Universitat Politècnica de València

Departamento de Informática
de Sistemas y Computadores

Improving Energy and Area Scalability of the Cache Hierarchy in CMPs

A thesis submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy
(Computer Engineering)

Author

Joan Josep Valls Mompó

Advisors

Prof. Julio Sahuquillo Borrás
Prof. Maria Engracia Gómez Requena

February 2017

Agradecimientos

Esta tesis es el resultado de mucho esfuerzo y dedicación a lo largo de estos años. Quiero dedicar este espacio para agradecer a todas aquellas personas que han estado a mi lado todo este tiempo por todo su apoyo y su comprensión.

En primer lugar quiero agradecerles a mis padres y a mi hermano por haber estado ahí siempre, tanto en los buenos como en los malos momentos. Gracias por haber estado siempre ahí para escucharme, por saber que puedo contar con vosotros para todo. Por vuestros ánimos, por soportarme día a día, por todo eso y mucho más, gracias.

Quiero agradecer a mis directores de tesis, Julio y María Engracia, por haber confiado en mí y haberme dado la oportunidad de llevar a cabo este trabajo. Por sus consejos e ideas que han permitido el buen desenlace de este trabajo. También a Alberto, sin cuya constante ayuda esto no habría podido salir adelante.

También me gustaría destacar la ayuda y el buen ambiente de trabajo creado por todos los miembros del grupo de investigación. Ha sido un placer trabajar junto a todos vosotros. Mi estancia por el Grupo de Arquitecturas Paralelas ha sido una experiencia que no olvidaré jamás.

No puedo olvidar dedicar unas palabras a mis amigos, tanto los que viven aquí como los que no. Han sido muchos años, muchas conversaciones. He compartido con todos vosotros experiencias de todo tipo, momentos que siempre perdurarán en mi memoria. Habéis sido testigos de todas mis quejas y siempre habéis estado presentes para sacarme una risa o para ayudar a despejarme. *Iron from Ice, brothers!*

Finalmente, quiero dedicar unas últimas palabras a Linux, mi perro, quien me ha acompañado durante prácticamente toda la carrera, a lo largo del máster y durante este doctorado. Te dejo estas líneas por ser el que más se alegra al verme cuando vuelvo a casa.

A todos vosotros, gracias.

Contents

List of Figures

List of Tables

Abbreviations and Acronyms

OS	**O**perating **S**ystem
CPU	**C**ore **P**rocessing **U**nit
CMP	**C**hip **M**ulti-**P**rocessor
SMP	**S**hared **M**emory **P**rocessor
SRAM	**S**tatic **R**andom-**A**ccess **M**emory
DRAM	**D**ynamic **R**andom-**A**ccess **M**emory
eDRAM	**e**mbedded **D**ynamic **R**andom-**A**ccess **M**emory
NUCA	**N**on **U**niform **C**ache **A**ccess
L1	First-level
L2	Second-level
LLC	**L**ast-**L**evel **C**ache
LRU	**L**east **R**ecently **U**sed
IPC	**I**nstructions **P**er **C**ycle
MPKI	**M**isses **P**er **K**ilo-**I**nstruction
MRU	**M**ost **R**ecently **U**sed
PS	**P**rivate-**S**hared
PDC	**P**rivate **D**irectory **C**ache
SDC	**S**hared **D**irectory **C**ache
WP	**W**ay-**P**rediction
MGD	**M**ulti-**g**rain **D**irectory
DWP	**D**ynamic **W**ay **P**artitioning
TLB	**T**ranslation **L**ookaside **B**uffer
TF	**T**ag **F**ilter
PIPT	**P**hysically **I**ndexed **P**hysically **T**agged
VIPT	**V**irtually **I**ndexed **P**hysically **T**agged

Abstract

As the core counts increase in each chip multiprocessor generation, CMPs should improve scalability in performance, area, and energy consumption to meet the demands of larger core counts. Directory-based protocols constitute the most scalable alternative. A conventional directory, however, suffers from an inefficient use of storage and energy. First, the large, non-scalable, sharer vectors consume unnecessary area and leakage, especially considering that most of the blocks tracked in a directory are cached by a single core. Second, although increasing directory size and associativity could boost system performance by reducing the coverage misses, it would come at the expense of area and energy consumption.

This thesis focuses and exploits the important differences of behavior between private and shared blocks from the directory point of view. These differences claim for a separate management of both types of blocks at the directory. First, we propose the PS-Directory, a two-level directory cache that keeps the reduced number of frequently accessed shared entries in a small and fast first-level cache, namely *Shared* Directory Cache, and uses a larger and slower second-level *Private Directory Cache* to track the large amount of private blocks. Entries in the Private Directory Cache do not implement the sharer vector, which allows important silicon area savings. Speed and area reasons suggest the use of eDRAM technology, much denser but slower than SRAM technology, for the *Private Directory Cache*, which in turn brings further energy savings. Experimental results show that, compared to a conventional directory, the PS-Directory improves performance while also reducing silicon area and energy consumption.

In this thesis we also show that the shared/private ratio of entries in the directory varies across applications and across different execution phases within the applications, which encourages us to propose Dynamic Way Partitioning (DWP) Directory. DWP-Directory reduces the number of ways with storage for shared blocks and it allows this storage to be powered off or on at run-time according to the dynamic requirements of the applications following a repartitioning algorithm. Results show similar performance as a traditional

directory with high associativity, and similar area requirements as recent state-of-the-art schemes. In addition, DWP-Directory achieves notable static and dynamic power consumption savings.

In addition, this dissertation deals with the scalability issues in terms of power found in processor caches. A significant fraction of the total power budget is consumed by on-chip caches which are usually deployed with a high associativity degree (even L1 caches are being implemented with eight ways) to enhance the system performance. On a cache access, each way in the corresponding set is accessed in parallel, which is costly in terms of energy. This thesis presents the PS-Cache architecture, an energy-efficient cache design that reduces the number of accessed ways without hurting the performance. The PS-Cache takes advantage of the private-shared knowledge of the referenced block to reduce energy by accessing only those ways holding the kind of block looked up. Results show significant dynamic power consumption savings.

Finally, we propose an energy-efficient architectural design that can be effectively applied to any kind of set-associative cache memory, not only to processor caches. The proposed approach, called the Tag Filter (TF) Architecture, filters the ways accessed in the target cache set, and just a few ways are searched in the tag and data arrays. This allows the approach to reduce the dynamic energy consumption of caches without hurting their access time. For this purpose, the proposed architecture holds the X least significant bits of each tag in a small auxiliary X-*bit*-wide array. These bits are used to filter the ways where the least significant bits of the tag do not match with the bits in the X-*bit* array. Experimental results show that this filtering mechanism achieves energy consumption in set-associative caches similar to direct mapped ones.

We would like to remark that the proposed schemes have been evaluated and compared against state-of-the-art approaches in terms of performance, energy and area. Experimental results show that the proposals presented in this thesis offer a good tradeoff among these three major design axes.

Resumen

Conforme se incrementa el número de núcleos en las nuevas generaciones de multiprocesadores en chip, los CMPs deben de escalar en prestaciones, área y consumo energético para cumplir con las demandas de un número núcleos mayor. Los protocolos basados en directorio constituyen la alternativa más escalable. Un directorio convencional, no obstante, sufre de una utilización ineficiente de almacenamiento y energía. En primer lugar, los grandes y poco escalables vectores de compartidores consumen una cantidad de energía de fuga y de área innecesaria, especialmente si se tiene en consideración que la mayoría de los bloques en un directorio solo se encuentran en la cache de un único núcleo. En segundo lugar, aunque incrementar el tamaño y la asociatividad del directorio aumentaría las prestaciones del sistema, esto supondría un incremento notable en el consumo energético.

Esta tesis estudia las diferencias significativas entre el comportamiento de bloques privados y compartidos en el directorio, lo que nos lleva hacia una gestión separada para cada uno de los tipos de bloque. Proponemos el PS-Directory, una cache de directorio de dos niveles que mantiene el reducido número de las entradas compartidas, que son los que se acceden con más frecuencia, en una estructura pequeña de primer nivel (concretamente, la *Shared Directory Cache*) y que utiliza una estructura más grande y lenta en el segundo nivel (*Private Directory Cache*) para poder mantener la información de los bloques privados. Las entradas en la Private Directory Cache no implementan el vector de compartidores, lo que conlleva importantes ahorros de energía y área. Debido a temas de área y latencia, se nos sugiere la utilización de tecnología eDRAM, mucho más densa pero más lenta que la tecnología SRAM, para la Private Directory Cache, consiguiendo así ahorros de energía mayores. Los resultados experimentales muestran que, comparado con un directorio convencional, el PS-Directory consigue mejorar las prestaciones a la vez que reduce el área de silicio y el consumo energético.

Ya que el ratio compartido/privado de las entradas en el directorio varia entre aplicaciones y entre las diferentes fases de ejecución dentro de las aplicaciones, proponemos el Dynamic Way Partitioning (DWP) Directory. El DWP-Directory reduce el número de

vías que almacenan entradas compartidas y permite que éstas se enciendan o apaguen en tiempo de ejecución según los requisitos dinámicos de las aplicaciones según un algoritmo de reparticionado. Los resultados muestran unas prestaciones similares a un directorio tradicional de alta asociatividad y un área similar a otros esquemas recientes del estado del arte. Adicionalmente, el DWP-Directory obtiene importantes reducciones de consumo estático y dinámico.

Esta disertación también se enfrenta a los problemas de escalabilidad que se pueden encontrar en las memorias cache. Las caches on-chip consumen una parte significativa del consumo total del sistema. Estas caches implementan un alto nivel de asociatividad (las caches L1 ya se implementan con ocho vías para potenciar las prestaciones del sistema). En un acceso a la cache, se accede a cada vía del conjunto en paralelo, siendo así un acción costosa en energía. Esta tesis presenta la arquitectura PS-Cache, un diseño energéticamente eficiente que reduce el número de vías accedidas sin perjudicar las prestaciones. La PS-Cache utiliza la información del estado privado-compartido del bloque referenciado para reducir la energía, ya que tan solo accedemos a un subconjunto de las vías que mantienen los bloques del tipo solicitado. Los resultados muestran unos importantes ahorros de energía dinámica.

Finalmente, proponemos otro diseño de arquitectura energéticamente eficiente que se puede aplicar a cualquier tipo de memoria cache asociativa por conjuntos y no solo a caches de procesador. La propuesta, la Tag Filter (TF) Architecture, filtra las vías accedidas en el conjunto de la cache, de manera que solo se mira un número reducido de vías tanto en el array de etiquetas como en el de datos. Esto permite que nuestra propuesta reduzca el consumo de energía dinámico de las caches sin perjudicar su tiempo de acceso. Para esto, la arquitectura sugerida mantiene los X bits menos significativos de cada etiqueta en una estructura auxiliar. Estos bits se utilizan para filtrar aquellas vías en las que estos bits menos significativos de la etiqueta no se correspondan. Los resultados experimentales muestran que este mecanismo de filtrado es capaz de obtener un consumo energético en caches asociativas por conjunto similar de las caches de mapeado directo.

Nos gustaría señalar que los esquemas propuestos han sido evaluados y comparados contra otras propuestas del estado del arte en prestaciones, energía y área. Los resultados

experimentales muestran que las propuestas presentadas en esta tesis consiguen un buen compromiso entre estos tres importantes pilares de diseño.

Resum

Conforme s'incrementen el nombre de nuclis en les noves generacions de multiprocessadors en xip, els CMPs han d'escalar en prestacions, àrea i consum energètic per complir en les demandes d'un nombre de nuclis major. El protocols basats en directori són l'alternativa més escalable. Un directori convencional, no obstant, pateix una utilització ineficient d'emmagatzematge i energia. En primer lloc, els grans i poc escalables vectors de compartidors consumeixen una quantitat d'energia estàtica i d'àrea innecessària, especialment si es considera que la majoria dels blocs en un directori només es troben en la cache d'un sol nucli. En segon lloc, tot i que incrementar la grandària i l'associativitat del directori augmentaria les prestacions del sistema, això suposaria un increment notable en el consum d'energia.

Aquesta tesis estudia les diferències significatives entre el comportament de blocs privats i compartits dins del directori, la qual cosa ens guia cap a una gestió separada per a cada un dels tipus de bloc. Proposem el PS-Directory, una cache de directori de dos nivells que manté el reduït nombre de les entrades de blocs compartits, que són els que s'accedeixen amb més freqüència, en una estructura menuda de primer nivell (concretament, la Shared Directory Cache) i que empra una estructura més gran i lenta en el segon nivell (Private Directory Cache) per poder mantenir la informació dels blocs privats. Les entrades en la Private Directory Cache no implementen el vector de compartidors, fet que provoca importants estalvis d'àrea i energia. Per motius d'àrea i latència, se'ns suggereix la utilització de tecnologia eDRAM, molt més densa però més lenta que la tecnologia SRAM, per a la Private Directory Cache; aconseguint així majors estalvis d'energia. Els resultats experimentals mostren que, comparat amb un directori convencional, el PS-Directory aconsegueix millorar les prestacions a la vegada que redueix l'àrea de silici i el consum energètic.

Ja que la ràtio compartit/privat de les entrades en el directori varia entre aplicacions i entre les diferents fases d'execució dins de les aplicacions, proposem el Dynamic Way Partitioning (DWP) Directory. DWP-Directory redueix el nombre de vies que emmagatzemen entrades compartides i permeten que aquest s'encengui o apagui en temps

d'execució segons els requeriments dinàmics de les aplicacions seguint un algoritme de reparticionat. Els resultats mostren unes prestacions similars a un directori tradicional d'alta associativitat i una àrea similar a altres esquemes recents de l'estat de l'art. Adicionalment, el DWP-Directory obté importants reduccions de consum estàtic i dinàmic.

Aquesta dissertació també s'enfronta als problemes d'escalabilitat que es poden trobar en les memòries cache. Les caches *on-chip* consumeixen una part significativa del consum total del sistema. Aquestes caches implementen un alt nivell d'associativitat (les caches L1 ja són implementades en huit vies per potenciar les prestacions del sistema). En un accés a la cache, s'accedeix a cada via del conjunt en paral·lel, essent així una acció costosa en energia. Aquesta tesis presenta l'arquitectura PS-Cache, un disseny energèticament eficient que redueix el nombre de vies accedides sense perjudicar les prestacions. La PS-Cache utilitza la informació de l'estat privat-compartit del bloc referenciat per a reduir energia, ja que només accedim al subconjunt de vies que mantenen blocs del tipus sol·licitat. Els resultats mostren uns importants estalvis d'energia dinàmica.

Finalment, proposem un altre disseny d'arquitectura energèticament eficient que es pot aplicar a qualsevol tipus de memòria cache associativa per conjunts i no només a les caches de processadors. La proposta, la Tag Filter (TF) Architecture, filtra les vies accedides en el conjunt de la cache, de manera que només un reduït nombre de vies es miren tant en el array d'etiquetes com en el de dades. Això permet que la nostra proposta redueixi el consum dinàmic energètic de les caches sense perjudicar el seu temps d'accés. Per a aquest propòsit, l'arquitectura sugerida manté els X bits menys significatius de cada etiqueta en una estructura auxiliar. Aquests bits s'utilitzen per filtrar aquelles vies en les que aquests bits menys significatius de l'etiqueta no es corresponguin. Els resultats experimentals mostren que aquest mecanisme de filtre és capaç d'obtenir un consum energètic en caches associatives per conjunt similar al de les caches de mapejada directa.

Ens agradaria senyalar que els esquemes proposats han sigut evaluats i comparats contra altres propostes del estat de l'art en prestaciones, energia i àrea. Els resultats experimentals mostren que les propostes presentades en aquesta tesis conseguixen un bon compromís entre aquestros tres importants pilars de diseny.

Chapter 1

Introduction

This chapter first presents the scalability problems found in the cache hierarchy of current CMP architectures and which are the focus of this dissertation. Then, a brief description of those issues that have been addressed in the literature is introduced. Afterwards, the objective of this dissertation and the different contributions of the thesis are discussed. Finally, a summary about how the rest of this dissertation deals with scalability is given.

1.1 Problem Description

As transistor technology miniaturizes, silicon resources become more abundant. Consequently, the core count is continually increasing in current shared-memory chip-multiprocessors (CMP). CMP systems must be designed to accommodate specific area and power budgets. Both power and area technological constraints represent major design concerns since they prevent future manycore CMPs from scalability with future increasing core counts.

Power consumption is mainly distributed among cores and large on-chip cache memories in current designs. Cache memories occupy a large percentage of the CMP area [1] to mitigate the huge penalties of accessing the off-chip main memory,and consume an important percentage of the overall power budget Giving more silicon area and power to the cache hierarchy and related structures (e.g. directory caches) leaves less space and power for cores, which could force CMP designs with simpler cores so yielding to lower performance, especially when running single-threaded applications [2]. In this regard, directory caches and processor caches are among the critical components in the on-chip cache hierarchy, therefore their design should be revisited in order to provide new memory structures that enable scalability.

Directory-based coherence is the commonly preferred approach in current CMPs over snoop-based coherence, since the former approach keeps track of cached blocks to avoid the use of broadcast messages. Two main design choices have been used in both research proposals [3, 4] and commercial processors [5–7] to implement CMP directories: *Duplicate Tags* and *Sparse Directories*. Both approaches present different design concerns. The Duplicate Tags approach does not scale for large systems due to the high energy consumed by the highly associative lookups, required to build the *sharer vector*. Sparse directories use a cache-like structure, the directory cache, which is implemented as a typical processor cache to keep track of the cached blocks. This approach reduces the associative lookups but presents two main disadvantages: i) it implements a large sharer vector that is expected to introduce important on-chip area and leakage overheads in future CMPs [8], and ii) copies in the processor caches being tracked must be invalidated when the associated directory entry is evicted. In spite of these disadvantages, due to power reasons, directory caches are the preferred approach and they are the focus of this thesis.

The key challenge when addressing scalability in sparse directories lies on reducing the overhead in area and power introduced by the sharer vector. Many research focusing on compression [9], hierarchical representation [10], multigrain-directories [11] has been done to address this challenge. This research mainly aims to reduce the average sharer vector length of a monolithic cache directory. Unlike this research, we effectively address the sharer vector overhead in terms of area and energy based on the characteristics of the blocks being tracked.

Most of the cache memories power consumption is due to dynamic power while a small fraction is due to static power. Dynamic power consumption comes from switching activity of transistors during cache accesses while static power consumption comes from current leaking, even when the cache is not being accessed. Cache designers must reach a compromise among performance, cost, size, and power/energy dissipation.

Dynamic energy consumption depends on the transistor switching activity, thus the more frequently the cache is accessed (e.g. L1 caches) the higher the dynamic consumption. This concern is even more important in CMPs than in monolithic processors since in CMPs, processor caches can be accessed both from the processor side and from the interconnection network side due to coherence requests, which increases the number of accesses. Due to performance reasons, processor caches are being deployed with a high associativity degree. In high-performance microprocessors, all the cache ways in the target set are accessed concurrently on every cache lookup. Thus, the associativity degree defines the number of accessed tags on each cache access. In addition, caches include one comparator per way, and compare as many tags as the number of ways. As a consequence, the dynamic energy dissipated per access increases with the cache associativity.

Previous research on the design of energy-efficient caches addressing dynamic power has focused on minimizing the internal transistor activity during a cache access. That activity comes from reading, comparing, and writing tags in the tag array, and from reading and writing data in the data array.

Important cache energy reduction approaches have focused on monolithic processors in the past such as Cache Decay [12], Drowsy Caches [13], and Way Guard [14]. Some of them, *e.g.* [15], were originally developed to reduce cache access time, but subsequent research has proven that these schemes provide important energy savings. However,

since these schemes are not directly applied to CMPs, recent research [16] has dealt with energy savings on CMPs when running parallel workloads.

In summary, in this work we address the scalability problems in terms of area and energy of the aforementioned structures (i.e. directory caches and processor caches) in CMPs. These caches need to accommodate to specific area and power budgets, hence becoming a main design concern, as represent a large percentage of both the on-chip area and of the overall energy budget. In this thesis we rely on the analysis of the behavior and key characteristics of the workloads to propose architectural approaches that face this problem.

1.2 Dealing with Scalability in Future CMPs

Many efforts have been carried out in both the industry and academy to deal with power and area focusing on the cache subsystem, including processor caches, off-chip caches and directory structures. Regarding the latter structures, directory caches have been proven to provide effectiveness and scalability, both in terms of power and area, for a small to medium number of cores. However, these design issues must be properly faced for future systems since the pressure on achieving good cache performance increases with the core counts. There are two main ways to tackle these issues: i) architectural solutions to achieve a good trade off among performance, area, and power; and ii) mingling disparate technologies in a power and/or area aware designs. Both ways can be applied independently or together, as proposed in this work. Below, we summarizes the main features of the considered RAM technologies.

On-chip caches have been typically implemented with SRAM technology (6 transistors per cell) which consumes important amounts of power and area. A few years ago, technology advances have allowed to embed DRAM (eDRAM) cells in CMOS technology [17]. An eDRAM cell integrates a trench DRAM storage into a logic circuit technology. Table 1.1 highlights the main properties of these technologies. Compared to SRAM, eDRAM cells have both less power consumption and higher density but lower speed. Because of the reduced speed, eDRAM cells have not been used in manufactured first-level caches of high-performance processors. In short, both technologies present diverse features regarding density, speed, and power.

Table 1.1: Comparing Technological Features of SRAM versus eDRAM.

Technology	**Density**	**Speed**	**Power**
SRAM	low	fast	high
eDRAM	high	slow	low

These CMOS compatible technologies have been used both in the industry and the academia to design processor caches. For instance, in some modern microprocessors [18–20] SRAM technology is employed in L1 processor caches while eDRAM cells are used to allow huge storage capacity in last level caches. Regarding academia, some recent works [21, 22] mingle these technologies in several cache levels. In short, both technologies properly combined at different (or even the same) cache structures can be used to address speed, area, and power in the cache subsystem.

1.3 Objectives of the Thesis

The main objective of this thesis is to address the aforementioned problems in terms of energy and area in cache-like structures for future CMPs and offer architectural designs to reduce them. This general objective can be broken down in two main sub-objectives according to the type of cache being studied.

Regarding directory caches, we pursue to devise architectural solutions that take advantage of the workload characteristics (e.g. amount of private and shared blocks). For this purpose we will analyze the behavior of parallel workloads from the directory perspective. Based on this behavior, we will try to reduce the cache directory area and energy consumption.

Regarding processor caches, the main objective is to reduce the dynamic energy consumption without hurting the performance. Again, a study of the workload characteristics (e.g. number of cache accesses or distribution of the less significant bits of the tag address). Then, based on these characteristics, we will devise mechanisms to filter the number of accessed ways.

1.4 Contributions of the Thesis

This thesis makes four major contributions, two new directory caches and two filter mechanisms –one of them for processor caches and other that can be applied to any set-associative cache–. Below, these contributions are listed.

- **The PS-Directory**. This thesis shows that private and shared blocks have a different behavior from the directory point of view. Based on this analysis, we propose the PS-Directory as a new architectural design for directory caches. It separates the traditional directory in two main structures, one destined to keep entry of private blocks and the other one to keep track of shared blocks. Each structure is tailored to better adjust to the requirements of each entry type. The combination of SRAM and eDRAM for this design is also evaluated. Results show that energy reductions about 27% are achieved. Additionally, directory area is reduced by 26.35%, while also increasing performance by 14%.

- **The DWP-Directory**. We found that the number of entries of each type in the directory varies both inter and intra applications. Static designs may not be able to adjust to these fluctuations or may do so at the expense of an overhead in both area and energy. To deal with this fact, we propose the DWP-Directory, a flexible architectural design that dynamically adapts the associativity that the structure destines to each type of entry. Results show that the proposal reduces the static and dynamic energy consumed by 31.5% and 59.9%, respectively.

- **The PS-Cache**. Data caches are designed with a high-associativity degree due to performance reasons. We propose a filter mechanism based on the private-shared state of entries in order to reduce the number of ways accessed during each lookup in order to reduce the dynamic energy consumed. Results show energy reductions by 22% and 40% for a L1 and L2 cache, respectively, for both a snoopy and directory coherence protocols.

- **The TF-Architecture**. We also devise a mechanism to filter the accessed number of ways in any set-associative cache, as are typical processor and directory caches. For that the tag array will be divided in two structures and the least significant bits of the tag will be employed to perform the corresponding filtering. Hence, energy consumption can be saved. Results show that the proposal can reduce up

to 87.75% and 89.13% the average number of ways that are looked up on every access to the L1 and L2 caches, respectively; which translates in energy savings by 74.9% and 85.9%.

1.5 Thesis Outline

This dissertation is composed of six chapters. Chapter 2 includes the background and related work. Chapter 3 describes the experimental framework and the evaluation methodology. Chapter 4 presents the PS-Directory and the DWP-Directory architectures proposed in this work. Chapter 5 discusses the PS-Cache and the TF-Architecture filtering mechanisms proposed. Finally, Chapter 6 summarizes this thesis, discusses future work, and enumerates the related publications.

Chapter 2

Background and Related Work

This thesis focuses on two main cache memory structures of the cache hierarchy: directory caches and processor caches. This chapter introduces some basic concepts. First, the current standard for cache memories is presented. Then, different concepts related to coherence protocols are discussed. Afterwards, the 16-core CMP baseline system used throughout this work is discussed. Finally, this chapter summarizes research with similar focus to the one of this thesis, that is scalability of directory and processor cache. This summary is broken down according to the memory structure being studied.

2.1 Background

This section presents some background about the cache hierarchy, NUCA caches, and coherence protocols to help understanding the remaining of this work.

2.1.1 Cache Hierarchy

Since late 1960s, computer architects have implemented cache memories to mitigate the huge gap between processor and main memory speed [23]. Cache memories are a relatively smaller, faster memory which stores copies of the data from frequently used main memory locations. Although originally only one level of cache was available to CPUs, current microprocessors implement two or three cache levels and have a split L1 cache for data and instructions, respectively. High level caches are fast and are designed for performance while lower level caches (e.g. LLCs) are designed for capacity. Hence, LLCs are designed as large memory structures, which significantly increases leakage consumption, in order to keep as much information as possible so reducing capacity misses. Therefore, their sizes range from several hundreds of KB up to several tens of MB [20].

Caches have been typically built with Static Random-Access Memory (SRAM) technology since it is the fastest electronic memory technology. However, SRAM incurs in high *leakage* or static energy consumption, which is a major design concern given that this consumption aggravates as the transistor size shrinks. Because of this reason, SRAM has been left in recent processors for high-level caches and other technologies (e.g. eDRAM) are being recently used in the much larger LLC.

A placement policy is employed to decide in which cache set and cache way a main memory block should be stored. If the placement policy permits choosing any entry in the cache to hold the block, the cache is called *fully associative*. Unfortunately, this is only practical for a small number of entries. On the other hand, if given block can be allocated in just one place in the cache, the cache is *direct-mapped*. This solution offers the best-case time and energy consumption, but the severe strictness on the block location option allows multiple blocks to share the same entry, so increasing the conflict misses. Thus, performance is being penalized when using a *direct-mapped* organization. Because of these reasons, current caches implement a compromise in which each main

memory block can go to any one of N ways in a cache set and are described as N-way set associative. The associativity is a trade-off between performance and consumption, and serves as a middle ground between the fully associative and direct mapped caches.

In order to keep low the number of conflict misses, current L2 and LLCs implement a high number of ways (e.g., 16 or more ways).

2.1.2 Non Uniform Cache Access (NUCA)

The improvements in the integration scale have brought an increase in memory cache's capacity and deeper cache hierarchy (L1, L2, L3, ...), that, in general, occupies a large percentage of the chip area. The problem lies in that the bigger the cache size, the higher the access latency. In a conventional memory design consisting of multiple banks, the worst case scenario is taken into account, which in this case is defined as the time required to access the furthest away cache bank. When cache sizes were not as big, this was not a serious problem, but nowadays it introduces a considerable penalization in the access time.

Non-Uniform Cache Access (NUCA) caches [24] were proposed to solve the on-chip wire delay for future large integrated caches. These schemes embed a network into the cache itself, which allows data to migrate within the cache in some NUCA schemes. Migration policies cluster the working set in the cache region nearest to the processor, which in turn reduces the access time as accessing the nearest banks is less costly in terms of latency.

In Figure 2.1 we can observe several cache configurations and their respective average access times. In the upper row, a traditional UCA along an inclusive multilevel UCA are presented. With this additional level in the memory hierarchy we can reduce the high latencies of the UCA. Then, several design variants of a NUCA, according to how data are mapped into banks, are presented. In the S-NUCA-1 (Static-NUCA-1), the less significant bits are used to select in which bank the block can be found or should be stored into. All banks are connected through data and address buses, which imposes a *wire overhead* of 20.9%, which is a serious performance problem. In the S-NUCA-2, the overhead is reduced up to 5.9% through the use of a 2D interconnection network. Finally, the D-NUCA (Dynamic-NUCA) organization shares a design similar to the S-NUCA-2.

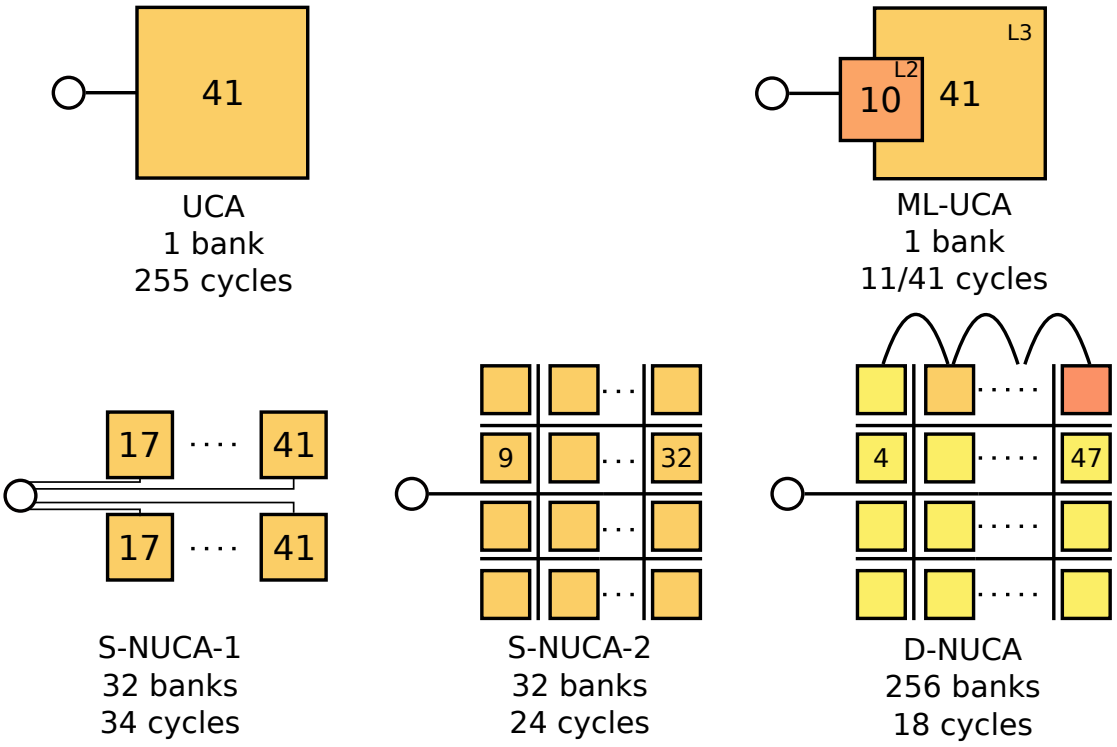


FIGURE 2.1: Average access time depending on the memory configuration.

In this configuration, a block does not have a fixed bank assigned, but regardless of its utilization rate it will be found on nearest or furthest banks. That is, the dynamic migration of data between the banks of the cache is allowed. In this way, NUCA caches can improve the average memory access latency.

2.1.3 Coherence Protocols

In order to maintain the coherence among all processors of a CMP the use of cache coherence protocols is required. In this section, a brief summary about the design options of these protocols will be made.

