The Tofu Interconnect

The Tofu interconnect uses a 6D mesh/torus topology in which each cubic fragment of the network has the embeddability of a 3D torus graph, allowing users to run multiple topology-aware applications. This article describes the Tofu interconnect architecture, the Tofu network router, the Tofu network interface, and the Tofu barrier interface, and presents preliminary evaluation results.

Yuichiro Ajima
Tomohiro Inoue
Shinya Hiramoto
Toshiyuki Shimizu
Fujitsu

Yuzo Takagi
DeNA

Fujitsu developed the Tofu interconnect as a national project of Japan's Ministry of Education, Culture, Sports, Science, and Technology (MEXT).[1] The project aims to develop one of the world's most powerful general-purpose supercomputer systems and grand challenge applications, and to build a core research hub for computational science (see http://www.nsc.riken.jp/index-eng.html). The Tofu interconnect provides the scalability to implement the 10-petaflops K computer system, which has more than 80,000 nodes. Several research groups will share the system, which will occasionally be totally occupied by a grand challenge job. Therefore, Tofu's network topology and features work together to ensure the effectiveness of the same distributed parallel algorithm on debug partitions, production partitions, and grand challenge mode.

The Tofu interconnect transfers both message passing interface (MPI) and I/O data. The network topology is a 6D mesh/torus. Each cubic fragment of the 6D mesh/torus network has the embeddability of a 3D torus graph so users can run multiple topology-aware applications. Existing highly scalable interconnects such as the Blue Gene torus[2] and Gemini[3] also use torus topology. The Blue Gene torus uses a partition switch to run multiple topology-aware applications, whereas Gemini has no special feature. Gemini constructs a single network with high-bandwidth links, whereas Blue Gene divides bandwidth into two networks: torus and tree. The Tofu interconnect is designed to provide a single network with high-bandwidth links and to run multiple topology-aware applications.

Results from preliminary evaluations show the effectiveness of the technologies used in the Tofu interconnect. The Tofu interconnect's one-hop look-ahead scheduling enhances the effective switching throughput; its direct descriptor technique reduces the command overhead by 0.24 μs; and its quad-rail communications approach improves the effective single-node throughput of random permutation traffic.

Design issues

Tofu is the main component of the InterConnect Controller (ICC) ASIC.[4] As Figure 1 shows, each ICC chip has a Tofu network router (TNR), four Tofu network interfaces (TNIs), a Tofu barrier interface (TBI), a host bus interface, and two PCI Express root complexes. The TNR provides 10 links to construct the 6D mesh/torus network: six in three scalable axes (*X*, *Y*, and *Z*), and four in three fixed-size axes (*A*, *B*, and *C*). The TNIs are data transfer interfaces that are compactly designed with fully wired logic to integrate the quad interfaces into a chip. The TBI provides offload capability

0272-1732/12/$31.00 © 2012 IEEE Published by the IEEE Computer Society

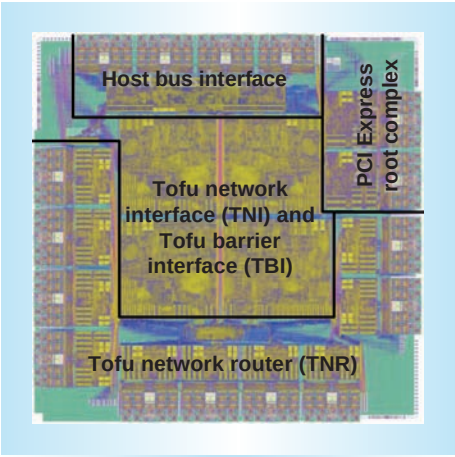

Figure 1. A micrograph of the ICC chip. The chip integrates all active components of the Tofu interconnect: a Tofu network router (TNR), four Tofu network interfaces (TNIs), a Tofu barrier interface (TBI), a host bus interface, and two PCI Express root complexes.

for synchronization. The host bus connects a Sparc64 chip[5] to the Tofu interconnect and PCI Express devices.

Figure 2a shows the ICC chip structure and interconnections. Four Sparc64 chips on a board are interconnected with the *A*- and *C*-axes links. Three boards in a Tofu unit are interconnected with the *B*-axis links. Tofu units are interconnected with the *X*-, *Y*- and *Z*-axes links that form a 3D torus. The *Z*-axis links connect 17 Tofu units in two racks: 16 for computing nodes, and one for I/O nodes. The *X*-axis and *Y*-axis are expandable according to the number of columns and rows of racks.

Figure 2b shows a topological model of the *ABC* 3D mesh/torus forming a Tofu unit. In the event of a single-board failure that decreases the *B*-axis's length, the 3D torus graph's embeddability is unaffected because the *ABC* 3D topology remains cubic.

One of the big challenges in building the K computer was system reliability. For instance, a mean time between failures (MTBF) of five years per node, assuming commodity processing nodes, would bring about two failures every hour in the 80,000-node system. We needed about one-hundredth of that failure rate. To minimize the failure rate, we integrated all active components of the Tofu interconnect into a single ICC chip, protected major data paths using error-correction code, and

