Hybrid caches: design and data management

ALEJANDRO VALERO BRESÓ

EDITORIAL
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Universitat Politècnica de València

Departamento de Informática
de Sistemas y Computadores

Hybrid Caches: Design and Data Management

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

*Doctor of Philosophy
(Computer Engineering)*

Author

Alejandro Valero Bresó

Advisors

*Prof. Julio Sahuquillo Borrás
Prof. Salvador V. Petit Martí*

September 2013

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

First Edition, 2013

© Alejandro Valero Bresó

© of the present edition:
Editorial Universitat Politècnica de València / www.lalibreria.upv.es

ISBN: 978-84-9048-137-0 (printed version)
Publishing reference: 5662

Doctoral Committee

- Prof. Ramon Canal Corretger

  Universitat Politècnica de Catalunya, Barcelona, Spain.

- Prof. José F. Duato Marín

  Universitat Politècnica de València, Valencia, Spain.

- Prof. María Engracia Gómez Requena

  Universitat Politècnica de València, Valencia, Spain.

- Prof. Antonio González Colás

  Intel Labs and Universitat Politècnica de Catalunya, Barcelona, Spain.

- Prof. David R. Kaeli

  Northeastern University, Boston, MA, USA.

Agraïments

Aquest treball ha requerit de l'esforç de moltes persones perquè es pogués dur a terme. M'agradaria començar pels meus directors de tesi Julio i Salva. Gràcies per haver confiat en mi i per transmetre'm els vostres coneixements i il·lusió per la reçerca en arquitectura de computadors. Als meus pares Carmen i Antonio, les meues àvies Carmen i Vicenta, la meua nóvia Amelia, i en definitiva a tota la meua família i amics, gràcies pel vostre suport, companyia i afecte al llarg de tots aquests anys.

Vull agrair l'esforç de la resta d'autors de les nostres publicacions científiques, José, Pedro, Vicent i Ramon, que amb els seus coneixements i experiència m'han ajudat a millorar la qualitat del treball que es presenta en aquesta tesi.

També m'agradaria destacar l'ajuda prestada i el magnífic ambient de treball creat per tots els becaris, professors i altres membres que han format part del Departament d'Informàtica de Sistemes i Computadors durant aquest temps.

Moltes gràcies per tot!

Contents

Abstract

Cache memories have been usually implemented with Static Random-Access Memory (SRAM) technology since it is the fastest electronic memory technology. However, this technology consumes a high amount of leakage currents, which is a major design concern because leakage energy consumption increases as the transistor size shrinks. Alternative technologies are being considered to reduce this consumption. Among them, embedded Dynamic RAM (eDRAM) technology provides minimal area and leakage by design but reads are destructive and it is not as fast as SRAM.

In this thesis, both SRAM and eDRAM technologies are mingled to take the advantatges that each of them offers. First, they are combined at cell level to implement an n-bit *macrocell* consisting of one SRAM cell and *n-1* eDRAM cells. The macrocell is used to build n-way set-associative hybrid first-level (L1) data caches having one SRAM way and *n-1* eDRAM ways. A single SRAM way is enough to achieve good performance given the high data locality of L1 caches. Architectural mechanisms such as way-prediction, swaps, and scrub operations are considered to avoid unnecessary eDRAM reads, to maintain the Most Recently Used (MRU) data in the *fast* SRAM way, and to completely avoid refresh logic. Experimental results show that, compared to a conventional SRAM cache, leakage and area are largely reduced with a scarce impact on performance.

The study of the benefits of hybrid caches has been also carried out in second-level (L2) caches acting as Last-Level Caches (LLCs). In this case, the technologies are combined at bank level and the optimal ratio of SRAM and eDRAM banks that achieves the best trade-off among performance, energy, and area is identified. Like in L1 caches, the MRU blocks are kept in the SRAM banks and they are accessed first to avoid many eDRAM reads. Nevertheless, refresh logic is not removed since data locality widely differs in this cache level. Experimental results show that a hybrid LLC with an eighth of its banks built with SRAM technology is enough to achieve the best target trade-off.

This dissertation also deals with performance of replacement policies in heterogeneous LLCs mainly focusing on the energy overhead incurred by refresh operations. In this thesis it is defined a new concept, namely MRU-Tour (MRUT), that helps estimate reuse information of cache blocks. Based on this concept, it is proposed a family of MRUT-based replacement algorithms that randomly select the victim block among those having a single MRUT. These policies are enhanced to leverage recency of information for a few blocks and to adapt to changes in the working set of the benchmarks. Results show that the proposed MRUT policies, with simpler hardware complexity, outperform the Least Recently Used (LRU) policy and a set of the most representative state-of-the-art replacement policies for LLCs.

Refresh operations represent an important fraction of the overall dynamic energy consumption of eDRAM LLCs. This fraction increases with the cache capacity, since more blocks have to be refreshed for a given period of time. Prior works have attacked the refresh energy taking into account inter-cell feature variations. Unlike these works, this thesis proposes a selective refresh policy based on the MRUT concept. The devised policy takes into account the number of MRUTs of a block to select whether the block is refreshed. In this way, many refreshes done in a typical distributed refresh policy are skipped (i.e., in those blocks having a single MRUT). This refresh mechanism is applied in the hybrid LLC memory. Results show that refresh energy consumption is largely reduced with respect to a conventional eDRAM cache, while the performance degradation is minimal with respect to a conventional SRAM cache.

Resumen

Las memorias caché o *cache* han sido implementadas normalmente con tecnología *Static Random-Access Memory* (SRAM) ya que es la tecnología de memoria electrónica más rápida. Sin embargo, esta tecnología consume una gran cantidad de corrientes de fuga, lo cual es un problema de diseño importante porque el consumo de corrientes de fuga incrementa a medida que el tamaño del transistor encoge. Se están considerando tecnologías alternativas para reducir este consumo. Entre ellas, la tecnología *embedded Dynamic* RAM (eDRAM) ofrece por diseño un área y corrientes de fuga mínimas pero las lecturas son destructivas y no es tan rápida como SRAM.

En esta tesis, ambas tecnologías SRAM y eDRAM se mezclan para conseguir las ventajas que cada una de ellas ofrece. En primer lugar, se combinan a nivel de celda para implementar una *macrocelda* de *n*-bits consistente en una celda SRAM y *n-1* celdas eDRAM. La macrocelda se utiliza para construir caches híbridas de datos de primer nivel (L1) asociativas por conjuntos de *n*-vías que tienen una vía SRAM y *n-1* vías eDRAM. Una sola vía SRAM es suficiente para conseguir buenas prestaciones dado que la localidad de los datos en caches L1 es elevada. Mecanismos arquitectónicos como predicción de vía, intercambio de datos (*swaps*) y operaciones de *scrub* se consideran para evitar lecturas eDRAM innecesarias, mantener los datos más recientemente utilizados (MRU) en la vía SRAM *rápida* y eliminar completamente la lógica de refresco. Los resultados experimentales muestran que, comparado con una cache convencional SRAM, las corrientes de fuga y área se reducen considerablemente con un impacto escaso en las prestaciones.

El estudio de los beneficios de las caches híbridas también se ha llevado a cabo en caches de segundo nivel (L2) actuando como caches de último nivel (LLCs). En este caso, las tecnologías se combinan a nivel de banco y se identifica el ratio óptimo de bancos SRAM y eDRAM que consigue el mejor compromiso entre prestaciones, energía y área. Como en las caches L1, los bloques MRU se mantienen en los bancos SRAM y se acceden primero para evitar muchas lecturas eDRAM. Sin embargo, la lógica de refresco no se elimina ya que la localidad de los datos difiere ampliamente en este nivel de cache. Los resultados experimentales muestran que una LLC híbrida con un octavo de sus bancos implementados con tecnología SRAM es suficiente para conseguir el mejor compromiso.

Esta disertación también se ocupa de las prestaciones de las políticas de reemplazo en LLCs heterogéneas centrándose principalmente en la sobrecarga de energía incurrida por las operaciones de refresco. En esta tesis se define un concepto nuevo, llamado MRU-Tour (MRUT), que ayuda a la estimación de información de reuso de los bloques de cache. Basándose en este concepto, se propone una familia de algoritmos de reemplazo basados en MRUT que seleccionan aleatoriamente los bloques víctima entre aquellos que tienen un solo MRUT. Estas políticas se mejoran para hacer uso de la recencia de información de unos pocos bloques y adaptarse a los cambios en el comportamiento de las aplicaciones. Los resultados muestran que las políticas MRUT propuestas, con menor complejidad *hardware*, mejoran las prestaciones de *Least Recently Used* (LRU) y de un conjunto representativo del estado del arte de algoritmos de reemplazo para las LLC.

Las operaciones de refresco representan una fracción importante del consumo total de energía dinámica de las LLC eDRAM. Esta fracción incrementa con la capacidad de cache, ya que una cantidad mayor de bloques tienen que ser refrescados en un periodo de tiempo dado. Algunos trabajos anteriores han atacado la energía de refresco teniendo en cuenta las variaciones de los componentes entre celdas. A diferencia de estos trabajos, esta tesis propone una política de refresco selectiva basada en el concepto de MRUT. La política ideada tiene en cuenta el número de MRUTs de un bloque para seleccionar si el bloque se refresca. De esta manera, muchos refrescos realizados en una política de refresco típica y distribuida se omiten, es decir, en aquellos bloques que tienen un solo MRUT. Este mecanismo de refresco se aplica en la memoria LLC híbrida. Los resultados muestran que el consumo de energía de refresco se reduce ampliamente respecto a una cache convencional eDRAM, mientras que la degradación de prestaciones es mínima respecto a una cache convencional SRAM.

Resum

Les memòries cau o *cache* han estat normalment implementades amb tecnologia *Static Random-Access Memory* (SRAM) ja que és la tecnologia de memòria electrònica més ràpida. No obstant això, aquesta tecnologia consumeix una gran quantitat de corrents de fugida, la qual cosa és un problema de disseny important perquè el consum de corrents de fugida incrementa a mesura que la grandària del transistor encongeix. S'estan considerant tecnologies alternatives per reduir aquest consum. Entre elles, la tecnologia *embedded Dynamic* RAM (eDRAM) ofereix per disseny un àrea i corrents de fugida mínimes però les lectures són destructives i no és tan ràpida com SRAM.

En aquesta tesi, ambdues tecnologies SRAM i eDRAM es barregen per aconseguir els avantatges que cadascuna d'elles ofereix. En primer lloc, es combinen a nivell de cel·la per implementar una *macrocel·la* de *n*-bits consistent en una cel·la SRAM i *n-1* cel·les eDRAM. La macrocel·la s'utilitza per construir caches híbrides de dades de primer nivell (L1) associatives per conjunts de *n*-vies que tenen una via SRAM i *n-1* vies eDRAM. Una sola via SRAM és suficient per aconseguir bones prestacions atès que la localitat de les dades en caches L1 és elevada. Mecanismes arquitectònics com predicció de via, intercanvi de dades (*swaps*) i operacions de *scrub* es consideren per evitar lectures eDRAM innecessàries, mantenir les dades més recentment utilitzades (MRU) en la via SRAM *ràpida* i eliminar completament la lògica de refresc. Els resultats experimentals mostren que, comparat amb una cache convencional SRAM, els corrents de fugida i àrea es redueixen considerablement amb un impacte escàs en les prestacions.

L'estudi dels beneficis de les caches híbrides també s'ha dut a terme en caches de segon nivell (L2) actuant com caches d'últim nivell (LLCs). En aquest cas, les tecnologies es combinen a nivell de banc i s'identifica el ràtio òptim de bancs SRAM i eDRAM que aconsegueix el millor compromís entre prestacions, energia i àrea. Com en les caches L1, els blocs MRU es mantenen als bancs SRAM i s'accedeixen primer per a evitar moltes lectures eDRAM. No obstant això, la lògica de refresc no s'elimina ja que la localitat de les dades difereix àmpliament en aquest nivell de cache. Els resultats experimentals mostren que una LLC híbrida amb un vuitè dels seus bancs implementats amb tecnologia SRAM és suficient per aconseguir el millor compromís.

Aquesta dissertació també s'ocupa de les prestacions de les polítiques de reemplaçament en LLCs heterogènies centrant-se principalment en la sobrecàrrega d'energia incorreguda per les operacions de refresc. En aquesta tesi es defineix un concepte nou, anomenat

MRU-Tour (MRUT), que ajuda a l'estimació d'informació de reús dels blocs de cache. Basant-se en aquest concepte, es proposa una família d'algorismes de reemplaçament basats en MRUT que seleccionen aleatòriament els blocs víctima entre aquells que tenen un sol MRUT. Aquestes polítiques es milloren per fer ús de la recència d'informació d'uns pocs blocs i adaptar-se als canvis en el comportament de les aplicacions. Els resultats mostren que les polítiques MRUT propostes, amb menor complexitat *hardware*, milloren les prestacions de *Least Recently Used* (LRU) i d'un conjunt representatiu de l'estat de l'art d'algorismes de reemplaçament per les LLC.

Les operacions de refresc representen una fracció important del consum total d'energia dinàmica de les LLC eDRAM. Aquesta fracció incrementa amb la capacitat de cache, ja que una quantitat major de blocs han de ser refrescats en un període de temps donat. Alguns treballs anteriors han atacat l'energia de refresc tenint en compte les variacions dels components entre cel·les. A diferència d'aquests treballs, aquesta tesi proposa una política de refresc selectiva basada en el concepte de MRUT. La política ideada té en compte el nombre de MRUTs d'un bloc per seleccionar si el bloc es refresca. D'aquesta manera, molts refrescs realitzats en una política de refresc típica i distribuïda s'ometen, és a dir, en aquells blocs que tenen un sol MRUT. Aquest mecanisme de refresc s'aplica en la memòria LLC híbrida. Els resultats mostren que el consum d'energia de refresc es redueix àmpliament respecte a una cache convencional eDRAM, mentre que la degradació de prestacions és mínima respecte a una cache convencional SRAM.

List of Figures

List of Tables

Abbreviations and Acronyms

Adp	**Ad**a**p**tive
AIP	**A**ccess **I**nterval **P**redictor
Alw	**Alw**ays
BIP	**B**imodal **I**nsertion **P**olicy
BRRIP	**B**imodal **R**e-**R**eference **I**nterval **P**rediction
Cond	**Cond**itional
Conv	**Conv**entional
DC	**D**ivide-and-**C**onquer
DIP	**D**ynamic **I**nsertion **P**olicy
DRAM	**D**ynamic **R**andom-**A**ccess **M**emory
DRRIP	**D**ynamic **R**e-**R**eference **I**nterval **P**rediction
ECC	**E**rror **C**orrecting **C**odes
EDAP	**E**nergy-**D**elay-**A**rea **P**roduct
eDRAM	**e**mbedded **D**ynamic **R**andom-**A**ccess **M**emory
ED^2P	**E**nergy-**D**elay squared **P**roduct
FP	**F**loating-**P**oint
IPC	**I**nstructions **P**er **C**ycle
Int	**Int**eger
LIP	**L**east Recently Used **I**nsertion **P**olicy
LLC	**L**ast-**L**evel **C**ache
LRU	**L**east **R**ecently **U**sed
LvP	**Liv**e-time **P**redictor
L1	First-level
L2	Second-level
MPKI	**M**isses **P**er **K**ilo-**I**nstruction

MRAM	**M**agnetic **R**andom-**A**ccess **M**emory
MRU	**M**ost **R**ecently **U**sed
MRUT	**M**ost **R**ecently **U**sed-**T**our
PRAM	**P**hase-change **R**andom-**A**ccess **M**emory
RRIP	**R**e-**R**eference **I**nterval **P**rediction
SRAM	**S**tatic **R**andom-**A**ccess **M**emory
SRRIP	**S**tatic **R**e-**R**eference **I**nterval **P**rediction
VC	**V**ictim **C**ache
WP	**W**ay-**P**rediction

Chapter 1

Introduction

This chapter introduces some concepts and presents the motivation for the work developed in this thesis. First, different semiconductor memory technologies are discussed, showing their advantages and shortcomings. Then, the concept of Last-Level Cache (LLC) is presented, and several issues related to these memories are discussed. Finally, a summary about how the rest of this dissertation deals with hybrid cache designs and their data management is given.

1.1 Background

Computer architects have implemented cache memories [1] since late 1960s to mitigate the huge gap between processor and main memory speed. Caches have been normally 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 [2]. Thus, many leakage reduction techniques have been proposed to mitigate this problem. Alternative technologies are also being considered. For instance, Dynamic RAM (DRAM) technology provides minimal leakage currents by design, but it has not been used to build caches because DRAM is difficult and expensive to implement in logic-circuit technology.

Current microprocessors implement two or three cache levels to reduce the difference in speed between processor and main memory. To reduce this penalty, LLCs are designed as large memory structures, which significantly increases leakage consumption. Due to this reason, LLCs are suitable for logic-compatible embedded DRAM or simply eDRAM technology. However, the main drawbacks of eDRAM are that its reads are destructive, it requires refresh operations, and it is not as fast as SRAM. Finally, researchers also have noted that the LRU replacement algorithm does not reach good performance in LLCs. One of the main reasons is that data locality in these memories is filtered by the inner cache levels.

