DCOS: Cache Embedded Switch Architecture for Distributed Shared Memory Multiprocessor SoCs

Daewook Kim, Manho Kim and Gerald E. Sobelman
Department of Electrical and Computer Engineering
University of Minnesota, Minneapolis, MN 55455 USA
Email: {daewook,mhkim,sobelman}@ece.umn.edu

***Abstract*— Shared memory is a common inter-processor communication paradigm for on-chip multiprocessor SoC (MPSoC) platforms. The latency overhead of switch-based interconnection networks plays a critical role in shared memory MPSoC designs. In this paper, we propose a directory-cache embedded switch architecture with distributed shared cache and distributed shared memory. It is able to reduce the number of home node cache accesses, which results in a reduction in the inter-cache transfer time and the total execution time. Simulation results verify that the proposed methodology can improve performance substantially over a design in which directory caches are not embedded in the switches.**

I. Introduction

Rapid advances of silicon and parallel processing technologies have made it possible to build multiprocessor systems-on-chip (MPSoCs). In particular, packet-switched MPSoCs, [1], which are called networks-on-chip (NoC) [2], are becoming increasingly attractive platforms due to their better scalability, higher data throughput, flexible IP reuse and by solutions to clock skew problems associated with bus-based on-chip interconnection schemes.

Distributed shared memory (DSM) [3] or distributed shared cache (DSC) [4] is an architectural approach which allows multiprocessors to support a single shared address space that is implemented with physically distributed memory. A DSM or DSC multiprocessor platform is also called non-uniform memory access (NUMA) [5] or non-uniform cache architecture (NUCA) [6], since the access time depends on the physical location of a data word in memory or cache. Coherence protocols allow such architectures to use caching in order to take advantage of temporal and spatial locality without changing the programmer's model of memory or cache.

While interconnection networks provide basic mechanisms for communicating, in the case of shared address space processors, additional hardware is required to keep multiple copies of data consistent with each other. Specifically, if there exist multiple copies of the data in different caches or memories, we must ensure that different processors are using the freshest data. Snoopy cache coherence is typically associated with MPSoC systems based on broadcast interconnection networks such as a bus or a ring. In bus-based interconnection systems, all linked processors snoop on the bus for transactions and can maintain cache coherence. However, this snooping protocol [5] is not a scalable solution and it causes serious bus traffic for cache coherence. An obvious solution to this problem is to propagate coherence operations only to those processors that must participate in the operations. This solution requires us to keep track of which processors have copies of various data values and also the relevant state information for them. This state information is stored in a place called the directory, and the cache coherence scheme based on such information is called directory cache coherence. In a distributed shared memory MPSoC that connects all the processors through switches, the directory cache coherence scheme can be applied. In the conventional directory cache protocol, each directory resides in a distributed shared memory bank or distributed L2 cache bank and it contains entries for each memory or cache block. An entry points to the exact locations of every cached copy of a memory block and maintains its status for future reference. The classical full-map directory scheme proposed by Censier [7] uses an invalidation approach and allows for the existence of multiple unmodified cached copies of the same block in the system. However, in such a system, each directory with distributed shared memory or cache is distributed among all nodes in the system to provide a closer local memory or local cache and several remote memories. While local memory access latencies can be tolerated, the remote memory accesses generated during the execution can reduce the performance of applications.

In this paper, we present a method to mitigate the impact of remote memory access latency. We propose a switch architecture for low-latency cache coherency of a distributed shared memory MPSoC platform which we denote as **DCOS**, **D**irectory **C**ache **O**n a **S**witch. The proposed architecture was applied to our proposed MPSoC platform that features packet switched cache coherent NUMA and NUCA in an on-chip system as shown in Figure 1. We have tested and evaluated this architecture using the RSIM [8] distributed shared memory MPSoC simulator. Some core parts of the simulator were modified and new directory cache modules with a crossbar switch were added in order to model our system. Simulations from the SPLASH-2 benchmark suite [9] were performed. The results show a substantial reduction of average read latency and execution time compared to a platform in which directory caches are not embedded into the switches.