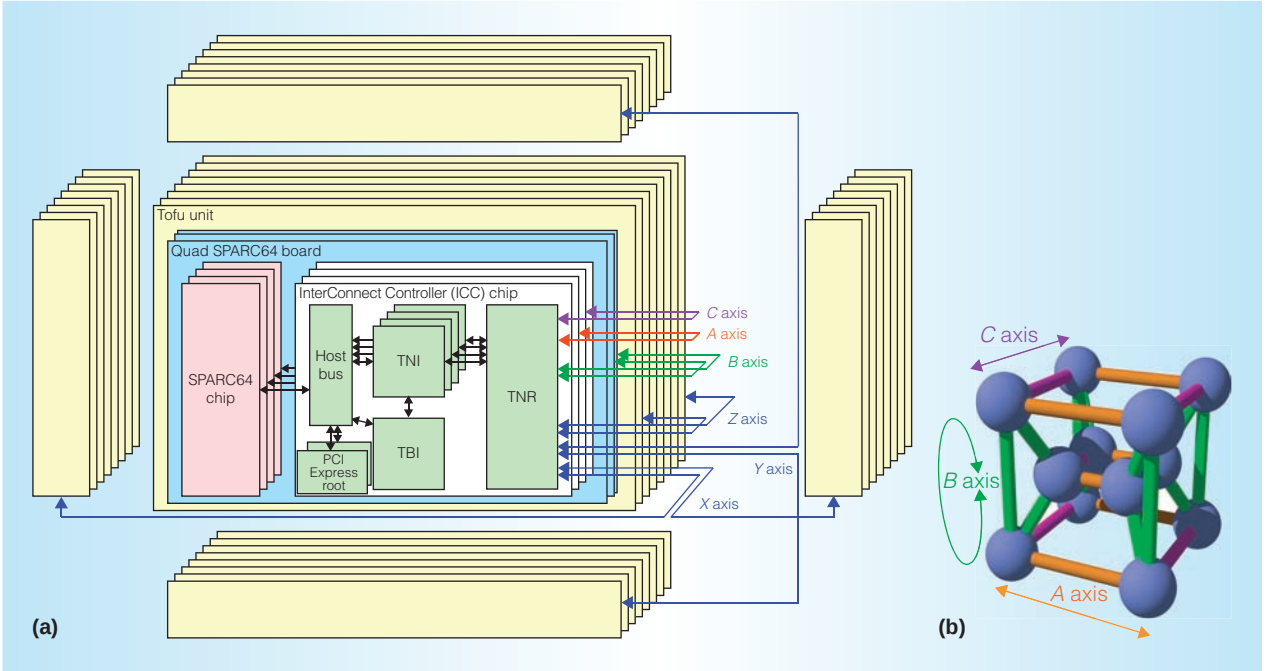

Figure 2. The ICC chip structure and interconnections along six axes (a). A topological model of the *A*-, *B*-, and *C*-axes (b).

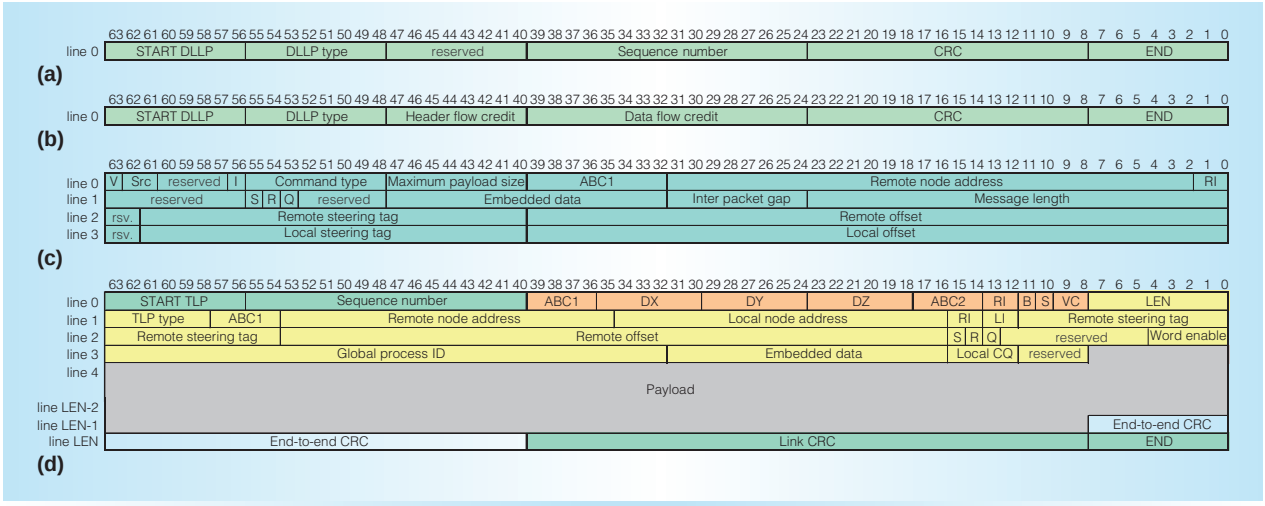

Figure 3. Examples of packet and command format in the Tofu interconnect: an ACK data-link-layer packet (DLLP) (a), an UpdateFC (updated flow control) DLLP (b), a Put command (c), and a Put transport-layer packet (TLP) (d).

water-cooled the chip. The ICC chip requires no external retimers or optical modules and uses passive electrical cables for the interconnections. The water-cooling system keeps the ICC's junction temperature at 50°C lower than a typical air-cooled condition. Thermal acceleration of semiconductor device failure mechanisms can be expressed using the Arrhenius equation for physico-chemical reaction rates,[6] meaning that lower temperatures reduce the failure rate.

One scalability issue for massive parallel applications is hotspot contention, which is a concern especially along *XYZ* scalable axes for the Tofu interconnect. Hotspots get worse when the utilization of links or virtual channels is unbalanced. The Tofu interconnect uses features that disperse traffic and improve utilization balance. Quad-rail network interfaces and a multipath routing algorithm exploit path diversity, and a novel virtual-channel scheduling algorithm uses multiple virtual channels effectively. The routing algorithm and the virtual-channel scheduling algorithm are deterministic to simplify end-to-end protocols and let us design high-throughput and low-latency TNI pipelines with wired logic.