2.1.3.1 MOESI Protocol

There are many alternatives in the design of coherence protocols depending on the possible states of the blocks stored in their private caches (e.g. L1). These alternatives are often named with acronims that include the first letter of the states they make use of: MOESI, MOSI, MESI, MSI, etc. Each state represents some read and write permissions for the block stored in the private cache. In this thesis, a MOESI protocol has been

used, which has a high number of states (other protocol employ a subgroup of them). The used states are the following:

- M (Modified): A block in modified state has the only valid copy of the data. The core maintains this copy in its cache and has write and read permission over the block. The other private caches cannot have a copy. The copy in the shared L2 (if present) is obsolete. When another core requests this block, the cache with the block in modified state must provide it.

- O (Owner): A block in owner state has a valid copy of the data, but in this case, other copies in a shared state may coexist. There can be only one block in owner state. The core maintaining a copy of this block has read permission, but cannot modify it. When a core tries to modify it, coherence actions are needed to invalidate the other copies. In this way, the owner state is similar to the shared one. The difference resides in the fact that the owner is responsible for providing a copy of the block after a cache miss in other private caches, since the copy in the shared L2 (if present) is obsolete. Furthermore, the evictions of blocks in owner state always need writeback operations.

- E (Exclusive): A block in exclusive state has a valid copy of the data. The other private caches cannot have a copy of this block. The core with this copy has read and write permissions. The shared L2 may also have a valid copy of the data block.

- S (Shared): A block in shared state has a valid copy of the data. Other cores may also have a copy in shared state and one of them in owner state. If no private cache has the block in owner state, the shared L2 also has a valid copy of the block and is responsible for providing it, if requested.

- I (Invalid): A block in invalid state has no valid copy of the data. The valid copies may be found either in the shared L2 cache or in some other private cache.

Figure 2.2 depicts the state transition diagram for a MOESI cache coherence protocol. Often, when a new block is stored in the cache, another block must be evicted from the cache. Since evictions of blocks always result in the cache transitioning to the I state we choose not to show these transitions in the diagram. When a processing core needs

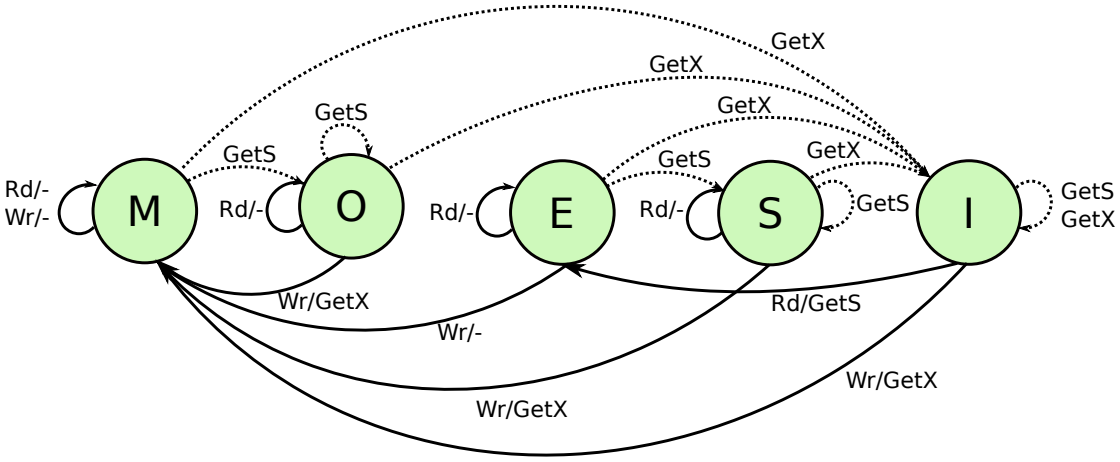


FIGURE 2.2: State transition diagram for a MOESI protocol.

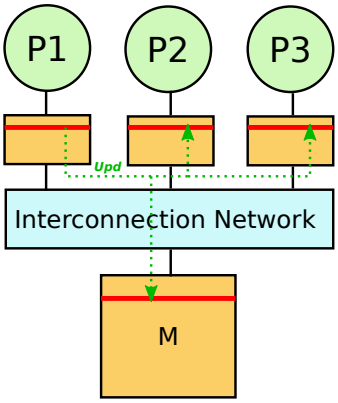


FIGURE 2.3: Update protocol working example.

read permission for a particular cache block (*Rd*) it issues a *GetS* request if it has not read permission for that block (*Rd/GetS*). Otherwise, if the processing core has read permission for that block any request is generated (*Rd/-)*. On the other hand, when the processing core requires write permission (*Wr*) it sends a *GetX* request. In the diagram, solid arrows correspond to transitions caused by local requests while dashed arrows represent transitions due to requests generated by remote processing cores.

2.1.3.2 Update and Invalidation Protocols

When a block that is already stored in the cache of multiple cores is locally written, it is necessary to take specific actions in order to keep the coherence of the memory system. There exist two main approaches: update protocols and invalidation protocols.

In an update protocol, when a core writes to a block, all other copies in the rest of the CMP caches are updated. Subsequent accesses to that block will have an updated

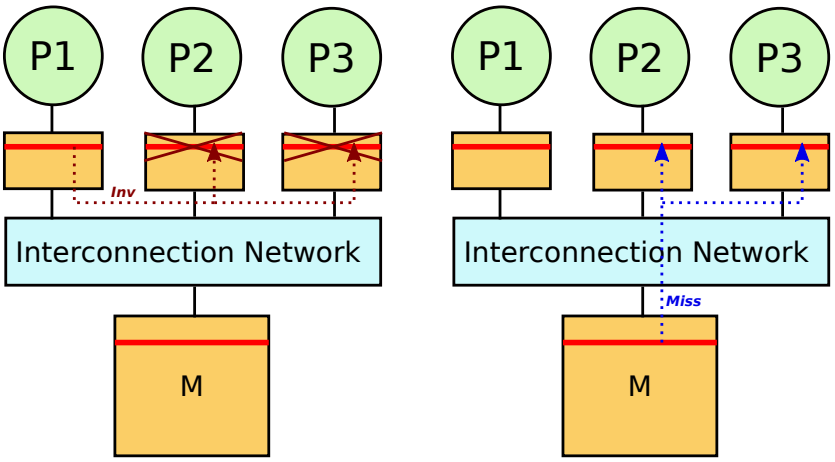


FIGURE 2.4: Invalidation protocol working example.

value of it. In order to do this, it is necessary that this update is notified to the rest of the caches, indicating the implicated block and the updated value. Once the block is updated, the read operations of other cores do not suffer a cache miss. This is specially good when there are writings made by a processor followed by a sequence of readings of other cores. Figure 2.3 depicts an example of such a protocol. Cores P1, P2 and P3 share a block in their respective private caches. Then P1 modifies the block which makes the coherence protocol to trigger an update message to both the other cores and to main memory. After this process, all cores share a coherent copy of the block.

On the other hand, the invalidation protocols, when a core writes to a block, all other copies in the other caches are invalidated. In this case, subsequent accesses will suffer a cache miss. After solving the miss, the cache obtains an updated copy. With this method, once a block is invalidated, new modifications by the same core will not trigger new invalidations. Therefore, the best case scenario for an invalidation protocol is when there are multiple consecutive write operations by the same core. Figure 2.4 shows an example of an invalidation protocol in progress. After P1 modifies the block, an invalidation message is sent to the other cores. A subsequent request of the invalidated block by P2 and P3 triggers a cache miss.

2.1.3.3 Directory-based and Snoopy Protocols

When implementing the coherence mechanisms in the cache system, for example the invalidation or update protocols explained in the previous section, there are two main

approaches: snoopy and directory-based protocols.

In snoopy protocols, the coherence requests must be sent to the caches of all cores, even if they do not have a copy of the block, and they are responsible of finding whether the request affects them in any way or not. For implementation purposes, these protocols require from an interconnection network that simplifies broadcasting (MPs with shared centralized memory, a common bus). Furthermore, the system needs to implement a monitoring mechanism of the bus in order to intercept the invalidation/update requests and adds specific lines to the bus in order to support the target protocol. The main disadvantage of this kind of protocols is that the higher the number of cores of the CMP, the bigger the traffic introduced in the network, thus limiting the system scalability.

Directory protocols aim to solve the scalability shortcomings of snoopy protocols. For this purpose, one of the main objectives of directory-based protocols is the avoidance of broadcasting. Instead, communication only takes place among those processors which are likely to have a copy in their caches. To this end, a structure that stores if a memory line is in a private cache, which processors have a copy (the *sharer vector*), plus another bit to indicate if the line is clean or dirty is used. In a full directory scheme, all information of all memory lines is kept. For example, for a system with n cores, each one with its private cache, a boolean array of size $n + 1$ bits is typically used. If a bit of the sharer vector ($i = 1, ..., n$) is set to true, that means that processor i has a valid copy of the line. The first bit (i.e. $i = 0$) indicates if the line is clean or dirty in that core. An array in which all elements are zero means that the line is found exclusively in main memory. If the first bit ($i = 0$) is active, the line is dirty and only one of the other bits can be active as well.

In order to store the information of all memory lines in a full directory, an important amount of area is needed, which brings scalability concerns. Notice that each directory entry grows linearly with the number of cores, which means a quadratical growth for the directory as a whole. A solution to this problem is reducing the size of a full directory by using a cache-like structure, also referred to as *directory cache*, to keep track of the cached blocks. As any other cache, the capacity of the directory cache is limited, hence, unlike a full directory, the coherence of all processor caches cannot be kept at the same time. In this approach, when a directory entry is evicted due to space constraints (i.e.

a new directory entry is required but no space is available), the blocks in the private caches must be invalidated in order to keep the coherence. This eviction is performed even if the block is still in use in the processor cache.

2.1.3.4 Type of Misses

When accessing a given cache due to a memory reference instruction, it may happen that the requested block is not allocated. This miss compels to look for the block in the lower level of the cache hierarchy or, in case that the miss occurs in the LLC, to access the main memory. There are different types of misses, some inherent to the cache behavior itself, and some to shared memory systems.

The so called 3C misses (cold, capacity and conflict) are inherent to private caches. A cold miss rises on the first access to a block, and the block must be brought first into the cache. They are also called compulsory or first reference misses. These misses are mostly unavoidable, unless there is some kind of prefetching in the system that *warms up* the cache. Capacity misses occur because blocks are being evicted from the cache due to space constraints. They happen when the program working set is much larger than cache capacity and can be solved by employing bigger caches. Lastly, conflict misses appear in case of set-associative or direct-mapped caches. They occur when several blocks are mapped to the same set and such a set is occupied while other sets are not, hence a block of the target set needs to be evicted. They are also called collision misses. A fully associative cache removes this type of misses.

Coherence misses appear in shared memory systems. Coherence must be kept across all private caches. When a core modifies a block, the copies of the block in the other private caches must be invalidated under an invalidation protocol. A subsequent access by another core can trigger this type of miss. Update protocols do not suffer from these misses.

Directory caches have limited space and there is a high probability that they cannot keep track of all the entries in the private caches. As mentioned before, when a directory entry is evicted, the blocks in the processor caches must be invalidated. The result of a block access that has been evicted because there was no free space in the directory cache are

the so called *coverage misses*. Directory protocols which can keep track of all blocks allocated do not have any coverage misses.

2.2 Baseline Architecture

As the previous section shows, there are a lot of design choices regarding the composition of a system. In this section present the baseline architecture that will be employed in the remainder of the thesis.

A tiled CMP architecture consists of a number of replicated *tiles* connected by a switched on-chip direct network. Different tile organizations are possible so, to focus the research, this thesis assumes that each tile contains a processing core with primary caches (both instruction and data caches), a slice of the L2 cache, and a connecting switch to the on-chip network. Cache coherence is maintained at the L1 caches. In particular, a directory-based MOESI coherence protocol with invalidation requests is employed with a directory cache storing coherence information. Both the L2 cache and the directory cache are shared among the different processing cores but they are physically distributed among them, that is, the L2 cache is implemented as a NUCA architecture [24]. Therefore, a fraction of accesses to the L2 NUCA cache is sent to the local slice and the rest to the remote L2 slices. In addition, L1 and L2 caches are non-inclusive, that is, some blocks stored in the L1 caches may not have an entry in the L2 cache (but in the directory). Figure 2.5 shows the organization of a tile (left side) and a 16-tile CMP (right side), which is used as baseline for experimental purposes throughout the work done in this dissertation. The detailed configuration parameters can be found in Section 3.4.

2.3 Related Work

As mentioned above, this thesis addresses scalability issues, in terms of area and energy, of cache-like structures in CMPs. More concretely, this work focuses on directory caches and processor caches. There have been many research works regarding these topics and this section summarizes some of the most recent and relevant proposals of the state-of-the-art.

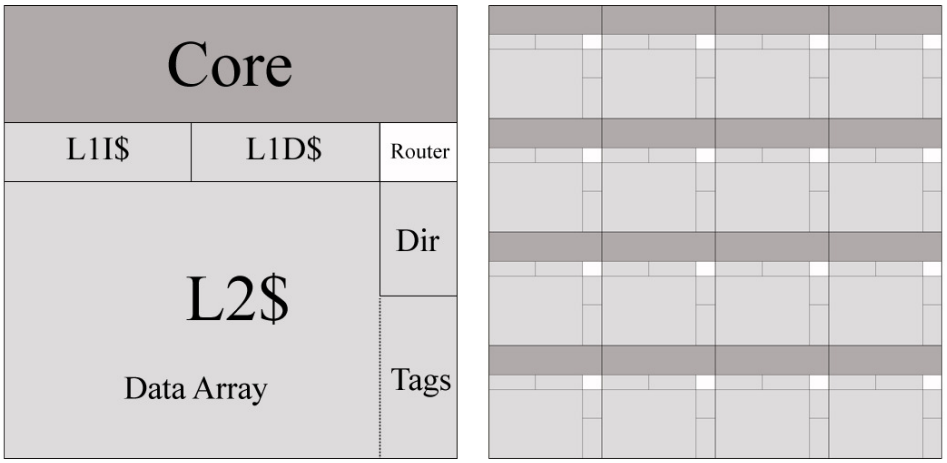


FIGURE 2.5: Organization of the tile assumed in this work and a 4×4 tiled CMP.

2.3.1 Directory Caches

Cache coherence is needed in shared memory systems where multiple cores have copies of the same memory block. Part of this thesis focuses on directory-based protocols, which are the commonly adopted solution for a medium to large core count. These protocols use a coherence directory to track which private (e.g. L1) processor caches share each block. The directory structure is accessed to carry out coherence actions such as sending invalidation requests to serialize write operations, or asking a copy of the block to the owner (e.g. the last processor that wrote it).

Traditional directory schemes do not scale with the core count, which is the current trend in the microprocessor industry. Thus, implementing directories that scale up to hundreds of cores in terms of power and area is a major design concern. Directory implementations, both in academia and industry, follow two main approaches: duplicate-tag directories and sparse directories.

2.3.1.1 Duplicate-tag and directories

Duplicate-tag directories maintain a copy of the tags of all tracked blocks in the lower cache level (e.g. the L1 core cache). Therefore, this approach does not raise directory induced invalidations. The sharer vector is obtained by accessing the highly associative directory structure. This approach has been implemented in modern small CMP systems [6, 7] and is the focus of recent research works [8, 25]. The main drawback of this

approach is the required associativity of the directory structure, which must be equal to the product of the number core caches by the associativity of such caches. This means that a directory access requires a 512 associative search for 64 8-way L1 caches. Duplicate-tag directories are area–efficient, however, the highly associative structures yield to a non-scalable quadratic growth of the aggregated energy consumption [26], so this approach becomes prohibitive for a medium to large core count.

The high power consumption incurred by duplicate-tag directories has led some research to focus on providing high associativity with a small number of ways. Cuckoo Directory [26] uses a different hash function to index each directory way, like skew–associative caches. Hits require a single lookup but replacements require from multiple hash functions to provide multiple candidates, so giving the illusion of a cache with higher associativity but at the expense of higher consumption and latency.

2.3.1.2 Sparse directories

Sparse cache directories [27] are organized as a set-associative cache like structure indexed by the block address. Reducing the directory associativity makes this approach more power–efficient than duplicate–tag directories. Each cache directory entry encodes the set of sharers of the associated tracked block. Conventional approaches use a bit vector, that is, a bit per-core cache, to encode the sharers. In this scheme, the per-core area grows linearly with the core count and the aggregated directory area grows quadratically, since the number of directory structures increases with the number of cores. Previous research works have focused on reducing directory area by focusing on the entry size.

To shorten the entry size some approaches use compression [28–31]. In [9, 28] a two-level cache directory is proposed. The first-level stores the typical sharer vector while the second-level uses a compressed code. When using compression, area is saved at expenses of using an inexact representation of the sharer vector, thus yielding to performance losses. Hierarchical [10] representation of the sharer vector has been also used for entry size reduction purposes. However, hierarchical organizations impose additional lookups on the critical path so hurting latency. Sparse directories may reduce area by reducing the number of directory entries but at the expense of performance since directory evictions force invalidations at the core caches of the blocks being tracked.

Unlike typical sparse directories, a recent scheme [32] uses different entry formats of the same length. Lines with one or a few sharers use a single directory entry while widely shared lines employ several cache lines (multi-tag format) using hierarchical bit vectors. This scheme requires extra complexity and accesses for managing dynamic changes (expanding/contracting) in the format.

Multi-grain directories (MGD) [11] also uses different entry formats of same length and tracks coherence at multiple different granularities in order to achieve scalability. Each MGD entry tracks either a temporarily private memory region or a single cache block with any number of sharers. This proposal is limited to a range of directory interleaving (those higher or equal to the size of a memory region) in order to achieve maximum benefits.

Finally, other proposals [4] focus on reducing the number of entries implemented in the cache directory instead of focusing on the sharer vector. While this approach does not affect the performance, it requires modifying the OS, the Page Table, the processor TLBs and the coherence protocol.

The Hybrid Representation directory [16, 33] considers a different representation for private and shared entries. It proposes a single-cache directory and both types of entries are mingled in the same cache structure. Contents of a private entry are permitted to move to a shared one and vice versa, according to the state of the block.

2.3.2 Processor Caches

CMPs usually accelerate their memory access by using one or more levels of processor caches, being them responsible of a significant percentage of the overall CMP die area [1], and of an important consumption of the overall power budget. This thesis proposes some energy-efficient cache designs that takes advantage of a private-shared detection of the blocks referenced by applications. Hence, this section first reviews some related work about energy-efficient cache designs, and then, it discusses some other optimizations based on a private-shared detection.

2.3.2.1 Energy-efficient cache designs

Caches consumption comes from both leakage (or static) and dynamic consumption. Regarding leakage reductions, Powell *et al.* [34] proposed a Gated-Vdd technique that aims to reduce leakage for instruction caches by reconfiguring them and turning off unused lines. Kaxiras *et al.* [12] proposed the Cache Decay, an approach that reduces the leakage power of processor caches by turning off those cache lines that are predicted to be dead, i.e., not referenced by the processor before they are evicted. Alternatively, Flautner *et al.* [13] exploited the fact that in a particular period of time only a subset of the cache lines are accessed to propose the Drowsy Caches. Different from the previous proposals, the voltage is reduced, but not cut off, for those cache lines that are not being accessed. In this way, the content of the cache line is not erased.

While techniques that aim to save leakage focus on reducing (or cutting off) voltage, dynamic energy saving techniques try to minimize the number of data read and written on every cache access. For example, Albonesi [35] proposed Selective Cache Ways, a cache design able to enable only a subset of the cache ways when the cache activity is not high. The prediction of ways was previously proposed by Calder *et al.* [15] to reduce the access time of set-associative caches. Ghosh et al. [14] proposed Way Guard, a mechanism for large set-associative caches that employs bloom filters to reduce dynamic energy by skipping the lookup of cache lines that do not contain the requested data according to the bloom filter.

Other techniques focus on reducing both leakage and dynamic consumption, for example, by reducing the area of the cache tags, like in the TLB Index-Based Tagging [36], or by performing run-time partitioning, like in the Cooperative Caching scheme [37] or in the ReCaC scheme [38].

The idea of splitting a processor cache in different structures to better exploit application characteristics has been thoroughly addressed in the literature [39]. This optimization may help performance, reduce energy consumption or improve area scalability depending on the data specification and later classification. This thesis will study the benefits of splitting caches in a directory cache instead of in processor caches.

2.3.2.2 Private-shared optimizations

The detection of private and shared data can be employed to optimize performance and power. Kim *et al.* [40] detect the sharing degree of memory pages to reduce the fraction of snoops in a token-based protocol. In this way, they can replace broadcast messages with multicast ones, thus reducing the energy consumption of the interconnection. The R-NUCA approach by Hardavellas *et al.* [41] detects private and read-only pages and maps them efficiently in a distributed NUCA cache. By mapping private pages to the closest NUCA bank to the core accessing them, the access latency is reduced, but also the amount of traffic generated, which will impact in the energy consumed by the network. Cuesta *et al.* [4] deactivate coherence for private pages, thus avoiding their tracking by directory caches. This enables to reduce the directory size up to eight times while still maintaining performance. Reductions in directory area also leads to reduce both dynamic and leakage consumptions.

Some previous works rely on the compiler and/or memory allocator to classify memory pages in order to either remove coherence for private pages [42] or improve data placement [43, 44]. In [43], a data ownership analysis of memory regions is performed at compilation time. This information is transferred to the page table by modifying the behavior of the memory allocator by means of hooks. This proposal is further improved in [44] by considering a new class of data, named as practically-private, which is mapped to the NUCA cache according to a first-touch policy. In [42], private data is not stored at the LLC with the aim of avoiding cache thrashing for private blocks. These works mark statically data as private either by the memory allocator or at compile time, when privacy of some data cannot be guaranteed.

SWEL [45] is a novel hardware-based coherence protocol that uses a private-shared block classification at the directory to allocate shared read-write blocks only at the shared LLC, so removing the need of coherence maintenance for them. The main drawback of that proposal is the latency penalization of accessing shared read-write blocks, which must be served by the LLC cache. POPS [46] decouples data and coherence information in the shared LLC to reduce access latency to this information and to improve the aggregate NUCA capacity. It also employs a directory private-shared classification (this time with the help of a predictor table). Spatio-temporal Coherence Tracking [47] also classifies private and shared data at the directory, accounting for temporal private data. It tries

to find large private regions to merge them in the directory to save directory space. Finally, Ros and Kaxiras [48] proposed VIPS, a complexity-efficient coherence protocol that employs write-back caches for private data for efficiency reasons and write-through caches for shared data for protocol simplicity.

Chapter 3

Experimental Framework

This chapter describes the experimental framework used to carry out the experiments presented in this thesis. This framework includes the simulation packages, the followed methodology and the used workloads.

The remainder of this chapter is organized as follows: Section 3.1 details the various simulation tools employed in the development of this dissertation. Section 3.2 presents the different benchmarks used to evaluate the different proposals. Section 3.3 explains the methodology and the metrics used in the evaluations. Finally, Section 3.4 summarizes the common system parameters utilized in the simulations.

3.1 Simulation tools

In this section we describe the simulation tools employed through this thesis. The proposed schemes have been evaluated with full-system simulation using Virtutech Simics [49] along with the Wisconsin GEMS tool set [50], which enables detailed simulation of multiprocessor systems. The interconnection network has been modeled using GARNET [51], a detailed network simulator included in the GEMS tool set. The area requirements of the proposals, and the cache access times used in the experiments, have been calculated using the CACTI 5.3 tool.

3.1.1 Simics-GEMS

Simics [49] is a functional full-system simulator capable of simulating several types of hardware including multiprocessor systems. Full-system simulation enables us to evaluate our ideas running realistic workloads on top of actual operating systems. In this way, we also consider the operating system behavior. Differently from trace-driven simulators, Simics allows dynamic change of instructions to be executed depending on different input data.

GEMS (General Execution-driven Multiprocessor Simulator) [50] is a simulation environment which extends Virtutech Simics. GEMS is comprised of a set of modules implemented in C++ that plug into Simics and add timing capabilities to the simulator. GEMS provides several modules for modeling different aspects of the architecture. For example, Ruby models memory hierarchies, Opal models the timing of an out-of-order SPARC processor, and Tourmaline is a functional transactional memory simulator. Since we assume simple in-order processing cores we only use Ruby for the evaluations carried out in this dissertation.

Ruby provides an event-driven framework to simulate a memory hierarchy that is able to measure the effects of changes to the coherence protocols. Particularly, Ruby includes a domain-specific language to specify cache coherence protocols called SLICC (Specification Language for Implementing Cache Coherence). SLICC allows us to easily model different cache coherence protocols and it has been used to implement the protocols evaluated in this thesis.

The memory model provided by Ruby is made of a number of components that model the L1 caches, L2 caches, memory controllers, and directory controllers. These components model the timing by calculating the delay since a request is received until a response is generated and injected into the network. All the components are connected using a simple network model that calculates the delay required to deliver a message from one component to another.

Garnet [51] is a detailed interconnection network model inside GEMS. It consists of a detailed *fixed pipeline* model and an approximate *flexible pipeline* model. The *fixed pipeline* model is intended for low-level interconnection network evaluations and models the detailed features of a state-of-the-art network. The *flexible pipeline* model is intended to provide a reasonable abstraction of all interconnection network models, while allowing the router pipeline depth to be flexible adjusted.

3.1.2 CACTI

CACTI (Cache Access and Cycle Time Information) [52] provides an integrated cache and memory access time, cycle time, area, leakage, and dynamic power model. By integrating all these models together, users can have confidence that trade-offs among time, power, and area are all based on the same assumptions and, hence, are mutually consistent.

CACTI is continually being upgraded due to the incessant improvements in semiconductor technologies. Particularly, we employ the version 5.3 for the results presented in this work. We are mainly interested in getting the access latencies and area requirements of both cache and directory structures that are necessary for implementing our proposals, assuming a 32nm process technology.

3.2 Benchmarks

The proposed schemes have been evaluated with a wide range of scientific applications from the SPLASH-2 benchmark suite [53], the ALPBench suite [54], the PARSEC suite [55], scientific applications, and commercial workloads. Table 3.1 shows the list of applications considered in the different studies.

TABLE 3.1: Simulated Applications and input sizes.

SPLASH-2 benchmark suite	
Name	**Input Size**
Barnes	16K particles
Cholesky	tk15
FFT	64K complex doubles
FMM	16K particles
LU	512x512 matrix
Ocean	514×514 ocean
Radiosity	room, -ae 5000.0 -en 0.050 -bf 0.10
Radix	512K keys, 1024 radix
Raytrace	teapot –optimized by removing locks for unused ray ids–
Volrend	head
Water-Nsq	512 molecules
PARSEC suite	
Name	**Input Size**
Blackscholes	simmedium
Swaptions	simmedium
ALPBench suite	
Name	**Input Size**
FaceRec	script
MPGdec	525_tens_040.m2v
MPGenc	output of MPGdec
SpeechRec	script
Scientific benchmarks	
Name	**Input Size**
Tomcatv	256 points
Unstructured	Mesh.2K
Commercial workloads	
Name	**Input Size**
Apache	4000 transactions
SPEC-JBB	4000 transactions

3.2.1 Barnes

The *Barnes* application simulates the interaction of a system of bodies (e.g. galaxies or particles) in three dimensions over a number of time steps, using the Barnes-Hut hierarchical N-body method. Each body is modeled as a point mass and exerts forces on all other bodies in the system. To speed up the inter body force calculations, groups of bodies that are sufficiently far away are abstracted as point masses. In order to facilitate this clustering, physical space is divided recursively, forming an octree. The

tree representation of space has to be traversed once for each body and rebuilt after each time step to account for the movement of bodies.

The main data structure in Barnes is the tree itself, which is implemented as an array of bodies and an array of space cells that are linked together. Bodies are assigned to processors at the beginning of each time step in a partitioning phase. Each processor calculates the forces exerted on its own subset of bodies. The bodies are then moved under the influence of those forces. Finally, the tree is regenerated for the next time step. There are several barriers for separating different phases of the computation and successive time steps. Some phases require exclusive access to tree cells and a set of distributed locks is used for this purpose. The communication patterns are dependent on the particle distribution and are quite irregular. No attempt is made at intelligent distribution of body data in main memory since this is difficult at page granularity and not very important for performance.

3.2.2 Cholesky

The blocked sparse Cholesky factorization kernel factors a sparse matrix into the product of a lower triangular matrix and its transpose. It is similar in structure and partitioning to the LU factorization kernel, but has two major differences: i) it operates on sparse matrices, which have a larger communication to computation ratio for comparable problem sizes, and ii) it is not globally synchronized between steps.

3.2.3 FFT

The *FFT* kernel is a complex one-dimensional version of the radix-$\sqrt{n}$ six-step FFT algorithm, which is optimized to minimize inter-processor communication. The data set consists of the n complex data points to be transformed, and another n complex data points referred to as the *roots of unity*. Both sets of data are organized in $\sqrt{n}$ x$\sqrt{n}$ matrices partitioned so that every processor is assigned a contiguous set of rows which are allocated in its local memory. Synchronization in this application is accomplished by using barriers.

3.2.4 FMM

Like Barnes, the *FMM* application also simulates a system of bodies over a number of time steps. However, it simulates interactions in two dimensions using a different hierarchical N-body method called the adaptive Fast Multipole Method. As in Barnes, the major data structures are body and tree cells, with multiple particles per leaf cell. FMM differs from Barnes in two main respects: i) the tree is not traversed once per body, but only in a single upward and downward pass (per time step) that computes interactions among cells and propagates their effects down to the bodies and, ii) the accuracy is not controlled by how many cells a body or cell interacts with, but by how accurately each interaction is modeled. The communication patterns are quite unstructured, and no attempt is made at intelligent distribution of particle data in main memory.

3.2.5 LU

The LU kernel factors a dense matrix into the product of a lower triangular and an upper triangular matrix. The dense $n \times n$ matrix A is divided into an $N \times N$ array of $B \times B$ blocks ($N = NB$) to exploit temporal locality in each sub matrix elements. To reduce communication, block ownership is assigned using a 2-D scatter decomposition, with blocks being updated by the processors that own them. The block size B should be large enough to keep the cache miss rate low, and small enough to maintain good load balance. Fairly small block sizes ($B = 8$ or $B = 16$) strike a good balance in practice. Elements within a block are allocated contiguously to improve spatial locality, and blocks are allocated locally to processors that own them.

3.2.6 Ocean

The *Ocean* application studies large-scale ocean movements based on eddy and boundary currents. The algorithm simulates a cuboidal basin using a discretized circulation model that takes into account wind stress from atmospheric effects and the friction with ocean floor and walls. The algorithm performs the simulation for many time steps until the eddies and mean ocean flow attain a mutual balance. The work performed

every time step essentially involves setting up and solving a set of spatial partial differential equations. For this purpose, the algorithm discretizes the continuous functions by second-order finite-differencing, sets up the resulting difference equations on two-dimensional fixed-size grids representing horizontal cross-sections of the ocean basin, and solves these equations using a red-back Gauss-Seidel multi grid equation solver. Each task performs the computational steps on the section of the grids that it owns, regularly communicating with other processes. Synchronization is performed by using both locks and barriers.

3.2.7 Radiosity

Radiosity is an application of the finite element method to solving the rendering equation for scenes with surfaces that reflect light diffusely. Unlike rendering methods that use Monte Carlo algorithms (such as path tracing), which handle all types of light paths, typical radiosity methods only account for paths which leave a light source and are reflected diffusely some number of times (possibly zero) before hitting the eye; such paths are represented by the code "LD*E". Radiosity is a global illumination algorithm in the sense that the illumination arriving at the eye comes not just from the light sources, but all the scene surfaces interacting with each other as well. Radiosity calculations are viewpoint independent which increases the computations involved, but makes them useful for all viewpoints.

3.2.8 Radix

The *Radix* program sorts a series of integers, called *keys*, using the popular radix sorting method. The algorithm is iterative, performing one iteration for each radix r digit of the keys. In each iteration, a processor passes over its assigned keys and generates a local histogram. The local histograms are then accumulated into a global histogram. Finally, each processor uses the global histogram to permute its keys into a new array for the next iteration. This permutation step requires all-to-all communication. The permutation is inherently a sender determined one, so keys are communicated through writes rather than reads. Synchronization in this application is accomplished by using barriers.

3.2.9 Raytrace

This application renders a three-dimensional scene using ray tracing. A hierarchical uniform grid is used to represent the scene, and early ray termination is implemented. A ray is traced through each pixel in the image plane and it produces other rays as it strikes the objects of the scene, resulting in a tree of rays per pixel. The image is partitioned among processors in contiguous blocks of pixel groups, and distributed task queues are used with task stealing. The data accesses are highly unpredictable in this application. Synchronization in Raytrace is done by using locks. This benchmark is characterized for having very short critical sections and very high contention. Barriers are not used for the *Raytrace* application.

