Using Switch Directories to Speed Up Cache-to-Cache Transfers in CC-NUMA Multiprocessors *

Ravi Iyer *
Server Architecture Lab
Intel Corporation
Beaverton, OR.
ravishankar.iyer@intel.com

Laxmi N. Bhuyan
Dept of Computer Science
Texas A&M University
College Station, TX.
bhuyan@cs.tamu.edu

Ashwini Nanda
Scalable Server Group
IBM TJ Watson Research
Yorktown Heights, NY.
ashwini@watson.ibm.com

Abstract

In this paper, we propose a novel hardware caching technique, called switch directory*, to reduce the communication latency in CC-NUMA multiprocessors. The main idea is to implement small fast directory caches in crossbar switches of the interconnect medium to capture and store ownership information as the data flows from the memory module to the requesting processor. Using the stored information, the switch directory re-routes subsequent requests to dirty blocks directly to the owner cache, thus reducing the latency for home node processing such as slow DRAM directory access and coherence controller occupancies. The design and implementation details of a* ***D****i****R****ectory* ***E****mbedded* ***S****witch* ***AR****chitecture, DRESAR, are presented. We explore the performance benefits of switch directories by modeling DRESAR in a detailed execution driven simulator. Our results show that the switch directories can improve performance by up to 60% reduction in home node cache-to-cache transfers for several scientific applications and commercial workloads.*

1 Introduction

Cache coherent non-uniform memory access (CC-NUMA) multiprocessors provide a scalable design for shared memory but they continue to suffer from long remote memory access latencies due to comparatively slow memory technology and network transfer latencies. While the memory write latency can be hidden through the use of write-buffers and relaxed memory consistency models, application performance still depends heavily on the memory read latency. Read requests to memory can be generally classified into two types: accesses to clean blocks obtained from memory and accesses to dirty blocks obtained from another processor's cache. The first problem dealing with clean memory blocks has been addressed using data prefetching techniques and some recent papers [5, 7]. However, the second problem dealing with dirty blocks has not yet been addressed successfully. Read requests to dirty blocks get serviced via direct cache-to-cache transfers supported in most modern CC-NUMA systems. The service time for these dirty cache-to-cache transfers on current systems is roughly 1.5 to 2 times longer than a clean memory access. For example, the access latency for dirty blocks is 1.6 times that for clean blocks on the SGI Origin 2000 [3]. This increase in access latency is due to the slow DRAM directory lookup, several message transfers over the interconnect and coherence controller occupancies. This affects the performance of commercial workloads such as TPC-C [6, 12] as well as several scientific applications [8, 4].

In this paper, we propose a directory caching technique to reduce the cache-to-cache transfer latency in CC-NUMA multiprocessors running scientific and commercial workloads. By embedding a small fast SRAM directory cache within each switch, called *switch directory*, in the network, we capture ownership information as data flows through the interconnect and provide it to future requests from processors that re-use this data. Additionally, these requests are also directly forwarded to the owner cache, avoiding the subsequent network hops towards the home node, directory controller occupancies and slow DRAM directory access. In this paper, we first present the communication behavior of several scientific applications and commercial workloads to identify the potential for switch directories. When considering design issues for switch directories, we find that current crossbar switches have large amounts of buffer space to efficiently pipeline messages. However, we observe that increasing the buffer size beyond a certain value does not have much impact on the application performance of shared memory multiprocessors. Thus the large amount of buffer space in current switches, such as SPIDER [2], can be better utilized by organizing them as a switch cache or switch directory.

Related work in this area includes the design of directory caches [8], the MIND scheme [9], the switch cache framework [5] and the GLOW extensions to SCI [7]. Michael et al. [8] presented the design issues for SRAM directory caches and showed significant performance benefits for scientific applications running in a CC-NUMA environment. Our proposed switch directory may be visualized as a generalization of this scheme by providing multiple levels of SRAM directory caching. However, our scheme particularly targets cache-to-cache transfers between clusters of processors. Finally, switch directories re-route processor requests directly to the owner cache and avoid not only the DRAM directory latency, but also subsequent network hops, bus transfer delays and controller occupancies at the home node. The MIND scheme [9] embeds full-map directories within the switches of the MIN but suffers from restrictions imposed by the inclusion property requiring larger directories towards the root of the hierarchy. Kaxiras [7] presents extensions to the SCI protocol, GLOW, to provide scalable reads and writes to widely shared data. In [5], we presented a switch cache framework to serve recently ac-

*This research has been supported by NSF grants MIP-9622740, CCR-9810205, and an IBM Partnership Award.

*This work was done while Ravi Iyer was at Texas A&M University.

cessed widely shared data within a small cache in the crossbar switch. In this paper, we target a different class of scientific and commercial applications called communication intensive workloads. Our proposed switch directory framework is aimed at reducing the latency of cache-to-cache transfers in CC-NUMA multiprocessors.

There are several issues to be considered while designing a multi-level directory caching technique. These include directory design issues such as technology & organization, cache coherence issues, switch design issues such as arbitration, and message flow control issues such as appropriate routing strategy, message layout etc. The contribution of this research is the detailed design and performance evaluation of a switch directory interconnect employing a **D**i**R**ectory **E**mbedded **S**witch **AR**chitecture (DRESAR) based on multiported set associative SRAM directory caches. The performance evaluation h of the switch directory framework is conducted using five scientific applications running on a detailed execution driven simulator (RSIM). Commercial workload traces for TPC-C and TPC-D obtained using COMPASS [10] at the IBM T.J. Watson Research Center is also used in a trace-driven simulator to evaluate the performance impact of switch directories.

