The Preliminary Evaluation of MBP-light with Two Protocol Policies for A Massively Parallel Processor – JUMP-1 –

Inoue Hiroaki† Ken-ichiro Anjo† Junji Yamamoto∗ Jun Tanabe†
Masaki Wakabayashi† Mitsuru Sato‡ Hideharu Amano† Kei Hiraki††
†Keio University ∗Real World Computing Partnership ‡Fujitsu Laboratories ††The University of Tokyo
3-14-1 Hiyoshi, Kouhoku, Yokohama, Kanagawa, Japan 223-8522
e-mail: mbp@aa.cs.keio.ac.jp

Abstract

A massively parallel processor called JUMP-1 has been developed to build an efficient cache coherent-distributed shared memory (DSM) on a large system with more than 1000 processors. Here, the dedicated processor called MBP (Memory Based Processor) -light to manage the DSM of JUMP-1 is introduced, and its preliminary performance with two protocol policies –update/invalidate– is evaluated.

From results of its simulation, it appears that simple operations like the tag check and the collection/generation of acknowledgment packets are mostly processed by the hardware mechanisms in MBP-light without aids of the core processor with both policies.

Also, the buffer-register architecture adopted by the core processor in MBP-light is exploited enough to process a protocol transaction for both policies.

1. Introduction

A Cache Coherent Non-Uniform Memory Access machine (CC-NUMA) is one of hopeful candidates for future common high performance machines. Unlike bus-connected multiprocessors, the system performance can be enhanced scalably as to the number of processors. Moreover, parallel programs developed in small multiprocessors can be transported easily.

A number of CC-NUMA systems have been developed: Stanford DASH[10] / FLASH[8], MIT Alewife[1], SGI Origin2000[9] and the Sequent NUMA-Q[11] are representatives. Such systems work efficiently with tens or hundreds of processors. However, when thousands of processors are connected, a large amount of memory and hardware are required to manage the DSM

JUMP-1 is a prototype of a massively parallel processor with cache coherent DSM developed by collaboration of 7 Japanese universities[4]. The major goal of this project is to establish some techniques required to build an efficient DSM on a massively parallel processor. A lot of novel techniques are introduced in the DSM of JUMP-1 for this purpose. In order to satisfy both high degree of performance and flexibility, a dedicated processor called MBP(Memory Based Processor)-light is proposed to manage the DSM of JUMP-1. MBP-light [5] consists of a simple core processor and hardwired controllers which handle memory systems, bus and network packets.

Various types of cache coherence protocols can be utilized on the DSM of JUMP-1, including an update policy which has never been implemented on traditional CC-NUMA systems. Although this type of protocol requires a lot of packet transfers, it can be useful for some applications which aggressively access the shared data. MBP-light provides dedicated hardware mechanisms to support implementation of such protocols. In this paper, two protocol policies implemented in JUMP-1 are described and evaluated.

2. A Massively Parallel Processor – JUMP-1 –

As shown in Figure 1, JUMP-1 consists of 256 clusters connected each other with an interconnection network called RDT(Recursive Diagonal Torus)[15]. The RDT includes both torus and a kind of fat tree structure with recursively overlayed two-dimensional square diagonal tori structure. Each cluster provides a high speed point to point I/O network connected with disks and high-definition video devices.

Each cluster is a bus-connected multiprocessor, as shown in Figure 2, including four RISC processors (SuperSPARC+), MBP-light which is directly connected to a cluster memory, and RDT router chip for interconnection network[13]. MBP-light, the heart of JUMP-1 cluster, is the custom designed processor which manages DSM, synchronization, and packet handling.

