White paper

FUJITSU Supercomputer PRIMEHPC FX1000

An HPC System Opening Up an AI and Exascale Era

Fujitsu Limited

Contents

Overview of the FUJITSU Supercomputer PRIMEHPC FX1000

Introduction
Fujitsu has led the development of supercomputers with the latest technologies for more than 40 years, after developing the first supercomputer in Japan in 1977. The FUJITSU Supercomputer PRIMEHPC FX1000 (or simply PRIMEHPC FX1000) is a state-of-the-art supercomputer that makes the exascale computing achieved by the supercomputer "Fugaku" more accessible, opening up an AI and exascale era.

High-performance design for HPC and AI areas
The PRIMEHPC FX1000 is a massively parallel computer equipped with the A64FX processor, which has been designed by Fujitsu for HPC and AI areas. The A64FX employs an Arm architecture that is widely used in smartphones and other such devices. The processor is also the first in the world to implement the Scalable Vector Extension (SVE), the vector extension for HPC and AI areas of Armv8-A instruction sets. The A64FX CPU chip has 48 compute cores and 4 assistant cores along with 4 stacks of the 3D stacked memory High Bandwidth Memory 2 (or simply HBM2) mounted as the main memory in the same package. The Tofu interconnect D (or simply TofuD) incorporated in the A64FX connects nodes to one another with 20-lane high-speed signals to construct a system in a highly scalable 6D mesh/torus configuration.

Highly reliable direct water cooling
The A64FX is mounted in the CPU memory unit (CMU), which prevents the semiconductor temperature from rising by circulating chilled water across cold plates to cool the A64FX, optical transceivers, and DC voltage converters. Maintaining a low temperature keeps the component failure rate low.

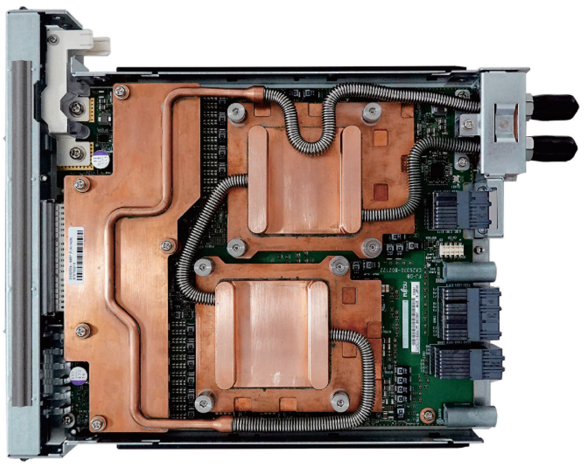


Figure 1 PRIMEHPC FX1000 CPU memory unit

Main unit and rack configuration
The PRIMEHPC FX1000 main unit can accommodate 24 CMUs, 3 boot disks, 3 service processors for system monitoring, 6 low-profile PCI Express expansion slots, and 12 power supply units. Up to 4 main units are mounted at each of the front and rear of a rack. The maximum number of nodes per rack is 384.

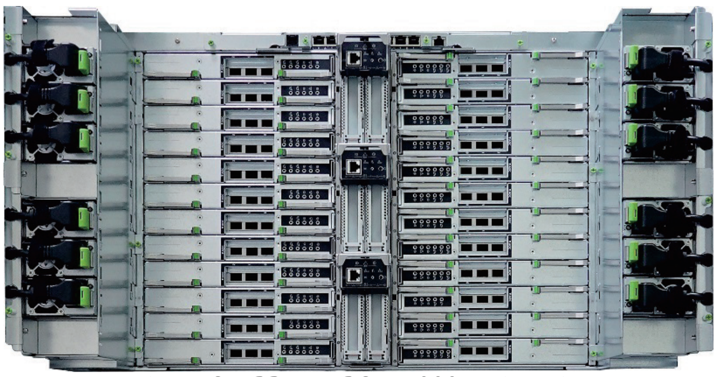


Figure 2 PRIMEHPC FX1000 main unit

System configuration
The peak performance of the PRIMEHPC FX1000 is 1.297 petaflops per rack. The maximum configuration is 1,024 racks, and the peak performance of 1.328 exaflops.