The rest of this paper is organized as follows. Section 2 motivates our research based on application characteristics. Section 3 presents the switch directory framework. The design and implementation details of a crossbar switch directory architecture are presented in Section 4. Its performance benefits are analyzed in Section 5 using extensive simulation experiments. Finally, Section 6 concludes this paper.

2 Background and Motivation

Communication in a shared memory system occurs through reads and writes to memory. While the latencies for writes can be hidden by using write buffers and weaker consistency models, application stall time due to memory reads still remains the major concern. In this section, we measure read requests generated in five different scientific applications and two commercial workloads and classify them based on the method by which they were serviced. The chosen scientific applications are the Fast Fourier Transform (FFT) [13], Transitive Closure (TC), Successive-Over-Relaxation of a grid (SOR), Floyd Warshall's all-pair-shortest-path Algorithm (FWA) and Gaussian Elimination (GAUSS). The two commercial workloads used are the online transaction processing benchmark (TPC-C) [14] and the decision support benchmark (TPC-D) [15]. The measurements described below were collected through extensive simulations of a 16-processor CC-NUMA system. Detailed description of our simulation methodology is presented later in Section 5.1.

Memory reads can be divided into two types depending on how they are serviced. Based on the directory state of the block, a read request can either find data clean in memory or dirty in another processor's cache. In Figure 1, the percentage of clean memory accesses and dirty cache-to-cache transfers are shown for different applications. We find that two of the five scientific applications, FFT and SOR, require 60% to 70% of the reads to be serviced as cache-to-cache transfers. The remaining three scientific applications require 15% to 30% cache-to-cache transfer reads. For the commercial workloads, we find the percentage of cache-

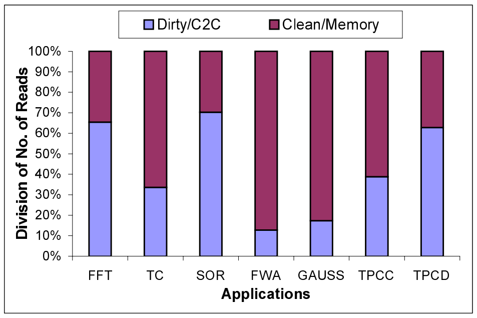


Figure 1. Fraction of Clean vs. Dirty Memory Reads

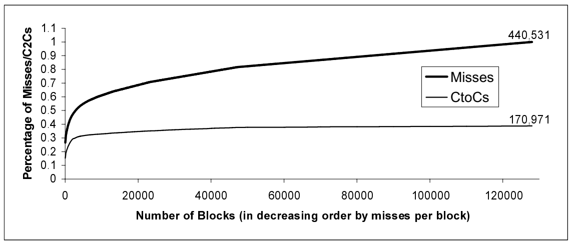


Figure 2. Access Frequency of TPC-C Blocks

to-cache transfers is 38% and 62% respectively for TPC-C and TPC-D. It should be noted here that cache-to-cache transfers are 1.5 to 2 times more costly than clean memory accesses because of the slow DRAM directory access and the multiple messages transferred over the interconnect. As a result, we find that while the percentage of cache-to-cache transfers in the FFT application was approximately 65% (from Figure 1), the average read latency component was almost 74%. Similarly, for TPC-C, this percentage increases from 38% for the number of cache-to-cache transfer reads to 49% for the average read latency component due to cache-to-cache transfers.

The percentage of cache-to-cache transfer misses shown in Figure 1 takes into account the read misses to all blocks touched during the execution. However, it may be the case that most of the cache-to-cache transfers occur only during reads to a selected set of the blocks. Figure 2 shows the distribution of read misses and cache-to-cache transfers over the total number of blocks touched during the execution of TPC-C. On the x-axis, the blocks are listed in decreasing order of read misses per block, i.e. the block that accounts for the highest number of misses is listed on the far left, while the block with the least number of misses is listed at the far right. On the y-axis, the percentage of overall read misses and those that required cache-to-cache transfers for all memory blocks is plotted in a cumulative fashion. Overall, there were approximately 440,000 read misses to roughly 130,000 blocks as can be seen at the extreme right in the figure. The total number of cache-to-cache transfers were measured to be approximately 170,000. We also observe that there were a small set of blocks with a high number of misses per block. Only 10% of the blocks account for roughly 88% of the cache-to-cache transfers. In summary, *we find that such workloads access a few communication intensive blocks frequently, accounting for a significant number of read misses that require cache-to-cache transfers*. Our aim is to develop a latency reduction technique that efficiently serves such read misses.

3 The Switch Directory Framework

Communication intensive blocks that are frequently accessed may require three network transfers to obtain the data. In the first network transfer, the requester sends a request to the home node for directory lookup. After performing the directory lookup, the state of the cache block is determined. If the block is clean, data is read from the memory and a reply is sent to the requester. However, if the data is in dirty state, the second network transfer occurs as the request is then forwarded to the owner of the memory block. The owner's cache is read and the third message is the cache-to-cache transfer of data from the owner to the requester. Finally, another message is also sent back to the home node to update the block in the main memory. As a result, the interconnection network is the only global location in the CC-NUMA system that sees requests, replies and coherence messages that travel from node to node.

