MOBILE HOME NODE: IMPROVING DIRECTORY CACHE COHERENCE PERFORMANCE IN NOCS VIA EXPLOITATION OF PRODUCER-CONSUMER RELATIONSHIPS

A Thesis

by

TARUN SONI

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 2010

Major Subject: Computer Engineering

MOBILE HOME NODE: IMPROVING DIRECTORY CACHE COHERENCE PERFORMANCE IN NOCS VIA EXPLOITATION OF PRODUCER-CONSUMER RELATIONSHIPS

A Thesis

by

TARUN SONI

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 2010

Major Subject: Computer Engineering

ABSTRACT

Mobile Home Node: Improving Directory Cache Coherence Performance in NoCs via Exploitation of Producer-Consumer Relationships. (August 2010)

Tarun Soni, B. Tech., Indian Institute of Technology Roorkee, India

Chair of Advisory Committee: Dr. Paul V. Gratz

The implementation of multiple processors on a single chip has been made possible with advancements in process technology. The benefits of having multiple cores on a single chip bring with it a new set of constraints for maintaining fast and consistent memory accesses. Cache coherence protocols are needed to maintain the consistency of shared memory on individual caches. Current cache coherency protocols are either *snoop* based, which is not scalable but provides fast access for small number of cores, or *directory* based, which involves a directory that acts as the ordering point providing scalability with relatively slower access. Our focus is on improving the memory access time of the scalable directory protocol.

We have observed that most memory requests follow a pattern where in one of the processors, which we will dub the $Producer$, repeatedly writes to a particular memory location. A subset of the remaining cores, which we will dub the $Consumers$, repeatedly read the data from that same memory location. In our implementation we utilize this relationship to provide direct cache to cache transfers and minimize the access time by avoiding the indirection through the directory. We move the directory temporarily to the Producer node so that the consumer can directly request the producer for the cache line. Our technique improves the memory access time by 13% and reduces network traffic by 30% over standard directory coherence protocol with very little area overhead.

To my family

ACKNOWLEDGMENTS

I am very grateful to my advisor Dr. Paul V. Gratz, for giving me this opportunity to work under him. Dr. Gratz with his constant guidance and support helped me move in the right direction throughout my research. His expertise helped me better understand and facilitated my progress on the topic. Without his insight and expert advice this work would have been impossible. I also want to thank him for all the facilities and support he has given me. I hope to take with me all the teachings and experience I gained from working with him.

I would like to thank Dr. Duncan M. Walker and Dr. A. L. Narasimha Reddy for agreeing to be my committee members.

I would also like to express my sincere acknowledgment to Ehsan Fatehi for his help with M5 simulator, Subodh Prabhu for help with tsim_ocin simulator and other CAMSIN group members for their encouragement and intellectual discussion which helped me shape this work.

I would also like to thank my parents for the confidence they had in me without which I would have never been able to pursue and complete my master's study.

Finally, I would like to thank Texas A&M University and the Department of Electrical and Computer Engineering for giving me the opportunity to pursue my masters degree.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Performance improvements which can be achieved through increasing the chip frequency have greatly reduced due to the *memory wall* and the *power wall*. The Memory wall is the increasing disparity between processor and memory speeds. The improvements in memory access time have not been keeping pace with the improvements in computational logic frequency, greatly reducing the improvement in system performance. Power wall is a manufacturing limit to the maximum operating frequency of a chip caused by the exponential increase in power consumption with factorial increase in frequency. The continuous increase in clock frequency leads to rise in power consumption and heat dissipation to levels too expensive to cool. The current rise in highly parallelizable applications and the need to run multiple applications simultaneously gives an opportunity for thread level parallelism. Multiple cores, running at lower frequency can efficiently utilize this thread level parallelism to achieve good performance improvements.

Chip multi-processors(CMP) [1, 2, 3] have been made realizable with the increased gate density available in current technology and have become the focus of recent research to enhance performance. One type of CMP is the Shared Memory multi-processor system which provides a single memory image to the programmer so that parallel programs can exchange information and synchronize with one another to achieve better performance. In large scale CMPs, the memory of a shared memory system is physically distributed across different sites to have faster memory accesses for better performance. *Memory access latency*, the time taken by a core to access

---

The journal model is *IEEE Transactions on Automatic Control.*

data, is still high and further reduced by attaching a cache to each core. Caching allows shared data to be replicated in multiple sites simultaneously which makes it imperative to have a mechanism to ensure a coherent memory. This mechanism is referred to as the cache coherence protocol.

The primary goal of a cache coherence protocol is to provide an uniform memory image such that a modification in data is observed semantically by all the processors. Therefore, the cache coherence protocol has to notify all caches sharing a copy about any modifications done to the data by a processor. The coherence protocol construction plays a crucial role in the overall performance of the Shared memory system. The design of a cache coherence protocol is split into two parts: a specification of state changes of cache blocks and the implementation that is used to accomplish that specification.

A. Cache Coherence Protocols

Various protocols have been devised for maintaining cache coherence, like MSI, MESI, MOESI [4], write-once [5], Synapse [6], Berkeley [7] and many more [8]. They can be broadly classified into write-invalidate and write-update protocols. A write-invalidate protocol has the writing processor invalidate copies of all other processors whereas write-update has the writing processor force other processors to update their copies. Write-invalidate has lower traffic in the connection compared to write-update but has more remote misses.

I have used *MOESI* protocol [4] in my thesis. MOESI is a type of write invalidate protocol which has all of the possible states commonly used in other protocols. *Modified* state means the cache has the only, most recent, correct copy of the data and the memory has incorrect (stale) data. *Owned* state implies the cache line holds the most recent, correct copy and the memory stale but there are other cache lines sharing a copy of the data. This avoids the need to write modified data back to memory before sharing it. *Exclusive* cache line holds the only copy which is same as the data in memory and the processor can modify and change it to Modified state. *Shared* state means there are multiple copies of the data and the memory also shares the same data. *Invalid* cache line means it does not hold a valid copy of the data. The implementation of transition between these states is explained later in the thesis.

B. Cache Coherence Implementations

The coherency protocol implementation in a CMP can be broadly categorized into Snoop based [5, 9, 10, 11] and Directory based [12, 13, 14, 15] cache coherence. The selection of the implementation generally depends on the connection mechanism amongst the cores which could be either a shared bus or a packet switched type interconnection network.

Snoop based protocols are generally used for shared bus architectures. Shared-Bus based architectures [16] have all the processors connected to a single bus. Any transaction by a core is visible to all the remaining cores, and appropriate action can be taken if an operation threatening the coherence is detected. All the cores snoop on the bus and update their state machines on every transaction happening on the bus. When a core reads an address not in its cache, it broadcasts a read request on the snoopy bus. Memory or the cache that has the updated copy responds to the request by supplying the data. If a core wishes to write to an address its cache does not own exclusively, the other cores need to invalidate their copy or update it to the new value. The bus also provides ordering of transactions since a request must first gain access of the bus as master which can be done by only one core at a time. The main drawback of this implementation is that only a single core at a time can broadcast data access requests on the bus. As the number of cores increase, the contention for bus increases causing an increase in access latency. Also a bus also has a physical limitation on the number of cores it can be connected to while transferring data at a certain rate. These drawbacks makes the bus based architecture limited in the number of cores it can support.

Network on Chip (NoC) [17, 18] based architectures have each core connected to a router and all the routers are connected through a packet switched network. This architecture is highly scalable since the network can route multiple requests at the same time. The drawback of this network is maintaining cache coherency is relatively difficult as cores do not have visibility for all the transactions. Snoop based protocols can be extended to a NoC but this would mean a request would need to be broadcasted on the packet switch network to all the cores for them to be able to snoop an the request. The other problem is ordering of requests received at a core, since in a packet switched network we cannot know which request was sent first. There have been implementations to provide this ordering of requests in snoop based protocols in NoCs. The main advantage with Snoop Based protocols is you can have faster direct cache to cache transfers as all cores snoop on a request. A core having a copy of the requested cache line can directly respond back. The drawback is even with the implementation on scalable NoCs as the number of cores increases the performance degrades. As the number of cores increase so does the number of packets broadcasted, increasing the network traffic drastically.

