MITIGATING CACHE ASSOCIATIVITY AND COHERENCE SCALABILITY CONSTRAINTS FOR MANY-CORE CHIP MULTIPROCESSORS

Thesis By

Malik Al-Manasia

In Partial Fulfilment of the Requirements for the Degree of

Doctor of Philosophy in Computer Engineering

School of Computing and Communications

Faculty of Engineering and Information Technology

University of Technology Sydney

CERTIFICATE OF ORIGINAL AUTHORSHIP

I certify that the work in this thesis has not previously been submitted for a degree nor has it been submitted as part of requirements for a degree except as part of the collaborative doctoral degree and/or fully acknowledged within the text.

I also certify that the thesis has been written by me. Any help that I have received in my research work and the preparation of the thesis itself has been acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

Signature of Student:

Date:

Acknowledgments

O Allah, being able to thank You is the greatest blessing You have blessed me with, You are the one who gave me the capability to accomplish this significant task of my life.

I would like to express my deep and sincere gratitude to my supervisor, Dr. Zenon Chaczko, for his persistent guidance, encouragement, and support throughout my period of study at the University of Technology, Sydney. His advice was critical in helping me lay the right foundation for my research, and thereby develop the right perspective in moving forward.

There are no words that could express my true gratitude to my beloved parents, wife, relatives and friends for their constant motivation, endless support and understanding. This thesis would have been impossible without every one of them. I married my wife half-way into my Ph.D. canditature, for which I appreciate her programming assistance, emotional support and unfaltering understanding. I extend a special gratitude towards my parents, the two people who raised me, who offered their full support, loved me and made the person I am today, for which I express my deepest gratitude.

Abstract

Chip Multi-Processor (CMP) designs have become dominant in the processor market. The evaluation and development of CMPs is essential for product improvement. Up to date, CMPs have presented many challenges for system designers, including cache memory system scalability. My research aims to implement a highly scalable CMP cache memory system using an associative cache, with enhanced replacement policy and a scalable cache coherent protocol.

This thesis establishes a novel Adaptive Hashing and Replacement Cache (AHRC) design, which can maintain high associativity with an advanced method of replacement policy. The AHRC design can improve associativity and keep the possible number of locations of each block (or ways) to a minimum. For the AHRC, the Adaptive Reuse Interval Prediction (ARIP) replacement policy was used because of its ability to resist both scan and thrash.

This research involved simulating several workloads on a large-scale CMP with AHRC as the last-level cache. The results demonstrated that AHRC has better energy efficiency and higher performance than conventional caches. Additionally, larger caches that utilise AHRC are the most suitable in many-core CMPs, as they support scalability as opposed to smaller caches. Scalable cache coherence protocols are essential for CMPs systems, in order to satisfy the requirement for more dominant high-performance chips with shared memory. However, the limited size of the directory cache, associated with larger systems, may result in recurrent directory entries, evictions and invalidations of cached blocks thus compromising system performance.

This thesis proposes the Private/Shared, Read-Only/Read-Write, Invalid/Valid scalable coherence protocol called PROI. This novel protocol implements a slight modification on the caches' tags, allowing it to differentiate between the private and shared data on a block granularity level. Also, PROI employs a dynamic writing policy with self-invalidation and self-downgrade for each L1 cache and can sustain system coherence and performance, scale with the raised number of cores and reduce area, energy, and performance associated costs with the coherence mechanism. The result indicates that PROI can reduce various variables, including the miss ratio of the private L1 cache by 17%, the network traffic, application runtime of approximately 6%, and energy consumption by about 35%. Therefore, utilising AHRC, ARIP, and PROI can mitigate the cache scalability constraints significantly and maintain the performance level while enhancing energy consumption of the CMP cache.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AHRC	Adaptive Hashing and Replacement Cache
ARIP	Adaptive Reuse Interval Prediction
BIP	Bimodal Insertion Policy
BSD	Berkeley-Style Open Source
CC-NUMA	Cache Coherent Non-Uniform Memory Architecture
CMP	Chip Multi-Processors
CPI	Cycles Per Instruction
CPU	Central Processing Unit
CSA	Computer System Architecture
DIP	Dynamic Insertion Policy
DMA	Direct Memory Access
DPIIP	Dynamic Promotion with Interpolated Increments Policy
DRAM	Dynamic Random Access Memory
DRF	Data-Race-Free
DSM	Distributed Shared Memory
DSP	Digital Signal Processor
DVFS	Dynamic Voltage and Frequency Scaling
ECC	Error Correction Codes
EEN	Explicit Eviction Notification
FIFO	First in First Out
FS/A	Full-System / Application-Level

ICEmon	In-Cache Estimation Monitor
IIC	Indirect Index Cache
IPC	Instruction Per Cycle
IPSEL	Insertion Policy Selection
ISA	Instruction Set Architectures
KIPS	Kilo Instructions Per Second
L1	Cache Level 1
L2	Cache Level 2
L3	Cache Level 3
LIFO	Last in First Out
LIP	LRU Insertion Policy
LLC	Last Level Cache
LoD	Level of Details
LRU	Least Recently Used
MLP	Memory Level Parallelism
MPKI	Misses Per 1000 Instructions
MRU	Most Recently Used
NACK	Negative Acknowledgment
NRU	Not Recently Used
NUCA	Non-Uniform Cache Access
NUMA	Non-Uniform Memory Architecture
OPT	Optimal Replacement Policy
OS	Operating System
PC	Personal Computer
PIPP	Promotion/Insertion Pseudo-Partitioning
PPSEL	Promotion Policy Selection

PROI	Private/Shared, Read-Only/Read-Write, Invalid/Valid
PSEL	Policy Selection
PTE	Page Table Entry
QoS	Quality-of-Service
RO	Read Only
ROI	Region of Interest
RRIP	Re-Reference Interval Prediction
RRPV	Re-Reference Prediction Value
RSWEL	Reconstituted SWEL
RW	Read Write
SDM	Set Dueling Monitor
SIPP	Single-step Incremental Promotion Policy
SMP	Symmetric Multi-Processor
SMT	Symmetric Multi-threaded
SRAM	Static Random Access Memory
SWEL	Protocol states are Shared, Written, Exclusivity. Level
SWMR	Single-Writer/Multiple-Readers
TADIP	Thread-Aware Dynamic Insertion Policy
TLB	Table Lookaside Buffer
TMA	Trap-based Memory Architecture
UCP	Utility-based Cache Partitioning
UIUC/NCSA	University of Illinois/NCSA Open Source License
VIPS	Valid/Invalid Private/Shared
W	watt

I. Principal Theory and Concepts

Chapter 1 **Introduction**

1.1. Background

Enhancement of computing processors is guided greatly by Moore’s Law, which has forecasted that the number of transistors per silicon area will double every eighteen months (Moore 2006). Computer architects are embarking on a fundamental shift in how the transistor bounty is used to increase performance, while Moore’s Law is expected to continue at least into the next decade (Marty 2008). Figure 1.1 illustrates the transistor count has been doubling every two years. Furthermore, silicon transistors can only keep shrinking for another few years and Moore’s Law will have its end.

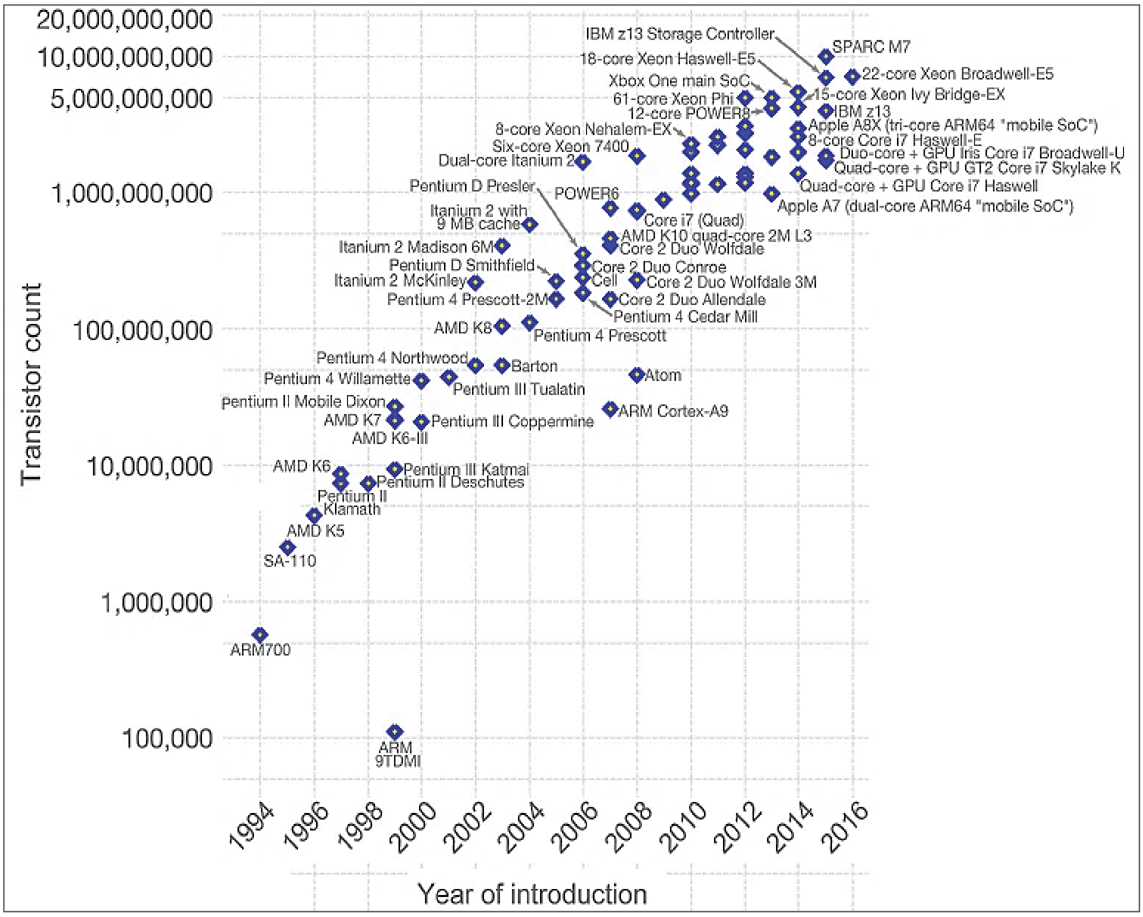


Figure 1.1 Moore’s law for some integrated circuits intensities (Roser 2016)

There is a strong correlation between power and processor clock rate. When the clock rate is enhanced, the power will also rise; after which, the temperatures will also increase (Rao 2009). Multi-core processors take advantage of this relationship by combining multiple cores, with each core able to be run at a lower frequency. By splitting up the power “provided to a single core” nominally between all cores (Geer 2005), the performance will improve, whereas the power and temperature will remain under control. Figure 1.2 shows this main advantage and the significant performance enhancement over the single core processor (Rao 2009; Spjut et al. 2009).

Also, multi-core technology can improve system efficiency and application performance for computers running multiple applications and multi-threaded programs simultaneously (Geer 2007; Ku et al. 2009). In the rest of this thesis, the terms multi-core and many-core will be used interchangeably, and the terms processor, core and processor core interchangeably.

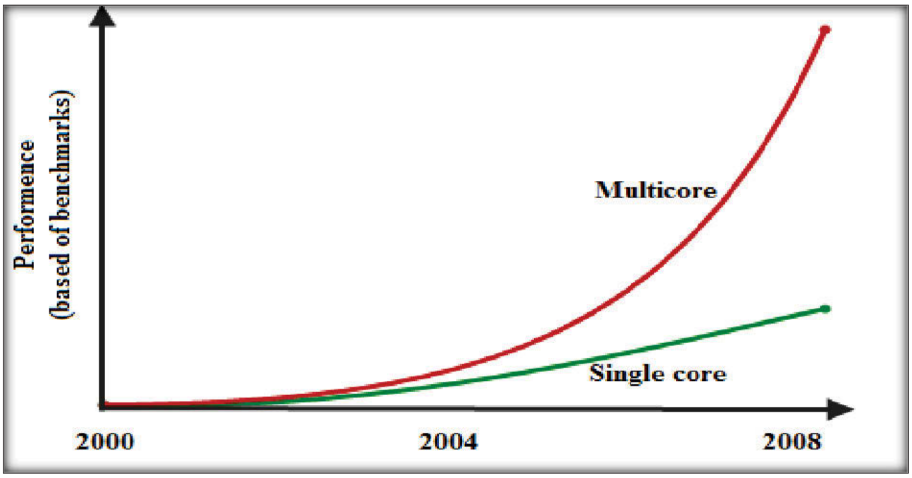


Figure 1.2 Multi-core performance compared to a single core (Rao 2009)

Regarding multi-core processors, cache coherency refers to the credibility of data stored in each core's cache. Multicore processors may contain distributed and shared caches on the chip, so this study should account for the coherence protocols to assure that when a core reads from memory, it reads the current piece of data and not a value that has been updated by a different core.

In multi-core processors architecture, caches are used for two main purposes; to reduce memory traffic and decrease the average memory latency. Thus, caches are crucial to maintain and increase the system performance; however, the possibility of cache incoherence in the private caches is the issue. For example, if four cores share the same block of the datum in their private caches, and one of these cores writes the datum, then the other cores may provide the old value because their caches are not up to date.

The modern multipurpose CMPs, which work for small-scale operations, are constructed using caches that are associative array functioning to store raw data and address, data pairs. The memory access functions to look up the caches. If however, the data block is non-existent, then it is taken from the preceding level until it is recovered from the intermediate cache or main memory. Cache is more advantageous in capturing the locality in an implicit and transparent manner to enable the ease of operation, rather than using explicitly-addressed, software managed local stores.

1.2. Rationale

Some researchers predict, albeit without a strong proof, that hardware cache coherence is going to disappear, as soon as, the number of cores increases to large numbers (Martin, Hill & Sorin 2012). In addition to the complexity arguments, there are a few researchers who believe that hardware cache coherence has poor scaling of the storage and the traffic on the interconnection network that coherence requires. These factors lead them to conclude that future many-core chips will communicate with software-managed coherence instead of hardware cache coherence. The purpose of this study is to investigate and mitigate the cache scalability challenges in manycore architectures.

1.2.1. Cache Scalability

Shared memory is the paramount low-level communication model in today's mainstream multi-core processors. Moreover, hardware cache coherence is paramount in the market for two reasons (Martin, Hill & Sorin 2012): technically, because the performance provided by hardware cache coherence is superior – incomparable with software-implemented coherence, and for its compatibility with legacy and current software.

In the case of current scale with cores amounting to double digits, the hierarchy of the cache is usually organised as indicated in Figure 1.3. To start with, each core has one or more levels of private caches, which provide fast and energy-efficient access to the critical working set of the running threads. Secondly, CMPs further include a large, fully shared last-level cache (LLC). A single shared LLC is preferred over multiple, private LLCs, in order to increase cache utilisation and provide faster inter-core communication. This usually occurs through shared caches instead of making use of the main memory. More concisely, in order to keep the numerous transfers visible to software, CMPs carries out an elaborate protocol that facilitates the communication between various caches. This further allows either several read-only copies or a single read-write copy of each of the cache lines.

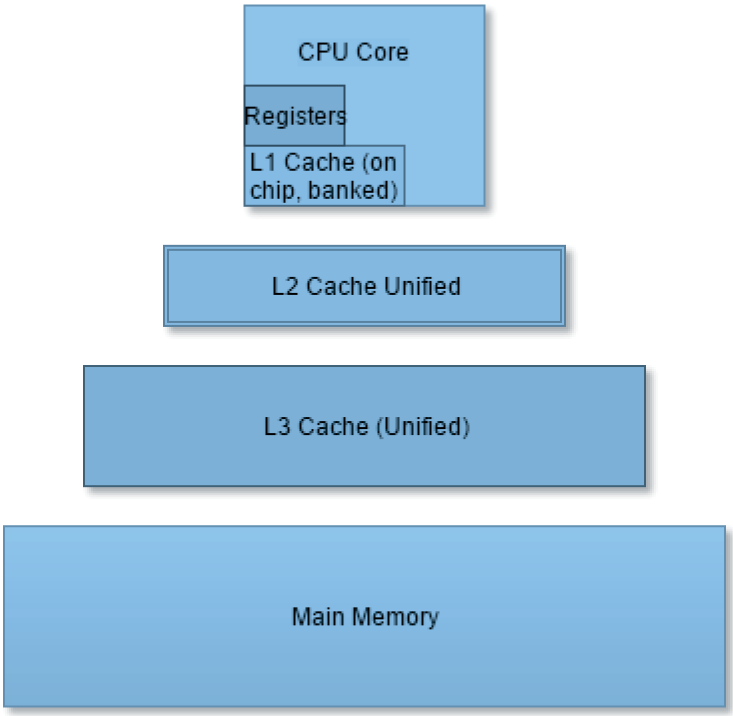


Figure 1.3 General Cache Hierarchy Levels

The current setup works in the CMPs with fewer cores making several aspects of CMPs, which are hard to scale when possessing hundreds or thousands of cores. To start with, large-scale CMPs need caches that have a greater degree of associativity, which is usually very costly to implement using modern methods. Secondly, the currently used cache implementations for coherence procedures are hard to scale beyond a few tens of cores, thus requiring considerable space, energy, and complex operations.

Finally, this research aims to focus on policies of practical cache replacement approach which attempt to emulate perfect replacement by predicting the reuse interval of a cache block. The most commonly used replacement policy, least recently used (LRU), gives a prediction of a near immediate reuse interval on cache misses and hits. Those applications that show a distant reuse interval have proved to be working poorly under LRU. This is because such applications have a working-set that is larger than the cache or experiences frequent regular bursts of reference to non-temporal data which is known as scans. To improve the performance of such workloads, this research will focus on the replacement of cache by utilising the knowledge of reuse interval prediction.

1.2.2.Multicore and Mobile Technology

A decade ago, central processing unit (CPU) manufacturers encountered the emergence of new performance challenges. One such challenge relates to the sustenance of high CPU speeds while maintaining their temperatures low enough using practical techniques. This has, however, been challenging especially after the developers realised that they could not increase speeds beyond the 4GHz limit without having to derive new methods of cooling because of the rapid heat generated by such speeds. For example, maintaining such high speeds would require that the developers opt for unrealistic and costly cooling methods, such as using liquid nitrogen and antifreeze coolants (Patrizio 2013). Nonetheless, through research, the developers found a suitable solution for this heat menace, they have discovered that using multiple cores would distribute the heat between the processors, and thus would ultimately aid in reducing the amount of heat generated by the increased speeds (Marcu et al. 2008; Patrizio 2013).

Elsewhere, portable devices (e.g. smartphones and tablets) developers have also been experiencing similar problems regarding hardware improvement. Portable devices are often limited in terms of space, which makes heat dissipation a daunting task to many manufacturers today. For example, installing a 4GHz processor would mean increased speed for the devices; however, the heat generated by the processor would be way above the levels that could be dissipated by obvious mechanisms. Quad-core CPUs such as Qualcomm’s Snapdragon and Nvidia’s Tegra 3 are some examples of multiple core processors developed using this technique. In this respect, the developers are also opting to install multiple cores instead of single-high order cores (NVIDIA 2010). The multiple core architecture works by reducing the clock speeds while at the same time reducing the voltage supply in a bid to achieve high levels of power efficiency and thereby lessen the amount of heat generated by the CPUs.

However, the scalability that functioned poorly for the personal computer (PC) and server markets (to some degree) is exacerbated by portable device developers, because the nature of the development environment is entirely dissimilar. Most users use their smartphones in a similar manner to their PCs, implying that they expect the two devices performance to be at par. In this case, managing the battery life for smartphones become the main challenge that can only be addressed using multi-core CPUs.

Furthermore, the increasing demand for higher functionality devices acts to increase the costs associated with the computational hardware and software resources and developments. In this view, designers would arguably avoid developing non-scalable and non-programmable hardware solutions as this would be difficult to attain or would make their manufacturing costs high. Instead, the designers opt for programmable multi-core solutions as they can achieve the desired optimal performance, power consumption at lesser costs, and provide a great extent of flexibility (Marcu et al. 2008).

Ideal as it seems, the chip-level multiprocessing architecture is not without challenges. Developing a chip to mount the multiple processors is among the main challenges (Gepner & Kowalik 2006; Hubbard et al. 2008). Preferably, the multiple processors generate a substantive amount of heat, which can pass out by the mounting chip. In this case, the designers are tasked with ensuring that the designed chip distributes the generated heat evenly to avoid instances of growing hot spots (Schauer 2008).

Furthermore, mobile technology possibly makes the scalability problem even worse, since power consumption and chip area should always be minimised while trying to improve processing performance. Cache scalability, which is currently not a big problem, is also expected to be problematic to mobile hardware designers in the near future. Considering that mobile CMPs designers achieve coherency between caches using snooping protocols, it is apparent that problems with bandwidth saturation and messages congestion will emerge since the snooping protocols can only scale to a limited number of cores. Hence, improving the mobile cache would require a reorganisation and analysis of the cache configuration in order to maintain the small chip area and low power consumption, while providing the optimal level of performance (Davanam & Lee 2010).

1.3. Aims of the Study

As mentioned earlier, some forecasters anticipate that the period of hardware cache coherence is approaching its end. For this reason, this research will focus on some of the most important factors about the scalability of many-core CMPs, including storage cost and interconnection traffic, and energy efficiency – all while maintaining the level of performance.

Storage: to maintain consistency in the private cores' cache; hardware cache coherence needs to track which of the private caches are caching a specific block. This storage should scale as long as the number of cores scale, but this scaling must be graceful.

Coherence traffic: which represent the coherence protocols communication messages and this traffic must not grow more and more, if the number of cores grows increasingly. This thesis will prove the ability of hardware cache coherence of scaling to a huge number of cores.

Energy efficiency: Caches usually consume a significant amount of energy in modern CMPs. Energy efficiency is important to mobile computing applications (e.g. cellular phones, notebook computers, and consumer electronics). They not only require high performance, but also low-energy consumption for longer battery life. Another driving force behind designing for energy efficiency is that power consumption is becoming the limiting factor in integrating more transistors on a single chip or on a multi-chip module due to cooling, packaging and reliability problems.

1.4. Research Hypotheses

It is possible to construct a highly scalable CMP cache memory system with improved performance, reduced power consumption and optimized usage of the CMP directory memory area by implementing a highly associative cache with enhanced replacement policy and a scalable cache coherence.

Until the beginning of this century, CPU performance was improved by increasing the number of transistors. Because of the speed of light and heating problems, this solution can't improve performance anymore without big changes. Figure 1.4 demonstrate this in the year (2004). The new solution was inspired by the principle of multiprocessor architecture systems but in a different level of complexity. Multicore architectures work well up to a few number of cores and in (2010) as Figure 1.4 shows other problems had arisen like parallelism and scalability. This research will focus on the scalability issues of the cache memory for large-scale many-core systems, where it will take care of performance, power consumption and complexity of the memory system at the same time.

Figure 1.5 represents the research question, the proposed solutions, and the validation methodology. More details of this figure will be described in Chapters 3, 4, 5, and 6.

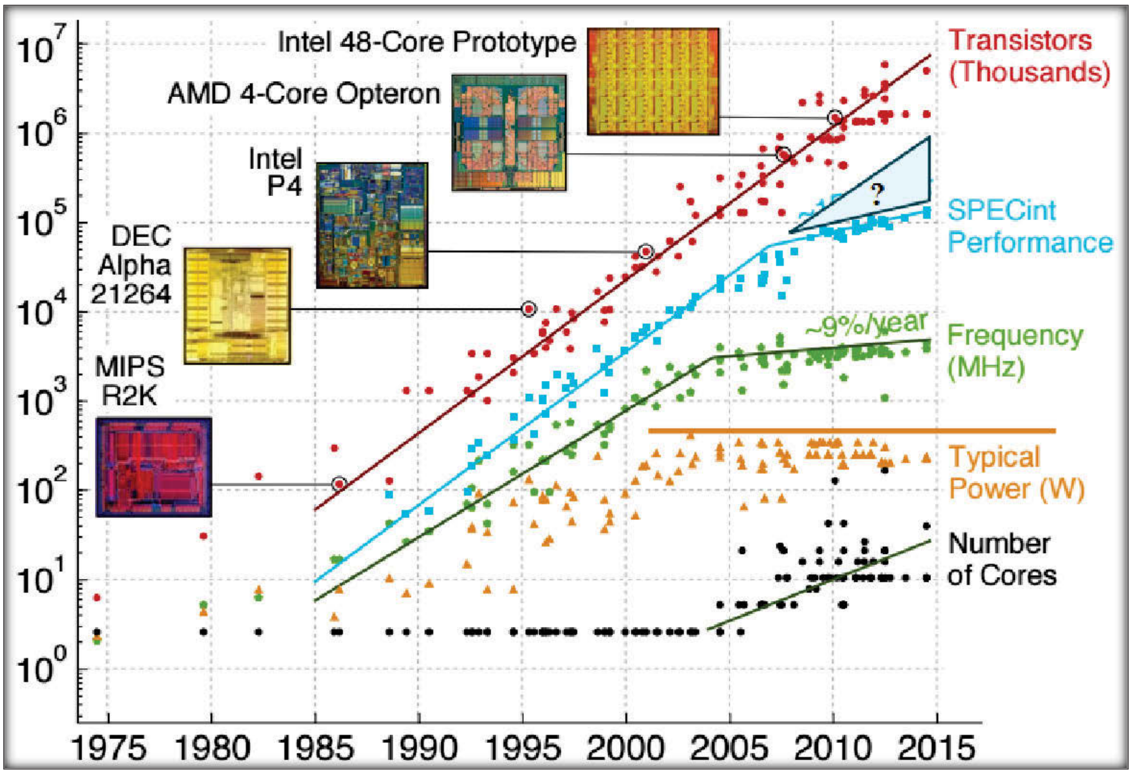


Figure 1.4 Processors improvement (Batten 2016)

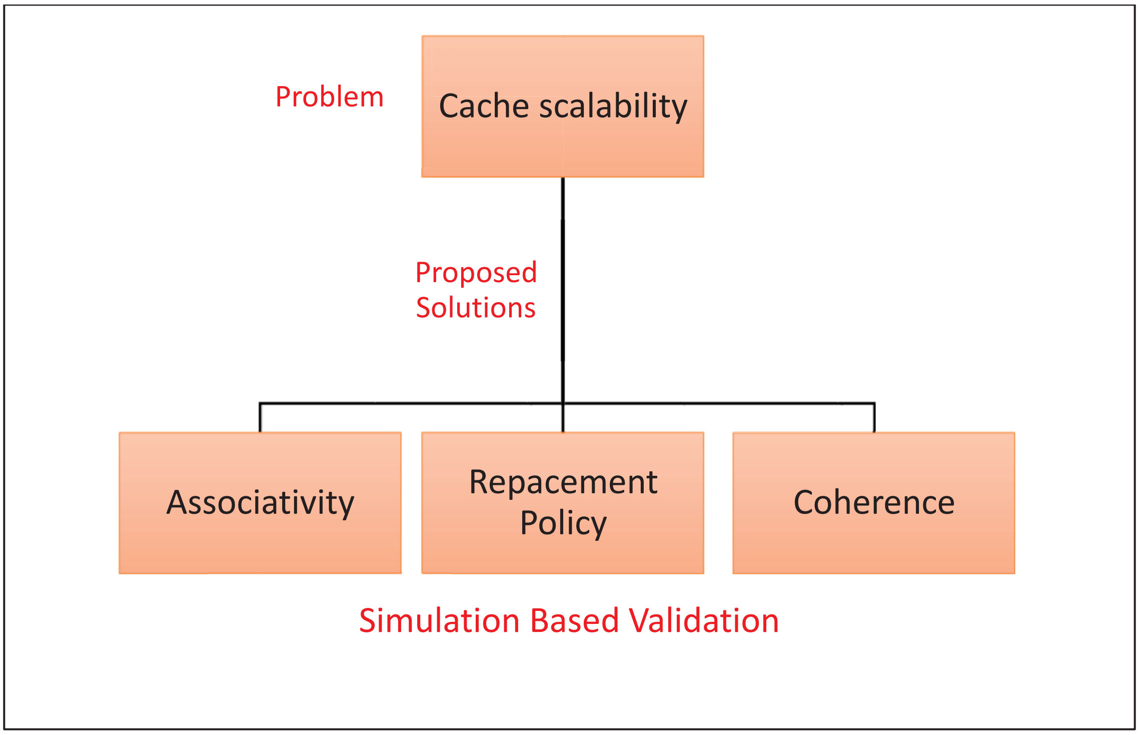


Figure 1.5 Research Hypothesis

1.5. Contributions to Knowledge

The value of this research will address hardware cache scalability problems in CMPs. This thesis will introduce three main contributions and techniques that enable scalable cache hierarchies that can be shared efficiently, and improve the performance and efficiency of modern CMPs with tens of cores – furthermore enabling cache hierarchies to scale to several hundreds of cores efficiently.

The contribution of this research should be perceived as:

- Investigating the simulation tools that focus on cache evaluation in chip multiprocessors.
- Evaluating different cache coherence mechanisms and understanding the cache scalability issues.

- Implementing a design of cache that delivers high associativity to lessen conflict misses at little cost with an adaptive replacement policy.
- Proposing a coherence directory protocol to provide the delusion of one level of shared memory, resulting in caches that appear transparent to software without the need of plentiful space, too much bandwidth, or increasing the coherence protocol complexity.

1.6. List of Publications

- Al-Manasia M., Chaczko Z. & Ounzar A. 2016, **'PROI: Block-based Coherence Bypass Protocol'.** Submitted.
- Al-Manasia, M., Chaczko, Z. & Ounzar, A. 2016, **'AHRC: An Optimized Cache Associativity'**, *IEEE 18th International Conference on High Performance Computing and Communications (HPCC 2016),* IEEE, Sydney, Australia Dec. 12-14, 2016, pp. 811-7. ISBN (978-1-5090-4297-5/16).
- Al-Manasia, M., & Chaczko, Z. (2015). **'Evaluation of Cache Coherence Mechanisms for Multicore Processors'**. In *Computational Intelligence and Efficiency in Engineering Systems* (pp. 307-320). Springer International Publishing. ISSN (1860-949X). ISBN (978-3-319-15719-1).
- Al-Manasia, M., & Chaczko, Z. (2015). **'An Overview of Chip Multi-Processors Simulators Technology'**. In *Progress in Systems Engineering* (pp. 877-884). Springer International Publishing. ISSN (2194-5357). ISBN (978-3-319-08421-3).
- Al-Manasia M. & Chaczko Z., **'Simulation of Manycore Computer System Architecture'** In Proceedings of the 23rd INTERNATIONAL CONFERENCE ON SYSTEMS ENGINEERING ICSEng, August 19-21, 2014, Las Vegas, NV, USA.

- Al-Manasia M. & Chaczko Z., **‘A Survey of Computer System Architecture Simulators, Case Study: Sniper’**, In Proceedings of the 2nd Asia-Pacific Conference on Computer-Aided System Engineering, APCASE 2014, 10th – 12th February 2014, South Kuta, Indonesia, page(s) 014-015, ISBN 978-0-9924518-0-6.

1.7. Outline of Thesis

As Table 1.1 on pages 16-17 shows, the thesis is divided into two main parts:

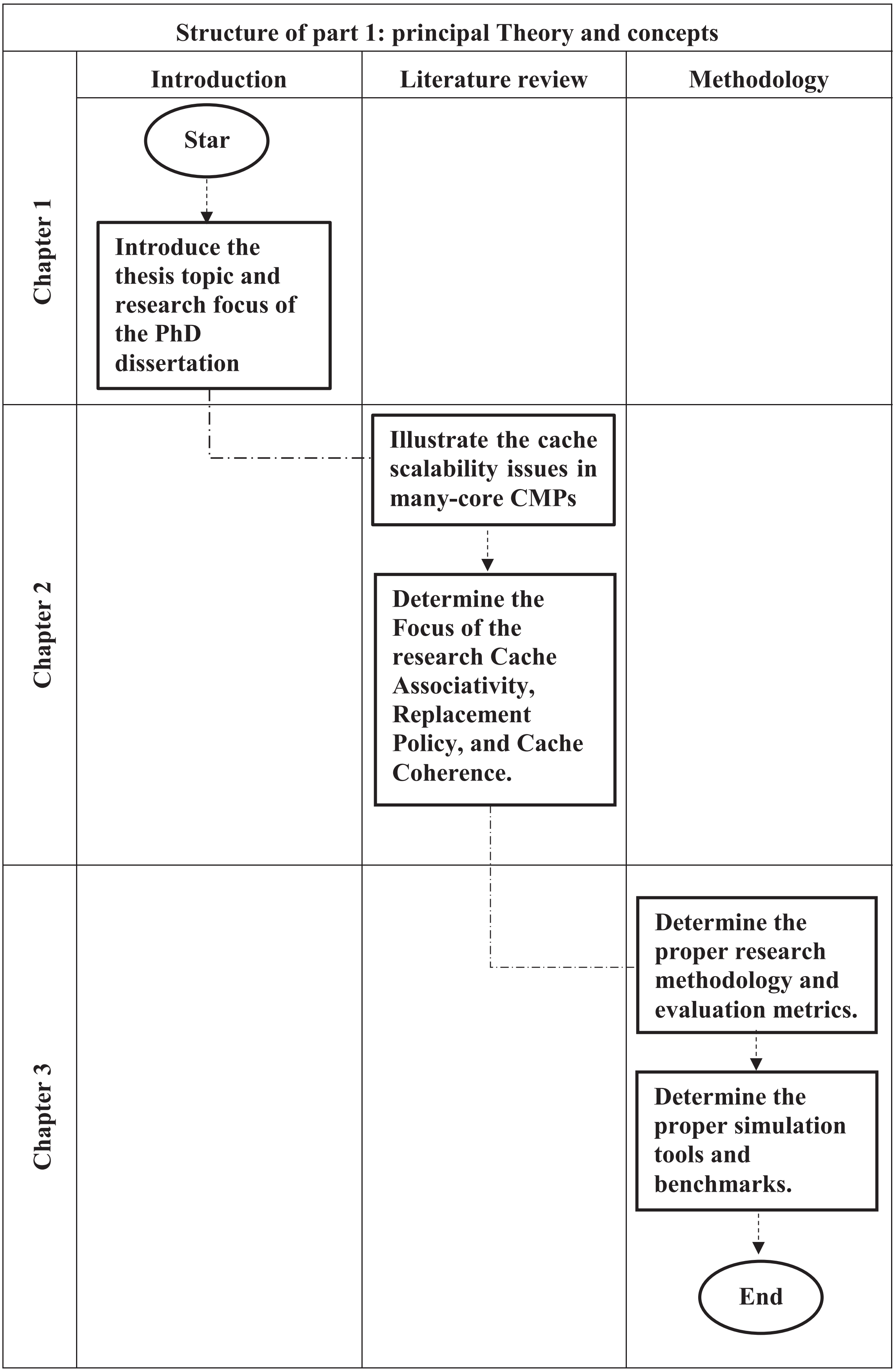
- Inaugurate the aims, goals and formulating the hypothesis.

This section is sub-divided into three chapters that explain the stated objectives and expected contribution of this study. Moreover, a presentation of the background on the cache associativity, replacement policies, and cache coherence concepts in CMPs systems and an overview of prior solutions for cache scalability. Finally, a discussion of appropriate simulation tools, performance metrics, methodology, and workloads used for evaluation have been handled in detail.

- Regulating the approaches and techniques to validate the hypothesis.

The second part covers four chapters, providing an action study of coherence mechanisms that insight the different cache coherence approaches in the CMP era. Then, it is focused on research contributions which include approaches and novel ideas that explain the experimental process needed to validate the thesis aims and hypothesis.

The final discussion and future work of the thesis are explained and offer reflections on the research in the last chapter along with the bibliography.