3.2.10 Volrend

The *Volrend* application renders a three-dimensional volume using a ray casting technique. The volume is represented as a cube of voxels (volume elements), and an octree data structure is used to traverse the volume quickly. The program renders several frames from changing viewpoints, and early ray termination is implemented. A ray is shot through each pixel in every frame, but rays do not reflect. Instead, rays are sampled along their linear paths using interpolation to compute a color for the corresponding pixel. The partitioning and task queues are similar to those in Raytrace. Data accesses are input-dependent and irregular, and no attempt is made at intelligent data distribution. Synchronization in this application is mainly accomplished by using locks, but some barriers are also included.

3.2.11 Water-Nsq

The *Water-Nsq* application performs an N-body molecular dynamics simulation of the forces and potentials in a system of water molecules. It is used to predict some of the physical properties of water in the liquid state.

Molecules are statically split among the processors and the main data structure in Water-Nsq is a large array of records that is used to store the state of each molecule. At each time step, the processors calculate the interaction of the atoms within each molecule

and the interaction of the molecules with one another. For each molecule, the owning processor calculates the interactions with only half of the molecules ahead of it in the array. Since the forces between the molecules are symmetric, each pair-wise interaction between molecules is thus considered only once. The state associated with the molecules is then updated.

Although some portions of the molecule state are modified at each interaction, others are only changed between time steps. Most synchronization is done using barriers, although there are also several variables holding global properties that are updated continuously and are protected using locks.

3.2.12 Blackscholes

The *Blackscholes* application is an Intel RMS benchmark. It calculates the prices for a portfolio of European options analytically with the Black-Scholes partial differential equation (PDE).

Blackscholes stores the portfolio with numOptions derivatives in array OptionData. The program divides the portfolio into a number of work units equal to the number of threads and processes them concurrently. Each thread iterates through all derivatives in its contingent and calls function BlkSchlsEqEuroNoDiv for each of them to compute its price.

3.2.13 Swaptions

The *Swaptions* application is an Intel RMS workload which uses the Heath-Jarrow-Morton (HJM) framework to price a portfolio of swaptions. The HJM framework describes how interest rates evolve for risk management and assets liability management for a class of models. Its central insight is that there is an explicit relationship between the drift and volatility parameters of the forward-rate dynamics in a non arbitrage market. Because HJM models are non-Markovian the analytic approach of solving the PDE to price a derivative cannot be used. Swaptions therefore employs Monte Carlo (MC) simulation to compute the prices.

The program stores the portfolio in the swaptions array. Each entry corresponds to one derivative. Swaptions partitions the array into a number of blocks equal to the number of threads and assigns one block to every thread. Each thread iterates through all swaptions in the work unit it was assigned and calls the function HJM Swaption Blocking for every entry in order to compute the price. This function invokes HJM SimPath Forward Blocking to generate a random HJM path for each MC run. Based on the generated path the value of the swaption is computed.

3.2.14 FaceRec

FaceRec uses the CSU face recognizer to recognize images of faces by matching a given input image with images in a given database. The application has been modified so that a separate input image is compared with each image in the subspace to emulate a typical face recognition scenario. The application is first trained with a collection of images. Then, the training data is written to a file so that it can be used in the recognition phase. This is done off line, so only the recognition phase will be used for reporting results.

At the start of the recognition phase, the training data and the image database are loaded. The subspace matrix is created from the image database. The rest of the recognition phase is divided in two sub-phases: projection and distance computation. During the projection phase, the input image given is normalized and projected in the large subspace matrix. Each thread is given a set of columns from the subspace to multiply. During the distance computation phase, the difference between each image in the subspace and the given one is computed by finding the similarity (distance). Each thread is responsible for computing distances for a subset of images in the database.

3.2.15 MPGdec

The *MPGdec* benchmark is based on the MSSG MPEG-2 decoder. It decompresses a compressed MPEG-2 bit-stream. The original image is divided in frames. Each frame is subdivided into 16x16 pixel macro blocks. Contiguous rows of these macro blocks are called a slice. These macro blocks are then encoded independently. The main thread identifies a slice (contiguous rows of blocks) in the input stream and assigns it to another

thread for decoding. The problem here is that the input stream is also variable length encoded. Thus, the main thread has to at least partly decode the input stream, in order to identify slices. This operation results in a staggered assignment of slices to threads and limits the scalability of extracting parallelism.

We have divided this benchmark in transactions, where each transaction is the decoding of one video frame. In particular, the execution of a transaction comprises four phases. First, it performs variable-length Huffman decoding. Second, it inversely quantizes the resulting data. Third, the frequency-domain data is transformed with IDCT (inverse discrete cosine transform) to obtain spatial-domain data. Finally, the resulting blocks are motion-compensated to produce the original pictures.

3.2.16 MPGenc

The *MPGenc* benchmark is based on the MSG MPEG-2 encoder. It converts video frames into a compressed bit-stream. The encoder uses, in principle, the same data structures as the decoder. The encoding process is parallelized by assigning different slices to each thread. However, since these slices can be determined very easily in uncompressed data, the encoding process can be parallelized without much effort by assigning different slices to different threads. The ALPBench version has been modified to use an intelligent three-step motion search algorithm instead of the original exhaustive search algorithm and to use a fast integer discrete cosine transform (DCT) butterfly algorithm instead of the original floating point matrix based DCT. Also, the rate control logic has been removed to avoid a serial bottleneck.

We have divided this benchmark in transactions, where each transaction comprises several phases: motion estimation, form prediction, quantization, discrete cosine transform (DCT), variable length coding (VLC), inverse quantization, and inverse discrete cosine transform (IDCT).

3.2.17 SpeechRec

SpeechRec uses the CMU SPHINX3.3 speech recognizer to convert speech into text. The application has the major phases: feature extraction, Gaussian scoring, and searching the language model or dictionary. First, the feature extraction phase creates 39-element

feature vectors from the speech sample. The Gaussian scoring phase then matches these feature vectors against the phonemes in a database. It evaluates each feature vector based on the Gaussian distribution in the acoustic model (Gaussian model) given by the user. In a regular workload, there are usually 6000+ Gaussian models. The goal of the evaluation is to find the best score among all the Gaussian models and to normalize other scores with the best one found. As this scoring is based on a probability distribution model, multiple candidates of phonemes are kept so that multiple words can be matched. The final phase is the search phase, which matches the candidate phonemes against the most probable sequence of words from the language model and the given dictionary. Similar to the scoring phase, multiple candidates of words (hypotheses) are kept so that the most probable sequence of words can be chosen.

Both the Gaussian scoring and searching phases, which take most of the execution time, have been parallelized. A thread barrier is used for synchronization after each phase. Gaussian models and the number of active nodes are divided equally among the threads in each respective phase for the needed calculations.

3.2.18 Tomcatv

Tomcatv is a highly vectorizable program for the generation of two-dimensional boundary-fitted coordinate systems around general geometric domains such as airfoils, cars, etc. It is based on a method which uses two Poisson equations to produce a mesh which adapts to the physical region of interest. The transformed non-linear equations are replaced by a finite difference approximation, and the resulting system is solved using successive line over relaxation. This benchmark is one of the most sensitive to the speed of the memory system.

3.2.19 Unstructured

Unstructured is a computational fluid dynamics application that uses an unstructured mesh to model a physical structure, such as an airplane wing or body. The mesh is represented by nodes, edges that connect two nodes, and faces that connect three or four nodes. The mesh is static, so its connectivity does not change. The mesh is partitioned spatially among different processors using a recursive coordinate bisection partitioner.

The computation contains a series of loops that iterate over nodes, edges and faces. Most communications occurs along the edges and faces of the mesh. Synchronization in this application is accomplished by using barriers and an array of locks.

3.2.20 Apache

The *Apache* benchmark is a simple command-line tool that allows one to fire requests to an URL, and see how fast the web-application could process them. It is designed to measure the performance of Apache HTTP server, but the tool can also be used to look at the server response time for any particular URL.

3.2.21 SPEC-JBB

The *SPEC-JBB* benchmark is used for evaluating the performance of server side Java. It evaluates the performance of server side Java by emulating a three-tier client/server system (with emphasis on the middle tier). The benchmark exercises the implementations of the JVM (Java Virtual Machine), JIT (Just-In-Time) compiler, garbage collection, threads and some aspects of the operating system. It also measures the performance of CPUs, caches, memory hierarchy, and the scalability of shared memory processors (SMPs).

3.3 Metrics and Methodology

All cache coherence protocols evaluated in this dissertation have been implemented using the SLICC language included in GEMS. Other protocols, like Token, are already provided by the simulator. All the implemented protocols have been exhaustively checked using a tester program provided by GEMS. The tester program stresses corner cases of cache coherence protocols to raise any incoherence by issuing requests that simulate very contended accesses to a few memory blocks.

The proposals presented in this work have been evaluated not only in terms of performance, but also on-chip area required, and energy consumption have been considered since these design issues, as mentioned above, are major design concerns in actual CMPs.

TABLE 3.2: System parameters

Common Memory Parameters	
Cache hierarchy	Non-inclusive
Cache block size	64 bytes
Split L1 I & D caches	64KB, 4-way (256 sets)
L1 cache hit time	2 cycles
Shared single L2 cache	512KB/tile, 8-way (1024 sets)
L2 cache hit time	2 (tag) and 6 (tag+data) cycles
Memory access time	160 cycles
Directory Parameters	
Single directory cache	256 sets, 4 ways (same as L1)
Single directory cache hit time	2 cycles
Coverage ratio	1×
Common Network Parameters	
Topology	2-dimensional mesh (4x4)
Routing technique	Deterministic X-Y
Flit size	16 bytes
Data and control message size	5 flits and 1 flit
Routing, switch, and link time	2, 2, and 2 cycles

All the experimental results reported in this thesis correspond to the parallel phase of each program. We have created benchmark checkpoints in which each application has been previously executed to ensure that memory is warmed up and, hence, avoiding the effects of page faults. Then, we run each application again up to the parallel phase, where each thread is allocated to a particular core. The application is then run with full detail during the initialization of each thread before starting the actual measurements. In this way, we warm up caches to avoid having a huge amount of cold misses.

For evaluating the performance, we measure the total number of cycles employed for each application. Although the IPC (instructions per cycle) constitutes a common metric for evaluating performance improvements, it is not appropriate for multi threaded applications running on multiprocessor systems [56]. This is due to the spinning performed during the synchronization phase of the different threads. For example, a thread can be repeatedly checking the value of a lock until it becomes available, which increases the number of completed instructions (and thus the IPC) but actually the program is not making any progress.

On the other hand, area and energy consumption of the studied approaches have been evaluated using the CACTI tool described in Section 3.1.2.

3.4 Common System Parameters

In this dissertation, the proposed approaches are modeled and evaluated considering tiled CMP architectures like the described in Section 2.2. The baseline system employs a directory-based coherence protocol in order to maintain coherence among the private caches. Our base directory scheme is an on-chip distributed sparse directory with a bit-vector sharing code in each entry. The protocol stores the blocks in the private caches considering MOESI states, and implements a non-inclusive LLC (L2 in our study) cache. Invalidation acknowledgements are directly sent to the requester.

Table 3.2 summarizes the values of the main system parameters used to carry out the experiments.

Chapter 4

Directory Scalability

This chapter describes two proposals that address the scalability problems of directory caches in CMP systems, namely the Private-Shared Directory (PS-Directory) and Directory Way Partitioning Directory (DWP-Directory). Both proposals rely on the different behavior of shared and private entries in the directory in order to achieve energy and area savings and in this way improve the directory scalability.

The implemented schemes have been evaluated and compared to other state-of-the-art approaches (i.e. Multi-Grain Directories and Hybrid Representation). Results show significant energy and area savings, while maintaining, or even improving, performance.

The remainder of this chapter is organized as follows. Section 4.1 first analyzes the behavior of blocks from the directory point of view. Afterwards the PS-Directory scheme is detailed and evaluated. Section 4.2 discusses the variation of associativity requirements of different types of blocks along the execution time of applications. Then, the DWP-Directory and its repartitioning algorithm are explained. Finally, the most representative results obtained in the experiments are shown and discussed.

4.1 PS-Directory

This section first analyzes the distinct type of behavior exhibited by blocks from the directory perspective. Based on this behavior, this section then proposes the Private-Shared Directory (PS-Directory), a two-level directory architecture, discusses its basic working behavior, and evaluates this approach in terms of performance, area, and energy.

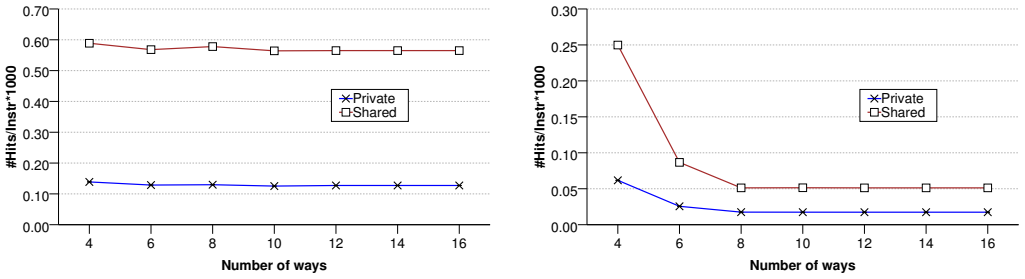
4.1.1 Analyzing the Behavior of Private and Shared Blocks from the Directory Point of View

As a previous step to design the directory, we analyzed the behavior of private and shared blocks from the directory point of view. The results of this analysis can be outlined in four key observations and one finding. As explained below, these five key points advocate to organize directory caches in two independent structures, one for tracking private blocks and the other for shared blocks. We will refer to the directory structure keeping track of private entries as *Private Directory Cache* (PDC) and to the one keeping track of the shared entries as *Shared Directory Cache* (SDC).

- **Observation 1:** *Directory entries keeping track of private blocks do not require the sharer vector field.*
- **Observation 2:** *Most data blocks in parallel workloads are private.*

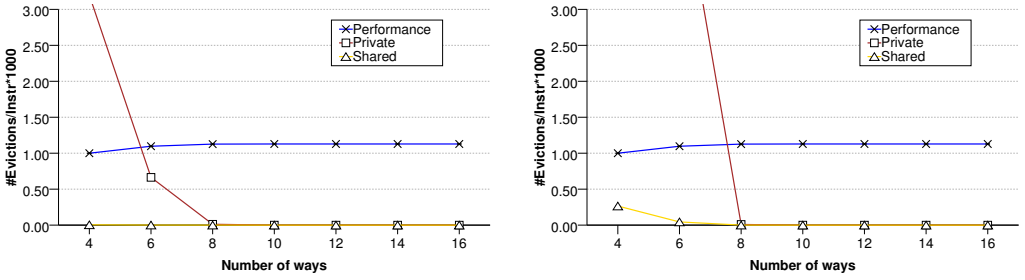
According to these two observations, the PDC should be designed narrower and taller than the SDC, that is, with shorter entries but with a higher number of them. Due to the smaller entry size in the PDC, important area savings can be achieved, especially for systems with a large number of cores, thus offering scalability. Notice that the larger the PDC is (in comparison with the SDC), the more area savings can be obtained, thanks to the missing sharer vector field.

- **Observation 3:** *Most directory hits concentrate on shared entries.*
- **Observation 4:** *Almost all directory entries for private blocks are accessed only once.*



(a) Barnes (b) Blackscholes

FIGURE 4.1: Number of hits to private and shared entries per kilo instruction in a conventional directory.



(a) Barnes (b) Blackscholes

FIGURE 4.2: Number of evictions of private and shared entries per kilo instruction in a conventional directory and its effect on performance.

These observations emphasize that private blocks access the directory either when they are not stored in the processor cache (e.g., the first access to a block or invalidations rising due to directory evictions) or when a write-back is performed (e.g., due to space constraints in the processor cache). The first case will cause a directory miss, while the second case will hit in the PDC and will invalidate the corresponding entry. On the other hand, shared entries are accessed more times due to several cores access the same block. Thus, most directory hits are due to shared blocks. According to this rationale, the SDC should be accessed first so preventing likely useless accesses to the PDC, which will result in energy savings.

Figure 4.1 depicts the number of directory hits (differentiating between shared and private entries) per kilo instructions committed, varying the number of ways in the directory cache and keeping constant the number of sets[1]. Two benchmarks, *Barnes*

[1]Experimental conditions are defined in Section 3.4.

from the SPLASH-2 benchmark suite [53] and *Blackscholes* from PARSEC [55], have been used to illustrate these observations.

As can be seen, the number of hits in entries tracking shared blocks is about 5× larger than that in entries tracking private blocks in Barnes. Entries of private blocks are only looked up again in case a block is replaced either from the directory or from the processor cache, and then asked again by the processor. In both cases, the directory entry is removed, thus when the corresponding private block is looked up in the directory, a miss will occur. Private entries are scarcely accessed in spite of being the number of them much larger than that of shared entries. Results for Blackscholes show minor differences for a higher number of ways because the number of directory evictions is noticeably reduced in this benchmark as the directory capacity increases. With a lower number of evictions, the number of L1 coverage misses will also decrease. Hence the directory will be accessed less frequently. These results suggest that while shared blocks should have a reduced directory access time for performance, this time is not so critical for private blocks. Keeping this observation in mind, we study the potential benefits of using a power and area aware technology to implement the private cache.

- **Finding 1:** *Shared directory entries have much less associativity requirements than private directory entries.*

To quantify the proper associativity degree, we ran experiments with a conventional (or single–cache) directory varying the number of ways. We identified and quantified the number of evicted directory entries that cause subsequent misses in the processor caches, and classified them into private and shared according to the type of the block that was being tracked. Then, the effect of both types of blocks on performance was measured. Misses in the processor caches that occur due to an eviction of a directory entry will be referred to as *coverage misses* as also done is some recent works [4, 57].

We found that private and shared entries have different associativity requirements. Figure 4.2 illustrates the results for two different workloads. Results reveal that the number of evicted shared blocks provoking coverage misses slightly varies with the number of ways, while the number of private blocks drops dramatically. The number of evicted private blocks is really high for a low associativity degree, which translates into significant performance degradation.

Assuming a typical LRU replacement policy and taking into account that entries in the directory tracking private blocks are not accessed again, the time an entry is busy tracking a given block works out like a FIFO policy; that is, in absence of locality, the impact of private blocks on performance mainly depends on the number of ways available to them. If it is too low, it is likely that the block will be forced to leave the processor cache, even though it is still being used, thus increasing the number of coverage misses. On the other hand, with a higher number of ways, we give them more chances before eviction. It can be observed that around 8 ways is enough to stabilize the number of evictions of private blocks as well as the system performance.

4.1.2 PS-Directory Organization and Basic Behavior

The main goal of this approach is to take advantage of the different behaviors exhibited by shared and private directory entries to design scalable directory caches while, at the same time, improving their performance. Figure 4.3 depicts the proposed two-level organization consisting of the Private Directory Cache (PDC) and the Shared Directory Cache (SDC). The PDC is designed with narrower entries since they do not require the sharer vector, and with a larger number of entries because of the expected high number of private blocks. On the other hand, the SDC has a reduced number of entries, due to the smaller number of blocks expected to be shared. This second structure implements the sharer vector in each of its entries and, therefore, this field is only implemented in a small fraction of directory entries.

When an access to a memory block misses in the processor cache, the corresponding block is looked up on the directory for coherence maintenance. Then, if the access results in a directory miss, no other private cache holds it and the block is provided by the corresponding NUCA slice (or by the main memory) to the requesting processor cache, and an entry is allocated in the directory cache to track that block. In the case of using the PS-Directory, this entry is allocated in the PDC since the block is held at this point of time by a single cache. Then, the core identifier is stored in the owner entry field.

On subsequent accesses to that memory block by the same processor, it will find the block in its L1 cache, so no additional accesses to the directory cache are required. On the other hand, when that block is evicted from the processor cache, two main actions are

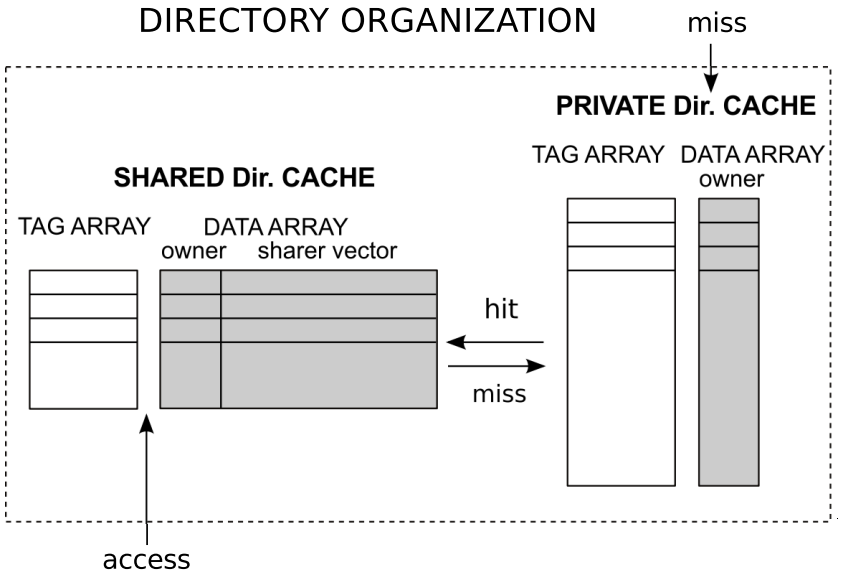


FIGURE 4.3: Private-Shared directory organization.

carried out: i) the data block is written back in the NUCA cache, and ii) the directory cache is notified in order to invalidate the entry of that block (stored in the PDC). Thus, a subsequent access to that block will result in a directory cache miss. This means that the PDC access time does not affect the performance of private blocks since these blocks are provided directly to cores by the NUCA cache or main memory.

If a block being tracked by the PDC is accessed by a core other than the owner, the block becomes shared, since two different private caches will hold a copy of the block. This means that its entry is moved to the SDC and the sharer vector updated accordingly. From then until eviction, coherence of this block is tracked in the SDC. That is, the proposal allows only unidirectional movements from the PDC to the SDC. Bidirectional transfers of entries among both caches have been also explored but the extra hardware cost does not justify the scarce performance benefits.

Regarding timing, directory caches are typically accessed in parallel with the NUCA cache. On a directory hit, the data block can be provided either by the NUCA cache or by a remote processor cache (i.e. the owner). In case the data block is provided by a remote processor cache, the NUCA access is canceled. Analogously, in the PS-Directory both directory cache structures could be accessed simultaneously; however, since most directory accesses concentrate on shared blocks, the PS scheme only accesses the SDC in parallel with the NUCA slice. This way provides major energy savings with minimal performance penalty. Figure 4.4 depicts the timing of this design choice.

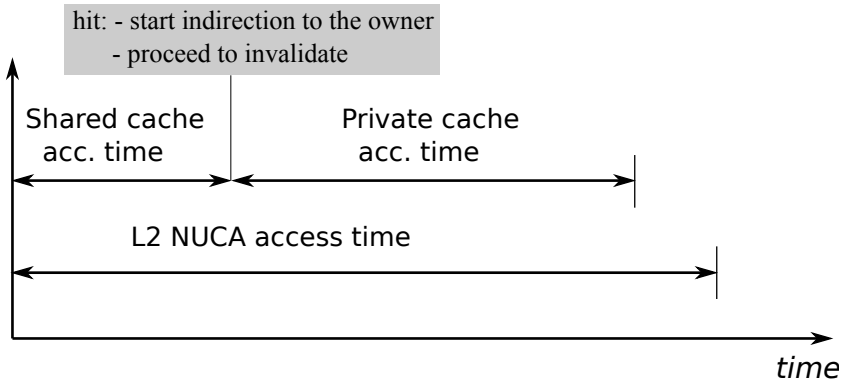


FIGURE 4.4: Parallel access of the Shared cache and the NUCA cache. Private cache is only accessed on a miss in the Shared cache.

Depending on the coherence protocol, specific coherence actions can start as soon as a hit rises in the SDC; for instance, read requests can be forwarded to the owner of the block, or invalidation requests can be issued to the caches sharing the block in case of write requests. On a miss in the SDC, the PDC is accessed. As mentioned above, this access could be also performed in parallel with the Shared cache but at the expense of power while bringing minimal benefits on performance. On a miss in this cache, which is the most frequent case, there will be neither energy or performance gains nor losses by accessing both directory structures in parallel instead of sequentially since both structures have to be accessed, and the sum of their access times is still lower than the NUCA access time that is accessed in parallel. The main difference appears on a Private cache hit. By accessing both directory structures in parallel, the directory access time would be slightly reduced on a private directory hit, but at the expense of higher and unnecessary energy consumption on a shared directory hit. Since hits on the shared directory are more frequent, making this access sequential was the preferred design choice.

Figure 4.5 summarizes the actions carried out by the directory controller on a coherence access, which works as follows:

- When a coherence request reaches the directory, the directory controller looks up first the SDC since it is more likely that the access results in a hit in this cache due to the higher fraction of accesses to shared entries. On a hit, the controller updates (if needed) the sharer vector, performs the associated coherence actions, and cancels the NUCA access (depending on the block state). On a

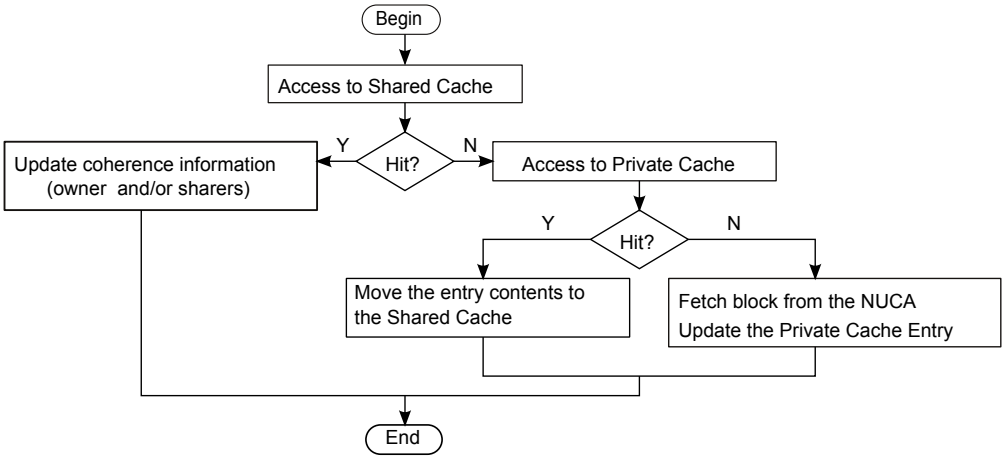


FIGURE 4.5: Directory controller flow-diagram.

miss in the SDC, the controller looks up the PDC. This sequential timing has, on average, negligible impact on performance since most directory accesses are to shared blocks, and most accesses to private blocks fetch the block from the NUCA cache.

- A hit in the PDC means that the block is shared because another core already has a copy of it in its cache. The processor that accessed it the first time will not access the directory again because its cache already holds the block, unless a data cache or directory eviction occurs and then the entry will miss in the directory again. Hence, the directory entry is moved to the SDC. This way ensures that entries for private blocks are retained in the PDC while shared entries are filtered and moved to the SDC.

- On a directory miss, the corresponding block entry is allocated in the PDC to keep track of the missing block. As there is no coherence information stored for that block in any of the two directory caches, then the block is not actually being cached by any processor. Thus, the block is assumed to be private to the core accessing it and the owner information (requesting processor) is updated with the core identifier.

- In the proposed implementation, when a block entry is replaced from any of both directory caches it leaves the directory after performing the corresponding invalidations in the processor caches, and no movement to the other cache directory is allowed.

The PS-Directory proposal reduces area by design with respect to conventional caches implemented with the same number of entries since directory entries in the PDC (most of the total directory entries) are much narrower. In addition, power is also reduced by accessing smaller cache structures sequentially. Nevertheless, the use of two independent organizations with different design goals, speed for the SDC and capacity for the PDC, suggests that using specific technologies addressing these design issues could provide the proposal further energy and area savings.

Low-leakage technologies or transistors with low leakage currents could be used in the PDC, whose number of entries is much higher and its access time is not critical for performance. This chapter explores the use of eDRAM technology in the PDC which provides, as experimental results will show, important area and leakage savings.

4.1.3 Experimental Evaluation

Different configurations for the PS-Directory have been evaluated with a 1× *coverage ratio*, if not stated a different ratio. This ratio indicates the number of directory entries per processor cache line. For instance, in the 1× ratio, each directory cache slice has the same amount of entries as an L1 cache. Two PS-Directory configurations have been evaluated varying its shared-to-private ratio (1:3 and 1:7), that is, the number of entries in the PDC is three and seven times greater, respectively, than that of the SDC. These two directory configurations have been chosen for comparison purposes, because they have the same number of entries (computed as the sum of entries in both directory caches). Additionally, we perform a sensitivity study with lower coverage ratios for our PS-Directory in order to show the significant reduction in directory area and power that it can achieve without degrading the system performance.

Table 4.1 shows the access time and characteristics of the studied directory structures. The first row, labeled as single cache, refers to the conventional single-cache approach (sparse directory) used as baseline. Then, two different PS architectures are presented. Values for the PDC were calculated both for SRAM and eDRAM technologies, and for different coverage ratios. Since CACTI provides latencies in *ns*, we rounded these values to obtain an integer number of processor cycles. The L2 cache access time was assumed to be 6 cycles, and the remaining access times were scaled accordingly. Notice that eDRAM speed is much slower than SRAM speed.

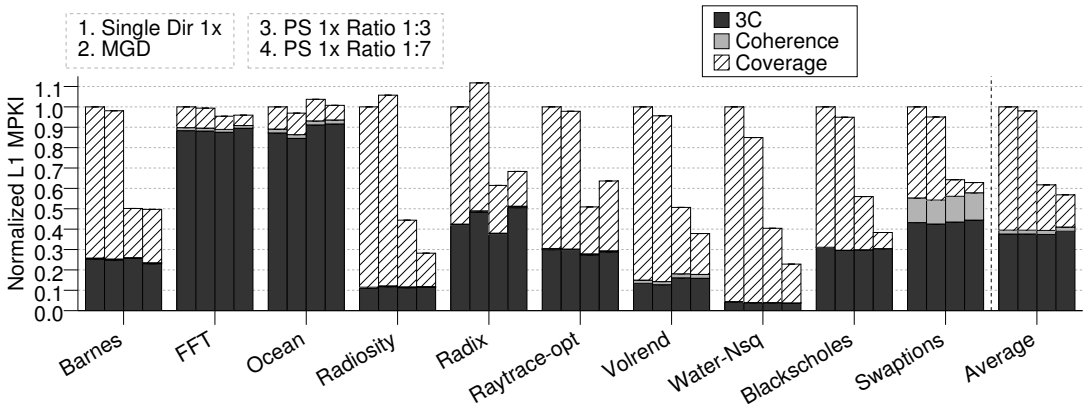
TABLE 4.1: Access time in processor cycles for the different directory caches.