Table 1 PRIMEHPC FX1000 system specifications

	1-rack configuration	Maximum configuration
Number of racks	1	1,024
Number of main units	8	8,192
Number of nodes	384	393,216
Peak performance		
Double precision	1.297 Pflops	1.328 Eflops
Single precision	2.595 Pflops	2.657 Eflops
Half precision	5.190 Pflops	5.315 Eflops
Memory capacity	12 TiB	12 PiB
Memory bandwidth	393 TB/s	402 PB/s
Interconnect bandwidth	31 TB/s	32 PB/s
Number of PCIe expansion slots	48	49,152
Connection topology	2x4x4x2x3x2 2x2x8x2x3x2	32x32x32x2x3x2

Arm Processor A64FX for HPC and AI Areas

A64FX overview
The A64FX CPU chip is manufactured using 7 nm process technology, and each chip contains approximately 9 billion transistors. Figure 3 is a photograph of the CPU chip.
The CPU chip has 48 compute cores and 4 assistant cores (52 cores in total). The total peak performance of the 48 compute cores is 3.3792 Tflops for double-precision floating-point operations. The A64FX CPU chip is equipped with 4 pairs of HBM2 input and output interfaces, a TofuD interface, and a PCIe interface.

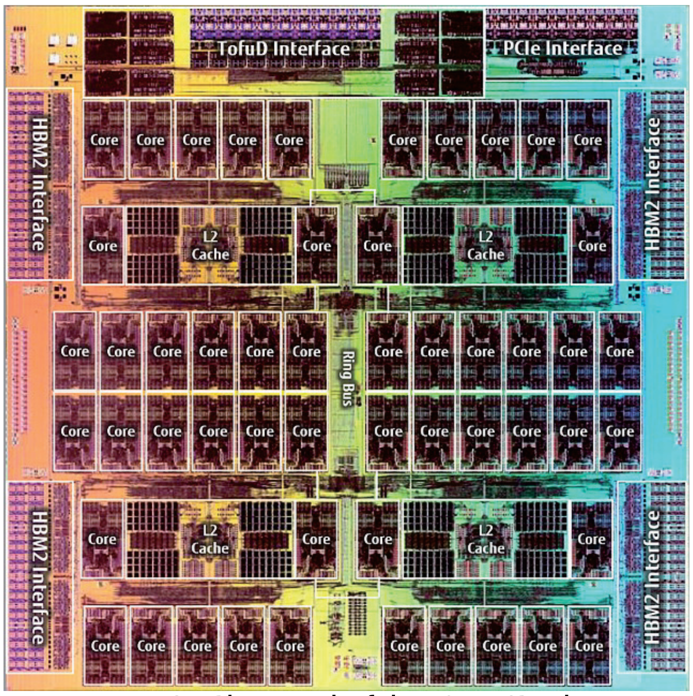


Figure 3 Photograph of the A64FX CPU chip

Scalable Vector Extension
The A64FX implements SVE for Armv8-A instruction sets. The SIMD bit width has been extended to 512 bits, which is twice that of the SPARC64™ XIfx. Half-precision floating-point numbers (FP16), which are used for AI, are now supported. SVE details are provided on pages 4 and 5.

Assistant core
The assistant cores mainly execute OS and I/O processing to reduce OS jitter on the compute cores. Reducing OS jitter, which is a cause of latency in collective communication, improves the efficiency of parallel processing.

Core Memory Group
A very important issue for a many-core processor like the A64FX is how to connect cores, caches, and memory. The A64FX employs an approach called Core Memory Group (CMG), which divides cores into four groups. A single CMG consists of 12 compute cores, 1 assistant core, an L2 cache, and a memory controller. Cache consistency is maintained among the four CMGs. The system software can handle the CMGs as NUMA nodes.

Heterogeneous integration of CPUs and memory
The A64FX heterogeneously integrates a CPU chip and the 3D stacked memory HBM2 into a single package using 2.5D packaging technology to provide a high theoretical bandwidth of 1,024 GB/s. Details of the heterogeneous integration of the CPU and memory in the A64FX are provided on page 6.

Built-in Tofu interconnect D
The TofuD built into the CPU is Fujitsu's proprietary interconnect, resulting in a massively parallel system driven by the A64FX. TofuD details are provided on pages 7 and 8.

I/O connections
The A64FX is equipped with 16 lanes of the standard PCIe Gen3 interface for I/O connections.

