To appear ISPAN'94

Overview of the JUMP-1, an MPP Prototype for General-Purpose Parallel Computations

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], 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

We describe the basic architecture of JUMP-1, an MPP prototype developed by collaboration between 7 universities. The proposed architecture can exploit high performance of coarse-grained RISC processor performance in connection with flexible fine-grained operation such as distributed shared memory, versatile synchronization and message communications.

1 Introduction

Recent progress in semiconductor technology enables practical use of MPP systems in numerical intensive computations. However, current MPPs are still very inefficient for general-purpose computations. Our main goal is to construct a truly general-purpose massively parallel processor that covers application fields not only for vector super-computers but also for main-frames and workstations. Potential application fields for our MPP system include:

- Numerical intensive computations,
- Graphics, scientific visualization and animation.
- Pattern recognition and robotics,
- Databases and OLTP, and
- Discrete problems.

Almost all the current commercial highly parallel processing systems are based on distributed memory architecture and use message passing as an only method to communicate between processing elements. MPP systems with shared memory architecture have various advantages over those with distributed memory architecture as follows:

1. Shared memory architecture allows natural description of problems including natural description of parallelism and synchronization.
2. Recent optimizing techniques for parallel processing require prefetching, thread-migration and remote-caching. Optimized compilers potentially have more opportunity to improve performance in shared memory systems than in distributed memory systems.
3. Shared memory systems allow more flexible operating environments such as data / process migration, dynamic change in partitioning and dynamic load distribution.
4. Shared memory system provides more efficient and more powerful protection mechanism based on memory access protection for remote accesses and messages.

Currently, three different shared memory architectures are proposed: centralized shared memory systems, bus-connected snooping memory systems and distributed shared memory systems. The former two organizations are not scalable and excluded for MPP systems.

Locality in memory references are required in distributed shared memory organization for efficient computation. But an application program which does not have any locality in memory references is also not suitable for MPP systems of any kinds. Therefore, we can adopt this limitation for practical computations. Furthermore, a shared memory system has advantage in process and/or data migration capability which can exploit dynamic locality that cannot be utilized in distributed memory systems.

The next issue is selection of proper protocols. Almost all the current shared memory systems use either invalidate or update protocol for consistency preserving operations. However, both protocols have their own advantages and disadvantages and one protocol cannot solve all the situations in application programs. The update protocol is essential to accomplish inter-cache communication but that also has false-sharing problem. We have proposed a new coherent protocol that dynamically switch appropriate protocols [1, 2].

The structure of directory which holds sharing information is important to reduce network traffic for coherent memory operations. We have proposed a pseudo-fullmap directory method [9] that dynamically switch between a hierarchical fullmap directory method, a point-to point directory method and an ordinary fullmap directory method.

The main objective of above two methods are to provide flexible interface to compilers and operating systems for optimizing system performance. Although we can keep the length of threads in application programs fairly long by fusing a number of threads necessary for optimizing communication, above mentioned methods requires programmable memory management operations whose thread length is very short. For example, a global read operation may requires access to the page table (or TLB) and sending remote-read request, a global write operation requires sending remote-write request, sending invalidate or update requests and possibly change the contents of directory and participate memory barrier synchronization.

Thus, a processing element(s) in a node executes both long threads (application) and very short threads (memory operation in globally shared locations). A single processor architecture cannot efficiently executes programs that is abundant in globally shared memory operations. We have proposed a complementary processor architecture (CPA) for efficient execution of both coarse-grained local computations and fine-grained global computations.

2 Efficient distributed shared memory architecture

As stated in the previous section, realization of efficient distributed shared memory (DSM) system is one of the most important issues for constructing a general-purpose MPP system. The basic components for a distributed shared memory are organization of cache memory, organization of cache directories, address translation mechanism and consistency preserving operations [5, 3, 4, 6]. But distributed shared memory systems so far is less efficient than distributed memory systems made of the corresponding hardware technology. Consequently, it is still used only as a testbed of research works. We have proposed a new distributed shared memory system, Strategic Memory System (SMS) for improving performance of a distributed shared memory system and construct a general-purpose shared memory MPP system. Main features of SMS are as follows:

- Globally shared virtual address space and two-stage TLB implementation.
- Directory attached to every page but data transfers by a cache block.
- Pseudo-fullmap directory cache memory based on a hierarchical broadcasting network.

- Dynamic consistency protocol that can be optimized by the access pattern of an application program.
- Caching directory information to TLB for accelerating consistency preserving operations.
- Integrate synchronization and consistency preserving operations for reducing network traffic and instruction overheads.

