A Scalable Solution for Cache Coherence in Many-core Systems using Share-pattern Aware Cache Segmentation and Hybrid Network-on-Chip

by

Antara Ganguly
MT14057

June, 2016

Indraprastha Institute of Information Technology
New Delhi
Under the Supervision of
Dr.Sujay Deb
Advanced Multicore Systems Lab, ECE Department, IIIT Delhi

Submitted in partial fulfillment of the requirements
for the Degree of M.Tech. in Electronics and Communication Engineering,
with specialization in VLSI and Embedded Systems

Certificate

This is to certify that the thesis titled "**A Scalable Solution for Cache Coherence in Many-core Systems using Share-pattern Aware Cache Segmentation and Hybrid Network-on-Chip**" submitted by **Antara Ganguly** for the partial fulfilment of the requirements for the degree of *Master of Technology* in *Electronics and Communication Engineering* is an original research work carried out by her under my guidance and supervision at Indraprastha Institute of Information Technology, Delhi. This work has not been submitted anywhere else for the reward of any other degree/dilpoma.

Dr. Sujay Deb
Indraprastha Institute of Information Technology, New Delhi

"God does not play dice with the universe."

-Albert Einstein

Abstract

With the advent of many-core era, scalable hardware support for cache coherence has become vital to system performance. Cache coherence protocols are provided in order to ensure that multiple cached copies of a single memory block are kept up-to-date. As the number of cores being integrated on a single chip is growing rapidly, scalability of cache coherency presents a promising research opportunity. Cache coherency models are broadly based on either snoopy coherence protocol or directory-based coherence protocol. While snoopy coherence is unscalable because of its dependence on ordered networks that are inherently difficult to scale, directory-based coherence is weighed down by its requirement of excessive directory area overhead and inaccurate tracking because of compressed sharer bits. In this work, we propose a scalable cache coherency model for multicore and many-core processors through a hardware and software co-design. We begin with modeling the performance metrics of current cache coherence protocols for high-performance multicore systems interconnected through regular packet-switched network-on-chip architectures and identify the possible bottlenecks imposed by them on system performance. We, then, design and develop network-on-chip architecture augmented with wireless interconnects for efficient handling of broadcast traffic. We propose a segmented design applicable for every level of cache memory according to the sharing pattern of the memory blocks among the cores. Finally, we design and implement an efficient and scalable cache coherence algorithm/protocol that can exploit the proposed wireless interconnects based network-on-chip architecture and the share-pattern aware cache segmentation. We demonstrate that our proposed architecture improves upon the results produced by some well-known multicore architectures employing conventional protocols for cache. Also, owing to the modularity of the proposed design, it can be extended to be used in the future many-core systems by increasing the levels of hierarchy of interconnects, memory and cache coherency.

Acknowledgements

I would begin by thanking Sujay Sir for all the guidance he has offered and faith he has shown in me. I can not thank him enough for the amount of motivation he has provided me throughout my master's. I thank Apala Ma'am for the inspiration that she has been to me and for having helped me in locating my interests and pursuing research in them. I owe it all to her support and I am always going to be thankful to her for that. I would also like to thank Harsha sir and Hemanta sir for helping me with any big or small issue that I faced in the course of my thesis work.

I am grateful to Ma and Baba for making me who I am today. Your **love, support and belief** in me are the only things that keep me going even in the not-so-easy walks of life. Thank you Didi and Rahul Da for your support, for being patient with me the whole time, for everything. You both are my linchpin. And Nishant, thank you for being there, always.

Antara

Contents

List of Figures

List of Tables

Abbreviations

Acronym	**W**hat (it) **S**tands **F**or
ILP	**I**nstruction **L**evel **P**arallelism
MPI	**M**essage **P**assing **I**nterface
NoC	**N**etwork-**o**n-**C**hip
I/P	**I**nput/**O**utput
NUCA	**N**on-**U**niform **C**ache **A**rchitecture
SMP	**S**ymmetric **M**ulti**p**rocessing
CMP	**C**hip **M**ulti**p**rocesser
WDM	**W**avelength **D**ivision **M**ultiplexing
MB	**M**emory **B**lock
CRP	**C**ache **R**eplacement **P**olicy
WB	**W**rite-**b**ack
RD	**R**ea**d**
WR	**Wr**ite
MSI	**M**odified **S**hared **I**nvalid
FSM	**F**inite **S**tate **M**achine
ISA	**I**nstruction **S**et **A**rchitecture
INV	**Inv**alidation coherence messages
ACK	**Ack**nowledgement coherence messages

Keywords:

Multicore, Many-core, Cache coherence protocol, Cache write policy, Network-on-chip, Broadcast traffic, Wireless interconnects

Dedicated to Baba, Ma and Didi...

Chapter 1

Introduction

1.1 Motivation

After hitting the power wall, the memory wall and the ILP wall, designers shifted their focus to developing multicore designs (Figure 1.1) instead of relying on a standalone processor for high performance computing applications [1, 2]. As multicore systems have multiple processors working towards a unified end task, designs have become more communication-bound and less computation-bound (Figure 1.2). Multiple cre-

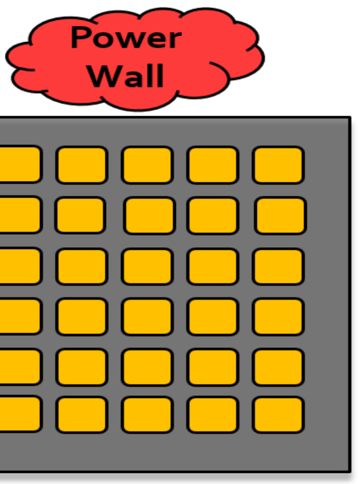

FIGURE 1.1: Evolution of Multicore Systems

ators, modifiers and readers of memory contents coexist on a single chip. Such complex systems are implemented by giving access of memory blocks to the multiple processors through creation of their multiple copies and storing them in the local caches (owned jointly or uniquely by the cores). This approach makes the maintenance of coherency among the numerous copies of the same memory block vital for correctness and accuracy in overall processing. That is where cache coherency protocols come into picture. However, implementation of cache coherency protocols in a multicore system poses certain design challenges [3]. Coherence messages, that is, both requests as well as responses, contribute to a large percentage of the chip traffic and hence, eat up significant part of the bandwidth of the on-chip networks. Intimation of coherence messages to destination processors adds significant overhead to overall runtime of the system. Storing the details of the sharer cores imposes storage overheads on the system which increases with increasing number of cores being integrated on a chip. And, above all, usage of the on-chip resources for implementation of the coherence protocols results in a lot of energy consumption over and above the general energy requirements of the chip. In order to resolve these issues, coherence protocol designs should be such that they add minimal traffic to the actual on-chip traffic by removing unnecessary requests and responses from the network. Latency overheads projected by the protocols should be trivial when compared to the overall system performance. Also, the protocol design should not demand large storage spaces. Taking care of traffic, latency and storage automatically check the amount of energy that is burned by the implementation of the cache coherence protocol on the chip. The proposed protocol involves a hierarchical approach towards coherence maintenance with a specially designed cache system and hybrid network-on-chip implementation promising better performance and bandwidth projections as compared to that of multicore systems with same number of processors but employing conventional protocols, cache hierarchies and interconnects. In a nutshell, this thesis is about pattern aware integration at many levels of a multicore system: communications models, memory models and coherency models.

1.2 About multicore systems

1.2.1 The multicore revolution

It was around the year 2005 that a huge revolution took place in the field of computing called the advent of multicore systems. Intel announced its product lines named Core and Xeon [4, 5], AMD came in with Opteron [6], Sun Microsystems with Niagra [7], IBM with Power4 [8] and so on. All these architectures had one major thing in common. All of them modeled multiple processors as a single processor in order to enhance system performance on the whole. Such systems came to be known as multicore systems and the number of cores being integrated in a system has been increasing ever since. But why is there a need of multiple cores in order to achieve high system performance? Instead, why can the industry not count on the operating frequency of a single core? It is because of the concept of power wall into which the researchers fumbled and realized

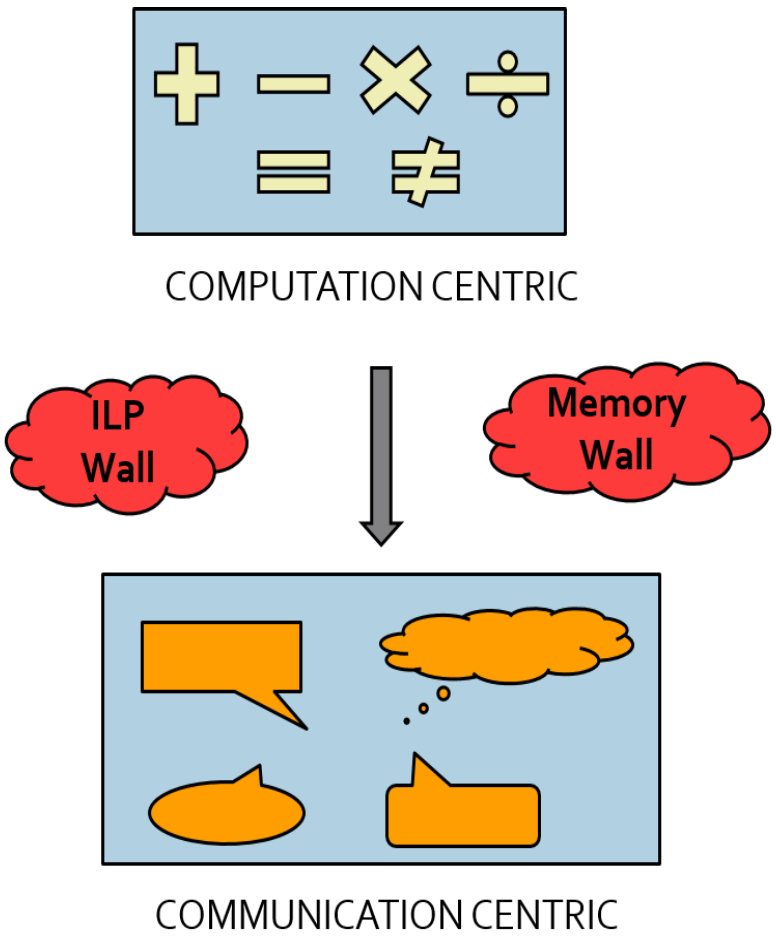

FIGURE 1.2: Shift of focus from Computation to Communication

that it is impossible to go beyond the current speed of a standalone processor without incurring gigantic and hence, unaffordable on-chip power consumption levels. And that is how the large avenue of intensive research in multicores opened up. Thus, in order to achieve high system performance, multiple cores are integrated on a single chip and one can think of mainstream chips with multiple thousands of cores being available in market in the near future [9].

1.2.2 Memory and interconnects in multicore and many-core systems

Any multicore system can be broadly divided into three parts, the processors, the on-chip memory and the interconnects, as seen in Figure 1.3. Further, on-chip memory or cache has levels (Figure 1.4) where each level has a distinct size, sharing property and distance from the processor. As one moves away from the processor, the access latencies of the cache levels worsen, however, the storage capacities increase and hence, a key system trade-off exists between the two parameters of the cache levels [10]. The interconnects, on the other hand, are interwoven links of communication between the multiple processors. Bus-based systems were used as interconnects for a long time until recently when problems of scalability started to appear with more number of cores being integrated into a single system. Then, the packet-switched networks-on-chip (Figure 1.4) were introduced which projected guaranteed throughput, low latencies and hence, high scalability owing to their point-to-point network layout [11, 12]. With further increase in number and hence, distance between processors in a chip, these conventional NoCs suffered from high latency, power consumption and routing problems. This led to the introduction of high-bandwidth single-hop long-range wireless links, especially, in place of multi-hop wired paths [13].

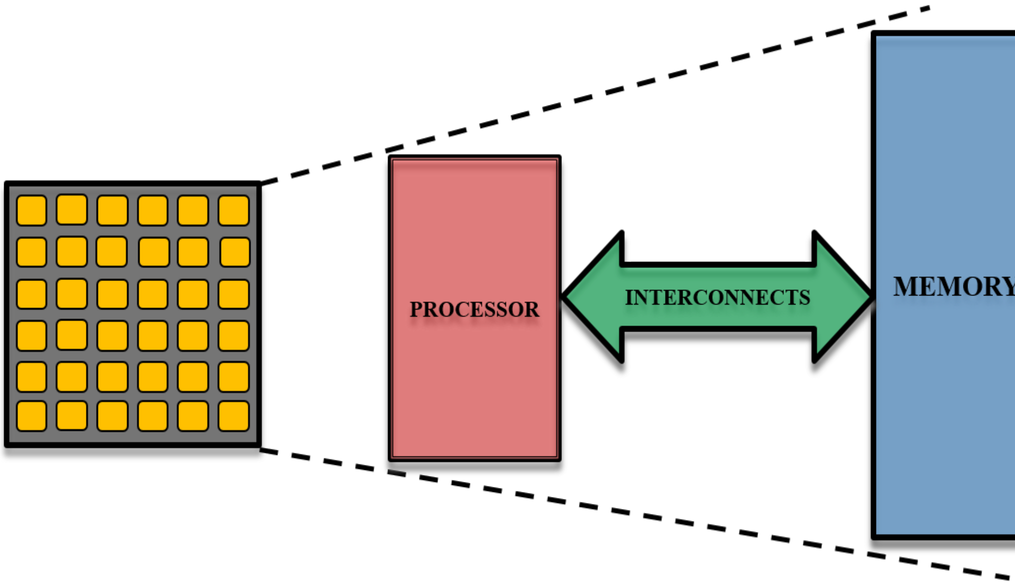

FIGURE 1.3: Blocks of a Multicore System

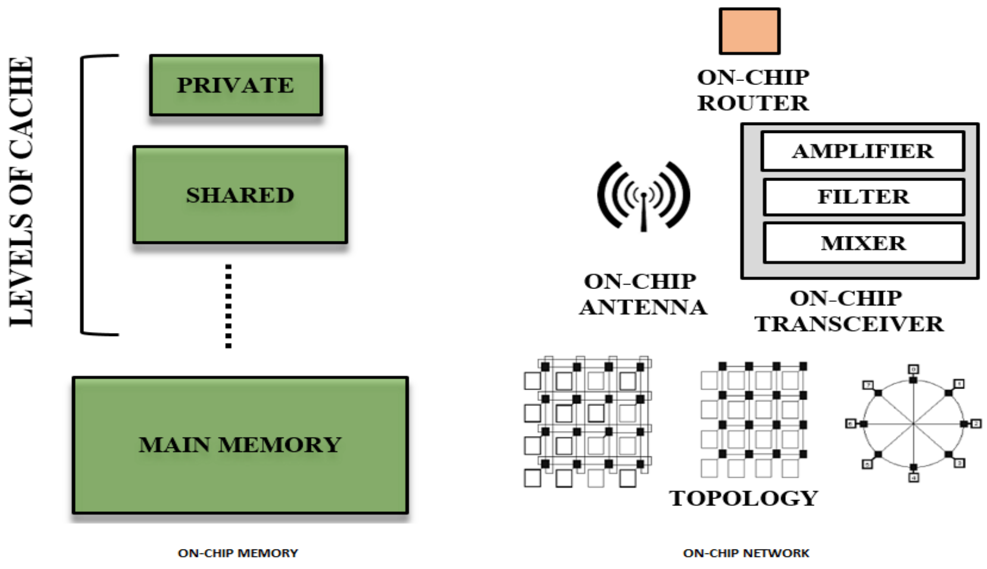

FIGURE 1.4: On-chip Resources of a Multicore Chip

1.2.3 Cache coherency in multicore systems

Cache hierarchies are used in the multicore systems to conceal the latency and traffic overheads incurred by off-chip memory accesses. Implementation of cache memory is more convoluted in multicore systems as compared to that in uniprocessor systems [14]. The cores need to communicate with each other for which shared memory model is employed. However, in order to take care of latency, there are private memory for each core too (Figure 1.5). Thus, with shared as well as private memory in place, multiple copies of memory block are being generated which lead to the possibility of incoherence among all the existent copies of the same memory block.

