Performance Improvement by N-Chance Clustered Caching in NoC based Chip Multi-Processors

A B. Tech Project Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of

Bachelor of Technology

by

Rakesh Yarlagadda
(07010138)

and

K Sanmukh Rao
(07010123)

under the guidance of

Dr. Hemangee Kapoor

to the

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI
GUWAHATI - 781039, ASSAM

CERTIFICATE

*This is to certify that the work contained in this thesis entitled "**Performance Improvement by N-Chance Clustered Caching in NoC based Chip Multi-Processors**" is a bonafide work of **Rakesh Yarlagadda** (**Roll No. 07010138**) and **K Sanmukh Rao** (**Roll No. 07010123**), carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati under my supervision and that it has not been submitted elsewhere for a degree.*

Supervisor: **Dr. Hemangee Kapoor**

Assistant Professor,

May, 2011

Department of Computer Science & Engineering,

Guwahati.

Indian Institute of Technology Guwahati, Assam.

Acknowledgements

We would like this opportunity to acknowledge the support and help of our guide Dr. Hemangee Kapoor. On one hand she gave us the independence of choosing our own research topic, working as per our schedules, exploring our ideas and on the other hand she kept us motivated, made sure that we did not loose our focus and warned about the fallacies in our ideas and helped us in working on them.

We are also grateful to our Head of Department, Dr. Purandar Bhaduri, for providing a nice working atmosphere in the department.

We owe our gratitude to the department staff who worked hard to ensure proper working of all the computing facilities.

And, above all, we are thankful to our families and friends without whose support, nothing could have been possible.

Abstract

Cache management is one of the key factors that affect the performance of present day Chip Multi-Processors. The main aspects that govern the cache management are the access latency and the cache space utilization. This paper proposes 3-chance clustered caching cache management scheme in NoC based multi-core cache coherent systems, where it targets to address both the issues. The L2 banks are formed into a cluster and are non inclusive. The cache management policy concentrates on increasing the life time of a cache block by giving up to 3 chances by rotation of data among the L2 caches and clustering keeps the data close to the processors thereby decreasing the access latency. The caches act as non-inclusive for increasing the cache space, which increases the access latency but is reduced by 2 level directory protocol implemented for cache coherence and cache clustering. The evicted L1 cache bocks are stored in the Cluster home L2 bank and the evicted L2 cache blocks follow the 3-Chance rotation algorithm. Experiment results based on full-system simulation show that for 2D-MESH, 3-Chance Clustered caching can increase the performance by $9 \sim 15\%$.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

From the beginning of computer architecture the basic entities are CPU (Central Processing Unit) and Main Memory. The instructions and the data are present in the memory and they are fetched and executed in the processor. Now the transactions with the memory are slow because of its hardware constraints and to increase its performance cost factors come up. So the researchers came up with Cache: small and high speed memory which sits in between processor and main memory. This cache is known as L1 cache. Now the memory operations latency has been reduced if a L1 cache hit occurs, but if not the request is sent to the main memory there by increasing the penalty. Now the researchers came up with putting a bigger cache in between L1 and Main memory, this cache is called L2 cache. L2 cache is much bigger than the L1 but much smaller than the Main memory. Now the data movement goes from Main memory to L2 and then to L1 and finally serving the processor.

As we have seen the memory hierarchy in Fig: 1.1, now the question comes to what should be the memory hierarchy in a multiprocessor system. A simple implementation that can come to mind will be processors with L1 cache each, and all processors sitting on a

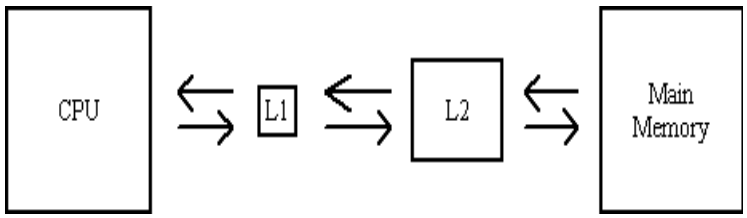

Fig. 1.1 Memory Hierarchy

chip with their L1's and the L2 cache and Main memory off chip. But this is not a well designed or it has many draw backs and its efficiency can be increased a lot. Now the main drawback in the above design is both the L2 and Main memory have become off-chip, by which the latency to access them has increased a lot. The on-chip transactions are faster than the off chip transactions. So researchers have come up with keeping the L2 cache on-chip by dividing into small banks of memory and keeping each bank with in a tile where there exists processor and L1. So now each tile in a chip contains a micro-processor, L1 cache and L2 cache bank. But this design is way ahead, i.e. in between some other designs have come up which are discussed now and again this design will be taken up. This design is the base for this report's work and is shown below.

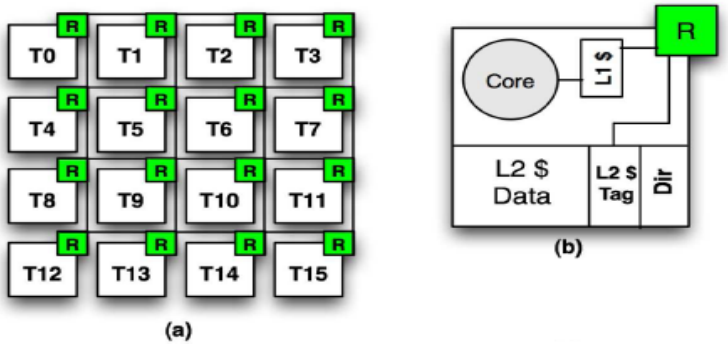

Fig. 1.2 (a) 16-Core tiled CMP model (b) The micro architecture of a single tile [HCM09]

The earlier designs that have been proposed for multiprocessor chips are based on their interconnection system i.e. Bus based architecture and Network-on-Chip architecture. Based on the interconnection model the cache coherence protocols have been developed,

those are Snooping protocol for Bus based, and Directory based protocol for NoC architecture. Fig: 1.2 represents Directory based protocol for NoC architecture and Fig: 1.3 represents Snooping protocol for Bus based architecture. The concept of cache coherence is not discussed in detail in this report but still a brief introduction and the basic protocols are explained.

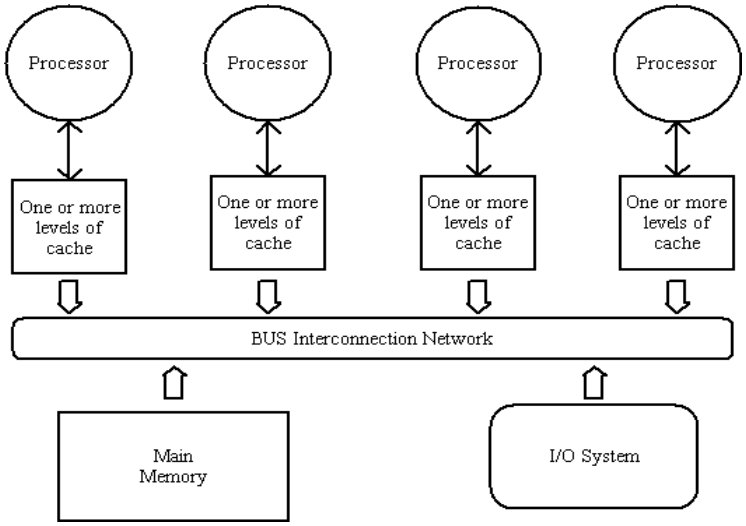

Fig. 1.3 Bus Interconnection

In a multiprocessor system if a shared block of memory is cached into various caches of the processors and if a particular processor modifies the data then this transaction should be visible to the readers of the block. A memory system is coherent if any read of a data item returns the most recently written value of that data item. This can be achieved through software as well as hardware methods. Hardware coherence is implemented through Snooping and Directory protocols. In snooping the cache controller snoops the interconnect medium i.e. the bus and constantly checks for sharing, or invalidate or write signals and by which it updates its state for a particular block. This scaled well for small number of processors but as the processors it does no scale well for the bandwidth and other requirements. The Directory based protocol is where the state of a memory block is stored in a central location and the list of the caches that has this is maintained there itself and

not in the individual caches.The directory maintains the coherence between the caches by sending messages over the interconnection medium. This protocol scales well as the processors number increase. This report focuses on the NoC architecture with Directory based cache coherence protocol. The next part of the report contains sections which discuss the recent designs and then defines the problem and solutions to it, and also the softwares that are used and the architectures that are simulated and compared.

1.2 Organization of The Report

This chapter provides a basic overview of cache concepts and cache-coherence protocols. Chapter 2 gives a brief overview of the cache-coherency protocols available in the literature. In chapter 3, we define the problem statement and propose a 3-Chance Clustering Caching Algorithm. Chapter 4 details the architecture of the simulator used and elaborates the evaluations and results. Chapter 5 concludes the thesis and throws some insight on where the future works might lead to.