Directory cache	# Ways	1× # Sets	**Access Time (cc)** 1×	0.5×	0.25×	0.125×
Single cache	4	256	2	2	2	-
SDC 1:3	2	128	2	2	2	2
PDC 1:3 SRAM / eDRAM	6	128	2 / 4	2 / 4	2 / 3	2 / 3
SDC 1:7	2	64	2	2	2	2
PDC 1:7 SRAM / eDRAM	7	128	2 / 4	2 / 4	2 / 3	2 / 3

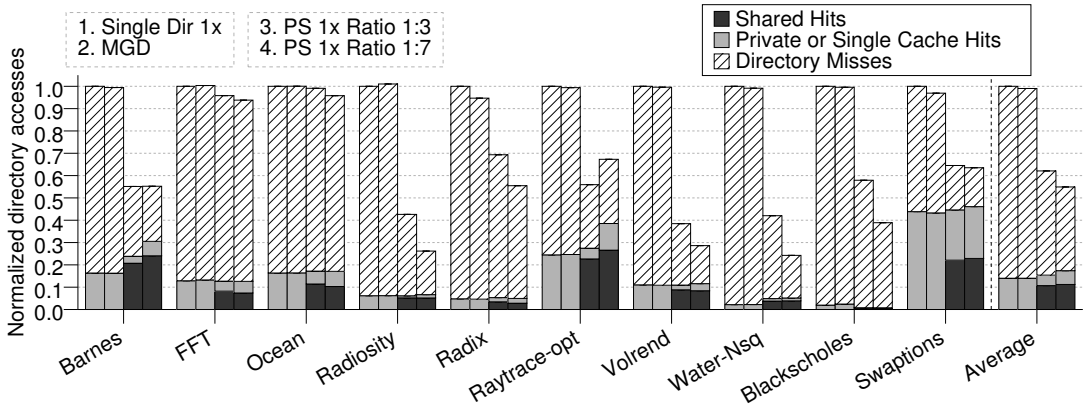
Apart from comparing the PS-Directory against a conventional directory cache with as many entries as the sum of the PDC and the SDC, the PS-Directory has been also compared against the recently proposed state-of-the-art Multi-Grain Directory (MGD) scheme [11]. As presented in Section 2.3 MGD uses different entry formats of same length and tracks coherence at multiple different granularities (either region or single cache entries) in order to provide scalability. By using a single entry instead of using one entry per block in the private region, the coherence directory size can be reduced. Region entries rely on a presence vector to indicate which blocks of the region are allocated in the private L1 cache. On a directory miss, a region entry is allocated in the directory. When a second private cache tries to access a block in a private region, the appropriate bit in the region's presence vector is reset and a block entry is allocated in the directory. Block entries work the same way as they do in conventional sparse directories. In the presented results, the associativity of the MGD is 4 ways as in our baseline, the memory interleaving is 1KB, and the number of entries is 0.5× that of the conventional and PS-Directories. This coverage ratio has been chosen for the MGD, as suggested by their authors, with the aim of providing scalability in terms of area and power by grouping blocks in regions.

4.1.3.1 Impact of PS-Directory on Performance

This section analyzes the performance of the proposed PS-Directory compared to the conventional *sparse* directory and to a multi-grain directory (MGD). The performance of the directory cache must be addressed because it may significantly affect the system performance. Effectively, every time a directory entry is evicted, invalidation messages



(a) Normalized L1 MKPI.



(b) Normalized directory accesses.

FIGURE 4.6: Normalized misses with respect to a conventional single-cache directory.

are sent to the corresponding processor caches for coherence purposes. These invalidations will cause *coverage misses* upon a subsequent memory request to those blocks, therefore impacting on the final performance.

Figure 4.6(a) shows the L1 MPKI (Misses per Kilo Instructions) classified in *3C* (i.e., cold or compulsory, capacity, and conflict), *Coherence*, and *Coverage*. As observed, the PS-Directory cache is able to remove most coverage misses caused by a *single cache* or sparse directory approach with the same number of entries (by 84.2% and 68.2% for 1:7 and 1:3 private-to-shared ratios, respectively). Essentially, this reduction in coverage misses comes from removing conflict misses in the directory cache, which are mainly caused by private directory entries as shown in Section 4.1.1. Therefore, by adding two additional ways to the PDC (at the cost of reducing the number of sets, so the

number of entries remains the same) most directory conflict misses can be avoided. To illustrate where benefits come from, lets study the 1:3 ratio. This ratio provides the same number of sets to the SDC and to the PDC, with 2-way and 6-way associativity, respectively. In other words, this PS organization has exactly the same number of sets as the 4-way single cache and, on average, the same number of ways per set. Thus, this scenario clearly shows that critical *private* sets are efficiently handled by the PS scheme. To sum up, performance benefits mainly come from identifying that the private entries suffer from conflict misses and selectively adding or removing associativity to specific structures depending on the requirements of the type of the entries.

The MGD directory reduces the L1 coverage misses by 3.2% with respect to the single conventional directory. Notice that the MGD is able to reduce the number of coverage misses with half the number of entries than the sparse directory. Nevertheless, this reduction is much lower than the one achieved by the PS-Directory.

Performance of a multilevel directory cache can be quantified as the number of coherence requests that find the required coherence information in the directory, that is, as the overall directory hit ratio regardless of the directory structure that provides such an information. Figure 4.6(b) presents the accesses to each PS-Directory cache classified in misses and hits. In case of a hit, it is also classified in the directory structure that currently has the entry (Private Directory or Shared Directory caches).

Notice that, as expected by design, the PDC shows on average a poor hit ratio despite of the much higher number of entries (3× and 7× times the entries of the SDC), and most directory hits concentrate on the SDC, which corresponds to the smaller and faster directory structure. Remember that each hit in the PDC refers to a private block that becomes shared. Although the 1:7 ratio could seem to have a too small SDC, it provides on average better results than the 1:3 ratio, since it reduces the number of accesses to the directory. *Ratio 1:3* and *ratio 1:7* reduce the number of accesses to the directory by 37.9% and by 45.1%, respectively, while the MGD directory only reduces this number by 1%.

Reducing both the number of coverage misses in the processor caches and the access latency to the directory cache translate into improvements in execution time as shown in Figure 4.7. This figure compares the performance of the studied directory schemes with that of a perfect directory cache. A directory cache is said to be perfect when it does

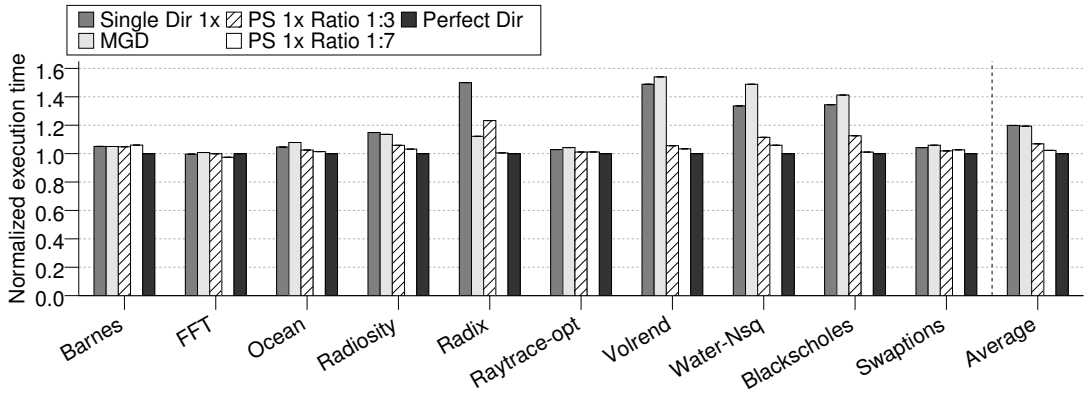


FIGURE 4.7: Normalized execution time with respect to a perfect directory.

not incur in performance degradation, that is, there are no coverage misses. Therefore, a perfect directory cache provides the same performance as a duplicate tags approach but it offers more scalability. Nevertheless, unlike the proposed scheme, there is no reasonable implementation of a duplicate tag approach. Benchmarks with high coverage miss values (i.e. Radix or Blackscholes) are the ones that benefit the most from our proposal or similar ones like MGD. The higher the reduction of coverage misses, the shorter the execution time. Compared to the single directory cache, the PS-Directory reduces execution time on average by 13.6% and 11.1% for the 1:7 and 1:3 shared-to-private ratios, respectively. Compared to the perfect cache, the single cache increases the execution time on average by 22.3%, yielding in some case to unacceptable performance (e.g. an increase by 60% in Radix). However, performance drops of our proposal with respect to the perfect cache are only by 6.4% and 2.9% for the ratios 1:3 and 1:7, respectively.

The small reduction of coverage misses achieved by MGD also brings, on average, small performance gains (by 3.9%) over the conventional single-cache directory. Compared to the PS-Directory, the MGD presents a slow-down of 11.6% and 16.7% considering the 1:3 and 1:7 ratios, respectively. This is due to the fact that entries tracking shared blocks are more frequently accessed at the directory, thus, a shared cache with shorter access time can positively impact on the cache miss latency. In short, results show the PS-Directory as a simple and effective design, which is able to reach performance close to a perfect directory with reduced hardware complexity.

4.1.3.2 Impact of PS-Directory on Area and Energy

This section analyzes how the PS-Directory is able to reduce area and energy consumption compared to a conventional single directory cache and the state-of-the-art MGD; while, as studied above, increasing performance.

Table 4.2 shows the area required for different PS schemes and the single directory cache. Both SRAM and eDRAM technologies, as stated in the *Private (Technology)* column, have been considered for the PDC design, while the smaller SDC is always implemented with fast SRAM technology. As expected, all the PS configurations are able to reduce area, even those entirely implemented with SRAM technology. In particular, compared to the single cache, the PS configurations with SRAM Private caches save by 18.51% and 25.48% of area for 1:3 and 1:7 shared-to-private ratios, respectively. These savings come because the PDC does not include the sharer vector field. In addition, when eDRAM technology is considered, these reductions grow up to 25.02% and 33.12% for 1:3 and 1:7 shared-to-private ratios, respectively.

TABLE 4.2: Area (in $mm^2 * 1000$) of the different PS configurations for 16 cores compared to the Single cache directory.

Directory	Shared	Private (Technology)	Total	Area (%)
Single	19.51	–	19.51	100.00%
PS 1:3	6.40	9.50 (SRAM)	15.90	81.49%
	6.40	8.22 (eDRAM)	14.63	74.98%
PS 1:7	3.45	11.08 (SRAM)	14.54	74.52%
	3.45	9.60 (eDRAM)	13.05	66.88%

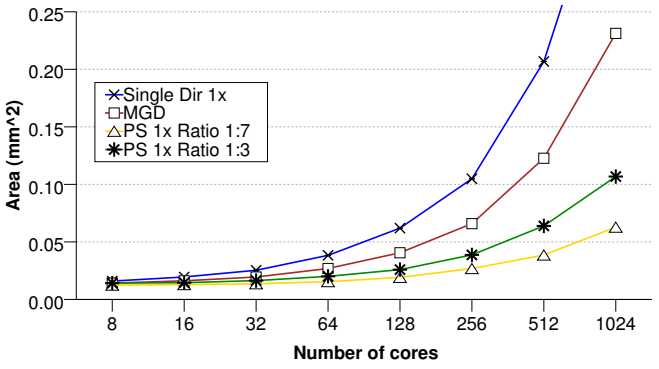


FIGURE 4.8: Scalability analysis in terms of area of the single cache, MGD and the proposed PS-Directory.

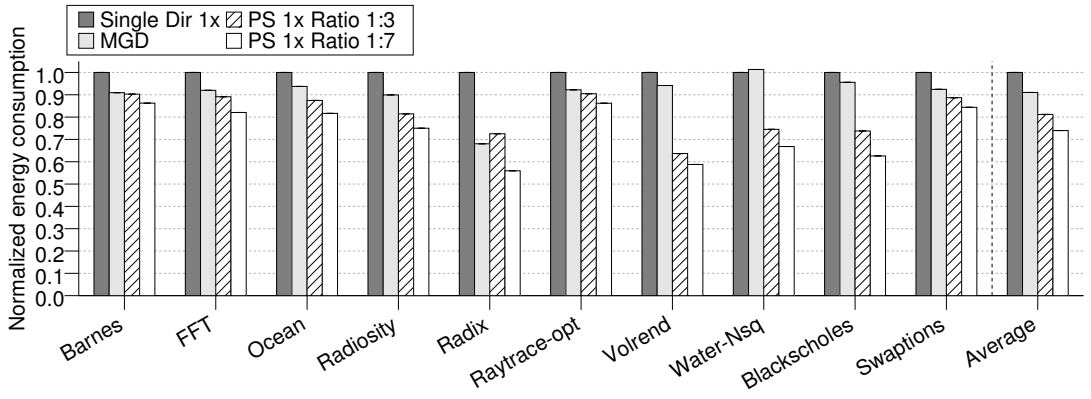
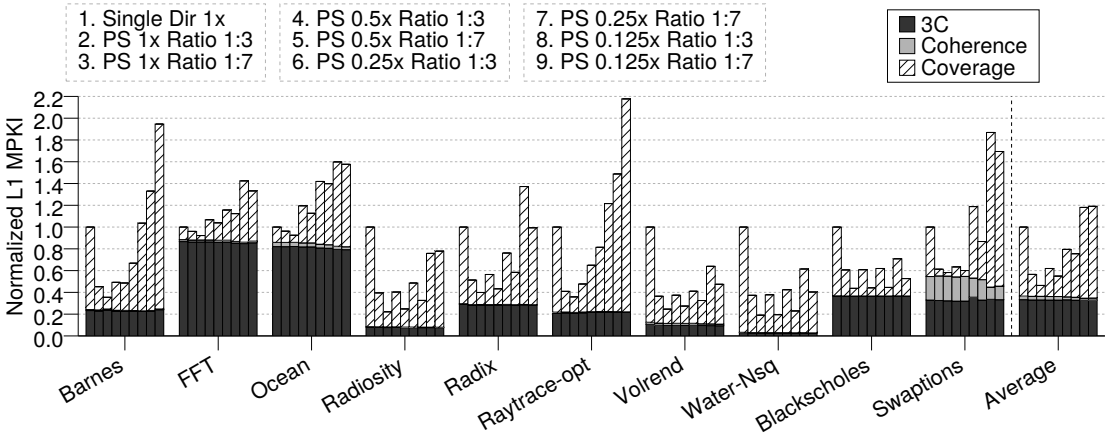


FIGURE 4.9: Normalized energy consumed by the directory with respect to a single-cache directory.

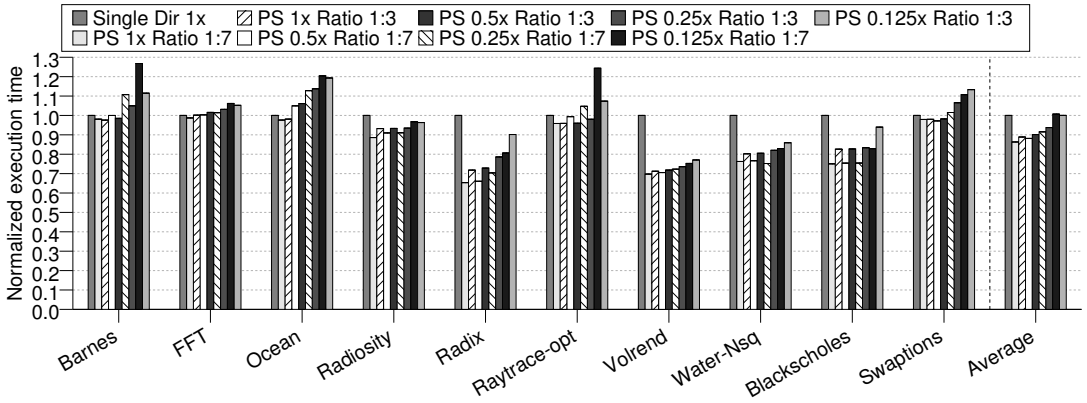
Figure 4.8 depicts the required per-core silicon area for the studied directory configurations. As observed, the single cache directory and the MGD require more area than any of the PS configurations. Additionally, their area requirements grow faster with the number of cores. Notice that in spite of using half the number of entries of a PS-Directory, the MGD scales poorer than the PS-Directory. The PS-Directory is able to reduce by 84.3% (ratio 1:7) and 73.3% (ratio 1:3) the area required by the conventional directory for a 1024–core system, even though all of them have the same number of entries. Thus, the PS-Directory overcomes one of the biggest problems that sparse directories present, namely, their scalability.

On the other hand, the PS-Directory attacks energy consumption by design, especially leakage, since it uses two structures with less complexity and less storage capacity than a single conventional directory cache.

Figure 4.9 shows the total energy consumed during the benchmarks execution, normalized with respect to the single cache directory. SRAM technology has been assumed in the PDC of the PS-Directory. We can observe that a PS-Directory with the same number of entries as a single cache directory can save around 27% and 20.5% of the energy consumption of the single cache directory for the 1:7 and the 1:3 ratios, respectively, while MGD *only* reduces by 8.9%. This means that a PS-Directory, with either 1:3 or 1:7 ratio, is able to improve the multi-grain scheme in terms of energy. In short, the PS-Directory reduces energy consumption by 18.7% with respect to the state-of-the-art MGD approach. Moreover, when taking eDRAM technology in the PDC into



(a) Normalized L1 MPKI.



(b) Normalized execution time.

FIGURE 4.10: Normalized performance with respect to the conventional single-cache directory.

consideration, the savings are as high as 87.3% and 81.3% for the 1:7 and the 1:3 ratios, respectively, with respect to the single cache directory.

4.1.3.3 Directory Coverage Ratio Analysis

This section evaluates the impact on performance of reducing the directory coverage ratio, that is, the number of entries in the PS-Directory cache. As the number of implemented entries is reduced in the directory cache, a degradation in performance is expected, but at the same time, area and energy consumption will improve. The ideal directory cache size is the one that entails negligible impact on performance while at the same time allows area and energy savings.

TABLE 4.3: Area (in $mm^2 * 1000$) of the different PS configurations for 16 cores compared to the 1× Single cache directory.

Coverage	Directory	Shared	Private	(Technology)	Area	Relative Area (%)
1×	Single	19,51	–		19,51	100,00%
	PS 1:3	6,33	9,50	(SRAM)	15,83	81,15%
	PS 1:7	3,28	11,08	(SRAM)	14,37	73,65%
	PS 1:3	6,33	8,22	(eDRAM)	14,56	74,61%
	PS 1:7	3,28	9,60	(eDRAM)	12,88	66,02%
0.5×	PS 1:3	3,28	4,80	(eDRAM)	8,09	41,47%
	PS 1:7	1,74	4,80	(eDRAM)	6,55	33,60%
0.25×	PS 1:3	1,74	3,01	(eDRAM)	4,76	24,39%
	PS 1:7	0,84	3,01	(eDRAM)	3,85	19,76%

TABLE 4.4: Static and dynamic energy consumption of the different PS configurations for 16 cores compared to the 1× Single cache directory.

Configurations		P leakage (mW)				E read (pJ)			
Coverage	Directory	Shared	Private	(Technology)	Total	Shared	Private	(Technology)	Total
1×	Single	4,2346	–		4,2346	0,0048	–		0,0048
	PS 1:3	1,1877	2,2572	(SRAM)	3,4450	0,0027	0,0028	(SRAM)	0,0055
	PS 1:7	0,6404	2,6334	(SRAM)	3,2739	0,0016	0,0032	(SRAM)	0,0049
	PS 1:3	1,1877	0,5123	(eDRAM)	1,7001	0,0027	0,0067	(eDRAM)	0,0094
	PS 1:7	0,6404	0,5977	(eDRAM)	1,2382	0,0016	0,0078	(eDRAM)	0,0094
0.5×	PS 1:3	0,6404	0,4114	(eDRAM)	1,0518	0,0016	0,0035	(eDRAM)	0,0052
	PS 1:7	0,3650	0,4799	(eDRAM)	0,8450	0,0010	0,0041	(eDRAM)	0,0052
0.25×	PS 1:3	0,3650	0,3276	(eDRAM)	0,6927	0,0010	0,0027	(eDRAM)	0,0037
	PS 1:7	0,2181	0,3822	(eDRAM)	0,6003	0,0007	0,0032	(eDRAM)	0,0039

Figure 4.10(a) shows the L1 MPKI classified in *3C*, *Coherence*, and *Coverage* (as Figure 4.6(a)) for different coverage ratios. As shown, with the only exception of a 0.125× coverage ratio, the proposal still incurs, on average, in less L1 cache misses than a single conventional directory cache allowing a significant reduction in directory cache area. For a 0.125× coverage ratio, the increase in the number of cache misses is roughly 20%, on average. This increase in coverage misses translates into a degradation in execution time with respect to the 1× coverage ratio PS-Directory. However, with respect to a single directory, the execution time is still shortened, even for a 0.125× coverage ratio, as shown in Figure 4.10(b). Therefore, if reducing silicon area is a target design goal, which would be the main reason for a lower coverage ratio, one can opt for reducing the area overhead of the directory without losing performance with respect to a conventional directory. The PS-Directory is able to improve the performance of a conventional single directory cache while using 8 times less entries.

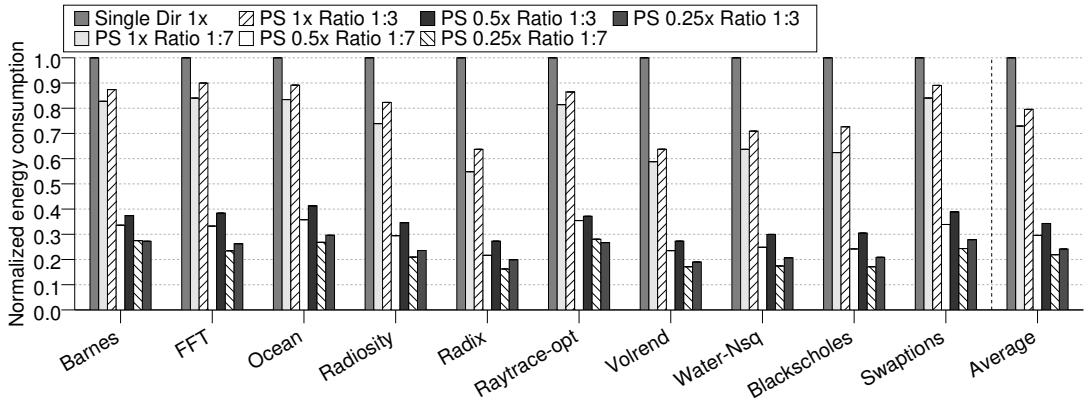


FIGURE 4.11: Normalized energy consumed by the directory with respect to a single-cache directory.

Table 4.3 shows the area required for different PS schemes with different coverage ratios[2] and the single directory cache. As expected, all the PS configurations are able to reduce area, even those with the same number of entries (1×) as the conventional directory cache. This is due to the fact that the PDC does not implement the sharer vector field. When the directory coverage ratio is reduced (i.e., 0.5× and 0.25× coverage ratios), area savings significantly increase up to $80,24\%$ for the 0.25× 1:7 configuration, while still improving the system performance (as shown previously). Comparing the results for both shared-to-private ratios, we can see that configurations with 1:7 ratio are more area efficient since they are able to reduce area from 12% up to 26% (depending on the directory coverage ratio) over configurations with 1:3 ratio, while providing similar performance results.

Table 4.4 shows the energy (dynamic and static) consumed by the PS-Directory cache with different coverage ratios and the 1× single directory cache. As observed, the 1× and 0.5× PS configurations consume more dynamic energy per access than the conventional cache, but this is highly offset by the much lower leakage consumed by the PS configurations, which is highly reduced even using SRAM technology in the Private cache. Leakage over the conventional cache is reduced from 19% (i.e. 3.4450/4.2346) for the SRAM 1× 1:3 configuration up to 86% in the eDRAM 0.25× 1:7 configuration. Comparing 1:3 and 1:7 shared-to-private ratios, the 1:7 configurations are able to reduce leakage consumption from 5% up to 15% with respect to the 1:3 configurations. Taking into account these values, Figure 4.11 shows the energy consumed during the execution

[2]Results for 0.125× are not shown because CACTI is not able to provide results for so small caches.

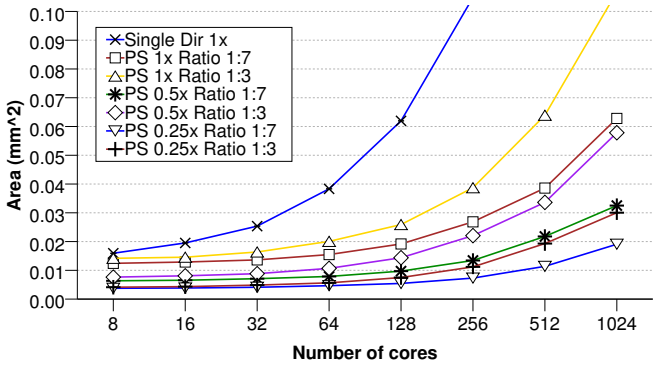


FIGURE 4.12: Scalability analysis in terms of area of the PS-Directory.

of the benchmarks by the PS-Directory normalized with respect to the energy consumption by the single-cache directory. Lower coverage ratios lead to less energy consumed at the cost of performance degradation.

Figure 4.12 depicts the area per core scalability for the studied directory configurations. As observed, the conventional directory cache exhibits the worst area behavior with significant area differences across the PS-Directory configurations. These differences increase with the number of cores. It requires even more area for 128 cores than all the PS configurations with up to 1024 cores, with the only exception of PS 1× 1:3.

As stated in the previous section for a 1× coverage configuration, the PS-Directory is able to reduce by $26,71\%$ (ratio 1:7) and $15,71\%$ (ratio 1:3) the area required by the conventional directory cache for a 1024–core system using both the same number of entries. Of course, the area is further reduced with smaller coverage ratios. In particular, for the 0.5× PS configurations, the PS-Directory requires only by $14,47\%$ (ratio 1:3) and $8,13\%$ (ratio 1:7) the area required by the single cache directory, and for the 0.25× PS configurations only $7,52\%$ (ratio 1:3) and $4,77\%$ (ratio 1:7) the area required by the single cache directory.

4.2 DWP-Directory

This section presents the second major contribution of this thesis. First, we present a characterization of the dynamic associativity requirements of the applications. Then, we introduce the proposed Directory Way Partitioning (DWP) architecture, and discuss

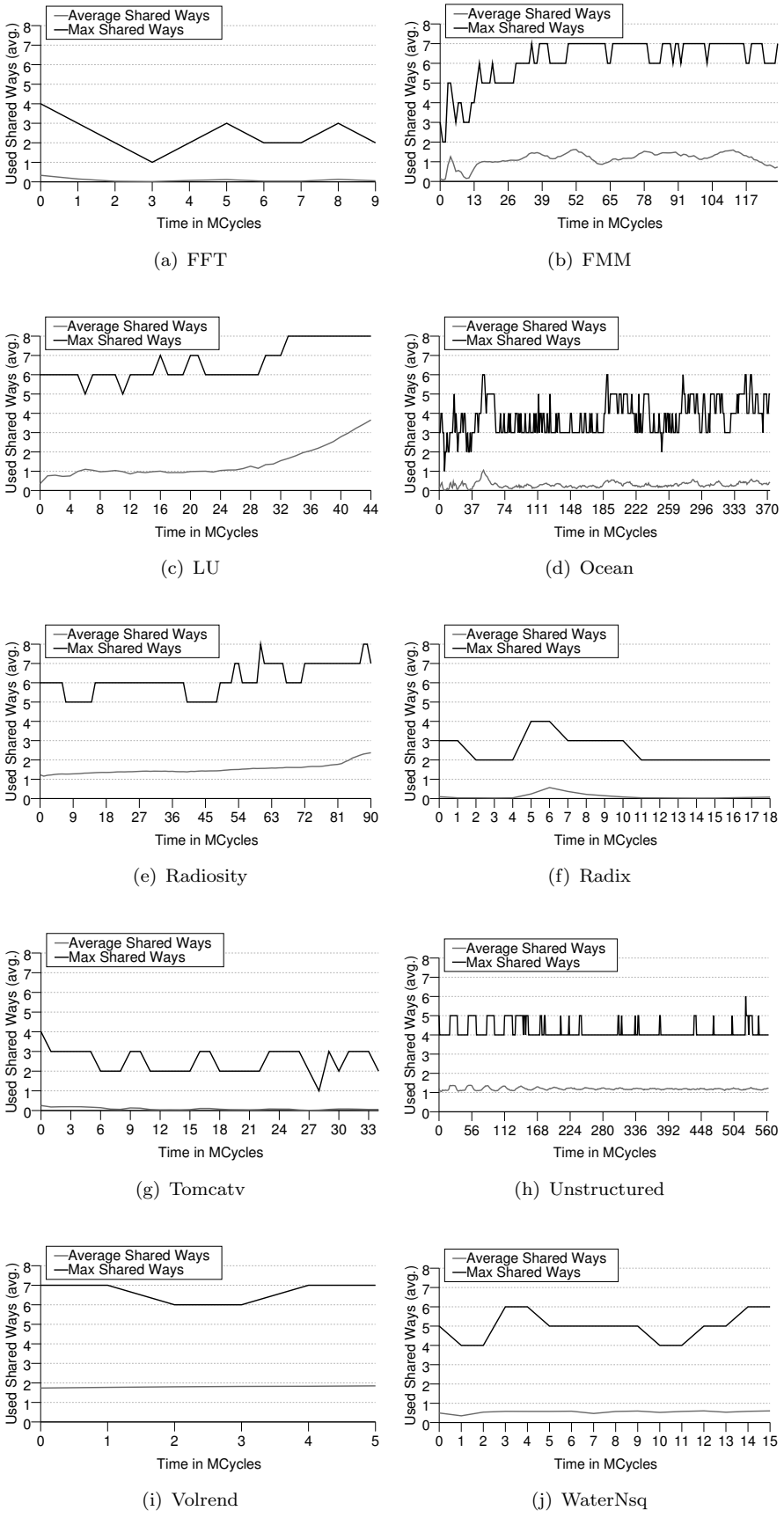
its basic behavior and the devised repartitioning approach. Finally, the proposal is evaluated against two state-of-the-art approaches.

4.2.1 Application Characterization

This section characterizes the applications used in our evaluation (Section 3.2) by studying the dynamic requirements of shared entries in the cache directory at run-time. The characterization study shows that while at some point in time some applications may require a single shared entry in a set, some others may require almost all the entries in a set to track shared blocks.

As a first design step, we analyze the dynamic requirements of shared directory entries across a representative subset of parallel workloads in order to find out how many shared entries should be supported to achieve the same performance as a conventional directory. As we support less shared entries, we can obtain more energy reductions. For this purpose, we ran parallel workloads and, for each of them, we measured the number of entries actually tracking shared blocks along the execution time. According to dynamic variability in the run-time demands of shared entries, there are some differences among applications, yet some general observations can be concluded. Figure 4.13 plots the dynamic evolution of the number of shared ways averaged across all the cache sets and directory banks, and the maximum number of shared entries in any set for each application assuming a 8-way directory cache.

It can be observed that, a static approach with $S = 2$ and $P = 6$ (S being the associativity given to shared entries and P the associativity given to private entries), which has been shown to be the best performing one in recent proposals [16, 33, 58, 59], fails to adequate to specific sets at a given point in time, since typically there is always one (i.e. labeled as `Max` in the plots) or some sets that require more than two ways for shared blocks. Yet, most of the applications have scarce set requirements, on average, to track shared blocks. Only `Radiosity` and `LU` require on average more associativity to track shared blocks than the deployed in the aforementioned proposals, but only during a small fraction of its execution time. This will inevitably lead to performance losses. Therefore, the solution to improve performance lies in adding extra shared entries. However, this way also would be at the cost of area and energy expenses, thus the key challenge lies in investigating the number of entries an efficient directory should deploy in order to



(a) FFT (b) FMM

(c) LU (d) Ocean

(e) Radiosity (f) Radix

(g) Tomcatv (h) Unstructured

(i) Volrend (j) WaterNsq

FIGURE 4.13: Average and maximum number of shared ways per set over the execution time across all the directory banks.

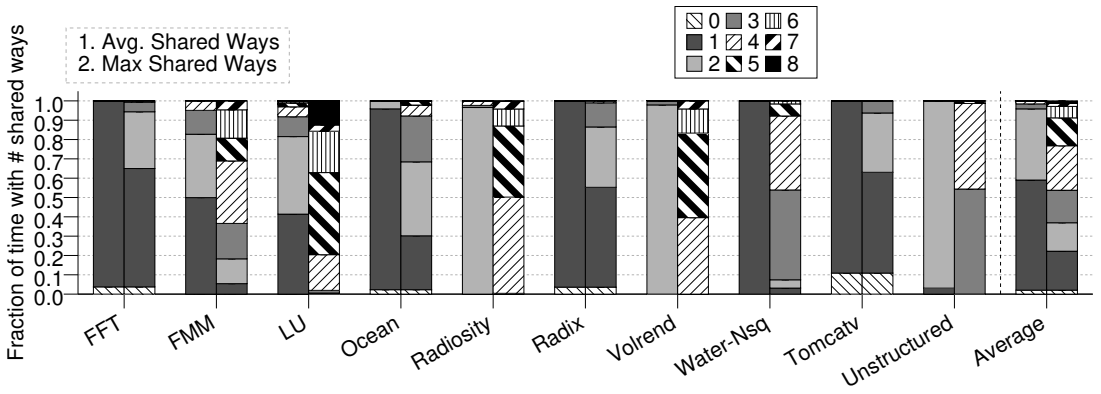


FIGURE 4.14: Fraction of time with # shared entries in a set.

achieve the best area and energy savings while sustaining the performance of a conventional all shared-entry directory (i.e. directories using sharer vector in all their entries). On the other hand, notice that there are also many other applications which do not need more than one shared way for most of its execution time (i.e. `FFT`, `Ocean`, `Radix`, `Tomcatv` and `Waternsq`). The additional shared associativity in the directory is not required in these cases, which in turn brings additional energy consumption and area that could be otherwise avoided.