A cache coherence protocol is the set of conventions that ensure that changes in the values of shared memory entries are communicated throughout the chip in a timely manner to maintain the overall order of execution. The existing cache coherence protocols are built on either snoopy protocols for small systems or directory-based protocols for large-scale shared memory systems (Figure 1.6). These protocols, as has been confirmed, are not scalable beyond a certain threshold number of cores [15].

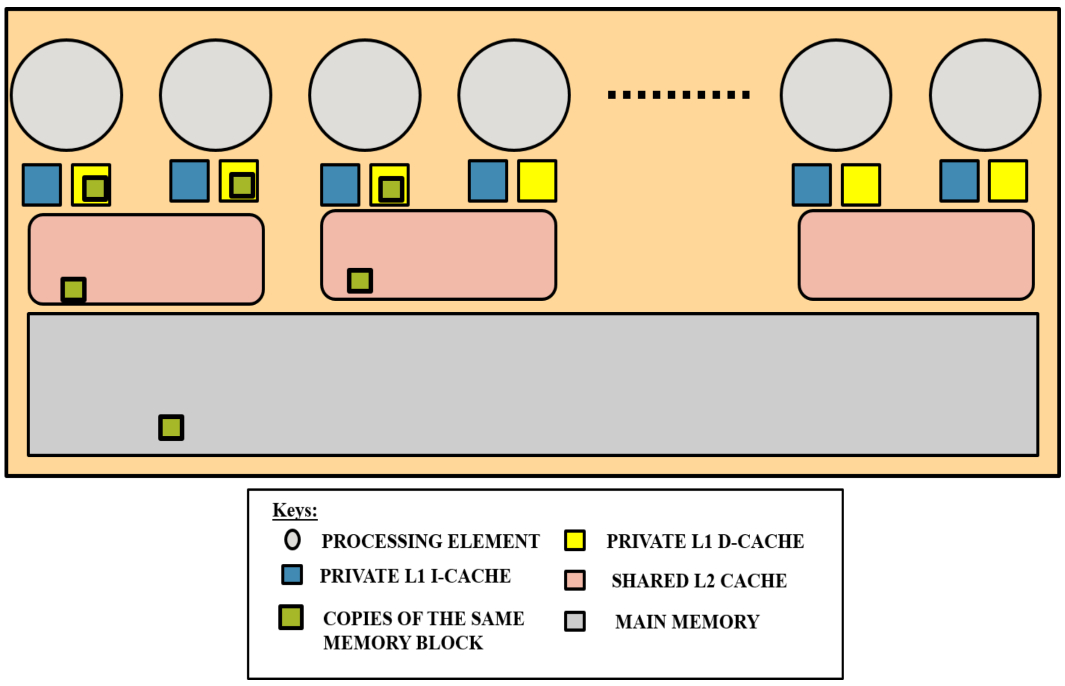

Figure 1.5: Private and Shared Memory Levels in a Multicore System

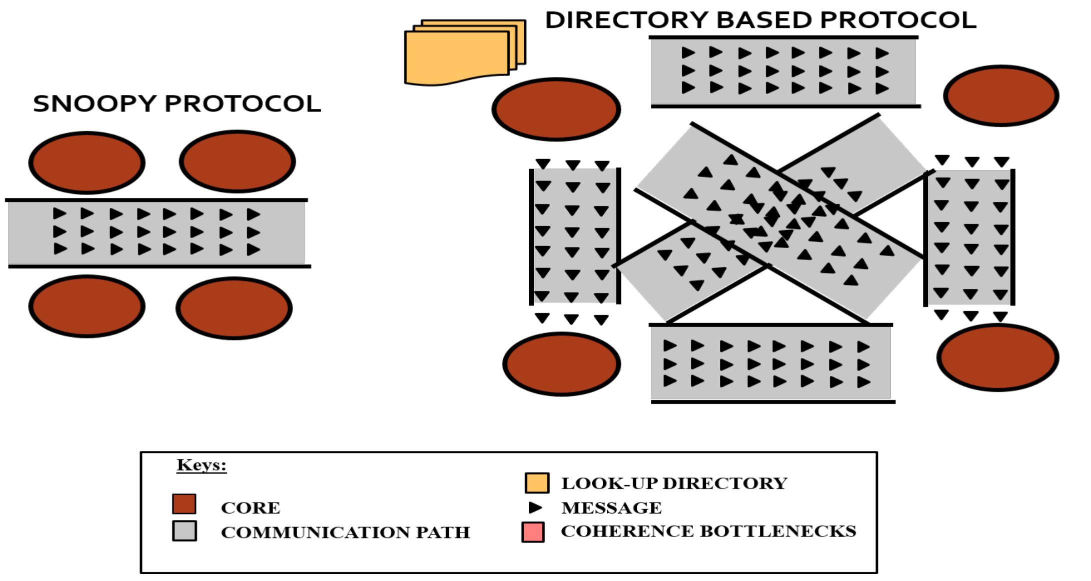

FIGURE 1.6: Overview of Snoopy protocol and Directory-based protocol

1.3 Problem Statement

This thesis proposes a hardware software co-design for a scalable cache coherence protocol tailored for the growing multicore systems. The problem statement can be broken down into the following parts:

- Modelling the existing cache coherency protocols used in high performance computing multicores namely directory based and snoopy in order to understand the extent to which they affect the scalability of multicores,
- Identifying the particular bottlenecks posed by these protocols on the traffic of the scaling multicore systems systems,
- Using these statistics, designing a clustered multicore architecture employing hybrid network on chip which has both wired as well as wireless links. The wired links take care of point-to-point communication for closely located cores and the wireless links carry out the long range broadcast communication efficiently,
- To decrease the traffic and hence, network energy consumption being projected because of coherency protocols, designing a cache system with intra-level segmentation with each segment mapped to memory block in a share-pattern aware manner leading to a significant decrease in traffic of coherence messages and
- Ultimately, designing a scalable cache coherency protocol which exploits the memory as well as interconnect structures in a way that it is scalable for a large number of cores.

With results, it is demonstrated that the proposed model for coherency in multicores and many-cores improves upon the results produced by multicore architectures employing conventional cache coherence, topology, memory and on-chip interconnects.

1.4 Outline

This dissertation is organized into seven chapters. In the next chapter (chapter 2), a concise literature review of various multicore cache coherence designs and their hardware implementations is presented with a gist of multicore evolution and coherence traffic. Chapter 3 presents a background of modular topology for multicore system that employ shared memory and hybrid network-on-chip as mode of inter-core communication promising reduction in latency and power usage. Then, a share-pattern aware design for cache memories has been proposed and discussed that significantly reduces coherence traffic and overall runtime by intelligently mapping memory blocks to appropriate segments of cache. Chapter 4 proposes a hierarchical coherence protocol that exploits the clustered architecture as well as the cache design presented in the previous chapter which is not only scalable but also is suitable for todays communication-centric multicore systems. In chapter 5, the simulators and the experimental set-up have been explained to give an insight into the evaluation methods being used in the next chapter. A comparison of the proposed cache coherence for many-core system with conventional cache coherence protocols on the grounds of on-chip traffic, overall latency and energy consumption has been presented in chapter 6. In chapter 7, a conclusion has been discussed about the advantages of using the proposed design on the system level backed up with the presented results. Also, possible extensions of the proposed work have been suggested and discussed concisely.

Chapter 2

Literature survey

This chapter presents an overview of the existing work related to cache coherence problem in multicore and many-core systems. The scope and the amount of related work is large, so this chapter focuses on the aspects most fundamental and related to the research in this dissertation. The papers relevant to the work done in this thesis have been studied and divided into six groups: ones dealing with multicore architecture on th whole, ones that dealt with the architecture of the cache particularly in multicores, ones that studied the coherence traffic related statistics in multicore chips, ones concentrating on cache coherence designing, ones that presented a study of on-chip networks in multicores and finally, the ones that proposed designs based on usage of on-chip networks for cache coherence.

2.1 Multicore architecture

In order to understand cache coherence in multicore systems and its predicament, it is vital to understand the architecture of the multicore systems. Borkar [30] analyses the integration of hundreds to thousands of small cores, to deliver outstanding system performance considering the constraints of power, memory bandwidth and on-chip interconnects and so on. Experimental analysis and observations of the emerging homogeneous multicore systems has been provided by Kayi, El-Ghazawi and Newby in their paper [26] where they use micro-benchmarks and full system simulation on x86 and SPARC architectures. The paper discusses the effect of multicore architectures in cluster performance and also, the overheads posed by cache coherence.

Butchy et al. use numerical simulation and applications in [27] to provide an overview of existing and emerging multicore and many core technologies as well as accelerator concepts. The paper stresses on hardware-aware computing by discussing interfaces needed to bring the hardware architecture and the implementation of efficient numerical algorithms closer.

2.2 Cache architectures in multicores

Many researchers around the world have presented a thorough study on various cache architectures and their nuances. Hackenburg et al. [17] demonstrate a comparison of different ccNUMA multiprocessor systems with integrated memory controllers and coherent point-to-point interconnects. They use advanced benchmarks for latency and bandwidth measurements in the memory subsystem and draw a close relation between micro-architectural differences and the performance of the memory subsystem.

In [16], Kim et al. propose several designs that treat the cache as a network of banks and facilitate nonuniform accesses to different physical regions. Their work suggests that NUCA designs reduce access latency, increase performance stability and are scalable forms of cache architectures. Molka et al. [18] present fundamental details of Intel Nehalem micro-architecture with its integrated memory controller, quickpath interconnect, and ccNUMA architecture. The paper provides a good insight of latency and bandwidth trends among different locations in the memory subsystem.

2.3 Cache coherence traffic in multicores

Besides cache architecture, a set of researchers have studied and presented their evaluations related to the coherence traffic and its generation mechanisms. Understanding of the traffic patterns and the related protocols/policies is important in order to be able to comprehend the system bottlenecks caused by coherence protocols. Ros and Kaxiras [19] say that directory, state bits, invalidation messages and all such overheads of a coherence implementation are generated in an effort to abstract coherency from memory consistency model. While they modify the cache write policies in order to implement their coherence protocol, Williams, Fensch and Moore [20] implement a new set of links to forward L1 misses optimistically to reduce load on actual on-chip network.

In [21], Kayi et al. investigate multicore specific performance metrics for cache coherency and memory bandwidth/latency/contention for a better understanding of the emerging multicore architectures. Qian, in his paper [24] about inclusive caches, elaborates on the Nehalem architecture and analyzes advantage of the MESIF cache coherence protocol by comparing with the standard MESI protocol. Ramos and Hoefler model communication in cache-coherent SMP systems [29] and develop an intuitive performance model for cache-coherent architectures testing it on Xeon Phi. They present several algorithms for complex parallel data exchanges and compare them with Intel OpenMP and MPI libraries.

2.4 Cache coherence in multicores

As has been discussed in the last chapter, implementation of cache coherence in the emerging multicore systems is a huge challenge because of the latency, storage, traffic and power overheads. Some papers base their research on these bottlenecks and try to minimize them through their modified versions of existing coherence protocols.

Zhang and Jesshope describe an on-chip COMA cache coherency protocol [25] to support the microthread model of concurrent program composition employing dynamic mapping and scheduling. With DiCo-CMP [22, 23], Ros, Manuel and Jos propose a cache coherence protocol for tiled CMP architectures. Their protocol makes the cache store up-to-date sharing information and ensures totally ordered accesses for every memory block on a miss.

Using an arbitration-free passive optical crossbar [28], Zhou et al. propose a design that supports multicasting and many-to-one communication for implementing cache coherency protocols and on-chip interconnect in many-core processors. This paper describes the design, fabrication and evaluation of an arbitration-free passive crossbar based on a microring resonator matrix that can be used to route WDM signals across the chip.

Vladimir et al. [31] in their work on cache coherency use two modified states, the first modified state indicates the most recent copy of a modified cache line and the second modified state indicates a stale copy of the modified cache line. Merchant presents an efficient method for handling multiple conflicting snoop requests with minimal stalling on the external bus. They do so by using blocking conditions to maintain and update a snoop queue for maintaining cache coherence in a computer system with caching unit [32]. In another coherency design [33], Agarwal, Bratt and Mattina configure the cache memories to have a portion in which each line is dynamically managed as either local to the associated processor core or shared among multiple processor cores.