1.1.1 Memory Technologies

1.1.1.1 Static Random-Access Memory (SRAM)

Until now, SRAM has been the predominant technology used to build memory cells in computer systems. SRAM cells are typically implemented with six transistors (6T cells). Figure 1.1 illustrates the structure of a 6T SRAM cell. These cells are usually designed for speed, which is the main reason because of they are used in cache memories, especially in the first levels of the memory hierarchy. However, the major drawbacks of SRAM-based caches are that they occupy a significant percentage of the overall die area in current multicore processors and consume an important amount of energy, especially

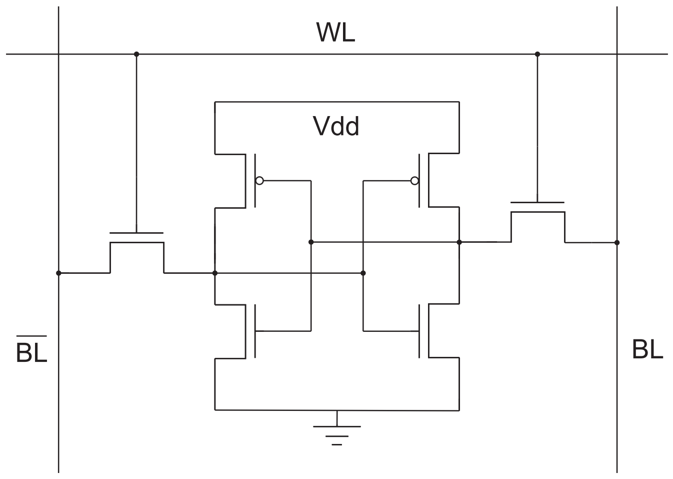


Figure 1.1: 6T SRAM cell. Labels *WL* and *Vdd* refer to the *wordline* and supply voltage, respectively, whereas labels *BL* and */BL* refer to the *bitline* and its complementary, respectively.

leakage energy which is proportional to the number of transistors. Furthermore, this design concern aggravates as the transistor size continues shrinking. Therefore, using SRAM cells impacts both in area and leakage currents.

1.1.1.2 Leakage Reduction Techniques for SRAM

Leakage energy reduction in SRAM cells has been widely investigated in the last decade. The proposed techniques can be classified into two main categories depending on whether the cache block state is preserved or not. In the first category, the supply voltage to selected cache lines is reduced and these lines remain in state-preserving low-power mode, so increasing the access time to such lines [3] [4]. In the second group, the supply voltage to the cache blocks with poorer locality is removed; thus, losing the block information [5] [6]. Subsequent accesses to such blocks will result on a cache miss; thus, the next level of the memory hierarchy must be accessed.

1.1.1.3 Dynamic Random-Access Memory (DRAM)

Leakage currents can also be reduced taking into account alternative technologies like DRAM, which is typically used for main memory. Figure 1.2 depicts the structure of a 1T-1C DRAM cell. Unlike SRAM cells, DRAM cells only require an active power supply during the memory access so their leakage currents are reduced by design. For

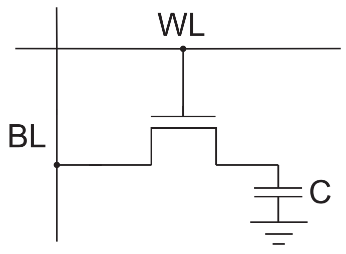


Figure 1.2: 1T-1C DRAM cell. Label C refers to the capacitor of the cell.

a given technology node, these cells require less area than SRAM cells since they are implemented with only one capacitor and the corresponding pass transistor. However, these cells present two major drawbacks that make them inappropriate for processor caches: low speed and information loss.

Regarding the first issue, technology advances have permitted to embed DRAM cells using CMOS technology [7]. An eDRAM cell integrates a trench DRAM storage cell into a logic-circuit technology and provides similar access delays as those presented by SRAM cells. As a consequence, some recent commercial processors such as some IBM POWER processors use eDRAM technology to build huge second-level (L2) caches or LLCs [8] [9] [10] [11].

The latter drawback is caused because capacitors in both DRAM and eDRAM cells store data as charge, which is lost either when contents are read (destructive read) or progressively with time. The elapsed time since the capacitor is charged until the contents are lost is referred to as retention time. To avoid capacitor discharges, contents are periodically read out and written back in a process known as refreshing. Refresh operations consume additional energy and affect performance since they compete for memory with regular processor requests. This is a major design concern in first-level (L1) caches, where data availability is critical. On the other hand, the refresh energy problem is expected to aggravate in future technologies with much higher memory storage capacities. For instance, as reported in [12], refresh consumption is expected to rise up to 50% in future 64Gb DRAM chips.

An important advantage of eDRAM technology is that it can be manufactured by logic technologies with minimal changes in the manufacturing process. This means that SRAM and eDRAM technologies can be mingled in the same die to obtain the best

Technology	Speed	Density	Leakage	Refresh logic	Destructive reads	logic-circuit compatible
SRAM	fast	low	high	no	no	yes
DRAM	slow	high	low	yes	yes	no
eDRAM	slow	high	low	yes	yes	yes

Table 1.1: Memory technology characteristics.

of each technology as already performed by some companies [13]. Table 1.1 summarizes the main design characteristics of the discussed memory technologies.

1.1.1.4 Other Memory Technologies and Cells

Apart from the technologies discussed in this chapter, several technologies are being developed, especially non-volatile technologies, such as Magnetic RAM (MRAM) and Phase-change RAM (PRAM or PCM) [14] [15] [16].

The non-volatile property of these technologies allow them to consume less leakage than eDRAM. MRAM and PRAM also provide high density, their reads are not destructive, and they do not require refresh operations. However, manufacturing constraints prevent from mixing them with logic-circuit technology using conventional two-dimensional (2D) chips. In addition, the low speed and dynamic energy consumed by these technologies, in particular for write operations, suggest that they are more appropriate for main memory storage instead of caches.

Other research works have focused on the design of new *DRAM-like* cells for caches. Liang *et al.* [17] proposed the 3T1D (three transistors and a diode) DRAM cell. The speed of this cell is comparable to the speed of 6T SRAM cells. Thus, 3T1D cells can be used for critical latency structures such as L1 data caches. However, although reads are non-destructive, the diode charge get lost over time, requiring from refresh schemes that might have a severe impact on performance. The 3T1D cell can be smaller than the 6T SRAM cell but, the smaller the cell size the lower the retention time of the diode capacitance.

Juang *et al.* [18] proposed a dynamic cell from a 6T SRAM cell which does not include the two transistors connected to Vdd that restore the charge loss due to leakage currents. Thus, the circuit results in a non-static cell with only 4 transistors (the quasi-static 4T cell). This cell offers an easy method for DRAM implementation in a logic process production, especially in embedded systems. Compared to 6T SRAM cells, the 4T cells require less area. In contrast, the data access is a bit slower and destructive. Like in the 1T-1C cell, this problem can be solved by re-writing the read data immediately after the read operation or before the retention time expires.

1.1.2 Last-Level Caches

LLCs are designed as very large memory structures 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 [11] [19]. In addition, in order to keep low the number of conflict misses, current LLCs implement a high number of ways (e.g., 16 or more ways).

1.1.2.1 Replacement Algorithms

Typically, cache memories exploit temporal locality by implementing the Least Recently Used (LRU) replacement algorithm. This algorithm acts as a stack that places the Most Recently Used (MRU) block on the top of the stack (MRU position) and the LRU block, which is the evicted block when space is required, on the bottom (LRU position). Although this algorithm works well in L1 caches with a low number of ways; with high associativities, like 8 and 16 ways that are currently found in LLCs, strict LRU is too expensive to implement. Therefore, approximations to LRU are the norm in commercial processors but their performance start to deviate from the strict LRU [20].

On the other hand, the performance of the LRU algorithm in LLCs is quite far from the optimal replacement strategy referred to as Belady's algorithm [21]. There are several reasons that explain why the LRU algorithm does not work well in LLCs with high associativity. First, most accesses that hit in the inner caches are hidden to LLC so information about temporal locality is lost. Second, LRU suffers from thrashing effects in those workloads whose working set is greater than the available cache size, resulting in cyclic accesses to blocks that *walk* through the stack without being reused. Third, LRU forces a block to descend down to the bottom of the stack before eviction, which may severely impact on performance since most blocks that are brought into the LLC are not referenced again once they leave the MRU position (see Section 4.2). For example, in a

16-way cache, a given block that leaves the MRU position and is not referenced again will not be evicted until fifteen distinct blocks mapping to the same set are accessed. This means that, instead, some useful blocks might be removed from the cache set so hurting the performance. Recent research works have focused on how to improve this shortcoming by predicting when a block can be evicted while it is still walking the LRU stack to the bottom [22] [23] [24].

The following replacement policies are some of the most representative state-of-the-art alternatives to the LRU policy in LLCs to date, and they will be compared against the policy proposed in this thesis. Other replacement strategies can be found in Section 4.6.

The Bubble algorithm [25], proposed by Zhang and Xue, unlike the LRU scheme, uses a queue instead of a stack that works as follows. An incoming block is allocated at the bottom of the queue, which is the location with the lowest access frequency. Anytime a block hits again, it is promoted one-position upwards the queue. In this way, the blocks closer to the top of the queue evince a higher access frequency than those closer to the bottom. When there is a lack of space, the block to be evicted is selected either from the bottom or the top of the queue, depending on whether the previous access to that set resulted in a cache miss or a cache hit, respectively. This work also presents a Divide-and-Conquer technique referred to as DC-Bubble, which divides the blocks in each cache set into independent groups, so that each group has its own replacement logic. In this scheme, when a block is fetched, the target group within the set is randomly selected. These schemes exploit both recency of information and frequency information and adapt to changes in the working set.

In [26], three adaptive insertion policies based on LRU are proposed. The first one, referred to as LRU Insertion Policy (LIP), inserts all incoming blocks in the LRU position, and then they are promoted to the MRU position if they are referenced again. This behavior prevents LIP from the effect of cache thrashing. The second one, namely Bimodal Insertion Policy (BIP), differs from LIP in that every x cache misses, the incoming block is inserted in the MRU position. This policy adapts to changes in the working set and, like LIP, provides thrashing protection. Finally, the third policy, referred to as Dynamic Insertion Policy (DIP), dynamically combines LRU and BIP using the Set Dueling strategy. DIP uses a small fraction of the cache sets to measure the performance of each policy, and applies to the remaining sets the policy that achieves the best performance.

Jaleel *et al.* [27] propose a family of algorithms based on Re-Reference Interval Prediction (RRIP) of cache blocks to deal with cache thrashing and bursts of accesses to non-temporal data. The simplest version, called Static RRIP (SRRIP) policy, uses a saturating counter per cache block to predict whether the block will be re-referenced sooner or later in the future. On a cache insertion, the distance prediction of the incoming block is set to be in the *distant future* (i.e., the counter is set to its maximum value), while subsequent accesses to this block reduce its distance prediction to be in the *near future* (i.e., decrease the counter by one each time the block is accessed). The victim block is selected among those blocks predicted to be accessed in the distant future. If there are not candidates, all the counters are increased by one until one of them saturates. An enhancement of SRRIP is the Bimodal RRIP (BRRIP) policy. It differs from SRRIP in that every x cache misses the counter of the incoming block is set to its maximum value minus one. Finally, both SRRIP and BRRIP policies were used together using Set Dueling as done in [26], resulting in the Dynamic RRIP (DRRIP) policy.

1.1.2.2 Refresh Mechanisms

Prior research works have concentrated on reducing the refresh energy by avoiding unnecessary refresh operations in off-chip DRAM memories. Regular read accesses to memory implicitly trigger a refresh operation since DRAM contents are written back after they are read. Based on this fact, some works have focused on delaying periodic refreshes of frequently requested data [28]. Other works take into account the inter-cell variation in retention time to enhance the refresh logic in order to adapt the refresh period for each memory row [12] [29] [30]. Error-Correcting Codes (ECC) have been also used to recover lost data due to extended refresh periods [31]. Finally, refresh can be skipped for data identified as *useless* [29]. Please refer to Section 5.3 for further information.

Like in their counterparts off-chip DRAM memories, refresh operations in eDRAM LLCs represent an important fraction of the total dynamic energy consumption of these devices as shown in Figure 1.3[1]. As observed, refresh energy increases with the cache capacity

[1]These results have been obtained with the machine parameters and methodology presented in Section 5.2.

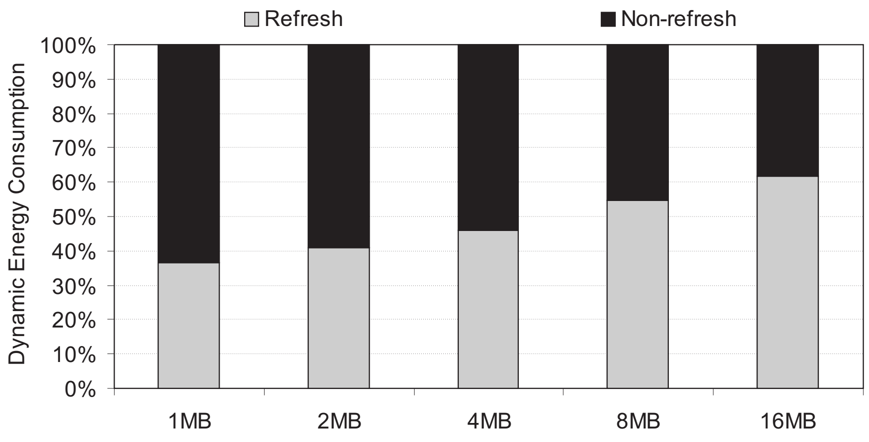


FIGURE 1.3: Dynamic energy split into expenses due to refresh and non-refresh operations in conventional eDRAM LLCs.

and grows up to 62% for a 16MB cache. This is because more cache lines have to be refreshed for a given retention time, which in eDRAM caches is by thousand times shorter than in off-chip DRAM memories [32].

Due to their different characteristics, the refresh techniques proposed for off-chip DRAM devices are not suitable for on-chip eDRAM caches. First, the access time among external DRAM Dual In-line Memory Modules (DIMMs) and the next level of the hierarchy (e.g., disks) is at least six orders of magnitude higher (from ns to ms), thus less aggressive techniques should be used since the misspeculation penalty (i.e., a useful memory row is not refreshed and then requested) is much higher. In contrast, techniques especially designed for on-chip caches can be much more aggressive. Second, main memory is not organized as a cache, so techniques such as way-prediction cannot be applied. Third, external memory works at coarse (row or page) granularity, typically a couple of KB. Instead, on-chip caches work with relatively smaller data blocks (e.g., 64 bytes).

The refresh problem in eDRAM caches has been previously attacked taking into account inter-cell feature variations [32] [33]. These works are orthogonal to the selective refresh proposed in this thesis. In [33], Emma *et al.* proposed to learn the appropriate refresh period from each cache set via a regressive process. Initially, this process assumes the worst-case refresh period for the entire cache. Then, refresh periods are increased step by step until ECC detect data losses. In this way, the best refresh period for each cache set is detected.

Wilkerson *et al.* [32] proposed an ECC optimization to identify expired data due to enlarged refresh periods. This approach provides both single-bit and multi-bit failure detection. The single-bit error can be corrected, while those sections of the cache with multi-bit errors are disabled to avoid the high latency and complexity of multi-bit error correction.

Another recent work [34] delays periodic refreshes taking into account the implicit refresh of regular accesses. In addition, authors propose a refresh policy that makes use of a 5-bit counter per cache line. The counter is set to its maximum value when the line is accessed or written back, and it is decremented on each periodic refresh to the line. When the counter reaches zero, the line is written back and set to valid if dirty, or invalidated if non-dirty. This mechanism requires more hardware complexity and area than the single-bit per line refresh policy proposed in this dissertation.

1.2 Objectives of the Thesis

The main objective of this dissertation is leveraging SRAM and eDRAM technologies in order to build a hybrid cache hierarchy that optimizes performance and reduces energy consumption. In addition, architectural design issues like refresh mechanisms and replacement algorithms are tackled in both L1 data caches and L2 caches acting as LLCs. Compared to a conventional SRAM cache, the performance degradation of hybrid caches is minimal because the most likely data blocks to be referenced are stored in *fast* SRAM technology, while eDRAM technology is used to store the remaining blocks so providing significant leakage and area savings.

For LLCs, based on the observation that most cache blocks are not referenced again once they leave the MRU position of the LRU stack, a set of replacement algorithms aware of this property are proposed to reduce hardware complexity and to obtain better performance than the traditional LRU policy. Finally, regarding the refresh problem, a selective refresh mechanism is devised. Compared to a conventional refresh method, the proposed selective refresh reduces the number of periodic refreshes by skipping refresh operations in those blocks that are candidates for eviction.

1.3 Contributions of the Thesis

The four major contributions of this thesis are described below:

- A hybrid eDRAM/SRAM cache architecture for L1 data caches is developed. Both semiconductor technologies are mingled at cell level to form an n-bit *macrocell*, which consists of one SRAM cell, *n-1* eDRAM cells, and *n-1 bridge* transistors that communicate the SRAM cell with the corresponding eDRAM cells. The macrocell is used to build the data array of n-way set-associative hybrid caches. These memories implement one SRAM way and *n-1* eDRAM ways. Architectural mechanisms are devised to maintain the MRU data in the fast SRAM way and to completely avoid refresh logic.