1.3 List of Contributions

The followings are the contributions of the algorithm proposed

- Forming of Clusters
- Storing of L1 evicted cache blocks in Cluster Home L2
- Rotation of L2 evicted cache blocks
- 2-Level Directory Protocol for Cache Coherence

Chapter 2

Review of Prior Works

2.1 Literature Overview

2.1.1 Basic Cache Concepts

Let us take the architecture where the processors are connected by Network-on-Chip and each tile has a L1 cache. Splitting the L2 cache into banks and placing the L2 banks in the tiles is next design that has been proposed and yet the question is whether these banks should be private or shared. The meaning of private and shared is that when an L2 bank is shared by on-chip processors then it is shared where as if each L2 bank is dedicated to only its own processor then it is private scheme. Both the schemes have their own advantages and disadvantages. There is another scheme where the degree of sharing varies from shared to private i.e. in 16 core chip the sharing degree can be (1, 2, 4, 8 and 16) where 1 is private and 16 is shared. [HKS+07] proposes these schemes and also gives a simulated results on these designs where the result vary depending on the application running. Sharing degree 2 means that every two CMP cores share their L2 cache banks.

$$AMAT = (1 - MissRate_{L1}) \times HitTime_{L1} + MissRate_{L1} \times MissPenalty_{L1} \qquad (2.1)$$

$$MissPenalty_{L1} = AAL_{L2} + MissRate_{L2} \times MissPenalty_{L2} \tag{2.2}$$

The performance of any design depends on Average Memory Access Time (AMAT). In the equations AMAT basically depends on two factors. The L2 average access latency and L2 miss rate. That is how well the design can balance these two factors, if the average access latency is decreased by decreasing the sharing degree and making the cache private it necessarily need not decrease the AMAT because it may increase the miss rate as the cache capacity will decrease by decreasing the sharing degree. So in private L2 cache design the L2 bank access latency is very less there by data transfer can be done quickly but this effects the other factor the L2 miss rate as the L2 bank is small. It is the opposite in the shared scheme as the cache capacity is increased thereby decreasing the L2 miss rate but it effects average access latency as the L2 bank where the data resides may be far away from the processor. Various designs have been proposed that try to balance these factors and achieve improvement in the AMAT.

2.1.2 Group Caching

[ZFQ+09], says that the AMAT can be reduced by keeping the memory block closer to the processor by loading the block in the L2 bank which is closer to the processor, and if two or more processors require the same memory block then instead of having a single copy multiple copies of same data should be loaded into the L2 banks closer to the destined processors. But increasing the number of copies should be controlled as it can lead to decrease in the overall L2 capacity. So they try to achieve this by forming cache groups as shown in Fig: 2.1. The 16 tiles are divided into four groups where each group contains four tiles with four L2 banks. The L2 banks within the group are in shared scheme and the groups overall are in private scheme. The coherence in between the groups is maintained through query messages and invalidation messages. Query messages are used to fetch data whereas invalidation messages are used to mark data blocks as invalid.

If a data miss occurs in a L1 cache of a Tile then a read request is sent to corresponding L2 of the group where that L1 cache belongs, and if the L2 cache has the data block it is sent to the requested L1 else query messages are multi-casted to other groups to check whether they have any copy of the data block. If some cache group has it then there is no need of off-chip memory access or Main memory access. The other cache groups who have the data block will reply to the query message, and the cache group which is closer to the requester sends data. While writing data each cache group's distributed directory has a tag that tells who all have the data block, by which the data invalidation messages are sent and this can also be used while block replacement.

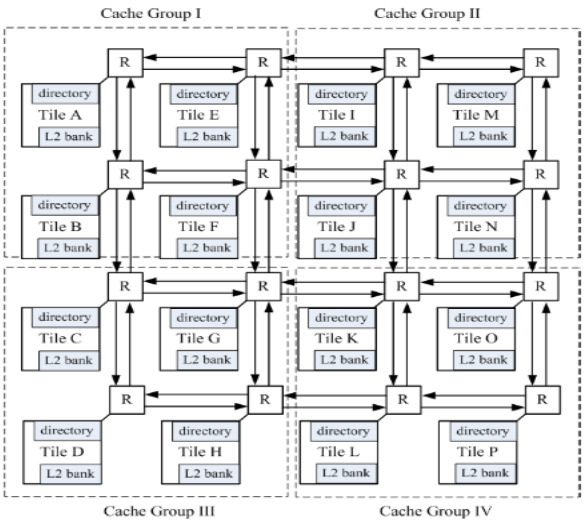

Fig. 2.1 Applying group caching [ZFQ+09]

The tag bits are used to send invalidation messages as well as while block replacement selecting those blocks who have a copy in other cache groups than removing a unique block with out any copies Fig: 2.2. Now coming to the factors of access latency and miss rate how [ZFQ+09] tried to balance these, by making small groups of four it reduced the average access latency as the L2 banks are near by and the miss rate by controlling the number of duplicates that is not removing a unique block in block replacement and going for a duplicate one. Group-caching increased the performance by 2%-8% compared to private

and shared schemes, and network energy consumption reduced by 11%-13%.

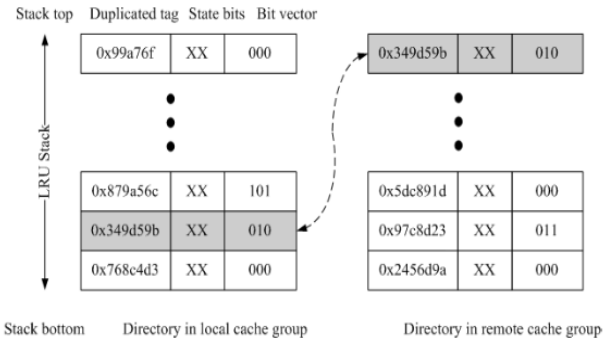

Fig. 2.2 Distributed Directory tag bits [ZFQ+09]

2.1.3 Dynamic Cache Clustering

In a 16-core chip where there are threads running on each processor the data requirement needed by it or the amount of cache space required by it is unknown. It changes with time and with respect to the program that is executing. Some applications require less L2 space where as some require more space. [HCM09] says that by fixing the sharing degree of the L2 banks that is selecting one degree out of (1, 2, 4, 8, 16) is giving fixed amount of space to the processors where those processors might not actually need that and require more than that. That is if the sharing degree is 1 that is private scheme and if a thread is running on a processor which initially requires less data requirement and further as the time goes on if its data requirement increases, initially the L2 bank of the processor serves it requests well but as the time goes the number hits reduces as the cache capacity is less and the over all performance reduces. If in the case where the sharing degree is 16 that is shared scheme, for a processor where the thread running on it requires less data requirement but the data blocks map to the L2 banks that are far away even though the data blocks are on-chip the access latency increases due to remote L2 bank accesses. So, the optimal way of handling this problem is not to fix the sharing degree of a system, but to dynamically change the number of L2 banks that can be accessed by a processor depending on the processor's data

requirement.

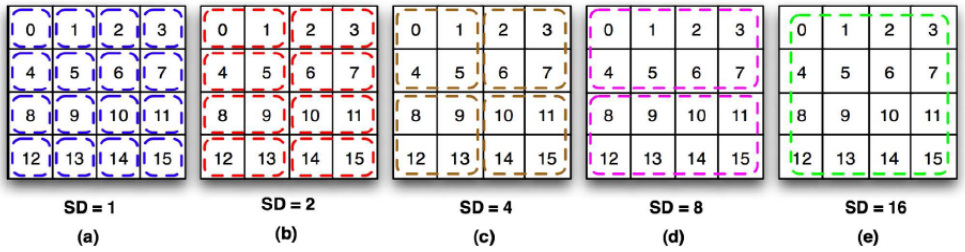

Fig. 2.3 Fixed Sharing Schemes with different sharing degrees (SD). (a) SD1 (b) SD2 (c) SD4 (d) SD8 (e) SD16 [HCM09]

According [HCM09] each core or processor can access to a set of L2 banks and the number varies between (1, 2, 4, 8, 16) in 16-core system. The L2 banks that can be accessed by it form a cache cluster to that particular core. The number of L2 banks in a cache cluster of a core i is called *Cache Cluster Dimension of core* i (CDi). By changing the cache dimension of a processor by depending on its AMAT dynamically the usage of the L2 banks can be optimized. So, when the threads are running each processor has its own cache dimension by which the L2 banks are utilized to its maximum. At periodic intervals each processor's AMAT is calculated and if the value is more than the previous one then the present cache dimension has increased the memory access time for the processor and currently is not suitable and therefore the cache dimension of the processor should be changed accordingly.

