ISSN 1349-1121
JAXA-RM-14-011E

JAXA Research and Development Memorandum

High Sustained Performance and Scalability on a Multicore-Based Massively Parallel Cluster of JAXA Supercomputer System

Yuichi Matsuo, Naoyuki Fujita and Ryoji Takaki

March 2015

Japan Aerospace Exploration Agency

High Sustained Performance and Scalability on a Multicore-Based Massively Parallel Cluster of JAXA Supercomputer System*

Yuichi Matsuo*1, Naoyuki Fujita*1, and Ryoji Takaki*1

ABSTRACT

Multicore processing has become a major trend in high-performance computing. However, there are two major issues with multicore technologies: 1) how to make full use of modern multicore CPU capability with reduced user workload, and 2) how to achieve better scalability up to a high degree of parallelism for memory-intensive applications such as computational fluid dynamics. First, the multicore-based massively parallel cluster of the Japan Aerospace Exploration Agency (JAXA) supercomputer system, which uses the Fujitsu FX1 as the core computer, is described. Notable features such as a high-speed barrier network, enhanced memory bandwidth, and an integrated mechanism to hide latency, i.e., the integrated multicore parallel architecture, which comprises a hardware barrier, L2 cache shared among cores, and an associated automated parallelization complier, are described. Second, the process to achieve significantly high sustained performance (>90%, high-performance Linpack benchmark) on the multicore CPU cluster system is explained. Finally, performance measurement results for JAXA applications are provided.

Keywords: Sustained performance, Scalability, Multicore, Massively parallel cluster system, JAXA supercomputer system

1. INTRODUCTION

The Japan Aerospace Exploration Agency (JAXA) has been involved in various activities in aerospace fields, including research, development, and application of aerial/space vehicles. The 20 consecutive successful launches of the H-IIA launch vehicle No. 26 in 2014 was a significant achievement. Since the era of the National Aerospace Laboratory of Japan, a predecessor of JAXA, JAXA has recognized the importance of numerical simulations using high performance computers. JAXA has promoted the installation and operation of large-scale high performance computing systems like the Numerical Simulator (NS) [1,2,3]. Real applications in the aerospace field tend to require large-scale simulations with high-speed computing or large amounts of memory, i.e., large-scale parallel computers.

All of the recent large-scale parallel computers used in scientific computations face the following two challenges. The first is how to utilize a multicore CPU easily and efficiently. The ability to increase CPU clock speed shows signs of leveling off, and the methodology to increase CPU performance is changing from clock-up to increasing the number of cores in a CPU. The second issue is how to improve parallel scalability to allow applications to use an extremely large number of cores effectively. Currently, practical applications use several hundreds of cores, and applications that use tens of thousands of cores are expected. However, the methodology to ensure parallel scalability with tens of thousands of cores has not been established.

Under the abovementioned circumstances, JAXA operated the Central Numerical Simulation System (CeNSS), a symmetric multiprocessor (SMP) cluster-type large-scale parallel computing system from October 2002 to April 2009. In April 2009, it was replaced with a multicore-based scalable parallel cluster with approximately 12,000 cores, peak performance of 120Tflops, and main memory of 94TB. This is the main component of the JAXA Supercomputer System (JSS).

The remainder of this paper is organized as follows. First, we review the history of supercomputing systems at JAXA. Then, we present the concept and design of the JSS and describe the new multicore-based scalable parallel cluster system comprising a Fujitsu FX1 node, which is the primary computing engine of the JSS. We also present the results of a performance evaluation using the high performance Linpack (HPL) benchmark and the current aerospace computational fluid dynamics (CFD) applications on the cluster.

* Received 19 December, 2014

*1 Supercomputer Office, Security and Information Systems Department

2. BACKGROUND

2.1 Numerical Simulator III

Since October 2002, JAXA operated a terascale SMP-cluster supercomputer system, called NS-III [3]. The NS-III had approximately 1,800 scalar processors, peak performance of 9.3Tflops, and main memory of 3.6TB. The main computing subsystem of the NS-III was the CeNSS, which had 18 cabinets, i.e., physical hardware units. Each cabinet was a Fujitsu PRIMEPOWER HPC2500 server with 128 CPUs and shared memory of 256GB. At maximum configuration limits, it was able to act as a 128-way SMP system. The CPU was the SPARC64^{TM} V scalar chip with a 1.3GHz clock. The theoretical peak performance per CPU was 5.2Gflops and 665.6Gflops per cabinet. Each chip employed a shared L2 cache of 2MB. A cabinet could be partitioned into two or four nodes according to the computational requirements. Here from an operating system perspective, a node is a logical unit. In the CeNSS, each compute cabinet was partitioned into four nodes, where each node was a 32-way SMP with a 64GB shared memory, giving a total of 56 compute nodes. All nodes were connected to a crossbar interconnect network through one data transfer unit per node.

Regarding the programming environment, we adopted the so-called hybrid programming paradigm, i.e., we used the "thread parallel" model within a node with automatic parallelism or OpenMP, and we used the "process parallel" model with the message passing interface (MPI) or XPFortran (similar to HPF) among nodes. For example, a triple do-loop, which is commonly used in CFD, is parallelized as follows. The outermost do-loop can be parallelized by process parallelism, and the remaining inner do-loops can be parallelized by thread parallelism if the loops are independent. The use of automatic parallelization strongly helped users execute applications written for the vector processor system in the past and on the new scalar processor system. Program modifications were not necessary because automatic parallelism can only be attained by specification during compilation and execution. In practice, hybrid parallelization, i.e., the combined application of process and thread parallelism, is quite difficult for users; thus, automatic parallelism is better than OpenMP. In the case of inner do-loop parallelization, it is better to parallelize the outer do-loops as often as possible in terms of the parallelization overhead. However, when the compiler failed to parallelize outer do-loops due to the limited capability of the automatic parallelization compiler, the compiler parallelized the innermost do-loop. Consequently, the performance and scalability of thread parallelism was not as good as expected. Therefore, the benefit of the large-size SMP was not attainable.