Table 2 A64FX specifications

Number of cores	Compute cores	48
	Assistant cores	4
Peak performance		
Double precision		3.3792 Tflops
Single precision		6.7584 Tflops
Half precision		13.5168 Tflops
L2 cache capacity		32 MiB
Memory capacity		32 GiB
Theoretical memory bandwidth		1,024 GB/s
Theoretical interconnect bandwidth		68 GB/s x 2 (in/out)
Theoretical I/O bandwidth		15.75 GB/s x 2 (in/out)
Process technology		7 nm CMOS FinFET
Number of transistors		Approximately 9 billion

Scalable Vector Extension for Arm Instruction Sets

SVE overview

SVE extends scalable vector operations to Armv8-A instruction sets. As a lead partner of Arm, Fujitsu worked collaboratively to develop the SVE specifications. Conventionally, the defined SIMD bit width supported by Armv8-A for vector operations is 128. In contrast, the SIMD bit width supported by SVE is in a range of 128 to 2,048, depending on the hardware implementation. The A64FX supports 512-, 256-, and 128-bit wide operation modes. The data types supported by SVE include not only common double-precision floating-point numbers and single-precision floating-point numbers but also half-precision floating-point numbers (FP16), which accelerates deep learning. Also supported are 16- and 8-bit integer vector operations, and inner product operation instructions are used to accelerate inference in deep learning. SVE supports the same instruction types as the SPARC64, that is, 4-operand FMA (Floating-point fused Multiply-Add) instructions, the Gather/Scatter instruction, math function acceleration instructions, and Predicate operations. In addition, loops whose count is unknown can be converted into SIMD in advance by the newly introduced First Fault Load instruction.

SVE register structure

SVE includes 32 scalable vector registers. The lower 128 bits of the scalable vector registers are shared with the Armv8-A SIMD & FP registers. In addition to the scalable vector registers, SVE includes 16 predicate registers. The bit width of the predicate registers is one-eighth that of the scalable vector registers. Figure 4 illustrates the registers.

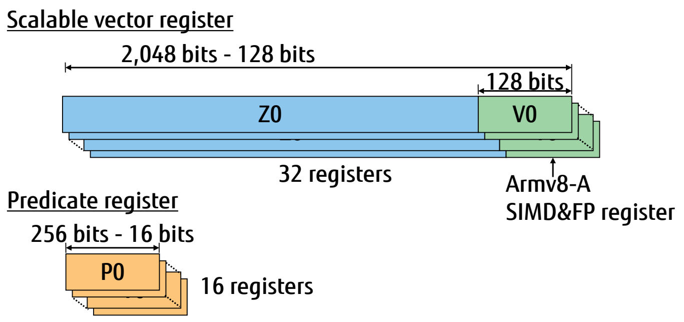
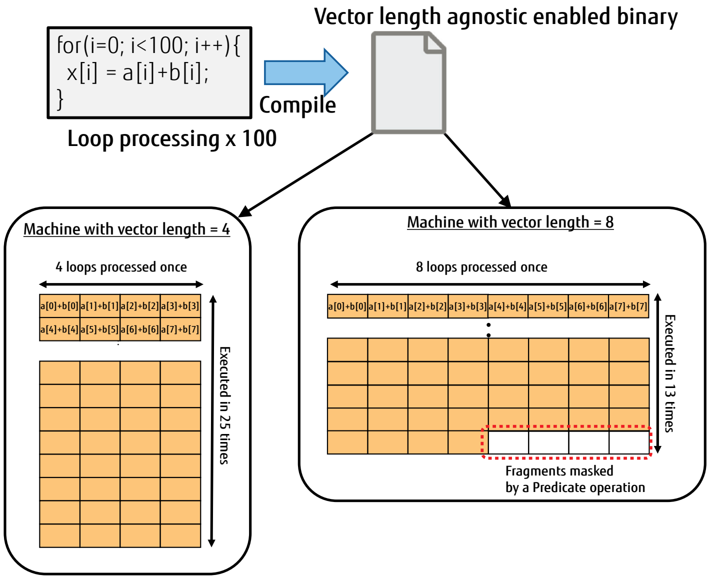


Figure 4 SVE register structure

Vector length agnostic

SVE makes it possible to create binaries that can operate without dependence on the SIMD bit width implemented by hardware. This feature is called vector length agnostic. Binaries that are vector length agnostic can be executed without recompilation, even on an SVE machine implementing a different SIMD bit width.

As shown in Figure 5, in cases where a program that executes a loop 100 times is compiled into binaries independent of the vector length, the number of elements (vector length) is calculated by an instruction, and a code that adjusts the loop count is generated. The calculation is based on the SIMD bit width of the machine. That code loops 25 times on a machine with a vector length of 4 and 13 times on a machine with a vector length of 8. If the original loop count is not a multiple of the vector length, fractional elements are masked by a Predicate operation.



```
for(i=0; i<100; i++){
  x[i] = a[i]+b[i];
}
```

