Basic Design Technical Report for the New Flagship System “FugakuNEXT”

Version 1.1 (Public Version)

RIKEN

2026.5.29

Overview

The Ministry of Education, Culture, Sports, Science and Technology initiated, in August 2022, a survey and research project on next-generation computing infrastructure for the purpose of examining the next flagship supercomputer from the perspectives of systems, collaboration with new computing principles, and operational technologies, toward the development of next-generation high-performance computing infrastructure. RIKEN, a National Research and Development Agency that is the operating body of the supercomputer "Fugaku" (hereinafter referred to as "Fugaku") based on the "Act on the Promotion of Public Utilization of the Specific Advanced Large Research Facilities," has participated in the survey and research project on next-generation computing infrastructure and has been conducting investigations and examinations on the desirable system and operational directions for next-generation computing infrastructure. In June 2024, it was selected as the implementing body to promote the development and deployment project of the next flagship supercomputer. Subsequently, in order to develop and deploy a flagship supercomputer that can be operated around 2030, the Ministry of Education, Culture, Sports, Science and Technology launched the relevant development and deployment project in January 2025.

RIKEN, following the "Final Report on Next-Generation Computing Infrastructure" of the Ministry of Education, Culture, Sports, Science and Technology HPCI Planning Promotion Committee in June 2024, and taking into account the examination results in the survey and research project on next-generation computing infrastructure, is proceeding with the development and deployment project of a new flagship system that will succeed "Fugaku."

The codename of this system is hereinafter referred to as "FugakuNEXT," and its development and deployment project is referred to as the "FugakuNEXT" Project. In order for Japan's science, technology, and innovation to lead the world and contribute to the development of society and industry, it is indispensable to realize a flagship system for "AI for Science" that achieves world-leading performance in both simulation and AI, and further performs processing through tight coupling between simulation and AI, while automating and advancing science including scientific hypothesis generation and validation. Looking ahead to the future development of AI for Science, the goal is to develop and deploy a system that achieves more than 5–10 times the effective computational performance for existing HPC applications, and achieves execution performance of 50 EFLOPS or more with peak performance at the Zetta scale for AI processing, and to achieve tens of times acceleration in application execution that exceeds the overall 5–10 times improvement in effective computational performance through the fusion of simulation and AI.

RIKEN selected Fujitsu and NVIDIA as joint development partners through a public solicitation, and conducted the basic design of "FugakuNEXT" together with Fujitsu from June 2025 and with NVIDIA from August 2025, both through February 2026.

This report consists of three chapters. Chapter 1 describes the development policy and the proposed development schedule after the basic design. Chapter 2 describes the system target performance of "FugakuNEXT," and Chapter 3 summarizes the detailed examination results of the basic design.

INDEX

1. Development Policy

1.1 Overview of the Basic Design of the “FugakuNEXT” Project

RIKEN, a National Research and Development Agency, following the “Final Report on Next-Generation Computing Infrastructure” of the Ministry of Education, Culture, Sports, Science and Technology HPCI Planning Promotion Committee in June 2024, and taking into account the examination results in the survey and research project on next-generation computing infrastructure, started in January 2025 and has been proceeding with the development and deployment project of a new flagship system that will succeed “Fugaku” (the “FugakuNEXT” Project). In order for Japan’s science, technology, and innovation to lead the world and contribute to the development of society and industry, it is indispensable to realize a flagship system for “AI for Science” that achieves world-leading performance in both simulation and AI, and further performs processing through tight coupling between simulation and AI, while automating and advancing science including scientific hypothesis generation and validation. Looking ahead to the future development of AI for Science, the goal is to develop and deploy a system that achieves more than 5–10 times the effective computational performance for existing HPC applications, and achieves execution performance of 50 EFLOPS or more with peak performance at the Zetta scale for AI processing, and to achieve tens of times acceleration in application execution that exceeds the overall 5–10 times improvement in effective computational performance through the fusion of simulation and AI.

In “FugakuNEXT,” with application-first as the principle, in order to achieve the above goals under power constraints, a system is assumed that adopts, as the basic configuration, a high-bandwidth and heterogeneous node architecture that combines a power-efficient CPU component capable of effectively utilizing assets such as application software cultivated in “Fugaku,” and a bandwidth-oriented computational acceleration component. As part of this project, the basic design for the development of “FugakuNEXT” has been carried out since April 2025. In the basic design, while RIKEN serves as the development entity and conducts development throughout FY2025, Fujitsu was selected as a joint development partner from June 2025 and NVIDIA from August 2025 through public solicitation, and the basic design was jointly conducted with both vendors through February 2026.

The items implemented in the basic design are mainly as follows. For hardware, the functional and performance definitions of system components such as CPU and GPU processing elements, system hardware, and network, as well as the selection of device technologies to be used, were carried out. For system software, specification studies and functional design were advanced, and prototype implementations were also carried out. In addition, these activities were carried out through collaboration between RIKEN and the joint development vendors, while advancing co-design together with application developers assumed as users of “FugakuNEXT,” in order to achieve high effective performance in various applications.

1.2 Development Policy of the “FugakuNEXT” Project

In the development of “FugakuNEXT,” it is required not only to realize a world-leading system, but also to promote the project with consideration for maximizing scientific outcomes, advancing and inheriting domestic technologies with global appeal to the information industry, and deployment to the global market. Figure 1-1 summarizes the development strategy of “FugakuNEXT.” With the aim of strengthening Japan’s science, technology, and industrial competitiveness, three pillars are particularly established:
(1) Made with Japan, (2) technological innovation, and (3) sustainability/continuity.

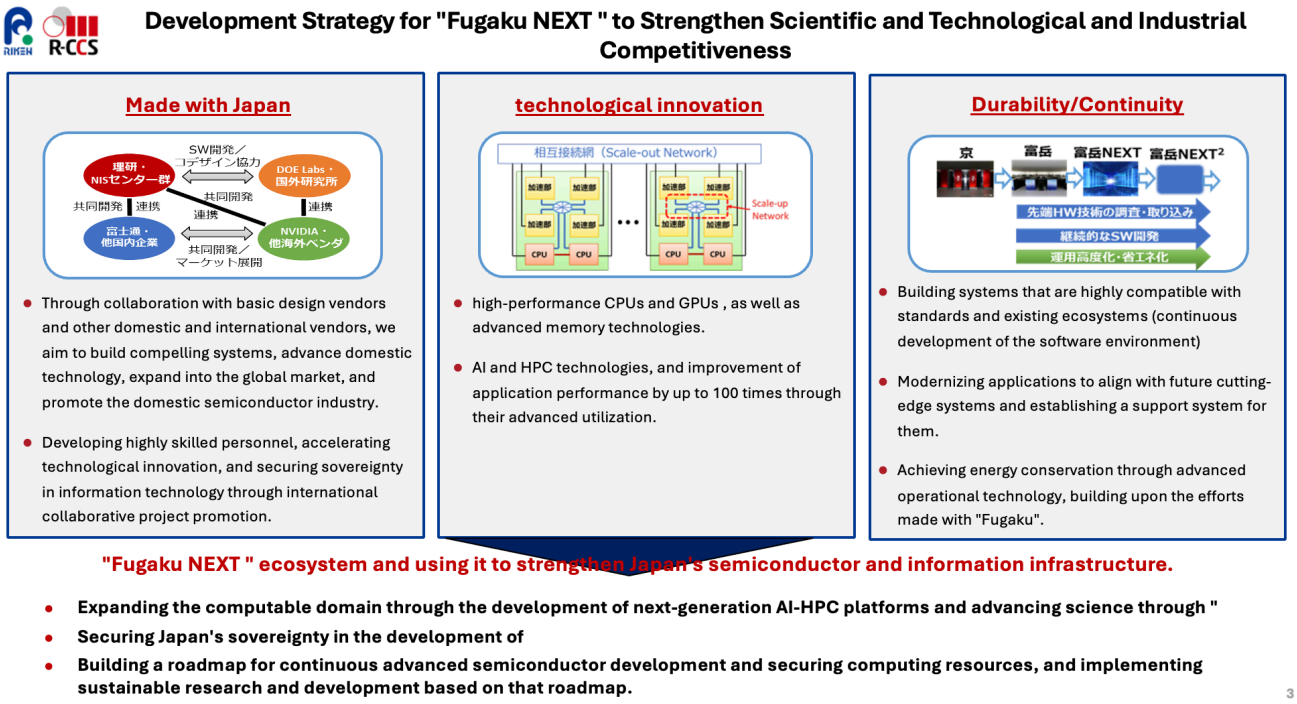


Figure 1-1

Based on the above development strategy, in order to proceed while maximizing cost-effectiveness, the following development policies were established in the basic design and the work was carried out.

1. As for the CPU component, in order to enable the utilization of application and system software assets accumulated in “Fugaku” to date, it shall, as a principle, have binary-level compatibility with “Fugaku.”
2. As for the acceleration component, it is necessary that an open software ecosystem be established so that it becomes easy for users to use from the perspectives of programming and performance optimization. In addition, in order to enable efficient code porting even before the operation of “FugakuNEXT,” the acceleration component architecture must be widely available at present. Therefore, GPUs, which already have a track record of use in large-scale supercomputers, are introduced as the computational acceleration component.
3. Regarding the hardware system itself, such as enclosures and cooling technologies, which are deliverables, and their design assets, it is also important that they be incorporated as part of the ecosystem and develop globally. For that purpose, it is necessary to develop a system that does not become high-cost and can adapt to a wide range of demand. Therefore, in designing each component and the system, open standards should be adopted as much as possible, and the reliability level should be designed to be within a necessary and sufficient range.
4. For system software such as compilers and libraries, from the perspective of continuous functional expansion and code maintenance after the completion of development, open-source

software shall be adopted as much as possible. In addition, the developed software shall, in principle, be released as open-source software so as to contribute to the development of “FugakuNEXT” and the community through modifications and functional extensions by users and development stakeholders. Even when commercial software is introduced, it shall be a principle to adopt software that is periodically updated, including in the future.

5. For the overall system, major components, and software to be developed in the “FugakuNEXT” Project, sufficient examination shall be conducted regarding cost and performance so that they have high market competitiveness. In addition, design shall be carried out while making efforts to understand the functions of computational resources required at the time of deployment.

1.3 Development Schedule of “FugakuNEXT”

Figure 1-2 shows the development schedule of “FugakuNEXT” assumed at present.

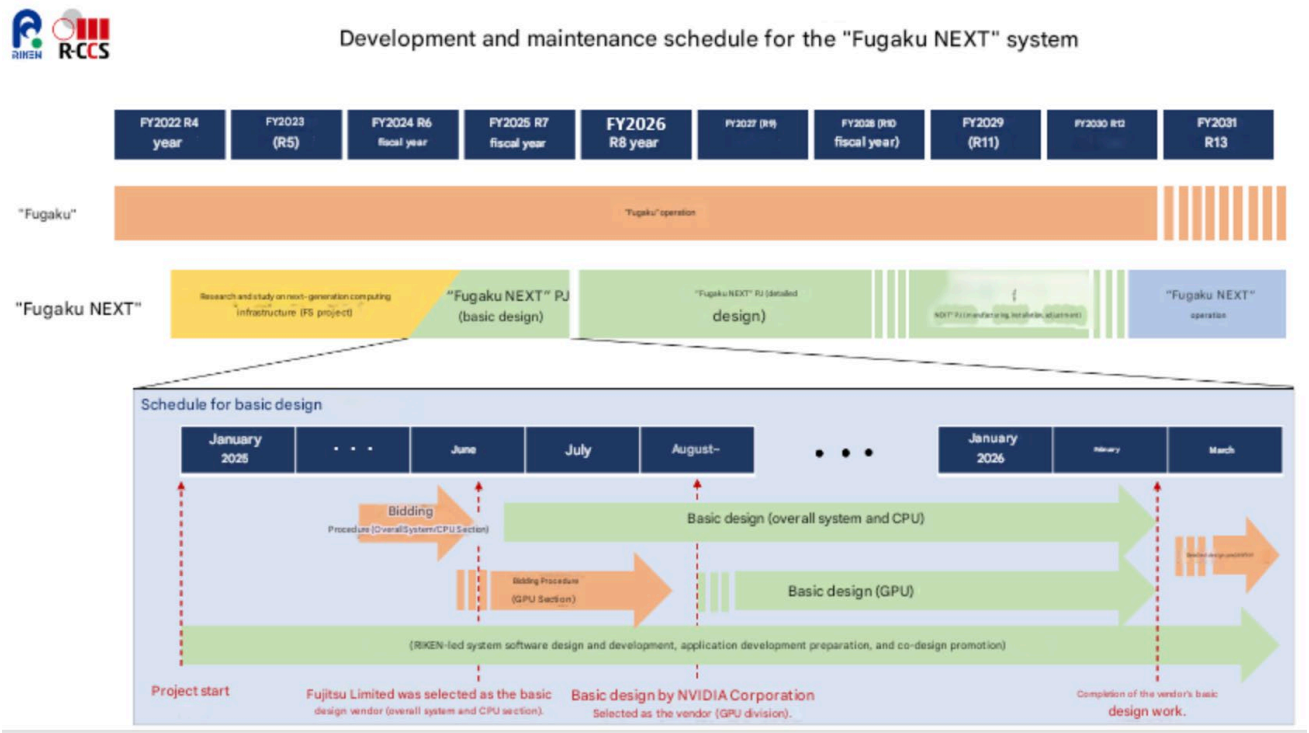


Figure 1-2

2. Target Performance

2.1 System performance targets

The system target performance values shown in Table 2-1 were defined as specifications, and the basic design was carried out. These performance values were compiled in the survey and research project on next-generation computing infrastructure of the Ministry of Education, Culture, Sports, Science and Technology, based on the following performance target items presented in the “Final Report on Next-Generation Computing Infrastructure.” With a system power consumption limit of 40 MW, target values are set mainly for FP64 vector performance for conventional HPC applications, matrix operation performance for AI-oriented low-precision computations such as FP16, BF16, and FP8, performance when utilizing sparsity, and main memory size and bandwidth.

- Effective computational performance of 5–10 times or more compared to the current level for existing HPC applications
- Effective performance of 50 EFLOPS or more for AI processing, assuming peak performance at the Zetta FLOPS scaleTarget of tens of times application acceleration overall through the fusion of simulation and AI

Table 2-1 Target performance in Basic design

item	CPU	Acceleration section	"Fugaku"	"Fugaku" ratio
Total number of nodes	Over		158,976	
Theoretical FP64 vector performance	48 PFLOPS or higher	2.6 EFLOPS or higher	537 PFLOPS	x5.7 or higher
Theoretical FP16/BF16 matrix calculation performance	1.5 EFLOPS or higher	150 EFLOPS or more	2.15 EFLOPS	x70.5 or higher
Theoretical FP8 matrix calculation performance	3.0 EFLOPS or higher	300 EFLOPS or more	-	
The same theoretical performance considering	-	600 EFLOPS or more	-	
Main memory size	10 PiB or more	10 PiB or more	4.85 PiB	x4.1 or higher
Main memory bandwidth	7 PB/s or more	800PB/s or more	163 PB/s	x4.9 or higher
Total power consumption	40 MW or less (compute nodes)		Approximately 30 MW	

3. Detailed Results of Basic Design Study (System Study Details)

This section describes each item examined in the system basic design, including architecture, application development, system software, storage and data management, facility, power, and cooling design, and operation design. Regarding the architecture, after describing the overall system, compute nodes, and Scale-out network that were examined, the co-design items established for the basic design are presented. In addition, an overview is provided of the performance models of CPU and GPU, memory performance characteristics, and investigations on application communication patterns.

Regarding application development, the development organization and plan, application development environment, application development, co-design feedback from applications, development of new applications for the FugakuNEXT era, and collaboration are described. Regarding system software, the programming environment, numerical computation libraries and middleware, communication libraries, and AI software are described. Regarding storage and data management, the architecture, support for utilization of storage systems, performance measurement methods and benchmarking methodologies, configuration studies and cost estimation under multiple budget assumptions, and items to be examined in the detailed design are described.

3.1 Architecture

3.1.1 Overall system

3.1.1.1 Overview

Figure 3-1 shows an outline of the overall system configuration of “FugakuNEXT.” The overall system of “FugakuNEXT” consists of the main system including compute nodes, shared storage which is an external storage device that shares data with the overall system, and peripheral systems including front-end computers.

The main system of “FugakuNEXT” has the following features.

- Equipped with Arm CPU FUJITSU-MONAKA-X
  - ➢ High HPC and AI performance by supporting Armv9 SVE2/SME2
- Heterogeneous compute node architecture
  - ➢ High power efficiency by a configuration combining FUJITSU-MONAKA-X and GPU
- Compute node design assuming commercial deployment to a wide range of users including Data Centers (DC)
  - ➢ Adoption of a compute node chassis that can be mounted in standard racks, enabling easy deployment into users’ existing facilities and rack equipment

The main system has a Scale-up network that connects GPUs for parallel computation using multiple GPUs, and a Scale-out network that connects compute nodes for parallel computation using multiple compute nodes. In addition, the overall system has a storage network for exchanging data with external storage devices, and a system management network used for management and control of compute nodes and peripheral systems within the overall system.

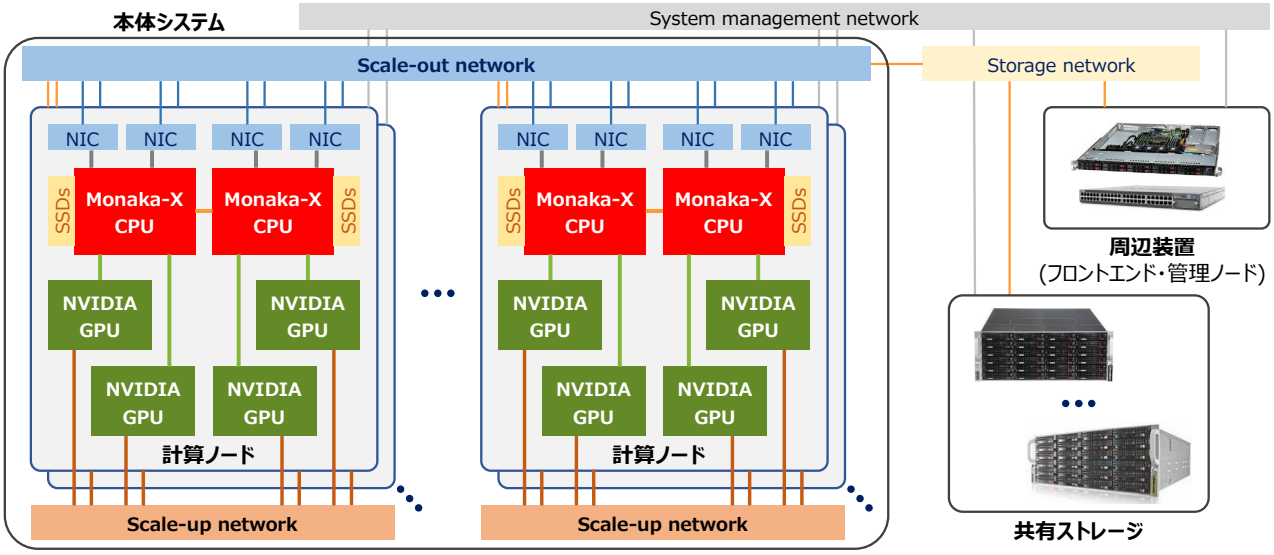


Figure 3-1 Overview of system design

3.1.2 Compute node

3.1.2.1CPU

The “FugakuNEXT” system is equipped with the CPU “FUJITSU-MONAKA-X” developed by Fujitsu. The “FUJITSU-MONAKA-X” CPU inherits the technology of the CPU “FUJITSU-MONAKA” for next-generation data centers, while providing functional enhancements for AI inference and strengthening memory and IO bandwidth, thereby enabling application to a wide range of fields such as HPC, data centers, and edge computing.

The features of the “FUJITSU-MONAKA-X” CPU are shown below.

- Cutting-edge technology (1.4 nm)
- 3D chiplet
- Many-core
- Low power consumption (ultra-low voltage)
- Reliability and security
- Vector engine (SVE2)
- Matrix multiplication engine (SME2)
- Tight GPU coupling (NVLink-C2C)

The “FUJITSU-MONAKA-X” CPU adopts a 3D chiplet, and achieves high performance, low power consumption, and low cost simultaneously by shortening data movement through three-dimensional interconnection between dies and by adopting mature technology for uncore-related dies.

3.1.2.2GPU

3.1.2.2.1 Basic GPU System Configuration

The FugakuNEXT GPU adopts a modular multi-chip module (MCM) architecture, overcoming the physical scaling limitations of conventional hardware. By adopting a tile-based logical design, it significantly improves compute throughput and memory capacity while ensuring high-speed connectivity across the node. This scalable approach enables a flexible balance between peak performance and operational efficiency according to diverse workload requirements.

3.1.2.2.2 GPU Computational Performance

The central principle in GPU compute design is to achieve an optimal balance between vector computation (streaming multiprocessor (SM) throughput and thread-level parallelism) and tensor computation (matrix operation performance). This key design trade-off determines the effectiveness of the system in two primary domains, depending on the degree of emphasis placed on each type of computation.

- HPC simulation workloads that depend on high-performance scalar and vector computation
- AI workloads centered on intensive matrix operations

3.1.2.2.3 Memory design

As a key factor that significantly affects GPU performance, multiple memory configuration options were presented in the basic design. The final selection among these options will be examined in more detail in the detailed design phase, taking into account factors such as technology maturity, product requirements, and schedule constraints.
In addition to the initial baseline, several architectural candidates are evaluated. The final selection will be made primarily based on the specific requirements of the expected workloads,

optimizing the balance among capacity, bandwidth, and power efficiency.

3.1.2.3 Scale-up Connectivity

In order to determine the most effective scale-up strategy, a comparative evaluation of two established interconnect architectures is conducted in the initial design phase. Although it may be extended in the future to consider various GPU counts, the current evaluation focuses on the following two configurations, which are existing system standards with proven track records at NVIDIA.

- NVL72 (switch-based scale-up): A large-scale configuration in which 72 GPUs are interconnected via a dedicated NVLink switch fabric. This approach prioritizes maximum aggregate throughput and unified memory across high-density compute nodes.
- NVL4 (bridge-based scale-up): A more compact configuration in which four GPUs are interconnected using NVLink bridges. This model represents a typical intra-node interconnect within a single computing tray.

3.1.2.3.1 NVL72

The NVL72 configuration is a high-density evolution of NVIDIA's current-generation architecture and is optimized for large-scale computation. It is largely similar to the current-generation system (Grace Blackwell GB200 NVL72), but with higher overall density. In the Vera Rubin generation, which is scheduled for release after the second half of 2026, each rack is planned to be equipped with 36 CPUs and 72 GPUs, with nine switches and nine NVLink switches arranged centrally. To enable 72-way interconnect, a passive copper backplane is installed within the NVLink domain, and space is allocated for its installation.

3.1.2.3.2 NVL4

While the NVL72 configuration emphasizes switch-based integration, NVL4-based racks provide a modular alternative that shifts architectural complexity from the backplane to the network layer. The NVL4 rack configuration differs from the centralized switch tray model of the NVL72 design in Grace Blackwell and Vera Rubin. In this configuration, GPUs are interconnected at the node level via NVLink bridges, eliminating the need for a passive copper backplane and central NVLink switches. By omitting NVLink switch trays, it becomes possible to integrate standard leaf switches (L1) directly into the rack. Although these leaf switches occupy more rack slots compared to compact NVLink switch trays, integrating them within the rack enables first-level scale-out connectivity within the same physical enclosure, effectively improving the overall system footprint.

3.1.3 Scale-out Network

The Scale-out network connects compute nodes in “FugakuNEXT” and is responsible for communication for parallel computation using multiple compute nodes. The Scale-out network mainly consists of network switches, NICs, cables, and related components. In this fiscal year, studies were conducted on network topologies for efficiently connecting a large number of GPUs, as well as on assumed performance and feasibility based on technology roadmaps.

3.1.3.1 Fujitsu’s plan

In the Scale-out network configuration assumed by Fujitsu, a Rail optimized fat-tree (Quad-rail optimized fat-tree) topology with four rails is adopted. This configuration is vendor-neutral for estimating power consumption and cost.Sixty-four compute nodes form one group, and each group has four Tier-1 switches. The four Tier-1 switches within a group each correspond to one rail, and among the Scale-out NICs installed in each compute node, those with the same index are connected to the same Tier-1 switch.

3.1.3.2NVIDIA’s plan

The basic design of the Scale-out network was examined using the Vera-Rubin generation technology of NVIDIA, which is scheduled to be introduced in the second half of 2026, as the baseline. The proposed system incorporates the following new technologies.

- Spectrum-X Ethernet: optimized adaptive routing and congestion control functions
- Spectrum-X switch: the next-generation switching silicon designed to deliver higher radix and lower latency than its predecessors
- ConnectX NIC: advanced network interface cards facilitating the massive injection bandwidth required per node
- The Scale-out network topology of the NVIDIA proposal consists of a classical two-tier Fat-tree configuration composed of L1 (Leaf) and L2 (Spine) layers, ensuring a non-blocking and high-bandwidth fabric.
- The L1 (Leaf) switches function as the primary aggregation points within a rack. Physically, the switches may not be placed within the computing rack.
- The L2 (Spine) switches function as inter-rack connections for global communication across the entire system.

As an example, in a configuration with four leaf switches per NVL72 rack, an oversubscription ratio of up to 1:1 can be supported. In a more realistic example with an oversubscription ratio of 4:1, a configuration with 144 L2 switches and up to 1152 L1 switches can be considered. The number of L2 switches required for the design is determined based on this oversubscription ratio and the total number of racks (or GPUs). In the default configuration of AI systems built by NVIDIA to date, a two-tier Fat-tree structure with an oversubscription ratio of 1:1 is commonly adopted. In the detailed design phase, requirements for HPC will be defined, and the design will be concretized, including the taper ratio.

The proposed topology assumes a baseline network architecture for the NVL72 configuration, but it can also be applied to configurations based on NVL4, and the performance of the Scale-out network is consistently maintained regardless of the Scale-up configuration selection. In configurations using NVL72 and NVL4, the two-tier Fat-tree structure is the same; however, the number of GPUs per rack, the scale and type of L1 switches, and cable routing differ, and therefore discussions across areas and working groups will be required in the detailed design phase. On the other hand, since the Scale-out network is designed independently of Scale-up, it is assumed that a consistent design can be applied regardless of the Scale-up configuration such as NVL72 or NVL4.The Spectrum-X network switches adopted in the NVIDIA proposal have hardware-based packet-level adaptive routing and proactive congestion control functions. This

enables high-performance and flexible communication control, such as reducing the frequency of data retransmissions and prioritizing specific flows. In addition, by utilizing In-band Network Telemetry, degradation of communication performance across the entire system due to a single workload can be prevented. Furthermore, a multi-plane architecture prevents fatal job interruptions due to partial failures. As a result, a highly fault-tolerant and robust network system can be constructed.

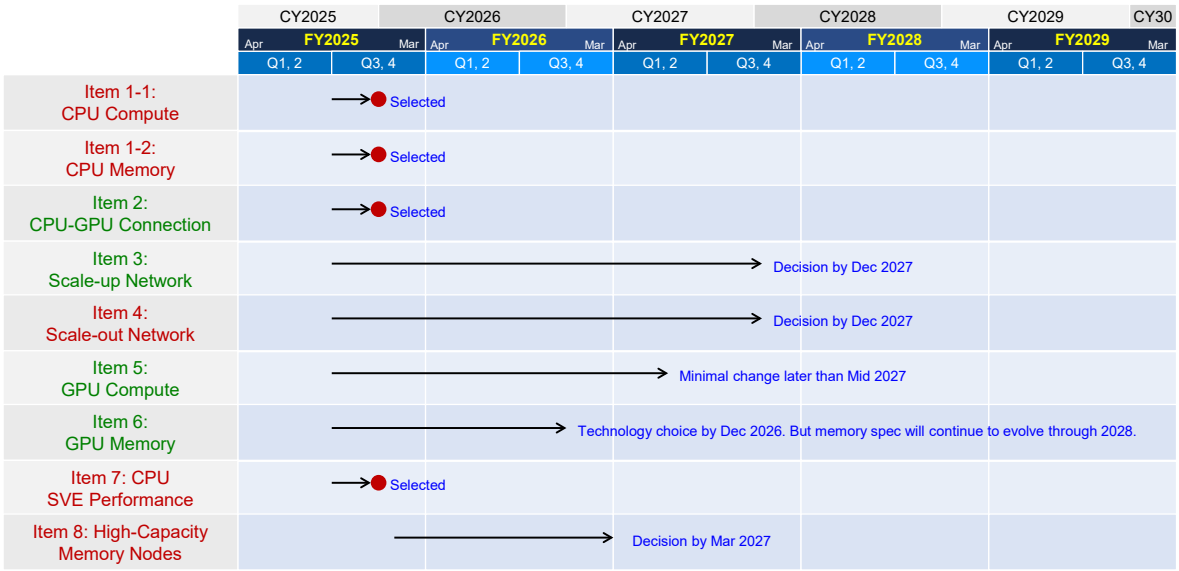
Finally, consideration was given to the introduction of Co-Packaged Optics (CPO) with the aim of improving power efficiency, reducing failure rates, and minimizing link flaps. In order to optimize the overall system power consumption, comprehensive evaluation and design are required, including the selection of network, CPU, GPU, and switches.

3.1.4 Co-design items

In the basic design, in order to explore the design space through co-design with applications for technically feasible options and to select the primary candidate technologies and specifications, the following architecture co-design items, Item 1 through Item 8, were established.

Item 1: CPU Specification
Item 2: CPU-GPU Connection
Item 3: Scale-up Network
Item 4: Scale-out Network
Item 5: GPU Compute Specification
Item 6: GPU Memory
Item 7: CPU SVE Performance
Item 8: Necessity of High-Capacity Memory Nodes

Table 3-5 Milestone of codesign



In addition, for Items 1-1, 1-2, 2, and 7, the primary candidates were selected during the system basic design period. The details of several items are described below.

3.1.4.1 Item 3: Scale-up Network

This item is a technical design option related to the Scale-up network, and the deadline for selecting the primary candidate is the end of December 2027. For multiple options regarding Scale-up domain size (number of GPUs) and bandwidth per GPU, it examines the desirable selection from the perspective of application performance sensitivity. It examines to what extent increasing the Scale-up domain size can improve actual application performance.

At present, this item is examined independently from Item 4 (Scale-out network). In the basic design, only technical options have been compiled, and co-design will continue to be conducted going forward.

3.1.4.2 Item 4: Scale-out Network

This item is a technical design option related to the Scale-out network, and the deadline for selecting the primary candidate is the end of December 2027. For multiple options regarding

injection bandwidth per GPU in the Scale-out network, it examines the desirable selection from the perspective of application performance sensitivity. Here, detailed information on network topology is not presented, but a Fat-tree-based configuration is assumed. In addition, since this is bandwidth per GPU, the injection bandwidth per node with a 4-GPU configuration is four times this value. Bandwidth per CPU is not considered here, but at present it is assumed that the per-node bandwidth corresponding to 4 GPUs can be utilized.

In the basic design, only technical options have been compiled, and co-design will continue to be conducted going forward.

3.1.4.3Item 5: GPU Compute Specification

This item is a technical design option related to GPU compute performance, and the deadline for selecting the primary candidate is mid-2027. Examples of compute performance per GPU socket at the present stage are presented, and the sensitivity of application effective performance to these is examined. In addition, items planned for future examination include the ratio between vector compute performance and tensor (matrix) compute performance, and the balance of computational precision in tensor cores.

In the basic design, only examples of compute performance have been presented, and co-design will continue to be conducted going forward

3.1.4.4Item 6: GPU Memory

This item is a technical design option related to GPU memory performance, and the deadline for selecting the primary candidate is the end of 2026. For various options of GPU memory technologies with different capacities and bandwidths, it examines the desirable selection from the perspective of application performance.
In the basic design, only technical options have been compiled, and co-design will continue to be conducted going forward.

3.1.4.5Item 8: Necessity of High-Capacity Memory Nodes

This item is a technical design option related to CPU nodes equipped with large-capacity memory, which differ from standard compute nodes, and the deadline for selecting the primary candidate is the end of March 2027. It examines whether there are HPC applications or AI frameworks that require CPU nodes with larger memory capacity than standard compute nodes. In other words, it asks whether there are applications or AI frameworks for which the memory capacity in Item 1-2 is insufficient. In addition, if such cases exist, it is also necessary to investigate the number of such applications and the expected usage of large-capacity memory.

Although further technical evaluation is required for feasibility, at present, CPU nodes with, for example, the following two types of memory capacities are assumed.
doubled memory capacity in 4x CPU sockets, or quadruple memory capacity in 2x CPU sockets

In the basic design, only example memory capacities of large-capacity memory nodes have been presented, and co-design will continue to be conducted going forward.

3.1.5 Investigation of Performance Models of CPU and GPU and Memory Performance Characteristics

In the design of FugakuNEXT, it is important to estimate application performance for CPUs, GPUs, and compute nodes composed of CPUs and GPUs. For the purpose of application performance estimation, investigations and validation of performance models, as well as investigations of performance characteristics of basic components such as memory, were conducted.

For the current CPU, the A64FX CPU of "Fugaku" (hereinafter referred to as the CPU), performance models for estimating the computational performance of application software were investigated, and the correctness of candidate models was validated using several benchmark programs provided by RIKEN on CPUs different from the CPU. In addition, for the current GPU (hereinafter referred to as the GPU), performance models for estimating the computational performance of application software were investigated, and the correctness of candidate models was validated using several benchmark programs provided by RIKEN on GPUs different from the GPU.

Furthermore, memory performance characteristics of each of CPU and GPU, as well as memory performance characteristics between CPU/CPU, CPU/GPU, and GPU/GPU, were investigated, and the corresponding memory characteristic models were also investigated and validated.

3.1.5.1CPU performance model

Based on profiling information of applications obtained using the CPU, a model was constructed to estimate performance on a different target CPU. The following were examined as candidate CPU performance models.

First, cycle accounting information (cycle account execution time) for the CPU is obtained using a profiler. Next, the execution time per core on the target CPU is estimated by multiplying each element of the cycle accounting information (instruction commit time, L1D cache access stall time, L2 cache access stall time, memory access stall time, etc.) by corresponding coefficients. Then, the execution time considering parallel processing efficiency is calculated using the number of cores when running the application on the target CPU. Finally, considering the Roofline model and others, the upper limit of effective performance is corrected so that the estimated performance becomes a realistic value.
As coefficients used to estimate the execution time per core, for example, the following were considered.

- Ratio of CPU frequency
- Ratio of the number of floating-point operation pipelines
- Ratio of SIMD length (including SIMD utilization rate)
- Ratio of floating-point operation latency
- Ratio of integer operation latency
- Ratio of L1 cache access latency
- Ratio of L2 cache access latency
- Ratio of memory access latency

3.1.5.2 GPU performance model

Based on profiling information of applications obtained using the GPU, a model was constructed to estimate performance on a different target GPU. The following were examined as candidate GPU performance models. First, the following metrics obtainable from NVIDIA Nsight Compute are used in the performance model

- Number of active warps (smsp_warps_active.avg.per_cycle_activ)
- Number of issued integer operation instructions
- Number of issued LoadStore instructions
- Number of issued single-precision operation instructions
- Number of issued double-precision operation instructions
- Ratios of issued instruction counts derived from the above information
- Memory bandwidth / cache hit rate information (used to determine whether loads are from cache or main memory)
- Number of SMs
- Number of active cycles
- Number of instructions per cycle within an SM

Based on these, a model was constructed to obtain the number of execution cycles while considering latency corresponding to each type of instruction.

3.1.5.3 Performance Estimation of CPU and GPU Using Candidate Performance Models and Its Validation

The applications used for model construction and evaluation consist of a computational kernel and non-kernel parts. The computational kernel is implemented either by selecting CPU or GPU execution using #ifdef, by an implementation described with OpenACC, or by a combination of these. Therefore, the computational kernel can operate on both CPU and GPU.
For measurement for data acquisition, measured data for both the computational kernel and the non-kernel parts are obtained using the A64FX CPU. In addition, measured data for the computational kernel are obtained using an A100 GPU, which is different from the validation GPU. Using these measured data, the performance of both the computational kernel and the non-kernel parts is estimated using the CPU performance estimation model, and the performance of the computational kernel is estimated using the GPU performance estimation model.

For validation measurements, measured data for both the computational kernel and the non-kernel parts are obtained on the CPU part of GRACE-HOPPER, and measured data for the computational kernel are obtained on the GH200 GPU of GRACE-HOPPER. By comparing the obtained measured data with the estimation results, the validity of performance estimation for the computational kernel and the non-kernel parts is verified. In addition, for the computational kernel, validation is performed by comparing estimation results including GPU with measured data.

For CPU performance estimation and validation, the following three applications were used.

- FrontFlow/Blue
- GENESIS
- SCALE

For these applications, the performance of the GRACE CPU was estimated from profiling data obtained on the A64FX CPU, and validated by comparison with measured performance on the GRACE CPU. As a result, cases were observed where the estimated execution time was longer than the measured value and cases where it was shorter. For cases where the estimated execution time was shorter than the measured value, it was confirmed that adjusting coefficients for memory access stalls and floating-point operation stalls brought the estimation closer to the measured value. In the future, it is necessary to improve accuracy by understanding cache access behavior and instruction scheduling behavior and by establishing a mechanism to determine coefficients. For cases where the estimated execution time was longer than the measured value, it is possible that optimization for the GRACE CPU has an effect,

and it is considered that accuracy may be improved by unifying compilers. These optimizations and further analysis are future tasks.

For GPU performance estimation and validation, the following three applications were used.

- FrontFlow/Blue
- GENESIS
- SCALE

For these applications, the performance of the GH200 GPU was estimated from profiling data obtained on the A100 GPU, and validated by comparison with measured performance on the GH200 GPU.

3.1.5.4Memory Performance Characteristics

In constructing the model of memory performance characteristics, the following memory performance metrics were measured. At that time, sequential, stride, and random access patterns were used. In addition, the machines shown in Table 3-6 were used for the measurements.

- CPU–local memory
- CPU–CPU (memory of a different sub-node)
- GPU–local memory
- CPU–GPU
- GPU–GPU

For the measured data, fitting was performed using several model equations proposed by RIKEN, and extraction of specific parameters and validation of the model equations were conducted. As a result, for each memory performance characteristic, it was possible to model the performance characteristics with a mean squared relative error (MSRE) well below the target accuracy of less than 10%.

Table 3-6 Specifications of Memory Performance Measurement Environment

Item	Fugaku	R-CCS (GH200 (gc-h200))	R-CCS (A100 (qa-a100))
Type	A64FX	Grace / H100	A100
Cores	48 (12/CMG)	72	–
L1 Size	64KB/way	128KB/core	192KB/SM
L2 Size	32MB/16way (8MB/CMG)	1MB/core	50MB (H100) / 40MB (A100)
L3 Size	–	114MB	–
Memory Size	32GB (8GB/CMG)	480GB	96GB (H100) / 80GB (A100)
Memory Bandwidth	1,024GB/s (256GB/s per CMG)	384GB/s	4,000GB/s (NVLink-C2C: 900GB/s) / 2,039GB/s (NVLink: 600GB/s)
Cache Line Size	256B	64B	128B
Compiler	FCC-4.12.1	GCC-11.5.0	NVHPC 25.9 / NVHPC 25.7

3.1.6 Investigation of Communication Patterns

For the purpose of performance estimation of the Scale-out network and feedback to the design, communication pattern information required by important benchmark applications was collected, and an environment for its analysis was constructed. Benchmark applications were executed on the supercomputer Fugaku, and communication pattern information was collected using MPI traces. In addition, based on the obtained communication pattern information, a tool was developed to visualize the communication status of each node over time, and a tool was developed to visualize, in stacked time graphs, how much and what kind of communication each node performed.

MPI traces of the benchmark applications executed on Fugaku were obtained using two tools: Structural Simulation Toolkit (SST) and Score-P. In SST, binary trace files in DUMPI format are obtained, and in Score-P, OTF2 files are obtained as MPI trace data. Tools were also developed to read the communication status of each node from these files and visualize it in graph form.
The benchmark applications for which communication traces were performed are shown in Table 3-7. MPI traces for GENESIS were obtained using Score-P, while traces for the others were obtained using SST_DUMPI. Going forward, in order to provide feedback to the design of the Scale-out network, studies will be conducted based on simulation and modeling using application communication patterns.

Table 3-7 Applications Used for Communication Pattern Tracing and Their Execution Sizes

Application Name	Description	Nodes	Processes
SALMON	First-principles calculation application targeting light–matter interaction	48	192
FrontFlow/Blue	High-precision prediction of non-compressible fluid dynamics; useful for usage analysis of ESI	768	3072
GENESIS	High-speed simulation assuming cellular environments; supports sub-angstrom calculations	4	16
LQCD-DWF-HMC	Hybrid Monte Carlo simulation (HMC) for lattice quantum chromodynamics (LQCD) using domain wall fermions (DWF)	2048	8192

3.1.7 Evaluation and Verification of High-Speed Link Connection Between CPU Component and Acceleration Component

As a result of advancing the basic design of the high-speed link connection method between the CPU component and the acceleration component, it was decided to consider NVLink-C2C (NVLink Fusion technology proposed by NVIDIA) as a strong candidate. Accordingly, prior to the start of the detailed design scheduled for the next fiscal year, evaluation and verification of the NVLink Fusion approach were conducted with the aim of accelerating the examination of the high-speed link connection method between the CPU component and the acceleration component and increasing the likelihood of achieving the project objectives.

3.1.7.1 Requirements for High-Speed Link Connection Between CPU Component and Acceleration Component

Evaluation and examination of the high-speed link connection between the CPU component and the acceleration component were conducted for the following specifications.

1. Cache coherence is maintained between the CPU component and the acceleration component.
2. The protocol complies with NVLink Fusion.
3. The bandwidth complies with NVLink-C2C.
4. As the CPU component, the Fujitsu CPU “FUJITSU-MONAKA-X” proposed in the basic design of “FugakuNEXT” is assumed.
5. As the acceleration component, an NVIDIA GPU proposed in the basic design of “FugakuNEXT” is assumed.

3.1.7.2 Evaluation and Verification Items

Evaluation and verification were conducted for the following items.
First, regarding logical connectivity, based on the above specifications, the following examinations were conducted on the logical connection method between the CPU component and the acceleration component, and the architecture was determined.

- Number of GPU sockets per node
- CPU–GPU connection method (protocol, data link, PHY, number of lanes, frequency, cache coherence control, etc.)
- NVLink Fusion CPU-side logical specification
- Consistency with FUJITSU-MONAKA-X ARM CHI logical specification
- Interface specification with GPU associated with NVLink Fusion

In addition, the following methods were examined as approaches for evaluating and verifying the logical connection method.

- Software simulation using VIP
- High-speed simulation using an emulator

Next, regarding the physical design, based on the above-described high-speed link connection specifications between the CPU section and the accelerator section, the following studies were conducted on the physical connection methods of the LSI, package, motherboard, and peripheral circuit components in the connection between the CPU section and the accelerator section.

- LSI floorplan
- Pinout of LSI and package
- Cross-section of package and motherboard

In addition, the following studies were conducted as methods to evaluate and verify the physical connection approach.

- Signal integrity verification methodology
- Power integrity verification methodology

Finally, regarding semiconductor technology, the following investigations and studies were conducted on the semiconductors and I/O circuits required to realize the examined logical connections and physical design, and the semiconductor technology node and I/O circuit IP to be adopted were determined.

- Specifications of semiconductor technology optimized for NVLink Fusion
- GRS (Ground-Referenced Signaling) of NVLink Fusion physical PHY
- Physical PHY for GPU connection associated with NVLink Fusion

In addition, estimations were conducted for circuit volume, area, power, and performance related to these.

3.2 Application development

3.2.1 Development Organization and Plan

3.2.1.1 Overview, Policy

In the system basic design, applications that can utilize FugakuNEXT and produce outcomes are selected, developed, and tuned. In addition, feedback to the determination of system design parameters is provided in this process. These series of activities are carried out from the perspective of system co-design in the design and development project.

Without falling into co-design based on a small number of limited applications, development of a wide range of applications is promoted, and development of frameworks to utilize them for co-design as needed is carried out in parallel. These frameworks are sequentially utilized in co-design, and enhancement of the frameworks is also advanced simultaneously.

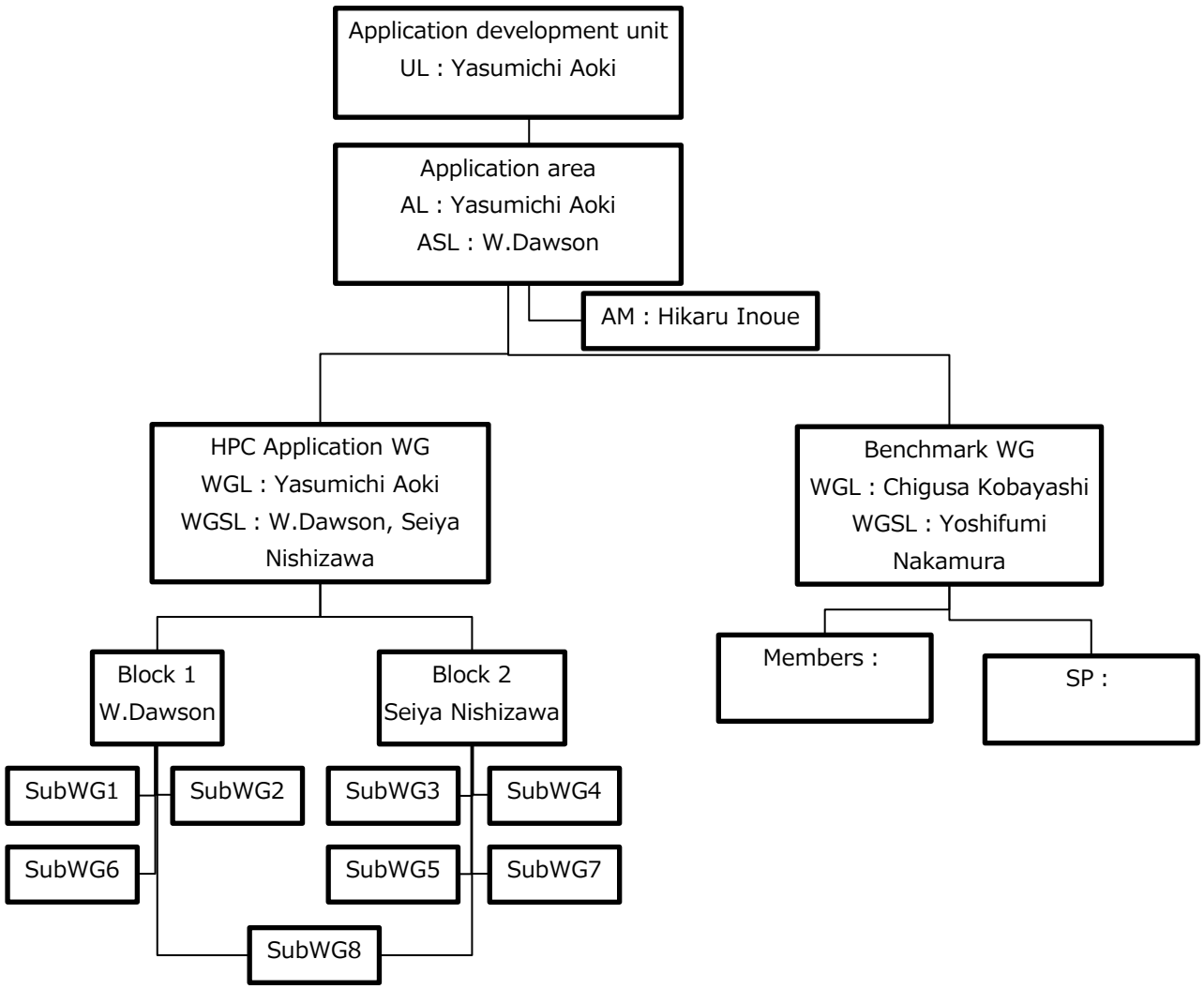
As the organization to promote such development, an application area is established under RIKEN Next-Generation Computing Infrastructure Development Division = Next-Generation Computing Infrastructure Application Development Unit, and collaboration with the application community and vendors, joint development of applications, benchmarking, and performance estimation assuming FugakuNEXT are conducted.

As the structure of the application area, sub-working groups are established for each application domain, and an application working group is established as an organization to oversee them. In addition, a benchmark working group is established to conduct application evaluation, benchmarking, and development of continuous development environments used for them, and development is carried out under a two-working-group structure as an area.

In constructing these structures, the organizational structure of the application study group of the post-“Fugaku” FS (Feasibility Study), which had been conducted mainly by RIKEN up to FY2024, was inherited, while partially modified according to activity results, and the application sub-working group structure was formed. The FS benchmark group was elevated to a working group in consideration of the importance and workload of its activities.

The application domains evaluated in FS and the selection of domain representatives were conducted by consulting application domains based on the “Computational Science Roadmap 2023” (https://cs-forum.github.io/roadmap-2023/) compiled by the HPCI Consortium Computational Science Forum. In this basic design, as a policy, the management of each sub-working group is newly assigned mainly to RIKEN R-CCS members (with a few exceptions), while collaboration with the application community continues to be carried out by the domain representatives from FS. In addition, members are added from within R-CCS for each domain as necessary. This enables collaboration with the community while also enabling close coordination within the division with other areas, resulting in efficient execution of operations.

3.2.1.2Project organization



The HPC Application WG consists of the following blocks and SubWGs.

Block	Organization
Block 1	BOG: William Dawson
	AD: Keiichiro Fukazawa, Kengo Nakajima
Block 2	BOG: Seiya Nishizawa
	AD: Takeshi Iwashita, Hirofumi Tomita

Sub WG	Organization
① Life Sciences [Sugita Team, TAMA Team]	OG: Shingo Ito LS: Kei Terayama
② New Materials & Energy [Nakajima Team, Yunoki Team]	OG: William Dawson LS: Yohei Yamaji
③ Weather & Climate [Tomita Team, Miyoshi Team]	OG: Shigenori Otsuka LS: Tomoo Kodama

	Yukio Adachi, Yuta Kawai, Arata Amemiya, James Taylor, Yuta Tarumi, (Unashish Mondal), Seiya Nishizawa
④ Earthquake & Tsunami Disaster Prevention [Tomita Team]	LS/OG: Kohei Fujita, (Seiya Nishizawa)
⑤ Manufacturing [Tsubokura Team]	OG: Junya Onishi LS: Ryoji Takagi AD: Chisachi Kato
⑥ Basic Science [Aoki Team]	OG: Issaku Kanamori LS: Tomoya Takiwaki, Issaku Kanamori
⑦ Smart City, Digital Twin, Society 5.0 [Yamaguchi Team]	LS/OG: Hiromi Yamaguchi AD: Takashi Shimokawabe Fukuji Tanaka
⑧ Scientific Computing & Machine Learning Algorithms	OG: Atsushi Suzuki LS: Daisuke Takahashi Rio Yokota

In addition, the following roles are defined for this unit.

UL: Unit Leader AL: Area Leader
ASL: Area Sub Leader
WGL: WG Leader WGSL: WG Sub Leader
AD: Advisor BOG: Block Organizer
LS: Liaison OG: Organizer
SP: S upport Personnel AM: Area Manager

Roles and Responsibilities

Role	Responsibility
UL	Responsible for this unit
AL	Responsible for the area
AM	Project management of the entire area Coordination with external parties of the area Preparation of minutes of plenary meetings
WGL	Responsible for the WG Management of WG organization Operation of development review meetings
WGSL	Project management of the WG Preparation of minutes of development review meetings Preparation of minutes of development review meetings
BOG	Operation of block meetings
AD	Advisor for application development
OG	Operation of SubWG Management of SubWG organization Preparation of minutes of review meetings
LS	Collaboration with the development community

- **Application Working Group**

  Development starting from the selection of applications in collaboration with the application community is carried out with this organization (RIKEN project members) as the core or contact

point, in cooperation with application developers. Development work is performed on a SubWG (Sub-Working Group) basis; however, as a unit for information transmission and sharing, blocks are formed by grouping four SubWGs, and development work is managed by block leaders. Block 1 consists of SubWG 1, 2, 6, and 8, and Block 2 consists of SubWG 3, 4, 5, 7 (,8). SubWG 1–7 correspond to HPC application domains, and SubWG 8 is responsible for numerical algorithms used in HPC (including machine learning). Each SubWG has an Organizer (OG) responsible for overall management and a Liaison (LS) responsible for collaboration with the development community. In principle, OG is assigned from full-time members of the R-CCS computational science team, and LS is assigned from individuals who served as domain representatives in the Feasibility Study (FS).
The questionnaire survey for the application community described later is conducted along the line of Block → SubWG → community developers.
Applications for initial evaluation in this fiscal year (EEA: Early Evaluation Applications) were selected, and for six of them, joint work with vendors has been initiated. These six application groups are referred to as EEA1.

- **Benchmark Working Group / Benchmark WG**

This WG is responsible for developing a framework that uniformly handles application software development, benchmarking, and performance estimation processes, and for publishing it both inside and outside the project. It collaborates closely with the Application Working Group and is also involved in incorporating target applications into the framework and conducting benchmarks. In this fiscal year, a Benchmark WG member was assigned to each EEA to support development. In addition, development of the Bench Kit used as the framework, examination of Benchpark utilization methods, and execution of hackathons were conducted in collaboration with the co-design WG and Benchpark developers at Lawrence Livermore National Laboratory (LLNL).

- **Collaboration with Vendors**

In the basic design of this fiscal year, points of contact with vendors in application development are consolidated into the co-design element technology study meetings. These study meetings consist of three parties: RIKEN, Fujitsu, and NVIDIA, and were held eight times at a frequency of once per month. In order to prepare agendas and deepen discussions for each study meeting, Sync-Up meetings were held in advance as bilateral meetings between RIKEN–Fujitsu and RIKEN–NVIDIA. In these meetings, presentation and examination of co-design elements and options from the architecture, progress of application development, and survey reports in the application area were conducted, and decisions and confirmations of tasks contributing to co-design, as well as exchange of information to be shared among both vendors and RIKEN, were carried out. From RIKEN, mainly members from the Application Area and Architecture Area participated, and the Application Area Leader served as the chair.

- **Collaboration with Other Areas**

As described in collaboration with vendors, co-design based on application software and feedback from application developers has been carried out mainly through collaboration between the Application Area and the Architecture Area. Based on the presentation of co-design elements and options from the architecture, feedback from applications was collected and shared. In addition, information on programming environments used by applications and coding using AI was shared with the System Software WG responsible for the basic design of those environments. From the next fiscal year onward, especially in providing development environments to application developers both inside and outside the project, collaboration with the Operations Technology Area will become important. Figure 3-11 schematically shows the inter-area collaboration in this fiscal year.

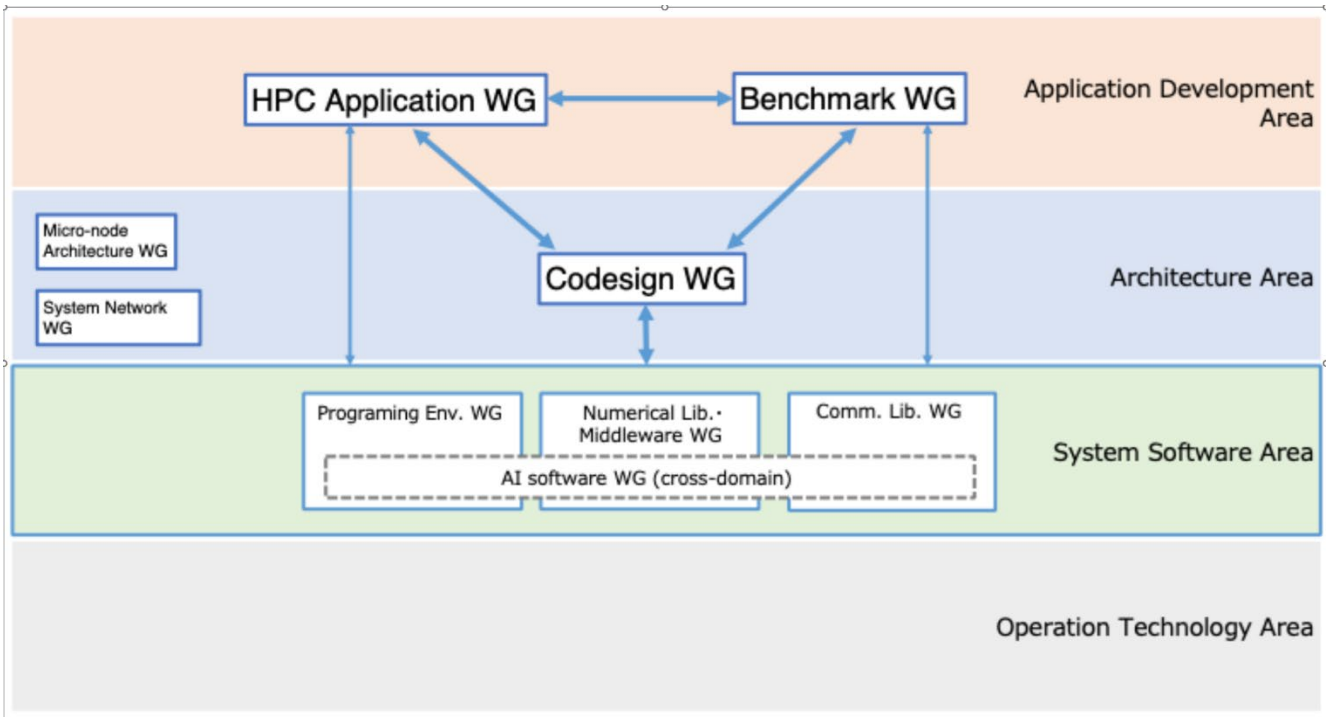


Figure 3-11

3.2.1.3Development Plan

- **FY2025**

The development items for FY2025 in FugakuNEXT application development are as follows. The corresponding sections are also indicated.

A: Development Environment Construction and Application Registration (3.2.2)
(1) Construction of continuous benchmark development environment
(2) Registration of applications in the environment

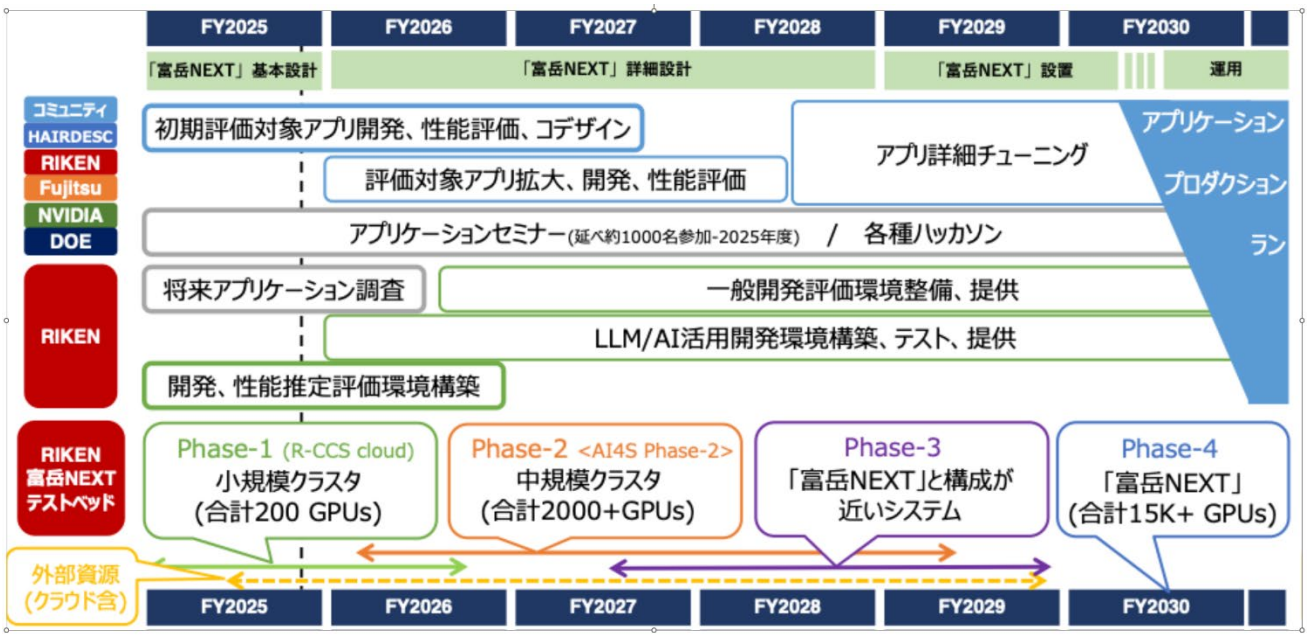
B: Application Development and Co-design (3.2.3, 3.2.5)
(3) Selection of initial evaluation target applications
(4) Determination of evaluation and co-design policy
(5) Narrowing down of applications to be evaluated in this fiscal year
(6) Initial performance evaluation of applications
(7) Application GPU tuning
(8) Evaluation of applications on current systems
(9) Performance estimation targeting FugakuNEXT

C: Co-design Not Based on Application Benchmarks (3.2.4)
(10) Feedback to system design items based on surveys of the application community
(11) Feedback to system design items based on analysis of Fugaku operation statistics

D: Development of New Applications Toward the FugakuNEXT Era (3.2.6)

- **Plan Until the Start of FugakuNEXT Operation**

The development plan of the Application Area until the start of FugakuNEXT operation, and the related testbed introduction plan, are shown below.


FY2025
FY2026
FY2027
FY2028
FY2029
FY2030
「富岳NEXT」基本設計
「富岳NEXT」詳細設計
「富岳NEXT」設置
運用
コミュニティ
HAIRDESC
RIKEN
Fujitsu
NVIDIA
DOE
初期評価対象アプリ開発、性能評価、コデザイン
評価対象アプリ拡大、開発、性能評価
アプリ詳細チューニング
アプリケーション
プロダクション
ラン
アプリケーションセミナー(延べ約1000名参加-2025年度) / 各種ハッカソン
RIKEN
将来アプリケーション調査
一般開発評価環境整備、提供
LLM/AI活用開発環境構築、テスト、提供
開発、性能推定評価環境構築
RIKEN
富岳NEXT
テストベッド
Phase-1 (R-CCS cloud)
小規模クラスタ
(合計200 GPUs)
Phase-2 <AI4S Phase-2>
中規模クラスタ
(合計2000+GPUs)
Phase-3
「富岳NEXT」と構成が
近いシステム
Phase-4
「富岳NEXT」
(合計15K+ GPUs)
外部資源
(クラウド含)
FY2025
FY2026
FY2027
FY2028
FY2029
FY2030

3.2.2 Application Development Environment

3.2.2.1 Development Plan

In this section, application development, execution, and performance evaluation are treated as an integrated process, and the contents examined and prepared from the perspective of the application development environment are described.

For co-design throughout the basic design and detailed design phases, it is required to provide quantitative information that contributes to the examination of various hardware requirements. Furthermore, through the development of application codes (algorithms) optimized to maximize the performance of next-generation systems, evaluation is conducted from the perspective of execution performance. For such examination and evaluation of hardware requirements, detailed application performance evaluation is indispensable in order to understand computational characteristics and bottlenecks of applications. At that time, in order to ensure comparability and reproducibility of evaluation results, a framework is required in which not only applications but also library and compiler versions and build conditions are clearly managed. A benchmark framework for systematically conducting performance evaluation is planned, and a design policy for a framework suitable for the purposes of this project has been formulated. In addition, since it is important to construct a continuous environment independent of specific vendors, the basic policy is to build a development and execution environment that actively utilizes open-source software (OSS) used internationally. However, considering barriers to entry for application developers and the status of application readiness, at this basic design stage, instead of consolidating into a single framework, multiple frameworks with different operational characteristics are combined to promote participation of a wider range of evaluation applications. By using the framework, updates to code through development and performance values can be obtained in a one-to-one relationship. Therefore, these data can serve as training data for AI-based development such as conversion to GPU code. In the detailed design phase, it is necessary to consider whether these code updates and performance values can be used as training data.

This framework is planned to be used not only for Early Evaluation Applications (EEA) described later, but also for applications and AI applications examined in system software. In addition, regarding performance evaluation methods, not only existing methods but also evaluation tools developed in system software are expected to be incorporated and used. Currently, development and performance evaluation are conducted mainly for 14 EEAs; however, toward the detailed design phase, it is necessary to include software with a wider range of programming models (such as GPU-resident models) and parallel models.

3.2.2.2 Benchpark

Benchpark is a Python-based open-source framework aimed at automating application benchmarking for HPC environments, primarily developed by Lawrence Livermore National Laboratory (LLNL). It adopts a configuration that uses Spack for software build and Ramble for execution workflow management and result collection, and is characterized by the ability to explicitly define the software stack including compilers and MPI and execute benchmarks in a reproducible manner. This configuration enables strict reproducibility testing including dependencies and long-term regression verification, and is effective as a foundation for systematically conducting strong scaling and weak scaling tests. In addition, by integrating with CI mechanisms, build and basic execution validation can be continuously performed. On the other hand, since environment construction based on Spack involves dependency resolution and generation of a large number of files, it may require time depending on the scale of the configuration and the usage environment. In particular, on Lustre, construction time tends to increase due to concentration of metadata operations. However, performance improvements

of Spack itself are continuously being pursued. In this framework, Benchpark is assumed to be used for evaluations that require detailed dependency management and high reproducibility.

3.2.2.3Benchkit

Benchkit is a lightweight benchmark framework based on Bash, which enables benchmark execution, CI/CD control, and aggregation of results executed across multiple sites. In this framework, Benchkit is positioned as the entry point for continuous evaluation. In Benchkit, after benchmark execution, result files in JSON format and detailed Performance Analysis (PA) information can be uploaded, and a mechanism is provided to extract, organize, and visualize necessary values on an aggregation page. In addition, by replacing the performance estimation model, comparative evaluations such as calculating performance ratios relative to Fugaku can be performed. These processes are pipelined through CI/CD mechanisms, enabling continuous execution from execution to result aggregation, performance estimation, and publication. Since it is based on Bash, it can be used without significant modification of existing build and execution scripts, and provides a configuration with low barriers to entry for external applications and multi-site participants.

In this framework, Benchkit is used as a layer responsible for lightweight continuous evaluation and performance aggregation, and Benchpark is used in combination for evaluations that require strict reproducibility including dependencies. It is also possible to invoke Benchpark from Benchkit, and the two are complementary. As future studies and implementations progress, useful components such as performance aggregation mechanisms and estimation mechanisms will be considered for integration into Benchpark or enhancement of interoperability in stages. This aims to construct an extensible development infrastructure that achieves both operational efficiency and reproducibility.

3.2.2.4Performance evaluation methodology

For performance evaluation, an extensible architecture that loosely couples continuous benchmark execution with performance estimation and analysis is considered as the basic policy. At present, Benchkit is used as an implementation example; however, this design is not fixed to a specific implementation, and it is assumed that an appropriate configuration will be concretized through future studies and operation.

For result data generated by benchmark execution, the basic approach is to manage separately JSON-format files that store lightweight metadata and TGZ-format archive files that contain detailed performance analysis information. By associating both using a common UUID, it is possible to achieve both minimal metadata management and flexible storage of large data for analysis. Based on this UUID, a configuration is being considered to connect to various pipelines for performance estimation and additional analysis.

For the performance estimation mechanism, it is assumed to adopt a configuration separated from the benchmark execution infrastructure itself. Rather than embedding specific performance models or estimation methods rigidly within the infrastructure, a promising option is to call estimation modules implemented as external tools as part of the CI/CD pipeline. By adopting such a decoupled design, it is considered possible to flexibly apply different estimation methods for each application and to update or replace performance models. In addition, assuming that performance estimation models may contain confidential information, operational forms that do not disclose estimation logic or internal parameters are also considered.

Furthermore, the scope of publication of benchmark results and performance estimation

results is being considered to be flexibly controllable on a per-application basis. The aim is to support operational modes that consider constraints on information disclosure, such as non-disclosure of raw data and authenticated publication.

This mechanism is still at the basic design stage and is planned to be refined step by step based on future implementation and operational status.

3.2.2.5Practical Use of the Development Environment

This section shows, in Figure 3-12, the overall flow of how applications are introduced, built, and evaluated using the prepared application development environment and performance evaluation framework.

First, as shown in the upper left of the figure, the source code and input data of each application are managed using a version control system in repositories on the Internet or on internal servers (in the case of non-public applications).

The benchmark framework acquires these information as necessary and performs application build, execution, and performance evaluation on each computing system. The results of execution and performance evaluation are automatically transmitted to a database server for performance evaluation, and project participants can share and review these evaluation results. With this mechanism, it becomes possible to perform continuous and reproducible performance evaluation while clearly managing the software configuration subject to evaluation.

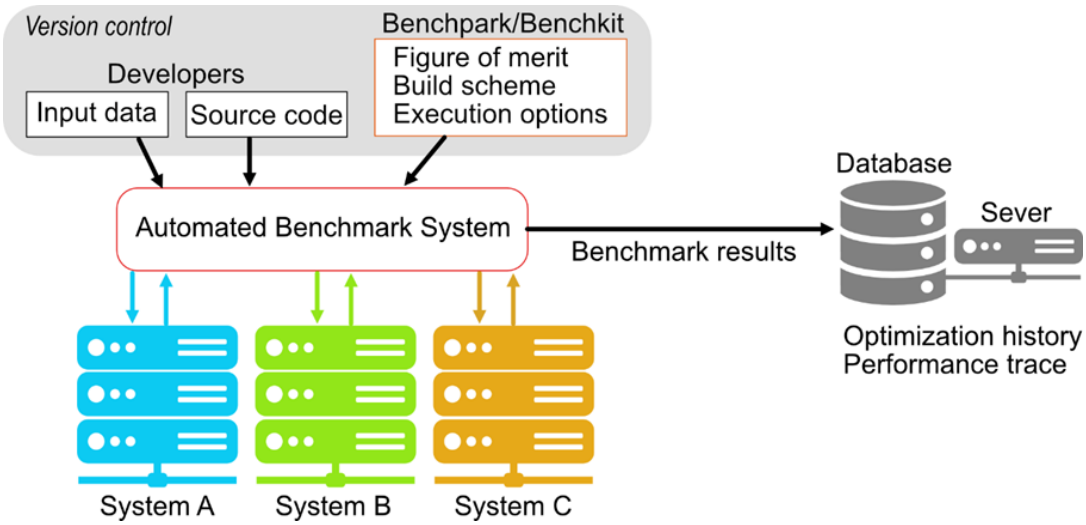


Figure 3-12 Benchmark framework

3.2.2.6Toward Public Release of the Development Environment

The performance evaluation framework constructed in this project has already been publicly released and is available as a platform for conducting application performance evaluation.

Benchpark https://github.com/RIKEN-RCCS/benchpark,
Benchkit https://github.com/RIKEN-RCCS/benchkit

On the other hand, regarding the method of publishing data accumulated as performance evaluation results, it is necessary to consider usage forms and scope of disclosure; therefore,

examination in the detailed design phase is required.
In addition, regarding the source code of applications subject to evaluation, since many software are open-source software (OSS), many are already publicly available. On the other hand, input data may contain sensitive information related to future research contents and plans, and therefore, users have indicated that it should be handled separately from source code. Accordingly, the method of publishing input data needs to be examined in the detailed design phase.
Furthermore, for non-public applications, it is necessary to consider contractual conditions with copyright holders and relevant regulations such as export control, and discussions with each developer are currently ongoing.

3.2.3 Application Development

3.2.3.1 Policy for Selecting Target Applications for Development

Application development in this project has the primary objective of providing effective feedback to the architecture design of FugakuNEXT. Therefore, rather than merely focusing on portability or extension of existing performance, representative application groups that will drive future large-scale scientific computing are selected as targets. The basic policy for selection is as follows.

- High importance within the domestic research community
- Possession of scale and complexity aimed at achieving scientific breakthroughs on FugakuNEXT
- Diversity in computational characteristics to expose design trade-offs
- Feasibility of establishing a development structure with a view toward future upstream merging

3.2.3.2 Early Evaluation Applications（EEA）

(1) Overview

Early Evaluation Applications (EEA) are core application groups for clarifying performance characteristics, memory requirements, and communication characteristics at the early stage of architecture design.

The selected applications are as follows.

Application name	SubWG	type	Computational Characteristic Classification
Genesis	SubWG1	Molecular dynamics simulation	3
UT-Heart	SubWG1	Cardiac electrophysiology and electromechanical coupled simulation	5 / 6
SALMON	SubWG2	First-principles electronic structure calculation (TDDFT)	5
mVMC	SubWG2	Strongly correlated electron system simulation using quantum Monte Carlo method	1
SCALE-LETKF	SubWG3	Weather simulation and data assimilation (Local Ensemble Kalman Filter)	1, 4, 5
E-Wave	SubWG4	Seismic wave propagation simulation	6
FrontFlow/blue	SubWG5	Fluid analysis using unstructured grid finite volume method (CFD)	6
FFVHC-ACE	SubWG5	High-fidelity fluid analysis using unstructured grid finite volume method	5
LQCD-DWF-HMC	SubWG6	Lattice QCD simulation (Domain-Wall Fermion, HMC)	5
MHDTurbulence	SubWG6	Magnetohydrodynamic turbulence simulation	5
Sionna + SUMO	SubWG7	Digital twin / large-scale social simulation	5, 6
PETSc	SubWG8	Large-scale parallel linear solver framework	
PyTorch/Megatron	SubWG8	Large-scale language model training (distributed deep learning framework)	

(2) GPU Support Status and Computational Characteristic Classification
For each EEA, the GPU support status, programming model used, and computational characteristics (B/F ratio, memory access pattern, computational intensity) were organized.
The computational characteristics were classified based on the following criteria:

1. Matrix-multiplication type (high computational intensity)
2. Low B/F, simple loop type
3. Cache-blocking applicable type
4. Low B/F with complex loop structure
5. High B/F with contiguous memory access
6. High B/F with indirect memory access

As a result, it was clarified that many applications are **memory bandwidth-bound or communication-bound**, while purely compute-bound applications are limited.
In the detailed design phase, constructing an application set with broader computational characteristics will be a key challenge.

3.2.3.3Security Export Control

When disclosing non-open-source software applications to overseas vendors, appropriate security export control must be conducted to prevent the proliferation of weapons of mass destruction, excessive accumulation of conventional weapons, leakage of sensitive technical information, and to maintain international trust.
At RIKEN, the Research Integrity and Economic Security Office conducts the review. Departments requesting disclosure must prepare appropriate documentation and submit it for review.
The applications subject to review are:

- FFB
- FFVHC-ACE
- E-Wave

Although these applications are not developed by RIKEN, since RIKEN is responsible for disclosure to overseas vendors in this project, RIKEN must conduct the export control review.
As of September 2025, the following documents were prepared and submitted:

1. **Classification Determination Sheet (Technology)**
   Includes:
   - Technology name
   - Description
   - Classification result
   - Rationale
   - Supporting materials
     (Reference: “Matrix Table of Goods and Technologies”)

2. **Review Form (Technology Provision / Joint Research)**
   Includes:
   - Technology name
   - Overview
   - Recipient information
   - End use
   - Providing organization

3.2.3.4Consideration of Joint Research Agreements

When jointly developing proprietary (non-open-source) software with vendors, it is necessary to clearly define ownership of intellectual property rights in addition to confidentiality obligations. In this project, RIKEN, NVIDIA, Fujitsu, and application developers are expected to collaboratively develop applications by leveraging their respective strengths. Therefore, establishing a joint research agreement among stakeholders is required.
The basic framework of the agreement is as follows:

1. **Contract Structure**

- Four-party joint research agreement (ERI Univ. of Tokyo, Fujitsu, NVIDIA, RIKEN)
- Joint development including implementation and algorithm design

2. **Source Code Disclosure Scope**

- Limited to project members (RIKEN, Fujitsu, NVIDIA developers)
- Restricted to project-related use

3. **Benchmark Performance Data**

- Openly published
- Used for academic and technical dissemination

4. **Intellectual Property Rights**

- **Background IP**
  - Retained by each party
  - Shared within project scope upon agreement

- **Foreground IP**
  - Freely usable among participating parties (including modification and commercialization rights)
  - Licensing to third parties decided by mutual agreement

3.2.3.5Testbed Usage Policy

A testbed environment will be provided as a platform that application developers can freely use. This environment will be equipped with the latest GPUs prepared by RIKEN, and in the future, prototype systems such as MONAKA, with the goal of enabling a seamless transition to FugakuNEXT.
The testbed is planned to be widely accessible to users. However, the source code of applications developed within this environment will be utilized as training data for application enhancement support functions. Therefore, users of the testbed will be required to agree to the terms of use, including the following provisions. Users will, of course, also be able to utilize the enhancement support functions.

- Source code may, in principle, be used for reinforcement learning of the application enhancement support functions.
- The intellectual property rights of AI models trained for the enhancement of these support functions shall belong solely to RIKEN.

3.2.3.6 EEA1 Initial Evaluation Applications – Group 1

EEA1 consists of applications selected for early joint evaluation with vendors.
Selection criteria:

- Rapid availability to vendors
- Close technical collaboration feasibility
- Clearly identifiable GPU optimization challenges

Selected applications:

- Genesis
- SALMON
- SCALE-LETKF
- E-Wave
- FrontFlow/blue
- LQCD-DWF-HMC

NVIDIA received all six applications.
Fujitsu received two:

- SCALE-LETKF
- LQCD-DWF-HMC

3.2.3.7 Summary of SubWG ActivitiesThis section summarizes the individual reports submitted by each SubWG.

Detailed descriptions of activities in each domain, the rationale for selecting Early Evaluation Applications (EEAs), computational characteristics, design challenges, and future key topics are provided in the supplementary document *"Application WG SubWG Reports."* In this section, key points directly relevant to architecture design considerations in the basic design phase are extracted and organized from a cross-cutting perspective.

While each SubWG report includes domain-specific scientific background and community activities, this section focuses primarily on the following aspects:

- Computational characteristics of the Early Evaluation Applications
- GPU support status and porting challenges
- Requirements for memory capacity, bandwidth, and communication performance
- Precision requirements and the applicability of mixed precision
- Potential for CPU–GPU collaborative utilization

For detailed individual reports, refer to Appendix A.

SubWG1: Life Sciences

In the life science domain, the molecular dynamics simulation code GENESIS and the cardiac multi-scale coupled simulation code UT-Heart were selected as Early Evaluation Applications. Both have optimization experience on "K" and "Fugaku" and are based on hybrid parallelization using MPI and OpenMP.

GENESIS targets multi-scale MD including all-atom models, coarse-grained models, and QM/MM. Although partial CUDA implementations exist, most of the computation is still executed on CPUs. Particle interaction calculations and memory access account for the main computational cost, and dependence on memory bandwidth is strong. By advancing GPU residency in the future, GPU memory capacity and bandwidth will become the factors determining performance.

UT-Heart is characterized by large-scale sparse matrix computation including electrophysiology–mechanics coupling and involves time-evolution iterative computation and large-scale I/O.

From this domain, the continued importance of CPU-side performance, sufficient GPU memory capacity, and large-capacity memory demand associated with multi-scale support were identified as important design points.

SubWG2: Materials and Energy

In this domain, the time-dependent density functional theory code SALMON and the multi-variable variational Monte Carlo code mVMC were selected.