In the past, we used a SIMPLEX mode for job execution. Users were able to accurately predict the elapsed time for a job subject to the computational cost. The CeNSS had a function similar to the SIMPLEX mode. However, the elapsed time fluctuated whenever a user re-ran a job. Consequently, we could not generate accurate job execution plans. The fluctuation in the elapsed time was due to the inherent characteristics of the SMP machine. When only one process executes on one node, the process can use all of the node's memory bandwidth. However, when N processes execute on the same node, one process can only use $1/N$ of the total memory bandwidth. The memory bandwidth that each process can occupy decreases or is affected by other processes because, in practice, it is difficult to assign only one process to one node. In particular, some memory intensive jobs strongly affected the elapsed time of other jobs running on the same node. The fluctuation of the elapsed time not only makes job execution planning difficult but also makes program tuning terribly difficult. Whether the fluctuation of the elapsed time is due to program tuning or memory contention caused by other jobs has not been identified. The influence among jobs sharing the same node is an inherent characteristic of the SMP machine, and a fundamental solution is desired.

2.2 JAXA Supercomputer System; JSS

The JSS is the first supercomputer system implemented since the creation of JAXA in October 2003. It is expected to contribute significantly to JAXA's missions by fully utilizing numerical simulations. In addition, the JSS is a symbol of "One JAXA," the integration of Japan's three space agencies. In addition to previous activities in aeronautics, active expansion to the space field is an important part of JAXA's long-term vision, which is referred to as JAXA 2025. Numerical simulations with the powerful supercomputer can make great contributions to JAXA's mission through improved reliability, reduced cost and lead time, and creation of innovative concepts for aerospace systems.

The primary requirement for the JSS is computational power, and the peak performance should be at least 10 to 20 times greater than that of the NS-III. Unsteady three-dimensional simulations with quite fine grid resolution, typified by large eddy simulations, are indispensable for current numerical simulations in the aerospace field. Such simulations require the management of large amounts of three-dimensional and time-dependent data. Therefore, in addition to computational power, the JSS should have high-performance data management capabilities, such as high-speed I/O and a large-scale data archive function. To apply numerical simulations directly to aerospace vehicle development, the system should be user friendly. For example, it should be able to handle commercial application programs and open-source software, achieve optimization easily, enable flexible job scheduling, and demonstrate operational and middleware reliability to realize a large number of parametric studies. Moreover, a seamless and secure environment is important to enable the JSS to be used by remote JAXA field centers. Finally, to

avoid interfering with the ongoing JAXA projects, which depend largely on supercomputers, smooth and rapid transition from the NS-III to the JSS is important.

JAXA introduced the JSS in April 2009. The system consists of the following: a large-scale parallel computing system, a storage system, a shared memory system, and a remote access system, as shown in **Figure 1**.

The large-scale parallel computing system comprises the Main (M) system and the Project (P) system. Both systems comprised Fujitsu FX1 clusters, which are multicore-based scalable parallel clusters. The peak performance of the M and P systems is 120TFlops and 15TFlops, respectively. The amount of main memory of the M and P systems is 94TB and 6TB, respectively. The storage system has 1PB for disk storage and 10PB for tape storage. The effective performance of the disk I/O is greater than 25GB/s. The hierarchical storage management has been introduced to manage large data transfers between disk and tape storage. The shared memory system consists of an application (A) system and a vector (V) system. The A system comprises a Fujitsu SPARC Enterprise M9000 node, which has 1TB main memory and 32 scalar processors. The V system comprised three NEC SX-9 nodes. The NEC SX-9 has a peak performance of 4.8TFlops, 3TB main memory, and 20 TB local storage. Each SX-9 node has 16 vector processors and is connected with an IXS crossbar network.

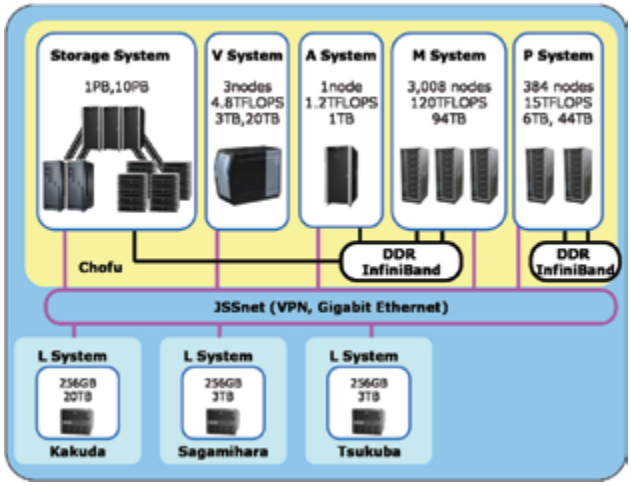


Figure 1: Overview of the JSS

The main JSS systems are installed at the Chofu field center and remote access systems are installed at major field centers. These remote access systems act as file servers and front-end servers at each field center, and they are connected through a virtual private network called JSSnet.

The JSS has approximately 15 times greater peak performance, 25 times more memory, and 20 times greater storage than those of NS-III. The JSS also has a high memory bandwidth (i.e., B/F = 1; ratio of bytes per flops) to enhance the effective computing performance, full-band interconnect, and powerful I/O capability. Moreover, the V system and the A system are prepared for vector users and pre-post processing, respectively. The large-scale parallel computing system and the storage system are the key parts of the JSS, whose details are described later.

3. FX1 CLUSTER DESCRIPTION

3.1 System overview

The JSS consists of several subsystems; however, the central system is a scalable parallel cluster system based on Fujitsu FX1. The FX1 cluster is a massively parallel system in which 3,008 compute nodes are connected with a full bisectional bandwidth fat-tree interconnection Infiniband™ DDR (Double Data Rate) network. **Figure 2** shows the configuration of the FX1 cluster, and its major specifications are listed in **Table 1**.

Each compute node contains one quad-core SPARC64™ VII chip (2.5GHz clock speed), two specially designed chipsets called JSC, DIMM memory modules, power units, and HCA for inter-node communication. A compute node has a theoretical performance of 40Gflops with 32GB memory. The CPU chip has a mechanism called integrated multicore parallel architecture (IMPACT), which comprise a hardware barrier among cores, 6MB shared L2 cache, and the associated compiler technology. This architecture enables automatic and high-speed parallel-thread execution on a multicore CPU without adding OpenMP directives. Moreover, a dedicated chipset to realize a high memory bandwidth enables high performance.

Essentially, the FX1 cluster performs data communication using a fat-tree network. It has another network, called the high-speed barrier network, which speeds up barrier synchronization and global collective operations, and mitigates the operating system interference among nodes.