Another implementation of coherence protocols is the Directory-based coherence protocols. A directory node acts as an ordering point and all the transactions go through the directory point. We can store information of all the sharers of the cache line as an entry at the ordering point. A request then does not need to be broadcasted throughout the network and is only sent to the relevant cores that share the particular cache line. Directory based coherence protocols [13] involve sending a memory access request to a directory which then grants permission, stalls or denies the transaction based on the current state of the memory address. The directory maintains a directory entry for each cache block and records the cache locations in which the block is stored. The elimination of broadcast cache coherence messages overcomes the major limitation of scaling machines to large-scale multiprocessor systems. This implementation is scalable as there is no rapid rise in packets with increase in number of cores. The drawback being each transaction has to go through a directory. There are large amounts of packet going to the directory from all the cores which can slow down the directory access and congestion around the router connected to the directory. This drawback is taken care by having smaller distributed directory caches servicing requests for different memory locations. One drawback for both centralized and distributed directory coherence protocol is, as the number of cores scale the average distance to the directory node increases thereby worsening the directory access time which in turn slows down memory access.

In this thesis, we provide an enhancement over the distributed directory cache coherence protocol by having direct cache to cache transfer using Producer-Consumer relationships. The drawback of a directory based protocol is the indirection through a directory which could possibly be located very far from the actual core sharing the data. In our work we try to tackle this problem by avoiding this indirection wherever possible to get performance benefits. This reduction is done by temporary movement of the directory from the $Default\ home$ node to the $Producer$ node which provides the data. This is possible because of a temporally stable Producer-Consumer relationship observed amongst different cores. Once a Consumer is formed it directly requests the Producer for the required data avoiding the indirection needed through the directory. We explain the implementation of this Mobile Home Node Directory Coherence protocol (Mobile Directory Coherence Protocol) in the later sections.

C. Organization

The rest of the thesis is organized as follows. Chapter II talks about previous relevant work in this area. Chapter III gives a background for the work and Chapter IV explains the mobile directory coherence protocol. Chapter V talks about the evaluation and Chapter VI the conclusion and the direction of future work.

CHAPTER II

PREVIOUS WORK

CMPs allow for integration of all system functions including compute processor, caches, communications processor, interconnection networks, and coherence hardware onto a single die [1, 2, 3]. Traditional approaches to cache coherence are broadcast-based snoopy protocols and directory-based protocols.

A. Snoop-Based Coherence

Broadcast-based snoopy protocols have been the most commonly used approach to building symmetric multiprocessors (SMPs) [9, 10, 11]. Snooping keeps caches coherent using a totally ordered network to broadcast coherence transactions directly to all processors and memory [5]. Snooping protocols are successful because they obtain data quickly (without indirection) and avoid the overhead of sequencing invalidation and acknowledgment messages. However, the main limitation of these protocols is that they rely on ordered interconnects, which do not scale beyond a moderate number of cores. They also have the bandwidth overhead of broadcasts.

In the past, there has been considerable effort to retain and scale snoopy protocols by adapting them for split transaction buses [10], hierarchical buses [11], and address broadcast trees [9] that provide a logical bus ordering. Expanding further, existing products, like the IBM Power4 and Power5, retain and scale snoopy coherence protocols onto a ring interconnect [19]. One of the reasons why so much effort has been devoted towards continuously scaling and supporting snoopy protocols, initially designed for bus-based systems, is that they enable direct cache-to-cache transfers and thus do not incur directory indirection for cache misses. For workloads that have fine-grain sharing, direct cache-to-cache transfers provide a huge advantage over going to an ordering point and suffering indirection. But the limitation of scalability is still a major concern and there is a shift towards having packet switch on chip interconnects for CMPs.

There have been techniques to have a total order of snoop requests over unordered networks for achieving the best of both scalability from unordered networks and direct cache-to-cache transfer. Bilir et. al. [20], Marty et. al. [21] have previously shown that snoopy protocols depend on the logical order and not the physical time at which requests are processed, i.e., the physical time at which a snoop request arrives at nodes is not important, as long as the global order in which all nodes in the system observe a particular request remains the same. Much of the work is focused on achieving this logical ordering over unordered networks. Logical ordering of broadcasted messages is achieved through globally-ordered numbers attached to the snoop requests [22, 23], or through a response message traversing the entire logical ring, collecting responses from all nodes [24]. These techniques provide an in-network ordering technique that enables broadcast based snoopy coherence protocols to scale on unordered interconnects. There are other methods that used broadcast for direct cache-to-cache transfer [25] or hybrid of broadcast and directory [20]. But still these implementations require broadcasting of requests over the network for direct cache-to-cache transfers. This adds unnecessary traffic to the network leading to power wastage and may also lead to heavy traffic slowing down the network all together. It is not really feasible for very large scale systems with hundreds of cores on a chip.

B. Directory Based Coherence

Directory-based protocols [12, 13, 14, 15] on the other hand transmit coherence transactions over an arbitrary point-to-point network to distributed ordering points which, in turn, redirect the transaction to a superset of processors caching the block. This implementation does not need an ordered interconnect because it uses explicit message acknowledgements to achieve request ordering and update the directory in a manner that appears atomic. This enables highly scalable interconnects, such as packetized meshes allowing larger systems. Directory protocols are also not broadcast in nature. This imposes lower bandwidth requirements on the interconnect fabric. However, they have higher unloaded latency because of the overheads of directory indirection, along with an additional cost associated with the storage and manipulation of directory state.

Modifications to Directory based protocols have been proposed to use the sharing patterns and other data access patterns to get faster accesses. Bilir et. al. [20] propose a hybrid of directory and broadcast. Jerger et. al. [26] propose a combination of modification in network and directory coherence protocol following sharing patterns for low traffic applications. Prefetching was proposed by Byrd et. al. [27], and Nesbit and Smith [28] to hide long miss latencies but if overly aggressive can increase network traffic. Producer initiated mechanisms were proposed by Abel-Shafi et. al. [29] and Koufaty et. al. [30], in which data is sent to the Consumer caches directly through remote writes or speculative updates to try and update even before the data is requested, leading to wastage in cases where the data is not needed. Another such speculative update mechanism, proposed by Cheng et. al. [31], is used where once a Producer-Consumer relationship is formed when data written in the Producer is sent directly to the Consumer. These implementations have a stricter sense of Producer-Consumer relationship to avoid wasted speculative updates.

In our work we provide both direct cache-to-cache transfer by avoiding the indirection through the directory point and also reduced network traffic since number of packets needed from request to response are reduced. We form a Producer-Consumer relationship as soon as a sharer is invalidated due to a write request. We use this relationship only to avoid the indirection through the directory. There is no prefetching or speculative updates involved avoiding any wasted packets and wasted power. A Consumer directly requests the Producer on a read miss allowing for cache-to-cache transfer. Since no broadcasting is involved it provides good scalability and prevents unnecessary packets in the network thereby reducing the traffic as well as saving power.

CHAPTER III

BACKGROUND

In this chapter we will discuss the MOESI protocol model and the Baseline Directory Coherence Protocol with which we compare our Mobile Directory Coherence Protocol. We now discuss the detailed state transition diagrams for MOESI protocol used in both the implementations.

A. MOESI Protocol

The protocol states stand for *M*odified, *O*wned, *E*xclusive, *S*hared, *I*nvalid.

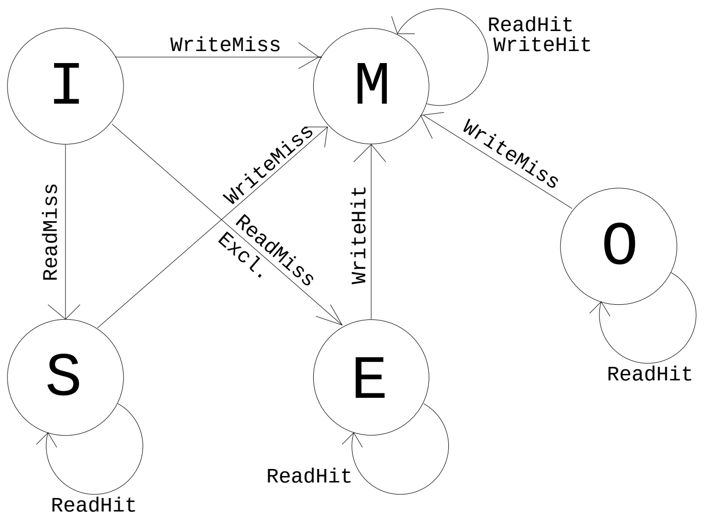


Fig. 1. State Transition Diagram for MOESI Protocol for CPU Requests

The Figure 1 shows the various states and the transition mechanism between each of the MOESI states for misses sent to the cache controller by the core directly connected to it.