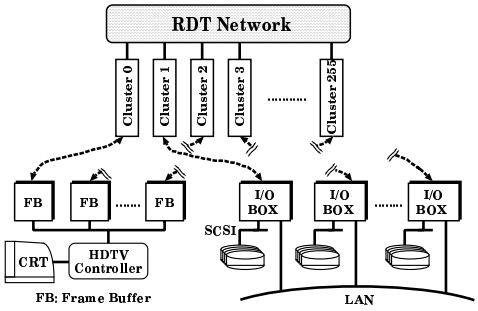

Figure 1. The Structure of JUMP-1

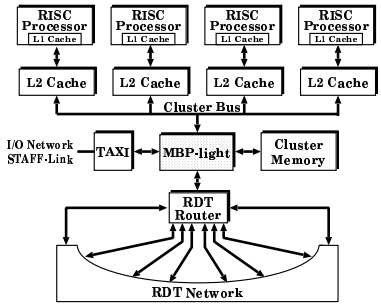

Figure 2. The Structure of JUMP-1 Cluster

In traditional CC-NUMAs – DASH / FLASH, Alewife and NUMA-Q –, the DSM is managed in a cache line size, and data in other clusters is copied into a cache attached to each processor. The consistency protocol is a simple invalidate policy, the interconnection network is a simple mesh or ring, and the directory scheme is based on one-to-one data transfer.

Although such a mechanism works efficiently in those systems with a limited number of processors, it is not suitable for a system with thousands of processors. For example, a large amount of memory for cache and directory is required. The invalidate policy based on one-to-one data transfer often causes a network congestion when many processors share the same data.

In order to address these problems, the following methods are used in JUMP-1.

1) Each processor (SuperSPARC+) shares a global virtual address space with three-stage TLB implementation. The directory is attached not to every cache line but to every page, while the data transfer is performed by a cache line. Some parts of cluster memory are available as L3 (Level-3) cache which stores the copies of other cluster memory.

2) Various types of cache coherence protocols can be utilized dynamically, including not only an invalidate policy but also an update policy. In traditional CC-NUMA systems, an update policy has never been implemented since it requires a lot of packet transfers. However, it can be useful for some applications which require frequent data exchange by accessing shared data aggressively. For efficient implementation of such an update protocol, MBP-light provides dedicated hardware mechanisms to multicast network packets and collect acknowledgment packets.

3) Reduced Hierarchical Bitmap Directory schemes (RHBDs) are introduced[7] to manage directory efficiently. The hierarchical structure of RDT is suitable for an efficient implementation of the RHBD.

4) Each processor provides a custom sophisticated snoop cache as the L2 cache. Various cache protocols including cache injection are supported by this chip. A relaxed consistency model is implemented using write buffers in the L2 cache[2]

The detail schemes of DSM management for JUMP-1 are described in [12].

3. The Structure of MBP-light

3.1. The Design Policies

In protocol processors for the recent CC-NUMA, – the MAGIC of FLASH and the SCLIC of NUMA-Q – a packet is split into the header and the data part by a hardwired logic. A powerful core processor treats the header part, while a hardwired logic treats the data part which is only transferred between buffers. When the packet is sent again, the header and data part are also quickly combined by a hardwired logic.

Unlike such a traditional method, the following design policies are adopted in JUMP-1.

1) Generation and collection of acknowledgment packets must be done quickly to introduce an update policy. Therefore, they are managed with a dedicated hardware mechanisms. Another custom logic is provided to check tags in the cluster memory.

2) All processes mentioned above require a complicated large hardwired logic. Thus, a simple 16-bit core processor (MBP Core) is introduced to reduce the total hardware requirement. Although the core processor is simple, a packet can be processed quickly by adopting the *buffer-register architecture* which can treat a packet buffer as a special 68-bit register.

Figure 3 shows the structure of MBP-light depending on above policies. MBP-light consists of three modules: RDT

Interface to treat network packets, MMC (Main Memory Controller) to control cluster memory and cluster bus, and MBP Core which is the core processor. RDT Interface and MMC provide their own hardware mechanisms, and work independently from MBP Core.