As the cache dimension of the processor varies the L2 bank where a cache block has to reside for a particular processor changes with the cache dimension. Therefor a mapping function required which maps to the appropriate L2 bank depending on the processor and its cache dimension. The default L2 bank that the cache block should reside when the sharing degree is 16 is called *Static Home Tile* SHT and the L2 bank that is assigned due to the cache dimension and the processor position is called *Dynamic Home tile* DHT. If a processor i requests a cache block B, and if CDi is smaller than 16, B is mapped to DHT which is different than the SHT. The mapping function can be chosen of our choice.

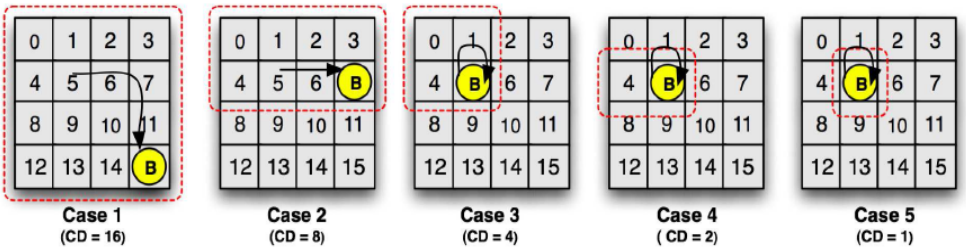

Fig. 2.4 An example of how the mapping strategy works. Each case shows the dynamically selected L2 bank of the requested cache block B upon varying the cache cluster dimension (CD) of processor 5. [HCM09]

[HCM09] suggested a runtime monitoring algorithm that changes the cache dimensions of the processors by collecting the AMAT's of each processor. The algorithm starts like, a process P starts running on core i with initial cache cluster CDi = 16. After a period of time T, the AMAT*(i, prev)* of the processor i is recorded and now the cluster size is reduced to half and again after time T new AMAT*(i, current)* is recorded. Now if the

$$AMAT(i,\ current) < AMAT(i,\ prev) \tag{2.3}$$

then shrink in the size of the cluster has increased the performance of the processor and if

$$AMAT(i,\ current) > AMAT(i,\ prev) \tag{2.4}$$

then the decrease in cluster size did not benefit the processor and the cache cluster size is again doubled. This strategy is applied for every processor and is repeated after every time period T there by always trying to optimize the cache dimension. Now coming to the factors of average access latency and miss rate how [HCM09] tries to balance these factors, it directly takes both the factors into account by calculating them dynamically and changing the cache cluster size thereby reducing the latency as well as increasing the cache capacity.

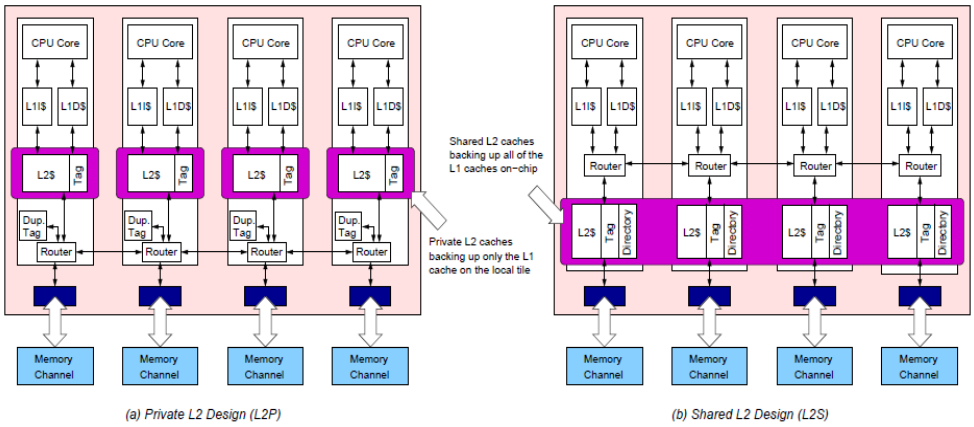

Fig. 2.5 The two baseline L2 designs. The private L2 design treats each L2 bank as a private cache. The shared L2 design treats all L2 banks as part of a global shared cache. [ZA05]

2.1.4 Victim Replication

In [ZA05] it classifies the designs into two categories as discussed before private and shared L2 design schemes and is shown in Fig: 2.5. It introduces a simple and hybrid design called the *Victim replication*. The basic idea is that assume a shared L2 scheme when a L1 miss occurs in a processor a request is sent to the *Home L2 Bank* where the data block has to be present and the request is served and the block is cached into the L1 cache of the requested processor. Now when that block is selected for removal not because of invalidation by writes but because of block replacement due to conflict or capacity miss then that *victim block* instead of completely removing that block from the tile place that in its own L2 Bank and tagging it as a *replica*. So now the the L2 banks act as overall shared but also private. But when it come to evicting the blocks in L2 banks preference is given to global data blocks than replicas as there will be a copy of the replica in the Home L2 Bank. Now the data blocks in a L2 bank can be classified into three categories invalid, global and replica data blocks. Now when a L1 data block is evicted then if there is an invalid or global data block with no sharers or any replica created before, then that block is removed and a new replica is created for that L1 block, else no replica is made.

Now coming to the factors that effect AMAT the access latency and miss rate and how

[ZA05] balances these factors, it creates replicas in its own L2 bank there by reducing the access latency and by giving preference to the global data blocks it removes the replicas so that the cache capacity is balanced to reduce the miss rate.

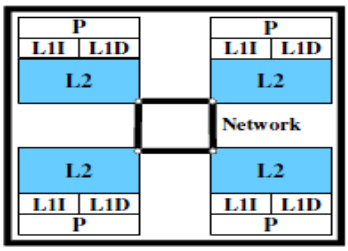

Fig. 2.6 Cooperative Caching (The shaded area represents the aggregate shared cache formed via cooperation) [CS06]

2.1.5 Co-operative Caching

In [CS06] it is similar to [ZA05] but differ in implementation. In [ZA05] if a L1 cache evicts a block, it is cached in its L2 bank where as in [CS06] if a L2 bank evicts a block due to conflict or capacity miss then it is not completely removed but it is spilled onto near by L2 bank. Cooperative caching will attempt to use remote L2 caches to hold data that would generally not fit in the local L2 cache, if there is space available in a remote L2 cache. Basically Cooperative caching starts for example a chip with four processors with their L1's and L2 banks in their tiles. The L2 Banks are private to start with but cooperate among them to create a overall shared scheme on the chip. *Singlets* are termed to those blocks where only one single of the data block exists in the chip. Now as the L2 banks are private they contain replicas, so there should be mechanism to control the number of replicas and protecting the singlets by spilling them onto other caches. Cooperative caching ensures three policies that are

- facilitates cache to cache transfer of on-chip clean blocks to remove unnecessary off-chip accesses to data hat are there in other L2 Banks

- replaces replicas to provide space for singlets
- to cache evicted singlets from local L2 cache Bank on to remote L2 Banks where there is space in the remote banks or the data block is unused there

These three policies are ensured by Cache Coherence Engine which sits in the middle of the four tiles and controls the data movement so as to ensure the above three policies. Now coming to how Cooperative caching try to balance between the factors, by making the caches private it reduced the L2 access latency and by cooperative sharing between the L2 banks it increased the cache capacity there by balancing the miss rate and by controlling the cooperation between the L2 banks it optimizes between private and shared schemes and shifting between them.

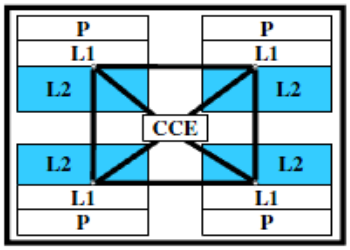

Fig. 2.7 Cache Coherence Engine [CS06]

Till now the the ideas that were discussed are related to L2 design, the next two ideas are related to design of the cache coherence protocol that is Directory based, how to implement the Directory based protocol in the chip and how can the implementation actually plays a role in the L2 access latency will be discussed.

2.1.6 Directory Based Cache Coherence

In [DC90] Directory based cache coherence protocols a directory is maintained which contains the information of the caches or the banks that contain the data blocks and also their permissions whether in shared mode or exclusive mode and whether the data is dirty and to written back to the memory or not. This information is stored in the directory

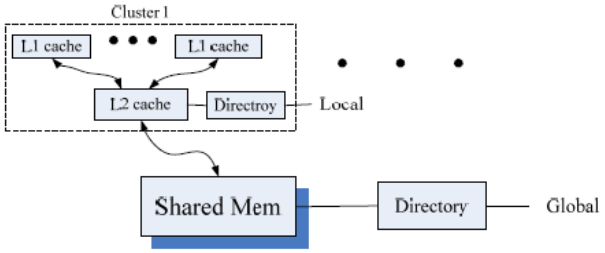

Fig. 2.8 Memory Hierarchy [YZ06]

in many ways. Mainly full map directory, limited directory, and chained directory. In full map directory each processor is assigned a bit for every data block and also a dirty bit. The processor bits tell which all processor's L1 cache contain the block and the dirty bit tells whether the block is dirty or not. The limited directory instead of so many to processors in the full map it maintains only the processor's id which contain the data block and ensures that only one processor can contain the data block by which restricting the sharing, and chained directory removes this limitation but increases the access latency. This report's problem which is yet to define focusses on full map directory and the next two ideas which are to be discussed, one modifies the full map directory where as the other implements it differently in terms of hardware.