The Figure 2 shows the various states and the transition mechanism between each

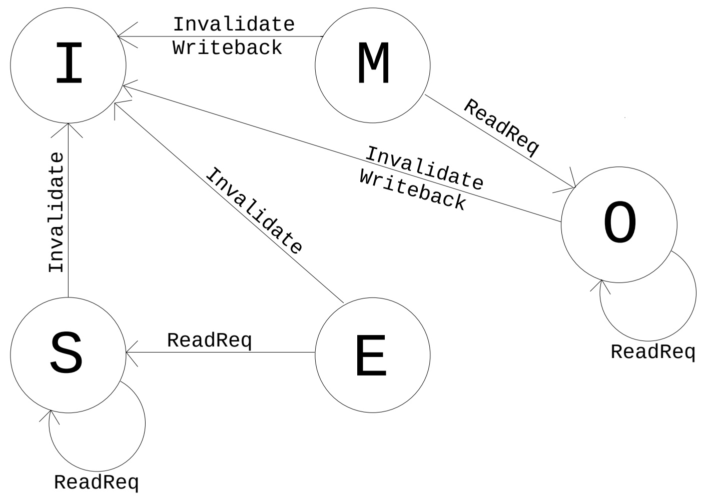


Fig. 2. State Transition Diagram for MOESI Protocol for Network Requests

of the MOESI states for requests received at the cache controller from the network. Now we describe each of the states in detail.

- Modified State: In this state the cache line stored in the cache has both a *dirty* and an *exclusive* copy. Dirty copy means the cache line has the most current data and the copy in memory is stale. Exclusive copy means no other cache has a copy of the cache line. On a read request from another core a cache line in Modified state shares the dirty copy and the state changes to Owned. The exclusive bit is reset but the dirty bit is still set in the owner cache. The cache line in this state when removed from the local cache due to block replacement, or an invalidation caused by read exclusive or a write request from a different core causes a writeback to the upper level of storage.

- Owned State: In this state the owner cache still has a dirty copy of the cache line. Again this could lead to a writeback same as in the Modified state case if the copy in the local cache is invalidated. The copies in other caches remain in shared state after the cache line is removed from the owner cache unlike Modified which does not have any copies in other caches since it is the only copy. On a write request the cache line changes back to Modified state on receiving the invalidation acknowledgements and setting the exclusive bit.

- Exclusive State: In this state there is a single but a clean copy of the cache line. It does not need a writeback if invalidated and changes to Shared state on a read request. It changes to Modified state on a write hit. A cache line when read from memory for the first time and has no sharers the directory responds back with an exclusive message, with data response for a read request, to set the cache line to Exclusive state.
- Shared State: In this state only the valid bit is set and the cache has a clean copy. There are multiple clean copies in other caches for the cache line in this state. It invalidates all the copies on a write request or read exclusive request and changes to Modified or Exclusive state respectively on receiving the acknowledgements. It remains in the same state if there is a ReadReq from another core on this state. It invalidates without any writeback to the memory.
- Invalid State: There is not a copy of the cache line at this local level. The data is read from the higher level of storage or local caches of other cores for read, write or read exclusive requests and changes to Shared, Modified or Exclusive states respectively.

B. Baseline Directory Coherence Protocol

The standard directory coherence protocol is based on the Stanford DASH [13] coherence protocol. The block diagram for this implementation is shown in Figure 3.

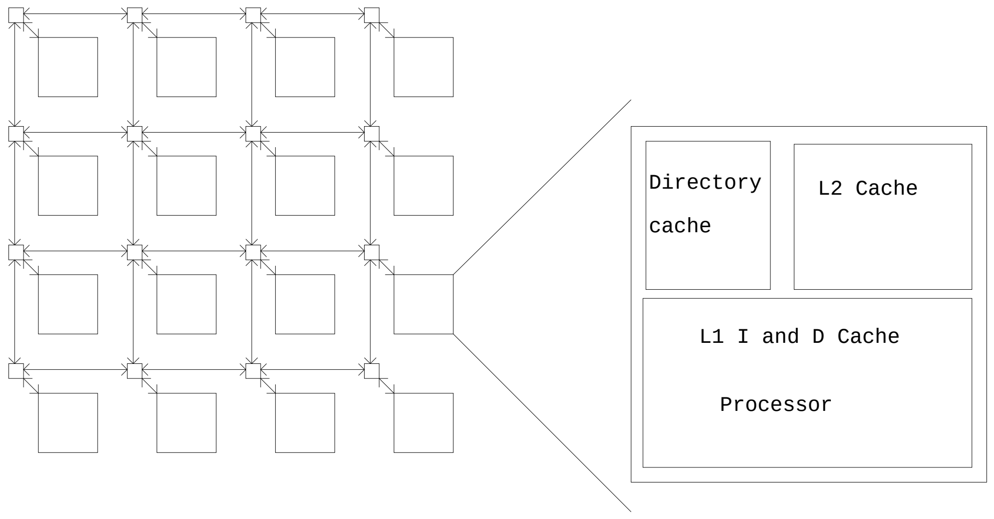


Fig. 3. Block Diagram of 4x4 Mesh

The directory is distributed across all the nodes and each cache line maps to one of the directory nodes. A core, depending on the address of the cache line, sends its requests to the corresponding directory. The types of message requests from cores and the coherence traffic that ensues are shown as follows.

- Read Request:

  The read request mechanism is shown in the Figure 4. When there is a read miss on the local cache line for a read done by a processor, the core sends out a read request to the directory home node. If the directory entry is missing it looks for it in the upper level of cache and updates its local entry. The directory forwards the request to one of the sharers or to the memory depending on the directory entry for the cache line and awaits response. The sharing core or the memory on receiving the request responds back with the data in its reply message. The directory core on receiving the reply sends the data back to the requesting core and updates the sharer information. The responding core resets

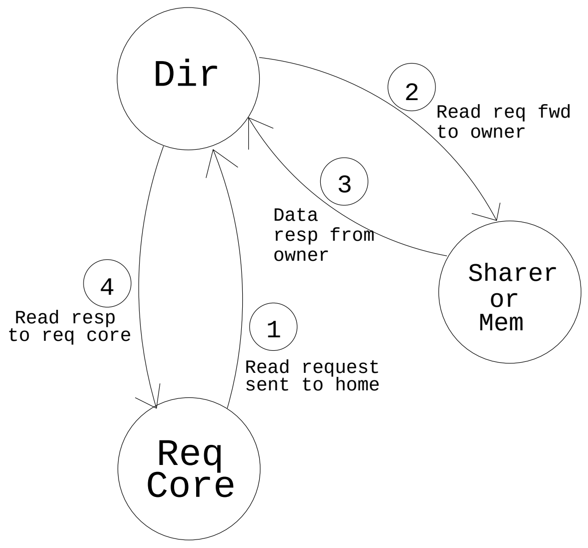


Fig. 4. Read Request Coherence Mechanism in DCP

the exclusive bit if it has been set. The dirty bit is not changed on the core and depending on the dirty bit value the state becomes either Owned or Shared.

- Write Request:

  The Write request mechanism is shown in the Figure 5. When the local cache does not own an exclusive copy of the cache line, it is a write miss and a write request message is sent to the directory. The directory forwards an invalid request to all the sharers with a data request to one of the sharers and waits for the acknowledgements. The core receiving an invalid request invalidates its local copy and sends back an acknowledgement and the core receiving the data request sends back a copy of the cache line with its acknowledgement. The directory while it is waiting for acknowledgements marks the directory entry as busy. Once the directory receives all the acknowledgements, the directory state is updated and it sends acknowledgement and the data response back to the

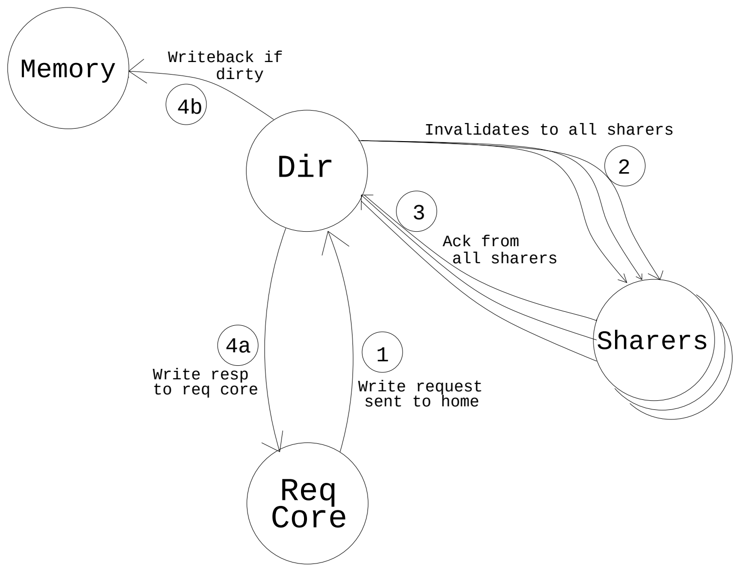


Fig. 5. Write Request Coherence Mechanism in DCP

requesting core. The requesting core on receiving the data and acknowledgements updates the cache line with new data and sets the exclusive bit and the dirty bit. The cache line with both the dirty and the exclusive bit being set is in Modified state.