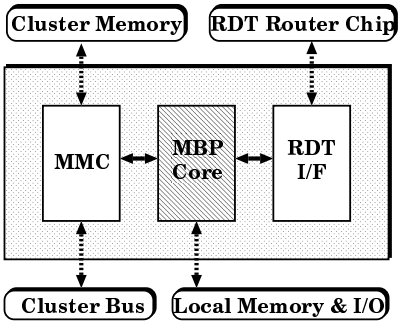

Figure 3. The Structure of MBP-light

3.2. MMC

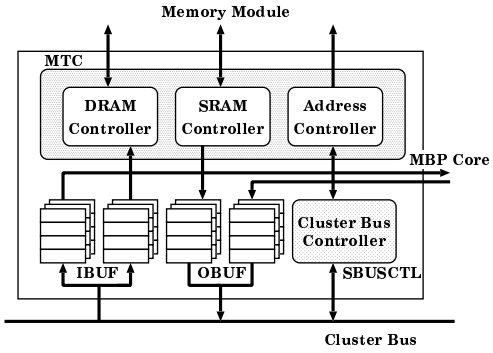

Figure 4. The Structure of MMC

MMC (Main Memory Controller) manages the cluster memory which consists of SDRAM for data and SRAM for tags. It also controls the cluster bus which connects four SuperSPARC+ processors with the L2 cache. When a processor misses the L2 cache, the cluster memory must be accessed and MMC checks the tag. Depending on the status of a cache line, MMC interrupts to MBP Core, and the software on the core processor is invoked.

As shown in Figure 4, it consists of the cluSter BUS ConTroLler (SBUSCTL) which manages cluster bus and packets, Input/Output BUFfer (I/O BUF) to store bus packets and Memory Timing Controller (MTC) which controls read from/write to SDRAM and SRAM.

3.3. RDT Interface

RDT Interface is directly connected with RDT router chip and manages network packet transfer.

The update policy requires a lot of network packets since a large number of processors tend to share a cache line. For avoiding network congestion, packets must be multicast in the network (one-to-one transfer is so inefficient). Since RDT network used in JUMP-1 provides the hierarchical multicast mechanism, it can be done without network congestion[7]. For a protocol processor, a fast generation and collection of acknowledgment packets are essential. RDT Interface provides two dedicated mechanisms: Ack Generator and Ack Collector for this purpose, as shown in Figure 5.

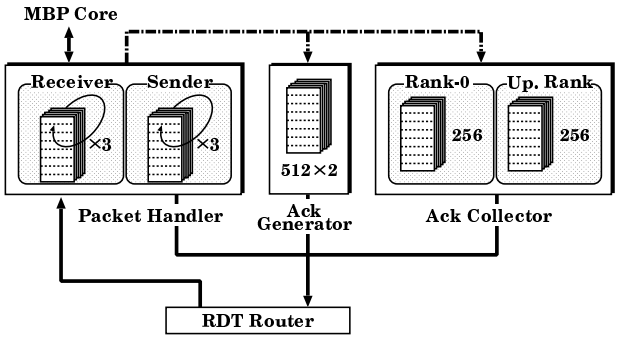

Figure 5. The Structure of RDT Interface

Ack Generator provides two cache systems called Net Cache and Ackmap Cache. Both caches are accessed when a packet with coherent message is received. Net Cache, a direct mapped cache with 512 entries, is accessed by the address of the DSM in the packet header. It stores the information whether the accessed line is cached in the cluster (in L2 or L3 cache) or not. At the same time, Ackmap Cache is accessed by the source cluster number of the receiving packet, and the bitmap which shows the returning path is obtained. Using the above information, an acknowledgment packet when the accessed data is cached, or a not-acknowledgment packet when the accessed data is not cached is automatically generated.

On the other hand, when an acknowledgment packet is received, another cache system called Ack Cache in Ack Collector is accessed by the key in the packets. Ack Cache is a direct map cache which provides 128 entries for each hierarchy of the embedded tree in RDT network. The number of packets which must be collected is registered in the cache entry, and the number is decremented when a packet arrives. When the number becomes zero, another acknowledgment packet for the upper hierarchy is generated, or MBP Core is interrupted.

In both cache systems, if a miss occurs, the program of the MBP Core will be interrupted for replacing or generating the entry. In this case, the performance is much degraded.

3.4. MBP Core

3.4.1 The Buffer-Register Architecture

The structure of MBP Core is shown in Figure 6. MBP Core consists of a pipeline with four stages treating 21-bit instructions and 16-bit data. 21-bit $\times$ 64K local memory which stores instructions and local data is connected. The MBP Core takes the I/O mapped approach, and another 64K address is provided for I/O devices and the dedicated hardware for the barrier operation. The MBP Core also provides the 16-bit $\times$ 256 internal memory which is used for the table jump and to store bitmaps for an acknowledgment packet.