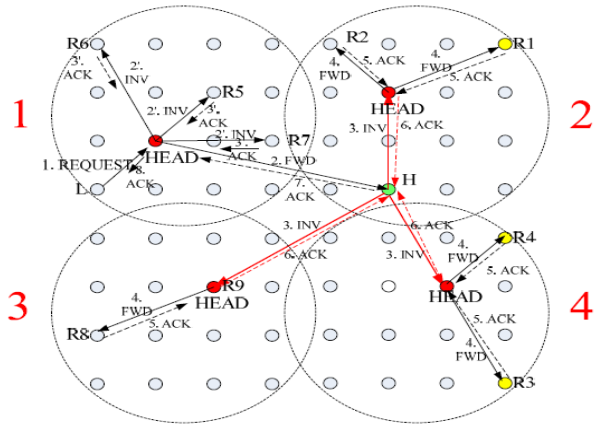

Fig. 2.9 Clustered perspective [YZ06]

2.1.7 Hierarchical Cluster Based Cache Coherence

In [YZ06] it identifies the draw backs of the full map directory, that is memory overhead for all the bits and tries to solve it by dividing the directory into two levels of directory. All the processors are divided into clusters of processors where each cluster has a L2 cache bank and directory which maintains information regarding that banks processors, and a global directory which maintains information about the cache clusters sits near the Shared Main Memory. The directory inside the cluster is called the local directory and the directory which maintains the information about clusters is the global directory. Each cluster has a *Head Node* which is selected among the processors which is close to all the other processors in the cluster. In the Head node the L2 Bank and the Local Directory reside. The cache coherence is divided into two levels. One is intra-cluster and other is inter-cluster. For read or a write miss first a request is sent to the local directory which if a copy exists in the cluster the information is sent else a request is sent to the global directory where if a write a miss invalidation messages are set to the other cluster head nodes else for a read miss the data is sent if no other cluster has the data else the information is sent regarding which cluster has the data. Now coming to the factors, the memory latency by dividing the directory and maintaining the local directory, and the miss rate by L2 Bank for an entire cluster by which the capacity is increased instead of small banks.

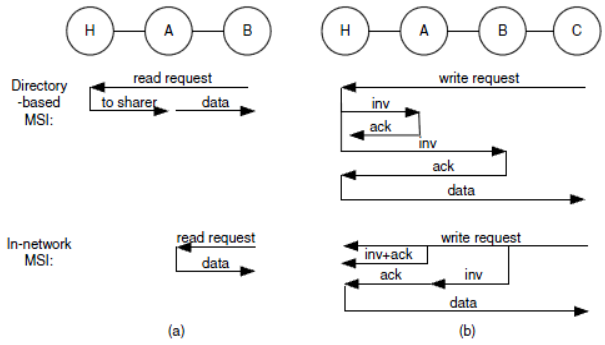

Fig. 2.10 In-network optimization scenarios [NE06]

2.1.8 In-network Cache Coherence

In [NE06] the idea of having directory inside the tile is basically removed. In all the designs discussed till now, when a read miss occurs in a L1 then a message is sent to the Home tile where that block may reside, as that is where the directory of it also exists. The directory tells where the block is in, in main memory or in some other L1 or L2 and sends the information to us and then a request is again sent to the remote node to send the data block. Its a hand shake between directory and then with the remote node, but if the remote node is in the path which is traversed by the query message sent to the directory then if there is a method such that the message can be replied directly instead of going to the directory. This lead to the idea proposed in [NE06] where the directories are moved to the network routers instead of keeping inside the tiles. The directories are implemented or kept in a cache which resides in the network routers. Now when a read-miss message travels to the Home node along which encounters a sharer then as the directory is in the router while passing the message to the next router it is checked with the directory cache in its router and if a hit occurs the cache that is in the particular tile replies to the requester instead of the Home node and the data block is send. The only overhead is the caches in the routers and checking the query messages at every router, but on the other side the latency is reduced drastically i.e. an entire handshake is removed.

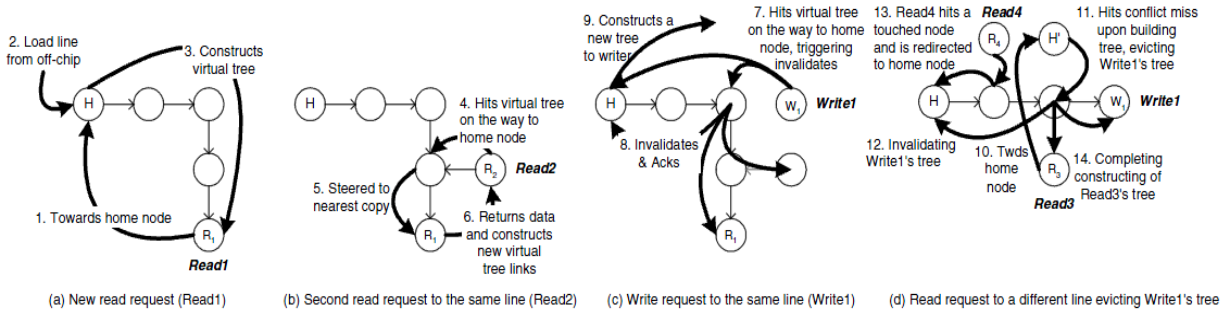

Fig. 2.11 Various scenarios of reads and writes in in-network implementation [NE06]

When a sharer replies to a query message it sends a the data block to the requestor and

along which a tree is constructed and every directory cache along the path makes an entry of the data block and points to the root node that is the node which has started the tree. The tree slowly enlarges as the number of sharers increase but if write miss occurs then the tree is broken down and exclusive permissions are given to the requestor and after which a new tree is constructed with the requestor as the root node. But here if a tree is constructed but there is no space in the directory cache then an existing tree is evicted to make space fr the new tree. Here when a tree is evicted the entire tree is evicted without any disjoints. Now if two trees simultaneously construct and evict themselves then a deadlock occurs by which both of them should halt for a random amount of time and start again. Even though it faces some problems the improvement in latency issue is very huge that is an average of $\sim$45% improvement in average memory access latency.

2.2 Summary

In this chapter a detailed analysis of how the cache-coherence protocols developed since its inception is provided, and their performances compared. In the next chapter, the problems in the existing protocols are defined and analysed and a solution is proposed in form of a 3-Chance Clustered Caching Algorithm.

Chapter 3

Problem Definition and Proposed Solution

3.1 Problem Definition

Cache management is one of the key factors that affect the performance of present day Chip Multi-Processors. The main aspects that govern the cache management are the access latency and the cache space utilization. The problem can be stated as the reduction of Average Memory Access latency (AMAT) so that processors need not waste time for the data. AMAT as we have seen in Eq: 2.1 and Eq: 2.2 depends on L2 access latency and L2 miss rate. The cache management policy concentrates on increasing the life time of a cache block by giving up to 3 chances by rotation of data among the L2 caches and clustering keeps the data close to the processors thereby decreasing the access latency. The caches act as non-inclusive for increasing the cache space, which increases the access latency but is reduced by 2 level directory protocol implemented for cache coherence and cache clustering. The evicted L1 cache bocks are stored in the Cluster home L2 bank and the evicted L2 cache blocks follow the 3-Chance rotation algorithm.

3.2 3-Chance Clustering Caching

3.2.1 Non Inclusive Formation of Clusters

Before going into the Cluster formation the architecture of the CMP is explained. Fig: 3.1 shows the architecture of the [TPS+04] 2D-MESH 16-tile CMP model. Tiled CMP architectures can scale well as the number of processors increase [ZA07]. The architecture employs a 2D mesh switched network, and each tile is connected to the 2D mesh network via the router attached to each tile. Fig: 3.1(b) shows the inside of a tile which contains a processor, a dedicated L1 cache, a L2 bank and a Directory which serves its purpose for coherence between the caches and for off chip memory accesses. The cache controllers of L1, L2 and Directory communicate via the NoC fabric through messages for data transfer as well as for coherence requests. The MESH network employs XY routing algorithm [MWM04].