Vector length agnostic enabled binary
Compile
Loop processing x 100
Machine with vector length = 4
4 loops processed once
a[0]+b[0] a[1]+b[1] a[2]+b[2] a[3]+b[3]
a[4]+b[4] a[5]+b[5] a[6]+b[6] a[7]+b[7]
Executed in 25 times
Machine with vector length = 8
8 loops processed once
a[0]+b[0] a[1]+b[1] a[2]+b[2] a[3]+b[3] a[4]+b[4] a[5]+b[5] a[6]+b[6] a[7]+b[7]
Executed in 13 times
Fragments masked by a Predicate operation

Figure 5 Conceptual image of vector length agnostic

Data formats and instruction types

SVE provides double-, single-, and half-precision floating-point numbers of operations across most floating-point instructions.. Even when the data format changes according to the required precision, operations can be performed with the same instruction type. Therefore, operation throughput is easier to predict. Figure 6 shows the supported floating-point number formats.

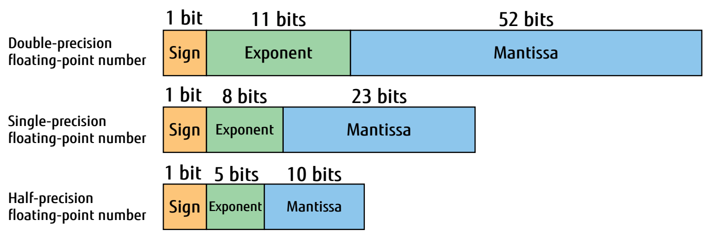


Figure 6 Supported floating-point number formats

SIMD integer operations are also substantial. In addition to 64- and 32-bit operations, 16- and 8-bit operations can be performed too. Among them, 16- and 8-bit operations support the dot product operations effective for inference processing.

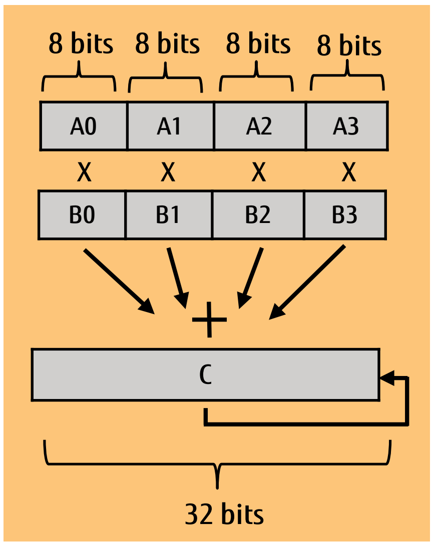


Figure 7 Dot product operation on 8-bit integers

Like on the SPARC64, the math function acceleration instructions of the SVE support trigonometric and exponential functions.

SIMD conversion of do-while loops/break syntax

The newly introduced First Fault Load instruction and FFR register in SVE make it easier to convert into SIMD the do-while loops/break syntax for exiting a loop through in-loop processing.
Usually, when a loop structure is converted into SIMD, the multiple times of in-loop processing written in the program are batch processed by the SIMD operation. However, if the loop count depends on the in-loop processing, any attempt at forced conversion into SIMD may forcibly terminate the application because a fault would occur. This is due to memory access exceeding the data area used by the program, as shown in Figure 8.

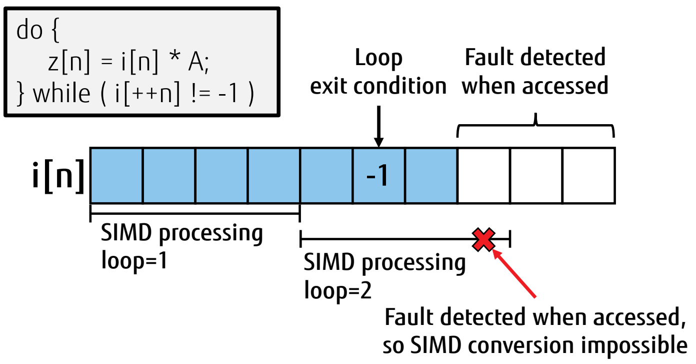


Figure 8 Difficulty of converting a Do-while loop into SIMD

The First Fault Load instruction introduced in SVE can be used to address this problem. When a fault in memory access is detected, the First Fault Load instruction performs a different operation according to the element with the detected fault. If the fault is detected in the first element of the SIMD operation, the instruction actually generated the fault. If the fault is detected in any other element, the instruction records the fault detection information in the FFR register and suppresses memory access to the subsequent elements after the one with the detected fault. The information registered in the FFR register can be used for masking by a Predicate operation. Figure 9 shows an example of SIMD conversion using the First Fault Load instruction. The loop can be properly processed by reading the FFR register information recorded by the First Fault Load instruction and using it for a Predicate operation.

```
        ptrue p0.d
        ld1rd z1.d, p0/z, [x2]
.loop:
        setffr
        ldff1d z0.d, p0/z, [x1, x3, lsl #3]
        rdffr p1.b, p0/z
        cmpeq p2.d, p1/z, z0.d, #-1
        brkbs p2.b, p1/z, p2.b
        mul z0.d, p2/M, z0.d, z1.d
        st1d z0.d, p2, [x0, x3, lsl #3]
        incp x3, p2.d
        b.last .loop
```

