計算機アーキテクチャ　102−10
(1993. 10. 21)

超並列プロトタイプ計算機 JUMP-1 の構想

平木 敬[*1], 天野 英晴[*2], 久我 守弘[*3], 末吉 敏則[*3], 工藤 知宏[*4], 中島 浩[*5], 中條 拓伯[*6],
松田 秀雄[*6], 松本 尚[*1], 森 眞一郎[*5]

[*1]東京大学 工学部, [*2]慶応大学 理工学部, [*3]九州工業大学 マイクロ化総合技術センタ,
[*4]東京工科大学 情報工学科, [*5]京都大学 工学部, [*6]神戸大学 工学部

本稿では超並列プロトタイプ計算機 JUMP-1 のアーキテクチャについて概説する。JUMP-1 は効率的な分散共有メモリの実現を最大の目的として設計されており，メモリ・アクセスのレイテンシ削減のために様々な機構が用意されている。例えば 3 レベルのキャッシュは invalidate と update の双方のコヒーレンス・プロトコルをサポートするとともに，プロセッサ間での高速な通信／同期のための機能を備えている。またメモリの一貫性の分散管理に適したスケーラブルなネットワークを提案し，従来のものに比べて高い性能が得られることを評価によって確認した。さらに，画像や大規模ファイルの高速な入出力のために，JUMP-1 には多数のシリアル・リンクを用いたスケーラブルな I/O システムも備えられている。

Overview of a Massively Parallel Computer Prototype: JUMP-1

Kei Hiraki[*1], Hideharu Amano[*2], Morihiro Kuga[*3], Toshinori Sueyoshi[*3],
Tomohiro Kudoh[*4], Hiroshi Nakashima[*5], Hironori Nakajo[*6], Hideo Matsuda[*6],
Takashi Matsumoto[*1] and Shin-ichiro Mori[*5]

[*1] The University of Tokyo, [*2] Keio University, [*3] Kyushu Institute of Technology,
[*4] Tokyo Engineering University, [*5] Kyoto University, [*6] Kobe University

This paper summarizes the architecture of a massively parallel computer, JUMP-1. The most important objective of the JUMP-1 is the implementation of an efficient distributed shared-memory system reducing access latency. To cope with it, we introduced a three level cache system not only managing its coherency by both invalidate and update protocols, but also providing inter-processor communication and synchronization mechanism. We also proposed a scalable interconnection network fit for the distributed management of memory consistency, and a scalable I/O system for high speed image and file processing. As for the interconnection network, the performance evaluation result shows its advantage to conventional scalable networks.

1 Introduction

To construct a new scheme of information processing in 21st century, we are now pursuing a joint university project called JUMPP (Japan University Massively Parallel Processing Project). This project aims to give fundamental paradigms and technologies for 10^6 scale massively parallel processing. The research items of the project are computation models, programming languages, operating systems, and hardware systems.

It is also planned to build a prototype system as the substantiation of these research works, and as the platform for further research. The prototype hardware system is called JUMP-1, on which a prototype operating system, prototype language processing systems, and various applications will be implemented. The JUMP-1 is a general purpose parallel computer with a distributed shared-memory system. It has 1024 processors together with intelligent caches, memory management co-processors, a high performance interconnection network, and image and file I/O systems.

In the following sections, Section 2 to Section 6, the architecture of the JUMP-1 and its components are summarized. The performance evaluation results of the interconnection network is shown in Section 7. Section 8 gives the conclusion and our research schedule.

2 Global Architecture

As a general-purpose processing system, the JUMP-1 must satisfy various requirements, which may contradict each other. The global architecture of the JUMP-1 is designed to satisfy the most fundamental issues for massively parallel processing systems. They are:

Efficient globally-addressable memory

Since a shared-memory programming model is as general as a sequential programming model in uniprocessor, efficient globally-addressable memory is crucial to realize general-purpose massively parallel processing.

The shared-memory system reduces overheads on implicit process communications through shared variables if a shared-memory is properly supported by hardware and architecture.

Optimized shared memory system that utilize update protocol

Existing snoop-cache protocols are not sufficient for reducing cache-cache traffic to achieve efficient parallel processing in numerical intensive computations and fine-grained computations. This is because numerical intensive applications require very frequent and wide memory accesses and because fine-grained computations require frequent interprocessor communication and synchronization through a shared-memory.

At the same time, other difficulties such as memory access latency and synchronization overheads must be solved for efficient executions on massively parallel processing systems.

Efficient message-passing and synchronization supports

In applications that are not data-parallel, overheads related to message passing and fine-grained synchronization are major sources of deficiency. One approach to these problem is to add separate message handling and synchronization mechanisms to a processor. However, those mechanisms limits merits of shared virtual address space unless they are implemented on separate virtual name spaces.

For avoiding excessively complication and overheads on name space translation, message-passing operation and synchronization must be realized on a single shared virtual memory system.

Optimized to both coarse-grained and fine-grained programs

When application programs are regular numerical incentive computations, performance on coarse-grained computations are the most important issue. While, very fine-grained operations are necessary for irregular problems and implementation of efficient shared virtual memory. They are also corresponding to local computations and global computations respectively. Concurrent execution of local and coarse-grained computations and global and fine-grained computations is essential to achieve general-purpose massively parallel processing.

