Vol.2011-OS-118 No.7
2011/7/27

Software managed memory system for many-core architectures

Masahiro SANO[†1] and Kenji KISE[†2]

We propose an efficient software managed memory system for many-core architectures. The system has an advantage that there is no or a little additional hardware cost. In our proposal, by using management cores, it is possible to make the memory system more flexible and achieve high performance without any special hardware. A hardware support with TLB is also considered in order to achieve the further performance. In order to evaluate the performance of our proposal, we implement original system named C5. We show the effectiveness of the C5 through performance measurements with SPLASH-2 benchmark suite.

1. Introduction

A many-core processor tends to integrate a large number of cores in its single chip. In the many-core chip, it is difficult to maintain the cache coherence with only hardware. Therefore not shared memory but distributed memory architectures are used in many-core processors in general.

The conventional shared memory parallel programming is easier than distributed shared memory programming. We have many legacy codes of shared memory programming. For these reasons, there is a strong demand to use the conventional parallel programming in distributed shared memory architectures.

In order to meet this demand, there is a method called software distributed shared memory(S-DSM). S-DSM has been studied and explored mainly in cluster computers for a few decades[1)]. This method of S-DSM provides a globally shared virtual memory that all nodes in the distributed memory architecture can access.

The method of S-DSM used in cluster computers is not simply adapted to many-core architectures because of differences between hardware organizations. Therefore we propose an efficient S-DSM or software managed memory system (SMMS) for many-core architectures.

The three policies of our system are described below. (1) It minimizes the additional hardware implementation cost. (2) It supports the easy parallel programming. (3) It resolves the bottleneck of main memory access.

The rest of this paper is organized as follows. In Section 2 we state the background of this study. Our proposal is described in Section 3. In Section 4 we evaluate our proposal. Finally, conclusions are given in Section 5.

2. Background and Related Works

In cluster computers the size of a local memory (the main memory of the computer) is large enough to keep shared data of S-DSM systems. Therefore it is implemented with only single-level memory hierarchy.

On the other hand, in many-core architectures the size of an on-chip local memory is severely restricted. And an off-chip main memory is needed to keep all shared data. In this way S-DSM in many-core architectures will be a system of two-level hierarchy comprised of local memory and main memory.

From the differences of hardware structures between single-level and two-level hierarchy, the following two problems occur. (1) The method of S-DSM used in cluster computers is not simply adapted to many-core architectures. (2) The main memory performance becomes a bottleneck because the number of memory controllers is smaller than the number of nodes in many-core architectures.

We describe the related works briefly. DSMCBE[2)] for Cell Broadband Engine is a study realizing S-DSM in multi-core processors. DSMCBE maintains the coherency of entry consistency by software. Although it achieves good performance, the uniqueness of entry consistency makes the programming difficult. It has still a disadvantage that programmers should optimize the size of shared data to fit the local memory.

Rigel[3)] and Intel SCC[4)] are studies that maintain the cache coherence with software in shared memory architectures. Both studies need to use low-grained memory operations to maintain the cache coherence and the programming for these architectures is difficult.

†1 Yahoo Japan Corporation
†2 Tokyo Institute of Technology
⋆1 This work was done while Masahiro SANO was a graduate student in Tokyo Institute of Technology. This work is not related to Yahoo Japan Corporation.

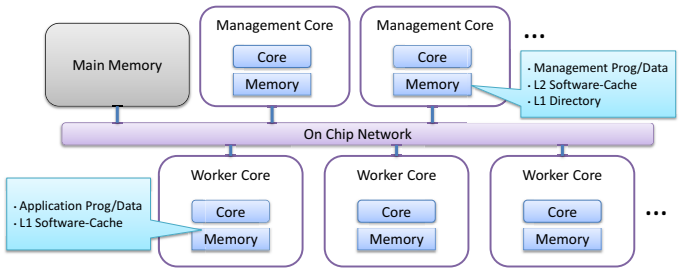

Fig. 1 System architecture of C5 software managed memory system model

Our target many-core architecture has tiled-nodes of same hardware structures. As one of the tiled-node architectures, we use M-Core[5]. Its single node has a simple processor element, a small node memory, a communication controller to communicate with other nodes, and a route that transfers packets from/to adjacent nodes. The details of M-Core are described in the paper[5].