SALMON mainly consists of stencil-type computations targeting electron dynamics and has high requirements for memory bandwidth and latency. Instead of conventional k-point parallel benchmark configurations, a supercell expansion problem setting was newly constructed to evaluate computation patterns aligned with future scientific challenges.
On the other hand, mVMC is a compute-bound application that heavily uses dense linear algebra operations and is positioned as important for evaluating double-precision floating-point performance. However, GPL license constraints and lack of a GPU version are issues.
In the materials domain, in addition to conventional first-principles calculations, AI-integrated computations such as machine learning interatomic potentials are becoming increasingly important. While double precision is required in many methods, there are also regions where mixed precision can be applied.
From a design perspective, FP64 performance, memory bandwidth, and support for mixed precision operations are important.

SubWG3 Weather and Climate Domain
In the weather and climate domain, SCALE-LETKF, which combines the numerical weather prediction model SCALE and the local ensemble transform Kalman filter LETKF, was selected. This application has a composite structure including large-scale three-dimensional grid computation, matrix operations, MPI communication, and large-scale I/O.
The SCALE part is dominated by stencil-type computation and has a strong memory bandwidth-bound tendency. The LETKF part uses matrix operations but GPU porting has not been completed.
From this domain, memory bandwidth-oriented design, evaluation of matrix computation support functions, and assessment of applicability of low-precision computation were identified as design issues.

SubWG4 Earthquake and Tsunami Disaster Prevention Domain
In this domain, the earthquake wave propagation code E-Wave based on unstructured finite element methods was selected. Sparse matrix-vector multiplication and conjugate gradient methods are the main computations, and it is a typical memory- and communication-bound application where neighbor communication and Allreduce via MPI occur frequently.
Furthermore, in this domain, demonstration studies of CPU-GPU cooperative utilization were reported, showing the possibility of reducing execution time and energy consumption simultaneously by reducing iteration counts on the GPU using initial value estimation with CPU memory.
From a design perspective, the importance of high-bandwidth memory, low-latency communication, and high-speed CPU-GPU interconnect is suggested.

SubWG5 Manufacturing Domain
In the manufacturing domain, the unstructured grid LES code FrontFlow/blue and the structured grid LES code FFVHC-ACE were selected. The former has indirect memory access characteristics, while the latter is stencil-based, and they have different memory access characteristics.
Both target large-scale turbulence analysis, and memory capacity, bandwidth, and inter-node communication performance determine performance. GPU execution is possible, but optimization opportunities remain.
From this domain, achieving both memory capacity and bandwidth, ensuring communication performance, and promoting GPU-first optimization were identified as important design issues

SubWG6 Basic Science Domain
In the basic science domain, the lattice QCD code LQCD-DWF-HMC and the magnetohydrodynamic turbulence code MHDTurbulence were selected.
LQCD mainly consists of stencil computation and iterative solvers, and global reduction occurs

frequently. Mixed precision algorithms are introduced, and communication performance and memory bandwidth are dominant. It is also suggested that pod size may affect performance.
MHDTurbulence is a time-evolution finite difference computation on a three-dimensional uniform grid and has strong dependence on memory bandwidth. Improvement of portability through introduction of Kokkos is also under consideration.
From this domain, communication performance, mixed precision support, and GPU-to-GPU direct connection performance were identified as important.

SubWG7 Smart City Domain
In this domain, Sionna for wireless field simulation and SUMO incorporating Agentic AI were selected. It has a structure where highly branching and irregular memory access computation by ray tracing and high computational intensity computation including neural network inference are mixed.
Currently, evaluation is at the component level, and MPI parallelization and integrated simulator construction are future issues.
From a design perspective, branching performance, GPU computational performance, and large-scale parallel communication performance are important.

SubWG8 Numerical Computation and Machine Learning Algorithms Domain
In this domain, the sparse matrix solver library PETSc and PyTorch/Megatron as an LLM training platform were targeted.
PETSc is a memory-bound library centered on SpMV and assumes distributed computation in a GPU-mixed MPI environment. There is a possibility of increasing computational intensity through preconditioning.
On the other hand, PyTorch/Megatron is a deep learning workload with high computational intensity and mixed precision assumptions, where high-speed communication between GPUs determines performance.
From this domain, the balance between memory bandwidth and computational performance, support for low precision computation, and ensuring GPU-to-GPU direct connection performance were identified as important design considerations.

3.2.4 Codesign Feedback from Applications

This section outlines the research activities related to codesign. The architecture team formulated the system basic design incorporating multiple options for each design choice. The application team was responsible for providing feedback on these design options.

3.2.4.1 Collection Methodology

(1) Developer Survey (DA) A/B

In the early design phase, developer surveys were conducted through each SubWG. The main points are as follows.

- Nature of kernels remaining on CPU
- Usefulness of CPU matrix engines
- GPU memory capacity requirements
- Node usage scale
- Necessity of CPU-GPU coherency
- Importance of latency

In Survey B, more specific investigations were conducted on the B/F ratio of CPU-resident kernels and node memory requirements.

(2) Fugaku Job Statistical Analysis (FY2024)

To analyze broader application characteristics, statistical analysis was conducted on jobs actually executed on the Fugaku supercomputer. Targeting FY2024, jobs with execution time of one hour or longer were selected for analysis. The number of analyzed jobs was 3,129,958, and node-time-weighted analysis was performed for these to evaluate the following.

- FLOPs efficiency distribution
- Memory usage distribution
- B/F ratio estimation
- Communication volume

3.2.4.2 Analysis Results for Key Design Items

(1) CPU Design

From the Developer Survey, the following opinions were predominant.

- Kernels remaining on the CPU are mainly I/O, pre/post processing, and branch-heavy processing
- High arithmetic intensity kernels are limited
- Demand for matrix engines is not clea

From Fugaku actual data as well, there are few cases achieving high FP efficiency on the CPU side, with 734 jobs achieving more than 30% of peak performance and only 93 exceeding 50%, confirming that overall there is a strong memory bandwidth-bound tendency.

(2) CPU-GPU Connection

From the Developer Survey, the following opinions were obtained.

- Strong demand for high bandwidth
- Coherent memory contributes to improved productivity
- This is a design item strongly coupled with memory capacity selection

(3) Scale-up / Scale-out Network

From the Developer Survey, the following opinions were obtained.

- Typical usage scale is 25–64 nodes
- A pod size of 72 GPUs can cover half

- Bandwidth is prioritized, latency is also important in some cases

(4) GPU Memory Capacity

In the survey, 128–256GB was the main range, and there were also cases requesting 512GB or more. However, it is considered necessary to re-evaluate consistency with memory bandwidth-bound characteristics. Capacity requirements depend mainly on the following factors.

- FFT stack
- Replica/ensemble
- Large-scale sparse matrices
- Number of particles

3.2.4.3Feedback of each System Version

This section describes the hardware design options presented by the architecture team and the corresponding feedback from applications.

System v0.5

Codesign Item 1: CPU Specification

- ✧ Related survey items: Developer Survey (ref:4.2.4.4) (A) 1, 2
- ✧ Related job statistics: Memory usage distribution
- ✧ Key findings from developer survey:
  - The use cases for the CPU-side matrix engine remain speculative. Examples include small dense matrix operations, analysis, and proxy AI tasks. Regarding precision, double precision was suggested to be useful, but this result lacks clear justification and requires validation.
  - No questions were conducted regarding the two types of CPU memory configurations.

- ✧ Key findings from job statistics:
  Total memory capacity: 4×730 (=2,920) GB/node covers 50% of jobs, and 4×1600 (=6,400) GB covers 90%.
  Assuming no duplicate copies exist in both GPU and CPU memory, CPU-side memory capacity:
  - If GPU is 128 GB each, 4 GPUs/node: 2,408 GB covers 50% of jobs, 5,888 GB covers 90%.
  - If GPU is 256 GB each, 4 GPUs/node: 1,896 GB covers 50% of jobs, 5,376 GB covers 90%.

  From the data, it is not possible to determine which SME option is better. If the GPU design is very aggressive (i.e., FP4/FP6/FP8 only), CPU SVE could act as a strong risk mitigation factor; thus, this could have been evaluated if GPU specifications were clearer.
  Job statistics suggest that larger memory capacity should be favored. While compute capability grows faster than memory capacity, application developers can adapt; therefore, memory projection should be conservative.
- ✧ Recommendation:
  - SME: A or B
  - Memory: A

Codesign Item 2: CPU-GPU Interconnect

- ✧ Related survey items: Developer Survey (ref:4.2.4.4) (A)6
- ✧ Related job statistics: None
- ✧ Key findings from developer survey:

- Performance: As high CPU–GPU bandwidth and as many concurrent transfers as possible are desired. Developers prioritize performance over ease of programming. Some concerns exist regarding overhead from memory synchronization.
- Cache coherence: Contributes to productivity but should remain optional and must not reduce peak bandwidth or user control. Most developers (though not all) already explicitly manage data transfers. At least one developer expressed concern about performance degradation on systems without hardware coherence when targeting Unified Memory.

Developers prefer higher-performance interconnects and tend to support NVLINK. Some applications rely on unified memory and are concerned about performance degradation on platforms without such support.

✧ Recommendation: A

Codesign Item 3: Scale-up Network

✧ Related survey items: Developer Survey (ref:4.2.4.4) (A) 5
✧ Related job statistics: Node usage on Fugaku, B/F communication volume
✧ Key findings from developer survey:
- Scale: ~25–64 nodes (100–256 GPUs)
- Medium scale: 100–300 nodes
- Overall trend: Many applications scale up to pod limits (≤576 GPUs); grand challenge cases use the full system

✧ Key findings from job statistics:
- Pod size of 72 GPUs covers ~50% of jobs; 576 GPUs cover ~90%
- Scale-up bandwidth: 0.4–4 TB/s/GPU (0.01–0.1 B/F × 200 PF × 20%)

Job statistics support selecting a pod size of 72 GPUs. Developers aim for larger scales and will need to program for hierarchical network performance. If NVL72 hardware is available, more detailed evaluation would be possible.

✧ Recommendation: B3

Codesign Item 4: Scale-out Network

✧ Related survey items: Developer Survey (ref:4.2.4.4) (A) 7
✧ Related job statistics: B/F communication volume
✧ Key findings from developer survey:
- Bandwidth is most important
- Latency is important for allreduce/inner-product operations, 3D FFT, and small messages; the system should avoid throughput degradation due to many concurrent small transfers

✧ Key findings from job statistics:
- Scale-out bandwidth: 600 GB/s/GPU (0.3 B/F × 10% × 200 PF × 20%)
- 20% → optimistic compute efficiency
- 10% → estimated off-pod communication ratio

Analysis suggests 600 GB/s is sufficient. Further analysis is needed on the types of off-pod (long-distance) communication.

✧ Recommendation: B

Codesign Item 6: GPU Memory

✧ Related survey items: Developer Survey (ref:4.2.4.4) (A) 3, 4
✧ Related job statistics: Fugaku memory use data
✧ Key findings from developer survey:
- 128–256 GB per GPU covers many real-world cases
- High capacity: 512 GB–1 TB required for large arrays, FFT stacks, replica/ensemble, super-particles, sparse direct solvers

- Outliers: >1 TB required for extreme cases such as global LES, very large MD/sparse problems, even several TB in extreme cases
- Majority view: Increasing bandwidth cannot compensate when working set exceeds device memory

✧ Key findings from job statistics:

- 730 GB/GPU covers 50% of jobs; 1600 GB covers 90

Both statistics and survey suggest large memory is required. However, this must be considered carefully. The study only evaluated total memory requirements, while the key factor is memory required by a single bandwidth-bound kernel. Data used only in other kernels can reside in CPU memory. Also, users were asked to think in FP64 FLOPS, which may lead to overestimation of performance. If asked in terms of memory bandwidth, users might reduce problem size.

✧ Recommendation: A

Codesign Item 7: CPU SVE Performance

✧ Related survey items: Developer Survey (ref:4.2.4.4) (A) 1

✧ Related job statistics: None

✧ Key findings from developer survey:

- CPU kernels will continue to be used in many codes: I/O, compression/decompression, preprocessing/postprocessing, MPI control, low-parallelism kernels, branch-heavy processing, quadruple precision
- Some applications plan CPU–GPU co-execution; others aim for full GPU execution

Additional survey items could include use of CPU SME for AI or CPU usage under GPU memory constraints. Fugaku job statistics should be further analyzed to confirm current SVE usage.

✧ Recommendation: B

System v1.0

Codesign Item 1a: CPU Specification – Matrix Compute Engine

Method: Investigated kernels likely to remain on CPU. Examined their B/F ratios, whether the matrix compute engine can be utilized, whether standard operations can saturate cores or if reducing core count is acceptable, evaluated usefulness of optional features for AI, and assessed emulation performance outlook.

- Survey on CPU-resident kernels and B/F ratios (Developer Survey (ref:4.2.4.4) (B) 1).
- Follow-up survey on matrix compute engine (Developer Survey (ref:4.2.4.4) (B) 2)
- Requested AI team to evaluate usefulness of optional features.
- Requested numerical library team to evaluate emulation performance.

✧ **Key findings from the investigation:**

Developers still do not have a clear view of how to utilize the CPU matrix compute engine. The main reason is that most target applications do not use dense matrix multiplication as a primary kernel. Some CPU-resident kernels were identified as vectorizable, but few had high computational intensity. It was also suggested that CPU could be used when data must be placed close to the network, but this depends on the scale-up network design

✧ **Key findings from AI team:**

The AI team investigated numerical precision used in deep learning applications to understand the need for optional features. They found that FP32/FP64 are primary precisions for some HPC-oriented tasks, while FP16/BF16 are required as fallback for training and inference. FP8 is becoming the primary standard for both training and inference. INT8 is essential for compatibility across existing systems and vendors. FP6 and FP4 were identified as promising future options. Higher precision accumulation remains necessary for training. Mixed precision is widely used in transformer operations. Overall, there was little evidence supporting the usefulness of options such as F16F16

or B16B16, and no use cases were identified for I64I64. In contrast, FP6 and FP4 options showed meaningful use cases.

- ✧ **Key findings from numerical library team:**
  A performance model for floating-point emulation on FUJITSU-MONAKA-X was developed by the numerical library group. For single and double precision emulation, the Ozaki method using Int8 is recommended. For matrix size 16384, emulated FP64 performs better than native Option A. For matrix size 2048, about 20% performance degradation is expected compared to native Option A peak performance.
- ✧ **Recommendation: B**

Codesign Item 1b: CPU Specification – Memory

Method: Modeled total CPU+GPU memory requirements per node and evaluated memory size and bandwidth (B/F) required by CPU-side kernels.

- Survey on total memory requirements (Developer Survey (ref:4.2.4.4) (B) 3).
- Survey on CPU kernel-specific memory requirements (Developer Survey (ref:4.2.4.4) (B) 4).

- ✧ **Key findings from developer survey:**
  For many applications, CPU memory requirements were small because developers plan to move almost everything to GPUs. However, there were exceptions with significantly different memory requirements, including very large capacities. Developers still require large memory on GPUs, which drives overall system memory demand. Concerns were raised that additional CPU memory may be required when staging data on CPU before sending it to the network.
  If GPU memory capacity decreases, CPUs with larger memory may become more important. Developers suggested imposing memory bandwidth limits on CPU kernels, but these kernels are not expected to dominate overall cost. These design options are not fully fixed, and since adding memory capacity later is relatively easy, Option A may be a cost-effective choice.
- ✧ **Recommendation: A**

Codesign Item 2: CPU-GPU Interconnect

Method: Evaluating with unoptimized applications may be misleading. Instead, evidence based on NVIDIA-optimized applications was requested. NVIDIA provided advantages of NVLINK and performance-related information.

- ✧ **Key points from NVIDIA report:**
  NVIDIA presented applications where NVLINK provides performance advantages, including simulation applications, deep learning, and HPL benchmarks. However, due to the complexity of modeling CPU and GPU simultaneously, explicit comparisons between options were not provided. Some applications require more memory than GPU provides, and high-speed interconnect allows data to reside on CPU. Examples comparing cache-coherent vs managed memory performance were also shown (e.g., NEMO). NVIDIA noted that benefits of cache coherence are difficult to isolate but are mainly beneficial for fine-grained tasks. High-speed interconnect and unified memory models also help achieve speedup in early porting stages.
  Since EEA applications are not yet GPU-optimized, modeling performance differences between the two options is difficult. Both NVIDIA and RIKEN currently lack methodology to simulate performance across different interconnect speeds. Therefore, codesign decisions cannot be made purely based on specific application requirements. Developers expressed strong preference for high-speed interconnect. NVIDIA data suggests that even optimized applications require high-speed interconnect for various tasks. The benefit is also tied to GPU memory choices, providing flexibility if smaller GPU memory is selected. Most developers prefer explicit control of data movement, but some (e.g., SALMON developers) rely on unified models and are concerned about performance

and code quality degradation without coherent memory.

✧ **Recommendation: A**

Codesign Item 7: CPU SVE Performance

Method: Identified kernels running on CPU and analyzed required SVE performance. Examined FLOPS utilization relative to peak vector FLOPS ("Vector Pipeline Business") and considered latency differences from A64FX.

- Survey on CPU kernels (Developer Survey (ref:4.2.4.4) (B) 1).
- Job statistics on FLOPS utilization.

✧ **Key findings from job statistics:**
The method evaluates current SVE efficiency on Fugaku. If peak FLOPS are rarely achieved, CPU kernels remaining in FugakuNEXT are unlikely to be compute-bound.

✧ **Analysis of FY2024 Fugaku job mix (jobs >1 hour, total 3,129,958 jobs):**
Most jobs run below 20% of peak FP performance.
Only 734 jobs exceed 30%, and only 93 exceed 50%.

✧ **Further analysis (FS2020 data):**
No application satisfies both conditions: FP efficiency >30% and memory bus utilization <60%.

✧ **Examples:**
GENESIS: FP ~4%, memory ~5%
SCALE: FP ~3.71%, memory ~9.22%
LETKF: FP ~0.07%, memory ~0.10%
LQCD-DWF-HMC: FP ~12.5%, memory ~67%
FFB: FP ~11.51%, memory ~43.09%
EbE-method: FP ~6.20%, memory ~14.53%
SALMON (ground state): FP ~3.94%, memory ~9.77%
SALMON (real-time): FP ~7.32%, memory ~8.11%
From Fugaku job data, most workloads are memory bandwidth-bound. Developer surveycs also indicates CPU-resident kernels have low arithmetic intensity. Therefore, 2-flow is considered the optimal option.

✧ **Recommendation: A**

Other Feedback

The importance of 512-bit SVE vectors in FUJITSU-MONAKA-X CPU was evaluated. Due to ongoing GPU porting efforts, it is difficult to predict future CPU bottleneck kernels. Instead, selected EEA codes were executed and analyzed. On Fugaku, SVE width can be manually reduced from 512-bit to 256-bit using the -Ksimd_reg_size=256 option, with appropriate intermediate setup. Some applications showed performance improvement with 512-bit SVE compared to 256-bit. However, results include effects from compiler optimizations, and the impact of SIMD width cannot be isolated precisely.

3.2.4.4Developer Survey

Developer Survey (A) Questionnaire

1. CPU Design: What do you expect from the CPU of FugakuNEXT? In FugakuNEXT, do you expect that kernels executed on the CPU will remain in your application? Also, are there examples in your field of codes that utilize both CPU and GPU simultaneously?
2. As one possibility, adding a matrix computation engine to the CPU is being considered. In some applications, it has been shown that this engine can be effective by running heavy computational kernels on the GPU while executing AI on the CPU, thereby avoiding data swapping. Do you have any ideas for utilizing a CPU-based matrix computation engine in your application? If so, do you have any requirements regarding the numerical precision that the matrix engine should support (e.g., half precision, single precision, double precision)?

3. Assume a system where each GPU has approximately 200 TFLOPS in FP64 and 10 PFLOPS in FP16. How much memory would be required to run your application on such a GPU? Please provide specific values such as 128GB, 256GB, 512GB, 768GB, or 1TB. Here, assume that the CPU side has a large amount of memory. In addition, please describe any constraints that determine the required memory capacity (e.g., existence of large data structures that are difficult to distribute across GPUs).
4. Related to the above question, we ask about the trade-off between memory capacity and bandwidth. Suppose that GPU memory size decreases while bandwidth increases significantly. If your application has strict memory capacity constraints, would a substantial increase in bandwidth help avoid issues related to memory capacity?
5. Again, assume the GPU performance described in Question 3. Suppose FugakuNEXT has 13,600 of these GPUs distributed across 3,400 nodes (4 GPUs per node). In FugakuNEXT, it is being considered to form groups called “pods” consisting of 4 to 576 GPUs, where communication within a pod is connected by a scale-up network that is faster than the scale-out network across pods. When determining the size of a pod, the number of nodes typically used in practical applications is important. In your application, how many nodes do you typically expect to use?
6. Regarding the coupling between CPU and GPU, NVLINK Fusion and PCI Express are being considered. An advantage of NVLINK is that memory between CPU and GPU is coherent. Do you think this feature would significantly improve the efficiency of developing your application?
7. Regarding the system network, both bandwidth and latency may affect application performance. In your application, is latency important?
8. In HPC, depending on the code, limiting factors such as memory bandwidth, cache size, and computation time differ. In your scientific domain, are there applications that are not constrained by memory bandwidth?
9. In HPC simulation applications in your field, are there cases where AI tasks constitute the most computationally expensive part?

Developer Survey (B) Questionnaire

1. In the previous survey, we asked about expectations for the CPU. This time, we ask about computationally expensive kernels that are expected to remain on the CPU. What is the approximate byte/flop ratio of these kernels? Also, is vectorization of these kernels possible?
2. Previously, we asked whether you had ideas on how to utilize a matrix computation engine on the CPU. If you have come up with any new ideas since then, could you share them? In the previous question, we provided one example where simulation is executed on the GPU while AI runs on the CPU. Another possible case is when a kernel includes matrix operations, but the associated data is not desirable to move to the smaller memory on the GPU.
3. In the previous survey, we asked how much memory is required per GPU. This time, we would like to ask how much memory is required for the entire node (CPU + GPU). Please assume that each node has 4 GPUs. Previously, estimates were made assuming that each GPU has 200 TFLOPS performance. However, the code may be limited not by FP64 compute performance but by memory bandwidth. Therefore, in this assumption, please assume that the memory bandwidth per GPU is 50 TB/s.
4. We assume that CPU memory can be used for the following two purposes:
5. Storing data that will later be transferred to the GPU for processing
6. Storing data that will be processed on the CPU side
7. Consider computationally expensive kernels that are expected to remain on the CPU. How much memory capacity is required for the data processed by these kernels (use case 2 above)?

3.2.5 Early Application Evaluation Results

Based on the application performance estimates reported by NVIDIA in the Basic Design Report, performance ratios relative to Fugaku were calculated. The Fugaku performance values used in the ratio calculation were measured by RIKEN during the basic design phase. Among the six EEA1 applications, E-Wave was not measured on Fugaku in the basic design phase; therefore, results for the remaining five applications are shown below.

In NVIDIA's performance estimates, three types of memory configurations and twelve types of network configurations are assumed for the FugakuNEXT hardware. The impact of network configurations is evaluated only for LQCD-DWF-HMC, while the impact of memory configurations is evaluated for all applications.

3.2.6 Development of New Applications for the FugakuNEXT Era

3.2.6.1AI Applications

FugakuNEXT is expected to become a state-of-the-art computer for artificial intelligence research. With a design based on the latest AI accelerators, significant acceleration is anticipated for AI training and inference tasks compared to Fugaku. The growth rate of AI computational capability is expected to be significantly higher than that of conventional FP64 operations, making the integration of AI methods essential for simulation application developers to achieve revolutionary advances. The goal of the application domain is to anticipate future AI use cases on FugakuNEXT and support system readiness for them.
Most of the EEA applications do not incorporate AI as a computational bottleneck. Unlike HPC, the AI domain tends to rely on standard libraries rather than highly customized code. In addition, because this field is rapidly evolving, it is not realistic to directly select specific AI EEA applications in the same manner as simulation codes. Instead, we attempted to understand how new patterns will emerge and how they will impact system design. Furthermore, since NVIDIA is expected to internally focus on tuning for commercial AI use cases (e.g., LLM inference), we considered that the initial focus should be placed on AI4Science and AI-HPC integration.

Codesign Key Points
The following are key codesign considerations related to AI applications:

- GPU memory capacity
- Scale-up network size
- Scale-out network performance
- Ratio of FP64 vector performance to AI compute performance
- System heterogeneity
- Necessity of dedicated nodes

Both co-location (placing AI and HPC workloads on the same node) and disaggregated deployment (separate partitions for AI and HPC connected via network) are important system configuration patterns and must be equally considered. When partitions are separated, software can compensate for hardware limitations, whereas tightly coupled configurations require careful hardware design. The coupling between scale-up network size and GPU memory capacity must also be considered. In addition, the benefits of “AI as a Service,” where users interact with the system in specialized ways or execute on specialized nodes, should be taken into account, as this may influence user interaction strategies (e.g., using Kubernetes instead of SLURM).

List of Computational Patterns:
Invocation of inference within HPC applications:

- A key challenge is achieving interoperability between AI libraries and HPC codes without overhead
- Codesign point: In co-location, sufficient GPU memory capacity is required. The balance between FP64 vector performance and AI matrix performance is important.
- Codesign point: In dedicated partitions, low-latency scale-out networks are required.

Invocation of training within HPC applications:

- Training completes faster than running HPC and simulation as separate processes.
- Codesign point: This operation is throughput-oriented and prioritizes throughput over latency. Very high scale-out bandwidth is required when using dedicated partitions.
- Invocation of HPC applications from inference:
- This pattern is synchronous and likely requires tight coupling
- Codesign point: GPU memory capacity and the balance between FP64 vector performance and AI matrix performance

- Invocation of HPC applications from training:
- Efficient coupling is required to keep training epochs short and achieve low latency.
- Codesign point: GPU memory capacity and the balance between FP64 vector performance and AI matrix performance.
- AI inference as a service:
- Provides model catalogs or custom models and enables interaction different from batch scheduling.
- Consideration: Is the cost of model switching acceptable when users access large model catalogs?
- Codesign point: System heterogeneity, GPU memory capacity, and scale-up network size.
-

AI training as a service:

- Hyperparameter optimization with small models is important and requires strong scaling.
- Codesign point: System heterogeneity, GPU memory capacity, and scale-up network size.
- AI fine-tuning as a service:
- Codesign point: System heterogeneity, GPU memory capacity, and scale-up network size

AI coding:

- More critical for system software and resource management than for codesign itself.
- Codesign point: Assuming thousands of agents executing code in parallel, high CPU performance becomes most important.
- Integration of agentic AI and scientific workflows:
- The exact form of realization is not yet clear.
- Codesign point: CPU may play a critical role as the orchestrator of AI tasks.

“End-to-end” AI inference:

- Are there scenarios where users do not use AI inference as a service?
- What differs between LLMs and AI4Science models (model size, context window size, prompt caching, etc.)?
- Codesign point: GPU memory capacity and scale-up network size.
- “End-to-end” AI training:
- Are there cases where users prefer to build models in custom environments rather than using training-as-a-service?
- Codesign point: Is the entire system used to build foundation models? How does this differ from LLM training?

Future Work

In this fiscal year, AI usage patterns on FugakuNEXT were investigated, and their impact on design decisions was examined. As future codesign feedback, each of these use cases needs to be further evaluated while considering trade-offs. These efforts will also focus on identifying AI applications that can be directly integrated into benchmark sets. Examples include MILP with LAMMPS or GROMACS, weather prediction with NICAM, and event detection in operations using vision transformers. These activities will continue through discussions with the broader community via SubWGs to anticipate emerging AI applications and use cases.

3.2.6.2Utilization of Low-Precision Arithmetic Units

The increasing demand for computational resources to execute deep learning has had a significant impact on the hardware domain. In recent years, accelerators have been equipped with dedicated matrix operation engines (such as Tensor Cores) optimized for deep learning computations, which exhibit characteristics substantially different from those of conventional HPC-oriented hardware. In particular, they demonstrate extremely high efficiency for low-

precision computations (FP16, FP8, INT8). Hardware vendors are increasingly prioritizing the optimization of these low-precision operations over FP64/FP32 precision, which has been widely used in computational science. Therefore, leveraging these new technologies is an important strategy for achieving dramatic performance improvements in FugakuNEXT.

GPU Model	FP64 (Vector)	FP64 (Matrix)	FP16 (Matrix)	INT8 (Matrix)
A100 SXM	9.7 TFLOPS	19.5 TFLOPS	312 TFLOPS	624 TOPS
H100 SXM	34 TFLOPS	67 TFLOPS	990 TFLOPS	1979 TOPS
B200 HGX	37 TFLOPS	37 TFLOPS	2250 TFLOPS	4500 TOPS

Table：Comparison of performance of high-end GPUs across generations. The processing throughput of matrix operations is shown without considering sparsity.

The development of low-precision hardware has wide-ranging impacts. First, for applications that can be reduced to matrix multiplication, it is important to establish strategies to overcome finite-precision computation. For compute-bound codes that cannot be treated as dense linear algebra, co-design of hardware and algorithms is required. For methods that are not compute-bound, advances in low-precision numerical methods provide approaches to reduce memory bandwidth and communication costs. All of these efforts are supported by the development of advanced programming models capable of efficiently handling low-precision data types.

Low-Precision Computation and Co-Design

The greatest opportunities exist in applications based on dense linear algebra. Developers have long recognized that expressing algorithms in the form of matrix multiplications enables extraction of maximum performance from hardware. Accordingly, there are many state-of-the-art HPC codes in which matrix multiplication constitutes the bottleneck. As indicated by the studies in SubWG2, even tensor contraction-based methods, particularly in physics and chemistry, present opportunities to transition toward matrix multiplication-centric approaches. These methods will likely become increasingly prevalent, as their computational cost is reduced on modern hardware. Errors arising from low precision can be mitigated using emulation algorithms such as the Ozaki scheme, the Markidis method, or BF16×9. However, it remains an open question whether emulation is practically efficient when only extremely low precisions such as FP4 or FP6 are available. Another issue identified through discussions with developers is the importance of establishing emulation techniques for processing a large number of small matrix multiplications in parallel.

Another class of compute-bound applications dependent on matrix multiplication is deep learning. Low-precision strategies are widely recognized in this field, and the preference for precisions such as BF16, FP8, and INT8 has been largely driven by the development of large language models. In computational science, however, many models still rely on FP64 or FP32. For example, SubWG2 investigated the precision requirements of machine-learning-based interatomic potentials and found that while FP32 and TF32 are used, other precision settings have rarely been explored. AI models in scientific domains tend to be relatively small and require highly accurate numerical predictions, necessitating high-precision floating-point computations such as FP32 or FP64. From a co-design perspective, GPU co-design is essential to ensure efficient execution of these models. On the application side, it is important to validate floating-point emulation and explore algorithms that can utilize low-precision types.

Many codes cannot easily exploit Tensor Cores. Not all compute-bound applications can be reduced to dense linear algebra; the GENESIS molecular dynamics code is one such example among the EEA. In certain problem settings, parts of these codes may utilize Tensor Cores (for example, by using the fast multipole method instead of Fourier transforms). However, this does not constitute a general solution. The FugakuNEXT development team must carefully determine

the balance between FP64/FP32 vector performance and AI matrix computation performance to properly support these applications.

Most EEA applications are memory bandwidth-bound, and some are highly sensitive to network latency. Historically, improvements in memory performance have lagged behind increases in computational performance. Therefore, even if FP64/FP32 vector performance is significantly reduced, overall throughput may still improve through increased memory bandwidth. The objective is to enhance memory and network performance by using lower-precision numerical formats (such as FP16 and BF16). Traditionally, most GPUs have natively supported only FP32, and even today, consumer hardware offers limited FP64 performance. Based on prior studies, application developers can adopt reduced-precision methods (with successful examples reported in molecular dynamics). With advances in AI technologies, programming languages supporting low-precision data types have also evolved. In addition, new tools (e.g., RAPTOR from R-CCS) have emerged to automatically profile precision requirements within code.

Future Work
The ratio between FP64/FP32 vector performance and AI low-precision matrix performance is a key co-design item planned for the next fiscal year. To provide feedback on this item, surveys will be conducted among SubWG members and application developers to understand the characteristics of compute-bound applications and how other research groups are utilizing low-precision computation. In collaboration with the numerical library team, performance improvements from applying floating-point emulation will be estimated using codes based on dense linear algebra. Representative use cases of AI applications to be analyzed will also be identified. Furthermore, compute-bound applications that do not rely on matrix operations will be selected, and the minimum floating-point precision requirements will be experimentally evaluated on hardware with limited FP64 support. During the remaining project period, collaboration with application developers will continue to identify use cases where compute-bound or memory bandwidth-bound kernels can benefit from low precision, and tuning support will be provided. The techniques and knowledge obtained from these efforts will be shared with the research community to enable computational scientists to fully leverage the benefits of FugakuNEXT.

3.2.6.3Quantum Computing

Quantum–classical hybrid computing has recently emerged as an approach to solving problems that are difficult for classical computation, such as those represented by the sign problem, by incorporating quantum computation. This trend is expected to continue and further develop in the FugakuNEXT era, and therefore it is necessary to include it as a category of target applications for evaluation. In the next fiscal year, it will be considered to either add a group to existing application SubWGs or establish a new SubWG to advance the addition and examination of target applications.

3.2.7 External Collaboration in Application Development

3.2.7.1 Interviews on the Future Enabled by "FugakuNEXT"

We have been continuously conducting interviews with leading experts in fields that utilize HPC, regarding science and society in the 2030s enabled by FugakuNEXT. We have obtained a wide range of insights, including recommendations on hardware architecture, proactive use of AI in simulations, promotion of open standards, real-world applications anticipating social and industrial impact, human resource development to support GPU porting and tuning, and the formation of a sustainable ecosystem. These insights are being leveraged in the development of FugakuNEXT.

Domain	Affiliation / Organization	Title	Name
Life Sciences	RIKEN, BDR, Multimodal AI Infrastructure Technology Research Team	Team Director	Ryosuke Kojima
Materials and Energy	Graduate School of Science, The University of Tokyo	Professor	Shinji Tsuneyuki
Weather and Climate	National Institute for Environmental Studies, Environmental Information Department (Planning and Support Division), Research Information Office	Acting Head	Hisashi Yashiro
Earthquake and Tsunami Disaster Prevention	Earthquake Research Institute, The University of Tokyo, Earthquake and Tsunami Disaster Prediction Research Center	Professor	Tsuyoshi Ichimura
	Japan Agency for Marine-Earth Science and Technology (JAMSTEC), Research Institute for Marine Geodynamics, Earthquake and Tsunami Prediction Research Center	Director	Takamine Hori
Manufacturing	Nihon University	Research Professor	Chikayuki Kato
	Kobe University	Professor	Makoto Tsubokura
	Tohoku University	Professor	Soji Kawai
Fundamental Science	High Energy Accelerator Research Organization (KEK), Institute of Particle and Nuclear Studies, Theory Center	Director / Professor	Shoji Hashimoto
	University of Tsukuba	Professor	Takenori Ohsu
Digital Twin	Graduate School of Information Science, Osaka University, Department of Information Networking	Professor	Hirozumi Yamaguchi

3.2.7.2 Hosting of the "FugakuNEXT" Application Seminar

"FugakuNEXT," currently under development as the successor to Fugaku, is expected to serve as a next-generation computational infrastructure supporting Japan's scientific and technological capabilities and industrial competitiveness. To realize this vision, further advancement of HPC applications and the development of human resources to support them are essential. Under this background, the Next-Generation Computational Infrastructure Application Development Unit plans seminars to broadly share the latest trends and practical knowledge—including AI utilization—in HPC application development for the FugakuNEXT era. In FY2025, the following seminars were conducted, with a total of approximately 1,000 participants.

No.	Date	Speaker	Title	Participants
1	May 28	Yasumichi Aoki	Application Development Area Kickoff	–
2	July 17	Yoshifumi Nakamura	Beyond CI/CD: Continuous Performance-Driven Principles for Future HPC Systems and Applications	65
3	July 31	Mohamed Wahib	Vision AI for Science and Engineering Applications	76
4	Aug 20	Katsuhisa Ozaki	Ozaki Scheme I: Emulation Method for Matrix Multiplication	72
5	Aug 27	William Dawson	Applying the Ozaki Scheme to HPC Applications	70
6	Sep 25	Daichi Mukogi	Challenges and Prospects in Automatic Generation of HPC Codes Using Generative AI	95
7	Oct 2	Hitoshi Murai	Programming Environment for FugakuNEXT	123
8	Oct 21	Takashi Shimokawabe	Miyabi: JCAHPC's New GPU-Based Supercomputer and Application Porting Activities to GPU	65
9	Oct 30	Kohei Fujita	Simultaneous Use of CPU and GPU for Fast and Energy-Efficient Implicit Wave Simulation	77
10	Nov 10	Ryo Yoshida	Integration of Simulation and the Real World through Sim2Real Machine Learning: Toward the Discovery of New Materials	85
11	Dec 2	Andreas Herten	Building JUPITER: From Procurement to Deployment	67
12	Jan 21	Ryosuke Kojima	Development of Multimodal AI Frameworks and Foundation Models in Life Sciences	61
13	Feb 18	Keiya Hirashima	Star-by-Star Simulations of the Milky Way Accelerated by AI Surrogate Modeling	60
14	Feb 26	Tomonori Shirakawa	Quantum-HPC Application Development for Quantum Many-Body Systems: Selected-Basis Diagonalization and Tensor-Network Approaches	64

In addition, in collaboration with the public relations division, information is disseminated via the official website of the RIKEN Center for Computational Science:
Japanese: https://www.r-ccs.riken.jp/fugaku-next/app-seminar/
English: https://www.r-ccs.riken.jp/en/fugaku-next/app-seminar/
Seminars will also be held in FY2026 in a similar manner; however, they will be organized around specific themes with a deeper, more focused exploration of each topic. Furthermore, this seminar series has been supported by the Kobe Center of the Foundation for Computational Science (FOCUS), which assisted in disseminating announcements via mailing lists to HPCI account holders. In FY2026, the seminars will be promoted more broadly to enhance awareness of FugakuNEXT and to contribute to strengthening application development capabilities.

3.2.7.3Domestic collaboration

In the activities of the application area of this project, collaboration with potential user application developers of FugakuNEXT is essential. As described in the development framework, the Application Working Group has established a collaborative structure with the domestic HPC application community, and initial evaluation target applications were defined through these activities.

In the latter half of the fiscal year, HAIRDESC (Center Director: Taeyoo Park) was established as

a new center within RIST. In addition, the Grand Challenge Program—successor to the current MEXT program for accelerating outcome creation—was announced with a planned start in FY2026. HAIRDESC has been assigned to support and accompany the GPU porting of applications under this program.
RIKEN is not only a collaborating institution of HAIRDESC, but is also working toward tighter collaboration through regular review meetings, with the goal of incorporating applications aiming to use FugakuNEXT under the Grand Challenge Program into the co-design process.

3.2.7.4International collaboration

International collaboration in application development is important both for the global expansion of the FugakuNEXT-centered application ecosystem and for constructing a system capable of efficiently leveraging internationally proven and widely demanded ecosystems.
As part of this effort, a hackathon utilizing Benchpark—a CI/CB framework for EEA developers—was held at the RIKEN Center for Computational Science in collaboration with developers of Benchpark from Lawrence Livermore National Laboratory (LLNL).
Going forward, collaboration will be expanded beyond LLNL to include U.S. DOE laboratories and other international partners. Through these efforts, we aim not only to advance the development of FugakuNEXT but also to establish an environment that benefits its users.

3.3 System software

3.3.1 Programming Environment

3.3.1.1 System, CPU Part

3.3.1.1.1 Compiler, Programming Languages

3.3.1.1.1.1 Programming Languages

In applications in the HPC domain, C, C++, and Fortran are mainly used as programming languages. In order to achieve performance, compilers for these languages are important. In "FugakuNEXT," compilers that support these programming languages will be provided. The compilers for the CPU part will also support programming language extension specifications such as Arm C Language Extensions (ACLE) and OpenMP. The versions of supported language standards and extension specifications depend on the compilers described later.
In applications in the AI domain, in addition to these, Python is mainly used. In order to achieve performance, libraries for numerical computation for Python are important. The Python interpreter and its libraries will be examined in the detailed design in collaboration with other WGs.

3.3.1.1.2 Compilers

3.3.1.1.2.1 Complier for CPU

In "Fugaku," the compute node had only a CPU (A64FX) as the computing device. In order to maximize the functionality and performance of A64FX, a dedicated compiler was developed and provided to users instead of using existing open-source or commercial compilers. In "FugakuNEXT," there will be two types of computing devices: CPU (FUJITSU-MONAKA-X) and GPU (manufactured by NVIDIA). For compiling code executed on the GPU part, the NVIDIA HPC SDK (including the NVIDIA CUDA Toolkit here) will be provided in order to maximize GPU functionality and performance. For compilers used to compile code executed on the CPU part, it is necessary to determine what to provide and how to realize it. The following describes these.

3.3.1.1.2.1.1 Candidate compilers

In programming code executed on the GPU part, there are two methods: one is to separate the processing executed on the GPU part from the CPU part at the level of functions/subroutines and describe it in a dedicated programming language (CUDA code), and the other is to describe the processing executed on the GPU part by embedding it within the CPU-side processing using directives or C++ templates (GPU offloading code).
Compilers included in the NVIDIA HPC SDK can compile not only CUDA code and GPU offloading code but also CPU code. In addition, there exist LLVM- and GCC-family compilers as open-source compilers widely used in the AI/HPC domain, which can compile not only CPU code but also GPU offloading code. Furthermore, it is also conceivable to develop and provide a dedicated compiler in order to maximize the functionality and performance of FUJITSU-MONAKA-X. This compiler family is tentatively referred to as the "Fujitsu compiler," and the command names for each programming language compiler are tentatively defined as fjclang (for C), fjclang++ (for C++), and fjflang (for Fortran).
These are organized by compilation target code and programming language in Table 3-8.

Table 3-8 candidate compilers

Target compiled code	Compiler type	compiler(command name)		
		C	C++	Fortran
CUDA code	NVIDIA HPC SDK	(nvcc)[1]	nvcc nvc++	nvfortran
GPU offloading code (OpenACC/OpenMP/Kokkos)	NVIDIA HPC SDK	nvc	nvc++	nvfortran
	LLVM	(clang)[2]	(clang++)[2]	(flang)[2]
	GCC	(gcc)[2]	(g++)[2]	(gfortran)[2]
CPU code	NVIDIA HPC SDK	nvc	nvc++	nvfortran
	LLVM	clang	clang++	flang
	GCC	gcc	g++	gfortran
	Fujitsu compiler	(fjclang)[3]	(fjclang++)[3]	(fjflang)[3]

3.3.1.1.2.1.2 Requirements for Feasibility of Compilers for the CPU Part

Here, the requirements for the feasibility of compilers for the CPU part are described, whether existing NVIDIA HPC SDK, LLVM, and GCC can satisfy these requirements in the FugakuNEXT generation, and under what conditions a Fujitsu compiler would be required.
However, the nvcc command included in the NVIDIA HPC SDK internally separates the portions of the source code executed on the GPU part and those executed on the CPU part, and compiles the CPU-side portions using a CPU compiler (host compiler). As the host compiler, nvc++, clang++, g++, etc. can be selected. Therefore, in the following, nvcc is not considered, and instead of the NVIDIA HPC SDK, the three compilers nvc, nvc++, and nvfortran are collectively referred to as the "NVIDIA HPC compilers."
As requirements for the feasibility of compilers for the CPU part in "FugakuNEXT," in addition to the general requirements for C/C++/Fortran compilers for the Arm architecture, the following are mainly required:

- Continuous support for new programming language standards in the front-end
- Support for SVE2/SME2 in Arm C Language Extensions (ACLE) in the front-end
- Support for the instruction set architecture (ISA) of FUJITSU-MONAKA-X in the back-end
- Support for the microarchitecture of FUJITSU-MONAKA-X in the back-end

3.3.1.1.2.1.2.1 New programming language Support

"FugakuNEXT" will be operated over several years, and during that period, programming language standards and programming language extension standards such as C, C++, Fortran, and OpenMP will be updated, and it is expected that software executed on "FugakuNEXT" will also use new language standards.
Therefore, the compilers provided in "FugakuNEXT" are required to be continuously updated to support new language standards even after the start of operation of "FugakuNEXT."
NVIDIA HPC compilers, LLVM, and GCC are, as of the time of writing, continuously updated to support new language standards, and it is expected that they will satisfy this requirement even

[1] nvcc is a compiler for CUDA C++, which is based on C++; however, it is used as C-based within the range where C++ and C are compatible.
[2] The GPU offloading functionality of OSS compilers depends on the OSS support status
[3] Tentative name in case providing a Fujitsu compiler.

after the start of operation of “FugakuNEXT.”

3.3.1.1.2.1.2.2 ACLE Support

ACLE is an extension specification of the C language for the Arm architecture, and various data types, intrinsic functions, macros, etc. continue to be added along with the evolution of the Arm architecture. The implementation status of ACLE differs depending on the compiler and its version.
For “FugakuNEXT,” it is desirable that ACLE supporting data types and intrinsic functions for Scalable Vector Extension version 2 (SVE2) and Scalable Matrix Extensions version 2 (SME2), which are important for software in the AI/HPC domain, is implemented.
In LLVM and GCC, as of the latest versions at the time of writing (LLVM 21.1.8 and GCC 15.2), ACLE supporting these features is implemented. In NVIDIA HPC compilers, in the implementation of ACLE in the latest version at the time of writing (NVIDIA HPC SDK 25.11), SVE2 is supported, but SME2 is not supported.
By adding support for SME2 to ACLE in NVIDIA HPC compilers, this requirement can also be satisfied by NVIDIA HPC compilers.

3.3.1.1.2.1.2.3 ISA Support

FUJITSU-MONAKA-X conforms to the instruction set architecture (ISA) of the Arm architecture. In the Arm architecture, the ISA continues to be enhanced, and support for it is also progressing in each compiler. Compiler support for ISA can be broadly classified into two categories: enabling the use of instructions included in the ISA, and utilizing those instructions for performance optimization.
In LLVM and GCC, Arm Ltd. takes the lead in advancing support for instruction usage. Support for instruction usage mostly involves modifications to architecture-specific parts, and since it has little impact on other architectures, it is rarely rejected by the development community. Therefore, such modifications are often incorporated smoothly.
NVIDIA HPC compilers use an LLVM-based back end and, as of the time of writing, are periodically rebased to newer versions of LLVM. Therefore, if instruction usage is supported in LLVM, it can also be supported in NVIDIA HPC compilers.
In LLVM and GCC, instruction utilization is advanced by sub-communities within the LLVM/GCC development community, including Arm Ltd. Instruction utilization may require modifications not only to architecture-specific parts but also to parts common to other architectures. Therefore, modifications with the following characteristics may be rejected by other members of the development community and not incorporated, due to reasons such as increased difficulty in compiler maintenance and disadvantages outweighing advantages:

- Limited applicability
- Dependence on non-standard specifications
- Optimizations that are not enabled by default
- Optimizations with small performance improvement effects

NVIDIA HPC compilers also have difficulty incorporating modifications that were not accepted into the LLVM development community, in order to facilitate rebasing the back end to new versions of LLVM. Therefore, if functions essential to “FugakuNEXT” cannot be incorporated into any of the above compilers, they will not satisfy the requirements, and a Fujitsu compiler will be required.

3.3.1.1.2.1.2.4 Micro Architecture Support

Support for microarchitecture is similar to ISA support. In LLVM and GCC, modifications that can be made within existing mechanisms or that do not affect other CPUs are likely to be incorporated, whereas modifications that do not meet these conditions may not be incorporated. NVIDIA HPC compilers also have difficulty incorporating modifications that were

not accepted into the LLVM development community. Therefore, if functions essential to "FugakuNEXT" cannot be incorporated into any of the above compilers, they will not satisfy the requirements, and a Fujitsu compiler will be required.

3.3.1.1.2.1.3 Implementation Method of the Fujitsu Compiler

In "Fugaku," a dedicated compiler developed by Fujitsu was provided; however, after the start of operation, version upgrades corresponding to new language standards have not been carried out. In order to perform version upgrades corresponding to new language standards at low cost, it is desirable to base them on existing compilers.

In the survey and research on next-generation computing infrastructure (system research) conducted up to FY2024, LLVM was investigated as a candidate, and it was concluded that LLVM is suitable as the base compiler for "FugakuNEXT." Therefore, if a Fujitsu compiler is required in "FugakuNEXT," it is desirable that its base be LLVM.

Table 3-9 shows the classification of general compiler source/build/distribution methods utilizing open-source software (OSS).

Table 3-9 Classification of General Compiler Source / Build / Distribution Methods Using OSS

	Class	Source / Build / Distribution	Examples of LLVM
1	Official release version	OSS source without modification is built and distributed by the developer	• LLVM release packages https://github.com/llvm/llvm-project/releases/
2	Custom build version	OSS source without modification is built and installed according to the environment	• Fugaku OSS
3	OS distribution version	OS developers apply minimal modifications (e.g., backports) to OSS source, build it to match the OS configuration, and distribute it as standard OS packages	• Red Hat Enterprise Linux 10 (clang/clang++) • Ubuntu 24.04 LTS (clang/clang++/flang-new) • SUSE Linux Enterprise Server 16 (clang/clang++)
4	OSS pre-integration version	Modifications planned to be submitted to OSS are pre-applied to OSS source, built, and distributed	• Clang for NVIDIA Grace (clang/clang++) https://developer.nvidia.com/grace/clang
5	OSS based version	OSS source with proprietary modifications applied is built and distributed	• Arm Toolchain for Linux (armclang/armclang++/armflang) https://developer.arm.com/Tools%20and Software/Arm%20Toolchain%20for%20Linux • AMD ROCm (amdclang/amdclang++/amdflang) https://www.amd.com/ja/products/software/rocm.html • oneAPI (C/C++) (icx/icpx) https://www.intel.com/content/www/us/en/developer/tools/oneapi/dpc-compiler.html

			• Technical Computing Suite (clang mode) (fcc -Nclang/FCC -Nclang) • NVIDIA CUDA Toolkit (nvcc) https://developer.nvidia.com/cuda/toolkit
6	Proprietary compiler	Built independently by utilizing OSS source with proprietary modifications (often non-public) as components, while using separate frontends, etc., and distributed as a standalone compiler	• NVIDIA HPC SDK (nvc/nvc++/nvfortran) https://developer.nvidia.com/hpc-sdk • oneAPI (Fortran) (ifx) https://www.intel.com/content/www/us/en/developer/tools/oneapi/fortran-compiler.html

Table 3-10 Characteristics of Each Classification of OSS-Based Compilers

No.	Classification (for convenience)	Characteristics
1	Official release version	Packages tested by the OSS development community can be used. However, supported platforms and included modules are limited, and may not be optimal for a given environment. (Example: the -mcpu option is not specified during build, leaving room for improvement in compiler performance, i.e., translation performance.)
2	Custom build version	
3	OS distribution version	Can be built according to the target environment. Even if issues occur, the source code remains OSS as-is, making it easier to obtain support from the OSS development community.
4	OSS pre-integration version	Supported by the OS distribution. Some distributions accept feature requests from partners and may backport OSS-submitted fixes to future versions.
5	OSS-based version	Includes only modifications expected to be merged into OSS. Positioned as upstream staging, incorporating bug fixes and partial early feature integrations.
6	Proprietary compiler	Includes proprietary (non-upstream) modifications aligned with specific objectives. May include proprietary libraries. Used when implementing proprietary features, tuning, or integration with custom libraries. Capable of incorporating features rejected by the OSS community. Command names are often changed.
		Enables construction of a highly customized compiler. In most cases, the LLVM backend is used for code generation, but the frontend is proprietary, resulting in incompatibility with LLVM compilers from a user perspective.

If a Fujitsu compiler is required in “FugakuNEXT,” it is in cases where proprietary modifications for ISA support or microarchitecture support—rejected by the LLVM development community—are required for “FugakuNEXT.” Otherwise, it is desirable to use LLVM as-is. Therefore, if a Fujitsu compiler is required in “FugakuNEXT,” it is desirable to adopt the above “OSS-based

version." In addition, when providing a Fujitsu compiler, it is desirable to rebase the Fujitsu compiler in accordance with LLVM version updates.

3.3.1.1.2.1.4 Analysis of needs of Fujitsu compiler

The following disadvantages can be identified when providing a Fujitsu compiler:

- For operators of "FugakuNEXT," maintaining proprietary modifications while performing bug fixes and continuously rebasing to new LLVM versions incurs maintenance and operational costs.
- For users of "FugakuNEXT," the number of compiler options for CPU code increases, making selection and usage more complex.

Since the "Fujitsu compiler" targets CPU code, demand is expected to be low for GPU-centric applications. At the basic design stage, no cases have been identified where CPU cost dominates performance and compiler support becomes a critical issue, making it difficult to assess demand. Therefore, at present, the necessity of a Fujitsu compiler with proprietary modifications applied to LLVM cannot be demonstrated. In the basic design, the Fujitsu compiler is considered unnecessary, and it will be re-evaluated in the detailed design according to requirements. In the detailed design phase, it is desirable to clarify issues through detailed examination of development items and performance evaluation based on the results of each WG's basic design, and to determine the need for the Fujitsu compiler.

3.3.1.1.2.1.5 Summary of CPU compiler

Based on the above, NVIDIA HPC compilers, LLVM, and GCC are highly likely to satisfy the feasibility requirements for CPU compilers, although there remains a possibility that future requirements may not be fully met. In addition, since NVIDIA HPC compilers are based on LLVM, if NVIDIA HPC compilers implement functionality and performance equal to or greater than LLVM, including the frontend, in the "FugakuNEXT" generation, LLVM will be unnecessary.
In conclusion, it is desirable to provide NVIDIA HPC SDK, LLVM, and GCC as shown in Table 3-8 as the compilers for "FugakuNEXT." However, LLVM may be omitted if, in the detailed design phase, NVIDIA HPC SDK is determined to provide sufficient functionality and performance. The Fujitsu compiler is not required in the basic design and will be reconsidered in the detailed design according to requirements. If a Fujitsu compiler becomes necessary, it is desirable to provide it as an "OSS-based version" based on LLVM and to rebase it in accordance with LLVM version updates.

3.3.1.1.3 LLVM Development

In addition to providing LLVM (Clang/Flang) as compilers for CPU code and GPU offloading code, the backend of NVIDIA HPC compilers is based on the LLVM backend and is periodically rebased. Furthermore, if a Fujitsu compiler is provided, it will also be based on LLVM. Therefore, if required functionality for "FugakuNEXT" does not exist in LLVM, development on LLVM will be necessary.
This section describes items that should be examined, designed, and developed for LLVM in the detailed design phase and beyond. It is desirable that these examinations, designs, and developments be carried out in cooperation with the LLVM development community.
In addition to the items described here, feature improvements requested through co-design with applications and compiler issues identified through detailed examination of development items and performance evaluations should also be examined, designed, and developed in cooperation with the LLVM development community in the detailed design phase and beyond.

3.3.1.1.3.1 Front-end

3.3.1.1.3.1.1 Language Standard Support

Table 3-11 shows an overview of language standard support for Clang/Flang planned in LLVM version 22.1.0, scheduled for release in February 2026. The language standards listed include the latest version at the time of writing and the immediately preceding version.

Table 3-11 L Overview of Language Standard Support in LLVM 22.1.0

Language standard		Support status
C	ISO/IEC 9899:2018 (C17)	Almost all features implemented
	ISO/IEC 9899:2024 (C23)	Most features implemented
C++	ISO/IEC 14882:2020 (C++20)	Almost all features implemented
	ISO/IEC 14882:2024 (C++23)	Most features implemented
Fortran	ISO/IEC 1539-1:2018 (Fortran 2018)	Almost fully implemented except for features related to images and coarrays
	ISO/IEC 1539-1:2023 (Fortran 2023)	Under development

An overview of OpenMP standard support for Clang/Flang planned in LLVM version 22.1.0, scheduled for release in February 2026, is shown in Table 3-12.

Table 3-12 Overview of OpenMP Standard Support in LLVM 22.1.0

OpenMP Standard	C/C++ Support Status	Fortran Support Status
OpenMP 3.1	All features implemented	All features implemented
OpenMP 4.0	All features implemented	Most features implemented
OpenMP 4.5	All features implemented	Under development
OpenMP 5.0	Almost all features implemented	Under development
OpenMP 5.1	Most features implemented	Under development
OpenMP 5.2	Most features implemented	Under development
OpenMP 6.0	Under development	Under development

Details of language standard support in the latest LLVM development version are provided at the following URLs:

- C: https://clang.llvm.org/c_status.html
- C++: https://clang.llvm.org/cxx_status.html
- Fortran: https://flang.llvm.org/docs/FortranStandardsSupport.html
- OpenMP (C/C++): https://clang.llvm.org/docs/OpenMPSupport.html
- OpenMP (Fortran): https://flang.llvm.org/docs/OpenMPSupport.html

3.3.1.1.3.1.2 Programming Model Support

In “FugakuNEXT,” the following programming models are planned to be provided for utilizing multicore CPUs and GPUs: OpenACC, OpenMP, Kokkos, standard language parallelism (C++ parallel algorithms and Fortran DO CONCURRENT construct), CUDA C++, and CUDA Fortran. Among these, OpenACC, CUDA C++, and CUDA Fortran are primarily for accelerators such as GPUs.

The compilers included in the NVIDIA HPC SDK support OpenACC, OpenMP, standard language parallelism, CUDA C++, and CUDA Fortran. LLVM supports OpenMP as shown in Table 3-12, and standard language parallelism is being developed within the LLVM development community. Kokkos supports NVC++ and NVCC included in the NVIDIA HPC SDK, as well as LLVM Clang.

Therefore, from the compiler perspective, there are no development items for these programming models. If development is found to be necessary during the detailed design phase,

it will be carried out in cooperation with RIKEN.

3.3.1.1.3.2 Backend

3.3.1.1.3.2.1 FUJITSU-MONAKA-X ISA Support

FUJITSU-MONAKA-X adopts Armv9 as its ISA. The following describes compiler support to leverage ISA features, including extensions. Among the adopted extensions, SVE2 and SME2 are important for performance.

3.3.1.1.3.2.1.1 Support for Extensions

There exist extensions, such as SVE2 and SME2, whose support can be optionally selected depending on the implementation. The compiler must allow users to specify whether instructions for each extension may be generated. LLVM supports such user directives up to the latest Arm extensions.
In addition, when extensions are enabled, optimization implementations to utilize those instructions may be required. For example, in LLVM, development is underway to enable vectorization of loops computing histograms using instructions newly introduced in SVE2.
If application benchmarks indicate that instructions of extensions adopted in FUJITSU-MONAKA-X can be effectively utilized, the addition of such optimizations will be considered.

3.3.1.1.3.2.1.2 SVE2 Support

SVE2 defines a vector instruction set and allows selection of different vector widths depending on implementation. Since it can process multiple times more data simultaneously compared to scalar instructions, it is desirable to use vector instructions as much as possible.
There are two main approaches to using vector instructions: automatic vectorization by the compiler and explicit programming using intrinsics corresponding directly to vector instructions. Most users do not perform explicit vector programming but rely on automatic vectorization, so supporting this is desirable. Automatic vectorization is a complex feature involving data dependency analysis and cost models for selecting vectorization strategies. It is necessary to verify whether optimal instruction sequences can be generated in various cases and improve them if necessary. Specific verification and improvement items will be examined in the detailed design phase and beyond.
For cases requiring vectorization of complex loop structures or sophisticated optimizations such as rearranging vector lanes, explicit vector-aware programming by users is required instead of automatic vectorization. For this purpose, Arm C Language Extensions (ACLE) defines intrinsic functions directly corresponding to SVE2 instructions. Since programming is easier than using assembly language, it is desirable to support this interface.

3.3.1.1.3.2.1.3 SME 2 Support

SME2 is an extension for accelerating matrix operations, with matrix registers and outer-product operations as its main features. The primary usage of SME2 instructions is expected to be implementation of mathematical libraries such as BLAS using low-level interfaces such as intrinsics or assembly. Applications utilize SME2 through such libraries.
As with SVE2, ACLE defines intrinsic functions for SME2, and it is desirable to support them.
In addition, approaches where the compiler automatically generates instructions (similar to automatic vectorization), or where the compiler transforms high-level mathematical representations (such as MLIR linalg dialect), are also possible. These approaches reduce user burden but are complex, and care must be taken to ensure expected results in target use cases.
If these features are required based on application benchmarks, the scope and content of support will be examined according to requirements.

3.3.1.1.3.2.2 FUJITSU-MONAKA-X Microarchitecture support

To fully exploit the performance of FUJITSU-MONAKA-X, microarchitectural information of FUJITSU-MONAKA-X will be incorporated into LLVM. In addition, enhancements and additions of optimization passes will be examined to provide effective optimizations for typical code patterns appearing in AI/HPC applications. Details are described below for each item. If new compiler issues are identified through performance evaluation during the detailed design phase, appropriate measures will be considered.

3.3.1.1.3.2.2.1 -mcpu Option Support (-mcpu=fujitsu-monaka-x)

Microarchitectural information of FUJITSU-MONAKA-X (such as latency and pipeline characteristics) will be added. This information will be used as parameters for various optimization passes, including instruction scheduling and software pipelining described below, enabling generation of binaries tuned for FUJITSU-MONAKA-X. These settings will be enabled when -mcpu=fujitsu-monaka-x is specified.

3.3.1.1.3.2.2.2 Cost model support

Microarchitectural information of FUJITSU-MONAKA-X (such as latency and pipeline characteristics) will be added. This information will be used as parameters for various optimization passes, including instruction scheduling and software pipelining described below, enabling generation of binaries tuned for FUJITSU-MONAKA-X. These settings will be enabled when -mcpu=fujitsu-monaka-x is specified.

3.3.1.1.3.2.2.3 Various optimizations

To maximize the performance of FUJITSU-MONAKA-X in AI/HPC applications, improvements to various LLVM optimizations will be examined. Examples include:

- Loop optimizations
- Instruction scheduling
- Optimization control via pragma directives

The selection of optimizations to be developed and the specific improvements will be examined in the detailed design phase.

3.3.1.1.3.2.3 Runtime

LLVM provides its own OpenMP runtime library. There may be room for tuning, such as memory copy processing between CPU and GPU, for the ISA and microarchitecture adopted by FUJITSU-MONAKA-X. Such tuning will be considered as necessary in the detailed design phase.
In addition, NVIDIA HPC SDK also provides its own OpenMP runtime library. Therefore, as necessary, it will be considered to reflect tuning of the OpenMP runtime library in LLVM into NVIDIA HPC SDK in cooperation with NVIDIA.

3.3.1.1.4 GCC Development

GCC will also be provided as a compiler for CPU code and GPU offloading code. However, since the compilers included in NVIDIA HPC SDK are based on LLVM, the Fujitsu compiler—if provided—will also be based on LLVM, and LLVM provides functionality and performance comparable to GCC, LLVM will be prioritized for development targeting “FugakuNEXT.”
This section describes items to be examined, designed, and developed for GCC in the detailed design phase and beyond.

3.3.1.1.4.1 Backend

3.3.1.1.4.1.1 -mcpu Option support (-mcpu=fujitsu-monaka-x)

As with LLVM, GCC will also support specifying fujitsu-monaka-x via the -mcpu option

3.3.1.1.5 Programming tools

3.3.1.1.5.1 Performance profiler

Performance profilers will be examined in the detailed design phase.

3.3.1.1.5.2 Debugger

Debuggers will be examined in the detailed design phase.

3.3.1.1.6 Other software

Other software related to compilers and programming languages, such as Portable Hardware Locality (hwloc) and numactl/libnuma, will also be examined for development necessity in the detailed design phase in cooperation with other WGs.

3.3.1.2GPU

The Programming Environment Working Group reported on the following items and provided early access to the next-generation Flang-based Fortran compiler currently under development to the FugakuNEXT software study team.

3.3.1.2.1 Various Compilers

The current configuration and support status of NVIDIA HPC components are shown below. Bug fixes and new feature additions are managed based on customer requests and application requirements, and updates for each component are provided through the NVIDIA HPC SDK release process.

3.3.1.2.1.1 Compiler

NVC is a C compiler for NVIDIA GPUs and x86 and Arm CPUs. Based on command-line options, it invokes the C compiler, assembler, and linker for the target processor. It complies with ISO C11 and supports GPU programming via OpenACC and OpenMP offloading, as well as multicore CPU programming via OpenACC and OpenMP

3.3.1.2.1.2 C++compiler (NVC++)

NVC++ is a C++17-compliant compiler for NVIDIA GPUs and x86 and Arm CPUs. Based on command-line arguments, it invokes the C++ compiler, assembler, and linker for the target processor. It supports ISO C++17, C++17 parallel algorithms, and GPU and multicore CPU programming via OpenACC and OpenMP. Support for C++20 and C++23 is currently in progress. In addition, preview support is provided for some features planned for C++26 (reflection, mdarray, submdspan, senders), and final versions will be supported after standardization

3.3.1.2.1.3 Fortran compiler (NVFORTRAN)

NVFORTRAN is a Fortran compiler for NVIDIA GPUs and x86 and Arm CPUs. It fully supports ISO Fortran 2003 and partially supports ISO Fortran 2008, 2018, and 2023. It supports ISO Fortran

parallel features (do concurrent and many array intrinsic functions), GPU and multicore programming via OpenACC and OpenMP, and GPU programming via CUDA Fortran.

3.3.1.2.1.4 NVCC

NVCC supports NVC++, as well as GCC/G++ and Clang with ACLE extensions on Arm. ACLE extensions for GCC are scheduled to be included in the CUDA 13.2 release. Host compiler support for Clang follows official Clang releases.

3.3.1.2.1.5 GCC

GNU Compiler Collection (gcc, g++, gfortran) also supports C, C++, and Fortran and provides an additional compiler option. It supports OpenMP directives for CPU targets and OpenACC and OpenMP offloading for GPU targets.

3.3.1.2.1.6 LLVM

NVIDIA's engineering team is focused on upstreaming optimizations in the intermediate layer of LLVM IR and below, with the goal of broadly improving compiler performance. Specific areas of effort include dependency analysis and loop optimizations.
NVIDIA provides builds based on the latest Clang source tree (top-of-tree builds), enabling performance improvements to be utilized quickly and easily. The Clang compiler supports OpenMP for both CPU and GPU targets, and support for OpenACC is currently in progress.
In addition, NVIDIA actively contributes to the Flang project. Flang is expected to become a compiler that fully supports Fortran 2023, including OpenACC, OpenMP, CUDA Fortran, and parallel features of the Fortran language.

3.3.1.2.2 Support for Fujitsu CPU

NVIDIA HPC compilers (NVC, NVC++, NVFORTRAN) support the FUJITSU-MONAKA-X CPU. NVCC is planned to support the Fujitsu Clang compiler as a host compiler.

3.3.1.2.3 Parallel Programming Models

NVIDIA HPC SDK compilers support multiple interoperable parallel programming models, including directive-based offloading, standard language parallelism, and explicit CUDA programming. The basic policy is to provide flexible interoperability that aligns with users' current environments.

3.3.1.2.3.1 OpenACC

NVIDIA HPC compilers fully support OpenACC 2.7 and partially support versions 3.0 and later based on customer demand. They support GPU offloading and multicore CPU programming.

3.3.1.2.3.2 OpenMP

NVFORTRAN, NVC++, and NVC support a subset of OpenMP features for both CPU and GPU. With a focus on OpenMP 5.0 features, they provide a highly portable and extensible programming environment. Unsupported directives and API calls are ignored as much as possible to ensure portability.
If they cannot be ignored, if the compiler cannot uniquely interpret them, or if execution would be incorrect, appropriate errors are issued at compile time or runtime.
For the latest list of supported OpenMP 5.0 target offload features, refer to:
https://docs.nvidia.com/hpc-sdk/compilers/hpc-compilers-user-guide/index.html#openmp-

subset
Support for OpenMP features in HPC compilers is determined by customer requests and application needs. Therefore, features beyond those currently supported, including those added after OpenMP 5.0, will be evaluated for support based on application requirements.

3.3.1.2.3.3 Kokkos

The use of abstraction programming models such as Kokkos enables improved productivity and portability. A mature CUDA backend is provided and continuously validated with the NVIDIA software stack. NVIDIA actively collaborates with the Kokkos project to enable support for NVIDIA GPUs.
An experimental OpenACC backend also exists. Multiple CPU backends, including OpenMP and C++ threads, are supported, enabling single-source portability across CPU and GPU. NVIDIA feeds back multiple improvements related to the CUDA backend upstream to improve performance on GPUs.

3.3.1.3Advanced Components

3.3.1.3.1 Study of Programming Models for GPUs

Currently, in the HPC domain, the following GPU programming models are widely used:

- **OpenACC**
  OpenACC is a directive-based programming model for heterogeneous computing. It was proposed in 2011 by Cray, CAPS, NVIDIA, and PGI, but as of 2026, support is effectively continued only by NVIDIA.
- **OpenMP**
  OpenMP is a directive-based API for shared-memory parallel programming, proposed in 1997 by a consortium of multiple HPC vendors. Later, in version 4.0 (2023), offloading functionality for accelerators was added.
- **Kokkos**
  Kokkos is a C++ library for performance portability targeting CPUs, GPUs, and other accelerators. Development started in 2012 at Sandia National Laboratories under the U.S. Exascale Computing Project, and since 2025 it has been developed under the support of the High Performance Software Foundation under the Linux Foundation.
- **Standard Language Parallelism**
  Here, syntax provided by programming language standards for describing loop parallelism is referred to as *standard language parallelism*. Examples include C++17 and later parallel algorithms (stdpar) and the DO CONCURRENT construct in Fortran 2008 and later.

The results of comparing OpenACC, OpenMP, CUDA, Kokkos, and standard language parallelism in terms of performance, ease of programming, and portability are shown in the table below.

Programming Model	Performance	Ease of Programming	Portability	Remarks
OpenACC	△	○	△	
OpenMP	×	○	○	Performance is expected to improve as compilers mature.
CUDA	○	×	×	
Kokkos	△ (depends on backend)	△	○	
Standard Language Parallelism	×	○○	○	

Considering the support status on NVIDIA GPUs, performance, and future prospects, FugakuNEXT adopts OpenACC as the primary GPU programming model, and also recommends Kokkos. In addition, other programming models will also be supported.
In addition to the above, attention will be paid to future trends of SYCL [0], an open standard for programming heterogeneous systems, as well as Julia [1], which has attracted attention in recent years, and its accelerator extension, Julia for Accelerators (JACC) [2].

[0] https://www.khronos.org/sycl/
[1] https://julialang.org/
[2] Pedro Valero-Lara, William F. Godoy, Het Mankad, Keita Teranishi, Jeffrey S. Vetter, Johannes Blaschke, and Michel Schanen. 2025. JACC: Leveraging HPC Meta-Programming and Performance Portability with the Just-in-Time and LLVM-based Julia Language. In Proceedings of the SC '24 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis (SC-W '24). IEEE Press, 1955–1966. https://doi.org/10.1109/SCW63240.2024.00245

3.3.1.3.2 Kokkos

F2Kokkos

As described above, Kokkos targets C++ and therefore cannot be directly applied to Fortran. To address this, the Fortran Language Compatibility Layer (FLCL) [3] of Kokkos provides functionality mainly for converting Fortran arrays into Kokkos multidimensional views. By using FLCL, users can replace loops in Fortran programs with calls to functions written in Kokkos.

[3] https://github.com/kokkos/kokkos-fortran-interop

Based on FLCL and the Omni compiler infrastructure [8], which is a source-to-source compiler framework previously developed by RIKEN, a tool called F2Kokkos is developed to enable interoperability between Fortran and Kokkos. An example of using F2Kokkos is shown below.
The user inserts a parallel_for directive immediately before the target loop (Figure 3-13).
F2Kokkos transforms the target loop (Figure 3-14) and generates Kokkos code (Figure 3-15) and an interface (Figure 3-16).

[8] https://omni-compiler.org/

```
subroutine sub
real :: x(xil:xiu,xjl:xju), y(yil:yiu,yjl:yju), a
!$kks parallel_for data(x,y,a) on(i,j) tile(t1,t2)
do j=jl, ju
  do i=il, iu
    y(i,j) = y(i,j) + a * x(i,j)
  end do
end do
end
```

Figure 3-13 original Fortranprogram

```
subroutine sub
use :: sub_kokkos_mod
real :: x(xil:xiu,xjl:xju), y(yil:yiu,yjl:yju), a
call kokkos_sub0(to_nd_array(x), to_nd_array(y), a,
                 xil, xjl, yil, yjl,
                 il, iu, jl, ju,
                 t1, t2)
end
```

Figure 3-14 Transformed Fortran program

```
#include <Kokkos_Core.hpp>
#include “flcl-cxx.hpp”
extern “C” {
  void kokkos_sub0(flcl_ndarray_t *nd_array_x,
                   flcl_ndarray_t *nd_array_y,
                   float a,
                   int xil, int xjl, int yil, int yjl,
                   int il, in iu, int jl, int ju,
                   int t1, int t2){
    auto x = flcl::view_from_ndarray<float**>(*nd_array_x);  ! Allocated on CudaUVMspace
    auto y = flcl::view_from_ndarray<float**>(*nd_array_y);
    parallel_for(“sub0”, Kokkos::MDRangePolicy<Kokkos::Rank<2>>({il,jl},{iu+1,ju+1},{t1,t2}),
      KOKKOS_LAMBDA(int i, int j) {
        y(i-yil,j-yjl) = y(i-yil,j-yjl) + (*a) * x(i-xil,j-xjl);
    });
  }
}
```

Figure 3-15 Kokkos code

```
module sub_kokkos_mod
use, intrinsic :: iso_c_binding
use :: flcl_mod
interface
  subroutine kokkos_sub0(nd_array_x, nd_array_y, a, xil, xjl, yil, yjl, &
                         il, iu, jl, ju, t1, t2) bind (c)
    use, intrinsic :: iso_c_binding
    use :: flcl_mod
    type(nd_array_t) :: nd_array_x
    type(nd_array_t) :: nd_array_y
    real :: a
    integer(c_int) :: xil, xjl, yil, yjl
    integer(c_int) :: il, iu, jl, ju
    integer(c_int) :: t1, t2
  end subroutine
end interface
end module
```

Figure 3-16 Fortran Intreface

In this fiscal year, as a prototype of F2Kokkos, a parallel_for directive for handling the parallel_for pattern of Kokkos was developed, and its performance was preliminarily evaluated. Table 3-13 shows the elapsed time when the processing of repeating the loop in Figure 3-17 1000 times was executed on the NVIDIA GH200 Grace Hopper Superchip of the R-CCS cloud. For comparison, the elapsed time of equivalent code using OpenACC is also shown. In the evaluation, NVIDIA HPC SDK 25.9-0 and Kokkos 4.7.01 were used.

```
!$kks parallel_for (x,y,a) on (i,j)
  do j=1, 4096
    do i=1, 4096
      y(i,j) = y(i,j) + a * x(i,j)
    end do
  end do
```

Figure 3-17 F2Kokkos test code

Table 3-13 F2Kokkos Evaluation results (sec)

F2Kokkos	F2Kokkos w/ fence	Original OpenACC	nomanagedalloc
2.95	3.38	9.49	3.32

The elapsed time of the version transformed by F2Kokkos (F2Kokkos) was 2.95 seconds, whereas the elapsed time of the OpenACC version (Original OpenACC) evaluated for comparison was 9.49 seconds.

In the F2Kokkos version, when a memory barrier (fence) operation, Kokkos::fence(), was inserted immediately after the parallel_for pattern appearing in the transformed code (F2Kokkos w/ fence), the elapsed time became 3.38 seconds.

On the other hand, when the option -gpu=mem:unified:nomanagedalloc was specified at

compile time for the OpenACC version (nomanagedalloc), the elapsed time was 3.32 seconds. Since the performance of F2Kokkos w/ fence and nomanagedalloc is almost equivalent, it was indicated that when utilizing the unified memory of GH200, F2Kokkos may achieve performance comparable to OpenACC. However, further investigation is required regarding the necessity of the memory fence immediately after parallel_for.

Creation of Japanese Documentation
To promote the adoption of Kokkos in Japan, the official documentation [4] was translated into Japanese. This work will be continued in subsequent years in accordance with updates to the official documentation, with the aim of merging it into the official documentation.

[4] https://kokkos.org/kokkos-core-wiki/index.html

3.3.1.3.3 OpenACC and OpenMP

Both OpenACC and OpenMP are directive-based programming models, and their GPU programming functionalities can be mutually translated. In addition to existing tools that convert OpenACC to OpenMP [5][6], tools for converting OpenMP to OpenACC will also be studied.

[5] https://github.com/intel/intel-application-migration-tool-for-openacc-to-openmp
[6] J. E. Denny, S. Lee and J. S. Vetter, "CLACC: Translating OpenACC to OpenMP in Clang," 2018 IEEE/ACM 5th Workshop on the LLVM Compiler Infrastructure in HPC (LLVM-HPC), Dallas, TX, USA, 2018, pp. 18-29, doi: 10.1109/LLVM-HPC.2018.8639349.

Furthermore, support for Solomon [7], a unified directive developed at the University of Tokyo that can be translated into both OpenACC and OpenMP, will also be considered.

[7] Y. Miki and T. Hanawa, "Unified Schemes for Directive-Based GPU Offloading," in IEEE Access, vol. 12, pp. 181644-181665, 2024, doi: 10.1109/ACCESS.2024.3509380.

3.3.1.3.4 Compilers

LLVM will be supported as the primary compiler for each language, and development will be carried out in cooperation with CPU vendors and accelerator vendors.
On the other hand, the LLVM Fortran frontend (Flang) is currently under active development by the community, but there remain issues particularly in the quality of support for relatively recent language standards (Fortran 2008 and later). Therefore, a test suite for the Fortran 2008 language standard (approximately 270 cases) will be developed and provided to the community (including CPU and accelerator vendors).

3.3.1.3.5 Programming tools

Programming tools such as performance profilers (CPU execution performance, GPU execution performance, communication performance) and debuggers will be studied in the detailed design phase.

3.3.2 Numerical Libraries and Middleware

This chapter describes the numerical libraries and middleware that should be provided in FugakuNEXT, including items to be considered at the basic design stage and the necessary and expected items for detailed design and development

3.3.2.1 Role of Numerical Libraries and Middleware

This chapter describes the numerical libraries and middleware that should be provided in FugakuNEXT, including items to be considered at the basic design stage and the necessary and expected items for detailed design and development.