Figure 9 Example of SIMD code converted by First Fault Load

Gather/Scatter instruction

HPC applications may read and write non-contiguous data by using an index stored in an integer array to indirectly reference another array. SVE supports the Gather/Scatter instruction to accomplish such access. In the following example of the Gather instruction, the integer register X0 contains the first address of a referenced array. Also, Z1 in the scalable vector register contains the index to the referenced array, and P0 in the predicate register contains a load mask. Thus, the data to be indirectly referenced can be gathered and stored in the register only for the elements for which P0 is 1.

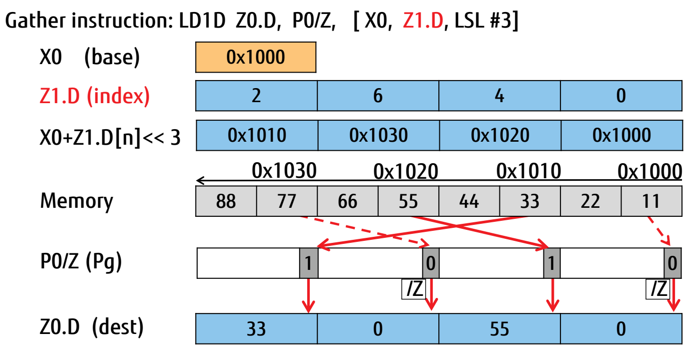


Figure 10 Gather instruction operation

Heterogeneous Integration of CPUs and 3D Stacked Memory HBM2 Using 2.5D Packaging Technology

Heterogeneous integration of CPUs and memory

The IC packages implemented in the PRIMEHPC FX100 consist of a CPU chip alone in an IC package and 3D stacked memory in different IC packages. The CPU package and eight 3D stacked memory packages are connected by wiring on the printed circuit board (PCB).
In contrast, the A64FX for the PRIMEHPC FX1000 heterogeneously integrate a CPU chip and 3D stacked memory into a single package using 2.5D packaging technology. Figure 11 is a photograph of the A64FX 2.5D package. The CPU chip and four HBM2 stacks are closely arranged and connected by high-density and fine-pitch wiring.

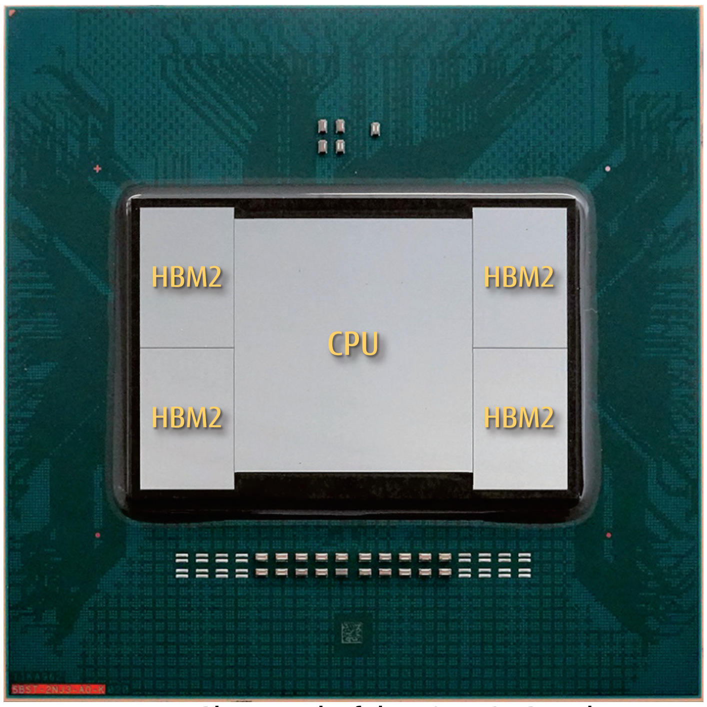


Figure 11 Photograph of the A64FX 2.5D package

2.5D packaging technology

The CPU chip and four HBM2 stacks of the A64FX are mounted on a silicon interposer (or simply Si-IP).
Figure 12 is a schematic sectional diagram that shows the structure of the 2.5D package. The Si-IP is a silicon substrate with wiring layers and through-silicon vias (TSVs), and no transistors are formed. The Si-IP is joined to the package substrate using interconnect technologies such as Cu pillars and C4 bumps. The CPU chip and HBM2 are connected to the Si-IP wiring layer by micro-bumps with a pitch of 40 to 55 μm. The package substrate is connected to the PCB by solder ball terminals called Ball Grid Arrays (BGAs) that have a pitch of 1 mm. From a comparison of the pitches of the micro-bumps and the BGAs, the Si-IP wiring is about 20 times denser than the PCB wiring.