- Since the macrocell design could be expensive to implement in high-associative caches, this thesis combines both SRAM and eDRAM technologies at bank level to build hybrid L2 caches. Like in the hybrid L1 cache, the MRU contents are stored in SRAM technology. Since temporal locality is much less predictable in L2 caches, the optimal ratio of SRAM and eDRAM ways is explored to achieve the best trade-off among performance, energy, and area.

- A family of low-cost replacement algorithms for LLCs is introduced, which exploits reuse information by leveraging the concept of MRU-Tour (MRUT). The number of MRUTs of a block is defined as the number of times that the block is placed in the MRU position while it resides in cache. In this dissertation it is shown that most blocks are not accessed again once they leave the MRU location. Thus, these policies select as candidates for eviction those blocks having a single MRUT.

- A selective refresh policy to minimize the number of periodic refresh operations is proposed. The devised mechanism aims to avoid energy wasting due to refreshing useless blocks, which are identified by exploiting the MRUT concept. The selective refresh policy does not refresh those blocks that have experienced one MRUT, and it is applied in both pure eDRAM and hybrid eDRAM/SRAM L2 caches.

1.4 Thesis Outline

This dissertation is composed of six chapters. Chapter 2 and Chapter 3 introduce the hybrid eDRAM/SRAM architecture for L1 and L2 caches, respectively. Chapter 4 presents the MRUT replacement algorithm. Chapter 5 describes the selective refresh mechanism, and finally, Chapter 6 summarizes this thesis, discusses future work, and enumerates the related publications.

Chapter 2

Hybrid eDRAM/SRAM L1 Data Cache Architecture

This chapter introduces the hybrid eDRAM/SRAM architecture for L1 data caches. First, the implementation of the macrocell and the architecture design issues are presented. Then, experimental results including performance, energy consumption, and area are discussed.

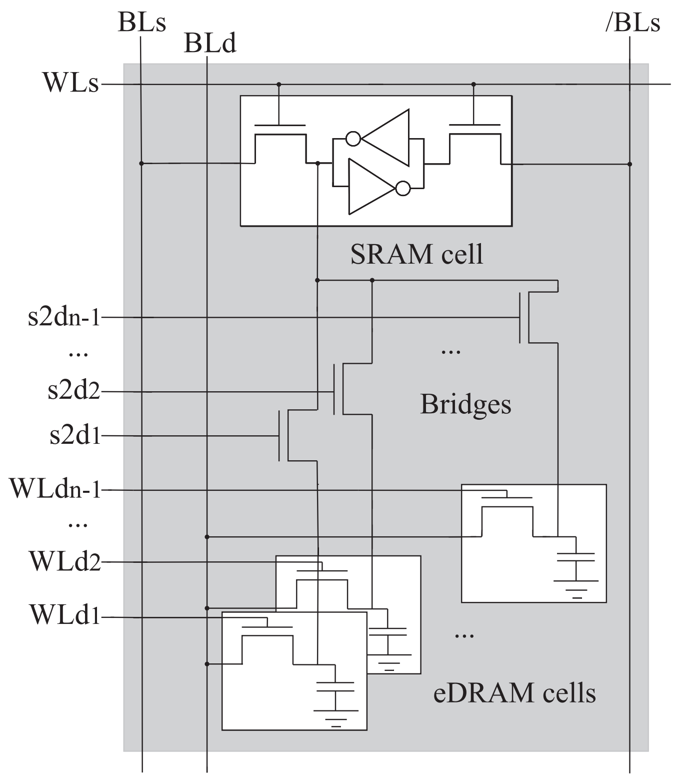


FIGURE 2.1: Block diagram of an n-bit macrocell.

2.1 Memory Cell Proposal

The main components of an n-bit macrocell are a typical SRAM cell, *n-1* eDRAM cells, and *n-1* bridge transistors communicating the SRAM cell and the corresponding eDRAM cell. Figure 2.1 depicts the implementation of an n-bit macrocell. The *static part* is limited to only one SRAM cell mainly due to leakage and area reasons. It has the same structure as a typical 6T SRAM cell. Thus, read and write operations in this part are managed like in a typical static cell through the bitline (*BLs*) and its complementary (*/BLs*).

The *dynamic part* is composed of *n-1* eDRAM cells, each one working like a typical 1T-1C DRAM cell. Each capacitor has an NMOS pass transistor (controlled by WLd_i) that keeps the capacitor charge insulated from the bitline (*BLd*). Like in the static part, read and write operations are performed as in a conventional eDRAM cell through the corresponding pass transistor.

Information movements among SRAM to eDRAM cells imply two main steps in a conventional design: i) read the SRAM cell content and write it to an intermediate buffer and ii) read the buffer and write its content to the eDRAM cell. The main novelty of the proposed design is that bridge transistors enable unidirectional transfers of data from the SRAM cell to the eDRAM cells. Each bridge transistor is controlled by a distinct $s2d_i$ signal. In this way, the information state in the SRAM cell can be copied directly to one of the capacitors, that is, no intermediate buffer is required. These transfers will be referred to as internal since no bitline is involved.

Notice that implementing a cache using macrocells instead of splitting the L1 cache in two different organizations (a standard SRAM cache and an independent eDRAM cache) has important advantages. First, some resources (e.g., decoder and wordline) can be shared. Second, when using two independent caches, the internal transfers (i.e., moving a block from static to dynamic part) involve more circuitry and wire delays, which might prohibitively increase latency and energy dissipation.

In order to check the correctness of the electronic behavior, the proposed cell has been modeled in NGSPICE, a Berkeley's Spice3f5-based circuit simulator. NGSPICE allows MOSFET behavior to be accurately simulated since it uses BSIM4 MOSFET model. All simulations used the Predictive Technology Models (PTM) [35]. Transistor features have been taken from the 2007 ITRS [2] for a 45nm technology node.

Two main design issues were addressed to ensure the correct functionality of the proposed cell: i) to check that the capacitor is properly charged and ii) the absence of flips when moving data from the SRAM cell to an eDRAM cell. Regarding the former issue, the main problem in a typical 1T-1C cell is the voltage degradation when writing a logic '1' to the capacitor. This is due to NMOS pass transistors incur a voltage drop equal to Vth when they transfer a logic '1'. Thus, in order to charge the capacitor to the maximum Vdd voltage, wordlines are usually boosted to a voltage $Vpp = Vdd + Vth$. In eDRAM cells, Vth and Vdd are normally set to 0.4 and 1.1V, respectively, for a 45nm technology node. Regarding the macrocell, the wordlines controlled by WLd_i and $s2d_i$ must be also boosted.

Concerning the second design issue, read operations in conventional SRAM cells must be preceded by precharging high both bitlines. Precharge operations are necessary to optimize the cell speed, area, and stability relationship [36]. This is mainly due to the different features of NMOS and PMOS transistors. In this way, flips are avoided inside the cell since NMOS transistors are stronger (i.e., they can drive more current) than

PMOS transistors. Analogously, to prevent flips in the proposed design, the capacitor of the eDRAM cell must be precharged to Vdd.

Figure 2.2 illustrates how the internal transfer operation between the SRAM cell and three eDRAM cells works in a 4-bit macrocell, highlighting both the precharge process and how flips are avoided[1]. Two internal transfers labeled as *eDRAM1* and *eDRAM2* are shown for illustrative purposes, both writing a ’0’ and a ’1’.

The number of bits in the macrocell device defines the number of ways of macrocell-based caches. In other words, n-bit macrocells are required to implement an n-way set-associative hybrid cache. Hence, these memories will have one way built with SRAM cells (SRAM way) and *n-1* ways implemented with eDRAM cells (eDRAM ways).

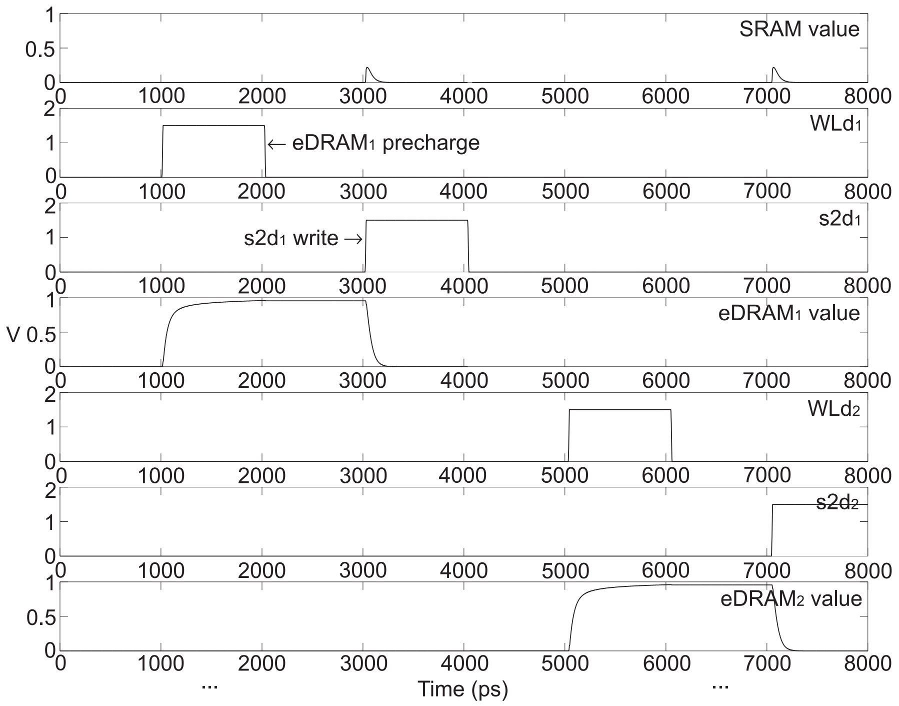
The proposed macrocell reduces both leakage and area. Since leakage is mainly produced by SRAM cells, the macrocell reduces leakage proportionally to the number of implemented eDRAM cells. For instance, let’s assume a 4-bit macrocell that has one SRAM cell and three eDRAM cells; ideally, as only the SRAM cell has leakage, the 4-way hybrid cache would achieve 75% (i.e., 3/4) less leakage than a conventional cache with the same capacity implemented with SRAM cells (see Section 2.3.3 for a deeper and accurate analysis). On the other hand, area is also reduced since eDRAM cells are implemented with less transistors than SRAM cells (see Section 2.3.4). Finally, notice that the higher the cache associativity degree, the higher the leakage savings and area reduction achieved with macrocells.

As read operations in an eDRAM cell are destructive and capacitors progressively lose their charge with time, refresh actions are normally required. In this context, new architectural innovations are presented to avoid extra logic and energy dissipation due to refresh operations.

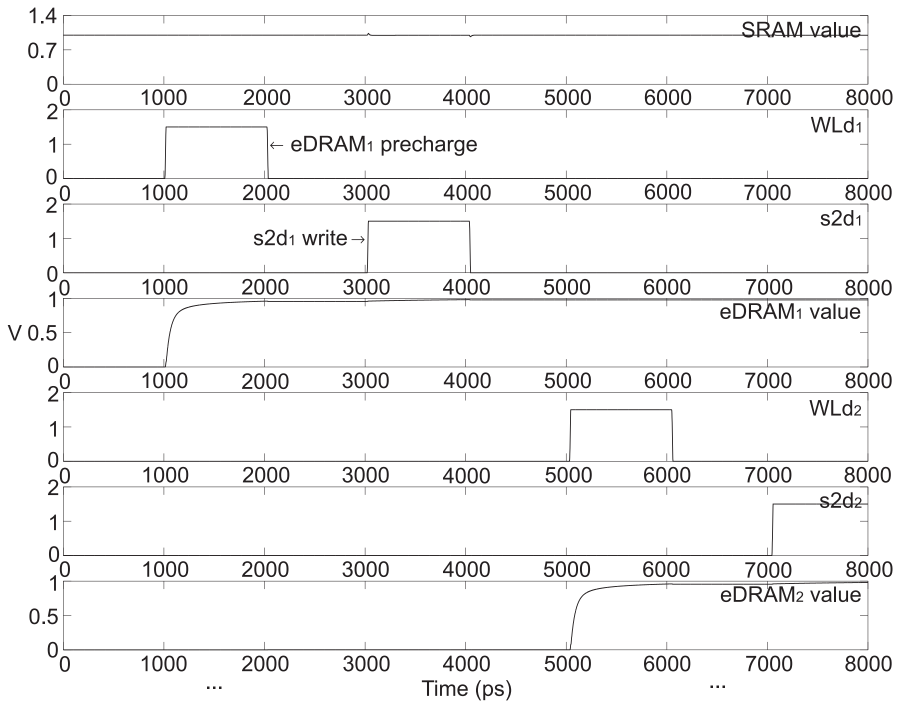
2.2 Architectural Design Issues

As mentioned above, the proposed n-bit macrocell is designed in order to implement the data array of an n-way set-associative cache, with one SRAM way and *n-1* eDRAM

[1]Bridge transistor features have been assumed the same as the pass transistors of the eDRAM cells, whose channel length and width are 45nm and 90nm, respectively.



a) Transfering a logic '0'



b) Transfering a logic '1'

FIGURE 2.2: Static to dynamic write operation details.

ways. From now on, it is assumed that the SRAM cells implement the *way-0* of the cache.

The tag array of hybrid caches is assumed to be built with SRAM cells, since it is much smaller than the data array. Thus, much lower energy and area benefits can be obtained in this structure, and implementing it with eDRAM technology would significantly affect the access time.

The architectural proposal devised to manage this cell consists of three main strategies, detailed below: i) accessing the eDRAM cells only in the case of a hit in the corresponding tag, ii) increasing the percentage of hits in the SRAM blocks in order to minimize the number of data movements between SRAM and eDRAM cells, and iii) minimizing the number of writebacks to L2.

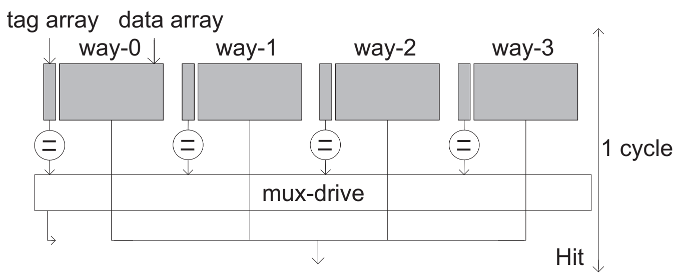
2.2.1 Accessing the eDRAM Ways

Typically, to achieve high performance, modern microprocessors overlap the access to the tag and data arrays. In this way, the requested data can already be available in the case of a successful tag comparison (i.e., a cache hit). Figure 2.3(a) shows the accessed parts (dark boxes) in a 4-way cache. The narrow box in the left of each way represents the tag array and the right box represents the data array. This working behavior is not adequate for data arrays implemented with macrocells, since regardless of the result of the access (i.e., cache hit or cache miss), reading the eDRAM cells is destructive and the design does not include refresh logic. Due to this reason, architectural solutions have been devised to avoid that cache blocks lose their contents.

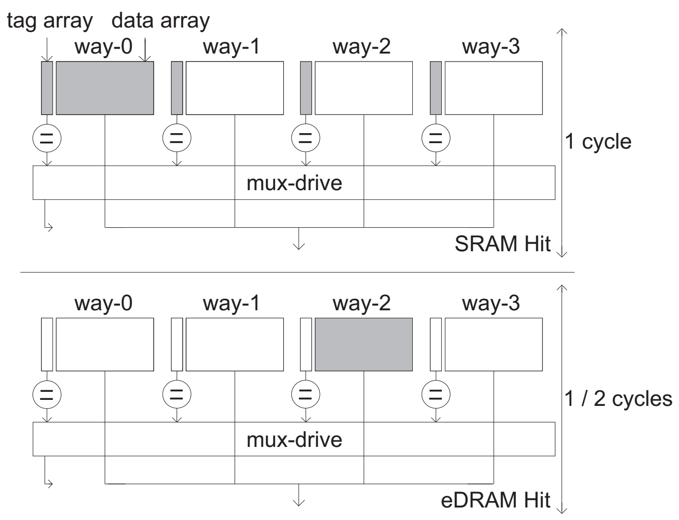
In order to guarantee correct execution, the problem is not that a given cache block loses its state with time but that the state becomes unrecoverable. In other words, the information loss is not a problem for correctness when it can be recovered from another location, for instance, from the L2 cache.

To deal with this problem, the devised solution has two main design goals: first, to reduce the number of state losses due to reads, and second, to provide a recoverable location without negatively impacting on performance.