The ICC chip is implemented in the Fujitsu 65-nm process. The chip is 18.2 mm × 18.1 mm in size with 48 million logic gates, 11.7-Mbit static RAM (SRAM) cells, and 128 lanes of differential I/O signals. Each ICC chip uses 80 lanes in 10 Tofu links, 32 lanes in the host interface, and 16 lanes in two PCI Express complexes. Each Tofu link has eight signal lanes with a 6.25 gigabits per second (Gbps) bit rate, and 5 gigabytes per second (GBps) raw bandwidth per direction. The TNI, TBI, and TNR operate at 312.5 MHz.

Tofu interconnect architecture

The Tofu interconnect uses a layer model that includes physical, data link, network, and transport layers. The TNR implements the physical- data-link-, and network-layer functionality and protocols. The TNIs and TBI implement transport-layer protocols and user interface functionality.

Data link layer

The data link layer provides reliable data transfer on links to the upper layer. The data-link-layer protocol detects packet loss or data error and retransmits failed packets. Each sender side of a link has an 8-Kbyte buffer for retransmission. To detect packet loss or data error on the receiver side, each upper-layer data payload is enveloped between a sequence number and a cyclic redundancy check (CRC) code.

A data-link-layer packet (DLLP) transports a control message to the other end of a link. When the receiver side receives data with a valid sequence number and CRC code, it sends an ACK DLLP to the sender side. Figure 3a illustrates the packet format.

If it detects packet loss or data error, the receiver side sends a negative acknowledgement (NACK) DLLP to the sender side to request retransmission. DLLPs also transport other control messages, such as link-initialize control and upper-layer flow control.

Network layer

The network layer delivers a packet from a source TNI to a destination TNI via the Tofu network. The routing algorithm has three stages. The first stage goes through the *ABC* axes; the second traverses the *XYZ* axes; and the last travels through the *ABC* axes again. Each routing stage uses fixed-order minimal routing. The routing algorithm is oblivious, and a routing header contains two sets of *ABC* coordinates. The extra *ABC* axis routing stage provides path diversity.

The network layer provides four virtual channels. Two of these are for the two traversals on the *ABC* axes and deadlock avoidance on the *XYZ* axes. The other two are for the separation of request packets and response packets. The flow control of a virtual channel is credit based. Each channel has an independent 8-Kbyte receive queue and the space in the queue is notified to the sender side ahead of packet transfer on the link. This notification uses an UpdateFC (update flow control) DLLP. Figure 3b shows the packet format.

Transport layer

The transport layer delivers end-to-end communication. The TNIs provide remote direct memory access (RDMA) and system packet communication. The TBI provides Tofu barrier communication.

An RDMA Put transfers data from the local node to the remote node, and an RDMA Get transfers data from the remote node to the local node. Each TNI has three sets of control queues (one kernel and two user) to handle RDMA communication. Each control register set of user control queues maps to a user process address space independently. The host processor enqueues a 32-byte command to start a Put or Get transfer. Figure 3c shows the format of a Put command. A single command transfers a message with a maximum size of 16 Mbytes. The TNI segments the message into TLP payloads, which have a maximum length of 1,920 bytes. The interpacket gap field, which controls the transmission rate of TLPs, is used for static congestion control of uniform communication patterns. Figure 3d shows the packet format of the Put TLP. The S flag indicates that the payload should be written after all preceding out-of-order memory accesses. Functionalities and benefits of the interpacket gap and the S flag are described elsewhere.[4]

System packet communication is used for TCP over Tofu communication and network booting of a node. Kernel control queues can transmit system packets. The TNI has a set of packet queues that receive the system packets.

The barrier channel (BCH) handles Tofu barrier communication. The TBI has eight BCHs. Each control register set of BCHs is independently mapped to a user process address space.

Virtual address space

Each TNI provides 4 million 40-bit virtual address spaces for RDMA communication. Each virtual address space is designed to map code, data, heap, and stack memory structures separately. Each address space is identified by a 22-bit steering tag (STag). The global memory location is addressed by a 19-bit node address, a 2-bit interface number, a 22-bit STag, and a 40-bit offset. The TNI internally uses a memory block structure to trace host-side memory paging. Each memory block has its own page table, which resides in the main memory. A virtual address space is a subspace of a memory block.

Tofu network router

The TNR consists of a 15-port crossbar and 10-link control modules, as Figure 4 illustrates. The crossbar is constructed from a 5×3 array of 5:3 crossbars. Ten Tofu links and four TNIs connect to the crossbar ports. Each port has a bandwidth of 5 GBps. A TNR's total switching capacity is 70 GBps.

To reduce routing latency, a link-control module forwards a packet before that packet is completely received. If a link-control module detects an error in a packet, it marks the packet as bad and asks the sender to retransmit the packet.

To enhance the effective switching throughput, the TNR uses a one-hop look-ahead virtual-channel scheduling algorithm for packets moving forward along the *X*-, *Y*- or *Z*-axis. An ordinary date-line virtual-channel scheduling algorithm sends a packet to a different virtual channel only when the packet moves across the date line. The one-hop look-ahead virtual-channel scheduling sends a packet to a different virtual channel not only when the packet goes through the date line but also when the next hop is the last hop along the axis. This algorithm lets the last hop packets bypass blocked preceding packets and improves the switch's effective throughput.

Tofu network interface

Each TNI consists of the following modules: manager of transmit (MTX), transmit assemble (TAS), manager of receive (MRX), receive buffer (RBF), manager of steering table (MST), manager of memory block (MMB), manager of transmit order queue prefetch (MTP), manager of packet buffer queue prefetch (MPP), and DMA arbiter.