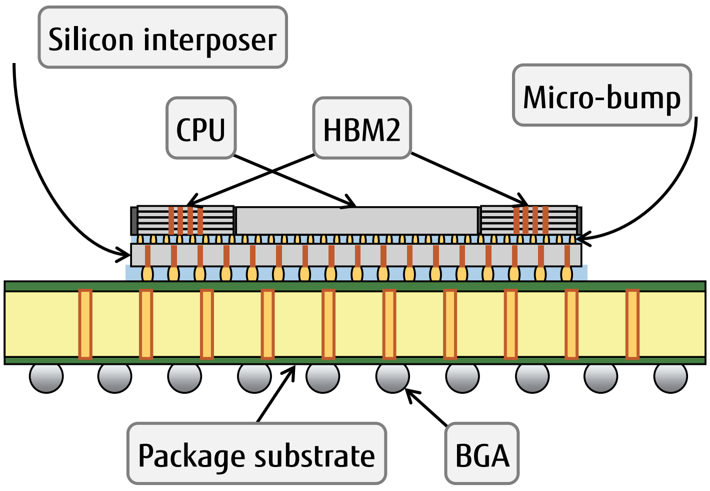


Figure 12 2.5D package structure (schematic sectional diagram)

3D stacked memory HBM2

The HBM2 used in the A64FX is 3D stacked memory for 2.5D implementation. The HBM2 stacks up to eight memory dies where TSVs have formed, and has eight independent 128-bit wide channels.
Table 3 shows the specifications of the main memory in the A64FX using four HBM2 stacks. The data signal transfer rate of the HBM2 is 2.0 Gbps, and the memory bandwidth per 128-bit wide channel is 32 GB/s. The memory bandwidth of a single stack is 256 GB/s, which is the total with the eight channels, and the memory capacity is 8 GiB. Therefore, the memory bandwidth of the A64FX using four HBM2 stacks is 1,024 GB/s, and the memory capacity is 32 GiB.

Table 3 A64FX main memory specifications

Memory bandwidth		1,024 GB/s
Memory capacity		32 GiB
Number of HBM2 stacks per package		4
HBM2	Data signal transfer rate	2.0 Gbps
	Data width	1,024 bits
	Memory bandwidth	256 GB/s
	Memory capacity	8 GiB

Tofu Interconnect D for High-Density Systems

TofuD overview
The TofuD is the interconnect used in the A64FX to construct a massively parallel system with more than 100,000 nodes. Based on the Tofu interconnect 2 (or simply Tofu2) for the PRIMEHPC FX100, the TofuD has improved functions for high-density systems and enhanced fault tolerance.

6D mesh/torus network
The TofuD configures a massively parallel system with more than 100,000 nodes using a 6D mesh/torus network. Originally developed for the K computer, this network continues to be used for the Tofu2 and TofuD. Figure 13 shows a model of the 6D mesh/torus topology. The lengths of the X, Y, and Z axes in the six dimensions may vary depending on the system configuration. The lengths of the other axes (A, B, and C) are fixed at 2, 3, and 2 (respectively). Because nodes are interconnected in 6 dimensions, each node is equipped with 10 ports for connections.
A 1-dimensional/2-dimensional/3-dimensional virtual torus represents the network topology in the view for a user. The user specifies the number of dimensions and size of the virtual torus space to map the space to a 6D mesh/torus network and reflect rank numbers in the space. This virtual torus method improves system fault tolerance and availability because an area containing a faulty node remains available as a torus.