Scalable network architecture

Scalable network architecture is a widely recognized requirement. However, scalability of the shared virtual memory space is far more important for general-purpose massively parallel processing than scalability of data traffic which is determined by the property of physical transmission lines. Therefore network architecture that supports scalable shared virtual memory protocols is necessary for constructing scalable massively parallel systems.

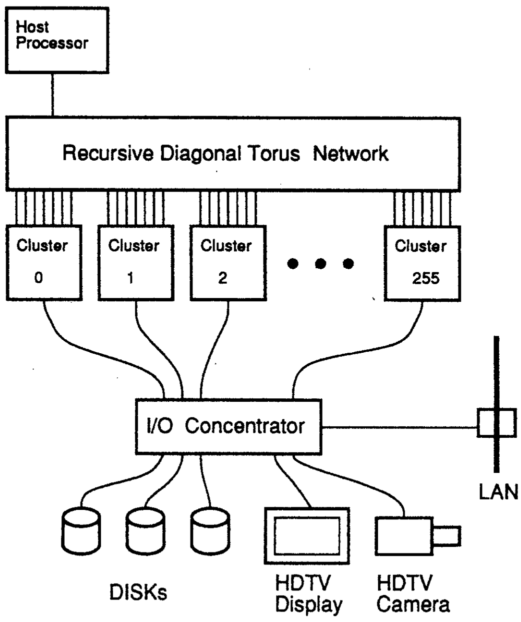

Figure 1: Global architecture of JUMP-1

The second requirement for scalable network architecture is that the system must smoothly scale to any system size. In this point of view, several network topology like hypercubes, hyper-crossbar cannot smoothly scale. Partitioning capability is another scalability requirement that is important under multi-user / multi-programming environment. The performance of the partition must be scale down as the size of partition decreases.

Scalable I/O subsystems

Data transfer between I/O devices and processing elements has different characteristics from communication between processing elements. Major differences are (1) locality of connections, (2) time duration of connections and (3) allowed latency. When a partitioned operating system is used, I/O data transfer cannot be routed on a interconnection network because I/O connections has no locality. Therefore, a separate I/O network and I/O subsystem is required to achieve scalable an I/O subsystem.

The JUMP-1 adopts clustered architecture (Figure 1) as a basic architecture. Clustered architecture has several good properties for massively parallel systems because (1) clustered architecture reduce the cost of interconnection network for typical numerical intensive applications, (2) it improves utilization of the interconnection network, (3) it allows more efficient memory consistency protocols and synchronization protocols and (4) it matches current physical implementation technology. For optimizing compilers and application programmers, clustered architecture has additional benefits. Since local processors are more tightly coupled than non-clustered system, optimizing compilers can exploit short-range parallelism in application programs. Furthermore, a program on a cluster can directly access large amount of memory, which is essential to implement irregular applications and operating systems.

As shown in figure 1, the JUMP-1 consists of 256 processor clusters and three different networks. The first network is Recursive Diagonal Torus (RDT) network for interconnecting processor clusters as described in the later section. The second network is an I/O network for disks and high-definition video devices. The I/O network is a point to point high-speed serial link with network concentrator which dynamically change the assignment of I/O devices to clusters. The third network is a maintenance network for booting, instrumentation and debugging, which is a tree-structured SCSI buses.

Figure 2 shows the block-diagram of a cluster. A cluster consists of 4 coarse-grained processors (CPU), 2 fine-grained processors that is directly connected to a main memory (Memory-Based Processor, or MBP), 4 secondary caches (L2 cache) that interface between a CPU and an MBP, two network interface processors (NIPs), a network router, an I/O network interface and a common bus.

A CPU is a off the shelf RISC processor (SUN SuperSparc) for the main part of the application programs because current RISC shows the best performance in sequential computations with locality in memory references by introducing large amount of process context (registers and system resources). The characteristics of an MBP is to complement a CPU for short thread fined-grained computations with frequent context switches. Since there are no commercial fine-grained MPU, the MBP is a custom design processor with associated support hardware as described later. An L2 cache memory is a secondary cache memory from CPU and the target of cache injection [1] with synchronization functions. Therefore, a CPU and an MBP form a decoupled architecture for global memory access operations. A main memory, also called L3 cache, has synchronization tags on each word for