We can attempt to solve large-scale problems using IMPACT and the high-speed barrier network without considering a parallel programming model that is suitable for multicore CPUs and degradation of parallel efficiency in large-scale parallel execution.

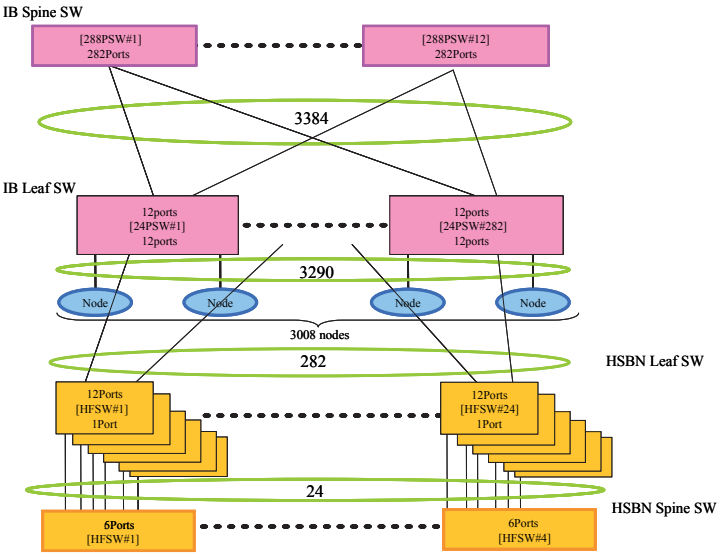


Figure 2: FX1 cluster configuration

Table 1: FX1 cluster specifications

CPU	Microprocessor	SPARC64™ VII
	L2 cache	6 MB shared
Node	# of CPU	1
	Memory	32 GB
	Memory bandwidth	40 GB/s
	I/O	Infiniband™ DDR HCA
Chassis	# of nodes	4
	Height	5U
Rack	# of chassis	8
System	# of racks	94

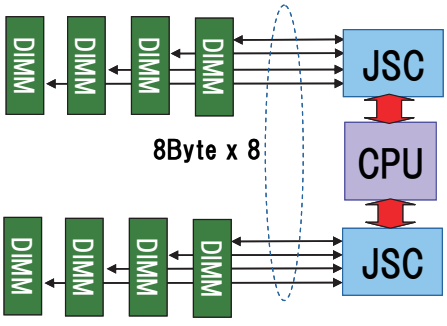
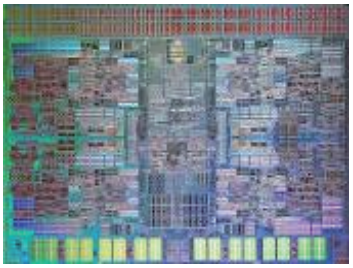


Figure 3: FX1 node structure



(a) SPARC64™ VII chip

(b) Compute node board

(c) Chassis

(d) Rack

(e) System

Figure 4: FX1 system packaging

The structure of a single FX1 node is shown in **Figure 3**, where two JSC chipsets are implemented. This enables a high memory bandwidth (1 Byte/Flop). **Figure 4** shows the FX1 components: a quad-core SPARC64™ VII chip, a board (i.e., compute node), a chassis that contains four boards, and a rack that contains eight chassis. In total, 94 racks are installed in the 3,008-node FX1 cluster.

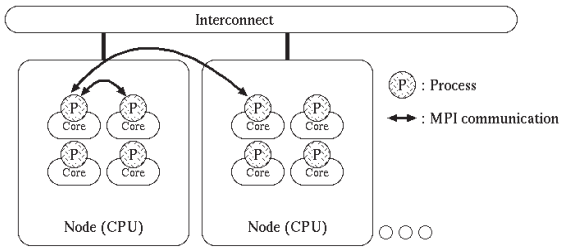
3.2 FX1 cluster features

The major features of the FX1 cluster are as follows.

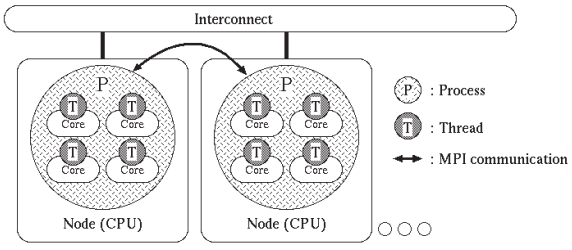
- ***SPARC64™ VII processor:*** The SPARC64™ VII is a quad core CPU based on the SPARC V9 architecture. It is designed and produced by Fujitsu. Each core is a 64-bit microprocessor with a clock speed of 2.52GHz augmented with two M&A floating point operation units that provide two fused multiply add operations per cycle for a total of 10.08Gflops per core or 40.32Gflops per CPU.
- ***Simple but high-performance compute node:*** A compute node is composed of a single CPU, memory module, IO module, and a DC-DC converter. This is quite simple and compact, which makes it possible to have a high memory bandwidth and short memory latency, which leads to high sustained performance.
- ***Integrated multicore parallel architecture (IMPACT):*** IMPACT is accomplished by a combination of hardware and software technologies. IMPACT consists of a shared L2 cache among the SPARC64™ VII cores and a hardware barrier among the cores. The FX1 employs an advanced automatic parallelization compiler for the software. Highly efficient automatic parallelization of the innermost loops is possible with IMPACT. Note that this used to be difficult; however, we can now handle a multicore CPU as a single CPU virtually. In addition, in combination with the special chipset that enables a high memory bandwidth (1 Byte/Flop), high parallel thread performance can be obtained.
- ***High performance and high function interconnect:*** The compute nodes are connected with multiple interconnection networks.
  1. *High-bandwidth data network:* This network is the main network of the FX1 cluster. It handles MPI packets or network file access. The network is a full bisectional bandwidth fat-tree network with Infiniband™ DDR, which alleviates sustained performance degradation caused by communication conflicts and minimizes communication time variation. The fat-tree network comprises 24 24-port Infiniband™ leaf switches and 12 288-port Infiniband™ spine switches. This two-stage fat-tree network results in a hop count of at most five with low latency, and I/O nodes and an application node that serve as pre/post processing are also connected with Infiniband™.
  2. *High-speed barrier network:* This network is used to speed up the inter-node barrier and global collective operations, such as broadcast and reduction, and serves to mitigate interference from operating systems, both of which could be bottlenecks at a high degree of parallelism. The high-speed barrier network is implemented such that it is added to the high-bandwidth network, and specially designed high-function switches are used.