Structure of part 1: principal Theory and concepts

Introduce the thesis topic and research focus of the PhD dissertation

Illustrate the cache scalability issues in many-core CMPs

Determine the Focus of the research Cache Associativity, Replacement Policy, and Cache Coherence.

Determine the proper research methodology and evaluation metrics.

Determine the proper simulation tools and benchmarks.

Structure of part II: Contribution to Research		
	Evaluation action study	Experiments
Chapter 4	Star Analysis and evaluation of coherence scalability in CMP. Comparing the different cache coherence approaches in CMP.	First experiment simulations context aims to describe the need for optimising a scalable coherent cache regarding area, complexity, energy consumption and performance.
Chapter 5		Implementing an Adaptive Hashing and Replacement Cache (AHRC). A design that has the ability to maintain high associativity with an advanced method of replacement policy.
Chapter 6		PROI, a scalable coherence protocol able to distinguish between the private and shared data on block granularity level. PROI can scale with the raised number of cores and reduce area, energy, and performance associated costs with coherence mechanism.

Table 1.1 Thesis Part I and part II Structure

The thesis is divided into three main parts. The details of the thesis outline are illustrated below:

Part I: Principal Theory Concept

Chapter 1: Introduction

Pages: 2 - 20

Chapter 1 aims to discuss the research topic along with the research theme. The chapter explains the research questions and analyses the hypothesis. Moreover, it elaborates the expected outcome which is reflecting the thesis aim to answer the research questions and validate the hypothesis.

Chapter 2: Cache Scalability

Pages: 21 - 52

The literature review provides a comprehensive review of cache associativity, replacement policies, and cache coherency concepts. It discusses the issues in cache scalability along with describing the needs for designing an optimised and scalable CMP cache system that is practically necessary for future many-core system architectures.

Chapter 3: Research Methodology and Justification

Pages: 53 - 77

Chapter 3 depicts the details of the methodologies and approaches needed to implement the simulations of the experimental work. The evaluation of appropriate simulation tools, performance metrics, methodology, and workloads used for evaluation have been examined in this chapter.

Part II: Contribution to Research

Chapter 4: Modelling and Evaluation of Cache Coherence Mechanisms for CMP

Pages: 79 – 96

Based on the literature review, explained in Chapter 2, this research action study tries to compare the different cache coherence approaches in CMP. The context aims to describe the need for optimising a scalable coherent cache – regarding area, complexity, energy consumption and performance.

Chapter 5: AHRC: An Optimised Cache Associativity

Pages: 97 - 118

Chapter 5 describes a new design of cache associativity and replacement that is called Adaptive Hashing and Replacement Cache (AHRC). This design has the ability to maintain high associativity with an advanced method of replacement policy.

Chapter 6: PROI: Block-based Coherence Bypass Protocol

Pages: 119 - 150

The chapter describes PROI, a scalable coherence protocol that is able to distinguish between the private and shared data on a block granularity level and employs a dynamic writing policy with self-invalidation and self-downgrade for each L1 cache. PROI can scale with the raised number of cores and reduce area, energy, and performance associated costs with the coherence mechanism.

Chapter 7: Conclusion

Pages: 151 - 158

The chapter concludes the major findings resulted from this thesis proposals approaches. Moreover, it depicts the future work that can be performed based on the conducted research.

Part III: Bibliography and Publications:

Chapter 8: Bibliography

Pages: 159 - 185

Chapter 8 reports all bibliographies that have been referenced in this research dissertation.

Chapter 2 **Cache Scalability**

Figure 2.1 presents the high-level mapping of the literature review contents. The chapter reviews previous studies related to the field of cache associativity, replacement policy, and cache coherence. The literature review highlights the benefits and challenges of the existing methods and reflects the possible solutions for applying more optimised scalable cache for many-core CMPs. As demonstrated in Table 1.1 on pages 16-17, the chapter provides the fundamental information needed for elaborating the research methodology design and contributions to the knowledge of the thesis.

2.1. Cache Associativity

Set-associated arrays are useful in building caches and coherence directories. In these arrays, one can use multiple ways to determine cache associativity. Cache associativity refers to the ability to choose a better substitute candidate. Caches which are highly associative are usually preferred when reducing the number of conflict misses. However, an increase in alternatives will increase the cache latency and energy, and, therefore, impose a stringent trade-off on the cache design.

Here , the costs of associativity are illustrated by showing the area, hit latency, and hit energy of a set-associative cache as the number of ways scales from 1 to 32. Optimisation of this cache for latency × area × energy is done using CACTI 6.5 (Muralimanohar, Balasubramonian & Jouppi 2007). It further consists of a set-associative tag array with 64-bit tags and a data array with 64-byte lines. Lookups are sequential: first, all the tags in the set are read and compared with the lookup address; if one of them matches, the line is retrieved from the data array.

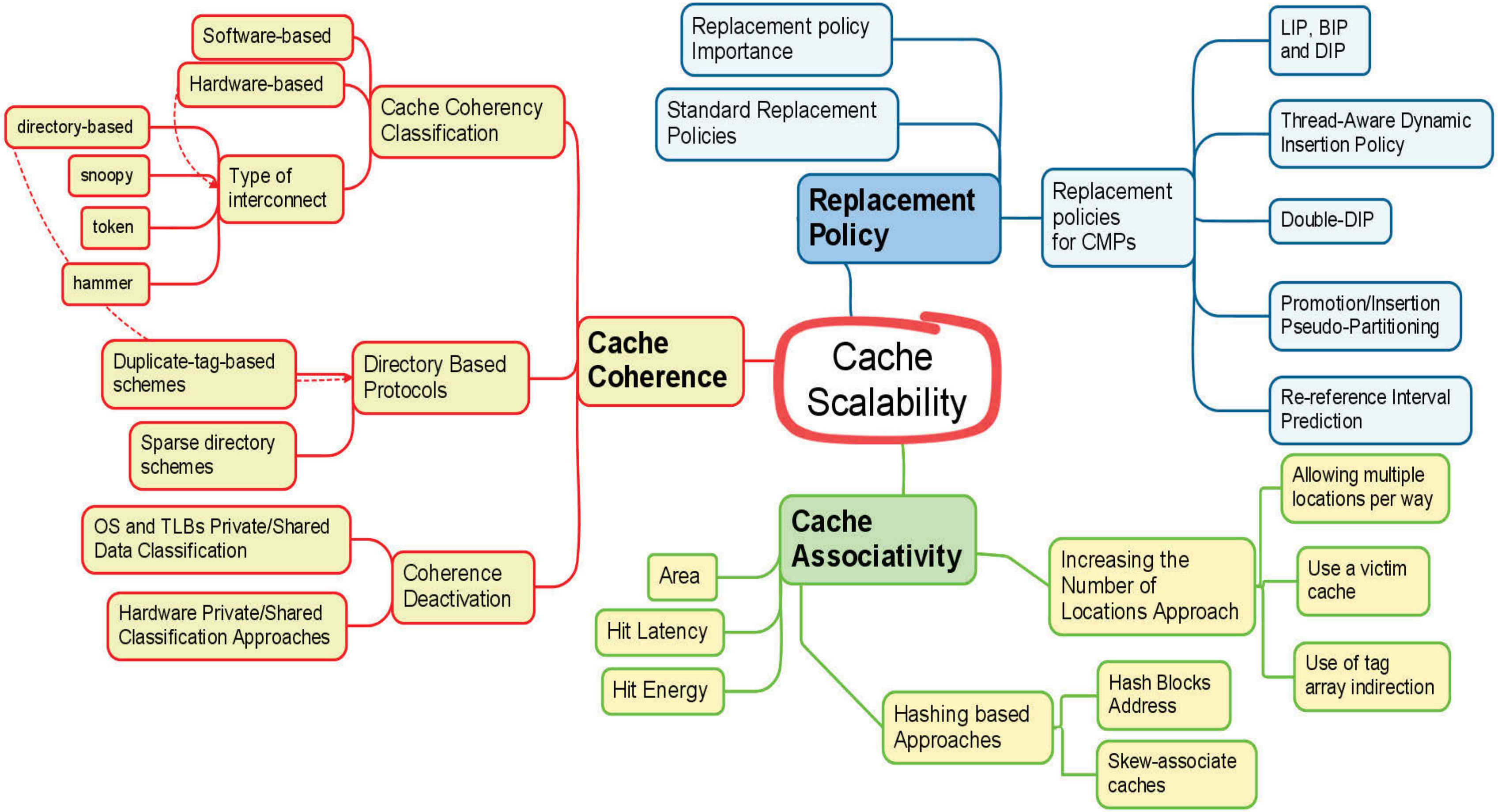

Cache Scalability
Replacement Policy
Replacement policy Importance
Standard Replacement Policies
Replacement policies for CMPs
LIP, BIP and DIP
Thread-Aware Dynamic Insertion Policy
Double-DIP
Promotion/Insertion Pseudo-Partitioning
Re-reference Interval Prediction
Cache Coherence
Cache Coherency Classification
Software-based
Hardware-based
Type of interconnect
directory-based
snoopy
token
hammer
Directory Based Protocols
Duplicate-tag-based schemes
Sparse directory schemes
Coherence Deactivation
OS and TLBs Private/Shared Data Classification
Hardware Private/Shared Classification Approaches
Cache Associativity
Area
Hit Latency
Hit Energy
Increasing the Number of Locations Approach
Allowing multiple locations per way
Use a victim cache
Use of tag array indirection
Hashing based Approaches
Hash Blocks Address
Skew-associate caches

Figure 2.1 Literature review high-level mind map

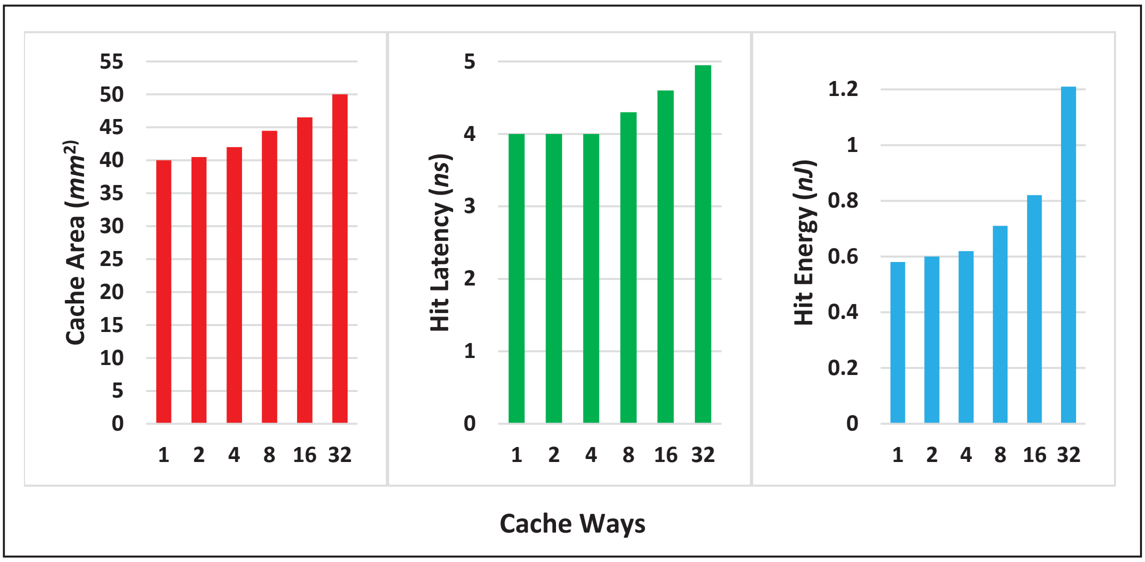


Figure 2.2 Area, hit latency and hit energy factors of an 8MB set-associative cache array with 1 to 32 ways

It is evident that caches, which are highly associative, carry a considerable cost. Many reasons contribute to this scenario since as the options are increased, one needs to read and compare more tags in parallel, which needs ports that are wider, added circuitry, and extra latency and energy. It is necessary to note that the difference is small with minimal ways; it may become larger than eight ways. For instance, with 32 ways, each hit must read 256 bytes of tags, which is 4 times the amount of data – read per the cache line size.

It is regrettable that at some point of scaling up the system, two tendencies aggravate the associativity vs. efficiency trade-off. On the one hand, limits on bandwidth, high latency, and high power of memory accesses increasingly need high associativity to limit the amount of off-chip accesses. Conversely, transferring towards many high energy-efficient cores implies that an appreciably excessive quantity of on-chip energy is consumed in the memory pyramid. In turn, this makes cache energy productivity increasingly more necessary.

In previous studies, researchers focused on fixing poor efficiency of set-associative arrays with substitute applications of highly associative arrays. Despite that, the most suitable strategies rely on the surging quantity of areas in which a block can be located. For instance, there are numerous locations in each way (Agarwal & Pudar 1993; Calder, Grunwald & Emer 1996; Rolán, Fraguela & Doallo 2009), victim caches (Basu et al. 2007; Jouppi 1990b) and more ranks indirection (Hallnor & Reinhardt 2000; Qureshi, Thompson & Patt 2005)). Increasing the quantity of probable positions of a block in the end upsurges the latency and energy of cache hits, and many of these schemes are more complicated than the conservative cache arrays that require, for instance, heaps (Basu et al. 2007), hash-table like arrays (Hallnor & Reinhardt 2000) or predictors (Calder, Grunwald & Emer 1996)). Otherwise, proper hashing can be useful in cache indexing, distributing the accesses, and averting worst-case access patterns (Kharbutli et al. 2004; Seznec 1993). Even if hashing based schemes assist in enhancing performance, the wide variety of places that contain a single block consistently impedes them.

It is worth to note that the implementation of associative lookups is considerably expensive; as such, the local store can be the appropriate option. Local stores are not associative, thus making them simpler, faster, and more energy-efficient than caches, however, they must be explicitly manageable either through using the programmer or the compiler, this is complex to implement effectively; for instance, those with large memories are complex thus making the processes of composing the software to be harder. In addition, it requires more instructions or direct memory accesses (DMAs) to facilitate moving contents of the local store. Specialised architectures target well-structured programs including streaming processors thus working well with a software-managed hierarchy (Kelm et al. 2009; Khailany et al. 2001). In addition, caches are a better fit for multipurpose CMPs. It is necessary to note that, in small-scale CMPs, previous research indicate that cache hierarchies achieve similar or better performance and better energy usage than the local stores when using popular optimisations like prefetching (Leverich et al. 2007).

In addition to simply increasing the ways of checking cache, and their parallel checking processes, there should be adequate previous operations to enhance the associativity. As such, they majorly depend on either making use of the hash function to enhance distribution of the cache accessibilities or increase block using the number of areas that can hold a block.

2.1.1. Hashing based Approaches

Hash Blocks Address: In computing the index, better hash functions on the address can be used rather than using subsets of blocked address bits to acts as a cache index. This is because hash functions are pathological, through spreading out to access patterns including the stride access that maps in similar sets. Moreover, hashing also confer slight improvement of access latency, and power overheads because of circuit addition. It also ensures addition of overheads in tag store because the tag is required to store the full block address. Simple hash functions are perceived to work effectively (Kharbutli et al. 2004), and the technique has been implemented in some commercial processes, as their last cache (Shah, Barren, et al. 2007).

Skew-associate caches: this technique use different hash functions to index computations (Seznec 1993). It uses unique block address that conflicts with the neighbouring set of blocks, where the fixed ones spread the conflicts to the other side. The Skew-associate caches result to reduced conflict misses, although its utilisation is higher than set-associative cache within a similar condition (Bodin & Seznec 1995). The technique breaks the set concepts, which makes it difficult to implement replacement policies since they are dependent on set ordering, such as approximating LRU using pseudo-LRU.

2.1.2.Increasing the Number of Locations Approach

Allowing multiple locations per way: This is also referred to as column-associative caches (Agarwal & Pudar 1993) since they allow the extension of direct-mapped caches for a block to be based into two locations depending on the two hash functions including the primary and secondary ones. The second location is checked by the Lookups, especially if the first location is a miss or rehash bit witness that a particular block among others in the set is in the secondary side. A hit in the secondary location results to the swapping of both secondary and primary locations, which improves the access latency.

However, the extension scheme assists in forecasting which location to investigate first has higher associativity (Zhang, Zhang & Yan 1997), the one to investigate first (Calder, Grunwald & Emer 1996), and the ones that identify unused set of caches to utilise them in storing the used schemes (Rolán, Fraguela & Doallo 2009). Among the challenges of several locations per way, include reduction to cache bandwidth because of multiple lookups, variable hit latencies, and requirement of more energy to swap the hits.

Use a victim cache: A victim cache is a fully, highly, and small associative cache that provide the storage of evicted blocks from the main cache until the time they are re-referenced or evicted (Jouppi 1990b). This helps in avoidance of conflict misses to the blocks that are re-referenced for a short time, although it is problematic in numerous hot ways with sizable conflict misses (Brehob 2003a). In order to demonstrate the concept, Scavenger (Basu et al. 2007) splits the cache space twice in equal large parts. He then organised the conventional set-associative cache and fully associative victim cache, as a heap. If the misses in the main cache is rare, the victim cache designs apparently work well. The miss in the main cache introduces energy consumption and additional latency in order to ensure the victim cache is checked regardless of whether the cache holds the block or not.

Use of tag array indirection: Another strategic alternative is implementing the tag and data arrays in a separate scope. This makes the tag array to become highly associative and adopt the pointers to non-associative data arrays. In this case, the Indirect Index Cache (IIC) (Hallnor & Reinhardt 2000) uses the hash table to implement the tag arrays by opening the chained hash for increased associativity. In addition, the V-Way cache (Qureshi, Thompson & Patt 2005) helps in implementing the tag arrays that are conventional set-associative, although it enlarges more than the data array in order to minimise the conflict misses. Unfortunately, the tag indirection concept suffers from increased hit latency because it is a requirement to serialise the access to the data and tag arrays. The IIC and V-Way cache have around 2×overhead in tag arrays, although the IIC contains a different hit latency.

Several elements of both designs increase the cache associativity and give proposals to the new replacement policies where some of them are specifically designed for the proposed model (Basu et al. 2007; Hallnor & Reinhardt 2000; Qureshi, Thompson & Patt 2005; Seznec 1993). Such aspects make it harder in determining the improvement metrics because of higher associativity, as well as better replacement policies. This leads to a conclusion that associativity and replacement policies are different aspects, but both focus on addressing the associativity.

2.2. Replacement Policies

This section discusses the recently proposed replacement policies to manage shared caches in CMPs and the details required for the cache design changes.

2.2.1.Overview

Caches are important in reducing the delay of memory accesses. This is because they provide the temporary fast-access storage unit for the data accessed by the cores. On a cache miss, data originates from the memory and is placed in a corresponding location in the cache. In the set-associative cache, the replacement policy must consider where to place the fetched cache block in the set. Accordingly, efficient cache replacement policy is always helpful because it reduces the cache miss rate, and, therefore, minimise the possibility of many cycles due to slow memory accesses.

In practice, the Least Recently Used (LRU) Policy is the most common policy for all levels of caches (Moinuddin et al. 2008; Qureshi et al. 2007). The LRU policy and L1 cache works perfectly since the latter benefits significantly from temporal locality because of its direct interface with the core. Moreover, the LRU policy matches the simplicity needed in the L1 cache design (Rajan & Ramaswamy 2007).

Unfortunately, references to L2 caches and last level caches lack temporal locality because the recently referenced lines are found in the L1 cache and are not requested from the upper levels again (Moinuddin et al. 2008). LRU causes the deadlines (aka zero-reused lines) (Xie & Loh 2009) to remain in the cache until they travel all the way from Most Recently Used (MRU) position to the LRU position, and only then they are evicted (Qureshi et al. 2007; Xie & Loh 2009).

Moreover, for memory intensive workloads having sets larger than the L2 cache, the LRU will cause thrashing. If the LRU policy is used for these workloads, the lines inserted in the cache will be referenced in the future, but because of capacity misses, new lines will replace them before being re-referenced. In such a case, the LRU causes a zero hit for the memory intensive workloads because all the cache lines are zero-reused lines (Moinuddin et al. 2008; Qureshi et al. 2007).

The design objective in L2 and LLC focuses on minimising the chance of lengthy accesses to upper storage memory that would be caused by a cache miss. Besides, the LLCs are characterised by large sizes and higher associativity (Rajan & Ramaswamy 2007). All these factors influenced recent studies' attempts to find optimised replacement policies capable of working better than LRU poor performance with last level caches (Megiddo & Modha 2004; Moinuddin et al. 2008; Qureshi et al. 2007; Qureshi et al. 2006; Rajan & Ramaswamy 2007; Subramanian, Smaragdakis & Loh 2006; Zebchuk, Makineni & Newell 2008). Many researchers were motivated by the ability to optimise replacement policies to gain greater control over the cache. In their studies, the researchers extended their studies to consider the problem of shared cache management in CMPs (Jaleel, Hasenplaugh, et al. 2008; Kron, Prumo & Loh 2008; Xie & Loh 2009).

In all these replacement policies, one must take into consideration multiple design issues when choosing. First, any policy that is implemented should be capable of working efficiently regardless of the type of workload. Second, it is important to ensure that any additional hardware requirement is kept to the minimum amount possible. For instance, its performance must not be below the performance of the LRU for different types of workloads. Lastly, the policy should result in negligible changes to the original cache design (Qureshi et al. 2007; Subramanian, Smaragdakis & Loh 2006).

2.2.2. Standard Replacement Policies

Belady (Belady 1966) suggested the optimal replacement policy (OPT) which provided an upper limit on the hit rate that can ever be achieved. According to the OPT policy, the right candidate for replacement is the one accessed furthest in the future. In order to implement this policy, one must know the lines to be accessed in the future and when the access will take place. Accordingly, this policy is never implemented.

The LRU, Least Frequently Used (LFU), random replacement, and First-In-First-Out (FIFO) are the most common replacement policies. However, the LRU policy is the standard replacement policy for modern on-chip caches. For a long time, there has been little or no improvement in replacement policies simply because of two major reasons: First, there is an assumption that the LRU policy works perfectly, yet this is the case for L1 caches only. Second, the proposed solutions need significant additional hardware (Subramanian, Smaragdakis & Loh 2006).

Given that LRU policy works poorly with last level caches prompted many studies attempting other optimised solutions that can achieve performance levels closer to OPT while maintaining low hardware overhead. Qureshi et al. (Moinuddin et al. 2008; Qureshi et al. 2007) gave suggestions of anti-thrashing policies that work well for memory intensive workloads. In addition, he proposed a dynamic policy with the ability to choose between the LRU or the anti-thrashing policy based on the workload.

Other researchers who proposed policies that combine both LRU and LFU included Subramanian et al. in (Subramanian, Smaragdakis & Loh 2006) and Megiddo and Modha in (Megiddo & Modha 2004). The policy proposed by Qureshi et al. (Qureshi et al. 2006) takes a replacement policy that exploits the Memory Level Parallelism (MLP) so that the lines likely to result in misses with the high MLP cost are the one that are least probably to be evicted. The policy brought forward by Rajan and Govindarajan (Rajan & Ramaswamy 2007) mimicked the optimal policy of Belady (Belady 1966) to produce the shepherd cache. In order to achieve lower overhead, Zebchuk et al. (Zebchuk, Makineni & Newell 2008) introduced changes to the shepherd cache.

2.2.3.Recent Replacement Policies for CMPs

In recent studies, replacement policies are used for the management of shared caches in CMPs. A huge proportion of the suggested policies are optimisations to uniprocessor policies. Qureshi et al. adjusted the dynamic policy to enable it select replacement policy for every core. Kron et al. (Kron, Prumo & Loh 2008) advocated the idea of promotion policies in combination with dynamic policy suggested in (Qureshi et al. 2007) so as to create a policy combining both dynamic replacement and promotion.

(Xie & Loh 2009) Xie and Loh tabled a policy that implicitly partitions the shared cache among cores using the insertion and promotion policies.

Replacement policies are used in managing shared caches in the CMPs. The option of explicit partitioning the shared caches is undesirable because it is less flexible. When using replacement policies for shared caches, the main goals should focus on achieving low hardware overhead, high performance, and scalability to the number of cores (Jaleel, Hasenplaugh, et al. 2008). In this section, the main discussion focuses on the proposed replacement policies used in shared caches in CMPs.

2.2.3.1. LIP, BIP and DIP

Qureshi et al. (Qureshi et al. 2007) proposed LRU Insertion Policy (LIP), Bimodal Insertion Policy (BIP) and Dynamic Insertion Policy (DIP). First, in LIP, all incoming lines are placed in the least recently used position in the recency stack. The reason for this action is that if any cache line is not going to be re-referenced in the future, they will not get a chance to pollute the cache. Thereafter, the line gets promoted to the most recently used (MRU) whenever it gets a hit while being at the LRU position. The LIP may retain some cache lines in the top of the recency stack even when they have stopped contributing to cache hits. In such a case, the LIP does not work well for all the workloads.

Second, the BIP is a hybrid of the LRU and LIP. The BIP places the incoming line in MRU position infrequently with a certain probability. The BIP guarantees protection against thrashing, but LRU-friendly workloads perform badly when this policy is implemented. For this reason, it is ideal to use the DIP.

Lastly, the DIP known to compete with the performance of the other policies in a dedicated set, before applying the winning policy to the remaining sets referred to as follower sets. Accordingly, the DIP works effectively among the other options.

2.2.3.2. Thread-Aware Dynamic Insertion Policy (TADIP)

In shared caches, the LRU is known for its poor performance because it serves cache accesses from different cores based on demand. In this case, a cache line requested by a core resides in the cache until its status changes to the least recently used even though it does not contribute to any hits during its residency in the cache.

Jaleel et al. (Jaleel, Hasenplaugh, et al. 2008) examined the allocation of cache resources based on benefit rather than demand. Workloads that do not benefit from more allocated cache resources are known as cache thrashing or streaming workloads. The workloads that benefit by contributing to hits are known as cache friendly workloads. Using cache friendly workloads and the LRU is recommended because these workloads benefit from more allocated resources; they should be served based on their demand. With respect to cache thrashing workloads, using BIP is recommended (Qureshi et al. 2007), because it helps in solving the thrashing problem as well as limit the resources allocated to these workloads.

Straightforwardly augmenting the DIP, as proposed in (Qureshi et al. 2007), for the shared cache is not adequate since it will pick either LRU or BIP for each of the workloads. It does not serve every workload with the strategy that best suites it. For that, Qureshi et al. proposed the Thread Aware Dynamic Insertion Policy (TADIP). The TADIP approach picks either LRU or BIP for every workload exclusively. In this way for N cores contending on the common cache, N-bit string is utilised to set the arrangement for every workload: LRU=0, BIP=1. To locate the best values for this N bit string, one arrangement is to execute all the 2N blends on the workloads and pick the string that yields the best execution. In any case, this technique is illogical; the yield shifts for various number of cores and diverse sorts of workloads and distinctive data sources. Another technique which can be utilised for little number of cores is utilising Set Duelling (Qureshi et al. 2007) with the goal that sets are devoted to every string for an aggregate of 2N sets; the string that accomplishes the best execution is utilised to choose the utilised strategies. This approach is not scalable and misutilise the cache.

To take care of this issue, Jaleel et al. (Jaleel, Hasenplaugh, et al. 2008) proposed two methodologies that pick the best arrangement for every core autonomously. In the initially proposed approach: TADIPIsolated, a policy selection (PSEL) counter is apportioned for every core. N+1 devoted sets are required for N+1 strings: one string of 0's known as the base-line string in which all cores are set to utilise LRU, and the rest N strings have just a single bit set with the end goal that in that every string one core is utilising BIP and the others utilise LRU. At the point when a miss happens for the base-line string devoted sets all PSEL counters are increased. At the point when a miss happens in one of the other N devoted sets, just the PSEL counter of the core utilising BIP will be decremented. Consequently, the performance of every core under LRU and BIP is utilised to update its counter autonomously. However, sometimes replacements are done because of references by different cores may influence the misses for the core under every policy. In TAPID-Isolated, when measuring the misses coming about because of every policy, it's expected that every other core is utilising LRU that is not the situation.

The second approach proposed by Jaleel et al. (Jaleel, Hasenplaugh, et al. 2008) is TAPID-Feedback. 2N strings are required, 2 strings for every core: in the primary string the core is set to utilise LRU and in the second string the core is set to utilise BIP, with different cores every utilising their chosen policies. This approach permits every core to choose the proper policy taking into account the sets chosen for other cores. Both TADIP-I and TADIP-F require generally low hardware – although they increment the total complexity of the replacement policy.

2.2.3.3. Double-DIP

Kron et al. (Kron, Prumo & Loh 2008) adjusts DIP (Qureshi et al. 2007) so that it is much-improved to work for CMPs by introducing the idea of promotion policies. When the designated policy in DIP is BIP, once a block is re-referenced it is promoted directly to the MRU position. In CMPs, this can result in one core using the storage more often to push other core's blocks to the LRU positions, thus being thrown out with a higher chance.

Kron et al. (Kron, Prumo & Loh 2008) suggested a policy that does not permit a block re-referenced by BIP to be promoted directly to the MRU position. Instead, the promotion is done slowly only one step up in the recency stack every time the block is re-referenced. The policy is known as Single Step Incremental Promotion Policy (SIPP). Kron et al., also suggested the use of set duelling (Qureshi et al. 2007) to dynamically choose from two possible promotion policies; either the ordinary policy or the SIPP depending on the performance of each on their dedicated sets. In order to accomplish this, two separate PSEL counters are used; one is used for the replacement policy (IPSEL) and the other for the promotion policy (PPSEL). As suggested by Qureshi et al. (Qureshi et al. 2007), the most significant bit in PPSEL is used in selecting one of the promotion policies.

Finally, Kron et al. improved their promotion policies to include different promotion increments in another policy referred to as Dynamic Promotion with Interpolated Increments Policy (D-PIIP). Kron et al added two increments of 2 and 4 rather than have only two options for the promotional values, that is, either 1 or directly to the MRU. In this case the most significant two bits are used to figure out how much the line should be promoted. This policy allowed for medium-level increments to be used where needed. The combination of DPIIP and DIP produces Double-DIP.

Even though the Double-DIP performs better than LRU for CMPs, it is unable to select different policies for different cores based on workloads. However, it offers a better way of managing the recency stack in shared cache where many workloads are competing for the cache.

2.2.3.4. Promotion/Insertion Pseudo-Partitioning (PIPP)

Xie and Loh (Xie & Loh 2009) proposed an approach that consolidates promotion and insertion policies to verifiably partition the cache among various cores. Dissimilar to different methodologies that expressly separate the cache into fixed partition, this approach takes into consideration capacity rearrangement where parts of the cache that are not utilised by one core are used by other cores. Xie and Loh (Zebchuk, Makineni & Newell 2008) utilise a component known as Utility-based Cache Partitioning (UCP) to choose the quantity of partitions apportioned to every core. This approach utilises extra tags to register the quantity of misses that would happen for every core if it's apportioned the whole cache. Contingent upon the outcomes, it partitions the cache among the cores so as to decrease the quantity of the global hits to the minimum. As indicated by these partitions, every core is given a priority that demonstrates where its blocks ought to be inserted in the recency stack. PIPP provides faster eviction of deadlines since they are not directly promoted to the MRU as in TADIP.

Aiming to diminish the extra hardware required for UCP in the extra tags, Xie and Loh proposed the In-Cache Estimation Monitor (ICEmon). These screens are by one means or another like the devoted sets utilised as a part of duel sampling proposed in (Qureshi et al. 2007). For every core, there is an ICEmon dedicated set. ICEmons are utilised to estimate the quantity of misses every cache would require in the event that it was designated the entire cache. In each ICEmon the quantity of blocks that can be utilised by other cores is restricted by the Private Monitoring Boundary; thus, blocks from other caches can't be apportioned higher recency than this limit. PIPP performs better than TADIP in their simulations for 2 and 4 core CMPs. Nonetheless, their approach upsurges the overall complexity of the replacement policy.

2.2.3.5. Re-reference Interval Prediction (RRIP):

LRU, LIP, BIP and DIP policy keep the age count of each cache block. RRIP on the other hand, maintains Re-Reference Prediction Value (RRPV). RRIP (Jaleel, Theobald, et al. 2010) describes a replacement scheme which is closely related to Seznec's pseudo-LRU. RRIP is inspired by the observations that effective replacement algorithms, such as LRU, are difficult and expensive to implement. Additionally, LRU has no record of the frequency of how data is accessed, and does not have the intelligence to know the fact that data recently put into the cache may be useless once it is used. RRIP implements a simple mechanism based on priority. It initiates with low priority and increases once there is future data access. There is a selection of victim blocks from the lowest priority group. If no blocks have minimum priority, then there is a reduction in priority of all the blocks as the search is repeated.

All studies included simulation for their proposed approaches in comparison with the conventional LRU and sometimes with other proposed replacement policies. However, each simulation used different cache sizes, different caches associativity and different benchmarks. Therefore, this report lacks accurate performance comparison between the studied policies. A unified simulation should be implemented for all policies to compare their performance which is unfeasible given the constraints.

2.3. Cache Coherency

This section illustrates the existing cache coherence techniques, various coherence protocols, and finally how to decrease the costs related to coherence and improving cache scalability.

2.3.1. Cache Coherency Classification

This section presents some related work on existing cache coherence techniques. It will focus on the aspects most fundamental and relatable to the research in this proposal.

Approaches for solving the cache problem fall into two major classes:

1. Software-based approaches which restrain caching of shared data to when is safe to do it, either done by the programmer, the compiler or the operating system.
2. Hardware-based which guarantee coherence by adding specific hardware capable of doing so. Hardware-based cache coherence solutions can also be classified according to two dimensions (Stenstrom 1990):

A. *The type of interconnect*: when processors are connected through a shared medium (i.e. a bus), protocols may use broadcasting to enforce coherence. These protocols are called *snoopy protocols*, and they apply to small-size bus-based systems because of their scalability shortcomings. In case no shared medium is present (the total order of messages cannot be guaranteed), snoopy protocols are usually replaced by *directory-based protocols*.

B. *The type of cache-coherence policy*: in this category, there are mainly two schemes: a *write-invalidation* and a *write-update* policy. In the former whenever a processor writes into a shared memory block *X* it invalidates all the copies of the block *X* present in other caches. On the other hand, in the latter one the caches are updated with the new value of *X*. Invalidation-based schemes are more common due to the fact that they are easier to implement and also more efficient for larger cache lines size than update-based schemes. Notice, however, that update-based schemes are more efficient when accessing heavily contended lines, since subsequent cache accesses result in a hit due to the update policy.

There are only few basic protocols used in cache coherence, but which of these protocols can stay with the future multi and many core architectures. The main four protocols types are: Snooping-based protocols (Goodman 1998), Hammer protocols (Hummel et al. 2006), Token protocols (Martin, Hill & Wood 2003b), and Directory-based protocols (Censier & Feautrier 1978).

- Traditional **snooping**-based protocols need an ordered interconnect to maintain cache coherence. Unfortunately, snooping protocols can't scale in terms of area requirements and power consumption (Bardisa 2009). On the other hand, there are few other protocols based on unordered networks such as: Hammer, Directory, and Token protocols.

- **Hammer** protocols - which is used by AMD (Ahmed et al. 2002) - have problems in scalability because of the broadcast invalidation messages (Bardisa 2009), which increase the traffic cost and, therefore, the power consumption.
- **Token** protocols can avoid indirection because it drops one of the hops in the critical path of cache misses. But also this type of protocols broadcast request to all tiles on every cache miss (Bardisa 2009). Thus, the interconnect will suffer from high traffic and power consumption.
- **Directory**-based protocols have been used for a long time in traditional multiprocessors, used in chip multiprocessors such as Sun UltraSPARC T2 (Shah, Barren, et al. 2007) and Piranha (Barroso et al. 2000). Directory protocols introduce indirection to solve cache misses which require only few coherence messages (Bardisa 2009), so this type of protocol is the most scalable in terms of network traffic and power consumption. However, directory protocols will suffer from two main problems: first, the directory memory overhead because this type of protocols needs to store the sharing information which increases linearly with the number of cores. Second problem is the indirection which causes long cache miss latencies.