3.3.2.2Requirements for Libraries and Middleware in FugakuNEXT

Numerical libraries and middleware are essential for application development, and as described above, their role is to solve mathematical problems that are difficult for users to implement. Since they are used in computationally intensive core components, their performance has a significant impact on applications.
Although various performance metrics exist for numerical libraries, they should be designed considering execution time (speed), power consumption, and computational accuracy.
In large-scale computations assumed for FugakuNEXT, the degree of parallelization also affects constraints on the computing architecture and problem size, as well as parallel efficiency. Furthermore, they should be callable not only from Fortran but also from C, C++, and newer languages such as Python and Julia, and must support general language bindings.
It is also important to consider whether the design is compatible with backend runtime models and whether it integrates with modern programming languages.
When considering the transition from Fugaku to FugakuNEXT, the role of numerical libraries and middleware must not be overlooked. FugakuNEXT represents a significant evolution from the architecture of Fugaku, transitioning from a CPU-only structure to a CPU+GPU configuration.
The memory hierarchy is also expected to become more complex, requiring consideration of trade-offs between capacity and bandwidth not only at the application level but also at the level of numerical libraries and middleware.
Furthermore, not only single-node usage but also distributed parallel computing across multiple nodes must be assumed. In particular, even for single-node computation, achieving high performance will likely require configurations utilizing one or more GPUs.
It is necessary to examine whether CPU and GPU computational resources can be used effectively and efficiently, whether smooth and gradual migration is possible without major changes to existing software structures, and whether maintainability and sustainability can be ensured, in order to advance the design of appropriate numerical libraries and middleware.
In the following, considerations are analyzed by dividing the overall system into the “CPU part” and the “accelerator part” using GPUs, as well as the “advanced component” required for overall integration, and the contents of discussions at the basic design stage are summarized.

3.3.2.3CPU

This section describes the basic design of numerical libraries targeting the CPU part of the FugakuNEXT system. First, issues are identified through current-state analysis, followed by the presentation of response policies to these issues. Then, survey results regarding the selection of target libraries for development are presented, and specific development items are listed.

3.3.2.3.1 Current State Analysis

In systems equipped with FUJITSU-MONAKA-X, which represent the FugakuNEXT system, the overall software stack is oriented toward a foundation based on open-source software (OSS),

and it is expected that OSS will also be utilized for mathematical libraries.
This section summarizes the current-state analysis conducted for the basic design of numerical libraries for next-generation systems equipped with FUJITSU-MONAKA-X. In this document, systems equipped with A64FX or FUJITSU-MONAKA are referred to as conventional systems, and systems equipped with FUJITSU-MONAKA-X are referred to as next-generation systems.
With the spread of AI technologies, there is a growing trend in inference processing toward execution on CPUs and mobile devices, reducing dependence on GPUs and enabling low-cost and energy-efficient environments. Therefore, providing high-performance matrix operations on CPUs is an important element in next-generation system development.
Based on this background, recent HPC and AI processors increasingly incorporate dedicated hardware and instruction sets for matrix operations. Since matrix operation performance significantly impacts overall system performance, processor vendors are focusing on accelerating this area. Considering these trends, it is important to effectively utilize the newly introduced matrix computation mechanisms in FUJITSU-MONAKA-X to maximize next-generation system performance.

3.3.2.3.1.1 Trends in New CPU Instructions and Data Types

In recent processor development, there has been a clear trend toward introducing new instruction sets and data types specialized for specific operations to improve performance in AI and data analytics workloads. Examples include SME (Scalable Matrix Extension) in the Arm architecture and AMX (Advanced Matrix Extensions) in the Intel architecture.
These instruction sets provide hardware-level support for matrix multiplication, which plays a central role in AI training and inference. In addition, support for low-precision data types such as bfloat16 and INT8 enables processing of more data simultaneously compared to double precision (FP64) and single precision (FP32).
This trend is driven by the observation that many AI workloads, especially in deep learning, do not necessarily require high precision. Instead, reducing precision (e.g., FP16, bfloat16, INT8) allows more compute units to be implemented within the same silicon area, reduces memory bandwidth consumption, and significantly improves throughput and energy efficiency.
As a result, there is a shift from high-precision floating-point computation to support for low-precision data types at the hardware level.
The FUJITSU-MONAKA-X processor is planned to adopt SME2, the successor to SME. SME introduces new instructions such as accumulation of outer products for matrix multiplication and supports low-precision matrix operations such as bfloat16, which are in high demand for AI workloads.
In conventional HPC applications, double precision has been the mainstream, and optimization using SIMD instructions such as SVE and AVX has been emphasized. Unlike AMX, SME supports single-precision real numbers and optionally double precision depending on CPU implementation. Effective utilization of SME for SGEMM and DGEMM (if supported) becomes a key challenge for performance.

3.3.2.3.1.2 Other Enhancements in FUJITSU-MONAKA-X

In processor generation transitions, not only instruction set extensions but also microarchitectural changes—such as cache size, memory bandwidth, and core count—significantly impact performance.
To maintain and improve performance of existing applications, tuning aligned with the new architecture is required. FUJITSU-MONAKA-X is expected to introduce changes in cache configuration and capacity compared to A64FX and FUJITSU-MONAKA, implying that optimal cache-blocking parameters will also change.
Additionally, an increase in core count is expected, affecting parallelization strategies. Consideration of synchronization overhead in large-scale parallel execution and optimization of

memory access under NUMA architecture are expected to contribute to performance improvements.

3.3.2.3.1.3 Approaches to Numerical Library Optimization

In optimizing numerical libraries, it is important to select effective approaches based on an understanding of the characteristics of the target computational domain.
The design philosophy of LAPACK is to concentrate computationally intensive operations into BLAS level-3 (matrix–matrix operations) and utilize highly optimized BLAS implementations tailored to each CPU architecture. Therefore, for FUJITSU-MONAKA-X, providing high-performance BLAS implementations that leverage SME instructions and consider cache configurations is critical.
In sparse matrix solvers, performance is highly dependent on the distribution pattern of nonzero elements. Since optimal data formats, preprocessing, and algorithms strongly depend on the physical background and discretization methods of the target problem, it is difficult for a single library to provide optimal performance for diverse sparse matrices. Thus, the applicability of general-purpose libraries is considered limited.
For FFT kernels, large radices are selected based on the number of processor registers to reduce the required B/F ratio. However, increasing the radix increases the number of data streams accessed simultaneously in the innermost loop, potentially causing cache line conflicts and performance degradation. Therefore, parameter tuning considering FUJITSU-MONAKA-X characteristics may be required.

3.3.2.3.2 Response policy

Based on the current-state analysis, this section presents the response policy for developing numerical libraries for FUJITSU-MONAKA-X.
The policy is to improve application performance by leveraging new features introduced in FUJITSU-MONAKA-X. In particular, optimization focusing on matrix multiplication (GEMM), identified as a core issue, using SME2 will be emphasized.

3.3.2.3.2.1 SME2 Support in BLAS Libraries

To address both HPC and AI workloads, matrix multiplication functionality leveraging SME2 will be provided through either the extension of OSS libraries or the utilization of vendor libraries such as Arm Performance Libraries.

3.3.2.3.2.1.1 SME Utilization in BLAS Libraries

From the perspective of continued use of existing HPC applications, providing optimized single-precision and double-precision matrix multiplication routines for FUJITSU-MONAKA-X is essential.
SGEMM and DGEMM are used not only through direct calls via standard BLAS interfaces but also extensively within level-3 BLAS routines and LAPACK routines.
For GEMM routines, computation kernels utilizing SME2 in addition to SVE2 will be developed. Performance improvements will also be extended to major BLAS/LAPACK routines that internally use GEMM.

3.3.2.3.2.1.2 Low-Precision Matrix Multiplication Support in BLAS Libraries

In AI applications, especially deep learning, maximizing computational performance is a critical challenge. As discussed in the current-state analysis, low-precision computation (e.g., BF16, INT8) is widely adopted to accelerate training and inference.
Dedicated hardware such as Arm SME and Intel AMX enables high-performance

implementation of low-precision matrix multiplication routines. For example, oneMKL provides functions such as cblas_gemm_bf16bf16f32 utilizing AMX, and such functions are used in frameworks like PyTorch.
To increase applications utilizing SME, it is meaningful to consider similar BLAS extensions. Therefore, SME2 kernels supporting low-precision data types widely used in AI will be developed to accelerate inference processing.

3.3.2.3.2.1.3 Investigation of SME2 Utilization Methods for FUJITSU-MONAKA-X

Effective utilization of SME2 instructions requires more than simply using new instructions; it is important to investigate processing methods suited to SME2 implementation in FUJITSU-MONAKA-X.

3.3.2.3.2.1.3.1 Consideration of Streaming SVE Mode Characteristics

SME instructions are executed in Streaming SVE mode. The vector length in this mode (SVL: Streaming Vector Length) may differ from the vector length in normal SVE mode (NSVL: Non-Streaming Vector Length), and it is recommended that SVL be equal to or greater than NSVL.
Depending on matrix multiplication parameters, either mode may be more suitable, and there is room to consider switching processing methods.

3.3.2.3.2.1.3.2 Use of Multi-Vector Instructions

SME introduces ZA storage, a two-dimensional matrix storage region sized according to SVL, and processes data in units called ZA tiles.
ZA tiles serve as destinations for matrix multiplication instructions, storing and accumulating results of outer products generated from SVE vector registers. Data transfer between memory and ZA tiles is performed via load/store instructions at the tile-slice level.
FUJITSU-MONAKA-X is expected to support multi-vector instructions introduced in SME2, enabling handling of multiple SVE vector registers simultaneously. Therefore, multiple approaches—such as tile-slice load/store or use of multi-vector instructions via SVE registers—must be evaluated.

3.3.2.3.2.1.3.3 Selection of Processing Methods by Data Type

The data types available for SME matrix multiplication depend on CPU implementation features. Non-widening operations accumulate results into ZA tiles of the same data type as inputs, whereas widening operations accumulate into higher-precision data types.
Since the number of ZA tiles varies depending on the data type, there is room for tuning the number of ZA tiles used within kernel loops.

Table 3-14 SME Matrix Multiplication Instructions

Feature		Input Data Type	Output Data Type	Instruction*
FEAT_SME		FP32	FP32	FMOP[AS], non-widening
		FP16	FP32	FMOP[AS], widening, 2-way
		BF16	FP32	BFMOP[AS], widening, 2-way
		INT8	INT32	[SU]MOP[AS], widening, 4-way
FEAT_SME (Optional)	FEAT_SME_F64F64	FP64	FP64	FMOP[AS], non-widening
	FEAT_SME_I16I64	INT16	INT64	[SU]MOP[AS], widening, 4-way
FEAT_SME2		INT16	INT32	[SU]MOP[AS], widening, 2-way
FEAT_SME2 (Optional)	FEAT_SME_F8F16	FP8	FP16	FMOP[AS], widening, 2-way
	FEAT_SME_F8F32	FP8	FP32	FMOP[AS], widening, 4-way
	FEAT_SME_F16F16	FP16	FP16	FMOP[AS], non-widening
	FEAT_SME_B16B16	BF16	BF16	BFMOP[AS], non-widening

* [SU] indicates signed or unsigned, and [AS] indicates Add or Subtract.

3.3.2.3.2.2 Optimization for FUJITSU-MONAKA-X Microarchitecture

In addition to utilizing SME2 instructions, performance of BLAS/LAPACK libraries will be maximized by tuning aligned with the microarchitectural characteristics of FUJITSU-MONAKA-X.

While leveraging knowledge obtained from optimizations for existing FUJITSU-MONAKA processors, changes introduced in FUJITSU-MONAKA-X will be taken into account.

Since cache configuration and capacity are expected to differ from A64FX and FUJITSU-MONAKA, cache-blocking parameters (block sizes) will be re-tuned to improve data locality. By setting optimal block sizes according to cache size and latency at each level, memory-access bottlenecks can be reduced and computational unit utilization improved.

Additionally, to accommodate the expected increase in core count, adjustments will be made to task granularity and load balancing among threads in parallel execution.

3.3.2.3.3 Selection of Target Libraries for Development

For numerical libraries targeting FUJITSU-MONAKA-X, two approaches are considered:

(1) use of vendor-provided libraries such as Arm Performance Libraries, and

(2) development based on open-source software (OSS).

The former has advantages in shorter development time and user support, while the latter provides greater flexibility in customization and licensing.

This development adopts an OSS-centered approach while also considering combined use of vendor libraries based on performance and development efficiency.

The following sections evaluate major BLAS libraries—OpenBLAS, BLIS, and LIBXSMM—by comparing SME support status, characteristics, and adoption level, and examine their priority as development bases.

3.3.2.3.3.1 OpenBLAS

OpenBLAS is an open-source BLAS/LAPACK implementation developed as a successor to GotoBLAS2, achieving high performance by providing manually optimized kernels for many CPU

architectures.
Regarding SME support, issues and pull requests as of September 2025 were extracted and summarized (Table 3-16).
An SME support request for Apple M4 (Issue #4715) exists, and SGEMM kernel code has been proposed, but remains in Work-In-Progress and has not been merged. Subsequent additions include limited SME usage paths under specific conditions. Mixed-precision sbgemm using widening SME instructions is not yet discussed.
SIMD utilization status is shown in Table 3-17.

Table 3-16 History of SME Support in the OpenBLAS Development Community

Issue / PR No.	Description	Opened	Closed / Merged
#4715	[issue] Request for ARM SME support (for Apple M4)	2024.5.23	Not yet
#4971	[PR] Enable HAVE_SME as VORTEX via auto-detection for Apple M4	2024.11.12	2024.11.13
#5011	[PR] Proposal of SME2 SGEMM kernel for Armv9-A architecture (Due to OpenBLAS structure, support for SYMM/TRMM is also required but not addressed)	2024.12.9	Not yet (WIP)
#5084	[PR] Support for SGEMM_DIRECT kernel using SME1	2025.1.19	2025.2.19
#5222	[PR] Fix ARMV9SME target in DYNAMIC_ARCH and add SME query code for macOS	2025.4.11	2025.5.11
#5324	[issue] test/sblat* fails to build due to undefined reference	2025.6.20	2025.7.9
#5365	[PR] Fix HAVE_SME setting in DYNAMIC_ARCH build using cmake	2025.7.9	2025.7.9
#5380	[PR] Support SME1-based DIRECT kernel for cblas_sgemm (with alpha and beta)	2025.7.15	2025.7.18
#5414	[issue] test_extensions/test_sgemmt.c fails when SME is used on Apple M4	2025.8.4	Not yet
#5423	[PR] Separate VORTEXM4 from VORTEX target and fix SGEMM_DIRECT support	2025.8.18	Not yet
#5429	[issue] Incorrect numerical results for matrix multiplication on Apple M4	2025.8.27	Not yet
#5450	[PR] Support strmm_direct kernel using SME1 in cblas_strmm; incorrect results	2025.9.18	Not yet

Table 3-17 SIMD Instruction Utilization in OpenBLAS

Function	Input / Output Data Type	SSE / AVX / AVX2	AVX-512	AMX	NEON	SVE	SME
BGEMM	BF16	△	○	×	△	○	×
SBGEMM	BF16 → FP32	△	○	○	△	○	×
SGEMM	FP32	○	○	–	○	○	△
DGEMM	FP64	○	○	–	○	○	×
CGEMM	Complex FP32	○	△	–	△	△	×
ZGEMM	Complex FP64	○	△	–	△	△	×

3.3.2.3.3.2 BLIS

BLIS (BLAS-like Library Instantiation Software) is an open-source framework for generating high-performance matrix operation libraries.
It combines architecture-independent software layers with architecture-specific optimized microkernels to achieve performance portability.
As shown in Table 3-18, BLIS currently does not support AMX or SME.

Table 3-18 SIMD Instruction Utilization in the BLIS Library

Function	Input / Output Data Type	SSE / AVX / AVX2	AVX-512	AMX	NEON	SVE	SME
BGEMM	BF16	×	×	×	×	×	×
SBGEMM	BF16 → FP32	×	×	×	×	×	×
SGEMM	FP32	○	○	–	○	○	×
DGEMM	FP64	○	○	–	○	○	×
CGEMM	Complex FP32	○	△	–	△	△	×
ZGEMM	Complex FP64	○	△	–	△	△	×

3.3.2.3.3.3 LIBXSMM

LIBXSMM is a specialized library designed to accelerate small matrix multiplication (Small Matrix Multiplication for x86). It was developed primarily by Intel contributors (Hans Pabst and Alexander Heinecke). Its key features are as follows.

- It is also used in Intel Extension for PyTorch (IPEX).
- Support for Arm and RISC-V architectures has been added to the implementation.
- There are constraints on arguments: no transpose for A, α = 1, β = 0 or 1.
- It performs Just-In-Time (JIT) code generation.
- It supports various data types: FP64, FP32, bfloat16, int16, and int8.

Unlike libraries such as OpenBLAS and BLIS, which are typically swapped at application link time, LIBXSMM uses the following interfaces.

When used via conventional function calls

- Add the prefix “libxsmm_” to [sd]gemm calls.
- If the arguments fall within the supported range, JIT execution is performed internally.
- The interception approach that keeps [sd]gemm calls unchanged was deprecated in August 2025.
- Only single-precision and double-precision GEMM are supported.
- Functions such as sbgemm, bgemm, or integer-type GEMM are not defined.

When used via C++ templates

- Specify the data type using libxsmm_mmfunction<TYPE>.
- JIT code generation is triggered in the constructor when creating an instance of libxsmm_mmfunction, and the GEMM kernel instruction sequence is generated in memory.
- The C++ interface does not appear to support mixed precision.
- Low-precision data types in C++23 are not yet considered, and the support policy is unclear (issue #877).

When used from C

- Specify data type information using libxsmm_create_gemm_shape.
- Input and output data types can be specified independently.
- Trigger JIT code generation and obtain a function pointer using libxsmm_dispatch_gemm.
- Execute matrix multiplication by calling the obtained function pointer.

Regarding SME support, related pull requests from the LIBXSMM development community’s GitHub repository were extracted and are summarized in Table 3-19.

Table 3-19 SME Support Progress in the LIBXSMM Development Community

PR No.	Description	Opened	Closed / Merged
#466	Support for A64FX (for reference)	2021.3.29	2021.3.29 (master branch)
#492, #496	AMX support for M1 (not merged)	2021.7.24	-
#910	SME support for M4 (FP32)	2024.10.16	2024.10.25 (feature_m4_sme branch)
#912	Fixes to files related to #910	2024.10.28	2024.11.2 (feature_m4_sme branch)
#915	Continued fixes to files related to #910	2024.11.5	2024.11.6 (feature_m4_sme branch)
#916	Merge of feature_m4_sme branch into main branch	2024.11.6	2024.11.8 (main branch)
#945	Fixes for CPU detection and others	2025.2.20	2025.2.26 (feature_m4 branch)
#954	Addition of environment variable LIBXSMM_AARCH64_USE_NEON	2025.3.24	2025.3.26 (feature_m4 branch)
#949	Merge of feature_m4 branch into main branch	2025.3.6	2025.3.26 (main branch)

Analysis of the JIT-generated FP32 SME compute kernel shows that it follows the same approach as OpenBLAS, executing matrix multiplication instructions consecutively across four ZA tiles.
Although SME instructions are expected to be applicable to mixed precision and integer GEMM, the current implementation supports FP32 only. Table 3-20 shows the SIMD instruction utilization status.

Table 3-20 SIMD Instruction Utilization in the LIBXSMM Library

Function	Input / Output Data Type	SSE/AVX/AVX2	AVX-512	AMX	NEON	SVE	SME
BGEMM	BF16	△	○	○	×	○	×
SBGEMM	BF16 → FP32	△	○	○	×	○	×
SGEMM	FP32	○	○	—	○	○	○
DGEMM	FP64	○	○	—	○	○	×

3.3.2.3.3.4 Priority Target OSS for Development

Although SME support has begun in some libraries, OpenBLAS provides only partial support and BLIS has none. LIBXSMM supports SME but is specialized and not a general BLAS/LAPACK replacement.
In terms of adoption, OpenBLAS is widely used (including Linux distributions) and already defines BLAS extensions for bfloat16.
Therefore, OpenBLAS is considered the primary candidate OSS for development due to its extensibility and broad architectural support.

3.3.2.3.4 Development items

Based on identified issues and response policies, the following development items are defined. The development scope focuses on BLAS-layer optimization centered on GEMM routines, considering resource-efficiency balance. Vendor libraries will continue to be evaluated during detailed design.

3.3.2.3.4.1 Introduction of SME2 Kernels into OSS BLAS Libraries

SME2 kernels will be introduced into OSS BLAS libraries. Target routines are SGEMM and DGEMM (if required CPU features are implemented).
This development is expected to improve performance of HPC applications performing matrix multiplications.

3.3.2.3.4.2 Extension of Low-Precision GEMM Support

Low-precision GEMM support (e.g., bfloat16 input / binary32 output) will be expanded in OSS BLAS libraries.
This is expected to improve performance of AI inference applications, especially in deep learning workloads.

Reference

[1] “Part 1: Arm Scalable Matrix Extension (SME) Introduction”, Arm Community blogs, May 23, 2024
（https://community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/arm-scalable-matrix-extension-introduction）
[2] “Part 2: Arm Scalable Matrix Extension (SME) Instructions”, Arm Community blogs, June 24, 2024
（https://community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/arm-sca lable-matrix-extension-introduction-p2）
[3] SME Programmer's Guide Version 1.0
[4] Arm Architecture Reference Manual for A-profile architecture

3.3.2.4GPU

In FugakuNEXT, discussions were conducted on whether a consistent numerical computing environment can be provided across both the CPU unit (FUJITSU-MONAKA-X CPU) and the acceleration unit (NVIDIA GPU), and whether existing Fugaku-targeted application assets can be applied to GPU support and acceleration with minimal modifications. The details are described below.
Particular emphasis was placed on the consistency and reliability (accuracy guarantees, compatibility, implementation differences) of foundational libraries such as mathematical functions, linear algebra, and sparse matrix solvers. Although bitwise identical results between CPU and GPU are not required, it was established as a fundamental requirement that functions such as exponential, trigonometric, hyperbolic, power, and Bessel functions produce results consistent within the bounds of ULP (Units in the Last Place). It is also agreed between RIKEN and NVIDIA that achieving bitwise identical results would require disabling FMA (fused multiply-add) operations and optimizations, leading to significant performance degradation, and is therefore impractical. The math.h implementation provided by the NVIDIA HPC SDK includes (device-specific) functions with explicitly stated ULP guarantees [3.2.1]. However, it cannot currently be verified whether these guarantees are fully consistent with those of the Fujitsu CPU-side compiler and math library implementations, as validation on the FUJITSU-MONAKA-X CPU is not yet possible. These validations are planned to be conducted at an appropriate stage in the detailed design phase, and depending on the results, additional modifications to the implementation may be required.

Reference

[3.2.1] CUDA C++ Mathematical Standard Library Function, CUDA Programming Guide v13.1,

https://docs.nvidia.com/cuda/cuda-programming-guide/05-appendices/mathematical-functions.html#cuda-c-mathematical-standard-library-function

3.3.2.4.1 Various Libraries

The correspondence between CPU-oriented numerical libraries assumed for the overall system and NVIDIA GPU numerical libraries is organized as follows.

3.3.2.4.1.1 NVPL — NVIDIA Performance Libraries

NVPL stands for NVIDIA Performance Libraries and refers to a set of libraries optimized for the Arm 64-bit architecture, designed to accelerate HPC applications. NVPL is compatible with existing standard mathematical APIs such as BLAS, LAPACK, and FFT, enabling acceleration of existing code without requiring source code modifications. The main components are as follows.
NVPL BLAS: Optimized implementation of standard BLAS (Basic Linear Algebra Subprograms)

- NVPL LAPACK: Dense matrix operations, eigenvalue computations, and linear equation solvers
- NVPL FFT: High-speed Fourier transform library compatible with the FFTW API
- NVPL RAND: High-performance random number generation library
- NVPL SPARSE: Linear algebra library for sparse matrices
- NVPL ScaLAPACK: LAPACK extension for distributed memory environments
- NVPL TENSOR: Tensor computation library

3.3.2.4.1.2 cuBLAS — CUDA Basic Linear Algebra Subroutines

cuBLAS is a BLAS library that operates on GPUs and enables high-performance execution of matrix and vector operations. It is provided as a standard component of the CUDA Toolkit and NVIDIA HPC SDK, and supports matrix-matrix multiplication, matrix-vector multiplication, and various vector operations.
The main features are as follows.

- Supports BLAS Level 1/2/3 APIs Supports multiple GPUs and mixed precision (FP16/FP32/FP64)
- Provides extended APIs (cuBLASLt) for low-precision/mixed-precision matrix multiplication and fused kernels for AI algorithms
- cuBLASXt: Multi-GPU support within a single process
- cuBLASLt: Flexible GEMM API

3.3.2.4.1.3 cuBLASDx — cuBLAS Device Extensions

cuBLASDx is a device-side extension library of cuBLAS that enables direct invocation of BLAS functions (e.g., GEMM) from within CUDA kernels. Unlike the host-side API cuBLAS, it allows low-latency and high-performance linear algebra operations within kernels.
The main functions are as follows.

- Execution of general matrix multiplication (GEMM) within kernels
- Performance improvement through fusion with other operations (e.g., memory access)
- Utilization of Tensor Cores and high-speed load instructions
- Currently provided as a preview or early release version

3.3.2.4.1.4 cuSOLVER

cuSOLVER is a GPU-based dense numerical linear algebra solver library built on cuBLAS, providing LAPACK-style APIs for higher-level linear algebra problems.
The main functions are as follows.

- Dense linear equation solvers (sparse functionality is planned to be migrated to cuDSS and AMGX-Next)
- LU decomposition, QR decomposition, eigenvalue computation, singular value decomposition
- Supports both single-GPU and multi-GPU configurations (cuSOLVERMp)

3.3.2.4.1.5 cuRAND — CUDA Random Number Generation

cuRAND is a high-performance random number generation library for GPUs, accelerating random number generation in GPU applications.

- Supports random numbers following various distributions such as uniform, normal, pseudo-random, and true random
- Allows direct generation and use within CUDA kernels
- Suitable for random-dependent workloads such as HPC simulations, statistical simulations, and Monte Carlo methods

3.3.2.4.1.6 cuSPARSE — CUDA Sparse Matrix Library

cuSPARSE is a library for efficiently executing sparse matrix computations on GPUs.

- Supports sparse matrix storage formats such as CSR
- Provides sparse matrix-vector multiplication, sparse matrix-matrix multiplication, sparse matrix operations, and format conversion
- Used for large-scale sparse data processing such as linear equation solving, graph analysis, and scientific computing
- Extended APIs for accessing Tensor Core sparsity features are provided in the related library cuSPARSELt

3.3.2.4.1.7 cuTENSOR — GPU-accelerated tensor linear algebra library

A library for tensor contraction, reduction, and element-wise operations on n-dimensional tensors. The related cuTENSORMp library provides tensor computation capabilities for multi-GPU and multi-node environments. cuBLAS, cuSOLVER, cuSPARSE, cuFFT, and cuRAND are included in the CUDA Toolkit and HPC SDK, while related GPU-accelerated libraries are additionally provided. These are classified into the following three categories.

- Device extension libraries: cuBLASDx, cuFFTDx, cuSOLVERDx, cuRANDDx, nvCOMPDx
- Linear solvers for sparse linear systems: cuDSS (direct solver), AMGX-Next (iterative solver)
- nvmath-python: A set of Python APIs enabling access to advanced features of NVIDIA libraries on CPU and GPU in single-node or multi-GPU/multi-node environments. nvmath-python.device enables access to Dx libraries and high-productivity kernel development in Python.

3.3.2.4.2 Ozaki Scheme

The Ozaki-I scheme has been productized within the cuBLAS library starting from CUDA 13.0 Update 1 released in October 2025. NVIDIA has made significant R&D investments toward the productization of the Ozaki scheme (e.g., ESC algorithms and the ADP framework) and maintains a roadmap to continue these efforts.

3.3.2.5Advanced Layer

3.3.2.5.1 Planned Development and Maintenance Items

From the perspective of advancement, considerations will be made for CPU-only, GPU-only,

CPU+GPU hybrid, and distributed parallel environments. The necessary functions of numerical libraries required for the deployment of the FugakuNEXT system are listed below, along with considerations at the basic design stage, identification of OSS that can be selected and prepared for detailed design, and discussion of requirements for those candidates. Items that require detailed performance analysis at this stage will be discussed individually in the next section.

3.3.2.5.1.1 BLAS:

BLAS (Basic Linear Algebra Subprograms) refers to the fundamental software group that provides functionality for dense matrix operations in basic numerical linear algebra. It provides fundamental operations for matrices and vectors. In particular, its functions are classified into levels 1 to 3 based on the order of computational complexity. Level 1 corresponds to vector-vector operations and norm calculations (O(N)), Level 2 corresponds to matrix-vector multiplication (O(N^2)), and Level 3 corresponds to matrix-matrix multiplication (O(N^3)). These functions are provided as sufficiently optimized software for each data type, function, and target hardware architecture, either as proprietary software from vendors or as OSS from development communities. In modern numerical libraries and mathematical software, high execution performance and productivity are achieved by combining performance-optimized code components while maintaining the abstraction level of numerical linear algebra problems. In FugakuNEXT development, it is necessary to provide BLAS optimized for the FUJITSU-MONAKA-X CPU, BLAS optimized for NVIDIA GPUs, and distributed BLAS that provides equivalent linear algebra functionality across multi-node environments. The development of BLAS for CPU and GPU environments is being investigated independently by each responsible vendor, and those results are referenced. As an alternative to vendor-provided options, high-performance OSS GPU BLAS such as magmablas in the MAGMA library is considered promising. In addition, with the many-core and many-processor evolution of CPUs and GPUs, it is necessary to support the simultaneous execution of many lightweight tasks such as batch processing. Batched BLAS has already been developed by RIKEN for A64FX, and extending this functionality to FUJITSU-MONAKA-X is straightforward. There is also the possibility that both CPU and GPU will support mixed-precision arithmetic units, making it important to consider interfaces and execution granularity control. For GEMM (matrix-matrix multiplication), discussions exist regarding mixed-precision algorithms such as the Ozaki scheme and other implementation techniques. These will be described in detail in a separate section.
For distributed BLAS implementations, options such as PBLAS and SLATE exist, but developing a library optimized for FugakuNEXT is difficult without collaboration with development communities. Other distributed GEMM implementations are also possible, but decisions on which option to adopt must be finalized by the detailed design phase. In C++26, a linalg package equivalent to BLAS functionality will be officially included. Discussions with programming language environment development regarding wrapper functions and runtime data management will be required in the detailed design phase.

3.3.2.5.1.2 LIN: Linear Solvers

Here, LIN refers to systems of linear equations and the fundamental matrix decomposition operations used internally. Solving dense linear systems has computational complexity of O(N^3), making large-scale execution rare, and sparse matrix iterative or direct methods are typically used. However, the following functions must be supported due to their importance even if not explicitly visible:
Methods that avoid direct computation of matrix inverses
QR decomposition required for orthogonalization of basis data
Cholesky decomposition as a form of QR decomposition
CPU implementations such as LAPACK and SLATE are expected, while GPU implementations

are limited to cuSOLVER and MAGMA. These will be selected and prepared for FugakuNEXT while monitoring development trends.

3.3.2.5.1.3 EVD/SVD: Eigenvalue and Singular Value Solvers

Eigenvalue decomposition (EVD) and singular value decomposition (SVD) are expected to be used not only in HPC but also in quantum computing. Since the K computer and Fugaku era, diagonalization has been central to energy calculations in quantum materials simulations. Recently, they have also become central numerical computations in data assimilation methods such as Kalman ensemble filters.

Eigenvalue algorithms are classified into iterative methods based on orthogonal decompositions (e.g., Jacobi and QR iteration methods) and methods based on Sturm sequences or determinant-based bisection. Parallel implementations often use the latter three-step approach (1. Householder tridiagonalization, 2. Cuppen's divide-and-conquer method, 3. Back transformation), which is adopted in many modern numerical libraries (e.g., LAPACK, SLATE, ScaLAPACK, ELPA, EigenExa, cuSOLVER).

Additionally, a two-stage algorithm that mitigates memory bandwidth limitations of SYMV-heavy Householder tridiagonalization is used in SLATE, ELPA, and MAGMA. Implementation complexity and scalability in distributed environments remain major challenges, requiring continued monitoring.

3.3.2.5.1.4 FFT: Fast Fourier Transform

FFT is the second most frequently used mathematical function in scientific simulations after matrix computations. CPU FFT libraries include FFTW, optimized HPC libraries such as ffte, and vendor-provided libraries such as SSL2 for A64FX. GPU FFT libraries include cuFFT (NVIDIA) and rocFFT (AMD). cuFFT also supports migration from FFTW.

Various GPU FFT libraries exist for different use cases, including VkFFT, cuFINUFFT, TurboFFT, TurboFNO, WIFFT-GPU, GLFFT, and NukadaFFT.

FugakuNEXT consists of many nodes with FUJITSU-MONAKA-X CPUs and NVIDIA GPUs. Therefore, FFT library requirements include:

1. Support for multi-GPU execution
2. Support for multi-CPU execution
3. If possible, dynamic allocation and cooperative execution between CPU and GPU depending on use cases

Item 3 requires new development and is a major challenge. For multi-GPU and multi-CPU support, heFFTe (Highly Efficient FFT for Exascale), developed under the DOE Exascale Computing Project, is a promising candidate. It supports distributed memory and multi-device environments, uses MPI communication, and supports multiple backends (FFTW, MKL, cuFFT, rocFFT, oneMKL), enabling portability. FugakuNEXT proposes development based on heFFTe to facilitate application migration.

3.3.2.5.1.5 PRNG: Pseudo-Random Number Generators

Widely used methods include F2-linear methods, Mersenne Twister, and xorshift, which provide lightweight, high-quality, and long-period properties. In parallel computing, it is important to support both long-period sequences and "jump" operations that advance states by large steps. Simply varying seeds across threads introduces proximity risks proportional to the square of the thread count. Jump operations mitigate this risk.

Although F2-linear methods support low-cost jump operations via characteristic polynomials, implementations vary and lack unified interfaces. For jumps beyond 2^64, multi-precision specifications are required, making interface design challenging.

3.3.2.5.1.6 MATH: Mathematical Functions

The following functional requirements are considered: which functions to support first, whether to support all functions defined in C99, whether to support both static and dynamic linking, and whether outputs should remain consistent across linking methods. Currently, MSVC shows different accuracy depending on linking type.
Other considerations include whether elementary functions differ across programming languages (relevant for custom implementations), performance requirements (latency vs throughput), and defining metrics or targets.
Accuracy is typically expressed in ULP, but since multiple definitions exist, a specific definition must be chosen. Specialized high-speed functions exist for certain ranges (e.g., Intel oneAPI, CUDA). For higher precision than FP64, full verification is infeasible, requiring defined validation methods. Vectorization support (multiple inputs) must also be considered.
Currently, FP32 accuracy is reported as up to 2.92e+05 ULP (MSVC), 0.93 ULP (MKL), and 1.35e+07 (CUDA). Detailed reports are available and updated semiannually. Advanced algorithms using FMA for high-speed and high-accuracy functions have also been proposed, and adoption should be evaluated.
CPU-GPU interoperability is also an issue. Vendor libraries generally lack precision compatibility. However, for portability and debugging, consistent results across CPU and GPU are required. Since both follow IEEE-754 FMA, a common framework for frequently used functions is desirable. Implementation options include inline functions or LLVM IR-level unification.

3.3.2.5.1.7 SPARSE: Sparse Matrices (Data Formats)

3.3.2.5.1.7.1 SpMV

Sparse matrix-vector multiplication (SpMV) is a key operation in solving linear systems arising in PDE simulations. For a sparse matrix A with n rows and nnz non-zero elements, SpMV resembles BLAS2 gemv: $y = \alpha op(A)x + \beta y$.
Unlike dense gemv (O(n^2)), SpMV has much lower computational intensity (~10–100 nonzeros per row). Formats include COO, CSR, CSC, and GPU-oriented formats such as SELL-C-σ. BSR enables SIMD efficiency by grouping elements into blocks.
SpMV applies to a single RHS vector, while SpMM applies to multiple RHS vectors, enabling cache reuse and higher computational intensity. Sparse matrix multiplication requires pre-identifying patterns and allocating memory.
SparseBLAS lacks standardization, with varying formats and APIs across vendors. However, inspector-executor models are commonly used for optimization.
Required features include:

- Separate functions per floating-point type in C interfaces
- SpMM storing vectors as dense matrices with dimension argument
- Internal optimization via data format transformation (CSR to BSR or SELL-C-σ)
- Access to permutation arrays for optimization
- Unified API for CPU and GPU implementations, with GPU using cuSPARSE and CPU format defined by final design

These routines should also serve as backends for KOKKOS-kernels and PETSc.

3.3.2.5.1.7.2 SPARSE functions for LIN, EVD, PRECOND

a. PETSc provides Krylov subspace solvers (CG, GMRES, BiCGStab) and preconditioners such as GAMG and additive Schwarz.
b. MUMPS provides multifrontal direct solvers with high BLAS3 utilization and is used in PETSc.
c. ARPACK and SLEPc provide eigenvalue solvers based on Arnoldi methods.
d. PETSc SNES provides nonlinear solvers (trust region Newton, nonlinear CG, BFGS).

e. PETSc TAO and IPOPT provide optimization libraries.

3.3.2.5.1.8 Middleware

Middleware is considered to include data management and frameworks above numerical libraries. However, definitions and selection policies remain unclear at the basic design stage. Some SPARSE-related software may also be classified as middleware, and further clarification will be made in final and detailed design phases.

3.3.2.6Libraries Requiring Performance Optimization

While previous sections assume vendor-provided or OSS-based solutions, some libraries require optimization or strategic development led by RIKEN to maintain international leadership. At the basic design stage, two items are highlighted: the Ozaki scheme driver and the EigenExa dense eigenvalue library.

3.3.2.6.1 GEMM: Ozaki Scheme Driver

The Ozaki scheme (Ozaki-I/II) emulates matrix multiplication using low-precision arithmetic. With improved GPU AI performance, low-precision GEMM performance has increased significantly, allowing double-precision-equivalent accuracy to be achieved using the Ozaki scheme. This is a novel technology originating in Japan.
To compensate for or exceed the loss of double-precision performance when transitioning from CPU to GPU, RIKEN will actively develop Ozaki scheme drivers in collaboration with CPU and GPU vendors.

3.3.2.6.1.1 Required Functional Requirements

The Ozaki scheme requires high-performance low-precision matrix multiply-accumulate operations. Considering current processor trends, at minimum, support for INT8 and FP8 (E4M3) inputs with approximately FP32 output precision is required. Interfaces for invoking such GEMM kernels are also necessary. Double-precision FMA operations are required internally.

3.3.2.6.1.2 Current-State Analysis

Ozaki-I using INT8 is supported in cuBLAS 13.0u2 and later. However, acceleration of the referenced algorithm has been proposed (https://doi.org/10.1177/10943420241313064), and there remains room for performance improvement. This implementation supports matrix multiplication of single-precision real, double-precision real, single-precision complex, and double-precision complex numbers. Performance model analysis of the accelerated algorithm estimates the following performance.
The vertical axis shows the measured performance of the INT8 matrix multiplication kernel (TOPS), and the horizontal axis shows the effective memory bandwidth (TByte/s).
Ozaki-II using INT8 is provided as an independently implemented OSS published on the R-CCS GitHub (https://github.com/RIKEN-RCCS/GEMMul8). This implementation supports matrix multiplication of single-precision real, double-precision real, single-precision complex, and double-precision complex numbers. It also supports both HIP and CUDA environments. Based on performance model analysis, the following performance is estimated.
The vertical axis shows the measured performance of the INT8 matrix multiplication kernel (TOPS), and the horizontal axis shows the effective memory bandwidth (TByte/s).
The vertical axis shows the measured performance of the INT8 matrix multiplication kernel (TOPS), and the horizontal axis shows the effective memory bandwidth (TByte/s).

Figure 3-18 Performance projection of Ozaki-I (DGEMM).

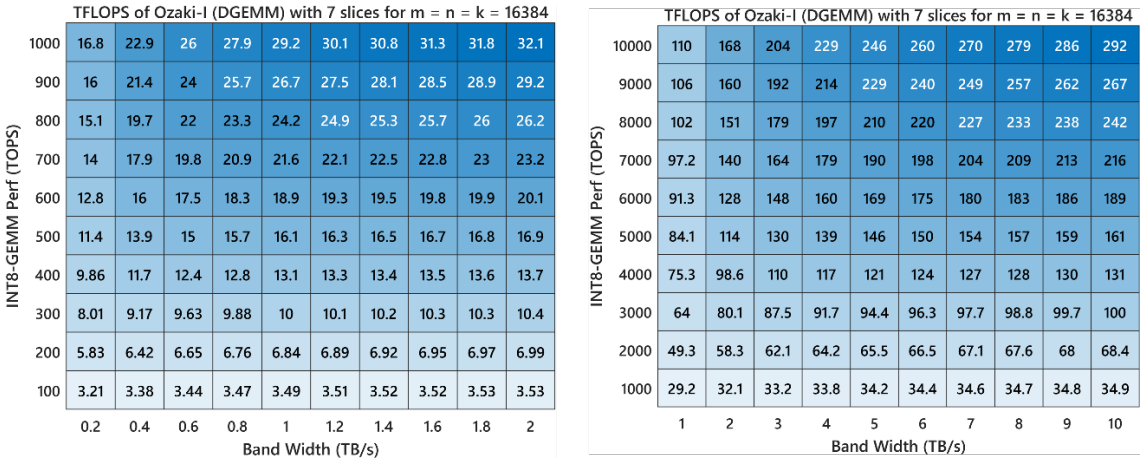


Figure 3-19 Performance projection of Ozaki-II (DGEMM).

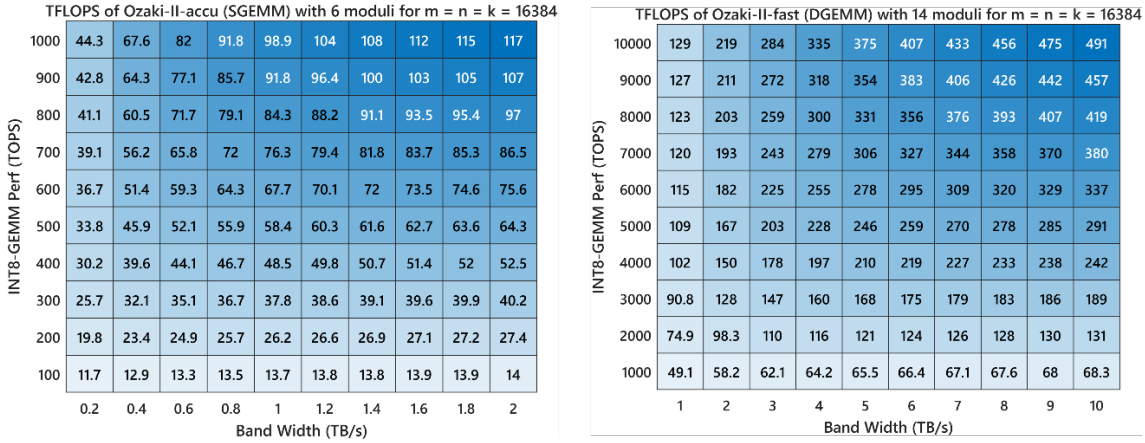
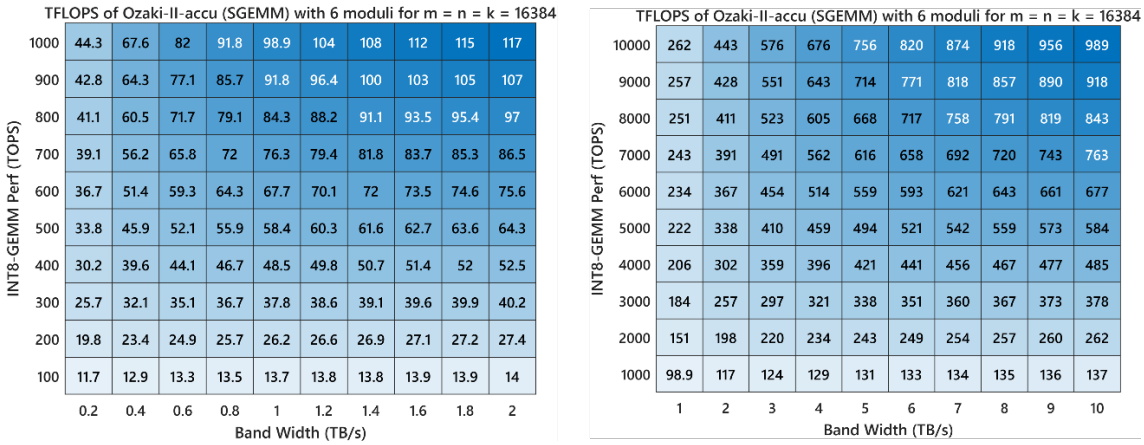


Figure 3-20 Performance projection of Ozaki-II (SGEMM)



3.3.2.6.1.3 Issues and Direction

The INT8 Tensor Core performance of NVIDIA Blackwell Ultra and NVIDIA Rubin has been significantly reduced compared to previous generations, and this trend may continue in the future. Since GPUs adopted in FugakuNEXT may have a similar configuration, it is necessary to develop OSS that supports not only 8-bit integers (INT8) but also 8-bit floating point (FP8_E4M3).

For OSS targeting the FUJITSU-MONAKA-X CPU, the design direction will be determined once the configuration of SME and performance indicators are finalized.

3.3.2.6.1.4 Development Items

If SME2 capable of operating with sufficient performance on the FUJITSU-MONAKA-X CPU is implemented, it will be possible to support the Ozaki scheme on both CPU and GPU and provide a seamless environment in terms of computational accuracy and performance. To achieve this, matrix multiplication kernels and interfaces for 8-bit integer and floating-point operations running on the FUJITSU-MONAKA-X CPU will be required in the future.

3.3.2.6.2 EigenExa: Eigenvalue Library

The importance of eigenvalue computation has been high, particularly in quantum physics and quantum chemistry calculations on conventional HPC platforms. There has been a history of developing open-source software focused solely on eigenvalue computation both domestically and internationally. Examples include ELPA mainly in Europe, DLA-future at CSCC, and EigenExa in the Fugaku project. In addition, comprehensive numerical linear algebra libraries such as ScaLAPACK and SLATE are widely used. EigenExa, discussed in this section, is recognized domestically and internationally as a dense eigenvalue computation library for parallel computing environments. Although early GPU support was desired beyond the Fugaku project, it was not realized. In the development of FugakuNEXT, not only GPU support but also GPU cluster support and performance optimization will be advanced. In addition to supporting FugakuNEXT, operation on major overseas supercomputers will also be considered, with design and development aiming at a de facto standard.

3.3.2.6.2.1 Required Functional Requirements

The required functions for the dense eigenvalue library EigenExa for FugakuNEXT are as follows.

1. High-performance execution with arbitrary configurations in CPU, GPU, and CPU+GPU mixed environments (CPU, GPU, distributed parallel computing across nodes, thread parallelism, etc.)
2. Double-precision and single-precision computation kernels
3. Residual correction iteration of eigenvalues by Ogita and Aishima
4. Mixed-precision operation control and performance acceleration using the Ozaki scheme

3.3.2.6.2.2 Current Status Analysis

The existing EigenExa achieves high parallelism and high performance in CPU-only cluster environments centered on Fugaku. However, since it internally uses libraries derived from netlib such as ScaLAPACK and LAPACK, it includes Fortran code. While some subprograms have been converted to C and C++, achieving high performance under a CUDA environment with flexible GPU offloading requires full migration to C++ and CUDA. Furthermore, K and Fugaku are highly parallel computers and adopt communication-avoiding algorithms that strongly consider communication algorithms. When analyzing mixed-precision operations expected in FugakuNEXT, EigenExa is designed assuming only double-precision computation, and transitioning to single precision or other data formats is not easy. On the other hand, attempts have been made on Fugaku hardware to improve precision while ultimately ensuring double-precision equivalent accuracy, and optimistic performance improvements have been achieved even with only SVE-based acceleration.

3.3.2.6.2.3 Issues and Direction

As analyzed in the “Current Status Analysis,” migration from Fortran to C++ and CUDA must be

carried out with the highest priority. This migration also aims to enable flexible implementation of mixed-precision operations and multi-data-type management described later. Next, it is necessary to expand to a multi-node version with inter-node communication suitable for FugakuNEXT. Currently, discussions are at the basic design level for programming environments and communication libraries, and implementation issues will be addressed during detailed design. It is also required to introduce code management techniques that can flexibly accommodate future migration to common programming languages after C++ and CUDA migration.

RIKEN has collaborated with the University of Fukui from 2003 to 2024 to complete the C++ conversion of divide-and-conquer routines in EigenExa. The know-how from this C++ conversion can be applied to other routines and parallel computing in general. During the FY2025 basic design phase, C++ conversion was performed for "parallel execution control," "communication functions," and the "block Householder back-transformation routine" used in post-processing of eigenvalue computation. It has also been confirmed that the converted C++ code maintains computational correctness and does not degrade performance. From the detailed design phase onward, C++ conversion of preprocessing Householder tridiagonalization routines, performance optimization, and full CUDA migration will be promoted, taking AI technologies into consideration.

Under conditions where significant acceleration of matrix multiplication using the Ozaki scheme is expected, there is strong affinity with residual correction iterative algorithms whose internal operations are matrix multiplications. Therefore, instead of the conventional approach of solving all stages with full precision, a dynamic mixed-precision selection approach will be prioritized, where computations are performed with moderately coarse precision and then refined iteratively starting from that initial value. Although support for mixed-precision operations on FUJITSU-MONAKA-X is currently undecided, this approach will result in the world's first CPU+GPU mixed-precision eigenvalue solver that actively utilizes mixed precision. While major functional and implementation changes are required, they are internal changes, and design and development will prioritize minimizing the burden on users so that neither CPU nor GPU usage requires significant modifications.

3.3.2.7Items Required for Performance Evaluation of Developed Libraries

3.3.2.7.1 Study of Benchmarking Methods

FugakuNEXT aims to achieve continuous performance evaluation of the system. This initiative is intended not only for the design and evaluation of future HPC system environments but also for identifying issues arising from system maintenance (failures, fixes, version upgrades) and contributing to the development and performance evaluation of applications running on HPC systems. As the system software area of FugakuNEXT aims to develop and share language environments as well as various libraries and tools, it is necessary to evaluate their impact on application performance. This section summarizes the benchmarking efforts required for continuous performance evaluation.

3.3.2.7.1.1 Preparation of Datasets

Continuous performance evaluation requires datasets for running benchmarks. Since many deliverables provided by the numerical library and middleware WG are in library form, evaluating performance directly is difficult. Users invoke numerical libraries from applications, so arguments used in library calls are required. When using libraries for system performance evaluation, it is desirable to prepare datasets that can both extract and limit system performance. Additionally, for continuous performance evaluation, datasets must be prepared to run on various system environments (CPU and GPU). Since it is difficult to prepare all such datasets in advance at the benchmarking stage, the following steps are required.

1. Create one typical sample problem and execution environment and generate the required

dataset
2. Create typical datasets for distributed and non-distributed parallel cases
3. Create datasets corresponding to system memory and cache capacity|
4. Create datasets that can run on multiple environments
5. Dump arguments used when invoking libraries during application execution and generate datasets from the dumped data

For steps 1–5, it is desirable to automatically generate dummy data.

3.3.2.7.1.2 Evaluation Method

The datasets and execution environments using sample problems will be placed on the benchmark framework described in the next section to enable continuous performance evaluation across various environments. The evaluation environment on the framework allows performance evaluation periodically or triggered by system or library changes. To detect errors caused by environment changes, checksum output is required for benchmarking. In addition, to evaluate performance, it is desirable to measure time across several characteristic phases. If hardware counter information can be obtained for each phase, it becomes possible to evaluate computation, communication, and I/O characteristics. The framework is expected to include mechanisms for predicting performance when system parameters change, enabling its use in system co-design.

In general, application developers often perform the work of integrating into a benchmark framework. However, since libraries are often adapted from externally developed software, the WG must carry out the work of integrating them into the benchmark framework.

3.3.2.7.2 Creation of Benchmark Suites Based on CB/CI/CD Methodologies

CI/CD stands for Continuous Integration and Continuous Delivery/Deployment. For FugakuNEXT development, Continuous Benchmarking (CB) has been proposed with continuous performance evaluation in mind. Within the application development area benchmark WG, two platforms, BenchKit and BenchPark, are prepared for FugakuNEXT performance evaluation. BenchKit is independently prepared by the benchmark WG, where source code and data are managed on GitHub, and build/run environments and scripts are provided to enable performance measurement and comparison across various system environments. It is easy to deploy and allows even general users unfamiliar with systems to easily measure performance using their own applications. BenchPark is a CB platform developed by the DOE Exascale Computing Project (ECP) for procurement, performance comparison, and architecture evaluation, and is a benchmark framework that enables HPC applications and mini-applications to run reproducibly under the same conditions by anyone. It requires environment setup for each system, and although the introduction barrier is high due to requirements such as Spack for portability and persistence, performance evaluation and analysis using BenchPark will be prioritized considering future generality in the system software area.

3.3.3 Communication Libraries

A basic design proposal for the communication library targeting around 2030 is described. By utilizing standard OSS communication libraries, the goal is to realize a communication library optimized for the FugakuNEXT hardware architecture.

3.3.3.1 Overall System / CPU Section

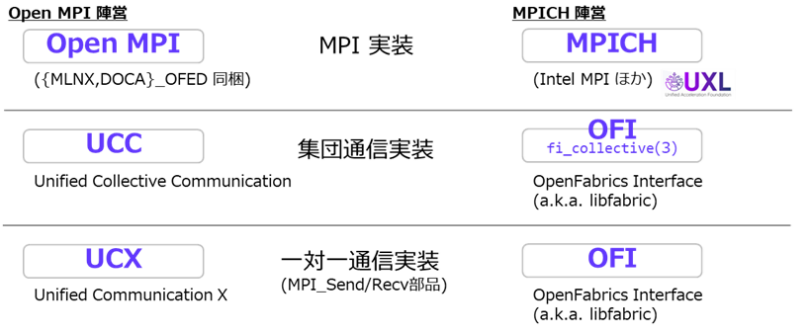
3.3.3.1.1 Trends in Communication Libraries

This section presents the results of an investigation into surrounding trends in communication libraries with a view toward around 2023.

3.3.3.1.1.1 MPI Standard

Comparison of Open MPI and MPICH
As OSS MPI libraries, Open MPI and MPICH are currently the two major communities. Figure 3-21 shows a comparison diagram of the communication layer implementations of Open MPI [1] and MPICH [2].

Figure3-21 Implementation comparison of the communication layers of Open MPI and MPICH



Some sites use MPICH as a base; however, when comparing Open MPI and MPICH, Open MPI has higher overall maturity, while MPICH has the characteristic of closely following the MPI standard. Nevertheless, Open MPI has the following advantages.

- Fujitsu MPI, up to the supercomputer “Fugaku,” has been based on Open MPI, and the accumulated software assets and know-how can be leveraged for FugakuNEXT.
- Considering that the accelerator vendor for FugakuNEXT has been decided as NVIDIA, NVIDIA’s current strong contributions to the Open MPI ecosystem are advantageous, particularly as Open MPI is ahead in GPU-aware MPI.
- NVIDIA provides products based on Open MPI.

From the above, Open MPI is selected as the primary candidate, and an evaluation platform for MPI applications will be established early toward the detailed design phase.

3.3.3.1.1.2 Communication Library Software Stack

Figure 3-22 shows the relationship diagram of system libraries. MPI aggregates various foundational software components and provides APIs by abstracting system resources. Figure 3-23 shows the internal structure of Open MPI. Starting from openmpi-5.0 [3], the openib component (InfiniBand / RoCE implementation) responsible for InfiniBand communication has been removed. As a result, Open MPI itself has effectively become a front-end for communication. Interconnect-specific processing has been moved to low-level communication libraries (UCX, OFI). Consequently, in Open MPI, only COLL effectively retains communication implementation, but replacement by UCC is progressing.

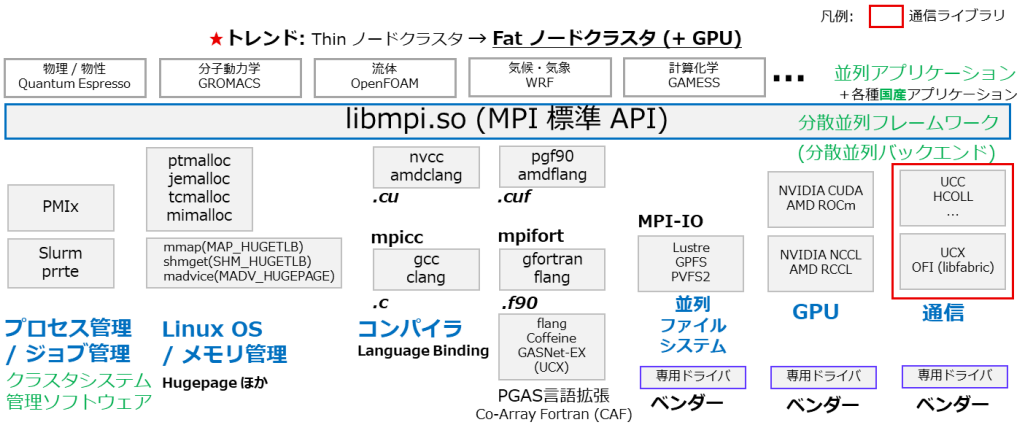


Figure 3-22 System Library stack

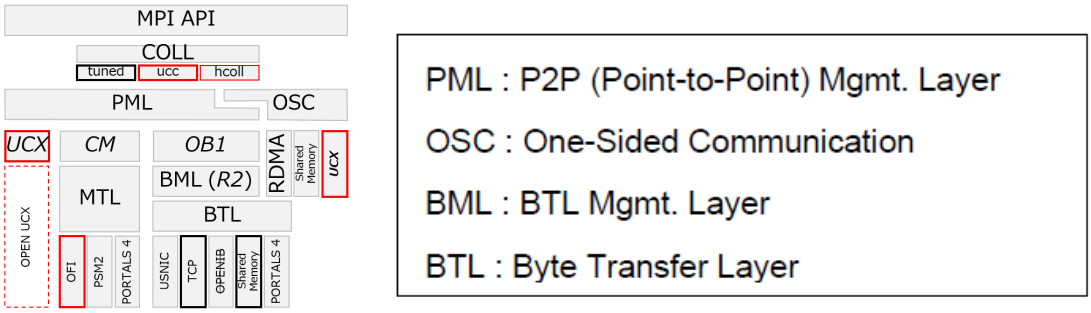


Figure 3-23 Open MPI internal stack

3.3.3.1.2 MPI Standard Trends

The compliance status of Open MPI with the MPI standard [3] is shown. Open MPI is expected to support the MPI 4.1 Standard in Version 6.0.0. Version 5.0 conforms to the MPI 3.1 Standard.

Table 3-21 Open MPI support status for MPI standards

MPI Standard	Release Date	Open MPI Support
MPI 5.0 Standard	2025/06/05	
MPI 4.1 Standard	2023/11/02	Planned in openmpi-6.0.0
MPI 4.0 Standard	2021/06/09	
MPI 3.1 Standard	2015/06/14	openmpi-5.0.8 (latest)
MPI 3.0 Standard	2012/09/21	

3.3.3.1.2.1 MPI Standard Trends - MPI 4.0 Standard

In the MPI 4.0 Standard (released on 2021/06/09) [4], due to the recent increase in data volume, the int-type count argument became insufficient, and a “large-count” API using MPI_Count was added. As a result, size limitations are expected to be relaxed, particularly in collective communication and MPI_File_*().

3.3.3.1.2.2 MPI Standard Trends - MPI 5.0 Standard

In the MPI 5.0 Standard (released on 2025/06/05) [5], a standard ABI (Application Binary Interface) specification was added to enable interoperability between different MPI implementations. With this specification, in addition to the current API (source compatibility), ABI (binary compatibility) will advance in the future. For example, a binary built with Open MPI can be executed in an MPICH environment.

3.3.3.1.3 Point-to-Point Communication Library Integration

3.3.3.1.3.1 UCX Overview

UCX (Unified Communication X) [6] is a low-level point-to-point (and one-sided) communication library. Its architecture is structured as shown in Figure 3-24 and Table 3-22 (from https://github.com/openucx/ucx). Among the UCX components, the implementation of interconnect-specific parts is handled by the Transport layer UCT (Unified Communication Transport) [7]. When adding interconnect-specific functionality, individual implementation in UCT is required; however, it can be implemented using the APIs provided on the UCT side.

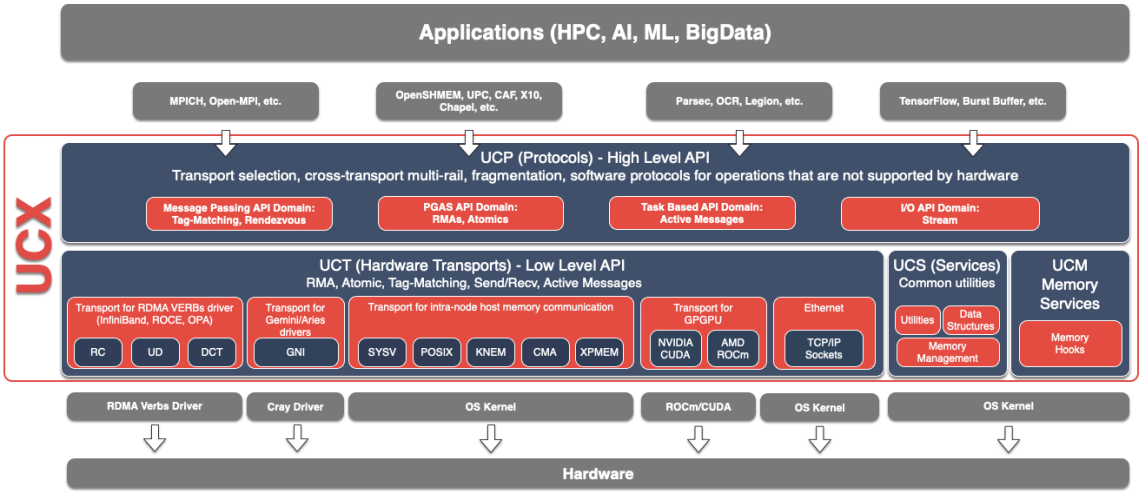


Figure 3–24 UCX Configuration

Table 3-22 UCX

Component	Role	Description
UCP	Protocol	Implements high-level abstractions such as tag-matching, streams, connection negotiation and establishment, multi-rail, and handling different memory types
UCT	Transport	Implements high-level abstractions such as tag-matching, streams, connection negotiation and establishment, multi-rail, and handling different memory types
UCS	Services	A collection of data structures, algorithms, and system utilities for common use

UCM	Memory	Intercepts memory allocation and release events, used by the memory registration cache

3.3.3.1.3.2 UCT Overview

UCT (Unified Communication Transport) is responsible for the Transport layer in UCX, and the implementation of interconnect-specific parts is also carried out within UCT. An overview of the UCT APIs is shown in Table 3-23. Since the specifications of the NIC driver for Fugaku NEXT have not yet been clarified, the UCT APIs to be used will need to be investigated and examined after the architecture of the Fugaku NEXT system has been finalized.

Table 3-23 UCT API Overview

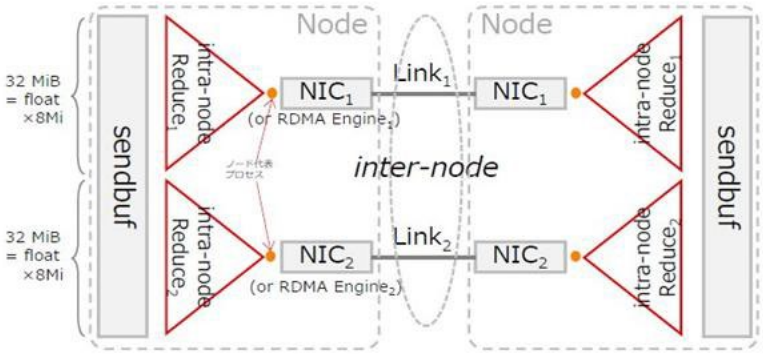
UCT API	Numbe of API functions
UCT Communication Resource	34
UCT Communication Context	7
UCT Memory Domain	14
UCT Active Message	7
UCT Remote memory access operations	6
UCT Atomic operations	6
UCT Tag Matching operations	8
UCT client-server operations	13

3.3.3.1.4 Integration of Collective Communication Libraries

3.3.3.1.4.1 Optimization for CPU Section

- Hierarchical Collective Communication Extension
  - Collective communication is a communication pattern frequently used in distributed parallel applications and is important in practice. MPI provides commonly used communication patterns as collective communication routines (22 routines). Collective communication implementations generally switch algorithms based on factors such as the degree of parallelism and message size to select the optimal algorithm. In recent years, with the advancement of chiplet technology, CPUs have become larger in scale, and memory hierarchies such as inter-socket NUMA have become more complex. As a result, there is increasing demand to utilize faster intra-node communication as

preprocessing in combination with inter-node communication. This has led to the emergence of “hierarchical” collective communication implementations that use different algorithms for intra-node and inter-node communication. However, at present, it is not clearly established whether these implementations are at the research level or production level. An example of hierarchical communication application is shown below. In the basic design, a fundamental investigation of “hierarchical” collective communication implementations was conducted. In addition, frameworks for hierarchical collective communication in Open MPI and Unified Collective Communication (UCC) are presented. In Open MPI, HAN (Hierarchical AutotuNed) [9], an implementation of hierarchical collective communication, has been incorporated. In UCC [8], which is one of the collective communication libraries, a hierarchical collective communication implementation, Hier, has also been incorporated into the Collective Layer.



例 1：Multi-Rail 構成での Double Tree による最適化例

	Open MPI	UCC
Name	HAN (Open MPI Hierarchical AutotuNed)	Hier (UCC Hierarchical Collective Layer)
git URL	https://github.com/open-mpi/ompi	https://github.com/openucx/ucc
Path on git	ompi/mca/coll/han/	src/components/cl/hier/

- The framework for intra-node collective communication in hierarchical collective communication in Open MPI and UCC is shown. Open MPI also incorporates an intra-node–specific collective communication implementation such as XHC (XPMEM-based Hierarchical Collective) [10], which is assumed to operate as a component of hierarchical collective communication implementations.

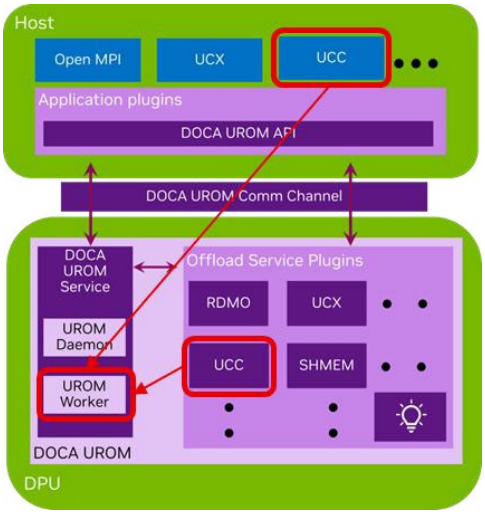
	Open MPI	UCC
Name	XHC (XPMEM-based Hierarchical Collective)	TL/SHM in NVIDIA HPC-X version UCC ?
git URL	https://github.com/open-mpi/ompi	https://github.com/openucx/ucc
Path on git	ompi/mca/coll/xhc/ openmpi-6.0.0 予定機能	Rev. 2.18.0 HPC-X バイナリのみ。ソース非公開

- In addition to the main hierarchical collective communication implementation, there are related frameworks such as intra-node collective communication (reduction operation component) and intra-node collective communication (inter-process communication component), which may offer opportunities for optimization.

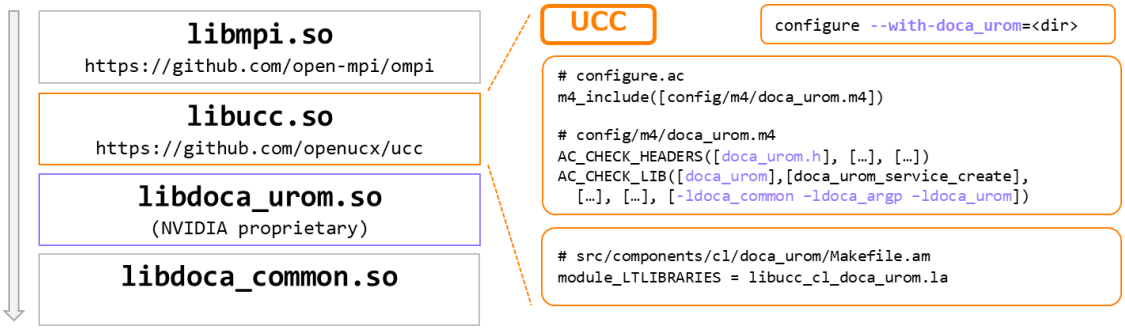
	Open MPI	UCC
Name	SMSC (Shared Memory Single Copy)	? (製品版のみで、ソース非公開のため不明)
git URL	https://github.com/open-mpi/ompi	?
Path on git	opal/mca/smsc/xpmem/ btl/sm 等と共通部	?

	Open MPI	UCC
Name	Open MPI **op/aarch64** MPI_Op 演算コンポ	UCC **EC** (Execution engine Context)
git URL	https://github.com/open-mpi/ompi	https://github.com/openucx/ucc
Path on git	ompi/mca/op/{base,aarch64}/	src/components/ec/cpu/

- Network Topology Optimization
  - Interconnects and network topologies in supercomputer centers may be system-specific. In the supercomputers “K” and “Fugaku,” proprietary NICs such as the Tofu interconnect and Tofu interconnect D, as well as torus/mesh network topologies, were used, and interconnect-optimized inter-node collective communication algorithms such as Ternary-X, which utilize multiple NICs and multiple paths simultaneously, were supported. At present, interconnects and network topologies are under consideration by the relevant WG, and the adopted hardware and technical specifications have not yet been determined. Therefore, consideration of algorithms optimized for the interconnect will be conducted after various specifications are finalized. In the basic design, as possibilities for collective communication offloading and in-network computing devices placed within the system or at communication switch ports, a fundamental investigation was conducted using UCC’s DOCA UROM (Unified Resource and Offload Manager) as an example of DPU offloading. NVIDIA DOCA UROM [11] enables part of HPC and AI parallel computing tasks to be offloaded from the host CPU to a DPU. Furthermore, UCC incorporates a mechanism for batch offloading of “collective communication” to the DPU, allowing collective communication processing to be offloaded to the DPU.



The left figure shows a schematic of NVIDIA DOCA UROM DPU offloading. The red boxes and arrows in the figure indicate the offloading of part of the host’s “parallel computing tasks” to the DPU via UCC, as well as the offloading of “collective communication processing.” The lower figure shows the call hierarchy structure of UCC and NVIDIA DOCA UROM. DOCA (software) itself is proprietary to NVIDIA, and its internal details are not disclosed.



- Device-Specific Optimization
  - Even in widely available modern NICs, technologies such as RDMA (Remote Direct Memory Access) and tag matching for fine-grained packet distribution are available. There are often various proprietary extensions, such as trigger-based request operations with configurable conditions and integration with in-network functions. However, if the available volume is limited and cannot be sufficiently deployed, these functions may not be usable. For communication devices such as NICs, the device vendor typically provides basic access through low-level point-to-point communication libraries such as UCX and OFI. However, for proprietary extensions, dedicated implementations at the collective communication level are often required. In addition, due to volume constraints, collective communication support may only be available under specific conditions. At present, this is under consideration by the relevant WG, and the adopted hardware and technical specifications have not yet been determined. Therefore,

optimization for the adopted hardware and network topology will be considered after various specifications are finalized. If components different from commodity hardware are adopted and affect layers above the physical layer, it is highly likely that communication drivers and parts of low-level point-to-point communication libraries (typically interconnect-specific components) will be affected. Optimization using device-specific features of communication devices such as NICs must be evaluated by the communication library WG in terms of feasibility for collective communication optimization, considering the scope, effectiveness, and availability of such features.

3.3.3.1.5 Integration with Other Software Components

- Integration of System-Specific Extensions
  - Communication libraries in the FugakuNEXT era are expected to utilize OSS and, if necessary, feed required features back to the OSS community as a basic development policy. However, optimized implementations for supercomputer center interconnects and network topologies may become system-specific extensions, and for proprietary collective communication implementations, it may be necessary to maintain separately managed source code. Open MPI adopts a Modular Component Architecture with low dependency, and for example, various collective communications are implemented as the mca/coll interface. Furthermore, Unified Collective Communication (UCC) [12], one of the collective communication libraries, provides a plugin mechanism. In UCC, external collective communication implementation binaries can be dynamically loaded through a plugin mechanism called Transport Layer Collective Plugins (TLCP). It is also possible to adjust plugin priority via environment variables. In the basic design, TLCP was used as an example for handling separately managed source maintenance, and a fundamental investigation was conducted.

```
$ mpirun -n 2 ¥
  -mca coll_ucc_enable 1 -mca coll_ucc_priority 100 ¥
  -x OMPI_UCC_TLCP_UCP_EXAMPLE_TUNE="alltoall:0-inf:inf" ¥
  osu_alltoall
```

UCC プラグイン (TLCP) の利用
環境変数でプラグインの priority を調整して有効化。但し環境変数名 *foo*_TUNE はプラグイン実装依存

```
$ cd ucc-1.5.0
$ ./autogen.sh
$ ./configure --with-tlcp=ucp_example ...
$ make
$ make install
```

UCC プラグイン (TLCP) の構築
configure --with-tlcp で “,” で連結してプラグインを明示的に指定するか、特殊な “all” か、いずれかで有効化

- Integration with System Memory
  - The Linux OS manages an address translation table in main memory to translate virtual memory addresses to physical memory addresses. To accelerate access, it also utilizes the Translation Look-aside Buffer (TLB) in the CPU. However, since the number of TLB entries is finite, a TLB miss occurs when lookup fails. In such cases, hardware must perform a page table walk to search the address translation table in main memory, which incurs significantly higher cost (time) compared to a TLB hit. To reduce the number of TLB entries consumed by workloads, there is a hardware feature called large pages, which increases the amount of memory represented by a single TLB entry by several hundred to several thousand times. In the supercomputers “K” and “Fugaku,” a dedicated large-page library called libmpg was provided to allow applications to easily use this feature. With the transition to OSS, it is necessary to consider alternatives to the libmpg library. In the basic design, as an example of enabling large pages using existing OSS, a method is shown to explicitly control the use of Hugepage (the Linux implementation of large pages [13]) via the glibc malloc function [14]. To enable Hugepage from MPI, it can be used by launching the MPI program with the environment variable GLIBC_TUNABLES set to enable glibc.malloc.hugetlb. Below is an example of

enabling Hugepage during MPI program execution and an explanation of the values specified for glibc.malloc.hugetlb. Whether Hugepage will be used in operation will be examined by the relevant WG in the detailed design phase.

mpirun での指定方法

```
$ mpirun ¥
        -n 4 ¥
        --report-bindings ¥
        --map-by ppr:4:node:pe=72 ¥
        -x GLIBC_TUNABLES="glibc.malloc.hugetlb=2" ¥
        ./mpi-hello.out
```

Tunable: glibc.malloc.hugetlb

This tunable controls the usage of Huge Pages on malloc calls. The default value is 0, which disables any additional support on malloc.

Setting its value to 1 enables the use of madvise with MADV_HUGEPAGE after memory allocation with mmap. It is enabled only if the system supports Transparent Huge Page (currently only on Linux).

Setting its value to 2 enables the use of Huge Page directly with mmap with the use of MAP_HUGETLB flag. The huge page size to use will be the default one provided by the system.

A value larger than 2 specifies huge page size, which will be matched against the system supported ones. If provided value is invalid, MAP_HUGETLB will not be used.

チューナブル・パラメータ: glibc.malloc.hugetlb

このチューナブル・パラメータは、malloc 呼び出しにおける Huge Page の使用を制御します。デフォルト値は 0 で、malloc の追加サポートは無効になります。

値を 1 に設定すると、mmap によるメモリ割り当て後に madvise と MADV_HUGEPAGE の使用が有効になります。これは、システムが Transparent Huge Page をサポートしている場合にのみ有効になります (現在は Linux のみ)。

値を 2 に設定すると、MAP_HUGETLB フラグを使用して mmap で Huge Page を直接使用できるようになります。使用される Huge Page サイズは、システムによって提供されるデフォルトのサイズになります。

2 より大きい値は Huge Page サイズを指定し、システムがサポートするサイズと比較されます。指定された値が無効な場合、MAP_HUGETLB は使用されません。

https://sourceware.org/glibc/manual/2.42/html_mono/libc.html

The known operational issues of Hugepage are described below. Hugepage has two types: a Hugepage pool type (pre-reserved type), which uses a memory pool dedicated to large pages, and a Hugepage non-pool type (on-demand type), which allocates large pages from OS-managed free (unused) memory when requested by an application and returns them to the OS free memory upon release.

The Hugepage pool type generally has a narrow search space for unused (free) large pages, and the cost of allocation and deallocation is almost constant and low. However, even if applications do not use large pages, the pre-allocated portion cannot be used as normal pages, which tends to reduce memory utilization efficiency.

On the other hand, the Hugepage non-pool type shares free memory with normal pages, improving memory utilization efficiency. However, since physically contiguous memory regions must be secured within large pages, it is known that the search cost can increase rapidly depending on the level of fragmentation.

Furthermore, in FugakuNEXT, changes in memory usage, such as GPU Unified Memory, may introduce new issues. If such issues are identified during detailed design, they will be referred to the relevant WG for further consideration.

Type	Pros.	Cons.
HugePage Pool Type (e.g., nr_hugepages)	When calling malloc(3), physical contiguous regions (pages) are carved out from the OS free memory in advance, reducing the cost of allocation (search is completed within the pool)	Memory in the pool is not included in the OS free memory, so depending on the application, Normal Pages may become insufficient
HugePage Non-Pool Type (e.g., nr_overcommit_hugepages)	When sufficient memory is available, Huge Pages and normal Pages can be flexibly allocated according to application requirements	During malloc(3) calls, latency and performance degradation may occur depending on memory conditions ※1

※1 If OS free memory is insufficient or fragmentation has progressed, contiguous physical memory regions of the required size may not be secured, and there is a concern that the OS may experience severe slowdown.

3.3.3.1.6 Language Interoperability

3.3.3.1.6.1.1 Fortran Language Binding

In the MPI specification, the Fortran language binding has been defined since MPI-1.0 and is supported by various MPI implementations including Open MPI. There are three ways to use the Fortran language binding: “USE mpi_f08”, “USE mpi”, and “INCLUDE mpif.h”. Among these, only “USE mpi_f08” conforms to the Fortran standard, and its use is strongly recommended. The characteristics of each method are as follows.

- “USE mpi_f08” supports the Fortran 2008 standard and later, and is the only usage method compliant with the Fortran standard. This method is strongly recommended.
- “USE mpi” is positioned as a method supporting standards prior to Fortran 2008, but it is not consistent with the Fortran standard and exists for backward compatibility.
- “INCLUDE mpif.h” has existed since the MPI-1 specification and is positioned as a method supporting the Fortran 77 standard. However, it was deprecated in MPI-3.0 and is planned for removal in MPI-4.1.

3.3.3.1.6.1.2 C Language Binding

In the MPI specification, the C language binding has been defined since MPI-1.0, similar to the Fortran language binding, and is supported by various MPI implementations including Open MPI. Unlike Fortran, the fundamental language specification is stable, and there are no particular considerations regarding its usage.

3.3.3.1.6.1.3 C++ Language Binding

The C++ language binding in the MPI specification was introduced in MPI-2.0 in 1997. However, it was deprecated in MPI-2.2 in 2009 and completely removed from the MPI specification in MPI-3.0 in 2012. The main reasons for its removal are as follows.

- The C++ language binding provides little added value over the C language binding.
- Maintenance cost is high.
- Name mangling in C++ is not standardized, making binary distribution difficult.