- Writeback:

  When the local copy of a data is being replaced in a cache it sends a request to the directory to remove it from the sharers list. And if it is the owner of a dirty cache line a Writeback to the upper level of cache or the memory occurs. Even on Invalidate request initiated by directory due to another core requesting exclusive access can cause a dirty cache line to initiate a Writeback.

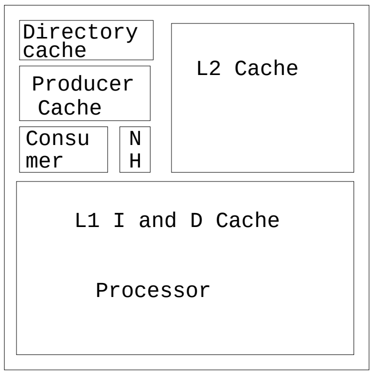
CHAPTER IV

MOBILE DIRECTORY COHERENCE PROTOCOL

The Mobile Home Node Directory Coherence Protocol we have designed and implemented exploits the Producer-Consumer relationship to get better access times. The number of cores and the topology uses is same as the Baseline Directory Coherence Protocol. The only difference is in the composition of the directory cache. We try to remove the indirection through the directory for data accesses by modifying the Baseline Directory Coherence Protocol. The indirection is removed by using the Producer-Consumer relationship observed amongst memory references. *Producer* node is the core which writes to a cache line multiple times without any other node writing to it in between. A node is deemed as a *Consumer* node if it accesses the cache line written by the Producer node in between the various writes. We utilize this relationship and make the Consumer node directly request the Producer node for the latest data avoiding any indirection through the directory node. We use small caches to store the Producer node and Consumer node information on every core. We see negligible overhead and a great amount of benefit by defining a Producer node for every exclusive request instead of waiting for multiple writes.

A. Composition of Directory Cache

We divide the standard directory cache into a set of four new smaller caches as shown in Figure 6. Firstly the standard directory cache which is used same as in the Baseline Directory Coherence Protocol. The *Producer cache* which stores the cache line state at the new node to which the directory has been moved. The *Consumer cache* which stores the location of the *Producer* at a particular core deemed as a Consumer based on it being an old sharer. The *New Home Node* cache stores the location of the new



NH – New Home Node Cache

Fig. 6. Core Block Diagram for Mobile DCP

home node at the default directory location.

B. Producer - Consumer Formation

Figure 7 shows the formation of Producer-Consumer relationship on a write request. A write request is sent to the directory which responds back with the directory entry and also sends out invalidates to other sharers. The sharers invalidate their cache copy and send out the acknowledgements to the requesting core. The requesting core on receiving the directory entry and all the acknowledgements knows the cache line is now exclusive and ready for writing. The sharers on invalidating also update the Consumer cache with the requesting core which now becomes the Producer cache. The Requesting core stores the directory entry in the Producer cache and can receive direct requests from the Consumer caches.

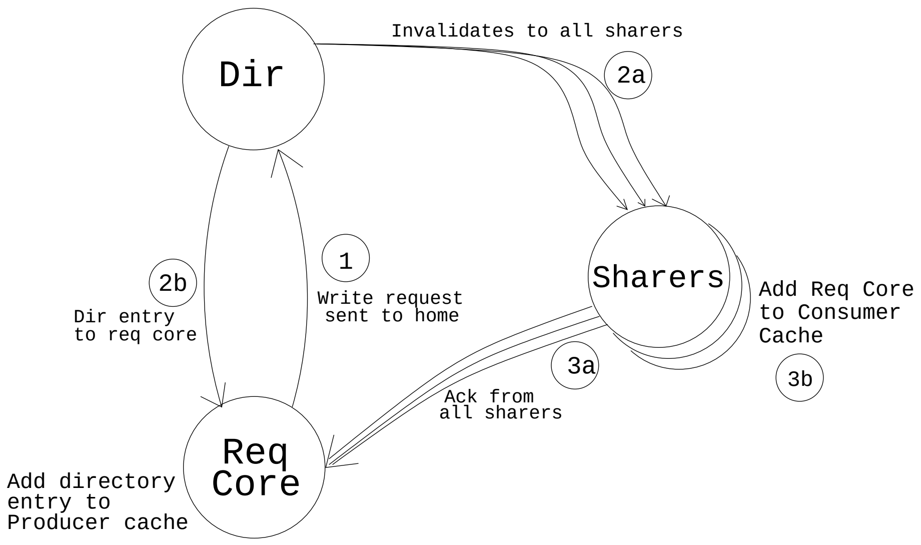

Invalidates to all sharers
Dir
2a
Sharers
1
2b
Write request sent to home
Dir entry to req core
Add Req Core to Consumer Cache
3a
3b
Ack from all sharers
Req Core
Add directory entry to Producer cache

Fig. 7. Producer-Consumer Formation Mechanism for Mobile DCP

C. Coherence Traffic

The type of messages and the coherence traffic that ensues for the Mobile Directory Coherence Protocol is shown as follows.

- Read Request:

  The read request mechanism is shown in the Figure 8. On a read miss in the local cache of a core, the cache controller checks for any Producer information for the address in the Consumer cache. If there is a hit on the Consumer cache a *Read Producer Request* is sent to the Producer core as shown by path 1b, else a Read Request is sent to the directory core as shown by path 1a. The directory core checks both the directory cache and the home node cache for that particular address. If there is a hit on the directory cache it works same as the Baseline Directory Coherence Protocol and forwards the request to one

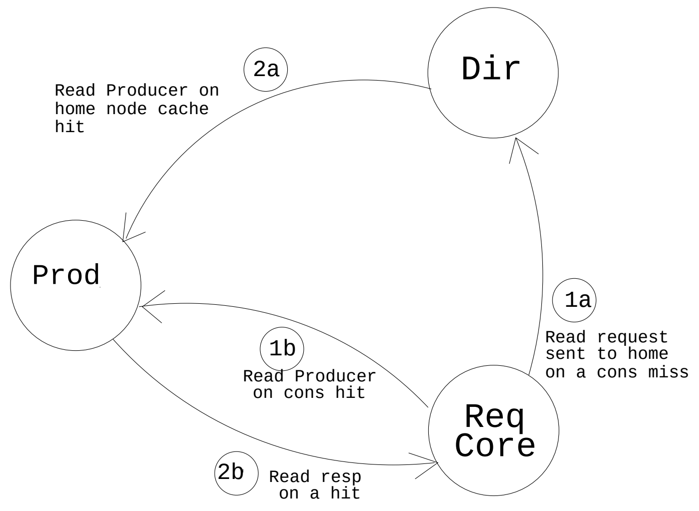

Dir
Prod
Req Core
2a
Read Producer on home node cache hit
1a
Read request sent to home on a cons miss
1b
Read Producer on cons hit
2b
Read resp on a hit

Fig. 8. Read Request Coherence Mechanism for Mobile DCP

of the sharers which responds with data. The directory sends a read response back to the requesting core. But if there is a miss in the directory cache and a hit in the home node cache it sends a *Read Producer Request* as shown by path 2a to the new home node. If there is a miss on both the caches it requests for directory entry information to the upper levels of cache. And if there are no sharers it requests for data from memory same as Baseline Directory Coherence Protocol.

We save on the indirection of accessing the directory if there is a hit in the Consumer cache and it directly requests the Producer cache to respond back for the data. This method allows you to have direct cache to cache transfers if the data sharing follows the Producer-Consumer pattern.

In the scenario the request is forwarded to the default directory and the directory entry is found, then the request is sent to one of the sharers. The network traffic follows the same pattern as the Baseline Directory Coherence Protocol.

- Read Producer Request:

  The Producer core on receiving a read Producer request responds back with the data and adds the requesting core to the directory entry stored in its Producer cache. In the case the local cache does not have a copy, that would mean the core is no longer the Producer then a read request is forwarded to the default directory node.