3.3 FX1 cluster parallelization

Figure 5 shows the parallelization models for a multicore CPU on the FX1 cluster. The flat-MPI model assigns each MPI process to each core in a CPU, and the IMPACT model assigns each thread to each core in a CPU. The IMPACT model is typically recommended with the FX1 cluster. The IMPACT model is a type of hybrid parallelization model that assigns each process (MPI or XPFortran) to each node (where each node is equal to a CPU), in which four threads are parallelized automatically. In this model, process parallelism should be written explicitly by users; however, thread parallelism is realized automatically by the automatic parallelism compiler.



a) Flat-MPI model



b) IMPACT model

Figure 5: Parallelization models (Flat-MPI and IMPACT)

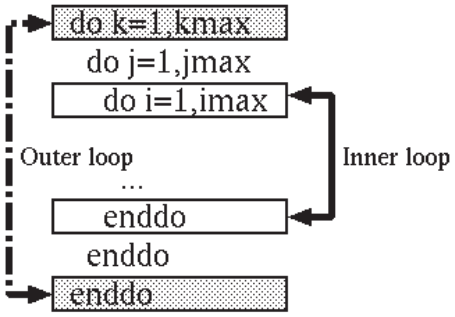


Figure 6: Loop parallelism

In thread parallelism, loop parallelization dominates the entire parallelization of the code. One important problem is which loop should be parallelized in the parallelization of multiple loops, as shown in **Figure 6**. Typically, parallelization of the outermost loop is the most efficient; thus, the automatic parallelism compiler attempts to parallelize the outer loop as often as possible. However, due to the loop complexity of real applications and limited capability of the compiler, the compiler tends to parallelize the innermost loop rather than the outermost loop. In innermost loop parallelization, the parallelization overhead has a considerable impact on thread scalability and decreases the parallel performance. Thus, the past automatic parallelism model could not deliver high performance in thread parallelization. On the other hand, IMPACT can achieve high performance even in the innermost loop parallelization due to a considerable decrease in the thread parallelization overhead. The high thread scalability in the innermost loop parallelization and the analytical capability enhancement of data dependency and operations in the loop increase the potential region and performance of thread parallelization. Moreover, efficient thread parallelization in the innermost loop theoretically improves thread scalability for applications optimized for the vector processor. In the case of vectorization, a part of the data dependency cannot be vectorized, e.g., inverse dependency. However, IMPACT can parallelize such a case. The IMPACT technology increases the potential region of automatic thread parallelization and achieves considerably good scalability.

Here, we present an example to show the superiority of IMPACT. We provide a detailed discussion of the overall performance of the FX1 cluster in Section X. The hardware feature of IMPACT is a shared L2 cache and a high-speed barrier among cores. The shared cache can decrease the number of unnecessary data transfers between caches in the thread parallelization and improve thread scalability, which is degraded by false sharing. **Figure 7** shows the effects of

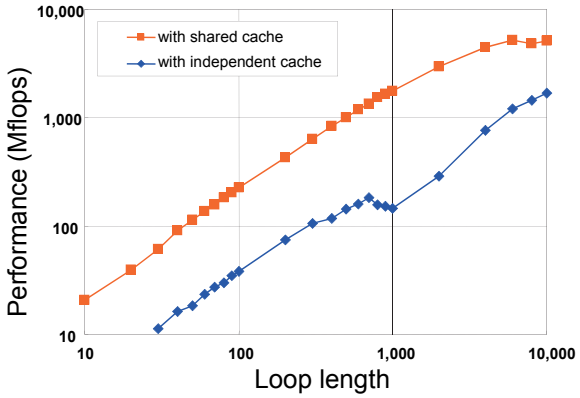


Figure 7: Effect of shared cache

the shared cache in DAXPY (EuroBen Kernel 8 [4]). The shared cache can improve performance dramatically.

The high-speed barrier among cores is implemented by the hardware, demonstrating a rate that is 10 times faster than when implemented by software. The performance achieved with IMPACT is approximately equal to the initiation overhead of a vector pipeline. These technologies decrease the thread parallelization overhead dramatically and achieve highly scalable thread parallelism.

4. MICRO-BENCHMARKS AND KERNEL PERFORMANCE

To understand the performance characteristics of the system, we measured the performance of individual components using micro-benchmarks and kernels.

4.1 Performance of micro-benchmarks

Here, the kernel performance of the FX1 cluster is compared in micro-benchmark programs with that of the Fujitsu HX600 [5] with an AMD Shanghai CPU, which has a clock speed of 2.3GHz and four cores (equivalent to the FX1).

Figure 8 shows the performance of STREAM TRIAD [6] with 4-core execution. The result of the FX1 cluster is 13.7GB/s, which is twice as fast as the HX600 due to FX1's superior memory bandwidth. Therefore, even for real applications (e.g., CFD) that require a high memory bandwidth, high memory performance can be expected. However, the effective performance (=measured/theoretical) of the memory access of the FX1 is approximately 34.25%, which is lower than that of the HX600. This is due to the adoption of the Chipkill technology in the FX1, which can correct a 4-bit memory error and improve the availability and reliability of systems

with very large amounts of memory. The effective memory access performance of the FX1 is approximately one-half that of the HX600, which has no Chipkill technology. This is consistent with the theoretical fact that adoption of the Chipkill technology decreases the effective performance of memory access by half.

Figure 9 shows the performance of DAXPY (EuroBen Kernel 8 [4]) with 4-core execution. The FX1 cluster with IMPACT shows much better performance than the HX600 at the region where the loop length is relatively small. This is due to the small overhead of thread parallelization realized by IMPACT.

Figure 10 and **Figure 11** show the effects of the high-speed barrier network on the performance of MPI barrier and Allreduce communications, respectively. The latency of both communication methods was measured with up to 256 nodes, showing a nearly constant time due to the high-speed barrier network. In Figure 11, the increase in the latency for six and eight nodes is due to software implementation rather than hardware implementation. This is due to hardware resource limitations in the real operating environment. However, this occurs for less than eight nodes, and the increased latency is not crucial; thus, this issue does not become a problem in actual operation.