3. The C5 Software Managed Memory System Model

In this section, we propose the software managed memory system (SMMS) model for many-core architectures named C5 based on the following three policies.

(1) It minimizes the additional hardware implementation cost.
(2) It supports the easy parallel programming.
(3) It resolves the bottleneck of main memory access.

Especially, the easy parallel programming is the key for programming of many-core architectures. In our S-DSM model, programmers need not to manage copies and communication of shared data. Furthermore, our model enables the simple porting of existing parallel programs written with P-threads.

3.1 The C5 SMMS Architecture

The system architecture of C5 is shown in **Fig. 1**. The system is comprised of three kinds of elements. There are a main memory, some management cores and many worker cores.

All cores are used as either worker core or management core. Note that we assume the homogeneous many-core processor, and the hardware configurations of worker core and management core are identical. The worker core executes an application program. The management core manages shared data by software. Using the sophisticated management of shared data on management cores, we solve two problems described in section 2. There is no additional hardware cost to implement our model because shared data management is implemented by software (it is a merit of S-DSM system).

Let's consider the first problem in section 2. (1) From the differences of hardware structures between single-level and two-level hierarchy, the method of S-DSM used in cluster computers is not simply adapted to many-core architectures.

In order to manage the shared data in our system, the management cores keep track of all shared data. The worker cores obtain shared data through management cores. By this way, this problem is resolved without additional hardware.

Let's consider the second problem. (2) The main memory performance becomes a bottleneck because the number of memory controllers is smaller than the number of nodes in many-core architectures.

All shared data is obtained through management cores. In order to reduce the number of accesses to main memory, the management cores work as L2 software cache and provide worker cores with the latest shared data.

3.1.1 Main Memory

The main memory is the memory all cores access globally. It keeps only the shared data. The shared data is read/written by management cores.

3.1.2 Worker Core and Management Core

The worker core executes an actual application program. It has all local data in order to execute the application and software cache (L1 Software-Cache) to store a part of shared data in the local memory.

The management core executes a system program to manage shared data. It does not execute an application program. In the system program, the management core waits requests from worker cores and provides services to them. The local memory of management core has (a) local data to execute the system program, (b) information of L1 cache entries (directory) and (c) software cache (L2 software-cache) to store a part of shared data.

If there is only a single management core, as the number of worker cores increases the number of accesses to the management core increases. Then it

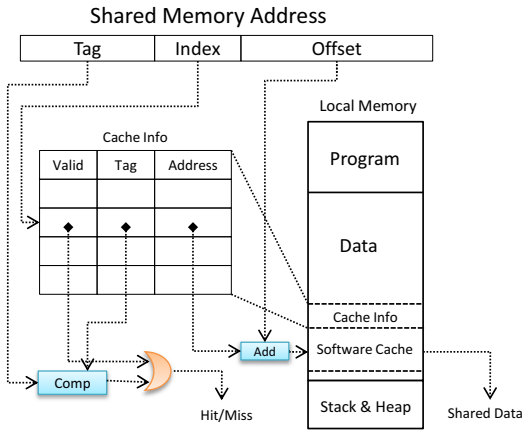

Fig. 2 Address translation mechanism from shared memory to local memory

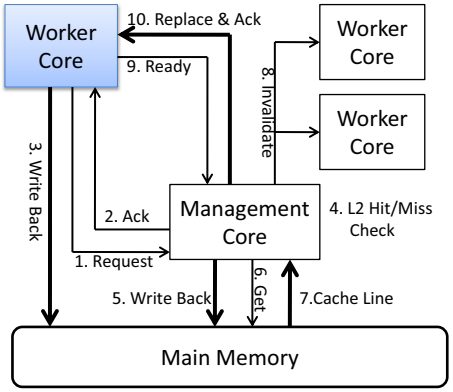

Fig. 3 Behavior of cache line replacement

prolongs the average response time for service. As a result, it diminishes the performance gain of parallel processing.

To use multiple management cores makes it possible to scatter the requests of worker cores. This distributed policy of management cores achieves good performance of parallel processing by eliminating the access concentration.

3.2 Software Cache and Synchronization Management

The cache operations to maintain cache coherence in both worker cores and management cores are implemented by software.