Additionally, in Open MPI, the C++ language binding is no longer supported starting from v5.0.0 released in 2023. When specifying the configure option “--enable-mpi-cxx” in Open MPI v5.0.0 or later, a configure error occurs and the build fails. Note that Open MPI v4.1.8 (the previous generation release still supported as of December 2025) supports the C++ language binding. Therefore, as long as Open MPI v4.1.x is supported, it is still possible to support the C++

language binding by using two generations appropriately. As described above, the C++ language binding has been removed from both the MPI specification and the latest version of Open MPI. Therefore, it is considered necessary to revise FugakuNEXT to exclude support for the C++ language binding.

3.3.3.1.6.1.4 Java Language Binding

The Java language binding is not part of the MPI specification and is specific to Open MPI [17]. Even in Open MPI v5.0.9 (the latest version as of December 2025), the configure option "--enable-mpi-java" exists and is supported. For Java, it is necessary to carefully manage risks to prevent Oracle JDK (commercial product) from being unintentionally included due to licensing issues.

- Pricing models (Oracle Java SE Universal Subscription) and policies (Oracle Technology Network (OTN) License Agreement for Java SE) have been revised multiple times.
- It is also necessary to consider how to reliably identify and manage users who utilize MPI Java Binding.

In addition, OpenJDK adopts a type of GPL license called "GPLv2 with the Classpath Exception", which also requires attention. However, this includes the Classpath Exception, meaning that if OpenJDK itself is not embedded in the MPI library and is only linked at runtime, it is not subject to GPLv2. As described above, since the Java language binding is an Open MPI-specific feature outside the MPI specification, it may be removed in the future depending on user demand and community direction. Furthermore, if supported, special care is required for license management. Based on these considerations, it is deemed necessary to revise FugakuNEXT to exclude support for the Java language binding.

3.3.3.1.6.1.5 Other Language Bindings

- Python Language Binding
  - The Python language binding is not included in the MPI specification. In Open MPI v5.0.0 and later, the configure option "--enable-python-bindings" has been added, but this enables Python bindings for PMIx and does not provide Python bindings for MPI. However, although not part of the MPI specification, MPI for Python (mpi4py) [20] is widely used as the de facto standard for MPI Python bindings. mpi4py follows MPI standards, and version 4.1.0 released in 2025 added support for MPI-5.0. It is expected that mpi4py will be used as the Python language binding in FugakuNEXT.
- Julia Language Binding
  - Julia is a programming language that combines ease of use similar to Python with execution speed comparable to C and Fortran, and has been gaining attention in the HPC field. Julia language binding is also not part of the MPI specification, but MPI.jl [17] exists as the de facto standard OSS implementation. In FugakuNEXT, it is expected that MPI.jl will be used when utilizing MPI from Julia programs.

3.3.3.1.7 Accelerator Integration

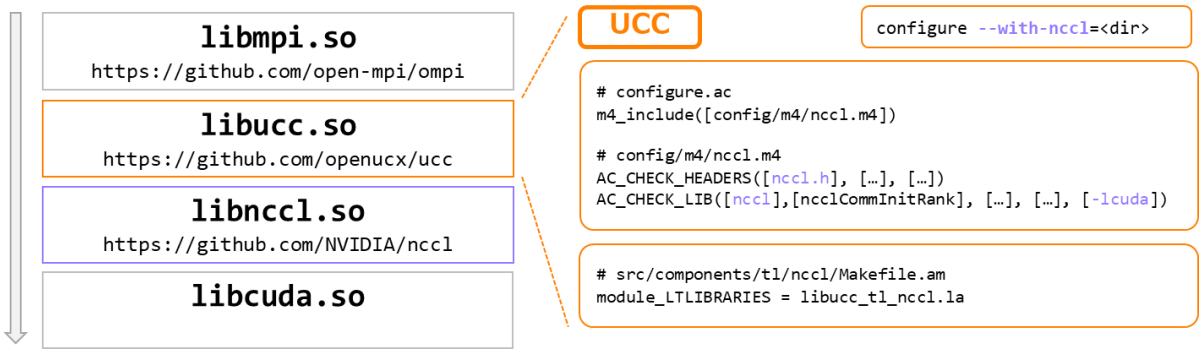
3.3.3.1.7.1 Inter-GPU Communication Library Integration

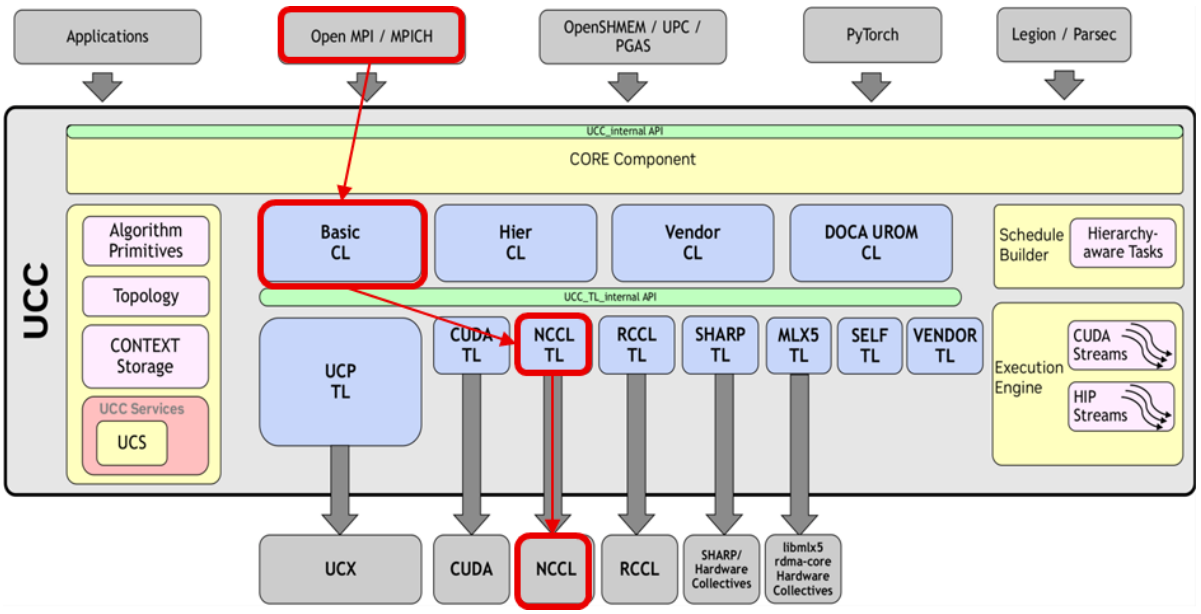
Collective communications such as Allreduce and Alltoall remain essential operations for applications, regardless of whether user data resides in host memory or GPU device memory. However, depending on the physical memory system, memory characteristics and hierarchy (internal topology) differ significantly, and the implementations of collective communication, including communication algorithms, also differ greatly. As a result, various xCCL (Collective Communication Library) implementations exist in addition to traditional general-purpose collective communication provided by MPI implementations. A key issue is how MPI implementations should configure these diverse collective communication libraries and how

applications should select and invoke them through MPI. It should be noted that xCCL is developed in a vendor-driven manner and continues to evolve and diversify. Although currently under discussion in related WAs/WGs, in the basic design, assuming NVIDIA-based GPUs for the accelerator, NCCL [21] is considered a de facto standard. As an example of accelerator integration, the invocation of NCCL, the collective communications supported by NCCL, and methods for enabling and using UCC and NCCL are presented.

3.3.3.1.7.2 Integration of Open MPI and NCCL

The integration between Open MPI and the inter-GPU collective communication library NCCL is described below.





The abbreviations in the above figure are as follows: CL: Collective Layer, TL: Team Layer, NCCL: Optimized primitives for inter-GPU communication (https://github.com/NVIDIA/nccl). UCC supports invoking NCCL, which is a collective communication library for inter-GPU communication. Open MPI does not have a direct implementation for calling NCCL and instead invokes it via UCC.

3.3.3.1.7.3 Collective Communications Supported by NCCL

<coll>	! root	root
allgather	ncclAllGather()	n/a
allgatherv	ncclGroupStart / ncclSend / ncclRecv / ncclGroupEnd	n/a
allreduce	ncclAllReduce()	n/a
alltoall	ncclGroupStart / ncclSend / ncclRecv / ncclGroupEnd	n/a
alltoallv	ncclGroupStart / ncclSend / ncclRecv / ncclGroupEnd	n/a
bcast	ncclBroadcast()	あり
reduce_scatter	ncclReduceScatter()	n/a
reduce_scatter_block	n/a	n/a
reduce	ncclReduce()	あり
barrier	Call allreduce with count = 1	n/a
gather	ncclSend	ncclGroupStart / ncclRecv / ncclGroupEnd
gatherv	ncclSend	ncclGroupStart / ncclRecv / ncclGroupEnd
scatter	ncclRecv	ncclGroupStart / ncclSend / ncclGroupEnd
scatterv	ncclRecv	ncclGroupStart / ncclSend / ncclGroupEnd

The above shows ucc_tl_nccl_<coll>_init() for the collective communications supported by NCCL. NCCL supports eight types of collective communications, and the others are complemented by ncclSend/ncclRecv.

3.3.3.1.7.4 Example of Enabling and Using UCC and NCCL

- Example of execution with UCC disabled in Open MP

```
mpirun -mca coll_ucc_enable=0 ¥
  osu_allreduce -d cuda -m 1000000000:1000000000
```

- Example of execution with UCC enabled in Open MP

```
mpirun -mca coll_ucc_enable=1 -mca coll_ucc_priority=100 ¥
  -x UCC_TL_NCCL_TUNE=allreduce:cuda:0 ¥
  osu_allreduce -d cuda -m 1000000000:1000000000
```

- Example of execution with UCC enabled in Open MP

```
mpirun -mca coll_ucc_enable=1  -mca coll_ucc_priority=100 ¥
  -x UCC_TL_NCCL_TUNE=allreduce:cuda:inf ¥
  osu_allreduce -d cuda -m 1000000000:1000000000
```

Option	Description
coll_ucc_enable	Equivalent to a feature gate
coll_ucc_priority	Priority for selecting UCC
-d cuda	Calls cudaMalloc() for memory allocation
UCC_TL_NCCL_TUNE	Priority per collective communication

The above shows options related to UCC and NCCL. By using NCCL, when data resides in GPU memory, it is expected that data movement will be minimized.

3.3.3.1.7.5 Integration of Communication Libraries Between CPU and GPU

Reducing data movement and global synchronization between the CPU and GPU (accelerator) is fundamental when handling large-scale data. In practice, to avoid stalling compute units on the accelerator, it is effective to suppress unnecessary data movement—including consideration of data locality—to mitigate memory bandwidth degradation, and to reduce idle time caused by synchronization waits.
In general, collective communication implementations can be built on top of low-level point-to-point communication libraries such as UCX. In the context of GPU-aware MPI [22], the conditions under which accelerator-specific transports within such point-to-point communication libraries operate, as well as their communication methods and performance characteristics across accelerator generations, are not clearly visible from MPI collective communication calls.
To address this issue, it is important to evaluate the characteristics of accelerator-specific transports in point-to-point communication libraries while aligning with future accelerator device information. Although currently under discussion in related WAs/WGs, assuming NVIDIA-based GPUs for the accelerator, UCX is considered a candidate GPU-aware point-to-point communication library, and further investigation of its integration is required.

3.3.3.1.8 Storage Integration

MPI IO provides not only an abstraction API for file resources (MPI_File_*) but also optimizations such as (1) Data Sieving, (2) Two-Phase I/O, and (3) File-system specific optimization (for Lustre: stripe count, stripe size, stripe offset). These techniques help reduce file system load by converting many small accesses into fewer large accesses.
In the Fugaku supercomputer, Fujitsu implemented MPI specifically for its file systems (FEFS, LLIO). In FugakuNEXT, usage will be considered according to the adopted file system implementation.
In Open MPI, both ompio [23] and ROMIO have been used as MPI IO components. However, the MPICH project has discontinued support for the standalone ROMIO library because it does not support the MPI_Count extension introduced in MPI Standard 4.0. Therefore, FugakuNEXT is expected to use the ompio component.
As for communication libraries, support will be considered in the detailed design phase after the storage WG determines the file system policy. The following section presents an example of integration when using the Lustre file system.

3.3.3.1.8.1.1 Integration of MPI IO and Lustre

Lustre [24] is licensed under GPLv2.0. When MPI IO integrates with Lustre, including Lustre headers causes GPLv2.0 to propagate to the MPI library. While using a GPLv2.0 MPI library does not propagate the license to user applications, distributing the MPI library together with the application may cause GPLv2.0 to propagate to the application, requiring careful attention. Integration with Lustre is achieved by including Lustre-provided headers during MPI library build time, enabling MPI IO to work with Lustre.

3.3.3.1.9 Integration with Communication Benchmark Tools

As a programming model, MPI implementations provide APIs that abstract various computing resources and serve as an aggregation point for many system software components beyond communication libraries. As a result, their structure is generally complex and wide-ranging.

Over time, individual system software components that make up MPI may be replaced or deprecated. In the short term, these libraries undergo frequent updates, and with the advancement of continuous integration and continuous delivery, release cycles have shortened, often occurring every three months. Consequently, MPI implementations—being integration points of many system software components—require increasing levels of release validation while continuing to evolve through proper release and quality management.

At the same time, MPI implementations are used by many performance-sensitive applications, and performance degradation in backend libraries can propagate to applications. It is therefore necessary to detect such degradation early through performance validation at each stage from libraries to applications. Particularly in unique, one-of-a-kind supercomputer systems, identifying the root cause of performance degradation—often specific to a particular application on that system—has historically required significant cost.

To address this issue, tools known as continuous benchmarking have emerged. For example, Benchpark [25] allows specification at three levels: system specification, benchmark specification, and experiment specification (measurement conditions for a specific benchmark on a given system), enabling customization for unique supercomputer systems.

In the basic design, comparisons were made among benchmarks included in Benchpark—GPCNeT, OMB, and SMB—with OMB used as an example.

- GPCNeT (Global Performance and Congestion Network Test)
  - https://github.com/netbench/GPCNET
  - Network congestion benchmark
    - network_test: benchmark without congestion
    - network_load_test: benchmark with congestion (80% of nodes act as congestors)
  - Originally used for analyzing adaptive routing (automatic detouring of congested links between switches)
  - Not intended for application developers

- OMB (Ohio State University Micro Benchmarks)
  - https://mvapich.cse.ohio-state.edu/benchmarks
  - GPU-capable general-purpose MPI benchmark (replacement for IMB: Intel MPI Benchmarks)
  - Includes benchmarks for point-to-point communication (10 types), collective communication (54 types), and one-sided communication (9 types), including support for both host and GPU memory

- SMB (Sandia MPI Micro-Benchmark Suite)
  - https://github.com/sandialabs/SMB
  - Benchmark suite for more realistic point-to-point communication performance
    - msgrate: point-to-point benchmark without cache effects
    - mpi_overhead: point-to-point benchmark under overlapping computation and communication

Among these three MPI benchmark tools, OMB [26] covers almost all MPI standard collective communications (22 routines × 3 types), except for MPI_{Scan, Exscan}. The list of OMB-7.5.1 MPI benchmark programs is shown below. SMB and GPCNeT are considered complementary tools.

collective / API	collective / blocking	collective / non-blocking	collective / persistent	Remarks
MPI_Allgather	osu_allgather	osu_iallgather	osu_allgather_persistent	
MPI_Allgatherv	osu_allgatherv	osu_iallgatherv	osu_allgatherv_persistent	
MPI_Allreduce	osu_allreduce	osu_iallreduce	osu_allreduce_persistent	
MPI_Alltoall	osu_alltoall	osu_ialltoall	osu_alltoall_persistent	
MPI_Alltoallv	osu_alltoallv	osu_ialltoallv	osu_alltoallv_persistent	
MPI_Alltoallw	osu_alltoallw	osu_ialltoallw	osu_alltoallw_persistent	UCC API ×
MPI_Barrier	osu_barrier	osu_ibarrier	osu_barrier_persistent	
MPI_Bcast	osu_bcast	osu_ibcast	osu_bcast_persistent	
MPI_Exscan	(not supported)	(not supported)	(not supported)	UCC API ×
MPI_Gather	osu_gather	osu_igather	osu_gather_persistent	
MPI_Gatherv	osu_gatherv	osu_igatherv	osu_gatherv_persistent	
MPI_Reduce	osu_reduce	osu_ireduce	osu_reduce_persistent	
MPI_Reduce_scatter	osu_reduce_scatter	osu_ireduce_scatter	osu_reduce_scatter_persistent	
MPI_Reduce_scatter_block	osu_reduce_scatter_block	osu_ireduce_scatter_block	(not supported)	
MPI_Scan	(not supported)	(not supported)	(not supported)	UCC API ×
MPI_Scatter	osu_scatter	osu_iscatter	osu_scatter_persistent	
MPI_Scatterv	osu_scatterv	osu_iscatterv	osu_scatterv_persistent	
MPI_Neighbor_allgather	osu_neighbor_allgather	osu_ineighbor_allgather	(not supported)	UCC API ×
MPI_Neighbor_allgatherv	osu_neighbor_allgatherv	osu_ineighbor_allgatherv	(not supported)	UCC API ×
MPI_Neighbor_alltoall	osu_neighbor_alltoall	osu_ineighbor_alltoall	(not supported)	UCC API ×
MPI_Neighbor_alltoallv	osu_neighbor_alltoallv	osu_ineighbor_alltoallv	(not supported)	UCC API ×
MPI_Neighbor_alltoallw	osu_neighbor_alltoallw	osu_ineighbor_alltoallw	(not supported)	UCC API ×
22 関数	*20 ベンチ*	*20 ベンチ*	*14 ベンチ*	

point-to-point	one-sided
osu_bibw	osu_acc_latency
osu_bw	osu_cas_latency
osu_latecy	osu_fop_latency
osu_latency_mp	osu_get_acc_latency
osu_latency_mt	osu_get_bw
osu_mbw_mr	osu_get_latency
osu_multi_lat	osu_put_bibw
osu_bibw_persistent	osu_put_bw
osu_bw_persistent	osu_put_latency
osu_latency_persistent	
10 ベンチ	*9 ベンチ*

3.3.3.1.9.1 General Communication Benchmark

At present, one of the well-known communication benchmark software packages is the Ohio State University Micro Benchmarks (OMB) [26]. In OMB, for MPI point-to-point communication, one-sided communication, and collective communication operations, there are 10 benchmarks, 9 benchmarks, and 54 benchmarks respectively, even when considering only C language call programs. Each benchmark has also been extended with options for so-called GPU-aware MPI, which specify user communication buffers on GPU device memory. In addition, OMB is included as a benchmark in Benchpark and is becoming a standard communication benchmark. An example of OMB usage is shown below.

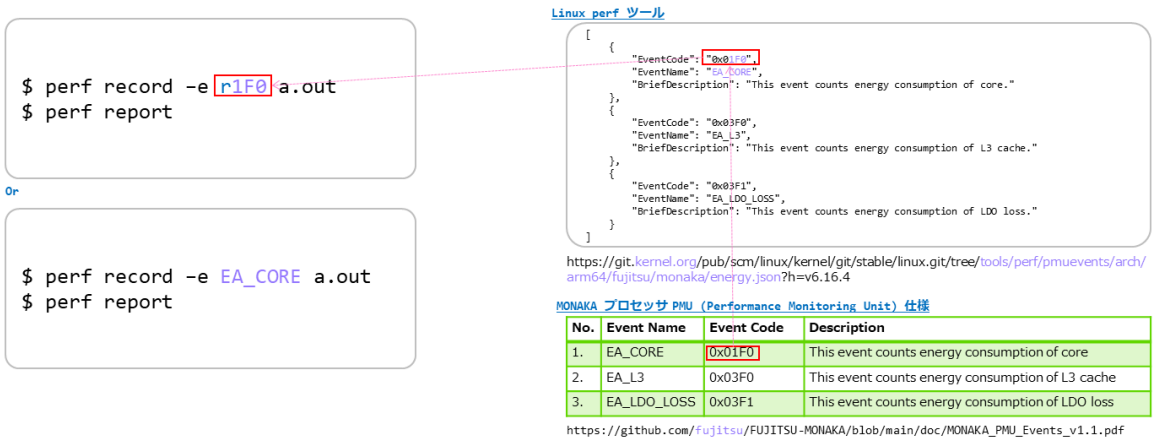
```
$ mpirun -n 2 osu_reduce --iterations=100 --warmup=15 ¥
    --message-size=$((32*1024*1024)):$((32*1024*1024)) --type mpi_float

# OSU MPI Reduce Latency Test v7.5
# Datatype: MPI_FLOAT.
# Size       Avg Latency(us)
33554432            11466.67


$ mpirun -n 2 osu_reduce --iterations=100 --warmup=15 ¥
    --message-size=$((32*1024*1024)):$((32*1024*1024)) --type mpi_float --full

# OSU MPI Reduce Latency Test v7.5
# Datatype: MPI_FLOAT.
# Size       Avg Latency(us)   Min Latency(us)   Max Latency(us)  Iterations
33554432            11252.42          11209.48          11295.35         100
```

In OMB, each process executes collective communication for a specified number of iterations (iters), aggregates the total elapsed communication time (timer), and calculates the (average) time per iteration (latency). Finally, MPI_Reduce with MPI_{MIN, MAX, SUM} is used to report the minimum, maximum, and average (avg_time) across processes. OMB continues to be actively extended, with rapid evolution and support, and has grown into a very large suite.

3.3.3.1.10 Fine-Grained Power Consumption Visualization

When acquiring power consumption of compute nodes, it is generally done using OS drivers (e.g., acpi_power_meter) that access power sensors placed on system boards via ACPI.

However, such measurements are typically at the board level or, when multiple sensors exist, at the socket level, and cannot capture finer granularity such as per CPU core. Therefore, it is common to estimate per-core power consumption using CPU Performance Monitoring Unit (PMU) events. In addition, many power sensors have update intervals on the order of milliseconds, which leads to accuracy issues when measuring power at finer granularity such as tens to hundreds of microseconds (e.g., function-level measurements). As a result, power measurement has been limited to very restricted scenarios such as evaluating individual computational kernels.
In A64FX, PMU events have been extended, and A64FX-specific Energy Analyzer (EA) events have been added. EA enables measurement of energy consumption (nJ) at the core level. EA is also planned to be supported in successor processors to A64FX [27]. However, the exact energy value (nJ) per EA counter increment is expected to be determined after 2027. An example of using the perf command is shown below.



The following shows performance counters related to power consumption in the FUJITSU-MONAKA processor PMU. The FUJITSU-MONAKA processor PMU provides seven performance counters related to power consumption, and by specifying them in the perf command, power visualization can be achieved.

No.	Event Name	Event Code	Description
1.	EA_CORE	0x01F0	This event counts energy consumption of core
2.	EA_L3	0x03F0	This event counts energy consumption of L3 cache
3.	EA_LDO_LOSS	0x03F1	This event counts energy consumption of LDO loss
4.	EA_MAC	0x0080	This event counts energy consumption of MAC
5.	EA_MEMORY	0x0090	This event counts energy consumption of memory
6.	EA_HA	0x00A0	This event counts energy consumption of HA
7.	EA_PCI (un-core)	0x0080	This event counts energy consumption of PCI

3.3.3.1.11 Items for Consideration and Policy in Detailed Design

No.	Item	Policy
1	Extraction of development items for communication libraries related to system- and hardware-dependent components	With the finalization of the FugakuNEXT system architecture, identify the items to be developed in the communication library. Once the system network, interconnect, topology, and other architectural elements are determined, examine architecture-specific items to be developed from a performance perspective.
2	Continuation of technology trend investigation	Continue investigating the technology trends described in this report and extract items to be developed.
3	Extraction of development items for communication libraries related to the TWI of the communication library WG	Identify items to be developed based on application evaluations in benchmark software development, which is a TWI of the communication library WG.

For item 1, the scope of consideration will be adjusted when the scale-out network, switches, and topology are determined in the detailed design phase. For item 2, identify areas where OSS bug fixes or performance improvements are likely to occur, extract items for investigation and development, determine priorities, and carry out development starting from higher-priority items. For item 3, continue investigations through discussions within the communication library WG.

3.3.3.2Accelerator

3.3.3.2.1 Communication Software for the Accelerator

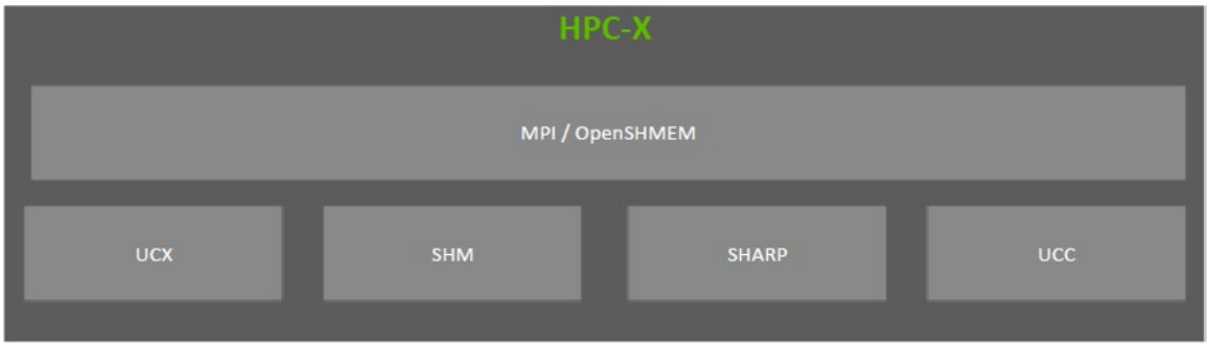
- MPI/OPENSHMEM: Supports conventional HPC application
- NCCL: Supports all AI use cases and some HPC use cases, including collective communication, point-to-point communication, and GPU-initiated one-sided communication
- NVSHMEM: Provides GPU-initiated communication, including one-sided and collective communication
- Widely used in AI workloads, especially for Mixture-of-Experts (MoE) parallelization
- Used in HPC applications such as Gromacs and numerical computation libraries
- NIXL: A general-purpose I/O transfer library for memory and storage
- UCC: Provides HPC collective operations for both MPI and OpenSHMEM
- UCX: A general-purpose low-level communication library that enables libraries such as MPI, OpenSHMEM, and NIXL to operate over NVIDIA networking

3.3.3.2.2 Technical Work Items (TWI) for the Accelerator

- Library and driver support: Provide stable and optimized versions of NCCL, UCX, MPI, and GPU drivers, along with hooks for telemetry collection
- Extension of benchmark tools: Enhance vendor-provided benchmark tools with new NCCL collectives and UCX transport options
- Exposure of performance counters: Provide telemetry data from NIC/GPU/CPU and enable integration into benchmarking workflows
- Collaboration in troubleshooting: Jointly analyze and address benchmark anomalies potentially caused by firmware, drivers, or communication libraries
- Roadmap alignment: Share updates on future library releases, offload capabilities, and adaptive routing mechanisms, and align them with benchmarking scope

3.3.3.2.3 HPC-X

HPC-X is a communication software package for HPC provided by NVIDIA and includes communication frameworks for MPI and OpenSHMEM. HPC-X includes implementations of MPI and OpenSHMEM, which are built on NVIDIA's high-performance RDMA communication middleware. In the MPI and OpenSHMEM implementations included in HPC-X, point-to-point communication, RMA (Remote Memory Access), and collective communication are implemented using UCX, UCC, and SHARP. These communication middleware components are designed to transparently leverage network offload capabilities and accelerators.



HPC-X plays a key role in maintaining MPI semantics and portability. MPI is a standard with a vast amount of existing application code. In FugakuNEXT, it is required to enable these existing codes to run naturally while achieving high performance. HPC-X makes this possible by incorporating the latest networking and GPU communication capabilities through UCX, while providing a stable implementation compliant with the MPI-3 standard. GPUDirect support in HPC-X is essential for many GPU-enabled applications and is also necessary for the gradual migration of CPU-only code to GPUs.

On the other hand, HPC-X is not a universal solution. For AI workloads that require top-tier collective communication performance, CUDA stream awareness, or GPU-initiated communication, NCCL or NVSHMEM is required. Therefore, in FugakuNEXT, HPC-X is expected to serve as the foundation for HPC applications, while AI-oriented communication is primarily handled by other libraries such as NCCL. These libraries are mutually compatible and can be used together in applications that require both functionalities or depend on libraries that do so.

From a benchmarking perspective, the evaluation criteria for HPC-X differ from those for NCCL. For HPC-X, the main evaluation targets include MPI-based point-to-point communication, collective communication, RMA performance, and GPU memory communication performance as GPU-aware MPI. In addition to peak performance, it is necessary to evaluate scalability with increasing node counts, stability across diverse communication patterns, and differences depending on the presence of network offloading. To reflect these aspects, conventional MPI benchmarks (e.g., OMB) need to be extended to match the actual configuration of FugakuNEXT. NCCL, on the other hand, provides its own collective communication benchmarks.

From the TWI perspective, HPC-X should be continuously evaluated using an automated communication benchmarking framework. As a foundational library, its performance and stability are directly linked to the reliability of the entire system. Therefore, rather than relying on manual evaluation for each configuration change or library update, an automated benchmarking framework capable of detecting performance degradation and anomalies at an early stage is required.

As design considerations, it is necessary to define which MPI implementation will be standard in FugakuNEXT and how the support scope of HPC-X will be defined. If HPC-X is adopted as the default MPI implementation, its update policy, coexistence with other MPI implementations, and impact on applications must be clearly defined. Another design consideration is the extent

to which OpenSHMEM will be used and supported. These issues will be continuously discussed among RIKEN, NVIDIA, and Fujitsu during the detailed design phase.

3.3.3.2.3.1 NCCL

NCCL (NVIDIA Collective Communications Library) is a GPU communication library provided by NVIDIA. It was originally developed mainly for collective communication and point-to-point receive operations and is supported by a large ecosystem consisting of AI frameworks, GPU hardware, cloud services, and AI companies. In the latest version, NCCL 2.29, host-initiated communication and device-initiated communication have been added. This development is based on the success of NVSHMEM in AI workloads such as DeepEP and aims to enable AI applications to execute all types of communication using a single communication library.

- The communication operations provided by NCCL are as follows.
  - Collective communication: AllReduce, Reduce, ReduceScatter, Broadcast, Gather, Scatter, AllGather, AllToAll
  - Point-to-point communication (bidirectional and one-sided): Send/Recv, Put/PutSignal/WaitSignal
  - Symmetric memory (NVL support): load/store access, multicast pointer, barrier
  - Device capabilities: GPU-initiated networking (GIN): put, flush, signal, barrier

  These communication operations run on GPU memory and function in coordination with the CUDA driver and runtime. NCCL operates over NVIDIA GPU interconnects including PCIe, NVLink, InfiniBand/RoCE, and third-party networks. It is used directly not only by HPC applications and libraries but also by deep learning frameworks such as PyTorch, JAX, and NeMo. NCCL sits between deep learning frameworks and CUDA, handling inter-GPU communication.

- NCCL includes the following functionalities.
  - Topology detection
  - Communication path discovery
  - Performance tuning
  - Buffer registration
  - CUDA kernels and various functions
  - Communication transports (shared memory, CUDA peer-to-peer, network)

  NCCL supports both intra-node and inter-node GPU communication. Within a node, it uses NVLink and PCIe, and across NVLink domains it utilizes RDMA-capable networks such as InfiniBand and RoCE. NCCL can also leverage network offloading mechanisms such as SHARP.

- Considerations for NCCL benchmarks
  From a benchmarking perspective, key evaluation axes include the performance of major collective operations such as AllReduce and Alltoall, scalability with increasing GPU counts, differences between NVLink generations, and behavior differences depending on the presence of network offloading. Additionally, how to handle the gap between communication patterns used in real applications and those represented by synthetic benchmarks such as nccl-tests is an important operational consideration. These points will be discussed in more detail in the detailed design phase.

3.3.3.2.3.2 NVSHMEM

NVSHMEM is a communication library for GPU clusters provided by NVIDIA, primarily targeting one-sided communication and collective communication. NVSHMEM pioneered the concept of GPU-initiated communication. This mechanism enables a level of integration between

computation and communication that cannot be achieved with host-initiated communication. While in MPI and OpenSHMEM communication operations are issued by the CPU or framework, a key feature of NVSHMEM is that GPU kernels themselves can directly issue communication operations. The effectiveness of this concept has been demonstrated in NVSHMEM and, as mentioned earlier, has recently been incorporated into NCCL as well. This design enables compute–communication fusion, allowing computation results generated within GPU kernels to be transferred immediately to other GPUs without explicit CPU involvement. As a result, reductions in latency and overhead can be expected. NVSHMEM adopts a one-sided communication model, which is fundamentally different from the bidirectional communication model commonly used in MPI. Therefore, NVSHMEM is more naturally suited to workloads with irregular communication patterns or many small messages. Accordingly, NVSHMEM's strengths in irregular communication and communication-intensive AI workloads such as MoE are considered to align with the workload characteristics expected in FugakuNEXT.

- Communication models provided by NVSHMEM
  - One-sided communication initiated by CPU and GPU
  - Collective communication initiated by CPU and GPU

  NVSHMEM is interoperable with MPI and OpenSHMEM and can be used in a hybrid manner combining MPI communication and NVSHMEM communication. The NVSHMEM API is divided into host APIs and device APIs, with the latter callable directly from GPU kernels. These APIs enable put operations and collective communication operations between GPU memory regions. NVSHMEM is used in both AI and HPC workloads.

- Representative examples are as follows.
  - AI training and inference using Mixture-of-Experts (MoE)
  - GROMACS
  - cuFFTMp
  - QUDA
  - LBANN

  NVSHMEM has played an important role as a library supporting communication patterns that were difficult to express with NCCL. However, NCCL has recently begun adding device-initiated communication and one-sided communication features, and the differences between NVSHMEM and NCCL are expected to shift toward other characteristics such as simplicity, flexibility, and extensibility. The NVSHMEM model is very clean and simple, but it does not support multiple communicators or resizing of the default communicator. On the other hand, it provides many communication functions including collective communication and one-sided point-to-point communication, all of which are available from both host and device. Additionally, the operational model for users is unified regardless of the environment, and programmers do not need to be aware of differences between NVLink domains and networks. NVSHMEM has characteristics that differ significantly from MPI, and these differences are in many cases similar to those between NCCL and MPI. Therefore, not all applications are suited to NVSHMEM, and it is expected to be used in combination with MPI/OpenSHMEM (HPC-X) and NCCL.

- Benchmarking perspective

  For host-initiated communication, NVSHMEM can be directly compared with other communication APIs. On the other hand, for device-initiated communication, it is difficult to define objective comparison metrics. For NVSHMEM communication issued from within GPU kernels, it is necessary to evaluate not only communication performance alone but also end-to-end performance combining computation and communication. Specific benchmarking methodologies will be defined through discussions in the detailed design phase.

3.3.3.2.3.3 NXIL

NIXL (NVIDIA I/O eXchange Library) is a general-purpose data transfer library that plays a role distinct from communication libraries such as MPI, NCCL, and NVSHMEM within NVIDIA's proposed communication and data movement stack. NIXL does not provide a full set of communication semantics including collective communication like MPI. Instead, it is designed to handle transfers across diverse data domains including memory and storage under a unified abstraction.

The design of NIXL adopts a clear separation of layers, with a North-Bound NIXL API in the upper layer and a South-Bound Backend API and multiple transfer backends in the lower layer. Applications and upper middleware specify only logical information such as source, destination, and data size, while the selection of actual transfer paths and mechanisms is delegated to NIXL. Expected lower-layer transfer backends include UCX (or libfabric), POSIX, GPUDirect Storage (GDS), GPU-NetIO, and custom backends for storage systems and object stores. NIXL integrates these backends through a plugin-based architecture. In the context of FugakuNEXT, NIXL is associated with inference serving frameworks such as Dynamo and SGLang that users may deploy on the system. In particular, it is an effective means for implementing AI patterns such as KV cache that require support across memory and storage hierarchies.

From a benchmarking perspective, NIXL is not an entity that can be evaluated using the same metrics as NCCL or MPI. While NCCL and MPI are mainly evaluated in terms of communication bandwidth, latency, and scalability, the primary evaluation axis of NIXL lies in the behavior of complex data movement across memory and storage domains. Specifically, important considerations include throughput between GPU and CPU memory, throughput between GPU memory and NVMe, overhead across intra-node and inter-node transfers, performance stability when using repost, effects of congestion and contention during large-scale concurrent transfers, and the presence of performance drift or resource leaks during long-term execution. These characteristics cannot be captured by single-run microbenchmarks and require automated long-duration, multi-configuration evaluation. From this perspective, automated benchmarking infrastructure and frameworks for visualization and long-term trend analysis should focus on inference serving frameworks (e.g., Dynamo) that capture the end-to-end performance enabled by NIXL.

As a design consideration, it is necessary to clearly define how or whether NIXL will be used in FugakuNEXT. If inference serving frameworks utilizing NIXL are to be deployed, the implementation policy of NIXL must be clarified. If UCX is included in the low-level network stack, the operational principles of NIXL are relatively clear.

3.3.3.3Advanced

This section summarizes the fundamental concepts and design considerations that should be organized prior to the detailed design phase for communication libraries in FugakuNEXT. Communication libraries are a foundational software layer closely related to the overall system configuration, execution environment, and operational conditions. Fixing designs that strongly depend on specific hardware configurations or usage patterns at an early stage is not appropriate from the perspective of future performance and extensibility.

Trends in communication libraries and their organization from system-wide, CPU, and GPU perspectives have been covered in the previous sections and related documents (basic design reports from Fujitsu and NVIDIA), and are assumed as prerequisites here. The focus here is not to determine a specific communication library configuration or implementation method, but rather to organize design philosophies, decision criteria, and considerations necessary to properly extract communication performance under hardware configurations and execution environments that will be determined in the future. Specifically, this section aims to provide guidelines for the detailed design phase by organizing approaches to understanding communication performance and isolating issues, as well as design considerations based on execution environments and operational conditions.

3.3.3.3.1 Approach to understanding communication performance and isolating issues

Communication performance is determined by multiple factors, including not only the implementation of communication libraries but also system configuration, execution environment, and operational conditions. Therefore, when performance degradation or variability is observed, it is important not to assume a single cause but to incorporate a framework for step-by-step isolation and understanding of contributing factors into the design in advance.

In this design, evaluation of communication performance is positioned not as a means to determine the performance of communication libraries alone, but as a design tool for understanding overall system behavior and identifying the location of issues. Application-level evaluation, communication profiling, and communication benchmarks each provide information from different perspectives, and should be used complementarily rather than relying on any single method.

Furthermore, understanding communication performance and isolating issues should not be treated as a one-time task but should be continuously carried out during both detailed design and operation phases. Therefore, this design assumes that communication benchmarks will be used not only for one-time evaluation but also in a form that enables continuous execution and result comparison.

Based on this concept, during the basic design phase, an environment was established on GH200 systems using Benchpark to enable execution of representative communication benchmarks such as OMB (OSU Micro-Benchmarks), allowing continuous verification of basic communication library behavior under consistent conditions. Similar benchmarking environments are expected to be incrementally developed for FugakuNEXT performance evaluation systems.

These communication benchmarks are not intended to evaluate communication performance in isolation but are positioned as auxiliary tools for understanding how differences in system configuration, execution environment, and operational conditions affect communication behavior. Continuous use of communication benchmarks is expected to facilitate trend analysis of performance changes and early-stage issue isolation, thereby supporting detailed design and operational phases.

3.3.3.3.2 Design considerations for communication considering execution environment and operational conditions

Communication performance is affected by operational conditions such as job placement policies, co-allocation, OS noise, and memory management methods. Therefore, communication library design should not assume that the execution environment is always optimal for communication.

In this design, elements with high dependency on execution environments are consciously separated within communication processing to prevent their effects from propagating across the entire communication functionality. On the other hand, aspects that are difficult to control solely within the communication library, such as storage configuration, job scheduler behavior, and detailed OS behavior, are not assumed to be solved within the communication library alone, but are instead delegated to operations or higher layers.

Additionally, execution environments and operational conditions are not fixed at system deployment and may change over time due to changes in usage patterns or operational policies. Communication design should therefore assume such changes and avoid configurations that strongly depend on specific operational conditions.

In particular, job placement policies and resource sharing approaches can contribute to variability in communication performance. This design assumes that such factors will be

absorbed through configuration settings, operational adjustments, and coordination with higher layers, ensuring that they do not impose direct constraints on the fundamental behavior of communication libraries.

3.3.3.3.3 Handover items for communication design (toward detailed design phase)

The fundamental concepts and design considerations for communication have been organized, and prerequisite conditions for the detailed design phase have been established. Specific configurations of communication libraries, optimization methods tailored to hardware characteristics, and implementation policies will be examined after the details of hardware configuration and execution environment are determined.

In the detailed design phase, discussions will proceed based on the following principles summarized in this section:

- Communication performance is not determined by a single factor, and should be approached through step-by-step understanding and issue isolation
- Communication benchmarks are not for performance evaluation alone but should be positioned as design tools for understanding behavior and isolating issues
- Design philosophy should maintain separation between communication library core behavior and performance variability caused by execution environments and operational conditions

Based on these principles, detailed design will be advanced step by step while considering consistency with other design elements such as computation, memory, I/O, and operations.

The communication library is intended to provide a foundational software layer that delivers communication functionality to parallel applications, taking into account differences in communication methods and execution environments, and enabling appropriate communication performance under hardware configurations that will be determined in the future. In this basic design phase, numerical communication performance metrics (latency, bandwidth, etc.) based on specific hardware configurations or execution conditions are not defined. These performance targets will be examined and established in the detailed design phase after clarifying conditions such as hardware configuration, execution environment, and communication patterns, and it is assumed that performance validation will be conducted under those defined conditions.

3.3.4 AI Software

3.3.4.1 Overall System / CPU

3.3.4.1.1 Positioning of This Document

This document summarizes the basic design of the software stack—specifically frameworks and libraries—for AI workloads on the CPU component, which falls under Fujitsu's responsibility in the FugakuNEXT project. Layers other than frameworks and libraries are treated as follows:

- The upper application layer is within the scope of the codesign working group. However, to clarify the use cases assumed by this software stack, cases where CPU utilization is expected are described in Section 3.3.4.1.2.
- Lower system software layers (job management, container management, OS, etc.) fall under the scope of the Operations System and Environment Working Group and are treated as independent topics; therefore, they are not covered in this document.

3.3.4.1.2 Assumed Use Cases

This section outlines expected CPU use cases in AI processing. AI workloads are broadly divided into training and inference. Training use cases are described in Section 3.3.4.1.2.1, inference in Section 3.3.4.1.2.2, and AI–HPC integration—specific to FugakuNEXT—in Section 3.3.4.1.2.3. Fine-tuning is included in Section 3.3.4.1.2.1 as it shares similar considerations with training.

3.3.4.1.2.1 AI Model Training / Fine-Tuning

Training workloads require high computational performance due to batch processing of large datasets and are typically executed on accelerators such as GPUs. However, surrounding processes are handled by CPUs, and in some cases CPU performance becomes the bottleneck. Assumed use cases:

- Reinforcement learning reward computation
  - CPU executes simulation or search processes used for reward calculation and may become the bottleneck
- Surrogate model training
  - GPU handles AI training, while CPU handles HPC simulation
  - In some cases, roles may be reversed
- Other cases
  - Lightweight fine-tuning,Data preprocessing and postprocessing

3.3.4.1.2.2 AI Inference

Inference workloads are smaller than training and may benefit from CPU execution depending on workload characteristics

- Cases where CPU outperforms GPU
  - Small workloads where GPU overhead dominates such as small-batch inference or lightweight models
  - Workloads unsuitable for GPU architectures such as branch-heavy algorithms
- Cases where CPU improves system performance
  - Multiple models of different sizes coexist within the system, with smaller models handled by the CPU (e.g., embedding models in RAG, draft models in speculative decoding)
  - Performance can be improved by having the CPU handle part of the inference processing (e.g., LLM decode processing, agents or experts in MoE and MoA [4] that do not fit in GPU memory

- High-load processing other than AI coexists within the system, and tasks are divided between CPU and GPU (e.g., when big data analysis is required as preprocessing for inference)

3.3.4.1.2.3 3. Fusion of HPC and AI

FugakuNEXT positions AI for Science as one of its concepts, and the fusion of HPC and AI is one of the important themes. Specifically, the roles played by HPC and AI vary depending on system requirements, but in this document the following use cases are assumed.

- AI for HPC application running: AI generates input data and execution conditions for HPC computations.
- AI for HPC application programming: AI assists in programming HPC applications.
- AI Surrogate for HPC: HPC computations are replaced by AI processing.
- HPC for AI training: Training data for AI is generated through HPC computations.
- HPC for AI agentic workflows: AI agents dynamically generate and launch HPC computation tasks via MCP servers, etc.

3.3.4.1.3 AI software stack

This section describes specific software stacks to realize the use cases in 3.3.4.1.2. The CPU software stack is composed only of OSS. The GPU software stack is outside the scope of this document as it is under the responsibility of NVIDIA. However, software that enables cooperative operation between CPU and GPU is important from the perspective of CPU co-design and is therefore included in this document. Some of these software components include NVIDIA open binaries.

The software stack is described in tabular form, and the table consists of the following elements.

- Software name: Name of the software.
- Usage: Purpose of the software, such as Training, Inference, Fine-Tuning, MLOps, etc.
- Layer: Layer within the software stack.
- Device: One of CPU, CPU/GPU, CPU&GPU.
  - CPU: Software used when performing AI processing on CPU.
  - CPU/GPU: Software used when performing AI processing on CPU, but also supports GPU and can be used in the same way for AI processing on GPU.
  - CPU&GPU: Software used for cooperative operation between CPU and GPU.
- Function: Main function of the software.
- Arm support: One of ◎, ○, △.
  - ◎: Software that Fujitsu has previously run on Arm systems. Not all functions are verified; it indicates successful execution for at least one input pattern. Performance is not considered.
  - ○: No execution record at Fujitsu, but support for Arm systems is declared by the software developer (OSS community, etc.).
  - △: It is currently unknown whether it runs on Arm systems. These software components will be verified during the detailed design phase, and if they do not run, alternatives with equivalent functionality will be considered.

The software stack in this document consists of mainstream OSS as of January 2026, and may be replaced by other software with equivalent functionality at the time of FugakuNEXT operation. Software versions are not explicitly specified, as it is appropriate to always use the latest versions. If required OSS does not support the FugakuNEXT CPU or does not fully utilize its capabilities, it is necessary to engage with OSS communities via issues or pull requests to improve functionality and performance.

3.3.4.1.3.1 Software for machine learning

This section presents software whose primary use is machine learning. In addition to commonly used software such as Scikit-Learn, NumPy, and Pandas, distributed parallel software such as DASK and Modin is also included for efficient use of large-scale computing resources.

Software Name	Usage	Layer	Device	Function	ARM Support
XGBoost	Data Analysis	Framework	CPU	Implementation of gradient boosting decision trees	◎
Scikit-Learn	Data Analysis	Framework	CPU	Machine learning tools and algorithms	◎
Statsmodels	Data Analysis	Framework	CPU	Python library for statistical modeling, testing, and data exploration	○
NumPy	Data Analysis	Library	CPU	Python library for numerical computation	◎
oneDAL	Data Analysis	Library	CPU	Optimized library for data analysis and machine learning	◎
SciPy	Data Analysis	Library	CPU	Python library for scientific computing	◎
DASK	Data Analysis	Parallel Computing	CPU	Parallel computing library for large-scale tabular data	◎
Modin	Data Analysis	Parallel Computing	CPU	Library that parallelizes and accelerates Pandas workflows	◎
Pandas	Data Analysis	Toolkit	CPU	Python library for data manipulation and analysis	◎

3.3.4.1.3.2 Software for deep learning

This section presents software whose primary use is deep learning (LLM-related software is described in 3.3.4.1.3.3, and surrogate model software in 3.3.4.1.3.4). In addition to deep learning frameworks such as PyTorch and TensorFlow, inference runtimes such as OpenVINO and ONNX Runtime are included for CPU inference. Libraries such as Arm Compute Library and oneDNN are also included to maximize CPU performance.

Software Name	Usage	Layer	Device	Function	ARM Support
OpenXLA	Inference	Compiler	CPU	Computational graph compiler for machine learning and deep learning	○
OpenVINO	Inference	Framework	CPU	General-purpose inference runtime framework	◎
ONNX Runtime	Inference	Framework	CPU/GPU	General-purpose inference runtime framework	◎
JAX	Inference	Library	CPU	Numerical computation library with automatic differentiation and JIT compilation	○
Triton Inference Server	Inference	Serving	CPU/GPU	General-purpose AI inference server	◎
diffusers	Inference	Toolkit	CPU/GPU	Toolkit for using diffusion models	○
PyTorch	Training Inference	Framework	CPU/GPU	Deep learning framework	◎
TensorFlow	Training Inference	Framework	CPU/GPU	Deep learning framework	◎
Arm Compute Library	Training Inference	Library	CPU	Deep learning library for Arm CPU	◎
oneDNN	Training Inference	Library	CPU	Deep learning library	◎
Eigen	Training Inference	Library	CPU/GPU	Linear algebra template library	○
DeepSpeed	Training Inference	Parallel Computing	CPU/GPU CPU&GPU	Library for distributed training and inference of large-scale models	△

3.3.4.1.3.3 Software for LLM

This section presents software for model training and inference in LLMs. It includes inference runtimes such as llama.cpp and vLLM, as well as fine-tuning tools such as trl and peft

Software Name	Usage	Layer	Device	Function	ARM Support
trl	Fine Tuning	Application	CPU/GPU	Fine-tuning tool for LLM using reinforcement learning	○
peft	Fine Tuning	Application	CPU/GPU	Fine-tuning tool for LLM models	○
llama.cpp	Inference	Framework	CPU/GPU CPU&GPU	Low-bit inference runtime framework for LLM	◎
Kleidi	Inference	Library	CPU	Low-bit matrix computation library	◎
Ollama	Inference	Serving	CPU	LLM inference server for GGUF format	○
Text Embeddings Inference	Inference	Serving	CPU/GPU	Inference server for embedding models	△
Text Generation Inference	Inference	Serving	CPU/GPU	Inference server for LLM	△
vLLM	Inference	Serving	CPU/GPU CPU&GPU	LLM inference server	◎
transformers	Inference	Toolkit	CPU/GPU	Toolkit for using transformer models	◎

3.3.4.1.3.4 Software for surrogate models

This section presents software for surrogate models, including GNN frameworks such as DGL and PyTorch Geometric, as well as scientific deep learning tools like PhysicsNeMO.

Software Name	Usage	Layer	Device	Function	ARM Support
DGL	Training Inference	Framework	CPU/GPU	Framework for GNN development	◎
PyTorch Geometric	Training Inference	Framework	CPU/GPU	GNN framework implemented in PyTorch	◎
PhysicsNeMO	Training Inference	Toolkit	CPU/GPU	Deep learning toolkit for scientific computing	◎

3.3.4.1.3.5 Software for MLOps/LLMOps

This section presents software for MLOps and LLMOps, including tools for managing AI model lifecycle and software for building LLM-based applications and AI agents.

Software Name	Usage	Layer	Device	Function	ARM Support
MinIO	MLOps	Application	CPU	High-performance S3-compatible object storage	○
MLFlow	MLOps	Application	CPU	Machine learning lifecycle management platform	○
LangChain	LLMOps	Framework	CPU	Framework for developing LLM-based applications	○
LangGraph	LLMOps	Framework	CPU	Framework for constructing and executing LLM workflows	○
LlamaIndex	LLMOps	Framework	CPU	Framework for LLM-based applications	△
Dify	LLMOps	Application	CPU	Platform for visual development and operation of LLM apps	○
lm-evaluation-harness	LLMOps	Application	CPU/GPU	LLM evaluation tool	△
LMCache	LLMOps	Application	CPU/GPU	KV cache management tool for LLM	△
OpenAI Agents SDK	LLMOps	Library	CPU	Python agents SDK for OpenAI-compatible APIs	△
LiteLLM	LLMOps	Server	CPU	Proxy server for LLM inference	△
DSPy	Prompt	Application	CPU	Prompt optimization tool	△
TextGrad	Prompt	Application	CPU	Prompt optimization tool	△
Milvus	VectorDB	Application	CPU	Vector database	◎
FAISS	VectorDB	Library	CPU	Library for vector databases	◎
OpenHands	AI Agent	Application	CPU	AI agent for coding and browser operations	△
AutoGen	AI Agent	Application	CPU	Platform for multi-agent AI development	△

3.3.4.1.3.6 Common libraries

This section presents commonly used libraries that serve as backends for applications and frameworks.

Software Name	Usage	Layer	Device	Function	ARM Support
Numba	Backend	Compiler	CPU/GPU	Python acceleration library	○
ArmPL	Backend	Library	CPU	Matrix computation library for Arm CPU	◎
OpenBLAS	Backend	Library	CPU	Matrix computation library	◎
OpenMP	Backend	Library	CPU	Multithreaded parallel computing library	◎
Open MPI	Backend	Library	CPU	Multiprocess parallel computing library	◎
OpenRNG	Backend	Library	CPU	Parallel random number generation library for scientific computing	○
oneTBB	Backend	Library	CPU	Multithreaded parallel computing library	◎

3.3.4.1.4 AI-related items in other WGs

AI-related considerations are also being conducted in other working groups. The main initiatives are as follows.

WG	Initiative
Co-design Working Group	AI use cases are included as part of hardware co-design considerations. Important AI-related use cases are also described in Chapter 2 of this document.
Operations system and user environment WG	Coexistence of HPC job-queue-based systems and container orchestration systems such as Kubernetes is being considered.
Numerical libraries and middleware WG	Feasibility of performance improvements such as SME2 support for AI computation libraries (e.g., Arm Kleidi) is being studied.

3.3.4.2GPU

This chapter describes the NVIDIA software stack, including AI frameworks and application domains.

3.3.4.2.1 AI frameworks, tools, libraries, SDKs

The major AI frameworks, tools, libraries, and SDKs are listed below.

- NVIDIA NGC Container Registry
  A container registry optimized for GPU-accelerated AI, HPC, and data analytics. Includes frameworks such as PyTorch and TensorFlow.
  https://catalog.ngc.nvidia.com/
- NVIDIA Deep Learning Software Stack
  A collection of GPU-accelerated libraries and SDKs designed to optimize deep learning training and inference workloads.
  https://developer.nvidia.com/deep-learning-software
- TensorRT / TensorRT-LLM

A high-performance inference runtime that optimizes trained models, including large language models, for NVIDIA GPUs.
https://developer.nvidia.com/tensorrt

- NVIDIA NeMo
  A framework for building, training, and customizing generative AI models such as LLMs, speech, and multimodal models.
  https://www.nvidia.com/en-us/ai-platforms/nemo/
- NVIDIA NIM (Inference Microservices)
  A set of microservices that package AI models as optimized and scalable API endpoints.
  https://www.nvidia.com/en-us/ai-platforms/nim/
- NVIDIA AI Enterprise
  An enterprise AI software platform providing supported frameworks, tools, and infrastructure for production AI.
  https://www.nvidia.com/ja-jp/data-center/products/ai-enterprise/
- Dynamo
  A software engine and orchestration layer designed to enable large-scale and efficient AI inference on NVIDIA platforms.
  https://developer.nvidia.com/blog/inside-the-nvidia-rubin-platform-six-new-chips-one-ai-supercomputer/

3.3.4.2.2 Python-based coding

The Python-based software components are listed below.

- cuPyNumeric
  A NumPy-compatible distributed array library that accelerates numerical computing on GPUs and multi-node environments.
  https://docs.nvidia.com/cupynumeric/latest/user/usage.html
- CUDA Python
  A set of tools that enable direct access to CUDA APIs and GPU acceleration from Python workflows.
  https://developer.nvidia.com/cuda/python

3.3.4.2.3 Visualization, digital twin

- Omniverse
  A platform for building and simulating physically accurate digital twins using real-time 3D and USD-based workflows.
  https://www.nvidia.com/ja-jp/omniverse

3.3.4.2.4 Representative AI software stacks for scientific domains

Representative AI and HPC software stacks for AI for Science are shown in the following sections by domain:

- Weather and climate
- Materials science
- Life sciences
- Industry / manufacturing / society / digital twin
- Open models for science

3.3.4.2.4.1 Weather and climate

Typical computation patterns are as follows:

1. Grid-based PDE (finite difference / finite volume / finite element)

2. Stencil computation (3D / 4D)
3. Spectral methods using FFT
4. Large-scale MPI + GPU (strong scaling / weak scaling)
5. Checkpoint / restart

Representative AI methods are organized as follows:

1. Deep learning-based weather prediction
   - End-to-end prediction of atmospheric variables using CNN, Transformer, and Graph Neural Networks
   - Used as surrogate models or complements to numerical weather prediction (NWP)
2. Spatiotemporal models for nowcasting
   - Short-term (minutes to hours) prediction of precipitation and severe weather
   - Based on radar/satellite data using CNN-RNN or Transformer
3. Physics–AI hybrid models
   - Combination of physical models and machine learning
   - Bias correction and replacement of computationally expensive subgrid models
4. AI-accelerated data assimilation
   - Approximation of assimilation steps using machine learning
   - State estimation using neural operators and surrogate models
5. Super-resolution / downscaling
   - Spatiotemporal resolution enhancement of coarse weather/climate data
   - Often implemented using CNN or diffusion models
6. Extreme weather and disaster prediction
   - Probabilistic prediction of typhoons, heavy rain, heatwaves, etc.
   - Combined with uncertainty quantification
7. Ensemble forecasting emulation
   - AI-based generation and compression of ensemble forecast
   - Enables fast probabilistic prediction at lower computational cost
8. Observation data processing and quality control
   - Automated analysis of satellite, radar, and in-situ data
   - Includes denoising, anomaly detection, and missing data completion
9. Physics-informed neural networks (PINNs)
   - Neural networks trained with physical constraints
   - Used for PDE approximation and physically consistent prediction
10. Neural operators for atmospheric dynamics
    - Operators mapping between function spaces
    - Useful for large-scale atmospheric flow modeling
11. Uncertainty quantification
    - Bayesian neural networks and ensemble-based method
    - Estimation of prediction uncertainty and risk
12. Climate pattern recognition
    - AI-based identification of large-scale climate patterns
    - Applied to long-term analysis and trend detection
13. Decision support AI
    - Integration of forecasting with optimization/recommendation systems
    - Used in disaster response, energy, agriculture, and transportation

NVIDIA software examples

- NVIDIA HPC SDK
- Earth-2 open models
  https://www.nvidia.com/en-us/high-performance-computing/earth-2/
  - FourCastNet: Achieves high prediction accuracy for multiple weather variables and is up to 60× faster than diffusion-based approaches.

- CorrDiff: Downscales continental-scale predictions to high-resolution regional forecasts up to 500× faster.
- Earth-2 Medium Range: Provides accurate forecasts up to 15 days across 70+ variables.
- Earth-2 Nowcasting: Generates high-resolution short-term forecasts within minutes
- Earth-2 Global Data Assimilation: Generates global atmospheric snapshots for initial conditions.

- PhysicsNeMo : An open-source Python framework for building, training, and fine-tuning physics AI models at scale. It provides utilities for constructing AI surrogate models that combine causality based on physical laws with simulation data and observational data, enabling real-time prediction. It also supports the development and validation of large-scale digital twin models across various physics domains, including computational fluid dynamics (CFD), structural mechanics, and electromagnetics.
  https://developer.nvidia.com/physicsnemo

3.3.4.2.4.2 Materials science

Typical computation patterns:

- Molecular dynamics (short/long-range interactions)
- Electronic structure calculations (sparse/dense solvers, eigenvalue problems)
- Particle-in-cell and Monte Carlo methods

Representative AI methods:

1. ML-based interatomic potentials
   - Neural network or graph-based models trained on DFT/MD data
   - Enable large-scale molecular dynamics simulations while maintaining accuracy close to DFT
2. AI-accelerated molecular dynamics
   - Integration of machine learning force fields into conventional MD workflows
   - Enable analysis over longer time scales and larger atomic systems
3. Surrogate modeling of physics simulations
   - Replace high-cost FEM, phase-field, and mesoscale simulations with AI models
   - Used for fast evaluation in materials design and optimization
4. Electronic property prediction
   - Machine learning models predicting bandgap, formation energy, and density of states
   - Often based on graph neural networks or message-passing architectures
5. High-throughput screening
   - Narrow down candidate materials from large compositional spaces using AI
   - Used in combination with automated DFT and database-driven workflows
6. Inverse material design
   - Propose new materials using generative models such as VAE, GAN, and diffusion models
   - Optimized for target properties and performance metrics
7. Structure–property learning
   - Discover correlations between atomic structures and properties in a data-driven manner
   - Support interpretability and hypothesis generation
8. Multiscale/multiphysics modeling
   - Bridge atomic, mesoscale, and continuum scales using AI
   - Used for predicting microstructure evolution and macroscopic behavior
9. Physics-informed ML
   - Neural networks trained with embedded physical constraints and governing equations
   - Improve robustness and physical consistency of predictions
10. Uncertainty quantification and active learning
    - Evaluate prediction uncertainty using Bayesian machine learning and ensemble methods

- Construct active learning loops for efficient generation of additional training data

11. Materials imaging analysis
    - Deep learning applied to microscopy images and tomography data
    - Perform segmentation, phase identification, and defect detection
12. Process optimization and control
    - Optimization of synthesis, processing, and manufacturing parameters
    - Integration of AI with experimental and simulation workflows

NVIDIA software examples

- NVIDIA HPC SDK
- NVIDIA ALCHEMI
  https://developer.nvidia.com/blog/revolutionizing-ai-driven-material-discovery-using-nvidia-alchemi/

3.3.4.2.4.3 Life sciences

Typical computation patterns:

- Molecular dynamics for proteins
- Medical image analysis
- Omics analysis
- Biomedical LLM/multimodal processing

Representative AI methods:

1. Protein structure prediction
2. AI-enhanced molecular dynamics
3. Drug discovery and virtual screening
4. Generative models for drug/protein design
5. Omics data analysis
6. Medical imaging
7. Multimodal data integration
8. Biological sequence LLMs
9. Single-cell and spatial biology analysis
10. Systems biology modeling
11. Uncertainty quantification
12. AI-driven experimental design

NVIDIA software examples

- NVIDIA HPC SDK
- Clara open models
  https://github.com/nvidia/clara
- BioNeMo Framework
  https://github.com/NVIDIA/bionemo-framework

3.3.4.2.4.4 Industry, manufacturing, society, digital twin

Typical computation patterns:

- CAE
- Surrogate models
- Design space exploration
- Digital twins

Representative AI methods:

1. Predictive maintenance
2. Quality inspection
3. Process optimization
4. Digital twin simulation
5. Anomaly detection
6. Supply chain optimization
7. Robotics
8. Generative design
9. Energy optimization
10. Human–machine collaboration
11. NLP-based systems
12. Closed-loop learning

NVIDIA software examples

- NVIDIA HPC SDK
- PhysicsNeMo
- Omniverse

3.3.4.2.5 Open models for science

Representative computation patterns:

- Multimodal scientific data analysis
- Scientific LLMs
- PINNs / surrogate models
- Bayesian models
- Inference pipeline optimization

NVIDIA software examples

- Inference optimization
  - TensorRT-LLM
    https://developer.nvidia.com/tensorrt
  - Dynamo

- Training
  - PyTorch / TensorFlow (NGC)
  - CUDA-X libraries (cuDNN, cuBLAS, cuTENSOR)
  - NCCL
- Generative AI
  - NeMo
  - NIM

3.3.4.2.6 Future collaboration

RIKEN and NVIDIA have agreed to collaborate on AI for Science. As a first step, five priority domains are defined, and NVIDIA will participate jointly:

1. Open foundation models for science
2. Domain science: physics
3. Domain science: bioscience
4. Domain science: materials science
5. Quantum computing

The knowledge obtained will be fed back into the FugakuNEXT AI software working group and reflected in architecture and software design.

3.3.4.3Advanced Components

This chapter organizes the core requirements of software and hardware necessary to realize AI capabilities in the FugakuNext system.

3.3.4.3.1 Requirements for CPU, Network, and System Software

This section focuses on the MONAKA CPU, dedicated interconnect, and vendor-provided system software required to enable modern AI workloads.

3.3.4.3.1.1 Support for System Software and Middleware

Requirements for system software are largely discussed in other chapters, but here the connectivity with AI applications is emphasized again.
To support agentic workflows and modern MLOps, system software must evolve beyond traditional batch-oriented operation models. In particular, containerization is essential, and a pipeline that enables secure “Build Once, Run Anywhere” must be provided.
Even if the system is designed around high-performance HPC-oriented formats assuming bare-metal execution (e.g., Apptainer/SIF), the ability to convert and run standard OCI/Docker images is essential to ensure compatibility with the AI ecosystem.
Modernization of orchestration strategies is also required. Whether adopting a “Cloud-over-HPC” approach, in which a temporary Kubernetes cluster is deployed on top of standard Slurm, or an “HPC-over-Cloud” approach, in which a Slurm controller runs on cloud infrastructure such as Kubernetes, the resource manager must provide an API-driven interface.
This enables higher-level decision-making systems such as MLOps platforms like Kubeflow and autonomous agents to programmatically submit and monitor jobs without relying on manual CLI operations.
Furthermore, the data layer must be capable of efficiently handling AI datasets that often consist of billions of small files. Since traditional POSIX file systems are not well suited for such workloads, it is necessary to integrate S3-compatible object storage as a first-class element of the storage hierarchy.

3.3.4.3.1.2 CPU (MONAKA) and Hardware Architecture

For the MONAKA CPU to function effectively as a cooperative processor for AI, Scalable Matrix Extension (SME) and Scalable Vector Extension (SVE) must be fully integrated within standard AI frameworks.

- Framework equivalence and synchronization: Major frameworks such as PyTorch, TensorFlow, and JAX must provide CPU backends that are functionally equivalent to the NVIDIA GPU stack and maintain version consistency. This enables researchers to execute workflows without being aware of the device, or to run hybrid workflows without version conflicts.
- Benchmarking: Fujitsu should provide comprehensive performance benchmarks for key computational primitives, focusing on FP16 and INT8 matrix operations, which are critical for Transformer architectures and emerging models.

This architecture must support a Unified Memory model to enable “Memory-Extended AI.” This allows high-bandwidth, zero-copy access from GPU to CPU main memory, enabling workloads that exceed GPU HBM capacity, such as massive embedding tables and large-scale graph analytics.
Furthermore, the runtime must support “Heterogeneous Split Compute,” enabling low-latency synchronization between CPU and GPU without the performance penalty of conventional PCIe

transfers, where the CPU handles complex branching logic and the GPU handles dense numerical computation.

For inference use cases, MONAKA should support the following two distinct execution patterns:

1. Capacity-driven inference: A method in which the large weights of large language models (LLMs) are offloaded to system memory and layers are streamed to the GPU as needed.
2. Inline/surrogate model inference: A method in which pre-trained surrogate models are continuously executed within running numerical simulations (C++/Fortran host code) using lightweight frameworks optimized for inference (e.g., ONNX Runtime, dedicated C++ libraries). This “inline inference” must be achieved with minimal overhead so as not to stall the main simulation loop.

3.3.4.3.1.3 Network and Interconnect

To ensure efficient distributed training, communication libraries optimized for the FugakuNext dedicated fabric are required. These libraries should provide APIs equivalent to NCCL and support low-latency collective communication essential for multi-node training and inference.

Data ingestion requires robust and dedicated infrastructure to handle large-scale datasets used in modern AI. The design should include data ingestion nodes equipped with high-performance CPUs and large memory buffers. These nodes must have external network (WAN) interfaces, independent of the internal cluster fabric.

On the software side, support for high-performance parallel data transfer tools such as Globus and Apache Kafka is required to automate and optimize large-scale data movement from external sources to in-cluster object storage.

3.3.4.3.1.4 Vendor Support and Integration

Close collaboration with vendors is essential to enable smooth porting. Early access to hardware prototypes, emulators, or performance simulators should be provided to enable early model adaptation. Additionally, collaborative efforts in performance estimation—such as estimating Blackwell-generation GPU-equivalent performance based on current execution traces—are required.

The compiler toolchain (compiler, profiler, debugger) should be integrated with formal methods. In particular, systems such as TADASHI must support polyhedral model extraction to enable correctness-guaranteed code transformations. Furthermore, support for JIT compilers via JAX and Numba is required to handle dynamic kernels in scientific AI applications.

3.3.4.3.1.5 Collaboration Across Working Groups

The development of this system software requires close collaboration with other working groups in FugakuNEXT. In particular, orchestration and containerization strategies must align with the Operations and User Environment WG, which manages coexistence between HPC queues and container orchestrators.

AI usage requirements are directly reflected in hardware co-design through the Codesign Review WG, while the Numerical Libraries and Middleware WG evaluates the performance feasibility of AI-oriented computational libraries such as Arm Kleidi.

3.3.4.3.1.6 Inference and Workloads Where CPU Has Advantages

While GPUs are the primary drivers of large-scale training, the MONAKA CPU must be actively utilized for workloads where it can outperform GPUs from mathematical or operational perspectives.

Specifically, architectural optimization is required for workloads with relatively small computational volume compared to GPU overhead (such as small-batch inference and small

language models (SLM)), as well as workloads unsuitable for GPU architectures that involve frequent branching (e.g., random forests).
In addition, the CPU should handle efficient execution of reward computation in reinforcement learning (e.g., via SmartSim), and management of small auxiliary models in hybrid configurations, such as embedding models in RAG and draft models in speculative decoding.

3.3.4.3.1.7 OSS Ecosystem and Arm Compatibility

To realize the CPU-oriented AI workloads described above, the system must provide a robust, precompiled open-source software (OSS) stack optimized for Arm.
Specifically, validated native support is required for foundational machine learning and data analytics frameworks such as XGBoost, Scikit-Learn, NumPy, Pandas, and Modin.
Furthermore, native support is required for deep learning and LLM-related components such as OpenVINO, llama.cpp, and vector databases (Milvus, FAISS), as well as surrogate model toolkits such as DGL, PyTorch Geometric, and PhysicsNeMO.

3.3.4.3.2 Requirements for GPU

This section focuses on the GPU acceleration layer, related libraries, and integration with the broader ecosystem, and outlines the requirements for FugakuNext to maintain competitiveness among world-leading AI supercomputers.

3.3.4.3.2.1 GPU Acceleration and Libraries

The platform must provide standard and optimized support for key computational primitives through libraries such as cuBLAS, cuDNN, and CUTLASS, and ensure full compatibility with major training frameworks such as PyTorch and TensorFlow.

3.3.4.3.2.2 Inference Software and Engines

NVIDIA must ensure robust support for high-performance inference engines such as vLLM and TensorRT-LLM. This software layer is critical and must natively support the following modern optimization features:

- Continuous batching to achieve maximum throughput
- Paged Attention for efficient memory management
- Prompt caching (e.g., Radix Tree) to accelerate multi-turn agents and RA
- Speculative decoding to reduce generation latency

These inference engines must be optimized for the specific hardware configuration of FugakuNext.
Furthermore, to support AI-for-Science, vendors should provide lightweight inline inference frameworks.

- Surrogate model execution: Libraries that can be directly embedded into high-performance simulation codes (C++/Fortran) and execute surrogate models such as PINNs and emulators with ultra-low latency.
- Zero-overhead integration: The framework must minimize context-switch overhead and enable zero-copy data transfer between simulation state and the inference engine.

In addition to standard libraries, it is important to track emerging architectures. For example, if transitions to state space models (SSM) such as Mamba or energy-based models progress, vendors should rapidly provide kernel updates and benchmarks to meet these scientific needs. Additionally, transparent benchmarks comparing proprietary technologies such as CUDA with open standards such as OpenACC are required to evaluate portability strategies.

3.3.4.3.2.3 Integration with MONAKA

To ensure system coherence, vendors must achieve hybrid consistency. This refers to implementing CPU-GPU integration similar to the Grace-Hopper platform for the MONAKA environment, adapting features such as Unified Memory and GPU Direct Storage, and ensuring that NVIDIA toolchains interoperate seamlessly and effectively with Fujitsu hardware.

3.3.4.3.2.4 Realization of AI for Science

To meet AI-for-Science requirements, the software stack must be designed to align with the unique computational patterns and methodologies of major scientific domains.

Meteorology and Climate: Support is required for deep learning-based weather forecasting, spatiotemporal nowcasting, and super-resolution downscaling. Integration with open models such as Earth-2 (including FourCastNet, CorrDiff, StormScope) is necessary.

Materials Science: Acceleration of molecular dynamics and electronic structure calculations using machine learning-based interatomic potentials is required. Workflows for inverse materials design using generative models and high-throughput screening must be established.

Life Sciences: Infrastructure optimized for protein structure prediction, omics data analysis, and generative drug discovery is required. Integration of frameworks such as BioNeMo and Clara, along with open models (e.g., La-Proteina, RNAPro), should enable large-scale biomolecular simulations and multimodal data analysis.

Industry and Digital Twin: Tools such as Omniverse and PhysicsNeMO must be utilized to enable modeling of complex physical phenomena, predictive maintenance, generative design, and real-time digital twin simulations.

3.3.4.3.2.5 Collaborative Research and Future Roadmap

To ensure that the software architecture continuously meets cutting-edge research demands, RIKEN and NVIDIA have initiated a collaborative research initiative from the detailed design phase. This partnership jointly promotes pilot projects on open foundation models across five key domains: science, physics, life sciences, materials science, and quantum computing.
Insights and optimization strategies obtained from practical applications in each domain will be continuously fed back to the FugakuNEXT AI Software WG, reflected in hardware architecture design guidelines, and utilized for further enhancement of the software stack.

3.3.4.3.3 Requirements for RIKEN (Upper Software Stack and Usage Model)

This section defines requirements for the advanced layer, focusing on AI application stacks, managed services, and agentic workflows that sit above the infrastructure provided by Fujitsu/NVIDIA.

3.3.4.3.3.1 AI Services (Inference and Training)

Inference should be provided as Managed Inference as a Service. Based on vendor-provided optimized inference engines, RIKEN will build a service layer that allows access to version-controlled model catalogs via APIs (HTTP/gRPC). The service backend should handle autoscaling based on load and multi-tenant access management.
Particularly important is support for BYOM (Bring Your Own Model), enabling researchers to securely upload and serve their own models within sandboxed environments.
Similarly, fine-tuning should be provided as a managed service that abstracts infrastructure

complexity. Users should be able to configure distributed training and checkpoints via UI/API, supporting both PEFT (LoRA/QLoRA) and full fine-tuning. This requires a model optimization toolchain for quantization, pruning, and compilation targeting both NVIDIA GPUs and MONAKA SME.

3.3.4.3.3.2 AI-Assisted Development and Agentic Workflows

To support autonomous AI-driven development, dedicated agentic infrastructure is required. This corresponds to a “Level 3 abstraction” or “control plane” execution model, where agents operate through predefined wrappers such as submit rather than raw shell commands.
The system should provide mock targets (lightweight CPU-only environments) and ephemeral sandboxes (fast container execution environments) to enable compile-test cycles in under one minute.
As a hybrid inference strategy for code generation, advanced planning should use secure gateways to state-of-the-art models, while sensitive internal code generation should use self-hosted fine-tuned models.
Finally, a verification flywheel must be implemented to ensure code quality. This includes continuous performance validation using tools such as Benchpark, validation of optimizations based on measured metrics, and transformation of verification loops using correctness guarantees from TADASHI (polyhedral models). User training should focus on guiding these structured decisions and improving the ability to review AI-generated code, ensuring that humans remain involved in critical design decisions.

3.3.4.3.3.3 Pilot Projects for Improving Portability

While vendors are expected to provide kernels optimized for their respective hardware, it is important to avoid strong vendor lock-in to CUDA and instead aim for a cross-platform software stack that can run on diverse hardware such as AMD GPUs.
From this perspective, Mojo is considered a promising candidate. However, immediate migration of the entire codebase carries significant risk. Therefore, it is proposed to launch pilot projects to evaluate Mojo’s effectiveness. As part of this effort, a carefully selected subset of software should be chosen, balancing technical feasibility and potential performance gains.

3.3.4.3.3.4 Collaborative Research and Future Roadmap

To ensure that the software architecture continuously meets cutting-edge research demands, RIKEN and NVIDIA have initiated the detailed design phase of a collaborative research initiative. This partnership jointly promotes pilot projects on open foundation models across five key domains: science, physics, life sciences, materials science, and quantum computing. Insights and optimization strategies obtained from practical applications in each domain will be continuously fed back to the FugakuNEXT AI Software WG, reflected in hardware architecture design guidelines, and utilized for further enhancement of the software stack.

3.4 Storage and data management

3.4.1 Introduction

This chapter describes the purpose of this document, the basic design overview and policy, and storage requirements.

3.4.1.1 Purpose of This Document

This document is a report on the findings of the storage review committee for the basic design of the next-generation computing platform "FugakuNEXT." This report describes the basic design proposal for the overall system storage architecture, design target values, and their validity and rationale, based on trends in external storage device technology and performance looking ahead to around 2030.

3.4.1.2 Direction

3.4.1.2.1 Overview

The basic design of the "FugakuNEXT" storage system aims to explore storage system options optimal for the computing environment of the 2030s and to present their feasibility and validity in order to meet the usage needs of the scientific and technological research sector, the industrial sector, and society. A comprehensive survey and study was conducted on the storage architecture—both for the system as a whole and its constituent components—anticipating future prospects based on current technological standards.

Next-generation computing infrastructure requires a storage system capable of delivering ultra-parallel I/O performance, supporting data-intensive computing, and ensuring long-term operational stability.The storage system for "FugakuNEXT" is based on a hierarchical configuration consisting of high-speed local storage (Tier 1) directly connected to compute nodes and high-capacity storage (Tier 2) shared by all compute nodes, aiming to meet the performance requirements (bandwidth, IOPS, capacity) demanded by each tier.The basic design involved detailed and multifaceted studies of hardware and file system architectures, technologies to support the utilization of the storage system, and performance evaluation methods.

3.4.1.2.2 Policy

The basic policy for the research and study regarding the basic storage design of “FugakuNEXT” is not limited to a feasibility study; rather, it involves comprehensively obtaining roadmap information on future technologies from major storage vendors and conducting a comprehensive comparison and evaluation of potential candidates with an eye toward technologies expected around 2030. The basic design study was conducted in accordance with the following items, and details are provided in the relevant chapters.

Storage Architecture Study (Chapter 3.4.2): After clarifying the usage scenarios for “FugakuNEXT,” this chapter examines system configurations that meet the requirements for Tier 1 and Tier 2, the functions and performance requirements expected for operation and management, and the feasibility of implementation.

- Study of Storage System Utilization Support Technologies (Chapter 3.4.3): We will examine data transfer technologies between Tier 1 and Tier 2, methods for integrating with external storage, and tools for the operational management and performance analysis of storage systems.
- Examination of Performance Measurement Policies and Benchmarking Methods (Chapter 3.4.4): Examine performance measurement policies for the first and second levels, as well as benchmarking methods for file systems.

- Cost Estimation (Chapter 3.4.5): To accommodate multiple budgets and phased implementation, we will examine options for each technical element, with a focus on hardware.
- Handover to Detailed Design (Chapter 3.4.6): Based on the results of the basic design review, we will identify items requiring further refinement or decision-making in the detailed design phase.

