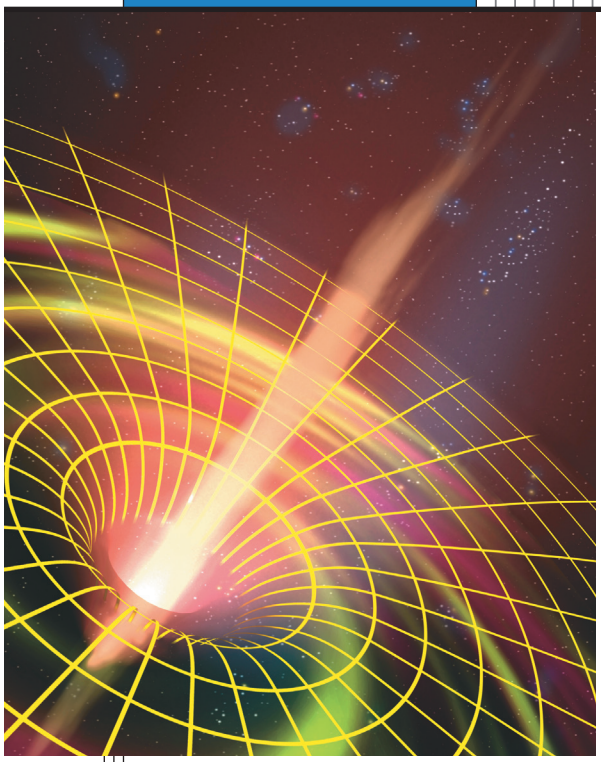

TOFU: A 6D MESH/TORUS INTERCONNECT FOR EXASCALE COMPUTERS

Yuichiro Ajima, Shinji Sumimoto, and Toshiyuki Shimizu, *Fujitsu*

A new architecture with a six-dimensional mesh/torus topology achieves highly scalable and fault-tolerant interconnection networks for large-scale supercomputers that can exceed 10 petaflops.

Researchers continue to improve high-performance computing systems by increasing the number of processor cores per node and nodes per system. To interconnect tens of thousands of nodes, many HPC systems employ mesh/torus topologies because of their high scalability and low cost/performance ratio.

On a mesh-connected system, topology-aware tuning is important for many applications. The system should be able to allocate a job contiguously and provide a torus topology for an individual job. To achieve high system utilization, a flexible-sized submesh is also important. To meet these needs, we have developed Tofu, an interconnect architecture that features a higher-radix mesh/torus topology (Tofu stands for "torus fusion" or "torus-connected full connection"). A Tofu system can be divided into an arbitrary size of rectangular submeshes just like a block of tofu, and provides a torus topology for each submesh.

TOFU INTERCONNECT ARCHITECTURE

Topology-aware tunability and system utilization are the primary trade-offs for mesh/torus-connected systems. Contiguous job allocation is intended to fulfill users' needs for topology-aware tuning, while noncontiguous job allocation maximizes system utilization. QCDOC (Quantum Chromodynamics on a Chip),[1] Blue Gene,[2] and QPACE (QCD Parallel Computing on the Cell Broadband Engine)[3] systems employ contiguous job allocation, while the Cray XT series[4] employs noncontiguous job allocation.

A contiguous job allocation scheme potentially requires additional hardware like partition switches to provide a consistent view of network topology wherever an application runs on a full system or subsystem. Assuming particular submesh shapes, architectural designers can reduce the number of necessary partition switches. However, a more flexible mechanism is required when assuming various job sizes. Because a multidimensional mesh topology can embed a ring topology, a higher-radix mesh/torus offers one solution. QCDOC employs a 6D torus topology and 12 network links per node.

We designed Tofu to be a 6D mesh similar to QCDOC but with a reduced number of network links, as a link's peak throughput is roughly inversely proportional to the number of network links per node. Tofu reduces links

 Published by the IEEE Computer Society

by restricting the length of some additional dimensions to two. A dimension of length two requires only one additional network link and is sufficient to combine with another longer dimension to embed a ring topology.