Since jobs which must be quickly processed are mostly managed by the hardware mechanisms in RDT Interface and MMC, MBP Core only processes a complicated part of the DSM protocol. It mainly decodes a packet, accesses a table, transforms the address, and generates the packet to send somewhere. The header of a packet in JUMP-1 is sometimes complicated and occupies several flits of the packet. Also, tags included in the data of a packet relate to a protocol control. Therefore, it is convenient to treat a packet buffer as a register. However, since the width of a packet buffer is 68-bit, it requires an enormous hardware to treat such buffers as common general purpose registers in the processor.

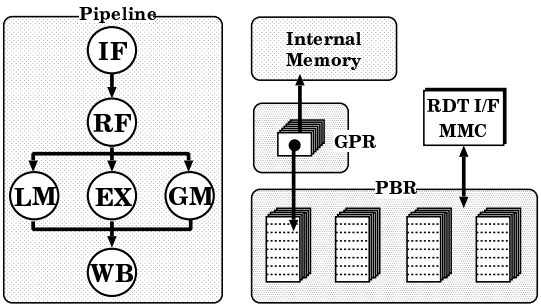

Figure 6. The Structure of MBP Core

To solve this problem, MBP Core provides 16 GPRs (General Purpose Registers) of 16-bit width and 112 PBRs (Packet Buffer Registers) of 68-bit width. The PBR is indicated by the content of the GPR, and accessed in the processor pipeline like a common register. While operations and data transfer between PBR and GPR or PBR and PBR are allowed, the content of the PBRs is transferred directly as a packet from/to MMC or RDT Interface. Since operations are mainly done between such a packet buffer and a register, we call this structure the *buffer-register architecture*. As an example of this architecture, the ADD operation between GPR and PBR (*i.e.* `ADDPG R1 R2(0)`) is shown in Figure 7.

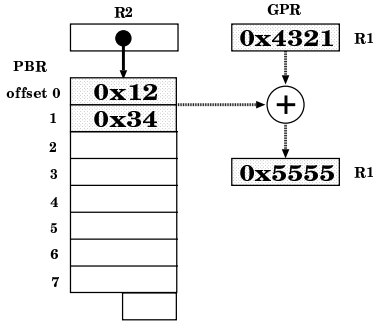

Figure 7. the ADD operation between GPR and PBR

Table 1. The Classification of MBP Core Instructions

Class	Type
WGG	operate between GPR and GPR
LMA	access to local memory
BRANCH	branch
WGI	operate between GPR and Imm.
BPI	operate between PBR and Imm.
RDT	control RDT Interface
MPP	transmit from PBR to PBR
WPG	operate between PBR and GPR
MMC	control of MMC
TJ	table jump
INT	control interrupt
IMA	access to internal memory
SPE	special instructions
NOP	no operation

Finally, the classification of instructions on MBP Core is shown in Table 1. MBP Core has about 85 instructions with 14 classes. In addition to the instructions described above, MBP Core has some instructions which control RDT Interface or MMC. Since it also has a dozen special instructions to process a protocol transaction quickly, it has a lot of advantages over a general RISC processor.

4. Chip Implementation

MBP-light is implemented on the Toshiba's 0.4 μm C-MOS 3-metal embedded array TC203E340. In order to cope with a large number of pins, TBGA (Tape Ball Grid Array) package is used.

The design of MBP-light is described in VHDL, synthesized with Mentor's Autologic-II, and verified with Toshiba's VLCAD. Using the behavior level simulator, a program of MBP Core for a simple protocol has been developed in parallel with the hardware design. The specification of MBP-light is shown in the Table 2.

The layout of MBP-light is shown in Figure 8. A lot of embedded RAMs are placed near four edges of the die surrounding random logics in the middle square part. A large RAMs are corresponding to the cache memory in RDT Interface, while small ones are used for PBRs.

Table 2. The Specification of MBP-light

Maximum clock (MHz)	50
Random logics	106,905
Internal memory(bits)	44,848
Area utilization(%)	38.3
The number of pins	352
Consuming Power (W)	3.1

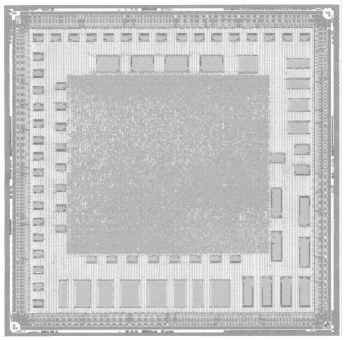

Figure 8. The Layout of MBP-light

5. Evaluation of MBP-light

5.1. Evaluation Method

Now, two clusters of JUMP-1 are available, and a simple protocol handler is implemented on MBP-light. The protocol handler supports two policies: invalidate and update policies. Since the program of MBP Core for the packet handler is so complicated, the description of the handler implementation is omitted here. The detail protocol is described in [2].