The 16 tiles are divided into four clusters and each cluster containing four cores. In a cluster the L2 act as non inclusive caches, where either one L2 has a data or any L1 has the data. The advantage of this policy is increase in the effective cache capacity of a cluster than [ZFQ+09] but comes with a disadvantage of access latency for example some L1 in the cluster has already cached the block and if an L1 requests for the same block none of the L2 banks will have the cache block because of the non inclusiveness. If the information regarding the L1 data is kept within the cluster at the directory then the directory will forward the request to the nearest L1 having the cache block thereby effectively reducing the overhead latency due to the non inclusiveness. As L1 hit time is much less than the L2 it further reduces the access latency caused by forwarding. The aim of the 3-Chance Clustered Caching is to increase the life time of a cache block as explained below.

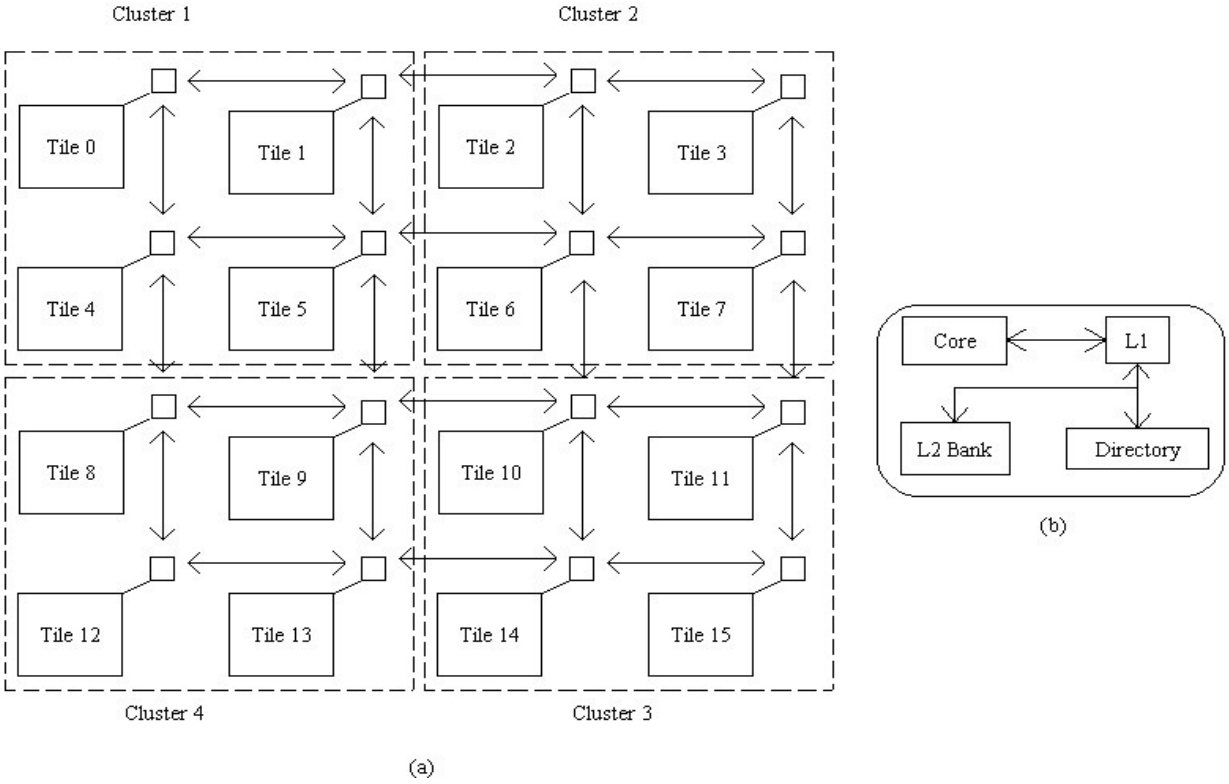

Fig. 3.1 (a) Clusters formed in a 16-core Tiled CMP (b) Architecture inside a tile

3.2.2 L1 evicted Cache Blocks

If an L1 evicts a cache block and if it is the only copy left in the cluster instead of completely removing it from the cluster it is stored in the Cluster Home L2. This way the life time of a unique cache block is increased. The addresses are mapped across the L2 banks and the Cluster Home L2 is the L2 bank that maps to that address. Here the stored cache block in the Cluster Home L2 can be reused again by any core in the cluster i.e. the evicted L1 block is localized in the cluster and is not private to any core which is in the case of [ZA05].

This way the number of shared copies in the cluster will always remain to the minimum thereby increasing the cache capacity yet attending to the access latency by the 2-Level Directory.

3.2.3 3-Chance Rotation Algorithm for the L2 evicted Cache Blocks

Every cache block in the L2 bank in the cluster is unique data block and is not present in any other cache in the cluster. So when a L2 eviction occurs the life time of the block can still be increased by forwarding the evicted cache block to neighbor L2 bank in the cluster in some order. We move the block in clockwise manner. This block can also be evicted from neighbouring L2 in future and thus rotated again until the cache block is given 3 chances, beyond that the cache block is evicted from the cluster. The 3-Chance rotation algorithm is variant of [MP94] where the value of N in N-chance algorithm is set to 1. After the last chance the data block is been sent to the Cluster Home Directory where it is written to the main memory if needed.

3.2.4 2-Level Directory Protocol

Every directory acts as a Cluster Home Directory (CHD) and Global Home Directory (GHD). The address space is mapped across the four directories in a cluster acting them as Cluster Home Directory to maintain coherence and for forwarding of data requests inside a cluster. Also the address space is mapped across the 16 directories on the chip which makes each directory Global Directory which maintains coherence across the clusters. Every entry in Directory can be a Cluster, Global or both thereby every entry in the directory has to accommodate for both the information. Fig 3.2 shows an entry in a directory.

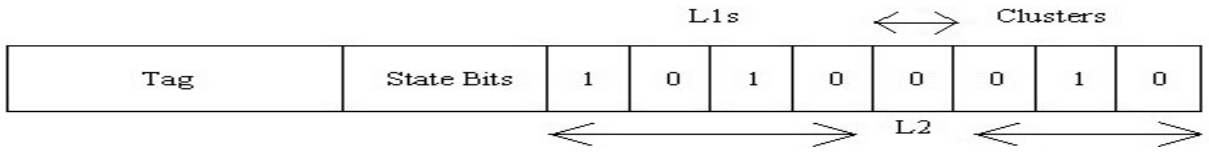

Fig. 3.2 An entry in the directory containing both cluster and global information of a data block.

The four L1 bits tells the information of L1s having the data in the cluster and L2 bit tells that the data is in some L2 bank in the cluster and the remaining three cluster bits is for Global information telling which other clusters have data.

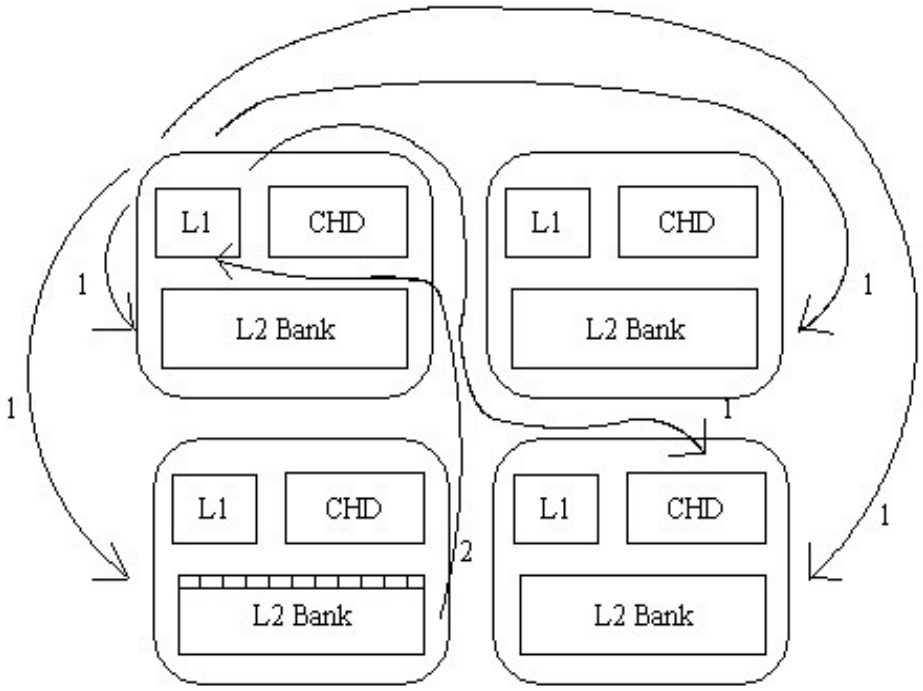

Fig. 3.3 Query request messages 1 are sent to the 4 L2 banks and CHD by requestor L1 for a particular address B which maps to that CHD. Response message 2 to the requestor L1 from a L2 bank having the data.