By storing ownership information for recently written blocks within the switches of the interconnect, we can possibly re-route subsequent processor requests directly to the owner cache and avoid the need for the slow DRAM directory lookup at the home node. Storing ownership information in the interconnect switches comes with the requirement that this information is kept coherent and consistent with the full-map directory and the base cache coherence protocol. The ideal topology for storing global coherent directory information in a system is the global bus. However, the global bus is not a scalable medium. Consequently, multiple bus hierarchies and fat-tree structures have been considered effective solutions to the scalability problem. The tree structure provides the next best alternative to hierarchical directory caching schemes.

3.1 The Switch Directory Interconnect

A multistage switching network for a 16-node CC-NUMA system is shown in Figure 3. In a shared memory system, communication between nodes is accomplished via read/write transactions and coherence requests/acknowledgments. The read/write requests and coherence replies from the processor to the memory use forward links to traverse through the switches. Similarly, read/write replies with data and coherence requests from memory to the processor traverse the backward path. Separating the paths enables separate resources and reduces the possibility of deadlocks in the network. At the same time, routing them through the same switches provides identical paths for requests and replies for a processor-memory pair that is essential to develop a switch directory hierarchy.

The basic idea of our directory caching strategy is to utilize the tree structure in the BMIN and the path overlap of requests, replies, and coherence messages to provide coherent directory information in the interconnect. By incorporating small fast directories in the switching elements of the BMIN, we can capture ownership information for frequently modified blocks and re-route subsequent requests directly to the owner cache. We use the term *switch directory* to differentiate these directory caches from the DRAM directory at the home node. An example of a BMIN employing switch directories is shown in Figure 3. An initial write request from processor P_i is served at the remote memory M_j. When the data/ownership reply flows through the interconnection network, the owner's pid is captured

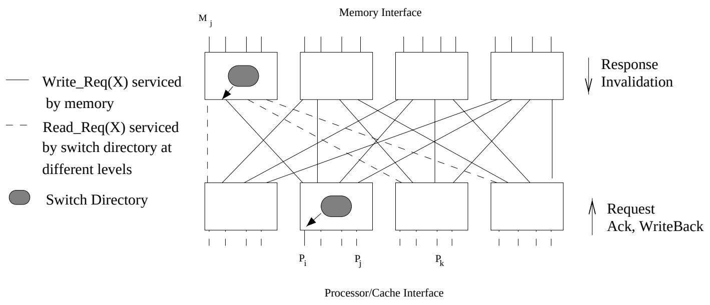
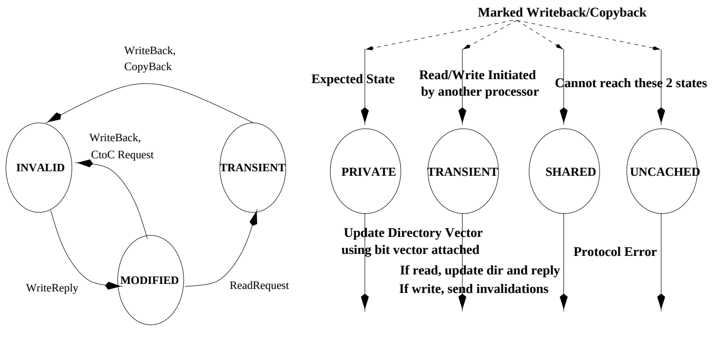

M j
Memory Interface
Write_Req(X) serviced by memory
Read_Req(X) serviced by switch directory at different levels
Switch Directory
Response Invalidation
Request Ack, WriteBack
P i
P j
P k
Processor/Cache Interface

Figure 3. Switch Directories in the Interconnect


WriteBack, CopyBack
WriteBack, CtoC Request
INVALID
TRANSIENT
MODIFIED
WriteReply
ReadRequest
Marked Writeback/Copyback
Expected State
Read/Write Initiated by another processor
Cannot reach these 2 states
PRIVATE
TRANSIENT
SHARED
UNCACHED
Update Directory Vector using bit vector attached
If read, update dir and reply
If write, send invalidations
Protocol Error

(a) Switch Directory State Diagram (b) Change in Directory Protocol

Figure 4. Switch Directory Protocol Execution

and stored in the directory cache at each switching element along the path. When a subsequent request to the same block arrives from a different processor, such as from P_j or P_k, it can be re-routed to the owner P_i and avoid any subsequent network hops to the home node and the slow DRAM directory lookup at the home node. The benefits of such a scheme is three-fold. First of all, it extends the directory functionality into the interconnect bringing it closer to consumer processors. Secondly, it reduces the cache-to-cache transfer latency by providing SRAM directory access and re-routing requests directly to the home node. Finally, it reduces the number of critical requests at the directory controller and potentially decreases the controller occupancy for other requests.

3.2 The Switch Directory Protocol

Many hardware cache-coherent systems employ a full-map directory scheme. In this scheme, each node (home node) maintains a bit vector to keep track of all the sharers of each block in its local memory. On every write, an ownership request is sent to the home node, invalidations are sent to all the sharers of the block and the ownership is granted only when all corresponding acknowledgments are received. At the processing node, each cache employs a three-state (MSI) protocol to keep track of the state of cache lines. Incorporating switch directories comes with the requirement that processor caches continue to remain coherent, and data access remains consistent with the system consistency model. Below, we describe the switch directory caching protocol in detail.

Our basic directory caching scheme can be represented by the state diagram in Figure 4 and explained as follows. *The switch directory only stores ownership information for*

Msg Type	Description
ReadRequest	Loads resulting in misses to remote memory.
WriteRequest	Stores resulting in misses to remote memory
WriteReply	Ownership reply for servicing write requests.
CtoC Request	Request forwarded to the cache when block is found to be in private state
CopyBack	Data sent to the home node to keep the memory consistent after a ctoc transfer.
WriteBack	Data sent from cache to memory when a dirty cache block is replaced.
Retry	Reply sent to initiate a retry for the request.