The performance is evaluated by a simple trace driven simulator. The trace is generated by an instruction-level simulator called (ISIS)[6]. It is a flexible simulation framework which can generate access traces of fully pipelined RISC processors. Here, the hit ratio of an instruction cache is assumed to be 100 % and only simple one-to-one data transfer is used in RDT network. Both update policy and invalidate policy distinguished in JUMP-1 are used in the evaluation. L2 cache is a direct mapped 1M-byte cache with 32-byte cache line. Although the L2 cache provides a lot of sophisticated functions, they are omitted in this simulation.

The latency parameters of the simulator are set as shown in Table 3.

Table 3. Latency Parameters

event	clocks
PU $\leftrightarrow$ L2 cache	1
L2 cache $\leftrightarrow$ CBUS	5
MMC $\leftrightarrow$ MBP	3
MBP $\leftrightarrow$ MBP	24

As benchmarks, four programs: FFT given 64K points, LU given 128-by-128 matrix, RADIX given 2M elements and OCEAN given 66-by-66 grids from SPLASH-2 benchmark suits[14] are selected. Since the trace generation takes an enormous computing time and disk storage, systems with up to 32 processors (8 clusters) are evaluated here.

5.2. Evaluation of Hardwired Mechanisms

5.2.1 The Hit Ratio of Net Cache and Ack Cache

First of all, we evaluate how often Net Cache of Ack Generator in RDT Interface hits with both policies. When the RDT packet arrives, an entry of Net Cache is accessed. If it hits, RDT Interface generates an acknowledgment packet. If not, RDT Interface must interrupt to MBP Core to make a new entry. It shows the ability to generate an acknowledgment packet in RDT Interface without invoking MBP Core. When it hits, the acknowledgment packet is returned only with 11 clocks (220 nsec) instead of more than 100 clocks processed by MBP Core.

The hit ratio of Net Cache with both policies are shown in Table 4 and 5 where CL means "clusters". The hit ratio with invalidate policy is overall low except OCEAN which has good spatial locality, while that with update policy is very high except RADIX which has little temporal locality[14].

Table 4. The Hit Ratio of Net Cache with Invalidate Policy

Appli.	hit ratio		
	2 CL	4 CL	8 CL
FFT	76.3 %	63.2 %	49.2 %
LU	91.5 %	91.0 %	88.0 %
RADIX	48.0 %	22.7 %	19.7 %
OCEAN	97.8 %	98.2 %	98.8 %

They show that an acknowledgment packet with update policy is almost automatically generated and MBP Core is interrupted rarely by RDT Interface. However, the number

of network packets which interrupt to MBP Core is actually important.

Table 5. The Hit Ratio of Net Cache with Update Policy

Appli.	hit ratio		
	2 CL	4 CL	8 CL
FFT	99.9 %	99.5 %	92.1 %
LU	99.9 %	99.9 %	99.9 %
RADIX	45.8 %	38.7 %	38.1 %
OCEAN	99.4 %	99.7 %	99.8 %

Then, Figure 9 shows the comparison of the number of network packets with two policies where regards that with update policy as 100 %. Also, the colored bar means the hit ratio.

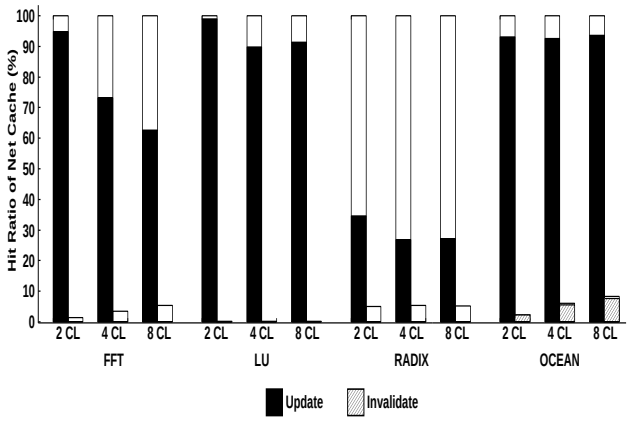

Figure 9. The Comparison of The Hit Ratio of Net Cache with Two Policies

The number of network packets with the invalidate policy is extremely smaller than that with the update policy since a large part of transactions can be processed with only communication inside the cluster. In other words, the dedicated mechanism of RDT Interface with the update policy is efficiently used. However, its ability with the update policy, is insufficient to suppress the number of network packets, compared to that with invalidate policy. Maybe, more clusters are required to be the same number of packets with both policies.