When a L1 read miss occurs 5 query requests are sent, 4 to all the L2 banks in the cluster and one to the cluster home directory. If any L2 bank has data it sends the data and invalidates itself for non inclusiveness and if no L2 bank has the data it either means that any L1 has the data or there is no data in the cluster Fig 3.3.

The Cluster Home Directory comes to know whether any L2 bank has the data or not. It forwards the request to the closest L1 cache near the requestor having the data if the cache block is present in any L1s of the cluster Fig 3.4.

If there is no data in the cluster the CHD forwards the request to the GHD if some other cluster has already brought the data. In that case the GHD searches the cluster bits and forwards the request to the closest CHD that contains the data and thereby the data is sent to the requestor. If there is no data present in any of the cluster the GHD makes an off-chip memory access and sends the data to the requestor updating the cluster bits Fig 3.5.

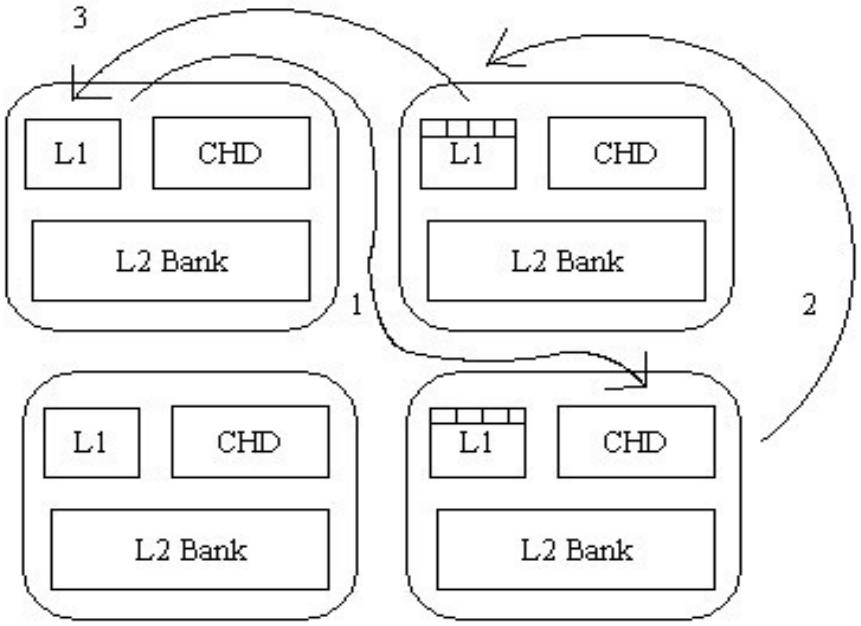

Fig. 3.4 If L2s dont have the data and some L1 in the cluster has, data request message 1 sent to the CHD is forwarded to the nearest sharer 2, and data response message 3 is sent by the L1 having the data.

If an eviction comes in the Directory entry preference is given to the Global entries than the local cluster ones. Due to 3-Chance rotation extra query request messages are required for other L2 bank as the data block can be present in any one of the L2 bank. This increases the network traffic which is an overhead but it increases the performance by increasing the life time of a cache block. The experimental setup and the results of the 3-Chance Clustered Caching are shown in the next section.

3.3 Summary

This chapter detailed about the problems in the existing cache-coherence protocols and a new algorithm was proposed addressing these problems. In the next chapter the improvement by the algorithm is verified quantitatively by simulating some benchmarks which reflect the real life problems.

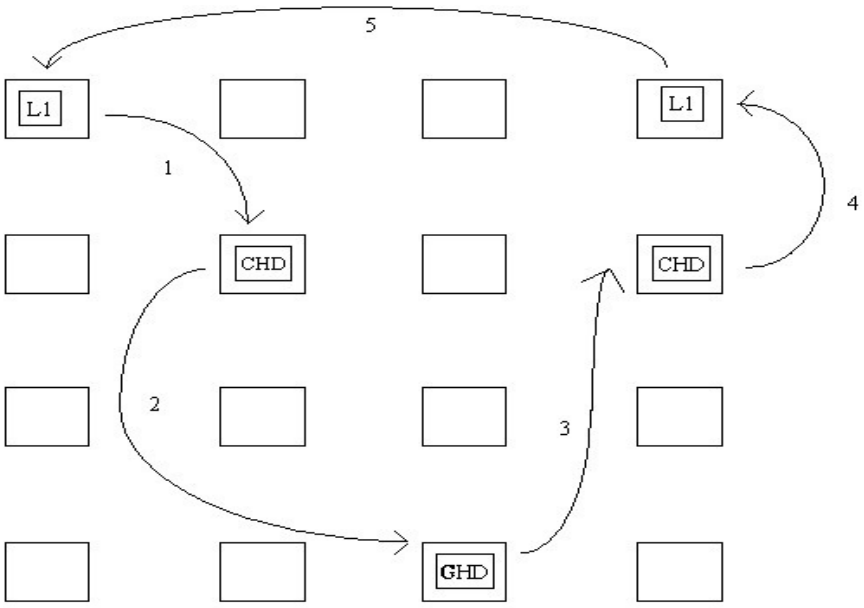

Fig. 3.5 If there is no data in the cluster the data request message 1 is forwarded to the GHD 2, and if some cluster has the data block the GHD forwards the request message 3 to the CHD of the cluster having the data which forwards 4 to the member of the cluster havind data and finally data response message 5 is sent to the requestor.

Chapter 4

Simulation and Results

4.1 Simulation Architecture

Full-system simulation is done to evaluate the performance of the new algorithm introduced in the previous chapter. The goal is to check how the new ideas effect the various characteristics like performance, latency, power consumption, and network load of an NoC based chip multiprocessor. Additionally, as the number of cores increase the interrelation between ideas and various system parameters are to be checked. Full-system simulation allows evaluation of system performance by running scientific/commercial workloads on top of actual operating systems. Simics full Functional simulator [MCE+02] from Virtutech (recently acquired by Wind River, a wholly owned subsidiary of Intel Corporation) is used. On top of that General Executiondriven Multiprocessor Simulator (GEMS) from Wisconsin Multifacet Project [MSB+05] is used for recording and comparing the memory transactions. Network interconnection is modeled using Garnet [APN09] which is a detailed interconnection network model inside GEMS. The power consumption is modeled using Orion [HSXLSM02] which is a power performance simulator for interconnection networks. A 16 core CMP is created virtually, on which unmodified Solaris 10 operating system is installed on the simulated machine and benchmarks from Splash2 [SME+95] benchmark suite are executed in order to evaluate the new architectures. The two architectures–Non-clustered

with shared L2 banks and Clustered with 3 chance clustering protocol – are simulated and the bench marks are run on them and the two architectures are compared with respect to the above parameters.

- Simics is installed along with the GEMS simulator on the server
- A 16-core setup is created on which an operating system Solaris 10 is installed
- MESI protocol is selected compiled and loaded on to the setup
- The Splash benchmark codes are compiled on the setup and the binary files are created
- The GEMS parameters are modified according to the target architecture and recompiled
- The Garnet and Orion modules are installed and are loaded for interconnection network and power characteristics
- Finally the Water-Spatial Benchmark is run on the setup and the results are compared

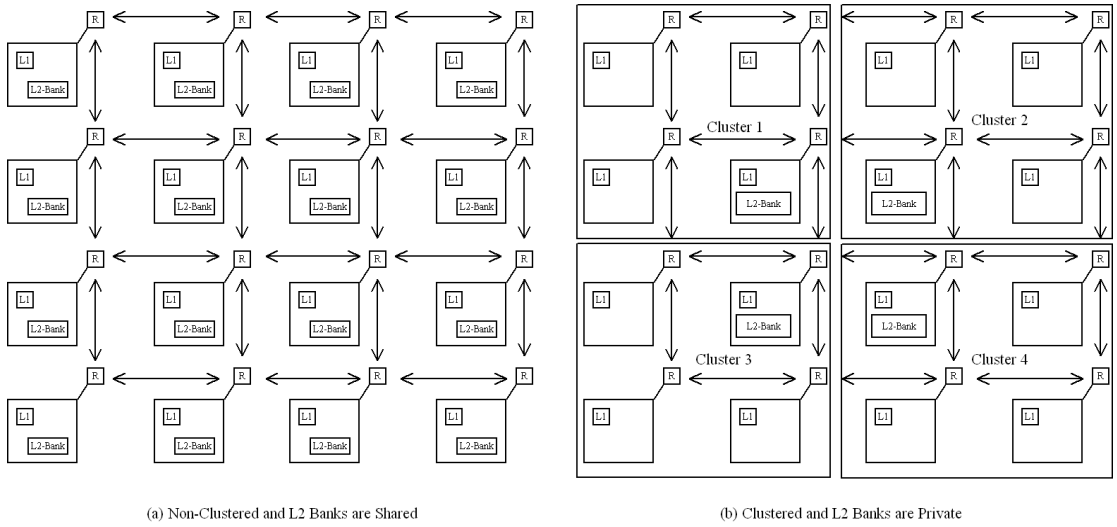

Fig. 4.1 Two architectures that are simulated in the simulation set up

4.1.1 Simics

Simics is a full system simulator. It provides virtual hardware that runs the same binaries as the physical hardware i.e. it is able run unmodified operating system and other softwares.