2.5 Study of on-chip network for multicores

Apart from working on the architectural and algorithmic detailing of the cache memory, it is very important to understand the relation of coherence traffic and the performance parameters of on-chip interconnects. Packet dataflows that are transported on the network have a huge impact on the system performance and power consumption of multicores and many-cores. Ye, Benini and Micheli in their paper [38] discuss these impacts quantitatively by introducing a packetized on-chip communication power model and evaluate the relationship between different design options (cache, memory, packetization scheme etc.) at the architectural level. A concise study of possible innovations needed to reduce NoC power consumption has been provided by Buckler, Burleson and Sadowski in their paper [42].

A study [36] done by Massas, Guironnet and Ptrot on memory write policies in shared memory multicore systems takes into account the difficulties related to on-chip communication using network-like interconnects. The paper provides a comparison of write-through and write-back policies based on cycle approximate bit accurate simulations. Kumar and Huggahalli assess the impact of cache coherence protocols on the processing of network traffic in their paper [40]. They observe and provide a detailed study of the performance parameters of I/O specific coherence protocols in the chosen platform.

2.6 On-chip network designs for multicore cache coherence

Researchers have also exploited the emerging technologies in the area of on-chip interconnects in order to provide a scalable coherency model. SCORPIO by Daya et al. [34] uses a separate fixed-latency, bufferless network for distributed global ordering of broadcast messages. In this paper, message delivery is said to have been decoupled from the ordering, allowing messages to arrive in any order and at any time and still, be correctly ordered. Li et al. a hybrid nanophotonic-electric on-chip network called SPECTRUM that optimizes throughput and latency [35]. They use a planar nanophotonic subnetwork that broadcasts latency-critical messages through a wavelength-division multiplexed (WDM) two-dimensional waveguide. Chou et al. [37] use a combination of 3D chip architecture and a ring-based interconnection projecting cost, latency, and power reductions.

ATAC by Kurian et al. [39] uses nanophotonic technology to implement a global broadcast network which is used by a directory-based cache coherence protocol to provide high performance and scalability. The design of PERCS by Armili et al. [43] for a high-productivity high-performance computing system is built using Hub chips that are integrated into the compute nodes. Another paper on interconnect based cache coherence study by Abadal et al. uses graphene-enabled wireless communication on the chip. They have wireless communication capabilities at the core level.

With CCNoC [41], Volvos et al. present a design that trains the NoC to work in accordance with server workloads via a pair of heterogeneous networks adaptive of the type of traffic moving through them. Data path width, router micro architecture, flow control strategy and delay are made to vary in accordance with dynamic traffic.

A thorough study of multicores and many cores, their cache architectures, their cache policies, their interconnect designs and finally, their cache coherence designs helped in building a strong background for developing a scalable cache coherency solution for multicore and many-core systems as presented in the following chapters.

Chapter 3

Share pattern aware cache segmentation

3.1 Motivation and Background

Research shows that three-fourth of the total number of memory accesses target the private data of an application, thus, establishing the dominance of private memory accesses over shared memory accesses in current server applications through industry standard benchmarks [45]. Cache coherence protocol, in general, is oblivious to the sharing pattern of the memory blocks in a multicore system [50]. Irrespective of a memory block being accessed by a single core or by a group of cores, the coherence protocols force every block to go through its predefined order of states which not only delays communication but also adds unnecessary coherence messages to the overall chip traffic [46]. This reduces system efficiency and consumes a huge amount of power which is even higher if the system works with broadcast oriented snoopy protocol [47–49]. On the other hand, if the implemented protocol is directory-based, then the status of every block being accessed has to be entered in the look-up directory. And chances are that many of these blocks do not require resolution of cache conflict because of their sharing pattern incurring unnecessary storage overhead. Hence, an optimised share-pattern aware cache design is highly desirable in order to save on system runtime, storage space and network traffic.

3.2 Building Larger Multicore Systems with Smaller Multicore Building Blocks

From combining uniprocessors to form a multiprocessor to combining multiprocessors to form a huger multiprocessor, the system building block of multicores (and now, many-cores) has come a long way in terms of both memory models as well as interconnect designs. For instance, Intel's Knights Landing, the second-generation Intel Xeon Phi product, is built of multiple tiles which act as building blocks for the architecture [74]. Each tile consists of two cores, two vector processing units (VPUs) per core and a specially designed memory architecture. Cores in a tile share L2 cache. Thus, multiple cores in the building block of multicore and many-core systems act as peers sharing a node exercising optimised shared as well as distributed rights on the available on-chip resources as suited for computing [52]. Such grouping of cores make the system modular, scalable and easy to debug. However, they also demand intricate hierarchical designing of the cache memory, the interconnects and the coherence protocols so as to avoid any compromise on the overall system performance [53–56].

From the point of view of on-chip memory designers, modular multicore architecture facilitates employing shared levels of cache memory for cores in a building block. Thus, instead of looking for a block in the entire cache level, a core can look for it only in its respective share of the cache level reducing latency and cache miss ratio [57].

As for interconnects, treating each module or group of nodes as a node in the architecture, communication between them can be modelled using a combination of single hop hardware implementation (say, wireless) and multi-hop wired NoC so as to reduce number of hops as compared to that in systems with only wired network-on-chip [58].

3.2.1 Shared memory vs MPI

Inter-core communication in multicore chips can take place via shared memory or MPI. [59]. Where the former requires a robust hardware implementation of on-chip memory, the latter requires extensive implementation of synchronisation based protocols to avoid interface speed mismatches [60–62]. The major advantages of shared memory over MPIs are [63]:

- Workload distribution: Shared memory model ensures that each core is given the address of the data it needs. In the MPI model, the actual data is sent to the cores.

- Data access: In shared memory model, the cores access the needed data directly as if the data is local. However, cores in MPI program are supplied with the actual data via messages.
- Data distribution awareness: In the shared model implementation, the programmer does not need to know about the data distribution, while in the MPI version, the programmer must explicitly specify where the data should be sent.

Further, in order to benefit from the cluster based multicore and many-core arrangement, each cluster can be treated as a node in itself and these nodes communicate with each other via a level of cache which is shared among them. Such hierarchical sharing of memory not only helps in controlling on-chip traffic but also, abstracts communication on the different levels from each other, which can be exploited in implementation of efficient cache coherency protocols.

3.2.2 Inter-core communication vs Intra-core communication

Once the cores of the multicore system have been demarcated into building blocks called clusters, on-chip communication can be classified into two categories: inter-cluster and intra-cluster. There is a huge gap in hop-count of communication in the two categories. Especially, for systems containing hundreds or thousands of cores, the inter-core distances for a pair of cores inside a cluster is much less when compared to that of cores located in different clusters, barring few border pairs. More hop count results in increased latencies, decreased throughput and greater power consumption [64]. After a lot of research, it had been proposed that number of hop-counts can be reduced by using long-range links that convert multiple hops into a single hop [65]. Of all implementation propositions made for such distant on-chip communication [66–69], wireless on-chip interconnects have emerged as the most promising solution to the limited scalability of conventional network-on-chip [70]. Wireless interconnects have their own design constraints though, especially, related to area and channel bandwidth and hence, judicial usage of wireless links is a must in a multicore architecture [71]. In terms of the cluster formation [72], the inter-cluster communication being comparatively short-range, wired network on chip is employed inside a cluster while the intra-cluster communication being long-range, wireless links are used to connect different clusters. Apart from management of hop-count, as we will see in the next chapter, wireless networks have other pluses like being broadcast-friendly which can can be exploited in the implementation of snoopy protocols.

3.3 Proposed design

3.3.1 Clustered architecture for on-chip memory and interconnects

The proposed architecture is a highly modular topology employing appropriately shared memory and efficient network-on-chip augmented with wireless links, as seen in Figure 3.1. Considering clusters of NXN cores, without loss of generality, four such clusters are combined to form 2NX2N cores system. The on-chip shared resources are of two types:

- Inter-cluster: L3 cache and wireless network-on-chip
- Intra-cluster: L2 cache and wired network-on-chip

L1I and L1D cache units are not shared and are private to a core, considering L1 to be separated into two parts, one for **I**nstruction and one for **D**ata. Thus, the system can be broken down into clusters which can further be broken down into nodes (Figure 3.2). A cluster consists of wired network-on-chip, a wireless hub and shared L2 cache. A node consists of the processor, private L1 I-cache and private L1 D-cache.

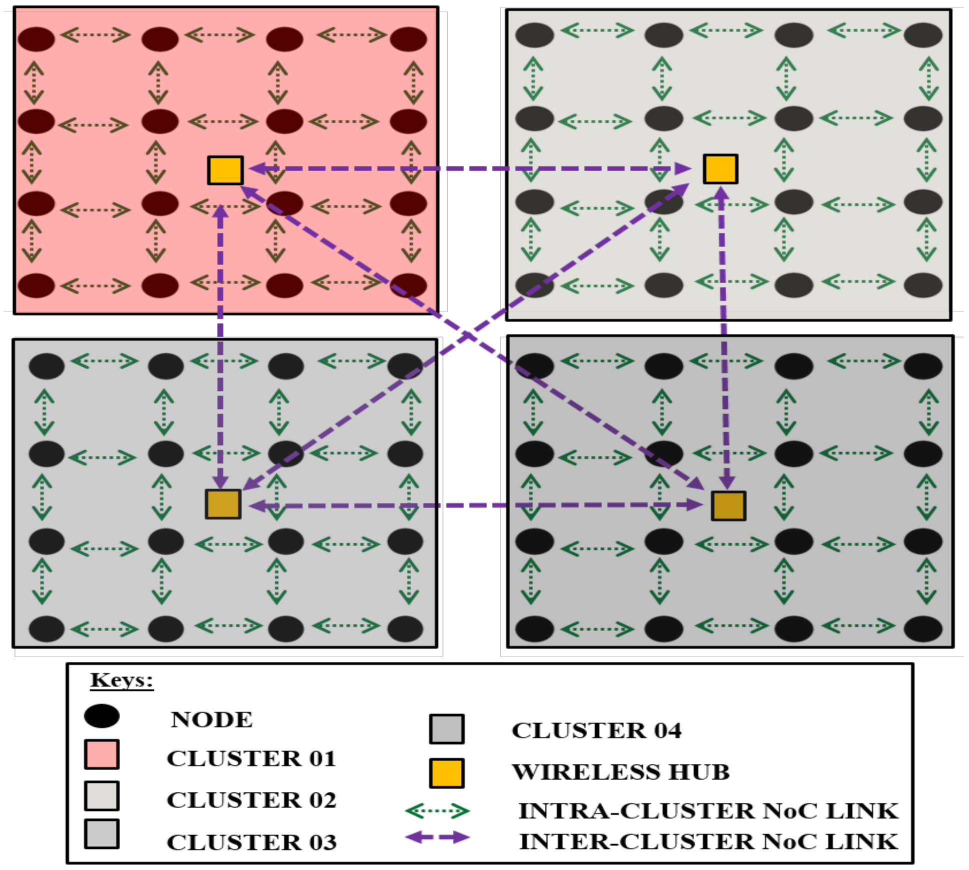

FIGURE 3.1: Representational image of a 8X8 multicore system demonstrating the topology of proposed design

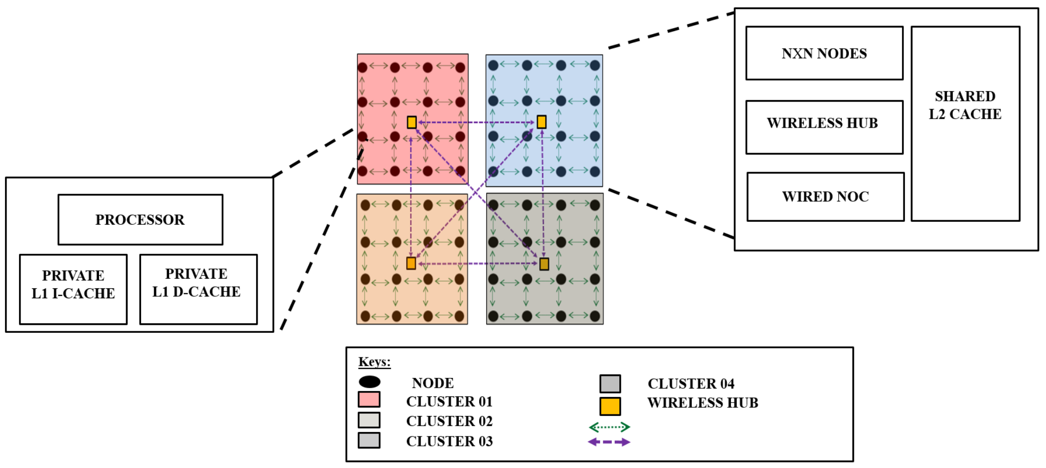

FIGURE 3.2: Block diagrams of a Node and a Cluster

3.3.2 Memory block categorisation based on sharing pattern

The memory blocks being used for processing by the cores of the system can be segregated into the following mutually exclusive sets:

- A = {x : x is a memory block accessed by a single core}
- B = {x : x is a memory block accessed by more than one core, all necessarily in a single cluster}
- C = {x : x is a memory block accessed by more than one core, at least one of which is part of a different cluster}

A refers to all data that will be used by a core only. B refers to all data that will be used by more than one cores but all the user cores are part of the one cluster only. C refers to all data that will be used by more than one cores but all the user cores are not necessarily part of the same cluster, at least one user core is part of a different cluster.

3.3.3 Sharing-pattern aware segmentation of cache levels

On the basis of the memory block categorisation, the cache levels are further divided internally to create a sharing-pattern aware memory hierarchy based on the clustering of the cores, Figure 3.3. As discussed in section 3.3.2, L1 cache is private to a core, L2 cache is shared by all cores in a cluster and L3 is shared by all the clusters. Based on the mentioned sharing trends, following are the segments into which caches are divided:

- L1 cache has three segments, L1A, L1B and L1C.

- L2 cache has two segments, L2B and L2C.
- L3 cache has one segment only, L3C.

As memory blocks of category ‘A’ do not require any sharing at all, L2 and L3 cache levels do not have segment ‘A’. Similarly, as memory blocks of category ‘B’ do not require any sharing beyond a cluster, L3 cache level does not have segment ‘B’ as well and so on (Figure 3.3).