A transmit order queue (TOQ) is a control queue, and a packet buffer queue (PBQ) is a packet queue. Both TOQ and PBQ are host-write, TNI-read queues. Figure 5 shows a block diagram of a TNI. The MTP, the MTX, and the TAS are packet-transmitting modules. The MTP reads commands from the main memory, the MTX decodes the commands and generates packet assemble requests, and the TAS assembles and transmits packets. The RBF, the MRX, and the MPP are packet-receiving modules. The RBF receives packets and passes packet headers to the MRX, which decodes the packet headers and generates payload store requests, and the RBF stores payloads in the main memory. When a received packet is a system packet, the MRX acquires packet buffer information from the MPP. The MST and the MMB manage virtual memory.

Direct descriptor

The TNI uses a direct descriptor technique that reduces the latency of descriptor fetching and does not affect the command throughput.[4] A direct descriptor register is a 64-byte memory-mapped register.

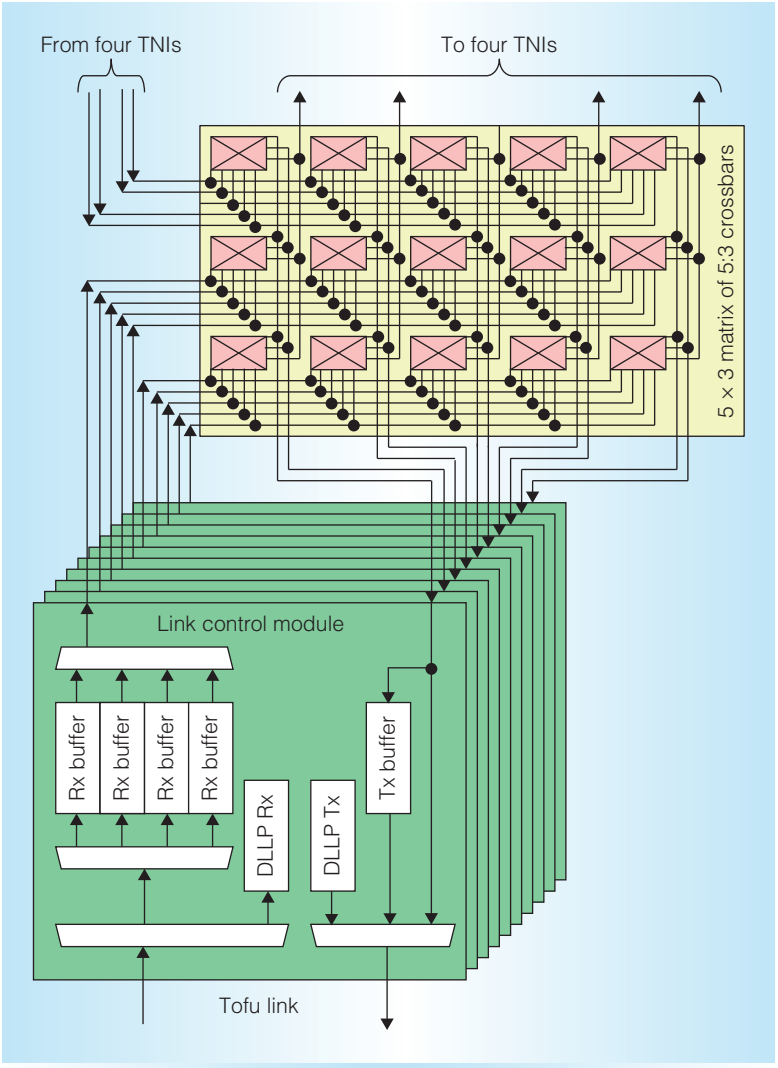

Figure 4. Block diagram of a TNR, which consists of a 15-ports crossbar and 10 link control modules.

A Sparc64 processor can write 64-byte data from eight floating-point registers with a single block write instruction. A direct descriptor register can contain double commands, which send 32-byte data embedded within them. The register reduces the communication latency of a 32-byte or shorter message. Writing a direct descriptor register also starts the fetching of subsequent commands.

Quad-rail communication

The four TNIs operate as independent quad-rails. This multirail communication multiplies the total throughput of nearest-neighbor traffic. Multirail communication also enhances effective throughput for

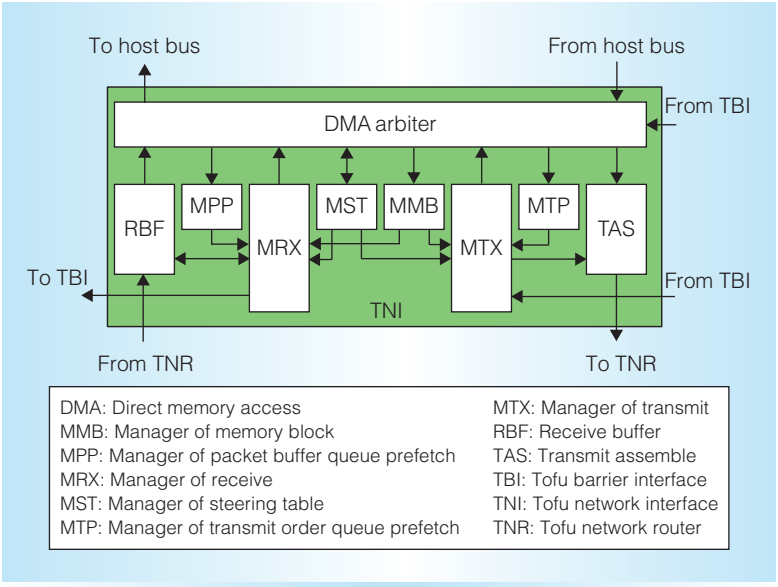

Figure 5. Block diagram of a TNI, which consists of three transmit modules (MTX, TAS, and MTP), three receive modules (MRX, RBF, and MPP), two virtual memory modules (MST and MMB), and a DMA arbiter.