The directory based protocol is used to maintain the cache coherence. The write-invalidate policy is used as write propagation scheme.

As a consistency model, release consistency[6] is used. We select a multiple reader and a single writer protocol. In this protocol, multiple writes by some worker cores to shared data which belongs to same cache line at same time are unacceptable.

In general, release consistency provides easy programming and enough scalability compared with other consistency models. The release consistency enables the programmers to reuse of existing parallel programs written with P-threads.

3.2.1 Shared Memory Access

In order to access to the shared memory in a worker core, we provide a special system function. In this function, the shared memory space is translated to the local memory space.

Fig. 2 shows the translation mechanism on worker core. A shared memory address is translated into a local memory address of accessed software cache line. By accessing the cache tag and valid bit, the function detects the cache hit or miss.

If the valid cache line exists in L1 software-cache, the worker core successfully loads from or stores to the translated address.

If the valid cache line does not exist in L1 software-cache, in order to obtain the line the worker core sends a request of cache line replacement to a management core. The worker core waits until the requested line is obtained.

The worker core loads from or stores to the translated address as soon as possible when the response which represents the requested process has been done from the management core is obtained.

3.2.2 Cache Line Replacement

The cache line replacement occurs if there is no valid L1 software-cache line in the local memory when a worker core accesses to shared memory space.

Each cache line is allocated to the one of management cores in address interleaved way. So, the management core to which a worker core sends the request depends on the shared memory address of the access miss.

Fig. 3 shows a behavior of cache line replacement. This is handled through (1) to (10) processes in order. (1) The worker core sends a request to a management core that is in charge of the shared memory address of the access miss. (2) The management core sends an acknowledgment back. (3) The worker core which receives the acknowledgment writes back the victim cache line to the main memory. (4) The management core confirms the requested cache line exists in the L2 software-cache. (5) The management core writes back a victim cache line to the main memory if needed. (6) The management core gets the requested cache line from the main memory. (7) The main memory sends the line to the management core. (8) The management core invalidates the corresponding cache line on other worker cores because the cache line is updated. (9) The management core waits the acknowledgment that represents the write back has done from the worker core. (10) The management core replaces the cache line of the worker core by data transfer and sends an acknowledgment after the replacement. When the data is available on the worker core, it continues to execute the application program.

Note that if the worker core does not write back, the processes of (2), (3) and (9) are skipped. If the management core does not write back, the process (5) is skipped.

3.2.3 Lock Primitive

The lock primitive for parallel programming is managed by the specific single management core. When a worker core acquires a lock, the worker core sends a request with a corresponding lock variable to the management core. The management core confirms whether the lock has already been acquired or not.

If the lock has not been acquired and it is available, the management core sends a response that indicates the lock has been acquired to the worker core.

If the lock is not available, the management core stores the request. The worker core does not continue the rest of application processes until the lock is acquired.

3.2.4 Unlock Primitive

A worker core writes back dirty cache lines when it releases a lock to the management core. At the time, the worker core sends an unlock request to all management cores which are in charge of the dirty cache lines.

Note that in order to release the lock after all cache lines are write back, the lock release request to the management core that controls locks is sent at the end.

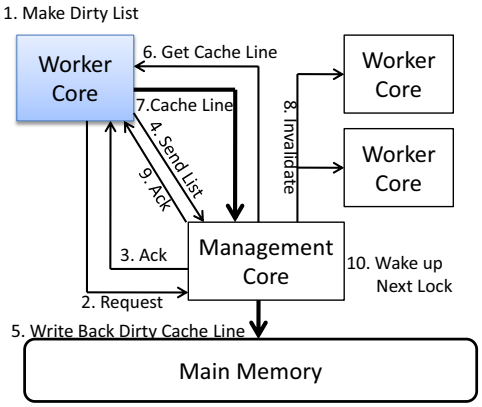

Fig. 4 Behavior of unlock primitive operation

Fig. 4 shows a behavior of unlock operation. (1) The worker core makes a list of dirty cache lines in the L1-software cache. (2) The worker core sends an unlock request to the management core. (3) The management core sends an acknowledgment back to the worker core. (4) The worker core sends the dirty line list. (5) The management core writes back the cache lines on the dirty line list. (6) The management core gets the cache lines on the dirty line list from the worker core. (7) The worker core sends the dirty cache lines to the management core. (8) The management core invalidates other worker cores which have the cache lines on the dirty list. (9) The management core sends an acknowledgment to the worker core which sends the unlock request. (10) The management core sends a response of a lock request to next worker core waiting for the lock.