Furthermore, in preparation for interviews with major storage vendors, a tripartite NDA was signed between RIKEN, Fujitsu, and the study group. To facilitate easy comparison of the interview results, the discussion points from this study group and the issues outlined in the "Report on the Results of Contracted Work for the Research Project on Next-Generation Computing Infrastructure" (hereinafter referred to as FS) were compiled into an interview sheet format, sent to each vendor, and responses were collected.A key achievement of using the interview sheets was the standardization of the level of detail in the information provided by each storage vendor.

3.4.1.3 FugakuNEXT Storage Requirements

This chapter maps the requirements in the FugakuNEXT Basic Design Specifications to the relevant sections of this report, as shown in Table 3 -24 .

Table 3-24 : Mapping of Specification Requirements and This Report

Specification Item			Specification Content	Section Number in This Report Chapter Number
General Information	3. Deliverables	(1) Final Report	Overall System Configuration (Overall configuration diagram, draft design specifications for compute nodes. Specifically regarding external storage devices, include the results of studies based on anticipated technologies and performance around 2030, along with their validity and rationale.)	3.4.2.1 3.4.2.3.1 3.4.2.4.1 3.4.5.2
	18. Terminology	(1) Units	Computational performance, storage capacity, and communication bandwidth shall be expressed in powers of 10; main memory capacity shall be expressed in powers of 2.	—
Technical Specifications	II-1 1. Development Requirements	5) Storage Requirements	The system shall include high-speed storage or cache available on compute nodes or for job-local use, as well as a storage system shared across all compute nodes, configured to achieve the required bandwidth, IOPS, and capacity at each tier. Furthermore, stable	3.4.2.1 3.4.2.3.1 3.4.2.4.1 3.4.4.1 3.4.4.2 3.4.5.2

			performance shall be maintained even during prolonged operation in actual production environments.	
	II-1 2. Overall System Configuration Specifications	(1) Configuration of “FugakuNEXT”	The system shall consist of a main system that primarily provides functions for processing application software, a front-end system that primarily provides functions for handling user tasks such as program development, and peripheral devices including a file system.	3.4.2.1
		(2) Overall Specifications of the Main System 5)	Each compute node must be able to access the file system.	3.4.2.1
		(3) Compute Node Specifications 10)	Assume a file system and hierarchical file system accessible from the compute node.	3.4.2.1
	II-1 2. (5) Peripheral Devices		Peripheral devices should include external storage devices and equipment for external network connections, as well as other functions necessary for the operation of the entire system.	3.4.2.1
		1)	External storage devices shall have a system configuration that takes into account access speed, capacity, and other factors. Access speed shall be measured as effective performance via the file system.	3.4.2.3.1 3.4.2.3.3 3.4.2.4.1 3.4.2.4.4 3.4.5.2
			① A two-tier hierarchical storage system based primarily on SSDs shall be implemented, allowing for phased expansion.	3.4.2.1 3.4.2.4.2
			② For the first tier, local storage shall be implemented, and the following performance targets shall be determined in the basic design: ・Bandwidth: Must be capable of writing the entire contents of a compute node’s memory in 2 minutes or less. Read performance must be equivalent or better. ・IOPS: Metadata processing (including file creation/deletion and read/write operations) must achieve 8K IOPS or more per single compute node.	3.4.2.3.1 3.4.2.3.3 3.4.5.2

			・Capacity: Must be at least three times the total memory size. ・For local storage, the first tier should consist of node-local storage installed on each compute node, as well as storage-dedicated nodes and near-node local storage shared across multiple compute nodes via the network.	
			③ For the second tier, implement shared storage accessible to all compute nodes, and determine the basic design with the following performance targets: ・Bandwidth: The entire contents of a compute node's memory must be writable to a Single Shared File via MPI-IO in 10 minutes or less. Read performance must be equivalent or better. ・Assumed Single Shared File method: A pattern in which each process accesses different regions of the same file. The file should be assumed to contain structured data of two or more dimensions (e.g., 2D data). ・ IOPS: The system must operate stably without interrupting I/O processing when I/O requests are made simultaneously from all nodes, achieving a performance of 1,000 IOPS or more per node. ・Capacity: Must be at least 30 times the total memory size. To optimize the balance between bandwidth, IOPS, capacity, and price, assume a two-tier storage system that uses SSDs as the primary storage while also incorporating HDDs.	3.4.2.4.1 3.4.2.4.4 3.4.5.2
		3)	Examine design approaches to maintain stable performance over long periods of operation, as well as benchmarking methods to quantitatively evaluate that stability, and document the results in the deliverables.	3.4.4.1 3.4.4.2
		4)	For the file system, use open-source software (OSS) as the foundation to enable continuous	3.4.2.3.4 3.4.2.4.5

			improvements based on user feedback.	
	II-2 1. 2 External Storage Devices	1)	Number of Levels and Implementation Method	3.4.2.1 3.4.2.3.1 3.4.2.4.1 3.4.3.1
	II-2 1. 2 External Storage Devices	2)	For each tier, consider the following factors based on the various budget scenarios and document the results in the deliverables a. Types of devices to be used b. Capacity, bandwidth, IOPS c. Redundancy methods d. Scalability and feasibility of phased implementation e. Power consumption f. Need for UPS g. Racks (standards, number of racks, rack size) and number of storage servers h. Budget breakdown	3.4.5.2
	II-2 1. 3 System Software	6)	Examine profiling methods for individual functions such as the overall application, accelerators, storage systems, and communication libraries; organize the necessary development items and document them in the deliverables.	3.4.3.3
	II-2 1.4 Installation Requirements	(2)	Examine the hardware configuration of peripheral devices, the number of racks, power requirements, etc., and document the results in the deliverables.	3.4.5.2
		(3)	Examine the power supply configuration for compute nodes and peripheral equipment (3-phase 4-wire 3P+N+E/415V or 480V) and document the results in the deliverables.	3.4.5.2
	II-4 System Specifications		Examine the following items and document the results in the deliverables. (1) Installation configuration, floor space, power consumption, cooling conditions, etc.	3.4.5.2

Additionally, from the FS studies conducted from FY2022 to FY2024, issues related to storage and file systems were identified and assigned management numbers (STK-xxx) in this basic design. The mapping between the FS issues and the corresponding sections in this report is provided at Table 3 -25 . In the basic design phase, vendor interviews were conducted for each issue to gather information. The examination of specific countermeasures will be carried out in the detailed design phase.

Table 3-25 Mapping of FS Issues and This Report

Management Number	Issue Description	Rationale for the Issue	Chapter Number in This Report
STK-001	SSF Performance Degradation (Increased Latency in Highly Parallel Jobs, Lock Acquisition Overhead)	FY 2023 User Interviews (asuca, Athena++, R2D2)	3.4.2.3.3 3.4.2.4.4
STK-002	Metadata Performance Degradation and I/O Errors (Performance degradation and evictions occur due to high-volume access to the same directory)	FY 2023 User Interviews (FFB/FFX, R2D2)	3.4.2.3.3 3.4.2.4.4
STK-003	File Open/Close Delays (Some responses delayed due to simultaneous access by multiple processes)	FY 2023 User Interviews (UTHeart, cause not yet investigated)	3.4.2.3.2 3.4.2.4.3
STK-004	Scaling issues during mass file creation (Limited to 2,000 nodes due to insufficient MDS performance)	FY 2023 User Interviews (Genomon)	3.4.2.3.3 3.4.2.4.4
STK-005	Insufficient system stability (IO errors caused by MDS load concentration, resulting in job failures and re-executions)	FY 2023 User Interviews (Multiple Applications)	3.4.2.3.3 3.4.2.4.4
STK-006	MPI-IO Asynchronous I/O Not Supported (Computation and I/O Cannot Overlap)	FY 2023 User Feedback (kinaco)	3.4.2.3.3 3.4.2.4.4
STK-007	Insufficient storage capacity and quota limits (restrict output and hinder the creation of results)	FY 2023 User Feedback (R2D2, kinaco)	3.4.2.3.2 3.4.2.4.3
STK-008	Users forced to modify applications due to I/O performance limits (risk of wasted computational results)	FY2024 User Interviews (Fugaku Users in General)	3.4.2.3.1 3.4.2.4.1
STK-009	Constraints of llio_transfer (cannot be placed on nearby SIOs, requires custom staging)	FY2023 User Interviews (NICAM)	3.4.3.1
STK-010	External Storage Transfer Load (Frequent transfers to HPCI shared storage or external research storage; insufficient speed)	FY2023 User Hearings (R2D2, kinaco)	3.4.3.2
STK-011	Storage system development has been completed, and modifications based on user feedback are not possible	Documented in the FY2024 Report	3.4.2.3.4 3.4.2.4.5
STK-012	Lack of a sustainable software development framework (modifications and improvements are necessary even after	As stated in the FY2024 Report	3.4.2.3.4 3.4.2.4.5

	deployment)		
STK-013	Storage requirements were not reflected in the hardware specifications, and performance is limited due to insufficient SSDs in I/O nodes	As stated in the FY2024 Report	3.4.2.3.1

3.4.2 Architecture

This chapter describes the overall configuration and use cases of the storage system, as well as the system configurations, expected functions, required performance, and selection of candidate file systems for both the Tier 1 and Tier 2 layers.

3.4.2.1Overall Architecture

The overall architecture of the "FugakuNEXT" storage system is shown in Figure 3-25 at .The system is configured as a two-tier hierarchical storage system, and each compute node (application) can access either the Tier 1 or Tier 2 storage system via a file system. In the figure, Tier 1 #1 through #3 indicate that the storage system's functions are selected based on the specific needs of each application. It is also possible for a single Tier 1 storage system to fulfill the roles of Tier 1 #1 through #3.

Data is transferred between the first and second tiers using staging and caching functions. Data is transferred to external storage, such as cloud storage, using data synchronization functions. Data transfer between tiers is described in Chapter 3.4.3.

Chapter 3.4.2 describes the hardware configuration, such as the devices used in Tier 1 and Tier 2, as well as considerations regarding the file system.

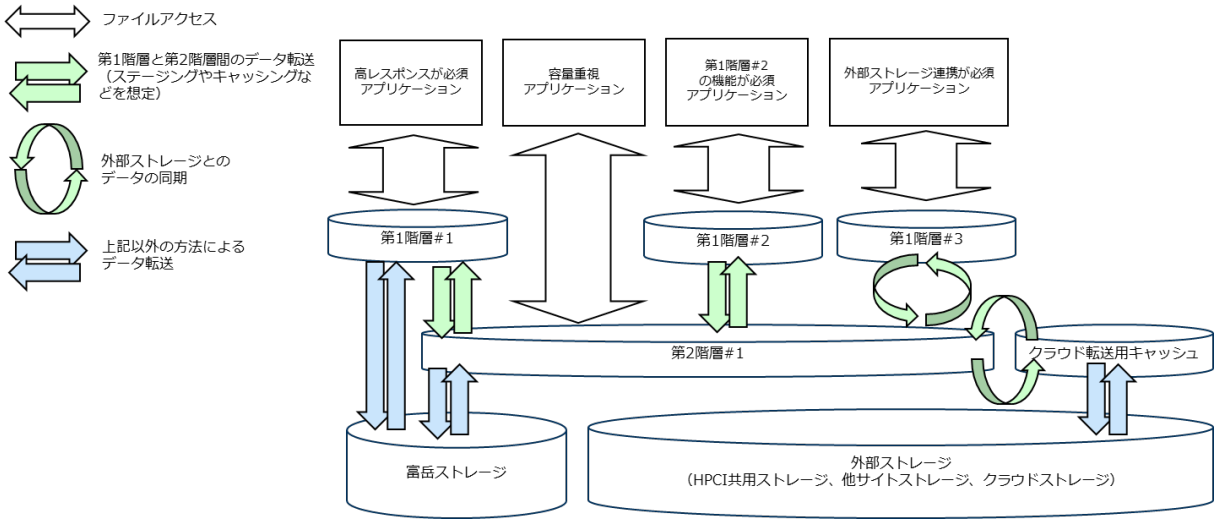


Figure 3-25 Overall Structure

3.4.2.2Usage Scenarios

The main use cases currently envisioned for "FugakuNEXT" users utilizing the hierarchical storage system are shown in from Figure3 -26 to3 -33 . Note that the colored areas in the figures indicate data transfer, and the meanings of the arrow types and colors are the same as in Figure3-25 at .

- Use Case 1
  - We consider cases involving the migration of applications from "Fugaku" and trial operations prior to the start of "FugakuNEXT" operations. For migration from "Fugaku," the purpose is to verify operation, and we assume computational execution on a single-node scale.Regarding trial operations prior to the start of "FugakuNEXT" operations, there was a history of requests for functional verification of critical applications and testbed environments even before the start of shared operations on "Fugaku," and it is anticipated that similar requests will arise prior to the start of "FugakuNEXT" operations.

Since trial operations take place before the start of shared operations, the intended users are primarily developers.

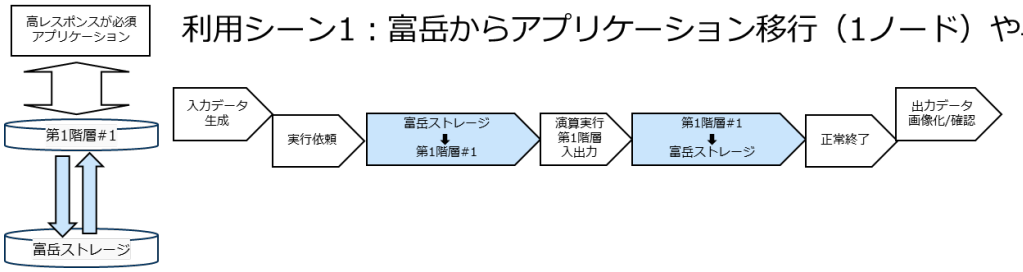


Figure 3-26 Use Case 1

- Use Case 2
  - ➢ We anticipate cases where the file system used for I/O in applications with high I/O volumes is fixed to the second tier only, as well as the migration of "Fugaku" applications that handle large volumes of files. This is used for workloads where the performance requirements of applications on the file system are low, and where using the first tier does not contribute to application performance.
    Since its use is also anticipated for workloads handling large numbers of files, it may contribute to increased load on the second tier.

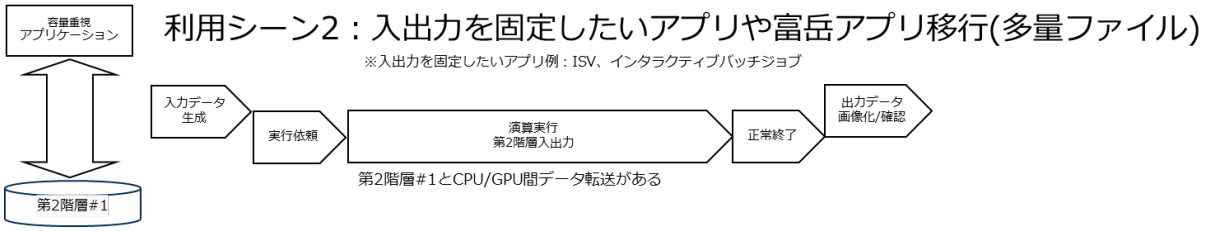


Figure 3-27 Use Case 2

- Use Case 3
  - ➢ This scenario assumes the use of unique features of different file systems within the first tier. For example, while the file system in Tier 1 #1 is typically used, a specific application uses the file system in Tier 1 #2. The file system is switched before the application is executed. Subsequently, the application performs its computations and transfers the output results to the second tier. Finally, the file system is restored to its original state.

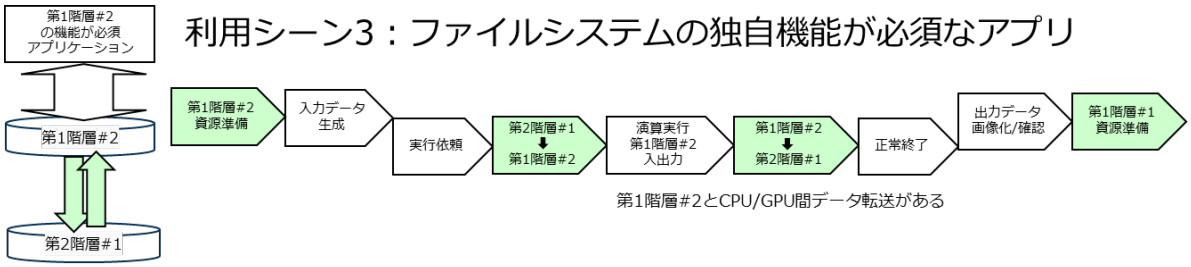


Figure 3-28 Use Case 3

- Use Case 4
  - Consider a scenario where the first tier is used to execute computations in applications that require high responsiveness. Data is transferred from the second tier to the first tier before and after computation execution via staging or caching. This approach is expected to be used in workloads where the application places high demands on the file system and where utilizing the first tier improves application performance.

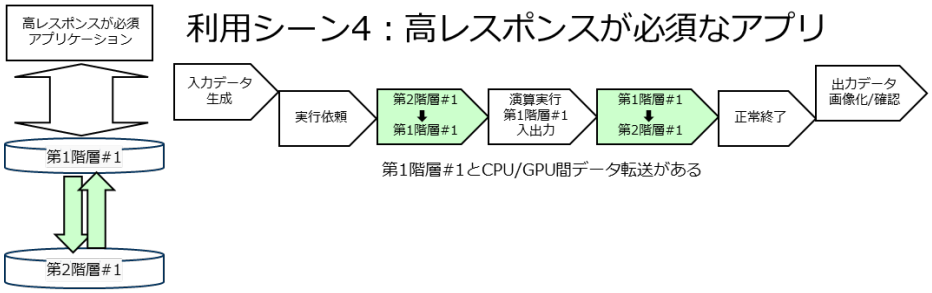


Figure 3-29 Use Case 4

- Use Case 5
  - This scenario assumes a workload where the application places high demands on the file system, involves a large volume of input data, and requires frequent and continuous output of calculation results, potentially leading to insufficient capacity in the first tier. In this case, it is necessary to periodically transfer the frequently and continuously output calculation results from the first tier to the second tier during the workload. It is assumed that input data is transferred from the second tier to the first tier only once, while subsequent output data is transferred from the first tier to the second tier multiple times.

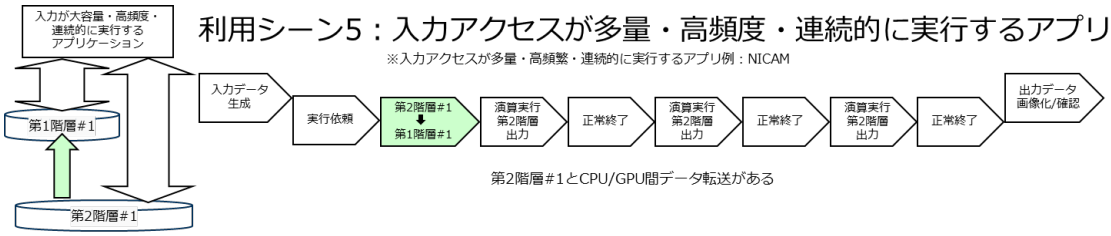


Figure 3-30 Use Case 5

- Use Case 6

This scenario assumes data transfer from "Fugaku" storage via the second tier during the migration of applications that utilize multiple nodes. It is intended for users of "Fugaku."

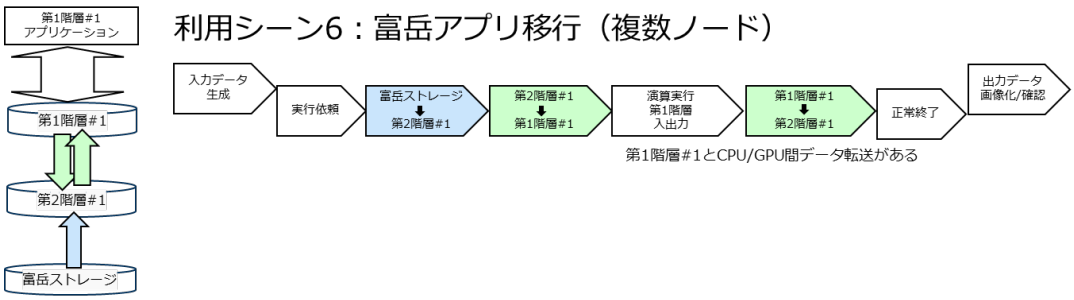


Figure 3-31 Use Case 6

- Use Case 7

This scenario assumes the use of applications that require integration with external storage. Input and output data are synchronized to the first and second tiers from external storage in parallel with the execution of computations. Output data is transferred to external storage.

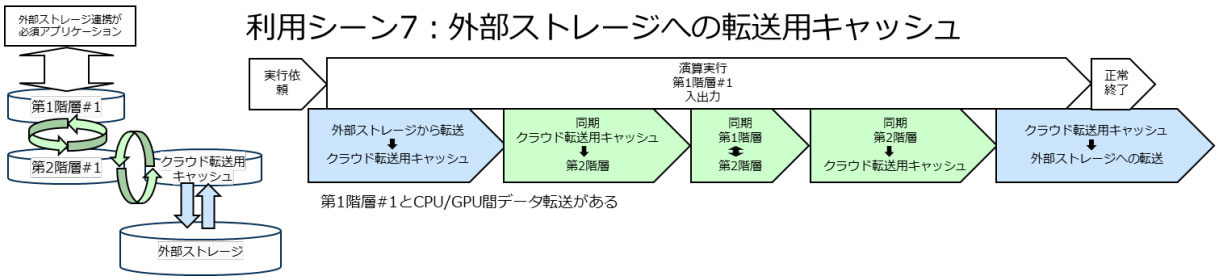


Figure 3-32 Use Case 7

- Use Case 8
  - This scenario outlines the lifecycle management of "FugakuNEXT" output data. After a workload is completed, the application's output data is transferred to external storage. The second tier is used for data archiving, and operational procedures and user guidance are required to ensure that external storage is utilized effectively to prevent storage capacity from becoming constrained.

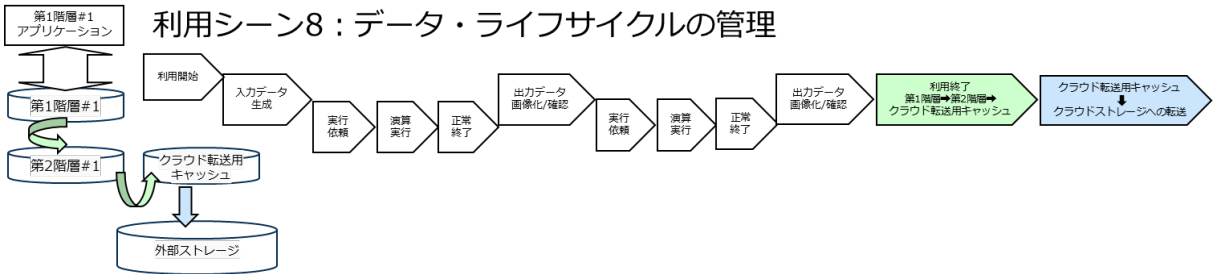


Figure 3-33 Use Case 8

3.4.2.3 Tier 1 Storage

This section describes the considerations for Tier 1 storage in the basic design. It incorporates the features required to realize the usage scenarios listed in the "3.4.2.2 " section, as well as file system challenges and issues raised during this review meeting.

3.4.2.3.1 System Configuration

This section describes the system configuration considerations for Tier 1 storage in the basic design.

3.4.2.3.1.1 Hardware Configuration Considerations

This section describes the hardware configuration considerations for Tier 1 storage.
The primary storage tier is based on an all-SSD configuration. The results of the evaluation of the devices to be used are described in the section at3.4.5.2.1 , and the results of the evaluation of the total storage capacity for the primary tier are described in the section at3.4.5.2.2 . In terms of hardware configuration, two types are under consideration: node-local storage and near-node-local storage.
Node-local storage utilizes SSDs installed on compute nodes. Since compute nodes are likely to be equipped with multiple SSDs, it is necessary to consider how to present these to users. The following are examples of potential use cases for node-local storage.

- Usage Scenario 1
  - Mount a separate file system (e.g., ext4, XFS) on each SSD installed on the compute node. While this approach allows for the utilization of each device's performance, it lacks redundancy and cannot handle device failures, resulting in poor availability. Usage Scenario 1 is shown in Figure 3-34.

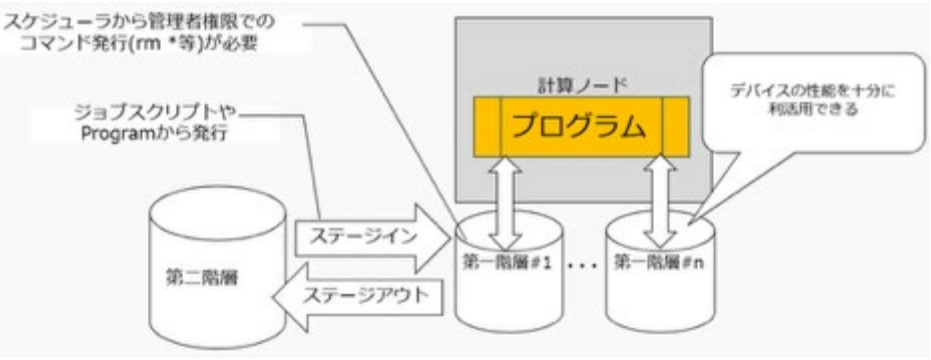


Figure 3-34 Node-Local Storage Usage Example 1

- Usage Scenario 2
  - The SSDs installed on the compute nodes are combined and utilized via software RAID for the file system. While this improves availability by allowing for device failures, performance considerations must account for the overhead associated with RAID. Usage Example 2 is shown in Figure 3-35 .

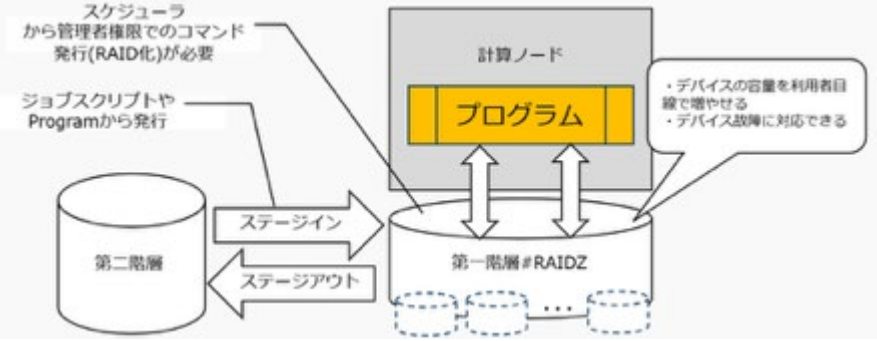


Figure 3-35 Node-Local Storage Usage Scenario 2

Near-node local storage is a method that utilizes SSDs installed on other nodes. The following are possible usage scenarios.

- Usage Scenario 1
  - The same SSD is connected via PCIe between two compute nodes and shared between them. While this improves availability because operations can continue on the paired node even if one node fails, it is expected to increase costs due to the complex hardware configuration and lead to more cumbersome operations. Furthermore, since application-level solutions such as Erasure Coding can also ensure availability, we consider the advantage of implementing this at the hardware level to be low.
- Usage Scenario 2
  - A dedicated storage node separate from the compute nodes is used, and each compute node accesses it via the network. While this improves operational stability by eliminating the impact of compute node failures, it is expected to increase costs due to the need for a dedicated node.
- Usage Scenario 3
  - The local storage of compute nodes used within a job is shared via the network. While this configuration is simple and offers cost advantages, the fact that storage is shared within a job means there are likely to be many considerations, such as how to control it from the scheduler.

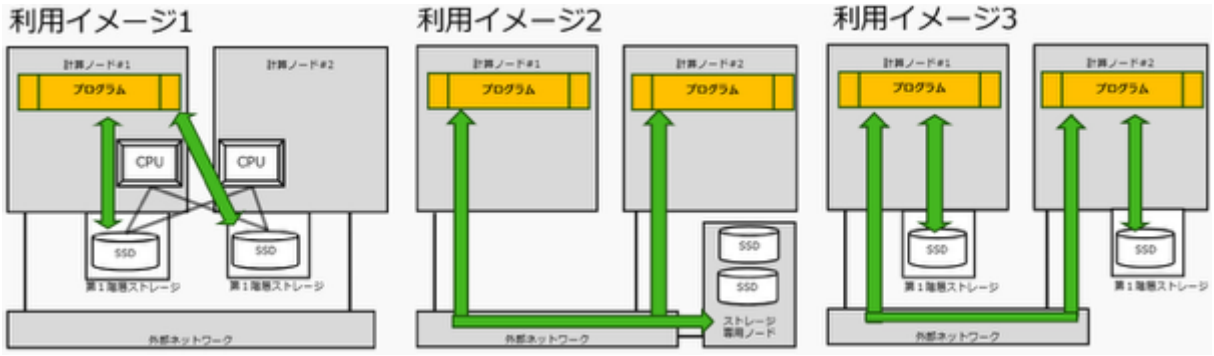


Figure 3-36: Near-Node Local Storage Usage Concept

In the basic design phase, we examined usage scenarios and configuration patterns for node local storage and near-node local storage. Each pattern has its own advantages and disadvantages in terms of performance, cost, and operability. In particular, near-node local storage is expected to involve increased configuration complexity and a greater number of considerations; therefore, at this stage, node local storage is considered the most viable option. During the detailed design phase, the hardware configuration will be determined through a more detailed comparative analysis based on information from each vendor.

For reference, details of the views on each pattern regarding near-node local storage are shown in Table 3-26 .

Table 3–26 : Detailed views on each usage scenario for near-node local storage

Category	Usage Scenario 1 Cross-link PCIe connection	Usage Scenario 2 Storage-dedicated node	Usage Scenario 3 Sharing within a Job
Configuration Features	・Since the Tier1 file system on the opposite node is accessed directly, a physical journal is required to ensure metadata protection and data integrity.	・Dedicated nodes allow for flexible selection of protection methods (replication, erasure coding, etc.).	・When sharing first-tier SSDs between nodes within a job, you need to decide whether to use simple remote block access or to apply erasure coding or similar techniques to enhance fault tolerance.
	・The physical journal area consumes a significant portion of SSD capacity.	・A scale-out configuration makes it easy to localize failures to individual nodes.	・A design that automatically swaps out to the second tier and performs recovery in the event of a compute node failure is also an option.
Prerequisites and Constraints	・Physical journals result in high SSD TBW consumption.	・Performance can be tuned based on the number of dedicated nodes and network bandwidth.	・Shared resources are job-based.
	・Due to direct PCIe connections, there are significant constraints when reconfiguring node layouts or replacing nodes.	・Bandwidth design must be tailored to job characteristics.	・Metadata management becomes complex when swap-out is used frequently.
Performance and Operational Concerns	Read performance drops to approximately half in practice.	・CPU load and latency increase, and performance fluctuations occur due to increased external network load.	・CPU load and latency increase, and performance fluctuates due to increased external network load.
	Suppresses performance fluctuations caused by		・Frequent swap-outs cause load peaks in the second tier.

	external network load, similar to local operations.		
Overall Assessment	・High consistency, but scalability and flexibility are significantly limited.	・Highly compatible with data center design and offers good operational stability.	・It offers the greatest flexibility, but performance and availability vary significantly depending on the chosen method.
	・Requires tolerance for SSD consumption and performance degradation.	・A strong option if dedicated hardware is available.	・Requires consideration of the most factors in actual operational design.

3.4.2.3.1.2 File system configuration consideration

This section describes the considerations for the file system configuration of Tier 1 storage.

- Basic Configuration
  - File System Type
    - In addition to compute node-local file systems (e.g., ext4, XFS, etc.) as the Tier 1 file system, a shared file system is also required to support SSF access from multiple compute nodes.
  - Supported Interconnect Types
    - The interconnect types between compute nodes are expected to be InfiniBand or Ultra Ethernet. For shared file systems, support for these interconnect technologies is required to enable inter-node communication.
  - Configuration Model
    - The file system is envisioned to consist of device nodes that store data and metadata, and services that manage them. For shared file systems, the presence or absence of management nodes in addition to compute nodes affects the configuration, so the components must be clearly defined.
- Virtualization Support
  - Virtualized environments, such as containers or virtual machines, are also considered as computing node environments. The file system must provide a method for accessing Tier 1 storage even within virtualized environments. Additionally, when configuring Tier 1 storage using a shared file system, integration with orchestration systems must be considered during selection.
- Data Placement/Metadata Management
  - In the case of a shared file system, it is anticipated that data and metadata will be distributed and stored across multiple nodes. In such cases, components that enable communication between compute nodes (e.g., Remote Procedure Call) are required as part of the file system architecture.
  - In "Fugaku," issues that limit I/O performance (STK-008/STK-013) exist because the shortage of SSDs in the first-tier storage and the memory capacity required for establishing connections were not taken into account. Therefore, it is necessary to clarify the requirements for compute nodes, such as the number of SSDs and memory capacity required.

3.4.2.3.2 Expected Functions

This section describes the functional requirements for the operation and management of Tier 1 storage.

3.4.2.3.2.1 Continuity, Stability, and Reliability

This section describes the functions expected to ensure continuous stability and reliability (data integrity) for the storage system.

- Redundancy and Fault Tolerance
  - To ensure stable and continuous operation, redundancy and fault tolerance functions at the file system level are required. The following five elements are considered for redundancy and fault tolerance.
- Protection Methods
  - Functions are required to prevent data loss even in the event of a disk failure. Examples include software RAID and Erasure Coding, which distributes data blocks and parity blocks across multiple disks. Additionally, factors such as the loss of data integrity in storage media or interface transmission paths, as well as end-to-end data integrity guarantees like T10 PI, should be included in the selection criteria.
- Metadata Redundancy
  - For shared file systems that require a metadata server, functionality is required to ensure operational continuity even in the event of a metadata server failure. Examples include an Active-Active HA (High Availability) configuration, in which two metadata servers are deployed and both are kept online at all times, or an Active-Standby HA configuration, in which one server is active and the other is on standby.
- Data Redundancy
  - For shared file systems, functionality is required to ensure operational continuity even if a storage server storing the data fails. One example is replicating the same data and storing it on multiple storage servers.
- Network Redundancy
  - For shared file systems, functionality is required to ensure continuous operation even if a network failure occurs. One approach is to use multiple network paths to enable simultaneous communication.
- Node Failover
  - For shared file systems in a near-node local configuration, a redundant configuration must automatically switch to a standby node in the event of a failure to ensure operational continuity. Additionally, the recovery time resulting from failure detection, node switching, and recovery processing must be short. Depending on the redundant configuration, an increase in load and a decline in performance may occur after a node switch.
- Data Reduction
  - "Fugaku" faces a challenge where storage capacity is insufficient, hindering the production of research results (STK-007). Therefore, features are needed to reduce the amount of data stored. Examples include data compression and deduplication.
- Load Prediction Function
  - To prevent node downtime and slow response times caused by high server load, it is necessary to monitor load conditions to detect and address issues proactively. One possible approach is to integrate with monitoring software such as Grafana or Prometheus.
- Countermeasures for Simultaneous Access by Multiple Processes
  - On "Fugaku," there is an issue where the response time for some processes is delayed when a large number of processes simultaneously perform file open/close operations (STK-003). Therefore, it is necessary to keep latency low even when there are metadata access requests from a large number of processes.
- Quota
  - To prevent specific users from monopolizing storage capacity, a quota function is

required. For example, it is conceivable to allow quotas to be set at the user, group, or project level.

- QoS
  - A feature is required to prevent specific users from monopolizing storage I/O requests within a compute node (node QoS). Additionally, a mechanism is required to prevent specific jobs from monopolizing I/O when multiple jobs are running (server QoS).

3.4.2.3.2.2 Security and data protection

This section describes the functions expected of a storage system to ensure security and protect data.

- Access Control/Authentication
  - Functions for controlling access to files and directories by other users or jobs are required. Support for POSIX ACLs, for example, is one possible approach.
  - In the case of shared file systems, functions are required to ensure that only trusted clients can access the server in order to prevent unauthorized access. Support for Kerberos authentication is one example.
- Encryption
  - Encryption is required to prevent eavesdropping and tampering with data in transit and at rest. Examples include encryption of communication channels using TLS and data encryption using AES.
- Key Management
  - Integration with a key management system is required to securely manage cryptographic keys used for data encryption. Examples of key management systems include support for Vault.

3.4.2.3.2.3 Future Potential and Scalability

This section outlines factors to consider regarding future potential and scalability when selecting a storage system with a view toward 2030.

- Applicability of New Technologies
  - Consideration of the application of new technologies, such as ZNS (Zoned Namespace) and SCM (Storage Class Memory), is required.
- Disk Technology
  - Consideration of NAND flash memory types (e.g., TLC, QLC, PLC) and form factors is required.
- Features Offering Technical Advantages
  - The presence or absence of features that constitute the file system's unique strengths is a key consideration.

3.4.2.3.2.4 Constraints and Operational Risks

This section lists items that could pose functional constraints or risks when operating the storage system.

- Constraints and Precautions
  - These include operational limitations such as the maximum number of files.
- Error Detection and Reconstruction
  - These include the availability and speed of reconstruction and automatic repair functions in the event of an error.
- Requirements and Constraints for Other Architectures
  - These include constraints on supported operating systems and the necessity of external

devices.

3.4.2.3.3 Examination of System Performance Requirements

This section describes the considerations regarding the required performance of Tier 1 storage. The performance of Tier 1 storage depends on the performance and number of devices (SSDs) installed in the compute nodes, as well as the performance of the file system. During the basic design phase, we conducted vendor interviews and gathered information regarding the following items necessary to meet the required performance.In the detailed design phase, we will organize the requirements for the storage system and, in collaboration with storage vendors, develop a design to ensure that a storage system meeting these requirements is implemented. Additionally, as necessary, we will perform performance estimates based on information provided by each vendor and organize the data to enable the validation of the proposed storage systems' feasibility and comparative analysis.

- Scalability
  - "Fugaku" faces the following challenges regarding metadata scalability. "FugakuNEXT" must also ensure metadata performance scalability even if the number of accessing compute nodes or the number of files handled increases.
    - Issues where performance degradation and increased frequency of I/O errors occur as the number of nodes accessing the same directory or the number of files increases (STK-002)
    - The issue where the load on the metadata server increases during the creation of large numbers of files, preventing scaling to the full system scale (STK-004)
    - The issue of jobs failing or being re-executed due to I/O errors caused by load concentration on the metadata server (STK-005)
    - With a phased deployment of compute nodes in mind, a method for evaluating target performance according to each phase must be considered (refer to Section3.4.4.1 regarding performance measurement policies).
- Performance Characteristics
  - As a selection criterion, it is necessary to confirm the availability of performance records, such as IO500.
- File System Performance and Device Performance
  - The system performance requirements and estimated performance with the assumed devices are described in Section3.4.5.2.2 . A file system that meets the required performance must be selected, even when considering the overhead associated with using the file system.
- Device-dependent performance
  - It is necessary to select devices that meet the required performance specifications, such as NVMe SSDs and SCM (Storage Class Memory).
- Parallel I/O
  - In the case of shared file systems, support for MPI-IO is required to handle parallel jobs from multiple compute nodes. Additionally, since the use of HDF5/NetCDF as parallel libraries is anticipated, support for these is also required.
  - On "Fugaku," support for MPI-IO asynchronous I/O has been identified as an issue (STK-006). Regarding MPI-IO asynchronous I/O, a detailed design study will be conducted, including an assessment of whether file system support is necessary.
- Consistency Relaxation
  - In addition to strict POSIX-compliant consistency, modes with relaxed consistency—used when prioritizing performance—are required.
- Representative Performance Metrics and Observations
  - Performance measurements must include metadata performance (IOPS) and sequential access performance (bandwidth). The discussion of measurement methods

is described in Chapter3.4.4 .

- SSF Performance Strength
  - In "Fugaku," performance degradation due to lock acquisition overhead during SSF access in highly parallel jobs has been identified as an issue (STK-001). Therefore, features are required to ensure SSF access performance even under high parallelism (e.g., I/O with relaxed consistency).
- GPUDirect Storage Support
  - To accelerate I/O from GPUs, the file system must support GPUDirect Storage. Details regarding the use of GPU-based Tier 1 storage will be examined during the detailed design phase and beyond.
- The Effectiveness of the Non-Shared Architecture (DASE)
  - We are closely monitoring the "DASE" (Distributed Anything Storage Engine) architecture—which separates compute and storage—adopted by the VAST Data Platform as a key technology for large-scale AI environments.
  - We will focus on its ability to physically eliminate metadata contention while scaling performance and capacity independently, and verify the feasibility of exabyte-scale expansion within a single namespace.
- Distributed Memory Integration and Ultra-High-Speed Data Feeding Technology
  - Through detailed design, we will also verify methods such as those provided by WEKA-IO's "Augmented Memory Grid" technology, which integrates distributed memory across thousands of nodes to ensure TB/s-class ultra-high-speed data supply capabilities (data feeding capabilities).
- Ensuring Low-Latency Hot Data Paths
  - The microsecond-level response performance seen in systems such as Scality (RING XP) is considered extremely useful as a hot data path for AI inference workloads, and we will scrutinize configuration proposals for this in the detailed design phase

3.4.2.3.4 Selection of File System Candidates

In the basic design, we extracted the elements required for the first-tier file system from items3.4.2.3.1 through3.4.2.3.3 . In the detailed design, we will select a file system that meets these elements based on information from each vendor. If it is difficult to satisfy all elements, we will examine the feasibility of implementation through functional development for each file system or by combining multiple file systems during the detailed design phase.

Furthermore, since "Fugaku" lacks a sustainable software development framework and faces challenges regarding continuous post-deployment modifications and improvements (STK-011, STK-012), these issues must be included in the selection criteria. Examples include the level of community activity for OSS and the vendor's maintenance/support framework for ISVs.

3.4.2.4 Tier 2 Storage

This section describes the considerations for Tier 2 storage in the basic design. It incorporates the functions required to realize the use cases listed in the3.4.2.2 section, issues related to the file system (FS), and issues raised during this review meeting.

3.4.2.4.1 System Configuration

This section describes the system configuration considerations for Tier 2 storage in the basic design.

3.4.2.4.1.1 Hardware Configuration Considerations

The second-tier storage is envisioned as a storage system shared across all compute nodes.

The storage system consists of external storage devices and storage servers that manage them. By deploying multiple storage servers, we will achieve the required second-tier storage capacity, bandwidth, and IOPS. The exact number of storage servers required will be determined during the detailed design phase.

Regarding the usage of Tier 2 storage, instead of using it as a single volume across the entire system, a configuration where it is virtually partitioned is also being considered. In that case, it is necessary to evaluate whether there are any performance issues and whether fault tolerance can be ensured. Specific implementation methods will be examined in the detailed design.

The results of the evaluation of devices to be used for Tier 2 storage are described in the section3.4.5.2.1 . Additionally, the results of the evaluation of the total capacity, bandwidth, and IOPS for Tier 2 storage are described in the section3.4.5.2.2 . The results of the evaluation regarding phased implementation are described in the section3.4.5.2.4 .

Based on the evaluation of devices for use in Tier 2 storage, it is anticipated that SSD write performance will be lower than read performance. Therefore, given the trade-off between capacity and performance, it may be difficult to meet the performance requirements for write performance. Since it will be necessary to reassess the performance requirements based on read performance, this will be addressed during the detailed design phase and beyond.

3.4.2.4.1.2 Filesystem configuration Consideration

This section describes the considerations regarding the file system configuration for Tier 2 storage.

- Virtualization Support
  - Virtualized environments, such as containers and virtual machines, are also anticipated as computing node environments. The file system is required to provide a method for accessing the second-tier storage even within virtualized environments.
- Data Placement/Metadata Management
  - Regarding data placement, a parallel configuration across multiple storage servers using striping is a possible approach. For metadata management, options include managing metadata on a single metadata server or distributing management across multiple nodes. Since these approaches affect the overall configuration, it is necessary to clearly define the chosen method.
  - In "Fugaku," the memory capacity required for establishing connections was not taken into account, resulting in an issue that limits I/O performance (STK-008). Therefore, it is necessary to clearly define the hardware requirements for metadata servers and storage servers, including memory capacity.

The following items are common to the first-level descriptions in Section3.4.2.3.1.2 .

- Basic Configuration
  - Supported Interconnect Types
  - Configuration Models

3.4.2.4.2 Phased Deployment

When introducing the Tier-2 storage system, 'phased deployment' refers to the process of introducing the storage system incrementally, for example by rolling out storage server enclosures in two or more stages, rather than installing the entire system all at once. Additionally, the term 'rolling update' refers to the gradual replacement or upgrade of part of the storage system.

After extensive consideration, it was concluded that there are several challenges to be addressed. Therefore, the specific solutions and deployment methodology will be determined during the detailed design phase.

Considerations;

- Phased Deployment
  - Budget

    If there is a discrepancy between the budget and the cost required to procure a storage system that meets the specifications, the performance and storage capacity may be limited to what is feasible within the available budget, potentially falling short of the specified requirements.

  - Implementation Timing/Installation Location/Relocation

    If the "FugakuNEXT" storage system is to be deployed in advance—for example, to facilitate data migration from the "Fugaku" storage system—it will be necessary to determine the deployment schedule, the building in which it will be installed, and, in some cases, whether relocation to a new building is required.

    Furthermore, depending on how long the "Fugaku" storage system remains operational, a single-phase deployment may be considered; however, since overhauls and other maintenance are required, it is currently undetermined how long it will remain operational.

  - File System Partitioning

    In the case of a phased deployment, since the entire storage system will not be installed all at once, it is necessary to consider how to partition the file system from the cabinets being introduced in stages so that users can access it. At this time, there are four proposals.

    It should be noted that, depending on the file system, it may not be possible to integrate the additional enclosure into the existing file system, so this must be taken into account.

Proposal to create a file system for each enclosure to be deployed

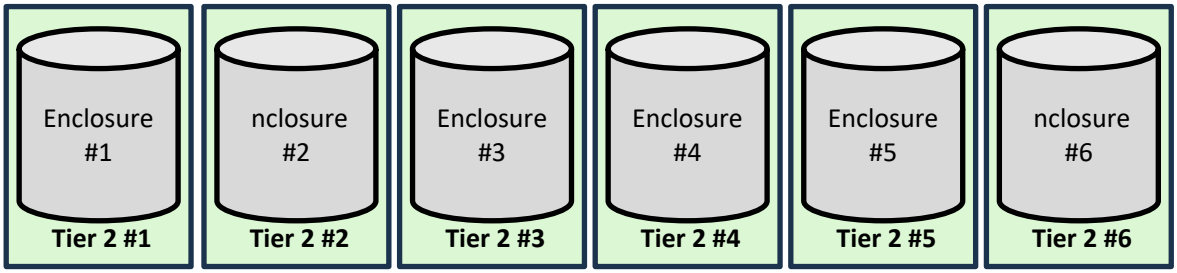


Figure 3-37 Proposal 1: Separation by Chassis

As demonstrated by the partitioning used in "Fugaku," the benefit of partitioning is that even if one file system stops, operations can continue because the other file systems remain operational.

Proposal to create file systems while adding enclosure to the system

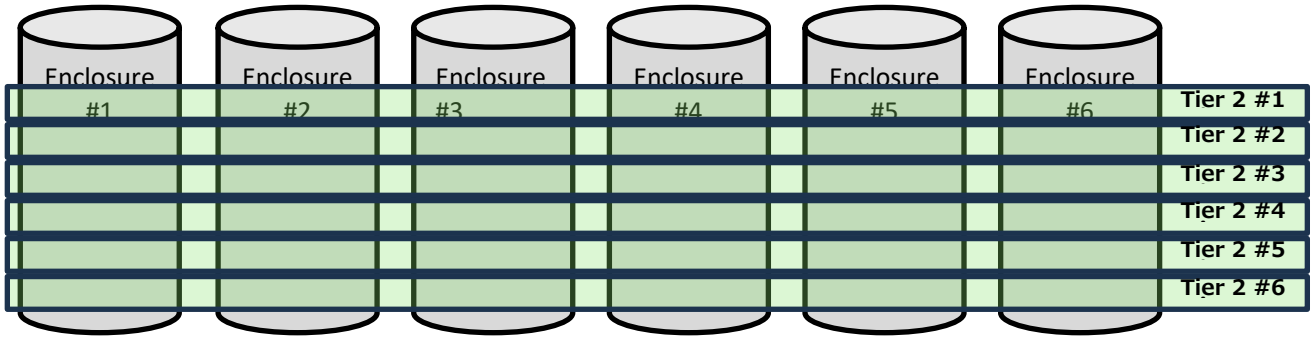


Figure 33-8 Proposal 2: Split Across Enclosures

A proposal to partition the file system by combining Proposal 1 and Proposal 2

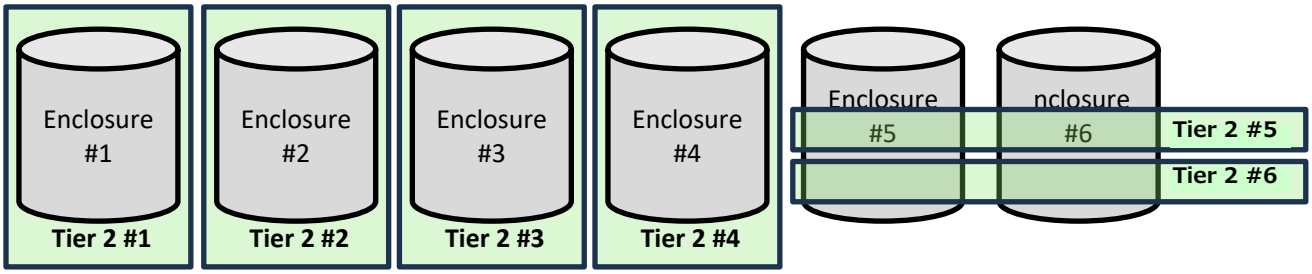


Figure 3-39 Proposal 3: Hybrid

A proposal consisting of one large-scale tier for user use, multiple small-scale tiers for testing, and one tier spanning all cabinets for full-scale testing

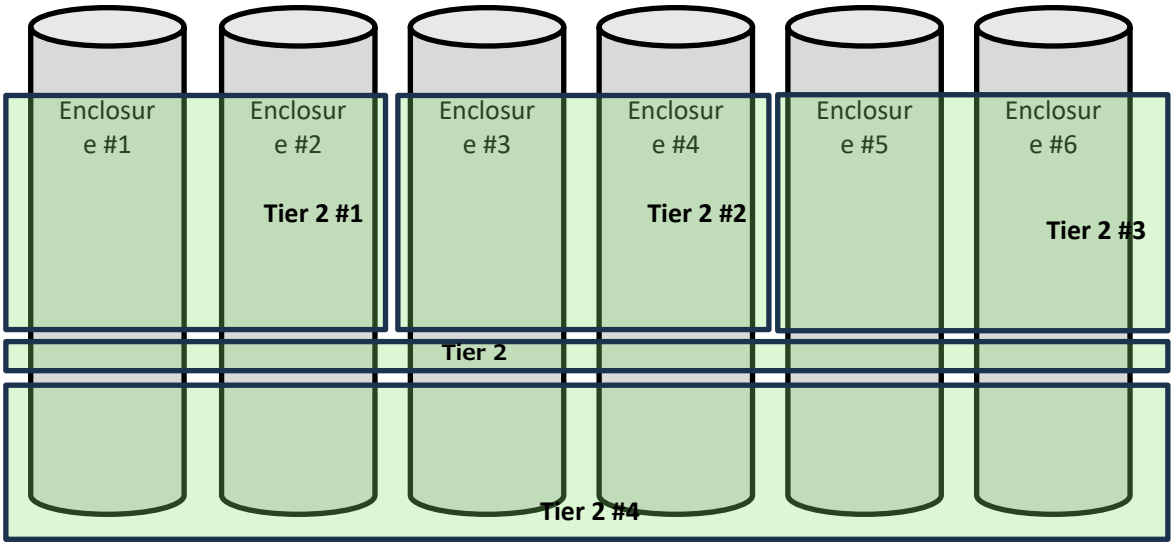


Figure 3-40 Proposal 4

- Rolling Type
  - Overall Performance
    Because updates are performed in stages, there may be periods when the entire storage system is not yet complete, making it impossible to measure overall performance. Similarly, overall performance may not be measurable if delays occur in the rolling

update schedule or if operational downtime constraints arise.
Additionally, if storage servers and other components do not have the same configuration, overall performance may be limited by the slowest controller or other component.
Note that with single-phase deployment, the entire storage system consists of the same generation and configuration, so overall performance can be measured all at once; however, the benefits of improved component performance and lower component unit prices cannot be realized for 5 to 6 years.

- System launch point
  It is necessary to consider at what stage the storage system is considered ready for operation (for example, the stage at which overall performance can be measured).

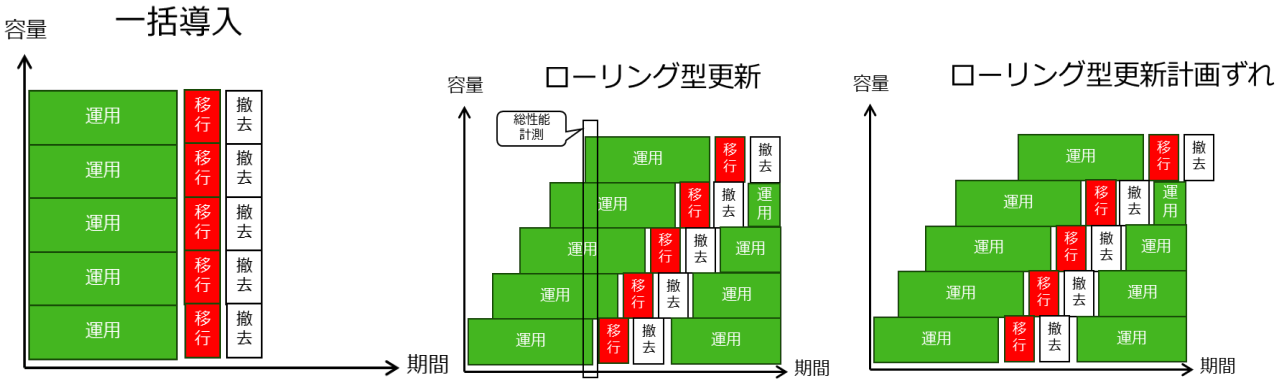


Figure 3-41 Rolling Deployment

3.4.2.4.3 Expected Functions

This section describes the expected functional requirements for the operation and management of Tier 2 storage.

3.4.2.4.3.1 Continuity, Stability, and Reliability

This section describes the functions expected to ensure continuous stability and reliability (data integrity) as a storage system.

- Rebalancing Function
  - To prevent data from becoming concentrated on specific storage servers, a function is required to distribute data evenly across storage servers.
- Snapshots
  - A function is required to take snapshots for recovery in the event of a file system failure.

The following items are common to the first-level descriptions in the "3.4.2.3.2.1 " section.

- Redundancy and Fault Tolerance
- Data Reduction
- Load Prediction Function
- Measures for simultaneous access by multiple processes
- Quota
- QoS

3.4.2.4.3.2 Security and Data Protection

This section describes the functions expected of a storage system to ensure security and

protect data. Each item corresponds to the first-level descriptions in Section3.4.2.3.2.2 .

- Access Control/Authentication
- Encryption
- Key Management

3.4.2.4.3.3 Future Potential and Scalability

This section describes items to be considered from the perspective of future potential and scalability in order to select a storage system with a view toward 2030.

- Configuration Flexibility
  - Flexibility is required to propose multiple configuration patterns based on specific requirements.

The following items are common to the first-level descriptions in the "3.4.2.3.2.3 " section.

- Applicability of New Technologies
- Features with Technological Advantages

3.4.2.4.3.4 Constraints and Operational Risks

This section describes items that may constitute functional constraints or risks in the operation of the storage system. Each item is common to the first-level description in Section3.4.2.3.2.4 .

- Constraints and Precautions
- Error Detection and Reconstruction
- Requirements and Constraints for Other Architectures

3.4.2.4.4 Review of System Performance Requirements

This section describes the considerations regarding the required performance of Tier 2 storage. Since the required performance of Tier 2 storage depends on the scale of the compute nodes, the storage configuration must be designed to meet these performance requirements. During the basic design phase, vendor consultations were conducted and information was gathered regarding the following items necessary to meet the required performance.In the detailed design phase, we will organize the requirements for the storage system and, in collaboration with storage vendors, develop a design to ensure that a storage system meeting these requirements is implemented. Additionally, as necessary, we will perform performance estimates based on information provided by each vendor and organize the data to enable the validation of the proposed storage systems' feasibility and comparative analysis.

- Scalability
  - In addition to the content described in Section 3.4.2.3.3 regarding the first tier, the second tier requires storage server scalability because the number of storage servers also affects performance. It is considered necessary to evaluate the balance between the number of servers required to achieve the target performance and the associated costs and capacity.
  - With a view toward the phased deployment of storage servers (see Section3.4.2.4.2), it is necessary to consider a method that allows target performance to be evaluated at each stage (for performance measurement policies, see Section3.4.4.1).

The following items are consistent with the content of Section 3.4.2.3.3, which describes the first tier.

- Performance Characteristics
- File System Performance and Device Performance

- Device-dependent performance
- Parallel I/O
- Consistency Relaxation
- Representative Performance Metrics and Observations
- The Strength of SSF Performance
- GPUDirect Storage Support

3.4.2.4.5 Selection of File System Candidates

For the second tier, a storage system—including both the file system and hardware—must be selected. In the basic design phase, we extracted the elements required for the second-tier storage system from the sections ranging from3.4.2.4.1 to3.4.2.4.4 .In the detailed design phase, file systems and hardware that satisfy these elements will be selected based on information from each vendor. If it is difficult to satisfy all elements, the detailed design phase will involve examining the feasibility of implementation through functional development of each file system or by combining multiple storage systems. Additionally, object storage will be included as a candidate for the second tier.

Furthermore, since “Fugaku” lacks a sustainable software development framework and faces challenges regarding continuous post-deployment modifications and improvements (STK-011, STK-012), these issues must be incorporated into the selection criteria. Examples include the level of community activity for OSS and the vendor’s maintenance/support framework for ISVs.

3.4.3 Support for Storage System Utilization

This chapter describes technologies that support the utilization of storage systems, including data transfer between the first and second tiers, integration with external storage, and tools for operational support and performance analysis of storage systems.

3.4.3.1 Data Transfer Between Tier 1 and Tier 2

This section describes the results of investigations and considerations regarding data transfer between the first and second layers at the basic design stage. Since this technology depends on the job scheduler and file system, specific considerations will be conducted during the detailed design phase or later, once the job scheduler and file system to be used by "FugakuNEXT" have been determined.

3.4.3.1.1 Transfer Technologies and Methods

Regarding data transfer technologies, we primarily investigated and evaluated file staging, and also examined transparent caching.File staging is a technology that limits an application's file access to the high-speed first tier and aims to reduce access time by performing "stage-in"—transferring input files used by the application from the second tier to the first tier before the application starts—and "stage-out"—transferring output files from the first tier to the second tier after the application ends.Transparent caching is a technology that achieves high-speed file access by utilizing the first tier as a cache for the second tier. Since the system handles data synchronization between the first and second tiers, applications can use the cache without being aware of its existence.

There are four types of data transfer methods. The specifics of each are described in the section titled "3.4.3.1.3 ."

- Transfer via the Job Scheduler Function
- Transfer via the file system function
- Transfer via general-purpose commands
- Transfer via other means

3.4.3.1.2 Transfer Patterns

The data transfer patterns anticipated for "FugakuNEXT" are shown in , Figure 3-42 . As stage-in destinations, we assume the node-local storage allocated to each compute node and the shared file system storage that can be shared among compute nodes. Additionally, the 1:N and SSF (Single Shared File) ⇔ FPP (File Per Process) notations in the figure indicate changes in the number and format of input files resulting from data transfer between layers.Taking (1) as an example, a single file on the second tier is staged into the node-local storage of N compute nodes (1:N transfer), and the file format changes from SSF (when accessing files on the second tier) to FPP (when accessing files on the first tier).

Note that among these patterns, (4), (6), and (8)—which involve splitting or merging files—are considered patterns that should be implemented on the application side and are therefore excluded; the focus is on (1), (2), (3), (5), and (7).

In use cases combining AI and simulation, since different sets of applications are executed in each, (2) may occur. The examination of patterns from an AI perspective, including the use of GPUDirect, will be conducted during detailed design.(3) refers to a pattern where input/output is performed on different files for each process (rank), with checkpoint/restart being a prime example. Checkpoint/restart is a function that creates checkpoints (records the application's execution state) upon instruction from the application and resumes execution from the checkpoint in the event of a failure; since this requires discussion in conjunction with the application, it will be examined during the detailed design phase.

O n "Fugaku," the llio_transfer command is used for staging files shared among processes.In

NICAM, one of the atmospheric models, input files are separated by process, and there is a requirement to stage files to a Storage I/O Node (SIO) near the node where the process is located. However, there is an issue (STK-009) where the specifications of the llio_transfer command cannot meet this requirement. Figure 3-42 shows that (3) corresponds to this issue.
Additionally, the `llio_transfer` command has the following issues, which will be addressed in the detailed design.

- There have been instances where applications failed to run because the llio_transfer command was not specified in the job script, and informing users about how to use the llio_transfer command has become a burden.
- Some Python files are 0 bytes in size, but because the llio_transfer command is designed not to transfer 0-byte files, the Python program cannot be executed.

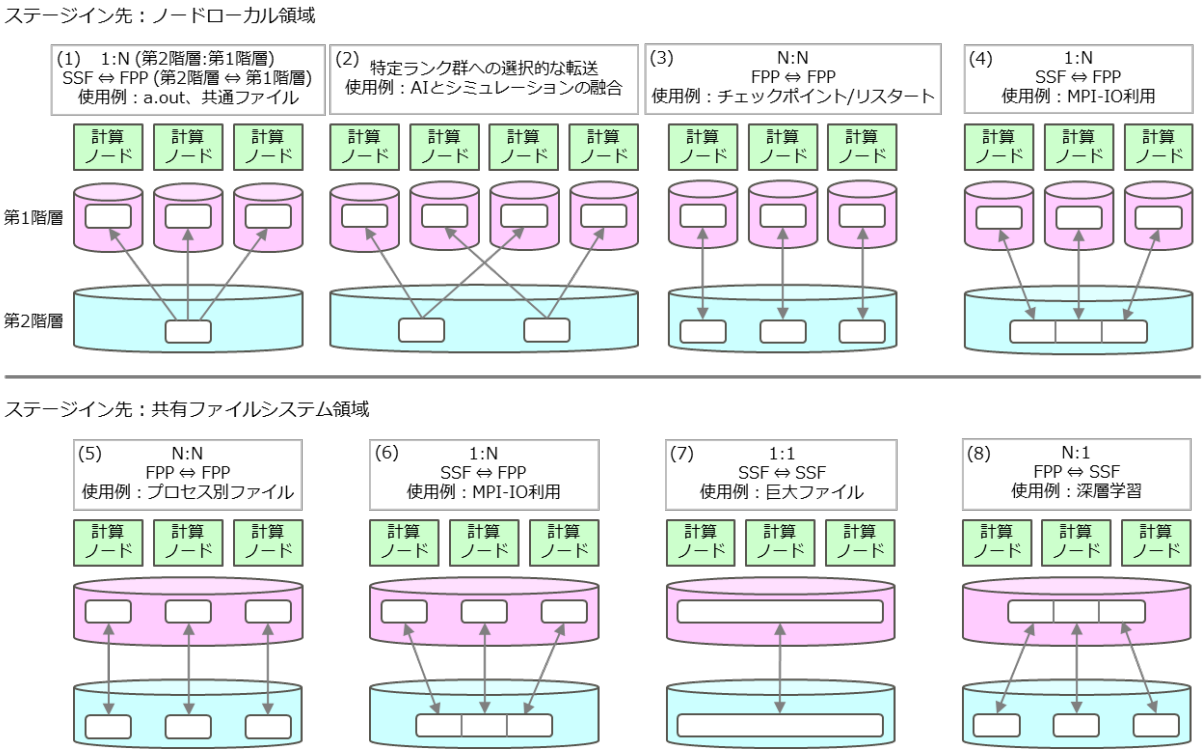


Figure 3-42 Transfer Patterns

3.4.3.1.3 Software Survey

We investigated the software available at the basic design stage for the data transfer methods described in the section. Note that specific considerations regarding support for each data transfer pattern described in the3.4.3.1.2 section will be conducted during the detailed design phase or later, after the job scheduler and file system to be used by "FugakuNEXT" have been determined.

1. Transfer via the Job Scheduler feature
   (1) PBS Professional
   - Overview
     The job scheduler provides a file staging feature.
   - Specification
     Specify using the -W option of the qsub command (job submission command) or the #PBS directive in the job script.
     • Stage In
       qsub -W stagein=<destination path>@<source host>:<source path>

or
#PBS -W stagein=<destination path>@<source host>:<source path>

• Stageout
qsub -W stageout=<source path>@<destination host>:<destination path>
or
#PBS -W stageout=<source path>@<destination host>:<destination path>

- Notes

Since staging operations are limited to the job script execution node, a different mechanism is required to support patterns where the node-local storage of multiple compute nodes serves as the destination or source, such as those described in (1) to (3) in the transfer patterns section of Figure 3-42 .

(2) Slurm Workload Manager

● Burst Buffer Lua Plugin

- Overview

The job scheduler provides a file staging function that utilizes the Burst Buffer Lua plugin.

- Specification Method

Describe the staging process in the Lua script called by the Lua plugin. A template for the Lua script is provided in the Slurm source code, which includes the required parameters, the timing of each function call, the return values of each function, and examples of how to implement each function.

- Notes

Since the staging process can be customized via the Lua script, it is expected to support a variety of transfer patterns.

● sbcast command

- Overview

Transfers files to each compute node assigned to the job.

- Specification

Write this in the job script and execute it.

Example: sbcast /global/a.out /local/a.out

Stages the file /global/a.out (located in the second directory level) into the /local directory of each compute node.

- Notes

As described in the "Specification Method," the sbcast command can be used for staging in, but since it cannot be used in the reverse direction—such as collecting output files from each compute node to the second level—a different mechanism is required for staging out. Additionally, since this is a command for broadcasting, it is not suitable for staging using transfer patterns other than (1) in Figure 3-42 .

2. Transfer via File System Features

(1) Lustre PCC (Persistent Client Cache)

- Overview

Uses the Lustre client's local storage (SSD) as a transparent cache for the file system.

- Specification

Execute the following command on the Lustre client.

• lfs pcc attach command

Associates the specified file with local storage. This causes the file to be cached on local storage, making it readable and writable.

• lfs pcc detach command

Detaches the specified file from local storage. The cache is flushed to the Lustre

server as needed.

- Note

  Although strictly speaking this differs from file staging, similar to file staging, it can reduce access times by limiting the application's file access to the fast first tier. It is also possible to automatically cache files on local storage based on specific rules.

3. Transfer via General-Purpose Commands

(1) cp

- Overview

  A general-purpose file copy command.

- Usage

  Write it in a job script and execute it.

- Notes

  Since file transfers between different nodes are not possible, to support patterns where the node-local areas of multiple compute nodes serve as the destination or source—such as those shown in (1) to (3) in the figure at Figure 3-42 —you must execute this command in combination with parallel execution commands such as pdsh or mpiexec.

(2) scp/rsync

- Overview

  A general-purpose file transfer command.

- Usage

  Write in a job script and execute.

- Notes

  Since files can be transferred from the job script execution node to other compute nodes, the system supports transfer patterns where the node-local storage of multiple compute nodes serves as the destination or source, as shown in (1) to (3) in the transferdiagram at Figure 3-42 . However, since parallel transfers to or from multiple compute nodes are not supported, the process must be repeated.

(3) pdcp/rpdcp

- Overview

  A general-purpose command that allows files to be transferred in parallel to multiple nodes.

  • pdcp command: Transfers files in parallel to multiple nodes.
  • rpdcp command: Transfers files in parallel from multiple nodes.

- Usage

  Write the command in a job script and execute it.

- Notes

  Since files can be transferred from the job script execution node to other compute nodes, it is possible to support patterns where the node-local areas of multiple compute nodes serve as transfer destinations or sources, such as (1) to (3) in the transfer patterns shown in , Figure 3-42 , .

4. Transfer via Other Methods

(1) mpiFileUtils

- Overview

  An MPI-based file manipulation tool. It runs as an MPI application.

  Tool Examples)

  • dbcast: Broadcasts a single file to each compute node.

- dcp: Recursively copies files or directories.
- dsync: Synchronizes two files or directories.

- Usage
  Execute by specifying the mpiexec or mpirun command in the job script.
- Stage-in (Example)
  mpiexec -n 128 dbcast /global/a.out /local/a.out
  or
  mpiexec -n 128 dcp /global/a.out /local
  Transfers /global/a.out from the second level to /local on each compute node.
- Stage out (example)
  mpiexec -n 128 dcp /local/* /global
  Transfers output files from /local on each compute node to /global in the second-level directory.

- Notes
  This tool is designed for use in large-scale systems and enables efficient transfer based on the number of nodes and processes. However, since this tool executes the same process on each compute node, special consideration is required if you wish to transfer different files for each compute node or specific rank groups, as shown in (2) or (3) of the transfer patterns at Figure 3-42 .

3.4.3.2 External Storage Integration

External storage is intended for use in data archiving and data sharing between different supercomputer systems. This section presents the results of our investigation into the following aspects of integration, such as data transfer between secondary-tier storage and external storage.

1. Methods of Integration with Tier-2 Storage
2. Data Transfer Methods with Tier-2 Storage
3.

Detailed integration and transfer methods with external storage will be examined during the detailed design phase and beyond, as they depend on the Tier 2 storage and external storage supported by "FugakuNEXT."

Furthermore, in Fugaku, there are challenges regarding transfer load caused by data movement frequency and insufficient speed when interfacing with external storage such as HPCI shared storage (STK-010); specific countermeasures for this issue will also be examined during the detailed design phase and beyond.

3.4.3.2.1 Integration Methods with Tier-2 Storage

Possible methods for integration between Tier 2 storage and external storage include providing a server accessible to both sides and performing data transfer using software or services for data transfer on that server, or introducing hierarchical storage management software to manage the system.

Since it is common practice to separate the data transfer server from the login nodes and compute nodes, we believe this approach is preferable for “FugakuNEXT” as well.

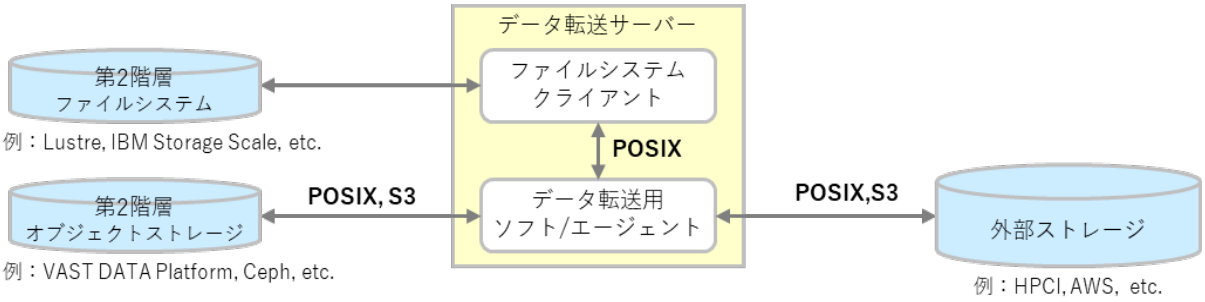


Figure 3-43 Data Transfer with External Storage

The following lists candidate external storage services for integration. Which external storage services to support will be determined during the detailed design phase or later. When making this determination, authentication methods for accessing each storage system will also be considered.

- HPCI Shared Storage
  Supported API: POSIX
- Amazon S3
  Supported API: S3
- Google Cloud Storage
  Supported APIs: S3 (compatible API), dedicated REST API
- OCI Object Storage
  Supported APIs: S3 (compatible API), dedicated REST API
- Azure Blob Storage
  Supported APIs: S3 (conversion tool required), Dedicated REST API

The following lists software and services that can be used to transfer data to and from external storage. All of these software and services support POSIX and S3 APIs for accessing the second-tier storage, and the S3 API—which has become the de facto standard—can be used for accessing the external storage.

- Rclone
  An open-source command-line program.
  It provides cloud-based equivalents of commands such as rsync, cp, mv, mount, ls, rm, and cat.
- Globus
  A data transfer service developed and operated by the University of Chicago.
  It features both a web-based GUI and a command-line interface.
- AWS DataSync
  A data transfer service provided by AWS.
  It features a GUI via the AWS DataSync console and a command-line interface.
  Exclusively for data transfer with AWS storage services.
- Google Storage Transfer Service
  A data transfer service provided by Google.
  It features a GUI via the Google Cloud Console and a command-line interface.
  Exclusively for data transfer with Google Cloud Storage.

The following tiered storage management software is considered suitable for use when managing both internal "FugakuNEXT" storage and external storage as a tiered storage system.

- ScoutAM
- HPSS

3.4.3.2.2 Transfer Methods to Tier 2 Storage

There are two methods for transferring data to external storage: automatic and manual.

- Automatic Transfer
  For example, by using Lustre's HSM functionality, data can be automatically transferred to external storage based on configured conditions.
  There are also other data transfer services that allow you to set policies and transfer data when specific conditions are met.
- Manual Transfer
  This method allows users to transfer file data by executing a transfer command at any time.

The following illustrates data transfer to external storage when using Lustre as the second-tier file system.

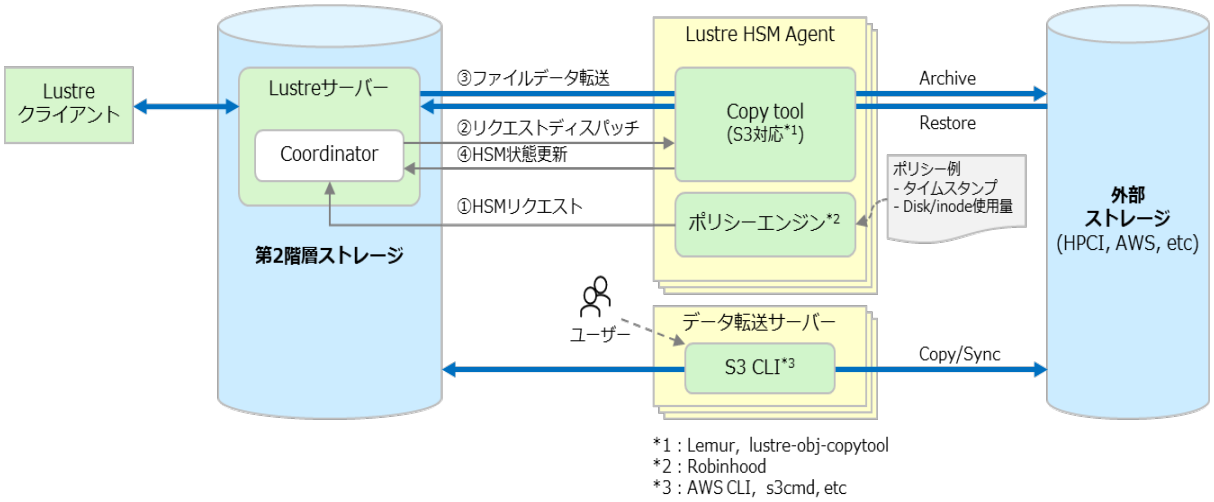


Figure 3-44 Data transfer between the second-tier (Lustre) and external storage

The types and number of agents and data transfer servers shown in the figure will be discussed in the detailed design phase and beyond. Figure 3-44 The overview of the tools used in the figure is as follows.

Used for automatic transfer

- Lemur
  A toolkit for HSM on Lustre.
- Lustre-obj-copytool
  A tool for archiving and restoring data between Lustre's HSM functionality and external object storage (such as S3-compatible services).
- Robinhood
  A versatile tool for managing the contents of large-scale file systems.
  It implements advanced features for the Lustre file system (such as listing/purging files by OST or pool, and reading MDT change logs).

Used for manual transfers

- AWS CLI
  A tool that allows you to manage AWS services using command-line shell commands.
- s3cmd

A command-line tool and client for uploading, retrieving, and managing data on Amazon S3, Google Cloud Storage, DreamHost DreamObjects, and other cloud storage service providers that use the S3 protocol.

Furthermore, when advancing the detailed design for the advancement of data management and policy-driven operations, the following points must be taken into consideration.

- Automation of Data Provenance Management
  - iRODS, which has a proven track record in academia, offers flexibility in provenance management and lifecycle automation for petabyte-scale scientific data by leveraging the latest "Policy Composition" framework
  - Suitability as a foundation for centrally managing logical namespaces and securely federating data with domestic and international research institutions, even in environments with mixed heterogeneous storage
- Metadata-Driven Governance and Cost Optimization
  - The effectiveness of visualization and "Cost Control" functions for billions of files using metadata-driven tools such as STARFISH Storage
  - An operational model that maintains data governance while optimizing storage costs through the detection of duplicates and visualization of access patterns in massive amounts of unstructured data
- In-situ (on-the-spot) analysis workflows:
  - The applicability of "in-transit" technologies, such as Intelligent Light (Kombyne), which perform visualization and data reduction directly in memory without moving data during the execution of ultra-large-scale simulations

3.4.3.3 Storage System Support and Performance Analysis Tools

3.4.3.3.1 Expected Functions

This chapter reports the results of our investigation and evaluation of tools designed to support the operation and performance analysis of storage systems. These tools are expected to support the maintenance of the FugakuNEXT storage system's health, the identification of bottlenecks, and the improvement of operational efficiency through the collection, visualization, and analysis of performance metrics. Based on the results of our preliminary design review, we have decided to proceed with development according to the following guidelines:

1. As a general rule, utilize the monitoring and analysis functions provided by the selected file system.
2. Any functional gaps in operations will be addressed by implementing the minimum necessary environmental configurations and developing auxiliary tools (e.g., scripts for integrating performance data, custom dashboards, etc.).

It should be noted that the specific items to be supplemented will be finalized during the detailed design phase after the file system has been decided.

3.4.3.3.2 Selection of Candidate Tools

To enable storage system operational support and performance analysis, we conducted a survey of representative technologies currently available. The survey focused on Lustre, which is widely adopted in flagship systems, and related tools for the VAST Data Platform, which is gaining traction in the AI infrastructure domain. Note that this survey is merely an example; final adoption will be contingent upon additional evaluation and consideration based on the characteristics of the target file system and operational requirements.

1. VAST Management System (VMS)

- Main Uses

Integrated management of VAST storage, performance monitoring, and data flow analysis.