Table 1. Operators and input data types of the Tofu barrier interface.

Operator	Data type
Barrier	None
AND, OR, XOR, MAX, SUM	64-bit integer
FPSUM	In 160/Out 320-bit floating point

random traffic patterns because the path diversity of the 6D mesh/torus improves the diversity of the link utilization.

Tofu barrier interface

The TBI provides the TNI with offload functionality for collective communication using an arbitrary algorithm within an arbitrary group of nodes. The TBI performs series of receives and sends without CPU intervention to avoid operating system jitter, which affects latency-sensitive applications. Each TBI consists of eight BCHs and 64 barrier gates. BCH control registers are mapped to user space, and barrier gate setting and status registers are in kernel space. Each barrier gate sends one packet after receiving one packet. Each BCH maintains a sequence of an arbitrary number of barrier gates. For example, we can use n barrier gates to achieve recursive doubling communication between n^2 nodes. Each barrier gate has double buffers to store the barrier packet of the next sequence in order to let other nodes repeat the sequence immediately after the previous one. The least-significant bit of the barrier sequence number, which is a 32-bit number contained in every barrier packet, identifies the buffer that is to store the received packet.

Each barrier gate has an arithmetic and logic unit to perform all-reduce collective communications. Table 1 shows the operator and data types supported. The TBI internally uses 320-bit format floating-point values to avoid the effect of cumulative rounding errors and to return the same result, given the same input values. The BCH takes a 160-bit floating-point input value containing a 155-bit significand and a 5-bit exponent, representing from −1,200 to +1,200 at 80 intervals. The BCH returns a 320-bit floating-point value containing a 235-bit significand represented in two 155-bit signed integer numbers and a 5-bit exponent. A barrier gate does not normalize a 235-bit significand, but just discards a small exponent one of two 155-bit digits. The final summation result is equivalent to selecting input values by exponent. We designed the 160/320-bit format to be easily converted from or into an ANSI/IEEE Std 754-1985 double-precision floating-point number.

This all-reduce summation method was developed by the Petascale System Interconnect project sponsored by Japan's MEXT; it has been implemented and proven successful in the highly functional switch of Fujitsu's FX1 Supercomputer.[7]

Preliminary evaluation

Our preliminary evaluation used a test system installed in the Fujitsu Numazu Plant and low-level APIs to measure the hardware's raw performance.

Direct descriptor

To assess the direct descriptor technique's effectiveness, we evaluated the difference in latencies when using the direct descriptor register and an ordinary fetch start register.

We measured the latencies for a two-node ping-pong benchmark that iterates 4-byte Put transmission and data polling 1 million times. The ping node and the pong node were neighborhood nodes in the *A* axis. Table 2 gives the result values from the evaluation. The average ping-pong communication latency using direct descriptor is 0.91 µs, and the latency using the fetch start register is 1.15 µs. The direct descriptor technique reduces the command fetch overhead by 0.24 µs.

Table 2. Measured latencies of Put ping-pong.

Trigger register	Command source	Latency
Fetch start	Main memory	1.15 µs
Direct descriptor	Register	0.91 µs

Single-rail point-to-point throughput

We evaluated three types of single TNI communication throughput. A *ping* benchmark sends messages consecutively and measures transmission throughput. The messages are of equal size in each benchmark, and we measure throughputs for various message sizes. The ping benchmark uses Put commands to transmit messages. Messages that are 8 bytes or shorter are piggybacked in a command.

A *ping-ping* benchmark simultaneously sends and receives consecutive messages and measures transmission throughput while the TNI is receiving messages.

A *ping-pong* benchmark measures communication throughput, including the routing delay overhead. In the ping-pong benchmark, messages longer than 8 bytes and shorter than 33 bytes are piggybacked in a two-descriptor Put command. For each benchmark, we repeat communication 1,000 times and measure the average throughput.

Figure 6 shows the results for the three benchmarks. Ping and ping-ping throughputs are suppressed by command throughput for 256-byte or shorter messages. For 512-byte or longer messages, ping and ping-ping throughputs saturate at around 4.76 GBps, for a bandwidth efficiency of over 95 percent. The results for the ping-ping benchmark show throughput near that of the ping benchmark and indicate no major back pressure in bidirectional communication. The ping-pong benchmark results saturate at message sizes of around 10 Kbytes. This result suggests that messages shorter than 10 Kbytes might cause latency-bound performance saturation.

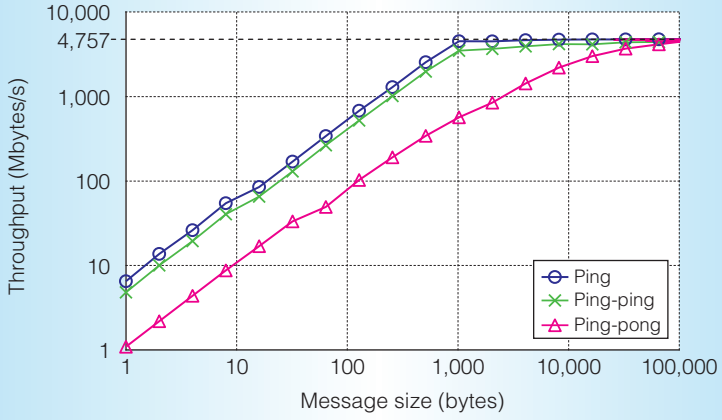

Figure 6. Results of evaluation of the ping, ping-ping, and ping-pong benchmarks. These benchmarks measure unidirectional, bidirectional, and round-trip throughputs of the Tofu interconnect.