Note that the process (10) is done only on the management core which controls locks.

3.2.5 Barrier Primitive

Like the lock management, the barrier primitive for parallel programming is managed by the specific single management core.

A worker core sends a barrier request to the management core that manages

barriers. The management core sends a response that represents the barrier has been done to all worker cores as soon as the management core received barrier requests from all worker cores.

Each worker core continues the application program when it receives the response. In our model, the shared data is not updated in the barrier primitive. The barrier is just used to synchronize the control flows.

3.3 Hardware Support for Fast Shared Memory Access

In the architecture discussed in the previous section, shared memory access is handled by only software. Therefore the latency of the shared memory access is large even if the access hits in L1 cache. The measured average L1 hit latency is about 35 cycles.

In order to speed up the shared memory access, we propose the architecture using TLB(Translation Look-aside Buffer). The following two processes are handled by hardware using TLB. (1)A cache line hit/miss detection. (2)A translation from shared memory address to local memory address. With these hardware supports, the access latency of L1 cache hit becomes 1 cycle. Note that the most processor has TLB and these hardware supports require few additional hardware.

With the hardware support with TLB, shared memory space is accessed by not a special function but load/store instruction. We defined that the target address space depends on most significant bit(MSB) of the address accessed by load/store.

The MSB of 0 represents the access to local memory space. The local memory address is not translated by TLB and is simply loaded/stored.

The MSB of 1 represents the access to shared memory space. On TLB hit, the shared memory address is translated to local memory address where the valid cache line stays by TLB. On TLB miss, an exception occurs and arranges the contents of TLB.

In our target MIPS architecture, the three types of TLB exceptions described as below occur and are handled by software.

TLB Refill Exception There is no valid entry in TLB. The valid entry is loaded from a page table.

TLB Invalid Exception The corresponding entry is invalid. Replace the cache line.

TLB Modified Exception The corresponding entry is write-protected. The entry is updated to writable.

Each entry of the page table has following three items. (1)A local memory address of a cache line in the local memory. (2)A shared memory address of the cache line. (3)Flags such as a validity of the cache line. The size of the page table is small $^{\star 1}$ enough to save in the local memory of each worker core.

When we use the hardware support with TLB, a problem occurs. It is hard to satisfy the constraint of release consistency that is all writes are reflected to other cores on unlock operation. Invalidations from management cores invalidate only entries of the page table. The invalidations of page table entries do not reflect TLB entries immediately. Worker cores evaluate the validity of the cache entry by information in the TLB entry. It is impossible to obtain the latest data unless the immediate TLB entry update.

In order to resolve the problem, we adopt the lazy release consistency that is a more relaxed consistency model than release consistency. The lazy release consistency imposes a constraint that all writes are reflected to other cores on lock operation. A worker core updates own all TLB entries on lock operation. The updates to TLB entries reflects the invalidations to the page table on unlock operation. In this way, the constraint of the lazy release consistency is met.

4. Evaluation of the C5 Software Managed Memory System

In order to evaluate our proposal, we implement a software managed memory system named C5, and evaluate its performance with SimMc that is a software simulator of M-Core architecture.

The simulation parameters for the evaluation are described below. The size of each local memory is 512KB. One of the nodes is treated as main memory. The main memory does not execute any programs. The size of the main memory is infinite and the access latency is 200 cycles. The capacity of software cache in each management core and worker core is 128KB. The software cache is direct-mapped cache. The cache line size is 1024B and the number of the cache entries is 128. The number of TLB entries is 128 and the replacement algorithm is fully

$\star 1$ The number of entries of TLB is equal to the number of L2 cache entries.