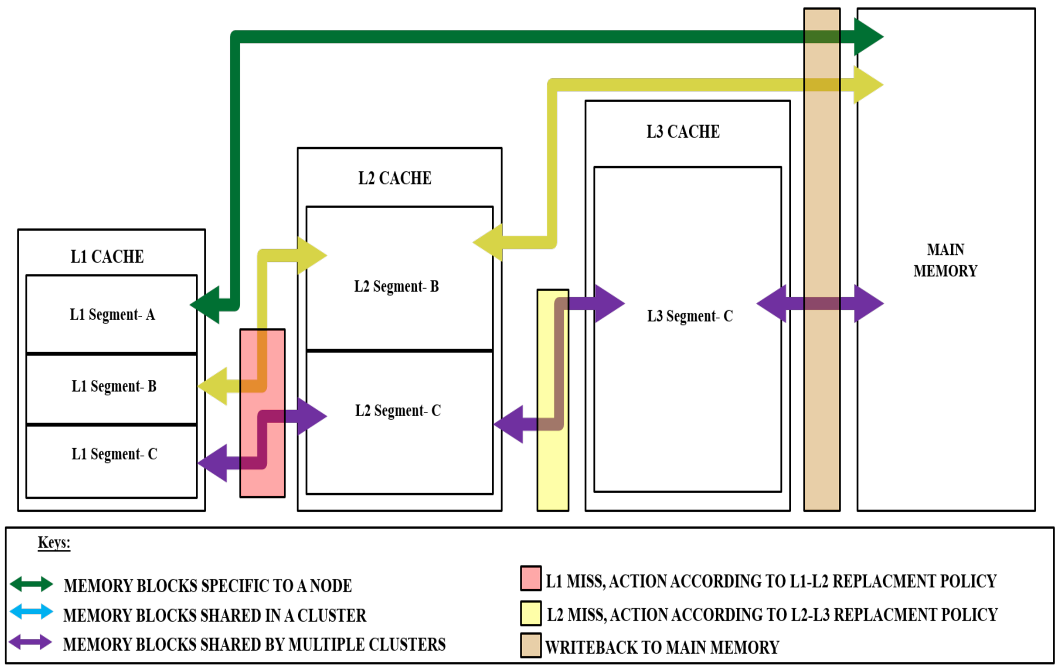
L1 CACHE
L1 Segment- A
L1 Segment- B
L1 Segment- C
L2 CACHE
L2 Segment- B
L2 Segment- C
L3 CACHE
L3 Segment- C
MAIN MEMORY
Keys:
MEMORY BLOCKS SPECIFIC TO A NODE
MEMORY BLOCKS SHARED IN A CLUSTER
MEMORY BLOCKS SHARED BY MULTIPLE CLUSTERS
L1 MISS, ACTION ACCORDING TO L1-L2 REPLACMENT POLICY
L2 MISS, ACTION ACCORDING TO L2-L3 REPLACMENT POLICY
WRITEBACK TO MAIN MEMORY

Figure 3.3: Sharing Pattern Aware Cache Architecture

3.3.4 Cache Write Policy and Cache Replacement Policy

Cache write policy decides what action is taken after a load operation occurs at one of the nodes. Choosing write-back over write-through in order to save network energy through minimised traffic [73], the entire process of writing back to main memory through all the levels of cache can be broken down to three steps considering a system implementation with three levels of cache. In order to make space for a new memory block in a cache level, an existing memory block is chosen in accordance with the cache replacement policy and is evicted from that particular cache level. However, before its elimination, the memory block is written back to the next cache level. After segmentation of the cache levels into A, B and C, it is evident that A type of data in L1 cache is not needed by other clusters and hence, is not written back to L2 cache. Instead, it is written back to main memory directly. Similarly, we find that B type of data in L2 cache can be written into main memory directly by bypassing the L3 cache. Such controlled write-back further reduces network traffic and system runtime. Figure 3.4 and Figure 3.5 depicts the flow of actions that take place in accordance with cache write policy and cache replacement policy when read and write operations occurs in segmented cache system respectively.

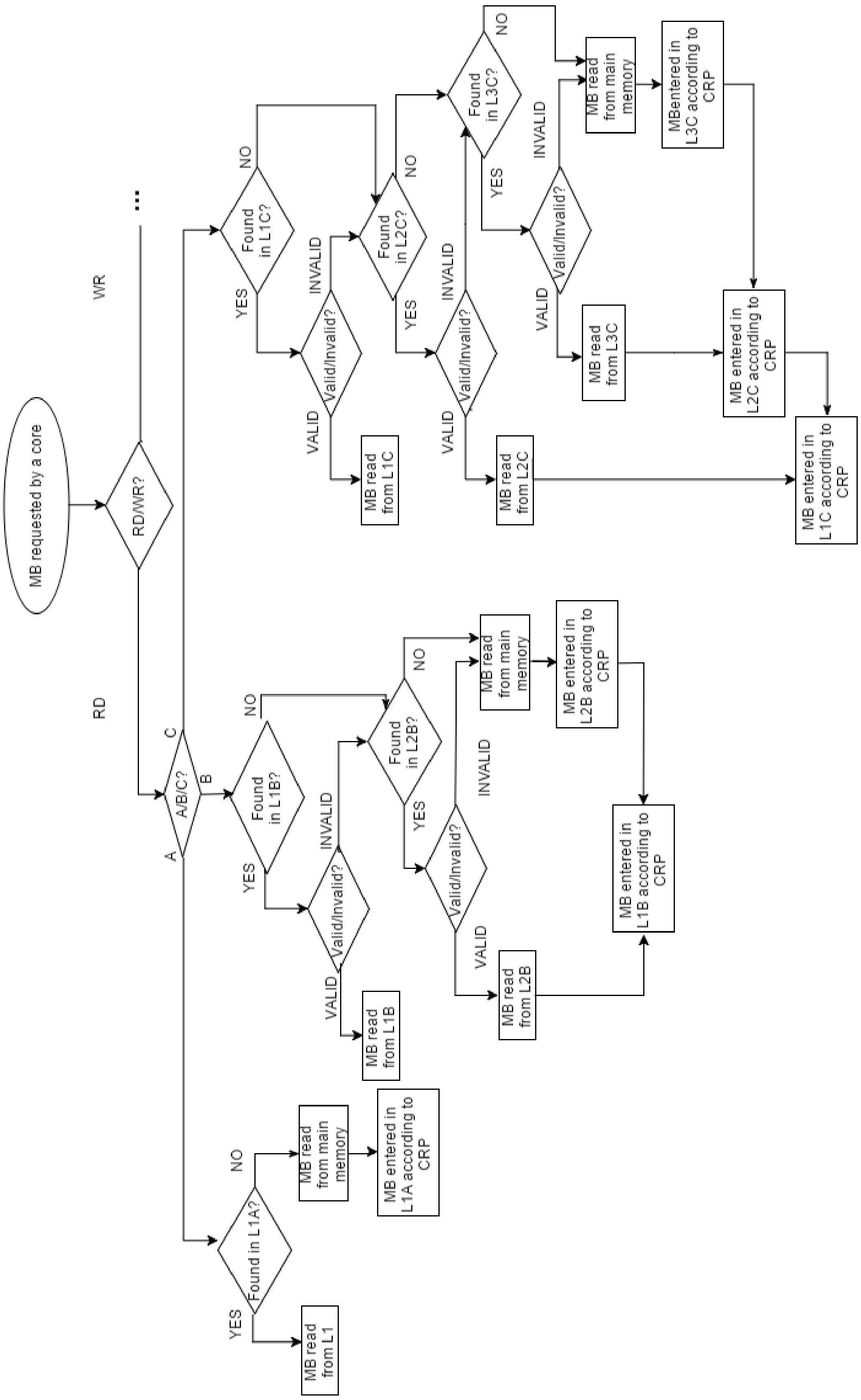

Figure 3.4: Read operation in segmented cache

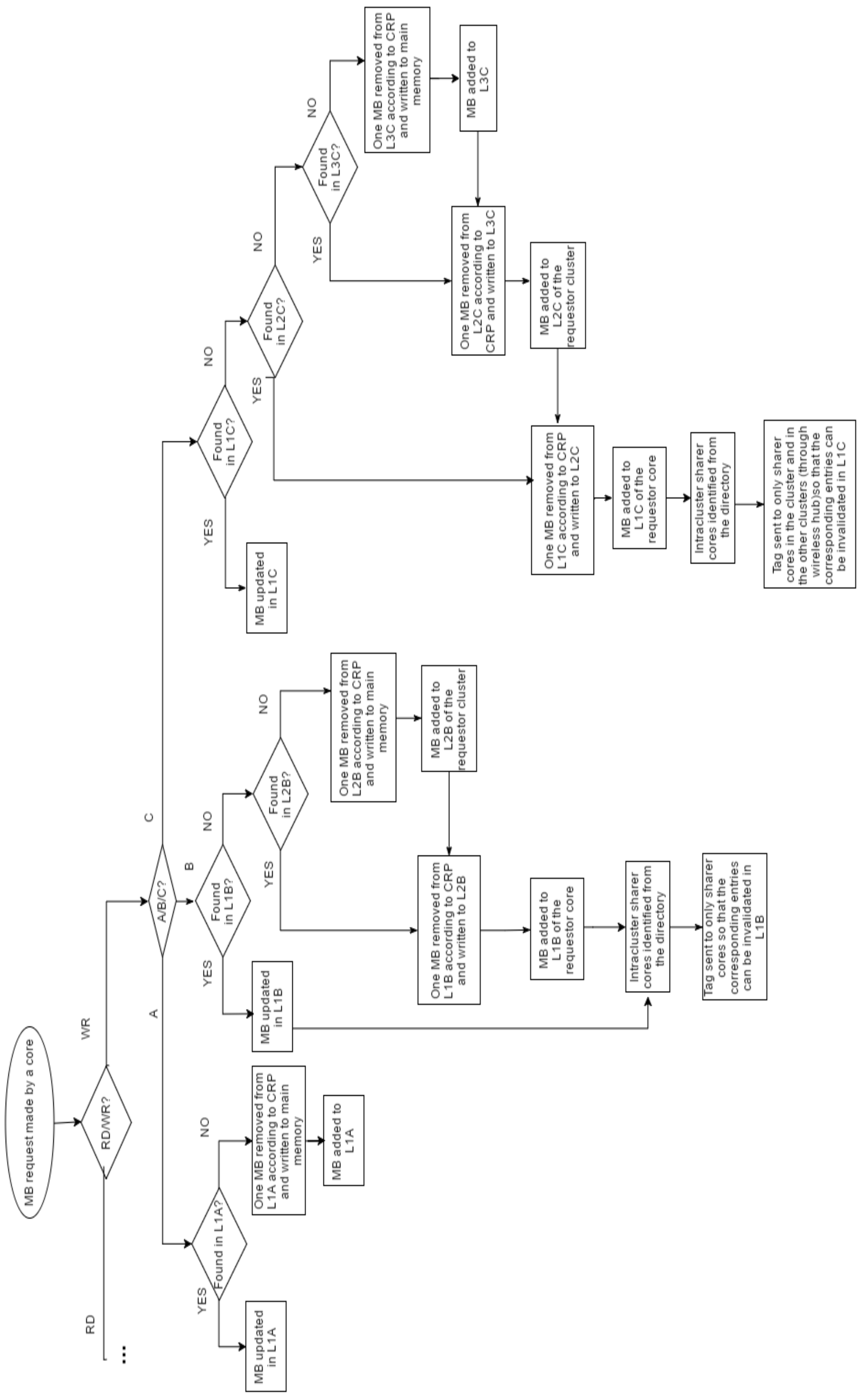
MB request made by a core
RD/WR?
RD
WR
...
A/B/C?
A
B
C
Found in L1A?
YES
NO
MB updated in L1A
One MB removed from L1A according to CRP and written to main memory
MB added to L1A
Found in L1B?
YES
NO
MB updated in L1B
Found in L2B?
YES
NO
One MB removed from L2B according to CRP and written to main memory
MB added to L2B of the requestor cluster
One MB removed from L1B according to CRP and written to L2B
MB added to L1B of the requestor core
Intracluster sharer cores identified from the directory
Tag sent to only sharer cores so that the corresponding entries can be invalidated in L1B
Found in L1C?
YES
NO
MB updated in L1C
Found in L2C?
YES
NO
Found in L3C?
YES
NO
One MB removed from L3C according to CRP and written to main memory
MB added to L3C
One MB removed from L2C according to CRP and written to L3C
MB added to L2C of the requestor cluster
One MB removed from L1C according to CRP and written to L2C
MB added to L1C of the requestor core
Intracluster sharer cores identified from the directory
Tag sent to only sharer cores in the clusters and in the other clusters (through wireless hub)so that the corresponding entries can be invalidated in L1C

FIGURE 3.5: Write operation in segmented cache

Chapter 4

Hardware aware cache coherence protocol

4.1 Motivation and Background

Cache-coherent shared memory is provided by almost all the market giants, Intel, AMD, IBM, Oracle (Sun) and ARM in mobile handsets, personal computers (desktops and laptops), server based applications and so on. It will not be wrong to say that ubiquity of hardware-based cache coherence protocols is not going to diminish as long as they can scale with the increase in number of cores being integrated on a single chip. Hardware solutions to incoherence of multiple copies of memory blocks always provide an abstraction to programmers when compared to the working of software oriented coherency systems, message passing in distributed memory systems or programming in scratchpad memories [75]. However, as research shows, usage of the on-chip resources by the existing coherency cache protocol designs project enormous traffic, storage, latency and energy overheads in the growing multicore systems [76].

This dissertation seeks to debunk the notion of unscalable cache coherence by presenting a scalable on-chip cache coherence protocol for multicore and many-core systems which owing to its hierarchical arrangement and hardware awareness will not be limited to performing well for only small multicore systems but can be easily extended to huge many-core systems as well. The proposed protocol is synergistically combined with the cluster-based on-chip resources and the share pattern aware cache segmentation from the previous chapter to provide a low latency, low energy consuming and reduced traffic projections.