Table 1. Messages Relevant to the Switch Directory

blocks that are modified, i.e. when a block is read to the processor cache in a dirty state, the owner's pid is cached in the switching elements. The switch directory protocol employs 3 directory states, namely MODIFIED, TRANSIENT and INVALID. The transitions of blocks from one state to another is shown in Figure 4a. The terminology of relevant messages flowing through the network is described in Table 1. The actions taken upon the arrival of each message at the switch directory can be described as follows. The few required changes to the cache and directory controller are also described whenever appropriate.

Write Replies: When write replies flow from the memory to the requester, they enter a new entry of ownership information for the block in each of the switch directories along its backward path.

Read Requests: Read requests enter the interconnect and check the switch directories along their path. No action is taken if the block is not present in the directory and the request is forwarded to the home node. A directory hit occurs when the block is found in MODIFIED state in the switch directory. In such an event, the switch directory consumes the read request by sinking the message. The switch directory then forwards a cache-to-cache transfer request for the requester to the owner's cache along the backward path. The state of the block is changed to TRANSIENT. When a read request finds the block in TRANSIENT state, there are two alternatives. One alternative is to store the requester's pid as a bit vector within each directory entry. The requester will then be served when a copy-back or write-back is sent from the owner's cache to the home node. Another alternative is to simply retry the request, thereby reducing the complexity of the switch directory. We employ this second alternative in our switch directory design also because we find that communication intensive blocks have very few sharers. However, note that in both the cases, the read request message should be sunk at the switch and not allowed to proceed to the home node.

Write Requests or Cache-to-Cache Transfer Requests: Write requests follow the forward path and check the switch directories along the way. Cache-to-cache transfer requests follow the backward path and check the switch directories along the path. If the block is present in MODIFIED state, then the block is invalidated and changed to INVALID state. The requests are allowed to progress to the next switch along their path. If the block is found to be in TRANSIENT state, then it means that a read transaction has been initiated from this switch directory. In such an event, a negative acknowledgement is sent to the requester and the write request is retried, while the cache-to-cache transfer request is sunk at the switch directory.

Write-Backs and Copy-Back Messages: When write-back and copy-back messages arrive at the switch directories following the forward path, they invalidate the block, if present. If the incoming message was a copy-back message, it may find the block in TRANSIENT state if the switch directory initiated a cache-to-cache transfer. In such a case, it carries the requester's pid to the home node to update the full-map directory and no further action is required at the switch directory. A write-back message will find the block in TRANSIENT state if a cache replacement forces the dirty line to be evicted before the cache-to-cache transfer request initiated by the switch directory reaches the owner's cache. In such an event, a reply is generated to the requester identified in the directory entry and using the data held in the write-back message. Finally, it is necessary that we keep the full-map directory updated with all the sharers. In order to do so, the message carries the identity of the requester to the home node. When the copy-back or write-back message reaches the home node, the directory vector is updated with the sharer pids. At this time, the directory controller can serve any requests held in the pending queue for the block. This functionality requires a minor modification in the directory controller for handling marked writeback and copyback requests.

Retries: Retries may arrive at the switch directory and find the block in TRANSIENT or INVALID state when queues at the cache controller are full or when other critical resources are unavailable. If the block is found in TRANSIENT state, then the retry message is re-routed to all requesters.

Finally, it should be noted that all messages generated by the switch directory are marked using a single bit in the header flit. This helps cache controllers and directory controllers to differentiate these messages from the regular messages.

4 Switch Directory Architecture and Design

Crossbar switches provide an excellent building block for scalable high performance interconnects. They mainly differ in two design issues: *switching technique* and *buffer management*. We use wormhole routing as the switching technique and input buffering with virtual channels since these are prevalent in commercial crossbar switches [1, 2].

4.1 Crossbar Switch Organization

Our base bi-directional crossbar switch has four inputs and four outputs. Each input link in the crossbar switch has two virtual channels thus providing 8 possible input candidates for arbitration. The arbitration process is the *age* technique, similar to that employed in the SGI Spider Switch [2]. At each arbitration cycle, a maximum of 4 highest age flits are selected from 8 possible arbitration candidates. The flit size is chosen to be 8 bytes ($4w$) with 16 bit links similar to the Intel Cavallino [1]. Thus it takes four link cycles to transmit a flit from output of one switch to the input of the next. Buffering in the crossbar switch is provided at the *input block* at each link. The input block is organized as a fixed size FIFO buffer for each virtual channel that stores flits belonging to a single message at a time. The virtual channels are also partitioned based on the destination node. This avoids out-of-order arrival of messages originating from the same source to the same destination. We also

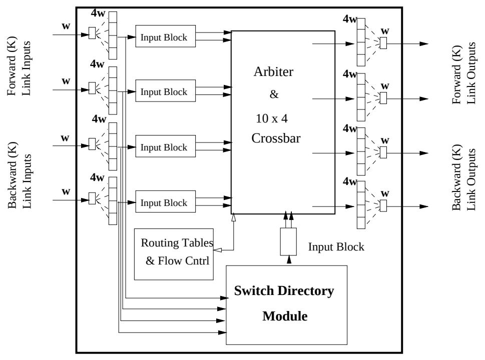


Figure 5. Switch Design and Organization

provide a bypass path for the incoming flits that can be directly transmitted to the output if the input buffer is empty.