To provide deeper insights in the most adequate number of shared ways, we quantified the fraction of time the directory is keeping track of any given number of shared blocks. Figure 4.14 shows the results across the studied benchmarks.

It can be seen that, on average, two or less directory cache ways able to keep track of shared blocks are required during 93.8% of the execution time, while only during a 3% of it more than four shared entries are in demand. Regarding maximum requirements in individual sets, it can be appreciated that, on average, during 76.8% of the execution time, there are no individual sets requiring more than four shared entries. This value makes sense since by definition, a shared block must be stored in at least two L1 caches, but since workloads are not ideally balanced, sometimes the accesses can concentrate on specific directory banks or sets. We experimentally found that this happens in some workloads like `Radiosity`.

The previous analysis, as well as experimental results, will confirm a directory with quarter or half its ways providing storage to track shared blocks is the most interesting

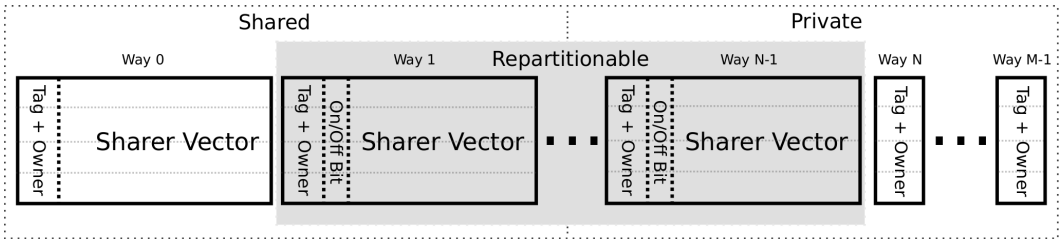


FIGURE 4.15: The DWP-Directory architecture.

design choices, that can provide the best trade off among performance, area, and energy.

4.2.2 DWP-Directory Architecture

The design of the DWP-Directory is mainly motivated by two observations discussed in Section 4.2.1. On the one hand, there are applications that need more than 3 or 4 shared ways during some phases of their execution, while there are some others that require nearly all the ways to track private blocks.

Keeping these observations into account, neither of them being supported for state-of-the-art approaches, the main goal of DWP-Directory is to provide support for both of them. Figure 4.15 depicts the structure of a generic DWP-Directory. Two types of entries are deployed: those having storage space to contain the sharer vector and those lacking the sharer vector. The directory deploys N shared entries and $M - N$ private entries per set, where M is the total associativity. Three areas can be appreciated: the most-left way is always shared, the $M - N$ most-right ways are always private, and the rest of ways in the middle can contain shared or private entries (i.e. repartitionable area, highlighted in dark). An entry in the repartitionable area include the `On/Off bit` that is set when the associated way is tracking shared blocks and reset when it tracks private blocks. When the bit is reset, the voltage supply to the sharer vector is removed since private blocks do not need it. Notice that this way allows energy savings, mainly leakage, with no performance penalty. In other words, with this design i) the private blocks do not consume the energy dissipated to hold the sharer vector, and ii) the directory size becomes smaller due to the removal of the sharer vector in part of its ways. An entry in a traditional directory under a MOESI protocol, apart from the tags, is comprised of the owner and a sharer vector field that requires $(log_2(C) + C)$ bits, being C the number of cores in the CMP. The higher the number of cores the larger the number bits that

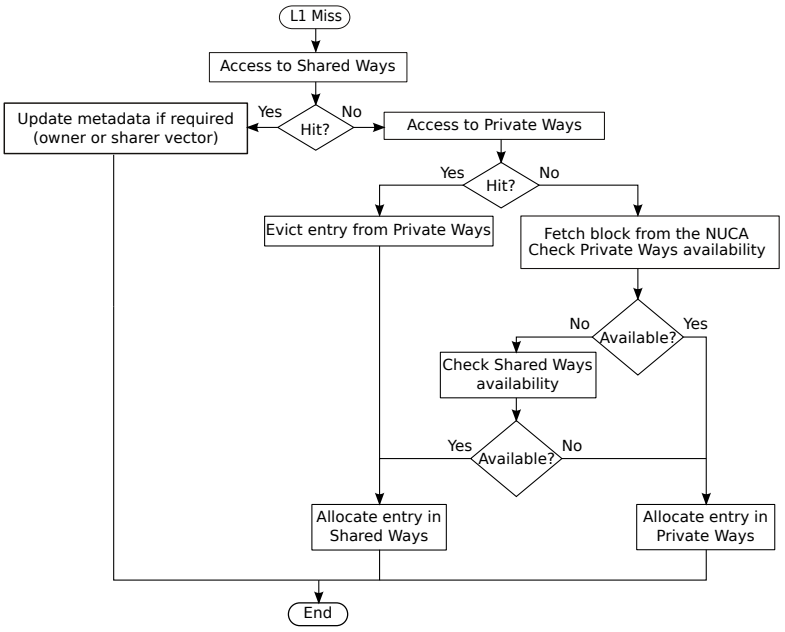


FIGURE 4.16: The DWP-Directory working flow chart.

can be saved with our proposal, i.e. $(M - N) \times C$ bits per set. To this amount, we should subtract a few N bits per set required for `On/Off bits`. The higher the number of cores the wider the sharer vector field since it requires one bit per core. Hence, in many-core systems the proposal would scale much better in both energy and area than traditional directory caches.

In summary, unlike existing approaches, which hardly limit the number of shared ways to 2 and private ways to 6, DWP-Directory implements a flexible sparse directory that can use all the ways to track private blocks, and it is able to track as many shared blocks as deployed sharer vectors.

4.2.3 Basic DWP-Directory Working Behavior

The DWP-Directory includes two types of entries: private and shared. Private entries are short, do not include the sharer vector, and are only able to keep track of private blocks. Shared entries are wider, implement the sharer vector, and can keep track of either shared and private blocks. Figure 4.16 depicts a flow chart that summarizes how DWP-Directory handles private and shared entries. On a miss in the L1 cache of a given core, the directory is accessed in order to maintain coherence. In a traditional directory, all the cache ways in the directory are accessed in parallel which translates into highly consuming searches.

To reduce dynamic energy consumption, the first lookup in the DWP-Directory only accesses the subset of ways tracking shared blocks. The reason to look up first these ways is that most of the accesses to the directory are to shared blocks [59]; thus, it is more likely to find the required block in the shared entries. Moreover, as discussed in Section 4.2.1, the number of active shared entries in the directory is, on average, lower or much lower than the number of private ways, so important energy savings can be achieved.

Upon a miss in the first lookup, the DWP-Directory searches the target block in the remaining entries, i.e. private entries. A hit in any of these ways means that the requesting core differs from the owner of the block, thus the block should become shared and the entry moves to a shared way. In case no shared entry is available, an entry should be evicted and all the copies of the block in the processor caches should be invalidated. Even though the DWP-Directory has potentially no limitation in the minimum number of shared ways, this dissertation does not evaluate the option of supporting no shared ways since the complexity of the coherence protocol increases. Notice that if there is no active shared way (i.e. all sharer vectors are deactivated), the previous owner of the block is invalidated and the new owner updated accordingly. New transitions are required in the protocol to take this case into account, while the DWP-Directory ensuring at least one shared way can work directly with the conventional coherence protocol. This case would be accounted as a shared entry eviction for the repartitioning algorithm as explained below.

If both directory lookups miss, a new entry is allocated. This entry is set as private since it only tracks a single copy. If there are free entries in the directory, an entry is selected, prioritizing private entries over shared entries. If all the entries are busy, the directory controller has to evict one of them. In such a case, the least recently used way, independently of being private or shared, is selected for eviction.

4.2.4 Repartitioning Approach

The DWP-Directory dynamically repartitions the number of shared entries enabled to keep track of shared and private blocks considering the run-time application needs. In other words, some of the shared entries are considered by design as private and their sharer vector field powered off for leakage savings. After a given number of accesses to

the directory, the DWP-Directory analyzes the eviction ratio between shared and private blocks and the number of private ways is readjusted taking into account the physical constraints.

The repartitioning mechanism is implemented with negligible hardware with only three main parameters. These parameters help the algorithm in decision taking about when a repartitioning should be triggered as a consequence of an increase or decrease of the demand of shared ways: an *interval length (IL)*, a *shared threshold (ST)*, and a *private threshold (PT)*. The selection of *IL* is quantified in number of accesses to the directory.

```
//For every access to the directory
directory_accesses++;
if (ctr != PT && ctr != ST) { //Ctr not saturated
  if (private_eviction_required) {
    ctr++;
  } else if (shared_eviction_required) {
    ctr--;
  }
}

if (directory_accesses == IL) {
  if (ctr == PT && shared_ways > 1) {
    private_ways++;
    shared_ways--;
  } else if (ctr == ST && shared_ways < N) {
    private_ways--;
    shared_ways++;
  }
  reset(); // Resets all counters
}
```

ALGORITHM 4.1: DWP repartitioning algorithm.

Algorithm 4.1 summarizes the pseudocode of the reconfiguration mechanism. This hardware algorithm acts on every directory access. Two global counters are used: *directory_accesses* and *ctr*. The former accounts for the number of accesses to the directory. The latter is an up/down counter that saturates at an upper threshold PT and at a lower threshold ST. Small top/down counters have a low implementation complexity and have been widely applied in the past, hence this design choice has been selected.

The algorithm works as follows. At the end of each interval of length *IL*, the repartitioning logic checks the value of the *ctr* counter to decide if the number of shared ways should be increased, decreased or remain in its actual value.

- The *ctr* counter is increased each time a private entry is evicted from the directory, and is decreased each time a shared entry is evicted.
- When the *directory_accesses* counter reaches IL:
  - If the counter saturates at its lower threshold *ST*, then additional shared entries are required. Thus, the most-left shared entry tracking a private block (Figure 4.15) is set as shared and its shared vector activated.
  - If the counter saturates at *PT*, then directory needs additional private ways in detriment of shared ones. In such a case, the most-right shared way in the *repartitionable area* (Figure 4.15) is set to private. Thus, its sharer vector is powered down and all sharers but the owner are sent an invalidation message.
  - If the counter is not saturated, then the system remains in its actual state for further *IL* accesses.
  - The counters *ctr* and *directory_accesses* are reset to 0.

This algorithm allows the proposal to dynamically adapt to the application phases, providing leakage savings without affecting performance. The reconfiguration of a way is done in all sets of the directory simultaneously in order to minimize complexity and to guarantee a very simple first lookup in the directory cache. Notice that the cost of evicting shared entries is higher than the cost of evicting private entries, but that will be taken into consideration when choosing the PT and ST thresholds values.

4.2.5 Experimental Evaluation

This section evaluates the DWP-Directory against a 4-way conventional or single-cache directory (which acts as the baseline), a 8-way conventional directory, and two state-of-the-art architectures: PS-Directory and Hybrid Representation [16].

Unlike the DWP-Directory proposal, the directory space assigned to each type of block in the aforementioned approaches is fixed and cannot be changed at run-time according to the needs of each particular workload during its execution.

TABLE 4.5: DWP-Directory System parameters

DWP-Directory Parameters	
Interval Length (IL)	500
Shared Threshold (ST)	10
Private Threshold (PT)	100

We evaluate both 16- and a 32-core CMPs configurations. Table 4.5 shows the specific DWP-Directory parameters used in the experiments.

All evaluated schemes, with the only exception of the baseline, implement a 8-way directory associativity. Both state-of-the-art architectures dedicate two ways to track shared blocks and the remaining ones to track private blocks (2:6 configuration). Since some workloads require a single shared way most of its execution time, as shown in the next section, a 1:7 configuration is also implemented for comparison purposes.

The DWP-Directory is sensitive to both directory and threshold parameters. Many configurations have been tested, however, as discussed in Section 4.2.1, only results for two of them are presented since experiments corroborate they as the most effective ones. One configuration implements half of its 8 ways without the sharer vector field, hereby referred to as *DWP-Directory (4:4)*, while the other implements the sharer vector in two of them, hereby referred to as *DWP-Directory (2:6)*. Both configurations share an *interval length (IL)* of 500 directory accesses, a *shared threshold (ST)* of 10 and a *private threshold (PT)* of 100. These thresholds were tuned to the studied workloads, showing minor differences for thresholds relatively high, thus no sensitivity analysis study is presented in this chapter.

4.2.5.1 Way Adaptation Analysis

The results of the dynamic adaptation of the proposal are shown in Figure 4.17. It depicts the average number of active shared ways for the studied DWP-Directory (2:6) and DWP-Directory (4:4) configurations. Each directory cache bank applies the repartitioning algorithm independently, hence the results are averaged across all the 16 banks of the CMP. For a significant number of applications both configurations present a similar behavior. This was expected, since most of the workloads have a higher demand for private entries. Thus, even though DWP-Directory 4:4 can have up to 4 active shared

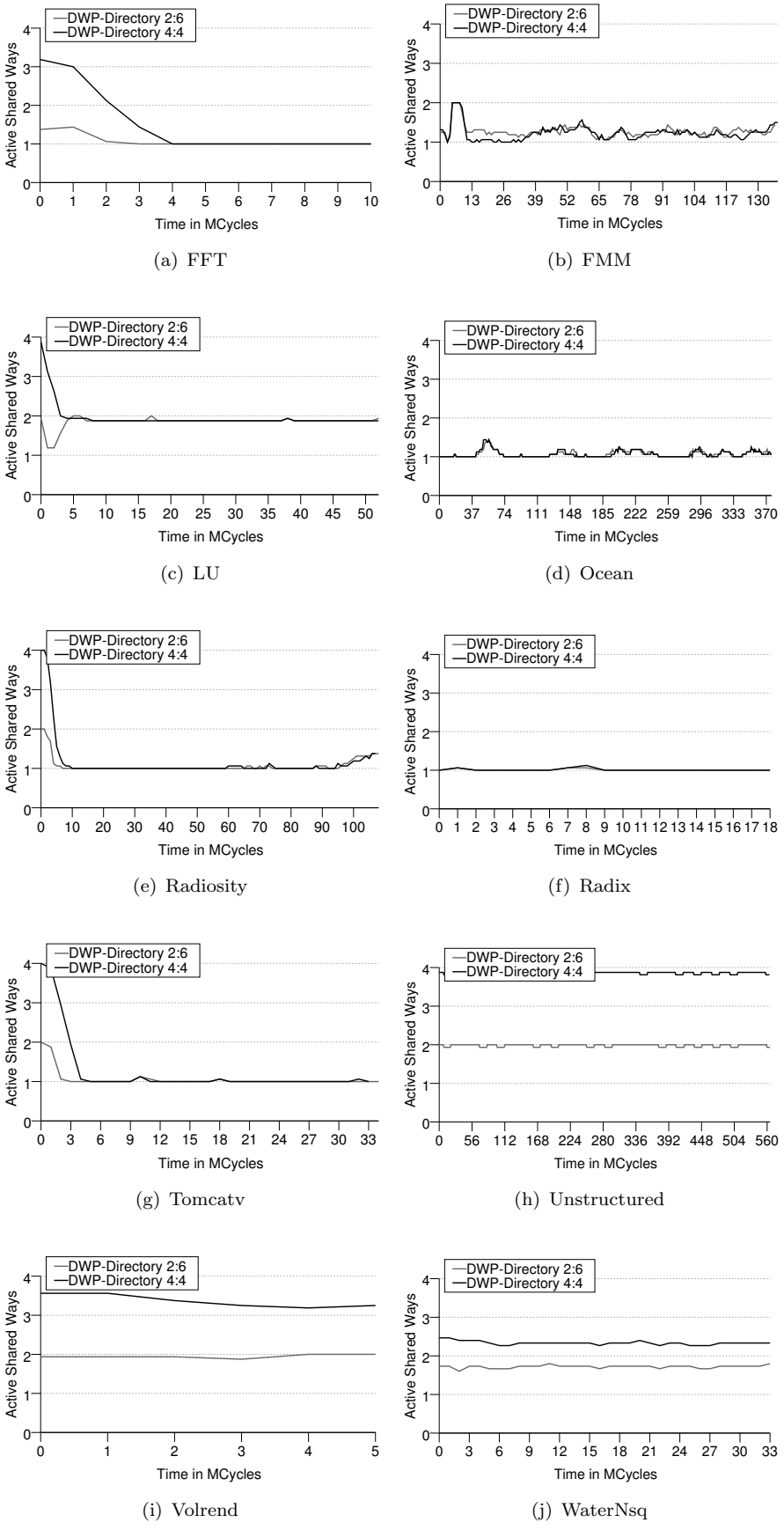


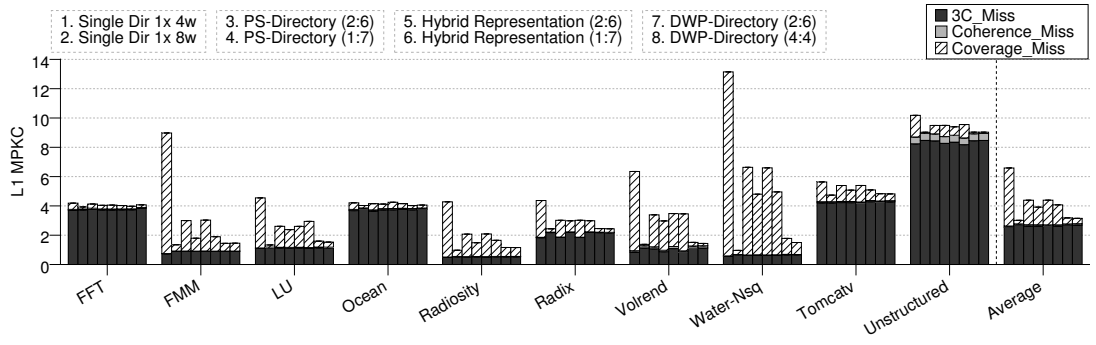
FIGURE 4.17: Average active number of shared ways across all tiles for DWP-Directory (2:6) and (4:4).

ways, it mostly varies between 1 and 2, just like the 2:6 configuration in 7 of 10 applications. An interesting observation is that most of the time just one shared way is enough, which means that static approaches with two shared ways are wasting non required energy budget. On the other hand, there are some exceptions where more shared ways are demanded, mainly in *Unstructured*, *Volrend*, and *WaterNsq*. In these workloads, the repartitioning algorithm would detect that 2 shared ways might be insufficient, however, static approaches would not be able to adequately meet these workloads' requirements.

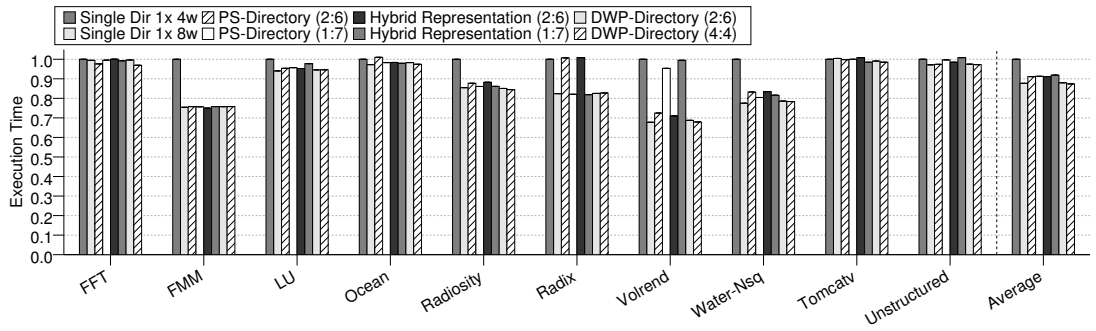
4.2.5.2 Impact of DWP-Directory on Performance

The impact of the DWP-Directory proposal on performance has been evaluated by analyzing the L1 Misses per kilocycles (MPKC) and the execution time and compared to the state-of-the-art schemes PS-Directory and Hybrid Representation. Every time a directory entry is evicted, invalidation messages are sent to those processor caches keeping a copy of the block and being tracked in order to be able to maintain cache coherence. These invalidations will cause coverage misses upon a subsequent memory request to those blocks, thus impacting on the final performance. Figure 4.18(a) shows the L1 MPKC across the compared schemes, which matches the number of directory accesses per kilocycles with respect to a 4-way single-cache directory in the studied 16-core CMP. The misses have been categorized in three types: 3C (capacity, compulsory and conflict), coherence and coverage as discussed in Section 2.1.3.4.

The evaluated schemes have negligible impact on 3C and coherence misses over the baseline. On the other hand, the aggregated associativity degree of the directory, as expected, has a big impact on the number of coverage misses. An increase from 4 to 8 ways in a single cache greatly decreases the number of coverage misses, approaching to the optimum performance that an ideal directory can achieve. The additional associativity allows more flexibility when keeping track of both shared and private entries in a set. Notice that even though most of the blocks are private and hence they whould require a higher number of entries, they are scarcely accessed in comparison to shared ones, so they can be prematurely evicted under an LRU replacement policy, when space constraints problems arise. Thus, additional associativity mitigates this problem.



(a) L1 Misses per kilocycles and per core.



(b) Execution Time.

FIGURE 4.18: Performance of the Single Directory, PS-Directory, DWP-Directory and Hybrid Representation, normalized with respect to a single-cache directory with 4 ways and 16 cores.

Regarding the state-of-the-art schemes, the PS-Directory reduces the number of misses by 34.5% and 40.6% for the 2:6 and 1:7 configurations, respectively. Hybrid Representation reduces this number by 34.3% and 38.2%. These reductions are achieved due to the different treatment of private and shared blocks. Since the associativity degree is partitioned, entries do not have the same allocation flexibility as a single-cache directory with the same associativity. Notice that the 1:7 configuration obtains the best results since, as discussed above, most of the applications present a low associativity requirement for shared entries. Yet, there are some exceptions in which the 2:6 configuration works best, *e.g.* in *LU* and *Unstructured* for the Hybrid Representation. Hence it can be seen that there is no optimal static configuration that satisfies every workload.

The DWP-Directory, which unlike the aforementioned schemes has the ability to adapt the private-shared partition size dynamically at run-time, obtains better results, reducing the number of misses by 49.8% and 50.4% in the 2:6 and 4:4 configurations,

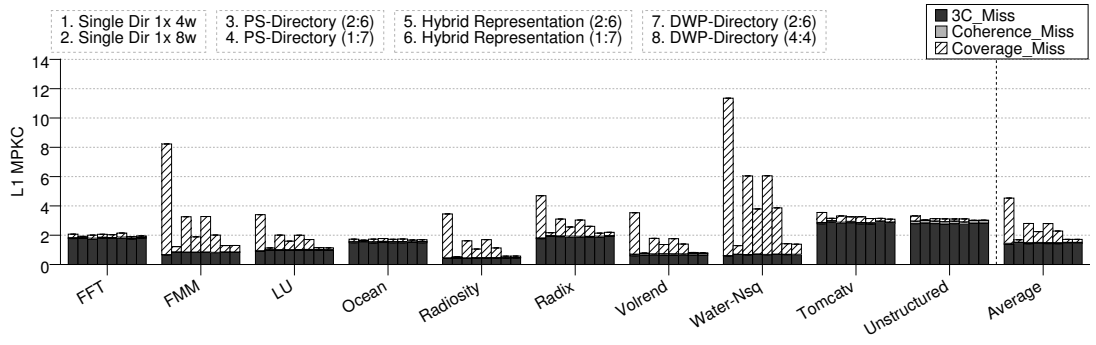
respectively. It performs similar as an 8-way single cache, with only a 1% degradation. Notice that following the characterization presented in Section 4.2.1, those applications with a higher maximum number of shared ways benefit the most our proposal, compared to the studied state-of-the-art schemes. On the other hand, applications with low shared requirements do not benefit as much. The dynamic adaptability of the proposal allows the directory a similar flexibility as the single-cache directory, while also keeping or improving most of the benefits of the studied state-of-the-art proposals in terms of area and energy reduction.

Reducing the number of L1 misses translates into a lower execution time of the applications, as shown in Figure 4.18(b). The reduction of misses achieved by the 8-way single-cache directory improves the execution time by 12.3%. The PS-Directory and Hybrid Representation both reduce the average application execution time by 8.9%. Meanwhile, the DWP-Directory reduces the execution time by 12.1% and 12.7% in the 2:6 and 4:4 configurations, respectively. As expected, applications with low MPKC values are the ones that have a lesser improvement in their execution time. Power-up and power-down delays of the proposal are taken into account in these results.

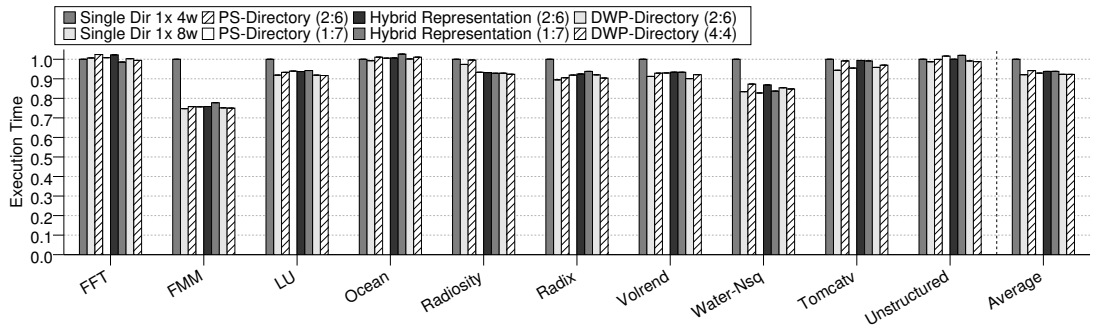
To explore how the proposal behaves on a higher number of cores, we launched experiments for a CMP with 32 cores. Figure 4.19(a) and Figure 4.19(b) show the L1 MPKC and the execution time, respectively. Results are similar as those presented for 16 cores. While the 8-way single cache reduces misses by 51%, the DWP-Directory 2:6 and 4:4 reduce them by 50.4% and 50.1%, respectively. The differences between our proposal and the 8-way single cache are smaller. In terms of execution time it translates into a reduction of 7.9%, 7.7%, and 7.7%, respectively. The state-of-the-art architectures achieve lower reductions but, as with 16 cores, a 1:7 shared-to-private way ratio performs on average slightly better than a 2:6 one.

4.2.5.3 Impact of the DWP-Directory on Energy Consumption

As mentioned above, static or leakage energy dominates the total energy consumption of the directory structure. Figure 4.20(a) shows the normalized leakage energy consumed by the directory structure with respect to the 4-way single cache.



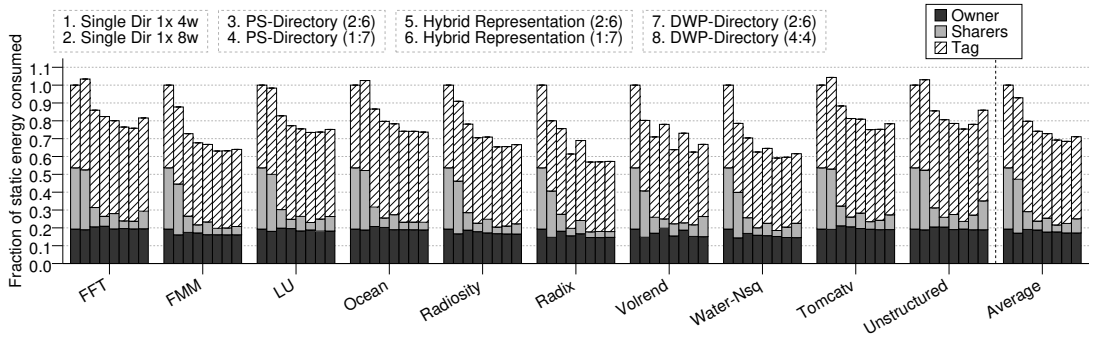
(a) L1 Misses per kilocycles and per core



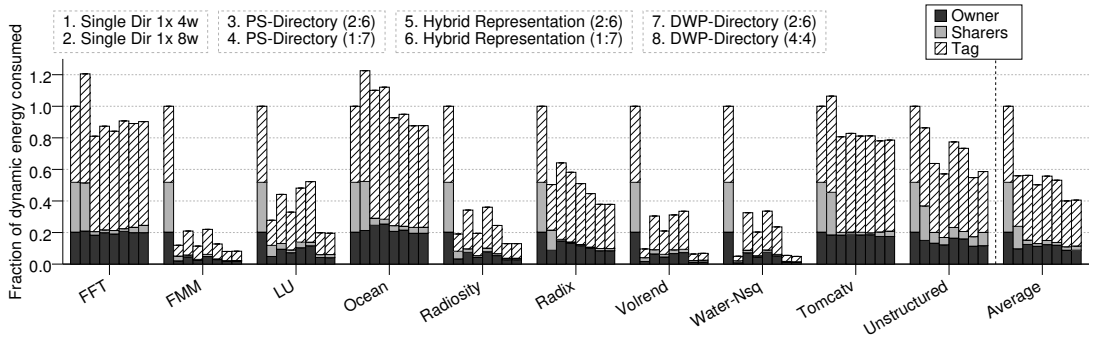
(b) Execution Time

FIGURE 4.19: Performance of the Single Directory, PS-Directory, DWP-Directory and Hybrid Representation, normalized with respect to a single-cache directory with 4 ways and 32 cores.

As can be seen, the 8-way single-cache directory reduces leakage by 7.1%, mainly due to the smaller execution time of the applications. The PS-Directory and the Hybrid Representation (2:6) achieve better energy savings by 20.3% and 27.2%, respectively, even though their execution time is slightly worse than the 8-way single-cache directory. These energy savings are the result of both schemes lacking the sharer vector field in some ways, namely those designated to keep track of private blocks, regardless of they are in a separate structure, like in the PS-Directory, or in the same set, as in Hybrid Representation. This allows the directories to consume less static energy, while the execution time of the application is not severely harmed as shown in the previous section. For this reason, configuration 1:7 consumes even less energy, since the sharer vector is present in one way less. The DWP-Directory reduces the static energy consumed by 31.5% and 28.9% for 2:6 and 4:4 configurations, respectively, which are the highest



(a) Static Energy



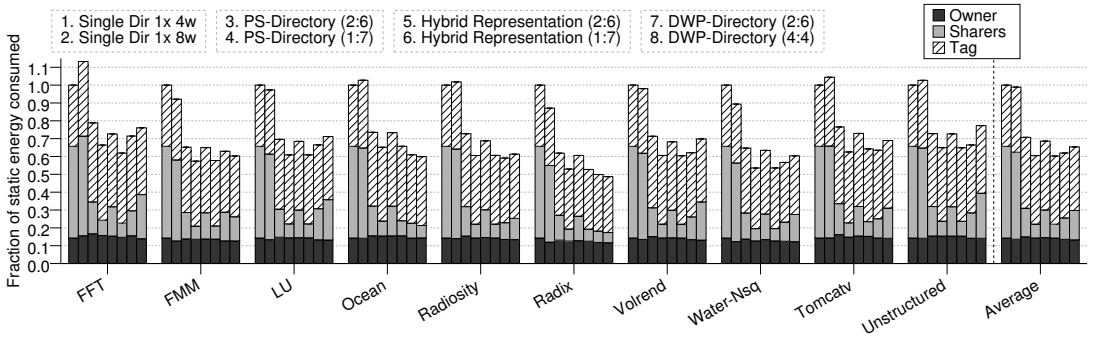
(b) Dynamic Energy

FIGURE 4.20: Normalized energy consumed of the Single Directory, PS-Directory, DWP-Directory and Hybrid Representation,with respect to a single-cache directory with 4 ways and 16 cores.