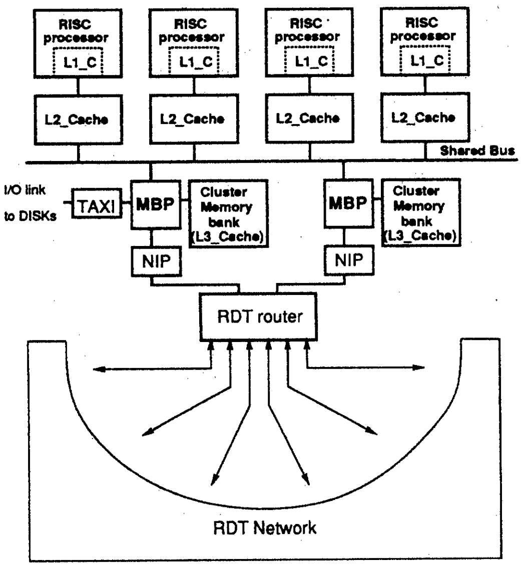

Figure 2: Block diagram of a cluster

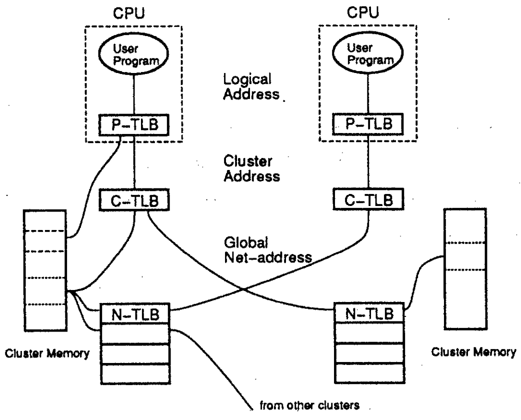

Figure 3: Address translation in JUMP-1

implementing I-structures, FIFO queues and other memory-based synchronization primitives.

A NIP is an interface processor to RDT network, which generates network routing information, assembles packets with error check code. A router is a node of RDT network with elastic barrier hardware. An I/O network interface is a high-speed serial transmitter/receiver (TAXI) with message buffers.

The global memory architecture of the JUMP-1 is a strategic memory system (SMS) which integrates shared virtual memory, synchronization and cache injection. The main features of the JUMP-1 SMS are as follows:

1. Three level virtual address space for flexible memory sharing, migration and protection (figure 3).
2. Dynamic switching of coherent protocols for optimizing access latency and network (bus) traffic (ICSCM, [2]).
3. Memory based synchronization on both L2 caches and main memories.
4. Integration of synchronization, communication and consistency protocols, which extends dynamic switching of coherent protocols to support producer-consumer synchronization efficiently [3].
5. Cache injection facility (CIF) for efficient communications by non-demand-driven memory operations [1].
6. Two level consistency protocol for efficient implementation of SMS. Snooping protocols are used to keep consistency in a cluster and a pseudo-full-map directory based protocols [4] are used to keep consistency among clusters.

Figure 3 outlines the address translation mechanisms in the JUMP-1. For implementing three level virtual address space, three translation lookaside buffers are used. A P-TLB is placed in a CPU and translates logical addresses to cluster addresses, which is used for accessing the main memory when the target is within the cluster. C-TLB is placed in an MBP and translates cluster addresses to global net-addresses, which is used for accessing remote memories, message passing. N-TLB is placed in an MBP and translates global net-addresses to cluster addresses. N-TLB also act as capability lists for protection on global memory accesses.

3 Processor and Secondary Cache

A processing element (PE) consists of a RISC processor, SuperSparc, and a secondary cache. SuperSparc has on-chip primary caches, a 5-way set associative 20 K-byte instruction cache, and a 4-way set associative 16 K-byte data cache. Both caches are physically-addressed and each cache line of them can be invalidate by a request from off-chip memory interface. These features and the write policy of the data cache, write-thorough and no write-allocate, give us a free hand of secondary cache configuration and its coherence control. Partial store ordering model also enables us to adopt weak ordering memory consistency scheme.

The secondary cache is 1 M-byte, direct mapping, write-back, unified, snoop cache. It has sophisticated mechanisms not only for multicache coherence control, but also for interprocessor communication and synchronization.

3.1 Coherence Control

The secondary cache supports both of two major coherence control protocols, write-invalidate and write-update. Programmers, compilers and/or operating system can specify one of these protocols as a page attribute according to the usage of data in the page. The attribute is cached as a part of cache tag.

Each line of the secondary cache has one of the following five states for multicache coherence control.

- invalid
- exclusive, dirty
- locally-shared, clean
- locally-shared, dirty
- globally-shared, clean

The states with *locally-shared* mean that copies of the line are only in a cluster, while a *globally-shared* line may be shared by two or more clusters. This distinction will reduce the load of MBP, because it can ignore a write for a locally-shared line and may not acknowledge the write with a bus transaction.

The load of MBP will also be reduced by inter-cache data transferring with *random reply* policy. When a PE misses its secondary cache and requests to obtain a line, every cache having the line tries to respond to the request as soon as it become free from its processor's requests. Then the cache with smallest load will reply the line, while others will cancel their trials. This policy has advantage over that with ownership concept to fix a cache responsible for the reply, because of better load balancing and shorter response time.

3.2 Communication and Synchronization

As described in the following section, MBP provides various operations for interprocessor communication and synchronization. The performance of these operations, however, would be limited because of the distance from a processor to MBP. Therefore, we introduced caching mechanisms for two important schemes, I-structure and FIFO, in order to reduce access latency of them[5].

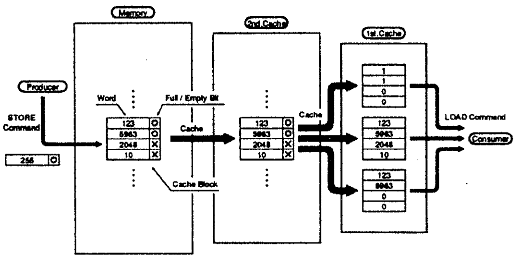

Figure 4: Cached I-structure

Each word in the secondary cache has a *full/empty* bit to indicate the presence of a valid data. Since the processor doesn't has such an additional bit, a special command, specified in a part of physical address, is available to load the bit as a data. Moreover, another load command performs a predefined action if the word is empty, while it obtains the valid data in the full case. The action for the empty case, specified as a page attribute, is loading a special data pattern or raising an interrupt synchronous to the load operation. As shown in Figure 4, the response data of ordinary and special load operations are cached in the primary data cache to minimize the latency.