4.2 Proposed hardware-aware cache coherence protocol

4.2.1 Hybrid interconnect awareness in cache coherency protocol

The two cache coherence protocol, directory-based and snoopy, limit the scalability of the multicore systems in different ways [77]. The directory-based cache coherence protocol stores the details of the sharer cores and current state of every memory block in a look-up table and hence, is able to communicate the changes/invalidation messages to the sharers only. Snoopy protocol, instead, broadcasts changes/invalidation messages made by a single core to all cores. While directory-based protocols use two-hop transfers (cache-to-directory-to-cache), snoopy protocols employ one hop transfers (direct cache-to-cache). It is to be noted that one protocol hop is not equal to one interconnect hop necessarily. Directory based protocols do not scale after a point as the size of directory incurs unmanageable memory overheads when the number cores goes beyond 40. Similarly, for snoopy protocols, beyond 15 cores, the multicast traffic is colossal. In

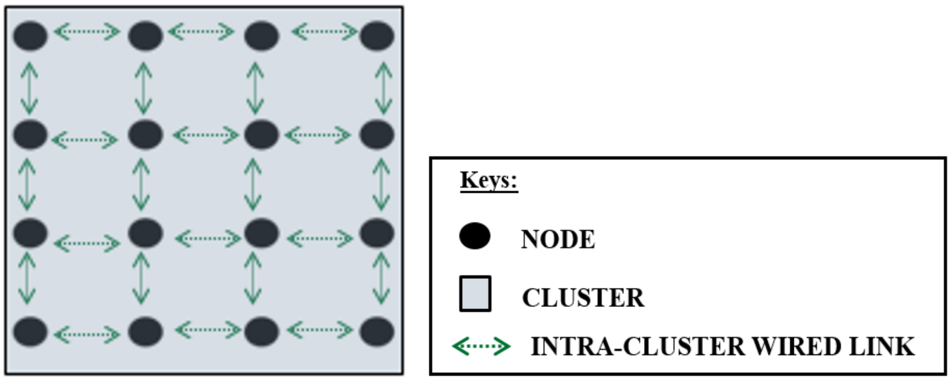

FIGURE 4.1: Intra-cluster Wired Interconnects

directory-based protocols, instead of broadcasting the coherence message to all cores, the destination of the coherence message is looked up from the sharer directory being maintained. As shown in Figure 4.1, packet-switched mesh networks inside the clusters of cores as presented in section 3.2.2 are used for directory-based coherence message. However, the mesh networks are natively unordered. For broadcasting a message from one node to all nodes, the hop-count is different in mesh networks as even though there is a source node is the same, there are multiple destination nodes. Hence, waiting time before the execution of the next queued task has to be implemented so as to abstract multiple unicast communication as a single broadcast communication. Wireless networks, however, provide a channel similar to the old bus-based systems that are apt for broadcasting a message from a single node to several nodes. As shown in Figure 4.2, each cluster in the proposed architecture acts as a node in the wireless network (through its wireless hub). Network or cores inside a cluster are abstracted at the inter-cluster level and do not interfere with the structuring of snoopy coherence. Thus, snoopy coherence protocol is employed at the inter-cluster level to:

- control coherence traffic,
- cut down directory storage area and
- reduce latency overheads in long range coherence communication.

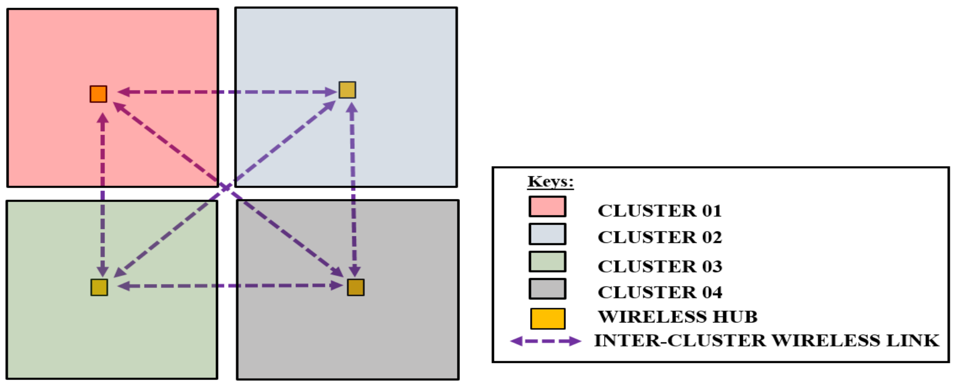

FIGURE 4.2: Intra-cluster Wireless Interconnects

4.2.2 Inner circle and outer circle of the cache segments

In order to capitalize on the sharing-pattern awareness in the cache architecture, the design of the proposed cache coherence protocol can be abstracted into two circles, the inner circle and the outer circle. As described in section 3.3.3, all L1A cache units are mutually exclusive to each other. Similarly, all L2B cache units are mutually exclusive to each other. None of the memory blocks belonging to either L1A or L2B will be copied into the cache of the same level. The inner circle is made of the L1B and L1C cache units and the number of inner circles is equal to the number of clusters on the multicore chip. The outer circle is made of the L2C cache units and there is only one outer circle in the current system. The inner circle and outer circle, thus, model only that part of intra-cluster on-chip memory and inter-cluster on-chip memory that require cache coherency. The inner circle employs directory-based protocol while the outer circle employs snoopy protocol.

4.2.3 Relation of the circles with coherence protocol type

A relation can be established between the type of circle (either inner or outer) and the cache coherence protocol being implemented in it based on the type of interconnect

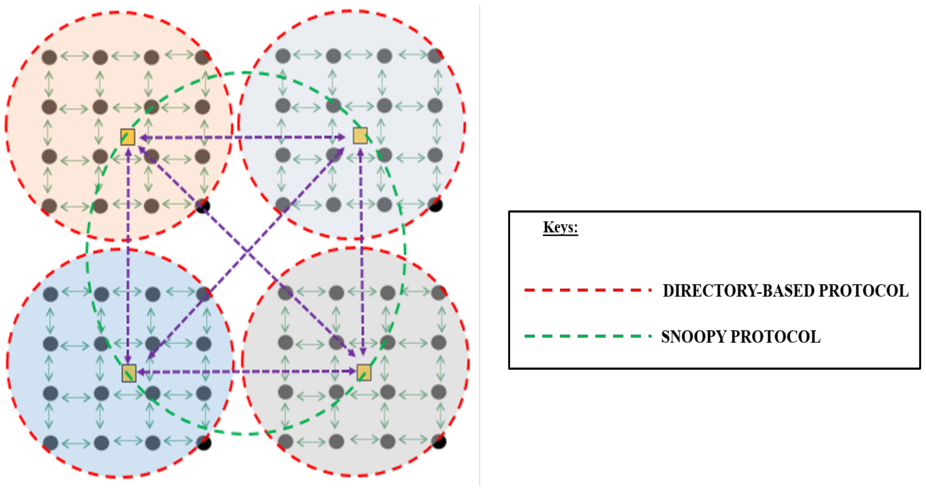

Figure 4.3: Inner and Outer Circles

implementation in that circle (Figure 4.3). As is evident, the inner circle consists of cache units inside a cluster that are owned by cores privately. These cores are connected via ordered mesh networks. Thus, as discussed in section 4.2.1, the coherence protocol implemented inside the inner circle is directory-based protocol. The outer circle, on the other hand, comprises of cache units that are not owned privately by cores, but, by clusters. The clusters are connected through a wireless network and thus, as presented in section 4.2.1, the cache coherence protocol implemented in the outer circle is snoopy protocol.

4.2.4 Cache segment awareness in cache coherency protocol

The inner circle and outer circle have been used to model only that part of intra-cluster on-chip memory and inter-cluster on-chip memory that require cache coherency. Requirement of coherence implementation in each segment of all the three cache level can be understood in the following manner:

- L1A: As L1A cache unit contains memory blocks being used by its owner core only and it does not share its contents with any other core inside its cluster or in any other cluster, it does not require implementation of cache coherence protocol to maintain coherency with other L1A cache units (Figure 4.4).

- L2B: As L2A contains memory blocks being used by cores in its owner cluster only and it does not share its contents with cores in any other cluster, it does not require implementation of cache coherence protocol to maintain coherency with other L2B cache units (Figure 4.5).

- L1B: As L1B cache unit contains memory blocks being used by more than one core in a cluster, it requires maintenance of coherency with other L1B cache units. This is done through directory-based protocol as already explained (Figure 4.5).
- L2C: As L2C cache unit contains memory blocks being used by cores in more than one cluster, it requires maintenance of coherency with other L2C cache units. This is done through snoopy protocol as already explained (Figure 4.6).
- L1C: As L1C cache unit contains memory blocks being used by cores in more than one cluster, it requires maintenance of coherency with other L1C cache units. This is done through directory-based protocol inside the respective cluster. In order to communicate the changes to L1C of cores in other clusters, snoopy protocol is used at the L2C level. Once, the change has been communicated to all the clusters' L2C, again directory-based protocol is used inside the clusters (Figure 4.6).

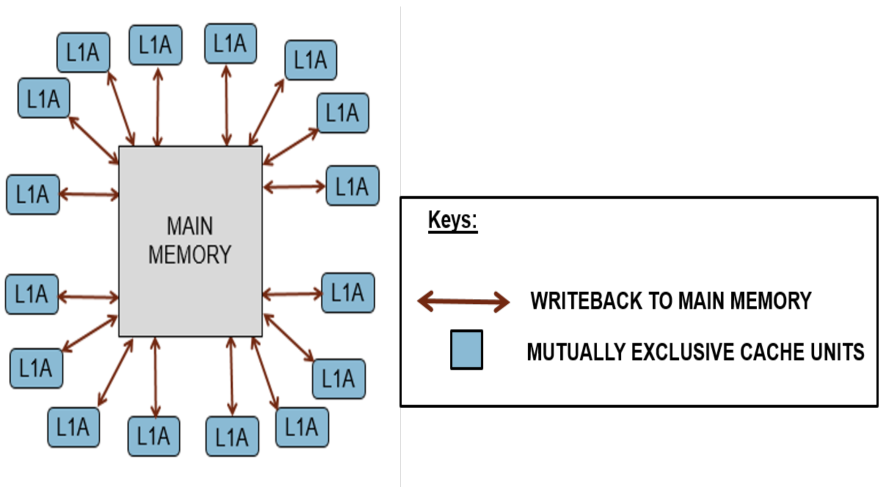

FIGURE 4.4: Hardware Aware Cache Coherence: Segment A

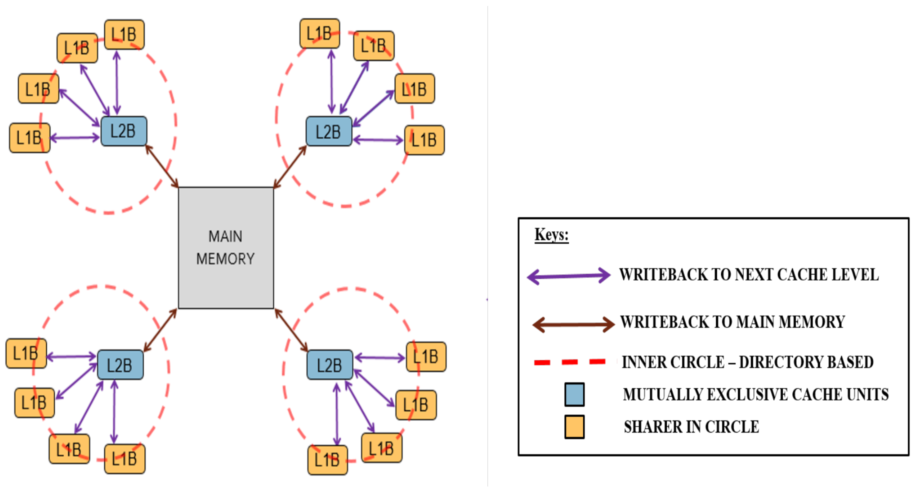
L1B
L2B
MAIN MEMORY
Keys:
WRITEBACK TO NEXT CACHE LEVEL
WRITEBACK TO MAIN MEMORY
INNER CIRCLE – DIRECTORY BASED
MUTUALLY EXCLUSIVE CACHE UNITS
SHARER IN CIRCLE

FIGURE 4.5: Hardware Aware Cache Coherence: Segment B

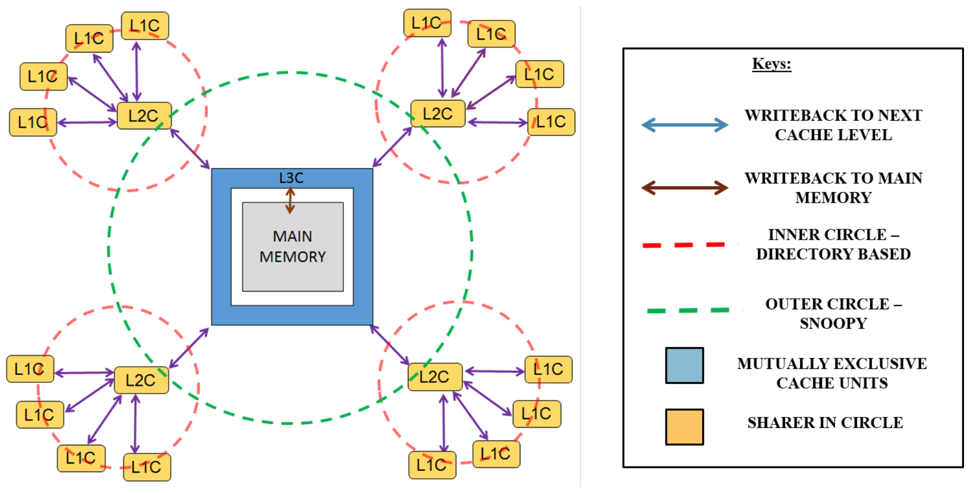
L1C
L2C
L3C
MAIN MEMORY
Keys:
WRITEBACK TO NEXT CACHE LEVEL
WRITEBACK TO MAIN MEMORY
INNER CIRCLE – DIRECTORY BASED
OUTER CIRCLE – SNOOPY
MUTUALLY EXCLUSIVE CACHE UNITS
SHARER IN CIRCLE

FIGURE 4.6: Hardware Aware Cache Coherence: Segment C

Chapter 5

Evaluation Methodology

In order to obtain results related to cache performance, simulation has been done on GEM5 full system simulator [78]. The system is simulated using timing-simple type of CPU which is non-pipelined and emphasizes on the modelling of the timing of memory accesses. The ISA used is x86 which enables the full system simulator to allow Linux booting and workload execution. Full system mode has been used along with RUBY memory system for a detailed implementation and hence, evaluation of the memory system. SLICC, a domain specific language, has been used to implement the new cache coherence protocol in GEM5. On compilation, the SLICC codes generate the necessary C++ codes required for the implementation of the cache coherence protocol. To evaluate the traffic based improvements, traces have been obtained from GEM5 simulator and have been injected into the NOXIM [79] network simulator. NOXIM simulator has been used to simulate the clustered architecture of the cores through implementation of wired links inside the cluster and modeling wireless links to connect the clusters to each other.