- Write request:

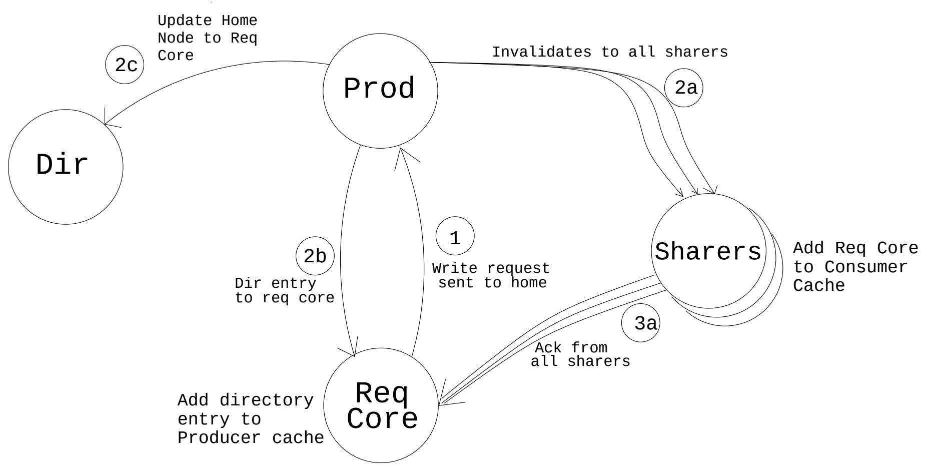

Update Home Node to Req Core
2c
Prod
Invalidates to all sharers
2a
Dir
1
Write request sent to home
2b
Dir entry to req core
Sharers
Add Req Core to Consumer Cache
3a
Ack from all sharers
Add directory entry to Producer cache
Req Core

Fig. 9. Write Request Coherence Mechanism for Mobile DCP

The write request mechanism in Mobile Directory Coherence Protocol is shown in the Figure 9. When there is a write happening in the core, first the local data cache is checked for an exclusive copy. If there is no exclusive copy but there is an entry in Producer cache it sends invalidation requests to all the sharers pointed by the directory entry and speeds up the write process. Since the write is happening at the new home node the invalidate messages can be directly sent and acknowledgements can be directly received avoiding indirection. The sharers on receiving invalidation requests invalidate their local copy and update their Consumer cache with the requesting core. The Producer awaits acknowledgement before setting the exclusive bit and dirty bit in the local cache and updating the state in the Producer cache.

If there is an entry in the Consumer cache of the requesting core, it forwards a write request to the Producer pointed by the entry otherwise it forwards the request to the default directory node. In this scenario the default node needs to be informed of the movement of directory entry by an explicit message.

If there is no entry in the Producer cache then the request is sent to the default directory which can work like in the case of Producer Consumer formation. If there is an entry in the new home node cache then it forwards the request to the new directory location.

Now, at the Producer node or the directory node it checks for a directory entry and it sends out invalidation requests to all the sharers and the entry back to the requesting core. The sharers on receiving invalidation requests invalidate their local copy and update their Consumer cache with the requesting core location before sending out their acknowledgements to the requesting core. At the requesting core on receiving the entry and the acknowledgements from all the cores it updates its Producer cache and the local cache. After the update an *Update Home Node* request is sent to the default directory node if the entry came from the requested Producer cache.

If there is no entry at the Producer cache a write request is forwarded to the default directory cache. And if there is no entry in the default directory cache but an entry in the home node cache, it forwards the request to that particular Producer updating its own home node cache. If there is no entry in both the caches then a directory entry request is sent to upper level of cache.

- Update Home Node:

  When a requesting core gains exclusive access through a direct request to a Producer core then an update home node request is sent to the default directory node to update its new home node cache with this new Producer.

- Writeback:

  If a cache line in local data cache is swapped out and the dirty bit is set. It checks for an entry in the local Producer cache, and is updated before sending to the default directory node. But if there is no entry in Producer cache an update request is sent to the directory. There is writeback to memory in either case. If the local cache line being swapped out does not have a dirty bit set then there is no writeback.

D. Exceptions and Correctness

We need to take care of special exception cases or race conditions that might arise in the Mobile Directory Coherence Protocol. We now explain the issues or exceptions that can occur and how we handle them in our implementation.

Stale data in Consumer Cache: In the case there is stale data in Consumer cache and it points to a Producer core which no longer has the directory entry, we address this issue by forwarding the request to the default directory node. The default home node can process the request itself or forward the request to any new home node to which the directory entry might have been moved. Finally, the Consumer cache entry is removed or updated to point to the new Producer node depending on the response message.

Acknowledgements or Requests received at the Producer core: A race condition occurs when there are multiple messages reaching a particular node whose order affects the output of the system. For Baseline Directory Coherence Protocol the directory acts as the ordering point for such messages and there are no races by design. In our Mobile Directory Coherence Protocol all the messages are also ordered by directories except for the scenario when the directory entry is in motion. Such race conditions can only happen for our protocol in Producer Consumer formation when the directory entry is in motion. A problem occurs if an acknowledgement is received before the directory entry has been received at the Producer node. We address this problem by adding an entry at the Producer cache as soon as an acknowledgement is received with all the presence vector bits set and on receiving the acknowledgements the bits corresponding to that core are reset. On receiving the new directory entry we bitwise and the current presence vector with the presence vector from the directory entry to check if all the acknowledgements have been received. We wait for all the bits to be reset before the new directory core knows it has an exclusive copy. Until all the bits are reset the write request on the cache line is still being processed which would mean any new requests on the cache line are stalled. This also ensures any new requests from the Consumer caches arriving at the Producer node before the directory entry is updated do not get sent back to the default node instead wait at this node for the write request to complete.

CHAPTER V

EVALUATION

A. Experimental Methodology

The performance of a *memory system*, consisting of the memory, cache and the interconnection network, can be considered to be the amount it contributes to the time needed to run a program. The time it contributes is the number of processor cycles wasted for a memory access due to memory system delays. In a multiprocessor, the performance is affected by the frequency of memory accesses and the latency of the memory system. The latency of the memory system depends on many factors like the network topology and speed, the number of processors and the size of the system, the frequency and size of the messages, and the memory access latency. The cache coherence protocol impacts the network traffic which added with the regular memory fills and spills determines the request rate, message frequency and size thereby impacting the overall latency of the memory system. In order to obtain the accurate latency of the memory system for both the protocols we need to implement detailed models of the cache coherence protocol and the interconnection network. Figure 10 shows the flow of the analysis methodology implemented. The steps involved in the evaluation are as follows:

- Generating the traces using M5 simulator: M5 simulator [32] is a modular platform in which major simulation structures like CPUs, Caches, etc. are represented as objects. M5's object orientation makes it easy to instantiate multiple CPU objects. We instantiate CPUs with ALPHA ISA and run PARSEC benchmarks [33] to generate the memory access traces. M5 is run in full system mode which simulates a complete system including a kernel, I/O devices

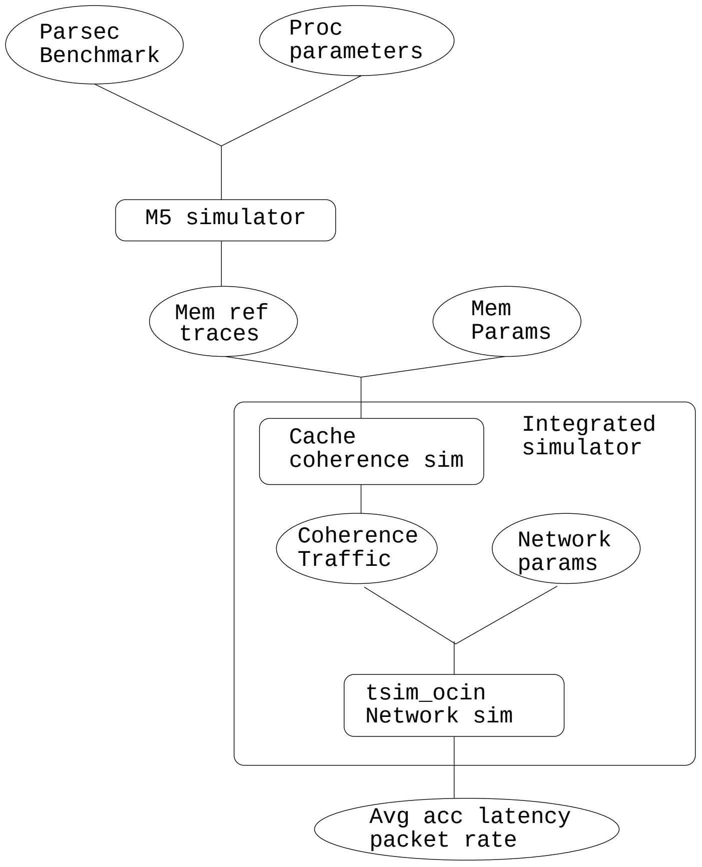


Fig. 10. Analysis Methodology Flow

Table I. CMP Configuration Parameters

Parameter	value
Number of cores	16
Clock frequency	2GHz
L1 D cache	64kB
L1 I cache	32kB
Line Size	64B

etc. The parameters passed to the M5 simulator are shown in the Table I. We use cross compiled binaries for the PARSEC benchmark for ALPHA ISA. The PARSEC benchmark applications are divided into three phases: initial serial phase, a parallel phase, and a final serial phase. The parallel phase is marked as the region of interest and we run the benchmarks in this region for the cycle accurate simulations [34].