All of these protocols have drawbacks in terms of many-core scalability. This research will focus on the weaknesses of these protocols and use a combination of the best techniques to propose a new competitive scalable protocol.

2.3.2. Directory Based Protocols

In highly parallel systems, interconnection structures that allow greater scalability are used, which makes the snoopy-based protocols unusable. This kind of architecture, typically a Non-Uniform Memory Architecture (NUMA) organisation for a tiled architecture (CC-NUMA when it is Cache Coherent), needs a more scalable coherence protocol. Scalability concept is commonly based in directory usage. Directory is a data structure that tracks for each block a record of the block state and the actual sharers of that block – that record is known as a directory entry. When a node misses a block on its private cache it first communicates with the home directory for that block in order to find the availability of it. Any modification on the block state (e.g. a write request) must be notified to the home directory. Every transaction may take further communication in order to ensure consistency for the block. The directory is also responsible of invalidating cache blocks if needed.

Directory-based mechanisms are independent of the interconnection used. The cache coherence protocol used can be either invalidation-based or update-based or hybrid, and the cache model can also have any number of states.

One of the main problems exhibited by directory-based cache coherence protocols is in regards to the storage resources required by the directories, which are increasingly greater as the number of nodes increases. In CMPs, where area and power constraints are a critical design issue, the use of directory-based cache coherence protocols compels us to reduce at most the silicon area required by directories. A lot of works have addressed this problem from different approaches. Most of them are focused on reducing either the number of entries or the entry size of the directory while maintaining system performance. Others even propose novel directory organisations with lower area requirements.

The directories of Chip Multiprocessors Coherence can be categorised into two classes:

Duplicate-tag-based schemes (Barroso et al. 2000; Golla 2006; Zebchuk et al. 2009) are power-inefficient and area efficient since they requires high associative structures (Ferdman et al. 2011a). **The sparse directory scheme** (Gupta, Weber & Mowry 1990a) is power-efficient, although its costs obtain significant area when over-provisioning the capacity of the directory, in order to avoid the conflicts in the structures with low-associativity directory (Ferdman et al. 2011a). The sparse directory organisations are abundant since they use representation of share bit vectors that is compressed (Acacio et al. 2001; Agarwal et al. 1988a; Chaiken, Kubiatowicz & Agarwal 1991; Chen 1993; Choi & Park 1999; Gupta, Weber & Mowry 1990a; O'Krafka & Newton 1990; Simoni 1992).

On the other hand, hierarchical directory organisations use area-sufficient vector storage that is uncompressed through numerous serialised lookups (Guo et al. 2010; Wallach 1992). Unfortunately, the techniques do not address the storage capacity of the number of vectors in the directory, but only checks the size of the sharer vectors. Moreover, the sparse directories undergo various conflicts, which lead to evictions of cached blocks that are difficult to be monitored. In order to avoid performance loss and reduce the frequency of conflicts, over-provisioning of directory capacity should be implemented on existing sparse directories (Gupta, Weber & Mowry 1990a; Singhal 2008). Even though the hierarchical and compressed designs are scalable, there are excessive practical area costs for the sparse directories because of over-provisioning capacity (Ferdman et al. 2011a).

Performance penalisation derived from forced invalidations can be mitigated at expense of increasing the directory size (at least, a coverage ratio between the number of cache entries and the number of directory entries equal to one is suggested). In order to invalidate cached copies, the sharing information is used. Such information may be either precise or imprecise. In this sense, there exists many proposals aimed to shorten the sharing information, such as, for example, the use of a limited number of pointers, segment directories, chained pointers, and the use of coarse vectors among others.

On the other hand, some proposals choose to reduce the number of directory entries by combining several of them into a single one.

As observed, there are many different approaches to improve the directory-based protocols in the literature, the following are the selected recent approaches.

2.3.2.1. Duplicate-tag

Duplicate Tag is where a copy of all tags in the cache track are maintained. The increment of the number of cores is a challenge for the duplicate tag store to implement their tasks (Marty & Hill 2007a). For instance, if the architecture has 64 cores, the duplicate tag store is required to have all 64-cache locations. However, the directories, which are located in off-chip memories, are not utilised to access the requested information because it takes a bit longer when compared to accessing the data from the on-chip memory. Fortunately, the on-chip memories store the directories through the storage of duplicate tags of every cache block in the system tiles. This solution does not exploit the locality of data and the growth of processors in the architecture because of the incrementing directory size. Some architectural models implement distributed directories such as Tilera Tile64 and SGI Origin 2000 architectures.

2.3.2.2. Sparse full-map

Highly associative structures are required in duplicate-tag based directories since they grow depending with core count, and preclude scalability because of prohibition to power consumption. In order to overcome the power barrier, the sparse directories reduce directory associativity, although it requires over-provisioning of the storage area for avoidance to invalidation rates.

However, the Sparse organisation helps in reducing the directory associativity through the use of low-order tag bits in order to extend the storage indexing and increase the directory sets for reduction of associativity. There is an extension of every directory through explicit sharer information, since the action of direct entries to frames of the cache.

In traditional Sparse directories, sharers were tracked by the use of bit vectors (Censier & Feautrier 1978). These bit vectors in every directory grows linearly based on the core count, which result in quadratic growth with the increase of core counts. More than 256MB of on-chip storage is required for consumption by the use of aggregate vector-based L1 directories at 256 cores.

2.3.2.3. Coarse-grain/Limptr

Every entry in Coarse-grain/Limptr holds very view pointers of sharer. However, the requirement of more sharers by the lines leads to invalidation, downgrades of insight of future invalidation of the existing sharers (Agarwal et al. 1988a; Kurian et al. 2010). This triggers an interrupt that can be handled by the use of software (Chaiken, Kubiatowicz & Agarwal 1991). The scheme reduces area overhead, although it causes a downgrade of messages that requires additional energy and bandwidth, as well as the increment of spurious invalidation (Sanchez & Kozyrakis 2012).

2.3.2.4. Hierarchical sparse

Hierarchical directories (Wallach 1992; Yang, Thangadurai & Bhuyan 1992) implement various levels of sparse directories where every level tracks the sharers of lower levels. In this case, the number of cores determines the growth of energy and area in a logarithmical aspect (Sanchez & Kozyrakis 2012). Moreover, the hierarchical organisations articulate more lookups, especially on the hurting latency, critical path, and require coherence protocol for complex hierarchical directories.

2.3.2.5. Tagless

Tagless directories (Zebchuk et al. 2009) represent sharer sets by the use of Bloom filters. It is highly area efficient since it does not provide storage for address tags. It is not energy-efficient at 1024 cores and is area efficient, since it reduces area and energy overheads with core count. Due to false positives, the scheme needs significant modifications for the coherence protocol, thereby increasing the bandwidth overheads (Sanchez & Kozyrakis 2012). Therefore, Tagless directories provides good area scalability but it is not energy-scalable. Due to the use of imprecise information, if a sharing pattern not represented occurs, the directory commonly resorts to broadcast invalidations at the expense of higher bandwidth utilisation.

2.3.2.6. SPACE

Space (Zhao, Shriraman & Dwarkadas 2010) notes that applications demonstrate the typical limitation of sharing patterns and introduce indirection levels for the reduction of sharing storage patterns, through the usage of full bit vectors and indexed sparse directory that hold the pointers into the pattern table. The schemes are not scalable and increase the range of sharers for efficient tracking, but needs additional bandwidth (Sanchez & Kozyrakis 2012).

2.3.2.7. WayPoint

WayPoint (Kelm et al. 2010) provides a proposal for the last-level cache to adopt a sparse full map directory, by using harsh table organisation in order to reduce directory overheads. This reduces significant complexity (Sanchez & Kozyrakis 2012), as well as reducing directory coverage.

2.3.2.8. Cuckoo Directory and SCD

The Cuckoo Directory (Ferdman et al. 2011a) implements the replacement process by the use of W-ary Cuckoo hashing. The process includes inserting a new line, the incoming line replaces the existing one, which is taken out, and reinserted in the available mapping position. This happens where the unused entries are unavailable. The process is repeated until an insertion threshold is reached or until an unused entry is found. However, the last line to be taken is evicted. It roughly provides the equivalent of a fully-associative directory at the cost of a more complex insertion procedure. In order to do so, Cuckoo uses a small-associativity hash table structure whose address bits are passed through different hash functions. Hits on the directory just require one access, but replacements need many hash functions in order to obtain various candidates, attaining the illusion of a more associative cache at the expense of greater energy consumption and latency.

ZCache (Sanchez & Kozyrakis 2010) has different approach of implementing the replacement process. In the model, the possible replacement candidates are all retrieved in a bread-first fashion. The least desirable candidate is selected by the use of information from replacement policy, and then performs the eviction actions for that candidate where a new one is inserted. This process requires few moves and lookups than the process used by Cuckoo Directory (Sanchez & Kozyrakis 2012). SCD provides the best combination of features from both schemes for implementing the replacement model (Sanchez & Kozyrakis 2012). However, the flexibility increases the complexity of implementation and additional energy to move the blocks between locations. SCD is therefore complex to validate and implement (Kurian, Khan & Devadas 2012). Table 2.1 compares the charachteristics of the former cache coherence schemes.

Scheme	**Scalable area**	**Scalable energy**	**Exact sharers**	**Dir-induced invalidations**	**Extra protocol complexity**	**Extra latency**
Duplicate-tag	Yes	No	Yes	No	No	No
Sparse full-map	No	Yes	Yes	Yes	No	No
Coarse-grain/limptr	No	Yes	No	Yes	Small	No
Hierarchical sparse	Yes	Yes	Yes	Yes	High	Yes
Tagless	No	No	No	Yes	Medium	No
SPACE	No	Yes	No	Yes	Small	No
Cuckoo Directory	No	Yes	Yes	Negligible	No	No
SCD	Yes	Yes	Yes	Negligible	Medium	No

Table 2.1 Cache coherence schemes charachteristics

2.3.3. Coherence Deactivation

There are two reasons for making coherence simpler in architectures with multiple cores. The first reason is to decrease the costs related to coherence (energy, performance, area) and the second is to improve scalability. Simplified coherence further enables varied accelerator architectures, and CPUs used for a wide range of applications to effortlessly intersect under coherent joint virtual memory. To accomplish this, various recent proposals try to minimise directory cost (Alisafaee 2012; Cuesta et al. 2011; Ferdman et al. 2011b; Kim et al. 2010), while a few aim at make coherence simpler by eliminating the directory restricted access in its entirety (Choi et al. 2011; Hossain, Dwarkadas & Huang 2011; Pugsley et al. 2010; Ros & Kaxiras 2012). A general helpful tool used by such approaches is data classification into shared and private (Choi et al. 2011; Ros & Kaxiras 2012), making the best use of the coherence protocol itself by employing silent self-invalidation rather than open coherence invalidations of shared data after synchronisation (Ros & Kaxiras 2012), or causing the reduction in the size of the directory (Alisafaee 2012; Hossain, Dwarkadas & Huang 2011; Pugsley et al. 2010).

It is possible to perform data classification by the OS (Cuesta et al. 2011; Hardavellas et al. 2009; Kim et al. 2010), Compiler (Li et al. 2010; Li, Melhem & Jones 2012), or hardware methods (Hossain, Dwarkadas & Huang 2011; Pugsley et al. 2010). While gaining the lowest hardware cost, classification by compiler is not clear to software. It is compatible with software/hardware co-design but needs recompilation and recoding for legacy software and there is a noteworthy effort to find out if there will be sharing of a variable at compile time. The approach of the OS does not enforce additional needs for dedicated hardware, as there is storage of page granularity data classification along with the Page Table Entries (PTEs) (Hardavellas et al. 2009). It is thus a proper choice for intricate effective optimisations. Nevertheless, it experiences adaptation and granularity issues, causing misclassifications that augment with time, thus corrupting classification quality.

2.3.3.1. OS and TLBs Private/Shared Data Classification

Several policies employ private/shared type of data classification to make coherence simple for data-race-free (DRF) semantics (Cuesta et al. 2011; Hardavellas et al. 2009; Kim et al. 2010). It is possible to eliminate private data from the coherence process to result in the reduction of the directory size. Whereas a few approaches depend on private/shared data that the OS provides based on a page, others up hold multi-granular method of classification in hardware. The page table identifies a page as private when only one core accesses it, and when a different core accesses the same page, its status changes to shared.

The problem with classification on the page level is that there is misclassifying of cache blocks that are private as shared within shared pages. Even when different cores access different cache blocks, the page still becomes shared. The transition of a page from private to shared leads to notification of the first core that tagged the page as private, to change the page's local classification. This process is necessary as the local classification dictates the employed write policy for a page's cache blocks. The transition of a page can occur at maximum once only, and once a page becomes shared, it can no longer revert back to being private.

The fact that the majority of write misses emerge from private blocks in a write-through protocol is a major observation that makes protocols such as VIPS-M (Kaxiras & Ros 2012) realistic. It is clear from the observation that VIPS-M employs an active write policy in the L1s: write-through for shared data and write-back for non-coherence *private* data. As a further optimisation, there is employment of a write-through buffer to combine writes to similar cache blocks. The emptying of this buffer happens at similar synchronisation points that cause self-invalidation. It also employs a similar OS classification technique to differentiate between *Read-Only* and *Read/Write* pages. There is usually no self-invalidation of cache blocks of Read-Only pages at synchronisation.

2.3.3.2. Hardware Private/Shared Classification Approaches

Numerous systems depend on the classification of hardware as shared or private for different reasons, the main one being the size reduction of the directory (Alisafaee 2012; Fang et al. 2013; Zebchuk, Falsafi & Moshovos 2013). To accomplish it, it is necessary to employ a multigrain directory that monitors coherence data on over one block size. The same is the recommendation of Alisafaee (2012) and Zebchuk et al. (2013), but their approaches vary on implementation. The system stores private areas in the directory and extracts shared cache blocks out of them. Likewise, Fang et al. (2013) approach describes private region entries with either shared or private block entries. The private ones have a different region as the owner while the shared ones have several cores accessing them.

Pugsley et al. (Pugsley et al. 2010) establish an adaptive description of their Shared, Written, Exclusivity. Level (SWEL) protocol, referred to as Reconstituted SWEL (RSWEL) in correspondence to classification adaptivity. The SWEL protocol entails a saturating decay counter that is 2-bit, for the reclassification of pages from shared to private. This is mainly because the lack of adaptive classification that permits the reclassification of data to private from shared, leads to the misclassification of all system data as shared that gets rid of optimisations related to the private data. Still, it is quite an intricate adaptation that happens periodically in bulk and needs elaborate time regulation that marks the decay counters for proper results. In addition, Zhao et al. (Zhao et al. 2013) concentrate on cache coherence adaptation by suggesting the Protozoa coherence protocol. The problem is Protozoa deals with adaptivity in data movement and coherence granularity. It adopts varying granularities for mitigating data movement transparency caused by coherence invalidations, rather than having set cache-block granularity for both coherence and data storage/movement actions. On the other hand, our approach deals with adaptivity when applied to private/shared data classification. Lastly, POPS puts forward a protocol that is most effective with shared or private data to satisfy the exchange between a shared and private LLC, but handles migrating data as private (Hossain, Dwarkadas & Huang 2011).

Chapter 3 **Simulation of Manycore Computer System Architecture**

3.1. Research Methodology

Research methodology represents the main portion of the research body because it refers to the steps taken by the researcher to respond to the research questions (Leedy & Ormrod 2005). It is evident that in researchers involved the computer architectures area, quantitative methodology is the frequent and most advantageous methodology. The majority of engineering research looks to recognise the determination of outcomes by minimising possible causes to a distinct set of variables or indicators (Borrego, Douglas & Amelink 2009). The use of quantitative methods suits deductive approaches, where a hypothesis or concept justifies the purpose statement, the variables, and how to answer the narrowly described research questions. The wording of the research question and the hypothesis under test govern the method of data collection, together with the statistical analysis technique employed to study the data (Creswell 2013).

The method of quantitative analysis largely relies on the positivism that sustains empirical research to measure and examine fundamental links between variables in a free values framework (Sale, Lohfeld & Brazil 2002). This is because it is possible to reduce all phenomena to empirical indicators that symbolise fact. It is a fact because of the reality of a single truth and is free of human perception, which makes the investigated thing and the investigator become independent entities.

Therefore, methods of quantitative research operate with numerical data form gathered from simulated experiment or a representative sample and analysed typically by statistical methods. The main objective is to discover the independent and dependent variables, getting rid of insufficient variables, which will reduce the difficulty of the problem so that it is possible to confirm or abandon the preliminary hypothesis. On the other hand, the qualitative method looks at the method of assigning meanings and its foundation is on constructivism and interpretation, considering that there are numerous realities and truths founded the creation of a social reality that is continually changing. Consequently, there is an intertwining between the study object and the investigator in such a manner that result in the creation of discoveries mutually in the situation's context that moulds the investigation (Sale, Lohfeld & Brazil 2002).

The present study uses a quantitative approach to authenticate the research hypothesis and respond to the research questions. Therefore, this thesis carried out of a widespread literature review that led to establishing the problem and creating a preliminary research model. Subsequently, the simulation environments, suitable and testable benchmarks, and the appropriate tools for simulating and investigating the study cases were determined. Then, case study data is gathered after three times of replicating the same experiments to attain additional accurate and strong results. In the end, the results were analysed and utilised to recognise the way they mitigate the constraints of the research problem.

3.2. Simulation Tools

Computer System Architecture (CSA) simulators are used to generally form and validate fresh designs and developments of CSA. CSA simulation is important and there are feasible criteria that distinguish between different simulators of CSA. Multi-dimensional features determine the classification of CSA simulators as well as their precision, performance, flexibility and functionality. The Sniper simulator has been chosen for a closer examination and testing. It proves its capability to level to hundred cores with an extensive range of performance and functionality.

3.2.1. CSA Simulators Technology Overview

A CSA simulator that is helpful in modelling different computer components and devices to estimate the output and the level of performance provided a certain input. Simulation of a computer happens to be the basic guiding force in the research and design of computer architecture. It is because the vast effort in the design of complex micro-architecture, hardly ever modelled in a diagnostic way, causing the method of prototyping each stage in the process of design to be excessively expensive. Nonetheless, a comprehensive simulation is just as expensive since it requires intensely crafted simulators, several machine cycles, and realistic workloads. The compulsory level of detail is attainable slowly as even the fastest simulators can take a number of weeks or months to complete. The domination of the designs of multi-core processors makes the process even worse. The host chip's entire throughput rises with increasing generations but, the single-thread performance that is a basic simulation element, does not reveal any improvement due to power limitations. It is also essential to be cautious about the fact that a target-architecture model slowly becomes complex because of the increasing cores count. It makes it difficult for the cores' augmented performance to become accustomed to the increasing complexity of the target.

Researchers recognise the issue, making them suggest simulation time reduction strategies. For example, there has been an application of the sampling method by earlier works (Perelman et al. 2003; Wunderlich et al. 2003) to achieve a statistical depiction of a certain application part in order to offer the same insights in a complete simulation, but in a small period of time. Different researchers prefer making the simulation parallel to reduce the time of a single simulation through using several threads to operate it (Chen, Annavaram & Dubois 2009; Miller et al. 2010; Mukherjee et al. 2000) . It is an appropriate method for the phase of design that requires a faster assessment of one or additional design points. Conversely, in the case of many parameters, operating multithreaded simulations concurrently offers improved simulation performance (Al-Manasia & Chaczko 2015b).

The strategies of decreasing simulation time are essential for simulators that are cycle-accurate (Martin et al. 2005a; Wenisch et al. 2005). They give research fundamental elements in computer architecture to improve comprehension of the stresses associated with workload from structures of micro-architectural designs. However, the fact that they are time consuming is a drawback in the examination of chips comprising several cores. In general, cycle-accurate simulators are not proper for large scale CMP architectures (Al-Manasia & Chaczko 2015b).

Literature is already present the awareness of grain simulation; it is also notable that common simulators give some level of abstraction. Most of them are in between micro-architectural pipeline modelling, all the way to emulation and functional simulation.

3.2.2.CSA Simulators Usefullness

The essentiality in simulations comes from the fact that their understanding makes it achievable to do different developments. **Firstly**, simulation is essential in migration because it is possible to maintain old systems operationally by stimulating them. As a result, simulators can help in easier system migration from old to new software and hardware systems, thus organisations can save on time and money. **Secondly**, simulation is essential when it comes to hardware development. According to Magnusson et al., the usefulness of simulators is also in computer systems modelling despite of their availability (Magnusson et al. 1998) , which makes the hardware development process simple since the hardware could be in the process of development or still absent. Furthermore, it can improve indirect costs reduction as it minimises the need for expensive hardware prototypes (Austin, Larson & Ernst 2002). Simulation saves on cost because of the use of a cheaper hardware to simulate costly resources.

Simulation is also helpful in software development for developers of device drivers and operating systems. It can be very difficult to debug programs that use kernel mode to run but simulators have debugging benefits that are non-intrusive, which make program development simpler. The function of simulators includes the provision of security because they operate in a controlled environment. It facilitates the use of new systems by companies before actual use in the production process, which is vital to boost the general security system of a company. Lastly, simulators help in restoring the simulated computer's earlier states. It does the representation of the memory, CPU registers, program counters, processes, and other active parts that define the existing computer state. This allows the user to simulate the activities that are useful in the debugging process. Using the states of a computer also assist in evading extensive boot-ups and configurations.

3.2.3. Taxonomy of CSA Simulators

The categorisation of CSA simulators happens based on their contexts. Their **scope** is the first categorisation factor, which include Full System, Micro-architecture, Application-based, Cycle Accurate and Instruction Set simulators. The technique of micro-architecture simulation is functional in design modelling and microprocessors' performance and its elements. Conversely, full system type model features as restricted modes and copy peripheral components' function in order to give the OS some support – this creates the possibility of running complex multithreaded workloads. Simics (Magnusson, Christensson, Eskilson, Forsgren, Hallberg, et al. 2002), gem5 (Binkert et al. 2011), RSIM (Pai, Ranganathan & Adve 1997) and SimOS (Rosenblum et al. 1995) employ this method. Application-based, also known as user level simulators, highlight on the user applications. It is much easier to develop and use them because they do not need models to time the device, large imaging disk, and OS booting. However, they are only useful in supporting basic workloads. Some that use this method include Graphite (Miller et al. 2010), Sniper (Carlson, Heirman & Eeckhout 2011), ZSim (Sanchez & Kozyrakis 2013), CMP$im (Jaleel, Cohn, et al. 2008) and McSimA+ (Ahn et al.). Cycle accurate and instruction set simulators are those with microprocessor simulation as the coverage and differ in details offered by them. Instruction set simulators stress on simulation of high speed of the processor functions, while cycle accurate simulators help in the processor's accuracy of timings.

The classification of CSA simulators may depend on **workload and input** that consist of execution-driven and trace-driven simulators. Trace-driven simulators utilise sessions recorded earlier when implementing simulated applications. It entails recording of memory operations and other vital instructions during the running of an application in an environment that generates trace (Goldschmidt & Hennessy 1993). Usually this form of environment can be the actual target hardware. However, it can also develop from the generations of software in case the hardware is absent or under development. An important factor to note is that irrespective of execution simplification, there is absence of a vibrant behaviour in trace-driven simulation that exists in multi-threaded programs operating on several processors concurrently. In case of the use of a different Symmetric Multi-Processor (SMP) system to run this form of simulation, it is possible to notice such discrepancy (Dikaiakos, Rogers & Steiglitz 1994; Goldschmidt & Hennessy 1993). Execution guided by trace also has the limitation of storage and time needed in session recording (Goldschmidt & Hennessy 1993). Goldschmidt and Hennessy, (Goldschmidt & Hennessy 1993), give a broad conclusion stating that it is not proper to use trace-driven simulation in the imitation of parallel systems or those that rely on timing.

Execution-driven simulators employ applications founding on a simulated processor. It does not need traces and it is possible to conduct it in a single machine (Goldschmidt & Hennessy 1993). Where there is similarity of the target and host, the interpretation of the instructions is directly to the host. It avoids the concerns of instruction concurrency and timing. The simulator’s accuracy is the only correctness affection of the outcome from an execution-driven (Goldschmidt & Hennessy 1993). It is a highly useful technique in target optimisation because it reveals all the data in utilisation or production by the target (Austin, Larson & Ernst 2002).

Lastly, another main category in the classification of CSA simulators is **detail**. It entails timing and functional simulators. Functioning simulators pay attention to the accomplishment of similar functions as those of the modelled components. Timing simulators on the other hand, try to concentrate on the correct reproduction of timing and performance features of the relevant targets as an extra characteristic to their functionalities.

3.2.4. CSA Simulators Quality Attributes and Evaluation Parameters

There are a variety of perspectives of balancing dissimilar factors in an optimal manner for functional improvement of the simulator concerning: flexibility, accuracy, performance, level of functionality and details. The ease of access of many simulators with particular designs for limited activities founded on distinct features comes because of the complication of the maximisation all of these aspects. Nevertheless, a few simulators have a more common approach. Below are the significant conditions that can depict the abilities of the simulators from diverse aspects:

1. **Performance**

It measures the simulator’s speed in accomplishing a particular workload. It bases its measurement on slowdowns, also referred to as the host amount of instructions, executed for each simulated target instruction. The measurement of a processor’s performance is in MIPS (Millions of Instructions Per second) employed in benchmarking simulators.

2. **Functionality**

It expresses what the simulator can achieve or perform following the modification. A functional simulator can operate a good element of the software that runs on the target, and it will work properly without problems. It also entails the Instruction Set Architectures (ISAs) supported by the simulator, the number and type of workloads it can operate.

3. Accuracy

It describes how properly a simulator simulates the target activity and it characterises to what level the performance of the simulated target matches the actual system performance. It is essential to note that a simulator that is more accurate sustains extra slowdowns; therefore, it will simulate an increased degree of details. Complete accuracy reveals the closeness of the simulation to the actual system, while relative accuracy reveals the correctness of a model between varied configuration settings.

4. Flexibility

It gives a depiction of whether the simulator’s design is versatile, its capability to run from different hosts and simplicity of porting it to different hosts. Flexibility also assesses the capacity of a simulator to have expandable modular design. All these requirements confirm the simulator’s flexibility.

There are several system and processor simulators available as shown in Table 3.1. Every simulator has its own qualities and serves its various purposes properly. A simulator that simulates the full system is mostly beneficial when the imitation entails broad kernel utility support of the OS or intense I/O activities. However, these simulators are rather slow and make it challenging to isolate the effect of architectural modifications from software and hardware stack interaction. Furthermore, since they depend on existing OSes, they typically do not maintain numerous core simulations well. Nevertheless, a simulation that is purely of application level is inadequate, even though time/space sharing and I/O activity are not the focus areas. For instance, thread scheduling in a processor with multiple cores is vital for both research interests and performance accuracy. As a result, it is advantageous for such simulators to handle threads separately from the real hardware and the host OS where the simulators run. For instance, Graphite (Miller et al. 2010) employs models with less detail like the one-IPC model, to attain enhanced speed in the simulation.

Simul-ator	Year, Availability	FS/A	LoD	Simulated Platform	SS	Extensibility	Special Features
Simics	1994, commercial/ free Academic	FS/A	OoO	UltraSparc, Alpha, x86, PowerPC, MIPS, ARM	+	API provided	Simulate "everything",
gem5	2003, free BSD	FS/A	OoO	Alpha, ARM, SPARC, MIPS, POWER and x86	+	Modular /OO design, Python configuration files	Written from scratch, pervasive OO, configurability
RSIM	1997, UIUC/NCSA	FS	OoO	SPARC V9/Solaris	+	Through modular design	Simulate adv. CPUs in network
SimOS	1994, freely available	FS	Variable	MIPS/SGI IRIX, Alpha/UNIX	+	Uncertain	Simulate entire system
Graphite	2010, free	A	No	x86	+++	Python config files	Parallel simulator
Sniper	2011, free	A	OoO*	x86	+++	Python config files	Interval simulation
CMP$im	2008, free	A	No	x86	++	Python config files	Memory system simulator
ZSim	2013, free	A	OoO*	x86	+++	Python config files	Fast interpreter design
McSimA+	2013, free	A	OoO*	x86	+++	Python config files	Offering a middle ground between FS/A

Table 3.1 Summary of Existing Simulators Categorised by Features

Sniper (Carlson, Heirman & Eeckhout 2011) utilises improved abstraction techniques like interval-based simulation to achieve additional accuracy with fewer performance operating cost. Although these simulators work well during space exploration in the early design stage, they fail in their accuracy for comprehensive micro-architecture-level studies of architectures with multiple cores. Graphite (Miller et al. 2010) and Sniper (Carlson, Heirman & Eeckhout 2011) happen to be faster simulators as they employ parallel simulation to enhance the simulation speed.

It is possible to use simulations driven by trace to exchange accuracy for speed in the simulation. However, it is unsuitable to use these for multithreaded applications as the information of the real-time synchronisation is usually missing with the use of traces. Thus, the simulations driven by execution are better. Alternatively, full-system simulators represent details of both OSes and micro-architecture-level to focus more on accuracy than simulation speed. As a substitute, it is advantageous to simulate the details of a micro-architecture with multiple cores while remaining quicker, compared to full-system simulators that have both software and hardware overhead. Because simulators focus on the simulation of thousand-core chips, for the moment they have to be a user-level simulator. No existing typical OS levels to thousands of cores, and there is also a limitation of the quantity of cores by ISAs, for instance, x86's advanced programmable interrupt controllers (xAPICs) only upholds nearly 256 cores (Sanchez & Kozyrakis 2013).

Simulators have different features, weaknesses and strengths, and significance to varying research areas. Whereas most development projects only have a few years, this study regards a simulator like SimOS to be too old for active employment in research now. ZSim and CMPSim have the issue that they are not obtainable publicly at the time of this study.

Some simulators can be integrated with useful timing simulators such as: TFsim and GEMS – use a timing-first approach to simulation, where a less extensive model simulates the timing of the processor, which is then later verified by a larger and more complex execution-drive simulator, such as Simics. Every simulator can simulate architectures of CMP, but just a few can simulate symmetric multi-threading (SMT). Architects largely depend on comprehensive cycle-level simulation. In some industrial developments, architects depend on actual cycle-accurate simulation. The main advantage of cycle-accurate simulation is clearly accuracy, although it has slow speed. Industry simulators normally operate at a speed of 1 to 10 kHz while academic simulators are not really cycle-accurate in comparison to real hardware. Hence, they are naturally faster because they have simulation speeds that range in the tens to hundreds of Kilo Simulated Instructions Per Second (KIPS), like gem5 (Binkert et al. 2011), GEMS (Martin et al. 2005a) and PTL-Sim (Yourst 2007).

There are quite a few challenges associated with cycle-accurate simulators in the era of the multi core processor. First, the performance of the simulation does not increase with more core numbers because of the single-thread aspect of these simulators. Second, simulating processors using large caches is increasingly difficult because the slow speed of the simulator does not facilitate simulating large dynamic instruction counts in a practical amount of time. Sampled simulation normally presumes comprehensive cycle-accurate simulation of the points under simulation, and the achievement of simulation speed is by limiting the amount of instructions that require detailed simulation. Increased abstraction simulation techniques employ a dissimilar, and orthogonal, simulation speeding up method: processor modelling happens at a higher abstraction level. This allows the speeding up of simulation and reduction of development time and simulator complexity.

3.2.5.Simics Simulator

Simics (Magnusson, Christensson, Eskilson, Forsgren, Hallberg, et al. 2002) is a full-system simulator and it permits the presence of an operative system, giving privileged devices and modes, allowing the implementation of additional complex multi-process or multi-threaded workloads, and of I/O or network applications. The use of Simics is in running unmodified binary codes for the preferred hardware at frequencies with acceptable performance. It is accurate, extensible, and scalable. The useful of Simics comes from the fact that it lets researchers simulate any form of digital system, from a FPGA or CPU to a full frame of components.

Simics maintains numerous ISAs such as MIPS, ARM, or x86 as well as other digital devices such as digital signal processors (DSP). Among the strongest points of a Simics is scalability, allowing varied systems of architecture, hundreds of I/O devices and diverse network protocols in one simulation. Therefore, it can simulate extremely large and quite complex systems to make sure of its usability in environments of the real world. An outstanding capability is the program execution in both reverse and forward direction, permitting the solving of mistakes of bugs in a different manner is difficult to determine. The virtual hardware that Simics implements runs similar binary software as the target system. It is an aspect that allows researchers to build, debug and set up software first on the actual machine and later implement it in the virtual device. Developers can employ third-party software because it can operate production binaries. Compared to other simulators like Sniper, Simics' main disadvantage is its speed.

3.3. Case Study: Sniper

This section presents one of the most suitable simulators for large scale CMP simulations called Sniper simulator which can stabilise the trade-off that happens between accuracy and performance.

3.3.1.Overview

Sniper is an accurate and high-speed x86, parallel simulator and it incorporates an interval simulation technique and extends the general functionality of the infrastructure of a Graphite simulator; allowing for accurate and speedy simulation. Exchanging speed of simulation for accuracy facilitates options of flexible simulation by Sniper when exploring different heterogeneous and homogeneous multi-core architectures. The following sections depict a Sniper simulator's architecture, the key components, simulation configuration, result tools and various visualisation options. To reveal the way to employ the sniper simulator in a practical way, this study shows how to begin a variety of test runs, results and analysis of recorded results.

Simulation tests rely on the simulator's pre-tailored test binaries. The development of the Sniper simulator is appropriate and has a setup configuration manual. The manual provides instructions on how to start and use the device in operations. When employing the sniper simulator one can perform timing simulations for multi-program workloads and applications with multiple threads and shared memory with tens to hundreds of cores. The performance speed of this operation is higher compared to that of most existing simulators. The sniper simulator's vital feature is the core model founded on interval simulation.

Interval simulation permits one to increase the abstraction level in architectural simulation. This assures speedy simulation development and evaluation periods. The sniper simulator is validated against Nehalem and multi-socket Intel Core2 systems, thus there are average performance estimation errors within 25% at the simulation speed of up to many MIPS (Carlson, Heirman & Eeckhout 2011).

The interval core model and the sniper simulator is critical in the non-core and studies of system level requiring a lot of details than the conventional one-IPC models, but where the cycle-accurate simulators execution is too slow to allow reasonable size workloads that is possible to simulate (Carlson & Heirman 2013). An additional benefit is that the model of the interval core supports the formation of Cycles Per Instruction (CPI) stacks that depict the cycle numbers that get lost due to the varying features of the system like the cache hierarchy or branch predictor. As a result, it results in improved understanding of the impacts caused by each part on the general system performance. This allows people to employ the sniper simulator for classification of applications and the co-design of hardware and software.

3.3.2.Sniper Configuration

Sniper configuration is accessible with configuration files and command line parameters. sniper/config/base.cfg is the default configuration where sniper exists in practice. The run-sniper script accepts the parameters of the command line of –c config file and –g config setting to configure or modify the sniper simulator. The assessment of configuration files or parameters happens from the left to the right in regards to the command line. In addition, newer values dominate the previous values.

3.3.3. Simulation Results Tools

The sniper simulator generates files at the termination of a simulation process, which is the sim.cfg that has every alternative of the used configuration and the sim.out file reveals the basic statistics while the sim.stats has the entire set of each recorded statistic at the simulation's key points.

Interval simulation is unique because allows for the creation of CPI stacks that summarise the place where time is spent. The CPI stack can be described as a stacked bar that shows different components that contribute to the overall performance as shown in Figure 3.1.

In order to estimate the consumed energy of a program, the sniper simulator integrates with the McPAT power and modelling framework. A person has the option to choose plot dynamic, static or total power of the chip for every component as illustrated in Figure 3.2.