The presence bit is also used to cache the words at the FIFO top. The secondary cache tries to prefetch eight words from a FIFO and gives each word to the processor bypassing the primary cache as the response of a special *dequeue* command. Since the presence bit of a prefetched word is turned empty if the word is beyond the FIFO bottom, a dequeue causing underflow will result the same as the load of an empty I-structure. Another type of FIFO is *packet FIFO* whose entry is a data packet up to 32-byte. The dequeue operation for this type FIFO removes the top packet and moves it to a buffer area. Since the buffer can be cached into both primary and secondary cache, the latency of the access to the contents of a packet is minimized. Moreover, the latency of moving the top packet will be hidden by a small prefetch buffer in the secondary cache.

4 Memory-Based Processor

Memory-Based Processor (MBP) is a fine-grained processing element for global operations that include management of memory consistency, memory based synchronization, message handling and user-level fine-grain operations. As shown in figure 5, an MBP is connected to L2 cache memory through a common bus, an RDT router through a network

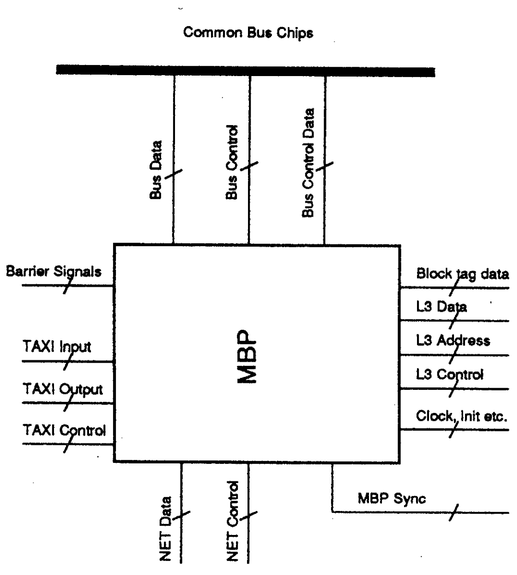

Figure 5: MBP connections

interface processor and I/O links. In this section, the function and the construction of the MBP are outlined. Detailed description of the MBP is found in [7, 8].

An MBP has following functions to complement coarse-grained CPU:

Strategic Memory System Management
Address translation, protection, data transfer and snooping operation to L2 cache memory, memory consistency preservation among clusters by a pseudo-fullmap directory scheme and accesses to remote memories are included in strategic memory system management [8]. The consistency protocol based on the pseudo-full map directory utilizes the hierarchical construction of the interconnection network for reducing update/invalidate requests on the interconnection network and for reducing the length of the entry of the directory.

Memory based synchronization
Memory based synchronization is the mechanism to implement I-structures [10], FIFO queues, memory barriers and Fetch and OP primitives on each memory location. In the JUMP-1, both L2 cache memories and main storages have a synchronization tag on each memory location and implement memory based synchronization.

Implementation of an elastic barrier
Although the name space of memory based barriers is virtual memory address space, the name space of the elastic barrier is the processor space because elastic barrier is supported by hardware. Since the number of processors in the JUMP-1 is too large for flat implementation of the barrier, elastic barrier is realized through RDT network, in which barrier operations are combined at higher levels in RDT network. MBP interfaces barrier requests from CPUs to RDT network router and manages partitioning of elastic barriers.

Cache injection
Cache injection facility (CIF) is a cache mechanism to inject data without a demand from a CPU request [1]. Cache injection is further divided into deterministic cache injection triggered by allread / allwrite cache protocols and speculative cache injection that is initiated by an MBP for realizing data-driven operation. In both cases, the target of cache injection is a block already on the cache or an empty block. The combination of cache injection facility and memory based synchronization can reduce network traffic related to processor communications.

Management of message passing operations

At an MBP, all the requests is interpreted as a message. An MBP receives messages from a CPU through L2 cache, from an RDT router through an NIP and from an I/O link. The MBP responds to a message in two different ways:

1. When the message invokes MBP activities, the message is written in an MBP message buffer and the corresponding MBP instruction pointer is added to MBP context queue.
2. When the message invokes CPU activities, the MBP tries to inject the message to L2 cache memory and modifies the active context queue by memory based synchronization operation.

Thread management for coarse-grained processing elements
Since locality is the key to archive high performance on coarse-grained processing elements, a sophisticated thread management that cannot be realized by a simple hardware context queue is indispensable. The JUMP-1 uses memory based FIFO queues as context queues

that is managed by an MBP. The basic principle of thread scheduling is based on Snoopy Spin-Wait method proposed in [12].

Macro Dataflow computations
In macro dataflow computations, an MBP functions as a matching memory, L2 cache works as an active macro-node queue by cache injection, and a coarse-grained processing elements execute macro-node and output tokens to subsequent nodes. Pipeline bubbles caused by non-firing input tokens are completely eliminated in the JUMP-1 by concurrent execution of MBP and CPU.