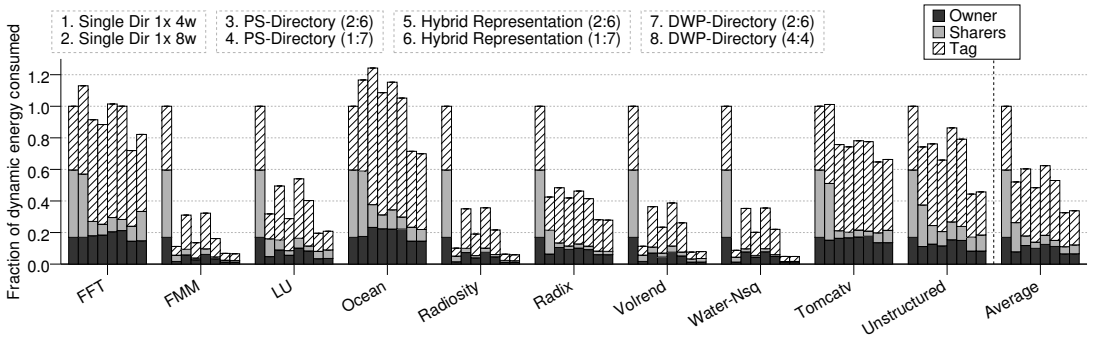
reductions of the evaluated directories. Notice that these leakage savings over state-of-the-art approaches come thanks to its repartitioning mechanism that allows DWP-Directory provisioning more shared ways when needed or even actually using none of them.

Results for dynamic energy are shown in Figure 4.20(b), also normalized with respect to the 4-way single cache. All the studied schemes, apart from DWP-Directory, achieve on average similar energy savings falling in between 44% and 50% over the baseline. The best scheme regarding this parameter greatly fluctuates across the applications, so there is no definitive best approach. Meanwhile, with the only exception of FFT, the DWP-Directory always achieves the best results. The consumption is reduced by 59.9% and 59.5% for the 2:6 and 4:4 configurations, respectively.

With 32 cores, in addition to maintain a performance gain similar as the one achieved



(a) Static Energy



(b) Dynamic Energy

FIGURE 4.21: Normalized energy consumed of the Single Directory, PS-Directory, DWP-Directory and Hybrid Representation, with respect to a single-cache directory with 4 ways and 32 cores.

in 16 cores, the proposal is able to achieve even better energy savings, offering a more scalable solution. Figure 4.21(a) and Figure 4.21(b) show the static and dynamic energy consumed in the 32 core CMP and normalized with respect to the 4-way single-cache. The leakage energy consumed by the 8-way single cache is only 1.1% better, despite the lower execution time. Meanwhile, the PS-Directory and Hybrid Representation 2:6 are able to reduce up to 29.3% and 31.3%, respectively, of this consumption. The energy savings are higher than those of the 16 core CMP mainly due to the larger amount of deployed sharer vectors. Since the mentioned schemes rely on the removal of the shared entry field, and this field increases its size with the number of cores, the overall number of bits that are eliminated is also higher. Lastly, the DWP-Directory is able to reduce up to 38% and 34.6% of the leakage energy consumed by the directory structure for the 2:6 and 4:4 configurations, respectively.

Regarding dynamic energy, the DWP-Directory is able to reduce up to 67.4% and 66.2%

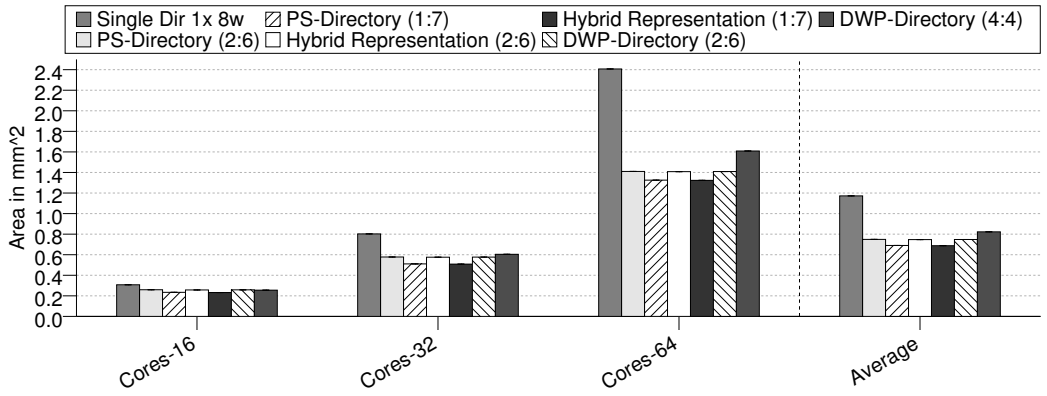


FIGURE 4.22: Area required for the different directories with an increasing number of cores.

for the 2:6 and 4:4 configurations, respectively, of the dissipated power, which is the highest across all the evaluated schemes.

4.2.5.4 Impact on Area Requirements

The on-chip area required to implement these directory structures is analyzed in this section. Results, obtained with CACTI, are shown in Figure 4.22 for the studied approaches. With a higher number of cores, the area requirement difference between the single cache and the proposal grows more and more. The DWP-Directory 4:4 requires only the 82.9%, 74.4%, and 66.8% area that a conventional single cache needs. The PS-Directory, Hybrid Representation and the DWP-Directory 2:6 scale similar to each other, and better than the 4:4 configuration, especially with 64 cores. This is mainly because the DWP-Directory 4:4 evaluated has a maximum of 4 shared ways, while the others only have 2. As results have shown, for a lower number of cores (i.e. 16 cores) 4 shared ways offer the best performance albeit with a small energy and area penalty with respect to a DWP-Directory with just 2 shared ways. Overall, DWP-Directory with a 2:6 configuration offers the best trade off among performance, energy, and area.

4.3 Summary

This Chapter has identified several key characteristics that clearly differentiate the behavior of private and shared blocks from the directory point of view. Based on these

observations, we have proposed the Private-Shared (PS) Directory, a directory cache that uses two different cache structures, each one tailored to one type of block (i.e., private or shared). The Shared Directory Cache, which tracks shared blocks is small, with low associativity and fast. The Private Directory Cache is aimed at tracking private blocks, which are highly dominant in current workloads. This structure does not store the sharer vector, is larger than the shared cache, and it is implemented with higher associativity.

This Chapter has also identified that the current needs of multithreaded applications, regarding shared and private data access from the directory point of view, varies dynamically along the execution time. Static private-shared structures are not able to properly adapt to this dynamic variation and, instead, on-demand based dynamic strategies are required. Based on these observations, we have introduced the Dynamic Way Partitioning (DWP) Directory, a sparse directory that sacrifices the sharer vector field from part of its ways in order to gain in both area and energy scalability. Furthermore, the implemented sharer vectors can be powered off or on according to whether the need of more shared ways dynamically rises or drops at run time, respectively.

Chapter 5

Filtering Techniques

In this chapter we propose two filtering techniques that can be applied to set-associative caches in the cache hierarchy system, namely the PS-Cache and the Tag Filter Architecture. These techniques reduce the number of tags and data entries checked when accessing a cache structure, which leads to reducing the dynamic power consumption. The main goal of the proposed approaches is to save dynamic energy in caches. For this purpose, our idea aims to discern which ways of a cache may contain the searched block, and save energy by only accessing those ways that may potentially contain the block.

The remainder of this chapter is organized as follows. Section 5.1 analyzes the memory access and motivates the necessity of designs that help reducing the consumption of highly accessed caches. Section 5.2 presents the proposed PS-Cache scheme and shows the experimental results obtained. Finally, Section 5.3 discusses a second proposed approach, the Tag Filter Architecture, and evaluates this proposal against other well-known schemes.

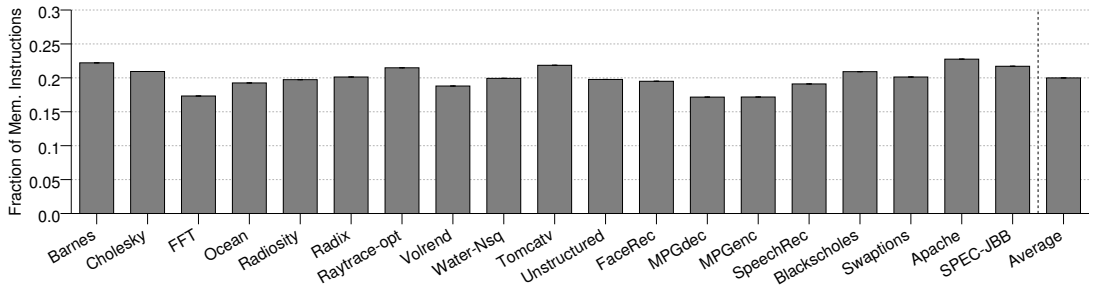


FIGURE 5.1: Fraction of memory instructions across the studied applications.

5.1 Analyzing the Cache Hierarchy Access

Memory reference instructions represent a significant percentage of the executed instructions, hence cache memories, specially L1 caches, are frequently accessed. We launched experiments to quantify the percentage of memory reference instructions in the studied workloads. Figure 5.1 shows the percentage of memory reference instructions in each individual benchmark executed on a 16-core CMP system. This value is roughly the same, around 20%, across the different benchmarks.

Therefore, a significant fraction of the total power budget is often consumed by on-chip caches. For example, in the Niagara2 processor [7], the 44% of the chip power is consumed by the L2 cache [60]. Reducing dynamic power consumption in caches of CMPs is an actual problem that is being under research [36, 37].

Also, when running multithreaded workloads, in addition to access the local cache, other caches (*e.g.*, remote caches) can be accessed for coherence purposes. Due to this fact, the number of accesses to the cache increases with respect to monolithic processors because of coherence requests issued by other cores. In other words, the cache is not only accessed from the processor side, but also from the interconnection network (NoC) side, therefore increasing the dynamic power consumption. In this context, the number of accesses coming from the NoC strongly depends on the type (snoop-based or directory-based) of the underlying coherence protocol.

As mentioned before, snoop-based protocols are based on broadcasting coherence requests to all the cores, which requires high bandwidth and energy consumption at the network but also at the caches, since all caches in the system are accessed on a coherence request. Thus, they are only appropriate for small system scales [40, 61, 62].

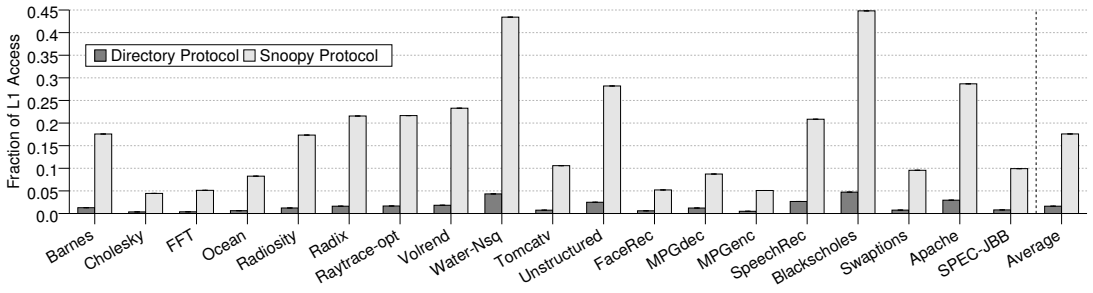


FIGURE 5.2: L1 Coherence lookups across the studied applications in both directory and snoopy protocols.

Directory-based protocols keep track of the various copies of cached blocks in a directory structure between the private and the shared cache levels [4, 8, 26]. This allows the processor to easily identify replicas of a block, so minimizing the coherence communication. Coherence requests are only sent to cores storing a replica, so only a subset of caches are looked up. This makes them more suitable for large-scale CMPs, since they reduce energy and bandwidth with respect to snoop-based protocols.

Figure 5.2 shows the fraction of cache accesses coming from the bus side in both types of protocols, snoopy and directory, in the studied 16-core system. As observed, this value is noticeable in snoopy protocols and it represents around one fifth of the total accesses; moreover, in some workloads this value is as high as 45%. In contrast, this value presents a scarce interest in directory-based protocols.

The previous discussion illustrates the importance of reducing dynamic power consumption in caches of CMP systems, and in snoop based protocols as well, where more coherence requests are issued by the cache controllers. To deal with this problem, this thesis proposes an architectural approach with the aim of taking advantage of different filtering mechanisms to reduce the number of ways accessed during each cache lookup.

5.2 PS-Cache

This section introduces and evaluates the proposed PS-Cache approach. The main goal of this approach is to take advantage of the classification of private (P) and shared (S) blocks to design a power-efficient cache architecture that is able to reduce the number of

ways looked up on each cache access. Instead of accessing all the ways in the corresponding set, as usually done, the PS-Cache only looks up a subset of them, in particular, those blocks whose type (private or shared) matches the requested block type.

The PS-Cache needs i) to keep blocks tagged as private or shared in the cache, and ii) a private-shared classification mechanism to indicate the type of the block to be looked-up. Blocks are tagged in the cache using a bit (the PS bit) attached to each cache line. This bit indicates the type of the block allocated to that line. In addition, although the PS-Cache can work with any private-shared classification mechanism to find out the type of the looked up block before accessing the data and tag arrays, this proposal assumes an OS-based private-shared mechanism similar to the one proposed by Cuesta *et al.* [4], which keeps the page information in a PS bit stored along with the TLB entry. Using such a coarse granularity presents its advantages and shortcomings. An interesting analysis about this fact can be found in [4]. In this way, the PS-Cache only accesses those ways whose PS bit value in the entry matches with the PS bit value given by the TLB for the looked up block.

5.2.1 The PS Page Classification Mechanism

The classification mechanism is based on OS support; therefore, the classification is performed at the page granularity. This means that all the blocks of the same page are classified with the same type. The sharing information is stored both in the page table and in the TLB that holds the translation for the most recently referenced pages. The sharing information comprises the PS bit and the *keeper* id, which is the first core that requested the page translation. This information is stored in the page table, and the TLB only stores the PS bit. On a memory reference, the core obtains the block type of the reference from the TLB when it is accessed with the purpose of getting the address translation. On a TLB miss, the page table is accessed (as usually done), but the devised classification mechanism also updates the sharing information in the page table and in the core TLB.

Figure 5.3 depicts an overview of how the classification mechanism works. The first miss (suffered by P0 in the example) sets the page status as private and the keeper field is set to P1. The page is set to private in the P0 TLB. On subsequent misses, if the page is found as private (which occurs in the second access from P1 in the example), it is

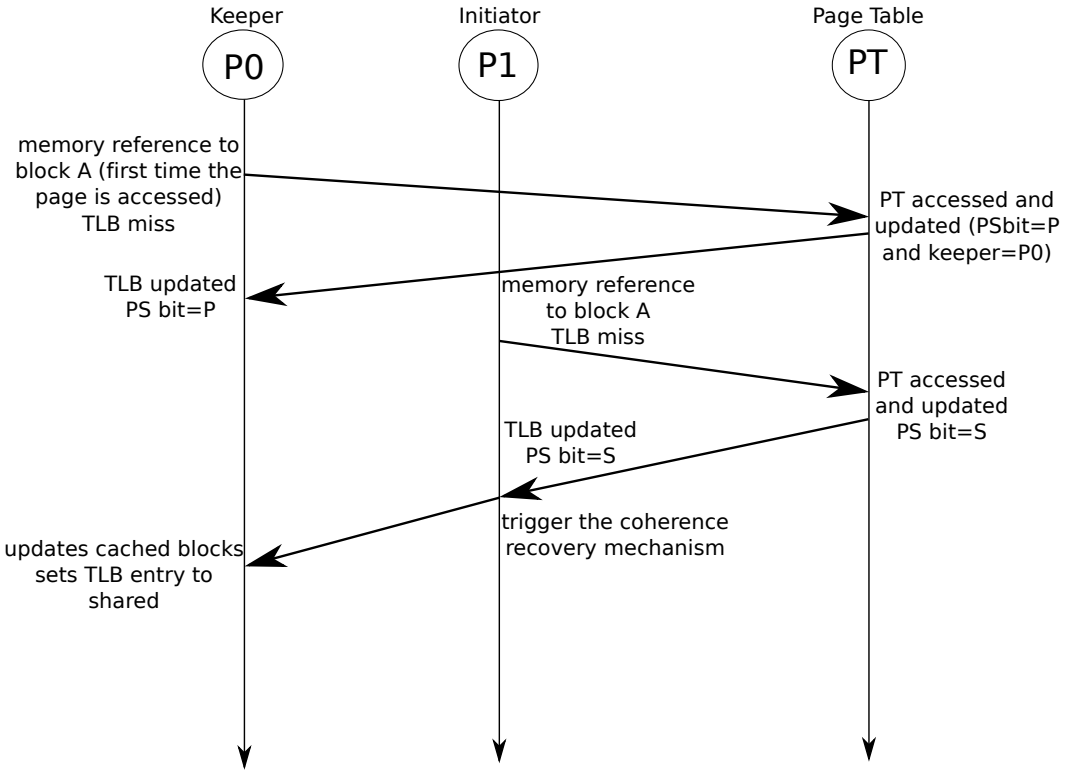


FIGURE 5.3: The PS Page Classification mechanism workflow. P0 and P1 are processors, and PT is the page table in main memory.

necessary to compare the keeper field (P0) with the core identifier requesting the access (P1). If the core identifier differs, then the page becomes shared. In order to update the page state, the page table (labeled as PT) entry is updated and a private-shared coherence recovery mechanism is triggered to maintain coherence between the page table and the keeper TLB and the PS bits in the caches.

The private-shared coherence recovery mechanism has to ensure that all the PS bits of the cached blocks of the page, as well as in the TLBs, keep the same type as their associated entry in the page table. For this purpose, the requesting core issues a *recovery request* to the page keeper (obtained from the page table entry). On the arrival of such a request, the keeper updates both the PS bit in the corresponding TLB entry and the PS bits of the cached blocks belonging to the given page. Notice that this recovery procedure is only required upon a Private-to-Shared transition. In this way the mechanism keeps the PS bit of every block in the cache coherent with the state of the page in the TLBs and in the page table. More details about the mechanism can be found in [4].

After solving the TLB miss, the sharing information is in the PS bit stored along with

the page translation in the TLB of the requesting core, and this PS bit is used by the core to check the type of the requested block (private or shared). As discussed above, the PS bit allows the mechanism to discern the group of ways in which the requested block can be found and, consequently, only these ways are accessed.

Although the private-shared classification employed in this chapter is performed by accessing the page table on every TLB miss [4, 41], the PS-Cache can also work along with a classification mechanism that employs TLB-to-TLB transfers [63], which can improve the overall performance of the system.

5.2.2 The PS-Cache Architecture

On the execution of a memory reference instruction, the cache controller first searches in the TLB to get the physical address[1]. As mentioned above, the TLB includes one bit per entry, the PS (Private-Shared) bit, that indicates how the page is classified. The value of this bit is read from the TLB entry jointly with the physical address. With this information, the cache controller proceeds searching the block in the cache. The TLB translation information is used to check the tags in the corresponding set, but as a novelty, the proposal also uses the PS-bit to avoid some of the ways to be accessed. The mechanism can be applied to any cache level, for illustrative purposes Figure 5.4 depicts an overview of the proposed cache architecture for the L1 cache.

The key difference is that in the PS-Cache only those ways matching the type indicated by the TLB are accessed, thus eliminating the energy consumption caused by looking up the other ways. As observed, each cache line has attached a PS (Private-Shared) bit which indicates the type of each block (according to the page table and the other TLBs). The PS bit provided by the TLB is compared with the PS bits of all the ways in the set. A simple logic is included to select the wordline (WL) of those ways whose PS bit matches the value of the ones obtained from the TLB for the page of the current memory reference. This means that, in the tag array, only a subset of the tags are read and then compared with the tag of the physical address and, in the data array, only a subset of data blocks are read. On a hit, the mux of the data array would select the proper data block from the ones accessed. This allows the proposal to reduce significant dynamic energy consumption across the memory accesses since in general, as

[1] If a TLB miss occurs, after solving the miss, the corresponding entry is in the TLB.

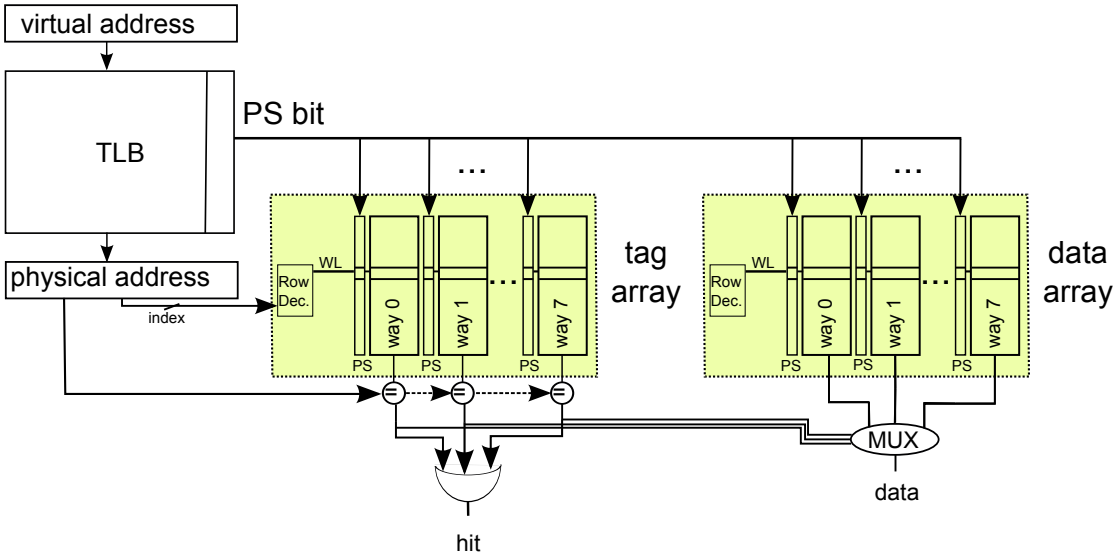


FIGURE 5.4: The PS-Cache architecture for L1 caches.

experimental results will show, only a small fraction of ways is required to be accessed in many memory operations.

The proposal also reduces dynamic energy consumption when accessing the cache from the NoC side (*i.e.*, coherence requests). In this case, only shared ways are looked up, since the classification mechanism ensures that before arriving the coherence request, the block is shared or it has been reclassified as shared by the private-shared updating mechanism.

In addition, to reduce the static power consumption, the proposed mechanism takes also advantage of the invalid bit. The power of all ways in invalid state is turned off and are also excluded from the process of looking the block up. This allows not only reducing the number of possible ways for the block (the lower the number, the less dynamic consumption), but also reducing the static energy consumption since power supply to these ways is cut off while they are in invalid state.

In case of accessing to L2 or L3 caches, the PS bit of the target block (already taken from the TLB) is carried in the miss request. On the other hand, PS bits in the cache entries are updated accordingly by the cache coherence protocol, so the PS bit of a request and the PS bit of the requested block are always coherent.

Regarding hardware complexity, the proposal requires minimal complexity. On the one hand, no extra information must be added to the TLB except a single bit (the PS bit) per

entry. Note that this bit can also be employed to optimize the cache coherence protocol as done in [4]. On the other hand, the proposal can be easily adapted to current caches. In fact, using a single wordline for all the ways in the set presents several problems due to, among others, many transistors are connected to the row wordlines and the column bitlines increasing the total capacitance, and thus, delay and power dissipation. As a consequence, to deal with this problem, current SRAM cache designs employ the divided wordline approach (DWL), which divides the wordline into a fixed number of blocks, for instance, one WL per cache way [64]. Notice, that our proposal takes benefit of this wordline scheme already working in current caches. Due to the low overhead of our scheme, the PS-Cache access time is not affected. In L2, the PS bit is known before accessing the cache and therefore does not affect the access time. Even regarding the first-level cache, the proposal could be integrated in most current deep pipelined processors because the access to first-level caches usually takes several stages; *e.g.*, the L1 hit time takes 3 cycles in the AMD Opteron X4 2356 (Barcelona) [65].

The previous discussion focused on typical physically tagged and physically indexed (PIPT) caches. However, the proposal could be also applied to other types of caches; for instance, virtually indexed but physically tagged (VIPT), like those of Intel processors. In these caches, the tag array is accessed in parallel with the TLB, and then the physical address is used to compare only those tags whose type matches the target one.

5.2.3 Experimental Evaluation

Two cache coherence protocols, a directory-based protocol and a snoop-based protocol, have been implemented and evaluated. Both protocols store the blocks in the private caches considering MOESI states, and implement a non-inclusive LLC (L2 in our study) cache. The directory protocol implements an on-chip directory cache, which increases its area overhead, while the snoopy protocol performs a broadcast on every write, and on every load in case the data is not found in the LLC. As analyzed in Section 5.1, snoopy protocols induce a higher number of coherence requests to the L1 caches, therefore a reduction in the average number of accessed ways results in higher energy savings than in directory based protocols. Energy results account for any access to the cache, including those that come as a consequence of the private-shared classification mechanism of page tables.

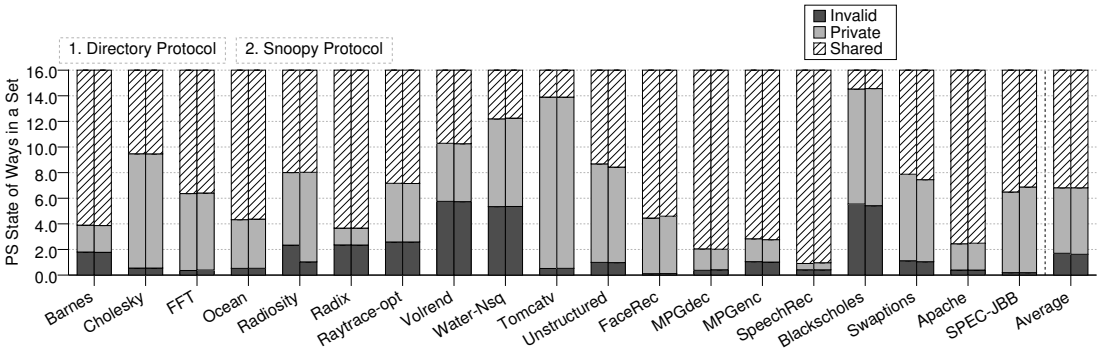


FIGURE 5.5: Average number of ways in a set of each type in the L2 cache for both studied protocols

Energy benefits of the proposal depend on the average number of ways that are looked up by the cache accesses. This chapter uses practically the same baseline system as in the previous chapter. However, since experimental results will strongly depend on the number of cache ways, we have used relatively highly set-associative caches (*i.e.* 8-way L1 caches, and a 16-way L2 caches) as also implemented in current processors (*e.g.* the IBM Power8 [66]).

5.2.3.1 Private-Shared Blocks Behavior Analysis

The number of accessed ways by our proposal changes mainly depending on which cache we are accessing to (L1 or L2), and on the type of block we are looking for.

The study starts with the L2 cache since it implements a higher number of ways, thus the proposal can potentially achieve higher energy savings in this cache.

Figure 5.5 depicts the average number of blocks of each type in the 16-way set associative L2 cache. Results are shown for the snoopy and directory MOESI-based protocols considered in the evaluation of this proposal. As observed, on average, there are around five private blocks in a set, whose access would result in important energy savings. Nevertheless, this number strongly depends on the application. There are some few applications with more than twelve private blocks per set on average (*e.g. tomcatv*), but as can be seen, most of the applications store shared blocks in most of the ways.

Figure 5.6 shows the average number of blocks of each type in the 8-way set associative L1 cache. Unlike L2 caches, the difference in the amount between private and shared

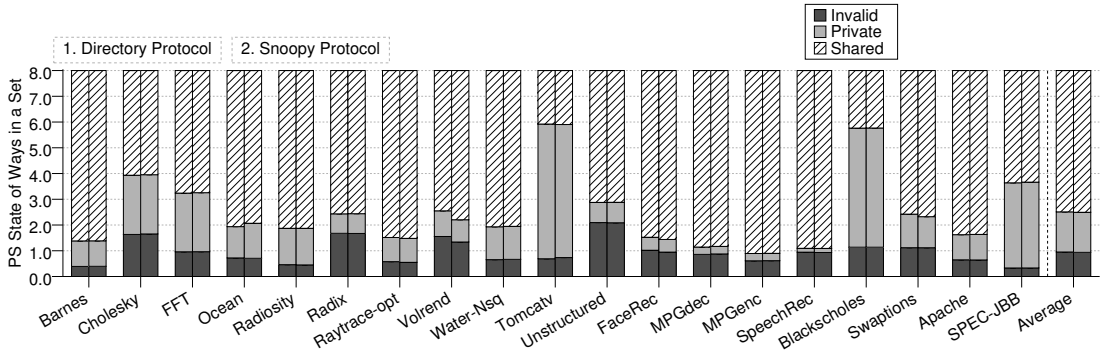
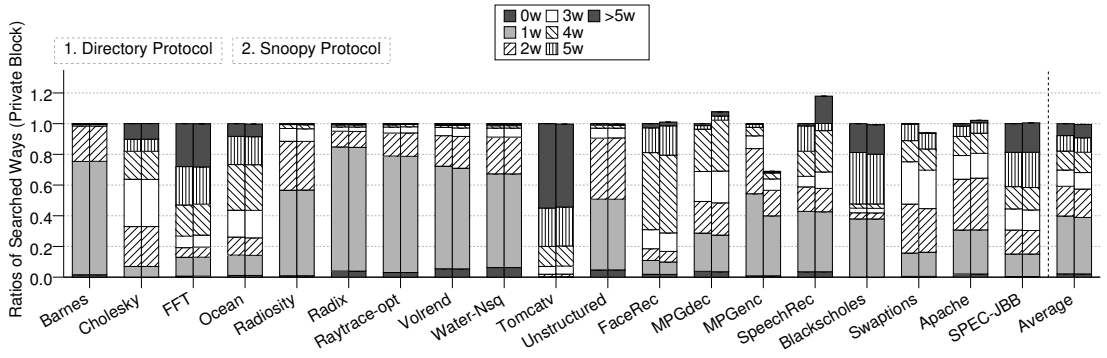


FIGURE 5.6: Average number of ways in a set of each type in the L1 cache for both studied protocols.

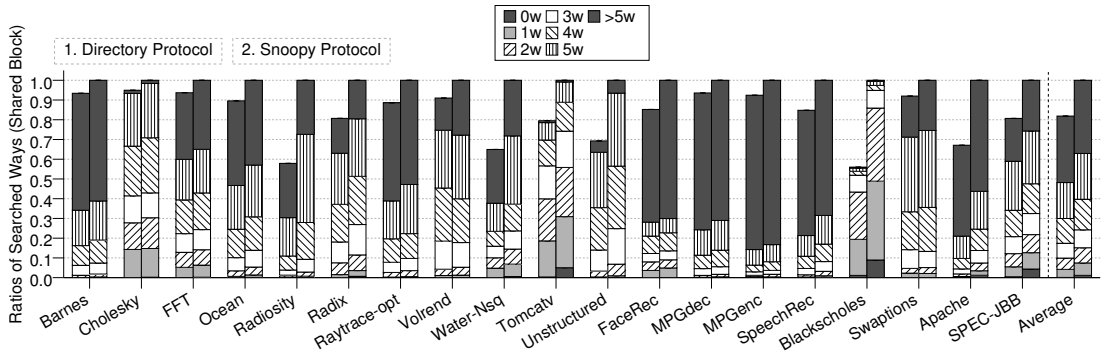
ways is higher, around two ways storing private blocks and five ways storing shared blocks.

Results confirm that the final number of accessed ways vary according to the type of block we are looking for and the application behavior. That is, the average number of blocks of each type seen on the arrival of a request to a private block can widely differ from that seen on the arrival of a request to a shared block.

To provide further insights on how much energy savings the proposal is able to bring, Figure 5.7(a) and Figure 5.7(b) show the distribution of the number of accessed ways on each access on the arrival of a private or shared request respectively in the L1 cache. The data in the first figure is normalized to the number of memory accesses when employing a directory protocol with PS-Cache, whereas the second figure is normalized to the number of memory accesses when employing a snoopy protocol with PS-Cache. As expected from Figure 5.6, most applications look up more ways when accessing shared blocks than when accessing private blocks, with only few exceptions such as *Tomcatv* and *Blackscholes*. An interesting observation is that when looking for a private block, most of the times (over 60% of the accesses) just one or two ways are looked up. Benefits, are lower when looking for a shared block, but even in this case, around 60% of times five or less ways are looked up, which will also bring important energy savings. Regarding the impact of the protocol, two main observations can be drawn. First, it can be appreciated that the rate of shared to private blocks accessed is quite similar in both protocols regardless of whether a shared or private block is requested. Second, major differences among protocols mostly appear when looking for a shared block, with



(a) Number of ways accessed when looking for a private block



(b) Number of ways accessed when looking for a shared block

FIGURE 5.7: Distribution of the number of ways accessed in the L1 cache normalized with respect to the snoopy protocol.