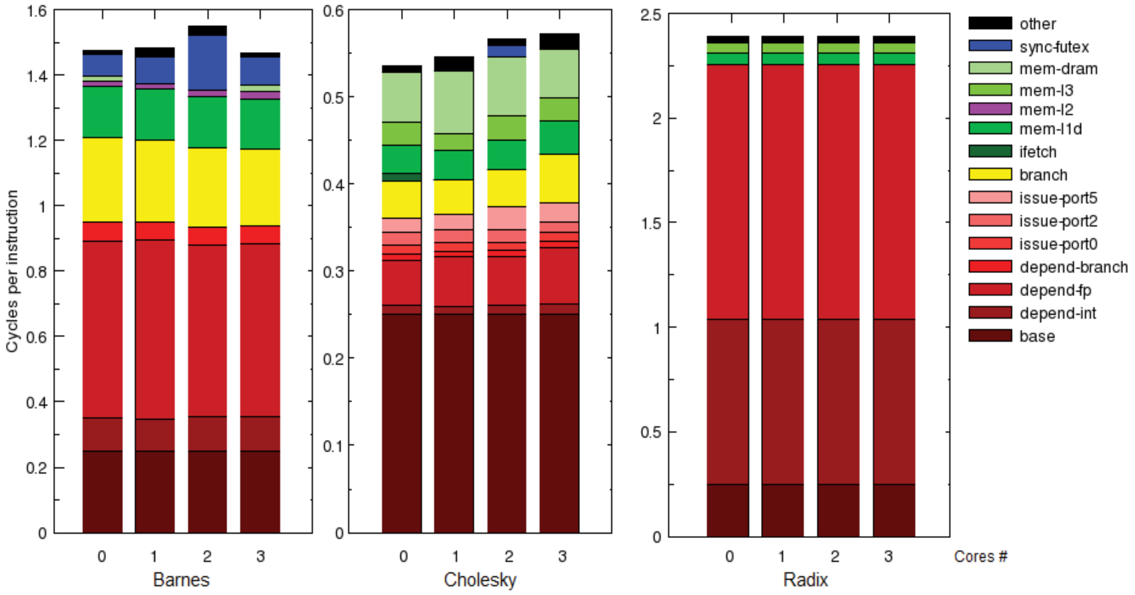


Figure 3.1 Cycles Per Instruction stack

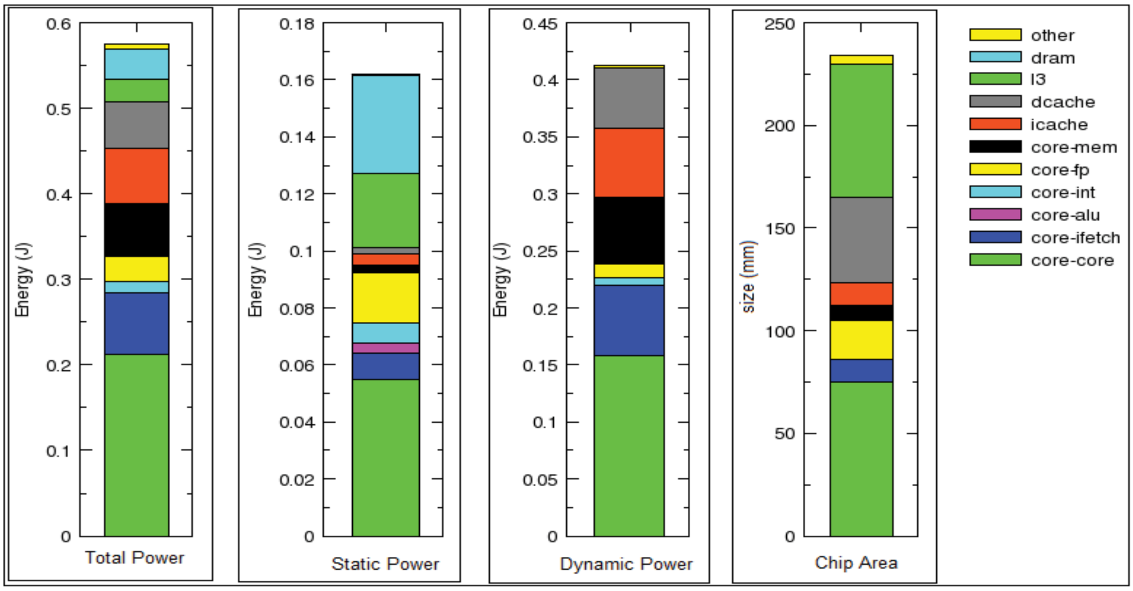
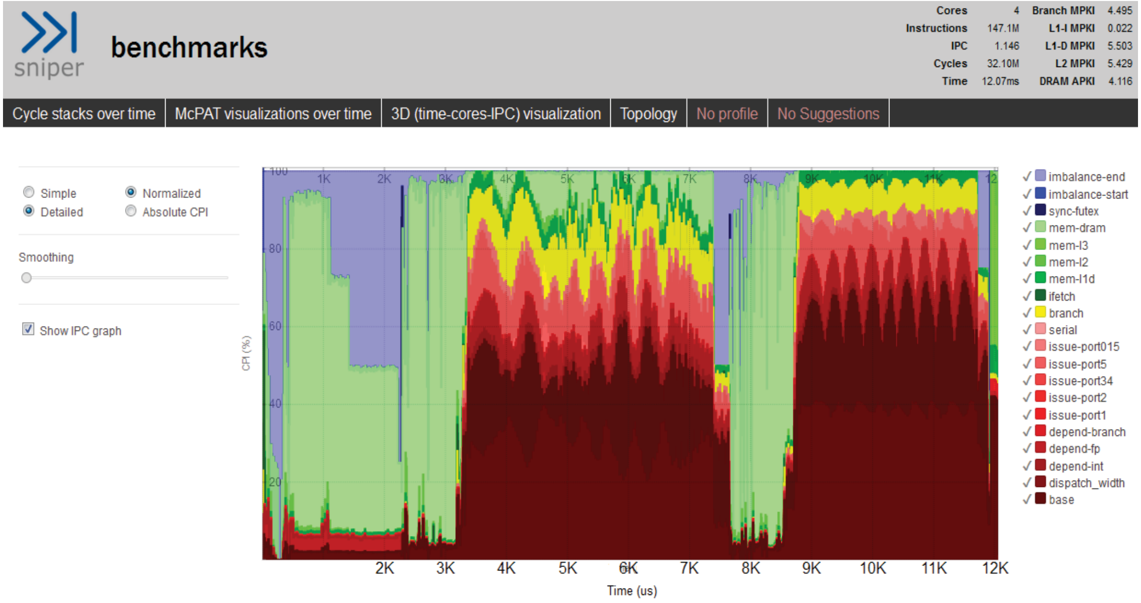


Figure 3.2. Power and area stacks

The options of sniper visualisations help users comprehend the behaviour of the software that works in the simulator. Currently, three main groups of visualisations exist in Sniper as illustrated in Figure 3.3. For example, they include cycle stacks that are plotted against time, McPAT plotted against time and 3D Time-Cores-IPC visualisations.


benchmarks

Cores	4	Branch MPKI	4.495
Instructions	147.1M	L1-I MPKI	0.022
IPC	1.146	L1-D MPKI	5.503
Cycles	32.10M	L2 MPKI	5.429
Time	12.07ms	DRAM APKI	4.116

○ Simple
● Detailed
● Normalized
○ Absolute CPI

Smoothing

☑ Show IPC graph

Figure 3.3 Cycle stacks plotted over time

3.3.4.Multiple Multi-threaded Workloads

One can run multiple, multi-threaded benchmarks simultaneously by installing the integrated benchmarks suite first. Thereafter, the user can utilise the benchmarking parameters to show the benchmarks and configurations that should be run. For instance, the command below can be used to run the example in Figure 3.3:

```
$BENCHMARKS_ROOT/run-sniper -c gainestown --benchmarks=splash2-fft-small-1,splash2-fft-small-1
```

There is no maximum number of threads that a user can run when utilising the --benchmark parameters. However, it is worth noting that this mode does not support the ROI handling or the multi-threaded workloads (Carlson & Heirman 2013).

3.3.5.Scripting

The sniper simulator has a Python interpreter that is used to request for statistics and update the configuration of hardware parameters when the simulator is in use. Consequently, this enables users to generate traces of IPC and make online DVFS updates among others.

3.3.6.Comparison between Sniper and Graphite

Graphite and Sniper have distinct characteristics and to understand them it is important to mention that Graphite is the foundation of Sniper whereby it removes, adds, and reworks on its characteristic, which is what distinguishes the two. This work looks into on the weaknesses and strengths of varying approaches. The configuration of the Sniper's default simulator is designed the same way as the Nehalem machine, the

Gainestown micro-architecture. Sniper generates the most noteworthy instruction latencies like math ops and FP ops. Previously, Sniper employed the Graphite configurations as the point of start but the developers of Sniper designed the models of today. For instance, they formed a parametric hierarchy of shared-cache and they provide support to sampling of multiple threads and a new Windowed version model of the MGI queuing model.

3.3.6.1. Unique Features in Graphite.

Simulators have numerous distinct features and to differentiate between Graphite and Sniper, Graphite maintains full-mode that is not in the current Sniper versions. Full-mode endures the allocation of single application simulation in different machines, which is helpful in the case of a big memory footprint whose simulation would be big enough for a single machine. Nevertheless, this can create a reduction in the simulation during the use of barrier synchronisation instead of relaxed synchronisation used automatically by graphite (Al-Manasia & Chaczko 2015b). Sniper no longer uses full-mode as it could not adjust to every system call throughout the whole simulator. It is essential to root all system calls to ensure their execution on the operator's behalf. Sniper employs light-mode completely, where most system calls pass directly to the operating system.

3.3.6.2. Unique Features of Sniper.

The integration of the assimilation of the CPI stacks and the core design of the interval simulation to enable more precise timing clear of the customary IPC models that are too sophisticated for the studies of network simulation is the most notable feature (Martin 2012).

Taking into consideration the specifications, there are some features permitting users to obtain faster simulations results from Sniper. The built-in benchmarks suite and benchmarks region of interest (ROI) integration provide an easier start for users. Sniper contained contemporary Linux environments like Red Hat, 32-bit, and Ubuntu, while also supporting current Java applications and Pin versions (Berkowits 2012). Furthermore, Sniper maintains several applications of multiple-threaded and single-threaded type like SIFT, trace format, or recent Pinballs including built-in infrastructure that carries out scheduling (Abts, Scott & Lilja 2003).

There are additional features of quality visualisation in the Sniper that enable better comprehension individual runs with information of topology for the generated architecture. The visualisation has McPAT incorporated into it, conveying proper results when run with Sniper core and different models. In addition, Sniper has the support of Python scripting and a powerful connection between the scripting, SimAPI, and the simulator. Another feature that makes sniper unique is its compatibility with message passing interface (MPI) applications together with the interchange that occurs between varying processes by apparent virtual-to physical mappings that the OS offers. At first, Sniper had added the support of true-DVFS that enabled every machine to operate its own frequency. The database statistics subsystems of Sniper are also sophisticated to permit routine report and graph generation from the databases. There is also direct addition of heterogeneous and hierarchical configurations support into the configuration files, that ease the creation of small tweaks to various configurations. More Sniper features include fault injection, loop tracer, micro-op support, and statistically allocated dynamic random access memory (DRAM) latencies (Carlson & Heirman 2013).

3.4. Experimental Setup

This section deals with the experimental setup and discusses the evaluation methodology.

3.4.1. System Setup

Our proposal evaluation uses either trace-based simulations with Sniper 6.1 (Carlson, Heirman & Eeckhout 2011) or full-system simulations with Simics simulator (Magnusson, Christensson, Eskilson, Forsgren, H, et al. 2002) depending on the time and implementation constraints of the study cases.

In conjunction with Sniper, the integrated McPAT (Li et al. 2009) is used to get a comprehensive timing, energy and area estimates for the CMPs that are being modeled.

In conjunction with Simics, the Wisconsin GEMS toolset (Martin et al. 2005b) is used to enable comprehensive CMP simulation. GEMS toolset include GARNET (Agarwal et al. 2009), which is a detailed on-chip network model simulator. Energy consumption of the simulated system has been calculated using the CACTI tool (Muralimanohar, Balasubramonian & Jouppi 2009).

The latencies assume a 32nm process technology. A 256-core system which has a fully shared LLC is simulated. The simulations were based on three objectives: energy consumption, the complexity of hardware (integration area) and performance. Table 3.2 provides the main parameters of the simulation environment along with benchmarks.

Processor Frequency & no. of cores	2.0 GHz, 256 cores
Split Private L1I & L1D	32KB, 4-way associative, 64B cache block size
L1 hit time	1 (tag) & 2 (tag+data) cycles
MSHR size & timeout	16 entries, 1000 cycles
Shared unified LLC	16MB, 16-way
LLC hit time	2 (tag) & 4 (tag+data) cycles
Off-chip access time	160 cycles
Cache hierarchy	Inclusive
Page size	8KB
Network	2D mesh (4x4)

Table 3.2 System Parameters

A wide range of shared-memory parallel workloads that need coherence from several suites were used including SPLASH-2 (Woo et al. 1995), SPEC, PARSEC (Bienia et al. 2008b), and several commercial server applications to evaluate the thesis proposals, as shown in Section 4.3.2, Table 5.2, and Table 6.3.

3.4.2. Evaluation Metrics

The tested metrics include the runtime, endpoint traffic, full system energy, the classification quality, miss ratio, network traffic, execution time, and the dynamic energy consumption. Throughout this research, the overall performance of different methods is demonstrated by the runtime (i.e. measuring the time necessary to complete certain amount of work). The metric instructions-per-cycle has been used by other works instead of runtime in judging performance improvements. However; system timing effects of multiprocessor workloads may alter the number of instructions executed. Therefore, Instruction Per Cycle (IPC) is not a suitable metric for evaluating the coherence protocols and systems.

Every reported experimental outcome corresponds to the benchmark parallel phase to avoid measuring thread forking. Several simulations runs for every benchmark were performed and small arbitrary perturbations were inserted in memory system timing to account for the inconsistency in multi-threaded workloads as in (Alameldeen & Wood 2003). A full system checkpoint (to provide a well-defined starting point) is used to initialise the system state and to simulate the execution until the end of the parallel phase. The number of cycles is recorded and referred to as application time in order to complete the parallel phase.

Endpoint traffic (in messages per miss) and interconnect traffic (in terms of bytes on interconnect links per miss) are other ways besides reporting runtime that measure and report the traffic. The endpoint traffic shows the amount of controller bandwidth needed for handling incoming messages. The total amount of link bandwidth used by the messages are indicated in the interconnect traffic as they traverse the interconnection.

3.5. Conclusion

A simulator that models a few core details of a micro-architecture at a speedy rate compared to the full-system simulators with hardware and software overheads is vital. Currently, the main objective entails the simulation of hundred to thousand-core chips, but there is lack of a typical OS that can scale this stage. Additionally, the ISAs places restriction on the number of cores, thus making it unfeasible to attain the preferred target.

The xAPICs of x86 can only support at utmost 256 cores. The selection of a simulator at this operation level should depend on the simulator type of the user-level. Thus, the most suitable choice for large scale CMP simulations is the Sniper simulator as it can stabilise the trade-off that happens between accuracy and performance. However, for a comprehensive and more accurate simulations, Simics simulator is preferable if time constraints are not an issue.

The system's process amount and cores of every packet continues to increase, thus creating the difficulty of simulating the development in sizes of a system within a practically short time. Compounding the increasing core numbers with bigger cache sizes, there is necessity for long, proper simulations to assess the next invention of system designs precisely. CSA simulators always provide a lot of abstraction levels having varying simulation speeds and detail levels. The importance of having different abstraction level limits itself to accurate simulation (low level) and fastness (high level) and permits the researcher to reason on various design elements. There is ease of use with Sniper simulator, and with it one can attain a balancing between performance and accuracy. Nonetheless, a few of the systems do not support various instructions not modelled correctly in Sniper. For instance, the 64-bit x86 and the SSE4 are a few of these instructions. However, Sniper simulation sustains playback instruction traces and simulates parallel applications in the pin based execution driven mode. Every simulated core can allow a trace of a single-thread, thus enabling the device to simulate multi-programmed workloads.

II. Contribution to Research

Chapter 4 **Modelling and Evaluation of Cache Coherence Mechanisms for Multicore Processors**

4.1. Introduction

Cache memory is a memory subsystem that stores data for fast access during persistent usage. The concept of caching is for increasing the memory speed of a slow affordable large memory by employing a small size of costly fast memory. Cache coherency defines the reliability of data stored in the cache of each core, in regards to multi-core processors. It is possible for processors with multiple cores to have shared and distributed caches on the chip, so accounting for coherence protocols is essential in guaranteeing that a core reads the current data from memory and not a value that another core updated. Figure 4.1 illustrates the problem of cache coherence by showing a shared L2 cache through four cores using private caches by a bus.

Each core goes into location X in a sequence. Core 1 reads X from L2 cache and takes a copy to its cache at the start. Core 4 then reads X from L2 cache and makes a copy to its cache. Afterward, core 4 modifies the location of X from value 8 to 5. Using a write-through cache, this updates the location of L2 cache; however, during action 4 when core 1 read location X once more, it read the previous value 8 from its cache instead of its actual value 5 from L2 cache. The caches that perform a write back make the situation even more challenging. The write of Core 4 would not revise L2 cache immediately; instead, it would just set the modified or dirty bit regarding the location X's cache block. L2 cache gets the cache block's contents written into it exclusively during the subsequent replacement of this cache block from the core 4's cache. The reading of the previous value is not inclusive to core 1; also the caches of core 2 and 3 will be missing during action 5 and 6 when both core 2 and 3 read the previous value of 8 rather than 5 from L2 cache. Lastly, if the write-back caches of more than one core have varying values to location X, L2 cache final value will not have any relation to the sequence wherein the writes to X happened instead the sequence wherein the replacement of the cache blocks with X will determine the end value. There should be enforcement of coherence between the caches to ensure correct execution. Implementation cost and performance are the main factors that affect this process.

Generally, there are two techniques for cache coherence in hardware; which are the directory-based and snooping-based protocols. The Snoopy methods of cache coherence need transmitting information to every cache controller. Nevertheless, cache messages increase with the increase in the amount of cores. Moreover, the required bus bandwidth that sends the messages and link the caches will be larger than available one, and then complete saturation of the bandwidth happens. Small-scale systems use these methods due to this constraint. However, the protocol based on the directory, scales to larger amounts of cores or processors compared to snoopy-based coherence protocol, as it allows actions of multiple coherence to occur simultaneously.

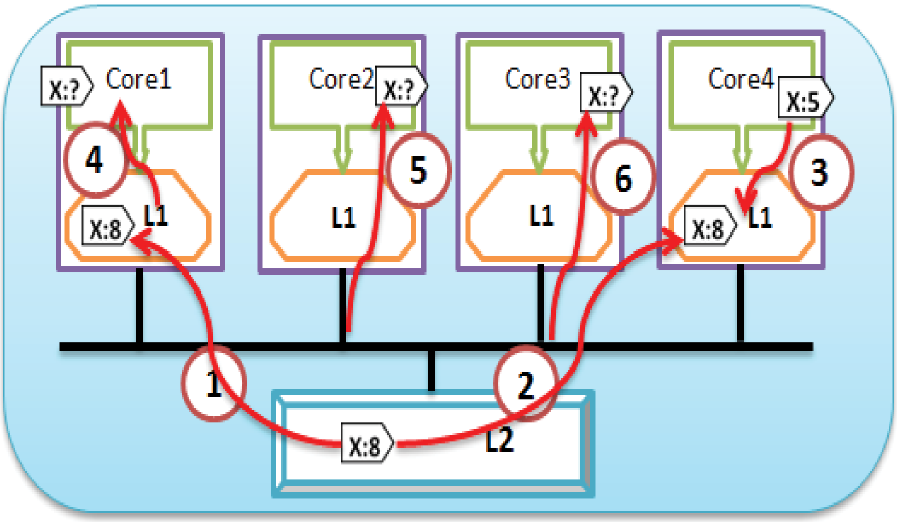


Figure 4.1 Cache Coherence Problem

Three vital attributes exist that influence the performance of the protocol of any cache coherence are elaborated as follows:

1. **Low-latency Cache-to-Cache Misses:** A cache-to-cache miss happens due to access of shared data that entails the provision of data by the cache of another processor. The latency reduction of cache-to-cache misses means the support of direct cache-to-cache misses by a coherence protocol (Martin 2003). To maintain the regular synchronisation and communication in these workloads well, it is necessary for systems to optimise cache-to-cache misses' latency (Barroso, Gharachorloo & Bugnion 1998).
2. **Scalability Challenges:** Because of the multithreaded future, it is necessary for hardware resources to scale efficiently. The hierarchy of memory is the main dilemma when it comes to hardware scalability. There is reliance of CMPs on big, multi-level hierarchies of cache to reduce the cost, constrained bandwidth and main memory high latency. Caches generally employ almost half of a chip area and a considerable system energy fraction. The key issues that limit the scalability of cache hierarchies include cache partitioning, associativity, and coherency.
3. **Bandwidth Efficiency:** It is essential for a cache coherence protocol to conserve bandwidth to reduce the cost and evade interconnect contention (as contention diminishes performance).

This chapter explains the relative behaviour of varying coherence protocols (Token-based coherence and conventional protocols). It intends not to (1) produce definitive throughput rates and execution times for the simulations or (2) evaluate such protocols on all system configurations. The aim is to accurately get relative evaluations and comparisons instead by employing an estimate of a system of a chip multiprocessor. The modelling and simulation of an entire system of the first-order timing outcomes for approximating an aggressive system of multiple cores running commercial loads are the ways to reach such an aim. This chapter aspires to attain the first-order outcomes – however, the same as the majority of architectural simulations – it does not seek to capture the features of the entire system in precise detail.

4.2. Background and Related Work

This section will illustrate conventional protocols, the Directory based and Snooping based protocols, together with the Token coherence protocol. It will describe how these protocols work, the advantages and drawbacks of each, and how to do the improvement of each.

4.2.1. Snooping Protocols

Snooping protocols entails the snooping of each bus transaction by the CMP cores, which then responds with proper state modifications for the equivalent cache lines based on two elements; the bus transaction type and cache line status. *The update-based* and *invalidation-based* protocols are two basic policies that categorise the coherence protocol based on snooping. Protocols based on invalidation include the Synapse (Frank 1984), the Illinois (Charlesworth 2002), the write-once (Goodman 1998), and the Berkeley (Katz et al. 1985) while protocols based on update include the Dragon (McCreight 1985) and the Firefly (Thacker, Stewart & Satterthwaite Jr 1988). Therefore, this study considers solely invalidation-based protocols because there has been more favour towards invalidation-based coherence over update-based coherence in the majority of the latest CMPs. The main current benefit of snoop-based CMPs is the fact that it has low miss latency, in particular for cache-to-cache misses. There is awareness by the responder that it has to transmit a response fast because a request goes directly to the system's memory modules and all other cores. Low latency in cache-to-cache miss is of great significance for workloads with substantial data sharing amounts. Responding with core cache's data when possible can minimise the average miss latency if cache-to-cache misses have reduced latency than fetching for memory data. Memory access that has low latency is due to the tightly-coupled environment of these systems (Martin 2003).

Previously, there have been two more advantages to snooping. Primarily, buses with shared wire were cost-effective interconnects for several systems, and coherence based on a bus provided an approach that was complex effective to implementing cache coherence. Second, snooping protocols based on a bus were comparatively straightforward. This benefit that was of great significance in the past is presently of much less significance. Latest snooping protocols that employ virtual buses are frequently as complex as or more compared to other approaches. The initial basic disadvantage of the snoop-based protocol is that, although there is an evolution in system designers away from shared-wire buses, designers that employ snooping are still bound to selecting interconnects that can offer virtual-bus performance when they opt for the interconnect. These interconnects related to virtual buses could be more expensive as they would need switch chips, may get lower bandwidth mainly because of root bottleneck, or might gain higher latency as every request requires to get to the root. In contrast, an unordered interconnect like a torus or grid of processors directly linked might have additional latency, cost, and bandwidth attributes (Martin 2003).

Naturally, snooping protocols remain broadcast-based protocols, which is the next basic disadvantage. It means they are protocols with the need of bandwidth that increases per the quantity of cores. It is a need that constraints system scalability, even after the elimination of the virtual or shared-wire bus bottleneck. Several proposals (Bilir et al. 1999; Martin et al. 2003; Sorin et al. 2002) plan to reduce the bandwidth need of snooping by utilising prediction based on the destination set, also referred to as predictive multicast – rather than broadcasting every request. Although they lessen request traffic, the main disadvantage of these proposals is that they depend on a totally-ordered interconnect. Below are a few methods that can be employed to enhance the Snoopy cache coherency protocols:

1. **3 Wired OR Signals** – first signal assertion occurs whenever any another cache besides the requester contains a block copy and a second one occurs when any cache contains exclusive block copy. The occurrence of the final third signal is during the completion of all snoop actions on the bus (Geer 2005) The requesting L1 and L2 examine the two signals after the third asserted signal. It is plausible to improve performance by applying these signals using L-Wires with low latency as all of them are on the critical path.
2. **Voting wires** – this is a different method employed to improve protocols based on Snoop that have low latencies. In general, transfers from cache to cache happen from modified state data, while there is one supplier (Galles & Williams 1994). even though one can retrieve a block from another cache instead of memory in the MESI protocol.

4.2.2. Directory-based Protocols

The distribution of memory in directory based protocols happens among varying processors and there is maintenance of the directory for every such memory. At present, most CMPs employ directory protocols as well in order to support cache coherency. Misses in L1 cache go to L2 caches and there is maintenance of a directory which stores the block status across every L2 cache. Home node, where there is storage of original data, receives the request to check if it has. The transmission of request from another cache's requester node, if unavailable the request moves to the remote node through the home node and initially looks for data from remote node, and transmits it later to the node that was requested. In addition, the write-invalidate-direct founded protocol is important in among the most common technology of CMPs currently in use. Protocols based on the directory aim at interconnection limitations and scalability avoidance associated with snooping protocols. In fact, they predate those protocols of snooping, with Feautrier and Censier (Censier & Feautrier 1978) and Tang (Tang 1976) carrying out early work on directory-based protocols during the late 1970s. Directory protocol systems, also referred to as cache-coherent non-uniform memory access (CC-NUMA) or distributed shared memory (DSM), are preferable when scalability in cores or processors quantity is a limitation of first-order design. In most cases, these protocols often forfeit speedy cache-to-cache misses in trade for this scalability.

Most studies talked about the benefit of directory protocols, perhaps considerably enhanced scalability. By merely getting in contact with those processors that might have cache block's copies (or less additional processors during the employment of an approximate directory application), the system's traffic increases linearly with the quantity of cores. On the contrary, broadcasts' endpoint traffic employed in snooping protocols increases quadratically (Martin 2003). Together with a scalable interconnect, one with a bandwidth that increases linearly per the cores' number, with directory protocol the system can scale to thousands of cores. When utilising large CMPs the two scalability problems encountered are the amount of directory state needed becomes a great worry, and reasonable interconnect is not truly scalable. Profound research on these two issues have been applied expansively, and systems in support of hundreds of processors are in use such as the SGI Origin 2000 (Loudon & Lenoski 1997).

The capacity to take advantage of arbitrary point-to-point interconnects is the next, and perhaps the more imperative, advantage of protocols based on directory. In general, the point-to-point interconnects have low latency and high bandwidth (Martin 2003).

The main drawbacks of directory protocols are two: One, the extra directory access and interconnect traversal, which is on the critical path of cache-to-cache misses. Therefore, there is simultaneous performance of the memory lookup using the memory-to-cache of directory lookup, with no penalty for misses. The latency of directory lookup is like that of DRAM in several systems, and so situating this lookup on cache-to-cache misses' critical path increases the miss latency of cache-to-cache considerably. Even though it is possible to reduce the directory latency with fast SRAM in order to cache or hold directory information, the additional latency offered by extra interconnect traversal is more difficult to mitigate. The miss latency of cache-to-cache enhances significantly with the mixture of these two latencies. With the frequency of cache-to-cache misses in much of significant commercial workloads, these enhanced miss latencies might have a grave consequence on system performance.

The manipulation and storage of directory state might be the second drawback of directory protocols. This was visible in earlier systems that employed devoted directory storage (DRAM or SRAM), thus increasing system cost. Conversely, saving the state of directory while eliminating the extra overhead of storage capacity; many current directory protocols have utilised DRAM and bit reinterpretation employed for error correction codes (ECC) (e.g., the S3mp (Nowatzyk et al. 1994), Alpha 21364 (Mukherjee et al. 2001), UltraSparc III (Horel & Lauterbach 1999), and Piranha (Barroso et al. 2000; Gharachorloo, Barroso & Nowatzyk 2000)). The increase of memory writes and reads storing these bits in memory increase the traffic in memory (Gharachorloo, Barroso & Nowatzyk 2000; Martin 2003). Among the best methods to enhance the Directory-based Cache Coherency protocols include:

1. **Exclusive Read Request for a block in a shared state**

   Because it is a clean copy, the L2 cache, once it gets a request from the L1 cache, it invalidates each L1 cache in this strategy. Prior to data transmission to the sender, other L1 caches will send an invalidate acknowledgment to that L1 cache. Usually, there is need by the L2 cache of a hop for a reply and for an acknowledgment. Hence, the hop's latencies and that of both the acknowledgment and reply messages should have similar values. Here, the sending of both messages occurs simultaneously through the corresponding low power PW-wires and low latency L-wires. This strategy improves performance and reduces power consumption.

2. **Read request for block in exclusive state**

   A read request will go to the exclusive owner for the L2 cache as the requesting L1 gets the L2 cache's data copy. The exclusive owner will then reply the requesting L1 cache, stating that the data that the L2 cache sent is valid given a clean exclusive copy. The exclusive owner sends a data copy to the requesting node if it requests for it, while updating the L2 cache data. The requesting cache only goes on after the exclusive owner sends a message. The L2 cache sends data via slow PW wires simultaneously. The exclusive owner should then send an acknowledgment message to the requester through L-wires of low bandwidth if the owner copy is a clean one, while if the copy of the owner is a dirty one, then the exclusive owner sends the message through the B-wires and uses the PW-wires to send the write back to the L2. This is an approach that mainly follows the performance improvement by transmitting the high prioritised data via the L-wires and the low prioritised one via PW-wires.

3. Using the techniques of which the coherence protocol is **proximity awareness,** improving the multi-core performance by lowering the redundant access to the memory off-chip.

4.2.3. Token-based Protocols

There is removal of the limits of directory indirection without giving up either interconnect decoupling from the coherence protocol or coherence decoupling from consistency, thus occurs using the protocol of token coherence proposed by Martin et al. (Martin 2003; Martin, Hill & Wood 2003a, 2003b). Token coherence employs counting of the token for the resolution of races, without the necessity to require an ordered interconnect or home node. It contains even more decoupling levels by separating the correctness substrate from the performance policy of the system. There is a further decoupling of the correctness substrate on avoiding starvation and invoking safety. The counting of token does not guarantee the satisfaction of a request in the end, although it guarantees safety. Therefore, prevention of starvation is possible through the correctness substrate. The processor sets off a relentless request in the detection of a potential starvation. With a reasonable arbitration method, the substrate then turns on at most a single persistent request for every block. The node of each system recalls all activated determined requests and forwards every block token to the initiator of the request. At last, the initiator carries out an operation of the memory (a store or load instruction) and disengages its constant request when it has the needed tokens. The development of performance policies of token coherence is such that it approximates an unordered protocol based on broadcast (stimulated by snooping protocols), a performance policy that is bandwidth-efficient that copies a directory protocol, and a projecting hybrid protocol that utilises prediction set by destination (Martin 2003; Martin et al. 2003).

TokenB protocol pays attention to cache-to-cache misses such as snooping protocols and not needing any interconnect ordering such as directory protocols. An outwardly obvious approach is using an unordered interconnect to directly transmit broadcasts.

4.3. Evaluation Methodology

This section deals with the experimental setup and discusses the evaluation methodology.

4.3.1. System Setup and Validation

The evaluation is performed in order to clarify the relative behaviour of diverse coherence protocols (Token Coherence based and traditional protocols). To carry out the analysis, the Simics simulator (Magnusson, Christensson, Eskilson, Forsgren, H, et al. 2002) that has a full-system multiprocessor is used with the Wisconsin GEMS toolset (Martin et al. 2005b).

The research illustrates the entire performance of various protocols through the runtime, such as measuring the time needed to accomplish a specific quantity of work. Other works have used the metric Instructions-Per-Cycle (IPC) rather than runtime in reviewing improvements of performance. Nonetheless; the effects of system timing of multiprocessor workloads may change the quantity of instructions executed as a result, IPC is not an appropriate evaluating metric for the coherence systems and protocols.

The parallel phase is where the measurement begins, to prevent the measurement of thread forking. The checkpoint of a full system is used to begin the state of a system and for execution simulation until the parallel phase ends. There is recording of the cycle amount, known as application time, in order to accomplish the parallel phase.

Traffic at the endpoint and interconnect are other means that calculate and report the traffic besides reporting runtime. The endpoint traffic demonstrates the quantity of controller bandwidth necessary for dealing with incoming messages. The interconnect traffic indicates the total link bandwidth utilised by the messages as they cross the interconnection. Table 4.1 shows the details of the way the modelling of the system's main features occur.

Private L1 Caches	64KB, 4-way set associative, split D/I, 1ns latency (2-cycle latency)
L2 Cache	Unified, 4MB, 6ns latency (12 cycles)
main memory	4GB, 80ns (160 cycles)
Coherence protocol	MOESI protocol
interconnect link	4GB/second or unbounded bandwidth 15ns latency (30 cycles)

Table 4.1 Simulation Parameters

4.3.2. Benchmarks

In this chapter, three commercial workloads of multi-threads were employed from the Wisconsin Commercial Workload Suite (Alameldeen et al. 2003): a Java middleware workload (SPECjbb), an Online Transaction Processing workload (OLTP), and a static web serving workload (Apache). The workloads mentioned previously run on a 16-core SPARC processor that operates with Solaris 9. The simulated system has a main memory of 4GB.

4.4. Analysis and Evaluations

The simulation of a full-system is necessary to analyse the demand system, as it enables for assessing the proposed systems when operating practical scientific applications upon the actual operating system. In addition, it is also getting the subtle timing outcome that is not possible to capture evaluation based on trace. Cycles per transaction is employed as the performance metric. Figure 4.2 demonstrates the normalised runtime that shows TOKENB is faster compared to SNOOPING using unlimited bandwidth links by (21–34%). In addition, TOKENB, compared to DIRECTOTY, is faster by (26–124%). This is because it avoids cache-to-cache misses of third interconnect traversal, the latency of the directory lookup, and eliminates memory controller’s blocking states.

The endpoint traffic of TOKENB is the same as or fewer than SNOOPING, whereas has additional interconnect traffic compared to SNOOPING. The endpoint traffic is as shown in Figure 4.3 (in normalised messages per miss that every coherence controller at the endpoint receives), and Figure 4.4 shows the interconnect traffic (in normalised bytes per miss).

TOKENB puts in some extra traffic overhead, but the overhead is less for all workloads. TOKENB and SNOOPING both make use of extra traffic for control messages of write-back. However, due to the detailed execution of decisions in SNOOPING concerning acknowledgment messages of write-back, SNOOPING utilises additional traffic for write-backs compared to TOKENB. SNOOPING transmits a write-back request as a marker message to itself and to the memory on the ordered interconnect. If the block still belongs to it, it gets the marker message and sends the memory this data. The ignorance of this implementation overhead brings the conclusion that these protocols produce very similar quantities of traffic.