Management of an RDT router
An RDT router requires initialization, establishment of a partition, set up of routing information and network clearing. Since an RDT router does not have managing capability, all these primitives are realized by an MBP connected to that router.

Execution of user-level fine-grain programs
Other than basic primitives motioned above, an user can run his fine-grain programs on MBP. Garbage-collection, transaction processing are examples of user-defined fine-grain programs. Programs are stored in a main memory and all the working spaces are also reserved in a main memory.

Figure 6 shows the block-diagram of the MBP. Among MBP functions listed above, those which are frequently used and have large influence on performance are implemented in MBP hardware function-blocks and other functions are realized by MBP-core programs. The following functions are selected for implementing by MBP hardware function-blocks:

Shared bus interface
Shared bus interface unit controls block-data transfer between L2 cache, cache injection and snooping protocols. Regular memory accesses are serviced by this unit.

Address translation
Successful cases of cluster addresses to net-addresses translation are performed by C-TLB hardware. Net-addresses to cluster addresses are translated by N-TLB. Housekeeping operations for C-TLB and N-TLB is realized by MBP-core programs.

Consistency preservation between clusters
Broadcast / selectable multi-cast communication necessary for the pseudo-full-map directory scheme is realized by inter-cluster access managing unit. Since RDT does not preserve communication orders, inter-cluster access acknowledge managing unit and per-processor access ordering unit control access order and collection of acknowledge signals for memory consistency protocols.

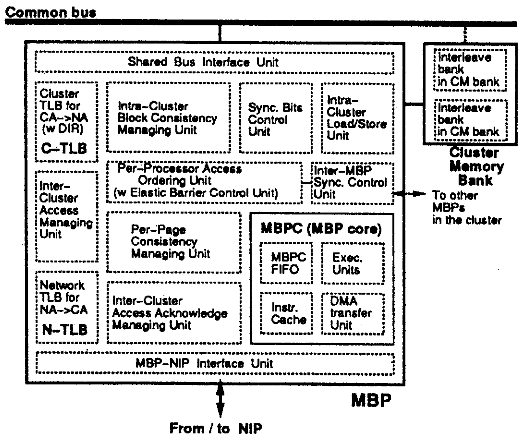
Common bus
Shared Bus Interface Unit
Cluster TLB for CA->NA (w DIR)
C-TLB
Intra-Cluster Block Consistency Managing Unit
Sync. Bits Control Unit
Intra-Cluster Load/Store Unit
Interleave bank in CM bank
Interleave bank in CM bank
Cluster Memory Bank
Inter-Cluster Access Managing Unit
Per-Processor Access Ordering Unit (w Elastic Barrier Control Unit)
Inter-MBP Sync. Control Unit
To other MBPs in the cluster
Per-Page Consistency Managing Unit
MBPC (MBP core)
MBPC FIFO
Exec. Units
Network TLB for NA->CA
N-TLB
Inter-Cluster Access Acknowledge Managing Unit
Instr. Cache
DMA transfer Unit
MBP-NIP Interface Unit
MBP
From / to NIP

Figure 6: MBP Block diagram

Basic memory based synchronization
Successful access in I-structures and other simple memory based synchronization primitives are processed in sync-bits control unit. Unsuccessful access in I-structures and other complicated primitives are processed by MBP-core programs.

Basic primitives for I/O links
DMA operations and hardware buffer controls for a TAXI network are implemented by MBP hardware function. Error handling primitives are realized by MBP-core programs.

MBP-core is a processor optimized to very short thread, network interface and memory with synchronization tags. The main objectives of the MBP is to emulate MBP functions that is not directly implemented by MBP hardwired function-blocks and execute user fine-grained programs. Detailed description of MBP-core architecture is described in [9].

5 Interconnection Network: RDT

The following properties are required for the interconnection network used in the JUMP-1. (1) Two dimensional torus structure can be easily emulated because parallel applications have been also developed in the JUMPP project on the torus connected multicomputer AP1000[13]. (2) In order to manage the distributed shared memory system with the hierarchical full-map directory, inherent tree structure with multiple roots is required. (3) Diameter must be smaller than that of hypercube with reasonable fixed number of degree.

To cope with these requirements, we proposed a novel interconnection network called the Recursive Diagonal Torus (RDT)[14].

The RDT is defined on a two-dimensional nearest neighbor torus ($N \times N$) which is called the base torus or rank-0 torus.

Here, let be the four additional links between node (x, y) and nodes $(mod(x \pm n, N), mod(y \pm n, N))$ on the rank-i torus. The resulting network is a torus called rank-(i+1) torus. n is called the **cardinal number**, and here, we set it 2. As an example, a rank-1 torus and a rank-2 torus are shown in Figure 7(a) and (b) respectively. Note that 8 independent upper toruses are formed on a torus.

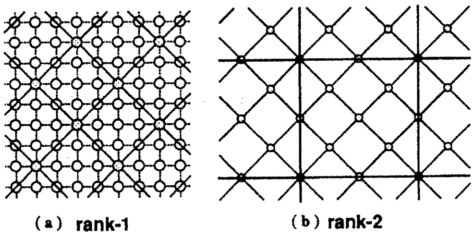

Figure 7: The RDT theory of organization

Recursive Diagonal Torus **RDT(n,R,m)** is a class of networks in which each node has links to form base (rank-0) torus and m upper toruses (the maximum rank is R) with the cardinal number n (here, n = 2). Note that, each node can select different rank of upper toruses from others.