To achieve the first design goal, the tags of all ways in the set are accessed in parallel with the data array of the block located in way-0 (see the upper side of Figure 2.3(b)). On a hit



a) Conventional cache



b) Hybrid cache

FIGURE 2.3: Access and timing of conventional and hybrid caches.

in way-0 (SRAM hit), the processor is satisfied and no eDRAM cell is subsequently read. This access could obtain the hit speed of a direct-mapped cache since the multiplexer control signals could be set early as done in way-prediction techniques [37] [38]. On a miss in way-0 but a hit in other tag associated to an eDRAM block, the corresponding data way is accessed subsequently. To illustrate this case, the lower part of Figure 2.3(b) represents a hit in way-2 (eDRAM hit). In this case, the contents are read and sent to the processor. However, since this operation is destructive, the capacitor state is lost. A straightforward solution to address this drawback is to implement a write-through policy so that a copy of the data can be always read from L2 if the capacitor is discharged. However, this would lead to a huge energy wasting in L2 caches as well as a poor performance.

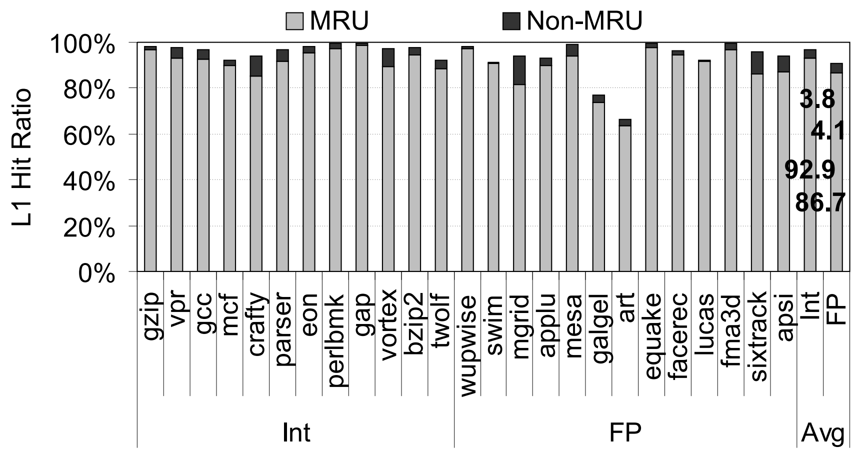


FIGURE 2.4: L1 hit ratio (%) in a conventional 64B-line 16KB-2way cache.

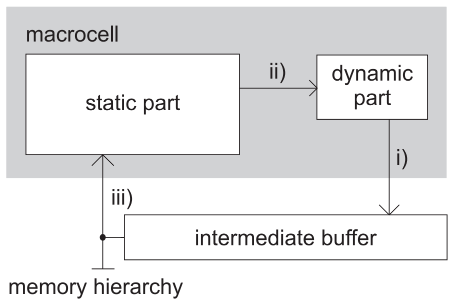


FIGURE 2.5: Swap operation details.

2.2.2 Increasing the Percentage of Hits in the SRAM Way

Some previous works [4] concluded that, because of data locality, in a 2-way set-associative L1 cache, most accesses hit the MRU blocks while few of them hit the non-MRU blocks. This observation can be appreciated in Figure 2.4[2], and it can be generalized to caches with higher associativities as shown in Section 2.3.2. Based on this assert, a simple *way-predictor* could be implemented by predicting the MRU way.

To increase the percentage of hits in the fast SRAM way of hybrid caches, the previous property is exploited by modifying the cache controller in order to keep the MRU block always in the SRAM way. To this end, the cache controller has been enhanced in order to internally manage a swap between eDRAM and SRAM cells in the case of an eDRAM hit. The swap operation consists of three main steps as shown in Figure 2.5. First, the contents in the eDRAM cells are transferred to an intermediate buffer. Then, the

[2]Details about the simulation environment can be found in Section 2.3.1.

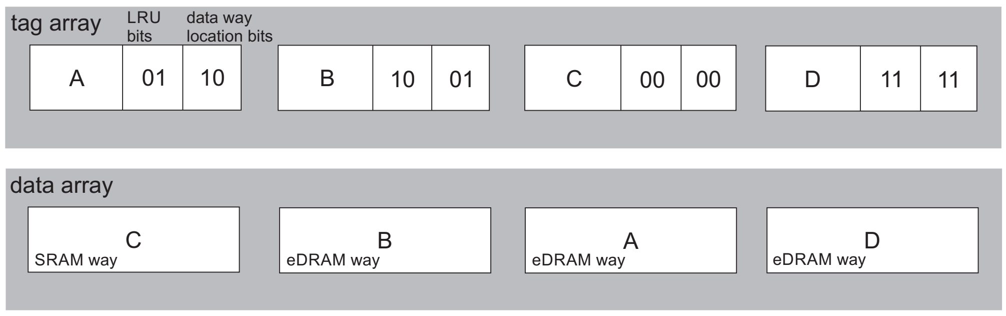


FIGURE 2.6: Diagram of a 4-way cache with the control information to maintain the mapping between tags and data blocks.

data stored in the SRAM cells are internally transferred to the eDRAM cells previously accessed. Finally, the contents from the intermediate buffer are written to the SRAM cells. If each step takes one cycle, the controller will be busy for three processor cycles. However, the processor can get the requested data earlier from the intermediate buffer. On a cache miss, a unidirectional transfer is triggered to move the data from the SRAM cells to the eDRAM cells that hold the data selected to be victimized by the LRU replacement algorithm. Then, the requested data, which comes from a lower level of the memory hierarchy instead of from the intermediate buffer, are stored in the SRAM cells.

Notice that writes in the eDRAM cells are always performed as internal movements from the SRAM cells and they are not always triggered by store instructions. They may be caused by a hit (either a load or a store) in the eDRAM cells.

The hybrid design allows capacitors to lose their charge with time, but those capacitors storing useful information (i.e., those cells accessed on a hit) will not need to be recharged since their information will be saved in the SRAM cells.

Finally, it has been assumed that the tag array is built with typical 6T memory cells, so tags do not need to be swapped. Therefore, control information may be required to track the mapping between tags and the associated data blocks. In particular, for a 4-way cache, two additional control bits (*data way location*) apart from the LRU bits are required as depicted in Figure 2.6. Notice that the LRU bits keep the access order of blocks, while the additional bits indicate the way in the data array where the associated block is stored. On the other hand, for a 2-way cache, no additional control bits are required, since in this case the LRU bit distinguishes between the SRAM and the eDRAM way. Remark that accessing these control bits is not in the critical path since they are read together with the tag array and the SRAM way.

2.2.3 Devised Writeback Policies

Although the architectural choices discussed in the previous sections avoid many capacitor state losses, they do not guarantee data preservation. Information loss may happen when a given dirty block stored in eDRAM cells is rarely accessed. Thus, incorrect execution could occur when accessing this block after its retention time expires. To deal with this shortcoming, two new writeback policies have been devised. These policies will be referred to as *early* writeback and *delayed* writeback.

The early writeback policy avoids the problem by preventing dirty blocks to be stored in eDRAM cells. As a block can only be written into eDRAM cells due to an internal swap operation, this policy checks the state of the block located in way-0 (SRAM way) when a swap arises and, if dirty, the block is written back to L2. The main drawback of this policy is that it might increase the number of writes to L2 so turning on excessive energy wasting. The early writeback policy assumes that the *valid bit* of each eDRAM way is implemented as a 1T-1C cell, including a capacitor with less capacitance than the macrocell capacitors to work as a *sentry bit.* In this way, if the capacitor of the sentry bit is not discharged, the contents of the associated data bits will be correct. On the contrary, if the capacitor of the sentry bit is discharged, the macrocell may still contain valid data, but the design conservatively assumes that the associated data has expired, what is equivalent to a non-valid content.

The delayed writeback policy is aimed at minimizing the number of writebacks to L2. To this end, this policy allows that dirty blocks move to the eDRAM cells. Once moved, capacitors maintain their state as dirty during the retention time. In this case, if the block is accessed again within the retention time, the block is moved to the SRAM cells so preserving its state, otherwise the state is lost. To avoid the latter case, the proposed solution is to access each block periodically (*scrubbing*) before the retention time expires, and if it is dirty, the block is written back to L2. This idea can be implemented with a single global binary counter [5] [17] for all the eDRAM blocks. The counter is decreased every cycle, and each time the counter reaches zero, an eDRAM block accessed following a given criterion (e.g., round-robin) is checked to be written back. The counter must be initialized to the retention time (in cycles) divided by the number of eDRAM blocks. Since this mechanism guarantees that every eDRAM block is checked before its retention time expires, the valid bit is not implemented as a sentry bit. Instead, this control bit is cleared each time the associated block is checked regardless of it is written back to L2 or not.

2.3 Experimental Evaluation

This section presents the simulation environment and experimental results of hybrid L1 caches implemented with macrocells. The presented results include an analysis of the hit ratio, performance degradation, retention time, and the impact of the devised writeback policies. In addition, leakage currents, dynamic energy, and area results are also estimated.

2.3.1 Simulation Framework

The macrocell-based hybrid caches have been modeled on top of an extensively modified version of the SimpleScalar simulation framework [39], which is a cycle-by-cycle trace-driven simulator that implements the microarchitecture of a superscalar processor. The simulation results include the execution time of the applications and the generated memory events (i.e., cache accesses, SRAM hits, eDRAM hits, misses, writebacks, and swaps) required to estimate leakage and dynamic energy, respectively. The CACTI 5.3 tool [40] [41] includes an analytical model for timing, energy, and area of caches and main memories, and it has been used to calculate the capacitances, retention time, leakage currents, dynamic energy per access type, and area for a 45nm technology node. The obtained values were used to feed the SimpleScalar simulator. The overall energy consumption was calculated combining the results of both simulators.

Experimental results were performed configuring the SimpleScalar for the Alpha Instruction Set Architecture (ISA) using the SPEC CPU benchmark suite [42]. Both Integer (Int) and Floating-Point (FP) benchmarks were run using the *ref* input set. Statistics were collected simulating 500M instructions after skipping the initial 1B instructions. Table 2.1 summarizes the main architectural machine parameters used throughout the experiments. These parameters and the presented simulation tools, ISA, benchmarks,

Microprocessor core	
Issue policy	Out of order
Branch predictor type	Hybrid gShare/Bimodal: gShare has 14-bit global history plus 16K 2-bit counters, Bimodal has 4K 2-bit counters, and choice predictor has 4K 2-bit counters
Branch predictor penalty	10 cycles
Fetch, issue, and commit width	4 instructions/cycle
ROB size (entries)	256
# ALUs	4 integer, 4 floating-point
Memory hierarchy	
L1 data cache	16KB, 32KB, and 64KB, 2-way and 4-way, 64 byte-line
L1 data cache access time	Split into hits in the static and dynamic parts: 1-cycle hit in the static part and x-cycle hit in the dynamic part, with $x = \{1, 2\}$ (see Section 2.3.2.2)
# L1 data cache banks	8
L2 unified cache	512KB, 8-way, 64 byte-line
L2 cache access time	10-cycle
Main memory access time	100-cycle

Table 2.1: Architectural machine parameters.

etcetera will be used to obtain all the experimental results presented in this thesis unless otherwise stated.

Multi-banked caches have been modeled to represent a more realistic approach. Each bank can support only one access at a time, but accesses to different banks can be performed concurrently. If a memory request refers to a bank that is already being accessed, that request must wait until the previous access (e.g., hit and writeback) finishes and the cache controller releases the corresponding bank. The waiting time also includes the time taken for a swap operation (if needed) in hybrid caches.

To reduce the likelihood of waiting due to bank contention, the cache sets are distributed among all the banks. Experimental results showed that, regardless of the cache scheme and organization, 8 banks provide less than 0.7% performance degradation with respect to an unlimited number of banks. Therefore, the number of banks has been set to 8 across all the experiments. This number of banks is reasonable and it is common to find other designs in the literature with more banks [43]. Finally, since one bank only can perform one swap operation at a given time, the number of banks also matches the number of intermediate buffers.

2.3.2 Performance

2.3.2.1 SRAM and eDRAM Hit Ratio

In an eDRAM-based cache, a read access requires a refresh operation since reads are destructive, even if the access results in a cache miss. Hence, to avoid *unnecessary* refresh operations in the hybrid design, eDRAM cells are only accessed after checking the tags. This section characterizes such accesses.

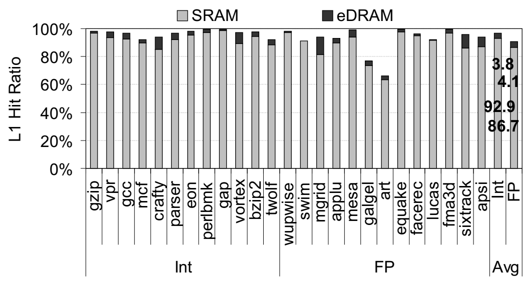
From now on, the terms *SRAM hit ratio* and *eDRAM hit ratio* are used to refer to the percentage of cache accesses that hit the SRAM and eDRAM cells, respectively. The sum of both gives the total L1 hit ratio. Figure 2.7 shows the results for different cache sizes (16KB, 32KB, and 64KB) and associativity (2 and 4 ways).

As an n-bit macrocell has only one SRAM cell, the storage capacity of the SRAM cells is the cache size divided by the number of bits of the macrocell, that is, divided by the number of cache ways. For instance, the static storage capacity of a 16KB-2way and a 32KB-4way cache is 8KB.

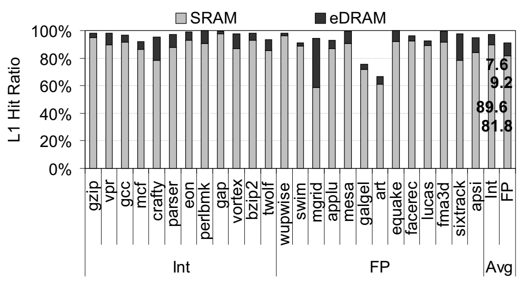
An important observation is that, due to the architectural mechanism that enforces the storage of the MRU block in the SRAM way, the number of hits in this way only depends on its storage capacity. For instance, the SRAM hit ratio of a 16KB-2way cache matches that of a 32KB-4way cache, since both caches have a static data array of the same size (i.e., 8KB). Hence, both arrays will have the same amount of misses, being the difference that the incoming block may be fetched from different memory structures (e.g., eDRAM cells or other level of the memory hierarchy).

Notice that for a given cache size, the higher the associativity degree the smaller the size of the SRAM way. Hence, although increasing the number of ways leads to a higher overall L1 hit ratio, the SRAM hit ratio decreases with the number of ways and the eDRAM hit ratio increases. In contrast, for a given associativity degree, enlarging the cache size results in a lower eDRAM hit ratio and a higher overall L1 hit ratio.

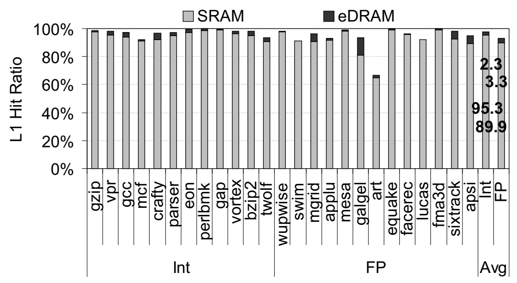
Results show the effectiveness of the devised architectural mechanisms since the eDRAM hit ratio is on average less than 4.1% for 2-way caches regardless of the benchmark type (Int or FP) and cache size. This value grows up to 9.2% for FP benchmarks in the 16KB-4way cache configuration, but in this case, the overall L1 hit ratio is also larger



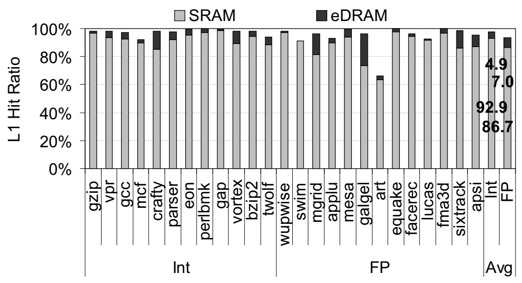
a) 16KB-2way



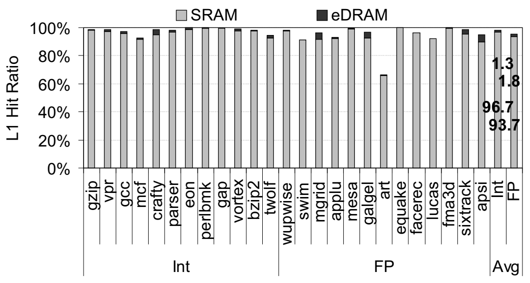
b) 16KB-4way



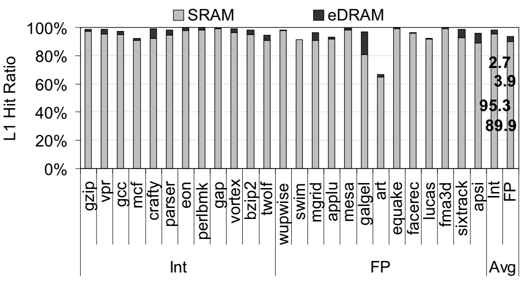
c) 32KB-2way



d) 32KB-4way



e) 64KB-2way



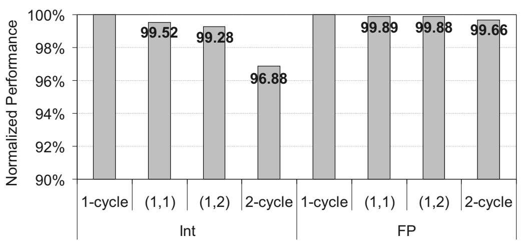
f) 64KB-4way

Figure 2.7: L1 hit ratio (%) for different hybrid cache organizations.