One-hop look-ahead virtual-channel scheduling

We evaluated the one-hop look-ahead virtual-channel scheduling algorithm using register-transfer level (RTL) simulations of two entire instances of ICC chips. In the simulation model, all *XYZ* links of two ICC chips are connected to one another, as Figure 7 illustrates. We used entire instances of ICC chips to avoid the ambiguity of an abstracted model, and we used RTL simulations to evaluate the ordinary date-line virtual-channel scheduling algorithm. Because the simulation model was enormous, we ran simulations on Cadence Incisive Palladium systems (http://www.cadence.com/products/sd/palladium_series).

To evaluate congested situations, packets are consecutively injected and traverse a maximum of four hops each, along the *XYZ* axes. The test bench randomly gives the number of hops to move along the XYZ axes when the packet is injected. Packets are injected via the four ABC links for each node and ejected via ABC links and TNIs. The effective

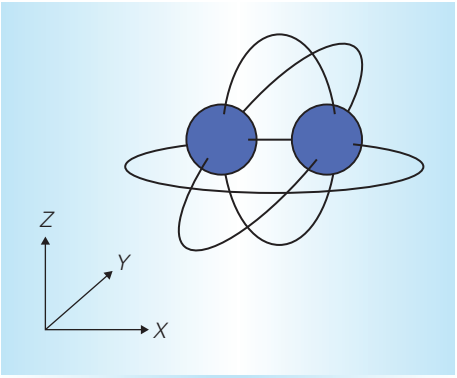

Figure 7. *XYZ* link connections for the register-transfer level (RTL) two-node simulation model used to evaluate congested situations. All *XYZ* links of the two ICC chips are interconnected.

switching throughput is measured between the 128 cycle intervals, and the collected results are represented in a histogram.

Figure 8a shows the results for ordinary date-line scheduling. The distribution peaks at around 18 GBps. Figure 8b shows the results for one-hop look-ahead scheduling. The peak of the distribution is smaller than for ordinary scheduling and has shifted to a higher throughput. In addition, the frequencies of throughput over 25 GBps have increased.

Multirail random-traffic throughput

We evaluated the effective single-node throughputs of random traffic using single-rail and multirail communication with a random-permutation-traffic benchmark in which each node transmits and receives 1,000 messages, and we measured the time between the first transmission and last reception. The *XYZABC* size of benchmark jobs varied from $2 \times 2 \times 1 \times 2 \times 3 \times 2$ to $2 \times 2 \times 16 \times 2 \times 3 \times 2$. The bisection bandwidth on the *Z* axis is a fixed 480 GBps. The expected single-node throughput decreases in inverse proportion to job size varied from 5 GBps at $2 \times 2 \times 4 \times 2 \times 3 \times 2$ to 1.25 GBps at $2 \times 2 \times 16 \times 2 \times 3 \times 2$.

Multirail interfaces are used along minimal multiple paths. When all of the *ABC* coordinates of the source node and those of the destination are different, the message is divided into four smaller messages and transmitted simultaneously via four different paths. When one or two of the *ABC* coordinates are different, the message is divided into two smaller massages. When all of the *ABC* coordinates are the same, the message is not divided.

Figure 9a shows the single-node throughput for single-rail communication. The peak

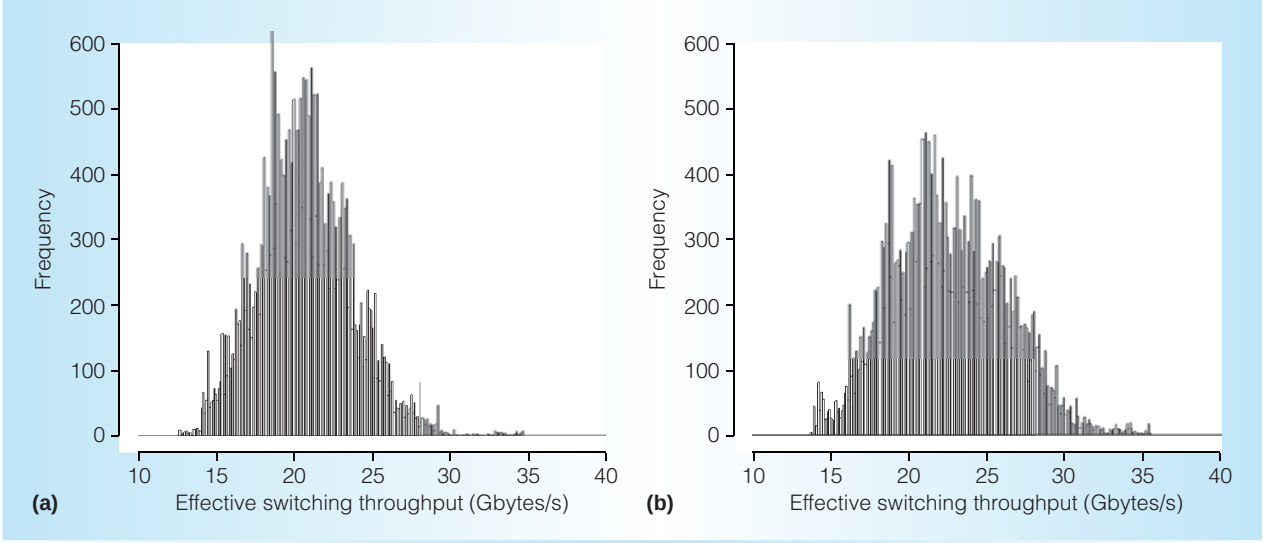

Figure 8. Histograms of effective switching throughput for ordinary date-line virtual-channel scheduling (a) and the one-hop look-ahead virtual-channel scheduling (b). These results show that the one-hop look-ahead scheduling improves effective switching throughput.