While organizing the switch directory, we are particularly interested in maximizing performance by serving flits within the cycles required for the operation of the base crossbar switch. Thus the organization depends highly on the delay experienced for link transmission and crossbar switch processing. Our switch directory organization is based on link and switch processing delays similar to those experienced in Intel Cavallino [1]. The internal switch core as well as the link transmission runs at 200 MHz. The switch takes four cycles to move flits from the input to the link transmitter at the output. The link also transmits a 8-byte flit in four 200 MHz cycles. Figure 5 shows the arbitration-independent organization of the switch directory. The organization is arbitration independent because the switch directory performs the critical operations in parallel with the arbitration operation. At the beginning of each arbitration cycle, a maximum of four input flits stored in the input link registers are transmitted to the switch directory module. In order to maintain flow control, all critical switch directory processing should be performed in parallel with the arbitration and transmission delay of the switch (4 cycles). In other words, four incoming requests need switch directory processing within four cycles. Messages generated by the switch directory are forwarded to the crossbar via the additional input block. In effect, the size of the crossbar increases from 8×4 to 10×4.

4.2 DiRectory Embedded Switch ARchitecture

In this section, we present a detailed hardware design of a crossbar switch directory called DRESAR (**Di****R**ectory **E**mbedded **S**witch **AR**chitecture). For a 4×4 crossbar switch, the objective is to serve four incoming messages within the four cycles of conventional crossbar switch processing. The implementation of DRESAR is shown in Figure 6. Its operation can be divided into *(1) process incoming flits, (2) switch directory access (3) FSM/protocol handling (4) CtoC & reply generation and (5) switch directory feedback.*

Process Incoming Flits: Incoming flits are processed based on their message type. A maximum of 3 bits are required in the header flit to encode the seven different types of messages as shown in Table 1. All other request types can be ignored since they do not require switch directory processing. Switch directory requests all require access to SRAM directory cache for snooping on the state of the block and ownership information, if in PRIVATE state. In a single cycle, the message headers are decoded and the request/address information is passed to the snoop registers.

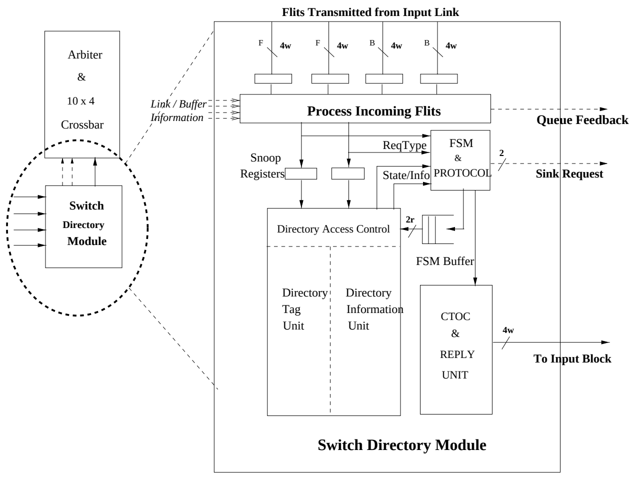


Figure 6. Implementation of DRESAR

Switch Directory Access: Since a maximum of four requests need to be processed within the remaining three cycles, we employ a 2-way multiported SRAM directory and two snoop registers for parallel access. Thus, two requests are served in a single cycle. As discussed in Section 3, the directory array stores one of three states for each block, namely MODIFIED, TRANSIENT and INVALID. In addition, the directory arrays also need to store ownership information, the first requester's pid and a bit vector for marking subsequent sharers. Assuming a 16-processor system, the size of each directory entry is a maximum of 10 bits: 4 bits for ownership information, 4 bits for a requester's pid and 2 bits for the directory state. Since the directory will store only a few entries (roughly 1024 to 4096), a 2-way multiported directory can be easily implemented by duplicating the directories. At the end of each access cycle, the state and directory information for two requests are read from the directory and forwarded to the FSM/protocol handler for further processing. Thus the four requests can be served within two cycles. Thus, out of four cycles of switch processing, the directory ports are busy in only two cycles for snooping. In the remaining two cycles, information updates and state changes for the accessed blocks are performed by the FSM/protocol handler to the directory entries.

FSM/Protocol Handling: In a single cycle, this component of the switch directory receives two requests along with their state and ownership/requester information from the directory. This component uses sequential logic to implement the state transition diagram shown in Figure 4. In addition, it also provides two *sink request* signals to the output transmitter of the crossbar switch. These signals are enabled if the messages do not need to proceed to the destination, as explained in Section 3. For example, if a read request finds the block in MODIFIED state, then the request need not be forwarded towards the memory module and will be handled by the switch directory. In such a case, the FSM component updates the state of the directory as required, generates a sink request signal to the appropriate output transmitter and signals the CtoC & Reply unit to forward a cache-to-cache transfer request to the owner node. Another important request that needs explanation is the write-back/copy-back request. If these requests find the block in TRANSIENT state, a reply is generated to the requester and its pid is attached to the message that continues towards the memory module. This information is stored in the directory arrays and is forwarded to the output transmitter in order to enable the appropriate bits in the appropriate header flit. Finally, note that all state changes to the directory entry are made during the two idle cycles (of four cycles) when the directory ports are free. Until these are processed, they are stored in the FSM buffer shown in Figure 6.