Next, the hit ratio of Ack Cache in RDT Interface is also an important issue, since MBP Core software is also invoked to collect acknowledgment packets for an invalidate/update request. Unlike the hit ratio of Net Cache, it is almost 100% with both policies in all applications. In most cases, acknowledgment packets are immediately returned and collected. Once all packets are collected, the entry is removed. Thus, the entry is not conflict unless the invalidate/update requests are frequently issued. Thus, most acknowledgment packets are collected only with 5 clocks (100 nsec) by Ack Cache controller.

As illustrated above, RDT Interface can execute a large part of protocol processing job with both policies without invoking MBP Core.

5.2.2 The Wake-Up Ratio of MBP Core from MMC

When a packets is issued on a cluster bus, MMC checks tags and judges whether the software on MBP Core is required or not. If MMC can process a packet, it takes about 10 clocks instead of more than 1000 clocks processed by MBP Core.

Here, the frequency of invoking MBP Core software called *wake-up ratio* is evaluated. The wake-up ratio is defined as

$$\frac{\text{the number of packets MMC can't handle}}{\text{the number of packets MMC receives}} .$$

The average wake-up ratio is shown in Table 6 and 7. As shown in these tables, the ratio with the update policy is about 30% independent from applications. However, the ratio with the invalidate policy depends heavily on each application and it increases in proportion to the number of clusters.

Table 6. The Wake-Up Ratio with Invalidate Policy

Appli.	wake up ratio		
	2 CL	4 CL	8 CL
FFT	17.7 %	36.3 %	50.0 %
LU	1.9 %	4.3 %	17.8 %
RADIX	40.4 %	54.2 %	59.8 %
OCEAN	12.5 %	28.0 %	38.1 %

Table 7. The Wake-Up Ratio with Update Policy

Appli.	wake up ratio		
	2 CL	4 CL	8 CL
FFT	33.2 %	33.6 %	33.1 %
LU	30.7 %	31.8 %	33.3 %
RADIX	36.2 %	36.3 %	36.0 %
OCEAN	30.8 %	30.8 %	29.6 %

The ratio with the invalidate policy is relatively higher than that with the update policy since most of packets used in the invalidate policy are caused by the coherence miss. In order to compare two policies in detail, the number of packets which interrupt to MBP Core should be evaluated.

As shown in Figure 10 where regards the number of packets with update policy as 100 % (the colored bar means the hit ratio), that with the invalidate policy is extremely

smaller than that with the update policy. In other words, although the ratio which invokes MBP Core is high to treat coherent misses with the invalidate policy, the number of packets which MMC can't handle is much smaller than that with the update policy. It comes from the frequent data exchange caused by the update policy. However, even with the update policy, MMC can execute a large part of protocol processing without invoking MBP Core.