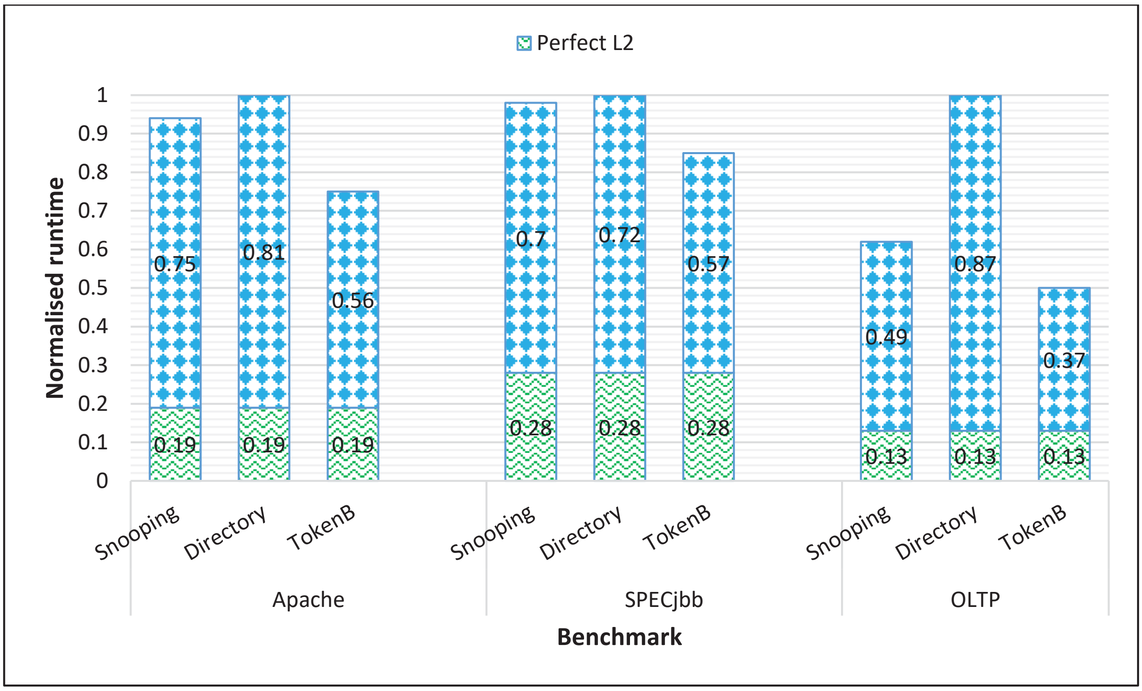


Figure 4.2 Runtime of TOKENB, SNOOPING, and DIRECTOTY using unbounded link bandwidth

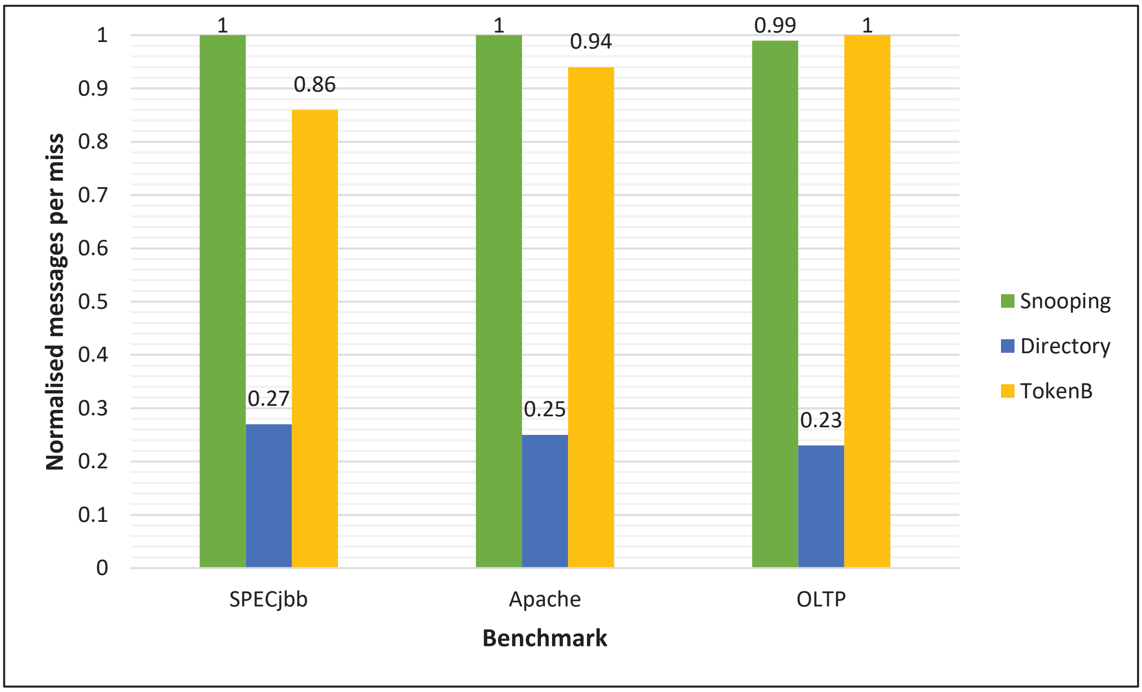


Figure 4.3 The endpoint Traffic of DIRECTOTY, SNOOPING, and TOKENB in normalised messages per miss

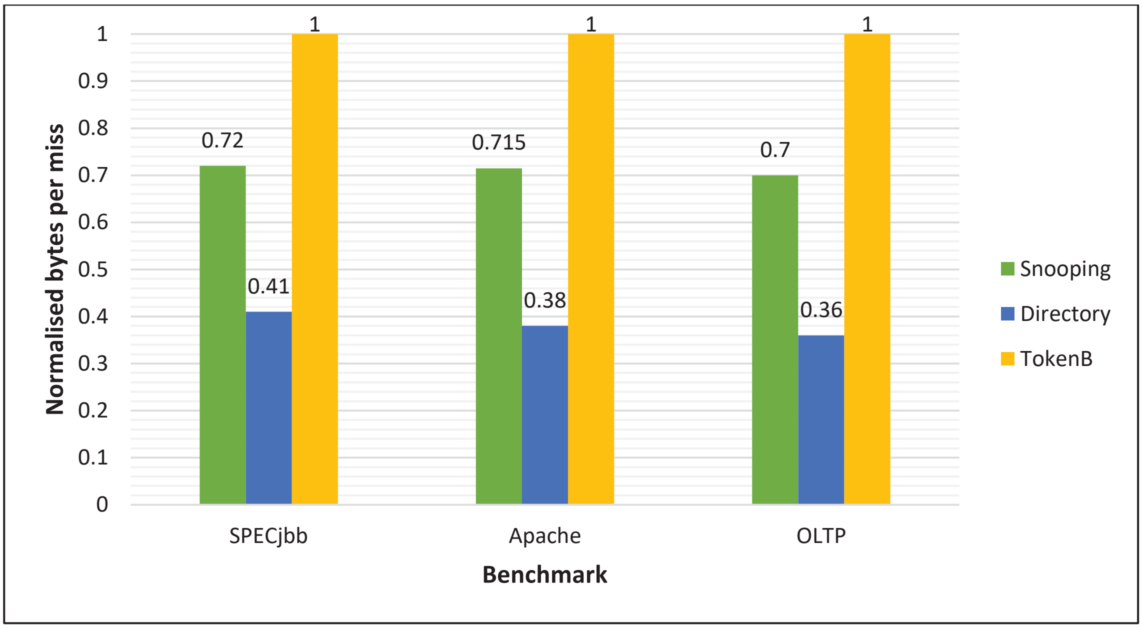


Figure 4.4 Normalised Interconnect Traffic of DIRECTOTY, SNOOPING, and TOKENB

In these figures, TOKENB produces more interconnect and endpoint traffic compared to DIRECTOTY, by about three times. Hence in the results, TOKENB requires coherence controllers with extremely higher bandwidth. TOKENB relies on broadcast, which constrains its scalability and is less scalable than DIRECTOTY, hence DIRECTOTY stays away from broadcast. TOKENB and SNOOPING-based protocols are not a proper option for systems with many cores on the same chip for their scalability limitations on bandwidth.

4.5. Conclusions

The methods of cache coherence are a major aspect aiming at attaining the goal of the development of exponential performance through extensive thread-level parallelism. This chapter studied the effective protocols and methods that are available, employed to attain cache coherent in architectures with multiple cores. Token based, Snooping-based, and Directory-based protocols are the protocols that modelled and analysed on the

Simics/GEMS simulator in this study. It further illustrates the strengths and weaknesses of each protocol, as well as discussions how to carry out their improvement. TOKENB is both faster compared to DIRECTOTY and better than SNOOPING with ample bandwidth. The superiority of TOKENB is to SNOOPING is because it employs similar quantities of traffic and is capable of outperforming SNOOPING by making use of a quicker, unordered interconnect. Furthermore, it is of a higher speed compared to DIRECTOTY in environments with ample bandwidth by avoiding putting latency in directory lookup and a third interconnect traversal on cache-to-cache misses' critical path. Alternatively, TOKENB uses a reasonable amount of extra interconnect traffic and significantly more bandwidth of an endpoint message than DIRECTOTY when it comes to small systems. Therefore, the performance of DIRECTOTY is better compared to TOKENB in an environment with limited bandwidth.

The choice of coherence protocol is as complicated and subtle today as it has ever been. For this study, the following chapters mainly focus on Directory protocols for their high scalability in comparison to other protocols.

Chapter 5 **AHRC: An Optimised Cache Associativity**

Chapter 5 **AHRC: An Optimised Cache Associativity 97**

5.1. Overview

Hardware resources require efficient scaling, because the future of computing technology seems to be intensively multithreaded. One of the main challenges in the scalability of computers hardware is the hierarchy of the memory. CMPs rely on large and multi-level hierarchies of caches, to reduce the cost of resources and improve systems performance. These multi-level hierarchies are the ones that also help to solve the issue of limited bandwidth, while minimising the latency of the main memory. Almost half of the area of the chip and a large percentage of the system energy is used by caches. One of the main problems limiting the scalability of cache hierarchies is called cache associativity. Caches consume a lot of energy to implement associative lookups. This affects the performance of the system by reducing the efficiency of caches.

This chapter describes a new design of cache associativity and replacement that is called - Adaptive Hashing and Replacement Cache (AHRC). This design has the ability of maintaining high associativity with an advanced method of replacement policy. AHRC can improve associativity and maintain the number of possible locations, where each block is kept as small as possible. Several workloads were simulated on a large-scale CMP with AHRC as the last-level cache. This thesis proposes an Adaptive Reuse Interval Prediction (ARIP) scheme for AHRC, which is superior to the NRU scheme that was described by Seznec. The results demonstrate that AHRC has better energy efficiency and higher performance as compared to conventional caches. Additionally, large caches that utilise AHRC are the most suitable in many core CMPs to provide a more significant improvement and scalability than the smaller caches. However, AHRC with a higher-level replacement may lead to loss of energy for workloads that are not sensitive to the policy governing the replacement process.

5.2. Introduction

There is an increasing demand for main memory bandwidth and latency. This continues to lead to Chip-multiprocessors' higher capacity and ability to relate easily with each other. Moreover, Moore's law allows room for CMPs to have many cores on one chip (Howard et al. 2010; Shin et al. 2010), however main memory still accesses limited bandwidth, and is of high energy and latency – all these affect the overall scalability of the CMPs. Contemporary chip design helps solve this problem as CMPs depend on complex memory hierarchies that have caches that are large and associative. This design normally causes the caches to occupy more than half the area of the chip, as they contribute to the consumption of both dynamic and static power (Kurd et al. 2010; Wendel et al. 2010).

The high associativity allows for flexibility in replacement, which in turn enables efficient utilisation of the limited cache capacity. Last-level caches in the current CMPs have high associativity of either 16-way or higher than 16-way (Hill & Smith 1989). This solves the issue with conflict misses that come up when frequently-accessed blocks are contained in the same set in the cache. The mode of doing this involves an increase in the number of ways based on the number of cores. However, the high associativity leads to increased static power and access latency/energy, which comes about due to the larger tags which are a result of the set associative lookup. Currently, the main challenge facing chip designers is to reduce the frequency of last-level cache misses in chips with multicores as they maintain the low on-chip hit latencies, placing a tough trade-off on cache design.

Modern designs provide a multitude of physical ways that caches can improve on associativity. Unfortunately, enhancing associativity by increasing the number of physical ways, leads to an increase in the latency and energy cost of cache hits. For instance, there is a limit in the level of associativity for first-level caches for most chips to two- or four-ways. Additionally, for the last-level cache, a 32-way set-associative cache has up to 3.3 times the energy per hit and is 32% slower than a 4-way design (Sanchez & Kozyrakis 2010). Most alternative techniques of improving associativity depend on increasing the number of locations where one can place a block (with e.g. multiple locations per way (Agarwal & Pudar 1993; Calder, Grunwald & Emer 1996; Rolán, Fraguela & Doallo 2009), victim caches (Basu et al. 2007; Jouppi 1990a) or extra levels of indirection (Hallnor & Reinhardt 2000; Qureshi, Thompson & Patt 2005)). As a result of increasing the number of possible block locations, there is an increase in the energy and latency of cache hits. Most of these approaches are more complicated than conventional cache arrays (requiring e.g. heaps (Basu et al. 2007), hash table-like arrays (Hallnor & Reinhardt 2000) or predictors (Calder, Grunwald & Emer 1996)). Alternatively, one can use hashing to index the cache. This spreads out the accesses and avoids worst-case access patterns (Kharbutli et al. 2004; Seznec 1993). Even though hashing based schemes improve associativity performance, the number of block locations still proves to be a limiting factor. This study makes the following contributions:

1) Proposing AHRC, a new design of cache that can improve associativity and maintain the possible number of locations of each block as small as possible. This design

is based on the understanding and knowledge that associativity is not determined by the number of locations in which a block can reside. Associativity is determined by the number of replacement candidates on an eviction.

2) Proposing an adaptive reuse interval prediction (ARIP) replacement policy that can resist both scan and thrash. An optimal replacement policy makes its replacement decisions using perfect knowledge of the reuse pattern of each cache block and replaces the block that will be reused furthest in the future. To be robust across all patterns of cache access, there is a need to dynamically determine whether an application is best suited to thrash resistant or scan-resistant (e.g. protected LRU). ARIP policy needs only 2-bits per cache block and can easily integrate into the already existing LRU approximation that is found in modern processors. ARIP identifies the replacement policy that is best suited for the application. ARIP can dynamically make a choice between thrash-resistance and scan-resistance which improves the associativity of AHRC significantly.

The rest of the chapter is organised as follows: Section 5.3 gives the necessary background on approaches to increase cache associativity. Section 5.4 presents the AHRC and ARIP design. Section 5.5 discusses the evaluation methodology, and Section 5.6 presents the results and evaluation of the AHRC as a last-level cache. Section 5.7 concludes this chapter.

5.3. Background and Impact

In addition to increasing the number of permutations of cache block locations and checking them in parallel, there is intensive research into alternative ways of improving the associativity of the cache. They highly depend on the hash functions to spread out cache accesses, or increasing the number of locations that a block can be in.

5.3.1. Hash Block Address:

Involves the use of a better hash function. One uses the hash function to compute the index in place of using a subset of the bits forming the original block address. Hashing spreads out access patterns that are pathological; this includes stride accesses which map to the same set all the time and slightly increases access latency. Additionally, it increases with increase in area and power as the circuitry increases. It also adds tag store overheads. This is because storage of the full block address must be in a tag. Research experiments report that simple hash functions perform well (Kharbutli et al. 2004). Some commercial processors employ this technique in their last-level cache (Sun Microsystems 2007).

5.3.2. Skew-associative Caches:

Seznec gave a proposition of a new cache-indexing scheme referred to as skewed associativity, to solve conflict misses in low-associative caches (Seznec 1993). The scheme uses simple hashing functions which combine index and tag bits to map an address to distinct sets in each way of the cache. Skewed-associative caches ensure that addresses will conflict in one way only.

Over time, blocks that conflict in one-way settle into non-conflicting locations in other ways. This reduces conflict misses and increases the effective associativity of the cache. However, skewed-associative caches go against the concept of a set and complicate cache replacement; least-recently-used (LRU) replacement proves to be difficult to implement and past skewed-associative designs have relied on simple not-recently-used (NRU) schemes (Bodin & Seznec 1995).

5.3.3. ZCache:

ZCache (Sanchez & Kozyrakis 2010) builds on Skew-associative caches to provide even greater effective associativity from a low associative (for instance, 4-way) cache. ZCache combines H3 hash-based indexing (Carter & Wegman 1977), a new bucketed pseudo-LRU replacement algorithm, and a replacement scheme that has multi levels, inspired by cuckoo hashing (Pagh & Rodler 2001). In ZCache, a victim block can displace other victims to alternate locations. This allows for the consideration of a large number of replacement candidates. The possibility of the cascading replacement happens because the hash function assigns each block to distinct alternate sets, enabling better cache utilisation without increasing the number of ways. However, the flexibility increases the complexity of implementation, along with additional energy to move the blocks between locations.

5.3.4. Cuckoo Directories:

These are similar to ZCache. A Cuckoo Directory (Ferdman et al. 2011a) uses skewed-associative lookup. Upon a miss, the Cuckoo Directory considers allocations that involve multiple displacements. ZCache limits the number of levels searched during an eviction operation. On the contrary, the Cuckoo Directory performs a *depth-first* search, however, it does not limit the search by levels. Instead, it continues to iterate until a valid insertion location is found, and data is then shifted as required.

5.3.5. RRIP:

RRIP (Jaleel, Theobald, et al. 2010) describes a replacement scheme which is closely related to Seznec's pseudo-LRU. RRIP is inspired by the observations that effective replacement algorithms, such as LRU, are difficult and expensive to implement. Additionally, LRU has no record of the frequency of how data is accessed, and does not have the intelligence to know the fact that data recently put into the cache may be useless once it is used. RRIP implements a simple mechanism based on priority. It initiates with low priority and increases once there is future data access. There is a selection of victim blocks from the lowest priority group. If no blocks have minimum priority, then there is a reduction in priority of all the blocks as the search is repeated.

5.3.6. Allow Multiple Locations per Way:

Column-associative caches (Agarwal & Pudar 1993) extend direct-mapped caches to allow a block to reside in two locations based on two hash functions (primary and secondary). There are lookups which check the secondary location. If the first is a miss, there is a rehash bit raised to show that the block is in the second location. To improve access latency, there is a swap in location between the primary and secondary location once there is a hit in the secondary location. There is extensive use of the scheme to predict which location to probe first (Calder, Grunwald & Emer 1996), higher associativity (Zhang, Zhang & Yan 1997), and schemes that explicitly identify the lesser used sets and use them to store the more frequently used ones (Rolán, Fraguela & Doallo 2009). The disadvantages of allowing multiple locations per way include the varying nature of hit latency and reduced bandwidth of the cache because of the multiple lookups, and the additional energy requirement to perform the swaps.

5.3.7. Use a Victim Cache:

The victim cache is a fully associative small cache that stores blocks removed from the main cache until they are either removed or re-referenced (Jouppi 1990a). It avoids conflict misses that are re-referenced after a short period. However, it works poorly with a small amount of conflict misses in many hot ways (Brehob 2003b). Scavenger (Basu et al. 2007) divides cache space into two equally large parts. These include a fully associative victim cache organised as a heap and a conventional set-associative cache. A miss in the main cache introduces additional latency and energy consumption which checks the victim cache whether it holds the block or not.

5.3.8. Use Indirection in the Tag Array:

Use of indirection is an alternative strategy to implement tag and data arrays separately. It makes the tag array highly associative, and has pointers to a non-associative data array. The Indirect Index Cache (IIC) (Hallnor & Reinhardt 2000) uses the tag array as a hash table using open chained hashing for increased associativity. Tag indirection schemes are affected by the extra hit latency, as they are forced to serialise accesses to the tag. The V-Way cache (Qureshi, Thompson & Patt 2005) uses a conventional set-associative tag array, but also makes it bigger than the data array. This makes conflict misses rare. Both the IIC and the V-Way cache have tag array overheads of around 2x, and the IIC has a variable hit latency (Sanchez & Kozyrakis 2010).

5.4. Design of AHRC and ARIP

This section presents our proposed design for cache associativity in conjunction with an optimised replacement policy.

5.4.1 AHRC

AHRC represents a new design of cache that this thesis is proposing. The technique can improve associativity and maintain the possible number of locations of each block (or ways) as small as possible. This design is based on the understanding and knowledge that associativity is not determined by the number of locations in which a block can reside. Associativity is determined by the number of replacement candidates on an eviction. Using different hash functions, AHRC can access each way just like a skew-associative cache (Seznec 1993). AHRC enlarge replacement candidates' numbers by using a three-level of replacement to keep the evicted block from a way into another that is not conflicted with the first. The best replacement candidate is selected by ARIP replacement policy. AHRC can access each way just like a skew associative cache, but with a higher level of replacement and simple relocation process. AHRC approach advocates using functions from the H3 family of universal hash functions (Carter & Wegman 1977; Ramakrishna, Fu & Bahcekapili 1997), which can attain many uniform and uncorrelated distributed hash values. AHRC modified the cache controller to include the hash functions, some additional state to store the hash values. AHRC use only three levels since the replacement process requires extra bandwidth, especially on the tag array, and needs extra energy. It is possible to have a block in only one location per way. Such blocks require only a single lookup. On replacement, AHRC can exploit different hash functions and thus, any block conflicting with the incoming block is moved to a non-conflicting location in a different way instead of being evicted to host the new block. This resembles the Cuckoo technique [19] of building space-efficient hash tables. For the AHRC to accommodate the incoming block during a miss, it walks the tag array by evicting the best one and performing a series of relocations. This process does not happen on the critical path. It happens off the critical path and concurrently with the miss and other lookups, its effect on access latency is thus reduced to zero.

5.4.2 ARIP

Policies of practical cache replacement approach attempts to emulate perfect replacement by predicting the reuse interval of a cache block. The most commonly used LRU replacement policy gives a prediction of a near immediate reuse interval on cache misses and hits. Those applications that show a distant reuse interval have proved to be working poorly under LRU. This is because such applications have a working set that is larger than the cache, or experiences frequent regular bursts of reference to non-temporal data which is known as scans. To improve the performance of such workloads, this research has proposed replacement of cache by utilising the knowledge of reuse interval prediction.

Replacement decisions in an optimal replacement are made using perfect knowledge of the reuse (or use pattern) of each cache block. The blocks that will be reused in the future are then replaced. On a miss, the replacement policies will always make a prediction on when the missing block will be reused next. Such predictions can be updated when further information about the block is available, for instance, on a re-reference.
Efficient utilisation of last-level cache is important in avoiding long latency misses of cache to main memory. Under LRU replacement, several studies have illustrated that

the filtering of the temporal locality by small caches causes the majority of the blocks inserted into the LLC not to be reused at all (Kaxiras, Hu & Martonosi 2001; Lai, Fide & Falsafi 2001; Qureshi et al. 2007; Xie & Loh 2009). Because of the LRU's poor performance for cache access patterns which results from filtered temporal locality, cache utilisation ends up becoming inefficient.

There are several other solutions (Basu et al. 2007; Johnson & Shasha 1994; Loh 2009; Rajan & Govindarajan 2007; Zhou, Philbin & Li 2001) that exist, but they drastically alter the organisation of the already existing cache or require additional hardware.

NRU uses one bit for each cache block. NRU provides a distant reuse interval prediction for the block that was not recently used and a near-immediate interval prediction for the block that was recently used. NRU always predicts that the missed or referenced block will have a near-immediate reuse and picks the replacement block from the distant future group. This study proposes NRU2 which is similar to the NRU but with two bits instead of one. NRU2 is more capable of resisting scan than NRU.

In case the reuse interval of all blocks turn out to be bigger than the available space, NRU2 does not efficiently utilise the cache. In such cases, NRU2 causes cache thrashing which lowers the cache hits to zero. To prevent cache thrashing, this study also proposes a bimodal NRU2 (BNRU2), which enters mainstream of cache blocks with a distant reuse interval of prediction and infrequently enters new coming blocks of cache with a long interval prediction. BNRU2 is similar to the bimodal insertion Policy (BIP) (Qureshi et al. 2007). BIP is an element of Dynamic Insertion Policy (DIP) that has a great role in preserving some of the working sets in the cache.

Using BNRU2 can significantly degrade the performance of cache for non-thrashing access patterns. To be robust across all patterns of cache access, this study proposes to dynamically determine whether an application is best suited to be thrash resistant or scan-resistant. This study proposes adaptive reuse interval prediction (ARIP) that can resist both scan and thrash. ARIP uses a similar to the expansion of DIP to shared caches. Two Set Dueling Monitors (SDMs) is utilised for every application, to progressively self-determine if the application ought to utilise NRU2 or BNRU2 within the sight of different applications.

ARIP policy needs only two bits for each cache block and can easily fit in the already existing LRU approximation that is found in up-to-date processors. ARIP identifies that it is best suited for the application and can dynamically make a choice between thrash-resistant and scan-resistant policies. ARIP recognises the replacement policy that is most appropriate for the application and can progressively settle on a decision between thrash-resistant and scan-resistant policies.

5.5. Methodology

This section discusses the experimental setup.

5.5.1. Infrastructure

The design of AHRC is evaluated using trace-based simulations with Sniper 6.1 (Carlson, Heirman & Eeckhout 2011). Sniper is an accurate and a parallel simulator that is a high-speed x86. It integrates the method of interval simulation and expands the functionality of graphite (Miller et al. 2010) simulation infrastructure, hence allowing for fast and accurate simulation. Sniper has a variety of flexible simulation options when exploring various heterogeneous and homogeneous multi-core architectures because of trading off simulation speed for accuracy. Table 5.1 gives a summary of the simulated system parameters. A 256-core system which has a fully shared L3 cache is simulated. The integrated McPAT (Li et al. 2009) is used to get a comprehensive timing, energy and area estimates for the CMPs that are being modeled. The latencies assume a 32nm process at 2GHz. The simulations were based on three objectives: energy consumption, the complexity of hardware (integration area) and performance which was measured in average clocks per instruction (CPI). CPI is calculated by the sniper, while energy consumption and hardware complexity are computed by McPAT.

5.5.2. Workloads

Different multithreaded benchmarks were used including PARSEC (Bienia et al. 2008a) and commercial server applications. Table 5.2 has a list of the workload suite together with their configurations. All applications are executed with their reference (maximum size) input sets. For the multithreaded workloads, the first 10 billion instructions were run immediately after fast-forwarding into the parallel region.

Cores	256 cores, x86-64 ISA, in-order, IPC=1 except on memory accesses, 2 GHz
L1 caches	32KB, 4-way set associative, split D/I, 1-cycle latency
LLC cache	8MB NUCA, 8 banks, 1MB bank, shared, inclusive, MESI directory coherence, 4-cycle average L1-to-L2-bank latency, 6–11-cycle L2 bank latency
MCU	4 memory controllers, 200 cycles zero-load latency, 64GB/s peak memory BW

Table 5.1 Main characteristics of the simulated CMPs

Benchmark	Configuration
zeus	16K connections, fastCGI, 12.8 billion instructions.
apache	16K connections, fastCGI, 12.8 billion instructions.
db2	100 warehouses (10GB), 64 clients, 2GB buffer pool, 6.4 billion instructions.
oracle	100 warehouses (10GB), 16 clients, 1.4GB SGA, 6.4 billion instructions.
fluidanimate	native workload, 6.4 billion instructions.
moldyn	19652 mo., boxsize 17, 2.56M interactions, 9 iterations

Table 5.2 Benchmark Configurations

The instructions in synchronisation routines were assumed (barriers, locks, etc.), to be able to determine when to end the execution process because synchronising can easily skew the results for instructions per cycle (IPC) for multithreaded workloads –since IPC will not remain constant across all executions (Alameldeen & Wood 2006). However, in the calculations, the assumed instructions were included. For multi-programmed workloads, the standard procedure for prior work has been followed (Jaleel, Theobald, et al. 2010): 20 billion instructions were fast forwarded for every process, simulate until all threads have executed at least 256 million instructions. Then, only the first 256 million instructions of each thread were considered into account for IPC computations.

5.5.3. Replacement Policy

The ARIP scheme is used for AHRC, which is superior to the NRU scheme that was described by Seznec. Exact LRU replacement is used for the conventional caches unless specified otherwise.

5.5.4. Energy and Power Models

Static and the dynamic power of every design of LLC were estimated using per-access energies and leakage power reported in (Sanchez & Kozyrakis 2010). These per-access values were extracted from McPAT (Li et al. 2009) and CACTI 6.5 (Muralimanohar, Balasubramonian & Jouppi 2007).

The additional runtime that is caused by LLC and main memory accesses were estimated, to generate the full estimates of system energy. A 200-cycle delay is assumed for main memory accesses and a 10-cycle delay for LLC accesses. Then, the total LLC energy is calculated under these two assumptions by considering both static and dynamic power. A fixed 2W is assumed per core and the energy of the main memory is calculated assuming DIMM with a 2.78 W background power and 51nJ per access energy (derived from the main memory power model in (Deng et al. 2011)) when assessing the full power of the system. Then, the full energy of the system was presented as the sum of the calculated core, main memory energies and LLC. Other components of the system have been neglected in this presentation. These memory estimates and core have been chosen to approximate the system as assumed in (Sanchez & Kozyrakis 2010); that is, a large class of Atom-class cores which have a slightly small on-chip cache and a low memory footprint.

5.6. Results, Analysis and Evaluations

The demonstration begins at the improvements in performance over the 4-way Set associative cache. Figure 5.1 contrasts the rate of miss of AHRC associativity, and a conventional design of 16-way set associative that is normalised to the miss rate of a 4-way set associative cache. All the results are normalised based on the formula:

$$N = \frac{X-Y}{Y}100\% \quad (1)$$

Where N is the normalised value which is presented in the figures, Y is the value of the 4-way Set associative cache, while X is the value of other techniques. Each bar indicates the percentage improvement in misses-per-1000-instructions (MPKI) over the 4-way cache.

The bar to the extreme left in each group shows the design of the 16-way set-associative. AHRC is then shown in the third bar. All the results are reported as improvements over a 4-way set-associative cache with LRU replacement, whilst AHRC uses ARIP replacement.

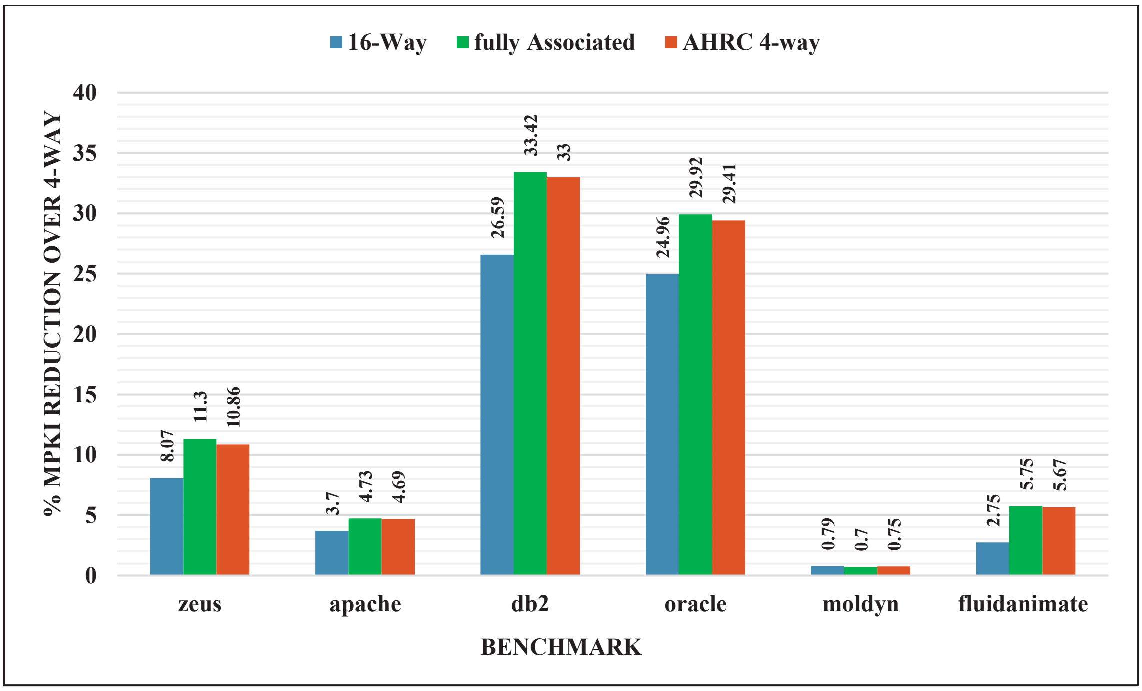


Figure 5.1 MPKI improvements over 4-way set associative caches for 16-way set associative, Fully Associative, and AHRC

The difference between AHRC and the fully Associative technique is negligible (see Figure 5.1). However, AHRC's concept is simplified and it involves less hardware overhead. Several workloads have proved to be insensitive to the replacement policy, thus gaining no benefit from multilevel replacement. For many workloads, single-level replacement (with 4-way lookup) is enough to match the performance of 16-way associative caches. Higher level lookups give the best marginal gains especially in commercial workloads scenarios, which (does not justify their energy and complexity overheads as shown below. The returns in performance are always diminishing because the replacement policy cannot effectively utilise the added associativity. Multi-level lookup can provide more substantial gains if the additional associativity is well utilised with an improved replacement policy and this is what makes AHRC superior to other designs.

Figure 5.2 shows how the benefits of multi-level lookup are associated with significant cost in average LLC power. In the same figure, the percentage increase in total LLC power is presented using a 4-way Set-associative cache (higher ways result in inferior results). When considering the full energy of the system, most of the added cache power is mitigated by energy savings from the improved performance. Figure 5.3 also shows that in a system that has a 1-DIMM main memory and 162 watt cores, AHRC gives the best trade-off for many of the commercial workloads. However, this benefit is not likely to warrant the complexity of additional implementation.