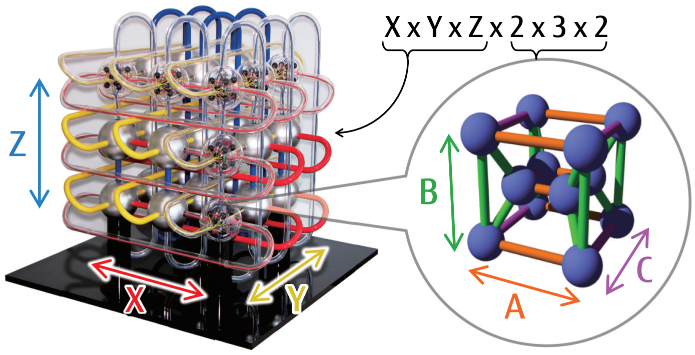


Figure 13 6D mesh/torus topology model

High-density system configuration
The TofuD interconnects nodes with ten 2-lane links, using electric transmission links between nodes in the same unit group and optical transmission links between nodes in different unit groups. The greater the number of nodes housed in a unit group, the greater the electric transmission ratio, which has a lower cost. The PRIMEHPC FX1000 is a high-density system using electric transmission links to connect up to 192 nodes mounted in 4 main units. The 2 nodes in the CMU are interconnected with the C axis. The 42 nodes in each main unit are interconnected with the Z, A, and B axes. The 4 main units are interconnected with the X and Y axes. The 6D structures of the CMU, each main unit, and the four main units are (X,Y,Z,A,B,C)=(1,1,1,1,1,2), (1,1,4,2,3,2), and (2,2,4,2,3,2), respectively. Interconnections through optical transmission use the X and Y axes for half of all the nodes, and the X, Y, and Z axes for the other half. Figure 14 shows how active optical cables (AOCs) are connected to a CMU. The number of 4-lane AOCs required per node for a TofuD 6D network structure is 0.625.

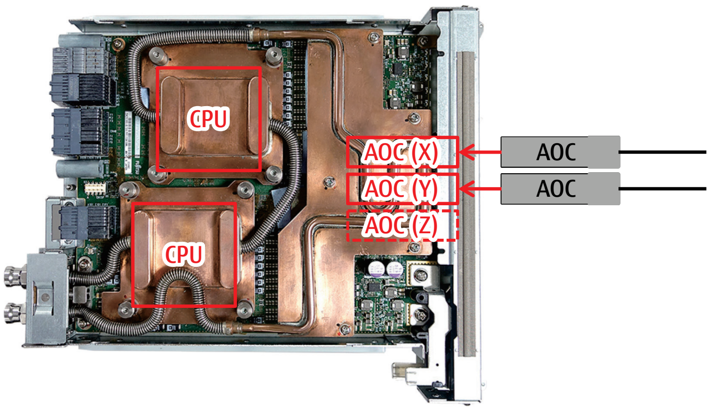


Figure 14 AOC connections to a CMU

Six RDMA engines
A torus is a network with a strong locality that prevents communications between adjacent nodes from interfering with each other. Concurrent communication with multiple RDMA engines makes it possible to utilize the locality of a torus network.
Four RDMA engines are implemented for the K computer and Tofu2, which utilize the locality of a network for high-speed collective communication using a dedicated communication library. The number of RDMA engines has increased to 6 for the TofuD in line with a significant improvement in memory bandwidth owing to the heterogeneous integration of CPUs and memory. This allows high-speed collective communication even in the supercomputer Fugaku, which has more than 100,000 nodes.

Tofu barrier enhancement
Jointly implemented in RDMA engines, the Tofu barrier is hardware for handling barrier synchronization and AllReduce collective communication with low latency. The K computer and Tofu2 assume that the Tofu barrier is used by one process per node and implemented in only one RDMA engine. In the A64FX, however, the number of CMGs has increased to 4, and the number of processes in a node is 4 or greater. This means more

opportunities for multiple processes in a node to collectively communicate using different communicators. Therefore, to increase Tofu barrier communication resources, the TofuD has incorporated the Tofu barrier in all six RDMA engines. In addition, the number of Tofu barrier channels per RDMA engine has increased from 8 communicators to 16 communicators. The AllReduce operations that can be executed at the same time as barrier synchronization have also been enhanced. The number of elements that can be reduced at one time has increased from 1 to 3 for floating-point numbers and from 1 to 8 for integers. In addition, a newly supported operation is MAXLOC for four elements.

Dynamic packet slicing

Since a massively parallel computer consists of an enormous number of parts, fault tolerance allowing system operation to continue even when a fault or failure occurs is important. The fault-tolerance function of the Tofu2, upon detecting a failure in a specific lane of a link, reduces the number of link lanes to continue communication.
A dynamic packet slicing function has been developed for the TofuD as a more advanced fault recovery function. Figure 15 and Figure 16 show dynamic packet operations. In split mode, which is normal operation, the sender divides a packet into slices and transmits them simultaneously in two lanes. Then, the receiver handles error detection in individual lanes separately and reports the error frequency to the sender. If the error frequency is high, a fault is assumed to have occurred, so the sender enters duplicate mode, where a packet is sent to two lanes without being divided. The duplicate mode of dynamic packet slicing reduces the effective bandwidth to half that of lane reduction with Tofu2, but it is different in that error detection in each lane continues. When the reported error frequency drops, dynamic packet slicing changes from duplicate mode back to split mode.