- Features
  Instant visualization of performance metrics such as bandwidth, IOPS, and latency, as well as capacity information such as data reduction rates and metadata usage. Supports multi-tenant environments and visualizes resource usage for each tenant.
- Key Features
  - Access via GUI Dashboard, CLI, and REST API
  - DataFlows (I/O path tracing)
  - Analytics (Detailed Metrics)
  - Top Actors (Tenant Usage)
  - VAST Catalog (Metadata Database)

2. LustrePerfMon

- Main Uses
  Long-term trend monitoring, time-series analysis, and dashboard visualization of Lustre environments.
- Features
  Utilizes an OSS stack (collectd + InfluxDB + Grafana) to enable flexible dashboard creation. Strong in time-series analysis. A standard monitoring platform widely used in the Lustre community; commercial versions are also available (e.g., ScalePOD Barreleye).
- Key Features
  - Metrics collection via Lustre plugins
  - Time-series data management with InfluxDB
  - Dashboard display via Grafana (by OST/MDT/OSS, by job)

3. MELT (Monitoring Extreme-scale Lustre Toolkit)

- Main Applications
  Wide-area monitoring, on-demand diagnostics, and root cause analysis in ultra-large-scale clusters.
- Features
  Achieves scalability across multiple networks and low overhead through an extended TBON (Tree-Based Overlay Network) structure. Monitors both servers and clients to identify I/O conditions in detail.
- Key Features
  - Measurement and display at the server, client, and job levels
  - Identification of high-load I/O via a top-command-like display
  - Operation and display via CLI

4. LASSi

- Main Applications
  I/O analysis, contention detection, and monitoring of performance fluctuations at the job/application level.
  Features
  Uses proprietary Risk/Quality metrics to quantify job-level read/write performance fluctuations for each MDS/OSS, analyzing file system slowdown risks and application-level I/O efficiency. Supports scheduler integration.
- Key Features
  - Generate job-level profiles from I/O statistics
  - Detection and identification of file system anomalies and performance degradation
  - Automatic generation of daily reports
  - Information retrieval via CLI

5. lltop / xltop

- Main Uses
  Real-time I/O load monitoring on a per-job basis.
- Features
  Features a top-command-like CLI with excellent responsiveness. Specializes in capturing instantaneous load. Simple implementation makes it easy to deploy.
- Key Features
  - Displays detailed I/O statistics by job and host
  - Advanced filtering and sorting capabilities
  - CLI-based operation and display

6. LMT (Lustre Monitoring Tool)

- Main Uses
  Server-side (MDS / OSS / LNet) load monitoring, history analysis, and failure analysis.
- Features
  History storage using Cerebro and MySQL, with visualization via ltop and lwatch. Strong in detailed server-side monitoring and long-term history management.
- Key Features
  - Metrics collection via Cerebro plugins
  - History storage using MySQL
  - Display via CLI (ltop) and GUI (lwatch / lstat)
  - Generation of daily and monthly reports via aggregation scripts

Summarizing the findings above, the following criteria were identified for tool selection.

- Selecting tools based on their characteristics and intended use
  - CLI-based tools (lltop, xltop, MELT, LMT)
    Lightweight and highly responsive, suitable for understanding instantaneous load, identifying noise sources, and automation.
  - GUI-based tools (VMS, LustrePerfMon)
    Excellent for intuitive operation, data visualization, time-series analysis, and integrated operational management. Can also complement automation through API integration.
- Comprehensive data collection and analysis capabilities
  - Optimization of metric granularity and collection intervals
    Requirements regarding data collection granularity (e.g., coarse granularity with low overhead during normal operations, fine granularity during abnormal conditions).
  - Scheduler integration
    Quickly identify I/O bottlenecks, noise sources, and their scope of impact caused by specific jobs or applications.
    Ensuring scalability and low overhead
    Minimizes monitoring overhead and enables wide-area monitoring even in hyperscale environments.
- Synergistic Effects Through Multi-Tool Integration
  - Considerations for Combined Use
    If a single tool cannot meet all requirements, combine multiple tools to leverage their respective strengths (e.g., real-time monitoring + long-term trend analysis).
  - Integration Capabilities
    Ease of integrated operation through common data formats, API integration, etc.
- Operational Structure and Costs
  - Expertise and Implementation Costs
    Evaluate the workload associated with implementation and maintenance.
  - Availability and Scope of Commercial Support
    Measures to mitigate risks during long-term operation.
  - Community support
    Frequency of OSS tool updates and level of active information sharing.

- Operational autonomy and security/resilience
  - AI-driven autonomous storage management
    Predictive diagnostics and anomaly detection for storage health using AI/ML models, as seen in NetApp BlueXP and VAST
  - Cyber resilience technologies enabling "autonomous defense and rapid recovery within minutes" against ransomware attacks and other threats
  - Next-generation encryption technology and data integrity
    Anticipating future threats, the introduction of **quantum-resistant cryptography (QRS)** protocols based on end-to-end encryption, as well as AI-powered data inconsistency detection and automatic correction capabilities

3.4.4 Performance Measurement Policy and Benchmark Methodology

This chapter describes the performance measurement policy for file systems and the benchmarking methods used to quantitatively evaluate the stability of performance measurements.

3.4.4.1 Performance Measurement Policy

Since the introduction of the Tier 2 storage system is expected to be phased, it is necessary to examine the target performance at each implementation stage. Therefore, the performance measurement policy is as follows.

- Target performance shall be determined by calculating bandwidth and IOPS based on the storage system scale at each phase of the phased deployment. The same applies even if the entire storage system is not deployed due to budget constraints or other reasons.
- The file system shall be identical at each stage of the phased deployment. Details regarding performance evaluation—such as whether it is sufficient to achieve the target performance based on the scale of the first deployed storage system—will be examined during the detailed design phase and beyond.
- The total memory of the compute nodes specified in the performance requirements refers to CPU memory excluding GPU memory, and at this stage, only read performance is considered.

3.4.4.2 Benchmarking Methodology

In describing the file system benchmarking methodology, the following assumptions are made.

- Whether the first-tier file system configuration will be a local file system or a shared file system is currently under consideration. Therefore, we will conduct our analysis to ensure that quantitative evaluation is possible regardless of the file system configuration. Additionally, the first tier will be evaluated across multiple compute nodes.
- For the quantitative evaluation of the file system, multiple measurements will be taken, and the average value will be used rather than the maximum value from the user's perspective. Furthermore, to evaluate stability, we will confirm that the variation from the average value remains within a certain range (e.g., ±10%). The number of runs and the range of variation will be determined during the detailed design phase or later.
- Since users of the file system tend to perform operations such as file creation within a single directory, the benchmark will also evaluate performance for a single directory.
- In evaluating file system performance, we also anticipate scenarios where data is transferred directly from a GPU to the file system, such as with GPUDirect Storage. Furthermore, the evaluation of file system performance using GPUs will be addressed during the detailed design phase and beyond. Note that the definition of "all memory on the compute node" in the performance requirements refers to memory connected to the CPU, not the GPU.
- Depending on the job, file open/close operations from some ranks may be delayed due to interference from other jobs. To evaluate stability, one possible approach is to run multiple benchmarks simultaneously and evaluate the results of each benchmark. Details will be discussed in the detailed design phase and beyond.

Table 3 -27 lists representative tools used for quantitatively evaluating file system performance. It also indicates benchmark programs that are available for applications where I/O performance is an issue on Fugaku. Note that evaluation methods using GPU-based tools and applications will be examined during the detailed design phase and beyond.

Table 3-27 : Representative Tools and Applications for Quantitatively Evaluating File System Performance

Tool/Application Name	Evaluation Details
IOR	・Measures I/O throughput performance ・Measures GPU I/O throughput performance
mdtest	・Measures metadata performance
IO500	・Comprehensive evaluation of file systems
NVIDIA GPUDirect Storage (GDSIO)	・Measure GPU I/O throughput performance
Elbencho	・Measuring I/O throughput performance ・Measures metadata performance ・Measures GPU I/O throughput performance
Deep Learning I/O Benchmark (DLIO)	・Measuring storage I/O performance during deep learning training
Athena++	・Application-based benchmarks ・File output for FPP and SSF
R2D2	・Application-based benchmarking ・SSF file output via MPI-IO
NICAM	・Application benchmarks ・FPP file output

The basic approach to performance measurement is outlined below

- Systematization of a multi-stage evaluation protocol

  We implement an evaluation framework that progressively clarifies performance from the physical limits of storage to the effective performance of applications across the following five phases. A basic example is shown below

  Phase 1: Understanding Physical Communication Limits (osu_micro_benchmarks)

  Identify the physical limits of the communication channel (pipeline) and verify how closely it matches physical response times in a virtual environment. Evaluation criteria include low latency at the 18–19 μs level and maintaining bandwidth close to theoretical values

  Phase 2: Basic Evaluation of Storage Alone (fio)

  Using simple, easily tunable tools, we verify the raw potential (Read/Write/IOPS) of the storage system itself as the saturation point under a single-client constraint

  Phase 3: Detailed HPC Workload Evaluation (IOR, mdtest, pfind)

  Evaluate resistance to high-concurrency write contention (lock contention) on a single shared file (SSF).

  For metadata performance, quantify the scalability limits during concentrated access to a single directory (hotspot) using metrics such as files/sec

  Confirm superiority over the OS standard serial `find` command through parallel search performance evaluation using `pfind`

  Phase 4: AI Effective Performance Analysis (DLIO Benchmark)

  Using the AI industry standard DLIO, introduce **AU (Accelerator Utilization)**—which indicates the utilization rate of computational resources (GPUs)—as the most important metric

Quantify the impact of data supply from storage on the computer's "wait time (data stall)"

Phase 5: Advanced Scenario and Stability Evaluation

Evaluate the effectiveness of QoS control and the rate at which effective performance is maintained during contention caused by the coexistence of multiple jobs and workloads

Points to Note:

Visualize the **performance gap (overhead)** that arises between the "storage potential (fio value)" obtained in Phase 2 and the "actual application performance (DLIO value)" obtained in Phase 4

Establish an analysis workflow to identify that the "true bottleneck"—where effective performance remains at only a fraction (e.g., approximately 10%) of its potential even when there is sufficient bandwidth on the storage side—lies in CPU-side data preprocessing or parallel control

To eliminate the influence of OS caching, we standardize file sizes set to at least twice the physical memory capacity and the use of Direct I/O

While adhering to international standards such as IO500, we measure sustainable throughput rather than transient performance by setting appropriate execution times, such as "Stonewall time"

An overview of each tool is provided below. For details on Athena++, R2D2, and NICAM, refer to the "Research Report on Next-Generation Computing Infrastructure (FY 2024) [System Research and Investigation (RIKEN) Team] FY 2024 Research Report (2)."

- IOR

  IOR is a benchmark tool capable of evaluating file system I/O performance using various I/O interfaces and access patterns. Supported I/O interfaces include HDF5, HDFS, S3, NCMPI (an API for parallel access to NetCDF), MMAP, and RAODS.Additionally, the following access patterns are available: "Single Shared File," where each MPI process performs I/O on the same file, and "File Per Process," where each MPI process performs I/O on a unique file. Furthermore, starting with version 4.0 of IOR, it is possible to evaluate GPU I/O performance using GPUDirect Storage. Therefore, it can be used for quantitative evaluation of bandwidth.

- mdtest

  mdtest is a benchmark tool that evaluates the metadata performance of a shared file system. It performs file creation, information retrieval, and deletion operations. You can specify whether each process performs these operations on a single directory or whether each process performs them under a unique directory. Therefore, it can be used for quantitative evaluation of IOPS.

- IO500

  IO500 is a benchmark tool developed to evaluate the storage performance of HPC (High-Performance Computing) systems. It evaluates performance using standardized test methods that simulate various access patterns encountered in real-world applications. This allows for the comparison of storage performance across different systems. The workloads used in IO500 are shown in the table at 3 -28 .The results of each IO500 test, run for 300 seconds, are used for evaluation. The IOEasy and IOHard APIs can use the same API as IOR.

  IO500 is a benchmark tool with a different purpose than evaluating storage performance requirements. Therefore, it is not suitable as a benchmark tool for evaluating storage requirements. However, it can be used to evaluate stability.

Table 3-28 Details of IO500 Measurement Patterns ( )

Item	Measurement Details	Tool Name	File Access Pattern
IOEasy	I/O Throughput Performance	IOR	・Read/write via FPP ・Any IO size can be specified
IOHard	I/O Throughput Performance	IOR	・Read/write via SSF ・Each process repeatedly reads and writes 47,008 bytes of data at regular intervals (47,008 bytes × number of processes) read/write
MDEasy	Metadata Performance	mdtest	・Each process creates, stat, and deletes a 0-byte empty file under its own directory
MDHard	Metadata Performance	mdtest	・All processes create, write, stat, read, and delete files under a single directory (writing and reading 3,901 bytes)
Find	File Search Performance	pfind	・Searches for files matching a pattern within a specified directory (targeting files created by MDHard)

- GDSIO
  An I/O benchmark tool for NVIDIA's GPUDirect Storage. It assigns a single process to a single GPU and measures the throughput and latency of the direct data transfer path between GPU memory and storage. There are no options to specify I/O interfaces or access patterns.
  Since this tool is designed to run on a single node, it is not suitable for evaluations involving multiple nodes or multiple processes.

- elbencho
  Elbencho is a benchmarking tool capable of evaluating the latency, throughput, and IOPS performance of file, object (S3), or block storage. It supports GPUDirect Storage, enabling direct evaluation of data transfer performance via the GPU. It does not support MPI and can be executed in various environments. The I/O interface is limited to POSIX. The Single Shared File and File Per Process access patterns can be used.
  It can be used for quantitative evaluation of bandwidth and IOPS. However, because it does not support MPI, it cannot evaluate the performance of writes using the Single Shared File method via MPI-IO, which is a Tier 2 requirement.

- Deep Learning I/O Benchmark
  A benchmark suite designed to emulate the I/O patterns and behavior of deep learning applications. It is adopted as the underlying technology for the MLPerf Storage Benchmark, an AI performance evaluation metric. It supports various data formats and dataset configurations and uses configuration parameters similar to those of actual deep learning applications. It emulates the I/O patterns of a wide range of deep learning applications.
  Since the evaluation method for this tool varies depending on the machine learning workload being evaluated, whether to use it for quantitative file system performance evaluation requires consideration during the detailed design phase or later.

Details regarding the input data and access patterns of the Deep Learning I/O Benchmark are as follows.

- Input Data (Training Data)
  - TFRecord, HDF5, NPZ, PNG, JPEG, CSV, etc.
- Access Patterns
  - Read: Training data
  - Write: Checkpoints (saving dummy data such as weights for each epoch)
  - Uses PyTorch/TensorFlow APIs
- Models (for emulating I/O patterns)
  - U-Met3D (image segmentation), ResNet-50 (image classification), BERT/LlaMA (natural language processing), CosmoFlow (cosmological parameter estimation), etc.
- Benchmark Output
  - Summary Report: Throughput, latency, total I/O operations, bytes, data read volume, output time, etc.
  - Profiling: Start and end times for each file I/O operation, read/write byte counts, file paths

3.4.4.2.1 Tier 1

This section describes benchmarks and verification methods for measuring bandwidth and IOPS at the first level. While some benchmark options are listed here, details will be determined after the memory size of the compute nodes is finalized.

Whether the Tier 1 file system will be a local file system or a shared file system is currently under consideration. The tools described in this chapter assume a shared file system. Therefore, in the case of a local file system, evaluations must be performed for "File Per Process" or individual directories per process, and the directory structure of the local file system used for evaluation must be the same on each compute node.

In the case of a shared file system, evaluations will be performed using a single shared file or a single directory.

- Bandwidth: It must be possible to write the contents of all memory on a compute node in 2 minutes or less. Read performance must be equivalent or better.
  - Local File System (Example)
    # IOR -a HDF5 -i 5 -F -w -r -t 1m -b <memory_size ÷ 1m> -o <outputfile>
    The -F option specifies "File Per Process." If not specified, it defaults to "Single Shared File."
  - Shared File System (Example)
    # IOR -a HDF5 -i 5 -w -r -t 1m -b <memory_size ÷ 1m> -o <outputfile>
    Verification Method
    The write and read times in the IOR output are each 2 minutes or less

- IOPS: Metadata processing (including file creation/deletion, read/write) IOPS must be 8K or higher per single compute node
  - Local File System (Example)
    # mdtest -F -i 5 -n 1000 -w 1024 -e 1024 -r -u -d <outputdir>
    The -u option specifies a separate directory for each process. If not specified, all processes will use the same directory.
  - Shared file system (example)
    # mdtest -F -i 5 -n 1000 -w 1024 -e 1024 -r -d <outputdir>
  - Verification Method
    The value of (file creation, write, read, and delete operations per second) ÷ number of nodes must be 8K or higher

3.4.4.2.2 Tier 2

This section describes the benchmarks and verification methods for measuring the bandwidth and IOPS of the second-level file system.Some benchmark options are listed below, but details must be determined separately once the memory size of the compute nodes has been finalized. Additionally, the evaluation of the second-level file system bandwidth uses the Single Shared File method via MPI-IO. Two access patterns for the Single Shared File method are considered, as shown in , Figure 3-45 . The segments in the figure represent the amount of data written by processes on each compute node.In access pattern (A), data equivalent to the memory size of a compute node is treated as a single segment and written. This write operation is repeated for all compute nodes. In access pattern (B), data equivalent to the memory size of a compute node is divided into multiple segments. The write operation for a single segment is repeated for all compute nodes. This is repeated until the write operations for all divided segments are complete.

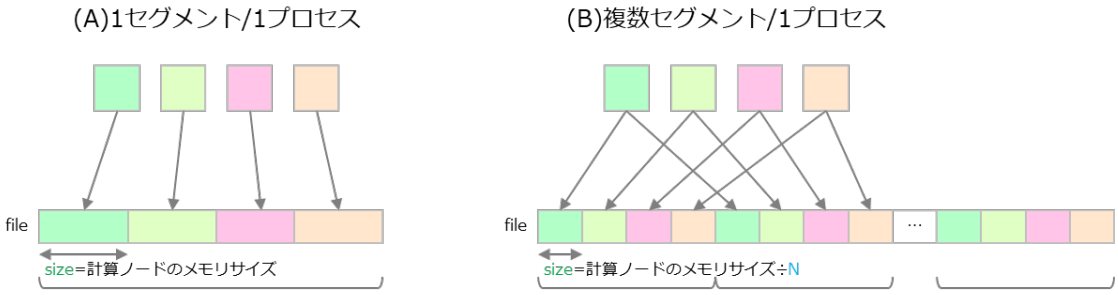


Figure 3-45 Access Patterns for 's Single Shared File

The access patterns described above can be implemented by specifying the -s option in IOR. The -s option specifies the number of segments, and the default value is 1. Examples of the options for I/O patterns (A) and (B) are shown below.

(A) IOR -w -r -t size -b size -s 1 -o /path/to/file

(B) (B) IOR -w -r -t size -b (size/N) -s N -o /path/to/file

- Bandwidth: The entire contents of the compute node's memory must be written to a file in 10 minutes or less using the Single Shared File method via MPI-IO. Read performance must be equivalent or better.
  - Shared File System
    # IOR -a MPIIO -c -i 5 -w -r -t 1m -b <memory_size ÷ 1m> -s 1 -o <outputfile>
  - Verification Method
    IOR output write and read times must be 10 minutes or less, respectively

- IOPS: For I/O operations requested simultaneously from all nodes, the system must operate stably without stopping I/O processing, achieving a performance of 1K IOPS or more per node.
  - Shared File System
    # mdtest -F -i 5 -n 1000 -w 1024 -e 1024 -r -d <outputdir>
  - Verification Method
    The output value for file creation, write, read, and deletion (operations/sec) divided by the number of nodes must be 1K or higher

3.4.5 Configuration Planning and Cost Estimation Under Multiple Budget Scenarios

This chapter describes the results of the technical survey conducted with storage vendors and the evaluation of technical options, taking into account multiple budget scenarios.

3.4.5.1 Expected Specifications

This section describes the expected specifications for Tier 1 and Tier 2 storage, focusing particularly on flash device specifications and the partitioning and deployment configuration of Tier 2 storage, based on the interview items from the technical survey conducted with storage vendors.

3.4.5.1.1 Device Types

This section describes the considerations regarding the expected specifications for the flash devices (SSDs) that constitute Tier 1 storage, from the perspectives of cost estimation and design feasibility.

1. Basic Information

Since the SSD generation, interface, and form factor affect maximum performance, implementation density, and maintainability, they are organized as basic information.

- Generation (PCIe Gen)
  - The interface generation must align with the performance targets. To ensure compatibility with the node architecture, the SSD must support the required generation and lane configuration. Examples include Gen5, Gen6, and Gen7.
- Manufacturer
  - To ensure procurement and long-term operation, continuous supply, firmware updates, and maintenance support must be guaranteed.
- Product Grade
  - Since operating temperature, warranty period, and durability guarantees vary, it is necessary to select a grade appropriate for the intended application. Examples include Enterprise, Datacenter, and Client.
- Product Name/Controller Name
  - To understand the generation, features, and track record, it is essential that the product name and controller information be clearly specified.
- Interface
  - To ensure physical connectivity and proper installation, the interface must be compatible with the slot configuration. Examples include PCIe x4, x8, U.2, U.3, E3.S, and E3.L.
- Protocol
  - To ensure OS/driver compatibility and management functions, the supported protocols must meet the requirements. Examples include NVMe, SATA, and SAS.
- Form Factor
  - A form factor must be selected that supports the node architecture, implementation density, and maintainability (hot-swap capability). Examples include E1.S, E3.S, E3.L, U.2, U.3, and M.2.

2. NAND Configuration Information

Since the NAND structure serves as a prerequisite for performance, endurance, and cost, it must be organized as configuration information.

- Cell Type
  - Cell types must be carefully evaluated to ensure a balance between capacity, performance, endurance, and cost. Examples include SLC, MLC, TLC, and QLC.

- NAND Chip Technology
  - Since these factors affect capacity, performance, and endurance, the number of layers and cell structure must be clearly defined. Examples include 3D NAND layer counts (96L–232L) and structures (CTF / FG).
- Controller Channel Configuration
  - Since this affects internal parallelism and maximum performance, the number of channels must meet the requirements. Examples include 8 to 16 channels.
- Page Size
  - Since this affects write efficiency and GC (Garbage Collection) behavior, the page size must be clearly defined. Examples include 16KB to 32KB.

3. Capacity and Controller Tier

Since capacity and the controller directly affect the number of units required, as well as unit cost and power consumption, the specifications must be defined.

- Capacity
  - To estimate the number of servers, number of racks, total weight, and cost, it is necessary to select the capacity per SSD. Examples include 7.68 TB, 15.36 TB, and 30.72 TB.
- Controller IC
  - Since this affects performance and stability, it is essential that supported features and track record are clearly defined.

4. Performance Information

To meet expected performance (bandwidth, IOPS, latency), key performance metrics must be defined.

- Latency
  - To evaluate the responsiveness of small I/O operations, latency must be understood. Examples include 80 µs for read and 15 µs for write.
- Sequential Read/Write
  - To estimate the maximum performance during large-capacity transfers, we assess sequential performance. Examples include Read 14,000 MB/s / Write 7,000 MB/s.
- Random Read/Write IOPS (4KB)
  - To evaluate small-block I/O, determine the 4KB random IOPS. Examples include Random Read 5,500K IOPS / Random Write 900K IOPS.

5. Reliability and Durability (Quality)

To ensure long-term operation, organize endurance and failure rate metrics.

- RND_TBW (Random Write Endurance) / SEQ_TBW (Sequential Write Endurance)
  - To evaluate the continuity of stable operation, determine the write endurance. Examples include RND 10 PBW / SEQ 30 PBW.
- AFR (Annual Failure Rate) / MTBF (Mean Time Between Failures)
  - Understanding failure rate metrics is required as a prerequisite for failure estimation and maintenance planning. Examples include AFR 0.35% / MTBF 2,000,000 hours.

6. Power Consumption and Thermal Design

Since this directly impacts rack power and cooling design, power and thermal specifications must be defined.

- Maximum Power Consumption During Operation
  - Since this affects the facility's power supply plan and operational cost estimates, the maximum power consumption must be determined. Examples include 25W (E3.S) / 40W (E3.L).
- Cooling Design and Thermal Management
  - To ensure continuous stable operation, the necessary cooling mechanisms and thermal

countermeasures must be implemented. Examples include airflow design, heat sinks, and thermal throttling support.

7. Physical Specifications

Organize dimensions and weight from the perspective of mounting constraints and mounting density.

- External Dimensions (W×H×D)
  - Used to verify physical interference and tray compatibility. An example is 76 × 111 × 7.5 mm.
- Weight
  - Verified from the perspectives of rack load capacity and serviceability. An example is 120 g.

8. Release and Pricing Information

Price information is compiled to serve as the basis for cost estimates.

- Release Date / Projected Release Date
  - This affects the implementation plan and procurement lead time. For example, sample shipment is scheduled for Q1 2028, and mass production is scheduled for Q1 2029.
- Reference Price
  - Confirm price information to serve as the basis for cost estimation.

9. Special Notes

Provide supplementary information to ensure consistency with requirements and to clearly convey important considerations. For example, include factors such as "E3.S" for capacity considerations or "Ex.x" for cooling considerations.

3.4.5.1.2 Partitioning and Installation Configuration

This section describes the considerations regarding the expected partitioning and deployment configuration in preparation for the phased introduction of Tier 2 storage and to accommodate multi-year budgets.

1. Types of Devices to Be Used

To balance required performance and cost efficiency, it is necessary to select the storage media configuration.

- All-Flash Configuration
  - A configuration capable of ensuring I/O responsiveness in high-performance areas (metadata and high-IOPS applications) is required. Examples include NVMe, E3.S, and Gen5-6 compatible solutions.
  - The configuration must be capable of meeting performance requirements while ensuring successful implementation and operation. Examples include all-NVMe configurations, ZFS-based configurations, and RAID-10/Erasure Coding (EC) configurations.
- Flash + HDD Configuration (Hybrid)
  - This configuration must be applicable to areas where high capacity and cost efficiency are prioritized. Examples include configurations that place frequently accessed data on flash storage.
  - To balance performance and capacity, the system must be capable of appropriately designing a cache hierarchy. Examples include NVMe + SAS HDD + HSM configurations.

2. Capacity and Performance Requirements

To clarify the prerequisites for cost estimation, we will organize the assumed values for total capacity, bandwidth, and IOPS.

- ALL-Flash

- Quantify total capacity to incorporate it into unit-based design and cost estimation. For example, the feasibility of a petabyte-class configuration is a possibility.
- To set performance targets, we quantify bandwidth. Examples include performance in the GB/s to TB/s range.
- To evaluate small-block I/O performance, quantify IOPS (4KB read/write).

- Flash + HDD
  - Organize the analysis from the perspectives of cache tier performance (burst bandwidth, write buffer performance), HDD tier sequential performance (sequential read/write performance of individual HDDs), and mixed I/O performance (variability).

3. Redundancy Methods

To ensure continuous operation even in the event of a failure, we will organize the redundancy methods.

- Storage Server
  - To ensure continuous operation even during failures and to simplify operational management, an appropriate mode configuration is required. Examples include Active/Active or Active/Standby. The switchover time (failover time) must also be verified.
  - Controllers To ensure data protection in the event of a controller failure, a synchronization method is required. Examples include NVRAM mirroring and ZIL logging.
  - To maintain stable I/O, a communication method that meets the requirements is necessary. Examples include InfiniBand, RoCE, and Ethernet.
- Shelf
  - To ensure shelf-level availability, the system must include configuration redundancy. Examples include dual-path and power supply redundancy. Additionally, verify the presence of cable redundancy.
  - To avoid path failures during I/O operations, connection redundancy is required. Examples include SAS switch redundancy or NVMe-oF path redundancy.
  - To avoid operational downtime, components must be redundant and easy to replace. Examples include redundant fans, redundant power supplies, and hot-swap capability.
- DISK
  - To prevent data loss in the event of a disk failure, a data protection mechanism is required. Examples include RAID 6, RAID 10, and Erasure Coding (EC). Additionally, verify the rebuild time.
  - To ensure recoverability in the event of a failure, metadata must also be redundant. Examples include HA configurations for MDS/MDT.
  - To maintain consistency during failures and reboots, consistency guarantees are required. Examples include journaling or write ordering.

4. Scalability and Phased Deployment

To enable phased implementation and support for multi-year budgets, it is necessary to clarify the units of expansion and configuration transitions.

- Scaling Units
  - To enable phased deployment, the expansion method must be divisible. Examples include expansion by node or by shelf. Verify whether scale-out is possible.
  - To enable expansion aligned with multi-year budgets, the unit of capacity addition must be clearly defined. Examples include expansion by shelf or by rack. Verify whether flexible phased deployment is possible.
- Configuration Diagram
  - To verify the feasibility of the initial deployment, the vendor must be able to present the current configuration (initial deployment configuration). For example, obtaining a configuration diagram provided by the vendor is one option.
  - To verify the feasibility of future expansion, it is required that future expansion configurations be presented for each expansion phase. Examples include phase-

specific configuration diagrams and the availability of cost estimates for scale-up.

- To ensure design flexibility, resource mapping must be flexible. For example, you could establish placement policies for MDS, OSS, and clients.

5. Power Consumption

Since this directly impacts power supply planning and operational cost estimates, it is necessary to define the units for power estimates.

- Power Configuration
  - To ensure alignment with power supply capacity, it is necessary to determine the maximum power consumption when the entire system is in operation. This is expected to be presented in kilowatts (kW).
  - To ensure the necessary level of detail for planning distribution panels and other equipment, it is necessary to determine the power consumption of the smallest divisible unit. Examples include the minimum node unit or shelf unit.
  - To evaluate the feasibility of phased deployment from a power perspective, it is necessary to determine the additional power consumption required during expansion. An example of this is the additional power required during scale-out.
  - To incorporate this information into the design of PDUs and other components, it is necessary to determine the power consumption at the rack level. Examples include the total power consumption of the entire rack (in PDU units).

6. Need for a UPS

To design in accordance with the protection policy during power outages, determine the necessity and capacity of UPS systems.

- UPS Configuration
  - To meet protection requirements during power outages, it is necessary to define whether a UPS is required and its capacity. Examples include the design of backup time and power supply paths. Additionally, the feasibility of short-term power outage countermeasures and UPS-less configurations must be confirmed.

7. Racks

Since this affects installation costs and portability, installation requirements (specifications, number of units, weight, and mounting configuration) must be clarified.

- Standards
  - To ensure the feasibility of rack mounting and layout design, it is necessary to clarify rack standards. Examples include EIA, JIS, and Open Compute.
  - To develop an installation plan that includes redundancy, it is necessary to specify the required number of units. For example, this may involve specifying the number of units for each configuration phase.
  - To ensure compliance with installation space and delivery conditions, rack dimensions (W×D×H) must be specified. Examples include 600×1,000×42U.
  - To meet load-bearing requirements, the total weight when fully loaded must be specified. Examples include the rack weight (total weight) when storage is installed.
  - To ensure proper implementation, maintenance, and cooling design, it is required to provide a mounting configuration diagram. For example, this could include a diagram visualizing the number and location of servers/shelves, as well as the air cooling paths.
  - To ensure proper cabling and configuration management, it is necessary to be able to specify the storage configuration components (number of servers, number of shelves, cabling configuration). For example, obtaining drawings provided by the vendor is one approach.

8. Budget Breakdown

To establish a multi-year implementation plan, it is necessary to organize a breakdown that

includes operating costs in addition to initial costs.

- Cost Breakdown
  - To understand the total cost of the overall plan, it is necessary to be able to calculate the cost at maximum capacity. Examples include initial implementation costs, annual maintenance fees, and power costs.
  - To evaluate phased deployment scenarios, it is necessary to understand the cost per smallest unit of deployment. Examples include price breakdowns by node or by shelf.
  - To ensure the accuracy of expansion budget estimates, it is necessary to be able to determine the additional cost per expansion unit.

9. Cooling and Environment

To meet installation environment constraints, it is necessary to clarify the cooling methods and their efficiency.

- Cooling Method
  - To meet facility equipment requirements, the cooling method must be compatible. Examples include air cooling, water cooling, and hot water cooling. Additionally, confirm whether hot water cooling or liquid immersion cooling options are available.
  - To ensure a viable thermal design for high-density configurations, it is necessary to ensure high cooling efficiency. Examples include intake and exhaust structures, as well as rack layout constraints.

10. Price and Proposal Information

Since this affects the decision to accept or reject the proposal and the total cost, the differentiating factors of the proposal must be clarified.

- Proposal Features
  - To expect benefits from added value, it is desirable that proprietary technologies and added value be presented. Examples include hot water cooling technology, proprietary EC, and QoS control.

3.4.5.2 Configuration Review

In this section, we have extracted items from the list of questions for storage vendors discussed in the previous section that particularly affect cost estimates, and present the options for anticipated specifications for each component of Tier 1 and Tier 2 storage.Please note that the options and estimated values presented in this section are subject to change based on future technological trends. Based on the results of this review and the information provided by each storage vendor, we will continue to evaluate future trends during the detailed design phase and determine the final configuration.

3.4.5.2.1 Types of Devices to Be Used

The basic policy for devices used in Tier 1 storage is to select SSDs based on the latest technology capable of stable supply, with the system's launch in 2030 in mind. Based on future storage technology trends and vendor information, the options for the primary SSD specifications currently anticipated are shown below. Note that, particularly regarding SSD capacity and Level Cell, we will collaborate with the architecture team and conduct further review during the detailed design phase.

- Basic Information (Basic Specifications Used as Assumptions for Estimates)
  - PCIe Generation: Gen5, Gen6, Gen7
    (While dependent on file system performance efficiency and SSD capacity efficiency, Gen7 is expected to be necessary to achieve performance targets. We will conduct a detailed review based on information provided by storage vendors and benchmarks, and

will also consider options such as a mixed deployment of Gen 6 and Gen 7 or a phased introduction of different generations.)
    - Product Grade: Enterprise, Data Center (Enterprise-grade, which ensures high reliability, durability, and performance during 24/7 operation, is the leading candidate. Classification depends on the vendor)
    - Interface: PCIe GenX x4, x8 (PCIe generation as mentioned above; 4 lanes are the mainstream)
    - Protocol: NVMe, SAS (NVMe delivers the highest performance)
    - Form factor: E3.S, E3.L, U.2, U.3 (EDSFF enables high-density, high-performance deployment)
    - Capacity: 4–8 SSDs per node (requires at least three times the installed memory capacity)
    - Weight: Varies by product. For E3.S/L, 100–150 g.
- NAND Configuration
    - Level Cell: SLC, MLC, TLC, QLC (Balance of capacity, speed, reliability, and cost)
    - NAND Chip Technology: 3D NAND
- Performance (Omitted due to NDA restrictions)
- Reliability and Durability (Omitted due to NDA information)
- Power Consumption and Thermal Design
    - Power Consumption: (Omitted due to NDA information)
    - Cooling Mechanism: Air-cooled/Water-cooled, Heat Sink Shape/Airflow Measures/Thermal Throttling Support

For Tier 2 storage, both Flash+HDD and All-Flash configurations were considered as candidates to achieve high capacity, high performance, and cost efficiency.Regarding the devices to be used for Tier 2 storage, our basic policy is to select SSDs and HDDs based on the latest technologies that can ensure a stable supply, with the system's launch in 2030 in mind. Based on future trends in storage technology and vendor information, the options for the primary specifications of SSDs and HDDs currently under consideration are shown below.

- Basic Information (Basic Specifications Used as Assumptions for Estimates)
    - PCIe Generation (SSD): Gen5, Gen6, Gen7 (Gen6 is expected to be the latest generation with a stable supply)
    - Product Grade: Enterprise, Data Center (Enterprise grade, which ensures high reliability, durability, and performance during 24/7 operation, is the leading option. Classification depends on the vendor)
    - Interface:
        - SSD: PCIe GenX x4, x8 (PCIe generation as mentioned above; 4 lanes are the mainstream)
        - HDD: SAS, NL-SAS
    - Protocol:
        - SSD: NVMe, SAS (NVMe maximizes performance)
        - HDD: SAS, NL-SAS
    - Form factor:
        - SSD: E3.S, E3.L, U.2, U.3
        - HDD: 2.5-inch, 3.5-inch
    - Capacity: (Omitted due to NDA information)
    - Weight:
        - SSD: Varies by product. 80–150g for 2.5-inch form factor
        - HDD: Varies by product. 650–750 g for 3.5-inch
- NAND Configuration (SSD)
    - Level Cell: SLC, MLC, TLC, QLC (considering the balance between capacity, speed, reliability, and cost)

- NAND Chip Technology: 3D NAND
- Performance
  - Based on the aforementioned specifications, the following performance levels are estimated to be achievable (excluding any technological breakthroughs over the next two years).
    - SSD (per unit): (Omitted due to NDA information)
    - HDD (per unit): (Omitted due to NDA information)
- Reliability and Durability
  - Based on the aforementioned specifications, the following performance is estimated to be achievable.
    - SSD: (Omitted due to NDA information)
    - HDD: (Omitted due to NDA information)
- Power Consumption and Thermal Design
  - Power Consumption: (Omitted due to NDA information)
  - Cooling Mechanism: Air-cooled/Water-cooled, Water-cooled door (device is air-cooled)
- Network Device Configuration
  - Scale-out Network (Computing Network)
    - Connection Standards: Ultra Ethernet or InfiniBand
    - Quantity: 4–8 per pair of storage servers (meta/data)
    - Notes: Calculations are based on the assumption that the storage network and compute network are shared, and that a mechanism for traffic path distribution is in place. Network separation depends on cost and will be further evaluated during detailed design.
  - Management Network
    - Connection Standard: 1GbE or 10GbE
    - Quantity: 6–14 per pair of storage servers (meta/data)
    - Remarks: The portion allocated to storage within the overall system management network

3.4.5.2.2 Capacity, Bandwidth, IOPS

Estimate the capacity, bandwidth, and IOPS of Tier 1 storage. Based on the device information described above, estimate the capacity, bandwidth, and IOPS of Tier 1 storage. The estimated and target values, assuming 5 TB of installed memory and 4,000 compute nodes, are as follows. Note that the estimated I/O performance values do not account for technological breakthroughs over the next two years.
(Details omitted as they contain NDA information)

Estimate the capacity, bandwidth, and IOPS of Tier 2 storage. Based on the aforementioned device information, the redundancy methods described below, and scalability, estimate the capacity, bandwidth, and IOPS of Tier 2 storage. Calculations were performed by comparing these estimates with the required specifications, assuming 5 TB of installed memory and 4,000 compute nodes.

- All-Flash case: (Omitted due to NDA restrictions)
- For Flash+HDD: (Omitted due to NDA information)

Since capacity and performance targets are based on the storage allocation within the overall budget, these target values are subject to constraints due to fluctuations in the allocated budget. For example, if the allocated budget is reduced to 50% of the initial plan, the capacity and performance targets will also be limited to approximately 50% of the initial values.

3.4.5.2.3 Redundancy Method

The following lists the options currently under consideration for the redundancy scheme of Tier 1 storage.

- All Compute Nodes: Node-local Storage, Near-node Local Storage
  - For details, refer to the section at3.4.2.3.1 . At this time, we are assuming a configuration in which the local storage of individual compute nodes is shared among jobs.
- Data Protection Methods: RAID (RAID6, RAID10), Erasure Coding, Mirroring
  - If each node's local storage consists of 4 to 8 high-performance SSDs and the data protection method within the node is RAID6 or RAID10, the actual usable capacity will be approximately 70% of the total physical device capacity.
  - Depending on the selected file system, data protection across multiple nodes can be implemented using Erasure Coding.

The following lists the options currently being considered for redundancy in Tier 2 storage.

- Storage Servers: Active/Active, Active/Standby, Multi-node Active
  - From the perspective of operational continuity (high availability) and performance improvement, the storage server is assumed to have an Active/Active high availability (HA) configuration.
- Physical Enclosure: Controller redundancy, dual network paths, redundant power supplies, multiple enclosure connections
  - For the controllers and shelves housing the storage servers, we also assume redundancy at the enclosure level (dual network paths, redundant power supplies, and multiple enclosure connections).
- Data protection methods: RAID 6 (or equivalent), Erasure Coding, Mirroring
  - Data protection methods for storage pools composed of SSDs and HDDs. When implementing redundancy such as RAID 6 or Erasure Coding, the actual usable capacity is approximately 70% of the total capacity of the physical devices.

3.4.5.2.4 Scalability and Phased Implementation

The following describes the options currently anticipated regarding the scalability and phased implementation of Tier 1 storage.

- Prerequisites
  - Tier 1 storage is assumed to be a single distributed file system that utilizes the local storage (composed of multiple SSDs) of each compute node and coordinates via the inter-node network.
- Deployment Phases: Sequential deployment in accordance with compute node construction / Preparation for future node expansion
- Scaling Units (Expansion Method): Per node, per group of nodes
  - If dedicated storage servers are required, expansion by node group is also anticipated.
- Capacity Addition Units: Per drive (SSD), per node, or per node group
- File system integration: Identical or different (at each stage)
  - Generally, it is assumed that expanded drives or node groups will be integrated into the existing Tier 1 file system
  - Depending on the selection of the first-tier file system, integration may not be necessary (dynamic creation of scratch areas during job execution)
- Scalability (Upper Limit): 5,000–10,000 compute nodes; 128 GB–50 TB per node
- Operational impact during expansion: Service interruption / Brief outage / None
  - Whether operational downtime of Tier 1 storage is required during file system expansion

List the options currently under consideration regarding the scalability of Tier 2 storage and the feasibility of phased implementation.

- Budget allocation: Lump-sum payment at initial deployment, multi-year budget (including equipment procurement costs)

- Number of implementation phases: 1–6
  - If the number of phases is small, the difference in capacity and performance between phases is large, the required budget for each fiscal year is large, and the implementation effort and operational impact are small.
  - If there are many phases, the difference in capacity and performance between phases is small, the budget required for each fiscal year is small, and the implementation effort and operational impact are large.
- Unit of division (expansion method): Node, shelf, rack
  - Even if expansion by node or shelf is supported, selecting expansion by rack may be considered for design and operational efficiency.
  - Consider not only the lifespan of the drives but also the lifecycle management of the entire enclosure, including controllers and other components.
  - It must be possible to separate and exclude equipment to be decommissioned from the Tier 2 storage system as phased deployment progresses.
- Capacity addition units: Drive (pool), shelf, rack
  - When expanding on a drive (pool) basis, controllers and racks are installed in advance, so the lifespan of the controllers must be taken into account.
- File system integration: Identical or different (at each stage)
  - At each stage, choose whether to configure a single file system (mounting multiple file systems in parallel from the compute nodes) or to integrate with an existing file system (or both).
  - It is possible to select different file system types at each stage and use them appropriately depending on the application. Consider whether integrated operation will allow you to enjoy the benefits of performance, capacity, and convenience.
- Total Capacity (Minimum–Maximum):
  - Due to factors such as the retention of existing equipment resulting from phased implementation, the total capacity may temporarily fluctuate within a range of 0.5 to 3 times the initial capacity. The capacity to be achieved at the start of operations and at peak capacity will be examined during the detailed design phase and beyond.
  - At this stage, estimates are based on the assumption that the maximum total capacity will be implemented in six phases.
- Operational Impact During Expansion: Service interruption / Brief outage / None
  - Whether operational downtime of Tier 2 storage is required during file system expansion

3.4.5.2.5 Power Consumption

Power consumption for Tier 2 storage is estimated based on the number of racks and servers specified in the "3.4.5.2.7 " section. The estimated results are shown below.

- Total Power Consumption (Tier 2 as a whole): (Omitted due to NDA information)

3.4.5.2.6 Need for a UPS

The necessity of a UPS was examined as follows.

- To safely shut down the storage system and protect data in the event of a power outage, "FugakuNEXT" requires an advanced power supply mechanism equivalent to a UPS, just as "Fugaku" does. "Fugaku" does not require a UPS because it relies on the facility's advanced power supply system (gas turbines).
- We provided input to the Construction, Installation, and Integration Review Committee, specifying an advanced power supply system equivalent to a UPS for storage as a mandatory requirement.
- In the case of a UPS, the capacity must be sufficient to ensure the time required to safely shut down the storage system after a power outage (20 to 30 minutes).

3.4.5.2.7 Racks (specifications, number of racks, rack size) and number of storage servers

This section presents the results of our estimates for the number of Tier 2 storage racks and storage servers, based on the aforementioned specifications (device selection, capacity/performance, redundancy, scalability, etc.). Rack specifications are assumed to be EIA standard (19-inch 42U racks). These estimates are based on currently available information and are subject to ongoing review in light of future technological trends and the results of system specification studies.

Table 3-29 Number of Storage Devices and Racks (Estimated) (Omitted due to NDA information)

Table 3-30 Number of Network Devices and Racks (Estimated) (Omitted due to NDA information)

Figure 3-46 Example Rack Installation Diagram (All-Flash Configuration: Storage Unit) (Omitted due to NDA information)

Figure 3-47 Example rack mounting diagram (Flash+HDD configuration: Storage unit) (Omitted due to NDA information)

Figure 3-48 Rack Installation Example (Common: Network Equipment) (Omitted due to NDA information)

Figure 3-49 Rack Installation Example (All-Flash Configuration) (Omitted due to NDA information)

Figure 3-50 Example Rack Layout (Flash+HDD Configuration) (Omitted due to NDA information)

3.4.5.2.8 Budget Breakdown

Since it is difficult to calculate a specific budget amount at this stage, we will closely monitor market trends and continue to gather vendor information. The budget for Tier 2 storage will be estimated on an ongoing basis starting from the detailed design phase, based on the quantity breakdown shown in Section3.4.5.2.7 .

3.4.6 Considerations for the Detailed Design

This chapter summarizes the items requiring further refinement and decision-making in the detailed design phase, based on the results of the basic design review. These are areas where a general direction was established during the basic design phase but where more specific specification decisions and technical verification are required. It also presents considerations necessary to ensure flexibility in anticipation of future operations and expansion.

Table 3-31 List of Considerations for Detailed Design

No	Consideration	Relevant Chapter	Remarks
1	Conduct a detailed comparison of node-local storage and near-node-local storage to determine the hardware configuration for Tier 1 storage.	3.4.2.3.1.1	
2	Organize requirements for the storage system and, in coordination with storage vendors, develop a design to ensure that a storage system meeting these requirements is implemented. Additionally, perform performance estimates based on information provided by each vendor as needed, and organize the data to enable verification of the validity and comparative evaluation of the proposed storage systems.	3.4.2.3.3 3.4.2.4.4	
3	Conduct an evaluation of MPI-IO asynchronous I/O, including whether file system support is required.	3.4.2.3.3 3.4.2.4.4	
4	Examine methods for utilizing Tier 1 and Tier 2 storage via GPUs.	3.4.2.3.3 3.4.2.4.4	
5	Based on information from each vendor, we will select a file system and evaluate the need for functional development and the need to combine multiple file systems.	3.4.2.3.4 3.4.2.4.5	
6	Determine the required number of storage servers.	3.4.2.4.1.1	
7	Regarding the use of Tier 2 storage, we will also consider a configuration where it is virtually partitioned rather than used as a single volume.	3.4.2.4.1.1	
8	Given the potential difficulty in meeting the write performance requirements due to the balance between capacity and performance, consider revising the performance requirements.	3.4.2.4.1.1	
9	We will explore specific solutions and implementation methods for a segmented deployment.	3.4.2.4.2	

10	Regarding data transfer between the first and second tiers, specific considerations will be made after the job scheduler and file system supported by "FugakuNEXT" have been determined.	3.4.3.1	
11	Examine data transfer patterns between the first and second tiers from an AI perspective, including the use of GPUDirect.	3.4.3.1.2	
12	We will examine checkpoints and restarts in conjunction with the application.[4]	3.4.3.1.2	
13	Examine the following issues regarding the llio_transfer command. ・Communicating usage instructions to users ・Transferring a 0-byte file	3.4.3.1.2	
14	Regarding support for each data transfer pattern between the first and second tiers, specific considerations will be made after the job scheduler and file system supported by "FugakuNEXT" have been determined.	3.4.3.1.3	
15	Determine the supported external storage.	3.4.3.2	
16	The method of integration between external storage and the second tier, as well as the data transfer method, will be determined through comparison and evaluation in conjunction with the software and data transfer services to be used.	3.4.3.2	
17	Examine specific countermeasures for transfer load issues caused by data movement frequency and insufficient transfer speeds when integrating with external storage, such as the HPCI shared storage currently under consideration for Fugaku.	3.4.3.2	
18	Regarding storage system operation support and performance analysis tools, identify development items for features lacking in the selected file system.	3.4.3.3.1	
19	We will examine methods for evaluating performance at each stage of a phased deployment.	3.4.4.1	
20	Examine stability evaluation methods, such as the number of runs and the range of variation, as benchmarking techniques.	3.4.4.2	
21	We will examine the evaluation of file system performance using GPUs as a benchmarking method.	3.4.4.2	
22	We will examine evaluation methods for benchmarking that account for the impact of external disturbances from other jobs.	3.4.4.2	
23	We will examine the evaluation of file system performance using applications as a benchmarking method.	3.4.4.2	

24	We will examine evaluation methods using elbencho and the Deep Learning I/O Benchmark as benchmarking methods.	3.4.4.2	
25	We will examine the IOR I/O size as a benchmarking method after determining the amount of memory installed on the compute nodes.	3.4.4.2.1	
26	We will examine plans for total capacity at the start of operations and when existing equipment remains in place during phased deployment.	3.4.5.2.4	
27	Continue reviewing the breakdown of the number of storage devices, racks, and budget based on information obtained from the storage vendor.	3.4.5.2.8	
28	Evaluate the necessity of developing a tool to distribute large volumes of files across N nodes, taking into account the functionality of the first-tier file system.		
29	We will continue discussions with the storage vendor regarding the storage architecture and share the results at the regular meetings of the Storage Review Committee.		
30	Further narrow down the design options for Tier 1 storage. Specifically, evaluate SSD capacity, Level Cell, PCIe generation, and the necessity of separating scale-out networks, and incorporate these findings into the architectural design.		
31	Collect device information (particularly regarding SSDs) adopted by vendors providing storage devices.		
32	Examine the constraints of Tier 2 storage (particularly interconnect type and power consumption) and incorporate them into the architectural and facility designs.		
33	If features requiring development within the FugakuNEXT budget arise, we will collaborate with RIKEN to drive the project forward. (e.g., if the Tier 1 file system lacks a staging function, or if integration between the staging function and the scheduler is not supported)		
34	Optimization of POSIX Compatibility Level and Effective Performance Regarding the balance between high POSIX		

	compatibility–as seen in DAOS 2.6 (DFuse) and similar systems–which allows existing applications to run without modification, and I/O acceleration, a final decision will be made based on hardware verification during the detailed design phase.		
35	Finalization of physical resource requirements for compute nodes To avoid “memory exhaustion” (e.g., due to connection information and metadata retention)–the primary cause of I/O instability on Fugaku–strictly determine the number of SSDs installed on compute nodes and the memory capacity required for metadata operations during the detailed design phase		
36	To prevent operational breakdowns caused by requiring users to directly handle various transfer tools (such as Globus, Rclone, and Aspera), we will determine the specific specifications for middleware that enables operational control and quality assurance In particular, incorporate into the detailed design measures to resolve transfer failures involving 0-byte files (an issue with llio_transfer) and API integration functionality with the job scheduler		
37	Software Ecosystem Continuity and Maintenance Framework To ensure continuous refinement and improvement after the system goes live, the status of OSS community activities and ISV support will be carried forward as key evaluation criteria for selection.		

References

Altair. (Date unknown). Altair PBS Professional. Accessed1 , 20: 2026,19 , : https://altairjp.co.jp/pbs-professional

17S chedMD. (1 , 2026). GitHub - SchedMD/slurm: Slurm: A Highly Scalable Workload Manager. Retrieved: 2026 ,1 ,19 , Source

OpenSFS and EOFS. (Date unknown). Lustre. Retrieved: 2026 1 19 , : https://www.lustre.org/

15L awrence Livermore National Security, LLC. (7 , 2025). GitHub - hpc/mpifileutils: File utilities designed for scalability and performance. Accessed: 2026 ,1 ,19 , Source

HPCI. (Date unknown). HPCI Shared Storage. Accessed: 2026 116 , : https://www.hpci-office.jp/using_hpci/hardware_software_resource/2026/hpci_2026_st-1

Amazon Web Services, Inc. (2026). Cloud Object Storage− Amazon S3. Retrieved: 2026 1 16 , Source: https://aws.amazon.com/jp/s3/

Google. (Date unknown). Cloud Storage | Google Cloud. Retrieved16 ,1 ,: 2026, . Source: https://cloud.google.com/storage?hl=ja

30O racle. (10 , 2025). Object Storage | Oracle | Oracle Japan. Accessed: 2026 ,1 ,16 , : https://www.oracle.com/jp/cloud/storage/object-storage/

: 2026Microsoft. (2026). Azure Blob Storage | Microsoft Azure. Retrieved16 ,1 , 2026, from, . Source

Craig-Wood, Nick. (2026). Rclone. Retrieved: 2026 1 19 , : https://rclone.org/

The University of Chicago. (2026). Globus. Retrieved: 2026 1 19 , : https://www.globus.org/

Amazon Web Services, Inc. (2026). What is AWS DataSync? - AWS DataSync. Retrieved: 2026 1 19 , Source: https://docs.aws.amazon.com/datasync/latest/userguide/what-is-datasync.html

Google. (2026). Storage Transfer Service | Google Cloud. Retrieved: 2026 1 19 , Source: https://cloud.google.com/storage-transfer-service?hl=ja

Versity Software, Inc. (2026). ScoutAM - Large Scale Archival Storage - Versity. Retrieved: 2026 1 19 , Source: https://www.versity.com/products/scoutam/

HPSS Collaboration. (2026). Home - HPSS Collaboration. Retrieved: 2026 1 19 , Source: https://hpss-collaboration.org/

28w hamcloud. (8 , 2019). GitHub - whamcloud/lemur: Lustre HSM tools. Retrieved: 2026 ,1 ,19 , : https://github.com/whamcloud/lemur

16C ompute Canada. (10 , 2020). GitHub - ComputeCanada/lustre-obj-copytool: Object copytool for Lustre HSM. Retrieved: 2026 ,119 from, .

CEA. (11 , 202422). Home · cea-hpc/robinhood Wiki · GitHub. Accessed: 2026 1 19 , : https://github.com/cea-hpc/robinhood/wiki

Amazon Web Services, Inc. (2026). What is the AWS Command Line Interface? - AWS Command Line Interface. Retrieved: 2026 1 19 , Source: https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-welcome.html

s3tools.org. (2026). Amazon S3 Tools: Command Line S3 Client and S3 Backup for Windows, Linux: s3cmd, s3express. Retrieved1 from: 202619, .

VAST Data. (Date unknown). Introduction to the VAST Management System. Retrieved: 2026 1 7 , : https://www.vastdata.com/resources/lab-guides/introduction-to-the-vast-management-system-lab-guide

4X iLi. (11 , 2019). GitHub - Lustre Monitoring System. Retrieved: 2026 ,1 ,7 , : https://github.com/DDNStorage/LustrePerfMon

26: 2026B rim, J. Michael. (4 , 2015). Monitoring Extreme-scale Lustre Toolkit. Retrieved7 ,1 2015, from, .

Sivalingam, K., & Richardson, H. (October 20, 2020). Application I/O Analysis with Lustre Monitoring Using LASSi for ARCHER. *High Performance Computing, 12321* (ISC High Performance 2020 International Workshops), 255–266. Retrieved from https://doi.org/10.1007/978-3-030-59851-8_16

25: 2026H ammond, John. (1 , 2011). GitHub - lltop. Retrieved7 ,1 2013, .

HammondJohn . (4 , 201225). GitHub - xltop. Retrieved: 2026 1 7 , : https://github.com/jhammond/xltop

RawlingsKen . (5 , 201611). Lustre Wiki - Lustre Monitoring Tool. Retrieved: 2026 1 7 , : https://wiki.lustre.org/Lustre_Monitoring_Tool

: 2026T he Lawrence Livermore National Laboratory. (9 ,7 , 2025). GitHub - hpc/ior: IOR and mdtest. Retrieved7 ,1 2025, from, . Source: GitHub - hpc/ior: IOR and mdtest: https://github.com/hpc/ior/tree/main

1IO500 Foundation. (10 , 2025). IO500. Retrieved7 ,1 , 20: 2026, . Source

NVIDIA. (12 , 20255). 1. Benchmarking and Configuration Guide— GPUDirect Storage Benchmarking and Configuration Guide. Retrieved: 2026 1 7 , : https://docs.nvidia.com/gpudirect-storage/configuration-guide/

: 2026B reuner, Sven. (12 , 2025,29). GitHub - breuner/elbencho: A distributed storage benchmark for file systems, object stores & block devices with support for GPU. Retrieved1 , 20257 , at: https://github.com/breuner/elbencho

UChicago Argonne, LLC. (1 , 20261). GitHub - argonne-lcf/dlio_benchmark: An I/O benchmark for deep learning applications. Retrieved: 2026 1 7 , : https://github.com/argonne-lcf/dlio_benchmark

3.5 Facility, Power, and Cooling Design

This section summarizes the results of studies and design policies regarding the construction, installation, and integration of FugakuNEXT during the basic design phase, and clarifies the prerequisites and constraints for handing over to the detailed design phase. Rather than delving into specific implementation methods or product selection, this section presents the design policies and assumptions that must be determined based on the installation environment conditions, such as the building structure, as part of the basic design.

The main scope covered in this report is as follows.

- ✧ Basic Policies for Power and Power Supply Requirements of Computers and Peripherals
- ✧ Basic policies regarding cooling conditions for computers and peripheral equipment
- ✧ Basic policies regarding rack specifications for computers and peripheral equipment
- ✧ Basic policies regarding conditions such as the necessity of raised flooring, dust protection, and fire protection for the installation of computers and peripheral equipment
- ✧ Status of consideration of various conditions, such as installation area and floor load, based on the estimated specifications

The following items are excluded from the scope of this section.

- ✧ Computing node specifications (CPU/GPU specifications, thermal density, power capacity, scale-up/scale-out network topology)
- ✧ Specifications for front-end computers (login nodes, web interface nodes, file transfer nodes, management server clusters, etc.), second-tier storage devices, and peripheral network equipment (external connection networks, management networks, control networks, storage networks)

The following items are excluded from this report and will be addressed during the detailed design phase and beyond.

- ✧ Detailed specifications for computer and peripheral racks, power supplies, power distribution units, cooling water distribution units, and other equipment, as well as specific hardware configurations
- ✧ Specifications for power distribution and cooling water piping between computer and peripheral equipment racks and the building

In this section, to examine computer rack specifications capable of achieving high computational performance and low environmental impact with a view toward the 2030 launch of FugakuNEXT, we will focus on rack power density, power supply conditions, and cooling conditions. We will survey trends in HPC system products launched since “Fugaku” and the Open Compute Project (OCP) to identify challenges for implementation.

3.5.1 Current Status Analysis, Constraints, and Challenges

3.5.1.1 Survey of HPC System/OCP Trends

3.5.1.1.1 Survey of HPC System and OCP Rack Trends

To achieve high computational performance in FugakuNEXT, it is essential to densely pack high-performance CPUs and GPUs; consequently, the power consumption per rack housing these components is expected to be high. To understand the current status regarding rack power density, power supply conditions, and cooling conditions, we conducted the following survey of HPC system products and OCP trends since the launch of "Fugaku."

- HPC System Products: HPE Cray Supercomputing EX Series, Lenovo Neptune, Atos Bull Sequana, NVIDIA MGX NVL72
- OCP Racks: OCP ORv3 HPR (High Power Rack)

The survey results confirmed the following major trends:

Power Density: Power per rack has already exceeded 100 kW, and for OCP racks, development plans are underway to reach 300 kW and eventually 1 MW.

Power Supply Conditions: Power reception via high-capacity 3-phase 480V AC is the mainstream approach. Additionally, high voltages of 48V DC or higher are used to reduce power loss in DC distribution within the rack.Furthermore, for racks of approximately 400 kW or more (with some exceptions), such as OCP ORv3 HPR v4, the introduction of HVDC (e.g., DC ±400 V) between AC and DC 50 V is being considered to further reduce power distribution losses.

Cooling conditions: The mainstream approach assumes the intake of chilled water from the building at high temperatures of 32°C or higher, and the adoption of highly efficient water-cooling technologies with a water-cooling ratio of 85% or higher.Coolant Distribution Units (CDUs), which are cooling water supply devices for Direct Liquid Cooling (DLC), come in two types: In-Row, which is installed alongside the rows of computer racks to be cooled, and In-Rack, which is installed inside the racks.As specific examples, the In-Row type is used for the HPE Cray EX4000, which has a high power consumption of approximately 400 kW, while the In-Rack type is used for the HPE Cray EX2500, which has a power consumption of approximately 140 kW; the appropriate method is selected based on the system's heat generation.

Rack floor load: The Atos BullSquana XH2000 and Lenovo ThinkSystem SC750 V4 Neptune have high floor loads of 2,000 kg/m^2 or more.

		HPE Cray		NSCC	Lenovo ThinkSystem	Atos BullSequana		NVIDIA MGX	OCP ORv3			
		EX4000 [1]	EX2500 [1]	ASPIRE 2A [2][3]	SC750 V4 Neptune [4]	XH3000 [5]	XH2000 [6]	NVL72 [7]	HPR v1 [8]	HPR v2 [9]	HPR v3 [9]	HPR v4 [9]
Rack Size [mm]	H	2489	2302	–	2011	–	2020	(2300)	(2300)	–	–	–
	W	1181	900		600		750	600	600			
	D	1740	1719		1200		1270	1,200	(1223)			
Weight [kg]		3629	1462	–	1836	–	2035	–	–	–	–	–
Floor load [kg/m^2]		1766	945	–	2550	–	2136	–	–	–	–	–
# of Blades		64	24	–	–	~ 38	32	18	–	–	–	–
Power per Rack (max)		400 kVA @480V	141 kVA @480V	–	–	120 kW	90 kW	120 kW	–	190 kW	300 kW	800 kW ~1 MW
Power Supply (Up to power delivery to the nodes)		AC 480 V DC 380 V	AC 480 V DC 380 V	–	AC 480 V DC 48 V	–	AC 400 V DC**	AC 480 V DC 50V	AC 480 V DC 50 V	AC 480 V DC 50 V	AC 480 V DC 50 V	AC 480 V DC ±400 V DC 50 V
Cooling Conditions Inlet Water Temp. [°C] Inlet Air Temp. [°C] Liquid-cooling ratio Cooling water temperature difference		W:32 A:– 100% –	W:32 A:– 100% –	W:40 A:– 100% –	W:45 A:40 100% –	W:40 A:– 97% –	W:40 A:– 95% –	W:45 A:35 85% –	–	–	–	–
CDU type		In-Row	In-Rack	In-Row	In-Rack / In-Row	In-Rack	In-Rack	In-Rack / In-Row	–	–	–	–

[1] HPE, "HPE Cray Supercomputing EX QuickSpecs", https://www.hpe.com/us/en/collaterals/collateral.a00094635enw.html

[2] National Supercomputing Centre Singapore (NSCC), "ASPIRE2A General QuickStart Guide",

https://help.nscc.sg/wp-content/uploads/2024/06/ASPIRE2A-General-Quickstart-Guide-1.pdf
[3] National Supercomputing Centre Singapore (NSCC), “NUS NSCC i4.0 DC: A Tropical Supercomputing DC”, https://www.nscc.sg/wp-content/uploads/2022/07/NUS-NSCC-i4.0-DC-V1.0-2.pdf
[4] Lenovo, “Lenovo ThinkSystem SC750 V4 Neptune Server Product Guide”, https://lenovopress.lenovo.com/lp2009-thinksystem-sc750-v4-neptune-server.html
[5] Atos, “Evolution of the Sequana System Architecture”, https://indico3-jsc.fz-juelich.de/event/149/contributions/603/attachments/178/256/2024_05_24_Evolution-of-the-Sequana-System-Architecture____Friedrich.pdf
[6] Atos, “BullSequana XH2000 Data Sheet”, https://atos.net/wp-content/uploads/2020/07/BullSequanaXH2000_Features_Atos_supercomputers.pdf
[7] NVIDIA, “MGX Accelerated Computing Rack and Trays Specification Revision 1.0”, https://www.opencompute.org/documents/mgx-accelerated-computing-rack-and-trays-specification-101024-pdf
[8] OCP, “ORv3 High Power Specifications”, https://drive.google.com/drive/folders/1t39vYwhEwFmjXH_bXEdU2AFnubRnj7H
[9] Meta, “ORv3 HPR ‘Next’ Rack and Power Solution for AI Systems”, https://www.dropbox.com/scl/fi/q7d9pjvghet5urai2uo68/6529-Metas-ORv3-HPR-Next-Ecosystem-Solution.pdf?rlkey=n7x63uhqsm9sq4znnu3nom04d&st=xejtidce&dl=0

3.5.1.1.2 Regarding the Adoption of Liquid Cooling for Power Shelves and Power Supply Units

The Power Shelf and the Power Supply Unit (PSU) mounted within it, as specified in OCP ORv3 High Power Rack (HPR), are currently designed with air cooling in mind.In future systems, as high-density and high-power configurations become more prevalent, heat generated by the power supply section is likely to become a limiting factor for the entire system. Therefore, the adoption of liquid cooling for the power supply and other components is a critical issue requiring consideration. In this study, we conducted interviews with PSU vendors to assess their current status regarding the implementation of liquid cooling.

Based on the results of our interviews, while server vendors have requested that we explore water-cooling solutions for PSUs, we confirmed that the interior of PSUs features high-density component layout, with the components requiring cooling distributed throughout the unit. Consequently, the space available for properly positioning cooling plates and heat pipes is extremely limited, and cooling structures tend to become increasingly complex.Furthermore, it was indicated that implementing liquid cooling may require either a reduction in power supply capacity or an increase in the enclosure volume. These constraints have made it clear that, at this stage, the commercialization of liquid-cooled PSUs remains limited.

In the detailed design phase, we will continue to explore water-cooling options in collaboration with vendors, while taking into account technical constraints and implementation challenges.

3.5.1.2 Constraints

This section clarifies the constraints that serve as prerequisites to be observed in advancing the design of FugakuNEXT. These are prerequisites that must be observed in the requirements and design studies in subsequent chapters.

3.5.1.2.1 Maximum Power for Computing Nodes and Peripheral Equipment

To control power costs during the operation of FugakuNEXT, the following maximum power constraints are established.

- The maximum power consumption of the entire system, including computing nodes and the network, shall be 40 MW or less during operation. However, peak power during design

based on hardware configuration may exceed 40 MW. We anticipate operating the system with total power consumption of 40 MW or less by utilizing power management mechanisms and control functions that limit the load on computing nodes and racks during operation.

- The maximum power consumption of peripheral equipment (login nodes, storage, management servers, external connection switches, etc.) shall be 2 MW or less.

3.5.1.2.2 Cooling Conditions for Computing Nodes

With the aim of reducing environmental impact and optimizing operational costs, the following cooling conditions are imposed on the FugakuNEXT design.

- As a general rule, almost all components of the computing nodes shall employ water cooling.
- Cooling conditions shall be met through free cooling using outside air, etc., without the use of chillers.
- Cooling shall be performed using only cooling water supplied from the building, and no cooling equipment using compressors shall be used.

3.5.1.2.3 Floor Space for Computing Nodes and Peripheral Equipment

To minimize infrastructure investment and maximize space utilization, the following floor space constraints are established.

- Four rooms, each partitioned into areas of approximately 500 m^2, shall be prepared for the installation of computing nodes.
- One room, partitioned into sections of approximately 500 m^2, shall be prepared for the installation of peripheral equipment.

Under these constraints, the system design must arrange components at high density within the smallest possible installation area to maximize space efficiency.

3.5.1.3 Challenges

Based on the survey of HPC systems and OCP trends, as well as the identified design constraints for FugakuNEXT, we present the challenges to be addressed in order to achieve high computational performance, reduce environmental impact, and optimize operational costs.

3.5.1.3.1 Establishment of high-density power supply, efficient power distribution, and power supply technologies

Current HPC systems have reached a power density exceeding 100 kW per rack, and power supplies of 300 kW or even 1 MW will be required in the future. In response to this, the following challenges exist.

- Improving power supply and distribution efficiency through the adoption of high voltages: To limit the increase in AC input circuits associated with rising total rack power, the adoption of power reception technology capable of handling 3-phase AC 400 V or higher is essential.Furthermore, as power increases, the rise in current flow makes it impossible to ignore heat generation caused by conductor resistance in the power distribution paths to compute nodes, as well as the rate of increase in conductor resistance due to rising conductor temperatures in these paths. It is crucial to increase the voltage of each component while reducing losses and heat generation, thereby increasing current density.
- Reduction of heat dissipation due to conversion losses in the Power Shelf/PSU: As the total power of the rack increases, losses resulting from voltage conversion within the PSUs inside the Power Shelf, as well as the heat generated by these losses, can no longer be

ignored. Therefore, the adoption of high-efficiency (low-loss) PSUs is extremely important. Additionally, while water-cooled PSUs are not yet common, we will continue to investigate the adoption of water-cooling technology.

3.5.1.3.2 Implementation of a cooling system capable of handling high heat generation

There is a need to efficiently cool the significant heat generated by the increasing density of racks. However, the following challenges exist.

- Thorough implementation of water cooling at the component level: Our basic policy is to apply water cooling technology to as many heat-generating components as possible—including CPUs, GPUs, memory, storage devices, and network interfaces—to maximize the proportion of the system cooled by water. Furthermore, we will not treat the manufacturing costs associated with water cooling as design constraints; instead, we will evaluate them from a full lifecycle perspective, including operational costs, reliability, and future scalability, to determine the optimal configuration.
- Adoption of high-temperature water cooling: Using low-temperature cooling water for compute nodes that require significant power increases chiller operation, which is disadvantageous from the perspectives of energy efficiency and environmental impact. Therefore, from the perspectives of reducing environmental impact and optimizing operational costs, it is important to adopt hot water cooling based on free cooling, which does not require chillers.Furthermore, considering the impact of rising component temperatures on reliability, it is necessary to ensure sufficient temperature margins and conduct thorough evaluations to achieve a design capable of stable operation even under high water temperature conditions.
- Preventing PUE Deterioration During Low Power Consumption: In HPC systems, power consumption fluctuates significantly between high-load and idle states. Cooling systems unable to adapt to these fluctuations provide excessive cooling during periods of low power consumption, which causes PUE to deteriorate. Therefore, co-design of the building, CDU, and computing systems is necessary, including the optimization of cooling capacity in response to power fluctuations.In addition to adopting CDU flow control functions, it is also important to equip the system with valve adjustment mechanisms to enable load-dependent flow control in the cooling water supply section from the CDU to the compute nodes.

3.5.1.3.3 Achieving High Installation Efficiency

From a cost optimization perspective, it is necessary to optimize and minimize the installation footprint. However, the following challenges exist.

- Selection of racks that balance high density and maintainability: High-density deployment increases the number of inter-rack cables, which can hinder maintenance operations. Therefore, it is important to optimize the design to improve maintainability while maintaining high-density deployment, and to consider efficient layouts for communications, power, and water-cooling piping.
- Addressing Floor Load Capacity: As racks become more densely packed, floor load capacity also increases. Consequently, it is necessary to reinforce the room's floor load capacity, distribute the load, and implement appropriate seismic countermeasures.Additionally, safety during the delivery of heavy racks is critical. Specifically, factors such as the dimensions of openings, load-bearing capacity, and the presence of steps along the delivery route must be evaluated. In this context, strategies to reduce the total weight during delivery—such as installing rack-mounted equipment after the racks have been brought in—should also be considered.

3.5.2 Requirements

In advancing the basic design of FugakuNEXT, we clarify the requirements that all components—including the entire system and the building structure—must meet. The following requirements are classified into three categories: computing node installation areas, peripheral equipment installation areas, and common items; specific numerical targets and conditions are presented for each. These requirements will serve as the basis for future system design, equipment selection, and infrastructure construction.

3.5.2.1 Requirements for the Computing Node Area

The computing node area will house computing nodes, scale-up network switches, scale-out network switches, management and control network switches, Coolant Distribution Units (CDUs) that supply coolant to these components, and power supply units. In addition to FugakuNEXT's computational performance targets, the following requirements are presented to reduce environmental impact and optimize operational costs.

3.5.2.1.1 Power and Power Supply Conditions

- It is required to achieve operation where the maximum power consumption of the computing nodes and the entire network does not exceed 40 MW, utilizing mechanisms such as power management systems that limit the load on computing nodes.
- Equipment and devices installed in the computer installation area must be capable of receiving power at 415 V or 480 V, 3-phase, 4-wire AC.
- To reduce power loss in DC power distribution within racks, the rack power distribution must be rated at 48 V DC or higher. Furthermore, appropriate conductors, conductor shapes, and cooling structures must be adopted to suppress losses in the power distribution path and the resulting deterioration of current density due to heat generation.

3.5.2.1.2 Cooling Conditions

- To reduce environmental impact and optimize operating costs, the cooling method is based on free cooling without the use of chillers. For this cooling method, the cooling water conditions shall be ASHRAE W4 (W45), and the air conditioning conditions shall be ASHRAE A4.All installed equipment and devices must be capable of operating under these conditions. However, in Kobe, where FugakuNEXT will be installed, it is assumed that ASHRAE W32 conditions can be met except during the height of summer. Under ASHRAE W32 conditions, the system must be capable of operating at a maximum power consumption of 40 MW; when conditions exceed ASHRAE W32, a reduction in the number of operational system units is permitted.
- It is required that studies be conducted with the goal of having Direct Liquid Cooling account for at least 95% of the computing nodes' power consumption.
- Assuming the utilization of exhaust heat after cooling, it is required to install a flow rate control mechanism based on power consumption so that the difference between the inlet and outlet water temperatures from the building is approximately 10 degrees.

3.5.2.1.3 Floor Space, Floor Load, and Rack Structure

- Assuming that four partitioned rooms of approximately 500 m^2 each will be provided for computer installation, the design must pursue space-saving measures and optimize the number of rooms used while considering operational costs and efficiency.

- The floor load of the racks must be designed to be 3,000 kg/m^2 or less; if this limit is exceeded, load distribution measures must be implemented. The basic policy is that raised flooring is not required.
- Racks must be capable of high-density installation from physical, cooling, and power supply perspectives. Furthermore, appropriate racks must be selected that ensure sufficient maintainability— —to reliably allow for cable insertion, removal, and wiring verification even under conditions of high cable density. Additionally, to avoid high costs, the use of open standards, such as those established by the OCP, is required to the greatest extent possible.
- To minimize maintenance time within the computer room, the structure must be designed to facilitate easy component replacement. Furthermore, for components and units that require significant maintenance time, the structure must be designed with portability in mind so that work can be performed in a separate room (work area).
- To prevent accidental contact by individuals entering the room, whether intentional or due to carelessness, each rack must be equipped with a lockable door.

3.5.2.2 Requirements for Peripheral Equipment Installation Areas

The peripheral equipment installation area shall house front-end computers (login nodes, file transfer nodes, management server clusters, etc.), second-tier storage devices, and peripheral network equipment (external connection networks, management networks, control networks, and storage networks). The following requirements are set forth to ensure that peripheral equipment achieves the target processing capacity while reducing environmental impact and optimizing operational costs.

3.5.2.2.1 Power and Power Supply Requirements

- The maximum operational power consumption of peripheral devices (such as login nodes, storage systems, management servers, and external connection switches) must not exceed 2 MW.
- Assuming that power is supplied via a backup power source, all equipment and devices installed in the peripheral equipment installation area must be capable of handling a three-phase, four-wire 415V power supply.

3.5.2.2.2 Cooling Conditions

- The peripheral equipment itself is air-cooled, assuming approximately 30 kW per rack, and requires that exhaust heat be removed via a Rear Door Heat Exchanger (RDHx) under ASHRAE W1 conditions.
- The air cooling conditions shall be ASHRAE A1, and equipment and devices capable of cooling under these conditions must be selected.

3.5.2.2.3 Installation area, floor load, rack structure, etc.

- It is assumed that one partitioned room of approximately 500 m^2 will be provided for the installation of peripheral equipment, and it is required that the equipment be housed within this space in the most space-efficient manner possible.
- The rack floor load must be designed to be 3,000 kg/m^2 or less; if this limit is exceeded, load distribution must be implemented. If free access is required, the computer installation party is responsible for installing the necessary flooring and delivery ramps within the required area.

- To enable high-density installation of equipment and devices, and to ensure maintainability even with increased cable density, racks must be EIA/ECA-310-E 19-inch racks with a height of approximately 48U, and racks with appropriate width and depth must be selected.
- To prevent accidental contact by individuals entering the room, whether intentional or due to carelessness, each rack must be equipped with a lockable door.

3.5.2.3 Common Requirements for Computer/Peripheral Equipment Installation Areas

3.5.2.3.1 Ensuring occupational safety and health for installation and maintenance personnel

Regarding the occupational safety of maintenance personnel, it is required that safety be ensured by providing adequate equipment for maintenance personnel, as well as by separating the computer installation area from the actual maintenance room and by limiting working hours.

3.5.2.3.2 Dust and Fire Prevention

Regarding dust and fire prevention, it is desirable that various laws and regulations and the conditions listed below be met; however, the various conditions shall be reviewed in light of system implementation and operational costs.

- To prevent physical damage, reduced cooling efficiency, and the induction of short circuits or corrosion caused by airborne dust in the computer installation area and peripheral equipment installation area, it is desirable to meet the cleanliness standards of "ISO 14644-1 Class 8" for airborne dust.
- To prevent the spread of fire in the event of a fire in the computer room or peripheral equipment room, and to prevent the spread of fire from equipment catching fire during a surrounding fire, it is desirable for the equipment and devices installed in these areas to have fire-resistant enclosures compliant with "IEC 62368-1: Audio/Video, Information and Communication Technology Equipment – Safety Requirements."

3.5.3 System Specifications

This section presents the results of studies on installation configurations, floor load, installation area, and cooling/power supply conditions for the computer installation area and peripheral equipment installation area, respectively.

Regarding the main system to be installed in the computer installation area, the Architecture Review Committee and Co-design Review Committee examined a reference configuration consisting of approximately 3,400 nodes, with each node comprising two CPUs and four GPUs, connected via a quad-rail "Rail-optimized fat-tree" scale-out network configuration using scale-up network switches. This reference configuration was used for architectural review purposes and should not be interpreted as a final system specification.

As of February 2026, there were two proposals for the form factor of the Tier-1 storage to be installed within the compute nodes: EDSFF E1.S and EDSFF E3.S. It is estimated that if EDSFF E1.S is adopted, the compute node will be 1RU, whereas if EDSFF E3.S is adopted—which allows for a higher power consumption—the compute node will be 2RU; therefore, this document examines both proposals.

3.5.3.1 Installation Configuration

This section illustrates the installation layout, including the service area, for the main racks to be installed in the computer installation area and the peripheral equipment installation area. Although this study does not include a front door, the detailed design phase will incorporate a front door into the analysis.

3.5.3.2 Compute Node Rack

The compute node rack to be installed in the compute area is expected to be NVIDIA's OCP-compliant MGX rack (with rear extender), which is based on the OCP ORv3 rack. It will house compute nodes, scale-up network switches, 4RU Tier-1 scale-out network switches, management and control network switches, and power shelves.The power supply (power shelf) is assumed to be capable of receiving 3-phase, 4-wire 415V or 480V power as specified for OCP ORv3 or MGX racks. It is assumed that the power shelf will supply power at a voltage of 48V DC or higher and distribute it to compute nodes and other components via high-current-capacity busbars.

While compute nodes, scale-up network switches, and scale-out network switches will utilize Direct Liquid Cooling (DLC), management and control network switches and the power shelf are assumed to be air-cooled. This document assumes that heat from these components will be recovered via a rear door heat exchanger to minimize the release of exhaust heat into the installation room.During the detailed design phase, we will continue to evaluate multiple cooling methods, including in-row heat exchangers, to select the optimal solution based on various requirements. The preliminary rack dimensions, including the rear door, are W600 mm x H2300 mm x D1482 mm. During the detailed design phase, we will determine the optimal service area dimensions based on the building conditions.