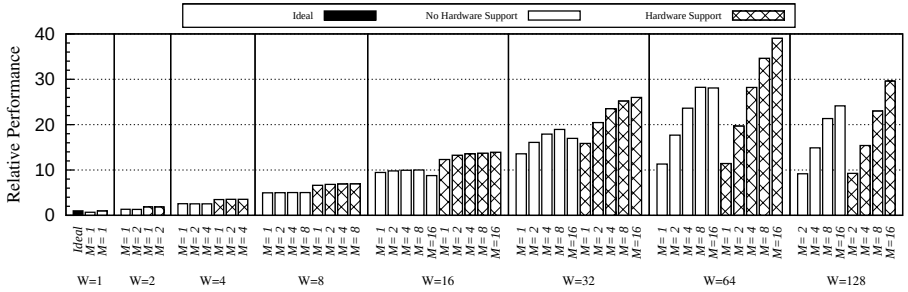

Fig. 5 Relative performance normalized to Ideal in Water. Ideal indicates a result on an ideal environment that has a perfect memory system. M and W represent the number of management cores and worker cores respectively.

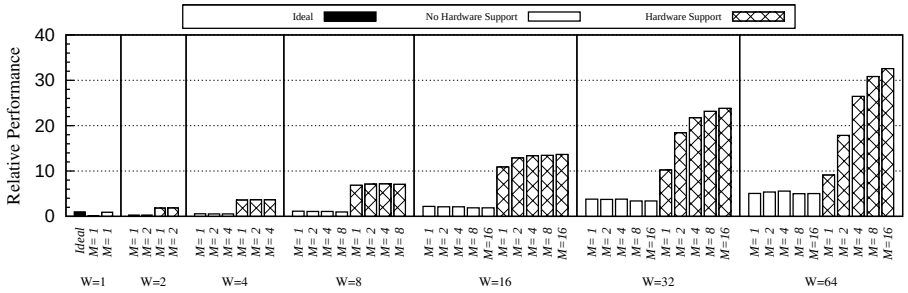

Fig. 6 Relative performance in LU

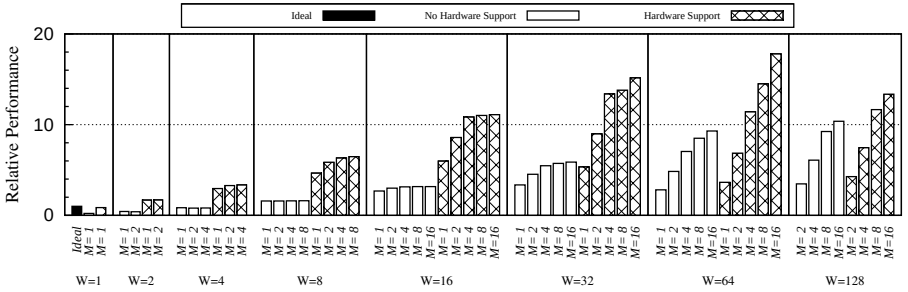

Fig. 7 Relative performance in FFT

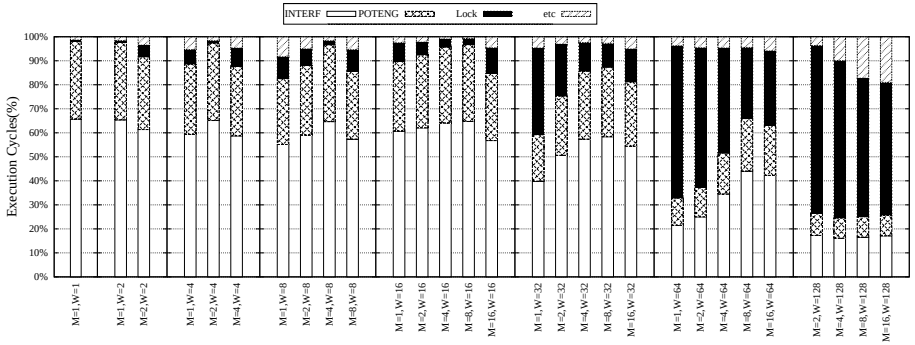

Fig. 8 Ratio of execution time focusing serial execution in Water

associative for the simplicity. Although the large associativity is adopted, it is enough if the associativity of TLB is same to the associativity of software cache.

The modified version[7] of SPLASH-2 Benchmarks Suite[8] is used to evaluate the C5 system. Water-Nsquared in kernels and LU and FFT in applications are used as benchmarks. Problem size of each benchmark is that Water is n=1728, LU is n=1024 and FFT is m=20. Each benchmark is modified to reduce thrashing. In an ideal environment(Ideal) that the local memory size is infinite, results of each benchmark with one core are used as a target for comparison. This environment assumes all memory access is performed in 1 cycle.