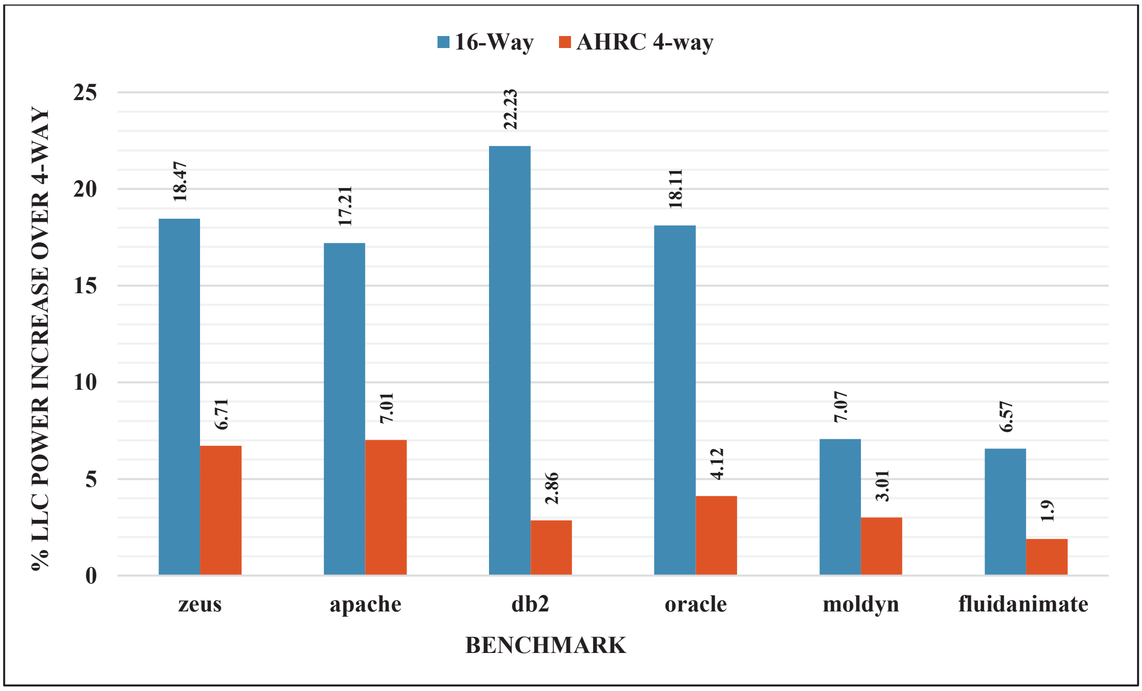


Figure 5.2 LLC power increase over 4-way set associative design

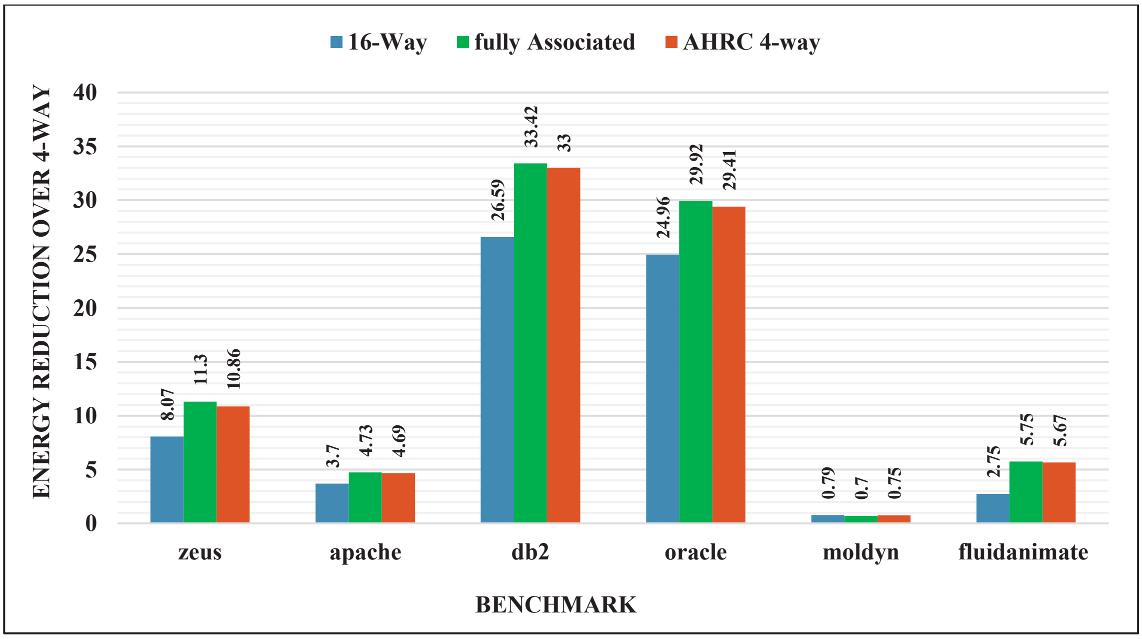


Figure 5.3 Full system energy reduction over 4-way set associative design assuming 2W per core

Notably, the ratio of LLC cache-to-core power that is assumed in this analysis is typical for a server-class system. Under the assumptions made in this analysis, the LLC cache is small (only 8MB for a 32nm process assumption) and only contributes a couple of percent of the total power of the system (the last-level cache has been reported as roughly 20% of chip-level power for Niagara 2 (Nawathe et al. 2008) and Nehalem (Processor)). Therefore, the power overheads of the LLC are often dwarfed by the core power and the energy savings in the main memory from the miss rates in our analysis. AHRC has also presented reductions in MPKI for a 32MB (Figure 5.4). The graph shows the results of 8MB in the sets of bars that are to the left, and 32MB results in the sets of bars to the right. The 32MB cache gives a greater improvement than the smaller designs. On balance, this study concludes that multilevel replacements of cache are likely justified for systems of the server-class. However, higher-level replacements will lead to a loss of energy.

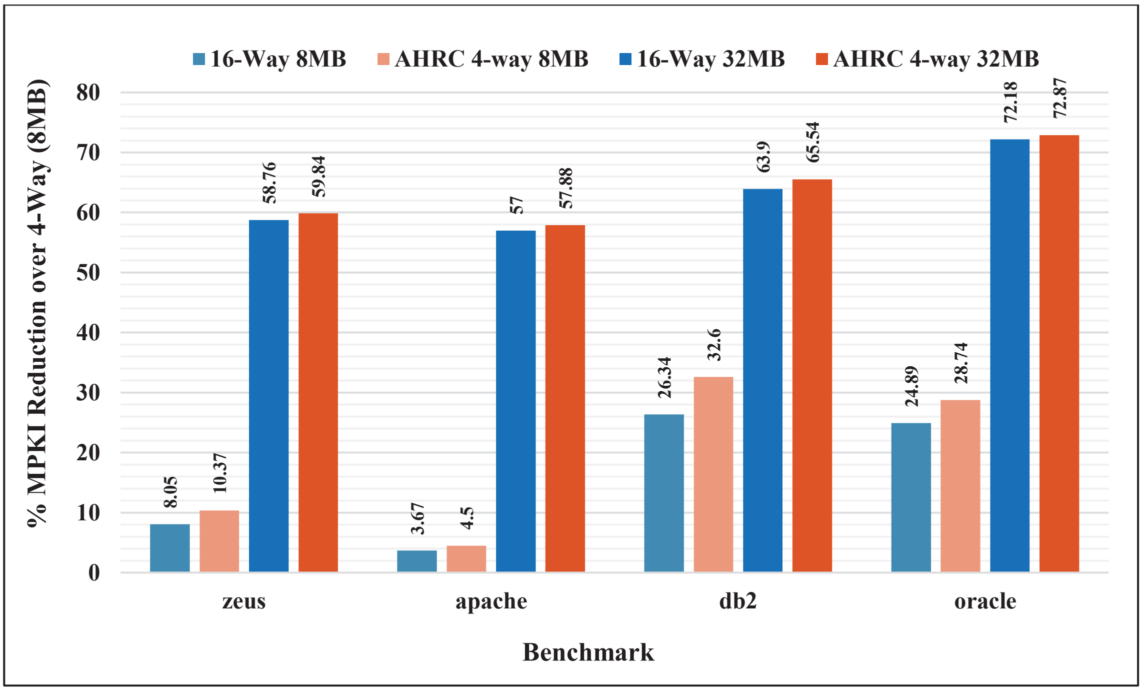


Figure 5.4 Comparisons of MPKI improvements over 4-way (8MB) set associative caches for 16-way set associative using LRU and AHRC using ARIP

5.7. Conclusion

This chapter has presented AHRC, which is a newly proposed design for caches, which use both multi-level replacements and hashed indices to achieve greater performance and associativity. An evaluation of using AHRC as the last level cache in a CMP proves that they give high connectivity with low overheads – regarding hit energy, hit time, and area. In both energy efficiency and performance, AHRC outperforms the traditional associative caches. A 4-way AHRC achieves higher performance than a 4-way set of associative counterparts, over a set of LLC miss-intensive workloads. Additionally, 4-way AHRC achieves better energy efficiency and higher performance than a 16-way set associative cache. AHRC does not emphasise the increase or reduction of associativity with the same hardware design. However, for workloads that are insensitive to replacement policy, AHRC with higher-level replacement results in a loss of energy.

There are several opportunities for further research available such as using AHRCs to design TLBs for multithreaded cores and build highly associative low-level caches. Also, policies of cache replacement that are specifically suited to AHRC, as ARIP still has space for further development.

Chapter 6 **PROI: Block-based Coherence Bypass Protocol**

6.1. Chapter Summary

Scalable cache coherence protocols are essential for CMPs systems, in order to satisfy the requirement for more dominant high-performance servers with shared memory. However, the small size of the directory cache of the increasingly higher CMPs cores may result in recurrent directory entries evictions. Accordingly, invalidations of cached data that will gravely corrupt system performance. According to prior studies, only a single core accesses a considerable fraction of data blocks, so it is needless to monitor these in the directory structure. Using the technique of uniprocessor systems and deactivating their consistency protocol is the best way to deal with those private blocks actively. The directory caches will stop the tracking of a substantial amount of blocks after the deactivation of the protocol, which will minimise their load and enhance their adequate size.

This thesis proposes the Private/Shared, Read-Only/Read-Write, Invalid/Valid (PROI): a scalable coherence protocol with minor modification on the level1's (L1s) and last level cache (LLC) tags, to distinguish between the private and shared data on block granularity level. PROI employs a dynamic writing policy with self-invalidation and self-downgrade for each L1 cache and is also able to maintain coherency at all times without performance degradation. PROI can scale with a raised number of cores and reduce area, energy, and performance associated costs with the coherence mechanism. PROI results show a significant reduction of almost 17% in miss ratio of private L1 cache, high network traffic reduction, a reduction in application runtime of approximately 6%. In addition, PROI has the capability to lessen the consumption of energy by almost 35%, on average.

6.2. Introduction

The last years have seen a rising demand for bigger and higher performing shared-memory multiprocessors (Conway et al. 2010; Le et al. 2007; Shah, Barreh, et al. 2007). These architectures have processors that speed up their memory access by employing one or more private cache levels that are transparent to the software, through a cache coherence protocol. Most shared-memory multiprocessors apply types of protocols founded on directories because they signify the most scalable strategy. The monitoring of system's memory blocks by conventional registers permits the simple identification of cached blocks without producing significant amounts of coherence traffic, as is the case in protocols based on a broadcast. Nevertheless, there is need of huge storage for monitoring all the memory blocks. A few latest proposals (Marty & Hill 2007b) and service systems like the AMD MagnyCours (Conway et al. 2010), have ways to avoid it as they only track cached memory blocks. Thus, small caches keep the directory information. Evicting the entry of a directory cache involves the invalidation of every cached copy of the related block because of a lack of filled directory (Cuesta et al. 2011). The increasing size of systems and the limited volume of directory caches can cause recurrent evictions of cache directory entries, which may lead to high miss levels (Jerger, Peh & Lipasti 2008a; Marty & Hill 2007b). Consequently, the miss levels of CMP caches may progressively increase, which results in severe performance deprivation.

Increasing the volume of directory caches leads to negative effects on both area requirements and access latency, which makes it a non-practical solution. As an alternative, it would be best to increase the efficiency of the directory caches' available space, supposing that it will be a limited resource, particularly in bigger systems. The fact that a considerable portion of the cache blocks that parallel workloads employ is private is the basis of our proposal to achieve this goal. Per our findings in a system with 256 cores, many of the accessed blocks are private because only one core has access to the memory blocks (60% on average) as seen on Figure 6.2. Even though there is no need for coherence maintenance when it comes to private blocks, directory caches still monitor them. Therefore, much of the kept information is unnecessary, which gravely interferes with their effectiveness. In contrast, not tracking private blocks would mean the increase of the accessibility of directory entries for the blocks that require coherence. It also means it would be possible to exploit the capacity of these directory entries in a more efficient manner.

Therefore, this study proposes a method that categorises blocks into shared and private blocks. Then, the coherency for private memory blocks will be disabled, which means directory caches will stop monitoring them. This study proposes a classification of private data at a fine granularity without the need of massive storage resources. The mechanism will consider the classification of all new blocks request from LLC as private by default. Thus, there will be deactivation of the coherence protocol for private data, that is, after cache misses, the retrieval of requested data from LLC will happen without monitoring and keeping track of cached copies. When there is a need to share a private block, the coherence mechanism helps to detect it and triggers a coherency recovery method that is responsible for restoring all blocks' coherence in the caches. From there on, the block becomes shared, which means it will start being tracked. This design only needs a data classification mechanism and a coherence protocol we called PROI, which has the capability to utilise the data classification efficiently. There is also no need for committed hardware structures since it exploits those already utilised by the CMP.

In order to satisfy the need for high-performance systems, it is essential to have efficient cache coherence protocols. Cache coherence protocols based on the directory are the modern approaches in many core CMPs to ensure that the data blocks are coherent at the final level private caches. Nonetheless, the increased amount of cores in a chip may interfere with the balance in regards to high consumption of energy, the area overhead, and high associativity needs of the directory structure. There have been proposals of varying directory organisations to allow increased scalability and lower overhead in coherence protocols depending on directories (Conway et al. 2010; Marty & Hill 2007b).

In this work, a runtime method is also employed to identify private data and further enhance the management performance of cache coherence in the system. More specifically, this study forms the cache coherence management founded upon the observation below:

Observation: *The granularity private accesses detection can play an important role in obtainable performance and energy saving benefits*.

The probability of discovering private data and more improvements will increase with a fine-grained examination of private data compared to page granularity. Figure 6.2 indicates the number of private accesses identified with a block granularity in comparison to a page granularity method for 14 dissimilar multithreaded applications in 256 cores system. The difference comes from the presence of one shared block in a page, which is adequate to alter other block's status in the page to shared. Whereas a page size of 8KB is used, it is expected the difference would be more striking for those architectures that employ page sizes of larger volumes for performance reasons. Figure 6.2 shows the greater likelihood to detect private accesses in block granularity as compared to a page granularity. This chapter will later discuss the possibility of carrying out block level detection with the PROI coherence protocol and its detection mechanism.

Previous research studies (Cuesta et al. 2011; Kim et al. 2010) proposed data classification methods using page granularity which help minimise coherence management transparency in directories, by not tracking each core's private blocks. However, suite to our observations, carrying out fine-grain data classification as opposed to page granularity can bring more benefits. As thus, this research makes the following contributions:

- Firstly, it would be best to use an aware directory based coherence protocol for data block classification that functions at a block level and assists in the detection of more private data blocks. Accordingly, it significantly minimises the block percentage that the directory needs to track, compared to the page level classification strategies. Therefore, there is room to employ smaller directory caches with reduced associativity in CMPs without affecting performance, thus helping the structure of the directory to grow with the rising number of cores.
- Secondly, this study proposes PROI: a scalable coherence protocol with minor modification on the L1s and LLC tags. PROI employs self-invalidation and self-downgrade for each L1 cache and is also able to maintain the coherency all the time without notable performance degradation.

The rest of the chapter is organised as follows: Section 6.2 highlights the cache coherence idea and present the main observation. Section 6.3 discusses the related work and motivation for our approach. Section 6.4 gives details on the background of directory caches, fast address translation, and private/shared data classification. Section 6.5 outlines the PROI protocol, its operations and states, and its detection mechanism, while Section 6.6 and 6.7 deals with the experimental setup and results evaluation. Finally, Section 6.8 concludes the chapter.

6.3. Related Work

There are two reasons for making coherence simpler in architectures with multiple cores. The first reason is to decrease the costs related to coherence (energy, performance, area) and the second is to improve scalability. Simplified coherence further enables varied accelerator architectures and CPUs used for a wide range of applications to effortlessly intersect under coherent joint virtual memory. To accomplish this, various recent proposals try to minimise directory cost (Alisafaee 2012; Cuesta et al. 2011; Ferdman et al. 2011b; Kim et al. 2010), while a few aim at make coherence simpler by getting rid of the directory restricted access in its entirety (Choi et al. 2011; Hossain, Dwarkadas & Huang 2011; Pugsley et al. 2010; Ros & Kaxiras 2012). A general helpful tool used by such approaches is data classification into shared and private (Choi et al. 2011; Ros & Kaxiras 2012), making the best use of the coherence protocol itself by employing silent self-invalidation rather than open coherence invalidations of shared data after synchronisation (Ros & Kaxiras 2012), or causing the reduction in size of the directory (Alisafaee 2012; Hossain, Dwarkadas & Huang 2011; Pugsley et al. 2010).

It is possible to perform data classification by OS (Cuesta et al. 2011; Hardavellas et al. 2009; Kim et al. 2010), Compiler (Li et al. 2010; Li, Melhem & Jones 2012), or hardware methods (Hossain, Dwarkadas & Huang 2011; Pugsley et al. 2010). While gaining the lowest hardware cost, classification by compiler is not clear to the software.

It is compatible with software/hardware co-design, but needs recompilation and recoding for legacy software and noteworthy effort to find out if there will be sharing of a variable at compile time. The approach of the OS does not enforce additional needs for dedicated hardware, as there is storage of page granularity data classification along with the Page Table Entries (PTEs) (Hardavellas et al. 2009). It is thus a proper choice for intricate effective optimisations. Nevertheless, it experiences adaptation and granularity issues, causing misclassifications that augment with time, thus corrupting classification quality.

The basis of our proposal is the observation of the private status of most parallel applications blocks. The exploitation of this fact will help us in proposing a hardware mechanism that does not keep track of those private blocks. Here, this section comments on a few studies that have some relation to our proposed approach. A few authors utilise the OS to identify shared and private pages. Hardavellas et al. (Hardavellas et al. 2009) employed this identification method to propose a competent data placement approach for disseminated shared caches (NUCA). However, the detection mechanisms are different, Hardavellas' application differs completely in data placement. In addition, it has no regard for coherence features.

Nevertheless, more attention is given to the use of identification of private and shared blocks in enhancing scalability and directory effectiveness. Conversely, Kim et al. (Kim et al. 2010) utilise OS detection to decrease the snoops' fraction in token protocols. The basis of that work is on the idea that, even though utmost blocks are identified as private, the tiny portion of shared blocks takes into consideration cache misses (Cuesta et al. 2011). Therefore, they suggest a refined way that identifies the shared data and their level of sharing. Thus, there is replacement of broadcast messages by multicast ones. The problem is that the proposal needs more hardware (significantly larger TLBs) and increase operating system overhead to achieve significant improvements.

In a different way, our mechanism is quite simple and does not need modification to the OS or complex modifications to the hardware. Additionally, instead of focusing only on cache misses of blocks with shared status, this study focuses on both portions of private and shared blocks. The effectiveness of our proposal is in reducing the number of tracked blocks while sustaining system performance.

A few other works employ the coarse-grain approach to monitor blocks, at the cost of causing higher storage needs to eliminate the redundant broadcast traffic. The suggestion of Region Coherence Arrays and RegionScout filters by Cantin et al. (Cantin, Lipasti & Smith 2005) and Moshovos et al. (Moshovos 2005) respectively, is because they offer dissimilar tradeoffs between implementation and accuracy costs. While RegionScout filters are less complex and require less storage, Region Coherence Arrays are more precise detecting shared regions and sort out more redundant broadcast traffic. On the other hand, RegionTracker offers a structure of coarse-grain enhancements that lessen the increase in storage and avoids the vagueness of earlier studies (Zebchuk, Safi & Moshovos 2007). However, the method involves extensive modifications in the design of the cache to enable region-level lookups. There are major differences in the fact that our proposal does not need the help of the OS, and does not increase the hardware complexity and storage overhead. The objective of our proposal is not only to reduce broadcast traffic, but to prevent the allocation of data blocks that do not need coherence maintenance. However, every one of the above techniques shares with our proposal: the concept of deactivating the mechanism of coherence when it is not crucial.

According to (Davari et al. 2015), the effect of traffic caused by a write-through cache is significant, resulting in benefits in general network traffic and, as a result, lower consumption of energy if used wisely. There is a use of adaptive methods related to intricate dual-grain by earlier works to identify shared and private areas, with the aim of lessening the directory size (Alisafaee 2012; Zebchuk, Falsafi & Moshovos 2013). This study aim, on the contrary, is to provide scalability and strong data classification.

In addition, other works exploitation of OS structures such as Ekman et al. (Ekman et al. 2002) who propose energy reduction by snooping for CMPs. This technique ensures that each TLB entry has a sharing vector to show the processors that share a page. The system transmits the sharing vector on each snoop request and stops any processes that do not need to share the page from doing a tag look-up within their caches. Consecutively, Jerger (Jerger, Peh & Lipasti 2008b) widens the tracking structure to monitor the current series of a region's sharers which proposed by Zebchuk et al. (Zebchuk, Safi & Moshovos 2007). Unfortunately, these methods involve significant hardware modifications and raise the storage requirements that make their application in real systems difficult to achieve. The need is to increase system performance, which will not work with less storage space. To help in reducing storage needs, our method only plans to keep information regarding the sharing degree of a block in the directory and deactivates the monitoring of private blocks.

There are more works that sustain cache coherence through the mechanism of combining hardware and software. Zeffer et al. (Zeffer et al. 2006) recommend a trap-based memory architecture (TMA) that identifies in hardware the violations of fine-grained coherence. It then activates a trap of coherence, in case of an occurrence and maintains software coherence in handlers of coherence. The trap-based considers a shared/private functionality in TLBs and depends on the operating system to notice when the private page becomes shared. The only issue is that TMA associates traps with coherence violations within operations of store or load (Cuesta et al. 2011). Moreover, TMA encourages more handling of the coherence trap by requiring that each processor gets more hardware support. As an alternative, there is the suggestion of a simple hardware device that implements in software a protocol of inter-node coherence (Zeffer & Hagersten 2007). To accomplish it, first coherence related to inter-nodes must verify the need to call up the software-coherence protocol. Next, there must be management by memory controller of evicted dirty remote-data from the LLC. This means higher software overhead although the hardware overhead is small.

Finally, Fensch et al. (Fensch & Cintra 2008) suggests a coherence format for tiled CMPs where there is a division of the coherence task between the hardware and the OS. In their format, there is avoidance of incoherence through disallowing the replication of data across the tiles. The achievement of this is through having the OS map pages divide private level-one (L1) caches, and having the hardware permit a partial and controlled migration of data via remote accesses of cache. The issue is the dependence of such approaches on the OS for system movement and data mapping, needs regularity in release and brings in more hardwired systems overhead, even though they do not affect the application software. Lastly, hardware and software differ when it comes to cache-block or page granularity because hardware methods work completely transparent to the software. But hardware methods can have prohibitive storage needs (Hossain, Dwarkadas & Huang 2011; Pugsley et al. 2010) or complexities in dual granularity (Alisafaee 2012; Zebchuk, Falsafi & Moshovos 2013).

6.4. Background and Motivation

This section reviews important points of what is acknowledged in the field of cache coherence and identify gaps or problems to address; providing a justification for the approach taken in the next section.

6.4.1. Directory Caches

Cache coherence protocols based on the directory have area and power consumption scalability, which makes them the best for controlling the coherency in a majority of core systems when compared to conventional protocols based on broadcast (Agarwal et al. 1988b; Gupta, Weber & Mowry 1990b). Nonetheless, the existing multi-core architectures with their needed power and latency and big last LLCs have come with fresh challenges. Caching is quite important in multiprocessor architectures as it plays two vital roles of uniprocessor and memory utilisation. As a uniprocessor, caching adds to reducing the time of instruction service and decreases the congestion in the memory in memory utilisation. Therefore, caching a division of directory entries prevents directory accesses' high power and latency overheads is commonplace. They give motivation to architects to cache a separation of entries. It is the aim of a directory cache to offer an effective means of storing data blocks copies in diverse coherent private caches, since its organisation can significantly affect overall system performance (Gupta, Weber & Mowry 1990b). The first vital decision related to the design of a directory cache is choosing the data blocks that need tracking. The second is the associativity and size requirement for a proficient directory cache.

Having directories with a duplicate tag is a common method for directory organisation in CMPs (Andr et al. 2000; Nanda et al. 2001). In contrast with other structures of the directory, this form of directory caches has more flexibility because they do not force any insertion among the levels of a cache. However, such directories come with overheads. The two main overheads are high associativity need that increase with system's core number and the cost of storage for duplicate tags. For example, a multi-core processor that has N cores, each with a K-way-set associative private cache, the directory cache must be N×K-way associative to sustain every private cache tag to prevent any invalidation. Therefore, this method is not suitable because of the high consumption of power and increased complexity in design due to high associativity. Our proposed framework avoids poor performance related to associative directory caches (caused by high counts of invalidation in the directory cache), as shown in Sections 6.4 and 6.6.

6.4.2. Fast Address Translation

TLBs are the main element for supporting fast virtual to physical addresses translation. However, in the majority of cases they are in the vital memory access path, which makes them the central motivation for numerous studies (Bhattacharjee & Martonosi 2009; Srikantaiah & Kandemir 2010) that focus on the methods for efficient and speedy translation from virtual to physical address through TLBs.

Private TLBs for every core is a pervasive way for categorising the TLBs in several core architectures. However, (Bhattacharjee, Lustig & Martonosi 2011) shows that such categorisation may cause poor performance and still maintain the TLB in the vital memory accesses path. The suggestion of a shared last level TLB (Bhattacharjee, Lustig & Martonosi 2011; Bhattacharjee & Martonosi 2009) to enhance sharing capability among the system cores is one way to overcome this. Another contrasting technique is attempting to raise sharing capacities between the cores by allowing a collaboration between the TLBs that are private (Srikantaiah & Kandemir 2010). In short, every individual core attempts to borrow the capability of private TLBs in different cores in the system previous to discovering the translation with a result of a higher cost to the OS.

With the shared TLB approaches, there is a need for higher access latency and interconnection network of a high bandwidth. Consequently, they require higher associativity, which will cause higher consumption of power and that is not an option. On the other hand, the second approach needs to query other TLBs for every missed page translation occurred at each core, which can initiate both increased network traffic and design complexity. For instance, a previous study proposes snooping different system TLBs for finding the page entry in them (Srikantaiah & Kandemir 2010). The approach lacks scalability with increased core counts because of high energy utilisation and traffic overhead that snooping needs.

6.4.3. OS and TLBs Private/Shared Data Classification

Many of policies employ private/shared type of data classification to make coherence simple for data-race-free (DRF) semantics (Cuesta et al. 2011; Hardavellas et al. 2009; Kim et al. 2010). It is possible to eliminate private data from the coherence process to result in the reduction of the directory size. Whereas a few approaches depend on private/shared data that the OS provides based on a page, others up hold a multi-granular method of classification in hardware. The page table identifies a page as private when only one core accesses it, and when a different core accesses the same page, the status changes to shared.

The problem with classification on the page level is that there is a misclassifying of cache blocks that are private as shared within shared pages. Even when different cores access different cache blocks, the page still becomes shared. The transition of a page from private to shared leads to notification of the first core that tagged the page as private, to change the page's local classification. This process is necessary as the local classification dictates the employed write policy for a page's cache blocks. The transition of a page can occur once at maximum, and once a page becomes shared, it can no longer revert back to being private.

The fact that the majority of write misses emerge from private blocks in a write-through protocol is a major observation that makes protocols such as VIPS-M realistic (Kaxiras & Ros 2012). It is clear from the observation that VIPS-M employs an active write policy in the L1s: write-through for shared data and write-back for non-coherence private data. As a further optimisation, there is employment of a write-through buffer to combine writes to similar cache blocks. The emptying of this buffer happens at similar synchronisation points that cause self-invalidation. It also employs a similar OS classification technique to differentiate between Read-Only and Read/Write pages. There is usually no self-invalidation of cache blocks of Read-Only pages at synchronisation.

6.4.4. Hardware Private/Shared Classification Approaches

Numerous systems depend on the classification of hardware as shared or private for different reasons, the main one being the size reduction of the directory (Alisafaee 2012; Fang et al. 2013; Zebchuk, Falsafi & Moshovos 2013). To accomplish it, it is necessary to employ a multigrain directory that monitors coherence data on over one block size. The same is the recommendation of Alisafaee (2012) and Zebchuk et al. (2013), but their approaches vary on implementation. The system stores private areas in the directory and extracts shared cache blocks out of them. Likewise, Fang et al. (2013) approach describes private region entries with either shared or private block entries. The private ones have a different region as the owner, while the shared ones have several cores accessing them.

Pugsley et al. (Pugsley et al. 2010) establish an adaptive description of their SWEL protocol, referred to as Reconstituted SWEL (RSWEL) in correspondence to classification adaptivity. The SWEL protocol entails a saturating decay counter that is 2-bit, for the reclassification of pages from shared to private. This is mainly because the lack of adaptive classification that permits the reclassification of data to private from shared leads to the misclassification of all system data as shared, which gets rid of optimisations related to the private data. Still, it is quite an intricate adaptation that happens periodically in bulk, needing elaborate time regulation that marks the decay counters for proper results. In addition, Zhao et al. (Zhao et al. 2013) concentrate on cache coherence adaptation by suggesting the Protozoa coherence protocol. The problem is that Protozoa deals with adaptivity in data movement and coherence granularity. It adopts varying granularities for mitigating data movement transparency caused by coherence invalidations, rather than having set cache-block granularity for both coherence and data storage/movement actions. On the other hand, our approach deals with adaptivity when applied to private/shared data classification. Lastly, POPS puts forward a protocol that is most effective with shared or private data to satisfy the exchange between a shared and private LLC, but handles migrating data as private (Hossain, Dwarkadas & Huang 2011).

6.5. PROI Approach

Our assumed system comes with private L1 caches that request for data from a shared LLC. The point of our classification method will be within the Directory cache, since it will be observing all system requests. PROI protocol updates data classification together with LLC's data responses. A L1 miss result will mean the beginning of a new life of cache block. Therefore, there can be access to a cache block in many cores. It is possible for a cache block to have repeated access from a core before it goes into a *dead* time pending eviction. The eviction of a cache block means the end of its life from the cache due to its non-use. PROI is a directory-based coherence protocol designed with dynamic write policy (write-through for shared blocks and write-back for private blocks), self-invalidations, and no transient states in order to achieve a scalable coherence that fulfils the coherence definition of Single-Writer/Multiple-Readers (SWMR) (Sorin, Hill & Wood 2011) without constraints. PROI use Invalidations to tolerate only one writer at a time. PROI is founded on an analogous technique to VIPS due to its simplicity. However, it designed with scalability and a block data classification in mind as a directory-based protocol.

A block might still be in active utilisation in case of its replacement, but its eviction happens because of a cache conflict with partial associativity. Nevertheless, the end of life of a cache block can have substitute interpretations. A dead-block prediction/detection method like cache decay or a program instruction can openly signal termination, or it can happen through more sophisticated predictors of dead-block. Signalled termination provides a means of managing the length of cache block life through adjusting its dead time. PROI will use this feature to reveal this ability by employing a simple predictor like cache decay. Coherence invalidations are other events that can be reliable when it comes to establishing when to terminate the life of a cache block. The invalidated cache block might actually be in active use and the same core is going to request it in a while. For this reason, the termination of a cache block in PROI approach will only happen with the eviction of that block.

The existence of a single life of a cache block during all L1 caches classifies it as *private,* and it becomes *shared* if it has two or more overlapping lives. The best way to monitor the life of a cache block is by making the start and end of it visible to the classification method. An L1 miss result is the start of a cache block life, and a new LLC request marks it as such. The principal coherence protocol defines the end of life detection. PROI coherence protocol uses the PUT message for a write-back of block replacement and to mark the end of a cache block life.

PROI stores information related to data classification in the Directory cache. Instead of using a *sharer* field, PROI uses a small *keeper* field that contains the quantity of sharers for a shared classification block or the ID of the private core for a private block. The length of a keeper field is $\log_2 n$, in which *n* represents the number of cores (in this study, 8-bits for 256 cores). This is much smaller when compared to the size of the sharer field (1-bit for every L1 cache, which means 256-bits for a 256-core machine). PROI protocol is more scalable with the rising number of cores. In addition to the *keeper*, a 2-bits *state* field is used to distinguish between Private/Shared and Write/Read blocks to help to keep coherence between the caches as shown in Table 6.1. Shared/Private bit keeps tracking of the blocks if they are used by one or more cores. Read/Write bit keeps tracking if the block is read-only or has been written to. Also, there will be a 1-bit for each cache block in L1 for invalidations. Figure 6.1 introduces the PROI states diagram.

At the start, the entries have a null classification in the *keeper* field and *state* field. The *state* field will represent a *private* state after the initial L1 request, and the *keeper* field becomes the requesting core's ID. More access to a private cache block will change that classification to *shared*, and the *state* field will represent that. The *keeper* field will now have that cache block's number of sharers. The classification information may be updated as indicated by the algorithm 6.5.1.

A GET request for a shared cache block maintains the classification as *shared*, while the *keeper* field changes in value to reflect the sharing degree for that cache block. The message of acknowledgment for any coherence invalidation or write-back at the LLC for a shared cache block will also change the value of the *keeper* field. The block will stay as *shared* in case the degree of sharing in the *keeper* field becomes '1', unless a write-through for a cache block in a valid state occurs to reveal the owner of that block, then the block status will revert from *shared* to *private* by changing the *state* field, and store the core's ID into the *keeper*. This means it is possible for it to change back to *private* when the sharing degree is equivalent to 1.

Classification	**State**	**Comments**
Private	Private Read (PR)	Use write-back
	Private Write (PW)	Use write-back
Shared	Shared Read (SR)	Use write-through
	Shared Write (SW)	Use delayed write-through

Table 6.1. Protocol states and classification

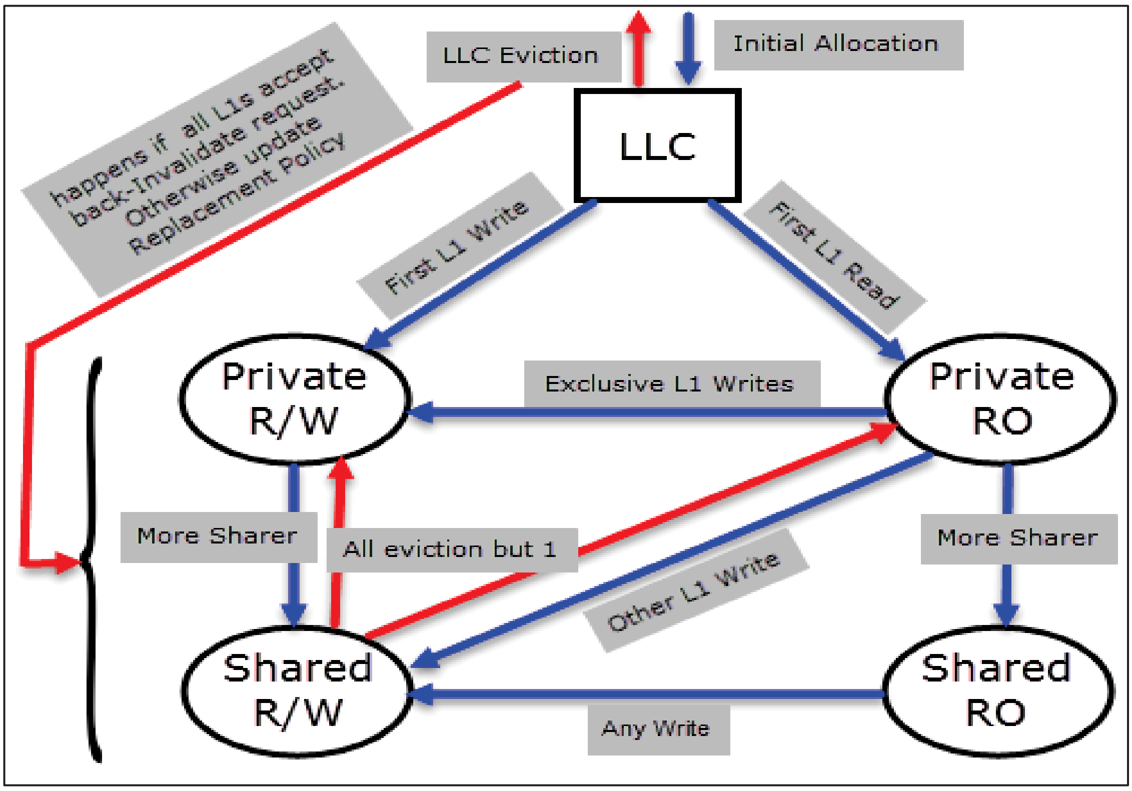


Figure 6.1: State Machine Diagram for the PROI protocol

PROI applies a coherence with self-invalidation that decreases network traffic by getting rid of the invalidation traffic and enhancing the overall performance. Where a core can self-invalidate any block in its L1 by setting the *invalid* bit for that block and it can be detected in future lookups by checking the *invalid* bit which shows if the miss was a coherence miss or from capacity, conflict, or compulsory group of misses.

There will be writing-back of private blocks with modified data into the LLC, which will mean the end of their life. However, there is no need for explicit eviction notification (EEN) when it comes to clean private cache blocks, because there are private blocks that maintain their status to a single core for the entire execution of the program.