Evaluation of all configurations has been carried out with PARSEC [80] benchmarks. Different PARSEC programs that have been used to simulate traffic traces of various applications of high performance computing (Blackscholes, Bodytrack, Canneal, Dedup, Facesim, Ferret, Fluidanimate, Freqmine, Streamcluster, Swaptions, Vips and X264). Multithreaded loads of the mentioned high performance programs with medium as well as large input sets have been used for the purpose of evaluation of the proposed system. Figure 5.1 shows the the order in which the simulators have been used along with their required input and obtained output. Table 5.1 summarizes the experimental setup used to obtain the results discussed in the next chapter.

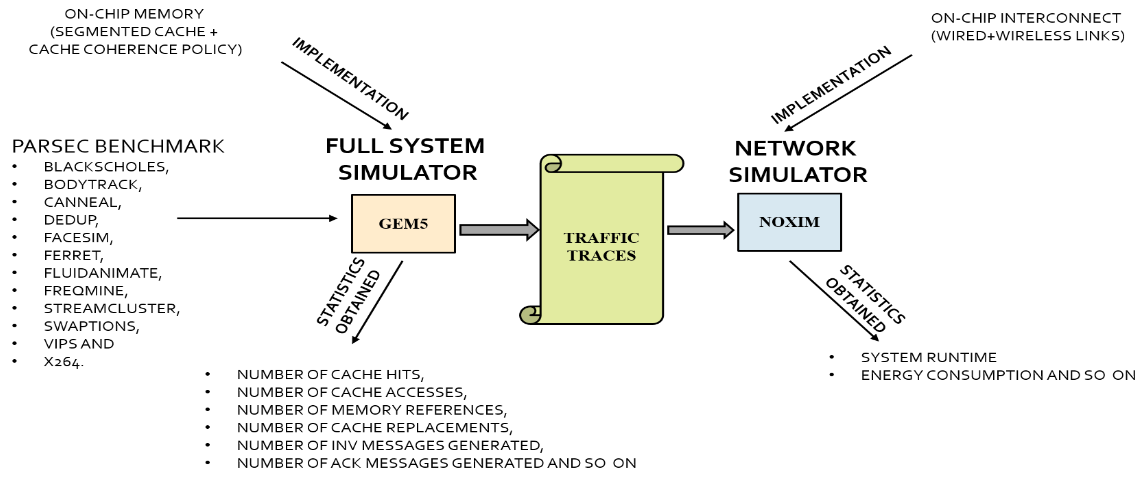

FIGURE 5.1: Implementation methodology

TABLE 5.1: Simulation Setup

Core	TimingSimple (non-pipelined, emphasizes on the modelling of the timing of memory accesses)
ISA	x86 (Linux booting and workload execution allowed by full system simulator)
L1 cache	Private to a core, split 4-way set associative write-back 16 KB I/D
L2 cache	Shared (private to a cluster), inclusive 4-way set associative 128 KB
Line Size	32 bits
FSM type for cache controllers	MSI (with suitable transient states)
NoC Topology	nXn Modified mesh (cluster formation)
Routing strategy	Intra-cluster XY routing, Intercluster Point to point routing

To test the effectiveness of sharing a level of cache among cores for inter-core communication, parameter being considered is the hit ratio of the cache level that is shared in one system and is distributed in the other. Efficacy of using long-range wireless links for inter-cluster communication, rather than using simple wired network has been evaluated by observing the system runtime. Impact of segmenting the levels of cache based on the pattern of sharing of memory blocks among the cores for inter-core communication has been assessed by obtaining the percentage of total traffic contributed by coherence messages, number of total memory references made and number of overall replacements made in L1 cache and L2 cache. Finally, the proposed cache coherence protocol utilizing the modular architecture and the segmented cache levels has been compared with conventional cache coherence protocols for system runtime and overall energy.

Chapter 6

Results and Discussion

6.1 Shared memory vs. MPI

To evaluate the advantages of shared resources in clustered architectures, performance of both shared memory as well as hybrid network-on-chip has been assessed. As is seen in Figure 6.1, the average hit ratio of the cache level (L2), which is shared in the shared memory model and is distributed in the MPI model, is better by 5.6% in the former system. Say, a system can afford to have x kB of L2 cache in total. Then, a

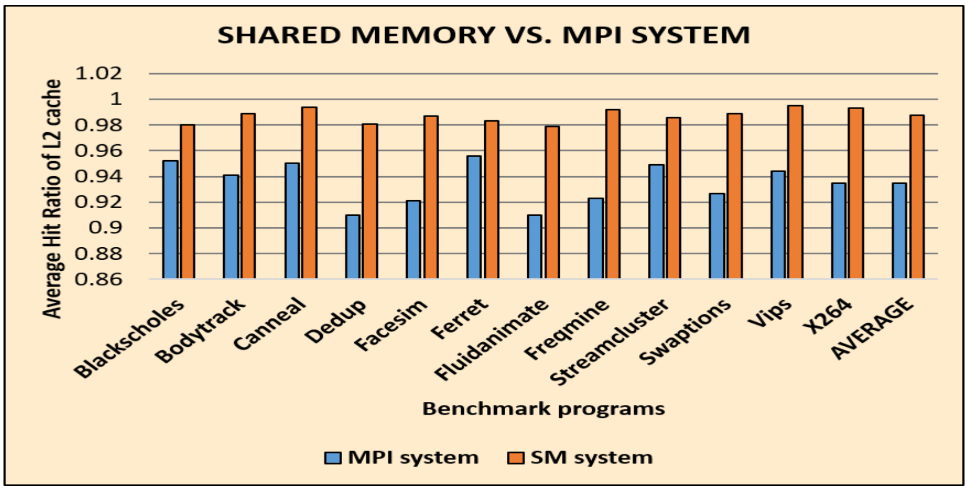

FIGURE 6.1: Average Hit Ratio Of L2 Cache using Shared Memory system

N-core system with 4 clusters will have 4 L2 cache units of x/4 size for shared memory model and N L2 cache units of x/N size for MPI model. Thus, the improvement in the number of cache hits can be attributed to the larger sized L2 caches in the shared memory model than that in the MPI systems. Also, even if there is an increase in L2 access time because of its larger storage space in shared memory systems, the number of main memory accesses made in the distributed memory system is much higher because of the higher cache miss ratio. Thus, the overall system runtime is worse for distributed systems when compared to that of shared memory systems and worsens with increase in number of cores.

6.2 Wired NoC vs. Hybrid NoC

In Figure 6.2, the system runtime shows a reduction of 17.43% for a clustered system with hybrid network-on-chip as compared to that of system without clusters and wired network-on-chip. This speedup can be accredited to the decrease in number of overall

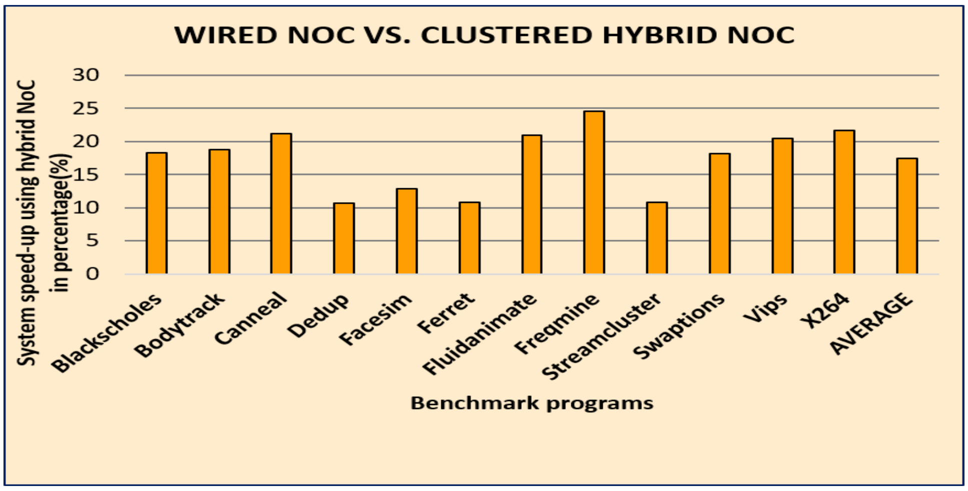

FIGURE 6.2: System Speed-up Using Hybrid NoC

hop counts because of addition of long-range single hop wireless links between clusters.

6.3 Share pattern aware cache design vs. conventional cache design

As is evident from Figure 6.3, system with share pattern aware cache hierarchy shows a reduction of 23.75% in the number of memory references made on an average by the cores as compared to that of system with conventional non-segmented cache architecture because the segmentation of cache levels results in mapping of specific cache areas in successive levels, thus, increasing the chances of hit over that of miss. Figure 6.4 shows that the coherence traffic across the cores drops by almost 30% in segmented cache implementation. The primary reason behind the lessened traffic is the elimination of INV/ACK messages between mutually exclusive cache units of the same level (A in L1 and B in L2). As the segments delegate the transfer of memory blocks according to their

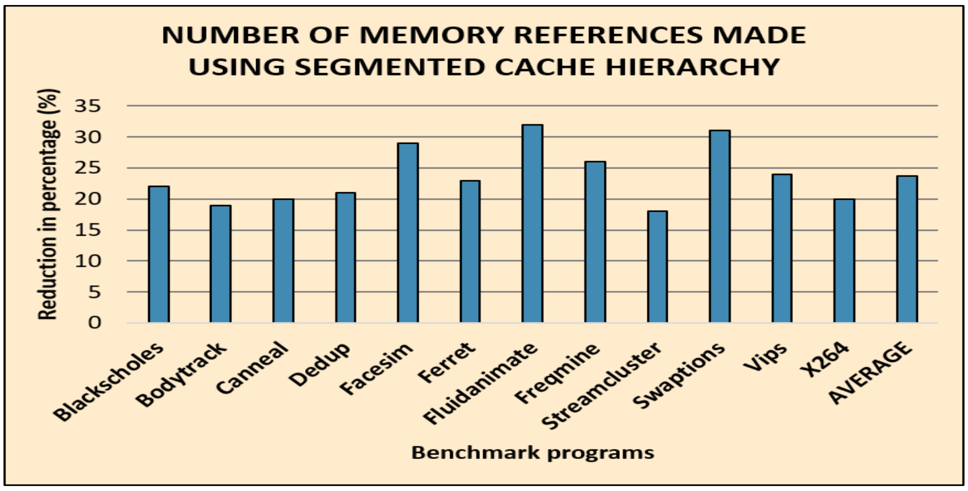

FIGURE 6.3: Reduction in number of Memory References made using Segmented Cache Hierarchy

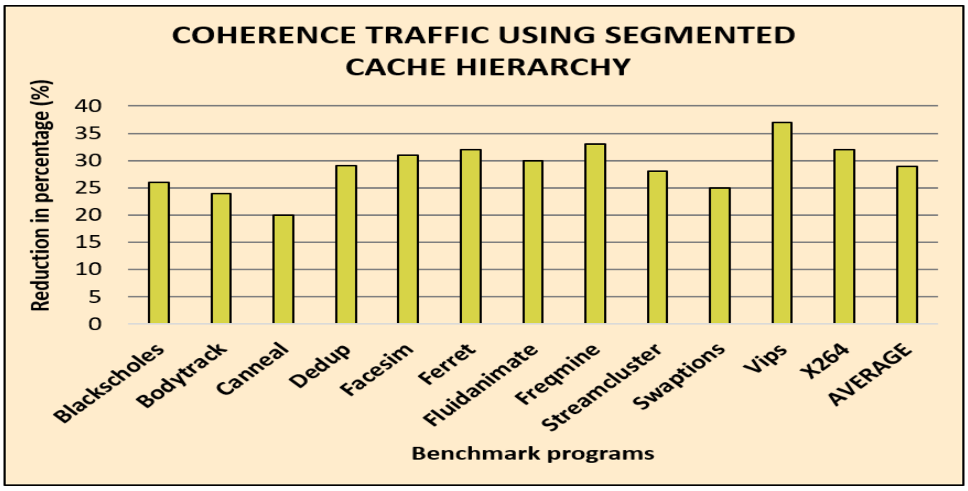

FIGURE 6.4: Reduction in Coherence Traffic using Segmented Cache Hierarchy

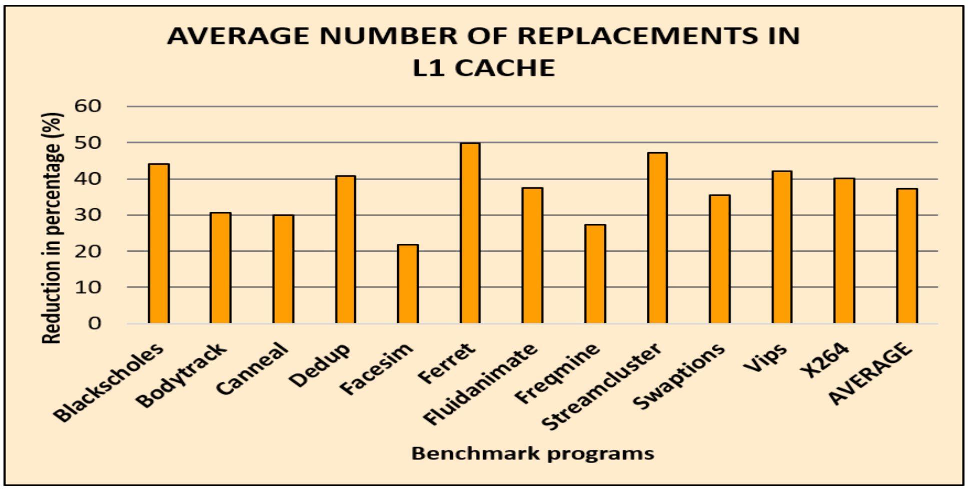

FIGURE 6.5: Reduction in average number of Replacements in L1 Cache using Segmented Cache Hierarchy