For MPI barrier communications, latency was measured with up to 512 nodes, showing 9.94 µs at 512 nodes. The high-speed barrier network can connect a maximum of 768 nodes, and measured good performance is expected for up to 768 nodes. However, the latency with 1024 nodes increases significantly to 248.51 µs. For more than 768 nodes, the high-speed barrier network cannot be used, and barrier synchronization is performed by software, which results in significant performance degradation.

The performance enhancement with more than 768 nodes remains a future issue. However, we can show the reasonable effectiveness of this approach, such as the high-speed barrier network for large-scale computation.

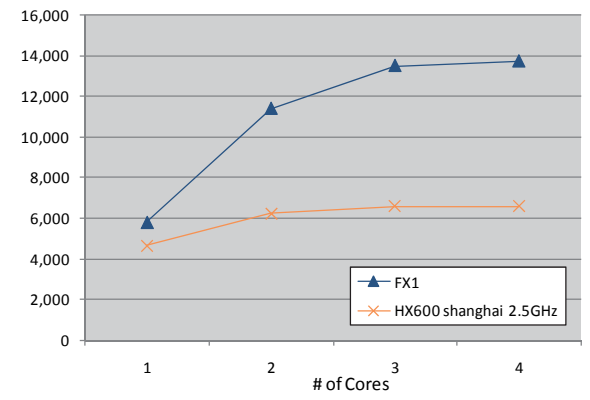


Figure 8: STREAM TRIAD performance

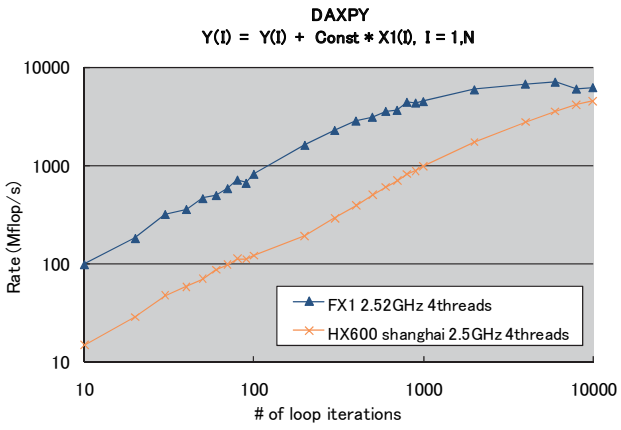


Figure 9: DAXPY performance

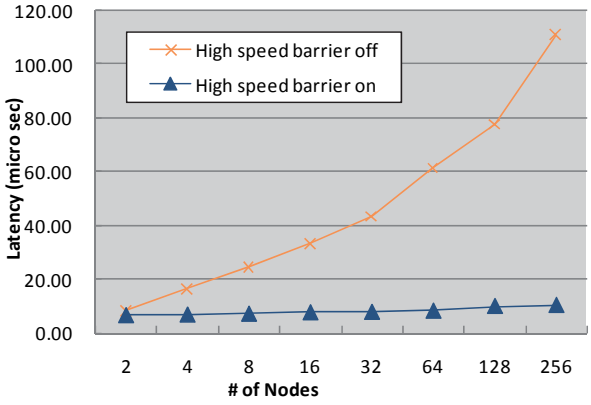


Figure 10: MPI barrier performance

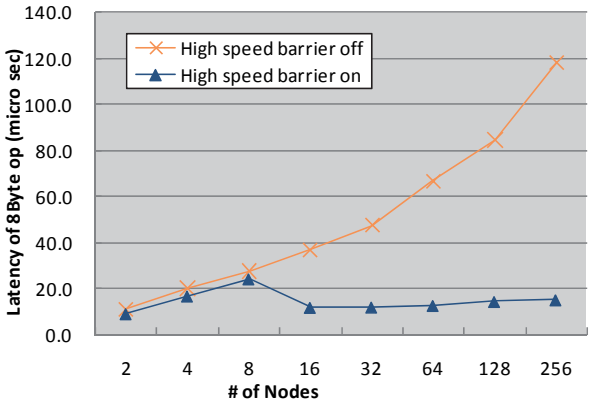


Figure 11: MPI Allreduce performance (8 bytes)

Table 2: NPB 3.1 Class C performance

Code	MOPS		Scalability
	1 Nodes (4 Cores)	256 Nodes (1,024 Cores)	
EP	17.5	4,258.9	242.8
BT	1,425.7	281,681.5	197.6
CG	396.1	65,979.3	166.6
FT	653.0	185,539.6	284.1
IS	74.2	6,356.1	85.6
LU	1,781.0	382,233.9	214.6
MG	1,327.9	494,243.3	372.2

4.2 Performance of NPB

The performance of the FX1 cluster was measured using NAS parallel benchmarks (NPB) [7]. The NPBs are a small set of programs designed to help evaluate the performance of parallel supercomputers, including eight benchmarks (e.g., EP, BT, CG, and several problem size classes: A, B, C). The measured results for NPB 3.1 Class C are listed in **Table 2**. The measurements were conducted with a single core and 256 nodes (i.e., 1,024 cores). A combination of MPI and IMPACT was used with 256 nodes. The average speed increase ratio with 256 nodes is approximately 220, showing good scalability for the speed-up problem with a fixed problem size. A compiler message during compilation showed an error for the automatic parallelization for EP and BT. Therefore, both cases were measured again with manual parallelization using OpenMP directives, which showed good scalability. This indicates that the current automatic parallelism compiler requires improvement.

4.3 HPL results

We measured HPL [8] on the FX1 cluster. The parameters and measurement results are summarized in **Table 3** and **Table 4**.

The current HPL results can be characterized by high sustained performance and high sustained efficiency ($R_{\max}/R_{\text{peak}}$). As can be seen in **Table 5**, systems are selected from the Top500 List [9] (November 2008), in which the number of cores is greater than 10,000 and efficiency is greater than 80%. Note that the FX1 has the highest sustained efficiency.

The reasons why such high sustained efficiency can be achieved with the FX1 cluster for HPL are summarized as follows.

High-performance BLAS

On the FX1, a high-performance BLAS library [10] is employed. This library is optimized such that the maximum performance of SPARC64™ VII can be achieved.

