Operating System for the K computer

● Jun Moroo ● Masahiko Yamada ● Takeharu Kato

For the K computer to achieve the world's highest performance, Fujitsu has worked on the following three performance improvements in the development of the operating system (OS). First, to bring out the maximum hardware performance of our original CPU and interconnect, we provided a mechanism of controlling hardware extensions directly from applications. As the second improvement, we have introduced the synchronization scheduling function that minimizes the synchronization wait time of parallel programs resulting from system interruptions by coordinating job runtime and system runtime between multiple nodes. Third, multiple page size support that allows use of more than one page size has been achieved for improved memory access performance and memory utilization efficiency. This paper also describes the performance improvement functions, usability and robustness of the OS developed.

1. Introduction

Supercomputers are used in large-scale simulation computations in a variety of science and technology fields. For weather forecasts, for example, the prediction area can be divided at the municipality level or even further at the station level to calculate values such as the temperature, wind direction, and wind speed for each sub-area, which enables highly accurate predictions to be made. Aerodynamic simulations on aircraft allow people to estimate how an aircraft will move without having to build a model or actual aircraft. In the development of new drugs, substances that are candidates for a medicine can be extracted out of a vast number of types and combinations to narrow them down to those with a curative effect.

Carrying out such large-scale simulations involves enormous amounts of memory and compute nodes and a huge number of files. Accordingly, an operating system (OS) capable of bringing out the maximum hardware performance of the supercomputer is essential.

This paper gives a description of the OS of the K computer[note)] that is capable of conducting large-scale simulations.

2. Software configuration of the K computer

Generally, supercomputers perform high-speed computations by:

1) Dividing huge computations
2) Operating many computers (compute nodes) concurrently in a coordinated manner
3) Transferring computation results at high speeds between compute nodes by using a set of standard communication libraries called Message Passing Interface (MPI)

The OS processes application startup and IO

note) "K computer" is the English name that RIKEN has been using for the supercomputer of this project since July 2010. "K" comes from the Japanese word "Kei," which means ten peta or 10 to the 16th power.

processing requests and takes charge of system processing including time of day control. For the OS of the K computer, we have maximized the hardware and software performance by improving the OS kernel and libraries. We worked on developing the OS of the K computer with a target of achieving 10 PFLOPS, a performance level ten times higher than the system at the time of its initial development.

The basic software of the K computer is composed of the OS and basic middleware (operations management software, language system) (**Figure 1**). The OS implements the architecture-dependent portion of the Linux kernel and has additional drivers for using the hardware of the K computer so that it can be used from the middleware in the upper level.

We have worked on developing the OS to install on the K computer, aiming for targets including improved usability, improved performance and improved robustness as shown in **Table 1**. The following sections present the respective targets.

3. Improved usability (effective utilization of user assets)

Some of the existing supercomputers use special OSes, on which software that has been