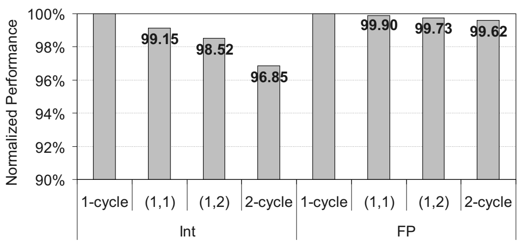
than in a 16KB-2way cache. Notice that a low eDRAM hit ratio is interesting due to two main reasons: first, the overall access time of the eDRAM cells requires additional processor cycles for tag comparison, and second, a hit in an eDRAM way incurs a swap between the static and dynamic parts.

2.3.2.2 Impact of eDRAM Access Time

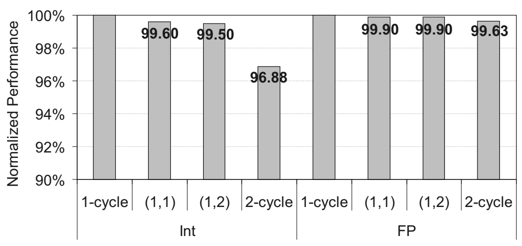
Results provided by CACTI indicate that an eDRAM cell can be accessed as fast as an SRAM, as also stated in [7], although writes can be slightly slower.



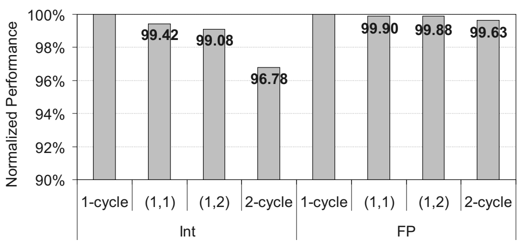
a) 16KB-2way



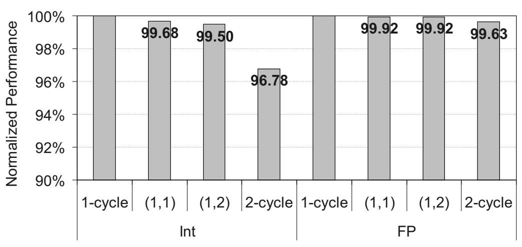
b) 16KB-4way



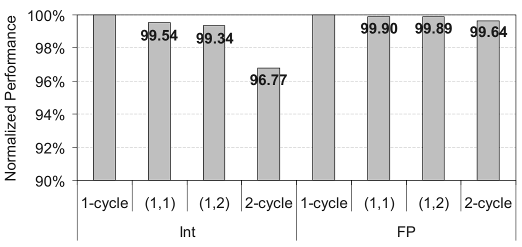
c) 32KB-2way



d) 32KB-4way



e) 64KB-2way



f) 64KB-4way

FIGURE 2.8: Normalized performance (%) with respect to the conventional cache for different access times (in processor cycles).

This section evaluates the impact on performance of the hybrid cache varying the access time of the eDRAM ways (1 and 2 cycles), while keeping constant (1 cycle) the access time of the SRAM way. Notice that a fast conventional cache with just 1-cycle access time imposes an upper-bound in performance (Instructions Per Cycle –i.e., IPC) to the proposal, since there is no performance loss.

Figure 2.8 plots the normalized performance with respect to the conventional cache with the same organization. For comparison purposes, the values of a 2-cycle access time conventional cache are also plotted. The X axis shows the access times of the different cache organizations. Conventional caches are labeled as *1-cycle* and *2-cycle*.

Access times of the hybrid caches are labeled as pairs *(1,x)*, where the first and second elements refer to the access time of the SRAM and eDRAM ways in processor cycles, respectively. Remember that a hit in an eDRAM way requires an additional processor cycle to check the tags.

The performance degradation of the hybrid cache mainly comes from the access to eDRAM ways and swap operations. Anyway, the performance degradation is always less than 2%. In contrast, the performance of the 2-cycle conventional cache drops down to about 3%. Regarding floating-point benchmarks, minor performance differences appear both in the slower 2-cycle conventional cache and in the proposed hybrid cache.

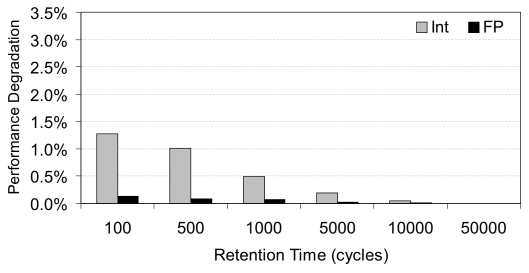
2.3.2.3 Impact of Retention Time

In the previous section, the performance degradation of the hybrid cache has been evaluated with respect to a conventional SRAM cache. In that study, a perfect capacitor with no charge loss has been assumed. Nevertheless, real capacitors lose their state after a given retention time. Therefore, accessing the eDRAM cells after this time requires the access to a lower level of the memory hierarchy to get the requested data; thus, adversely impacting on performance. This section explores the impact of the retention time on performance with respect to a hybrid cache with perfect capacitors.

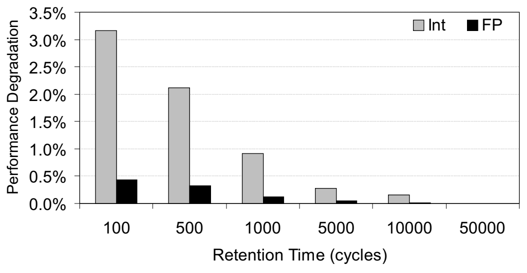
Figure 2.9 shows the results ranging the retention time from 100 to 50K processor cycles. Retention times longer than 50K cycles do not improve the cache hit ratio in any of the analyzed organizations; thus, no performance degradation can be observed.

For a given associativity degree, the larger the cache size (e.g., 64KB) the lower the performance losses (due to the larger static part of the cache). In contrast, for a given cache size, the higher the associativity degree the larger the performance degradation (due to the larger number of eDRAM hits –i.e., larger dynamic part). Finally, large dynamic parts require long retention times to reach the performance of the baseline.

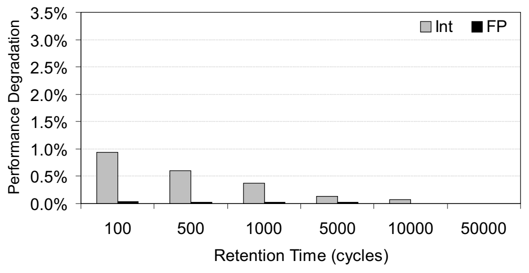
To sum up, the proposal just requires from capacitors that retain their charge for a few thousands of processor cycles in order to achieve its maximum performance. As obtained with CACTI, a relatively low capacitance of 10fF gives a retention time of about 63K processor cycles for a 1GHz processor speed. This is half the capacitance of typical eDRAM devices [40] [41]. Finally, notice that for a given retention time (in μs),



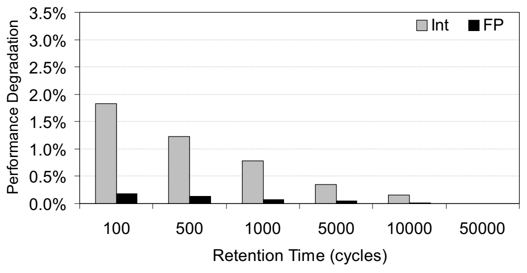
a) 16KB-2way



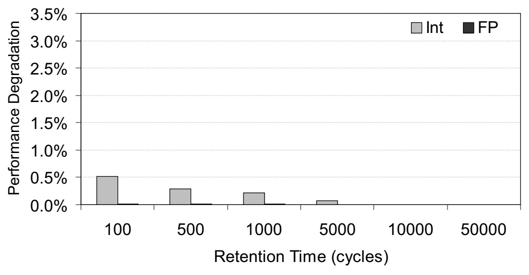
b) 16KB-4way



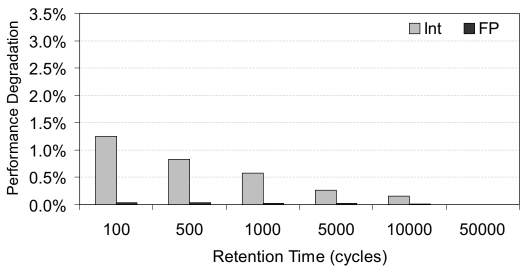
c) 32KB-2way



d) 32KB-4way



e) 64KB-2way



f) 64KB-4way

Figure 2.9: IPC losses (%) with respect to the perfect hybrid cache for different retention times (in processor cycles).

increasing the processor frequency implies smaller capacitances since the cycle time is reduced.

2.3.2.4 Writeback Policies

A writeback to L2 in a conventional writeback cache arises when a dirty block is evicted from the L1 cache. In addition to writebacks due to replacements, new types of writebacks can be identified in the devised policies. Regarding the early writeback policy,

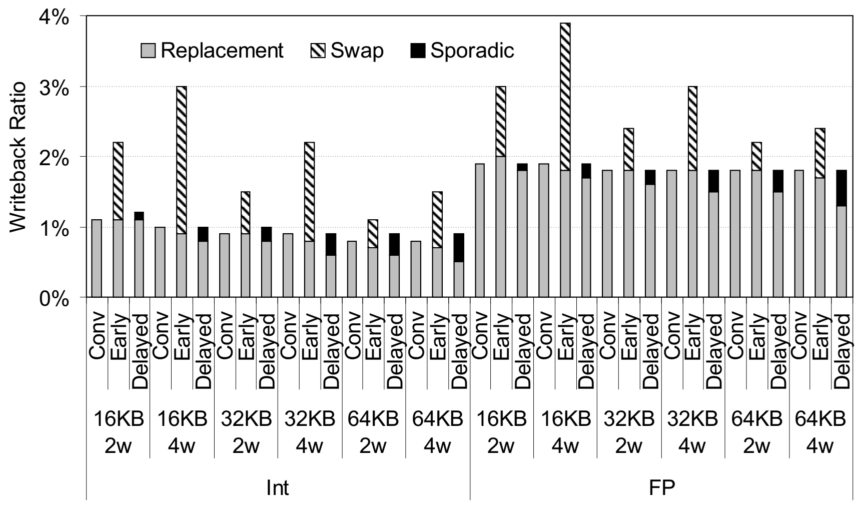


Figure 2.10: Writeback ratio (%) for the devised policies and the conventional cache.

two types of writebacks can be distinguished. First, the SRAM block must move to an eDRAM way in order to make room to the missing block. In such a case, if the SRAM block is dirty it must be written back to L2. In addition, a writeback can be also performed due to a swap risen as a consequence of an eDRAM hit. Thus, there are writebacks both due to replacements and due to internal swaps. Regarding the delayed policy, writebacks due to replacements only can occur in the eDRAM ways. In addition, writebacks can also rise driven by the internal counter. This kind of writebacks will be referred to as *sporadic* writebacks.

To evaluate the effectiveness of the devised policies, it has been obtained the corresponding writeback ratio for each writeback class, calculated as the number of written back blocks divided by the number of accesses to the L1 cache.

Figure 2.10 shows the average writeback ratio of both policies in the hybrid cache configurations for a 10fF capacitance. For comparison purposes, the figure also depicts the average writeback ratio of the conventional cache (Conv).

As observed, for a given cache organization and benchmark type, the replacement writeback ratio is quite uniform across the analyzed policies. Regarding the early policy, its swap writeback ratio leads to an important amount of blocks written back to L2. The comparison between this ratio and the sporadic ratio of the delayed policy leads to the conclusion that the early policy is too conservative since many blocks are written back unnecessarily, so incurring in an energy wasting.

In short, the proposed delayed writeback policy performs, without refresh, a similar amount of writebacks as a conventional cache with the same organization, so no significant energy wasting is incurred due to additional writebacks.

2.3.3 Energy Consumption

This section discusses the methodology used to obtain both dynamic and leakage energy consumed by each cache scheme. Then, energy results are presented and analyzed for different cache organizations.

2.3.3.1 Methodology

An access to a conventional cache consumes energy due to the fact that all the components in the cache set (i.e., all the tags and data ways) are accessed in parallel. In contrast, only the SRAM way of the data array is accessed in the hybrid cache, although in parallel with all the tags of the set. Therefore, on an SRAM hit, the hybrid cache consumes less dynamic energy than the conventional cache. Nevertheless, additional energy is consumed on an SRAM miss (i.e., an eDRAM hit or cache miss), since the cache controller must drive a swap operation between the corresponding eDRAM and SRAM cells.

To estimate the energy consumption, each of the three steps involved in a swap operation (see Section 2.2.2) has been modeled as follows: i) a destructive read in an eDRAM way, ii) a store in an eDRAM way, and iii) a store in the SRAM way. While the dynamic energy consumed in the second step involves only the row decoder, in the first and third steps it also involves all the data array components (e.g., bitlines, senseamps, subarray output drivers, and so on). In addition, the energy consumption due to wordline voltage boosting and precharging capacitors in the second step (see Section 2.1) has been also taken into account in the results.

On a cache miss, the requested block is fetched from L2 (or main memory) and the victim block of L1, if dirty, is written back to L2. These read and write requests to L2 have been also considered to compute the dynamic energy.

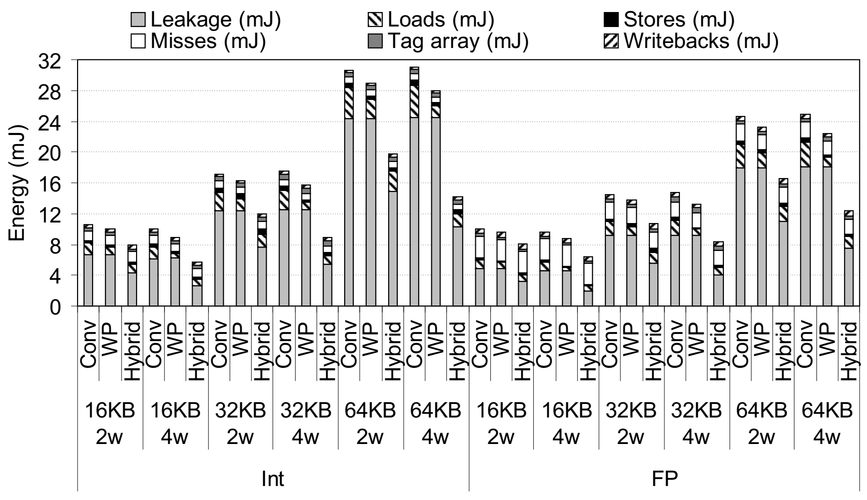


FIGURE 2.11: Energy consumption (in mJ) for the analyzed schemes.

To quantify energy savings, the proposed hybrid cache uses the ITRS high-performance device type for the SRAM cells and the logic-process based DRAM for the eDRAM cells. CACTI provides the dynamic energy consumed per access and categorized by cache component. Using these values, dynamic energy was calculated per access according to different events: SRAM hit, eDRAM hit, misses, and writebacks. Then, we measured the times that each event rises during each benchmark execution (i.e., 500M instructions). Finally, the overall dynamic energy was computed by multiplying the energy per access for a given event by the times that the event rises. Regarding leakage consumption, it is accumulated each cycle taking into account that there is minimal leakage energy associated to the eDRAM cells.

2.3.3.2 Energy Results

The proposed hybrid cache is compared not only to the conventional scheme but also to another conventional scheme implementing way-prediction as done by the proposal. This model will be referred to as WP scheme. In this way, the benefits due to the prediction technique can be identified. For the hybrid cache, 10fF capacitors and the delayed writeback policy were used.

Figure 2.11 illustrates the dynamic and leakage energy results for the analyzed schemes. The dynamic energy consumed by the L1 data array is classified according to the type of event (i.e., loads, stores, writebacks, and misses), whereas the tag array energy comprises all the accesses to this structure. Finally, the represented leakage refers to the leakage consumption of the whole L1 cache. The sum of all these values gives the total energy consumed by the cache memory.

Notice that leakage energy consumption is obtained by accumulating each cycle the leakage consumption of the entire cache. In contrast, dynamic energy is only consumed when there is an access to the cache. Thus, in those execution periods where the cache is not accessed, the leakage energy consumption dominates the overall energy consumption of the L1 cache. Such periods are usually common since many memory reference instructions access to the cache in a burst mode. Moreover, when the processor pipeline is stalled due to a long latency event (e.g., an L2 miss), the L1 cache is not accessed for hundreds of cycles.

Due to the aforementioned reasons, leakage dominates the overall energy consumption. As a consequence, the hybrid cache, which incurs in less leakage consumption because of the use of eDRAM cells, shows the best energy results. In particular, regarding integer benchmarks, the proposed memory reduces the overall energy consumption by 36% and 54% for a 64KB-2way and a 64KB-4way cache organizations with respect to the Conv scheme, respectively. This trend is similar in the remaining analyzed cache organizations as well as for floating-point benchmarks. Notice that the leakage energy consumption is quite similar in the Conv and WP schemes, since both schemes use only SRAM cells for data storage.