Table 3: HPL parameters

N	3,308,800
NB	440
P	32
Q	94
BCAST	2 ringM
DEPTH	1
L1	No-transposed
U	Transposed

Table 4: HPL measurement results

System	Fujitsu FX1 cluster
# of nodes	3,008
# of cores	12,032
R_{max}	110.6 Tflops
R_{peak}	121.3 Tflops
Efficiency	*91.19 %*
Execution time	60h 40 m

Table 5: HPL list sorted by efficiency

Rank	Computer	Cores	RMax	RPeak	Efficiency	Nmax	Exec Time [h]
	FX1	12,032	110,600	121,283	91.19%	3,308,800	60.65
14	Altix ICE 8200EX	12,288	128,400	146,736	87.50%	1,817,000	8.65
17	Altix ICE 8200EX	10,240	106,100	122,880	86.34%	1,535,480	6.32
46	Cray XT4	11,328	54,648	63,437	86.15%	-	-
75	Blue Gene/P	12,288	35,123	41,779	84.07%	-	-
56	Blue Gene/P	16,384	46,830	55,706	84.07%	933,887	3.22
57	Blue Gene/P	16,384	46,830	55,706	84.07%	933,887	3.22
24	Blue Gene/P	32,768	92,960	111,411	83.44%	-	-
32	Appro Xtreme-X3	10,000	76,460	92,000	83.11%	1,508,000	8.31
33	p5 575	12,208	75,760	92,781	81.65%	1,383,600	6.47
69	Blue Gene	16,384	37,330	45,875	81.37%	663,551	1.45
5	Blue Gene/P	163,840	450,300	557,056	80.84%	2,580,479	7.07
11	Blue Gene/P	65,536	180,000	222,822	80.78%	1,766,399	5.67
16	Blue Gene/P	40,960	112,500	139,264	80.78%	-	-
30	Cray XT5	10,400	76,800	95,680	80.27%	-	-
39	Altix	13,824	66,567	82,944	80.26%	1,478,736	9.00
4	Blue Gene	212,992	478,200	596,378	80.18%	2,456,063	5.74

Specifically, high performance DGEMM [10] makes a great contribution to the HPL performance improvement because 97% of the execution time of HPL is spent on DGEMM calculations. In this measurement, based on the DGEMM routine, which is highly tuned for SPARC64™ VII one core, and by further tuning to make the best use of the shared L2 cache suited for IMPACT, we can obtain 2% higher DGEMM performance. Thus, we can obtain 94.6% of the peak hardware performance. The BLAS library is used frequently in scientific computations; thus, supplying high performance BLAS will significantly improve the performance of such scientific computations.

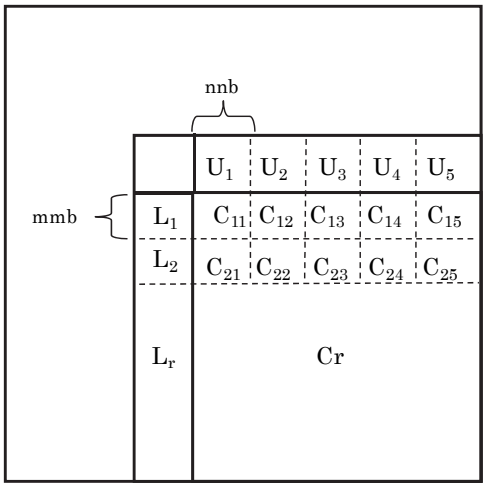


Figure 12: Segmentation of *L* panel for data transfer

Communication latency hiding

In HPL, communication is overlapped such that, while updating the *C* part, the *L* panel required for the next iteration can be transferred in the column direction. In the original HPL, the *L* panel communication between a sender and a receiver is synchronized because the *L* panel is large. Then, the sender must wait until the update of the *C* part is finished and the receiver is ready. To solve this problem, we modified the HPL such that the wait time of the sender is reduced by the following process: 1) update of the *C* part is divided into small sub-blocks (**Figure 12)**; 2) the receiver confirms whether the sender is ready to send at the end of each sub-block; 3) the send-receive action is completed after the negotiation; 4) all small sub-blocks are updated until the right edge; and 5) the rest of the *C* part (the greater part of *C* in Figure 12) is updated simultaneously. As a result, we can obtain a 1% HPL performance improvement. In addition, we reduced the number of processes that send and receive messages with IMPACT parallelization. As a result, the communication overhead was relieved on up to 12,032 cores.

Large problem size based on high hardware reliability

It is well known that the performance of HPL can be improved by the increasing problem size. The performance variation (peak performance ratio versus problem size) using 512 nodes on the FX1 cluster is shown in **Figure 13**. In HPL, the number of arithmetic operations is proportional to the cube of the problem size; thus, increasing the problem size will cause a dramatic increase in the execution time. Typically, a problem size with such an extended execution time (e.g., several hours) is selected because extended execution at over 90% load will involve a considerable reliability risk. In this measurement, under the conditions of maximum problem size (N_{max} = 3,308,800) inside the memory capacity and no restricted execution time, we eventually achieved *91.19%* average sustained performance for an extended execution time (*60 h 40 min*). As a result, a 2% performance improvement can be obtained as compared to the case in which the problem size is half. In Table 5, the execution time for the systems (except FX1) is less than 10 *h*, demonstrating that the FX1 cluster is highly reliable, even for scientific computations.

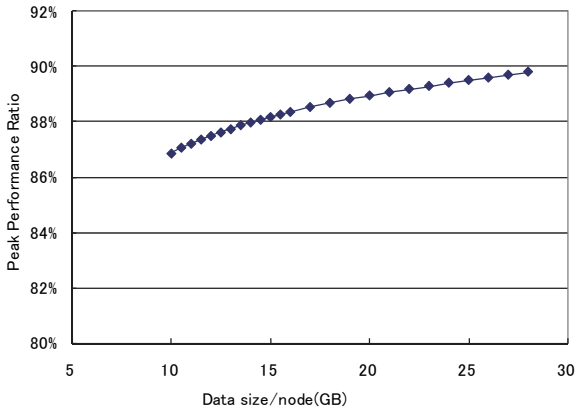


Figure 13: HPL performance and data size

5. JAXA PARALLEL APPLICATIONS

5.1 Overview of the applications

The performance of five typical applications used at JAXA (JAXA Benchmark Programs: JBP [11~15]) were measured on the FX1 cluster. An overview, typical computational results, and characteristics are listed in **Table 6**, **Figure 14**, and **Figure 15**, respectively. In Figure 15, the vertical and horizontal axes show the data communication ratio and the memory access ratio to the total CPU time, respectively.