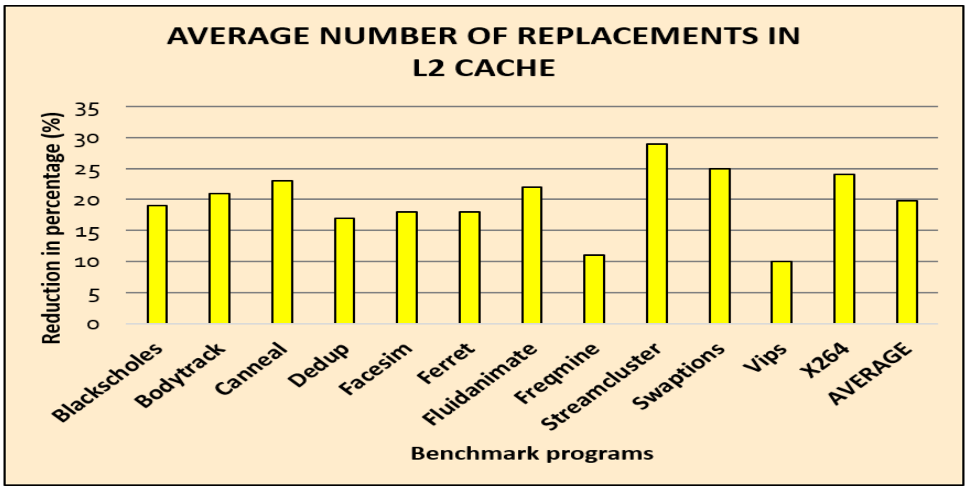

FIGURE 6.6: Reduction in average number of Replacements in L2 Cache using Segmented Cache Hierarchy

access areas (inter-core or intra-core), the cache replacements also show an improvement of 37.23% (Figure 6.5) and 19.75% (Figure 6.6) in L1 and L2 levels respectively.

6.4 Proposed cache coherence protocol vs. conventional cache coherence protocols

After simulations, it is observed that the proposed cache coherency protocol shows a speedup of 23.75% and 26.21% over directory-based (Figure 6.7) and snoopy protocol (Figure 6.8) respectively. The speedup in the system employing the proposed cache

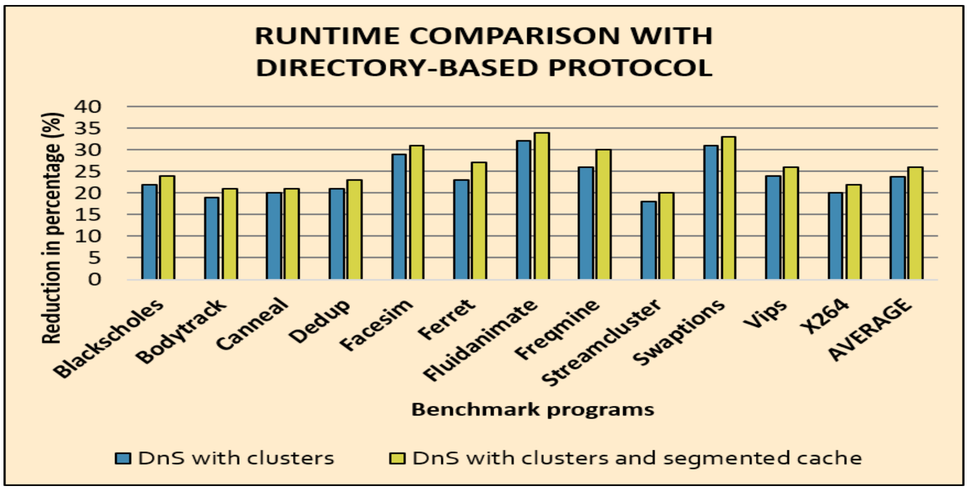

FIGURE 6.7: Reduction in Runtime compared to Directory-based protocol using Proposed Cache Coherence protocol

coherence protocol can be attributed to the judicial usage of wired network in the directory based protocol at the inter-cluster level and wireless channel for snoopy protocol

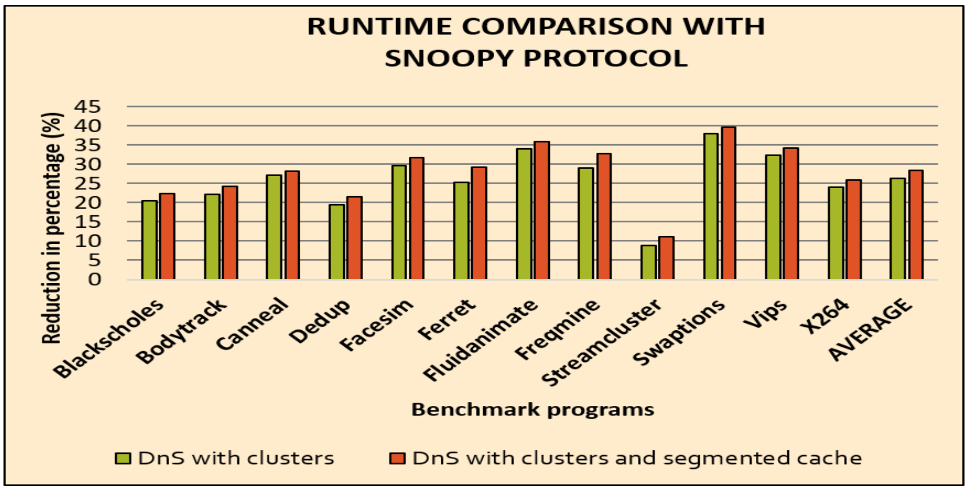

FIGURE 6.8: Reduction in Runtime Compared to Snoopy protocol using Proposed Cache Coherence protocol

at the intra-cluster level. The cache coherency when employed along with the proposed segmented cache projects a further reduction of 2.25% (Figure 6.7) and 2.19% (Figure 6.8) in the overall system runtime. The further reduction in system runtime in the share pattern aware cache system can be attributed to lesser number of cache replacements and hence, reduced main memory accesses accesses and lesser coherence messages. Also, the

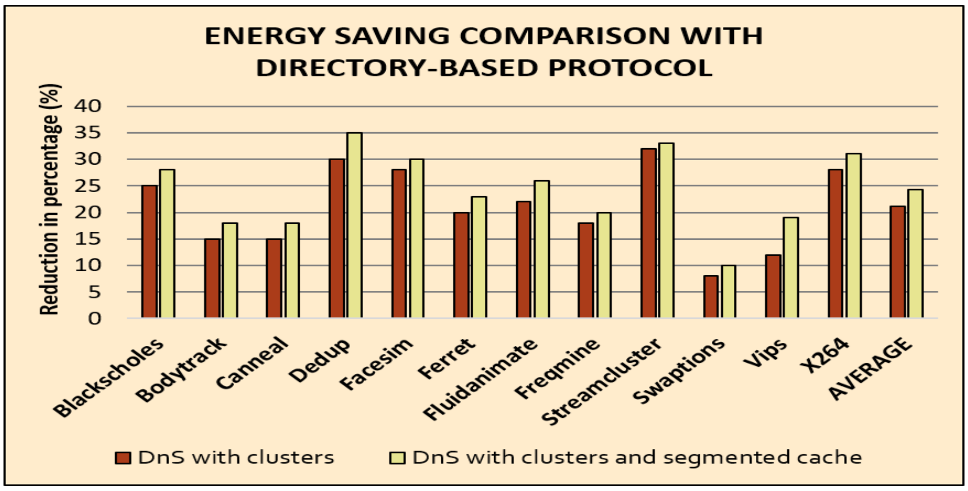

FIGURE 6.9: Reduction in Runtime compared to Directory-based protocol using Proposed Cache Coherence protocol with Segmented Cache Hierarchy

proposed cache coherence protocol projects energy savings of up to 21.08% and 23.5% over directory-based (Figure 6.9) and snoopy protocols (Figure 6.10) respectively which increases to 24.25% and 27.08% when used with proposed segmented caches. The reason behind better energy projections in the proposed system is the check that has been imposed on the usage of the on-chip interconnects because of reduces traffic.

Thus, it is demonstrated that the proposed cache architecture segmentation teamed

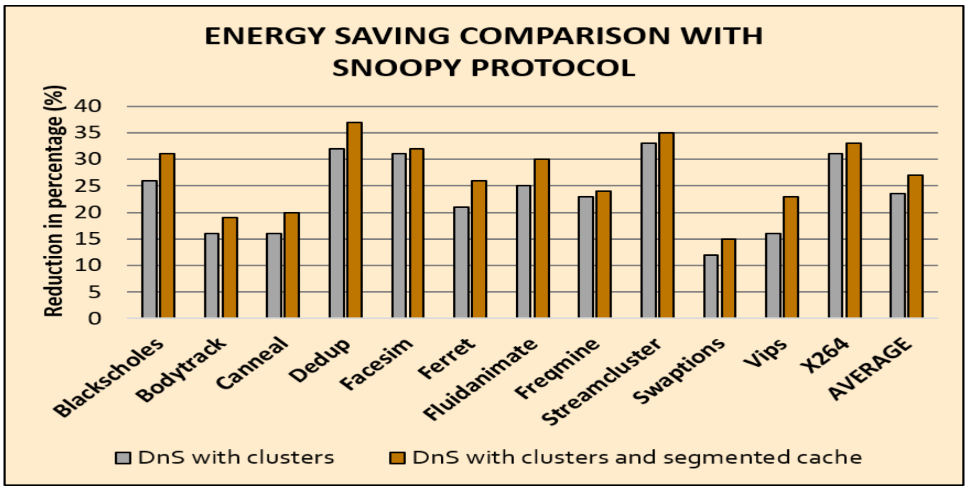

Figure 6.10: Reduction in Runtime compared to Snoopy protocol using Proposed Cache Coherence protocol with Segmented Cache Hierarchy

with the hierarchical cache coherence protocol improves upon the results produced by multicore architectures employing conventional architecture and coherence protocols for cache and on-chip interconnects.

Chapter 7

Conclusion and Future Work

7.1 Summary of work done

The thesis work was set out to design a scalable cache coherence protocol for multicore and many-core systems. In doing so, first, the existing cache coherency protocols used in high performance computing multicores, namely directory-based and snoopy, were explored in order to understand the extent to which they affect the scalability of multicores and many-cores. Then, the system bottlenecks imposed by these protocols on the traffic of the existent systems were identified that act as the reasons and motivation behind the need of developing a scalable coherence design.

Based on these findings, a modular architecture for both on-chip interconnects and cache memory was introduced that would enhance system performance of clustered multicore and many-core topologies. The on-chip interconnect architecture employed packet-switched interconnects for smaller distances as well as wireless links for long-distance communication in order to take care of unnecessary latency as well as energy overheads. The cache hierarchy was further segmented at each level according to the sharing pattern of the memory blocks among the cores and clusters. The segments were implemented in order to reduce coherence traffic, cache miss ratio and main memory access. Finally, a hierarchical solution of cache coherence protocol was proposed which exploited the mentioned interconnect and cache architectures to reduce the overall system runtime and energy consumption.

7.2 Conclusion supported by results

The average cache hit ratio improves by 5.6% in systems communicating through shared memory instead of message passing interface. An average speedup of 17.43% is seen in clustered systems with hybrid network-on-chip as compared to that of system without clusters and wired network-on-chip. Systems with cache hierarchy segmented according to sharing pattern show a drop of 23.75% in the number of memory references and of almost 30% on coherence traffic made on an average by the cores as compared to that of system with conventional non-segmented cache architecture. Also, the cache replacements show an improvement of 37.23% and 19.75% in L1 and L2 levels respectively. The proposed cache coherency protocol shows a speedup of 23.75% and 26.21% over directory-based and snoopy protocol respectively. The cache coherency when employed along with the proposed segmented cache projects a further reduction of 2.25% and 2.19% in the overall system runtime. The proposed cache coherency protocol projects energy savings of 21.08% and 23.5% over directory-based and snoopy protocol respectively which increases to 24.25% and 27.08% when used with proposed segmented caches.

7.3 Possible Extensions of the Proposed Design

The scale of study of coherence in multicore and many-core systems is extensive and multifaceted. To generate achievable improvement in terms of coherence traffic, implementation of efficient cache architecture can be explored further. Share patterns can be further modelled to find more correlations between data used by cores in the same cluster and in different clusters. These findings can be used to propose better cache replacement policies and hybrid cache write policies which aid in reducing coherence traffic. Also, tapping the access directions from each core based on the ISAs and the applications being used, a more dynamic as well as coherence friendly hardware framework can be developed for the on-chip resources.

7.4 Scalability

The proposed hardware aware cache coherence protocol along with the clustering of on-chip resources, on-chip network augmented with wireless links and segmented cache can be extended to the future many-core systems by increasing the levels of hierarchy of interconnects, memory and cache coherency.

Bibliography

[1] Asanovic, Krste, et al. *The landscape of parallel computing research: A view from berkeley.* Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, 2006.

[2] Villa, Oreste, et al. "Scaling the power wall: a path to exascale." *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis.* IEEE Press, 2014.

[3] Martin, Milo MK, Mark D. Hill, and Daniel J. Sorin. "Why on-chip cache coherence is here to stay." *Communications of the ACM* 55.7 (2012): 78-89.

[4] Held, Jim, Jerry Bautista, and Sean Koehl. "From a few cores to many: A tera-scale computing research overview." *white paper, Intel (2006).*

[5] Intel Press Release. Dual Core Era Begins, PC Makers Start Selling Intel-Based PCs. http://www.intel.com/pressroom/archive/releases/2005/20050418comp.htm, Apr. 2005.