CtoC & Reply Unit: After the required protocol handling, three types of messages may be generated by the switch directory, namely cache-to-cache transfer requests, read replies and retry requests. A cache-to-cache transfer request is generated when a read request arrives at the switch directory and finds the block in MODIFIED state. For such a message generation, only the header flit of the incoming message is required. A read reply is generated when a write-back or a copy-back message arrives at the switch directory and finds the block in TRANSIENT state. For such a message, this unit requires the data that follows the write-back header. Finally, a retry request is generated when a write request arrives at the switch directory and finds the block in TRANSIENT state. For this message generation, only the header flit is required. The generated message is assembled, prepared and then passed to the input block containing two virtual channels. Since all these requests are directed towards the processor interface, they get routed through the backward path in the interconnect. Note also that messages generated at the switch directory are marked using a 1-bit flag in the header flits. This flag can be used by directory and cache controllers to identify switch directory messages.

Switch Directory Feedback: A possible issue with switch directory processing is when the queues within the directory module gets full. These include the virtual channel queues in the input block and the FSM buffer. In such cases, feedback information in the form of queue sizes is provided to the crossbar arbiter to block incoming requests until a space becomes available. The modification to the arbiter for implementing this blocking is quite straightforward. We believe that such cases are very unlikely but can be easily avoided by blocking only messages that are critical to maintain coherence and allowing all other requests to bypass the switch directory and be serviced at the home node.

4.3 Designing a 8×8 Crossbar Switch Directory

In the previous subsection, we presented the design of a 4×4 crossbar switch directory. In this section, we discuss the implementation issues for an 8×8 switch directory with the same basic crossbar operation and delay parameters. As a result, our goal now is to process 8 (instead of 4) messages within 2 cycles. Consequently, doubling the crossbar switch size basically requires doubling the number of ports on the directory. In order to alleviate this requirement, we propose the addition of a small pending buffer that holds information for blocks in transient state. Requests processed via the switch are maintained in transient state at the directory for a brief period of time. Writeback, Copyback, CtoC and Retry messages only need to check the switch directory to find out if the block is in transient state. By storing the information for the few transient blocks (8-16 entries) in the pending buffer, switch directory processing for these messages can

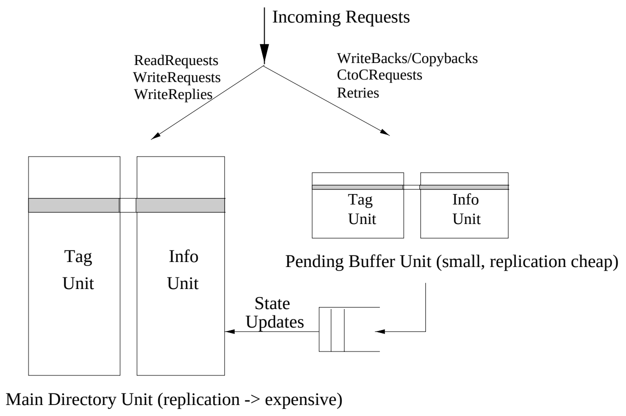


Figure 7. Pending Buffer Design for Large Switches

Multiprocessor System - 16 processors			
Processor		Memory	
Speed	200MHz	Access time	40
Issue	4-way	Interleaving	4
Cache		Network	
L1 Cache	16KB	Switch Size	8x8
line size	32bytes	Core delay	4
set size	2	Core Freq	200MHz
access time	1	Link width	16 bits
L2 Cache	128KB	Xfer Freq	200MHz
line	32bytes	Flit length	8bytes
set size	4	Virtual Chs.	2
access time	8	Buf. Length	4 flits
Switch Directories			
Dir Size	256-2048 entries	Assoc	4-way
Application Workload			
FFT	$16K$ pts	SOR	512x512
TC	128x128	GE	128x128
FWA	128x128		

Table 2. Execution-Driven Simulation Parameters

be moved from the main directory to the pending buffer. All resultant state changes for the main switch directory can be performed when the directory ports are free. The use of a pending buffer is shown in Figure 7. The only disadvantage that remains is when all requests require access to the main directory unit. In such cases, only true 4-way multiporting of the switch directory can help maintain the base crossbar switch. Since we do not foresee this occurring often, we believe that employing a 4-way multiported pending buffer organization and a 2-way multiported directory organization is more cost-effective than a 4-way multiported directory organization.

5 Performance Evaluation

In this section, we present a detailed performance evaluation of the switch directory multiprocessor using extensive execution-driven and trace-driven simulations.

5.1 Simulation Methodology

To evaluate the performance impact of switch directories on the scientific application performance of CC-NUMA

Commercial Workloads	
OLTP/DSS workloads	TPC-C/TPC-D
DBMS/Data Size	DB2/1GB
Trace Content	16 Million Memory Refs
Cache/Memory Latencies	
Cache Access Time	8 cycles
Local Memory Access	100 cycles
CtoC Transfer (Local Home)	220 cycles
Remote Memory Access	260 cycles
CtoC Transfer (Remote Home)	320 cycles
Switch Directory Interconnect	
Directory Size	256 to 2K entries
Directory Associativity	4-way
Directory Hit Latency	200 cycles

Table 3. Trace-Driven Simulation Parameters

multiprocessors, we use a modified version of the Rice Simulator for ILP Multiprocessors (RSIM) [11]. Our base system configuration consists of 16 nodes. Each node consists of a 200MHz processor capable of issuing 4 instructions per cycle, a 16KB L1 cache a 128KB L2 cache, a portion of the local memory, directory storage and a local bus interconnecting these components. With the network components running at the same frequency as the processor (200 MHz), note that any improvements in communication latency becomes a conservative estimate. A detailed list of simulation parameters is also shown in Table 2. The system employs the full-map three-state directory protocol, the MSI cache protocol to maintain cache coherence and a release consistency model. We modified RSIM to employ a wormhole routed bidirectional MIN using 8×8 switches organized in 2 stages as shown earlier in Figure 3. The crossbar switch operation is similar to the description given earlier.