II. Switch-based MPSoC with DCOS

Figure 1(a) shows a packet switched MPSoC platform with on-chip shared memory. Figure 1 (b) shows the platform with

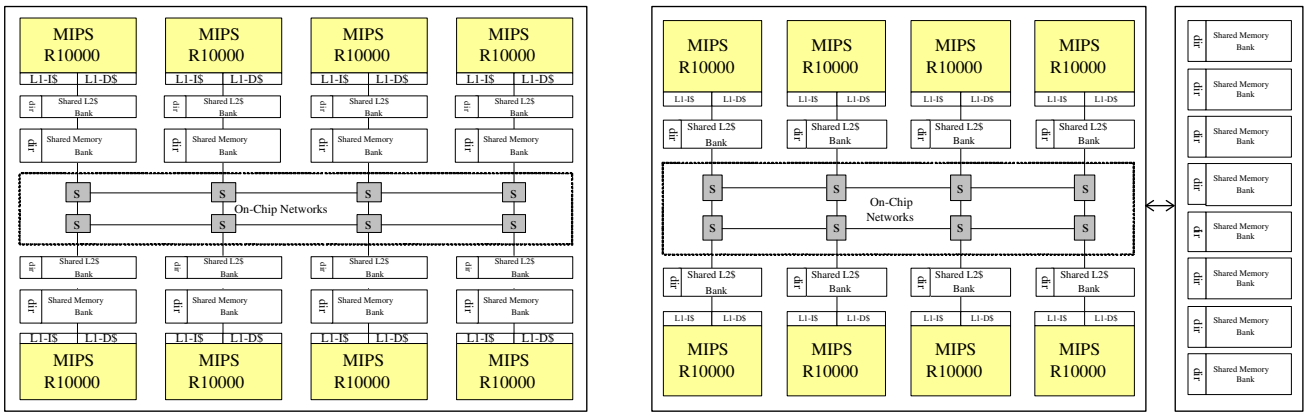

Fig. 1. DCOS architecture based MPSoC platform with (a) on-chip shared memory and (b) off-chip shared memory.

off-chip shared memory. The proposed DCOS architecture was implemented within each switch to reduce the cache-to-cache data transfer time and the switches were connected through a 4×2 2D mesh topology. Wormhole routing was adopted as the packet switching methodology. For our simulations, we used the MIPS R10000 core model which is supported by RSIM as shown in Figure 2. The main features of the MIPS R10000 include superscalar execution, out-of-order scheduling, register renaming, static and dynamic branch prediction, and non-blocking memory load and store operations. The first-level cache can either be a write-through cache with a no-allocate policy on writes, or a write-back cache with a write-allocate policy. The second-level shared cache is a write back cache with write-allocate. The directory cache coherence protocol we adopted in our switch is the modified-shared-invalidate (MSI) protocol [10]. The protocol was also implemented within each distributed shared memory bank and distributed shared L2 cache to compare with the DCOS-based platform to verify performance. The detailed DCOS MSI protocol and switch architecture are shown in Figure 3 and Figure 4.

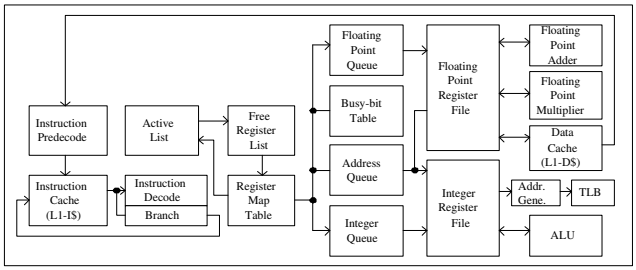

Fig. 2. MIPS R10000 core block diagram.