- Cache Coherence simulator: We have designed a simulator to accurately implement the protocols described in this thesis to get the access latency . The simulator determines the state of the cache block and the corresponding directory entry for each of the memory reference in the trace. This state consists of the cache tags and the directory pointers to all the sharers. First, we run the coherence simulator stand alone and compared the reduction in latency, assuming a fixed latency per hop, for the complete benchmark. We calculate the hop counts assuming dimensional order routing. This gives us an estimate of the performance improvement that can be expected and the simulations run time is lower compared to cycle by cycle simulation so a quick simulation is feasible. The parameters passed to the coherence simulator are shown in the Table II.

- Cache Coherence simulator with network simulator: We have integrated tsim_ocin [35] a network simulator which runs cycle accurate simulations. We generate the net-

Table II. Cache Coherence Parameters

Parameter	Baseline DCP value	Mobile DCP value
L2 Cache size	2MB	2MB
Associativity	4	4
L2 Latency(cycles)	8	8
Block Size	64B	64B
Directory size(entries)	16k	4k
Dir Latency(cycles)	4	4
Producer Size(entries)	-	8k
Consumer Size(entries)	-	256
New Home Size(entries)	-	16k

Table III. Interconnection Network Parameters

Parameter	value
Number of terminals	16
Hops/cycle	1
Wire delay(cycle)	1
Routing algo	xydor

work traffic based on the cache coherence traffic. The cache coherence simulator creates the messages to be sent over the network and the network depending on the cycle and the dependencies injects those packets. The request packets are injected based on the cycles obtained from the traces. The response or reply packets are injected into the network when the request packets reach the destination. This implementation gives us a much more accurate model of the traffic that will be seen over the network. Since the run time is very high we run only the region of interest for the benchmark traces. The network parameters passed to the simulator are shown in Table III.

We have used the values obtained from the network simulator to make an estimate of improvement in runtime and energy consumption. For runtime improvement we use the Average L2 access time as feedback in M5 simulator as L1 miss latency and estimate the runtime. For energy savings we normalize the product of total number of flits and the average hop count for Mobile Directory Coherence Protocol(Mobile DCP) with respect to Baseline Directory Coherence Protocol (Baseline DCP).

B. Results

The simulations were run on the PARSEC Benchmarks. GM stands for Geometric Mean which was calculated for computing the average of the Normalized values across all benchmarks.

1. Average L2 Cache Miss Latency

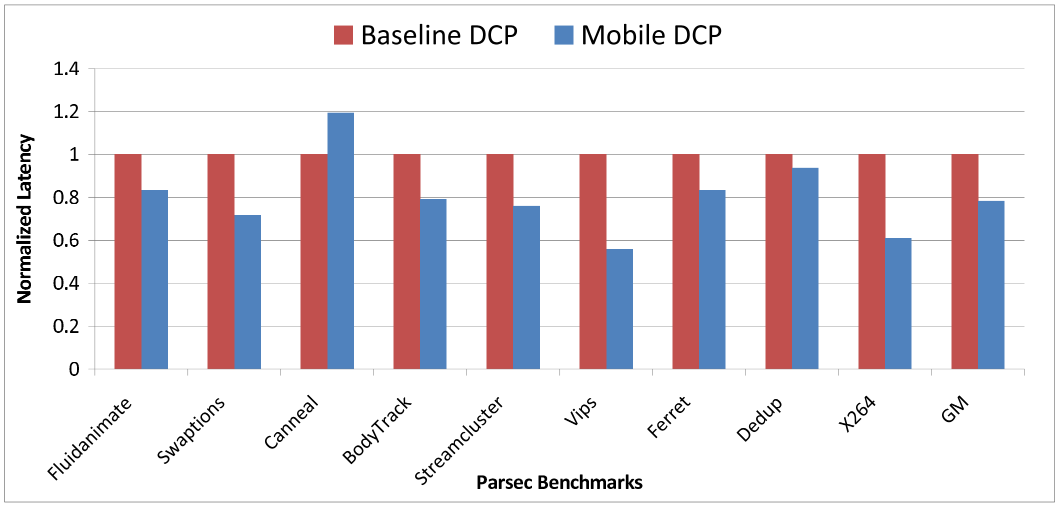


Fig. 11. Normalized Average L2 Cache Miss Latency

Figure 11 shows average L2 cache miss latency for all the benchmarks for the Mobile DCP normalized with respect to Baseline DCP. We see a maximum improvement of 45% and an average improvement of 22% across all benchmarks. This latency reflects the actual improvement offered by the Mobile DCP with respect to the Baseline DCP. The latency for the remote misses is reduced by removing the directory indirection thereby improving our overall L2 miss latency.

For Canneal benchmark we see latency degradation for the Mobile DCP compared to Baseline CP. The increase in access time is because there are very few Producer-Consumer relationships formed to get the benefit from the modification. The L2 cache miss rate is high implying a lot of block replacements at L2 cache level which breaks the Producer Consumer relationships if any formed. Another factor that can cause degradation for Mobile DCP is that the Directory cache for default home node is smaller and since the larger Producer Directory cache is not fully utilized due to lack of Producer-Consumer relationships resulting in many directory misses.

2. Average L2 Cache Access Latency

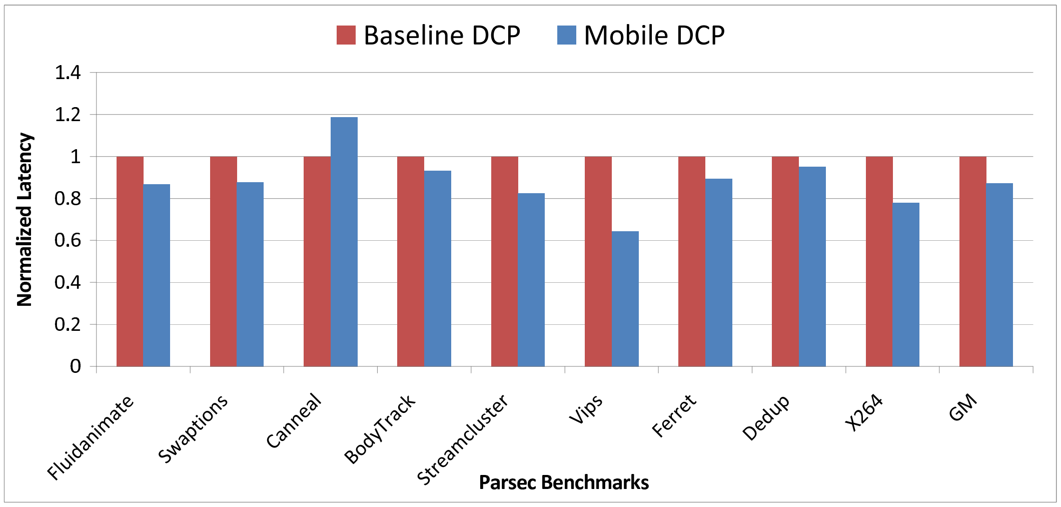


Fig. 12. Normalized Average L2 Cache Access Latency

Figure 12 shows the normalized average L2 cache access time. We see upto a maximum improvement of 35% and an average improvement of 13% across all benchmarks for the average L2 cache access latency for Mobile DCP with respect to the Baseline DCP. This improvement reflects the possible improvement that can be seen in the overall performance of the system. These values are given as feedback to M5 simulator as L1 miss latency to estimate the runtimes for the benchmarks.

3. Injection Rates and Average Packet Latency

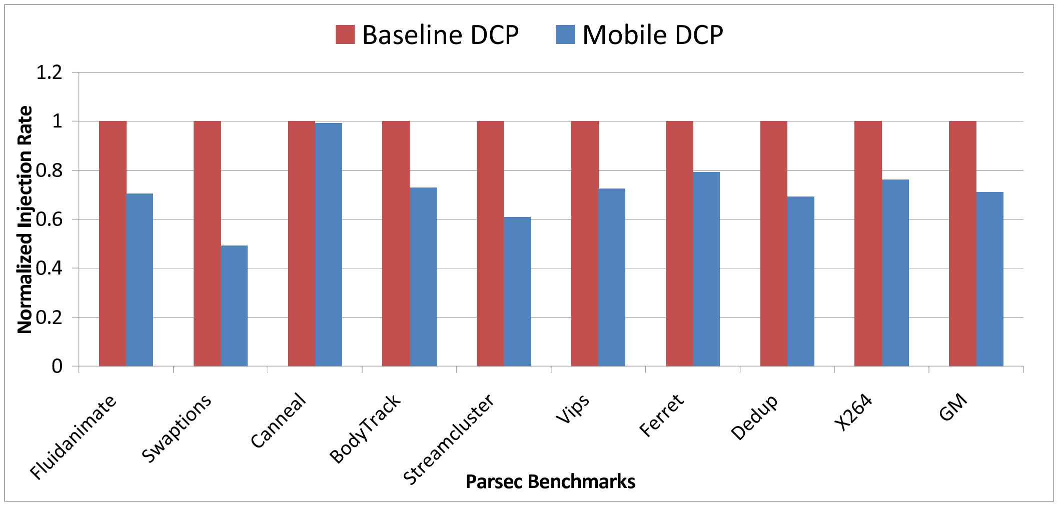


Fig. 13. Normalized Injection Rate

Figure 13 shows the better injection rates for all the benchmarks in the Mobile DCP implementation. We also see an improvement in average packet latency as show in the Figure 14. We see the improvement in network performance parameters like injection rate and average packet latency because the Mobile DCP reduces the network traffic by reducing the request and response packets. This reflects the overall improvement in network traffic.