Tofu has six coordinate axes: *x*, *y*, *z*, *a*, *b*, and *c*. The lengths of the *ac* axes are restricted to two, so each node has a total of 10 links. The length of the *b*-axis is restricted to three instead of two for fault tolerance. Twelve nodes having the same *xyz* coordinates constitute a *node group* and are interconnected by the *abc*-axes. A node group can be considered a unit of job allocation. Figure 1 shows examples of three node groups whose *xyz* coordinates are (0,0,0), (1,0,0), and (2,0,0). The spherical vertices represent nodes and the cylindrical edges *abc*-axes.

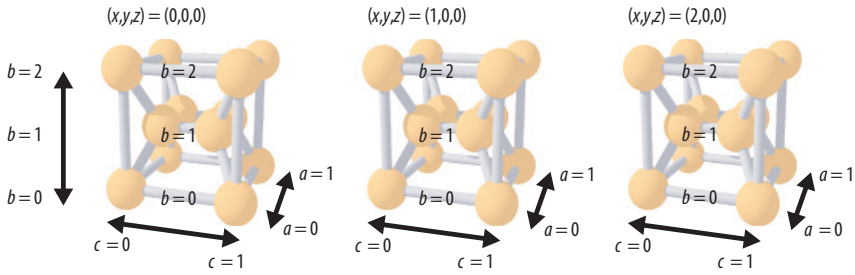

Figure 1. Example Tofu node groups. Twelve nodes (spherical vertices) having the same *xyz* coordinates constitute a node group and are interconnected by the *abc* axes (cylindrical edges).

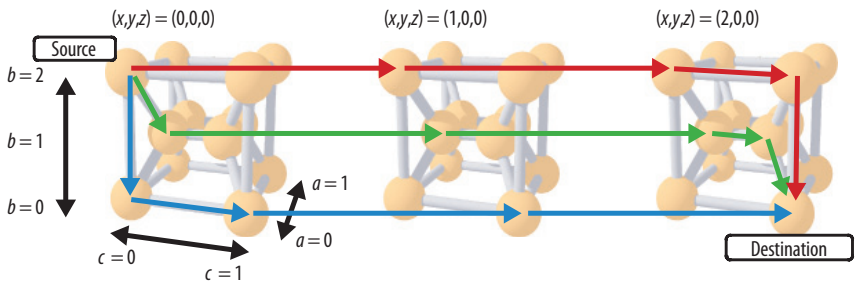

Figure 2. Multipath routing in Tofu. Example of 3 out of 12 paths from the node (0,0,0,0,2,0) to the node (2,0,0,0,0,1).

Multipath routing function

A routing algorithm that detours a unit under maintenance is necessary to enable hot-swap maintenance. We therefore developed an algorithm for Tofu that divides packet routing into three phases. First, a packet traverses the *abc*-axes to select a path at a source node group. It then moves from the source node group to the destination node group along the *xyz*-axes. Finally, the packet travels along the *abc*-axes again to a destination node at the destination node group. Although the routing path in each phase is minimal, the total routing path is not because the Tofu routing algorithm can take *abc*-axes twice. Tofu routing can relay through an arbitrary node in a source node group, so there are a total of 12 paths for an arbitrary destination.

Figure 2 shows examples of multiple paths. The arrows represent three routes from node (0,0,0,0,2,0) to node (2,0,0,0,0,1).

Torus mapping and fault tolerance

For many applications, topology-aware tuning is recommended to achieve the scalability of tens of thousands of nodes. Therefore, a Tofu system offers every job a network topology view of a 3D torus. It embeds a 3D torus view into a 6D mesh space by corresponding two axes of the 6D mesh to one application-view axis. The system sequentially allocates the coordinates of each application-view axis to form a loop in the plane composed of the 6D mesh's two axes.

Unfortunately, faults inevitably occur in extremely large systems, despite efforts to reduce the failure rate. Therefore, it is important to build a fault-tolerant system that isolates and replaces faulty components without stopping system operation. To achieve higher system utilization while tolerating faults and failures, Tofu has the ability to offer a 3D torus view on a submesh with a faulty node.