The RDT in which every node has links to form all possible upper toruses (RDT(n,R,R)) is called the perfect RDT (**PRDT(n,R)**) where n is the cardinal number (here, n=2) and R is the maximum rank. Although the PRDT is unrealistic because of its large degree ($4(R+1)$), it is important as a basis for establishing routing algorithm, broadcasting, and other message transfer algorithms on the RDT.

The JUMP-1 must be scalable to the system with ten thousand nodes (for example, array of 128×128 nodes or 256×256 nodes). In this case, m is set to be 1 (degree = 8). For this number of nodes, the maximum rank of upper toruses is 4. Thus, the RDT(2,4,1) is treated here.

In the RDT, each node can select different rank toruses from others. Thus, the structure of the RDT(2,4,1) also varies with the rank of toruses which are assigned to each node. This assignment is called the *torus assignment*. Various torus assignment strategies can be selected considering the traffic of the network. If the local traffic is large, the number of nodes which have low ranks should be increased. However, complicated torus assignment introduces difficulty to the message routing algorithm and implementation. For the JUMP-1, we selected a a relatively simple torus assignment shown in Figure 8.

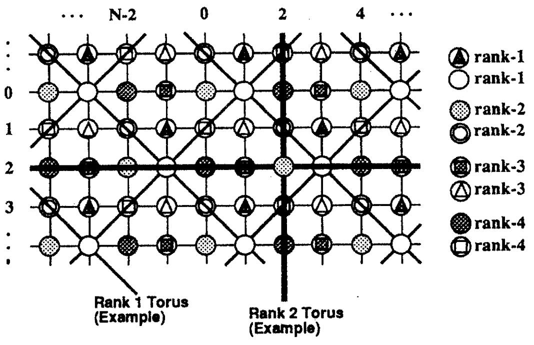

Figure 8: Torus assignment used in JUMP-1

The RDT provides the following features for the JUMP-1.

- A simple routing algorithm called the vector routing, which is near optimal and easy to be implemented, enables smaller diameter than that of the hypercube (11 for 2^{16} nodes) with smaller degree (8 links per node).
- Using its inherent hierarchical structure in upper toruses, the distributed multicast required for the hierarchical full-map directory can be efficiently implemented. Trees and the hypercube connection are easily emulated. The FFT and the bitonic sorting algorithms are also easy to implement.
- With the best use of the redundant structure, fault tolerant techniques can be easily applied on the RDT.

Precise definitions and discussions on the RDT are described in [14][15].

6 I/O System

6.1 Image and file I/O subsystems

The I/O system consists of image I/O and file I/O subsystems. The image I/O subsystem supports high quality image processing for scientific visualization. The file I/O subsystem provides a large amount of non-volatile data storage.

In vector-type supercomputers, I/O systems are commonly connected with a single high speed I/O channel, such as HIPPI, or an internal bus. In the JUMP-1 system, this approach would cause communication bottleneck at a specific cluster connected to the I/O channel when the other clusters request a large number of I/O operations. Thus we took an approach based on not a single centralized I/O channel but a number of distributed I/O links as shown in Fig. 9. Each link is composed of high speed serial communication LSI and FIFO as described below.

For example, in order to display a high-vision image (1920 x 1035 pixels per frame) on a monitor in real time (30 frames/sec.), a centralized channel is required to have high transmission bandwidth (about 180 MBytes/sec.). However, by dividing a channel into several links, say 32 links, the bandwidth can be reduced to about 5.6 MBytes/sec (45 Mbps). Several serial links from clusters are combined into frame buffers for image input/output, then image data are transferred to a image I/O devices such as an HDTV monitor or a camera through a fast video bus. As in an image I/O interface, a disk I/O interface is connected to JUMP-1 clusters via many fast serial links, then data from disk storage devices are transferred by an I/O controller through a SCSI bus.

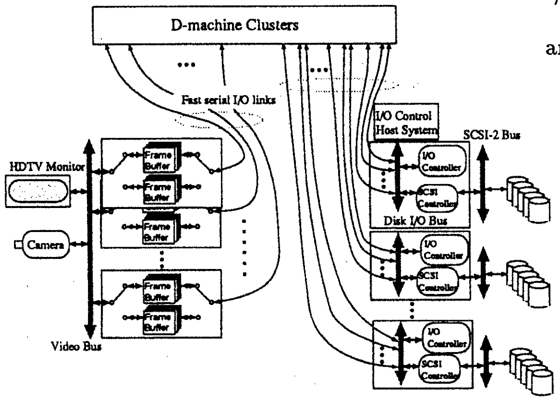

Figure 9: Organization of an I/O system

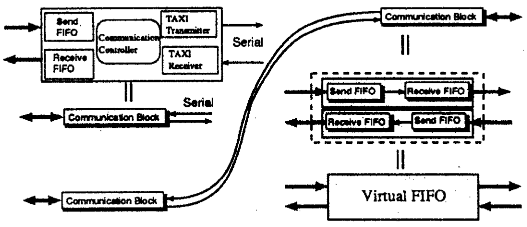

Figure 10: Virtual-FIFO organization

Since only connecting via fast serial communication links between an I/O interface and clusters cannot achieve a high performance I/O system, an efficient communication organization among them has to be implemented. Thus, we propose a Virtual-FIFO mechanism for an efficient I/O communication organization as described below.