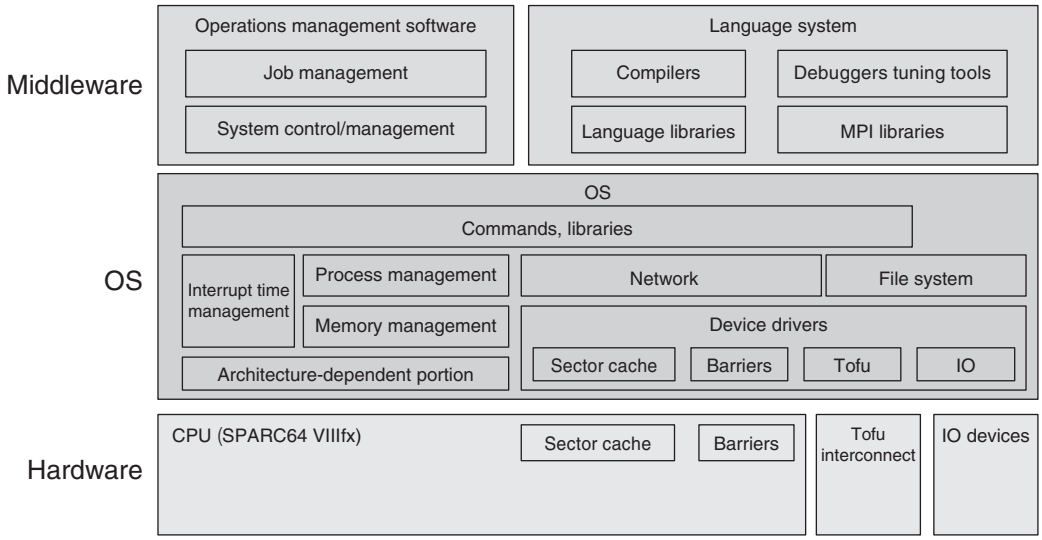

Figure 1
Software configuration of the K computer.

Table 1
Features of the K computer OS.

	Feature	Realization method
Improved usability	Effective utilization of user assets	Linux POSIX API
Improved performance	Direct control of hardware extensions	SIMD instruction support Hardware barrier Sector cache Extension of device drivers for hardware control registers
	Improved scalability	Synchronization scheduling with rsadc
	Improved memory performance and efficiency	Large page support
Improved robustness	Security RAS enhancement	User access protection Fallback for failure

developed by the user cannot be used. For the OS of the K computer, we have adopted Linux to allow continued use of the user's assets.

Similarly to general OSes, the OS of the K computer is equipped with process management, memory management, IO management (device driver), file system and network functions. Linux is used in more than 91% of the TOP500 ranked supercomputers and we have adopted it in view of making user applications portable. Users can use POSIX API, a UNIX application programming interface (API), on the K computer.

In addition, the OS of the K computer has API (library, command) extensions for using extended hardware, which allows easy use of hardware functions through the compiler and MPI library functions.

These contrivances enable the OS of the K computer to port general open source software (application programs, tools, etc.) by recompilation without having to change the source program.

4. Improved performance

4.1 Direct control of hardware extensions

SPARC64 VIIIfx is a SPARC64 VII-based CPU equipped with expanded registers, extended SIMD instructions, hardware barriers between cores and sector cache for speedup. For the OS of the K computer, we have prepared a mechanism with additional device drivers that allow these hardware extensions to be controlled directly from applications. The drivers map registers that control barrier synchronization and sector cache into the memory space of applications, by which high-speed control is achieved without involving system call overhead.

The sector cache is intended to be used to divide the cache in the CPU into two virtual regions to facilitate caching of the data that are repeatedly used by applications.

Specifically, the complier generates memory access instructions to cache less frequently used data in sector 0 and more frequently used data in sector 1. This prevents frequently used data from being expelled from the cache, and thus improves the execution performance of applications. The amount of frequently used data may vary depending on the functions in applications. Accordingly, the ratio of the sector cache used from user applications has been made changeable during program execution to allow the utilization efficiency of the cache to be optimized.

The performance improvement effect of cache control cannot be sufficiently achieved if overhead is generated due to the issuance of a system call when a device driver is called along with sector cache operation. For that reason, device drivers for the K computer have been designed and implemented to allow user applications to directly control the registers that control the sector cache. This has successfully reduced the sector cache control time from a few microseconds to a few nanoseconds.

4.2 Improved memory access performance

The OS of the K computer is equipped with large page support as a function to provide both high memory access performance and high memory utilization efficiency.

Generally, the virtual memory management unit provided for Linux for 64-bit SPARC manages the memory used by programs in 8 Kbyte page units.

A cache (TLB: translation lookaside buffer) is provided for speeding up the address translation performed by the hardware but the number of entries is limited. With a page size of 8 Kbytes, use of memory of a few Mbytes or larger causes the process to exceed the capacity of the TLB and overhead resulting from OS address translation is generated.