Figure 3a shows an example of mapping coordinates of an application-view axis with a length of 9 on a non-faulty submesh. In this case, the system chooses the *x*-axis and *b*-axis of the 6D mesh to be an allocation plane for the application-view axis. It maps coordinates 0 on node (0,0,0,0,0,0) and 5 on node (2,0,0,0,0,0).

Figure 3b shows an example of fault-tolerant mapping. Node (1,0,0,0,1,1) is down, and the length of the application-view axis is reduced to 8. The system removes the four nodes (1,0,0,0,1,0), (1,0,0,0,1,1), (1,0,0,1,1,0), and (1,0,0,1,1,1) from the coordinate allocation. This mechanism is also effective for the hot-swap maintenance scheme.

High system utilization

High system utilization is required for cost-effectiveness. Job allocation scheduling is especially effective and requires few additional hardware costs. For example, the RIKEN Super Combined Cluster (RSCC) system achieved 78-79 percent utilization during 2005-2006, but there was still room for improvement. Consequently, RIKEN and Fujitsu Laboratories collaboratively introduced the

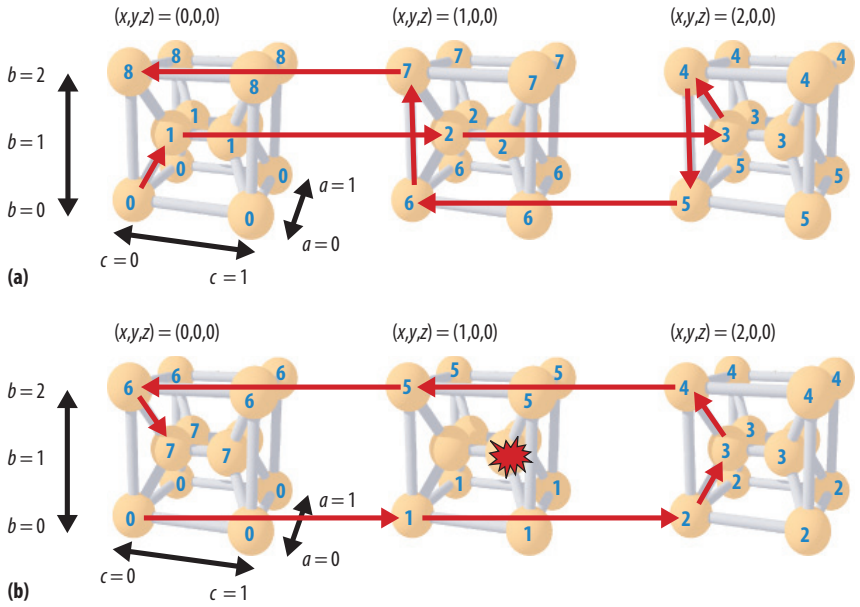

Figure 3. Fault-tolerant mapping in Tofu: (a) example of mapping on a nonfaulty submesh and (b) example of fault-tolerant mapping.

Meta Job Scheduler, job allocation scheduling software that performs backfill scheduling, which improved RSCC's system utilization to 92-93 percent during the following two years.[5] This was roughly equivalent to a 17 percent hardware enhancement.

For mesh-connected supercomputers, fragmentation is the main obstacle to high system utilization. Working from the premise that flexible submesh shapes would help fill small fragments of free nodes, we utilized dimensional combinations to map the same 3D torus view. In Tofu, the *b*-axis has a different length than the *ac*-axes; this asymmetricity increases a 6D mesh shape's variation to achieve the specified 3D torus view.