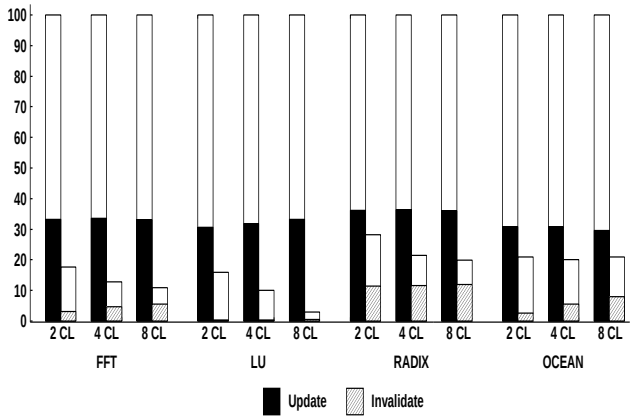

Figure 10. The Comparison of The Wake-Up Ratio with Two Policies

5.3. Evaluation of Instruction Set Architecture

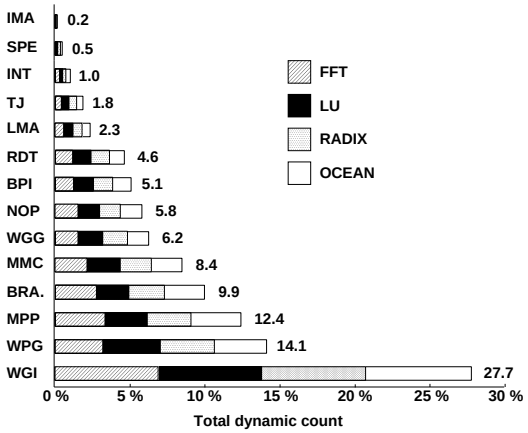

Figure 11. The Instruction Mix with Invalidate Policy

Here, the program running on MBP Core to process a protocol is decoded after MBP Core is interrupted by MMC or RDT Interface. In order to demonstrate the efficiency of MBP Core architecture, the instruction mix with two policies on 8 clusters is analyzed.

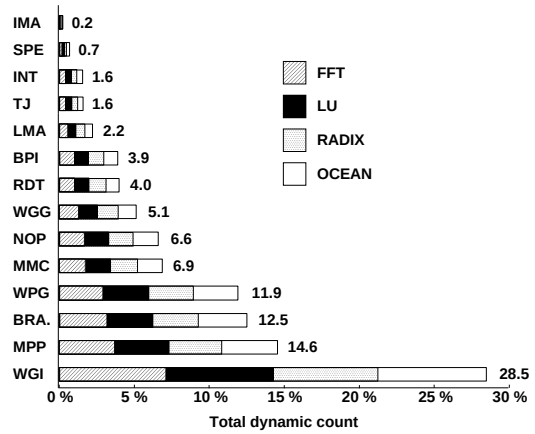

Figure 12. The Instruction Mix with Update Policy

As shown in Figure 11 and 12, frequently used instructions are almost the same in every application with both policies. Instruction classes related to *buffer-register architecture* are WPG, MPP and BPI in Table 1. The total ratio of those classes with invalidate policy is 31.5%. On the other hand, that with update policy is 30.3%. From these results, instructions for the buffer-register architecture adopted by MBP Core is efficiently used for both policies.

In our former study [5], we compared the core architecture with DLX[3] like 32 bit general RISC processor. The performance of the core processor is about 20% better than that of 32bit RISC processor in spite of a small hardware requirement because of the buffer-register architecture.

5.4. The Performance Comparison

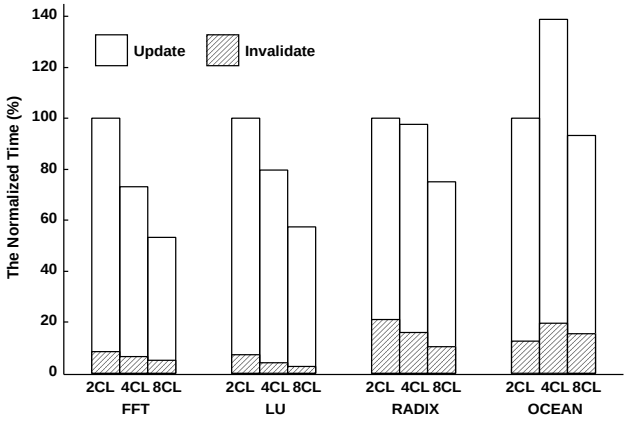

Figure 13. Normalized Time for Applications with Two Policies

Although the time for protocol management programs is not exactly modeled, we measured the performance of JUMP-1 with both policies. Figure 13 shows the preliminary evaluation results of execution time normalized to the time when update policy is used on two clusters. From this result, the execution time of update policy is much worse than invalidate policy in all applications. The overhead of update policy is caused by the protocol management program executed on MBP Core invoked with frequent packets arrival. However, in this simulation, the multicast mechanism in JUMP-1 is not effectively used and protocol management program on MBP Core is not well optimized for the update policy.

With both policies, the tendency of speedups is almost the same. In FFT and OCEAN, the performance is improved when the number of clusters increases, while it is not well improved in RADIX and LU. We will optimize protocol programs, and evaluate the system with larger number of clusters.