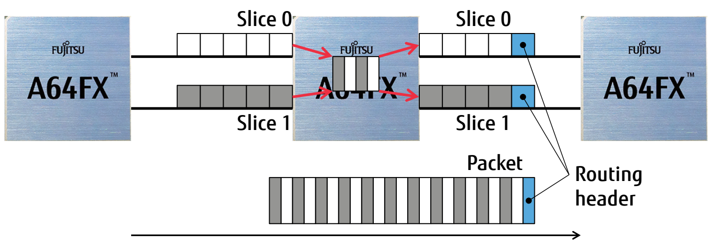


Figure 15 Dynamic packet slicing: Split mode

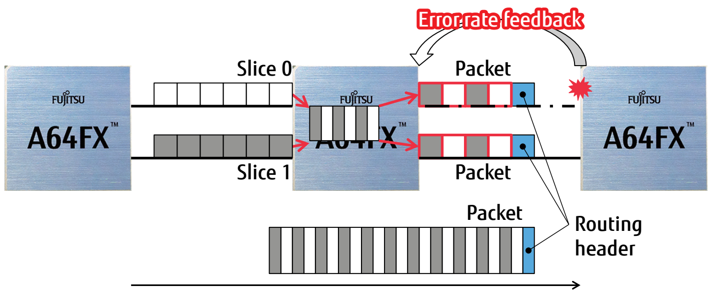


Figure 16 Dynamic packet slicing: Duplicate mode

Low-latency communication

The RDMA Put communication latency of the TofuD is 0.49 µs (shortest). This is shorter by 0.22 µs for the Tofu2 and by 0.42 µs for the K computer. Figure 17 shows a breakdown of the latency. The latency is lower because the Tofu2 introduced cache injection and integration into the CPU (so buses were deleted), and because the TofuD no longer needs to deskew between lanes in the physical coding layer. One-hop latency is approximately 80 µs.

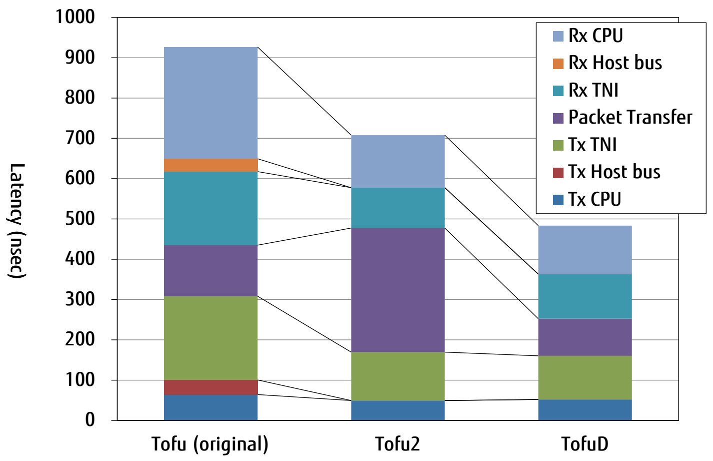


Figure 17 Breakdown of communication latency

Table 4 Tofu interconnect D specifications

Data transfer rate	28.05 Gbps
Encoding	64b/66b
Number of lanes per link	2
Link bandwidth	6.8 GB/s
Injection bandwidth	40.8 GB/s
Number of ports connected per node	10
Network topology	6D mesh/torus
Routing method	Extended dimension order
Number of virtual channels	4
Maximum packet length	1,984 bytes
Packet transfer method	Virtual cut-through
Flow control method	Credit-based
Delivery guarantee method	Link-level retransmission
RDMA communication function	Put/Get/Atomic RMW
Number of RDMA engines	6 (Concurrent communication is possible.)
Number of CQs per RDMA engine	12 pairs
Address translation method	Memory Region + Page Table
Number of Tofu barrier channels	96
Communication protection method	Global process IDs
Operating frequency	425 MHz

Reference
For more information about the PRIMEHPC FX1000, contact our sales personnel or visit the following website:

https://www.fujitsu.com/global/products/computing/servers/supercomputer/index.html

FUJITSU Supercomputer PRIMEHPC FX1000: An HPC System Opening Up an AI and Exascale Era
Fujitsu Limited
First Edition November 12, 2019
2019-11-12-EN