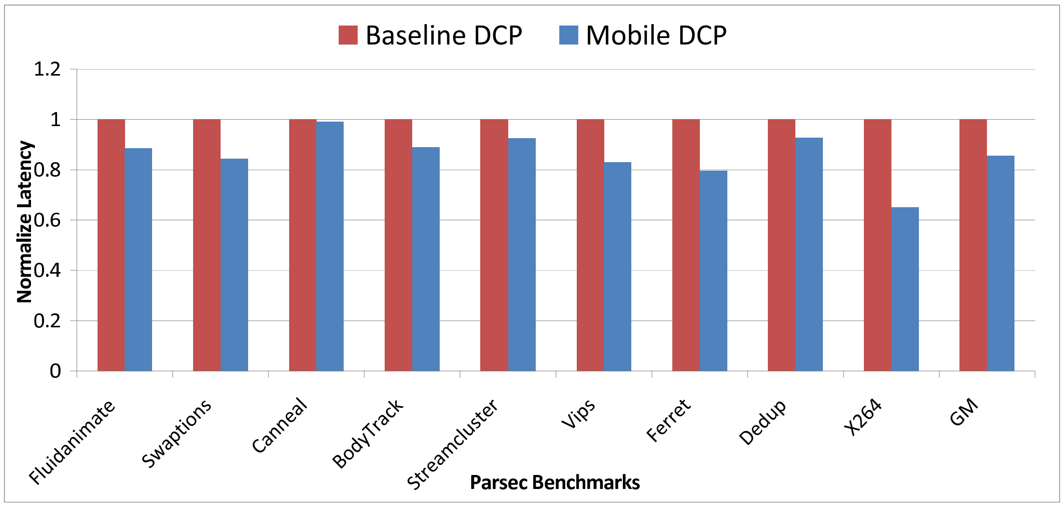


Fig. 14. Normalized Average Packet Latency

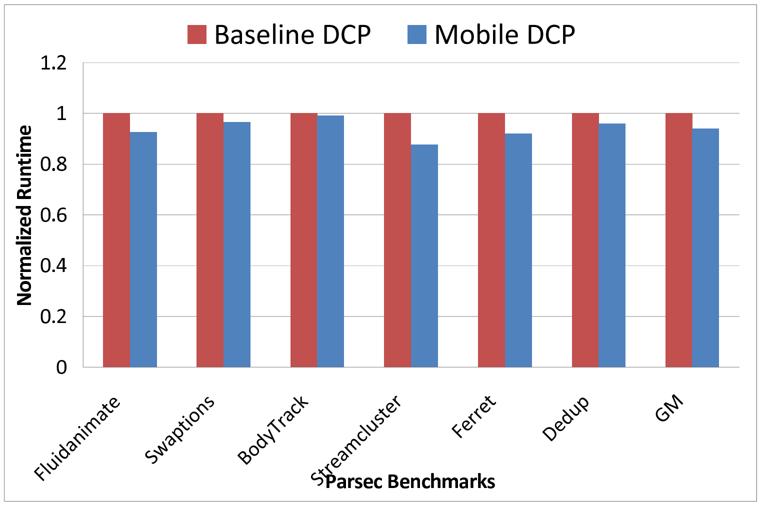


Fig. 15. Normalized Runtime

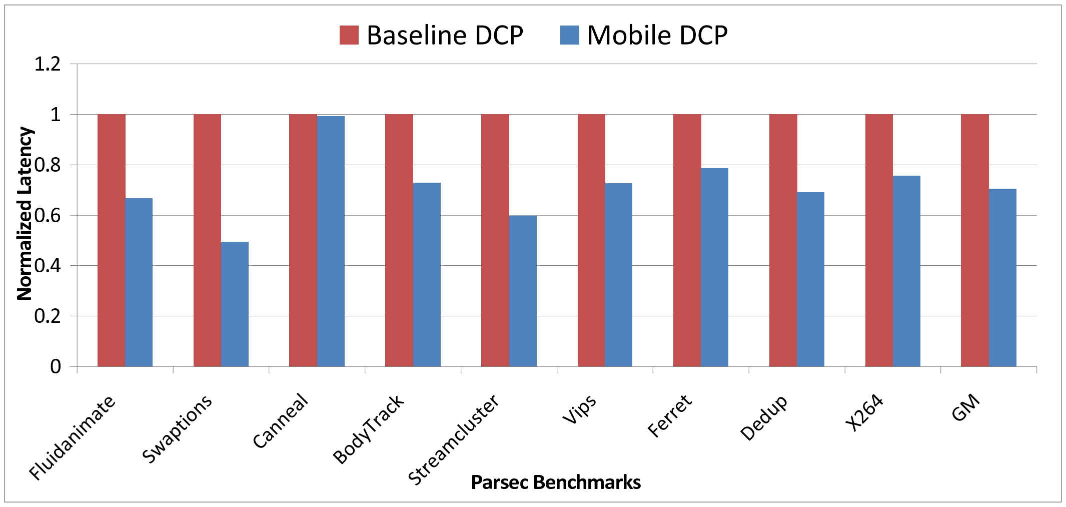


Fig. 16. Normalized Network Energy Consumption

4. Estimated Runtime and Energy Savings

Figure 15 shows the Normalized estimated runtimes and Figure 16 shows the Normalized estimated energy consumptions. We see an average runtime improvement close to 8% across all benchmarks and a maximum improvement of 18% in one of the benchmarks. We do not have runtime values for three benchmarks due to long run time and simulator issues with the modified direct miss latencies. The average energy savings are around 30% across all benchmarks. These estimates show the overall improvement the Mobile DCP provides over the Baseline DCP both in terms of performance and energy consumption.

5. Summary

We see an improvement in the average L2 miss and access latencies which improves the overall performance of the system. The average network packet latency and injection rates also improve due to the reduction in network packets needed for directory indirection. The improvement in network traffic leads to improvement in energy consumption of the network modules.

CHAPTER VI

CONCLUSIONS

The performance improvement and the energy and power consumptions depend on the network traffic depend on the cache coherence protocol. The coherence protocol can provide faster response to local cache misses and the coherence traffic impacts the network traffic and power consumptions drastically.

The previous work that has been done to minimize cache miss latencies by direct cache to cache transfers is based on ordering of snoop broadcasts [22] or use tokens with broadcast [25]. Most of the work for direct cache to cache transfers involves some sort of broadcasts which always causes an increase in traffic that could be detrimental with further scaling to more number of cores and leads to higher energy consumptions.

There are also methods that use Producer Consumer relationships or other such data sharing relationships for speculative updates in the form of prefetch or Producer initiated writes. These methods try to reduce the number of local misses by making speculative updates but causes wasted data transmissions causing increase in network traffic and thus increasing the energy consumed. They provide better performance by reducing the number of remote misses but the energy and power consumptions are increased.

In our Mobile Directory Coherence Protocol we see an improvement in network performance parameters like injection rate and average flit latency because the protocol reduces the network traffic by reducing the number of request and response packets. The network traffic reduction is reflected in the energy savings of close to 30% average across the benchmarks. We also see an improvement in the Memory access times by faster and direct cache to cache transfers through Producer-Consumer relationships. The improvement in memory access times is apparent from the reduction in L2 miss latency by 22% and L2 access latency by 13% which results in improvement of estimated runtime by 8% average across the benchmarks. We not only have better access times through direct cache transfers but also reduce the network traffic improving both performance and provide energy savings.