III. Directory Cache on a Switch for Shared L2$ and Shared Memory

A. DCOS Cache Coherence and Caching Flow

Systems for directory-based cache coherence combine distributed shared memory architectures with scalable cache coherence mechanisms. We exploit a full-map directory cache coherence protocol as our DCOS architecture. In this scheme, the directory resides in main memory and contains entries for each memory block. An entry points to the exact locations of every cached copy of a memory block and maintains its status. With this information, the directory preserves the coherence of data in each distributed shared cache bank by sending directed messages to known locations, avoiding expensive broadcasts. Figure 3 describes a simple example of the data sharing flow for DCOS cache coherence. For example, a presence bit of 1 denotes that cache #1 holds a copy of the memory block. The state entry assigned to a memory block holds the current state of the block: empty, shared, or modified/invalid.

Figure 3(a) presents the initial status of the DCOS directory cache, which shows no data items are copied into caches or memories. The directory entries are empty and therefore the state tag shows empty, marked as 'E'. Figure 3(b) presents the case where a data is shared with other caches and memories. The bold arrow pointing from the shared memory bank to the L1 cache of core 2 indicates the case where shared data is not available in the shared L2 cache due to a write-miss. Figure 3(c) shows that a write request of one processor leads to the invalidation of all other copies of the processor caches. The new data value is stored in the caches or memories and the entry state is changed to the modified state, marked as 'M'. The bold arrow pointing from the shared memory bank to the L1 cache of core 2 in Figure 3 also shows the direct data invalidation case when the shared L2 cache doesn't have the data due to a write-miss.

B. Directory Cache Embedded Switch Architecture

Figure 4 shows the overall DCOS architecture block diagram including our proposed directory caches. All the directory caches for both the shared memory bank and the shared L2 cache bank are embedded within the crossbar switch. The cache dir update and memory dir update inputs to the directory cache module indicate the signals required for the DCOS to update whenever the data states of the attached node

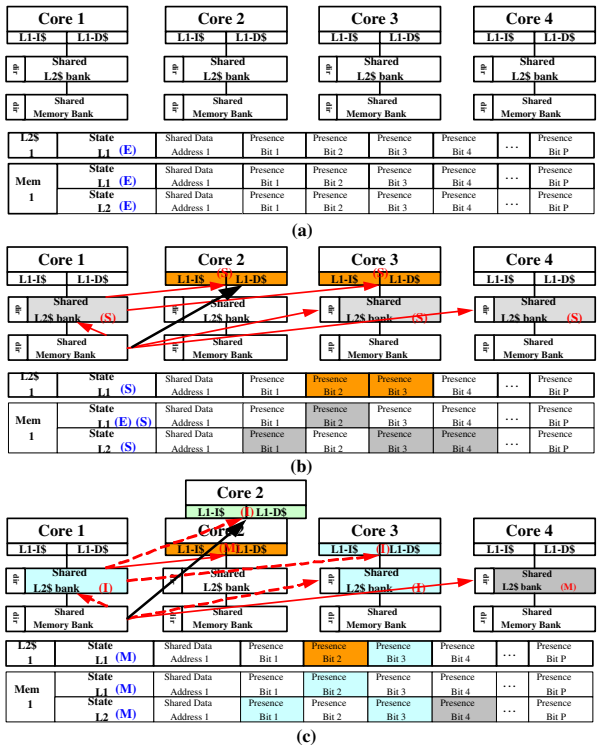

Fig. 3. Full-map directory cache coherent MSI protocol for DCOS. (a) Empty, (b) shared and (c) modified/invalid.

cache and node memory are changed. This information is sent to the arbiter through the directory controller. This leads to efficient routing up to the destination node. It reduces cache-to-cache transfer time and results in an overall performance improvement in terms of packet latency and execution time.

IV. Simulation Environment

We have used the RSIM simulator for distributed shared memory multiprocessor systems. Some core parts of the simulator written in C++ were modified for a shared L2 cache environment and the proposed directory cache module was added to the default switch block. Table I summarizes the parameters of the simulated platform.

The application programs used in our evaluations are FFT, Radix, Ocean, and Barnes from the SPLASH-2 benchmark suite. The input data sizes are shown in Table II.