Our switch directory system improves on the base system in the following respects. Each switching element of the bidirectional MIN employs a variable size directory that models the functionality of the DRESAR switch directory. We have selected some numerical applications to investigate the potential performance benefits of the switch directory interconnect. These applications are Fast Fourier Transform (FFT), Transitive Closure (TC), Successive-Over-Relaxation of a grid (SOR), Floyd-Warshall's all-pair-shortest-path algorithm (FWA) and Gaussian Elimination (GE). The input data sizes are shown in Table 2 and the memory characteristics were discussed in Section 2.

We also evaluate the performance benefits of a switch directory interconnect on two commercial benchmarks, the TPC-C online transaction processing benchmark [14], and the TPC-D decision support benchmark [15]. The traces for these two benchmarks were collected at the IBM TJ Watson Research Center using the COMPASS simulator [10]. We used a trace-driven approach for simplicity and limiting simulation execution time. Our trace-driven simulator models a single issue processor, a single layer of 4-way set associative 2MB cache, the MSI cache protocol and the full-map directory protocol. We model a release consistent system by assuming that all write requests are cache hits. Additionally, we model the switch directory interconnect. We use constant memory access latencies for cache miss service time. These parameters were derived from latencies measured on recent multiprocessors [3, 4]. The detailed trace-driven simulation parameters are shown in Table 3.

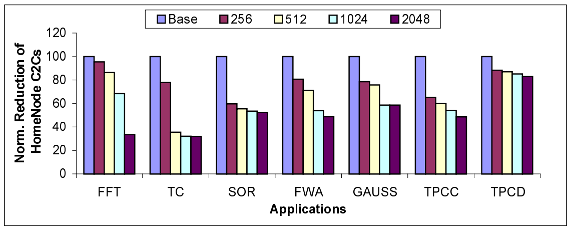


Figure 8. Reduction in Home Node CtoC Transfers

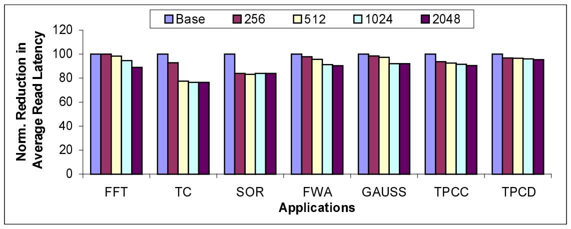


Figure 9. Reduction in the Average Read Latency

5.2 Results and Analysis

In this section, we compare the impact of switch directories (256 entries to 2048 entries) to the base system with no directory caching. The main objective of switch directories is to reduce the number of read requests arriving at the home node for directory lookup and cache-to-cache transfer service. Figure 8 shows this reduction in the number of home node cache-to-cache transfers. On the x-axis, the applications as well as the size of the switch directory is varied. On the y-axis, the number of home node cache-to-cache transfers is presented, normalized to the base multiprocessor. We find that directory caching reduces the number of home node cache-to-cache transfers by as high as 66% and 68% for the FFT and TC applications respectively. Improvements in the other three scientific applications range from 42% to 52%. The TPC-C commercial workload benefits from a reduction of up to 51%, while the TPC-D workload shows only a 17% reduction. Note also that for most applications, except FFT, the benefits seem to reduce as the number of entries in the directory are increased. A directory size of 1K entries seems to be the most reasonable.

By serving many of the cache-to-cache transfers via switch directories, the average read latency may be significantly reduced since slow DRAM lookup is avoided. Figure 9 shows the reduction in average read latency as a function of the number of directory entries. It should be noted here that this latency reduction directly depends on three factors, the number of switch directory hits shown in Figure 8, the fraction of cache-to-cache transfer reads shown earlier in Figure 1 and the ratio of latencies experienced for different types of read requests. From Figure 9, we find that the reduction in latency ranges from roughly 8% to 23% for the scientific applications. The reduction in TPC-C latency is as high as 10% and the reduction in TPC-D latency only reaches up to 5%. The low reduction in latency for commercial workloads can be attributed to our conservative estimate of switch directory hit latency as compared to home directory cache-to-cache transfer latency (see Table 3. When congestion at the network and memory nodes is taken into account, we expect the reduction in read latency to be significantly higher. Finally, the read latency directly af-

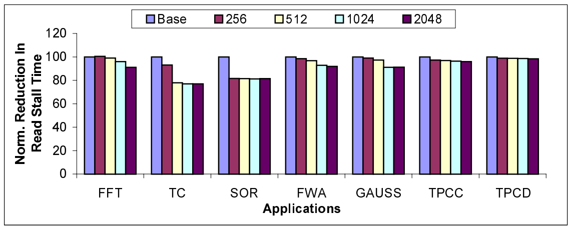


Figure 10. Reduction in the Read Stall Time

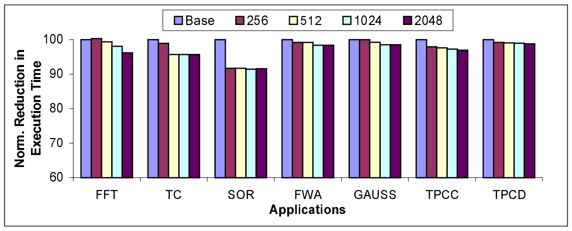


Figure 11. Execution Time Reduction

fects the read stall time experienced by the application and the stall time reductions are similarly shown in Figure 10.