If the page size to be managed with one entry is increased, high-speed access to memory of a few Gbytes or larger can be achieved by means of a TLB. However, managing the entire

data area with one-sized large pages may reduce the memory utilization efficiency because of the unused memory area in a large page. To address this problem, the OS of the K computer allows the page size to be selected from the options of 4 Mbytes, 32 Mbytes and 256 Mbytes.

For example, to run an application that uses a stack larger than the area for static data including global variables, a large page can be allocated to the stack and a small page to the data area. In this way, both improved memory utilization efficiency and improved performance thanks to the reduction of TLB misses can be achieved (**Figure 2**).

4.3 Improved scalability

We have developed a synchronization scheduler and statistics sampling mechanism for the purpose of improving the performance of parallel jobs across multiple nodes (scalability improvement).

4.3.1 Synchronization scheduler

A single computation job that runs on the K computer system is a parallel program deployed on multiple nodes and, every time a certain computation process is completed, mutual synchronization and communication take place between nodes connected via interconnects.

For system operation, the OS and operations software daemons (programs that run in the background for system management) need to run, and they run asynchronously regardless of the job.

This daemon operation causes interruptions (system noise) of a parallel program. In computation with tens of thousands of nodes, interruptions may increase in proportion to the number of nodes, leading to a significant performance degradation. There are two ways to solve this problem:

1) Dedicate a specific core of a node composed of multiple CPU cores to the OS to run the daemon for eliminating interruptions in the job on each node (spatial division)
2) Reduce interruptions in synchronization/communication of a parallel program to a certain level independently of the number of nodes by synchronized running of daemons between compute nodes (synchronization scheduler method) (temporal division)

With the method described in 1), when one of the eight cores of the CPU of the K computer is allocated to the system and seven cores to the job, the execution efficiency is limited to 87.5% (seven-eighths). Accordingly, we have developed the synchronization scheduler method described in 2) for the K computer.

With the synchronization scheduler of the K computer, the OS uses the barrier function of the Tofu interconnect to achieve inter-node synchronization. The synchronization scheduler enhances the process scheduler function of Linux to perform synchronization at intervals of 100 ms, for example, and run the daemons in 1 ms out of the 100 ms. The parallel program can

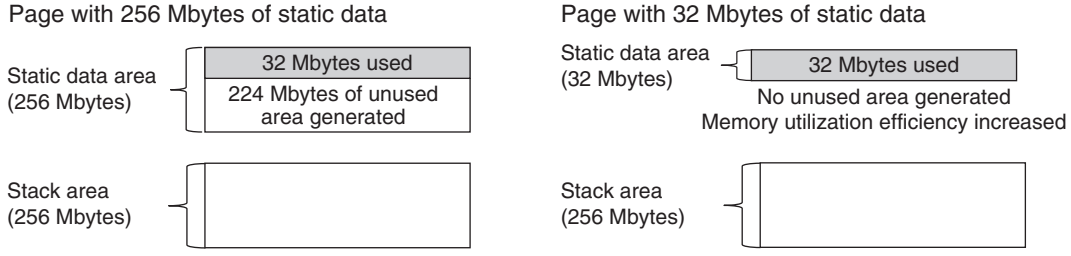

Figure 2
Effect of multiple page size support in process that uses 32 Mbytes of static data.

run 99% of the time in synchronization between all nodes without interruptions, which allows for a performance improvement in proportion to the number of nodes (**Figure 3**).

4.3.2 Statistics sampling function

The K computer reduces interruptions themselves by using the synchronization scheduler to lower the effect of daemons to jobs and optimizing daemon processes. As an example, this section describes the improvement of the statistics sampling function.

When a user application runs on a compute node, daemon processes that run periodically in the system can cause interruptions (system noise) to the user application and result in variation in execution performance of the user application (the execution performance may vary depending on the execution timing of the user application).

For this reason, reducing system noise on a compute node requires the daemon processes that run periodically in the system to be minimized.

To monitor to ensure normal operation of the system and analyze performance bottlenecks, the OS samples statistics including CPU and memory usage. Generally, for OS statistics, the *sadc* command is run via *cron*, a function intended for periodically executing system processes, to read kernel data and write to the statistics file. This method does not cause major system noise with small systems but with the K computer, which is an enormous system, this statistics sampling process may result in system noise.