Consider, for example, all the possible mesh/torus shapes that can map a 12 × 12 × 6 torus view for a 3D torus-connected system and a Tofu system:

3D torus-connected system: 12 × 12 × 6, 12 × 6 × 12, 6 × 12 × 12
Tofu system: 6 × 6 × 2 × 2 × 3 × 2, 6 × 2 × 6 × 2 × 3 × 2, 2 × 6 × 6 × 2 × 3 × 2, 6 × 4 × 3 × 2 × 3 × 2, 6 × 3 × 4 × 2 × 3 × 2, 4 × 6 × 3 × 2 × 3 × 2, 4 × 3 × 6 × 2 × 3 × 2, 3 × 6 × 4 × 2 × 3 × 2, 3 × 4 × 6 × 2 × 3 × 2

There are three possible shapes for the 3D torus-connected system and nine for the Tofu system. A larger number of allocation candidate mesh shapes would be expected to improve system utilization.

We simulated system utilization using a real workload log from the Parallel Workloads Archive (www.cs.huji.ac.il/labs/parallel/workload). We scaled a job's number of nodes, with properties of candidate mesh shapes corresponding to the number of nodes, and assumed a large job had redundancy nodes for fault avoidance. The simulation results with backfill job scheduling showed that system utilization of a single candidate job shape was about 70 percent and that of multiple candidates improved to about 80 percent. Based on these results, we are developing a new job allocation scheduler which supports job allocation from multiple job shape candidates and utilizes enhanced scheduling algorithms of the Meta Job Scheduler.

ADDITIONAL TOFU FEATURES

The Tofu interconnect architecture has several other features.

Throughput and packet transfer

Tofu has high-throughput links with 10 gigabytes per second of fully bidirectional bandwidth for each. The link throughput is roughly derived from the off-chip bandwidth and network degree 10. We implemented 100 GBps of the off-chip bandwidth for each node to feed enough data to a massive array of 128-Gflops processors.[6]

Packet length is variable to minimize packet header overhead and improve effective bandwidth. The minimum packet length is 32 bytes and the maximum is 2,048 bytes, including the header and cyclic redundancy check (CRC). Packets are transferred via virtual cut-through,[7] which achieves low latency by buffering packets only when the destination link is blocked.

Each link has four 8-Kbyte receive buffers and an 8-Kbyte retransmission buffer. The receive buffers correspond to four virtual channels and are used while the destination link is blocked. The retransmission buffer is used for link-level retransmission that recovers CRC errors.

I/O communication routing

Preparing a dedicated I/O interconnect ensures I/O bandwidth. To achieve a beneficial cost/performance tradeoff, computation and I/O communication should share bandwidth—this is one of the reasons why computation and I/O phases are often separated in a single application. However, even if computation and I/O communications share bandwidth, I/O communication should

not pass the area in which other jobs communicate for computation. To minimize the path on which I/O communication crosses other jobs, we arranged the coordinates of the I/O node and routing orders.

Figure 4 shows a desirable I/O communication path. A partial network of z coordinates equal to 0 forms the I/O network, and the remaining network of z coordinates larger than 0 defines the computation network. Rather than have I/O communication pass the x-axis and y-axis on the computation network, it is preferable that it only pass the z-axis. We designed two groups of virtual channels with different routing orders, ensuring that a desirable virtual channel could transmit a packet dedicated for I/O communication. Computation nodes transmit packets by virtual channels that have the z-axis as the first order, and I/O nodes transmit packets by virtual channels that have the z-axis as the last order.