Detailed description of SMS is shown in [12].

3 JUMP-1 architecture

JUMP-1 [8] adopts clustered architecture (figure 1) as a basic architecture. Clustered architecture has several good properties for massively parallel systems because (1) clustered architecture reduces the cost of an 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.

As shown in figure 1 JUMP-1 consists of 256 processor clusters and three different networks. The first network is Recursive Diagonal Torus 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 flexible I/O controlers 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 processor contexts (registers and system resources). The characteristics of an MBP is to complement a CPU for short thread fined-grained computations with frequent context switches [11]. Since there are no commercial fine-grained MPU, the MBP is a custom design processor with associated support hardware as described later. A L2

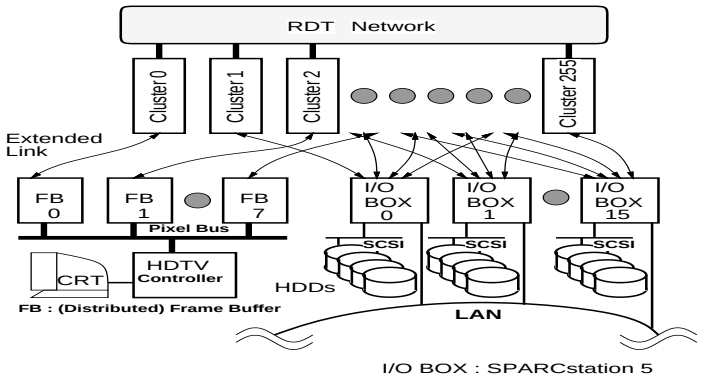

Figure 1: Global architecture of JUMP-1

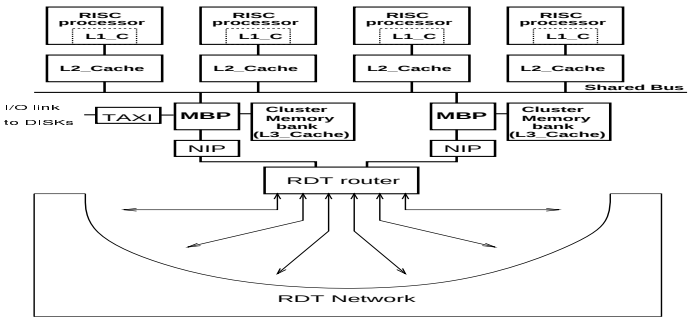

Figure 2: Block diagram of a cluster

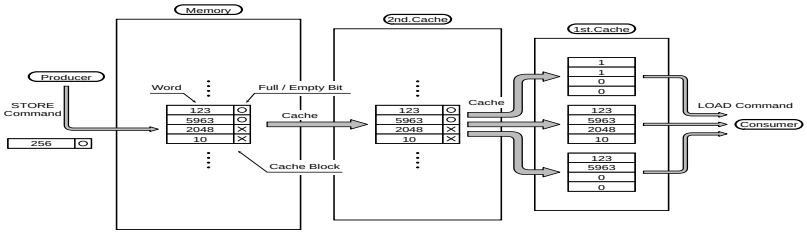

Figure 3: Cached I-structure

cache memory is a secondary cache memory from CPU and the target of cache injection [7] 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 implementing I-structures [10], 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[13]. An I/O network interface is a high-speed serial transmitter/receiver (TAXI) with message buffers.

3.1 Secondary Cache

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

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 multi-cache 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. 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[14].

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 3, the response data of ordinary and special load operations are cached in the primary data cache to minimize the latency.

3.2 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. An MBP is connected to L2 cache memory through a common bus, an RDT router through a network 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 [12].

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

Strategic Memory System 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 [12]. 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 JUMP-1, both L2 cache memories and main storages have a synchronization tag on each memory location and implement memory based synchronization.

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 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 is a cache mechanism to inject data without a demand from a CPU request. 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.

Thread management Since locality is the key to archive high performance on coarse-grained processing elements, a sophisticated thread management that cannot be re-

alized by a simple hardware context queue is indispensable. JUMP-1 uses memory based FIFO queues as context queues that is managed by an MBP.

Execution of user-level 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.

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
- Address translation
- Consistency between clusters
- Basic memory based synchronization
- Basic primitives for I/O links

3.3 Interconnection Network: RDT

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 JUMP1 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 JUMP1, we selected a a relatively simple torus assignment shown in Figure 4. 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 [15][16].