For the K computer, remote *sadc* function (*rsadc*), which collects compute node performance statistics in IO nodes, has been developed (**Figure 4**). We have aimed to reduce costs and improve reliability with the K computer by aggregating IO devices onto IO nodes in each rack. We have reduced the number of interruptions by offloading to these IO nodes the IO processes of performance statistics of the

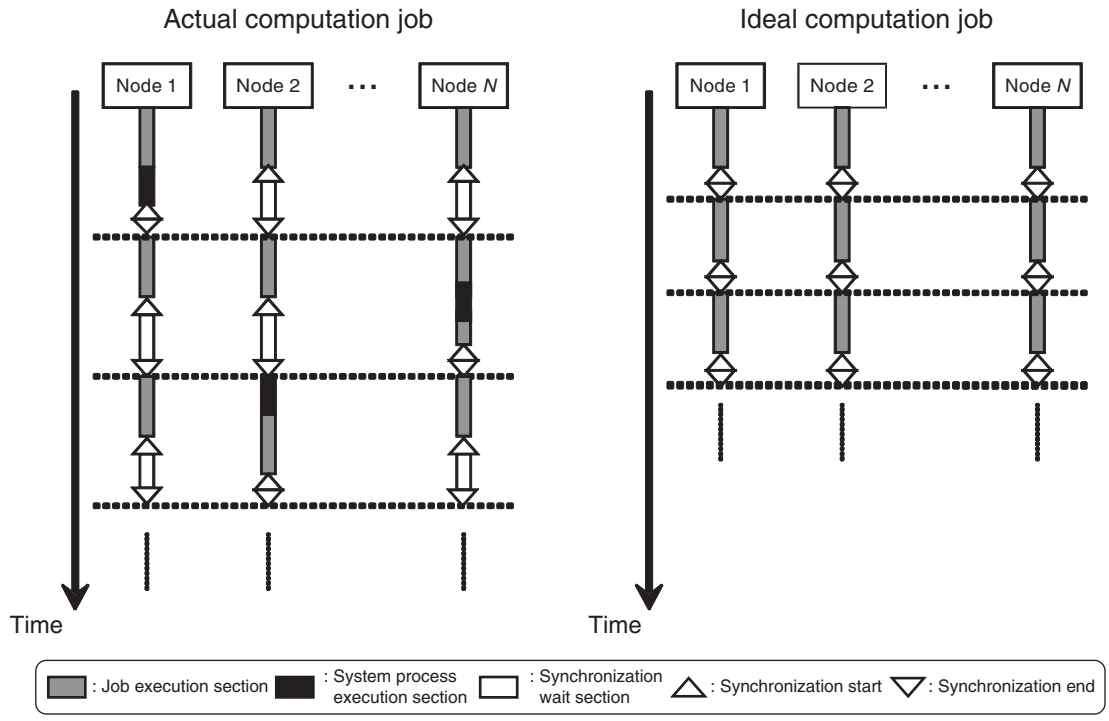

Figure 3
Operation of synchronization scheduler.

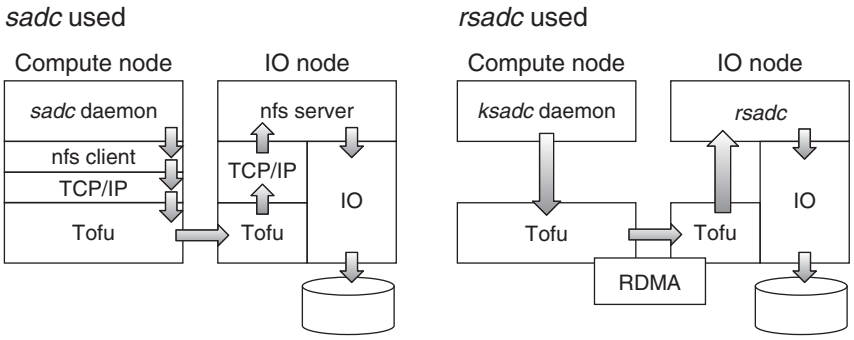

Figure 4
Schematic diagram of *rsadc*.

many compute nodes.