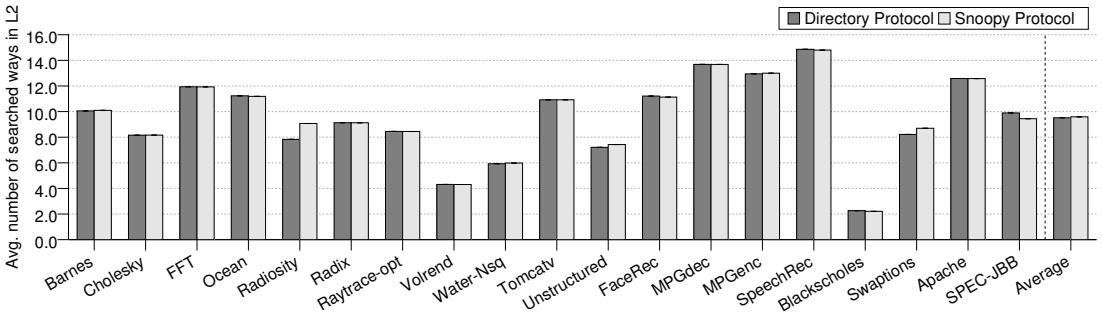


FIGURE 5.8: Average number of ways accessed in the L2 cache for the studied protocols.

the exception of *SpeechRec* when looking for a private block. In this case, a directory based protocol reduces the number of lookups on average around 20% with respect to the snoopy protocol. Moreover, this reduction can be as high as 70% in *Water-Nsq* when looking for a shared block.

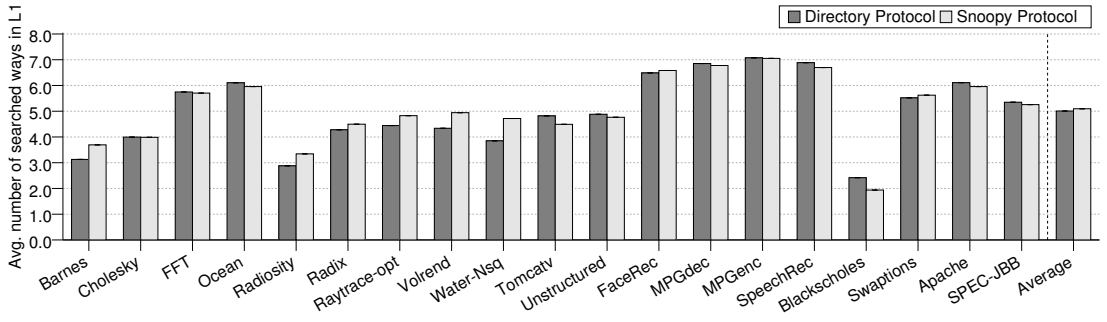


FIGURE 5.9: Average number of ways accessed in the L1 cache for the studied protocols.

Figure 5.8 shows the average number of accessed ways in the L2 per access. On average, a second-level cache implementing the PS-Cache architecture needs only to look 10 of its 16 ways up, although there are some cases (*i.e. BlackScholes*) in which this number can be as low as 2 ways per access.

As mentioned above, the proposal can be applied to any level of the cache hierarchy, thus this section also explores the benefits on the L1 cache in the studied system.

Figure 5.9 shows how many ways are looked up on average across all the benchmarks in the proposed 8-way first-level cache. Results show scarce differences between both types of coherence protocols, both of them accessing 5 ways on average. In some applications the PS-Cache greatly reduces the number of ways to be looked up (*e.g.*, only 2 in *Blackscholes*), while in others like *MPGenc* that presents a large number of shared ways, the impact is not so high.

5.2.3.2 Impact of PS-Cache on Energy Consumption

This section analyzes the impact of the proposal on the energy consumption of the caches.

Figure 5.10 shows the dynamic energy consumed by the L2 cache for the directory and a snoopy protocols considered in this dissertation. Conventional protocols and caches (labeled as baseline) have been included for comparison purposes. Results of the PS approach have been labeled with the name of the type of the protocol implemented (directory or snoopy) in the system. In this cache, there is not much difference between both coherence protocols. As observed, energy savings widely differ across benchmarks.

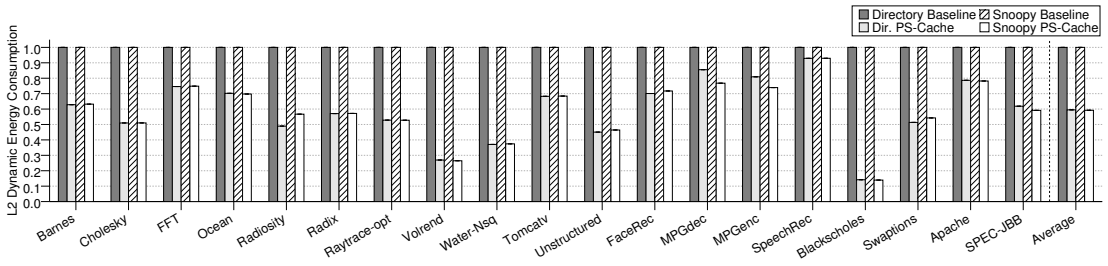


FIGURE 5.10: Reduction of the dynamic energy consumption in L2 across the studied protocols.

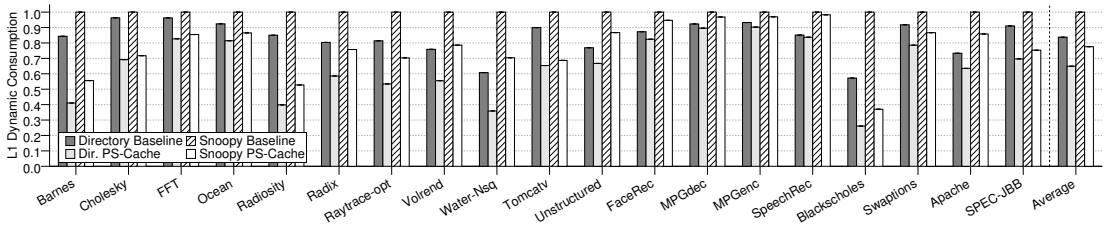


FIGURE 5.11: Reduction of the dynamic energy consumption in L1 across the studied protocols.

On average, the PS-Cache achieves by 40% of energy reduction in both coherence protocols. However, notice that in some cases, the overall energy consumption of the PS Cache is only by 12% that of the conventional system (*BlackScholes*), and even in the worst case, the benefits always exceed 8%.

Figure 5.11 shows the results for the L1 cache. On average, similar energy savings (*i.e.* by 22%) in percentage are brought by both snoopy and directory protocols in the L1 cache. An interesting remark is that a snoopy protocol with the PS-Cache architecture can consume less than a conventional directory. This is simply achieved through the selective look-up within the different ways of a set provided by the PS bit.

As suggested in the previous section, applications with a large number of private-block lookups, obtain higher energy reductions. For example, *Barnes* reduces dynamic power consumption by 44% and 51% for snoopy and directory protocols, respectively, and *Radiosity* by 47% and 53%, respectively. On the other hand, applications with a low number of private-block lookups offer no such benefits. Best example of this scenario is the *SpeechRec* benchmark, which only reduces the power consumption by 3%.

Results show that the dynamic energy consumption reduction is higher in L2 caches than in L1 caches, even more if we consider virtually-indexed physically-tagged L1 caches instead of physically-indexed physically-tagged ones. Hence, it can be concluded that the lower the cache level the higher the benefits provided by this technique.

5.3 Tag-Filter Architecture

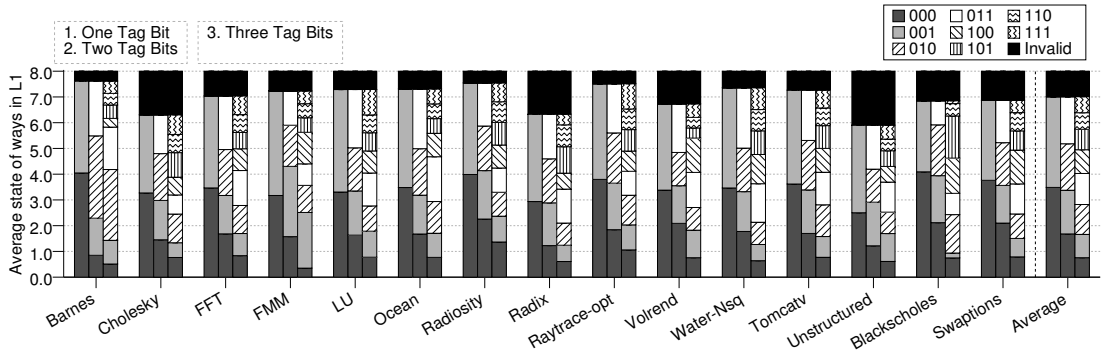
This section presents an analysis of the last tag bits distribution in caches and the Tag Filter (TF) Architecture, and introduces the the second approach proposed in this dissertation to reduce energy consumption in the processor caches. This scheme can be applied to any set-associative cache in a CMP, such as processor or directory caches.

5.3.1 Last Tag Bits Distribution

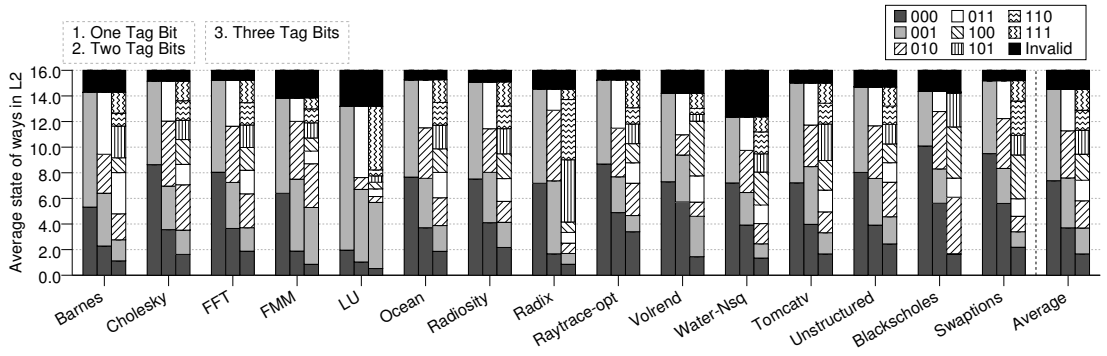
As mentioned in Section 5.1, a significant fraction of the total power budget is often consumed by on-chip caches. To deal with this problem, this work proposes an architectural approach based on the hypothesis of the homogeneous distribution of the least significant bits of the address tag across the ways of a set-associative cache [67].

We launched experiments to verify this hypothesis in the considered experimental scenario (see Chapter 3 for further details). Figure 5.12 shows the average distribution of the blocks across an 8-way L1 cache and a 16-way L2 cache on a 16-core CMP system. In Figures 5.12(a) and 5.12(b) on average there are 1 and 2 ways in *invalid state*, under the implemented MOESI protocol, in the L1 and L2 cache, respectively. Meanwhile, the remaining ways share a quite homogeneous distribution considering the lowest order tag bits of the allocated blocks. Therefore, there would be no need to access all the ways in a set if there were some mechanism able to filter accesses for a subset of ways that might potentially allocate the requested block. The homogeneous distribution of the lowest order tag bits makes our approach a perfect method for filtering accesses.

Directory caches are typically built as set-associative caches and, as experimental results will show, the least significant bits in the address tag follow a similar homogeneous distribution. A low associativity of the directory caches causes frequent evictions of directory entries that lead to extra coverage misses in the processor caches, thus as in



(a) Average number in the L1 cache.



(b) Average number in the L2 cache.

FIGURE 5.12: Average number of ways in a set of each type in the cache hierarchy varying the least significant bits.

any kind of cache, a higher associativity would improve the performance. However, their associativity is also limited due to power constraints.

5.3.2 TF-Architecture Scheme

The aim of our proposal is to achieve dynamic energy savings by filtering the number of accessed ways in the target set on each cache access. The terms Tag Filter Cache (TF-Cache) and Tag Filter Directory (TF-Directory) are used to refer to the Tag Filter Architecture when it is applied to a processor cache and to a directory cache, respectively.

The TF-Cache approach reduces the number of tags that are compared on each cache access and also the number of ways that are accessed in parallel in the data array. The final aim is to reduce dynamic power consumption in these cache components, which represents a large percentage of the total system power consumption. In a typical cache

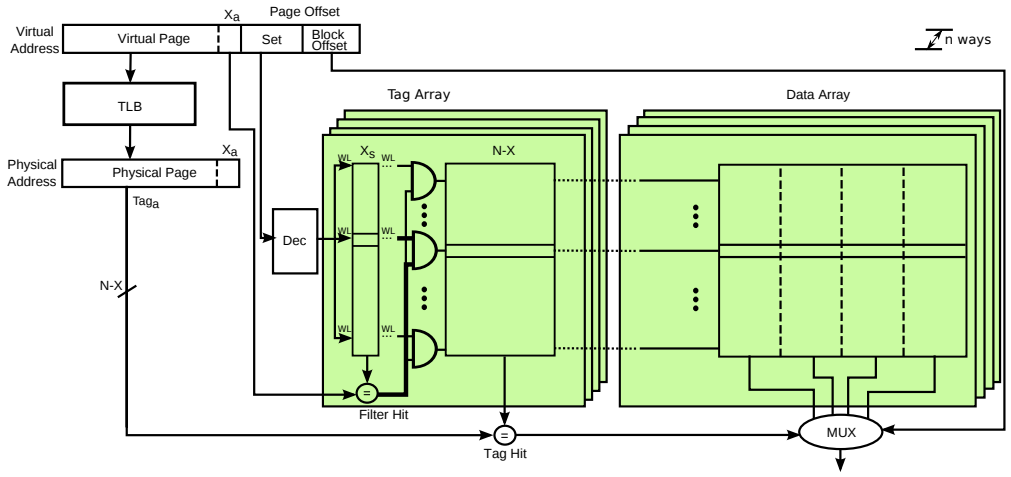


FIGURE 5.13: The TF Architecture for L1 caches.

access, to check if the searched block is in the cache, the entire tag in all the ways of the target set are compared with the one of the searched block. Although the comparison is done in all the ways, only one of them can potentially result in a hit. In first level caches, where performance is a key objective, the data array is looked up in parallel with the tag array, before knowing whether or not the target block is stored in the set.

For energy saving purposes, the proposed approach applies first, a small and fast filter to discard some of the accessed ways (both in the tag array and in the data array). Those cache ways that mismatch the *small tag* comparison are not accessed. The cache memory is upgraded with minimal hardware complexity as follows. The tag array is decoupled in two main structures: one X-bit-wide and the other one $N - X$ bits wide. The TF-Cache employs the least significant bits of the tags stored in a X-bit-wide table to reduce the number of accessed ways; that is, the entries in this table act as a filter to access the tag and the data arrays as explained below. Figure 5.13 depicts a block diagram of the TF-Cache for a first-level VIPT cache.

To allow the mechanism to work in current VIPT caches, the first comparison must start before the TLB output is known. For this purpose, we assume that the operating system (OS) is responsible to ensure that the X least significant bits of the virtual address are the same as those of the physical address. This assumption is reasonable since i) a uniform page address distribution is expected and ii) main memory capacities are by four orders of magnitude bigger than page sizes (*e.g.* a 32GB main memory [68] and a

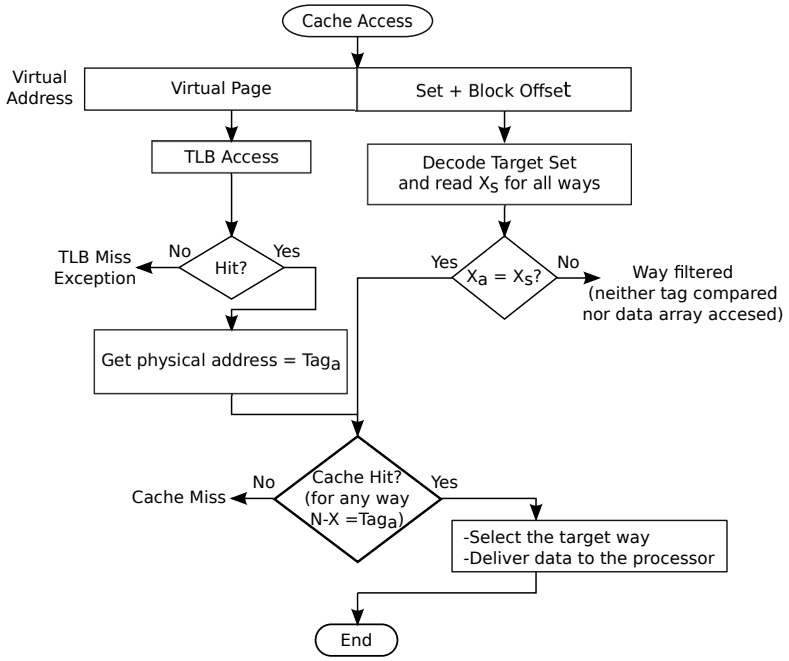


FIGURE 5.14: The TF Architecture working flow for L1 caches.

4KB page size), which allows the OS to have some allocation flexibility, and so, being able to tolerate this small restriction.

Unlike a typical cache that performs a single N-bit tag comparison for each way, the proposed mechanism performs two tag comparisons, one of X bits for each way in the set and the other of $N - X$ bits only for the ways that match the first comparison, as illustrated in the flowchart in Figure 5.14 and shown in the block diagram of Figure 5.13. Under the aforementioned assumption, the first comparison can be done once the output of the set decoder is provided, while the address translation in the TLB is being performed. In this step, only the X least significant bits of the virtual page (namely X_a in Figure 5.13) are looked up in all the ways of the target set (namely X_s). The few number of bits (we evaluated from 1 to 4) used in this comparison, allows it to be fast and effective (as experimental results will show), thus introducing negligible time penalty and important energy savings.

In case the first comparison fails in all the cache ways, the L2 cache is accessed. Otherwise, two main actions are performed in parallel. On the one hand, once the TLB provides the $N - X$ bits of the physical address (*i.e.* Tag_a), the remaining $N - X$ bits of the tag array are compared to those provided by the TLB. Notice, that this comparison involves a much larger number of bits, however, it is only performed in those

cache ways that succeed the first comparison. Similarly, those entries of the data array corresponding to those ways that matched the first tag comparison are accessed.

From a complexity perspective, the proposal requires minimal hardware complexity to be applied to current caches: an additional AND gate per set and way, plus an additional X-bit small comparator per way (the one shown in the box representing the tag array in Figure 5.13). For instance, in the evaluated caches, a simple circuitry consisting of 1K AND gates and 8 X-bit comparators, where X varies from 1 to 4 depending on the evaluated configuration. As mentioned above, the tag array is decoupled in two independent structures. Simple logic is required to drive the wordline (WL) signal to both the $N - X$ tag structure and the data array. As observed in Figure 5.13, the wordline is allowed to drive both the wide tag structure and the data array for a given way, but only in case the first comparison in that way succeeds. Notice that the AND gates do not remove the power supply since this would not preserve the data contents.

Regarding the TF-Directory, on a directory access the scheme works very close to the TF-Cache. The tag array is decoupled in two, a X-bit and a $N - X$ bits wide, tables and the access is split in two sequential steps. In this manner, the tags whose least significant bits do not match with the ones of the searched block are filtered in a similar way as explained in the L1 data cache. This design allows the mechanism to significantly reduce dynamic energy consumption across the cache accesses since, in general, only a small fraction of ways is compared in most of the accesses.

Finally, we would like to remark that the benefits of our proposal mainly vary with the cache associativity. The higher the associativity the larger the energy savings achieved by the TF Architecture. However, since our focus is on dynamic energy, varying the number of sets or the cache size would have a minimal impact on performance other than changing the capacity misses so incurring in less or more cache accesses. Anyway, it is expected that varying the number of accesses would maintain approximately the same percentage of filtered ways hence achieving comparable energy savings. A similar rationale could be applied for an increasing core count.

5.3.3 Experimental Evaluation

This section briefly describes the schemes that are considered for comparison purposes against the TF Architecture. Then, experimental results are presented and analyzed for both processor caches and directory caches.

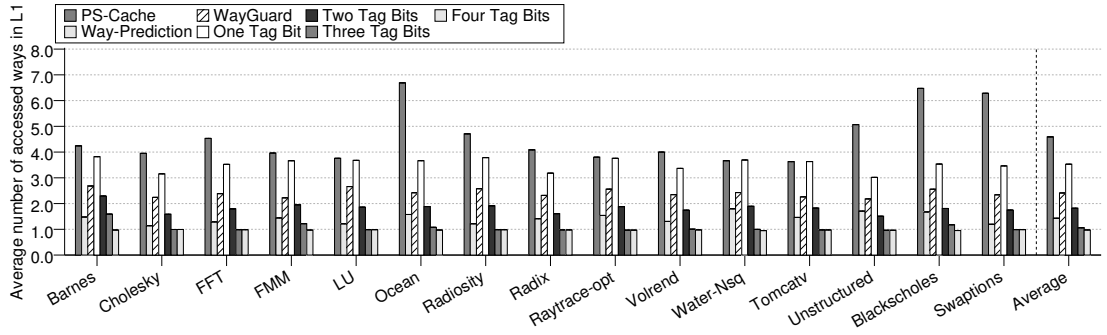
As in the previous proposal, the evaluation will consider highly associative processor caches (*i.e.* 8-way L1 caches, and 16-way L2 caches), as well as directory caches with a higher number of ways (*i.e.* 8 ways).

5.3.3.1 Compared Schemes

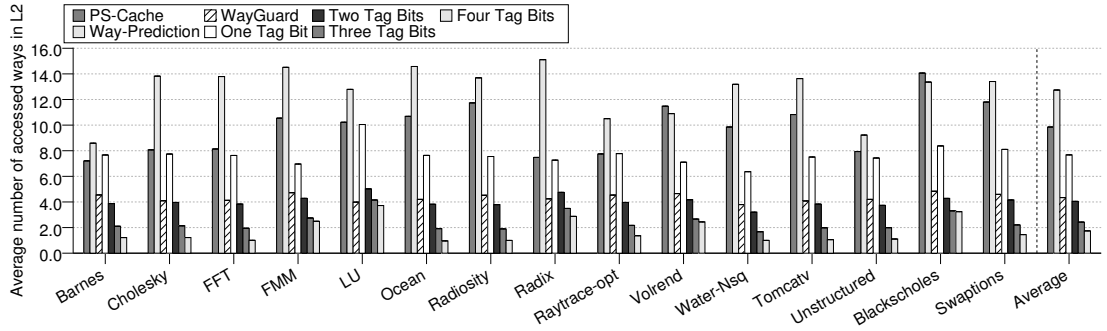
First, the TF-Cache is evaluated and compared to other state-of-the-art proposals that also reduce dynamic energy consumption by accessing a subset of the cache ways instead of all of them. These schemes are Way Prediction [15, 35] and two recent approaches: Way Guard [14] and PS-Cache, which was presented in Section 5.2.

Way Prediction techniques [15, 35] predict the way that is likely to keep the target data in advance, typically the way containing the MRU block, and only that way is accessed first. The problem rises when the prediction fails; in such a case, after performing the comparison of the MRU tag, all the remaining ways are accessed at a second phase to look up the target block. This means that on mis-prediction, energy wasting rises and latency increases, since additional cycles are required to solve the memory request.

Way Guard [14] has been proven to work efficiently in highly-associative caches. The mechanism implements a counting bloom filter associated to each cache way. Way Guard works as follows. First, a hash function is applied to a subset of bits of the address of the target block. The output of the hash is a m-bit index that is decoded to access the $2^m - 1$ entry bloom filter vector. If the bit is set to 1 then the associated cache way is accessed (both tags and data arrays), otherwise that way is not searched. Each entry of the bloom filter has associated an up/down counter (*e.g.* a 3-bit counter in the original work), that is decremented each time a cache line whose address maps to that position is evicted from the cache, and increased when the block is written to the cache. In the original paper, results are shown for m equal to $4\times$ the number of blocks in a cache. We will refer to this configuration as $WayGuard - 4\times$. This approach requires a decoder



(a) Average number of ways accessed in the 8-way L1 cache.



(b) Average number of ways accessed in the 16-way L2 cache.

FIGURE 5.15: Average number of ways accessed in the cache hierarchy in the studied schemes.

with 4× more outputs than the already implemented in the cache to index the target set.

To evaluate the TF-Directory we implemented the Tag Filter Architecture in two directory schemes: in a conventional single-level directory cache [27] and in the PS-Directory described in Section 4.1.

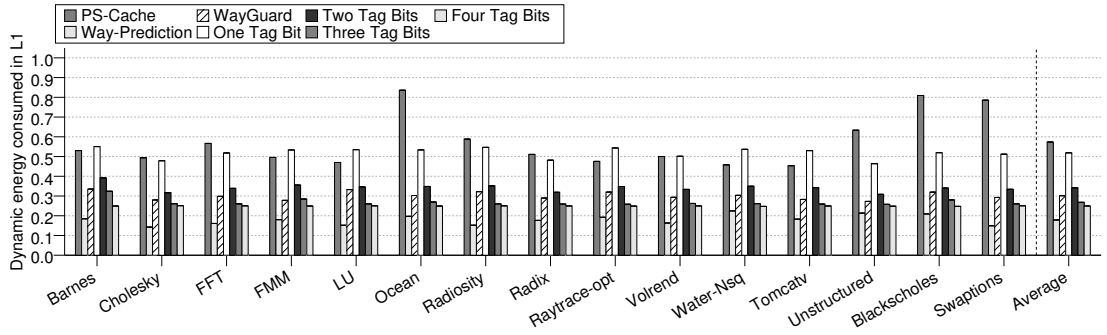
5.3.3.2 TF Architecture in Processor Caches

Figure 5.15(a) shows the average number of searched ways in the 8-way L1 cache across the studied techniques. The more bits (from 1 to 4) are used in the bit-array for filtering the ways, the less ways are accessed. On average, the number of accessed ways in a 8-way cache for 1-, 2-, 3-, and 4-bit tag filter is 3.53, 1.82, 1.06, and 0.98, respectively. This means that the accesses follow a uniform distribution when considering the least significant bits. Consequently, using three bits suffices to limit the number of ways that

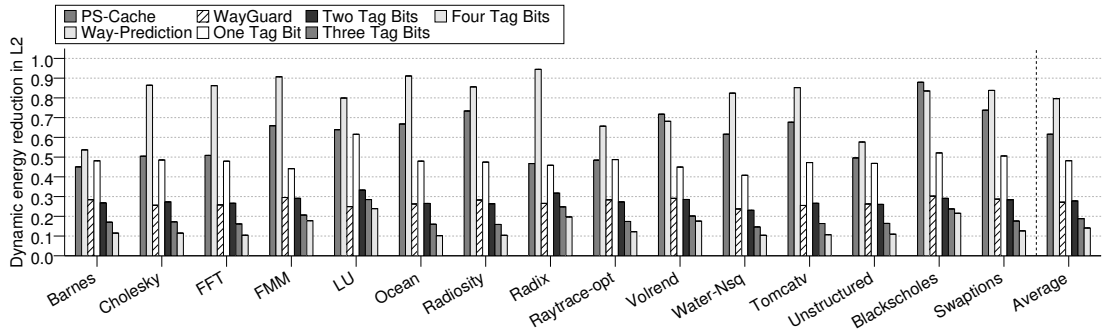
are looked up to just a single one, since our first-level cache has 8 ways, therefore allowing the consumption of a set-associative cache that uses this mechanism to be similar as that of a direct-mapped cache. There is no significant difference in the obtained results across the different applications for a given number of tag bits. Notice that it is possible to have an average number of ways accessed lower than 1, since it might happen that the least significant bits of the tag address have no match in the tag array. In this case, no way has to be accessed and the cache miss is triggered earlier than in a non-filtering approach. This rationale explains the results obtained for four bits.

Compared to the PS-Cache, the proposal always achieves better results even with just a single tag bit (*i.e.* X equal to 1 bit). The PS-Cache accesses on average to 4.6 ways and the results widely vary from one application to another. In some applications, like *Ocean*, there is almost no access reduction, whereas in others (*e.g. Tomcatv*) it can reduce it by about 50%. This variability in the results is due to the high variation in the private-shared access pattern across the applications. WayGuard and Way-Prediction access on average 2.41 and 1.43 ways, which remains mostly constant along all the studied applications. Thus, they perform better than the proposal with a single bit. Two bits are enough to surpass WayGuard and a third one is needed to surpass Way-Prediction. Using the MRU way as a prediction does prove to be good enough for first-level caches providing a good hit ratio.

Figure 5.15(b) shows the average number of searched ways in the 16-way L2 cache. The number of ways accessed on average is 7.68, 4.04, 2.43, and 1.74 for an X number of bits in the first comparison equal to one, two, three, and four bits, respectively. As in the L1 cache, the reduction balances evenly across all the studied applications. The trend shows that there is still room for improvement, but at the cost of increasing X. In comparison, the PS-Cache, Way-Prediction, and WayGuard access 9.85, 12.7, and 4.34 ways, respectively. Way-Prediction, which works really well for the L1 cache, performs poorly in lower levels of the cache hierarchy. The reason is that L1 caches filter many of the processor accesses, and thus, application locality is much poorer in lower levels. When the prediction hits, only a way is accessed, but when it misses the remaining ways have to be accessed. Therefore, the figure shows a poor hit ratio in the LLC. Also it is worth to note that a failed prediction also means a penalty in the access time since additional cycles are required in order to get the target data. That is, Way-Prediction is



(a) Dynamic energy consumed in the 8-way L1 cache normalized to a conventional cache.



(b) Dynamic energy consumed in the 16-way L2 cache normalized to a conventional cache.

FIGURE 5.16: Dynamic energy consumed in the cache hierarchy.

a hindrance for performance when applied in this level. Both Way-Prediction and PS-Cache perform worse than the TF Architecture even with one bit, whereas WayGuard performs almost as well as the proposal when employing a two-bit tag array.

As a consequence of reducing the number of accessed ways in caches, the dynamic energy consumption is also reduced. Figure 5.16(a) shows the dynamic energy consumed by the first-level cache. Results have been normalized to those of a set-associative cache in which all the cache ways are accessed, which also include the power overhead incurred by the extra comparators. The Tag Filter Cache is able to reduce the dynamic energy consumed by 48.1%, 65.8%, 73.2%, and 74.9% for a tag filter with one, two, three, and four bits, respectively. The marginal benefits of adding additional bits to the filter are fewer with each additional step, thus, the results for a five-bit filter do not differ much from those shown for a four-bit scheme. As expected, Way-Prediction shows the best results, being able to reduce dynamic energy consumption up to 82.1% in the Ocean application. The PS-Cache scheme obtains the worst results, since it is the scheme

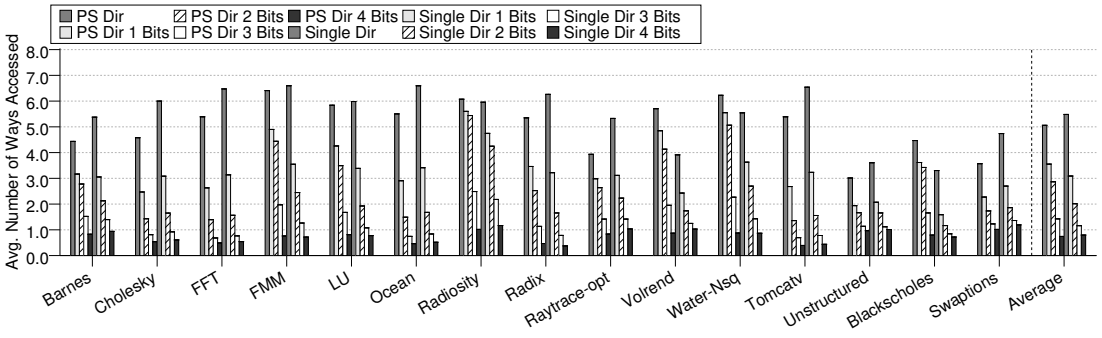


FIGURE 5.17: Average number of ways accessed in the directory per directory access across the studied schemes.

that accesses more ways. Figure 5.16(b) depicts the results for the L2 cache. The Tag Filter Cache is able to reduce consumption by 51.8%, 72.2%, 81.1%, and 85.9% for the different tag filter sizes, respectively. Again, one can see the diminishing benefits of further increasing the tag filter size. WayGuard achieves reductions similar as a 2-bit TF-Cache, whereas PS-Cache and Way-Prediction display no such improvements in comparison to the proposed architecture, reducing energy consumed only by 38.4% and 20.4%, respectively.