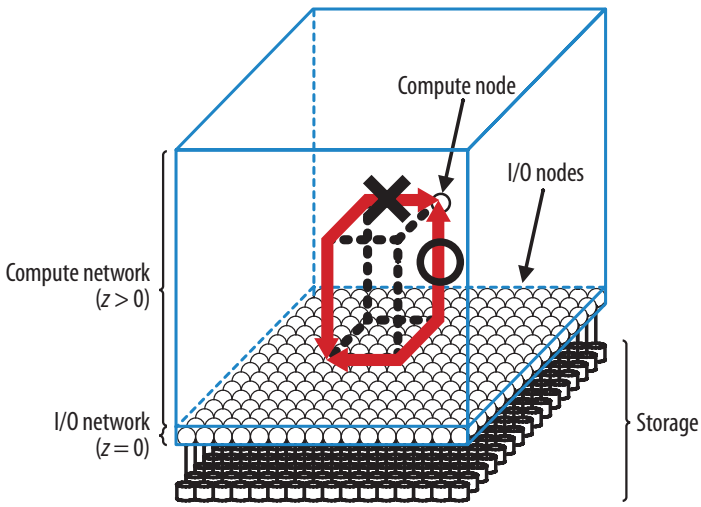

Figure 4. Desirable I/O communication path. A partial network of z coordinates equal to 0 forms the I/O network, and the remaining network of z coordinates larger than 0 defines the computation network. Computation nodes transmit packets by virtual channels that have the z-axis as the first order, and I/O nodes transmit packets by virtual channels that have the z-axis as the last order.

Multiple communications engines

Some applications can overlap communications to multiple nodes simultaneously. Parallel execution of communication primitives by multiple communication engines may improve performance, especially of communication-intensive and nearest-neighbor communicating applications like Lattice QCD. We therefore implemented four communication engines in each node in Tofu. As Figure 5 shows, each engine can transmit and receive packets simultaneously, transmit packets to any direction, and receive packets from any direction. Four communication engines can also improve the throughput of point-to-point communication by simultaneously using multiple paths between nodes.

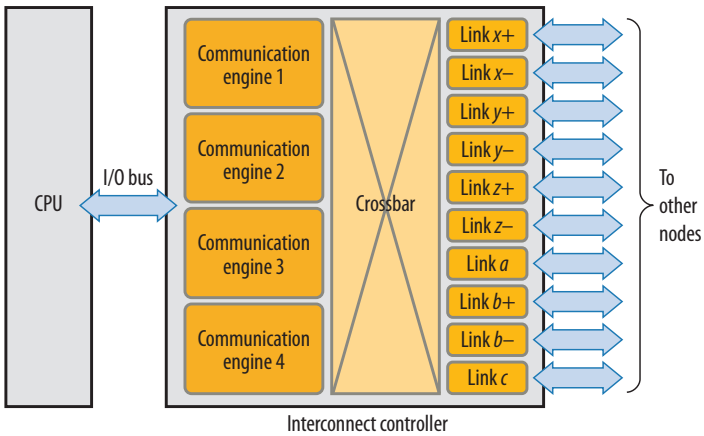

Figure 5. Block diagram of a node with four communication engines. Each engine can transmit and receive packets simultaneously, transmit packets to any direction, and receive packets from any direction.

Integrated collective function

The collective communication function's communication time often suppresses a highly parallel application's performance scalability. Collective communications are generally implemented by combining point-to-point communications, whose frequency increases with the number of nodes engaging in the collective communication function. Each point-to-point communication involves CPU processing time that often gets unexpectedly longer, and this variability greatly affects the completion time of a highly parallel collective communication function.

We addressed this problem by designing in the hardwired logic an integrated collective function, which processes frequently used collective communication functions (barrier and reduce) without any CPU intervention. Collective communication functions are operated by chained barrier gates. A barrier gate waits for and transmits one barrier packet. Figure 6 shows an example of a chain of four barrier gates. This chain of barrier gates repeats transmission and reception of a barrier packet four times to synchronize 16 nodes.

A chain of barrier gates can also perform the all-reduce operations of a 64-bit integer or a double-precision floating-point number. It converts a double-precision floating-point number into two 160-bit floating-point data to obtain the same result at all nodes with a single barrier