The *ksadc* daemon of compute nodes maintains the performance statistics of the individual nodes in the memory and carries out IO processes by means of *rsadc* on IO nodes. The command *rsadc* uses the remote direct memory access (RDMA) function of the Tofu interconnect to transfer the performance statistics of the individual compute nodes to IO node memory without daemon processes on compute nodes, which can be stored on disks.

For the K computer, we have developed a function for a system composed of many racks so that it can collect log files required for billing and system management from IO nodes of the many racks into one control node without accessing compute nodes. This has enabled the administrator to gain an understanding of the conditions of the entire system without causing interruptions in parallel jobs.

5. Improved robustness

The K computer, which is composed of a large number of hardware components, is a system required by various users to perform lengthy computations and it needs to be highly robust. In relation to hardware failures, for example, it is important to have reliability, availability and serviceability (RAS) functions, which allow continued operation even if one component fails and early detection of the point of failure for replacement. In addition, the system must be secure so that it is possible to share files within user groups or prevent other groups from viewing the files.

The K computer has been given high robustness by making use of hardware redundancy, RAS functions provided for Linux including multipath driver and security function and adding to Linux the RAS functions that have been enhanced in Fujitsu's existing supercomputers and servers.

The following subsections describe examples of the enhanced RAS functions and security functions.

5.1 RAS enhancement

The OS of the K computer has inherited the existing server technologies of Fujitsu to enhance the functions of intermittent memory error recovery and identification and notification of hardware failure points.

To address memory errors caused by alpha rays and such like, the OS of the K computer cooperates with the memory patrol function of the hardware to perform data correction, thereby making the system highly reliable.

The hardware performs memory read (patrol) at intervals specified by the OS. By using the error correcting code (ECC) mechanism, the OS writes data to the memory when any single-bit error is detected in the memory data, by which error correction including ECC is performed. Performing this error correction

periodically prevents correctable single-bit errors from developing into uncorrectable double-bit errors, which improves reliability.

Once any failed page is detected, the OS marks it to prevent it being allocated memory when any subsequent memory allocation request is issued, thereby retiring the page.

When any memory error is detected, the OS specifies the slot containing that memory and notifies the administrator of it so as to reduce the time from stopping the node to replacing the component. The RAS functions have been enhanced to allow precise identification of possible points of failure based on error events detected in relation to CPU and IO devices (PCI channel, Gbit Ethernet adapter, Fibre Channel adapter, etc.) in addition to memory.

5.2 Security functions

The K computer system is used simultaneously by many users and it is essential that it has security functions. However, security functions do not make sense if they affect the computing performance.

The security functions of the K computer have been achieved by combining the user management and file system functions normally used in UNIX-based OSes.

The OS supports the following security functions in relation to authentication by general user management.

- Password encryption
- Change of file and directory access rights
- Change of owners of files and directories
- Change of group ownership of files and directories

For files and directories, security management is provided based on the controlled access protection profile (CAPP), which controls access by "read," "write" and "execute" permissions for user, group and other.

6. Conclusion

This paper has described the features of the OS of the K computer. The development of an OS for a globally unparalleled large-scale system was a challenging task, and we needed to achieve both ultimate performance and usability. We intend to continuously work on developing the OS to provide the foundation for more easily conducting large-scale simulations on the K computer and help advance various fields in science and industry.

Jun Moroo
Fujitsu Ltd.
Mr. Moroo is currently engaged in software development for next-generation HPC.

Takeharu Kato
Fujitsu Ltd.
Mr. Kato is currently engaged in software development for next-generation HPC.

Masahiko Yamada
Fujitsu Ltd.
Mr. Yamada is currently engaged in software development for next-generation HPC.