3.5.3.2.1 Scale-out Network Switch Rack

The OCP ORv3 HPR rack is envisioned as the scale-out network switch rack to be installed in the computer installation area. It will house a 4RU Tier-2 scale-out network switch, management and control network switches, and a power shelf.The power shelf is assumed to be capable of receiving 3-phase, 4-wire 415V or 480V power as specified by OCP ORv3. It is assumed that the power shelf will supply power at a voltage of 48V DC or higher and distribute it to the 4RU Tier-2 scale-out network switches via high-current-capacity busbars.

While the scale-out network switches will utilize Direct Liquid Cooling (DLC), this document assumes that the management and control network switches and the power shelf (Power Shelf)

will use air cooling. To minimize the discharge of heat from these components into the installation room, this document assumes that heat will be recovered via a rear door heat exchanger.During the detailed design phase, we will continue to evaluate multiple cooling methods, including in-row heat exchangers, to determine the optimal solution based on various requirements. The preliminary rack dimensions, including the rear door, are W600 mm x H2300 mm x D1482 mm.

3.5.3.2.2 Peripheral Equipment Racks

Peripheral equipment will be selected from models compatible with EIA/ECA-310-E 19-inch racks. Although an increase in cable density is anticipated due to the higher density of peripheral equipment and the adoption of 0U PDUs, we have assumed a rack with a width of 800 mm, a depth of 1200 mm, and a height of 48U, capable of accommodating even a significant increase in the number of cables discussed in the architecture review meeting and regarding peripheral equipment.The 0U PDUs supplying power to the peripheral equipment will be selected during the detailed design phase to accommodate three-phase, four-wire 415V power. The rack size, including the rear door for recovering heat from the peripheral equipment, will be W800 mm x H2300 mm x D1482 mm.

3.5.3.3 Floor Load

The results of the floor load analysis for the computing node racks and the second-tier storage racks using HDDs—both of which are expected to have high rack density—are shown for the computing equipment installation area and the peripheral equipment installation area, respectively. The results were 2,302 kg/m^2 and 1,381 kg/m^2 respectively, both of which did not exceed 3,000 kg/m^2.Further analysis will be conducted during the detailed design phase. If the load exceeds 3,000 kg/m^2, measures such as reducing the installation density or distributing the load by installing steel plates will be considered.

3.5.3.3.1 Computational Node Rack

Assuming an MGX rack (NVL72) as the compute node rack, and using data from the Aivres KRS8500V3 (GB300 NVL72) and the nVent Rack Chiller RDHX PRO Active. We estimated that the mass of a 2RU compute node is approximately 1.2 times that of a 1RU unit, and that the mass of the Power Shelf is approximately 1.4 times that of a 1RU unit due to the increase in power output from 33 kW to 72 kW.

3.5.3.3.2 Tier 2 Storage Rack

Since the selection of peripheral equipment is scheduled to take place from 2027 to 2028, this document provides an estimate based on the values for the HPE Cray ClusterStore E1000 SSU-D (4U106) and the nVent Rack Chiller RDHX PRO Active, assuming that the second-tier storage (HDD storage) will be the heaviest among the peripheral devices.

3.5.3.4 Installation area

3.5.3.4.1 Computer Installation Area

Computer Installation Area: Based on the assumptions below, we evaluated whether the equipment could be installed in four partitioned rooms totaling approximately 500 m^2. Even in Plan 2, which involves a large number of racks, the area per room is approximately 203 m^2, indicating that installation is feasible. In the detailed design phase, we will also consider reducing the number of rooms required, taking into account the placement of surrounding equipment such as building columns and distribution panels.

[Assumptions]

- Dimensions of the 500 m^2 room: 25 m wide, 20 m long
- Racks for the computer installation area are evenly distributed across the four rooms
- The number of racks per island is tentatively set based on the number that can be cooled by a 2.1 MW In-Row CDU (*) (*Assuming Lite-on's CDU with the highest cooling capacity announced as of 2025)
- Excludes areas where installation is not possible (such as building columns and doorways) and peripheral equipment (such as distribution panels)
- To accommodate the service areas for each rack, a 1.8-meter service area was established in front of the racks, a 1.2-meter service area behind them, and a 1.4-meter service area on either side of each island, and the assumed racks listed below were considered accordingly.

	Option 1	Option 2
Compute Node Racks	36 units	54 units
Scale-out network switch racks	3	3
In-Row CDU rack	8 units	8 units
Floor space	12 m x 13.5 m = 162 m^2	15 m x 13.5 m = 202.5 m^2

3.5.3.4.2 Peripheral Equipment Installation Area

We evaluated whether peripheral equipment can be installed in a single partitioned room of approximately 500 m^2 based on the following assumptions. The result was approximately 441 m^2, indicating that installation is feasible. During detailed design, we will explore options to reduce the number of racks and conduct a review that takes into account the placement of peripheral equipment relative to building columns and distribution panels.

[Assumptions]

- The dimensions of the 500 m^2 room are 25 m wide by 20 m long
- Each island accommodates a maximum of 10 racks
- Excludes areas where installation is not possible (such as building columns and doorways) and peripheral equipment (such as distribution panels)
- Reflecting the service areas for peripheral equipment racks, a 1.8-meter service area is provided in front of the racks, a 1.2-meter service area behind them, and a 1.4-meter service area on either side of each island. The following racks are arranged accordingly

The number of peripheral equipment racks is tentatively set as follows

Front-end computer racks	2 units
Second-tier storage racks (data section)	84 units
Tier 2 storage racks (metadata section)	12 units
Storage network rack	3 units
Management and Control Network Rack	2 units
Floor space	22.4 m x 19.7 m = 441.3 m^2

For the secondary-tier storage, a Flash+HDD configuration was adopted due to the large installation area required.

3.5.3.5 Power consumption

3.5.3.5.1 Computer Installation Section

The Architecture/Co-design Review Committee is conducting estimates and interviews regarding the power consumption of major components such as CPUs, GPUs, scale-up, and scale-out network equipment. To ensure that maximum power does not exceed 40 MW, the system scale of compute nodes and networks ( ) will be determined in 2027, and operational measures such as limiting the number of active units and utilizing power capping functions will be implemented.Even if the entire system is evenly distributed across four rooms and power is limited by restricting the number of units in operation, we will continue to review the matter during the detailed design phase to ensure that the upper limit for each room is not restricted to an equal division of the 40 MW cap (i.e., 10 MW per room), thereby allowing the system to operate up to the 40 MW limit.

3.5.3.5.2 Peripheral Equipment Area

The total maximum power consumption for the Peripheral Equipment Installation Area—including the front-end computers, secondary-tier storage, peripheral network devices, and the rear doors used to dissipate their heat—must not exceed 2 MW, with a maximum power consumption of approximately 30 kW per rack. This will be addressed during detailed design, including equipment selection, through discussions regarding operational systems, the usage environment, and storage during review meetings.

3.5.3.6 Cooling Conditions

3.5.3.6.1 Computer Installation Area

Detailed design shall ensure that the equipment and devices installed in the computer installation area can operate under cooling water conditions ASHRAE W4 (W45) and air conditioning conditions ASHRAE A4. Taking into account the meteorological conditions in Kobe, where the facility is located, the detailed design shall examine options such as operating at a maximum power of 40 MW under ASHRAE W32 conditions during periods excluding midsummer, and implementing restrictions on the number of units in operation if this limit is exceeded.
We examined the water-cooling ratio for the compute node rack. We assumed that the compute nodes, scale-up/scale-out network switches, and the busbars distributing 48 V DC (50 V DC) within the rack would be fully water-cooled. All other management and control network switches, as well as the power shelves, were assumed to be air-cooled, with the power shelf loss (efficiency) set at 3.5% (96.5%).By assuming full liquid cooling for the compute nodes—which account for the majority of power consumption—and high-efficiency power supplies, the liquid cooling ratio reached approximately 97%. Even in this scenario, the power required for air cooling exceeds 10 kW. During the detailed design phase, we will evaluate the optimal cooling method by considering multiple approaches, including rear doors and in-row heat exchangers, while also collaborating with the architecture review committee to explore the feasibility of implementing liquid cooling for Power Shelves and management/control network switches.

When the system's total power consumption reaches 40 MW, a flow rate of approximately 58,010 LPM (liters per minute) is required to maintain a 10°C temperature difference between the cooling water supplied from the building and the cooling water returned to the building. On the other hand, assuming the system's standby (idle) power consumption is 30% (12 MW), the cooling water temperature difference would be approximately 3°C without flow control.From the perspective of waste heat utilization, we will examine a flow control method capable of maintaining the temperature difference between the inlet and outlet of the building cooling water at approximately 10°C even during prolonged idle periods during the detailed design phase. In this examination, we will utilize the flow control mechanisms built into the CDU. Furthermore, we will also examine the possibility of installing valves, etc., on the compute nodes to provide individual flow control capabilities.
The installation conditions for equipment to be installed in the computer room are shown below.

Water Cooling Conditions for the Computer Installation Area

Item	Specification	Remarks
Primary-side cooling water temperature	ASHRAE W4 (W45)	
Primary flow rate	3,045 LPM or more per CDU	Maximum cooling capacity of CDU: 2.1 MW
Maximum primary side pressure drop	0.2 MPa (T.B.D.)	
Maximum allowable pressure on the primary side	1 MPa (T.B.D)	
Primary side cooling water ΔT	Approx. 10℃	
Primary-side piping (coupler) diameter	3 inches (T.B.D.)	
Primary-side cooling water conditions		
BTA Concentration	10–100 ppm	
pH	7–9	
Conductivity	100 μS/cm or less	

Air-cooling requirements for equipment installed in the computer room

Item	Specification	Remarks
Supply Air Temperature	ASHRAE A4	
Humidity	ASHRAE A4	
Dew point	(T.B.D)° C or lower	No condensation
Airflow	(T.B.D) m³/min/rack	
Exhaust air temperature	(T.B.D) to (T.B.D) ℃	
Air ΔT	(T.B.D) ℃	

3.5.3.6.2 Peripheral Equipment Installation Area

Since the air-cooling conditions for the peripheral equipment installation area (ASHRAE A1) are standard for ICT equipment, cooling is expected to be feasible. Assuming the total heat dissipation is approximately 30 kW, nVent's rear door (RDHX PRO Active) has a cooling capacity of 44 kW when the cooling water temperature is 24℃ and the exhaust temperature is 27℃; therefore, cooling is expected to be feasible under the cooling water conditions (ASHRAE W1).Appropriate equipment will be selected during the detailed design phase. The water cooling, water quality, and air conditioning conditions are shown below.

Water Cooling Conditions for Peripheral Equipment Installation Area

Item	Specification	Remarks

Primary-side cooling water temperature		ASHRAE W1	
Primary flow rate		43.1 LPM or more per rack 2,873 LPM or more per room	At 30 kW/rack 2 MW per room
Maximum primary pressure drop		0.5 MPa (T.B.D.)	
Maximum allowable pressure on the primary side		1 MPa (T.B.D.)	
Primary side cooling water ΔT		Approx. 10℃	
Primary-side piping (coupler) diameter		1.5 inches (T.B.D.)	
Primary-side cooling water conditions			
	BTA Concentration	10–100 ppm	
	pH	7–9	
	Conductivity	100 μS/cm or less	

Air-cooling conditions for equipment installed in the surrounding area

Item	Specifications	Remarks
Supply Air Temperature	ASHRAE A1	
Humidity	ASHRAE A1	
Dew point	(T.B.D)° C or lower	No condensation
Airflow	(T.B.D) m³/min/rack	
Exhaust air temperature	(T.B.D) to (T.B.D) ℃	
Air ΔT	(T.B.D) ℃	

3.5.3.7 Dust-proof, fire-resistant

- Airborne Particle Standards: ISO 14644-1 Class 8 cleanliness is a common standard for ICT equipment, so compliance is expected to be feasible. Detailed design will proceed based on this standard.
- Compliance with "IEC 62368-1: Audio/Video, Information and Communication Technology Equipment – Safety Requirements" as a fire safety measure is expected to be achievable. Detailed design will proceed based on this standard.

3.5.3.8 Power Supply Requirements

This section presents the power supply requirements and power distribution diagrams for the main racks to be installed in the computer installation area and the peripheral equipment installation area.

- Number of Phases : 3-phase
- Voltage : AC 415 V ±10% (3-phase, 4-wire) or AC 480 V ±10% (3-phase, 4-wire) [Computer Room]
  : AC 415 V ±10% (3-phase, 4-wire) [Peripheral Equipment Area]

- Frequency : 50/60 Hz
- Transient Withstand Capacity : T.B.D. (To be determined during detailed design)

For the peripheral equipment installation area, assuming the use of standard peripheral devices, the voltage shall be AC 3-phase 4-wire 415V capable of supplying AC single-phase 240V. On the other hand, the computer installation area is compatible with either AC 3-phase 4-wire 415V or 480V, and a decision will be made during fiscal year 2026.

3.5.4 Installation Environment Conditions for the Building, etc.

This section outlines the installation environmental conditions for the building.

3.5.4.1 Floor Load

To facilitate high-density installation per rack, reduce environmental impact through space savings, and optimize operational costs, the floor load capacity is set at 3,000 kg/m^2 or higher.

3.5.4.2 Dust protection

To prevent physical damage, reduced cooling efficiency, and the risk of short circuits or corrosion caused by airborne dust in the equipment and devices to be installed, the areas where computers and peripheral devices are installed must meet the cleanliness standard of ISO 14644-1 Class 8 for airborne dust.

3.5.4.3 Power Supply Equipment

To ensure the stable operation of the installed equipment and devices, the following requirements apply to the power supply equipment.

- Number of Phases
  : 3-phase
- Voltage
  : AC 415 V ±10% (3-phase, 4-wire) or AC 480 V ±10% (3-phase, 4-wire) [Computer Installation Area]
  :AC 415 V ±10% (3-phase, 4-wire) [Peripheral Equipment Installation Area]
- Permissible voltage fluctuation
  : Within ±10% of the input voltage
- Power Supply Frequency
  : 60 Hz
- Power Supply Frequency Variation
  : Within ±3 Hz of the rated frequency

3.5.4.4 Cooling Equipment

To ensure the stable operation of the installed equipment and devices, the following conditions must be met for the cooling system:

Cooling water conditions for the computer installation area and peripheral equipment installation area

Item	Computer Installation Area	Peripheral Equipment Area
Primary-side cooling water temperature	ASHRAE W4	ASHRAE W1
Primary Flow Rate	58,010 LPM or more per system @40 MW	2,873 LPM or more per room @2 MW
Primary-side cooling water conditions		
BTA concentration	10–100 ppm	
pH	7–9	

	Conductivity	100 μS/cm or less

Air conditioning conditions at the installation site for peripheral equipment

Item	Specifications	Remarks
Supply Air Temperature	ASHRAE A1	
Humidity	ASHRAE A1	

3.5.5 Summary of Considerations

This section describes the major items finalized in the basic design, key considerations for the detailed design that require further review, and coordination points with other working groups and building design.

3.5.5.1 Items Finalized in the Basic Design

This section lists the items already finalized during the basic design phase. These serve as prerequisites for the detailed design.

- The maximum power capacity for computers and peripheral equipment is set at 40 MW for the computer installation area and 2 MW for the peripheral equipment installation area; power management systems and other measures shall be employed to achieve this.
- The computer installation area shall be capable of receiving AC 3-phase 4-wire 415V or 480V power, and the peripheral equipment installation area shall be capable of receiving AC 3-phase 4-wire 415V power; DC power distribution within the high-power-density compute node racks shall be at a high voltage of 48V or higher.
- Computers shall be designed for cooling using a free-cooling system that does not require chillers and shall be capable of operating under cooling water conditions of ASHRAE W4 (W45) and air conditioning conditions of ASHRAE A4. However, taking into account the meteorological conditions at the installation site in Kobe, it shall be possible to cool a maximum power consumption of 40 MW under ASHRAE W3 conditions, and operational restrictions shall be permitted if conditions exceed W3.
- The majority of the compute nodes shall utilize Direct Liquid Cooling.
- Peripheral equipment shall be capable of air cooling under ASHRAE A1 air-conditioning conditions, and models shall be selected that allow the exhaust heat to be recovered via the rear door under ASHRAE W1 cooling water conditions.
- To reduce the installation footprint, the racks for computers and peripheral equipment shall be capable of high-density mounting.
- It is assumed that free-access flooring is not required for the installation of computers and peripheral equipment.
- To ensure stable operation of computers and peripheral equipment and for fire protection, fire-resistant enclosures compliant with general airborne dust standards and safety regulations shall be provided.
- Based on the estimated specifications, it has been confirmed that the installation area and floor load are within the building specifications.

3.5.5.2 Considerations for Detailed Design

This section outlines the items to be considered during the detailed design phase.

- Rack specifications for computers and peripheral equipment, detailed specifications for power supplies, power distribution units, cooling water distribution units, and other equipment, as well as specific equipment configurations
- Power distribution specifications and cooling water piping specifications between the computer and peripheral equipment racks and the building
- Reduction of air-cooling power consumption for compute node racks and optimal cooling methods for air-cooled waste heat
- Redundancy policy and specific configuration for the CDU
- Flow rate control mechanisms in response to power fluctuations during high load and standby (idle) conditions. Based on a flow rate adjustment mechanism using CDUs, consider the possibility of adding individual flow rate control functions (valves, etc.) to compute nodes

- Optimization of peripheral equipment rack width and depth dimensions, taking into account the number of cables and maintainability
- Examination of service area dimensions optimized for building conditions
- Establishment of occupational safety and health measures for workers in the computer installation area and peripheral equipment installation area. In particular, regarding maintenance and replacement work in environments where indoor temperatures rise, consider measures to reduce work time and methods that allow work to be performed in a room separate from the installation site when prolonged work is required
- Calculation of CO2 emissions during product manufacturing and measures to improve accuracy

3.5.5.3 Coordination with other working groups and building design

This section outlines the items that should be addressed during the detailed design phase.

3.5.5.3.1 Items for coordination with the Architecture Review Committee

- The goal is to use hot water cooling and to have Direct Liquid Cooling (DLC) account for 95% or more of the computing nodes' power consumption
- The input rating for compute nodes shall be 48 V DC or higher.
- We will collaborate with the Architecture Review Meeting to advance studies on implementing DLC for power shelves and management/control network switches, as well as on reducing losses in power shelf and rack busbars and the associated heat generation.
- In collaboration with this study group, we will examine a flow rate control mechanism that responds to power fluctuations during high-load and standby (idle) periods. Additionally, we will explore the possibility of installing valves or similar devices on computing nodes to enable individual flow rate adjustment.
- To ensure occupational safety and health for workers in the computer installation department, we will examine structures that reduce work time, particularly for maintenance and replacement work in environments where indoor temperatures rise, as well as structures that allow work to be performed in a room separate from the installation site when long hours are required.

3.5.5.3.2 Coordination Items for the Storage Review Committee and the Operational System and Usage Environment Development Review Committee

- Select equipment that can be cooled under ASHRAE A1 air-cooling conditions and powered by single-phase 240V AC.
- Assuming the use of 48U 19-inch racks, we will collaborate with this study group to examine rack mounting configurations aimed at optimizing installation space.

3.5.5.3.3 Items for Coordination with Building Design

- Ensure that conditions regarding floor load, dust protection, power supply equipment, and cooling equipment are accurately reflected in the building design.

3.5.6 Data Center Digital Twin

To holistically optimize the design, installation, and operation of the building and supercomputers for FugakuNEXT, we considered constructing a digital twin of the data center and utilizing it for design and operations.In next-generation HPC/AI systems, which are becoming increasingly high-density and high-power, the interdependencies among power, cooling, layout, and operational loads are growing more complex, making optimization difficult through conventional static design studies alone. Therefore, it is crucial to conduct evaluations and verifications from the design phase through to operation using a virtual model that corresponds to the actual environment.

In this study, we aim to build an environment capable of comprehensively modeling the data center's structure, power consumption, cooling behavior, and operational status, and to enable the preemptive evaluation of the impact of changes in system configuration and operational conditions. This will facilitate design decisions based on quantitative evaluations regarding rack layout, cooling water flow control, operational policies under power constraints, and equipment redundancy configurations.

We plan to utilize ExaDigiT, an open-source platform currently under development, as the digital twin foundation. This platform is a framework for reproducing and analyzing the operational state of HPC systems and data centers, capable of comprehensively handling power consumption, thermal behavior, and workload execution status. This allows us to model the behavior of the entire system, including computers and building infrastructure, and utilize it as an evaluation platform to support design and operational optimization.

This fiscal year, to verify the effectiveness of this digital twin platform, we conducted model construction and evaluation targeting the cooling facilities at the Research Center for Computational Science and the AI4S supercomputer, which is scheduled for future deployment. In the detailed design phase, we plan to construct models of the FugakuNEXT computers and cooling equipment to advance the co-design of building and system designs, while also aiming to utilize the platform for continuous optimization and anomaly detection during the operational phase.In the future, we aim to evolve this into a dynamic digital twin that reflects the real-time status of the actual system by integrating with the operational data platform.

3.5.7 Status of Building Design

This section presents the current status of building design considerations based on the planned installation of "FugakuNEXT."

First, from the perspective of computational service continuity, rather than adopting the approach used during the transition from "K" to "Fugaku"—where the existing system was removed and a new system installed on the same site—we have decided to adopt a configuration that allows "Fugaku" and "FugakuNEXT" to operate in parallel for a certain period. Therefore, instead of utilizing the existing building, we plan to construct a new building to ensure service continuity and stability of the user environment during the system migration period.

The building is not merely a space for housing computers; it is a foundation that must be designed as an integral part of the computing system, taking into account power supply, cooling, maintenance and operation, and environmental impact.In this project, based on the findings of the feasibility study, we are proceeding with planning under the premise of a DBO (Design-Build-Operate) approach, which integrates design, construction, and operation to achieve overall optimization. This approach ensures consistency between the building infrastructure and the computing system while optimizing operational efficiency and lifecycle costs from the design phase onward.

A key priority in the building design is a dramatic improvement in energy efficiency. While conventional domestic data centers typically have a configuration where cooling power accounts for approximately 30–50% of IT power, FugakuNEXT aims to significantly reduce the cooling power ratio and achieve efficiency levels close to the world's highest standards. Specifically, the design goal is to reduce the cooling power ratio—which was approximately 35% in "Fugaku"—to around 10%.To achieve this, the project will adopt a hot-water cooling system based on free cooling that does not rely on chillers, thereby balancing improved energy efficiency with reduced environmental impact.

The facility development project is scheduled to begin with the publication of the solicitation guidelines in February 2026. Following a competitive dialogue process to select a contractor, the plan is to conclude a basic contract by the end of 2026.Multiple rounds of competitive dialogue and Q&A sessions will be held from March to June 2026 to refine the technical requirements and project conditions. Subsequently, proposals will be accepted in August of the same year, followed by presentations and hearings in September, with the preferred bidder scheduled to be selected in October. Following selection, a basic agreement and a basic contract will be signed to advance the project to the next phase.

Following the conclusion of the basic contract, the project will transition to the design and construction phase, with the completion and handover of the facility scheduled for December 2029. During this period, while finalizing the computer system specifications, the design of the power supply, cooling, building structure, and layout will be integrated and finalized, and construction and equipment installation will proceed in stages.

Following the handover of the facility, we will transition to the maintenance and management phase, which will continue through March 2036. During this phase, we will ensure the stable operation and maintenance of the building facilities while continuously optimizing them in coordination with the operation of the computing systems. Through these efforts, we aim to maintain highly efficient and stable computing operations over the long term.

As described above, this facility is not merely a standard construction project but is positioned as a next-generation data center model that achieves "high efficiency, low environmental impact, and high-density housing." In the upcoming detailed design phase, we will continue to optimize power, cooling, and installation conditions while ensuring alignment with system specifications and operational requirements.

3.6 Operational Design

This section reports on the status of discussions regarding the operational systems and user environment for FugakuNEXT at the basic design stage.The purpose of this section is to present an overview of the operational infrastructure for the next-generation flagship system, as well as to organize its design philosophy, the premises of the study, and the basic policies to be handed over to the detailed design phase. FugakuNEXT is not a system that simply extends conventional high-performance computing infrastructure; rather, it is designed as a computing infrastructure premised on a new usage model that integrates AI, simulation, and data analysis.

As Japan's core HPC system, the "Fugaku" supercomputer has supported a wide range of fields, from scientific and technical computing to industrial applications and the resolution of societal challenges. For conventional HPC workloads centered on large-scale numerical simulations, it has achieved high stability and reliability, earning high acclaim worldwide. However, in recent years, the demands surrounding HPC have undergone significant changes due to shifts in the research environment.Due to the rapid development of AI and machine learning, the expansion of data-driven research, the growing need for integration with external data platforms and the cloud, and the increase in interactive and API-driven usage, there are an increasing number of scenarios where a system architecture based on traditional batch-centric operations struggles to keep pace.

Through the operational experience with Fugaku, several structural challenges have come to light. These include a lack of flexibility to accommodate the diversification of workloads, the need to address demands for more sophisticated user interfaces, the increasing complexity of data integration with external systems, more stringent security requirements, and the significant scope of impact when making system changes. These challenges cannot be resolved through simple feature additions or partial modifications; they indicate the need to reexamine the design philosophy of the entire operational system.

"FugakuNEXT" is expected to adopt a GPU-centric computing infrastructure and serve as a core computing platform that integrates AI, simulation, and data analysis. In such an environment, it is essential to establish an operational infrastructure that ensures fair and highly efficient management of computing resources, supports the coexistence of diverse usage patterns, enables high-speed and secure data integration with external systems, incorporates robust security design, and features observability and automation.Furthermore, as the importance of AI-driven operations is expected to increase toward 2030, it is necessary to design an operational system that prioritizes sustainability and scalability rather than short-term optimization.

The scope of this report covers the basic policies for the operational management system of FugakuNEXT, the basic policies for the user environment, the approach to authentication, authorization, and account management, the policy for integration with external systems, and the organization of non-functional requirements such as availability, operability, scalability, and security. These are fundamental matters for ensuring system operation and must be finalized during the basic design phase.On the other hand, the selection of specific technologies and products, concrete configuration design, detailed implementation methods and parameter design, and the description of operational manuals and workflows are outside the scope of this report and will be left to the detailed design phase and subsequent processes.

This report first identifies challenges in the operational and usage environment of the current Fugaku system and analyzes requirements with a view toward 2030. Based on this, it presents an overall vision of the operational infrastructure required for FugakuNEXT and summarizes design considerations such as the service model, logical architecture, and aspects of availability, flexibility, and scalability.Next, this report outlines the basic design policies for each individual component, including workload management, system software, user services, operational support functions, network integration, and operational processes. In each section, items to be finalized as part of the basic design are clearly distinguished from those to be considered during the detailed design phase and beyond, and the prerequisites and design philosophies for handing over to subsequent phases are presented.

3.6.1 Current Status Analysis and Challenges/Requirements

3.6.1.1 Challenges from Fugaku

This section covers the period from the start of Fugaku operations through September 2025 and organizes challenges identified primarily through interviews with operations and maintenance staff and analysis of operational performance data. Rather than treating the identified challenges as isolated incidents, we clarify the underlying structural factors and classify them as root causes. Based on this, the objective is to derive design policies and requirements for FugakuNEXT for each root cause.

This analysis is not intended merely to list points for reflection. Its ultimate goal is to establish design guidelines for building a sustainable and scalable operational infrastructure, taking into account future changes in usage patterns and technological advancements. In other words, the aim is to systematize the challenges that have emerged with "Fugaku" from the perspective of preventing recurrence and to convert them into design constraints for "FugakuNEXT."

3.6.1.1.1 Discrepancy Between Insufficient Functional Development and Design Philosophy During the Operational Phase

Since its launch, “Fugaku” has been maintained and operated with the highest priority placed on stability and availability. As a result, no large-scale functional revisions or reorganization of the design philosophy were implemented during the operational phase; instead, the system has primarily been addressed through small-scale feature additions and partial expansions. While this accumulation of incremental improvements contributed to ensuring short-term stability, it gave rise to the following structural challenges in the long term.

First, there is a divergence from the original design philosophy. “Fugaku” was designed with a usage model centered on large-scale batch processing, and new workload types such as AI analysis, data-driven processing, and interactive use were not central to the original vision. Consequently, as usage demands have diversified in recent years, it has become apparent that the design philosophy itself is not fully aligned with these needs.

Second, there is a lack of systematic approach to functional expansion. As a result of repeated ad hoc modifications to address individual issues, the overall unity and consistency of the system gradually declined. This led to increased complexity in the system architecture, growing dependencies, and the personalization of maintenance tasks, which in turn resulted in rising development and maintenance costs and inconsistencies in the user experience.

Based on the above, “FugakuNEXT” requires a design philosophy that explicitly accounts for modifications after the system goes live. In other words, recognizing that a gap will inevitably emerge between the initial design and post-deployment reality, scalability and ease of modification must be incorporated from the design phase.

3.6.1.1.2 Insufficient Operational Specifications During Design and Associated Challenges

During the design phase of "Fugaku," discussions centered primarily on functional specifications and performance requirements, and the systematic definition of operational specifications was insufficient. As a result, management systems and operational procedures were established sequentially during the operational phase, creating a disconnect between design and operation. This has manifested in the following challenges.

First, there is a lack of automation for routine tasks. Many operational tasks are performed manually or semi-automatically, leaving room for efficiency gains and reduced human error through automation. In particular, large-scale node updates and configuration changes have led to prolonged task durations and increased operational workload.

Second, there is a lack of centralized information management. Operational information, configuration data, and monitoring data are managed across multiple systems, making it

difficult to ensure the consistency and integrity of information. This also impacts the speed of fault analysis and change management.

Third, there is a lack of observability. Although a monitoring infrastructure is in place to comprehensively grasp the system state, the data design is not sufficient for operational optimization or future advanced analysis. From the perspective of improving operational stability and enhancing the information provided to users, the design of monitoring and observability needs to be reorganized.

In light of the above, “FugakuNEXT” must explicitly define operational specifications from the design phase and incorporate operational processes into the design. We have identified the following as fundamental requirements: configuration design based on automation, centralized information management, and enhanced observability.

3.6.1.1.3 Constraints due to dependence on HPC middleware

"Fugaku" relies heavily on HPC middleware (Fujitsu's TCS), with much of the system control and job management depending on this middleware. While this configuration enabled efficient operation in the early stages, it gave rise to the following constraints in the long term.

First, there is a constraint on update flexibility. Heavy reliance on specific middleware expanded the scope of impact during OS updates or modifications to peripheral functions, making continuous improvement difficult.

Second, there was a delay in adopting the latest technologies. In ensuring compatibility with rapidly advancing technological areas such as AI platforms and cloud integration functions, the existing configuration was found to be a limiting factor in some cases.

Third, there is a divergence from standardization. In meeting modern security requirements and integrating with external systems, compatibility with standard OSS-based protocols and interfaces is crucial; however, highly proprietary configurations may limit this flexibility.

In light of the above, “FugakuNEXT” must avoid excessive reliance on specific middleware and adopt standardization and the utilization of OSS as its fundamental policy. This will enable continuous updates, ensure interoperability with other platforms, and facilitate flexible responses to security enhancements.

3.6.1.2 Requirements

"FugakuNEXT" is expected to evolve from a traditional large-scale MPI-centric computing environment into an integrated computing platform that accommodates a diverse mix of workloads, including AI, data analysis, interactive processing, and resident services. Therefore, this section outlines the required functions and design principles for each component, based on the anticipated usage patterns and technical environment as of 2030.

The requirements presented in this section do not prescribe the selection of specific technologies but rather define the requirements that should be finalized during the basic design phase.

3.6.1.2.1 Workload Management

The workloads handled by “FugakuNEXT” are expected to include not only conventional large-scale MPI jobs but also AI/ML processing, data analysis, event-driven processing, interactive processing, and resident services—all of which are expected to run simultaneously on the same platform. Consequently, operational elements such as scheduling, resource allocation, queue design, execution environment control, and usage visualization must be designed with the assumption of greater complexity than in the past.

In particular, with the widespread adoption of GPU-intensive nodes and heterogeneous configurations, a mechanism that enables the integrated management of multiple resources—such as CPUs, GPUs, memory, storage bandwidth, and network bandwidth—and allows for the

selection of optimal allocation methods based on job characteristics is essential. Furthermore, resource control must account for integration with external infrastructures, including energy-saving requirements, cloud bursting, and federated computing.

In light of the above, the workload management for “FugakuNEXT” must be constructed as a flexible management platform that integrates traditional HPC WLM functions with the management functions of cloud-native platforms.

3.6.1.2.2 System Software

System software must possess the flexibility and sustainability to comprehensively handle diversifying research paradigms—such as AI, data-driven processing, cloud integration, and federated computing—on a single platform, in addition to supporting traditional HPC applications.In particular, with the widespread adoption of GPUs, the software layers—including the OS, compilers, libraries, MPI, and I/O stacks—must be maintained in a structure that enables continuous updates and expansion while achieving advanced performance optimization. A systematic design that balances performance optimization and ease of updating is essential.

Regarding the application execution environment, in addition to conventional module management methods, the architecture must be designed to support multiple execution models, such as containers, and include mechanisms to ensure secure isolation while maintaining reproducibility and portability. Particularly for AI workloads, where library updates are frequent and dependencies tend to become complex, operations reliant on manual management are unsustainable. The establishment of an integrated management framework, including dependency analysis, version control, and vulnerability detection, is required.

Furthermore, it is important to establish a structure that can continuously incorporate vendor-optimized technologies while maintaining compatibility with the international HPC community. By establishing standard APIs and common interfaces, we will build an environment where application developers, users, and operators can work efficiently on a common foundation.

In light of the above, the system software must not only stably support the operational foundation of “FugakuNEXT” but also be developed as a flexible and sustainable software architecture capable of adapting to the diverse computational demands of 2030.

3.6.1.2.3 User Services

As an interface for users, we will develop an interface designed to enable a diverse range of users, regardless of their level of expertise, to efficiently utilize the advanced HPC infrastructure. In addition to the conventional CLI-based operation system, we require a mechanism that integrates diverse usage forms—such as web portals, GUI applications, and API integration—into a single user service layer, providing a consistent operating environment while accommodating differences in user skill levels.

With the expansion of interactive usage, visualization, and AI/data analysis applications, functions that were previously treated separately as front-end systems—such as the login environment, interactive execution environment, visualization environment, and data transfer functions—must be provided in an integrated manner as part of the user services. This will create an environment where users can perform computations, analyses, visualizations, and data operations tailored to their objectives through a consistent operational framework, without being aware of differences in execution modes or infrastructure configurations.

Furthermore, a design that prioritizes the user experience as an international research infrastructure—including multilingual support, visualization of usage status, a unified operating system, and account integration functions—is essential. User services must closely integrate with workload management, operational support systems, and operational data infrastructure to create a structure that allows users to handle jobs, data, and workflows in an integrated and intuitive manner.

Based on the above, the “FugakuNEXT” user services will be developed as an integrated UX platform that serves as the optimal entry point for both traditional HPC users and users in the AI and data science fields.

3.6.1.2.4 Operations Support System

An operational support system is expected to function not only as a means to streamline core operations such as account management, task management, and billing processing, but also as an integrated support infrastructure that underpins all research activities utilizing HPC. As usage patterns diversify and the number of users increases, operational tasks are becoming more sophisticated and complex. An operational framework reliant on individual systems or manual processes makes it difficult to ensure efficiency, transparency, and data consistency.

Therefore, it is essential to centrally manage user information, task information, job performance data, billing and allocation information, inquiry histories, and other data based on a unified data model. This enables improved operational efficiency, ensures transparency in processing, and maintains data consistency.

Furthermore, given the widespread adoption of external integration and federation, the system must possess the scalability to support integration with HPCI, data exchange with external systems, and user attribute management based on security requirements. The design must be flexible enough to adapt to future regulatory changes and shifts in usage patterns.

Furthermore, the operational support system must work in close coordination with the operational data platform and function as a foundation that supports the visualization and enhancement of operational status by comprehensively handling usage data, operational information, and audit information.

Based on the above, the operational support system will be positioned as a user, issue, and operational information management platform and developed as a core component that efficiently and securely supports the overall operation of “FugakuNEXT.”

3.6.1.2.5 Operations Infrastructure

The operational infrastructure must be designed with a focus on stability, observability, and advanced automation, based on the premise of stably operating a large-scale system comprising thousands of nodes over an extended period.It is difficult to cope with the increasingly complex operational environment—such as the advancement of heterogeneous configurations, the increase in GPU-intensive nodes, load fluctuations accompanying the expansion of AI workloads, and the sophistication of security requirements—using conventional operations that rely on individual scripts or partially distributed monitoring and management platforms.

Therefore, the operational infrastructure must be built as a platform that comprehensively handles various elements such as provisioning, configuration management, monitoring, log and metric collection, event management, incident response, user information management, and job information management. It is essential to adopt a design based on the fundamental principles of Infrastructure as Code (IaC) and Observability, enabling configuration changes, status monitoring, and incident response without excessive reliance on manual intervention.

It is necessary to establish a foundation that enables the visualization of operational status, performance optimization, and early detection of failure signs by centrally collecting, storing, and analyzing operational logs (including job information), performance metrics, event data, and audit information, thereby facilitating the advancement of autonomous operations.

Furthermore, assuming that the uses of compute nodes will diversify—such as batch processing, interactive use, and resident services—the infrastructure must possess the flexibility to rapidly adapt to changes in cluster configuration, node expansion, and operational policies. The design must clearly separate the roles of management, control, and compute systems, and localize the scope of impact in the event of a failure to ensure the reliability of the

entire system.

Based on the above, the operational infrastructure must be established as an integrated and autonomous operational core that supports the reliability, efficiency, and scalability of the entire “FugakuNEXT” system.

3.6.1.2.6 Security

Historically, HPC systems have been designed and operated on the premise of stable operation within closed environments. However, for the “FugakuNEXT” generation, connections to external clouds, observation equipment, industrial systems, the HPCI Federation, and overseas research institutions are expected to become commonplace. In such an environment, a traditional perimeter defense model alone is insufficient; a comprehensive security design based on communication and access control is required.

Therefore, the basic approach is a multi-layered defense structure combining strengthened authentication and authorization mechanisms, clear network segregation, and integrated management of audit logs. At the same time, the design must balance the requirements for high-speed data transfer with the security requirements associated with external connections. The structure will ensure high-bandwidth communication while logically and physically segregating control, management, and computing networks, enabling the application of appropriate defense levels based on usage.

To achieve both performance and security while uniformly supporting diverse usage patterns, "FugakuNEXT" adopts a security design based on a combination of communication visualization, anomaly detection, log management, and a separation architecture. Security must not be treated as an independent, single-function component but must be designed as a continuous, operations-driven mechanism integrated with the operational infrastructure.

3.6.1.2.7 Operational Processes

Operational processes must evolve from the manual maintenance, monitoring, and change management typical of conventional HPC operations to an operational model based on automation, standardization, continuous improvement, and AIOps integration.Due to increasing workload complexity, frequent library updates, cloud integration, and the use of federation with external organizations, the operational burden is expected to increase significantly compared to traditional HPC environments. In such an environment, it is difficult to maintain stable operations through ad-hoc responses or unsystematic procedures.

Therefore, it is necessary to clarify and standardize procedures (developing SOPs), systematize change management, accelerate incident response, introduce predictive maintenance, clarify service level objectives (SLAs/SLOs), and centralize document management. Furthermore, due to the diversification of the user base—which includes educational institutions, corporations, and international collaborative researchers—it is essential to maintain operational processes that ensure fairness and transparency in a mixed environment.Furthermore, the realization of AI-driven operations is also required.

Operational processes must be designed not merely as a set of procedure manuals, but as a system that incorporates a continuous improvement cycle driven by the utilization of operational data. This will establish an operational foundation capable of sustainably adapting to system expansion and changes in usage patterns.

3.6.1.2.8 Deployment and Operations Plan

The implementation and operation plan must be formulated not merely as a framework for launching a new system, but as a lifecycle plan that sustainably supports an increasingly complex and sophisticated research infrastructure. Implementation and operation are not separate processes; they must be managed integrally under a consistent design philosophy.

First, during the implementation phase, it is necessary to formulate a plan that spans all service

layers, including facility construction, power and cooling infrastructure, network design, and hardware installation, as well as user services, workload management (WLM), virtualization infrastructure, operational data infrastructure, and operational support systems. Implementation should not be considered complete merely upon the physical installation of hardware; rather, it should be premised on an integrated launch that includes the software stack, management infrastructure, and user experience layer.

In particular, for the transition from “Fugaku” to “FugakuNEXT,” it is essential to design the system with the continued use of existing user environments, application assets, and data assets as a prerequisite. It is necessary to minimize the impact on users by ensuring compatibility, establishing a migration support framework, setting a parallel operation period, and systematically implementing a phased rollout.Furthermore, the implementation plan must incorporate a system for systematically conducting multifaceted verification, including performance validation, confirmation of scheduling behavior, and verification of AI/ML workload operations. Ensuring stability during the initial implementation phase, user training, and documentation preparation are also considered critical components of the implementation process.

During the long-term operational phase following deployment, it is important to maintain a structure that enables continuous adaptation to technological advancements while assuming at least five years of stable operation. Since the frequency of software stack and library updates is expected to be high, it is necessary to maintain a design that incorporates a planned update strategy and scalability. Furthermore, given the need for connectivity with external infrastructure—such as HPCI collaboration, international joint research, and cloud utilization—it is essential to continuously maintain a configuration that prioritizes standardization and compatibility.

Furthermore, the sustainability of the operational framework is a critical factor. It is necessary to establish a system that minimizes reliance on individual expertise through the promotion of operational automation, thereby enabling the systematization and transfer of knowledge. Additionally, a comprehensive risk management framework must be established that takes into account external factors such as disaster risks and power constraints.

In light of the above, the implementation and long-term operation plan must be formulated as a strategic plan based on the core principles of sustainability, scalability, and ease of updates. This plan must ensure a smooth transition of users from “Fugaku” while remaining flexible enough to accommodate future expansions and institutional changes.

3.6.1.2.9 AI-Driven Operations

In “FugakuNEXT,” AI is positioned not merely as a computational resource for users, but as a foundational technology to enhance operations themselves. In conventional HPC operations, many tasks—such as troubleshooting, performance analysis, log analysis, responding to inquiries, configuration changes, and software updates—have relied on the experience and manual labor of operations staff.However, with the diversification of workloads, the increasing complexity of system configurations, and the rising frequency of software updates, it is becoming increasingly difficult to perform these tasks continuously and to a high standard relying solely on human labor. Therefore, the basic policy for "FugakuNEXT" is to incorporate AI into every stage of operational tasks—including execution, improvement, and the software development that supports them.

First, regarding operational tasks, the system is designed so that AI uses logs, metrics, events, job information, inquiry histories, and configuration data to detect signs of failure, estimate causes, suggest potential responses, answer inquiries, assist with configuration changes, and support operational decision-making. This will assist with investigations and decisions that were previously performed individually by operations staff, aiming to speed up responses, standardize quality, and reduce reliance on individual expertise.Furthermore, under certain conditions, we are considering the automatic execution of routine procedures—such as node

isolation, service restart, job resubmission, and notification issuance—with AI support.

Next, AI will be actively utilized in the development of software that supports operations. The operational infrastructure, operational support systems, user interfaces, and monitoring and analysis platforms for “FugakuNEXT” will require continuous improvements and functional enhancements even after operations begin, in response to policy changes, shifts in usage patterns, and the addition of new requirements.Therefore, we will utilize AI throughout the entire software lifecycle—from requirements gathering, design assistance, implementation, and testing to documentation and maintenance—with the aim of improving development speed and maintainability . This will enable us to perform modifications to operational software, which previously entailed a significant human workload, in a shorter timeframe and with greater flexibility.

More importantly, the use of AI should not be limited to temporary automation but should be incorporated into a continuous improvement cycle even after operations begin. For "FugakuNEXT," the premise is to continuously review and improve the operational processes themselves based on actual operational data accumulated in the operational data platform.For example, by analyzing incident response histories, user inquiries, job failure trends, performance variations, and resource utilization, AI will assist in reviewing operational procedures, adjusting thresholds, improving monitoring rules, and redesigning operational workflows. As a result, operational processes will not be static but will continue to evolve based on actual data.

Similarly, operational software will undergo continuous functional enhancements based on performance data collected after the start of operations. AI will organize and analyze user requests and operational challenges to identify necessary modifications and potential new features, thereby progressively advancing the operational software. In other words, "FugakuNEXT" aims to create a cycle in which AI is not only used to perform operations but also to improve operational processes and implement those improvements.

Based on the above, the AI-driven operation of “FugakuNEXT” involves the integrated realization of the automation and advancement of operational tasks, the streamlining of operational software development, and continuous improvement following the start of operations. As a result, operations will be transformed from a static framework into a foundation that continuously evolves in response to system usage patterns and technological advancements.

3.6.2 Overview of Operations

This section presents an overview of operations on FugakuNEXT, based on the requirements outlined in the previous section. The purpose of this section is not to explain the details of individual functions, but to clarify the design philosophy that underlies them.

FugakuNEXT is expected to evolve from a traditional computing environment centered on large-scale MPI jobs into an integrated computing platform that combines AI, data analysis, interactive processing, and resident services. In such an environment, each element—including workload management, execution environments, user interfaces, operational support, operational infrastructure, security, and operational processes—must be designed with diverse computing workloads in mind, making a coherent operational design for the entire system indispensable.

The core of this operation is the integrated management of heterogeneous workloads. On FugakuNEXT, in addition to conventional batch jobs, AI/ML processing, data analysis, interactive processing, and resident services will run simultaneously on the same platform. Therefore, it is necessary to manage multiple resources—such as CPUs, GPUs, memory, storage bandwidth, and network bandwidth—in an integrated manner and allocate them optimally according to job characteristics.Furthermore, a workload management mechanism that combines HPC workload management functions with cloud-native workload management functions is required to achieve both resource utilization efficiency and flexibility while enabling diverse execution models to coexist.

The execution environment and system software supporting these workloads must be designed as a framework that balances performance optimization with continuous updates. In particular, AI workloads involve frequent library updates and complex dependencies, making operations reliant on traditional manual management unsustainable. Therefore, we will establish an integrated management mechanism—including dependency management, version control, and vulnerability response—based on an execution environment that combines module management with containers.

Furthermore, the operational design will be built around the user perspective rather than the system architecture itself. For FugakuNEXT, we will integrate multiple access methods—including the CLI, web portal, and APIs—into a single user service layer, providing an environment where users can perform computations and analyses without being aware of differences in infrastructure configuration or execution methods.

The operational support system is positioned not merely as a management function, but as a foundation supporting the entire research activity. By comprehensively managing user information, project information, resource utilization data, job performance, and inquiry history, we will enhance operational efficiency and transparency while establishing the system as a foundation that supports users' research activities. Furthermore, this information will be linked to the operational data platform and utilized for visualizing and analyzing operational status.

The operational infrastructure is designed with observability and automation at its core.By comprehensively collecting and storing logs, metrics, events, and audit information to enable a multifaceted understanding of the system's state, we achieve early fault detection and performance optimization. Furthermore, we manage configuration and provisioning via Infrastructure as Code to enable operations that do not rely excessively on manual intervention for configuration changes or fault response. This enables sustainable operations while keeping operational costs low, even in large-scale and complex environments.

Regarding security, we combine network segmentation, tenant isolation, integrated authentication and authorization, and centralized management of audit logs to systematically address security requirements—which have often been overlooked in traditional HPC—from the design phase. Furthermore, while assuming connectivity and collaboration with external clouds and other organizations, we implement service provisioning and access controls that balance security and convenience.

Operational processes will transition from manual-centric operations to a system based on automation, standardization, and continuous improvement. By systematizing change

management, incident response, and problem management, and incorporating an improvement cycle based on operational data, we aim to achieve continuous improvement in operational quality. Furthermore, we will accommodate the diversification of user groups and ensure operations that uphold fairness and transparency.

Furthermore, deployment and operation will be designed holistically as a lifecycle rather than as separate phases. Assuming a transition from "Fugaku," we must systematically implement the continued use of existing assets, phased deployment, and parallel operation. Even after deployment, we will maintain an update strategy and scalability based on the premise of long-term operation, creating a foundation capable of adapting to technological advancements and changes in usage patterns.

As described above, the operation of FugakuNEXT will be built as an integrated operational platform centered on the integration of heterogeneous workloads, a sustainable software architecture, user-centric design, data-driven operations, automation, integrated security, and continuous improvement. This operational framework will realize a shift from traditional HPC focused on stable operation to a continuously evolving platform that supports research activities.

3.6.3 System Software

3.6.3.1 Purpose and Positioning

This section establishes the adoption policy for the OS and virtualization infrastructure on FugakuNEXT's compute nodes as a prerequisite for Detailed Design 1. The scope is limited to compute nodes; the configuration of the operational infrastructure, such as management nodes, will be addressed in later chapters.If the assumptions regarding compute nodes are unclear, inconsistencies may arise across chapters regarding the interpretation of job execution methods, software provisioning, and audit requirements. Therefore, this chapter prioritizes defining constraints to eliminate uncertainty in design decisions during subsequent phases, rather than comparing candidate approaches. Furthermore, this chapter assumes that multiple user jobs will run simultaneously within a single compute node and clarifies the conditions for multi-tenancy.

3.6.3.1.1 Scope

The main points addressed in this section are as follows.

- OS Selection Policy
- Design Policy for Reducing OS Jitter
- FUJITSU-MONAKA-X OS Support Policy
- Determining the Virtualization Method
- Container Runtime Selection Policy
- Items to Be Considered and Deliverables in Detailed Design Phase 1

3.6.3.1.2 Decisions

The decisions in this section will be applied as input conditions for Detailed Design Phase 1.

- In Detailed Design Phase 1, Ubuntu will be the OS selected for initial evaluation.
- The virtualization method for compute nodes shall be the container method.
- In Detailed Design 1, Apptainer will be the initial container runtime under evaluation.
- To address OS jitter, KPIs will be established and incorporated into an operational cycle of measurement, analysis, and improvement.
- The OS and container runtime for the production environment shall be finalized at least one year prior to the scheduled system installation date, in accordance with the selection policy.
- For multiple user jobs on the same node, execution boundaries will be separated on a per-container basis, and separation of processes, file systems, and networks will be mandatory.
- To protect resources in a multi-tenant environment, usage limits for CPU, memory, GPU, and local storage will be managed on a per-job basis.

3.6.3.1.3 Outstanding Issues

The following items remain pending.

- Production Environment OS and Container Runtime

3.6.3.2 OS

3.6.3.2.1 Distribution Selection Policy

The OS will be selected based on the premise that it meets the following conditions. In making the selection, we will prioritize operational continuity and security.

- Provides full support for various devices such as GPUs, NICs, and storage
- Long-term operational continuity must be ensured

- Security updates must be available on an ongoing basis
- Must have a proven track record of use in HPC environments

After conducting an extensive survey of Linux distributions, we identified Ubuntu as a leading candidate that meets the above criteria.The reasons for this selection include the fact that NVIDIA has chosen Ubuntu as the base OS for DGX OS, suggesting strong GPU support; the availability of LTS for long-term support; and a sufficient track record of adoption in HPC, although it is less extensive than that of Red Hat Linux. While Ubuntu will not be finalized as the OS for FugakuNEXT's compute nodes at the basic design stage, it will be evaluated and its operational environment verified during Detailed Design Phase 1.

3.6.3.2.2 OS Jitter Mitigation Policy

Since OS jitter manifests as performance differences between nodes during MPI synchronization, continuous countermeasures are necessary not only during the design phase but also during the operational phase. Therefore, OS jitter countermeasures will not be limited to measurement; we will define an improvement process based on the premise that measurement results will be reflected in operational improvements. Measurement items must allow for continuous comparison, and the following are envisaged as representative KPIs.

- Maximum Allowable Latency Variation
- Latency Variance During MPI Synchronization
- OS noise occurrence rate

KPIs will be reevaluated not only at the time of initial deployment but also with each update release. The evaluation results will be used to determine configuration changes and readjustments. Specific evaluation criteria, thresholds, and the improvement process will be defined in Detailed Design 1.

3.6.3.2.3 FUJITSU-MONAKA-X Support

As a general rule, FUJITSU-MONAKA-X features will be incorporated into open-source software (OSS) such as Linux, and it is assumed that the necessary hardware support will be provided by Linux distributions. In the basic design phase, we have compiled the current status of Linux support for the planned CPU features. The need for additional development will be evaluated during the detailed design phase and beyond.

This survey is based on the CPU specifications anticipated at the time of the Basic Design; if specifications are added or changed, the support status will be updated accordingly. The support status is categorized as "Supported," "Partially Unsupported," or "Undecided." "Supported" indicates no additional development is required; "Partially Unsupported" requires a determination of whether development is necessary during the Detailed Design phase; and "Undecided" will be reevaluated after the specifications are finalized.

FUJITSU-MONAKA-X Linux Support Status (December 2025 Survey)

CPU Features	Support Status	Details
Arm Feature Support (SVE2, SME2)	Supported	Both SVE2 and SME2 are already supported in Linux. However, this support is limited to the Linux kernel; individual support is required for various computing libraries, etc., depending on their respective implementation methods.

Power Control, Power Measurement	Supported	Both power management and power monitoring functions comply with ACPI specifications and are fundamentally supported in Linux. However, support may be required in the future if the version of the compliance specification is updated to a newer one.
Virtualization Features	Partially unsupported	Hardware virtualization features (memory virtualization, interrupt virtualization), VHE support, and some nested virtualization features are supported. However, the latest FEAT_NV3 feature within nested virtualization is not supported.
Security	Partially unsupported	Pointer Authentication (PAC), Branch Target Identification (BTI), and cryptographic instructions (AES, SHA1, SHA256, SHA512, SM3, SM4) are supported. Various features related to Confidential Computing (RMEv2, PCIe encryption, NVLink-C2C encryption) are not supported.
RAS Features	To be determined	Arm RAS and SBMR have been designated as compliance standards, but the detailed specifications within them are undecided at the basic design stage, and it is not yet possible to make a definitive judgment regarding Linux support.
Performance Counters	To be determined	Information regarding performance counters is currently undecided in the basic design, and it is not yet possible to make a definitive judgment regarding Linux support. However, it is generally the case that support for performance counters—particularly the mapping between event IDs and event names—must be implemented on a per-SoC basis, and it is highly likely that this will also be required for "FugakuNEXT."

3.6.3.3 Compute Node Virtualization

The virtualization method for compute nodes will prioritize performance requirements and adopt a container-based approach. The container-based approach excels in startup efficiency and resource utilization efficiency, and aligns with the HPC job execution model. The virtual machine approach is excluded from the standard execution method and will be considered on an exceptional basis in Detailed Design 1 only if necessary.Furthermore, assuming that multiple user jobs coexist on a single node, containers will be operated as tenant boundaries, and the separation of processes, file systems, and networks will be a mandatory requirement.

3.6.3.3.1 Container Runtime Selection Policy

Container runtimes shall be selected only from those that meet operational continuity and security requirements. As a general rule, selections must satisfy the following conditions.

- Long-term operational continuity
- Ongoing security updates
- Track Record of Use in HPC

Based on these conditions, Apptainer will be evaluated in Detailed Design 1, and verification of the operational environment will be conducted.

3.6.3.4 Evaluation Policy for Detailed Design 1

3.6.3.4.1 Items for Consideration

In Detailed Design 1, we will use the assumptions finalized in this chapter as input and verify pending items under conditions equivalent to actual hardware.

- Continued investigation of OS and container runtimes (including reselection based on technology trends)
- Multi-tenant isolation and resource management methods based on Apptainer
- Definition of KPIs, evaluation methods, and operational processes for OS jitter countermeasures

3.6.3.4.2 Deliverables

The following deliverables shall be prepared in written form:

- Results of the ongoing investigation into OS and container runtimes
- Procedures for implementing multi-tenancy using Apptainer
- OS Jitter Evaluation Method and Proposed Improvement Cycle

3.6.4 Workload Management

The purpose of this chapter is to establish the basic policy for workload management in FugakuNEXT and to define design criteria that will guide the selection of methods and operational design from the detailed design phase onwards. FugakuNEXT aims to provide both HPC workloads and workloads managed by Kubernetes on the same infrastructure. Therefore, the division of roles among management methods and the delineation of operational responsibilities will be clarified during the basic design phase.

This chapter defines the types of workloads under consideration, the positioning of candidate workload management software, and the items to be examined and deliverables to be finalized in Detailed Design Phase 1. Note that in this chapter, conventional job management software for managing HPC workloads is referred to as a job scheduler.

3.6.4.1 Purpose and Positioning

3.6.4.1.1 Scope

In this chapter, we organize workload management into two distinct systems—the job scheduler and Kubernetes—and define the respective domains each will cover as design policies. Additionally, we specify the range of candidate job schedulers and the scope of Kubernetes application, and present the items to be considered and deliverables for Detailed Design Phase 1, which assumes the concurrent use of both systems. The following are excluded from the scope of this chapter.

- Evaluation of the relative merits of each software and the final decision on adoption
- Implementation methods, configuration examples, specific configuration values, and detailed configuration diagrams
- Performance tuning methods and the finalization of operational procedures
- Final selection of individual products

3.6.4.1.2 Decisions

The basic configuration for workload management on FugakuNEXT will involve the concurrent operation of two systems: a traditional HPC job scheduler and Kubernetes. Based on this, the division of roles will follow the policy outlined below.

- As a general rule, HPC workloads centered on large-scale parallel computing, batch execution, and interactive use will be managed by the job scheduler
- As a general rule, AI training, inference, and service-oriented workloads will be managed by Kubernetes.
- The boundary conditions and exceptions for this division of roles will be defined in Detailed Design 1. Furthermore, to ensure the successful operation of this dual-system approach, the following constraints will be established.
- Resource usage information collected from each system shall be standardized using common metrics such as CPU time, GPU time, memory usage, and execution time to ensure consistency for billing and auditing purposes
- Resource allocation within nodes shall be provided based on predefined resource profiles, and over-allocation shall not be permitted under standard operations

3.6.4.1.3 Outstanding Issues

The following items will not be finalized in the basic design but will be reviewed and finalized in the detailed design.

- Job scheduler product name

3.6.4.2 Types of anticipated workloads

FugakuNEXT is designed to support both AI processing and service-oriented workloads, adopting a configuration where these two types of workloads coexist on the same platform. While it was previously common to provide these on separate systems, FugakuNEXT will offer them as an integrated platform ( ).

3.6.4.2.1 HPC Workloads

HPC workloads focus on large-scale parallel computing and include both batch and interactive jobs. Since these workloads require the simultaneous allocation of a large number of nodes and involve synchronized parallel execution, operations must prioritize fairness and resource utilization efficiency.

The main targets are as follows.

- Large-scale parallel computing
- Batch jobs
- Interactive Jobs

3.6.4.2.2 Workloads managed by Kubernetes

Workloads managed by Kubernetes are intended for AI training, inference, and service-based workloads. These workloads involve a mix of short-duration jobs, must adapt to fluctuating demand, and require the continuous provision of resident services; as such, they have operational requirements that differ from those of HPC workloads.

The primary targets are as follows.

- AI training and inference jobs
- Service-oriented workloads

Even for AI training and inference jobs, job schedulers are often used, so users are not necessarily required to use Kubernetes. For workloads managed by Kubernetes, it is necessary to investigate specific user use cases and evaluate them based on anticipated demand. In the basic design phase, use cases have not yet been investigated; during the detailed design phase, we will collaborate with the AI Software WG to identify use cases.

3.6.4.3 Selection Criteria for Workload Management Software for Conventional HPC

For FugakuNEXT, candidate job schedulers will be identified based on the following criteria.

- Operational track record in large-scale HPC environments
- Sustained development framework and maintainability
- Compatibility with dual-system operation and resource partitioning

The following three job schedulers are considered candidates in the basic design.

- Slurm
- PBS Pro
- Flux

The job scheduler is positioned as a core function for HPC workload operations; its selection and application criteria will be evaluated in Detailed Design Phase 1. The evaluation will reference Fugaku's operational track record and scale, with operational feasibility and flexibility serving as the primary criteria.

3.6.4.4 Positioning of Kubernetes-based Workload Management

In FugakuNEXT, Kubernetes will not serve as a replacement for conventional HPC job schedulers, but rather as a foundation for handling workloads suited to AI and service-oriented

applications. The scope of application, target use cases, and operational prerequisites will be identified, evaluated, and finalized in Detailed Design Phase 1.

3.6.4.5 Evaluation Policy for Detailed Design 1

In Detailed Design 1, candidate job schedulers and Kubernetes will be evaluated under conditions close to the production environment, and the adoption policy and scope of application will be finalized based on the results. The evaluation will use operational feasibility under a dual-system configuration as a common criterion.

3.6.4.5.1 Prerequisites

The prerequisites for the evaluation are as follows.

- From a security perspective, jobs must be executed from within containers
- Node resources must be partitioned by CPU, memory, and GPU units and made available to users
- The job scheduler and Kubernetes must adhere to the selection policies for standard OS and container runtimes

3.6.4.5.2 Considerations

The evaluation will focus on the following aspects.

- Evaluation of the job scheduler's performance and operational flexibility
- Feasibility of billing integration
- Identification of Kubernetes use cases and evaluation of feasibility
- Suitability of resource isolation methods for dual-system operation
- Implementation and operational overhead of staging and data distribution methods for large-scale jobs

3.6.4.5.3 Deliverables

Deliverables will be organized into written documents in a format directly usable for adoption decisions and operational migration.

- Job Scheduler Evaluation Results
- Kubernetes Scope, Prerequisites, and Use Cases
- Survey and Evaluation Results Regarding Billing Integration
- Results of the Resource Isolation Method Study
- Results of the study on staging and data distribution methods for large-scale jobs

3.6.5 User Software Stack

This section does not present the results of individual software selections for the software stack at the library layer and above in "FugakuNEXT." Instead, it defines the basic design, which includes the policy for providing a standard execution environment to support long-term use and continuous expansion after the start of operations, as well as diverse usage patterns, along with the approach to its maintenance and updates.The purpose of this basic design is to clarify, regarding the software stack at the library layer and above, what will be provided as "standard," what conditions must be met, what aspects will be determined in the basic design, and what aspects will be left to the detailed design.

3.6.5.1 Standard Software Stack Provision Model

The basic policy for the software stack at the library layer and above in "FugakuNEXT" is to provide it as a standard execution environment that enables users to conduct research and development continuously and efficiently. It is assumed that the standard stack will be configured as a cohesive, integrated environment, rather than a collection of individual software components.

The standard stack must meet the following conditions:

- It must be widely used and continuously developed within the international HPC software community.
- It must have a track record of use on "Fugaku," Virtual Fugaku, and related environments, and ensure continuity with existing application assets
- It must not be dependent on specific vendors and must allow for long-term maintenance and updates

Meeting these conditions ensures that the standard software environment for "FugakuNEXT" is not a temporary configuration but remains available for sustained use.

3.6.5.2 Organization of the Scope of Provision (Standard, Recommended, User-Managed)

For software at the library layer and above, the basic policy is to organize the scope of provision based on the following principles, with the aim of reducing operational burden and preventing the standard environment from becoming bloated.

- Standard Software
  Numerical computation libraries, parallel processing-related libraries, and major middleware that play a fundamental role in the use of "FugakuNEXT" will be included in the standard stack, which the Center will be responsible for maintaining and providing.
- Recommended Software
  Software that is highly useful depending on the field or usage scenario but is not essential for all users will be classified as recommended software, and the scope of its provision and maintenance will be limited.
- User-Managed Software
  Software dependent on specific projects or individual research will be managed by the user or the project team, and will not be included in the standard stack.

This basic design establishes this classification policy, and the specific classification of software will be determined during the detailed design phase.

3.6.5.3 Basic Policy on Updates, Compatibility, and Reproducibility

The standard software stack must be configured to allow for periodic updates to keep pace with technological advancements, while also being designed for long-term operation. At the same time, from the perspective of ensuring the reproducibility of research results, it is a fundamental

requirement that users be able to understand and control the impact of updates. Therefore, when updating the standard stack, it is assumed that configurations prioritizing stability will coexist with those addressing vulnerabilities and incorporating new technologies.Furthermore, the system must include mechanisms to preserve and utilize past computational environments in a reproducible form.

Regarding specific update frequencies, support periods, and methods for maintaining compatibility, this basic design outlines only the policy, with details to be examined and determined during the detailed design phase.

3.6.5.4 Distribution Format and Relationship with the Execution Environment

From the perspective of ensuring reproducibility and portability, the software stack at the library layer and above is based on integration with container technology. The standard stack is required to be based on a container environment, while also being configurable for use through module management and package management as needed. This enables the use of the same execution environment not only in on-premises environments but also in conjunction with cloud and external computing resources.

3.6.5.5 Alignment with the International HPC Software Community Stack

In designing the software stack for "FugakuNEXT" at the library layer and above, we will treat trends in international HPC software stacks—such as E4S and OpenHPC—as reference information for verifying the validity of our approach. The common objectives demonstrated by these initiatives—such as openness, portability, and scalability—will also be prioritized in the basic design of "FugakuNEXT."At the same time, we will avoid being tied to specific software sets or distribution models, and this basic design will define only principles and prerequisites.

3.6.5.6 Matters Decided in This Section and Matters Left to Detailed Design

The items to be determined in this section as part of the basic design are as follows.

- The software stack above the library layer will be provided integrally as a standard execution environment
- The stack will be centered on open source and prioritize compatibility with international HPC software stacks
- The scope of provision shall be classified into "Standard," "Recommended," and "User-Managed"
- The configuration shall be designed to balance updatability and reproducibility

On the other hand, the following matters will be left to the detailed design phase.

- Selection and versioning of individual software components
- Update frequency and specific support policies
- Specific methods and operational procedures for container usage, module management, and package management

3.6.6 User services

3.6.6.1 User Interface

Our basic policy is to move away from traditional CLI-centric systems and build an integrated environment where these three interfaces function homogeneously, aiming to provide a consistent user experience regardless of the user's skill level or purpose.This policy aims to resolve issues in Fugaku such as fragmentation between the CLI, Web, and API; duplicate implementations; lack of integration in the authentication and auditing infrastructure; absence of data model and API definitions; increased maintenance and expansion costs; and inconsistent user experience.

The "FugakuNEXT" UI is designed with a three-tier architecture centered on the API layer. The GUI and CLI function by calling a common API, ensuring output consistency. As a result, operations in the GUI and CLI are equivalent, and the addition of new features or updates are immediately reflected in both through API extensions.The API layer defines common data models for jobs, projects, accounts, and usage statistics, and comprehensively handles authentication, authorization, and auditing functions. It also provides a standard set of APIs that allow user-created tools and external systems to submit jobs, transfer data, retrieve execution status, and perform monitoring. On the backend, an operational data platform centrally manages user information, project information, job information, and usage status information, thereby improving the consistency and reusability of the entire system.

Based on the above design principles, we will address the major issues that have emerged in Fugaku as follows:

- Standardization of functions previously duplicated or fragmented across CLI and Web interfaces: Standardize these functions through API modularization to improve maintainability.
- Reorganization of commands and tools previously implemented in multiple languages and formats: Redesign based on unified development conventions and the API-first principle.
- Performance improvement: We will clarify performance requirements and achieve fast API responses by exploring asynchronous processing and caching mechanisms. This will eliminate response delays during Web portal and CLI operations, leading to a significant improvement in usability.
- DevOps: "FugakuNEXT" will adopt a DevOps approach to strengthen collaboration between development and operations. We will establish a system for integrated management of Git repositories, CI/CD tools, and documentation, and introduce automated testing and deployment to streamline the entire lifecycle from development to operations. The use of AI is fundamentally assumed for these processes.

Based on the above, the basic design of the "FugakuNEXT" user interface assumes the provision of three types of interfaces: GUI, CLI, and API.

On the other hand, the following items will be left to the detailed design phase.

- Details of Major GUI and Web-Based Functions
  Job operation portal, visualization and monitoring dashboard, execution environment management functions, request and inquiry functions, information provision sites (Docs / News / FAQ), and other functions to be implemented and their details
- Details of key CLI functions
  Functions to be implemented and their details, such as authentication, job operations, issue management reference, and file system access
- Details of Key API Functions
  Specification details, including API functions and response formats
  Data to be collected, including job operations, issue management, usage statistics, and monitoring metrics

3.6.6.2 Application Environment

" FugakuNEXT" aims to balance maintaining compatibility with existing applications and supporting new use cases so that applications across diverse fields can achieve maximum performance and efficiency. While inheriting the numerous high-performance application assets developed for Fugaku, it will realize a computing environment where any application brought in by users can run smoothly.

In the "FugakuNEXT" application environment, it is crucial to balance stable operation with user flexibility. In particular, ISV applications have complex dependency environments and licensing structures, and vendor support is essential for operational guarantees. Therefore, the system will allow users to select applications officially supported by FUJITSU-MONAKA-X and run them within containers.Furthermore, to minimize management overhead and operational risks, the default policy is to allow users to bring their own licenses; central management will be applied only when licenses are treated as resources under workload management. This ensures flexibility in user-specific license management and external integration. On the other hand, since applications and workflows other than ISV applications are highly diverse and difficult to manage uniformly, their deployment and management will be the responsibility of the user, thereby maintaining both system stability and user flexibility.

For the reasons stated above, the following items are determined as part of the basic design.

- For ISV applications, users will select those supported by FUJITSU-MONAKA-X and make them available for use within containers.
- The system will not provide ISV licenses; users must bring their own as needed.
- The Center will manage user licenses only when ISV licenses are used as resources under workload management; in all other cases, users will manage them themselves.
- The deployment and management of non-ISV applications (including workflows) shall be carried out under the user's responsibility.

On the other hand, the following matters will be left to the detailed design phase.

- Consideration of the method for selecting specific ISV application versions available to users
- Consideration of each ISV vendor's support policy for FUJITSU-MONAKA-X
- Examination of methods for users to bring their own licenses to be managed by the Center
  Types of license servers, access management for license servers, and measures to prevent license misuse by other users
- Examination of methods for user-managed license administration
  How to set up a license server within a tenant, and the network requirements for accessing external license servers

3.6.6.3 Package and Module Management

Package and module management is an essential mechanism in HPC environments for systematically managing diverse software, libraries, and dependencies, ensuring that users have a stable and highly reproducible execution environment. In HPC, there are numerous combinations of compilers, MPI, numerical computation libraries, and other components, and traditional standard Linux package management often cannot handle this complexity.Since version coexistence, dependency resolution, and ensuring environmental reproducibility are directly linked to the reliability of research results, a dual approach combining package management and module management with a systematic management infrastructure is necessary. Flexible build management, such as Spack, and simple environment switching via Environment Modules reduce the burden on users and significantly improve the efficiency of research activities.

For "FugakuNEXT," we will build upon the Spack operational expertise cultivated on "Fugaku" while incorporating the latest technological trends in software package management software within the HPC field to explore foundational technologies that maximize the performance and software management efficiency of "FugakuNEXT."

3.6.6.3.1 Features

The features of Spack and Environment Modules are as follows.

Features of Spack

- It automatically resolves diverse versions and dependencies, making complex environment setup easier
- It allows for advanced customization, enabling the creation of environments tailored to specific purposes, such as optimized builds
- Because the learning curve is steep, it can be difficult for beginners to get started

Features of Environment Modules

- Easy to set up and use, making it accessible even to HPC beginners
- Simply loading the pre-provided modules ensures a safe and reliable environment
- Since version and dependency consistency is guaranteed by the module designers, users have limited flexibility

3.6.6.3.2 Items to be decided in this section and items to be left to the detailed design

Based on the above, it is assumed in the basic design phase that users will be provided with two software package management functions: Spack and Environment Module.

On the other hand, the following matters will be left to the detailed design phase.

- Spack (public/private) operation mode
- Design of Environment Modules / Lmod
- Integration with the container environment
- Operation and maintenance design

3.6.6.4 Pre/Post Environment

Pre/Post is a user service that systematically supports the pre-processing and post-processing tasks necessary for users to execute computational jobs smoothly. It standardizes essential tasks beyond the computation itself—such as data formatting, partitioning, and input preparation during pre-processing, and result extraction, aggregation, preliminary analysis, and visualization preparation during post-processing—thereby significantly reducing the user's workload. It is a critical service for achieving the effective utilization of computational resources and improving user productivity.

Regarding the execution environment for Pre/Post, two scenarios are possible: using compute nodes or providing dedicated nodes. Pre/Post processing is an essential element for maximizing the value of simulation results, and it is particularly crucial to be able to handle the increasing scale and complexity of analyses expected in the "FugakuNEXT" era.

- Addressing Scaling
  As analyses become larger in scale, the volume of data is expected to increase even further compared to the Fugaku era. In simulation environments generating petabyte-scale or even exabyte-scale data, I/O and network transfers become significant bottlenecks in post-processing workflows. To resolve this issue, it is necessary to execute post-processing at the point where the data is generated—that is, on the compute nodes themselves.
- Addressing Increasing Complexity of Analysis Subjects
  In pre-processing, advanced support through inference is required to determine combinations of input data and execution parameters for analysis; in post-processing, it is required to extract features from multivariate simulation results. For example, in complex models that consider a wide range of physical phenomena and chemical reactions, finding appropriate parameter settings is difficult, and support based on advanced inference capabilities using AI and machine learning is considered indispensable. Computation nodes are the optimal environment for performing such inference.

For the reasons stated above, the basic design of the "FugakuNEXT" pre- and post-processing environment assumes the use of compute nodes. However, the following matters will be left to the detailed design phase.

- Organization of specific use cases for pre- and post-processing to be executed on "FugakuNEXT"
- Investigation of the system software required on compute nodes to implement the identified use cases
- Investigation of pre- and post-processing software (ISV or OSS) capable of implementing the identified use cases
- Optimizing data processing efficiency by utilizing the computational performance and parallel I/O of compute nodes for pre- and post-processing
- Reducing storage load by efficiently performing feature extraction and image generation through co-processing between solvers and post-processing, such as in-situ and in-transit processing

3.6.6.5 Container Registry

FugakuNEXT will introduce a proprietary web container registry service to enable the efficient and secure operation of diverse projects and jobs utilizing containers.The primary objectives of establishing a proprietary container registry are to distribute authenticated and signed containers, maintain data confidentiality, reduce traffic caused by external communications, and ensure operational consistency and account standardization across the entire "FugakuNEXT" system. This will provide a secure and high-speed image distribution environment without relying on external services, thereby enhancing service reliability.

3.6.6.5.1 Potential OSS Options

While the software used to implement this new container registry service is not necessarily limited to OSS, the following are the main open-source software (OSS) options currently available:

- Docker Registry (Distribution)
- Harbor
- Project Quay
- Zot

3.6.6.5.2 Decisions to Be Made in This Section

The items to be decided in this section as part of the basic design are as follows. For the following reasons, we have selected Harbor and Zot as candidates for the basic design phase.

- Harbor
  As the most widely adopted solution for enterprise use, it offers a comprehensive set of features required for a secure container registry, enabling the construction and operation of services at a low cost.
- Zot
  As it is lightweight and OCIv2-compatible, it is expected to grow as container registry software in the future, and we believe it will enable the construction of a modern container registry service starting in 2030, when "FugakuNEXT" becomes operational.

3.6.6.5.3 Considerations for the Detailed Design

The following lists the considerations for the detailed design when establishing a container registry service.

- Scale of Use
  Estimate the expected number of images, image sizes, and push/pull frequencies.
- Authentication and Security
  Consider security measures at the service level, such as authentication methods, container image signing, and vulnerability scanning.
- Infrastructure Configuration
- Consider the configuration of infrastructure components—such as servers, storage, networking, and databases—to align with the service usage scale and availability and security requirements.
- Monitoring
- In addition to monitoring common infrastructure components such as storage and networking, we will evaluate mechanisms to monitor metrics specific to the registry service, such as registry status, the number of images, and the number of pushes and pulls.
- Availability
- We will evaluate service availability metrics and target values.

3.6.6.6 VPN Connection Service

For “FugakuNEXT,” it is essential that researchers be able to access the system safely and efficiently from diverse user environments both on and off campus, as well as domestically and internationally. In particular, as data-driven research, cloud integration, and integrated workflows with external facilities increase, a connection method based on dedicated network paths (such as L2/L3 VPNs) linking external environments to “FugakuNEXT” is required.The basic design of "FugakuNEXT" assumes the provision of VPN connection services between external environments and "FugakuNEXT." Specific verification of VPN types (L2/L3, encryption methods), bandwidth, and availability design will be conducted during the detailed design phase and beyond.

3.6.6.7 Use of External Services

With the advancement of AI and data-driven research as well as complex workflows, integration with external cloud services and APIs has become a critically important mode of utilization.“FugakuNEXT” must meet diverse research requirements—such as improving researcher productivity, streamlining analysis, and integrating with external data platforms—by enabling secure and flexible integration with these external services. The basic design of “FugakuNEXT” assumes that the use of external cloud services, external APIs, and external computing services will be permitted.

The following outlines the items to be considered starting from the detailed design phase.

- Specific connection methods and network configurations for external clouds, APIs, and computing services
- Details regarding connection destination control (e.g., outbound allowlist)
- Security requirements and audit log management when using external services
- Data transfer methods, bandwidth control, and QoS

3.6.6.8 Service Catalog

The Service Catalog is a document that systematically compiles the various services provided by "FugakuNEXT," clearly outlining the features available to users, performance targets, limitations, security policies, network requirements, API integration, cost allocation, and lifecycle management. This enables users to accurately understand the scope of services, terms of use, and quality levels, while allowing the operations team to provide consistent services to all users.It also ensures service transparency and fairness, contributing to a

reduction in inquiries and the prevention of issues. By organizing the increasingly complex and diverse services of "FugakuNEXT," it serves as an indispensable foundation for simultaneously improving the user experience and enhancing operational efficiency. For the reasons stated above, "FugakuNEXT" treats the development of a service catalog as a prerequisite during the basic design phase.

However, the following items will be addressed during the detailed design phase:

- Service Overview (e.g., Name, Purpose, Target Users, Value Provided)
- Service Details (e.g., provided functions, service components, usage scenarios, dependent services)
- Terms of Use and Service Levels (e.g., availability, performance, availability, capacity, restrictions)
- Usage Procedures and Support (e.g., application method, approval workflow, contact point, backup policy)

3.6.6.9 User Documentation

User documentation is a collection of materials designed to help users learn how to use "FugakuNEXT" in a step-by-step and systematic manner. It covers a wide range of topics, including quick starts, user guides, data transfer procedures, programming environments, GPU migration, container operations, visualization, and API usage.These serve as practical support materials to facilitate a smooth start with the advanced HPC+AI platform, and their significance lies in enabling users to resolve issues independently. This is expected to reduce the support burden, accelerate research work, and improve the user experience , making them an essential element for the effective utilization of "FugakuNEXT."

3.6.6.9.1 Document Delivery Formats

The following lists the media formats envisaged for document delivery.

- PDF Format
  Overview: A format in which documents are distributed as a single completed booklet (or multiple PDFs).
  Advantages: Can be managed as a static, stable version. Suitable for offline viewing and printing.
  Disadvantages: A revised version must be created for each update. Searchability and information freshness are limited. Viewability depends on the device.
- Web (HTML) Delivery
  Overview: A format where content is provided as a hierarchical structure of chapters and pages on a website.
  Advantages: Updates can be made immediately and in parts. Excellent searchability and navigation. Easy integration with web portals. Access analytics are available.
  Disadvantages: Requires maintenance of the web system. Requires an online connection. Layout is constrained.