V. Simulation Results and Analysis

In this section, we present and analyze the performance results obtained through extensive simulations and evaluate the impact of the DCOS architecture. The main objective of switch directory caches is to reduce the number of cache-to-cache transfers and the total execution time. Figure 5 shows that both cache-to-cache transfer time and execution time of DCOS based schemes were substantially reduced over the non-DCOS scheme in terms of total consumed clock cycles. The parameter we varied was the size of the shared memory directory cache on a switch, which was varied from 512 to 2048 entries, while

TABLE I

Simulation Parameters

Processor	
Architecture model	MIPS R10000
Speed	1 GHz
Max. fetch/retire rate	4
Instruction Window	64
Functional units	2 integer arithmetic, 2 floating point
Memory queue size	32 entries
L1 Cache	
Line size	16 bytes
Write through	Directed mapped, 32 KB
Request ports	2
Hit time	2 cycles
Shared L2 Cache Bank	
Line size	64 bytes
Write through	Directed mapped, 128 KB
Hit time	15 cycles, pipelined
Shared Memory	
Access time	70 cycles (70ns)
Interleaving	4-way
Directory Cache in Switch	
Shared L2$ directory bank size	32 entries
Shared memory directory size	512 / 1024 / 2048 entries
On -Chip Switched Networks	
Topology	2D Mesh (4x2)
Flit size	8 bytes
Switch speed	250 MHz
Channel speed	500 MHz
Channel width	32 bits

TABLE II

Benchmark Applications and Input Sizes

Programs	Input Sizes
FFT	32 k
Radix	1M Keys, 1024 Radix
Ocean	100 × 100 Grid
Barnes-Hut	2048 Bodies

fixing the cache size of the shared L2 directory at 32 entries. As shown in Figure 5(a), the total execution time for each benchmark application was reduced proportionally as the size of the on-switch directory cache is increased. When comparing the execution time of the non-DCOS to the DCOS for 2048 entries, the overheads for FFT, Radix, Ocean and Barnes were reduced by 43.1%, 28.7%, 21.4%, and 27.9%, respectively. In addition, as shown in Figure 5(b), the total cache-to-cache transfer time overhead from home node to local node was analyzed to determine the impact of the DCOS scheme. Cache-to-cache transfer time also proportionally decreased as we increase the size of the shared memory directory cache on a switch. When compared, the cache-to-cache transfer time overhead of non-DCOS to DCOS for 2048 entries on FFT, Radix, Ocean and Barnes were reduced by 35.8%, 63.2%, 30.8%, and 43.2%, respectively. Based on these two performance metrics on the four benchmark programs, a substantial performance improvement is obtained with the proposed DCOS scheme.

VI. Conclusions

We have presented a novel directory-cache embedded switch architecture with distributed shared cache and distributed shared memory. This scheme is able to reduce the number of