Algorithm 6.1 classification information in PROI approach

New request

```
if Keeper is clear, then
        Keeper ← requester
        Its Invalid bit (I) ← clear
        Private-bit ← set
else if Private is set, then
          if Keeper = requester, then
               I ← clear
          else if keeper = other ID with invalid state
               keeper = requester
               I ← clear
             else
                 Keeper ← 2
                 I ← clear
                 Private-bit ← clear
             end if
          end if
        end if
     else
          Keeper ← Keeper +1
          I ← clear
     end if
```

Invalidation (for clean private and RO shared)

```
       I ← set
```

Eviction

```
If shared & RW, then
Keeper ← Keeper - 1
```

Access to a private cache block will mean a comparison between its *Keeper* field and the ID of the requesting core. The cache block stays private if the ID matches with the one in the *Keeper* field. This is where a silent eviction of a private-clean cache happens because of conflict/capacity misses and repeated requests by the former private owner. In the case of a difference in the IDs, the protocol will notify the previous private owner before responding. This type of recovery notification occurs only once in a unicast form for every transition from *private* to *shared*, or changing the *keeper* ID while keeping the private state as private – as only the previous private owner needs notification.

There are two possible scenarios: the cache block changes from *private* to *shared* if it still exists in the cache of the previous private owner. Consequently, the previous owner either carries out a dirty data write-back or transmits an acknowledgment if the private block is clean. The protocol then changes the cache block to shared and the value of the *Keeper* field becomes 2. However, the previous private owner gives a negative acknowledgment (NACK) if there has been a silent eviction of the cache block, which means its end life. In here, the protocol maintains the classification of the cache block as *private*, and the *Keeper* field writes the new owner's ID. The LLC must access the memory prior to responding with to the new requester, in case there is the eviction of data from the LLC as an end of life because of an eviction and not invalidation.

Read-only (RO) *shared* data does to go through self-invalidation just as *private* data, which yields an increased hit rate. The classification of a cache block as RO only happens in the case of no *write* observation in the cache block's LLC. The LLC will notify the cores with the cache blocks that have the first *write* to update them as read-write (RW) from RO. This operation is neither on the critical path or costly because it occurs only once for every transition from RO to RW.

Our approach uses inclusive cache hierarchy, which means all L1 block copies will be evicted if LLC evicts that block. A broadcast is required in case the block was shared. For this reason, PROI uses a similar technique to the Query Based Selection (Jaleel, Borch, et al. 2010) that prevents replacing cache blocks that are still in use in core caches. This is done using a Back-Invalidate request from LLC to L1 caches. If core caches reject the Back-Invalidate request, the replacement in LLC will be updated to keep the block as recently used.

6.6. Experimental Setup

This section deals with the experimental setup and discusses the evaluation methodology.

6.6.1. System Setup

Our proposal evaluation uses the Simics full-system simulator (Magnusson, Christensson, Eskilson, Forsgren, H, et al. 2002), with the Wisconsin GEMS toolset (Martin et al. 2005b) that enables comprehensive CMP simulation. GEMS toolset include GARNET (Agarwal et al. 2009), which is a detailed on-chip network model simulator. Energy consumption of the simulated system has been calculated using the CACTI tool (Muralimanohar, Balasubramonian & Jouppi 2009), with a 32nm process technology. Table 6.2 provides the main parameters of the simulation environment.

Processor Frequency & Core Count	3.0GHz, 256 cores
Split Private L1I & L1D	32KB, 4-way associative, 64B cache block size
L1 hit time	1 (tag) & 2 (tag+data) cycles
MSHR size & timeout	16 entries, 1000 cycles
Shared unified LLC	16MB, 16-way
LLC hit time	2 (tag) & 4 (tag+data) cycles
Off-chip access time	160 cycles
Cache hierarchy	Inclusive
Page size	8KB
Network	2D mesh (4x4)

Table 6.2. System Parameters

6.6.2. Benchmarks

A wide range of shared-memory parallel workloads that need coherence were used from several suites including SPLASH-2 (Woo et al. 1995), SPEC and PARSEC (Bienia et al. 2008b), to evaluate this study proposal, as shown in Table 2. Simulating these benchmarks is done with small to medium working sets because of practical time constraints. As shown in Table 2, The smaller data-sets is assumed to simulate the application as is the case for most works (Cantin, Lipasti & Smith 2005; Fensch & Cintra 2008; Jerger, Peh & Lipasti 2008a).

Benchmark	Application domain	Input Specifications
SPLASH 2 (8)		
Cholesky	High-Performance Computing	Tk15.O
FFT	Signal Processing	64K complex double
FMM	High-Performance Computing	16K particles
Ocean	High-Performance Computing	258×258
Radix	General	16K integers
Raytrace	Graphics	Teapot
Waternsq	Molecular Dynamics	512 molecules
Watersp	Molecular Dynamics	512 molecules
PARSEC (5)		
Blackscholes	Financial Analysis	simmedium
Bodytrack	Computer Vision	simmedium
Canneal	Engineering	simmedium
Fluidanimate	Animation	simmedium
Swaptions	Financial Analysis	simmedium
SPEC (1)		
Tomcatv	Vectorized mesh generation	256 points

Table 6.3. Workloads and Input Specifications

6.6.3. Evaluation Metrics

MESI, the directory based cache coherence protocol, is used to do a comparison of PROI approach. The exact tested metrics are the classification quality, miss ratio, network traffic, execution time, and the dynamic energy consumption.

To test PROI approach, its performance is measured with the associativity of directory cache equal to 4 with 64 set in the base setup. Every reported experimental outcome corresponds to the standard parallel phase. Several simulations runs were performed for every benchmark, and small arbitrary perturbations were inserted in memory system timing to account for the inconsistency in multi-threaded workloads (Alameldeen & Wood 2003).

6.7. Performance Evaluation

This section involves the revelation of the fact that this study proposal can lessen the number of blocks that the directory cache may track. It results in energy saving, which is a huge objective, and there are fewer cache misses by the processor that leads to performance improvements.

6.7.1. Private Blocks classification

The basis of PROI is the reality that a considerable quantity of accessed blocks during the parallel workloads execution is private. Figure 6.2 indicate the portion of memory blocks that CMP refers to that only one of the cores uses (i.e. private). Our observations revealed that almost 60% of the data are private on average. PROI attempts to detect all private data. Since it functions at a block granularity rather than at page level, hence it is possible to detect every private block as such because the page-based method will regard the entire page as shared if private blocks accessed by several cores. Therefore, Figure 6.2 show the portion of data that PROI mechanism identifies as private leads to 3 times more than the page-level classification.

6.7.2. Private Cache Misses

Among the key advantages of decreasing the directory cache eviction is the reduced invalidations of the private caches, which is the L1 cache in our system. This is mainly because of the fact that any cache block eviction means invalidation of blocks within any of the L1 caches. Figure 6.3 indicates the miss ratio of the cache in L1 for 14 dissimilar applications with multiple thread benchmarks with regards to the base case. PROI approach has a reduction of almost 17% in miss ratio of the private L1 cache.

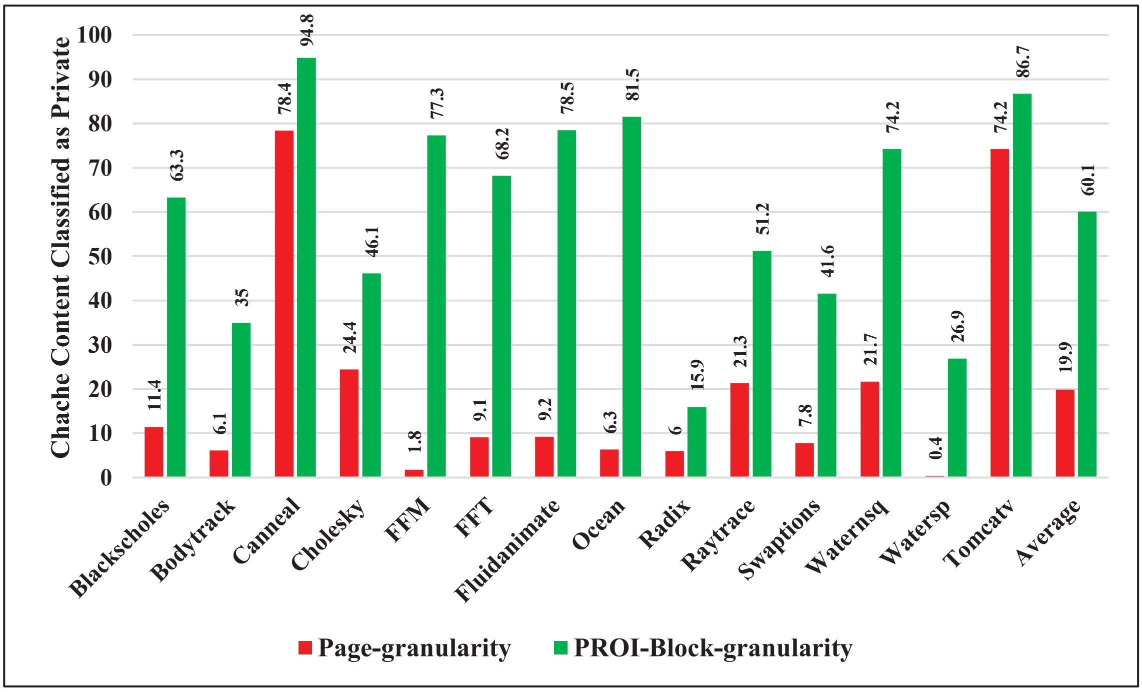


Figure 6.2: Percentage of detected private date with page and block granularities

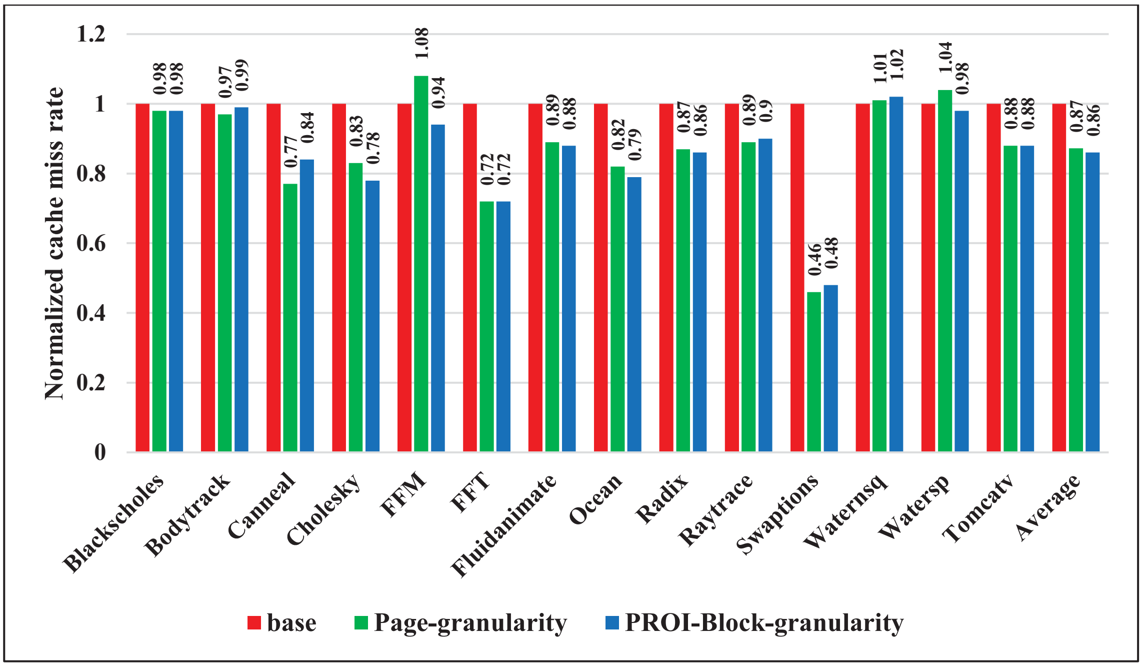


Figure 6.3: Cache miss rate normalised with respect to MESI

6.7.3. Impact on Network Traffic

Figure 6.4 can explain the network traffic decrease (23% on average) without a reduction in the miss level. It reveals the amount reduction of write-through traffic using PROI. Write-throughs are a huge network traffic contribution in protocols that use the approach of write-through for shared data. Thus, block granularity will lead to the decrease of network traffic by reducing shared data volume. Workloads like *Cholesky, Ocean* and *Tomcatv* extensively gain from finer granularity, because there is a reduction of write-through traffic. Other like *Blackscholes*, *Raytrace*, and *Bodytrack* do not have a big reduction of network traffic despite the effect of write-throughs. For those workloads, there has been quite an increase in the quantity of write-back, which eliminates the advantage of having reduced write-through traffic. This is an obvious feature of larger datasets such as these, which enforce regular replacements in L1 caches. *Canneal* has a very little volume of shared data which makes it insensitive to write-through.

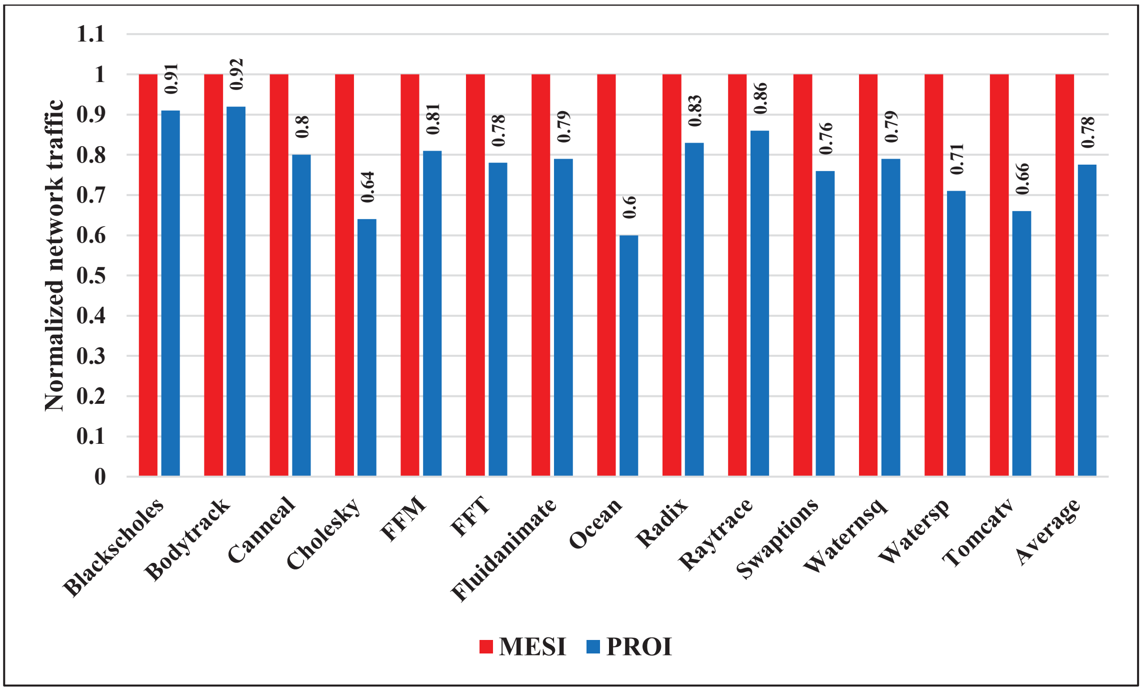


Figure 6.4: Network traffic normalised with respect to MESI

Generally, write-through traffic affect benchmarks more as compared to data traffic attributable to cache misses. Nevertheless, reduced movement of data does not cause network traffic reduction because of the lower rate of misses, but instead from noteworthy reduction in the volume of write-through. The decrease in cache misses is so remarkable that there is a reduction of coherence traffic as interpreted from Figure 6.3 and Figure 6.4.

6.7.4. Execution Time

It is possible to considerably reduce applications' runtime because of the decrease in the latency and amount of cache misses, as depicted in Figure 6.5. According to the data, PROI can result in reduction in application runtime of approximately 6%. The reduced execution latency of the applications comes as a result of the decrease in the amount of cache misses and less latency in their resolution, and the reduction of network exchanged messages. The critical parts of storage requirements in the system can also be exploited in PROI as it is possible to drastically decrease the directory cache size while achieving good performance.

6.7.5. Impact on Energy Consumption

Because of the decrease in network traffic and cache misses, it is possible for PROI approach to reduce energy consumption in the system. Figure 6.6 shows the active consumption of energy in regards to memory controllers, directory caches, and the interconnection network.

The energy consumption of private data requests is lower because they do not require access to directory caches. Additionally, it is a noteworthy reduction as the size of the directory cache reduced because they have lower access latency that offsets more accesses. Considering common utilisation of data classification, directory caches, and the interconnection network, PROI has the capability to lessen the consumption of energy by almost 35% on average.

The application's execution time is noticeably close to static energy consumption, but Figure 6.6 does not show this. Specifically, less consumption in static energy when it comes to memory controller and that of the network is directly proportional to runtime reduction (Figure 6.5). The static energy of directory caches relies on both their size and application runtime. Therefore, the employment of smaller caches can reduce the static power consumption significantly.

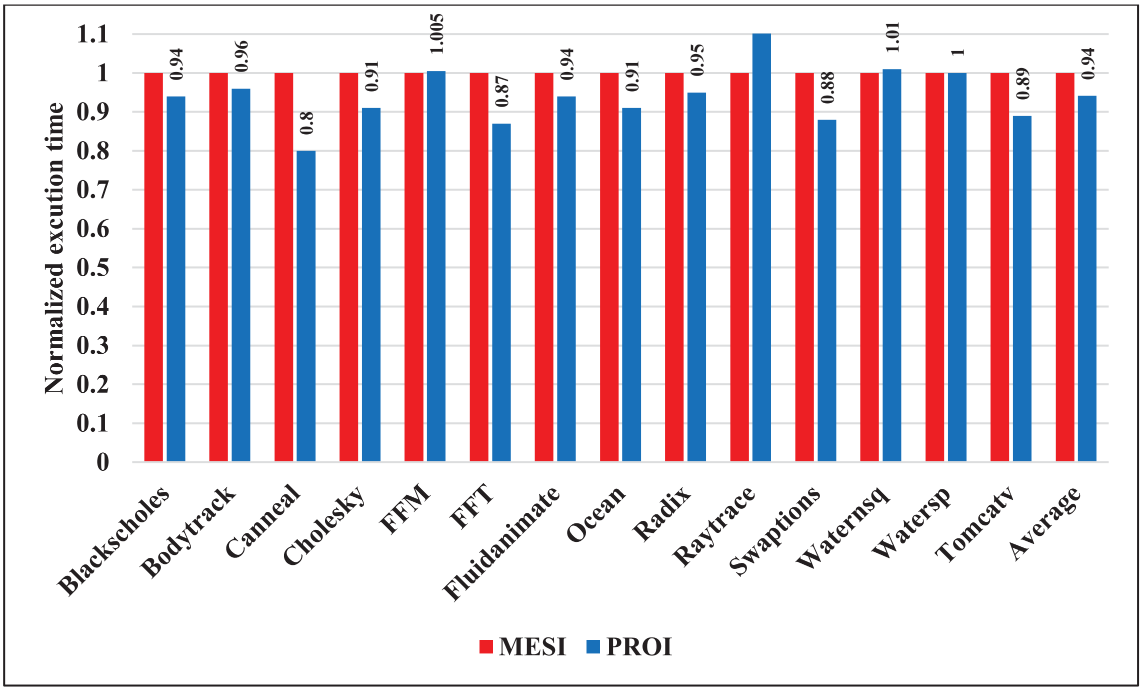


Figure 6.5: Execution time normalised with respect to MESI

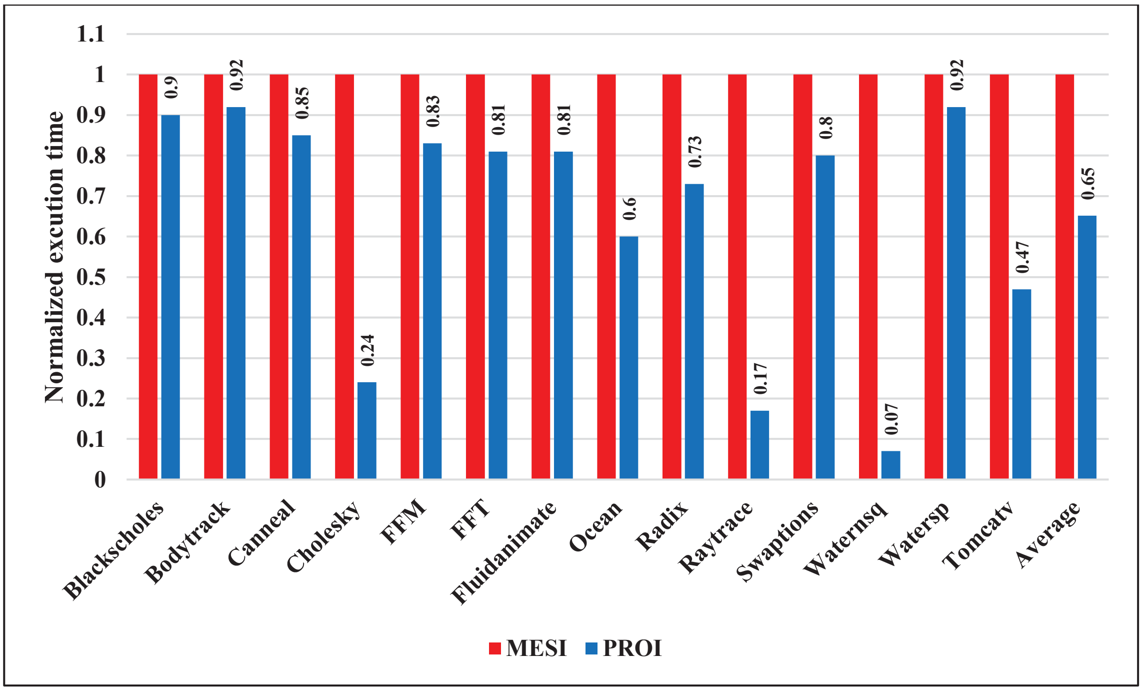


Figure 6.6: Energy consumption normalised with respect to MESI

6.8. Conclusions

Cache coherence protocols in the directory are the common approaches for controlling the coherency in a majority of core systems due to their area and power consumption scalability, when compared to conventional broadcast-based protocols. All cached architectures have writable shared information, which causes the issue of Cache Coherence. The problem with classification on the page level is that there is the misclassifying of cache-blocks that are private as shared within shared pages. This study reveals that it would be best to use a data block classification method that functions at cache block level and assists in the detection of more private data blocks. PROI focuses on a simple approach that does not need complex modifications to the hardware or OS. Studies show that benchmarks are less sensitive to data classification of the fine-grained approach in regards to miss rate. Therefore, instead of focusing only on cache misses of blocks with shared status, PROI has more focus on both portions of private and shared blocks. The proposal successfully shows the reduction of the amount of blocks that the directory cache will track. PROI leads to energy savings, which is a huge objective, and there are fewer cache misses by the CMP that leads to performance improvements. Furthermore, it is possible to considerably reduce applications' runtime in case of the decrease in the latency and amount of cache misses.

Chapter 7 **Conclusions**

This chapter highlights the main points of the thesis. It will summarise the key concepts of the chapters, followed by elaborating the contribution, limitation and future work of the dissertation.

7.1. Summary

In this thesis, the problems of cache scalability in many-core chip multiprocessors have been discussed.

Part one of the thesis presents the research goals and formulating the hypothesis. The dilemma of the cache scalability in many-core CMPs systems is the main research theme of this thesis. We present a background on the cache associativity, replacement policies, and cache coherence concepts in CMPs systems and an overview of prior solutions for cache scalability. Moreover, a discussion of appropriate simulation tools, performance metrics, methodology, and workloads used for evaluation in this thesis have been handled in detail.

To establish this theme, in part two of the thesis an action study of coherence mechanisms that insight the different cache coherence approaches in the CMP era have been studied. Then, it focuses on research contributions which include approaches and new ideas that explain the experimental process needed to validate the thesis aims and hypothesis.

In summary, the conclusion reflects on the accomplishments of the initial aims and the targets explained in Chapter 1 on page 10, along with the comprehensive review and discussion on existing cache associativity, replacement policies, and cache coherence approaches in Chapter 2.

This study depicts the details of the methodologies and approaches needed to run the simulations of the experimental work. The evaluation of appropriate simulation tools, performance metrics, methodology, and workloads used for evaluation have been examined. Section 3.5 summarise the research methodology applied in this dissertation.

Based on the literature review explained in Chapter 2, we have done an action study that compares the different cache coherence approaches in CMP. The context aims to describe the need for optimising a scalable coherent cache – regarding area, complexity, energy consumption and performance. Accordingly, we decide to focus on directory protocols for their scalability as concluded in Section 4.5.

The proposed approaches of the case studies have been discussed in Chapter 5 on page 106 and Chapter 6 on page 135. The outcomes of the experimental approaches have been demonstrated on page 113 and 144, which determines the effectiveness of our proposed algorithms. Finally, the contribution and future work of the thesis has been discussed in this chapter.

7.2. Thesis contribution

To summarise the dissertation research contributions, we have:

- Investigated the simulation tools that focus on cache evaluation in chip multiprocessors;
- Modelled and evaluated the hardware cache coherence mechanisms and described the cache scalability issue;
- Proposed a novel design of cache that delivers high associativity to lessen conflict misses;

- Implemented an adaptive replacement policy to work in conjunction with the proposed associativity approach; and
- Proposed an innovative coherence directory protocol to provide the delusion of one level of shared memory, without the need of plenty space, too much bandwidth, or increasing the overall complexity.

7.3. Discussion

The work presented in this thesis made a significant contribution to cache scalability in CMPs by proposing two novel approaches.

The thesis first explained the research methodology and simulation environment. In addition, it compared between the available simulators by highlighting their features and weaknesses. Thus, the most suitable choice for large-scale CMP simulations within this thesis was the Sniper simulator, as it can stabilise the trade-off that happens between accuracy and performance. However, for a comprehensive and more accurate simulation, Simics simulator was used (See Chapter 3 and (Al-Manasia & Chaczko 2015b)).

Following the research methodology approach, the thesis studied the effective protocols and methods that are employed to attain cache coherent in many-cores architectures. Token based, Snooping-based, and Directory-based protocols are the protocols that are modelled and analysed in this study. It further illustrates the strengths and weaknesses of each protocol, as well as, discusses how to carry out their improvement. TOKENB is faster compared to DIRECTOTY and better than SNOOPING mechanism with ample bandwidth, as it avoids putting latency in directory lookup and using a third interconnect traversal on cache-to-cache misses' critical path. On the other hand, TOKENB uses a reasonable amount of extra interconnect traffic and significantly more bandwidth of an endpoint message than DIRECTOTY. Therefore, the performance of DIRECTOTY is better compared to TOKENB in an environment with limited bandwidth and many-cores. The choice of coherence protocol is as complicated and subtle today as it has ever been. For this study, we focused only on Directory protocols for their high scalability in comparison to other protocols (See Chapter 4 and (Al-Manasia & Chaczko 2015a)).

Following the cache coherence action study, the thesis presented the Adaptive Hashing and Replacement Cache (AHRC), which is a new design for caches that use both multi-level replacements and hashed indices to achieve greater performance and associativity. An evaluation of using AHRC as the last level cache in a CMP proves that they give high connectivity with low overheads regarding hit energy, hit time, and area. In both energy efficiency and performance, the AHRC approach outperforms traditional associative caches. A 4-way AHRC achieves higher performance than a 4-way set of associative counterparts over a set of LLC miss-intensive workloads. Additionally, 4-way AHRC achieves better energy efficiency and higher performance than a 16-way set associative cache. AHRC does not emphasise the increase or reduction of associativity with the same hardware design. However, for workloads that are insensitive to replacement policy, AHRC with higher-level replacement results in a loss of energy (See Chapter 5 and (Al-Manasia, Chaczko & Ounzar 2016)).

Taking further enhancement in cache scalability, the thesis proposed another approach for optimising cache coherency in CMP. Cache coherence protocols in the directory are the common approaches for controlling the coherency in most core systems, due to their area and power consumption scalability in comparison to conventional broadcast-based protocols. All cached architectures have writable shared information, which causes the issue of Cache Coherence. The problem with classification on the page level is that there is misclassifying of cache blocks that are private as shared within shared pages.

This study reveals that it would be best to use a data block classification method that functions at the cache block level and assists in the detection of more private data blocks. PROI focuses on a simple approach that does not need complex modifications to the hardware or OS. Studies show that benchmarks are less sensitive to data classification of fine-grain approach in regards to the miss rate. Thus, instead of focusing only on cache misses of blocks with shared status, PROI has more focus on both portions of private and shared blocks. The proposal successfully shows the reduction of the amount of blocks that the directory cache will track. PROI causes energy savings, which is a huge objective, and there are fewer cache misses by the CMP that leads to performance improvements. Furthermore, it is possible to considerably reduce application runtime in case of a decrease in the latency and amount of cache misses (See Chapter 6).

7.4. Future work

The future work approaches in mitigating the cache scalability challenges based on this study have been listed below:

1. Utilise AHRCs to design TLBs for multithreaded cores;
2. Build highly associative low-level caches;
3. Design policies of cache replacement that are specifically suited to AHRC as ARIP still have space for more development;

4. Improving AHRC with a higher-level replacement and avoid the loss of energy for workloads that are not sensitive to the policy governing the replacement process; and
5. Implementing AHRC, ARIP, and PROI in a thousand-core CMP to examine their efficiency when they are combined within one CMP.

7.5. Final Remarks

In consideration of cache scalability approaches in large CMPs, the optimised AHRC cache associativity, ARIP replacement policy, and PROI coherence protocol show a significant scalability improvements in terms of area, latency, and energy. This section of the thesis highlights the following outcomes as the validation results of the illustrated research questions.

By applying AHRC and ARIP as a novel cache associativity mechanism:

- The difference between AHRC and the fully Associative technique is negligible. However, AHRC's concept is simplified and it involves less hardware overhead;
- AHRC design has the ability to maintain high associativity with an advanced method of replacement policy;
- AHRC can improve associativity and maintain the number of possible locations, where each block is kept, as small as is possible;
- ARIP scheme is superior to the NRU scheme and works efficiently with AHRC;
- Results demonstrate that AHRC has better energy efficiency and higher performance when compared to conventional caches; and

- Large caches that utilise AHRC are the most suitable in many-core CMPs to provide more significant improvement and scalability than smaller caches.

By applying PROI protocol and data classification mechanism as a novel cache coherence approach:

- PROI can scale with the raised number of cores and reduce area, energy, and performance associated costs with the coherence mechanism;
- PROI results show a significant reduction of almost 17% in miss ratio of private L1 cache, high network traffic reduction, a reduction in application runtime of approximately 6%; and
- PROI has the capability to lessen the consumption of energy by almost 35% on average.

III. Bibliography and publications

Bibliography

Abts, D., Scott, S. & Lilja, D.J. 2003, 'So many states, so little time: Verifying memory coherence in the Cray X1', *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, IEEE, p. 10 pp.

Acacio, M.E., González, J., García, J.M. & Duato, J. 2001, 'A new scalable directory architecture for large-scale multiprocessors', *High-Performance Computer Architecture, 2001. HPCA. The Seventh International Symposium on*, IEEE, pp. 97-106.

Agarwal, A. & Pudar, S.D. 1993, *Column-associative caches: A technique for reducing the miss rate of direct-mapped caches*, vol. 21, ACM.

Agarwal, A., Simoni, R., Hennessy, J. & Horowitz, M. 1988a, 'An evaluation of directory schemes for cache coherence', *Computer Architecture, 1988. Conference Proceedings. 15th Annual International Symposium on*, IEEE, pp. 280-9.

Agarwal, A., Simoni, R., Hennessy, J. & Horowitz, M. 1988b, 'An evaluation of directory schemes for cache coherence', *SIGARCH Comput. Archit. News*, vol. 16, no. 2, pp. 280-98.

Agarwal, N., Krishna, T., Peh, L.-S. & Jha, N.K. 2009, 'GARNET: A detailed on-chip network model inside a full-system simulator', *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, IEEE, pp. 33-42.

Ahmed, A., Conway, P., Hughes, B. & Weber, F. 2002, 'AMD opteron shared memory mp systems', *Proceedings of the 14th HotChips Symposium*.

Ahn, J.H., Li, S., Seongil, O. & Jouppi, N.P., 'McSimA+: A Manycore Simulator with Application-level+ Simulation and Detailed Microarchitecture Modeling', *ISPASS*.

Al-Manasia, M. & Chaczko, Z. 2015a, 'Evaluation of Cache Coherence Mechanisms for Multicore Processors', *Computational Intelligence and Efficiency in Engineering Systems*, Springer, pp. 307-20.

Al-Manasia, M. & Chaczko, Z. 2015b, 'An Overview of Chip Multi-Processors Simulators Technology', *Progress in Systems Engineering*, Springer, pp. 877-84.

Al-Manasia, M., Chaczko, Z. & Ounzar, A. 2016, 'AHRC: An Optimized Cache Associativity', *IEEE 18th International Conference on High Performance Computing and Communications*, IEEE, Sydney, pp. 811-7.

Alameldeen, A.R., Martin, M.M., Mauer, C.J., Moore, K.E., Xu, M., Hill, M.D., Wood, D.A. & Sorin, D.J. 2003, 'Simulating a $2 M Commercial Server on a $2 K PC', *Computer*, vol. 36, no. 2, pp. 50-7.

Alameldeen, A.R. & Wood, D.A. 2003, 'Variability in Architectural Simulations of Multi-Threaded Workloads', paper presented to the *Proceedings of the 9th International Symposium on High-Performance Computer Architecture*.

Alameldeen, A.R. & Wood, D.A. 2006, 'IPC considered harmful for multiprocessor workloads', *IEEE Micro*, vol. 26, no. 4, pp. 8-17.

Alisafaee, M. 2012, 'Spatiotemporal Coherence Tracking', paper presented to the *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, Vancouver, B.C., CANADA.

Andr, L., Barroso, Gharachorloo, K., McNamara, R., Nowatzyk, A., Qadeer, S., Sano, B., Smith, S., Stets, R. & Verghese, B. 2000, 'Piranha: a scalable architecture based on single-chip multiprocessing', *SIGARCH Comput. Archit. News*, vol. 28, no. 2, pp. 282-93.

Austin, T., Larson, E. & Ernst, D. 2002, 'SimpleScalar: An infrastructure for computer system modeling', *Computer*, vol. 35, no. 2, pp. 59-67.

Bardisa, A.R. 2009, 'Efficient and Scalable Cache Coherence for Many-Core Chip Multiprocessors', Universidad de Murcia.

Barroso, L.A., Gharachorloo, K. & Bugnion, E. 1998, 'Memory system characterization of commercial workloads', *ACM SIGARCH Computer Architecture News*, vol. 26, no. 3, pp. 3-14.

Barroso, L.A., Gharachorloo, K., McNamara, R., Nowatzyk, A., Qadeer, S., Sano, B., Smith, S., Stets, R. & Verghese, B. 2000, 'Piranha: a scalable architecture based on single-chip multiprocessing', *ACM SIGARCH Computer Architecture News*, vol. 28, ACM, pp. 282-93.

Basu, A., Kirman, N., Kirman, M., Chaudhuri, M. & Martinez, J. 2007, 'Scavenger: A new last level cache architecture with global block priority', *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, IEEE Computer Society, pp. 421-32.

Batten, C. 2016, *ECE 4750 Computer Architecture*, Data collected by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, C. Batten, Cornell University, School of Electrical and Computer Engineering, viewed 19 Dec 2016, <http://www.csl.cornell.edu/courses/ece4750/handouts/ece4750-overview.pdf>.

Belady, L.A. 1966, 'A study of replacement algorithms for a virtual-storage computer', *IBM Systems journal*, vol. 5, no. 2, pp. 78-101.