The following outlines a potential structure for document systematization.

- Volume-Based Structure
  Overview: A method of consolidating all information into a single large document (file or web page).
  Advantages: Partial updates are easy. Provides entry points tailored to user needs. Document structure can be flexibly expanded.
  Disadvantages: Information becomes scattered. Redundancy and duplicate descriptions occur. Not suitable for grasping the big picture.
- Aggregated Structure
  Overview: A method of breaking down information into multiple sections (pages) and providing them categorized by purpose or target audience.

Advantages: The big picture is easy to grasp. It is easy to maintain consistency in the content. Reference relationships are simple.
Disadvantages: Requires a full update. Users may have difficulty locating specific sections. Difficult to expand the structure.

3.6.6.9.2 Items to be Determined in This Section

The items to be determined in this section as part of the basic design are as follows.

- For "FugakuNEXT," user documentation will be created and provided.
- To ensure readability, the documentation will be provided via the Web in a consolidated format without fragmentation.

3.6.6.9.3 Items to be considered from the detailed design phase onward

From the detailed design phase onward, the following items will be considered.

- Timing of documentation release (e.g., whether to release the programming guide early)
- Document structure (Quick Start, Programming Guide, etc.)
- Usage policy for training (e.g., tutorials, pre-implementation training)

3.6.6.10 Multilingual Support

This section defines the basic design policy for multilingual support in “FugakuNEXT.” The service interfaces of “FugakuNEXT” (websites, systems, documentation, etc.) and technical support will officially support Japanese and English. However, the basic design assumes that English will be selected if only one language is to be chosen.This policy is based on the expectation that, while Japan will lead development and operation, the system will be used by researchers worldwide. This will provide an environment where the primary user base can smoothly utilize "FugakuNEXT" functions and information without encountering language barriers.

3.6.7 Operational Support System

The operational support systems for “FugakuNEXT” are a suite of systems that provide centralized and efficient support for a wide range of activities, from user procedures related to HPC system usage (such as applications, information retrieval, and inquiries) to system administration by operational managers (including user, project, and resource management, billing, and usage statistics).

"FugakuNEXT" will implement an operational support system with the aim of improving user convenience and streamlining system operations.

3.6.7.1 Design Policy Based on Requirements

This section describes the design policies for the basic design phase, aimed at addressing challenges inherited from "Fugaku" and requirements looking toward 2030. By implementing the design based on these policies, we aim to create a system capable of flexibly adapting to future institutional changes.

- Automation and Labor Savings
  To ensure stable 24/7 operation and reduce operational workload, we will automate as much as possible, ranging from user applications to account management and simple inquiry handling. AI will be utilized for this automation.
- Improving User Experience
- To eliminate the need to navigate multiple cumbersome systems, we will design an intuitive and user-friendly UI/UX for all system operations, including applications, information retrieval, and inquiries.
- Modularity and Loose Coupling
  We will explore architectural and functional designs that minimize tool modification costs in response to changes in operational systems or usage policies, and enable rapid and flexible responses by operational staff alone even after launch. We will establish a framework assuming that the actual development and programming will be performed by AI.
- High Reliability, Availability, and Scalability
  Given that “FugakuNEXT” will serve as mission-critical social infrastructure, the operational support system itself will be designed with redundancy as a prerequisite to ensure high reliability and availability. Furthermore, we will ensure high scalability to accommodate future increases in the number of users and types of usage.
- Seamless Integration with HPCI
  Building on the current integration with HPCI on "Fugaku," the system must seamlessly integrate with the unified HPCI account and function as an integral part of the HPCI system.

3.6.7.2Development Items

This section describes the items to be reviewed and decided during the detailed design phase.

3.6.7.2.1 Functional Requirements

The following functional requirements are defined with the aim of improving user convenience and streamlining system operations.

The user management function handles account issuance, management, and authentication, as well as permission granting, enabling account registration and usage period extensions. The issue management function streamlines resource allocation and access control. The resource usage management function ensures the fair allocation of HPC resources and visualizes usage status. The billing system automatically calculates billing data and generates invoices, improving the transparency and management of usage fees.The usage statistics function

visualizes system usage through data collection and analysis from the operational data platform, contributing to operational improvements. The ticket system integrates inquiry handling, FAQs, and chatbots to expedite user support. The HPCI integration function enables seamless integration with the existing HPCI ecosystem by synchronizing and linking HPCI accounts and usage plan/performance data.

3.6.7.2.1.1 User Management Features

Manages account issuance and administration, authentication, and the assignment and management of roles (permissions). The main features are as follows.

- Account Registration
  Registers new user accounts in the system based on user applications.
- Account Update
  Modifies registered account information (such as affiliation and contact details).
- Account Suspension/Reactivation
  Temporarily suspend or deactivate an account.
- Account Deletion
  Completely delete account information that is no longer needed from the system.
- Extend Account Usage Period
  Extend the usage period for accounts whose term is about to expire.

3.6.7.2.1.2 Project Management Features

Create groups for each task, add or remove members, and designate representatives and administrators. Allocate resources and manage access controls for each task. The main features are as follows.

- Project Registration
  Register a new research project in the system.
- Project Modification
  Modify information for registered projects (such as team members and allocated resources).
- Project Extension
  Extend the project period and update related information.
- Delete Project
  Delete information on completed projects from the system.

3.6.7.2.1.3 Resource Usage Management

Collect usage data for HPC resources such as CPU/GPU time, memory, and storage to ensure fair allocation. The collected data is integrated with the billing system. The scope of resources to be managed will be determined during the detailed design phase. The main functions are as follows.

- Resource Allocation
  Allocates resource quantities to tasks.
- Resource Request/Approval Workflow
  A series of processes in which the project leader requests resource expansion and the administrator reviews and approves the request.
- Resource Usage Display
  Visualize and display current resource consumption and remaining quantities.
- Resource Usage Limit Settings
  Sets upper limits on the amount of resources allocated to projects and users to manage consumption.

3.6.7.2.1.4 Billing System

Based on data collected through resource usage management, calculates usage fees at pre-set billing rates and generates billing statements. During the detailed design phase, the resources subject to billing and the billing mechanism are specified. The main functions are as follows.

- Access Control
  Manages available budgets and notifies users when budgets are exceeded.
- Billing Calculation
  Performs billing calculations based on the amount of resources used, points, and other factors.
- Billing Details/Invoice Generation
  Displays usage details and invoices for each user and group on the web.
- Billing Rate Settings
  Integration features to support various operational strategies, such as dedicated usage.
- Electricity Integration
  Integration features to implement systems such as "Fugaku Points."

3.6.7.2.1.5 Usage Statistics

Integrates with the operational data platform to analyze data such as system utilization rates and job statistics, and visualizes it on a dashboard.

- Usage Statistics Analysis
  Analyzes data such as overall system utilization, number of job executions, resource usage by user or group, and job performance.
- Statistical Data Analysis/Visualization
  Analyzes collected data, displays it as graphs on the dashboard, and generates reports.

3.6.7.2.1.6 Ticket System

Centralize user support, from receiving inquiries and incident reports to FAQs, automated responses via chatbots, and ticket management.

- Inquiry/Incident Reporting
  Centrally accept user inquiries, incident reports, and requests via the web
- FAQ/Knowledge Base
  Systematically compile and publish frequently asked questions (FAQs) and their answers
- Automated Responses via Chatbot
  Utilizes natural language processing to provide automated responses to simple inquiries and questions related to the FAQ
- Ticket Management
  Ticket status and history management

3.6.7.2.1.7 HPCI Integration Features

To ensure smooth operations under various HPCI programs, we aim to integrate with the broader HPCI ecosystem by seamlessly linking user information and usage data.

- HPCI Account Integration
  User Information Integration

3.6.7.2.2 Non-functional Requirements

To ensure reliability and sustainability, the following non-functional requirements are defined. Performance requirements include improved user interface responsiveness, high processing throughput during peak periods, and the ability to rapidly process statistical data on a daily basis . Availability requirements mandate a high uptime rate, excluding planned maintenance, and require the elimination of single points of failure through redundancy of key components such as web servers, database servers, and application servers. Scalability requirements include the ability to flexibly accommodate increases in the number of users and data volume.Maintenance and operation requirements include log management for operation histories and events, regular backups and rapid recovery, and the ability to apply patches and updates with minimal service downtime. Security requirements involve enhancing system safety and reliability through role-based access control (RBAC) and regular vulnerability assessments.

3.6.7.2.2.1 Performance Requirements

The system must provide rapid response times to user operations, high processing throughput during peak periods, and the ability to quickly process large volumes of statistical data on a daily basis and promptly display analysis results.

- Response Speed
  Response time for general operations via the user interface (login, information retrieval, simple applications, etc.)
- Processing Throughput
  Number of applications processed and inquiries received during peak times
- Data processing volume
  The ability to process large volumes of statistical data on a daily basis and display analysis results promptly.

3.6.7.2.2.2 Availability requirements

High uptime (excluding scheduled maintenance) and elimination of single points of failure through redundancy of key components are required.

- Uptime
  Annual system uptime (excluding scheduled maintenance).
- Redundancy
  Key components (Web servers, DB servers, application servers, etc.) must be duplicated to eliminate single points of failure.

3.6.7.2.2.3 Scalability Requirements

The system must possess scalability to handle increased traffic through resource additions and distributed deployment, as well as the flexibility to accommodate new technologies and features while minimizing the impact on the existing system.

- Scalability
  The architecture must be capable of easily accommodating increases in the number of users, concurrent connections, and data volume through the addition of resources and distributed deployment.
- Flexibility
  The system must be able to accommodate the introduction of new HPC technologies and usage models (e.g., integration with quantum computers, new AI frameworks) and the addition of new features while minimizing the impact on existing systems.

3.6.7.2.2.4 Maintenance and Operation Requirements

Requires log management of operation histories and errors for use in audits and failure analysis, as well as regular backups and recovery to enable rapid restoration in the event of data corruption, and patch application and updates that minimize service downtime.

- Log Management
  All system operation histories, errors, and events must be recorded and made available for auditing and fault analysis. Logs must be retained for an appropriate period.
- Backup/Recovery
  All system data must be backed up regularly and be capable of being restored promptly in the event of data corruption.
- Patch Application/Updates
  The system must have a mechanism in place to apply security patches and software updates while minimizing service downtime.

3.6.7.2.2.5 Security Requirements

System security is ensured through strict access restrictions based on role-based access control and regular vulnerability assessments.

- Access Control
  Implement role-based access control (RBAC) to strictly restrict unauthenticated and unauthorized access.
- Vulnerability Mitigation
  Conduct regular vulnerability assessments.

3.6.7.3 Matters to Be Determined in This Chapter and Matters to Be Left to the Detailed Design

The items to be determined in this section as part of the basic design are as follows.

- Implement an operations support system
- As a response to the requirements for the operations support system, establish a design policy centered on “operational automation and labor savings,” “improved user experience,” “modularity,” “high reliability, availability, and scalability,” and “HPCI integration”

On the other hand, the following matters will be left to the detailed design phase.

- Functional and non-functional requirements for development items
- System overview, system specifications, software selection, and implementation methods
- Operational policies and procedures

3.6.8 Operational Infrastructure

For “FugakuNEXT,” based on the operational track record of the current “Fugaku,” we aim to design an operational infrastructure capable of stably running over 4,000 hardware units and the various software programs operating on that hardware. In the basic design, the software to be considered for the operational infrastructure is classified into four categories: provisioning software, configuration management software, monitoring software, and operational data infrastructure software. This section presents the items to be considered for “FugakuNEXT” and the relevant software.

This section describes the evaluation of the following software required to enable system operations, as well as the considerations for ensuring efficient and stable system operations using this software.

- Provisioning Software
  Performs OS installation and uninstallation on hardware (servers)
- Configuration Management Software
  Performs hardware configuration changes, as well as the installation, uninstallation, and configuration changes of OS, middleware, and applications running on hardware (servers)
- Monitoring Software
  Collects information on hardware and software running on hardware (servers) and detects anomalies
- Operational data platform software
  Collects, stores, and manages information regarding hardware and software running on hardware (servers)

3.6.8.1 Provisioning Software

This section examines provisioning software. Provisioning refers to the process of installing an OS on a physical server to make it operational; software that automates and streamlines this process is called provisioning software.

Since "FugakuNEXT" requires the rapid construction of large-scale nodes, the automation and streamlining of server provisioning tasks are essential. In the current "Fugaku" operation, provisioning tasks are performed using the Fujitsu Software Technical Computing Suite; however, to address the challenges in the "Fugaku" system mentioned at the beginning, it is necessary to select alternative software.

In the basic design phase, based on the need for integration with various infrastructure components (such as DHCP and TFTP) during OS provisioning, as well as a track record of supporting large-scale environments, two provisioning software options—Foreman and OpenCHAMI—have been selected. A detailed evaluation of these software options will be conducted during the detailed design phase.

The reasons for selecting the software and the items to be considered in the detailed design are outlined below.

- Reasons for Selecting the Software
  Based on the results of the provisioning software comparison, Confluent and Warewulf cannot integrate with infrastructure components such as DHCP and DNS during OS provisioning. Consequently, configuration changes to these infrastructure components must be made separately before and after provisioning. This not only complicates operations when performing provisioning in parallel in large-scale environments but also makes configuration inconsistencies more likely to occur.On the other hand, while Foreman and OpenCHAMI do not raise concerns regarding their feature sets, Foreman offers greater flexibility in system configuration because it can perform provisioning for compute nodes distributed across multiple subnets entirely within a single subnet.Furthermore, from a scalability perspective, Foreman has a superior track record of operation in large-scale environments. Additionally, Foreman includes mechanisms for

functional expansion via plugins, demonstrating its consideration for extensibility. Although the differences described above exist between Foreman and OpenCHAMI, there are no issues regarding their core functionalities; therefore, Foreman and OpenCHAMI will be adopted as the provisioning software in the basic design.

- Items for Consideration in the Detailed Design
  In the detailed design phase, we will investigate and evaluate Foreman and OpenCHAMI, including the impact of functional differences on actual operations. The following outlines the items to be considered in the detailed design.
  - Determination of the Provisioning Software to Be Adopted
  - Clarification of the operational requirements for the "FugakuNEXT" system and verification of the selected software's functional and non-functional compliance with those requirements

3.6.8.2 Configuration Management Software

This section discusses configuration management software. Configuration management refers to the process of performing various operations (configuration management) on the hardware and software necessary for system operation using the interfaces and functions provided by the hardware and software. Software that uniformly manages, automates, and streamlines these operations is called configuration management software. Furthermore, within configuration management software, those limited to operations on hardware are called hardware configuration management, and those limited to operations on software are called software configuration management.

For "FugakuNEXT," system configurations are expected to become increasingly complex, as the scope of use is anticipated to expand beyond conventional HPC processing to include AI processing and other applications. Furthermore, the management of this software is expected to rely not only on conventional packages and containers (formerly Singularity) but also on new platforms such as Kubernetes. In this increasingly complex system, selecting configuration management software is necessary to streamline various operations on hardware and software, as well as system configuration and configuration changes.

In the basic design phase, we will adopt Ansible and AWX as configuration management software, taking into account the types of hardware and software subject to configuration management, the ease of defining configuration management, and the management of execution history. However, since there are concerns regarding AWX due to large-scale community-led revisions underway since 2024, we will continue to closely monitor the status of community activities. A detailed review of these selected software solutions will be conducted during the detailed design phase.

The reasons for selecting the software and the items to be considered in the detailed design are listed below.

- Reasons for Software Adoption
  Based on the results of the configuration management software comparison, there are concerns regarding Chef and Puppet because they use proprietary DSLs for describing system configuration definitions, which requires a learning curve. On the other hand, Ansible and Salt use YAML, which lowers the barrier to entry for describing system configuration definitions, and there are no particular concerns regarding a comparison of their functional features.There are differences between Ansible and Salt regarding the development of GUIs and the management of execution history. A key feature of Ansible is that it allows for configuration management via a GUI and the management of execution history through AWX, which is officially developed by the community. Furthermore, Ansible excels in supporting the operations expected in configuration management. Therefore, we have decided to adopt Ansible as the configuration management software.
- Items for Consideration in the Detailed Design

In the detailed design phase, we will primarily focus on Ansible to investigate and evaluate its ability to handle specific configuration management items and operations. The following outlines the items to be considered in the detailed design.

- Organization of configuration management items and their targets required for system operation following the determination of the hardware and software comprising the "FugakuNEXT" system, and determination of configuration management implementation methods
- Clarification of the operational requirements for the "FugakuNEXT" system and verification of the functional and non-functional adequacy of the selected software against these requirements

3.6.8.3 Monitoring Software

This section discusses monitoring software. Monitoring software is software that monitors the state of the system based on operational information (system data) regarding the system and the software running on it, and that detects anomalies within the system and visualizes system operational status.

With the expansion of use cases for "FugakuNEXT," the types of hardware and software required to perform these tasks are diversifying, and the system configuration is becoming increasingly complex. In such an environment, to enable early response to system failures and ensure stable system operation, it is necessary to select monitoring software that can "monitor hardware and software, detect and notify of anomalies, and track operational status."

The monitoring software consists of a "system data collection and storage function" for collecting and storing system data, and a "monitoring function" for enabling system monitoring and observability.

- System Data Collection and Storage Function
  A function that collects, stores, and manages information on the hardware and software within the system (system data). It consists of a "system data generation and acquisition unit" that acquires and generates information, and a "system data storage and management unit" that stores and manages the collected data.
- System Monitoring
  Based on the collected system data, this function detects the occurrence of normal or abnormal events within the system and, upon detection, notifies external parties via email or other means.
- System Observability
  Enables easy "monitoring of system operational status" and "monitoring of the behavior of each component comprising the system" using collected system data.

In the basic design phase, we will evaluate the "software implementing the System Data Generation and Acquisition Unit," "software implementing the System Data Storage and Management Unit," "software implementing System Monitoring," and "software implementing System Observability," and investigate the software listed below. A detailed evaluation of this software will be conducted during the detailed design phase.

Functions to be implemented by monitoring software	Software Implementing the Function	
System data collection and storage function	System Data Generation and Acquisition Module	OpenTelemetry Fluent Bit Prometheus Exporter Beats Elastic Agent
	System Data Storage and Management Department	Prometheus Elasticsearch OpenSearch
Monitoring Features	System Monitoring	OpenSearch Kibana Alertmanager
	System Observability	OpenSearch Kibana Grafana

3.6.8.4 Operational Data Platform Software

The operational data platform is a foundation that enables long-term improvements by accumulating and analyzing user data—such as operational information (system data) regarding the systems built on "FugakuNEXT" and the software running on them, as well as job statistics. Specific management targets and software considerations within the operational data platform will be addressed during the detailed design phase.

3.6.9 Security

3.6.9.1 Security Policy

"FugakuNEXT" is a large-scale shared computing infrastructure used by a diverse range of researchers both in Japan and abroad, and ensuring safety, availability, and reliability is a top-priority design requirement. In next-generation HPC, where computing nodes, storage, networks, and container platforms are closely integrated, a multi-layered security approach—combining network segmentation, tenant isolation, access control, auditing and logging, and operational audits—is essential, rather than relying on a single defense mechanism.This section outlines the "segmentation architecture," "network design," "tenant model," "audit framework," and "log management" that form the foundation of "FugakuNEXT," and presents the fundamental approach for balancing security and flexibility.

3.6.9.1.1 Considerations

Since security policies affect the overall structure of "FugakuNEXT," they must be examined from multiple perspectives, taking into account foundational technologies, external case studies, international standards, and operational requirements.

- Network Architecture Segmentation
  In "FugakuNEXT," considering that a large number of users will access the system simultaneously, network separation must be established as a fundamental requirement. Since mixing management networks with user data communications increases security risks, it has been confirmed that a design strictly separating both physical and logical layers is effective. Furthermore, by minimizing communication paths to the outside and implementing appropriate control and monitoring only when necessary, the system's defense against external threats can be enhanced.
- Tenant Isolation Method
  Since different users and workloads are executed for each project, it is necessary to provide an execution environment where they do not interfere with one another. Tenant isolation utilizing a container-based infrastructure is an effective method that can simultaneously isolate computing and network resources, thereby balancing flexibility and security. Furthermore, by encapsulating tenants—including user environments—using a VPN, it becomes possible to provide a secure execution environment.
- Investigation of Storage Access Isolation Methods
  The storage architecture must align with the tenant isolation structure. By securely mounting user home and project directories within the tenant environment, it is possible to ensure security by controlling accessible paths and making other project areas invisible.
- Audit and Log Design
  For secure operations and incident response, a mechanism to visualize activities across all layers and comprehensively collect and store necessary logs is essential. In particular, logs related to authentication and authorization, job execution, network communications, and system operations are necessary for situational awareness and investigative analysis. Furthermore, operational design that considers long-term storage and tamper prevention is also important.
- Development of a Security White Paper
  A white paper documenting the security architecture is necessary to explain the safety of "FugakuNEXT" to users and external organizations with a high degree of transparency. This not only ensures user confidence but also serves as proof of reliability in external collaborations.

3.6.9.1.2 Items to Be Determined in This Section and Items to Be Left to Detailed Design

The items to be decided in this section as part of the basic design are as follows.

- Tenants will be constructed using containers to provide a closed network space for each project.
- Storage will be implemented by mounting user directories into containers.
- A security audit will be conducted at the start of operations.
- The policy will be to collect and retain necessary audit logs.
- A security white paper will be prepared.

However, the following items will be addressed during the detailed design phase.

- Tenant implementation method (network policies, CNI configuration, service mesh, etc.)
- Details of the storage mounting method (access rights, encryption, path control)
- Frequency, scope, and implementation structure of security audits
- Detailed design of log types, retention periods, and access rights
- White paper items and update policy

3.6.9.2 Authentication and Authorization

Since "FugakuNEXT" requires the integrated management of diverse access paths—such as web portals, SSH, APIs, and workflow platforms—consistency in authentication methods and authorization frameworks is critical.Furthermore, as this encompasses a wide range of issues—including account management, access control, multi-factor authentication, and external integration—we will organize information regarding single sign-on (SSO), authorization models (such as RBAC), and the latest authentication methods. These are essential not only for improving the user experience but also for balancing security and operational efficiency.

"FugakuNEXT" must resolve the challenges associated with the multiple authentication infrastructures of the previous Fugaku system and integrate them into a single, state-of-the-art authentication infrastructure to prevent permission inconsistencies, streamline management, and reduce the burden on users.

For the reasons stated above, we will assume in the basic design phase that we will implement unified authentication using the latest authentication methods, including SSH and web, and build a Single Sign-On (SSO) environment. Details regarding the authentication and authorization technologies used for SSO, as well as the unified authentication methods for SSH and web, will be addressed during the detailed design phase and beyond.

3.6.9.3 Security Incident Response

“FugakuNEXT” is a large-scale computing infrastructure that accommodates a large number of users and diverse workloads; incidents such as system failures, attacks, and operational errors have the potential to lead to service outages, loss of research data, and the spread of impact to external parties.In particular, in an environment where computing resources, storage, networks, and external connections are closely interlinked, delays in decision-making or response during an incident could rapidly expand the scope of damage. Therefore, it is necessary to establish a systematic incident response framework from the perspectives of incident types, initial response, organizational roles, evidence management, recovery, and improvement, and to clarify the response structure required for “FugakuNEXT.”

For the reasons stated above, the formulation of incident response procedures to enable rapid recovery and response in the event of an incident is established as a prerequisite during the basic design phase. However, specific details regarding the items to be included in the response procedures, the organizational structure (including division of roles, chain of command, and external reporting flows), as well as the training framework, education programs, and improvement cycles, will be addressed during the detailed design phase and beyond.

3.6.9.4 Encryption

Since “FugakuNEXT” will handle diverse research data, personal information, and highly confidential deliverables, and given that usage patterns will become more sophisticated—including external connections and cloud integration—the implementation of encryption for both transmitted and stored data is essential for ensuring security.Furthermore, encryption is not a single element; it requires an integrated approach covering communication paths, stored data, and key management. The selection of specific methods and determination of their scope of application must be organized with due consideration for security levels, performance impacts, and operational overhead.

For “FugakuNEXT,” it is essential to incorporate encryption into the overall architecture to mitigate various risks, including eavesdropping, tampering, and unauthorized access by attackers, as well as operational errors and improper handling from within the organization. Particularly in large-scale systems where compute nodes, networks, and storage are intricately interconnected, it is crucial to clearly identify where encryption should be applied and to establish a consistent operational model that includes key management methods .

For the reasons stated above, “FugakuNEXT” assumes that encryption will be implemented where necessary as a prerequisite of the basic design phase. Decisions regarding which areas to encrypt, which methods to adopt, and how to manage keys will be addressed during the detailed design phase and beyond, taking into account performance requirements, operational constraints, and usage patterns.

3.6.10 Operational Processes

The operation of “FugakuNEXT” must stably meet a wide range of requirements, including the efficient utilization of computational resources, maintenance of service quality, rapid recovery from failures, and ensuring security. In particular, as requirements such as the increase in AI and data-driven workloads, integration with external platforms, and the enhancement of high availability and observability grow, operational processes form the foundation for ensuring the system’s reliability and sustainability.Therefore, it is decided in the basic design that "FugakuNEXT" will establish systematic operational processes in accordance with standards such as ITIL and ITSMOP. This will prevent operations from becoming dependent on specific individuals and support the establishment of an operational framework capable of continuous improvement. In practice, the system will be structured with the premise of operational automation utilizing AI.

The items to be examined in the detailed design are as follows.

- Operational Framework
  We will examine how to specify the scope of responsibility, authority, and escalation procedures within the operational framework and integrate them into daily operations. We will clarify the division of roles, authority, and areas of responsibility within the operational organization and define escalation rules.
- Maintenance Management
  We will examine how to standardize criteria for planned and emergency shutdowns, workflows, impact assessments, and notification methods. We will establish procedures for scheduled and unscheduled maintenance, notification methods, and job scheduling methods to minimize impact.
- Incident Management
  Examine how to define incident detection methods, response processes, priority classifications, communication flows, and recovery criteria. Standardize the entire process from fault detection through recovery and recurrence prevention to improve the identification of the scope of impact and response speed.
- Problem Management
  Examine how to incorporate root cause analysis methods, problem registration criteria, and processes for implementing permanent corrective actions. Conduct systematic root cause analysis of incidents and establish a mechanism to continuously incorporate configuration improvements and corrective measures.
- Change Management
  We will examine how to standardize each stage of the change management process: change requests, impact analysis, review and approval, implementation, verification, and rollback. We will execute changes such as OS updates, configuration changes, and infrastructure updates in a planned manner, and establish processes for pre-review, impact analysis, and approval.
- Service Catalog Management
  Determine how to define service items, terms of service, restrictions, and SLOs, and how to present them to users. Organize service details, SLOs, and constraints for users, and clarify the scope of service provision.
- Development of SOPs (Standard Operating Procedures)
  Examine the level of detail in operational manuals, update procedures, and methods for automatically updating procedures through CI/CD integration. Standardize operational procedures to ensure reproducibility and operational quality. Also consider update management integrated with CI/CD.
- Verification Procedures and Release Management
  Examine how to systematize the configuration of the verification environment, test items, and release criteria. Establish the infrastructure necessary to maintain release quality,

including the preparation of the verification environment prior to production deployment and configuration change management.

- Document Management
  Examine how to design the scope of documents to be created, update procedures, version control methods, storage locations, and access permissions. Establish a document system comprising configuration diagrams, procedure manuals, operational logs, and change histories, and build a mechanism to update these documents in line with changes.
- SLA/SLO Management
  Determine how to establish monitoring metrics, data collection methods, report formats, threshold settings, and improvement processes. Define service quality metrics (availability, response time, performance, etc.) and formulate SLO monitoring and reporting procedures.
- Data Protection, BCP, and DR
  Examine how to define backup configurations, retention periods, recovery procedures, and DR site operation methods. Organize backup policies, data retention policies, and DR procedures to ensure business continuity during disasters or major outages.
- Training and Knowledge Management
  Examine the structure of the training framework, training implementation plans, and methods for accumulating knowledge and improving searchability. Formulate a training framework for operators, establish a knowledge base, and define methods for the continuous accumulation of operational knowledge.
- Audit and Compliance
  Examine how to design the scope of audit logs, retention periods, access controls, and audit response processes. Establish a management framework for log management, evidence retention, and change history management capable of supporting internal and external audits.
- Lifecycle Management
  Examine how to systematize the records, approvals, and work processes required at each stage of implementation, update, and decommissioning. Organize configuration and service management processes covering the entire lifecycle from system implementation to decommissioning.
- Operational Evaluation and Improvement (Continual Improvement)
  Examine evaluation metrics, review cycles, methods for incorporating improvement proposals, and continuous management approaches for process improvement. Incorporate periodic evaluations of KPIs/SLOs and process improvement loops into operations to achieve continuous optimization.

3.6.10.1 System Implementation

A system implementation plan is essential for ensuring the smooth and safe deployment of "FugakuNEXT."It is necessary to organize the processes, roles, and risks that stakeholders must share—from hardware delivery through construction and testing, user migration support, parallel operation, and long-term maintenance—and to clarify the implementation sequence and responsibilities from the perspective of overall optimization. In particular, because the process involves complex elements such as verifying compatibility with Fugaku, data migration methods, and establishing operational frameworks, early formulation of the implementation plan will greatly contribute to preventing migration delays and ensuring stable system operation.
For the reasons stated above, "FugakuNEXT" will formulate a comprehensive system implementation plan that includes not only the system itself but also facility infrastructure. Detailed implementation plans (processes, organizational structure, building coordination, user migration plans, etc.) will be formulated after the detailed design phase. The following outlines the items to be considered after the detailed design phase.

- Building Construction and Facility Infrastructure Coordination

- Power Supply and HVAC Construction Plans
- Cabling Plan
- Synchronization of Construction Schedule and System Delivery Schedule
- System Construction and Implementation Schedule
- Equipment Delivery Plan
- System Construction Plan
- Performance and Operational Testing Plan
- User Migration Support Plan
- Support for Application Porting and Performance Verification
- Workflow Migration Guidelines
- Tutorials, Help Desk, and Technical Support System
- Parallel Operation and Switchover Plan
- Definition of the Parallel Operation Period for Fugaku and "FugakuNEXT"
- Details on Data Migration Methods and Mitigating Differences in Scheduler Settings
- Performance Verification and Compatibility Testing During the Parallel Operation Period

3.6.11 Long-Term Operation Plan

A long-term operation plan will be formulated for "FugakuNEXT" to ensure stable, long-term operation and a smooth transition to "FugakuNEXTNEXT." This plan aims to maximize continuity, reliability, and utility throughout the entire lifecycle of "FugakuNEXT."Furthermore, to ensure a smooth transition to the future system, “FugakuNEXTNEXT,” we will incorporate operational policies and migration methods that anticipate the next-generation system from the planning stage.

The details of the long-term operation plan will be examined during the detailed design phase and beyond. The items to be considered are listed below.

3.6.11.1 Guidelines for Formulating the Long-Term Operation Plan

The operation of "FugakuNEXT" must be planned for a long-term period of five years or more, spanning from the start of operation to the upgrade to the next-generation system. Therefore, rather than simply aiming for the stable operation of the current system, a continuous plan based on the premise of transitioning to "FugakuNEXTNEXT" is required.

3.6.11.1.1 Design of the Parallel Operation Period

The transition to "FugakuNEXTNEXT" will require parallel operation spanning multiple fiscal years. For parallel operation, the following elements must be clarified from the planning stage:

- A phased migration schedule organized by task and project
- Data migration methods (high-speed data migration, incremental synchronization, integration with external storage, etc.)
- Application build/optimization support (e.g., differences between GPU generations)
- Strategy for coexistence of WLM across both systems (queue separation, quota management)

3.6.11.1.2 Long-term hardware maintenance and upgrades

Assuming long-term operation, the following items will be examined in advance.

- Maintenance contract duration and renewal timing
- Securing replacement parts and managing end-of-life risks
- Assessment of operational impact during generational upgrades of GPUs, networks, storage, etc.

3.6.11.1.3 Software and Middleware Update Plan

Systematic management is also required for components with high update frequencies in long-term operations:

- Update plans aligned with the OS Long-Term Support (LTS) cycle
- Version update policies for WLM (Slurm/PBS/Flux, etc.) and Kubernetes

End

Appendix A:

1. Application Working Sub-Working Group Report

1.1 SubWG1 (Life Sciences)

1.1.1 Purpose and Structure

Sub-WG1 focused on representative applications in the life sciences field and conducted activities aimed at identifying application characteristics for FugakuNEXT, preparing for benchmark evaluations, and sharing information with the scientific community.

1.1.2 Selection of Applications for Early Evaluation

From the life sciences field, GENESIS (https://mdgenesis.org/) and UT-Heart (https://ut-heart.com/jp/index.html) were selected.

GENESIS is a molecular dynamics (MD) simulation package supporting multiscale (all-atom, coarse-grained, and QM/MM) models, consisting of various functions and analysis tools necessary for MD calculations. This application has held an important position since the development of “K” and has been adopted as a target application for “Fugaku.”Furthermore, while most of this application is written in Fortran, part of the code is already written in CUDA, enabling GPU-based computations.

This application has the following characteristics:

- MD calculations supporting multiscale models
- Hybrid parallelization using MPI and OpenMP
- Optimization for "Fugaku"
- Support for GPGPU computing for some calculations
- The development team includes members with optimization experience on "K" and "Fugaku"

Based on these characteristics, we determined that this is a useful application for evaluating next-generation computing architectures, including CPUs and GPUs, and positioned it as an application for early evaluation.

UT-Heart is a multiscale, multiphysics simulation software for the heart. Like GENESIS, this application has been closely involved since the development of "K" and "Fugaku" and has been optimized for Fugaku. Furthermore, since this application shares similar characteristics with GENESIS, it was selected as an early evaluation application.

1.1.3 Preparation and Submission for Benchmark Evaluation

Since GENESIS is open-source software, we provided the Benchmark WG with information on where to obtain the source code required for benchmark evaluation. We also provided the same information to NVIDIA and Fujitsu for the purpose of detailed performance evaluation by the vendors and consideration of future optimization.

Regarding UT-Heart, we are proceeding with the conclusion of a contract among the three parties: RIKEN, Fujitsu, and NVIDIA.

1.1.4 Sub-WG Meetings and Community Collaboration

We exchanged information with members of the SubWG and application developers in the life sciences field primarily through online meetings, email, and Slack. During SubWG meetings, we held ongoing discussions on the following matters.

- Policy for selecting applications for early evaluation
- Response to requests from the Benchmark WG
- Organization of potential applications from within and outside the field

With application developers in the life sciences field,

- we exchanged information regarding the project’s objectives, requirements, and progress
- Opinions and requests from developers
- Methods of collaboration with the Benchmark WG

. In particular, developers from GENESIS requested the implementation of a wide range of benchmark tests that include not only all-atom models but also coarse-grained models and QM/MM models. They also strongly requested the presentation of the project schedule and milestones.

1.1.5 Response to the Domain Survey

The two field surveys conducted were addressed within the Sub-WG.
Furthermore, to understand the current adoption status of AI applications in the life sciences field and with an eye toward the potential future performance evaluation of AI-powered applications, a list of applications was compiled based on discussions with the Sub-WG and application developers. The list is shown below.

Program Name	Purpose
AlphaFold2	Structure Prediction
RoseTTAFold	Structure Prediction
RFdiffusion	Structure Generation
PhysNet	Machine Learning Potential for MD
Boltzmann Generators	Conformational Sampling
DeepEMhancer	Enhancing Cryo-EM Density Maps
DeepFRET	Automatic analysis of single-molecule FRET data
ESM	Multimodal protein generative model
BioEmu	Sequence -> Approximated equilibrium distribution of structures
DeePMD-kit	Deep learning-based models of interatomic potential energy and force fields
Boltz2	Structure prediction, affinity prediction
ChemTSv2	Molecule design
kMOL	Property prediction
INGOR	Gene network estimation
ABLang2	antibody-specific language model
BindCraft	antibody and peptide design
AFsample2	protein sequence -> generate multiple conformations
Subsampled AF2	protein sequence -> generate multiple conformations
BoltzGen	Binder design
OpenFold3	structure prediction
ChemGLaM	PLM/CLM-based multimodal CPI prediction

1.1.6 Achievements and Remaining Challenges for This Fiscal Year

The selection of GENESIS as the application for early evaluation and the completion of evaluation preparations represent a major achievement of SubWG1. However, GENESIS

supports GPU processing for only a portion of its computations; it is not a fully GPU-resident application and relies on the CPU for the majority of its computations. Therefore, we believe it is necessary to proceed with making GENESIS GPU-resident while also adding other GPU-resident applications for evaluation purposes.Furthermore, regarding the benchmark targets, it is necessary to clarify the policy of conducting evaluations not only on the all-atom model but also on coarse-grained models and QM/MM models.

Regarding UT-Heart, the issue was that the tripartite agreement between RIKEN, Fujitsu, and NVIDIA took time, and preparations for evaluation could not be completed.

1.1.7 Key items for the next fiscal year (detailed design phase)

Regarding GENESIS, we believe it is important to proceed with discussions on how to integrate the application into the benchmark framework for the detailed design phase, while collaborating with the developers. At the same time, we will exchange views with Fujitsu and NVIDIA on application optimization strategies and deepen the discussion. Furthermore, we will investigate whether GPU-resident applications other than GENESIS (e.g., GROMACS, OpenMM) can be used as evaluation applications.

Regarding UT-Heart, we aim to conclude the contract at an early stage and, once concluded, proceed promptly with preparations for the detailed design phase, similar to GENESIS.

1.2 SubWG2 (New Materials and Energy Field)

1.2.1 Objectives and Structure

The primary objective of SubWG2 was to prepare the energy and materials research community for FugakuNEXT. Activities can be summarized as follows.

1. Investigation of application characteristics in the field
2. Setting of benchmark parameters
3. Information Sharing with the Scientific Community

The field of materials science involves an extremely diverse range of computational methods. Therefore, we aimed to identify applications that could leverage the strengths of FugakuNEXT and to gain a broad understanding of the entire field to facilitate co-design activities.

1.2.2 Selection of applications for early evaluation

The following two applications were selected for early evaluation.

1. SALMON – This software calculates electronic dynamics and optical responses resulting from the interaction between light and matter based on time-dependent density functional theory (TDDFT). It was selected based on its track record on Fugaku and the completion of its porting to GPUs. It is a stencil-based code consisting of repeated matrix operations and is a prime example of a computation with high memory bandwidth and latency requirements.
2. mVMC – Numerical simulation software for quantum lattice models using the multivariate variational Monte Carlo method. The primary reasons for its selection were its extensive track record of use in Japan and its accessibility via HPCI. Due to the scalability of the Monte Carlo algorithm and its heavy reliance on dense linear algebra operations, it is well-suited for evaluating the floating-point performance of FugakuNEXT.

1.2.3 Consideration of Alternative Applications

We examined the potential of the pre-installed software set for HPCI resources provided by the Research Organization for Information Science and Technology (https://www.hpci-office.jp/for_users/appli_software) as candidates for alternative applications and analyzed the suitability of each application.

- ABINIT-MP – An excellent candidate, but could not be selected due to a lack of human resources. Its code characteristics are closer to the life sciences field than to the materials field.
- AkaiKKR – Could not be selected for similar reasons.

- HΦ – mVMC was selected instead. Since the two codes are closely related, improvements to mVMC will also benefit HΦ.
- LAMMPS – Could not be selected because it is not a domestically developed code; it is a code that should be considered in the context of the potential for machine learning in materials science.
- NTChem – Has a small user base.
- OpenMX – An excellent alternative candidate, but could not be selected due to a lack of human resources.
- PHASE/0 – Has few users.
- Phonopy – The intrinsic computational cost depends on external applications.
- Quantum ESPRESSO – Not a Japanese-developed code, and performance evaluations by NVIDIA exist.
- SMASH – Currently not under active development.

ABINIT-MP, AkaiKKR, HΦ, and OpenMX are recommended for future consideration.

NVIDIA's application list was also considered as a candidate, but since Quantum ESPRESSO is the only materials-related application and SALMON has a proprietary kernel, it was deemed unsuitable as a proxy.

We also investigated computationally constrained applications, which are common in this field. Methods constrained by dense linear algebra are representative examples.

1.2.4 Computationally Constrained and AI Applications

We also examined applications constrained by computational performance. Such applications are common in this field, and methods constrained by dense linear algebra are particularly representative. However, there are challenges such as the difficulty of achieving peak performance in practical applications and competition with low-order scaling algorithms. Whether an application is constrained by computational performance depends on factors such as the problem definition and numerical representation.Representative examples include ground-state DFT calculations (10.1145/3295500.3357157), excited-state calculations (arXiv:2509.23018), high-precision wavefunction methods (10.1109/SC41406.2024.00015), tensor network methods (10.1021/acs.jctc.4c00903), and exact diagonalization lattice simulations (10.1016/j.cpc.2024.109093). While double-precision arithmetic is required for most of these, single-precision arithmetic is sufficient for wavefunction methods.

AI applications were also examined. The most important use case in the materials field is the Machine Learning Interatomic Potential (MILP). The use of MILPs is expanding, and they can have high computational requirements, such as achieving a large proportion of peak performance on supercomputers (10.1016/j.cpc.2020.107624).There are diverse variations of MILPs based on neural network architectures, chemical environment descriptors, and the presence or absence of additional procedures to enforce symmetry. While MACE is the most widely used, higher-performance force fields are also emerging (https://matbench-discovery.materialsproject.org/). In terms of accuracy, we focus on variations of the same force field that offer quality equivalent to or better than MACE.Examples include eSEN (specifying use for low precision), NequIP (mixed single- and double-precision), SevenNet (single-precision), ORB (single-precision), DPA3 (mixed single- and double-precision), and MACE (double-precision, justified in the paper). Beyond interatomic potentials, AI models for generating new molecules and materials using methods such as diffusion models are noteworthy emerging applications.

1.2.5 Preparation and Submission for Benchmark Evaluation

For both applications targeted for early evaluation, we created input files and scripts for running the benchmarks.

1. For SALMON, although a benchmark from the FS2020 project existed, we determined that it differed from the computational patterns required for future research. Specifically, the existing benchmark relies on large-scale "k-point" sampling, which results in an "embarrassingly

parallel″ structure consisting of numerous small-scale problems. However, this method is applicable only to bulk systems.Instead, we constructed a new benchmark based on increasing the supercell size. This approach is better suited to future scientific problems.

2. For mVMC, we prepared benchmarks based on discussions with the developers. We also prepared scripts to generate inputs of different sizes.

We confirmed that these benchmarks can be run on the Fugaku and Grace Hopper supercomputers. Regarding SALMON, it worked after modifying the build system (Github: be60c90). Furthermore, we shared the SALMON benchmarks with NVIDIA developers and provided comments on the differences compared to the data they collected in FS2020.

1.2.6 Sub-WG Review Meetings and Community Collaboration

SubWG2 held a meeting with the SALMON and mVMC developer communities. During the meeting, we discussed code integration strategies.

SALMON developers agreed to develop on a specific branch and to merge into the main GitHub repository as a general rule. CI will test all merge requests at .The developers decided to share a developer wiki containing coding conventions. Regarding contributions of CUDA code, they agreed to allow them only for deep routines (Hamiltonian application) and only on an optional basis. Furthermore, they requested testing cooperation from the CX framework regarding these changes and new compilers. Additionally, comments were made regarding the historical difficulty of maintaining code from external sources and the fact that CUDA is difficult for developers who are primarily physics researchers.

Due to mVMC's licensing constraints, it is important to formulate an appropriate strategy for NVIDIA integration. Initially, we believed that simply creating a Pfaffian library for GPUs would suffice, but to improve performance by executing multiple computations simultaneously, a corresponding driver is required.The code has been rewritten in Julia, and we are considering the possibility of releasing the Julia version under a more permissive license. Commits to the main development branch are possible, and developers will rebase them onto their respective active branches.

1.2.7 Response to the Field Survey

We shared survey forms with developers involved in the SALMON and mVMC projects. We received responses to both survey forms for SALMON and to the second survey form for mVMC.

1.2.8 Achievements and Remaining Challenges for This Fiscal Year

This year, we collaborated on code design activities in partnership with SALMON and mVMC developers. In addition to the surveys, we conducted specific computational experiments regarding vector width. Furthermore, we conducted a survey of the entire field to identify key applications for FugakuNEXT.

The remaining challenges are as follows.

1. SALMON is a priority target application due to the developers' high level of interest in FugakuNEXT. We must continue coordinating between the developers and NVIDIA/Fujitsu and provide as much support as possible to maximize GPU performance.
2. mVMC presents a challenge due to GPL licensing constraints and the lack of a GPU version, but it is the only EEA capable of truly leveraging FugakuNEXT's floating-point performance. We must actively collaborate with developers and allocate human resources to the development of mini-applications and GPU porting.
3. Since the materials field involves diverse computational methods and software, we should continue engaging with the community to advance code preparation for FugakuNEXT, while also promoting the development of new methods that can leverage the massive computational power provided by FugakuNEXT.

We are investigating the performance characteristics of mVMC on Fugaku using performance tools. After refining and summarizing this data, we need to assess the degree to which peak performance is achieved on FugakuNEXT.

1.2.9 Key Items for the Next Fiscal Year (Detailed Design Phase)

Future efforts are as follows.

1. Evaluate the impact of specific GPU design decisions on the SALMON and mVMC codes. Build an actual performance model and experimental platform.
2. Support the porting of mVMC to GPUs.
3. Tune the SALMON code to understand its performance sensitivity to specific GPU designs.
4. Provide AI workload benchmarks and support the "HPC Applications WG" project area.

1.3 SubWG3 (Meteorology and Climate)

1.3.1 Objectives and Structure

SubWG3 conducted activities targeting representative applications in the meteorology and climate fields, with the aim of organizing application characteristics for FugakuNEXT, preparing for benchmark evaluations, and sharing information with the field community.

During the basic design phase, the group focused particularly on selecting early evaluation applications that would contribute to hardware design and on preparing the information and materials necessary for their evaluation.

1.3.2 Selection of Applications for Early Evaluation

SCALE-LETKF was selected as the early evaluation application in the meteorology and climate field.

SCALE-LETKF is an application that combines the regional climate model SCALE with a data assimilation system based on the Local Ensemble Transform Kalman Filter (LETKF), and includes representative computational patterns used in numerical weather prediction.

This application has the following characteristics:

- Numerical model calculations, including large-scale three-dimensional grid calculations and time-step evolution calculations
- Data assimilation calculations centered on matrix operations in LETKF
- A computational structure dominated by MPI communication and memory access
- Large-scale file I/O for input and output data

Based on these characteristics, we determined that this application is useful for evaluating next-generation computing architectures, including CPUs and GPUs, and positioned it as an application for early evaluation.

1.3.3 Preparations for and Submission of Benchmark Evaluation

For SCALE-LETKF, we prepared the following materials necessary for benchmark evaluation and submitted them to the Benchmark Working Group.

- Complete set of source code
- Input data for benchmark execution
- Document summarizing compilation procedures, execution procedures, and methods for verifying execution results

These materials have been organized to enable evaluation across multiple architectures, including Fugaku and GPUs.

In addition, we provided the SCALE-LETKF source code and related data to NVIDIA and Fujitsu for the purpose of detailed performance evaluation by vendors and future optimization studies.

1.3.4 Sub-WG Meetings and Community Collaboration

During the project period, SubWG3 held a total of nine review meetings and continuously discussed the following items.

- Policy for selecting applications for early evaluation
- Response to requests from the Benchmark WG
- GPU support status and future development policy
- Organization of Potential Applications Within and Outside the Field

Regarding potential applications from within and outside the field, four applications have currently expressed interest in contributing to future benchmarks.

In addition, a workshop was planned for March 26, 2026, with the aim of sharing the status of the FugakuNEXT project for the meteorology and climate fields and discussing future prospects. This workshop is positioned as a forum for information sharing and exchange of opinions with the field community.

1.3.5 Response to the Domain Survey

The meteorology and climate sector compiled responses to a survey of application developers conducted by the project team.

- First Survey: 10 responses
- 2nd Survey: 5 responses

These represent contributions from the field as a whole, and SubWG3 was responsible for compiling and submitting them.

1.3.6 Impact Assessment on SIMD Length

To evaluate the impact of different SIMD widths on application performance, we compared the execution times of 256-bit SVE and 512-bit SVE for SCALE (excluding the LETKF portion) on Fugaku by changing the compiler options and runtime environment settings.

As a result, a speedup of approximately 15% was confirmed when using 512 bits compared to 256-bit execution.

On the other hand, caution is required when interpreting whether this performance difference can be explained solely by the difference in SIMD width. Many SCALE kernels are memory-bound or have structures containing many branches, so it is not necessarily intuitive that a significant performance improvement can be achieved simply by expanding the SIMD width.

In fact, in addition to the difference in SIMD length,

- differences in compiler optimization behavior
- changes in the scope of vectorization
- differences in instruction scheduling and register utilization

and other complex factors may be at play.

Although this evaluation is a comparison under limited conditions and does not strictly isolate the impact of SIMD length, we share it as foundational information for considering architectural design and compiler optimization strategies.

1.3.7 Achievements and Remaining Challenges for This Fiscal Year

A major achievement of SubWG3 during the basic design phase was the completion of the selection and evaluation preparation of early evaluation applications, centered on SCALE-LETKF. On the other hand, the following challenges have become clear regarding this year's implementation.

- The porting of the LETKF portion to the GPU is incomplete, and performance evaluation and optimization utilizing the GPU are important tasks for the future.
- Although the SCALE component can run on a GPU, there is still room for performance improvement at this stage.
- Although it was expected that each SubWG would submit one or two evaluation applications, only one was submitted in the meteorology and climate field.

1.3.8 Key Items for the Next Fiscal Year (Detailed Design Phase)

SubWG3 considers the following items to be important for the detailed design phase next fiscal year.

- GPU implementation of the LETKF component and reorganization of the computational structure

  The porting of the LETKF component to GPU has not yet been completed, and it is necessary to proceed with implementation and performance evaluation based on GPU

usage. In particular, it is important to review the computational structure, which centers on matrix operations, including data layout suitable for GPUs and the utilization of libraries.

- Performance Improvement of the SCALE Component on GPUs
  Although the SCALE component can be executed on a GPU, sufficient performance improvements have not yet been achieved. There remains room for further optimization from the perspectives of memory access and kernel decomposition methods.
- Consideration of Surrogation and Reorganization of Algorithmic Elements
  It is considered important to examine each component of the overall numerical weather prediction and data assimilation system from the perspective of surrogating computationally intensive parts and reorganizing algorithmic structures. This will allow us to consider not only conventional computational methods but also the potential for introducing AI and machine learning techniques.
- Exploration of the Potential for Low-Precision Computation and Matrix Operation Engines
  In computations that make heavy use of matrix operations, such as those involving LETKF, investigations into the applicability of low-precision arithmetic and the utilization of dedicated matrix operation engines may contribute to future performance improvements and enhanced power efficiency. During the detailed design phase, we will evaluate the feasibility of these approaches while clarifying their relationship with numerical accuracy requirements.
- Selection and Submission of Additional Applications with Different Computational Characteristics
  Given the diversity of applications in the meteorology and climate fields, it is important to select additional key applications based on criteria such as communication characteristics, memory-boundness, and the ratio of matrix operations, and to submit them as benchmarks. This will enable us to provide more multifaceted input for the architectural design of FugakuNEXT.

Through these efforts, we aim to provide broader and more realistic input for the architectural design and optimization studies during the detailed design phase.

1.4 SubWG4 (Earthquake and Tsunami Disaster Prevention)

1.4.1 Objectives and Structure

S ubWG4 conducted a survey targeting representative applications in the field of earthquake and tsunami disaster prevention to organize application characteristics for FugakuNEXT, prepare for benchmark evaluations, and investigate application development for the coordinated use of CPUs and GPUs on FugakuNEXT. During the basic design phase, the group focused particularly on selecting applications for early evaluation that would contribute to hardware design, as well as preparing the program samples and input data necessary for their evaluation.

1.4.2 Selection of Applications for Early Evaluation, Preparation for Benchmark Evaluation, and Submission

E-Wave was selected as the application for early evaluation in the field of earthquake and tsunami disaster prevention.

E-Wave is a seismic wave propagation calculation application based on the implicit unstructured grid finite element method, and it includes representative calculation patterns in the field of earthquake and tsunami disaster prevention.

This application has the following characteristics.

- Sparse matrix-vector multiplication calculations, primarily using random access, as employed in the unstructured grid finite element method
- Computations in the conjugate gradient solver that are limited by memory bandwidth
- MPI communication in adjacent domains for sparse matrix-vector multiplication, and MPI_Allreduce communication in the conjugate gradient solver

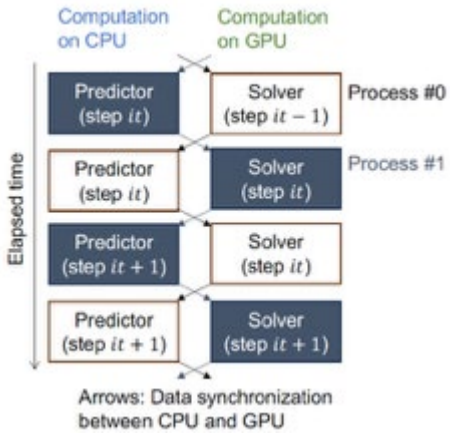
Based on these characteristics, we determined that this application is useful for evaluating next-generation computer architectures and designated it as an application for early evaluation.
Regarding E-Wave, we prepared the following materials necessary for benchmark evaluation and submitted them to the Benchmark Working Group.

- Complete set of source code
- Input data for benchmark execution
- Document summarizing compilation procedures, execution procedures, and methods for verifying execution results

These materials were organized to enable evaluation on multiple architectures, including Fugaku and NVIDIA GPUs. We conducted discussions to establish a research framework based on NDAs and joint research agreements so that this code could be utilized within the FugakuNEXT project.

1.4.3 Investigation into application development for CPU-GPU co-processing

When running the current E-Wave on the heterogeneous architecture consisting of CPUs and GPUs envisioned by FugakuNEXT, all computations will be performed on high-performance GPUs, while CPUs will be used solely for GPU control, resulting in CPU cores and CPU memory remaining unused.Utilizing CPU resources alongside GPU resources has the potential to reduce execution time and energy consumption. Therefore, SubWG4 investigated examples of CPU-GPU cooperative computing in this field and presented them at the FugakuNEXT Application Development Seminar (Presentation Title: Simultaneous use of CPU and GPU for fast and energy-efficient implicit wave simulation).Here, as an example of CPU-GPU cooperative computing development in this field, we introduced a method that simultaneously reduces both the execution time (time-to-solution) and energy consumption (energy-to-solution) in time-history simulations of numerous cases through the simultaneous utilization of resources [Ichimura et al. 2024].In this method, by using CPU computations to accurately estimate the initial solution for the next step based on past solution results stored in large-capacity CPU memory, we reduce the number of iterations of the iterative solver while maintaining the accuracy of the simulation results, thereby accelerating the computation.By assigning high-speed GPUs to computationally intensive solver executions and processing two cases in parallel, the method optimizes computational resources through the simultaneous use of CPUs and GPUs. When combined with other high-performance computing techniques, this approach achieves an 8.7-fold improvement in throughput and a 7-fold reduction in energy consumption compared to conventional GPU-only methods in a GH200 environment.
Based on these findings regarding CPU-GPU cooperative computing, we responded to the Developer Survey questionnaire.



Tsuyoshi Ichimura, Kohei Fujita, Muneo Hori, Maddegedara Lalith, Jack Wells, Alan Gray, Ian Karlin, John Linford: Heterogeneous computing in a strongly-connected CPU-GPU environment: fast multiple time-evolution equation-based modeling accelerated using a data-driven approach, Workshop on Accelerator Programming and Directives (WACCPD) 2024@SC24

1.5 SubWG5 (Manufacturing Sector)

1.5.1 Objectives and Structure

In the manufacturing sector, the application of HPC is advancing for first-principles high-precision analysis aimed at elucidating phenomena and fully replacing physical testing, as well as for multi-objective optimization that takes various constraints into account. In particular, in recent years, there has been a growing demand to shorten product development cycles to meet diverse market needs, leading to heightened expectations for innovation in design processes utilizing HPC.

SubWG5 conducted activities aimed at investigating the latest trends and needs regarding applications used in the manufacturing sector, and based on these findings, identifying target problems and the necessary computational resources for next-generation computing infrastructure.

During the basic design phase, the focus was particularly on selecting early evaluation applications that contribute to hardware design and on preparing the information and materials necessary for their evaluation.

1.5.2 Selection of Applications for Early Evaluation

Applications for early evaluation were selected from the field of turbulent flow analysis, a key challenge in the manufacturing sector. This is because in turbulent flow analysis, memory size, memory bandwidth, and inter-node communication performance often determine the scope of application, and significant performance improvements can be expected from the architecture of next-generation computing infrastructure, which emphasizes data transfer between GPUs and DRAM as well as data movement within DRAM.From this perspective, two applications with different memory access characteristics — namely, FrontFlow/blue (hereinafter FFB) and FFVHC-ACE—were selected as applications for early evaluation.

FFB is a general-purpose fluid analysis code based on Large Eddy Simulation (LES), which can accurately predict unsteady flow of incompressible and compressible fluids. It employs a finite element method with excellent geometric adaptability, enabling unsteady turbulent flow analysis around fluid machinery such as fans and pumps, as well as complex geometries, and the prediction of noise generated by the flow.Furthermore, it is optimized for high-speed operation on vector and scalar massively parallel computers and supports large-scale massively parallel computing through a domain decomposition method that implements automated optimal domain decomposition and merging.In fact, it has a proven track record on large-scale parallel computers such as "K" and "Fugaku," and a paper describing these results was selected as a finalist for the Gordon Bell Prize at SC20 (International Conference on High Performance Computing, Networking, Storage, and Analysis).

On the other hand, FFVHC-ACE is a fluid analysis code that enables high-fidelity compressible fluid LES analysis around complex geometries such as aircraft fuselages, and is characterized by the following three key technologies.

1) A hierarchical equidistant orthogonal grid method and a proprietary embedded boundary method that enable fully automatic grid generation for complex geometries
2) A wall model LES that enables LES analysis of high Reynolds number flows
3) A numerical scheme (KEEP scheme) that enables high-fidelity compressible fluid LES analysis

The major difference between FFB and FFVHC-ACE lies in the internal data structures used to store flow field information. Since FFB uses unstructured data, its computational structure relies heavily on list vectors in the main computational units.In contrast, FFVHC-ACE uses structured grid data, resulting in a computational structure dominated primarily by stencil operations. Since these two applications differ significantly in their memory access characteristics and computational patterns, we determined that they would serve as complementary benchmarks for evaluating next-generation computer architectures.

1.5.3 Preparation and Submission for Benchmark Evaluation

Regarding FFB, we have prepared the following materials necessary for benchmark evaluation and submitted them to the Benchmark Working Group. As for FFVHC-ACE, we have made the preparations listed below, but are currently coordinating regarding licensing and public availability.

- Complete set of source code
- Input data for benchmark execution
- Document summarizing compilation procedures, execution procedures, and methods for verifying execution results

These materials have been organized to enable evaluation across multiple architectures, including "Fugaku" and GPUs.

1.5.4 Sub-WG Discussion Meetings and Community Collaboration

During the activity period, SubWG5 held three review meetings and continuously discussed the following items.

- Selection policy for early evaluation applications (FFB, FFVHC-ACE)
- Response to requests from the Benchmark WG

1.5.5 Response to the domain survey

We compiled responses from the manufacturing sector to the survey for application developers conducted by the project team. These responses represent the consolidated opinions of the entire sector, and SubWG5 was responsible for organizing and submitting them.

1.5.6 Achievements and Remaining Challenges for This Fiscal Year

A major achievement of SubWG5 during the basic design phase was the completion of the selection of early evaluation applications centered on FFB and FFVHC-ACE, as well as the preparation for FFB benchmark evaluation. However, the following challenges have become apparent regarding this year's activities.

- Although FFB can run on a GPU, detailed profiling will be required in the future, and there is currently room for performance improvement.
- FFVHC-ACE requires support for GPUs.
- To proceed with detailed performance evaluations by vendors and future optimization studies, it is necessary to establish repository management and clarify licensing terms for providing source code and related data to relevant vendors.

1.5.7 Key Items for the Next Fiscal Year (Detailed Design Phase)

SubWG5 considers the following items to be important for the detailed design phase in the next fiscal year.

- Establishing an environment to facilitate vendor performance evaluations
  Regarding FFB and FFVHC-ACE, establishing a system for providing source code and related data is the top priority to facilitate detailed performance evaluations and optimization studies by vendors. Specifically, we must proceed with building and operating a repository based on the already agreed-upon policy, clarifying license terms, and establishing distribution procedures to enable the early launch of the evaluation environment.
- Promoting GPU-Based Implementation and Performance Optimization
  Based on the performance evaluation results from vendors, it is important to perform tuning focused on the main computational components and to progressively advance performance optimization with the GPU as the primary processing unit.

Through these efforts, rather than merely passively accepting evaluation results from vendors, we aim to organize computational characteristics and operational requirements from the perspective of application developers and provide feedback on hardware design. This will enable us to contribute broader and more practical input to architectural design and optimization considerations during the detailed design phase. It should be noted that the

disclosure of hardware design information is essential for efficiently advancing such collaborative activities.

1.6 SubWG6 (Basic Science)

1.6.1 Objectives and Structure

SubWG6 conducted activities targeting representative applications in the basic science field, with the objectives of organizing application characteristics for FugakuNEXT, preparing for benchmark evaluations, and sharing information with the scientific community.

1.6.2 Selection of Applications for Early Evaluation

From the basic science field, LQCD-DWF-HMC (https://github.com/i-kanamori/LQCD-DWF-HMC) and HDTurbulence (https://github.com/cfcanaoj/MHDTurbulence) were selected.

LQCD-DWF-HMC was selected from the particle physics and nuclear physics fields; it is a lattice QCD (Lattice Quantum Chromodynamics; Lattice QCD or LQCD) simulation code and an application focused on the physics of elementary particles—quarks and gluons.Lattice QCD simulations have evolved in tandem with the development of parallel computers and account for a significant portion of HPCI projects (18% of computing resources allocated to selected general projects on "Fugaku" in Phase A of FY2024 and 14% in Phase A of FY2025).The selected code employs Domain-Wall Fermions (DWF), which excel at handling chiral symmetry—a key property of QCD—to describe quarks, and uses the Hybrid Monte Carlo (HMC) method to generate the gluon field configuration.The core of the computation is a solver that solves partial differential equations using an iterative method; an extracted version of this component was also provided as a kernel application. The code is written in C++ and, with the exception of architecture-specific optimizations, is implemented within the scope of C++11.

This code has the following characteristics.

- It requires the handling of complex numbers
- Frequent communication with neighboring nodes occurs due to stencil calculations on the lattice
- Parallelization is a hybrid of MPI and OpenMP
- Global reduction associated with the iterative method also occurs frequently
- Memory bandwidth and communication are rate-limiting factors
- Mixed-precision algorithms with FP32 are used
- There is no need to prepare a separate file containing initial conditions at runtime
- GPU implementation uses OpenMP (for kernel applications, a CUDA version is also provided in addition to OpenMP)

Based on these characteristics, we have identified this as a useful application for evaluating next-generation computing architectures, including GPUs. The license inherits the GPLv3 from the original code set (Bridge++; https://bridge.kek.jp/Lattice-code/), but an MIT-licensed version for the kernel portion is currently in preparation.

MHDTurbulence was selected from the space sector; it is a turbulence simulation code that solves three-dimensional magnetohydrodynamics (MHD) equations, serving as a fundamental and general-purpose application for studying space plasma and interstellar medium.This code solves compressible fluid equations based on the finite volume method and can be applied to a wide range of astrophysical problems, including astrophysical phenomena such as the Sun, stars, and supernova explosions, as well as galaxy formation and evolution and accretion disk simulations.

This code is written in Fortran90 and has the following features.

- Large-scale finite difference calculations on a three-dimensional uniform lattice
- Iterative Updating Using an Explicit Time-Stepping Scheme
- Domain-decomposition parallelism using MPI
- GPU offloading implementation using OpenMP and OpenACC
- No need for a file containing initial conditions at runtime

We determined that these computational characteristics are highly dependent on memory bandwidth, communication performance, and computational performance (particularly vector processing performance), making them useful for evaluating next-generation architectures, including GPU accelerators. Furthermore, because the application is grounded in fundamental physics yet features a relatively simple computational structure, making it easy to conduct performance evaluations and optimization studies, we positioned it as an application for early evaluation. The license is BSD 3-Clause.

Furthermore, both code sets are highly portable because they use conservative language specifications and rely on few dependent libraries.

1.6.3 Preparation and Submission for Benchmark Evaluation

For LQCD-DWF-HMC, we prepared the following materials required for the benchmark evaluation and submitted them to the Benchmark Working Group.

- Complete set of source code
- Input data for benchmark execution
- Document summarizing compilation procedures, execution procedures, and methods for verifying execution results

These materials have been organized to enable evaluation on Fugaku and NVIDIA GPUs. Additionally, we provided similar materials to NVIDIA and Fujitsu for the purpose of detailed performance evaluation by the vendors and future optimization studies, and responded to individual inquiries. We also provided source code, parameters, and information regarding the execution environment for evaluating communication performance using Fugaku. We provided benchmark results regarding performance variations based on SIMD length.

There are two versions of this application's source code: a full application version and a kernel version. The full application version was initially provided as a tarball but was ultimately released on GitHub under the GPLv3 license. As of the time of this report, the kernel version remains a tarball; however, we are preparing to release a version with the license changed to MIT on GitHub after obtaining the developer's consent. Note that the kernel version tarball has already been incorporated into BenchKit.

Regarding MHDTurbulence, we prepared the following materials necessary for benchmark evaluation and submitted them to the Benchmark Working Group.

- Complete source code
- Documentation summarizing compilation procedures, execution procedures, and methods for verifying execution results

These materials have been organized to enable evaluation across multiple architectures, including both CPU and GPU. Initially, only the GPU version was available, but as part of this initiative, we have newly prepared the CPU version to enable benchmark execution on a variety of architectures.

1.6.4 Sub-WG Meetings and Community Collaboration

Within the SubWG, information was exchanged primarily via online channels and email.

For the broader fundamental science community, information regarding FugakuNEXT and application trends was shared at the 25th HPC-Phys Study Session, hosted by the Japan Institute for Computational Fundamental Science (JICFuS) in September 2025.

In the particle physics and nuclear physics fields within basic science, information regarding benchmarks is provided on an ongoing basis, primarily by developers involved in the research. We are also in contact with groups within the LQCD community that use different quark models than DWF, as well as researchers investigating AI acceleration algorithms.

We are sharing information regarding benchmarks primarily with Fugaku users in the space science field. Within the community, there is high interest in GPU offloading for more complex applications, including those using adaptive mesh refinement (AMR) and iterative methods. In the next phase of early evaluation, it is hoped that codes with these differing computational characteristics will also be included.

1.6.5 Response to Domain Surveys

The two domain surveys conducted were handled within the SubWG.

1.6.6 Achievements and Remaining Challenges for This Fiscal Year

The selection of LQCD-DWF-HMC and HDTurbulence as early evaluation applications, and in particular the fact that the former led to evaluation during the basic design phase, are major achievements of SubWG6. On the other hand, the following issues have become clear:

- Regarding LQCD-DWF-HMC, there is a discrepancy with the vendor's licensing requirements (specifically, that it must not be GPL). We plan to provide a non-GPL version of the kernel component in the future.
- Regarding the method of providing the source code, LQCD-DWF-HMC was not made publicly available. While the full application was released on GitHub in December 2025, the release of the kernel application has been delayed.

1.6.7 Key items for the next fiscal year (detailed design phase)

LQCD-DWF-HMC will expedite the release of a non-GPL licensed version of the kernel application while advancing communication optimization for GPUs. Additionally, the project will promote the use of QUDA, a related library developed by NVIDIA for GPUs. Algorithms utilizing half-precision arithmetic, currently under development in the original application (Bridge++), will also be incorporated.Furthermore, while it has been suggested that pod size may significantly impact performance in this application (approximately 12% according to the RFP), it has also been suggested that QUDA, which utilizes half-precision arithmetic, can reduce this gap. A detailed investigation of this impact will be conducted during the detailed design phase, and the findings must be fed back into the hardware design.

Currently, MHDTurbulence is divided into multiple implementations: OpenMP (CPU), OpenMP (GPU), and OpenACC. Going forward, we aim to establish a unified implementation framework across various computing environments by adopting Kokkos, thereby enabling easier and more equitable performance comparisons. Looking ahead to the detailed design phase, we will proceed with porting to Kokkos to streamline architecture-dependent components and improve portability.

1.7 SubWG7 (Smart City Domain)

1.7.1 Objectives and Structure

SubWG7 focused on representative applications in the smart city domain, working to organize application characteristics and prepare benchmarks for FugakuNEXT. During the basic design phase, the group prioritized selecting candidate applications for early evaluation to aid in assessing next-generation supercomputers, as well as making the minimum necessary preparations to enable benchmark execution.The target applications are a group of simulations involving multi-physics and multi-agent modeling, such as city-scale radio wave propagation and electromagnetic field analysis, evaluation of communication methods (millimeter-wave and terahertz), power and renewable energy behavior (e.g., ZEC including window-type perovskite solar cells), smart grids, EV charging and discharging behavior, and vehicle, pedestrian, and agent behavior.However, a fully integrated simulator has not yet been completed; we are currently in the process of gradually establishing execution environments and benchmarks for each component.

1.7.2 Selection of Applications for Early Evaluation

SUMO, which incorporates NVIDIA Sionna and Agentic AI, was selected as the application for early evaluation in the smart city field. Sionna features GPU-accelerated ray-tracing-based 3D radio field propagation and link-level simulation capabilities, while SUMO, incorporating Agentic AI, provides an evaluation platform for vehicle, pedestrian, and agent decision-making. Currently, both operate independently and are capable of generating benchmarks.

This application has the following characteristics:

- Spatial computations, including large-scale 3D ray tracing (searching for numerous rays and paths across high-resolution spatial data).
- Computations centered on matrix operations and neural network inference (beamforming, channel matrix generation, Agentic AI inference, etc.).
- High branching complexity and irregular memory access associated with event-driven behavior of numerous agents (dominated by conditional branching and dynamic data structures).

Based on these characteristics, we determined that the application is useful for evaluating the performance of next-generation computing architectures, including CPUs and GPUs, and positioned it as an application for early evaluation.

1.7.3 Preparation and Submission for Benchmark Evaluation

For both SUMO and Sionna, we prepared the following materials necessary for benchmark evaluation and submitted them to the Benchmark Working Group.

- Complete source code
- Input data for benchmark execution
- Document summarizing compilation procedures, execution procedures, and methods for verifying execution results

These materials have been organized to enable evaluation on the Fugaku architecture.

1.7.4 Sub-WG Review Meetings and Community Collaboration

During the project period, a total of two review meetings were held within SubWG7, as well as a review meeting with the Benchmark WG, to continuously discuss the following matters.

- Selection of Applications for Early Evaluation
- Presentation of Applications to the Benchmark WG and Sharing of Development Progress
- Response to requests from the Benchmark WG
- GPU Support Status and Future Development Policy

1.7.5 Response to the sector survey

We compiled responses for the smart city sector in response to a survey of application developers conducted by the project team.

- 1st Survey: 1 response
- 2nd Survey: 1 response

1.7.6 Achievements and Remaining Challenges for This Fiscal Year

The primary achievement of SubWG7 during the basic design phase was the selection of SUMO—which incorporates NVIDIA Sionna and Agentic AI—as an early evaluation application. We ensured each component could operate independently and established an environment capable of performing benchmarks. Additionally, we organized build and execution procedures designed for operation on the Fugaku architecture and completed the initial setup of the evaluation platform.

On the other hand, the following challenges have become apparent.

- The integrated simulator, which includes Sionna, SUMO, and electromagnetic and power system simulators scheduled for future introduction, remains incomplete; currently, evaluations are limited to individual components.
- Optimization of code for GPUs and area-partitioned parallelization using MPI have not been implemented or are limited, and large-scale scalability evaluations have not been conducted.
- Although development is proceeding primarily in the Singularity environment, preparations for a standard operating environment based on FugakuNEXT—including Spack support,

build verification with Fujitsu compilers, and compatibility with Benchpack/BenchKit—remain incomplete.

1.7.7 Key Items for the Next Fiscal Year (Detailed Design Phase)

The following items are important for SubWG7 in preparation for the detailed design phase.

- Simulator Integration
  Establish a configuration that enables integrated evaluation at the urban scale by linking Sionna, the Agentic AI-embedded SUMO, and the electromagnetic field and power system models scheduled for future introduction.
- MPI Support
  Implement MPI parallelization based on the partitioning of the urban space to enable scalability evaluation.
- Standardization of Build and Runtime Environments
  In addition to the current Singularity-based development environment, we will add support for Spack and conduct build and operational verification using Fujitsu compilers.
- Benchpack/BenchKit Support
  We will reorganize the application to conform to the operational framework of the Benchmark Working Group and prepare it for integration into a standard evaluation framework.

Through these efforts, we aim to establish an execution platform for integrated smart city simulations during the detailed design phase and provide effective input for architecture design and performance evaluation.

1.8 SubWG8 (Scientific Computing and Machine Learning Algorithms)

1.8.1 Objectives and Structure

SubWG8 focused on fundamental computational libraries for numerical simulation and deep learning. Its activities aimed to organize the characteristics of these libraries when used in applications on FugakuNEXT, configure mini-applications and prepare datasets for evaluating library performance in benchmarking, and share information with the scientific community.

1.8.2 Selection of Applications for Early Evaluation

From the fields of numerical simulation and deep learning, PTESc (https://petsc.org) and PyTorch (https://pytorch.com) were selected. These are collections of numerical computation libraries for parallel computing environments that bridge the gap between foundational numerical libraries, such as BLAS and LAPACK, and actual applications.

PETSc is a numerical computation library written in C and continuously developed by Argonne National Laboratory. It solves linear and nonlinear systems of simultaneous equations composed of sparse matrices — derived from the discretization of mathematical models described by partial differential equations—in a parallel computing environment based on data distribution.Sparse matrices are stored in the CSR format, a storage format for unstructured data, and the library implements the Krylov subspace method—a solution method for large-scale systems of linear equations involving sparse matrices — along with several high-performance preprocessing routines. The fundamental numerical operation here is SpMV, which computes the product of a sparse matrix and a vector. This library has the following features.

- Execution of MPI processes in a heterogeneous parallel environment combining CPUs and GPUs
- Separation of data distribution operations and linear/nonlinear solver algorithms via an abstracted interface to the matrix data format
- Interfaces not only for C but also for applications written in Fortran and Python

Based on these characteristics, we have identified it as a useful numerical library for evaluating next-generation computing architectures, including GPUs.

PyTorch is a fundamental numerical library for deep learning that provides an interface in the Python language. Large language models (LLMs) have evolved primarily around the Transformer architecture, but their foundation is supported by floating-point operations, data distribution, and communication for parallel execution. This library has the following features:

- It runs in a heterogeneous parallel environment mixing CPUs and GPUs, supporting direct data transfer between GPUs in addition to distributed execution via MPI.
- Due to its high computational granularity—where computation dominates over data communication—and high computational density, it fully utilizes the processing capabilities of the latest GPUs.
- Although there are training and inference phases, both are being accelerated through mixed-precision computation.

Based on these characteristics, we have positioned the PETSc library as a useful numerical library for evaluating next-generation computing architectures, including GPUs, using numerical computations that handle dense data complementary to the sparse matrix data handled by the PETSc library.

1.8.3 Preparation and Submission for Benchmark Evaluation

Since both PETSc and PyTorch are numerical computation libraries, evaluating their performance requires an application and a dataset that utilize the library. For PETSc, we created a C program that reads ASCII files containing sparse matrix data generated by the finite element method and saved in MatrixMarket format, stores this data in PETSc's internally distributed sparse matrix storage, and computes numerical solutions using the Krylov subspace method.The program is executed via a script that configures parameters for the selection of the Krylov subspace method and preprocessing, as well as for iterative execution and convergence testing. For the PyTorch evaluation, we selected LLM training using Megatron-LM (https://github.com/NVIDIA/Megatron-LM/). We chose both Dense and MoE models as targets and prepared parameters for distributed training.

I submitted a benchmark set consisting of the two scripts mentioned above, data sources, and parameter settings to the Benchmark WG.

1.8.4 Sub-WG Meetings and Community Collaboration

During the activity period, SubWG8 held a total of nine review meetings and engaged in ongoing discussions regarding the following items via Zoom meetings as well as the Slack channel " ."

- Policy for Selecting Applications for Early Evaluation
- Response to requests from the Benchmark WG
- GPU Support Status and Future Development Policy
- Promoting the Use of Numerical Libraries in Other Application WGs

We contacted the PETSc community directly to obtain information regarding the feasibility of execution in a GPU environment and development trends on compute nodes with multiple GPUs.

- Due to the characteristics of GPU hardware, preprocessing to reuse data immediately after computation has not been implemented
- PETScSF, a communication library for a new distributed environment supporting multi-GPU execution, has been implemented in the development version

1.8.5 Response to Domain Surveys

The two domain surveys were addressed within the Sub-WG. In particular, linear and nonlinear problems involving sparse matrices handled by PETSc are memory-limited. However, since some preprocessing implementations have the potential to increase computational density, we provided responses based on our considerations regarding the balance between memory bandwidth and computational speed in GPU environments.

1.8.6 Achievements and Remaining Challenges for This Fiscal Year

Initially, we envisioned an open-source elastic body simulation code written in Fortran90 as a mini-application utilizing PETSc. However, it was discovered that the NVIDIA Fortran compiler does not fully implement the latest language specifications, making it impossible to compile the Fortran interface portion of PETSc. Consequently, we decided to write the execution routines for the sparse matrix solver in C++ and to prepare the sparse matrices either within the evaluation application or as external datasets.

PyTorch development progresses rapidly, and executable versions of library objects and scripts for benchmarking are provided in a virtual environment with each version update. For performance evaluation on actual hardware, it was necessary to select a stable version. We selected PyTorch versions for both the Grace Hopper supercomputer—which has an instruction set architecture similar to FugakuNEXT—and the current Fugaku system.

1.8.7 Key Items for the Next Fiscal Year (Detailed Design Phase)

To run PETSc in an environment where multiple GPUs are directly interconnected, the PETScSF communication library must be used. Since this is a development version, collaboration with the PETSc development community must be strengthened.Since the availability of a Fortran interface would expand the user base for domestic applications, we will identify the compiler-specific features required to build the PETSc library. For sparse matrix-vector multiplication (SpMV) on GPUs, we currently use libraries provided by GPU vendors, but a SpMV backend using Kokkos Kernels has also been implemented. We need to evaluate whether PETSc should handle data distribution in mixed CPU-GPU environments or whether to utilize Kokkos’s capabilities.

Communication can sometimes be a bottleneck in LLM training. Since the use of a large number of parameters is essential for advanced training, it is necessary to examine whether execution can be performed efficiently in an environment where multiple GPUs are directly interconnected.