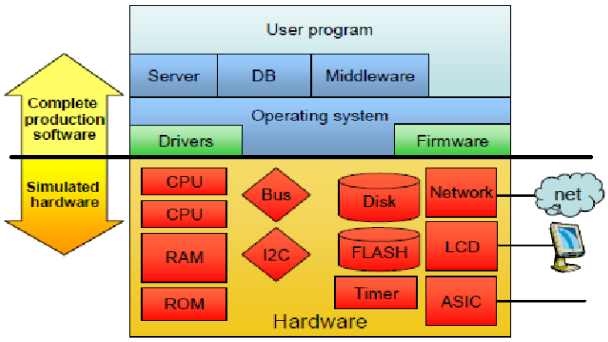

Fig. 4.2 Simics simulates all hardware components of a system, allowing the unmodified software to run just like on the physical hardware [MCE+02]

4.1.2 Gems

GEMS is a tool to characterize and evaluate the performance of multiprocessor hardware systems. It uses the existing full-system functional simulation infrastructure of Simics as the basis around which to build a set of timing simulator modules for modeling the timing of memory systems and processor cores. GEMS mainly consists of two primary modules Ruby and Opal. Ruby is a timing simulator of a multiprocessor memory system. It models caches, cache controllers, system interconnect, memory controllers and banks of main memory. GEMS uses a domain-specific language called Specification Language for Implementing Cache Coherence(SLICC) to specify cache coherence protocols. Opal models the timing of an out-of-order SPARC processor.

4.1.3 Garnet

Garnet is an interconnection network model inside a full-system simulation framework. Garnet's fixed pipeline model was used in the simulation. Fixed pipeline model is intended

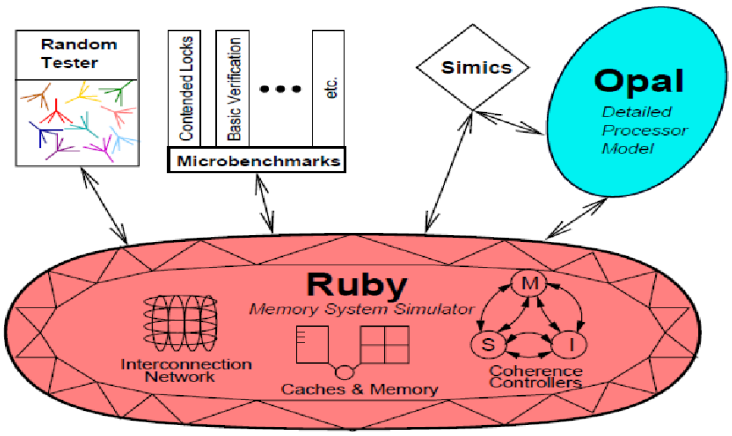

Fig. 4.3 A view of the GEMS architecture: Ruby, our memory simulator can be driven by one of four memory system request generators [MSB+05]

for low-level interconnection network evaluations and models the detailed features of a state-of-the-art network. Flit is the smallest item transferred on on-chip networks. The size of it also indicates bandwidth of links. Buffer size indicates the number of buffers inside routers. Garnet models the pipeline router which consists of five stages - RC (route computing), VA (virtual channel allocation), SA (switch allocation), ST (switch transfer) and LT (link transfer). Virtual network is a group of VCs (virtual channels) which are used to transfer the same type of coherence messages.

4.1.4 Orion

Orion is a power-performance interconnection network simulator that is capable of providing detailed power characteristics. It works with the Garnet "fixed pipeline" model. Various performance counters that are required for power calculations are recorded during the simulation. In the end, they are fed to Orion for reporting the interconnection network power.

Component	Parameters
Processor	UtraSPARCIII+
L1 I/D cache	64KB, 4-way, 3 cycles
L2 cache bank (non-cluster)	1MB, 4-way, 12 cycles
L2 caches (non-clustered)	16
L2 cache size (clustered)	4MB, 4-way, 12 cycles
L2 caches (clustered)	4
Memory bank	1GB, 4KB/page, 158 cycles

Table 4.1 System Parameters

4.2 Experimental Setup

Two architectures were designed and simulated in the simulation set up. One architecture is a shared scheme with 16 cores each with a processor, L1 split cache and L2 unified cache Bank, the other architecture has 16 cores each with a processor and L1 split cache. The processors are divided into four clusters with four processors in each cluster and a dedicated L2 bank for each cluster which are private and don't share data. Both architectures have a Directory cache and a Directory and a Main memory Fig: 4.1 and the setup parameters are listed in Table: 4.1. Unmodified Solaris 10 operating system was loaded in the simulated system and one program from Splash2 parallel benchmark suite was run till completion to measure the performance. The benchmark application used was: Water-Spatial. MESI cache coherence protocol is implemented and a comparison is done between cluster and non clustered architecture which is shown in Figure: 4.5

Garnets fixed pipeline model is used which models the pipeline routers and its parameters are listed in table: 4.2. Shared memory benchmarks from SPLASH2 suite [SME+95] are used in our experiment and their parameters are listed in table: 4.3.

4.3 Clustered Caching

As shown in the Fig: 4.5 the improvement in ruby cycles of the clustered version as compared to the non-clustered version is 15%.

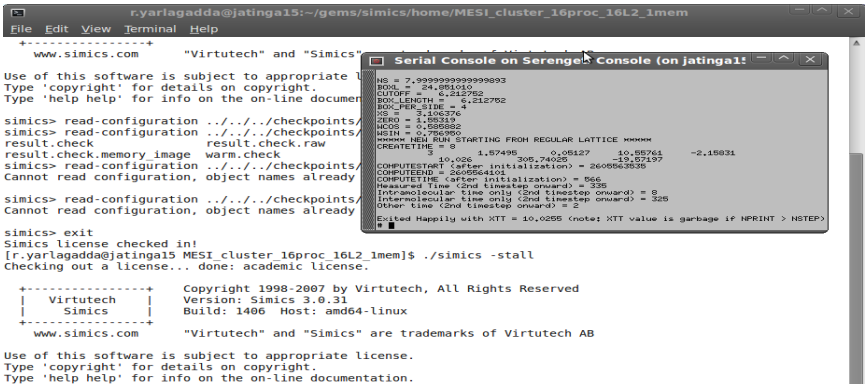

Fig. 4.4 Console of the Target Architecture

Network Configuration	Parameters
Flit Size	16 bytes
Buffer Size	4
Pipeline Stage	5-stage
VCs per virtual network	4
Number of virtual networks	5

Table 4.2 Parameter of Garnet's Fixed Pipeline Model

4.4 3-Chance Clustered Caching

As can be seen from the graph below that shows the performance improvement of the 3-Chance Clustered Caching over Shared L2 bank scheme, the improvement is 9 $\sim$ 15%. The reasons as explained earlier are that the increase in life time of the data block by non inclusive L2 banks and 3-chance rotation and reducing the access latency through clustering thereby keeping the cache block nearer to the requestor.