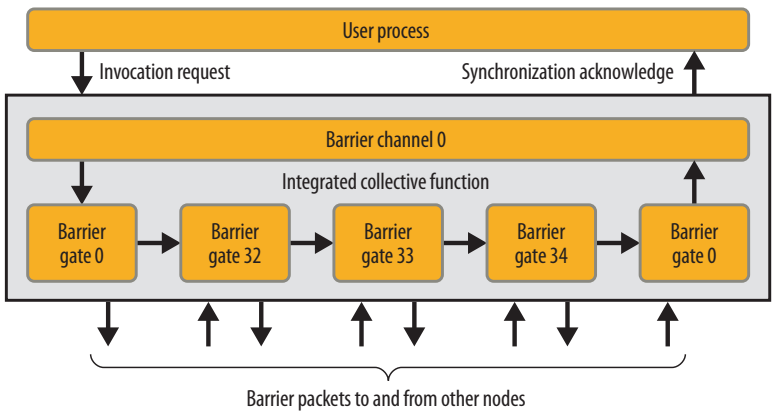

Figure 6. Example of four barrier gates operating and connected to a barrier channel. This chain of barrier gates repeats transmission and reception of a barrier packet four times to synchronize 16 nodes.

synchronization sequence. This floating-point calculation method was developed by the Petascale System Interconnect project sponsored by Japan's Ministry of Education, Culture, Sports, Science and Technology, and was previously implemented in the Highly Functional Switch of Fujitsu's FX1 supercomputer.[8]

High-performance computing systems are becoming denser, leaner, and more integrated, making it increasingly challenging to manage and repair failures. By effectively isolating components that require maintenance and repair, the Tofu interconnect architecture promises to be a fundamental technology for future exascale systems.

Acknowledgments

The authors thank Yuji Oinaga and the many Tofu project members for their valuable suggestions and efforts.

References

1. P. Boyle et al., "The QCDOC Project," *Nuclear Physics B—Proc. Supplements*, Mar. 2005, pp. 169-175.
2. The BlueGene/L Team, "An Overview of the BlueGene/L Supercomputer," *Proc. 2002 ACM/IEEE Conf. Supercomputing* (SC 02), IEEE CS Press, 2002; https://asc.llnl.gov/computing_resources/bluegenel/pdf/sc2002-pap207.pdf.
3. G. Goldrian et al., "QPACE: Quantum Chromodynamics Parallel Computing on the Cell Broadband Engine," *IEEE Computing in Science and Eng.*, Nov./Dec. 2008, pp. 46-54.
4. W.J. Camp and J.L. Tomkins, "Thor's Hammer: The First Version of the Red Storm MPP Architecture," *Proc. 2002 ACM/IEEE Conf. Supercomputing* (SC 02), IEEE CS Press, 2002.
5. T. Shigetani, "RIKEN Super Combined Cluster Operations Report" (in Japanese), Mar. 2009; http://accc.riken.jp/HPC/Symposium/2008/shige.pdf.
6. T. Maruyama, "SPARC64 VIIIfx: Fujitsu's New Generation Octo Core Processor for PETA Scale Computing," presentation, Hot Chips 21, 2009.
7. P. Kermani and L. Kleinrock, "Virtual Cut-Through: A New Computer Communication Switching Technique," *Computer Networks*, Jan. 1979, pp. 267-286.
8. T. Abe, T. Inari, and K. Seki, "JAXA Supercomputer Systems with Fujitsu FX1 as Core Computer," *Fujitsu Scientific & Technical J.*, Oct. 2008, pp. 426-434.

Yuichiro Ajima *is a system architect in the Next-Generation Technical Computing unit at Fujitsu Limited. His research interests are in technical computing system architectures. He received a PhD in information engineering from the University of Tokyo. Ajima is a member of the Information Processing Society of Japan (IPSJ). Contact him at aji@jp.fujitsu.com.*

Shinji Sumimoto *is a senior architect in the Next-Generation Technical Computing unit at Fujitsu Limited, and a research fellow at Fujitsu Laboratories Limited. His research interests are in high-performance cluster system architectures, especially high-performance communication, and large-scale computing architectures. Sumimoto received a PhD in electrical engineering from Keio University. He is a member of IPSJ. Contact him at s-sumi@labs.fujitsu.com.*

Toshiyuki Shimizu *is a director in the Next-Generation Technical Computing unit at Fujitsu Limited. His research interests include high-performance computer system architectures, particularly interconnect architectures for highly scalable systems. Shimizu is a member of IPSJ and the Institute of Electronics, Information, and Communication Engineers. Contact him at t.shimizu@jp.fujitsu.com.*