Table 6: JBP list

Code	Application in aerospace	Numerical Method	Parallel Strategy
P1	Combustion	FDM + Chemistry	MPI + IMPACT
P2	Aeronautics	FVM (Structured)	MPI + IMPACT
P3	Turbulence	FDM + FFT	XPF + IMPACT
P4	Space Plasma	PIC	MPI + IMPACT
P5	Aeronautics	FVM (unstructured)	MPI + IMPACT

FDM: Finite Difference Method, FVM: Finite Volume Method, PIC: Particle in Cell

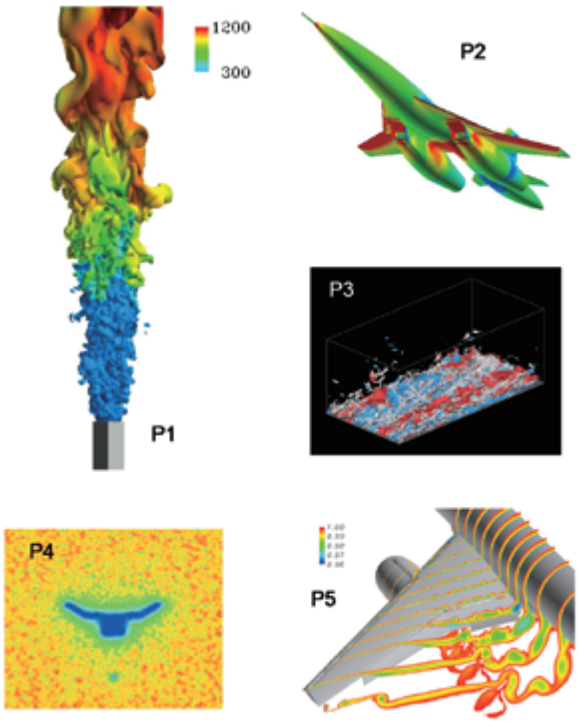


Figure 14: Typical JBP results

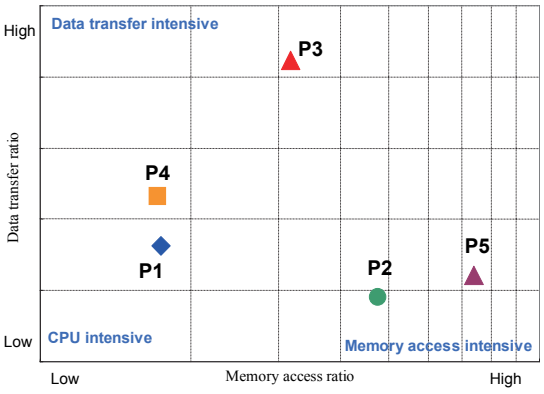


Figure 15: JBP characteristics

5.2 Single-node performance

Table 7 shows a comparison of the execution time for JPB between the FX1 and the HX600. Here, one FX1 node and one HX600 socket, each having four cores, were used. The average thread scalability of the FX1 is superior to that of the HX600, showing the effectiveness of IMPACT. Here, the Fujitsu automatic parallelism compiler without IMPACT features was used with the HX600.

P5 is a list access program, and its performance is limited by the memory bandwidth. Note that the FX1 and the HX600 have different L2 cache replacement algorithms. The FX1 has a conventional inclusive cache with an L2 cache of 6MB; the HX600 has an 8MB victim cache (6MB L2 cache, 2MB L1 cache). The cache increment of the HX600 improves the thread scalability of P5.

P3 is parallelized using XPFortran, and demonstrates a great deal of data communication between nodes as compared to other programs. Thus, the thread scalability of P3 is relatively lower than that of other programs.

5.3 Multi-node performance

Table 8 lists the results for the JBP scale-up problem, showing the execution time on a single node and on multiple nodes. Single-node execution is a single process execution with thread parallelization by IMPACT. The multi-node execution is multi-process execution, whose problem size is enlarged according to the number of processes.

Table 7: Single-node execution results

Code	FX1			HX600 shanghai 2.5GHz		
	1 Core [sec]	4 Core [sec]	Scale ratio	1 Core [sec]	4 Core [sec]	Scale ratio
P1	515.6	149.2	3.46	559.2	216.5	2.58
P2	242.5	80.6	3.01	238.9	93.6	2.55
P3	211.3	93.5	2.26	201.1	146.7	1.37
P4	619.0	172.6	3.59	560.8	171.9	3.26
P5	320.9	134.2	2.39	346.9	127.7	2.72
Ave.	-		2.94	-		2.50

Table 8: Multi-node execution results

Code	Execution on single- node		Execution on multi-node		
	# of grids	Exec time [sec]	# of grids	# of nodes	Exec time [sec]
P1	1,728,000	131.0	1,285,632,000	744	143.3
P2	512,000	71.0	384,000,000	750	91.5
P3	1,572,864	346.8	805,306,368	512	491.7
P4	65,536	164.0	49,152,000	750	193.0
P5	4,173	142.0	2,492,921	750	181.6

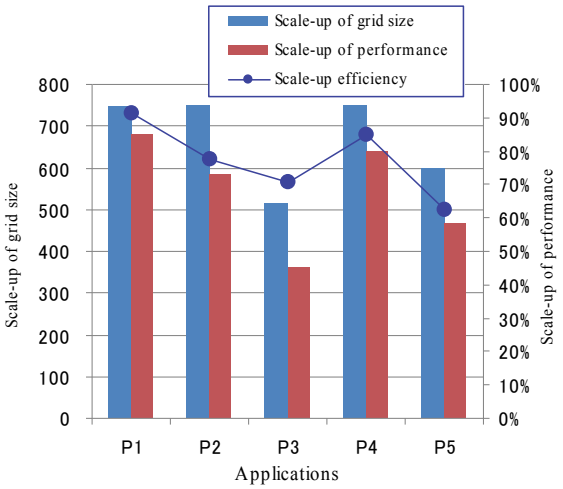


Figure 16: Scalability

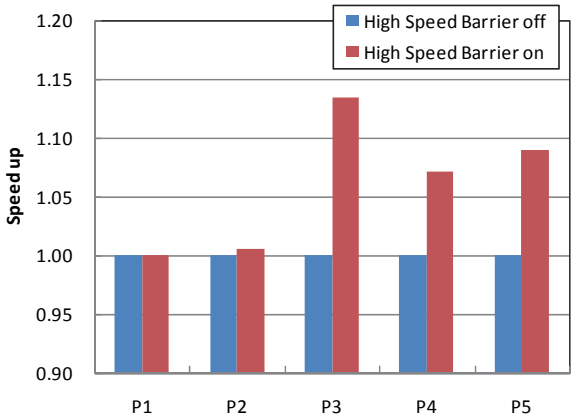


Figure 17: Effects of high-speed barrier network

Figure 16 shows the scale-up effect calculated using the results listed in Table 6. The average value of five JBPs is approximately 77.4%, ranging from 512 processes (2,048 cores) to 750 processes (3,000 cores). P3 and P4, which are data transfer intensive programs, show good efficiency, i.e., 70.5% and 85.0%, respectively. Thus, the good parallel scalability of the FX1 cluster can be identified in actual applications.