6.2 Virtual-FIFO mechanism via fast serial communication

A configuration of Virtual-FIFO mechanism is described in Fig. 10. A communication block consists of fast serial communication LSI (AMD TAXI) chip sets for transmitting and receiving, Send- and Receive-FIFOs and a communication controller which controls asynchronous communication by using a simple Xon/Xoff protocol. By connecting two communication blocks via a twisted pair or coaxial cable, a bi-directional first-in first-out facility is constructed between them. Combining some Virtual-FIFOs implements I/O subsystem which holds wide range I/O bandwidth and is flexible for I/O cable length.

Merits in I/O communication via Virtual-FIFOs are shown as follows.

- Relaxing distance constraint: Since the numbers of processing elements grows, an overall system configuration becomes larger, in considering physical distance between an I/O system and clusters, connecting via fast serial links is one of the most effective implementations for massively parallel computing systems.
- Enhancing communication throughput: A communication sequences consists of five phases, (a)writing into Send-FIFO (b)parallel-serial transforming (c)serial transmission (d)serial-parallel transforming (e) reading from Receive-FIFO. Since these five phases

can be overlapped in Virtual-FIFO organization, a communication throughput can be enhanced.

7 Performance Evaluation

7.1 INSIGHT : An Interconnection Network Simulator

An interconnection network simulator, called INSIGHT is developed to evaluate the performance of various interconnection networks toward realization of massively parallel computers [18][19].

INSIGHT provides salient features to investigate the desired interconnection network using *network description language* which is used to specify the characteristics of the interconnection network such as its topology, flow control, channel width, and others.

INSIGHT aims to evaluate the performance of an interconnection network for massively parallel computers containing thousands or tens of thousand processor elements. INSIGHT performs simulation based on the specifications of the target interconnection network written in network description language. Furthermore, the various communication patterns obtained from the execution of parallel programs are used in the simulation providing practical application tests to the interconnection network.

INSIGHT is useful tool for the performance evaluation and the support of development of desired interconnection network, because of that it can modify its parameters which is needed for simulations.

7.2 Performance Analysis of RDT

In this section, we will consider four types of interconnection networks having 4096 nodes without considering the bisection width, i.e., the RDT(2,4,1), the 2-dimensional torus (64×64), the 3-dimensional torus (16×16×16), and the 12-dimensional hypercube.

In the implementation, the simulation parameters for each interconnection network are set as follows:

Flow control method : store and forward

Routing method : deterministic routing

Channel width : 32 bits with bidirectional connection links

Packet size : 128 bits (Header size : 64 bits, Data size : 64 bits)

Data transfer frequency : 50 MHz

These are on the assumptions that data transfer between channels can be completed within 1 clock period, the routing process time in each node is set to 40 nanoseconds (i.e., 2 clock periods) assuming that decision making for the next node takes one clock period, and to pass through the exchange switch is also one clock period.

In this simulation, a message which is given a randomly generated destination node has been created with message size ranging from 4 bytes to 128 bytes. Then, we obtain the average network latency of the first generated 10,000 messages at 2 microseconds interval, and at increasing message creation interval.

The plot of the average network latency as a function of message creation interval for the RDT, 2-dimensional torus, 3-dimensional torus, and the 12-dimensional hypercube are shown in Figure 11.

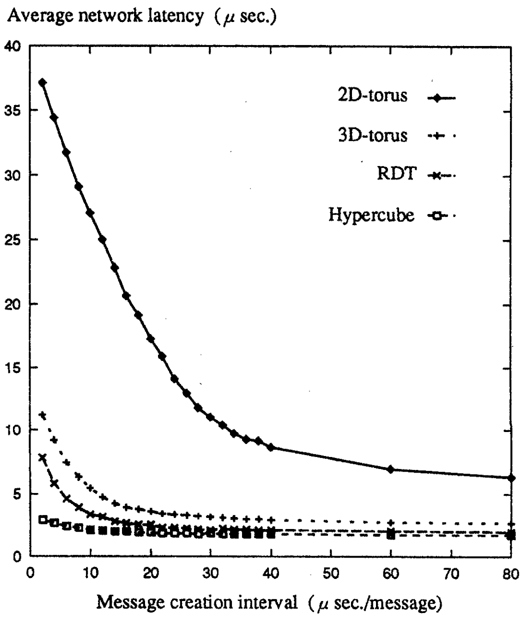

Figure 11: Message creation interval vs. Average network latency.

The 2-dimensional torus has a high latency because there are only few connection links per node and collision often occurs. In contrast, the hypercube which is a high-dimensional topology has low latency. The RDT(2,4,1) also has low latency, similar to the hypercube, in spite of the few communication links per node. Great improvement on the latency is clearly observed on the RDT(2,4,1) with low hardware cost.

8 Concluding Remarks

The architecture of the JUMP-1, a massively parallel computer prototype, has been described. This paper has also given unique features of its key components, the intelligent secondary cache, memory management co-processor MBP, interconnection network RDT, and I/O system with TAXI. The performance evaluation results, obtained from the network simulator INSIGHT, prove the efficiency of the RDT.