The ultimate metric for application performance is the execution time. Figure 11 shows the reduction in execution time normalized to the base system as a function of the number of switch directory entries. We find that the maximum reduction in execution time is about 9% for the SOR application. The FFT and TC application execution time reduces by up to 4%, while the remaining scientific application show negligible reductions in execution time. The TPC-C commercial workload shows an execution time reduction as high as 4%, while the TPC-D application shows an execution time reduction of 2%.

6 Conclusions

In this paper, we presented a novel hardware caching technique, called switch directory, to improve the CC-NUMA cache-to-cache transfer latency for communication intensive data in scientific and commercial workloads. A trace analysis of communication intensive applications showed that a significant amount of cache-to-cache transfers were required. Thus, remote memory access performance can be improved by caching ownership information in a global directory.

By incorporating small directory caches within each switching element, ownership information was captured as they flowed from the memory to the processor. This information was provided to subsequent read requests and were rerouted directly to the owner cache. This process avoided the need to access the slow DRAM directory at the home node. The main hindrance to this global caching technique was that of maintaining cache coherence and the full-map directory protocol. We organized the caching technique in a hierarchical fashion by utilizing the inherent tree structure of the network. To maintain full-map directory information accurately, writeback and copyback messages sent from the owner's cache to the home node carried the requester pids to the full-map directory. When such messages reached the directory, they not only updated the memory but also updated the directory state and bit vector. The directory caching technique was also kept non-inclusive and thus devoid of the size problem in a multi-level inclusion property.

We also presented the detailed design of a directory embedded switch architecture (DRESAR) and analyzed its performance. We employed a dual-ported set associative SRAM directory organization for a 4×4 crossbar switch cache. We also proposed a pending buffer design to scale to 8×8 crossbar switches. Our simulation results indicate that a directory holding 1024 entries was sufficient to provide up to 50% reduction in cache-to-cache transfers and a 10% improvement in execution time for some of the scientific applications. Improvements to the database applications were minimal, roughly 4% for TPC-C and 2% for TPC-D in execution time. We believe that by combining the proposed switch directory scheme with our previous switch cache framework [5], the CC-NUMA memory access latency can be further reduced.

References

[1] J. Carbonaro and F. Verhoorn, “Cavallino: The Teraflops Router and NIC,” *Proc. of 4th Symp. on High Performance Interconnects*, Aug. 1996.

[2] M. Galles, “Scalable Pipelined Interconnect for Distributed Endpoint Routing: The SGI SPIDER Chip,” *Proc. of 4th Symp. on High Performance Interconnects*, Aug. 1996.

[3] C. Hristea, et al., “Measuring Memory Hierarchy Performance of Cache-Coherent Multiprocessors Using MicroBenchmarks,” *Proceedings of Supercomputing: High Performance Networking and Computing*, 1997.

[4] R. Iyer, N. Amato, L. Rauchwerger and L.N. Bhuyan, “Comparing the Memory System Performance of the HP V-Class and SGI Origin 2000 Multiprocessors using Microbenchmarks and Scientific Applications,” *International Conference on Supercomputing (ICS'99)*, June 1999.

[5] R. Iyer and L.N. Bhuyan, “Switch Cache: A Framework to Reduce the Remote Memory Access Latency of CC-NUMA Multiprocessors,” *Fifth International Conference on High Performance Computer Architecture (HPCA-5)*, Jan 1999.

[6] R. Iyer et al., “A trace driven analysis of sharing behavior in TPC-C,” *2nd Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW'99)*, Jan 1999.

[7] S. Kaxiras and J. Goodman, “The GLOW Cache Coherence Protocol Extensions for Widely-Shared Data,” *Proc. of the International Conference on Supercomputing*, May 1996.

[8] M. Michael and A. Nanda, “Design and Performance of Directory Caches for Scalable Shared Memory Multiprocessors,” *Fifth International Conference on High Performance Computer Architecture (HPCA-5)*, 1999.

[9] A.K. Nanda and L.N. Bhuyan, “Design and Analysis of Cache Coherent Multistage Interconnection Networks,” *IEEE Transactions on Computers*, vol. 42, no. 4, April 1993.

[10] A.K. Nanda, Y. Hu et al., “The Design of COMPASS : An Execution Driven Simulator for Commercial Applications Running on Shared Memory Multiprocessors,” *Proc. of International Parallel Processing Symposium*, April 1998.

[11] V. Pai et al., “RSIM Reference Manual. Version 1.0,” *Department of Electrical and Computer Engineering, Rice University. Technical Report 9705.* July 1997.

[12] P. Ranganathan et al., “Performance of Database Workloads on Shared Memory Systems with Out-of-Order Processors,”*Proceedings of 8th Conference on Architectural Support for Parallel Languages and Operating Systems*, 1998.

[13] J. P. Singh, W.-D. Weber, and A. Gupta, “SPLASH: Stanford Parallel Applications for Shared-Memory,” *ACM SIGARCH Comp. Arch. News*, vol. 20, no. 1, pp. 5–44, March 1992.

[14] Transaction Processing Performance Council, *TPC Benchmark C*, http://www.tpc.org/, 1990.

[15] Transaction Processing Performance Council, *TPC Benchmark D*, http://www.tpc.org/, 1995.