Figure 17 shows the advantage of the high-speed barrier network for JBP. Performance with and without the high-speed barrier network was measured and compared. Note that there is no difference in the performance for P1 and P2 because these applications have low barrier synchronization and a small number of reduction operations. The use of the high-speed barrier network improves performance of P3, P4, and P5. This is due to the significant usage of barrier synchronization for P3 and P5, and the significant use of reduction operations for P4. The high-speed barrier network works well for applications with relatively high barrier synchronization and a large number of reduction operations.

Table 9 shows a JBP performance comparison between the FX1 cluster and the CeNSS [3], which was the previous system used at JAXA. The average speed-up ratio of JBP is approximately 11.36. This ratio is greater than 7.75, which is expected by the hardware enhancement, thereby showing the good execution efficiency of the FX1 cluster.

Table 9: Comparison between FX1 and CeNSS

Code	# of sockets	CeNSS [sec]	FX1 [sec]	Performance ratio
P1	744	1380.4	143.3	9.63
P2	750	1468.6	91.5	16.05
P3	512	3517.0	491.7	7.15
P4	750	3061.7	193.0	15.86
P5	750	1447.2	181.6	8.13
Ave.	-			11.36

6. CONCLUSION

To keep up with the trends of multicore processing and increased parallelism, we have installed the JSS, which uses the Fujitsu FX1 multicore-based scalable parallel cluster with IMPACT and high-speed barrier technologies. In this work, through performance measurements of HPL and JAXA's real applications on the M system, which is comprised primarily of the FX1, we were able to confirm the following: 1) effective utilization of the multicore CPU with IMPACT, and 2) high parallel-scalability with the high-speed barrier network.

In future, we will continue to evaluate and enhance the automatic compiler parallelization technology and expand the use of IMPACT to newly developed applications. In addition, we plan to investigate a suitable parallel programming model for petascale scientific applications.

ACKOWLEDGEMENTS

The authors would like to thank the JSS development and operation team at JAXA for their invaluable support. The authors would also like to thank the Fujitsu HPC team for their cooperation.

REFERENCES

[1] Miyoshi, H., Fukuda, M., Iwamiya, T., Nakamura, T., Tuchiya, M., Yoshida, M., Yamamoto, K., Yamamoto, Y., Ogawa, S., Matsuo, Y., Yamane, T., Takamura, M., Ikeda, M., Okada, S., Sakamoto, Y., Kitamura, T., Hatama, H., and Kishimoto, M., "Development and Achievement of NAL Numerical Wind Tunnel (NWT) for CFD Computations," *Proceedings of SC1994,* November 1994.

[2] Matsuo, Y., Nakamura, T., Tsuchiya, M., Ishizuka, T., Fujita, N., Ohkawa, H., Hirabayashi, Y., Takaki, R., Yoshida, M., Nakamura, K., Yamamoto, K., Suematsu, K., and Iwamiya, T., "Numerical Simulator III–Building a Terascale Distributed Parallel Computing Environment for Aerospace Science and Engineering," *Parallel Computational Fluid Dynamics, New Frontiers and Multi-Disciplinary Applications*, Elsevier (2003), 187-194.

[3] Matsuo, Y., Tsuchiya, M., Aoki, M., Sueyasu, N., Inari, T., and Yazawa, K.,"Early Experience with Aerospace CFD at JAXA on the Fujitsu PRIMEPOWER HPC2500", *Proceedings of SC2004*, November 2004.

[4] http://www.hpcresearch.nl/euroben/index.php

[5] Nakashima, H., "T2K Open Supercomputer: Inter-University and Inter-Disciplinary Collaboration on the New Generation Supercomputer," *Intl. Conf. Informatics Education and Research for Knowledge-Circulating Society (ICKS'08)*, Jan. 2008.

[6] http://www.cs.virginia.edu/stream/

[7] http://www.nas.nasa.gov/publications/npb.html

[8] http://icl.cs.utk.edu/hpl/index.html

[9] http://www.top500.org/

[10] http://www.netlib.org/blas/

[11] Mizobuchi, Y., Tachibana, S., Shinio, J., Ogawa, S., and Takeno, T., "A Numerical Analysis of the Structure of a Turbulent Hydrogen Jet Lifted Flame," *Proceedings Combustion Institute*, Vol. 29, (2002), 2009-2015.

[12] Takaki, R., Yamamoto, K., Yamane, T., Enomoto, S., and Mukai, J., "The Development of the UPACS CFD Environment," *Lecture Notes in Computer Science* 2858, Springer (2003), 307-319.

[13] Abe, H., Kawamura, H., and Matsuo, Y., "Direct Numerical Simulation of a Fully Developed Turbulent Channel Flow With Respect to the Reynolds Number Dependence," *Transaction of the ASME, Journal of Fluids Engineering*, Vol. 123, (2001), 382-393.

[14] Shinohara, I., Suzuki, H., Fujimoto, M., and Hoshino, M., "Rapid Large-Scale Magnetic-Field Dissipation in a Collisionless Current Sheet via Coupling between Kelvin-Helmholtz and Lower-Hybrid Drift Instabilities," *Phys. Rev. Lett*. **87**, 095001, 2001.

[15] Murayama, M., and Yamamoto, K., "Comparison Study of Drag Prediction for the 3rd CFD Drag Prediction Workshop by Structured and Unstructured Mesh Method," AIAA Paper 2007-0258, Jan. 2007.