Benchmarks(transactions)	Environment
Ocean	514 x 514 grid
Barnes	64K particles
Water	512 molecules
Fmm	16384 particles

Table 4.3 Workloads Parameter

	Ruby_Cycles	miss_latency	Average network latency	Total Link Power(W)	Total Router Power(W)	cycles_per_instruction
16 cores						
Cluster_Water	20468650	50.5393	14.2864	0.31648	2.06E+005	0.300567
Non-Cluster	24351369	64.7518	21.5923	0.496668	3.79E+005	0.287218
% Improvement	15.944561474	21.949196779	33.8356729	36.279365693	45.7571575951	-4.64768921

Fig. 4.5 The results generated by GEMS for both the architectures

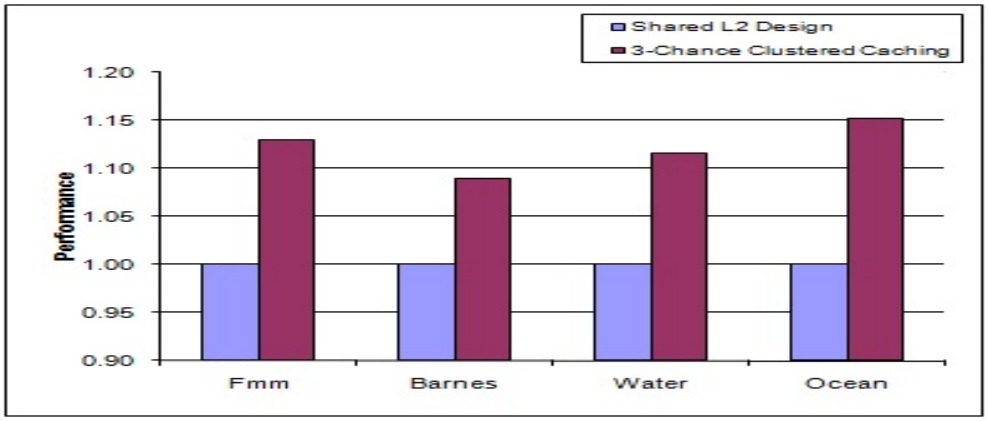

Fig. 4.6 Performance comparisons between Shared L2 Scheme and 3-Chance Clustred Caching for 16-node 2D-MESH CMP

The below graph shows the energy consumption and the 3-Chance Clustered Caching consumes 19 $\sim$ 34% more energy than the Shared L2 bank scheme and this can be explained due to the overhead of extra query messages sent to the L2 banks and forwarding of requests by the CHD.

4.5 Summary

In this chapter, the performance of the proposed algorithms were analysed and improvements of the range 9 $\sim$ 15% were registered when compared to the non-clustered algorithm.

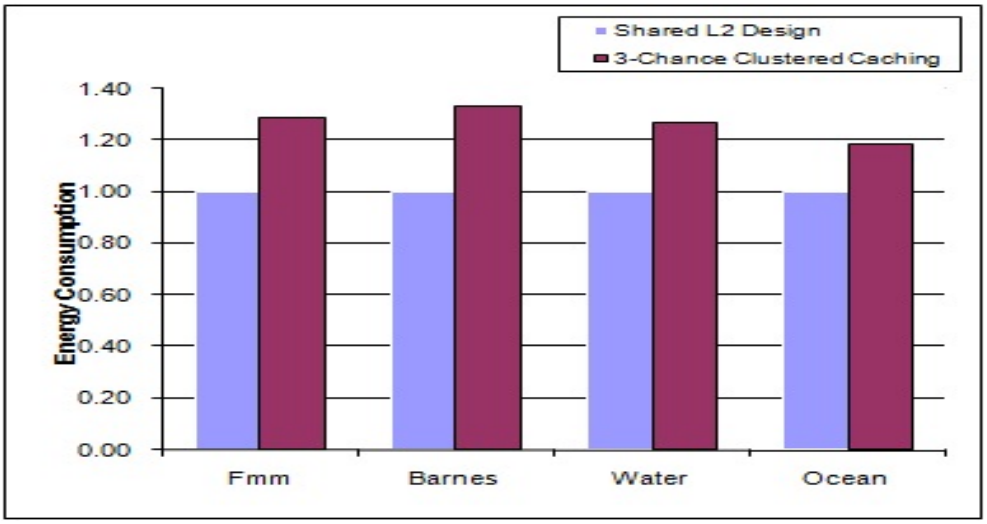

Fig. 4.7 Energy Consumption comparisons between Shared L2 Scheme and 3-Chance Clustred Caching for 16-node 2D-MESH CMP

Chapter 5

Conclusion and Future Work

5.1 Conclusions

For the upcoming CMPs cache management plays a key role in its performance. Cache managements main goals are to reduce the access latency and to increase the effective cache capacity. Increase in the life time of a cached data block reduces costly off-chip memory access and non inclusiveness of L2 banks increases the cache capacity. We propose 3-Chance Clustered caching scheme which aims at the above two and ultimately reducing the access latency and increasing the cache capacity. The main principle of this scheme is to keep the data block within the cluster as long as possible. For this the evicted L1 cache blocks are stored in the cluster home L2 thereby localizing the data and making it accessible to other cores and rotating the L2 evicted data blocks using 3-chance algorithm. For this to happen a directory at cluster level and at global level is required and a 2 level directory is implemented which takes care of the query requests and the coherence issues of the caches. The experiment results show that the proposed scheme shows increase in the performance by $9 \sim 15\%$ compared with Shared L2 bank scheme.

Due to increase in the query messages because of the rotation of data there is an overhead in the energy consumption and the new scheme increases the energy consumption

by 19 $\sim$ 34% Since the overheads of communications inside cache cluster play hinders the performance of the CMP our future work focuses on decreasing the query messages and also supporting n-chance rotation (intra cluster and inter cluster rotation) of L2 evicted data blocks further thereby a clusters unused cache space can be utilized more efficiently.

5.2 Future Works

The future works might focus on implementing the concept of dynamic clustering in the proposed algorithm thus addressing the issue of uneven utilization of L2 caches by different Cache groups. Also the number of query messages can be reduced by choosing an optimum tradeoff between the simplicity of algorithm and the network congestion due to query messages.

References

[APN09] N. Agarwal, L. Peh, and N.Jha. GARNET: A detailed on-chip network model inside a full-system simulator. In *IEEE International Symposium on Performance Analysis of Systems and Software,ISPASS,Proceedings*, pages 33–42, 2009.

[CS06] J. Chang and G. S. Sohi. Cooperative caching for chip multiprocessors. In *Proceedings of the 33rd annual international symposium on Computer Architecture*, pages 264–276, 2006.

[DC90] K Kurihara A Agarwal D Chaiken, C Fields. Directory-based cache coherence in large-scale multiprocessors. *Computer*, 23(6):49–58, June 1990.

[HCM09] M. H. Hammoud, S. Cho, and R. Melhem. Dynamic cache clustering for chip multiprocessors. In *Proceedings of the 23rd Intl. Conference on Supercomputing (ICS)*, pages 56–67, 2009.

[HKS+07] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, and S. W. Keckler. A NUCA substrate for flexible CMP cache sharing. *IEEE Transactions on Parallel and Distributed Systems*, 28(8):1028–1040, 2007.

[HSXLSM02] W. Hang-Sheng, Z. Xinping, P. Li-Shiuan, and S. Malik. Orion: a power-performance simulator for interconnection networks. In *Proceedings of 35th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 294–305, 2002.

[MCE+02] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hållberg, J. Högberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A full system simulation platform. *IEEE Computer*, 35:50–58, 2002.

[MP94] R. Wang T. E. Anderson M. Dahlin and D. A. Patterson. Cooperative caching: Using remote client memory to improve file system performance. 1994.

[MSB+05] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood. Multifacets general execution-driven multiprocessor simulator (GEMS) toolset. *Computer Architecture News (CAN)*, September 2005.

[MWM04] R. Mullins, A. West, and S. Moore. Low-latency virtual-channel routers for on-chip networks. In *Computer Architecture, 2004. Proceedings. 31st Annual International Symposium on*, pages 188–197, 2004.

[NE06] L. Shang N. Eisley, L-S. Peh. In-network cache coherence. In *P39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 321–332, 2006.

[SME+95] S.C.Woo, M.Ohara, E.Torrie, J.P. Singh, and A. Gupta. The splash-2 programs: characterization and methodological considerations. In *Proceedings. 22nd Annual Symposium on Computer Architecture*, pages 24 – 36, 1995.

[TPS+04] M.B. Taylor, J. Psota, A. Saraf, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, A. Agarwal, W. Lee, J. Miller, D. Wentzlaff, I. Bratt, B. Greenwald, H. Hoffmann, P. Johnson, and J. Kim. Evaluation of the raw microprocessor: an exposed-wire-delay architecture for ilp and streams. In *Computer Architecture, 2004. Proceedings. 31st Annual International Symposium on*, pages 2–13, 2004.

[YZ06] A. Jantsch L. Li M. Gao Y. Zhang, Z. Lu. Towards hierarchical cluster based cache coherence for large-scale network-on-chip. In *39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 321–332, 2006.

[ZA05] M. Zhang and K. Asanovic. Victim replication: Maximizing capacity while hiding wire delay in tiled cmps. In *Proceedings.32nd International Symposium on Computer Architecture*, pages 336–345, 2005.

[ZA07] M. Zhang and K. Asanovic. Victim migration: Dynamically adapting between private and shared cmp caches. 2007.

[ZFQ+09] W. Zuo, S. Feng, Z. Qi, J. Weixing, L. Jiaxin, D. Ning, X. Licheng, T. Yuan, and Q. Baojun. Group-caching for noc based multicore cache coherent systems. In *Proceedings of the conference on Design, automation and test in Europe (DATE)*, pages 755–760, 2009.