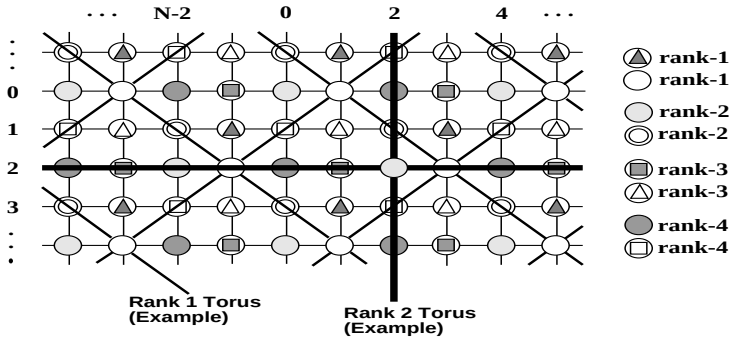

Figure 4: Torus assignment used in JUMP-1

3.4 I/O System

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.

We took an approach based on not a single centralized I/O channel but a number of distributed I/O links (STAFF link [17]. Each link is composed of high speed serial communication LSI and FIFO as described below.

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 STAFF link mechanism for an efficient I/O communication organization as described below.

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 STAFF link implements I/O subsystem which holds wide range I/O bandwidth and is flexible for I/O cable length.

4 Concluding Remarks

In this paper, we discuss the importance of flexible distributed shared memory in a MPP system for general-purpose computations. The main features of JUMP-1 memory system are:

1. Flexible consistency-protocol which dynamically switch protocols.
2. 2 level consistency protocol: snoopy cache within a cluster and the directory based method among clusters.
3. Pseudo-fullmap directory method that combines a hierarchical fullmap method.
4. Efficient synchronization protocols that is based on the fusion of consistent protocol and synchronizing structure.
5. Powerful and flexible realization of sharing and protection by three level addressing and three TLBs.
6. Efficiently implemented by fine-grained Memory-Based Processor.

Currently, JUMP-1 is under construction and will complete in the next year.

Acknowledgments

We would like to express our thanks to Prof. H. Tanaka, Prof. S. Tomita, 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] Matsumoto, T., "Fine-Grain Support Mechanisms", IPS Japan SIG Reports, Vol.89 No.60, ARC–77–12, pp.91–98 1989 (In Japanese).

[2] Matsumoto, T. et al., "MISC: A Mechanism for Integrated Synchronization and Communication Using Snoop Caches", Proc. of the 1991 Int. Conf. on Parallel Processing, Vol. 1, pp.161–170 1991.

[3] Agarwal, A., et al., "An Evaluation of Directory Schemes for Cache Coherence", Proc. 15th Int. Symp. on Computer Architecture, pp.280-289 1988.

[4] Chaiken, D. et al.,"Directory-Based Cache Coherence in Large-Scale Multiprocessors", IEEE Computer, Vol.23 No.6, pp.49-58 1990.

[5] Li, K.," IVY: A Shared Virtual Memory System for Parallel Computing", Proc. 1988 Int. Conf. on Parallel Processing, St. Charls, IL, pp.94-101 1988.

[6] Warren, D.H.D. and Haridi, S., "Data Diffusion Machine-a scalable shared virtual memory multiprocessor", Int. Conf. on Fifth Generation Computer Systems 1988, ICOT 1988.

[7] Matsumoto, T. and Hiraki, K., "Cache Injection and High-Performance Memory-Based Synchronization Mechanisms", IPS Japan SIG Reports, Vol.93 No.71, ARC-101-15, pp.113-120 1993 (In Japanese).

[8] Hiraki, K., Amano, H., Kuga, M., Sueyoshi, T., Kudoh, T., Nakashima, H., Nakajo, H., Matsuda, H., Matsumoto, T. and Mori, S., "Overview of a Massively Parallel Computer Prototype: JUMP-1", IPS Japan SIG Reports, ARC-102-10, pp.73-84 1993.

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

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

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

[12] Matsumoto, T. and Hiraki, K., "Distributed Shared-Memory Architecture Using Memory-Based Processors", Proc. of Joint Symp. on Parallel Processing '93, IPSJ/IEICE/JSSST, pp.245–252, 1993 (In Japanese).

[13] Matsumoto, T.,"Elastic Barrier: A Generalized Barrier-Type Synchronization Mechanism", (in Japanese). Trans. of IPS Japan, Vol.32 No.7, pp.886–896 (July 1991).

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

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

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

[17] Okada, T., Nakajo, H., Matsumoto, T., Kohata, M., Matsuda, H., Hiraki, K. and Kaneda, Y., "An I/O Access Method for the Massively Parallel Computer JUMP-1", IPS Japan SIG Reports, ARC-107-23, pp.177-184 1994.