Berkowits, S. 2012, 'Pin-a dynamic binary instrumentation tool', *Intel Developer Zone*.

Bhattacharjee, A., Lustig, D. & Martonosi, M. 2011, 'Shared last-level TLBs for chip multiprocessors', paper presented to the *Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture*.

Bhattacharjee, A. & Martonosi, M. 2009, 'Characterizing the TLB Behavior of Emerging Parallel Workloads on Chip Multiprocessors', paper presented to the *Proceedings of the 2009 18th International Conference on Parallel Architectures and Compilation Techniques*.

Bienia, C., Kumar, S., Singh, J.P. & Li, K. 2008a, 'The PARSEC benchmark suite: Characterization and architectural implications', *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, ACM, pp. 72-81.

Bienia, C., Kumar, S., Singh, J.P. & Li, K. 2008b, 'The PARSEC benchmark suite: characterization and architectural implications', paper presented to the *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, Toronto, Ontario, Canada.

Bilir, E.E., Dickson, R.M., Hu, Y., Plakal, M., Sorin, D.J., Hill, M.D. & Wood, D.A. 1999, 'Multicast snooping: a new coherence method using a multicast address network', *Computer Architecture, 1999. Proceedings of the 26th International Symposium on*, IEEE, pp. 294-304.

Binkert, N., Beckmann, B., Black, G., Reinhardt, S.K., Saidi, A., Basu, A., Hestness, J., Hower, D.R., Krishna, T. & Sardashti, S. 2011, 'The gem5 simulator', *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1-7.

Bodin, F. & Seznec, A. 1995, *Skewed associativity enhances performance predictability*, vol. 23, ACM.

Borrego, M., Douglas, E.P. & Amelink, C.T. 2009, 'Quantitative, qualitative, and mixed research methods in engineering education', *Journal of Engineering education*, vol. 98, no. 1, p. 53.

Brehob, M.W. 2003a, 'On the Mathematics of Caching', Michigan State

University.

Brehob, M.W. 2003b, 'On the mathematics of caching', Michigan State University.

Calder, B., Grunwald, D. & Emer, J. 1996, 'Predictive sequential associative cache', *High-Performance Computer Architecture, 1996. Proceedings., Second International Symposium on*, IEEE, pp. 244-53.

Cantin, J.F., Lipasti, M.H. & Smith, J.E. 2005, 'Improving Multiprocessor Performance with Coarse-Grain Coherence Tracking', *SIGARCH Comput. Archit. News*, vol. 33, no. 2, pp. 246-57.

Carlson, T.E. & Heirman, W. 2013, 'The Sniper User Manual'.

Carlson, T.E., Heirman, W. & Eeckhout, L. 2011, 'Sniper: exploring the level of abstraction for scalable and accurate parallel multi-core simulation', *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ACM, p. 52.

Carter, J.L. & Wegman, M.N. 1977, 'Universal classes of hash functions', *Proceedings of the ninth annual ACM symposium on Theory of computing*, ACM, pp. 106-12.

Censier, L.M. & Feautrier, P. 1978, 'A new solution to coherence problems in multicache systems', *Computers, IEEE Transactions on*, vol. 100, no. 12, pp. 1112-8.

Chaiken, D., Kubiatowicz, J. & Agarwal, A. 1991, *LimitLESS directories: A scalable cache coherence scheme*, vol. 26, ACM.

Charlesworth, A. 2002, 'The sun fireplane interconnect', *Micro, IEEE*, vol. 22, no. 1, pp. 36-45.

Chen, G. 1993, 'SLiD—A cost-effective and Scalable Limited-Directory scheme for cache coherence', *PARLE'93 Parallel Architectures and Languages Europe*, Springer, pp. 341-52.

Chen, J., Annavaram, M. & Dubois, M. 2009, 'SlackSim: a platform for parallel simulations of CMPs on CMPs', *ACM SIGARCH Computer Architecture News*, vol. 37, no. 2, pp. 20-9.

Choi, B., Komuravelli, R., Sung, H., Smolinski, R., Honarmand, N., Adve, S.V., Adve, V.S., Carter, N.P. & Chou, C.-T. 2011, 'DeNovo: Rethinking the Memory Hierarchy for Disciplined Parallelism', paper presented to the *Proceedings of the*

2011 International Conference on Parallel Architectures and Compilation Techniques.

Choi, J.H. & Park, K.H. 1999, 'Segment directory enhancing the limited directory cache coherence schemes', *Parallel Processing, 1999. 13th International and 10th Symposium on Parallel and Distributed Processing, 1999. 1999 IPPS/SPDP. Proceedings*, IEEE, pp. 258-67.

Conway, P., Kalyanasundharam, N., Donley, G., Lepak, K. & Hughes, B. 2010, 'Cache Hierarchy and Memory Subsystem of the AMD Opteron Processor', *IEEE Micro*, vol. 30, no. 2, pp. 16-29.

Creswell, J.W. 2013, *Research design: Qualitative, quantitative, and mixed methods approaches*, Sage publications.

Cuesta, B.A., Ros, A., Mar, #237, G, a.E., #243, mez, Robles, A., Jos, #233 & Duato, F. 2011, 'Increasing the effectiveness of directory caches by deactivating coherence for private memory blocks', *SIGARCH Comput. Archit. News*, vol. 39, no. 3, pp. 93-104.

Davanam, N. & Lee, B.K. 2010, 'Towards Smaller-sized Cache for Mobile Processors using Shared Set-associativity', *Information Technology: New Generations (ITNG), 2010 Seventh International Conference on*, IEEE, pp. 1-6.

Davari, M., Ros, A., Hagersten, E. & Kaxiras, S. 2015, 'The Effects of Granularity and Adaptivity on Private/Shared Classification for Coherence', *ACM Trans. Archit. Code Optim.*, vol. 12, no. 3, pp. 1-21.

Deng, Q., Meisner, D., Ramos, L., Wenisch, T.F. & Bianchini, R. 2011, 'Memscale: active low-power modes for main memory', *ACM SIGPLAN Notices*, vol. 46, no. 3, pp. 225-38.

Dikaiakos, M.D., Rogers, A. & Steiglitz, K. 1994, 'Fast: A functional algorithm simulation testbed', *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 1994., MASCOTS'94., Proceedings of the Second International Workshop on*, IEEE, pp. 142-6.

Ekman, M., Stenstr, P., #246 & Dahlgren, F. 2002, 'TLB and snoop energy-reduction using virtual caches in low-power chip-multiprocessors', paper presented to the *Proceedings of the 2002 international symposium on Low power electronics and design*, Monterey, California, USA.

Fang, L., Liu, P., Hu, Q., Huang, M.C. & Jiang, G. 2013, 'Building expressive, area-efficient coherence directories', paper presented to the *Proceedings of the 22nd international conference on Parallel architectures and compilation techniques*, Edinburgh, Scotland, UK.

Fensch, C. & Cintra, M. 2008, 'An OS-based alternative to full hardware coherence on tiled CMPs', *2008 IEEE 14th International Symposium on High Performance Computer Architecture*, IEEE, pp. 355-66.

Ferdman, M., Lotfi-Kamran, P., Balet, K. & Falsafi, B. 2011a, 'Cuckoo directory: A scalable directory for many-core systems', *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*, IEEE, pp. 169-80.

Ferdman, M., Lotfi-Kamran, P., Balet, K. & Falsafi, B. 2011b, 'Cuckoo directory: A scalable directory for many-core systems', paper presented to the *Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture*.

Frank, S.J. 1984, 'Tightly coupled multiprocessor system speeds memory-access times', *Electronics;(United States)*, vol. 1.

Galles, M. & Williams, E. 1994, 'Performance optimizations, implementation, and verification of the SGI Challenge multiprocessor', *System Sciences, 1994. Proceedings of the Twenty-Seventh Hawaii International Conference on*, vol. 1, IEEE, pp. 134-43.

Geer, D. 2005, 'Chip makers turn to multicore processors', *Computer*, vol. 38, no. 5, pp. 11-3.

Geer, D. 2007, 'For programmers, multicore chips mean multiple challenges', *Computer*, vol. 40, no. 9, pp. 17-9.

Gepner, P. & Kowalik, M.F. 2006, 'Multi-core processors: New way to achieve high system performance', *Parallel Computing in Electrical Engineering, 2006. PAR ELEC 2006. International Symposium on*, IEEE, pp. 9-13.

Gharachorloo, K., Barroso, L.A. & Nowatzyk, A. 2000, 'Efficient ECC-Based Directory Implementations for Scalable Multiprocessors', *Proceedings of the 12th Symposium on Computer Architecture and High-Performance Computing (SBAC-PAD 2000)*.

Goldschmidt, S.R. & Hennessy, J.L. 1993, *The accuracy of trace-driven simulations of multiprocessors*, vol. 21, ACM.

Golla, R. 2006, 'Niagara2: A highly threaded server-on-a-chip', *Fall Microprocessor Forum*.

Goodman, J.R. 1998, 'Using cache memory to reduce processor-memory traffic', *25 Years of the International Symposia on Computer Architecture (selected papers)*, ACM, pp. 255-62.

Guo, S.-L., Wang, H.-X., Xue, Y.-B., Li, C.-M. & Wang, D.-S. 2010, 'Hierarchical Cache Directory for CMP', *Journal of Computer Science and Technology*, vol. 25, no. 2, pp. 246-56.

Gupta, A., Weber, W.-D. & Mowry, T. 1990a, 'Reducing memory and traffic requirements for scalable directory-based cache coherence schemes'.

Gupta, A., Weber, W.-D. & Mowry, T. 1990b, *Reducing memory and traffic requirements for scalable directory-based cache coherence schemes*, Computer Systems Laboratory, Stanford University.

Hallnor, E.G. & Reinhardt, S.K. 2000, *A fully associative software-managed cache design*, vol. 28, ACM.

Hardavellas, N., Ferdman, M., Falsafi, B. & Ailamaki, A. 2009, 'Reactive NUCA: near-optimal block placement and replication in distributed caches', *SIGARCH Comput. Archit. News*, vol. 37, no. 3, pp. 184-95.

Hill, M.D. & Smith, A.J. 1989, 'Evaluating associativity in CPU caches', *Computers, IEEE Transactions on*, vol. 38, no. 12, pp. 1612-30.

Horel, T. & Lauterbach, G. 1999, 'UltraSPARC-III: Designing third-generation 64-bit performance', *Micro, IEEE*, vol. 19, no. 3, pp. 73-85.

Hossain, H., Dwarkadas, S. & Huang, M.C. 2011, 'POPS: Coherence Protocol Optimization for Both Private and Shared Data', paper presented to the *Proceedings of the 2011 International Conference on Parallel Architectures and Compilation Techniques*.

Howard, J., Dighe, S., Hoskote, Y., Vangal, S., Finan, D., Ruhl, G., Jenkins, D., Wilson, H., Borkar, N. & Schrom, G. 2010, 'A 48-core IA-32 message-passing processor with DVFS in 45nm CMOS', *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, IEEE, pp. 108-9.

Hubbard, T., Lencevicius, R., Metz, E. & Raghavan, G. 2008, 'Performance Validation on Multicore Mobile Devices', *Verified Software: Theories, Tools, Experiments*, Springer, pp. 413-21.

Hummel, M.D., Keller, J.B., Meyer, D.R. & Owen, J.M. 2006, 'System and method of maintaining coherency in a distributed communication system', Google Patents.

Jaleel, A., Borch, E., Bhandaru, M., Steely Jr, S.C. & Emer, J. 2010, 'Achieving non-inclusive cache performance with inclusive caches: Temporal locality aware (tla) cache management policies', *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, IEEE Computer Society, pp. 151-62.

Jaleel, A., Cohn, R.S., Luk, C.-K. & Jacob, B. 2008, 'CMP$im: A Pin-based on-the-fly multi-core cache simulator', *Proceedings of the Fourth Annual Workshop on Modeling, Benchmarking and Simulation (MoBS), co-located with ISCA*, pp. 28-36.

Jaleel, A., Hasenplaugh, W., Qureshi, M., Sebot, J., Steely Jr, S. & Emer, J. 2008, 'Adaptive insertion policies for managing shared caches', *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, ACM, pp. 208-19.

Jaleel, A., Theobald, K.B., Steely Jr, S.C. & Emer, J. 2010, 'High performance cache replacement using re-reference interval prediction (RRIP)', *ACM SIGARCH Computer Architecture News*, vol. 38, ACM, pp. 60-71.

Jerger, N.D.E., Peh, L.-S. & Lipasti, M.H. 2008a, 'Virtual tree coherence: Leveraging regions and in-network multicast trees for scalable cache coherence', paper presented to the *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*.

Jerger, N.E., Peh, L.-S. & Lipasti, M. 2008b, 'Virtual Circuit Tree Multicasting: A Case for On-Chip Hardware Multicast Support', *SIGARCH Comput. Archit. News*, vol. 36, no. 3, pp. 229-40.

Johnson, T. & Shasha, D. 1994, 'X3: A Low Overhead High Performance Buffer Management Replacement Algorithm'.

Jouppi, N.P. 1990a, 'Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers', *Computer Architecture, 1990. Proceedings., 17th Annual International Symposium on*, IEEE, pp. 364-73.

Jouppi, N.P. 1990b, 'Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers', *ACM SIGARCH Computer Architecture News*, vol. 18, ACM, pp. 364-73.

Katz, R.H., Eggers, S.J., Wood, D.A., Perkins, C. & Sheldon, R.G. 1985, *Implementing a cache consistency protocol*, vol. 13, IEEE Computer Society Press.

Kaxiras, S., Hu, Z. & Martonosi, M. 2001, 'Cache decay: exploiting generational behavior to reduce cache leakage power', *ACM SIGARCH Computer Architecture News*, vol. 29, no. 2, pp. 240-51.

Kaxiras, S. & Ros, A. 2012, 'Efficient, snoopless, system-on-chip coherence', *SOC Conference (SOCC), 2012 IEEE International*, IEEE, pp. 230-5.

Kelm, J.H., Johnson, D.R., Johnson, M.R., Crago, N.C., Tuohy, W., Mahesri, A., Lumetta, S.S., Frank, M.I. & Patel, S.J. 2009, 'Rigel: an architecture and scalable programming interface for a 1000-core accelerator', *ACM SIGARCH Computer Architecture News*, vol. 37, ACM, pp. 140-51.

Kelm, J.H., Johnson, M.R., Lumettta, S.S. & Patel, S.J. 2010, 'WAYPOINT: scaling coherence to thousand-core architectures', *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, ACM, pp. 99-110.

Khailany, B., Dally, W.J., Kapasi, U.J., Mattson, P., Namkoong, J., Owens, J.D., Towles, B., Chang, A. & Rixner, S. 2001, 'Imagine: Media processing with streams', *IEEE micro*, vol. 21, no. 2, pp. 35-46.

Kharbutli, M., Irwin, K., Solihin, Y. & Lee, J. 2004, 'Using prime numbers for cache indexing to eliminate conflict misses', *Software, IEE Proceedings-*, IEEE, pp. 288-99.

Kim, D., Ahn, J., Kim, J. & Huh, J. 2010, 'Subspace snooping: filtering snoops with operating system support', paper presented to the *Proceedings of the 19th* *international conference on Parallel architectures and compilation techniques*, Vienna, Austria.

Kron, J.D., Prumo, B. & Loh, G.H. 2008, 'Double-dip: Augmenting dip with adaptive promotion policies to manage shared l2 caches', *Proc. of the Workshop on Chip Multiprocessor Memory Systems and Interconnects, Beijing, China*.

Ku, W.-C., Chou, S.-H., Chu, J.-C., Liu, C.-L., Chen, T.-F., Guo, J.-I. & Wang, J.-S. 2009, 'VisoMT: a collaborative multithreading multicore processor for multimedia applications with a fast data switching mechanism', *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 19, no. 11, pp. 1633-45.

Kurd, N.A., Bhamidipati, S., Mozak, C., Miller, J.L., Wilson, T.M., Nemani, M. & Chowdhury, M. 2010, 'Westmere: A family of 32nm IA processors', *2010 IEEE International Solid-State Circuits Conference-(ISSCC)*.

Kurian, G., Khan, O. & Devadas, S. 2012, 'A Case for Fine-Grain Adaptive Cache Coherence'.

Kurian, G., Miller, J.E., Psota, J., Eastep, J., Liu, J., Michel, J., Kimerling, L.C. & Agarwal, A. 2010, '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, pp. 477-88.

Lai, A.-C., Fide, C. & Falsafi, B. 2001, 'Dead-block prediction & dead-block correlating prefetchers', *Computer Architecture, 2001. Proceedings. 28th Annual International Symposium on*, IEEE, pp. 144-54.

Le, H.Q., Starke, W.J., Fields, J.S., O'Connell, F.P., Nguyen, D.Q., Ronchetti, B.J., Sauer, W.M., Schwarz, E.M. & Vaden, M.T. 2007, 'IBM POWER6 microarchitecture', *IBM J. Res. Dev.*, vol. 51, no. 6, pp. 639-62.

Leedy, P.D. & Ormrod, J.E. 2005, 'Practical research', *Planning and design*, vol. 8.

Leverich, J., Arakida, H., Solomatnikov, A., Firoozshahian, A., Horowitz, M. & Kozyrakis, C. 2007, 'Comparing memory systems for chip multiprocessors', *ACM SIGARCH Computer Architecture News*, vol. 35, ACM, pp. 358-68.

Li, S., Ahn, J.H., Strong, R.D., Brockman, J.B., Tullsen, D.M. & Jouppi, N.P. 2009, 'McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures', *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, IEEE, pp. 469-80.

Li, Y., Abousamra, A., Melhem, R. & Jones, A.K. 2010, 'Compiler-assisted data distribution for chip multiprocessors', paper presented to the *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, Vienna, Austria.

Li, Y., Melhem, R. & Jones, A.K. 2012, 'Practically private: enabling high performance CMPs through compiler-assisted data classification', paper presented to the *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*, Minneapolis, Minnesota, USA.

Loh, G.H. 2009, 'Extending the effectiveness of 3D-stacked DRAM caches with an adaptive multi-queue policy', *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, IEEE, pp. 201-12.

Loudon, J. & Lenoski, D. 1997, 'The SGI Origin: A ccNUMA Highly Scalable Server, Silcon Graphics, Inc., presented at the Proc. Of the 24th Int'l Symp', *Computer Architecture, Jun*.

Magnusson, P.S., Christensson, M., Eskilson, J., Forsgren, D., H, G., #229, llberg, H, J., #246, gberg, Larsson, F., Moestedt, A. & Werner, B. 2002, 'Simics: A Full System Simulation Platform', *Computer*, vol. 35, no. 2, pp. 50-8.

Magnusson, P.S., Christensson, M., Eskilson, J., Forsgren, D., Hallberg, G., Hogberg, J., Larsson, F., Moestedt, A. & Werner, B. 2002, 'Simics: A full system simulation platform', *Computer*, vol. 35, no. 2, pp. 50-8.

Magnusson, P.S., Dahlgren, F., Grahn, H., Karlsson, M., Larsson, F., Lundholm, F., Moestedt, A., Nilsson, J., Stenström, P. & Werner, B. 1998, 'SimICS/sun4m: A virtual workstation', *Proceedings of Usenix Annual Technical Conference*, pp. 119-30.

Marcu, M., Tudor, D., Fuicu, S., Copil-Crisan, S., Maticu, F. & Micea, M. 2008, 'Power efficiency study of multi-threading applications for multi-core mobile systems', *WSEAS Transactions on Computers*, vol. 7, no. 12, pp. 1875-85.

Martin, M.M. 2003, 'Token coherence', UNIVERSITY OF WISCONSIN.

Martin, M.M., Harper, P.J., Sorin, D.J., Hill, M.D. & Wood, D.A. 2003, 'Using destination-set prediction to improve the latency/bandwidth tradeoff in shared-memory multiprocessors', *Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on*, IEEE, pp. 206-17.

Martin, M.M., Hill, M.D. & Sorin, D.J. 2012, 'Why on-chip cache coherence is here to stay', *Communications of the ACM*, vol. 55, no. 7, pp. 78-89.

Martin, M.M., Hill, M.D. & Wood, D.A. 2003a, 'Token coherence: A new framework for shared-memory multiprocessors', *Micro, IEEE*, vol. 23, no. 6, pp. 108-16.

Martin, M.M., Hill, M.D. & Wood, D.A. 2003b, 'Token coherence: decoupling performance and correctness', *Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on*, IEEE, pp. 182-93.

Martin, M.M., Sorin, D.J., Beckmann, B.M., Marty, M.R., Xu, M., Alameldeen, A.R., Moore, K.E., Hill, M.D. & Wood, D.A. 2005a, 'Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset', *ACM SIGARCH Computer Architecture News*, vol. 33, no. 4, pp. 92-9.

Martin, M.M.K., Sorin, D.J., Beckmann, B.M., Marty, M.R., Xu, M., Alameldeen, A.R., Moore, K.E., Hill, M.D. & Wood, D.A. 2005b, 'Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset', *SIGARCH Comput. Archit. News*, vol. 33, no. 4, pp. 92-9.

Marty, M.R. 2008, *Cache coherence techniques for multicore processors*, ProQuest.

Marty, M.R. & Hill, M.D. 2007a, 'Virtual hierarchies to support server consolidation', *ACM SIGARCH Computer Architecture News*, vol. 35, ACM, pp. 46-56.

Marty, M.R. & Hill, M.D. 2007b, 'Virtual hierarchies to support server consolidation', *SIGARCH Comput. Archit. News*, vol. 35, no. 2, pp. 46-56.

McCreight, E.M. 1985, 'The dragon computer system', *Microarchitecture of VLSI Computers*, Springer, pp. 83-101.

Megiddo, N. & Modha, D.S. 2004, 'Outperforming LRU with an adaptive replacement cache algorithm', *Computer*, vol. 37, no. 4, pp. 58-65.

Miller, J.E., Kasture, H., Kurian, G., Gruenwald, C., Beckmann, N., Celio, C., Eastep, J. & Agarwal, A. 2010, 'Graphite: A distributed parallel simulator for multicores', *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, IEEE, pp. 1-12.

Moinuddin, K., Jaleel, A., Patt, Y.N., Steely Jr, S.C. & Emer, J. 2008, 'Set-dueling-controlled adaptive insertion for high-performance caching', *Micro IEEE*, vol. 28, no. 1, pp. 91-8.

Moore, G.E. 2006, 'Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp. 114 ff', *Solid-State Circuits Society Newsletter, IEEE*, vol. 11, no. 5, pp. 33-5.

Moshovos, A. 2005, 'RegionScout: Exploiting Coarse Grain Sharing in Snoop-Based Coherence', *SIGARCH Comput. Archit. News*, vol. 33, no. 2, pp. 234-45.

Mukherjee, S.S., Bannon, P., Lang, S., Spink, A. & Webb, D. 2001, 'The Alpha 21364 network architecture', *Hot Interconnects 9, 2001.*, IEEE, pp. 113-7.

Mukherjee, S.S., Reinhardt, S.K., Falsafi, B., Litzkow, M., Hill, M.D., Wood, D.A., Huss-Lederman, S. & Larus, J.R. 2000, 'Wisconsin Wind Tunnel II: a fast, portable parallel architecture simulator', *Concurrency, IEEE*, vol. 8, no. 4, pp. 12-20.

Muralimanohar, N., Balasubramonian, R. & Jouppi, N. 2007, 'Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0', *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, IEEE Computer Society, pp. 3-14.

Muralimanohar, N., Balasubramonian, R. & Jouppi, N.P. 2009, 'CACTI 6.0: A tool to model large caches', *HP Laboratories*, pp. 22-31.

Nanda, A.K., Nguyen, A.-T., Michael, M.M. & Joseph, D.J. 2001, 'High-throughput coherence control and hardware messaging in everest', *IBM J. Res. Dev.*, vol. 45, no. 2, pp. 229-43.

Nawathe, U.G., Hassan, M., Yen, K.C., Kumar, A., Ramachandran, A. & Greenhill, D. 2008, 'Implementation of an 8-core, 64-thread, power-efficient sparc server on a chip', *Solid-State Circuits, IEEE Journal of*, vol. 43, no. 1, pp. 6-20.

Nowatzyk, A., Aybay, G., Browne, M., Kelly, E., Lee, D. & Parkin, M. 1994, 'The S3. mp scalable shared memory multiprocessor', *System Sciences, 1994. Proceedings of the Twenty-Seventh Hawaii International Conference on*, vol. 1, IEEE, pp. 144-53.

NVIDIA 2010, 'The Benefits of Multiple CPU Cores in Mobile Devices'.

O'Krafka, B.W. & Newton, A.R. 1990, 'An empirical evaluation of two memory-efficient directory methods', *ACM SIGARCH Computer Architecture News*, vol. 18, ACM, pp. 138-47.

Pagh, R. & Rodler, F.F. 2001, *Cuckoo hashing*, Springer.

Pai, V.S., Ranganathan, P. & Adve, S.V. 1997, 'RSIM: An execution-driven simulator for ILP-based shared-memory multiprocessors and uniprocessors', *Proceedings of the Third Workshop on Computer Architecture Education*, vol. 178.

Patrizio, A. 2013, *In Smartphones and Tablets, Multicore is Not Necessarily the Way to Go*, viewed July, 2014 2014, <http://blog.smartbear.com/mobile/in-smartphones-and-tablets-multicore-is-not-necessarily-the-way-to-go/>.

Perelman, E., Hamerly, G., Van Biesbrouck, M., Sherwood, T. & Calder, B. 2003, 'Using SimPoint for accurate and efficient simulation', *ACM SIGMETRICS Performance Evaluation Review*, vol. 31, ACM, pp. 318-9.

Processor, I.X. 2010, '5600 Series, Datasheet, Volume 1'.

Pugsley, S.H., Spjut, J.B., Nellans, D.W. & Balasubramonian, R. 2010, 'SWEL: hardware cache coherence protocols to map shared data onto shared caches', paper presented to the *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, Vienna, Austria.

Qureshi, M.K., Jaleel, A., Patt, Y.N., Steely, S.C. & Emer, J. 2007, 'Adaptive insertion policies for high performance caching', *ACM SIGARCH Computer Architecture News*, vol. 35, ACM, pp. 381-91.

Qureshi, M.K., Lynch, D.N., Mutlu, O. & Patt, Y.N. 2006, 'A case for MLP-aware cache replacement', *ACM SIGARCH Computer Architecture News*, vol. 34, no. 2, pp. 167-78.

Qureshi, M.K., Thompson, D. & Patt, Y.N. 2005, 'The V-Way cache: demand-based associativity via global replacement', *Computer Architecture, 2005. ISCA'05. Proceedings. 32nd International Symposium on*, IEEE, pp. 544-55.

Rajan, K. & Govindarajan, R. 2007, 'Emulating optimal replacement with a shepherd cache', *Microarchitecture, 2007. MICRO 2007. 40th Annual IEEE/ACM International Symposium on*, IEEE, pp. 445-54.

Rajan, K. & Ramaswamy, G. 2007, 'Emulating optimal replacement with a shepherd cache', *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*, IEEE, pp. 445-54.

Ramakrishna, M., Fu, E. & Bahcekapili, E. 1997, 'Efficient hardware hashing functions for high performance computers', *IEEE Transactions on Computers*, vol. 46, no. 12, pp. 1378-81.

Rao, W. 2009, 'Multi Processors, their Memory organizations and Implementations by Intel and AMD', <http://ece.uic.edu/wenjing/courses/fa08ECE569/ECE569/w21.pdf>.

Rolán, D., Fraguela, B.B. & Doallo, R. 2009, 'Adaptive line placement with the set balancing cache', *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, ACM, pp. 529-40.

Ros, A. & Kaxiras, S. 2012, 'Complexity-effective multicore coherence', paper presented to the *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*, Minneapolis, Minnesota, USA.

Rosenblum, M., Herrod, S.A., Witchel, E. & Gupta, A. 1995, 'Complete computer system simulation: The SimOS approach', *Parallel & Distributed Technology: Systems & Applications, IEEE*, vol. 3, no. 4, pp. 34-43.

Roser, M. 2016, *Technological Progress*, Published online at OurWorldInData.org2016, <https://ourworldindata.org/technological-progress/>.

Sale, J.E., Lohfeld, L.H. & Brazil, K. 2002, 'Revisiting the quantitative-qualitative debate: Implications for mixed-methods research', *Quality and quantity*, vol. 36, no. 1, pp. 43-53.

Sanchez, D. & Kozyrakis, C. 2010, 'The ZCache: Decoupling ways and associativity', *Microarchitecture (MICRO), 2010 43rd Annual IEEE/ACM International Symposium on*, IEEE, pp. 187-98.

Sanchez, D. & Kozyrakis, C. 2012, 'SCD: A scalable coherence directory with flexible sharer set encoding', *High Performance Computer Architecture (HPCA), 2012 IEEE 18th International Symposium on*, IEEE, pp. 1-12.

Sanchez, D. & Kozyrakis, C. 2013, 'ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-Core Systems'.

Schauer, B. 2008, 'Multicore processors–a necessity', *ProQuest discovery guides*, pp. 1-14.

Seznec, A. 1993, 'A case for two-way skewed-associative caches', *ACM SIGARCH Computer Architecture News*, vol. 21, ACM, pp. 169-78.

Shah, M., Barreh, J., Brooks, J., Golla, R., Grohoski, G., Gura, N., Hetherington, R., Jordan, P., Luttrell, M. & Olson, C. 2007, 'UltraSPARC T2: A highly-treaded, power-efficient, SPARC SOC', *Solid-State Circuits Conference, 2007. ASSCC'07. IEEE Asian*, IEEE, pp. 22-5.

Shah, M., Barren, J., Brooks, J., Golla, R., Grohoski, G., Gura, N., Hetherington, R., Jordan, P., Luttrell, M. & Olson, C. 2007, 'UltraSPARC T2: A highly-treaded, power-efficient, SPARC SOC', *Solid-State Circuits Conference, 2007. ASSCC'07. IEEE Asian*, IEEE, pp. 22-5.

Shin, J.L., Tam, K., Huang, D., Petrick, B., Pham, H., Hwang, C., Li, H., Smith, A., Johnson, T. & Schumacher, F. 2010, 'A 40nm 16-core 128-thread CMT SPARC SoC processor', *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, IEEE, pp. 98-9.

Simoni, R.T. 1992, 'Cache coherence directories for scalable multiprocessors', to the Department of Electrical Engineering.Stanford University.

Singhal, R. 2008, 'Inside Intel next generation Nehalem microarchitecture', *Hot Chips*, vol. 20.

Sorin, D.J., Hill, M.D. & Wood, D.A. 2011, *A Primer on Memory Consistency and Cache Coherence*, Morgan \& Claypool Publishers.

Sorin, D.J., Plakal, M., Condon, A.E., Hill, M.D., Martin, M.M.K. & Wood, D.A. 2002, 'Specifying and verifying a broadcast and a multicast snooping cache coherence protocol', *Parallel and Distributed Systems, IEEE Transactions on*, vol. 13, no. 6, pp. 556-78.

Spjut, J., Kensler, A., Kopta, D. & Brunvand, E. 2009, 'TRaX: a multicore hardware architecture for real-time ray tracing', *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 28, no. 12, pp. 1802-15.

Srikantaiah, S. & Kandemir, M. 2010, 'Synergistic TLBs for High Performance Address Translation in Chip Multiprocessors', paper presented to the *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*.

Stenstrom, P. 1990, 'A survey of cache coherence schemes for multiprocessors', *Computer*, vol. 23, no. 6, pp. 12-24.

Subramanian, R., Smaragdakis, Y. & Loh, G.H. 2006, 'Adaptive caches: Effective shaping of cache behavior to workloads', *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, IEEE Computer Society, pp. 385-96.

Sun Microsystems, I. 2007, 'UltraSPARC T2 supplement to the UltraSPARC architecture 2007'.

Tang, C. 1976, 'Cache system design in the tightly coupled multiprocessor system', *Proceedings of the June 7-10, 1976, national computer conference and exposition*, ACM, pp. 749-53.

Thacker, C.P., Stewart, L.C. & Satterthwaite Jr, E.H. 1988, 'Firefly: A multiprocessor workstation', *Computers, IEEE Transactions on*, vol. 37, no. 8, pp. 909-20.

Wallach, D.A. 1992, 'PHD: A Hierarchical Cache Coherent Protocol', Citeseer.

Wendel, D., Kalla, R., Cargoni, R., Clables, J., Friedrich, J., Frech, R., Kahle, J., Sinharoy, B., Starke, W. & Taylor, S. 2010, 'The implementation of POWER7 TM: A highly parallel and scalable multi-core high-end server processor', *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, IEEE, pp. 102-3.

Wenisch, T.F., Wunderlich, R.E., Falsafi, B. & Hoe, J.C. 2005, 'TurboSMARTS: Accurate microarchitecture simulation sampling in minutes', *ACM SIGMETRICS Performance Evaluation Review*, vol. 33, ACM, pp. 408-9.

Woo, S.C., Ohara, M., Torrie, E., Singh, J.P. & Gupta, A. 1995, 'The SPLASH-2 programs: characterization and methodological considerations', *SIGARCH Comput. Archit. News*, vol. 23, no. 2, pp. 24-36.

Wunderlich, R.E., Wenisch, T.F., Falsafi, B. & Hoe, J.C. 2003, 'SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling', *Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on*, IEEE, pp. 84-95.

Xie, Y. & Loh, G.H. 2009, 'PIPP: promotion/insertion pseudo-partitioning of multi-core shared caches', *ACM SIGARCH Computer Architecture News*, vol. 37, ACM, pp. 174-83.

Yang, Q., Thangadurai, G. & Bhuyan, L.N. 1992, 'Design of an adaptive cache coherence protocol for large scale multiprocessors', *Parallel and Distributed Systems, IEEE Transactions on*, vol. 3, no. 3, pp. 281-93.

Yourst, M.T. 2007, 'PTLsim: A cycle accurate full system x86-64 microarchitectural simulator', *Performance Analysis of Systems & Software, 2007. ISPASS 2007. IEEE International Symposium on*, IEEE, pp. 23-34.

Zebchuk, J., Falsafi, B. & Moshovos, A. 2013, 'Multi-grain coherence directories', paper presented to the *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, Davis, California.

Zebchuk, J., Makineni, S. & Newell, D. 2008, 'Re-examining cache replacement policies', *Computer Design, 2008. ICCD 2008. IEEE International Conference on*, IEEE, pp. 671-8.

Zebchuk, J., Safi, E. & Moshovos, A. 2007, 'A Framework for Coarse-Grain Optimizations in the On-Chip Memory Hierarchy', paper presented to the *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*.

Zebchuk, J., Srinivasan, V., Qureshi, M.K. & Moshovos, A. 2009, 'A tagless coherence directory', *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, IEEE, pp. 423-34.

Zeffer, k. & Hagersten, E. 2007, 'A case for low-complexity MP architectures', paper presented to the *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, Reno, Nevada.

Zeffer, k., Radovi, Z., Karlsson, M. & Hagersten, E. 2006, 'TMA: a trap-based memory architecture', paper presented to the *Proceedings of the 20th annual international conference on Supercomputing*, Cairns, Queensland, Australia.

Zhang, C., Zhang, X. & Yan, Y. 1997, 'Two fast and high-associativity cache schemes', *Micro, IEEE*, vol. 17, no. 5, pp. 40-9.

Zhao, H., Shriraman, A. & Dwarkadas, S. 2010, 'SPACE: Sharing pattern-based directory coherence for multicore scalability', *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*.

Zhao, H., Shriraman, A., Kumar, S. & Dwarkadas, S. 2013, 'Protozoa: adaptive granularity cache coherence', *SIGARCH Comput. Archit. News*, vol. 41, no. 3, pp. 547-58.

Zhou, Y., Philbin, J. & Li, K. 2001, 'The Multi-Queue Replacement Algorithm for Second Level Buffer Caches', *USENIX Annual Technical Conference, General Track*, pp. 91-104.