Since the proposed mechanism introducing no access time penalty, no performance evaluation results are shown.

5.3.3.3 TF Architecture in Directory Caches

This section evaluates the TF Architecture implemented both in a conventional single-level directory cache and in the recently proposed PS-Directory approach. Experimental results assume an 8-way conventional directory cache and a PS-Directory with a 2-way Shared cache and a 6-way Private cache. For each of them, we evaluated the effects of the proposal ranging the filter size from 1 bit to 4 bits in 1-bit steps.

Figure 5.17 shows the average number of accessed ways on a cache access in the studied schemes. As can be seen, when no filter is applied, a memory reference instruction accesses an average of 5.5 and 5.1 ways on each memory access for the directory cache and the PS-Directory, respectively. Unless some few exceptions, PS-Directory always accesses fewer ways than the conventional directory cache. The TF Architecture further improves these numbers. As happened in the TF-Cache, the more bits are used in the

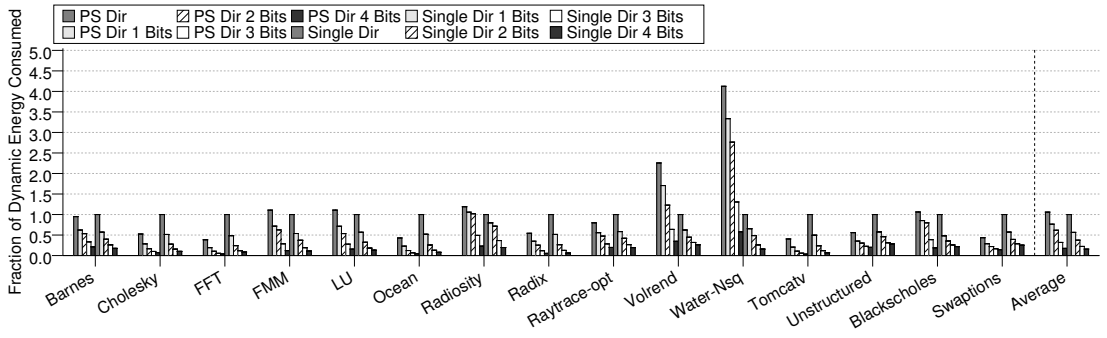


FIGURE 5.18: Normalized dynamic energy consumed by the directory across the studied schemes.

filtering, the less ways are accessed. Just one bit is enough to reduce to 3.1 and 3.6 the average number of accessed ways. The TF-Directory is able to reduce as much as 0.8 and 0.7 the accessed ways for directory cache and PS-Directory respectively, when employing 4 bits. The tag filtering behaves almost identically in both directory protocols which means that it is applicable to any other cache directory scheme. Also, as happened in the TF-Cache, directory misses could be detected earlier with this architecture if no tag comparison matches in the least significant bits.

Figure 5.18 shows the normalized dynamic energy consumed in the studied directory configurations. The proposal is able to achieve energy reductions by 30.2%, 43.6%, 71.4%, and 84.5% for an increasing tag filter size, respectively, in the PS-Directory. Analogously, reductions by 43.5%, 62.5%, 77.7%, and 84.2% are achieved in the single directory cache. Single-level directories take more advantage of tag filtering than many-level ones when few bits are selected in the filtering process. Nonetheless, results in energy consumed seem to converge as we increase the number of filtering bits. Comparing this figure with Figure 5.17, it can be appreciated that there is no direct correlation between energy consumption and average number of accessed ways. For this purpose we should account for the total number of accesses, which varies among the studied schemes since they produce a different number of invalidations due to evictions of entries in the directory cache. Below we present these results.

Figure 5.19 shows the normalized total number of ways accessed in the directory cache along the complete execution of the applications. These results effectively confirm that there is a direct correlation between the total number of accessed ways and dynamic energy consumed by the directory. As such, the lower this number, the more energy can

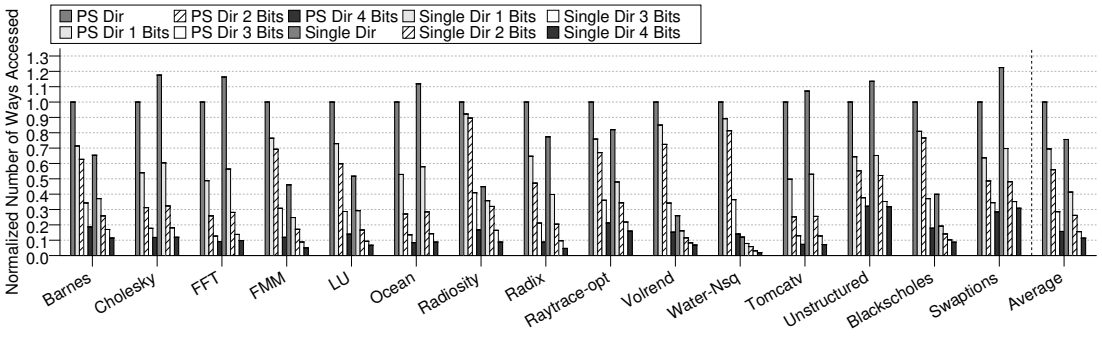


FIGURE 5.19: Normalized total number of accessed ways in the directory.

be saved. Remember that an access to the directory is triggered on a miss in the L1 processor cache for coherence maintenance. Even though the PS-Directory looks up a lower average number of ways per access than the single directory cache, it looks up on overall a higher number of ways because it performs more accesses to the directory. This is due to a higher number of the coverage misses. The tag filter is able to decrease the number of accessed ways from 69.4% with one single bit down to 15.7% with 4 bits in the PS-Directory and from 56.5% to 15.8% in the conventional directory cache. Although it seems to be a convergence in the total number of accessed ways as the tag filter size increases, slightly better results are achieved when the TF Architecture is applied to the conventional directory cache in comparison to the PS-Directory. The reason is the higher number of ways assumed in the conventional directory.

5.4 Summary

This chapter has proposed two approaches to save energy in processor caches and directory caches.

First, the PS-Cache is proposed. It is an energy-efficient cache design that only accesses a subset of the set ways, without hurting the performance. The PS-Cache assumes that blocks are classified at page level as shared or private, according to the TLB information. It also adds a single bit attached to each cache entry, which only activates the word line of the way if the block type matches the one provided by the TLB. In this way, dynamic energy consumption is largely saved. On the other hand, coherence requests to remote private caches only access the subset of ways that has blocks with the shared type.

Second, the TF Architecture (TF-Cache and TF-Directory) is proposed. As the previous approach, it aims to reduce dynamic power consumption in set-associative caches by accessing a subset of the set ways. The proposed mechanism saves a significant amount of energy by effectively reducing the number of searched ways by using a simple filtering mechanism based on the least significant bits of the address tag of the searched block. The proposal divides the tag array stored in the ways in two different segments. One of them, with few of the least significant bits and the other with the rest of bits of the tag. In order to filter the set ways, two sequential comparisons are performed. In the first comparison the least significant bits in the tag of the searched block are compared to the least significant bits stored in all the ways of the set. This comparison is performed very fast without waiting for the TLB output. Once we have the result of this comparison, the second is performed, comparing the rest of bits in the tag, but only for those ways that succeed the first comparison. If the data array is accessed in parallel with the tag array, then only those ways matching the first comparison are accessed in the data array. This filter choice is appropriate since, as results show, there is rather homogeneous distribution of the bit array field contents across the various ways of a set. This cache design can be implemented in any set-associative cache structure like private data or directory caches and in any cache level of the cache hierarchy.

Chapter 6

Conclusions

This dissertation has focused on the scalability issues found in two main types of CMP caches: directory caches and processor caches. Regarding directory scalability we have proposed two designs that achieve energy and area reductions by attacking the sharer vector present in these structures. By removing this field from a subset of the ways, and adapting the coherence protocol accordingly, notable savings can be obtained. Regarding processor caches, this work proposes several filtering mechanisms that reduce the number of ways looked up during each cache access. As a consequence the dynamic energy consumed by these heavily accessed structures is reduced.

In this chapter, the main contributions of these proposals are summarized, followed by a discussion about future work and an enumeration of the scientific publications related with this dissertation.

6.1 Contributions

Power consumption is a major design concern in current high-performance chip multiprocessors, which increases with the core count. On-chip caches often consume a significant fraction of the total power budget, and important research has focused on reducing energy consumption in these memory structures although typically at the cost of performance. Also, the increasing core counts in future manycore CMPs claim for scalable coherence structures in terms of power and area.

This work presents four main contributions that attack the scalability problem in terms of area and energy of the structures found in the cache hierarchy, i.e. processor and directory caches. Below we summarize the conclusions for each of them.

This work identifies five key characteristics that clearly differentiate the behavior of private and shared blocks from the directory point of view. Based on these observations, Section 4.1 has introduced the PS-Directory, a directory cache that uses two different cache structures, each one tailored to one type of block (i.e., private or shared). The Shared Directory Cache, which tracks shared blocks is small, with low associativity and fast. The Private Directory Cache is aimed at tracking private blocks, which are highly dominant in current workloads. This structure does not store the sharer vector, is larger than the shared cache, and it is implemented with higher associativity. Both, eDRAM and SRAM technologies, have been taken into consideration for the implementation of the Private Directory Cache.

Experimental results for a 16-core CMP show that, compared to a single directory cache with the same number of entries, the PS-Directory improves performance by 14% due to the separate treatment of private and shared blocks. Additionally, directory area is reduced by 26.35% mainly due to not storing the sharer vector for the private blocks, and by 33.98% when eDRAM technology is considered for the Private cache. Regarding energy consumption, reductions about 27% are achieved. Compared to the state-of-the-art MGD scheme, the PS-Directory increases the performance by 16.7% and reduces energy by 18.7%, being also much more scalable in terms of area. Finally, we would like to remark that the mentioned benefits are obtained with almost the same performance as the duplicate tags approach (i.e., perfect directory) but with a feasible implementation that scales with the number of cores.

This thesis shows that the current needs of multithreaded applications, regarding shared and private data access from the directory point of view, varies dynamically with execution time. Static private-shared structures are not able to properly adapt to this dynamic variation and, instead, dynamic strategies are in demand. Section 4.2 has presented the DWP-Directory, a sparse directory that sacrifices the sharer vector field from part of its ways in order to gain in both area and energy scalability. Furthermore, the implemented sharer vectors can be powered off or on as required according to whether the need of more shared ways rises or drops at run time, respectively. That is achieved by employing the repartitioning algorithm also proposed in this thesis.

Experimental results for a 16-core CMP show that, compared to a conventional directory cache with the same number of entries, DWP-Directory reduces the static and dynamic energy consumed by 31.5% and 59.9%, respectively, while having an almost negligible performance penalty when compared to a more energy and area demanding 8-way conventional cache, and having a lower execution time than a more power-efficient 4-way directory.

Section 5.2 has proposed the PS-Cache, an energy-efficient cache design that only accesses a subset of the set ways, without hurting the performance. The PS-Cache assumes that blocks are classified at page level as shared or private, according to the TLB information. It also adds a single bit attached to each cache entry, which only activates the word line if the block type matches the one provided by the TLB. In this way, dynamic energy consumption is largely saved. On the other hand, coherence requests to remote private caches only access the subset of ways that has blocks with the shared type.

Results have shown that in CMPs, implementing either directory-based or snoopy-based protocols, the PS-Cache can bring important energy savings. The proposal has been evaluated in both L1 and L2 caches showing energy reductions by 22% and 40% for both of them, respectively, indistinctly of the coherence protocol employed.

Finally, Section 5.3 has proposed the TF-Architecture to reduce dynamic power consumption in set-associative caches. The proposal divides the tag array in two different segments. One of them, with few of the least significant bits and the other with the rest of bits of the tag. In order to filter the set ways, two comparisons are performed. In the first comparison the least significant bits in the tag of the searched block are compared to the least significant bits of the tags stored in all the ways of the set. This comparison

is performed very fast without waiting for the TLB translation. Once we have the result of this comparison, the second is performed (if there is at least a hit), comparing the rest of bits in the tag, but only for those ways that succeed the first comparison. If the data array is accessed in parallel with the tag array, then only those ways matching the first comparison are accessed in the data array. This filter choice is appropriate since, as results show, there is rather homogeneous distribution of the bit array field contents across the various ways of a set. In this work we have applied the TF-Architecture to data caches (TF-Cache) and directory caches (TF-Directory).

Results show that TF-Cache can reduce up to 87.75% and 89.13% the average number of ways that are looked up on every access to the L1 and L2 caches, respectively; which translates in energy savings by 74.9% and 85.9%. Compared to other state-of-the-art schemes, TF-Cache achieves better results than the compared architectures, with the only exception of Way-Prediction in first-level caches by a small margin. Way-Prediction has been proven to be ineffective when applied to other levels of the cache hierarchy, whereas our TF Architecture works better at any level. Meanwhile, results for the TF-Directory show that up to 84% of the ways accessed by the conventional directory cache can be filtered, which translates in roughly the same percentage of energy savings.

6.2 Future Work

As for future work, other architectural solutions to the scalability problems focused on this work are planned to be researched.

Even though duplicated tags schemes offer better performance than their sparse schemes counterparts, due to the lack of coverage misses, the highly-associative lookups needed to build the sharer vector difficult the scalability, specially in terms of energy. Both directory types have advantages and disadvantages. The potential of combining both of them to overcome their natural inconveniences is worth researching. For instance, a two-level directory comprised of a small sparse directory that serves as a cache for the coherence information found in the second level: a duplicated tag directory. Directory misses will be avoided thanks to the second level, which will keep track of all blocks being stored in the private caches, while access latency and energy consumption will be minimized by the first level sparse directory.

Furthermore, most of the mechanisms detailed in this thesis are orthogonal to each other and can, hence, be applied simultaneously. To the best of our knowledge, no approach has been proposed dealing with such a kind of research. Additionally, the base system mentioned above can be further improved by also employing these ideas, more concretely way filtering for the duplicated tag directory and the dynamic adaptation to shared and private block requirements for the sparse directory.

Finally, other repartitioning algorithms could be deployed and evaluated to check if they adapt better than the currently proposed one.

6.3 Publications

The following papers related with this dissertation were accepted for publication in different international journals and conferences.

Journals:

- Joan J. Valls, Alberto Ros, Julio Sahuquillo, and María E. Gómez. PS directory: a scalable multilevel cache for CMPs. *Journal of Supercomputing*, volume 71, issue 8, pages 2847-2876, 2015.
- Joan J. Valls, Alberto Ros, Julio Sahuquillo, and María E. Gómez. PS-Cache: an energy-efficient cache design for chip multiprocessors *Journal of Supercomputing*, volume 71, issue 1, pages 67-86, 2015.
- Joan J. Valls, Alberto Ros, María E. Gómez, and Julio Sahuquillo. The Tag Filter Architecture: An energy-efficient cache and directory design. *Journal of Parallel and Distributed Computing*, volume 100, pages 193-202, 2017.

Conferences:

- Joan J. Valls, Alberto Ros, Julio Sahuquillo, María E. Gómez and José F. Duato. PS-Dir: A Scalable Two-Level Directory Cache. In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques* (PACT), pages 451-452, Minneapolis, Minnesota, USA, 2012.

- Joan J. Valls, Alberto Ros, Julio Sahuquillo and María E. Gómez. PS-cache: An energy-efficient cache design for chip multiprocessors. In *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques* (PACT), page 407, Edinburgh, Scotland, UK, 2013.

- Joan J. Valls, Julio Sahuquillo, Alberto Ros and María E. Gómez. The Tag Filter Cache: An Energy-Efficient Approach. In *Proceedings of the 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing* (PDP), pages 182-189, Turku, Finland, 2015.

- Joan J. Valls, María E. Gómez, Alberto Ros and Julio Sahuquillo. A Directory Cache with Dynamic Private-Shared Partitioning. In *Proceedings of the 23rd annual IEEE International Conference on High Performance Computing* (HiPC), pages 382-391, Hyderabad, India, 2016. This publication received a HiPC Best Paper Award.

In addition, other related papers have been published in domestic conferences:

- Joan J. Valls, Alberto Ros, Julio Sahuquillo and María E. Gómez. El directorio PS: Una caché de directorio multinivel escalable para CMPs. In *XXIII edición Jornadas de Paralelismo SARTECO*, pages 455-460, Elx, Spain, 2012.

- Joan J. Valls, Alberto Ros, Julio Sahuquillo and María E. Gómez. PS-Cache: Un diseño energéticamente eficiente para caches en CMPs. In *XXVI edición Jornadas de Paralelismo SARTECO*, pages 73-81, Córdoba, Spain, 2015.

- Joan J. Valls, Julio Sahuquillo, Alberto Ros and María E. Gómez. Reduciendo el consumo dinámico de energía con Tag Filter Cache In *XXVII edición Jornadas de Paralelismo SARTECO*, pages 525-532, Salamanca, Spain, 2016.

All works listed above are exclusively related with this thesis. The specific contributions of the Ph.D. candidate reside mostly in the implementation of the proposed techniques, the setup and execution of most simulation experiments, the writing of the paper drafts

describing the work as well as the presentation in the conferences. Along these processes, the co-authors have repeatedly provided useful hints and advices, which the Ph.D. candidate has then applied to make the work evolve into its final version.

References

[1] Rajeev Balasubramonian, Norman Paul Jouppi, and Naveen Muralimanohar. *Multi-Core Cache Hierarchies*. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2011.

[2] Michael R. Marty and Mark D. Hill. Virtual hierarchies. *IEEE Micro*, 28(1):99–109, January 2008.

[3] Michael R. Marty and Mark D. Hill. Virtual hierarchies to support server consolidation. In *34th Int'l Symp. on Computer Architecture (ISCA)*, pages 46–56, June 2007.

[4] Blas Cuesta, Alberto Ros, María E. Gómez, Antonio Robles, and José Duato. Increasing the effectiveness of directory caches by deactivating coherence for private memory blocks. In *38th Int'l Symp. on Computer Architecture (ISCA)*, pages 93–103, June 2011.

[5] Pat Conway, Nathan Kalyanasundharam, Gregg Donley, Kevin Lepak, and Bill Hughes. Cache hierarchy and memory subsystem of the AMD opteron processor. *IEEE Micro*, 30(2):16–29, April 2010.

[6] Luiz A. Barroso, Kourosh Gharachorloo, and Robert McNamara, et al. Piranha: A scalable architecture based on single-chip multiprocessing. In *27th Int'l Symp. on Computer Architecture (ISCA)*, pages 12–14, June 2000.

[7] Manish Shah, Jama Barreh, and Jeff Brooks, et al. UltraSPARC T2: A highly-threaded, power-efficient, SPARC SoC. In *IEEE Asian Solid-State Circuits Conference*, pages 22–25, November 2007.

[8] Jason Zebchuk, Vijayalakshmi Srinivasan, Moinuddin K. Qureshi, and Andreas Moshovos. A tagless coherence directory. In *42nd IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, pages 423–434, December 2009.

[9] Manuel E. Acacio, José González, José M. García, and José Duato. A two-level directory architecture for highly scalable cc-NUMA multiprocessors. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 16(1):67–79, January 2005.

[10] Song-Liu Guo, Hai-Xia Wang, Yi-Bo Xue, Chong-Min Li, and Dong-Sheng Wang. Hierarchical cache directory for cmp. *Journal of Computer Science and Technology*, 25(2):246–256, March 2010.

[11] Jason Zebchuk, Babak Falsafi, and Andreas Moshovos. Multi-grain coherence directories. In *46th IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, pages 359–370, December 2013.

[12] Stefanos Kaxiras, Zhigang Hu, and Margaret Martonosi. Cache decay: Exploiting generational behavior to reduce cache leakage power. In *28th Int'l Symp. on Computer Architecture (ISCA)*, pages 240–251, June 2001.

[13] Krisztián Flautner, Nam Sung Kim, Steve Martin, David Blaauw, Trevor Mudge, Stefanos Kaxiras, Zhigang Hu, and Margaret Martonosi. Drowsy caches: Simple techniques for reducing leakage power. In *29th Int'l Symp. on Computer Architecture (ISCA)*, pages 148–157, May 2002.

[14] Mrinmoy Ghosh, Emre Özer, Simon Ford, Stuart Biles, and Hsien-Hsin S. Lee. Way guard: A segmented counting bloom filter approach to reducing energy for set-associative caches. In *Int'l Symp. on Low Power Electronics and Design (ISLPED)*, pages 165–170, August 2009.

[15] Brad Calder and Dirk Grunwald. Predictive sequential associative cache. In *2nd Int'l Symp. on High-Performance Computer Architecture (HPCA)*, pages 244–253, February 1996.

[16] Peng Liu, Lei Fang Michael C. Huang, , Qi Hu, and Guofan Jiang. Building expressive and area-efficient directories with hybrid representation and adaptive multi-granular tracking. *IEEE Transactions on Computers (TC)*, May 2015.

[17] Richard E. Matick and Stanley E. Schuster. Logic-based eDRAM: Origins and rationale for use. *IBM Journal of Research and Development*, 49(1):145–165, January 2005.

[18] J. M. Tendler, J. S. Dodson, J. S. Fields, H. Le, and B. Sinharoy. POWER4 system microarchitecture. *IBM Journal of Research and Development*, 46(1):5–25, January 2002.

[19] B. Sinharoy, R. N. Kalla, J. M. Tendler, R. J. Eickemeyer, and J. B. Joyner. Power5 system microarchitecture. *IBM Journal of Research and Development*, 49(4/5):505–521, July 2005.

[20] Ron Kalla, Balaram Sinharoy, William J. Starke, and Michael Floyd. POWER7: IBMs next-generation server processor. *IEEE Micro*, 30(2):7–15, April 2010.

[21] Alejandro Valero, Julio Sahuquillo, Salvador Petit, Vicente Lorente, Ramon Canal, Pedro López, and José Duato. An hybrid eDRAM/SRAM macrocell to implement first-level data caches. In *42nd IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, pages 213–221, December 2009.

[22] Xiaoxia Wu, Jian Li, Lixin Zhang, Evan Speight, Ram Rajamony, and Yuan Xie. Hybrid cache architecture with disparate memory technologies. In *36th Int'l Symp. on Computer Architecture (ISCA)*, pages 34–45, June 2009.

[23] Alan Jay Smith. Cache memories. *ACM Comput. Surv.*, 14(3):473–530, September 1982. ISSN 0360-0300. doi: 10.1145/356887.356892. URL `http://doi.acm.org/10.1145/356887.356892`.

[24] Changkyu Kim, Doug Burger, and Stephen W. Keckler. An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches. In *10th Int'l Conf. on Architectural Support for Programming Language and Operating Systems (ASPLOS)*, pages 211–222, October 2002.

[25] Alberto Ros, Manuel E. Acacio, and José M. García. A scalable organization for distributed directories. *Journal of Systems Architecture (JSA)*, 56(2-3):77–87, February 2010.

[26] Michael Ferdman, Pejman Lotfi-Kamran, Ken Balet, and Babak Falsafi. Cuckoo directory: A scalable directory for many-core systems. In *17th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, pages 169–180, February 2011.

[27] Anoop Gupta, Wolf-Dietrich Weber, and Todd C. Mowry. Reducing memory traffic requirements for scalable directory-based cache coherence schemes. In *Int'l Conf. on Parallel Processing (ICPP)*, pages 312–321, August 1990.

[28] Manuel E. Acacio, José González, José M. García, and José Duato. A new scalable directory architecture for large-scale multiprocessors. In *7th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, pages 97–106, January 2001.

[29] David Chaiken, John Kubiatowicz, and Anant Agarwal. LimitLESS directories: A scalable cache coherence scheme. In *4th Int'l Conf. on Architectural Support for Programming Language and Operating Systems (ASPLOS)*, pages 224–234, April 1991.

[30] Guoying Chen. Slid - a cost-effective and scalable limited-directory scheme for cache coherence. In *5th Int'l Conf. on Parallel Architectures and Languages Europe (PARLE)*, pages 341–352, June 1993.

[31] Brian W. O'Krafka and A. Richard Newton. An empirical evaluation of two memory-efficient directory methods. In *17th Int'l Symp. on Computer Architecture (ISCA)*, pages 138–147, June 1990.

[32] Daniel Sanchez and Christos Kozyrakis. SCD: A scalable coherence directory with flexible sharer set encoding. In *18th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, pages 129–140, February 2012.

[33] Lei Fang, Peng Liu, Qi Hu, Michael C. Huang, and Guofan Jiang. Building expressive, area-efficient coherence directories. In *22nd Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, pages 299–308, September 2013.

[34] Michael Powell, Se hyun Yang, Babak Falsafi, Kaushik Roy, and T. N. Vijaykumar. Gated-Vdd: A circuit technique to reduce leakage in deep-submicron cache memories. In *Int'l Symp. on Low Power Electronics and Design (ISLPED)*, pages 90–95, July 2000.

[35] David H. Albonesi. Selective cache ways: On-demand cache resource allocation. In *32nd IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, pages 248–259, December 1999.

[36] Jongmin Lee, Seokin Hong, and Soontae Kim. Tlb index-based tagging for cache energy reduction. In *17th Int'l Symp. on Low Power Electronics and Design (ISLPED)*, pages 85–90, August 2011.

[37] Karthik T. Sundararajan, Vasileios Porpodas, Timothy M. Jones, Nigel P. Topham, and Björn Franke. Cooperative partitioning: Energy-efficient cache partitioning for high-performance cmps. In *18th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, pages 311–322, February 2012.

[38] Kamil Kedzierski, Francisco J. Cazorla, Roberto Gioiosa, Alper Buyuktosunoglu, and Mateo Valero. Power and performance aware reconfigurable cache for cmps. In *2nd Int'l Forum on Next-Generation Multicore/Manycore Technologies*, pages 1–12, June 2010.

[39] Julio Sahuquillo and Ana Pont. Splitting the data cache: A survey. *IEEE Concurrency*, 8(3):30–35, July 2000. ISSN 1092-3063. doi: 10.1109/4434.865890. URL `http://dx.doi.org/10.1109/4434.865890`.

[40] Daehoon Kim, Jeongseob Ahn Jaehong Kim, and Jaehyuk Huh. Subspace snooping: Filtering snoops with operating system support. In *19th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, pages 111–122, September 2010.

[41] Nikos Hardavellas, Michael Ferdman, Babak Falsafi, and Anastasia Ailamaki. Reactive NUCA: Near-optimal block placement and replication in distributed caches. In *36th Int'l Symp. on Computer Architecture (ISCA)*, pages 184–195, June 2009.

[42] Jiayuan Meng and Kevin Skadron. Avoiding cache thrashing due to private data placement in last-level cache for manycore scaling. In *Int'l Conf. on Computer Design (ICCD)*, pages 282–288, October 2009.

[43] Yong Li, Ahmed Abousamra, Rami Melhem, and Alex K. Jones. Compiler-assisted data distribution for chip multiprocessors. In *19th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, pages 501–512, September 2010.

[44] Yong Li, Rami G. Melhem, and Alex K. Jones. Practically private: Enabling high performance cmps through compiler-assisted data classification. In *21st Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, pages 231–240, September 2012.

[45] Seth H. Pugsley, Josef B. Spjut, David W. Nellans, and Rejeev Balasubramonian. SWEL: Hardware cache coherence protocols to map shared data onto shared caches. In *19th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, pages 465–476, September 2010.

[46] Hemayet Hossain, Sandhya Dwarkadas, and Michael C. Huang. POPS: Coherence protocol optimization for both private and shared data. In *20th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, pages 45–55, October 2011.

[47] Mohammad Alisafaee. Spatiotemporal coherence tracking. In *45th IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, pages 341–350, December 2012.

[48] Alberto Ros and Stefanos Kaxiras. Complexity-effective multicore coherence. In *21st Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, pages 241–252, September 2012.

[49] Peter S. Magnusson, Magnus Christensson, and Jesper Eskilson, et al. Simics: A full system simulation platform. *IEEE Computer*, 35(2):50–58, February 2002.

[50] Milo M.K. Martin, Daniel J. Sorin, and Bradford M. Beckmann, et al. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. *Computer Architecture News*, 33(4):92–99, September 2005.

[51] Niket Agarwal, Tushar Krishna, Li-Shiuan Peh, and Niraj K. Jha. GARNET: A detailed on-chip network model inside a full-system simulator. In *IEEE Int'l Symp. on Performance Analysis of Systems and Software (ISPASS)*, pages 33–42, April 2009.

[52] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P. Jouppi. Cacti 6.0. Technical Report HPL-2009-85, HP Labs, April 2009.

[53] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. The SPLASH-2 programs: Characterization and methodological

considerations. In *22nd Int'l Symp. on Computer Architecture (ISCA)*, pages 24–36, June 1995.

[54] Man-Lap Li, Ruchira Sasanka, Sarita V. Adve, Yen-Kuang Chen, and Eric Debes. The ALPBench benchmark suite for complex multimedia applications. In *Int'l Symp. on Workload Characterization*, pages 34–45, October 2005.

[55] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The PARSEC benchmark suite: Characterization and architectural implications. In *17th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, pages 72–81, October 2008.

[56] Alaa R. Alameldeen and David A. Wood. IPC considered harmful for multiprocessor workloads. *IEEE Micro*, 26(4):8–17, July 2006.

[57] Alberto Ros, Blas Cuesta, Ricardo Fernández-Pascual, Maria E. Gómez, Manuel E. Acacio, Antonio Robles, José M. García, and José Duato. Extending magny-cours cache coherence. *IEEE Transactions on Computers (TC)*, 61(5):593–606, May 2012.

[58] Joan J. Valls, Alberto Ros, Julio Sahuquillo, María Engracia Gómez, and José Duato. PS-Dir: A scalable two-level directory cache. In *21st Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, pages 451–452, September 2012.

[59] Joan J. Valls, Alberto Ros, Julio Sahuquillo, and María Engracia Gómez. PS directory: a scalable multilevel directory cache for cmps. *The Journal of Supercomputing*, 71(8):2847–2876, 2015.

[60] Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi. Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *42nd IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, pages 469–480, December 2009.

[61] Niket Agarwal, Li-Shiuan Peh, and Niraj K. Jha. In-Network Snoop Ordering (INSO): Snoopy coherence on unordered interconnects. In *15th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, pages 67–78, February 2009.

[62] Jason F. Cantin, James E. Smith, Mikko H. Lipasti, Andreas Moshovos, and Babak Falsafi. Coarse-grain coherence tracking: RegionScout and region coherence arrays. *IEEE Micro*, 26(1):70–79, January 2006.

[63] Alberto Ros, Blas Cuesta, María E. Gómez, Antonio Robles, and José Duato. Temporal-aware mechanism to detect private data in chip multiprocessors. In *42nd Int'l Conf. on Parallel Processing (ICPP)*, pages 562–571, October 2013.

[64] Bruce Jacob, Spencer Ng, and David Wang. *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann Publishers, Inc., 4th edition, 2007.

[65] David A. Patterson and John L. Hennessy. *Computer Organization and Design: The Hardware/Software Interface (The Morgan Kaufmann Series in Computer Architecture and Design)*. Morgan Kaufmann Publishers Inc., 4th edition, 2008.

[66] W. J. Starke, J. Stuecheli, D. M. Daly, J. S. Dodson, F. Auernhammer, P. M. Sagmeister, G. L. Guthrie, C. F. Marino, M. Siegel, and B. Blaner. The cache and memory subsystems of the ibm power8 processor. *IBM Journal of Research and Development*, 59(1):3:1–3:13, Jan 2015.

[67] Alberto Ros, Polychronis Xekalakis, Marcelo Cintra, Manuel E. Acacio, and José M. García. Ascib: Adaptive selection of cache indexing bits for reducing conflict misses. In *Int'l Symp. on Low Power Electronics and Design (ISLPED)*, pages 51–56, July 2012.

[68] 27-inch imac, technical specifications, available online (nov, 2014) at http://www.apple.com/imac/specs/.