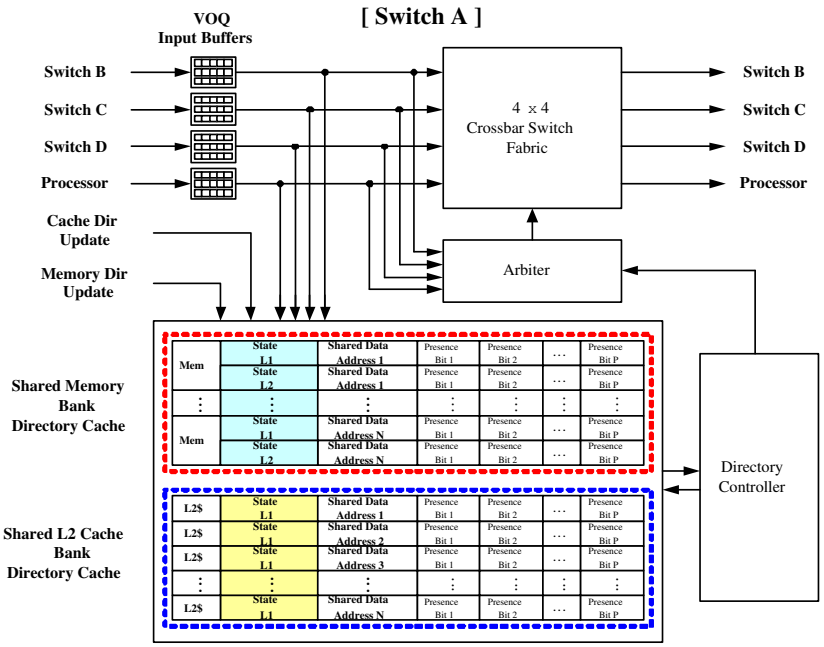

Fig. 4. DCOS architecture block diagram.

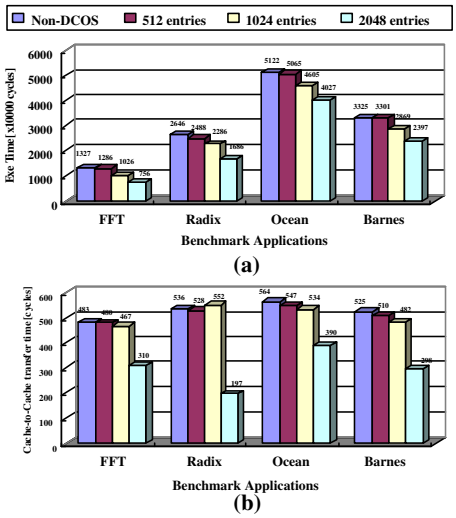

Fig. 5. Simulation results. (a) Execution time [$\times 10^4$cycles]. (b) Cache-to-cache transfer time [cycles].

home node cache accesses, which results in the reduction in inter-cache transfer time and total execution time. Simulation results verify that the proposed methodology can improve performance substantially over a platform in which directory caches are not embedded in the switches.

ACKNOWLEDGMENTS

We thank Sangwoo Rhim, Bumhak Lee and Euiseok Kim of the SAMSUNG Advanced Institute of Technology (SAIT) for their help with this manuscript. This research work is supported by a grant from SAIT.

REFERENCES

[1] M. D. Nava, P. Blouet, P. Teninge, M. Coppola, T. Ben-Ismail, S. Picchiottino, and R. Wilson, "An open platform for developing multiprocessor socs," *Computer*, vol. 38, no. 7, pp. 60–67, 2005.

[2] L. Benine and G. D. Micheli, "Networks on chip: a new paradigm for systems on chip design," in *Proc. of Design, Automation and Test in Europe Conf.*, jan 2002, pp. 418–419.

[3] J. Hennessy and D. Patterson, "Computer architecture: A quantitative approach, chapter six," Aug 2003.

[4] J. Ahn, K. Lee, and H. Kim, "Architectural issues in adopting distributed shared memory for distributed object management systems," in *Proceedings of the Fifth IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems*, Aug 1995, pp. 294–300.

[5] J. Hennessy, M. Heinrich, and A. Gupta, "Cache-coherent distributed shared memory: perspectives on its development and future challenges," in *Proceedings of the IEEE*, vol. 87, Mar 1999, pp. 418–429.

[6] C. Kim, D. Burger, and S. W. Keckler, "An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches," in *ASPLOS-X: Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*. ACM Press, 2002, pp. 211–222.

[7] L. Censier and P. Feautrier, "A new solution to coherence problem in multicache systems," *Computer*, vol. 27, no. 12, pp. 1112–1118, 1978.

[8] C. Hughes, V. Pai, P. Ranganathan, and S. Adve, "Rsim: simulating shared-memory multiprocessors with ilp processors," *Computer*, vol. 35, pp. 40–49, 2002.

[9] S. Woo, M. Ohara, J. Torrie, E.; Singh, and A. Gupta, "The splash-2 programs: characterization and methodological considerations," in *Proceedings. 22nd Annual International Symposium on Computer Architecture*, 1995, pp. 24–36.

[10] D. Culler and J. Singh, "Parallel computer architecture: A hardware/software approach, chapter five."