The architectural and functional design of the JUMP-1 has been finished, and we are now pursuing the logic design of four ASIC chips for each components. The system will be completed in the first quarter of 1995.

Acknowledgments

We would like to express our thanks to Prof. S. Tomita, Prof. H. Tanaka, Prof. Y. Kaneda, and all research members of the JUMPP project for their support.

A part of this research was supported by the Grant-in-Aid for Scientific Research on Priority Areas, #04235103, from the Ministry of Education, Science and Culture.

References

[1] T. Matsumoto, and K. Hiraki. Cache Injection and High-Performance Memory-Based Synchronization Mechanisms. In *IPSJ SIG Notes*, Vol. 93, No. 71, ARC-101-15, pp. 113–120, 1993. (in Japanese)

[2] T. Matsumoto. Fine-Grain Support Mechanisms. In *IPSJ SIG Notes*, Vol. 89 , No. 60, ARC-77-12, pp. 91–98, 1989. (in Japanese)

[3] T. Matsumoto, T. Tanaka, T. Moriyama and S. Uzuhara. MISC: A Mechanism for Integrated Synchronization and Communication Using Snoop Caches. In *Proc. of the 1991 Int. Conf. on Parallel Processing*, Vol. 1, pp. 161–170, 1991.

[4] T. Matsumoto and K. Hiraki. A Shared-Memory Architecture for Massively Parallel Computer Systems. In *IEICE Japan SIG Reports*, Vol. 92, No. 173, CPSY 92-26, pp. 47–55, 1992. (in Japanese)

[5] M. Goshima, C. Okada, T. Hosmoi, S. Mori, H. Nakashima and S. Tomita. The Intelligent Cache System for Fine-Grain Inter-Processor Communication. In *IPSJ SIG Notes*, Vol. 93, No. 71, 93-ARC-101, pp. 121–128, 1993. (in Japanese)

[6] T. Hosomi, S. Mori, H. Nakashima and S. Tomita. A Hardware Cache Conherence Scheme with the Assistance of Software. In *IPSJ SIG Notes*, Vol. 93, No. 20, 93-ARC-99, pp. 117–124, 1993. (in Japanese)

[7] T. Matsumoto. A Multiprocessor System with Memory-Based Processors and Register-Based Processors. In *Proc. of 43th Annual Convention of IPS Japan*, Vol. 6, 6Q-3, pp. 115–116, 1991. (in Japanese)

[8] T. Matsumoto and K. Hiraki. Distributed Shared-Memory Architecture Using Memory-Based Processors. In *Proc. of Joint Symp. on Parallel Processing '93*, pp. 245–252, 1993. (in Japanese)

[9] K. Hiraki and T. Matsumoto. Compsite Parallel Processing Architecture with Two different processing element with different grain size. In *IEICE Technical Reports*, Vol. 90, No. 144, CPSY 93-42, pp. 25–30, 1993.

[10] Arvind and R. A. Iannucci. A Critique of Multiprocessing von Neumann Style. In *Proc. 10th Int. Symp. on Computer Architecture*, pp. 426–436, 1983.

[11] K. Li. IVY: A Shared Virtual Memory System for Parallel Computing. In *Proc. 1988 Int. Conf. on Parallel Processing*, pp. 94–101, 1988.

[12] T. Matsumoto. Synchronization and Processor Scheduling Mechanisms for Multiprocessors. In *IPS Japan SIG Reports*, Vol. 89, No. 99, ARC-79-1, pp. 1-8, 1989. (in Japanese)

[13] H. Ishihata, T. Horie, S. Inano, T. Shimizu, S. Kato and M.Ikesaka, Third generation message passing computer AP1000. In *Proc. of the International Symposium on Supercomputing*, pp. 46–55, 1991.

[14] Y. Yang, H. Amano, H. Shibamura and T. Sueyoshi. Recursive Diagonal Torus: An interconnection network for massively parallel computers. To appear in *Prof. of the 5th IEEE Symposium on Parallel and Distributed Processing*, 1993.

[15] Y. Yang, H. Amano, H. Shibamura and T. Sueyoshi. Characteristics of the Recursive Diagonal Torus: an Interconnection Network for Massively Parallel Computers. In *IEICE Techinical Reports*, CPSY93-26, Aug. 1993. (in Japanese)

[16] H. Nakajo, M. Kohata and Y. Kaneda. An Implementation of a Distributed Image Generation System using a Fast Serial Link. *IEICE Technical Reports*, CPSY93-33, pp. 39–46, 1993. (In Japanese)

[17] Advanced Micro Devices. Am79168/Am79169-275 TAXI-275 Transmitter/Receiver Preliminary Data Sheet. 1993.

[18] H. Shibamura, M. Kuga and T. Sueyoshi. Development of an Interconnection Network Simulator for Massively Parallel Computer. In *IPSJ SIG Reports*, 92-ARC-97, pp. 121–128, 1992. (in Japanese)

[19] H. Shibamura, M. Kuga and T. Sueyoshi. An Interconnection Network Simulator for Massively Parallel Computers. In *Proc. of the Joint Symposium on Parallel Processing*, pp. 159–166, 1993. (in Japanese)