6. Summaries

A dedicated processor called MBP-light to manage the DSM in JUMP-1 is introduced, and its performance with two protocol policies are evaluated.

Its simulation appears that simple operations like the tag check and the collection/generation of acknowledgment packets are mostly done by the hardware mechanisms without MBP Core with both policies.

Also, a simple 16-bit RISC called the buffer-register architecture is adopted in MBP-light. It is greatly exploited for complicated protocol transactions with both policies.

The preliminary evaluation shows that the execution time with update policy is still much worse than that with invalidate policy. Since the multicast mechanism of JUMP-1 is not used in this simulation and the system scale is limited, the extensive simulation research is required.

The prototype of JUMP-1 with 16 processors is scheduled to be available within this year. We will optimize the protocol program on the prototype.

7. Acknowledgments

The authors would like to express their sincere gratitude to the members of the Joint-University project for their valuable advises and discussions. The authors also express their thanks to Mentor Graphics Japan for supporting design tools.

A part of this research was supported by the Grant-in-Aid for Scientific Research on Priority Areas, #04235130, from the Ministry of Education, Science and Culture. This research is also supported by Parallel and Distributed system Consosium.

References

[1] D. Chaiken and A. Agarwal. Software-Extended Coherent Shared Memory: Performance and Cost. In *Proc. of The 21st IEEE International Conference on Computer Architecture*, pages 314–324, April 1994.

[2] M. Goshima et al. The Intelligent Cache Controller of a Massively Parallel Processor JUMP-1. In *Proc. of Innovative Architecture for Future Generation High-Performance Processors and Systems*, pages 116–124, October 1997.

[3] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, San Francsico, California, second edition, 1996.

[4] K. Hiraki et al. Overview of the JUMP-1, an MPP Prototype for General-Purpose Parallel Computat ions. In *Proc. of IEEE International Symposium on Parallel Architectures, Algorithms and Networks*, pages 427–434, December 1994.

[5] I. Hiroaki et al. MBP-light: A Processor for Management of Distributed Shared Memory. In *Proc. of The 3rd International Conference on ASIC*, pages 199–202, October 1998.

[6] T. Kisuki et al. Shared vs. Snoop: Evaluation of Cache Structure for Single-Chip Multiprocessors. In *Proc. of The 3rd International Euro-Par Conference*, pages 793–797, August 1997.

[7] T. Kudoh et al. Hierarchical bit-map directory schemes on the RDT interconnection network for a massively parallel processor JUMP-1. In *Proc. of International Conference on Parallel Processing*, volume I, pages 186–193, August 1995.

[8] J. Kuskin et al. The Stanford FLASH Multiprocessor. In *Proc. of The 21st IEEE International Conference on Computer Architecture*, pages 302–313, April 1994.

[9] J. Laudon and D. Lenoski. The SGI Origin: A ccNUMA Highly Scalable Server. In *Proc. of The 24th IEEE International Conference on Computer Architecture*, pages 241–251, June 1997.

[10] D. Lenoski et al. The Stanford DASH Multiprocessor. *IEEE Computer*, 25(3):63–79, March 1992.

[11] T. D. Lovett and R. M. Clapp. STiNG: A CC-NUMA Computer System for the Commercial Marketplace. In *Proc. of The 23rd IEEE International Conference on Computer Architecture*, pages 308–317, May 1996.

[12] T. Matsumoto et al. Distributed Shared Memory Architecture for JUMP-1: A General-Purpose MPP Prototype. In *Proc. of IEEE International Symposium on Parallel Architectures, Algorithms and Networks*, pages 131–137, June 1996.

[13] H. Nishi et al. The RDT Router Chip: A versatile router for supporting a distributed shared memory. *IEICE transaction on Information and Systems*, E80-D(9):854–862, September 1997.

[14] S. C. Woo et al. The SPLASH-2 Programs: Charaterization and Methodological Considerations. In *Proc. of The 22nd Annual Symposium on Computer Architecture*, pages 24–36, June 1995.

[15] Y. Yang et al. Recursive Diagonal Torus: An interconnection network for massively parallel computers. In *Proc. of The 5th IEEE symposium on Parallel and Distributed Processing*, pages 591 – 594, December 1993.