Relative performances of Water, LU and FFT to Ideal are shown in **Fig. 5**, 6 and 7 respectively. In the figures, M and W represent the number of management cores and worker cores respectively. Note that results of LU are shown up to 64 worker cores because the problem size is small.

4.1 Water

Performances with/without hardware support are speed-up at a good rate up to 32 worker cores in Fig. 5. The number of management cores highly affects the performances on 64 worker cores. The performances on 128 worker cores are lower than the performances on 64 worker cores.

Fig. 8 shows ratios of each process in normalized execution time with hardware support. INTERF and POTENG are processes that compute force and potential respectively and have enough parallelism. Lock is a process of lock operations in Water. The etc in the figure is processes except these processes. In the figure, the

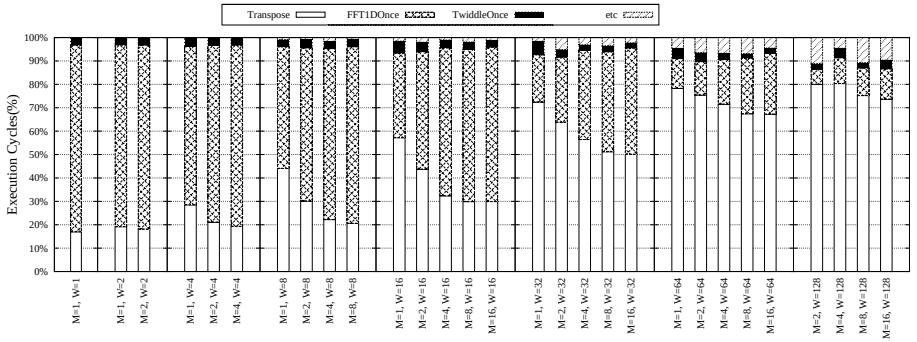

Fig. 9 Ratio of execution time focusing communication in FFT

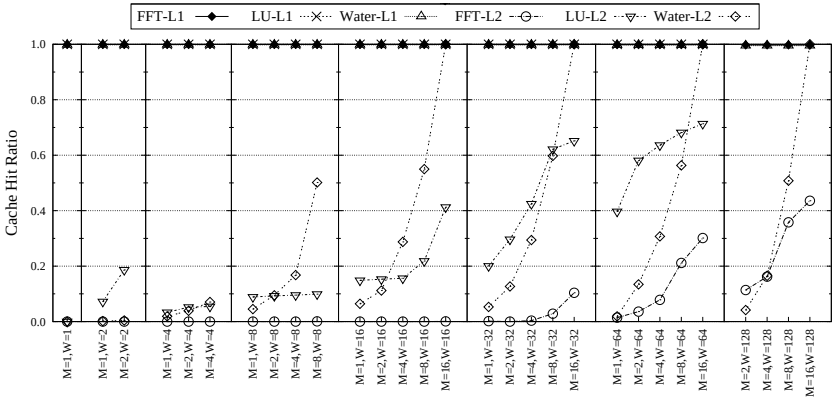

Fig. 10 Software L1 and L2 cache hit ratio with hardware support

more the number of worker cores increases, the more the ratio of Lock increases. This is because the Lock is not performed in parallel. Compared with Fig. 5, performances become down as the ratio of the lock increases. Consequently, Lock process is a major cause of performance bottlenecks in Water.

4.2 LU

Performances with hardware support are speed-up at a good rate up to 64 worker cores in **Fig. 6**. On the other hand, without hardware support, speed-up rate is small because the performance of 1 worker core is much lower than the performance of Ideal.

SPLASH-2 points out that there is load imbalance in LU and the performance is only 36 times speed-up in 64 nodes in a perfect memory system. In C5 system, the performance is 32 times speed-up in 64 worker cores. Therefore the poor parallelism of the algorithm is a major cause of performance bottlenecks in LU.

The performance differences between with/without hardware support result in the differences of the latency of shared memory access. The number of accesses to shared memory covers a large portion of all instructions in LU. Therefore the difference of access latency affects the performance severely.