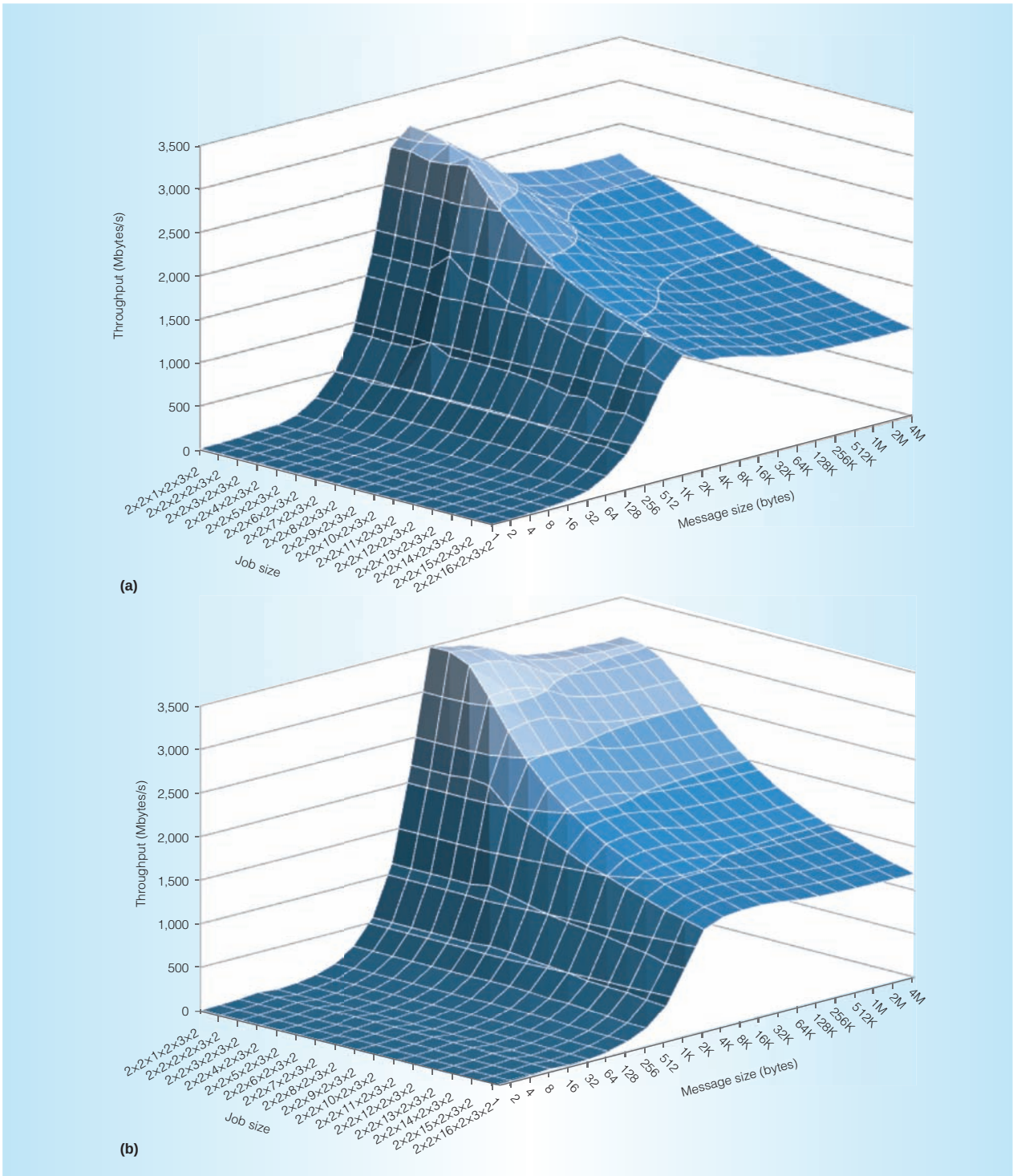

Figure 9. Message size versus single-node throughput of random permutation traffic using single-rail communication (a) and multirail communication (b). These results show that the single-node throughput of multirail communication is stable for large message sizes.

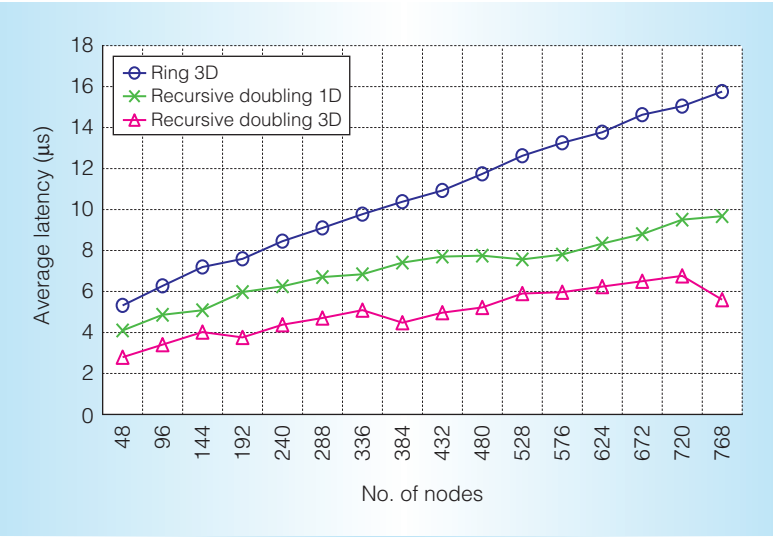

Figure 10. Tofu barrier latencies using different communication algorithms vary. The recursive doubling, order log *n* algorithm, and its topology-aware implementation reduce latencies.