Leakage consumption increases with the cache size for both the Conv and WP schemes. However, to appreciate significant differences in the hybrid cache, the associativity degree must be fixed. More precisely, the static part of the cache must be enlarged. Although theoretically the hybrid cache should reduce leakage by 50% and 75% for 2 and 4 ways, respectively, compared to the conventional caches, experimental results do not reach these values. The main reason is that this theoretical reduction is restricted to cells in the data array (where the eDRAM technology is being applied) while other parts of the cache structure, like the tag array and the remaining cache controller logic are not being considered.

Regarding dynamic energy, the WP cache exhibits the lowest consumption, closely followed by the hybrid cache. This is because both schemes save a significant amount of dynamic energy by applying way-prediction. However, the overall dynamic consumption is higher in the hybrid cache mainly due to swap operations, wordline voltage boosting, and precharge of capacitors. The consumption due to loads and stores also increases with the cache size (although more slowly than leakage consumption) since the dynamic energy consumed per access grows with the cache size. On the other hand, the consumption due to writebacks is quite uniform across the analyzed schemes since the delayed writeback policy performs a similar amount of writebacks as the Conv and WP schemes. Finally, larger caches show lower miss and writeback rates, thus devoting less energy to such events.

All the analyzed schemes implement the same tag array (i.e., using typical 6T SRAM cells). Therefore, the energy consumed by this structure slightly differs across the schemes for a given cache organization. Nevertheless, its contribution to the overall energy consumption increases with the cache size and associativity.

2.3.4 Area Savings

The n-bit macrocell saves area with respect to n conventional SRAM cells, since the former is partly implemented with eDRAM cells. The width and height of the proposed macrocell have been estimated with CACTI by adding the area of the conventional SRAM and eDRAM cells. In addition, the area overhead due to the wordline voltage boosting and the bridge transistors has been taken into account, assuming a conservative design where each bridge transistor occupies the same area as a 1T-1C cell.

The area of an SRAM and an eDRAM cell is $0.296\mu m^2$ and $0.062\mu m^2$ for a 45nm technology node, respectively. Using these values, since an n-bit macrocell consists of one SRAM cell and *n-1* eDRAM cells, the area of a 2-bit and a 4-bit macrocell is $0.566\mu m^2$ and $0.836\mu m^2$, respectively. These area values were used as input to CACTI to obtain the area of the entire hybrid cache.

Recall that eDRAM cells of the macrocell are assumed to be built with trench storage capacitors. These capacitors etch deep holes into the wafer and are formed in the silicon substrate instead of above it. The deeper the hole the higher the capacitance. Thus, the macrocell area is not affected by the capacitance value [44].

Table 2.2(a) shows the area (in mm^2) of the tag and data arrays of both conventional (Conv) and hybrid schemes varying the size (from 16KB to 64KB) of a 2-way

Cache size	Tag array (mm^2)	Data array (mm^2)		Reduction (%)
		Conv	Hybrid	
16KB	0.008	0.273	0.259	5
32KB	0.015	0.332	0.316	5
64KB	0.018	0.446	0.424	5

a) 2-way caches

Cache size	Tag array (mm^2)	Data array (mm^2)		Reduction (%)
		Conv	Hybrid	
16KB	0.008	0.446	0.318	28
32KB	0.015	0.498	0.355	28
64KB	0.020	0.602	0.431	28

b) 4-way caches

Table 2.2: Tag array and data array area (in mm^2) for both conventional and hybrid schemes.

set-associative cache. Table 2.2(b) shows the corresponding results for a 4-way set-associative cache. The last column of each table shows the total reduction (in percentage) achieved by the hybrid cache. This percentage increases with the number of eDRAM cells and it is quite uniform regardless of the cache size. It is around 5% and 28% for 2- and 4-way caches, respectively.

2.4 Summary

This chapter has presented the hybrid eDRAM/SRAM macrocell-based L1 cache architecture. The macrocell has been shown as an efficient device to implement cache memories, since its design deals with energy consumption and area while maintaining the performance. Architectural mechanisms such as swap operations, way-prediction, and special writeback policies have been considered to maintain the most likely data to be accessed in SRAM technology as well as to avoid unnecessary destructive reads and refresh logic required in typical eDRAM memories.

Experimental results have shown that, compared to a 4-way set-associative conventional SRAM cache with the same organization, a hybrid cache obtains energy and area savings up to 54% and 28% respectively; while having scarce impact on performance (lower than

2%). In addition, a relatively low capacitance of 10fF is enough to avoid performance losses due to retention time constraint.

The work discussed in this chapter has been published in [45] [46] [47].

Chapter 3

Hybrid eDRAM/SRAM L2 Cache Architecture

As L2 caches are larger than L1 caches, more energy and area savings can potentially be reached when the hybrid architecture is applied to them. This chapter presents the hybrid eDRAM/SRAM cache architecture for L2 caches. First, the distribution of cache hits across the ways of the L2 cache is studied to provide insights about the optimal distribution of SRAM and eDRAM ways. Then, the hybrid design and its associated architectural mechanisms are presented. The experimental evaluation is performed from both storage capacity and area points of view, and it covers performance and energy consumption. The optimal percentage of SRAM and eDRAM ways is identified using the Energy-Delay squared Product (ED^2P) and Energy-Delay-Area Product (EDAP) metrics.

3.1 Motivation

As shown in the previous chapter, most hits concentrate in the MRU blocks in L1 data caches. Therefore, for hybrid L1 eDRAM/SRAM caches, it is enough to build a single cache way with fast SRAM technology and force this cache way to store the MRU block for performance purposes. However, it is widely known that data locality in L2 caches is much poorer than it is in L1 caches, thus this implementation might yield to unacceptable performance in L2 caches. In addition, the macrocell design would become too complex and expensive to implement in huge LLCs with high associativity.

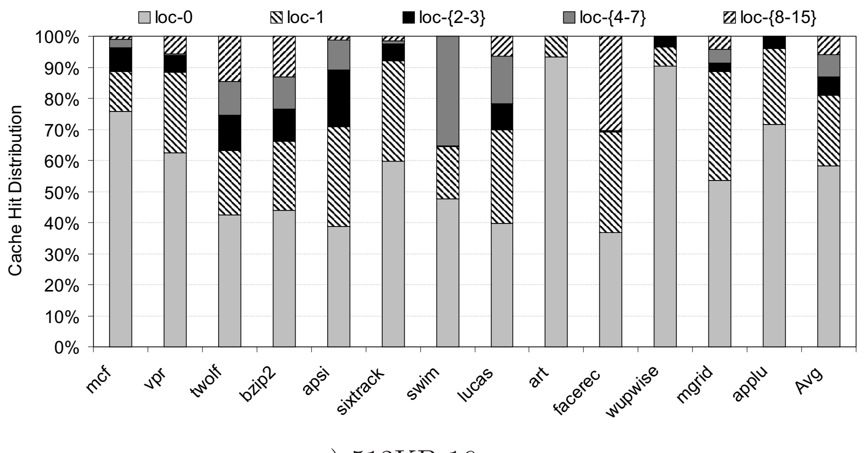
Due to the latter reason, the hybrid proposal works at a coarse granularity in L2 caches, that is, at cache bank level instead of cell level. To identify which is the most appropriate ratio of SRAM and eDRAM banks in terms of performance, energy, and area, a design space exploration for hybrid L2 caches is performed.

The two extremes of this design space exploration are defined by caches implemented with a single technology. These caches are referred to as pure SRAM and pure eDRAM caches. Both extremes provide, respectively, the maximum performance but higher energy expenses and area, as well as the minimum performance but lower leakage energy consumption and area. Between these points, the percentage of SRAM banks and eDRAM banks can be varied in order to benefit performance or energy and area savings.

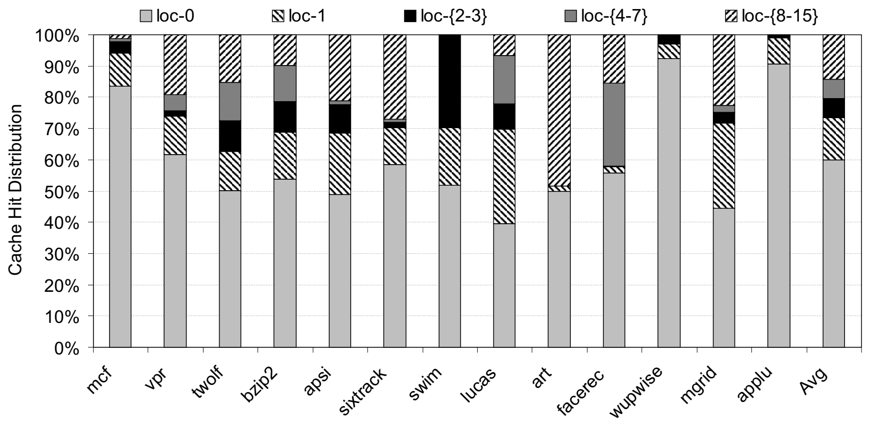
In summary, the design space exploration is useful to find out the optimal hybrid eDRAM/SRAM L2 cache, which provides the best trade-off among performance, energy consumption, and area.

For analysis purposes, two cache organizations have been considered, a 512KB-16way and a 1MB-16way set-associative LLCs (L2 caches) implementing the LRU replacement algorithm. With the aim to serve as a guide to estimate how many ways should be built with SRAM banks and how many with eDRAM banks, this section explores the distribution of cache hits.

The distribution of cache hits has been obtained with the aim of analyzing if hits concentrate in a few blocks at the top of the LRU stack. In such a case, these blocks should be stored in fast SRAM banks and a swap mechanism should enable the movement of blocks between banks built with different technologies. Figure 3.1(a) and Figure 3.1(b)



a) 512KB-16way



b) 1MB-16way

FIGURE 3.1: Percentage of cache hits across the locations of the LRU stack.

show the results of both conventional 512KB-16way and 1MB-16way L2 caches, respectively. Those applications exhibiting an L2 hit ratio greater than 85% in both cache organizations were skipped for this study, since they do not represent a realistic behavior in this level. Label *loc-0* refers to the location storing the MRU block, while *loc-15* is the position storing the LRU block. Label *loc*-{*x*-*y*} denotes hits in a location falling in between x and y in the stack, both inclusive.

Results indicate that, unlike L1 caches where more than 90% of hits concentrate in the MRU way, hits are distributed among different locations of the LRU stack in L2 caches. Although the distributions are clearly skewed to the first ways, the 512KB cache requires half the cache ways (8 ways) to cover by 95% of the cache hits, while this percentage

Cache configuration	SRAM ways	eDRAM ways	SRAM banks	eDRAM banks	SRAM ratio (%)
16S	16	0	8	0	100
8S-8D	8	8	4	4	50
4S-12D	4	12	2	6	25
2S-14D	2	14	1	7	12.5
16D	0	16	0	8	0

Table 3.1: Pure and hybrid caches with the corresponding number of ways, banks, and ratio (%) of SRAM banks

drops down to 85% for the 1MB cache. Notice too that, for the 512KB cache, the MRU way captures *only* around 50% of the cache hits in 7 of 13 applications. This number of applications grows up to 9 for the 1MB cache. Anyway, the percentage of hits in the MRU way is on average by 60% for both cache organizations. Thus, these results confirm that implementing only the MRU way with SRAM technology would yield to unacceptable performance. Finally, remark that there is no need to maintain a bidirectional relationship between cache ways and positions of the LRU stack (e.g., way 4 stores the block in position 4 of the stack) but only to find out which percentage of blocks should be stored in fast SRAM ways, which simplifies the design.

3.2 Hybrid Last-Level Cache Design

This section discusses how the hybrid L2 cache is accessed to avoid unnecessary destructive reads and introduces how the swap operations between eDRAM and SRAM banks are performed to keep the MRU data in the latter banks. In addition, a distributed refresh mechanism to avoid capacitor discharges is presented.

This work assumes that each cache bank stores a pair of ways, which results in an LLC with 8 banks for the studied 16-way caches. As each bank of the data array can be implemented with either SRAM or eDRAM technology, several design choices can be explored. Table 3.1 summarizes the studied design choices, specifying the number of SRAM and eDRAM ways and banks of each cache configuration and the ratio of SRAM banks. The conventional schemes in both extremes of the table are the pure SRAM (16S) and the pure eDRAM (16D), which have all their banks implemented with SRAM and eDRAM technology, respectively. The tag array of all the studied cache configurations

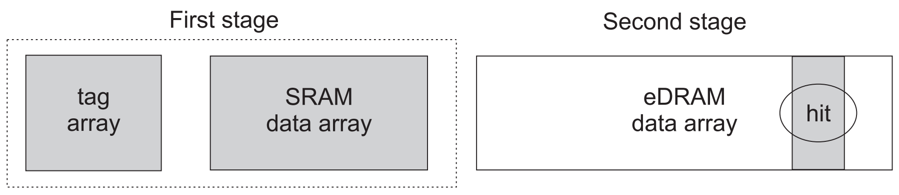


FIGURE 3.2: Diagram of the hybrid cache access. Dark boxes represent the accessed parts of the cache. The second stage is performed only on a hit in an eDRAM way detected in the first stage.

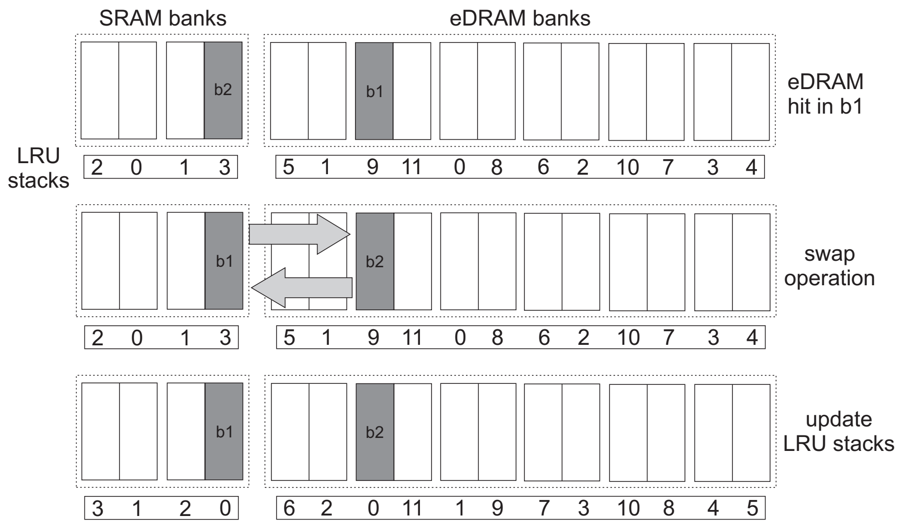
is assumed to be implemented with typical SRAM cells; thus, analogous to the hybrid L1 cache, the reading of any tag is not a destructive operation.

3.2.1 Accessing the Hybrid Cache: Bank-Prediction

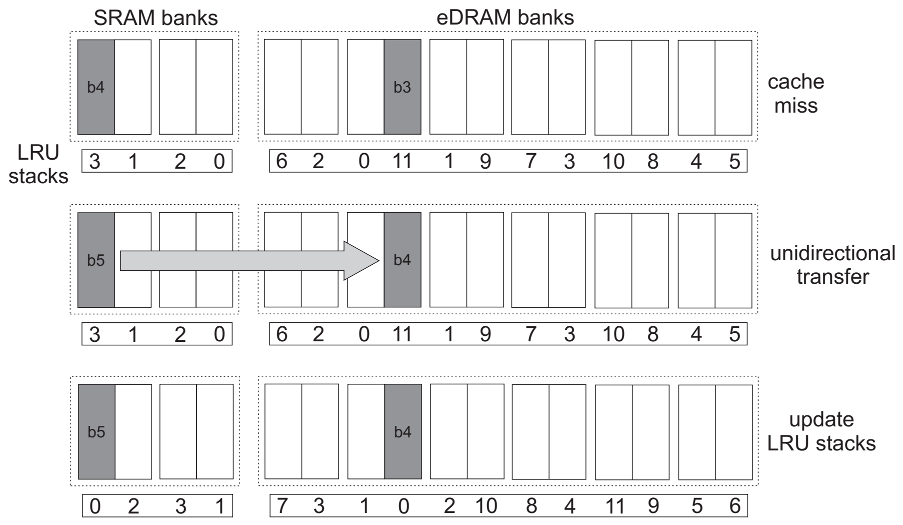
Like in the hybrid L1 design, the energy wasting of the destructive reads due to accessing all the cache ways is also avoided in hybrid L2 cache designs. However, unlike previous works, the proposed hybrid L2 caches predict various ways (instead of only one) that will be accessed during a first stage. Hence, the access of these memories is split into two stages as depicted in Figure 3.2. In the first stage, all the tags (which are built with SRAM technology) and all the SRAM banks (SRAM data array) are accessed in parallel. This technique will be referred to as *bank-prediction*. If the requested block is stored in an SRAM way, the access time of the hybrid cache is as fast as a hit in a conventional SRAM cache and the second stage is not performed (i.e., no eDRAM way is accessed). On a miss in the SRAM data array but a hit in a tag associated to an eDRAM way, the second stage is performed and only the target eDRAM way is accessed. In this case, the access time includes the tag comparison plus the access to the eDRAM data. On a cache miss, no eDRAM way is accessed and the requested block is fetched from main memory.