4.3 FFT

Performances with and without hardware support have the least speed-up of other benchmarks in **Fig. 7**. The performance with hardware support is only 9 times speed-up at a maximum. The performance without hardware support is only 20 times speed-up at a maximum.

Fig. 9 shows ratios of each process in normalized execution time with hardware support. Transpose, FFT1DOnce and TwiddleOnce are the three functions that have longest execution time in FFT. The etc in the figure is processes except these processes. The more the number of worker cores increases, the more ratios of Transpose increase. This is because the Transpose is not performed in parallel.

All worker cores start to copy heavy amount of shared data at the same time in Transpose, so the massive number of cache misses occurs. Management cores are unable to handle the cache misses without keeping the requests waiting. As a result, it is thought that performance enhancement by increasing the number of worker cores is not gained.

4.4 Cache Performance

Fig. 10 shows cache hit ratios of each benchmark with hardware support. L1 cache hit ratios are over 99% in all patterns. On the other hand, L2 cache hit ratios tend to rise as the number of management cores increases. This is because the total size of L2 cache capacity increases by increasing the number of management cores. In LU, L2 hit ratios rise as the number of worker cores increases by prefetching even in the same number of management cores.

5. Conclusions

In this paper, we proposed an efficient software managed memory model for many-core architectures with management cores that manage shared data. The three policies of our system are described below. (1)It minimizes the additional implementation hardware cost. (2)It supports the easy parallel programming. (3)It resolves the bottleneck of main memory access.

We assume two kinds of architectures in our proposal. The first architecture is distributed memory architectures that translate shared memory address to local memory address by software. The second architecture is distributed memory architectures in which each core has TLB for hardware support in order to speed up shared memory access not assuming special hardware.

C5 is a original software managed memory system our proposal model was adapted to. We showed the effectiveness of the C5 through performance measurements with SPLASH-2 benchmark suite and found following three things.

- The use of multiple management cores is the key to achieve good speedup with many worker cores.
- Hardware support by TLB is much better than no hardware support.
- In some benchmarks, the system without hardware support achieved sufficient performance.

As our future works, we evaluate our proposal with more realistic simulation model of main memory and consider additional hardware support with more appropriate balancing between software and hardware.

References

1) Judge, A., Nixon, P., Tangney, B. and Weber, S.: Overview of Distributed Shared Memory, Technical report, University of Bologna (1998).
2) Larsen, M.N., Skovhede, K. and Vinter, B.: Distributed Shared Memory for the Cell Broadband Engine (DSMCBE), *Proceedings of the 2009 Eighth International Symposium on Parallel and Distributed Computing(ISPDC)*, pp.121–124 (2009).
3) Kelm, J.H., Johnson, D.R., Johnson, M.R., Crago, N.C., Tuohy, W., Mahesri, A., Lumetta, S.S., Frank, M.I. and Patel, S.J.: Rigel: An Architecture and Scalable Programming Interface for a 1000-core Accelerator, *Proceedings of the 36th annual international symposium on Computer architecture(ISCA)*, pp.140–151 (2009).
4) Mattson, T.G., Riepen, M., Lehnig, T., Brett, P., Haas, W., Kennedy, P., Howard, J., Vangal, S., Borkar, N., Ruhl, G. and Dighe, S.: The 48-core SCC Processor: the Programmer's View, *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–11 (2010).
5) Uehara, K., Sato, S., Miyoshi, T. and Kise, K.: A Study of an Infrastructure for Research and Development of Many-Core Processors, *International Conference on Parallel and Distributed Computing Applications and Technologies(PDCAT)*, pp. 414–419 (2009).
6) Gharachorloo, K., Lenoski, D., Laudon, J., Gibbons, P., Gupta, A. and Hennessy, J.: Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors, *Proceedings of the 17th annual international symposium on Computer Architecture(ISCA)*, pp.15–26 (1990).
7) : The Modified SPLASH-2 Home Page. http://www.capsl.udel.edu/splash/.
8) Singh, J.P., Gupta, A., Ohara, M., Torrie, E. and Woo, S.C.: The SPLASH-2 Programs: Characterization and Methodological Considerations, *Proceedings of the 22nd International Symposium on Computer Architecture(ISCA)*, pp.24–36 (1995).