throughput is at a message size of around 1 Kbyte. The ratio to the expected throughput is about 60 percent at job size $2 \times 2 \times 4 \times 2 \times 3 \times 2$, about 80 percent at $2 \times 2 \times 8 \times 2 \times 3 \times 2$, and about 100 percent at $2 \times 2 \times 16 \times 2 \times 3 \times 2$. The throughput decreases as the message size increases because of unbalanced link utilization.

Figure 9b shows the single-node throughput for multirail communication. Compared to single rail, the single-node throughput for long messages does not decrease as much from the peak throughput, which is nearly equal to that of single rail. This stability likely results from the diversity of link usage. However, the peak throughput is at around 4 Kbytes. At message sizes smaller than 4 Kbytes, single-rail throughput is higher than multirail throughput. Therefore, messages should not be divided when the message size is smaller than 4 Kbytes.

Tofu barrier

The latency of barrier synchronizations using TBI varies depending on the communication algorithm. We measured the latencies of the Tofu barrier using a 3D ring, 1D recursive doubling, and 3D recursive doubling algorithm. A 3D algorithm consists of horizontal, vertical, and depth phases using the logical 3D torus coordinates given by the system. The benchmark job size varied from 48 to 768 nodes.

Figure 10 shows the results. The latency of the 3D ring increases linearly because the number of barrier gates increases as the *Z*-axis length of the job size increases. The latency for 1D recursive doubling is lower than that for the 3D ring because recursive doubling uses log2(N) barrier gates for an N node job. Three-dimensional recursive doubling further reduces the latency because the hop count is optimized.

High-performance computing systems are becoming larger and making it more difficult to exploit network locality. By enabling traditional 3D torus topology-aware communication optimizations, and by improving random communication performance, the Tofu interconnect architecture provides a migrate and upgrade path to many applications for future exascale systems. MICRO

References

1. Y. Ajima, S. Sumimoto, and T. Shimizu, ''Tofu: A 6D Mesh/Torus Interconnect for Exascale Computers,'' *Computer,* vol. 42, no. 11, Nov. 2009, pp. 36-40.
2. N.R. Adiga et al., ''An Overview of the BlueGene/L Supercomputer,'' *Proc. ACM/IEEE Conf. Supercomputing,* IEEE CS Press, 2002, doi: 10.1109/SC.2002.10017.
3. R. Alverson, D. Roweth, and L. Kaplan, ''The Gemini System Interconnect,'' *Proc. IEEE 18th Ann. Symp. High Performance Interconnects,* IEEE CS Press, 2010, pp. 83-87, doi:10.1109/HOTI.2010.23.
4. T. Toyoshima, ''ICC: An Interconnect Controller for the Tofu Interconnect Architecture,'' Hot Chips Conf. (Hot Chips 22), 2010; http://www.hotchips.org/uploads/archive22/HC22.24.510-1-Toyoishima-Tofu-icc.pdf.
5. T. Maruyama et al., ''Sparc64 VIIIfx: A New-Generation Octocore Processor for Petascale Computing,'' *IEEE Micro,* vol. 30, no. 2, Mar./Apr. 2010, pp. 30-40.
6. J.R. Black, ''Mass Transport Aluminum by Momentum Exchange with Conducting Electrons,'' *Proc. IEEE 6th Ann. Reliability Physics Symp.,* IEEE Press, 1967, pp. 148-159.
7. T. Abe, T. Inari, and K. Seki, ''JAXA Supercomputer Systems with Fujitsu FX1 as

Core Computer,'' *Fujitsu Scientific and Technical J.*, Oct. 2008, pp. 426-434.

Yuichiro Ajima is a system architect in the Next-Generation Technical Computing Unit at Fujitsu. His research focuses on high-performance computing system architecture. Ajima has a PhD in information engineering from the University of Tokyo. He is a member of the Information Processing Society of Japan and IEEE.

Tomohiro Inoue is an engineer in the Next-Generation Technical Computing Unit at Fujitsu. His research focuses on computer networks. Inoue has a PhD in information engineering from Hiroshima City University.

Shinya Hiramoto is an engineer in the Next-Generation Technical Computing Unit at Fujitsu. His research focuses on high-performance computing system architecture. Hiramoto has an ME in information engineering from Nara Institute of Science and Technology. He is a member of the Information Processing Society of Japan.

Toshiyuki Shimizu is a director of the Next-Generation Technical Computing Development Unit at Fujitsu. His research focuses on high-performance computing system architecture, particularly interconnect architecture for highly scalable systems. Shimizu has an MS in computer engineering from the Tokyo Institute of Technology.

Yuzo Takagi is a former engineer in the Next-Generation Technical Computing Unit at Fujitsu. He currently is a business planner, product manager, and data-mining analyst at DeNA Corporation. His research interests include astrophysics, plasma physics, fluid dynamics, Internet services, parallel computing, and high-performance computing system architecture. Takagi has an MS in astrophysics from the University of Tokyo.

Direct questions or comments about this article to Yuichiro Ajima, 1-1 Kamikodanaka, 4-chome Nakahara-ku, Kawasaki 211-8588 Japan; aji@jp.fujitsu.com.