3.2.2 Keeping the Last Accessed Blocks in SRAM Banks

To keep the MRU data in fast SRAM banks, the cache controller manages a swap operation between SRAM and eDRAM banks, similar to the swap performed in the proposed hybrid L1 architecture. To properly select the blocks to be transferred, the design maintains a separate LRU stack order in both SRAM and eDRAM data arrays, which allows reduce the required number of LRU control bits. Figure 3.3(a) and Figure 3.3(b)



a) eDRAM cache hit



b) Cache miss

FIGURE 3.3: Diagrams of the involved actions in an eDRAM hit and a cache miss to keep the MRU data in SRAM banks.

illustrate the 4S-12D cache configuration with the different actions carried out on an eDRAM hit and a cache miss, respectively, to maintain the MRU data in the SRAM data array.

On an eDRAM hit, the requested eDRAM block (labeled as *b1*) is transferred from its eDRAM bank to the SRAM bank that holds the LRU block of the SRAM data array (referred to as *b2*), which in turn is moved to the eDRAM bank. To properly perform the swap operation, block *b1* is temporarily placed in a buffer associated to the eDRAM bank while block *b2* moves to the target eDRAM bank. Finally, block *b1* is transferred from the auxiliary buffer to the SRAM bank. After this operation, the involved blocks are set as the MRU ones of each data array by updating both LRU stacks.

Remark that a block is not evicted from the SRAM data array neither moves to the eDRAM array until it becomes the LRU and is selected to be swapped. Before being evicted from the SRAM array, a block always resides in the same SRAM bank. In other words, an SRAM hit does not imply any data movement between banks.

On a cache miss, the LRU block of the eDRAM data array (block *b3*) is selected for replacement. The cache controller triggers a unidirectional transfer from the SRAM LRU block (block *b4*) to the eDRAM bank that contains the victim block, while the incoming data block that is fetched from main memory (block *b5*) is allocated in the SRAM bank. Of course, the control bits must be accordingly updated.

Finally, tags are not swapped in the hybrid L2 cache. Instead, four control bits (i.e., *data way location* bits) per tag are required to maintain the mapping between tags and cache ways of 16-way caches (see Section 2.2.2).

3.2.3 Distributed Refresh

Unlike hybrid L1 caches, refresh logic cannot be removed in the L2 cache. This is because data located in the L2 cache may be accessed after very long periods of time since L1 filters most cache accesses to L2. As a consequence, depending on the cache configuration, the retention time has to be set to millions of processor cycles to avoid the need of refresh operations, which corresponds to a capacitance far beyond from the typical capacitances of eDRAM devices [40] [41].

For both hybrid and pure eDRAM L2 caches, the capacitance has been set to 10fF, which is the optimal capacitance obtained for the hybrid L1 design. This capacitance corresponds to a retention time of 190K processor cycles for a 3GHz frequency. Analogous to the hybrid L1 cache, the eDRAM cells of the hybrid L2 scheme are considered to be implemented with trench capacitors.

To avoid capacitor losses, refresh cycles can be used either in a distributed or burst method in typical DRAM memories [48]. This thesis assumes the distributed refresh for eDRAM banks since it is the commonly used method. In order to mitigate the refresh penalty, the refresh operations have been interleaved among banks and performed following a round-robin policy, where each eDRAM block is regularly refreshed. Similar to the delayed writeback policy of the hybrid L1 approach, the period between two consecutive refresh operations is established as the retention time divided by the number of eDRAM blocks. This guarantees that all the eDRAM blocks are refreshed before their retention time expire.

Finally, notice that special writebacks policies are not required since data losses are avoided by the distributed refresh. Thus, like in conventional caches, writeback operations only occur during block replacements.

3.3 Experimental Evaluation

This section evaluates area, performance, and energy consumption for each studied distribution of cache banks. The analysis is carried out on the basis of both storage capacity and area. Performance results include both hit ratio and IPC analysis. The energy expenses are split into leakage and dynamic energy. In addition, the energy consumption of the distributed refresh has been considered too. Both ED^2P and EDAP metrics will be considered to determine which is the most efficient distribution of cache banks.

The architectural machine parameters are the same as those presented in the hybrid L1 cache analysis (see Section 2.3.1) with the following exceptions. In this study, the L1 data cache is assumed to be a conventional SRAM cache with a fixed 16KB-2way cache organization and an access time of 2 cycles. This configuration remains unchanged for the remainder of the thesis. The L2 cache is evaluated for both 512KB and 1MB cache capacities with an associativity of 16 ways. The access time of the tag array, SRAM banks, and eDRAM banks has been set to 2, 6, and 9 cycles, respectively. All the access time numbers were obtained with CACTI for a 3GHz processor speed, resembling that of most existing commercial processors. Remark that the access time (in ns) increases with the cache capacity; however, for the L2 cache organizations studied in this chapter, the differences are hidden when the access time is rounded up to processor cycles. Finally,

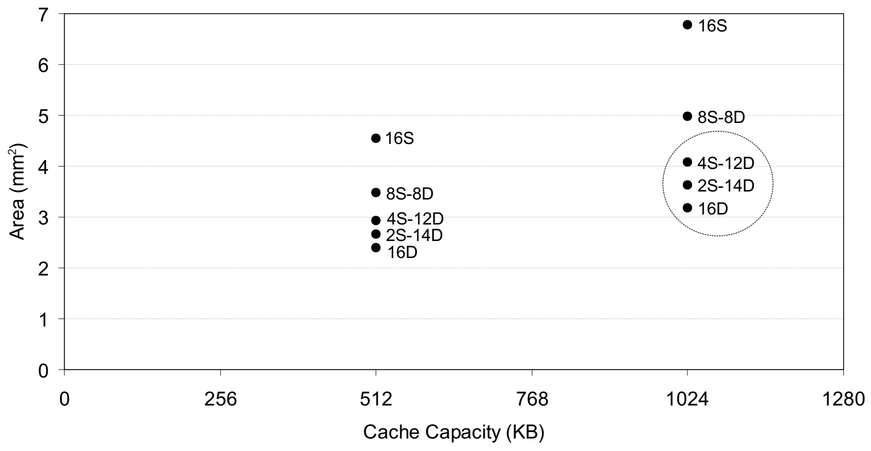


FIGURE 3.4: Area (in mm^2) of the analyzed caches. The circle groups those 1MB eDRAM-based configurations with less area than the 512KB SRAM cache.

bank contention in both hybrid and pure eDRAM caches includes the refresh memory events.

3.3.1 Area

To estimate the area of the studied caches, it is first obtained the area of an SRAM and an eDRAM bank. Then, these values have been accumulated according to the number of banks in each cache to calculate the area of the data array. The area overhead due to the control bits to keep the mapping between tags and ways has been taken into account in the hybrid caches.

Figure 3.4 plots the cache area results (in mm^2) including both tag and data arrays. Notice that no area benefits come from the tag array since it is built with SRAM technology regardless of the cache configuration. As can be seen, the higher the number of eDRAM banks of the data array the larger the area savings. Compared to the conventional 16S cache with the same capacity, the 16D cache is the scheme that most reduces the area (by 47% in the 512KB cache), closely followed by the 2S-14D hybrid cache (41%). These area reductions are larger for the 1MB cache size. In this case, area savings are up to 46% for the 2S-14D approach.

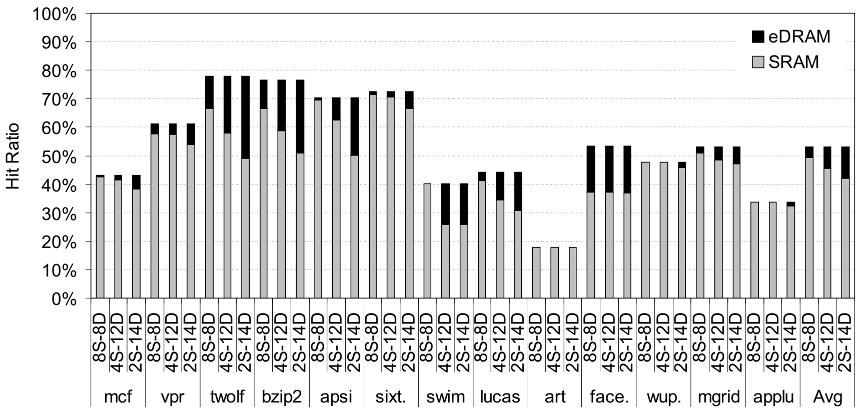
In the figure, it can be appreciated that 4S-12D, 2S-14D, and 16D (highlighted with the circle) 1MB eDRAM-based configurations present area savings with respect to the pure 512KB SRAM cache despite their storage capacity is twice as large. Based on this observation, it makes sense to compare different approaches not only on a capacity basis but also on an area basis. Thus, to perform the analysis on the basis of area, the highlighted 1MB eDRAM-based caches will be compared against the conventional 512KB SRAM cache. Note that some of these 1MB caches significantly reduce the area of the 512KB cache, thus this type of study provides conservative results for the highlighted configurations.

3.3.2 Performance

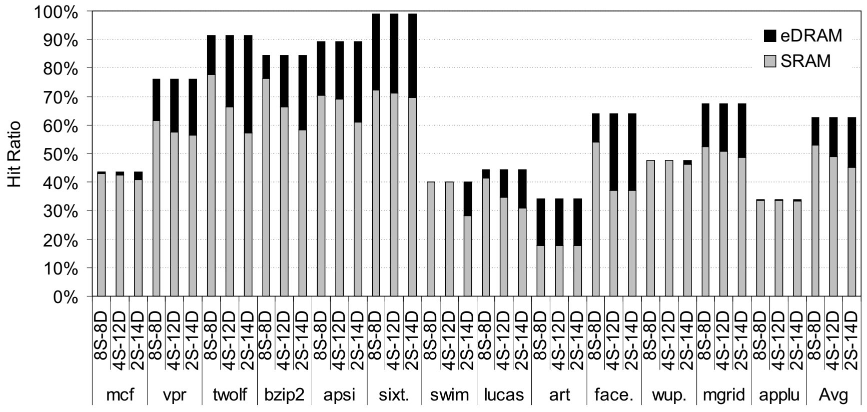
To provide insights in performance, the hit ratio in the different cache banks is quantified first. Remember that the hybrid designs do not allow information loss due to capacitor discharges. Thus, the total hit ratio matches the obtained in pure SRAM caches. Figure 3.5(a) and Figure 3.5(b) depict the results for the 512KB and 1MB caches, respectively.

As expected, for a given cache size, the hit ratio in eDRAM banks (eDRAM hit ratio) increases with the number of eDRAM ways. Nevertheless, this is not the case in a few applications. For instance, the eDRAM hit ratio in *art* holds constant for both 512KB and 1MB caches regardless of the cache bank distribution. For the 512KB cache, this behavior can be explained by looking the values of this benchmark in Figure 3.1(a), where only the MRU way and the following one (i.e., *loc-0* and *loc-1*) are enough to capture almost all the cache hits, so no eDRAM hit ratio can be observed in any hybrid cache organization. On the other hand, for the 1MB cache, this pair of ways capture around 50% of cache hits, while locations from 8 to 15 in the LRU stack capture almost all the remaining hits.

For the 512KB cache, the eDRAM hit ratio is on average by 4%, 7%, and 11% for 8S-8D, 4S-12D, and 2S-14D hybrid approaches, respectively. These percentages grow up to 10%, 14%, and 18%, respectively, for the 1MB cache. Similarly, for the 512KB cache, the SRAM hit ratio is on average by 49%, 46%, and 42% for 8S-8D, 4S-12D, and 2S-14D hybrid configurations, respectively, while for the 1MB cache, these percentages are by 53%, 49%, and 45%. This is due to, compared to the 512KB hybrid cache, the SRAM data array of the 1MB hybrid cache is twice as large. Overall, the 512KB cache achieves on average a higher miss ratio (by 47%) than the 1MB cache (by 37%).



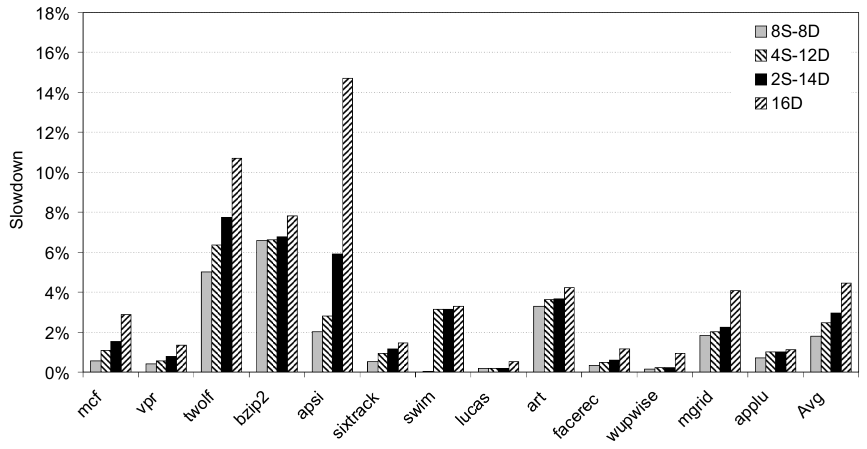
a) 512KB-16way



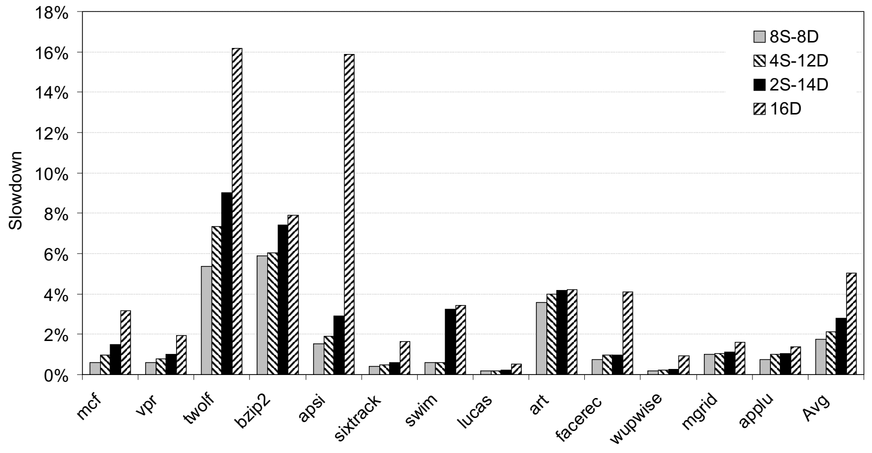
b) 1MB-16way

FIGURE 3.5: Hit ratio (%) split into hits in SRAM and eDRAM banks. XS-YD configuration refers to a hybrid cache having X ways in SRAM banks and Y ways built with eDRAM banks.

To achieve enhanced performance, it is important that the percentage of eDRAM hits remains as low as possible since an access to an eDRAM way takes more cycles than an access to an SRAM way. Performance losses due to bank contention also rise because of periodic refresh operations. These losses are not constant across the different cache configurations, since the elapsed time between two consecutive periodic refreshes becomes shorter as the number of eDRAM blocks increases. In addition, in the pure eDRAM cache (16D), reads require to refresh data since these operations are destructive, which also induces bank contention. In contrast, the bank contention on an eDRAM hit in hybrid caches is induced by the swap operation between the involved banks. The swap operation has been modeled as a read access to the target eDRAM bank, a write access



a) 512KB-16way



b) 1MB-16way

Figure 3.6: Slowdown (%) of the analyzed cache configurations with respect to the pure SRAM cache on the basis of capacity.

to that bank, and another write access to the target SRAM bank. Bank contention is considered in these three steps.

Figure 3.6(a) and Figure 3.6(b) show the slowdown of the studied 512KB and 1MB caches, respectively, compared to a pure SRAM cache with the same capacity (the lower is the better). As expected, the slowdown increases with the number of eDRAM ways regardless of the cache size. Enlarging the eDRAM data array results in a higher number of refresh operations, slow accesses to eDRAM data, and swap operations. In general, the performance loss is higher in those applications with a high eDRAM hit ratio. For instance, in *twolf*, the eDRAM hit ratio can be as high as 29% in the 512KB

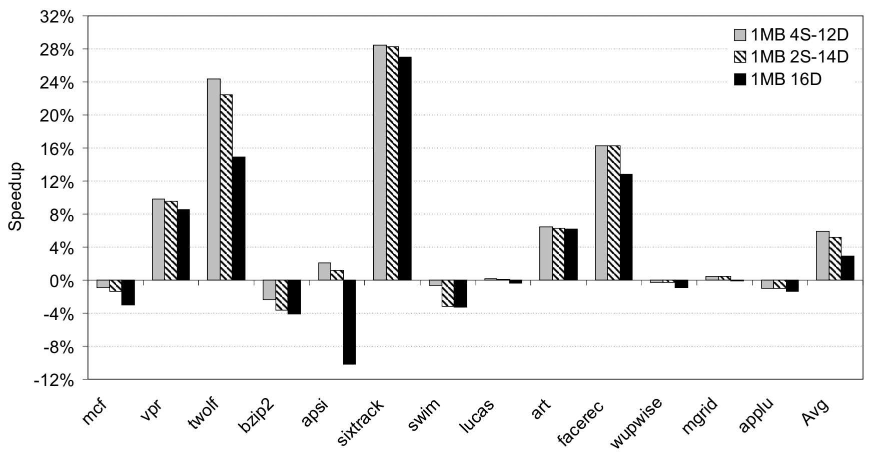


Figure 3.7: Speedup (%) of the selected 1MB eDRAM-based caches with respect to the SRAM scheme on the basis of area.

cache with the 2S-14D configuration, which leads to 7.7% slowdown. In comparison, for the 1MB cache, the eDRAM hit ratio is up to 34% and its slowdown is by 9%. The pure eDRAM architecture is strongly affected both by the slow access time and bank contention induced by refresh operations. For instance, in *twolf*, the slowdown is around 16.2% for the 1MB cache with the 16D scheme, resulting in a very poor performance.