We see an increase in directory misses for benchmarks that have a high miss rate due to the smaller directory cache size in our Mobile DCP. A possible future direction is to reduce these directory misses by investigating methods to merge the directory cache and the Producer cache. We can use a single larger cache for storing both default and mobile directory entries reducing the number of directory misses by providing a larger pool of cache lines reducing conflict misses. If the mobile directory entries are occupying a lower number of cache lines the remaining larger set of cache lines can be used for directory entries. The degradation caused by more directory misses due to larger cache miss rate can be tackled and we can get better performance even for Canneal Benchmark. Another direction to investigate for improving Mobile DCP is restricting the movement of directory by having better sharing patterns. We can use techniques like keeping a count of writes done by a particular core and have a threshold point for the count before the directory entry can be moved to that core.

REFERENCES

[1] "IBM Power6," http://www.ibm.com/developerworks/power/library/pa-expert1.html, 2004.

[2] J. Held, J. Bautista, and S. Koehl, "From a few cores to many: A tera-scale computing research overview," Tech. Rep., Intel, 2006.

[3] D. Wentzlaff, P. Griffin, H. Hoffmann, Liewei Bao, B. Edwards, C. Ramey, M. Mattina, Chyi-Chang Miao, J.F. Brown, and A. Agarwal, "On-chip interconnection architecture of the tile processor," *IEEE Micro*, vol. 27, no. 5, pp. 15–31, 2007.

[4] Advanced Micro Devices, *AMD64 Architecture Programmers Manual Volume 2: System Programming*, 3.14 edition, 2007.

[5] J. Goodman, "Using cache memories to reduce processor-memory traffic," in *Proc. of the 10th Annual International Symposium on Computer Architecture*, 1983, pp. 124–131.

[6] S. J. Frank, "Tightly coupled multiprocessor systems speed memory access times," *Electronics*, vol. 57, no. 1, pp. 164–169, 1984.

[7] R. H. Katz, D. A. Wood, S. J. Eggers, C. Perkins, and R. G. Sheldon, "Implementing a cache consistency protocol," Tech. Rep., University of California at Berkeley, Berkeley, CA, USA, 1984.

[8] J. Archibald and J.-L. Baer, "Cache coherence protocols: evaluation using a multiprocessor simulation model," *ACM Trans. Comput. Syst.*, vol. 4, no. 4, pp. 273–298, 1986.

[9] A. Charlesworth, "Starfire: Extending the SMP envelope," *IEEE Micro*, vol. 18, no. 1, pp. 39–49, 1998.

[10] M. Galles and E. Williams, "Performance optimizations, implementation, and verification of the SGI challenge multiprocessor," in *Proc. of the 27th Hawaii International Conference on System Sciences*, 1994, vol. 1, pp. 134–143.

[11] D. J. Schanin, "The design and development of a very high speed system bus the encore multimax nanobus," in *Proc. of 1986 ACM Fall Joint Computer Conference*, Los Alamitos, CA, USA, 1986, pp. 410–418.

[12] L. M. Censier and P. Feautrier, "A new solution to coherence problems in multicache systems," *IEEE Transactions on Computers*, vol. 27, no. 12, pp. 1112–1118, 1978.

[13] D. Lenoski, J. Laudon, K. Gharachorloo, A. Gupta, and J. Hennessy, "The directory-based cache coherence protocol for the DASH multiprocessor," in *Proc. of the 17th Annual International Symposium on Computer Architecture*, 1990, pp. 148–159.

[14] N. Eisley, L.-S. Peh, and L. Shang, "In-network cache coherence," in *Proc. of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, 2006, pp. 321–332.

[15] A. Agarwal, R. Simoni, J. Hennessy, and M. Horowitz, "An evaluation of directory schemes for cache coherence," in *Proc. of the 15th Annual International Symposium on Computer Architecture*, 1988, pp. 280–289.

[16] J. R. Goodman and P.J. Woest, "The Wisconsin Multicube: a new large-scale

cache-coherent multiprocessor," in *Proc. of the 15th Annual International Symposium on Computer Architecture*, 1988, pp. 422–431.

[17] W. J. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," in *Proc. of the 38th Annual Design Automation Conference*, 2001, pp. 684–689.

[18] A. Hemani, A. Jantsch, S. Kumar, A. Postula, J. Berg, M. Millberg, and D. Lindqvist, "Network on chip: An architecture for billion transistor era," in *Proc. of the 18th IEEE Norchip Conference*, 2000.

[19] B. Sinharoy, R. Kalla, J. Tendler, R. Eickemeyer, and J. Joyner, "Power5 system microarchitecture," *IBM Journal of Research and Development*, vol. 49, no. 4.5, pp. 505–521, 2005.

[20] E. E. Bilir, R. M. Dickson, Y. Hu, M. Plakal, D. J. Sorin, M. D. Hill, and D. A. Wood, "Multicast snooping: A new coherence method using a multicast address network," in *Proc. of the 26th International Symposium on Computer Architecture*, 1999, pp. 294–304.

[21] M. R. Marty and M. D. Hill, "Coherence ordering for ring based chip multiprocessors," in *Proc. of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, 2006, pp. 309–320.

[22] N. Agarwal, L.-S. Peh, and N. K. Jha, "In-network snoop ordering (INSO): Snoopy coherence on unordered networks," in *Proc. of the 15th International Symposium on High Performance Computer Architecture*, Raleigh, North Carolina, 2009, pp. 67–78.

[23] M. M. K. Martin, D. J. Sorin, A. Ailamaki, A. R. Alameldeen, R. M. Dickson,

C. J. Mauer, K. E.Moore, M. Plakal, M. D. Hill, and D. A. Wood, “Timestamp snooping: An approach for extending SMPs,” in *Proc. of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2000, pp. 25–36.

[24] K. Strauss, Xiaowei Chen, and J. Torrellas, “Uncorq: Unconstrained snoop request delivery in embedded-ring multiprocessors,” in *Proc. of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, 2007, pp. 327–342.

[25] M. M. K. Martin, M. D. Hill, and D. A. Wood, “Token coherence: Decoupling performance and correctness,” in *Proc. of the 30th Annual International Symposium on Computer Architecture*, 2003, pp. 182–193.

[26] N. D. E. Jerger, L.-S. Peh, and M. H. Lipasti, “Circuit-switched coherence,” in *Proc. of the 2nd ACM/IEEE International Symposium on Networks-on-Chip*, 2008, pp. 193–202.

[27] G.T. Byrd and M.J. Flynn, “Producer-consumer communication in distributed shared memory multiprocessors,” *Proceedings of the IEEE*, vol. 87, no. 3, pp. 456–466, 1999.

[28] K. J. Nesbit and J. E. Smith, “Data cache prefetching using a global history buffer,” in *Proc. of the 10th International Symposium on High Performance Computer Architecture*, 2004, pp. 96–96.

[29] H. Abdel-Shafi, J. Hall, S. V. Adve, and V. S. Adve, “An evaluation of fine-grain producer-initiated communication in cache coherent multiprocessors,” in *Proc. of the 3rd International Symposium on High-Performance Computer Architecture*, 1997, pp. 204–215.

[30] D. A. Koufaty, X. Chen, D. K. Poulsen, and J. Torrellas. Data, "Data forwarding in scalable shared-memory multiprocessors," in *Proc. of the 9th International Conference on Supercomputing*, 1995, pp. 255–264.

[31] L. Cheng, J. B. Carter, and D. Dai, "An adaptive cache coherence protocol optimized for producer-consumer sharing," in *Proc. of the 13th International Symposium on High Performance Computer Architecture*, 2007, pp. 328–339.

[32] N.L. Binkert, R.G. Dreslinski, L.R. Hsu, K.T. Lim, A.G. Saidi, and S.K. Reinhardt, "The M5 simulator: Modeling networked systems," *IEEE Micro*, vol. 26, pp. 52–60, 2006.

[33] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," Tech. Rep. TR-811-08, Princeton University, Jan 2008.

[34] M. Gebhart, J. Hestness, E. Fatehi, P. Gratz, and S. W. Keckler, "Running PARSEC 2.1 on m5," Tech. Rep., The University of Texas at Austin, Department of Computer Science, Oct 2009.

[35] S. Prabhu, B. Grot, P. V. Gratz, and J. Hu, "Ocin-tsim: DVFS aware simulator for NoCs," in *The 1st Workshop on SoC Architecture, Accelerators and Workloads*, 2010, pp. 48–55.

VITA

Tarun Soni received his B. Tech degree in electronics and communication engineering from the Indian Institute of Technology (IIT) Roorkee, India in June 2006. After completion of his undergraduate studies he worked as a Design Engineer at Texas Instruments in India until July 2008. He joined Texas A&M University in August 2008 to pursue his master's degree in Computer Engineering and graduated in August 2010. His research interests lie in the area of computer architecture.

Tarun Soni may be reached at:
102 WERC,
Texas A&M University,
College Station, TX 77843-3126
E-mail: tarun1985@gmail.com