[6] AMD Press Release. AMD Introduces the Worlds Most Advanced x86 Processor, Designed for the Demanding Datacenter. http://ir.amd.com/phoenix.zhtml?c=74093p=irol-newsArticleID=1049464, Sept. 2007.

[7] Kongetira, Poonacha, Kathirgamar Aingaran, and Kunle Olukotun. "Niagara: A 32-way multithreaded sparc processor." *IEEE micro* 25.2 (2005): 21-29.

[8] Tendler, Joel M., et al. "POWER4 system microarchitecture." *IBM Journal of Research and Development* 46.1 (2002): 5-25.

[9] Borkar, Shekhar. "Thousand core chips: a technology perspective." *Proceedings of the 44th annual Design Automation Conference.* ACM, 2007.

[10] Hennessy, John L., and David A. Patterson. *Computer architecture: a quantitative approach.* Elsevier, 2011.

[11] Kumar, Shashi, et al. “A network on chip architecture and design methodology.” *VLSI, 2002. Proceedings. IEEE Computer Society Annual Symposium on.* IEEE, 2002.

[12] Bjerregaard, Tobias, and Shankar Mahadevan. “A survey of research and practices of network-on-chip.” *ACM Computing Surveys (CSUR)* 38.1 (2006): 1.

[13] Deb, Sujay, et al. “Wireless NoC as interconnection backbone for multicore chips: promises and challenges.” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems,* 2.2 (2012): 228-239.

[14] Kayi, Abdullah, Tarek El-Ghazawi, and Gregory B. Newby. “Performance issues in emerging homogeneous multi-core architectures.” *Simulation Modelling Practice and Theory* 17.9 (2009): 1485-1499.

[15] Marty, Michael R. *Cache coherence techniques for multicore processors.* ProQuest, 2008.

[16] Kim, Changkyu, Doug Burger, and Stephen W. Keckler. “Nonuniform cache architectures for wire-delay dominated on-chip caches.” *Micro, IEEE* 23.6 (2003): 99-107.

[17] Hackenberg, Daniel, Daniel Molka, and Wolfgang E. Nagel. “Comparing cache architectures and coherency protocols on x86-64 multicore SMP systems.” *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on microarchitecture.* ACM, 2009.

[18] Molka, Daniel, et al. “Memory performance and cache coherency effects on an intel nehalem multiprocessor system.” *Parallel Architectures and Compilation Techniques, 2009. PACT’09. 18th International Conference on.* IEEE, 2009.

[19] Ros, Alberto, and Stefanos Kaxiras. “Complexity-effective multicore coherence.” *Proceedings of the 21st international conference on Parallel architectures and compilation techniques.* ACM, 2012.

[20] Barrow-Williams, Nick, Christian Fensch, and Simon Moore. “Proximity coherence for chip multiprocessors.” *Proceedings of the 19th international conference on Parallel architectures and compilation techniques.* ACM, 2010.

[21] Kayi, Abdullah, et al. “Experimental evaluation of emerging multi-core architectures.” *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International.* IEEE, 2007.

[22] Ros, Alberto, Manuel E. Acacio, and Jos M. Garca. “DiCo-CMP: Efficient cache coherency in tiled CMP architectures.” *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on.* IEEE, 2008.

[23] Ros, Alberto, Manuel E. Acacio, and Jose M. Garcia. "A direct coherence protocol for many-core chip multiprocessors." *Parallel and Distributed Systems, IEEE Transactions on* 21.12 (2010): 1779-1792.

[24] Qian, Bin-feng. "The research of the inclusive cache used in multi-core processor." *Electronic Packaging Technology High Density Packaging, 2008. ICEPT-HDP 2008. International Conference on.* IEEE, 2008.

[25] Zhang, Li, and Chris Jesshope. "On-chip COMA cache-coherence protocol for microgrids of microthreaded cores." *Euro-Par 2007 Workshops: Parallel Processing.* Springer Berlin Heidelberg, 2008.

[26] Kayi, Abdullah, Tarek El-Ghazawi, and Gregory B. Newby. "Performance issues in emerging homogeneous multi-core architectures." *Simulation Modelling Practice and Theory* 17.9 (2009): 1485-1499.

[27] Buchty, Rainer, et al. "A survey on hardwareaware and heterogeneous computing on multicore processors and accelerators." *Concurrency and Computation: Practice and Experience* 24.7 (2012): 663-675.

[28] Zhou, Linjie, et al. "Design and evaluation of an arbitration-free passive optical crossbar for on-chip interconnection networks." *Applied Physics A* 95.4 (2009): 1111-1118.

[29] Ramos, Sabela, and Torsten Hoefler. "Modeling communication in cache-coherent SMP systems: a case-study with Xeon Phi." *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing.* ACM, 2013.

[30] Borkar, Shekhar. "Thousand core chips: a technology perspective." *Proceedings of the 44th annual Design Automation Conference.* ACM, 2007.

[31] Pentkovski, Vladimir, et al. "Method and apparatus for shared cache coherency for a chip multiprocessor or multiprocessor system." U.S. Patent No. 6,976,131. 13 Dec. 2005.

[32] Merchant, Amit A. "Method and apparatus for maintaining cache coherency in a computer system with a highly pipelined bus and multiple conflicting snoop requests." U.S. Patent No. 5,893,151. 6 Apr. 1999.

[33] Agarwal, Anant, Ian R. Bratt, and Matthew Mattina. "Caching in multicore and multiprocessor architectures." U.S. Patent No. 7,805,575. 28 Sep. 2010.

[34] Daya, Bhavya K., et al. "SCORPIO: a 36-core research chip demonstrating snoopy coherence on a scalable mesh NoC with in-network ordering." *Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on.* IEEE, 2014.

[35] Li, Zheng, et al. "Spectrum: a hybrid nanophotonic-electric on-chip network." *Proceedings of the 46th Annual Design Automation Conference.* ACM, 2009.

[36] Massas, De, Pierre Guironnet, and Frdric Ptrot. "Comparison of memory write policies for NoC based multicore cache coherent systems." *Design, Automation and Test in Europe, 2008. DATE'08.* IEEE, 2008.

[37] Chou, Shu-Hsuan, et al. "No cache-coherence: a single-cycle ring interconnection for multi-core L1-NUCA sharing on 3D chips." *Proceedings of the 46th Annual Design Automation Conference.* ACM, 2009.

[38] Ye, Terry Tao, Luca Benini, and Giovanni De Micheli. "Packetized on-chip interconnect communication analysis for MPSoC." *Design, Automation and Test in Europe Conference and Exhibition, 2003.* IEEE, 2003.

[39] Kurian, George, et al. "ATAC: a 1000-core cache-coherent processor with on-chip optical network." *Proceedings of the 19th international conference on Parallel architectures and compilation techniques.* ACM, 2010.

[40] Kumar, Amit, and Ram Huggahalli. "Impact of cache coherence protocols on the processing of network traffic." *Microarchitecture, 2007. MICRO 2007. 40th Annual IEEE/ACM International Symposium on.* IEEE, 2007.

[41] Volos, Stavros, et al. "CCNoC: Specializing on-chip interconnects for energy efficiency in cache-coherent servers." *Networks on Chip (NoCS), 2012 Sixth IEEE/ACM International Symposium on.* IEEE, 2012.

[42] Buckler, Mark, Wayne Burleson, and Greg Sadowski. "Low-power networks-on-chip: Progress and remaining challenges." *Low Power Electronics and Design (ISLPED), 2013 IEEE International Symposium* on. IEEE, 2013.

[43] Arimilli, Baba, et al. "The PERCS high-performance interconnect." *High Performance Interconnects (HOTI), 2010 IEEE 18th Annual Symposium on.* IEEE, 2010.

[44] Abadal, Sergi, et al. "Graphene-enabled wireless communication for massive multicore architectures." *Communications Magazine, IEEE* 51.11 (2013): 137-143.

[45] Cuesta, Blas A., et al. "Increasing the effectiveness of directory caches by deactivating coherence for private memory blocks." *ACM SIGARCH Computer Architecture News.* Vol. 39. No. 3. ACM, 2011.

[46] Bui, Tuan, et al. *Measuring and characterizing cache coherence traffic.* Tech. Rep, 2004.

[47] Agarwal, Niket, Li-Shiuan Peh, and Niraj K. Jha. “In-network snoop ordering (INSO): Snoopy coherence on unordered interconnects.” *2009 IEEE 15th International Symposium on High Performance Computer Architecture.* IEEE, 2009.

[48] Bilir, E. Ender, et al. “Multicast snooping: a new coherence method using a multicast address network.” *ACM SIGARCH Computer Architecture News*27.2 (1999): 294-304.

[49] Agarwal, Niket, Li-Shiuan Peh, and Niraj K. Jha. “In-network coherence filtering: snoopy coherence without broadcasts.” *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture.* ACM, 2009.

[50] Ravishanicar, . V., and James R. Goodman. “Cache implementation for multiple microprocessors.” (1983)

[51] Sodani, Avinash, et al. “Knights Landing: Second-Generation Intel Xeon Phi Product.” *IEEE Micro* 36.2 (2016): 34-46.

[52] Hamid, Norhazlina, Robert John Walters, and Gary B. Wills. “Performance evaluation of multi-core multi-cluster architecture (MCMCA).” *Delivery and Adoption of Cloud Computing Services in Contemporary Organizations* 219 (2015).

[53] Chen, Xiaofang, et al. “Hierarchical cache coherence protocol verification one level at a time through assume guarantee.” *High Level Design Validation and Test Workshop, 2007. HLVDT 2007. IEEE International.* IEEE, 2007.

[54] Lee, Suk-Bok, et al. “A scalable micro wireless interconnect structure for CMPs.” *Proceedings of the 15th annual international conference on Mobile computing and networking.* ACM, 2009.

[55] Blake, Geoffrey, Ronald G. Dreslinski, and Trevor Mudge. “A survey of multicore processors.” *IEEE Signal Processing Magazine* 26.6 (2009): 26-37.

[56] Sarangi, Smruti R., Abhishek Tiwari, and Josep Torrellas. “Phoenix: Detecting and recovering from permanent processor design bugs with programmable hardware.” *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture.* IEEE Computer Society, 2006.

[57] Hwang, Kai, and Naresh Jotwani. *Advanced Computer Architecture, 3e.* McGraw-Hill Education, 2011.

[58] Ganguly, Amlan, et al. “Scalable hybrid wireless network-on-chip architectures for multicore systems.” *IEEE Transactions on Computers* 60.10 (2011): 1485-1502.

[59] Barney, Blaise. “Introduction to parallel computing.” *Lawrence Livermore National Laboratory* 6.13 (2010): 10.

[60] Buono, Daniele, Tiziano De Matteis, Gabriele Mencagli, and Marco Vanneschi. Optimizing Message-passing on Multicore Architectures Using Hardware Multi-threading. 2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (2014).

[61] Friedley, Andrew, Greg Bronevetsky, Torsten Hoefler, and Andrew Lumsdaine. Hybrid MPI: Efficient Message Passing for Multi-core Systems. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on - SC 13 (2013).

[62] Psota, James, and Anant Agarwal. RMPI: Message Passing on Multicore Processors with On-Chip Interconnect. High Performance Embedded Architectures

[63] Kavi, Krishna, et al. “Shared memory and distributed shared memory systems: A survey.” *Advances in Computers* 53 (2000): 55-108.

[64] L. Benini and G.D. Micheli, Networks on chips: A new SoC paradigm, Computer, vol.35, no.1, pp.7078, 2002.

[65] U.Y. Ogras and R. Marculescu, Its a small world after All: NoC performance optimization via long-range link insertion, IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.14, no.7, pp.693706, 2006.

[66] Ye, Yaoyao, et al. “3D optical networks-on-chip (NoC) for multiprocessor systems-on-chip (MPSoC).” *3D System Integration, 2009. 3DIC 2009. IEEE International Conference on.* IEEE, 2009.

[67] Pang, Jun. “Chromophore-based nanophotonic network-on-chip and computing systems.” (2013).

[68] Abadal, Sergi, et al. “Graphene-enabled wireless communication for massive multicore architectures.” *IEEE Communications Magazine* 51.11 (2013): 137-143.

[69] Carpenter, Aaron, et al. “Using transmission lines for global on-chip communication.” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 2.2 (2012): 183-193.

[70] S. Deb, A. Ganguly, K. Chang, P. Pande, B. Beizer, and D. Heo, Enhancing performance of network-on-chip architectures with millimeter-wave wireless interconnects, Proc. IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP 2010), pp.7380, July 2010.

[71] Shouyi, Y. I. N., et al. “Hybrid wired/wireless on-chip network design for application-specific SoC.” *IEICE transactions on electronics* 95.4 (2012): 495-505.

[72] Shreedhar, Tanya, and Sujay Deb. “Hierarchical Cluster based NoC design using Wireless Interconnects for Coherence Support.” *2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID).* IEEE, 2016.

[73] Jouppi, Norman P. *Cache write policies and performance.* Vol. 21. No. 2. ACM, 1993.

[74] Sodani, Avinash, et al. “Knights Landing: Second-Generation Intel Xeon Phi Product.” *IEEE Micro* 36.2 (2016): 34-46.

[75] Martin, Milo MK, Mark D. Hill, and Daniel J. Sorin. “Why on-chip cache coherence is here to stay.” *Communications of the ACM* 55.7 (2012): 78-89.

[76] Bryant, Ray, et al. “Scaling linux to the extreme.” *Proceedings of the Linux Symposium.* 2004.

[77] Sorin, Daniel J., Mark D. Hill, and David A. Wood. “A primer on memory consistency and cache coherence.” *Synthesis Lectures on Computer Architecture* 6.3 (2011): 1-212.

[78] Binkert, Nathan, et al. “The gem5 simulator.” *ACM SIGARCH Computer Architecture News* 39.2 (2011): 1-7.

[79] Catania, Vincenzo, et al. “Noxim: An open, extensible and cycle-accurate network on chip simulator.” *2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP).* IEEE, 2015.

[80] Bienia, Christian, et al. “The PARSEC benchmark suite: characterization and architectural implications.” *Proceedings of the 17th international conference on Parallel architectures and compilation techniques.* ACM, 2008.