The slowdown for a 512KB cache is on average by 1.8%, 2.5%, and 3.0% in the 8S-8D, 4S-12D, and 2S-14D hybrid approaches, respectively. In comparison, minor differences appear in these percentages for the 1MB hybrid cache. In the 16D configuration, the slowdown grows up to 4.5% and 5.0% for the 512KB and 1MB caches, respectively.

Finally, the analysis on the basis of area is presented. Figure 3.7 plots the speedup of the selected 1MB eDRAM-based caches with respect to the conventional 512KB SRAM cache. As observed, the hybrid caches, with much longer access time when accessing eDRAM data, improve performance in 8 of 13 applications. The speedup in applications like *vpr* and *twolf* comes from the fact that the hit ratio significantly increases with the cache capacity (see Figure 3.5). For example, the hit ratio of *vpr* is by 61% and 76% for the 512KB and 1MB cache size, respectively. On the contrary, in some other benchmarks such as *mcf* and *bzip2*, the increase of the hit ratio (if any) does not compensate the bank contention of the higher number of refresh operations. Again, regardless of the application, the performance increases with the number of SRAM ways. To sum up, the speedup of both 2S-14D and 4S-12D hybrid caches is on average by 5.2% and 5.9%, respectively, while this percentage drops down to 3.0% for the pure eDRAM cache.

Type of energy	16S	8S-8D	4S-12D	2S-14D	16D
Total leakage (mJ)	70.7	42.6	28.2	21.0	13.8
Tag array (mJ)	0.2	0.2	0.2	0.2	0.2
SRAM hits (mJ)	4.9	2.3	1.1	0.5	0
eDRAM hits (mJ)	0	0.2	0.3	0.3	4.9
Swaps (mJ)	0	0.9	1.0	1.1	0
Writebacks (mJ)	0.3	0.3	0.3	0.3	0.3
Misses (mJ)	7.5	4.2	2.5	1.7	7.4
Refreshes (mJ)	0	1.1	1.7	2.0	13.7
Total dynamic (mJ)	13.0	9.2	7.1	6.0	26.5
Total consumption (mJ)	83.7	51.9	35.3	27.0	40.3
Total reduction (%)	–	38.0	57.9	67.7	51.8

a) 512KB-16way

Type of energy	16S	8S-8D	4S-12D	2S-14D	16D
Total leakage (mJ)	120.2	70.2	44.4	31.6	18.9
Tag array (mJ)	0.2	0.2	0.2	0.2	0.2
SRAM hits (mJ)	8.7	3.5	1.6	0.7	0
eDRAM hits (mJ)	0	1.2	0.9	0.7	9.5
Swaps (mJ)	0	1.5	1.6	1.8	0
Writebacks (mJ)	0.3	0.3	0.3	0.3	0.3
Misses (mJ)	8.6	4.8	2.9	1.9	9.3
Refreshes (mJ)	0	3.2	4.9	5.7	24.7
Total dynamic (mJ)	17.8	14.6	12.5	11.4	44.1
Total consumption (mJ)	138.0	84.8	56.9	43.0	63.1
Total reduction (%)	–	38.5	58.8	68.8	54.3

b) 1MB-16way

Table 3.2: Leakage and dynamic energy consumption (in mJ) and the reduction (%) compared to the pure SRAM scheme.

3.3.3 Energy Consumption

This section analyzes both leakage and dynamic energy consumption (in mJ) of the studied L2 caches. Table 3.2(a) and Table 3.2(b) show the results for the 512KB and 1MB cache organizations, respectively. Leakage energy includes the consumption of both tag and data arrays. To provide insights in energy savings, it has been analyzed separately the dynamic energy of the tag array, which is looked up on every cache access, and the data array energy, which has been classified into several categories according to the access type. Six different categories have been identified: SRAM hits, eDRAM hits, swaps, writebacks, misses, and refreshes.

The SRAM hits category denotes the energy consumed by the access to the SRAM array, whereas the eDRAM hits category takes into account the previous access to the SRAM banks (which is useless, as there is an SRAM miss) and the access to the target eDRAM bank. The swaps category includes the expenses of transferring the involved blocks in the swap operation of hybrid caches. This category also includes the consumption of the unidirectional transfers from SRAM to eDRAM banks that arise on a cache miss in hybrid approaches. The writeback category considers the energy of accessing just the bank containing the victim block. The misses category includes the energy required to access the SRAM banks, the eDRAM ones (if any), and the access to the bank where the incoming block is allocated. The refresh category takes into account the energy consumed by the periodic refresh and the energy due to restoring the eDRAM contents after a destructive read. Notice that, as there is not information loss due to capacitor discharges regardless of the cache scheme, the energy penalty of accessing to main memory has not been considered since it is the same for all the studied caches.

As observed, both eDRAM-based and hybrid approaches reduce leakage currents by design thanks to the use of eDRAM banks. Indeed, leakage decreases with the number of eDRAM ways. Compared to the 16S cache, the 2S-14D approach reduces leakage by 70% for the 512KB cache. This percentage grows up to 74% for the 1MB cache size. As expected, for a given cache scheme, the 1MB caches consume a larger amount of leakage with respect to the 512KB caches since they double the cache capacity.

Regarding dynamic energy, the tag array consumption is almost negligible compared to the energy of the data array. As expected, the 16S cache is by far the scheme that consumes more energy in the SRAM hits category since all the cache ways are accessed in parallel, while the values in this category decrease with the number of eDRAM ways. As opposite, the eDRAM hits energy increases with the number of eDRAM ways, although the fact of accessing the SRAM ways ahead of the eDRAM target may prevent from obtaining low values even for a low number of eDRAM ways. This is the case of the 8S-8D configuration for the 1MB cache (1.2 mJ).

The expenses of the swap operation do not represent an important fraction of the dynamic energy consumption. The worst case can be found in the 2S-14D configuration for the 512KB cache, where the swap energy consumption (1.1 mJ) represents about 18% of the total dynamic energy (6.0 mJ). Nevertheless, in spite of this fact, this is the configuration that most reduces the dynamic energy among all the studied caches.

The consumption due to writebacks slightly affects the total energy, and it is roughly the same across the studied schemes. In contrast, noticeable differences appear in the misses category. The 16S and 16D caches consume a large amount of energy because of the entire data array is involved on each access. For the hybrid caches, the consumption due to misses decreases with the number of SRAM ways.

The energy due to refresh operations increases with the eDRAM ways, since more eDRAM blocks are checked to be refreshed. Recall that this category also includes the energy expenses due to writing the contents after a destructive read in the 16D approach.

Overall, for a given cache scheme, the dynamic consumption increases with the cache capacity, similar to the leakage expenses. Compared to the 16S scheme, the 2S-14D configuration reduces dynamic energy by 36% in the 1MB cache. This percentage is up to 54% for the 512KB cache size. Note also that, compared to the conventional scheme, the 16D approach doubles the dynamic consumption in the 512KB cache, while in the 1MB cache the consumption increases by a $2.48x$ factor.

An interesting observation is that, for the study on the basis of area, both 4S-12D and 2S-14D configurations for the 1MB cache reduce the total dynamic energy in spite of having a capacity twice as large. In contrast, the 1MB 16D cache significantly increases the total dynamic energy with respect to the 512KB SRAM approach.

Regarding total energy consumption, for a given cache capacity, the 16D configuration reduces it by 52% and 54% for the 512KB and 1MB caches, respectively. This percentage grows up to 58–59% and 68–69% when using the 4S-12D and 2S-14D configurations, respectively. However, for the 8S-8D approach, the obtained energy savings are lower than those of the pure eDRAM scheme. This is mainly due to the high leakage consumption when the hybrid cache capacity is half SRAM-based. Compared to the 512KB 16S cache, both 4S-12D and 2S-14D hybrid schemes for the 1MB cache reduce the overall energy by 32% and 49%, respectively. This percentage drops down to 25% for the pure 1MB eDRAM cache.

Finally, remark that, in general, the leakage contribution represents a high percentage of the total energy consumption. This is due to leakage is always consumed independently of whether the cache is being accessed or not, while there is not dynamic energy consumption in L2 if blocks are found in L1 (except from the periodic refresh).

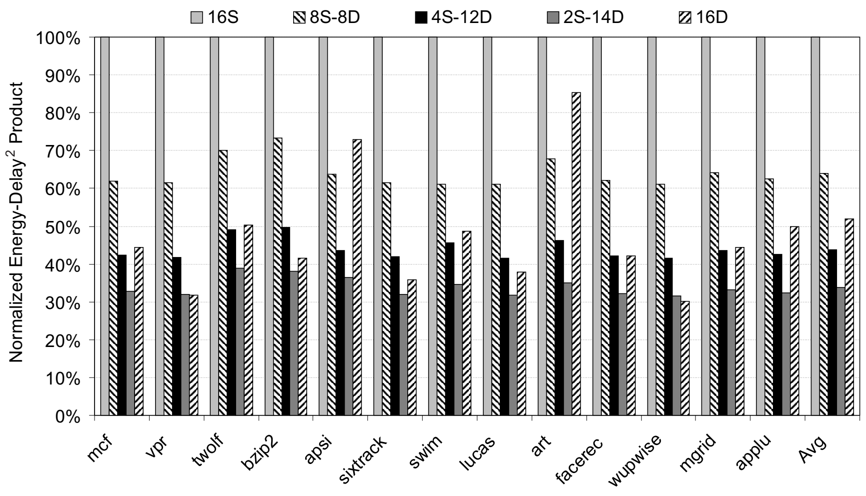
3.3.4 Energy-Delay Squared Product

This section evaluates the trade-off between performance and energy consumption using the ED^2P metric. Figure 3.8(a) and Figure 3.8(b) show the normalized ED^2P of the studied configurations for both 512KB and 1MB caches, respectively (the lower is the better).

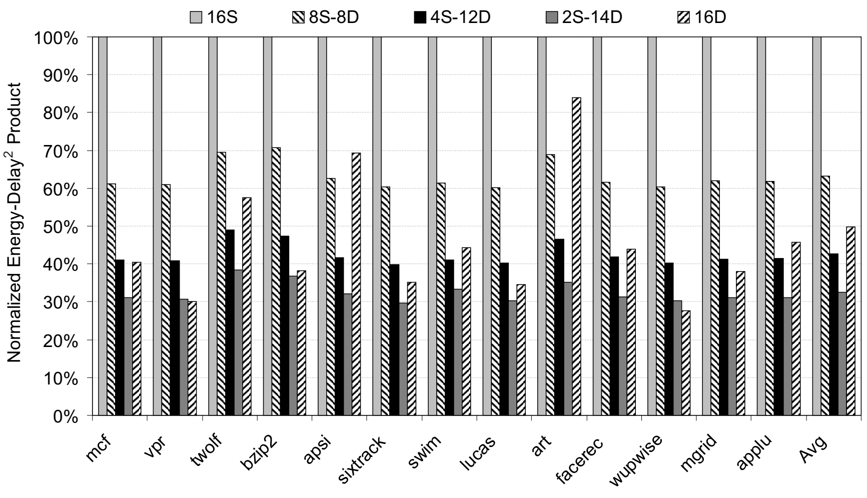
Results indicate that compared to the pure SRAM cache, all the studied configurations reduce the ED^2P despite the lower performance obtained. This is mainly due to hybrid and pure eDRAM caches significantly reduce leakage currents by design. This fact points out the importance of eDRAM-based Last-Level Cache designs, and especially hybrid-based designs, since 4S-12D and 2S-14D present on average a higher reduction of ED^2P compared to the 16D scheme. In particular, for the 512KB cache, the 4S-12D approach reduces ED^2P in 7 of 13 benchmarks over the 16D scheme, while 2S-14D does it in 11 workloads. Although the 8S-8D scheme performs better than the pure eDRAM cache (see Figure 3.6), it does not reduce the ED^2P with respect to this scheme. As mentioned above, this is mainly because the 8S-8D approach consumes a high amount of leakage energy.

On average, regardless of the cache organization, the reduction of ED^2P is quite uniform. It is up to 56–57% and 66–67% in 4S-12D and 2S-14D configurations, respectively. These results are by 48–50% in the 16D approach. Therefore, a hybrid cache design with 12.5% of its banks built with SRAM technology (2S-14D) or 25% (4S-12D) is a better cache design option than a pure eDRAM cache.

Figure 3.9 plots the normalized ED^2P of the 1MB caches with respect to the 512KB SRAM cache. The reduction of ED^2P on the basis of area is not as large as the analyzed reduction on the basis of capacity. The reason is that, although the eDRAM-based caches perform better than 512KB 16S on average (see Figure 3.7), the energy savings



a) 512KB-16way



b) 1MB-16way

FIGURE 3.8: Normalized ED^2P (%) with respect to the pure SRAM approach on the basis of capacity.

are considerably reduced. Moreover, for the 1MB 16D cache, the ED^2P is above 100% in *apsi* and *art*.

Regardless of the cache configuration, *twolf* and *sixtrack* applications achieve a larger ED^2P reduction than in the analysis on the basis of capacity. This fact can be explained by looking at Figure 3.7, where these benchmarks obtain the highest speedups. On average, the ED^2P reduction is by 40%, 54%, and 30% for 4S-12D, 2S-14D, and 16D, respectively.

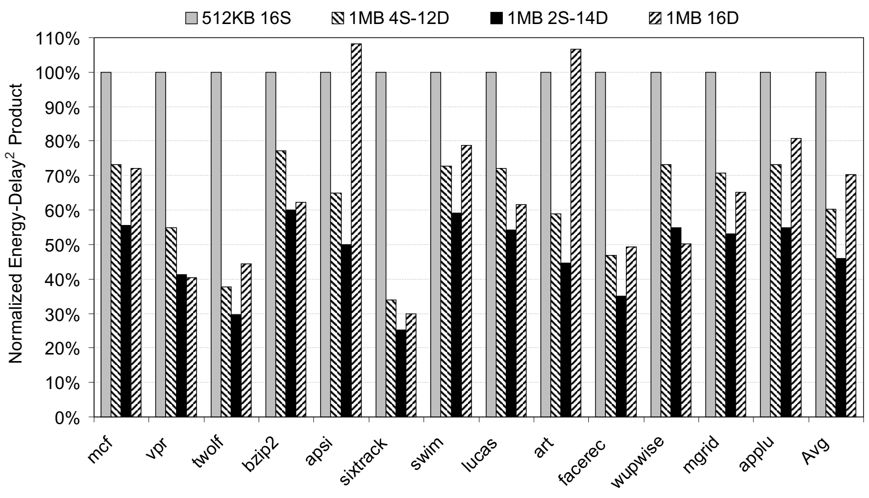


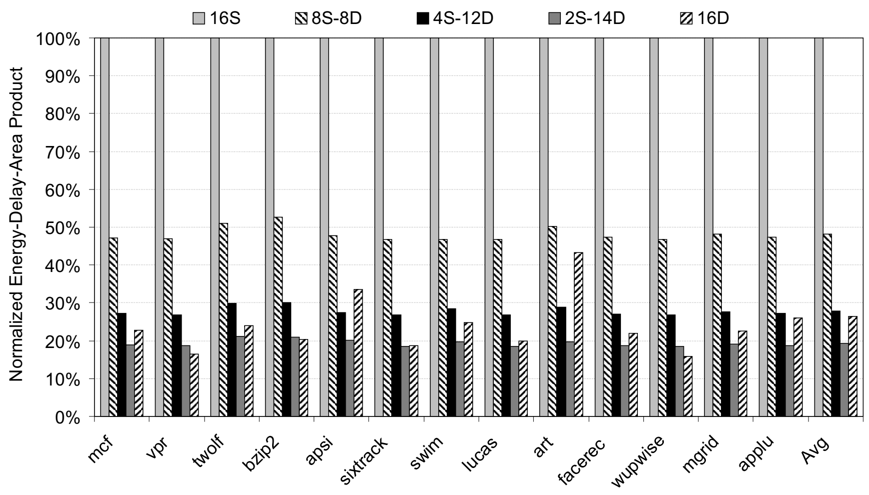
FIGURE 3.9: Normalized ED^2P (%) with respect to the pure SRAM scheme on the basis of area.

3.3.5 Energy-Delay-Area Product

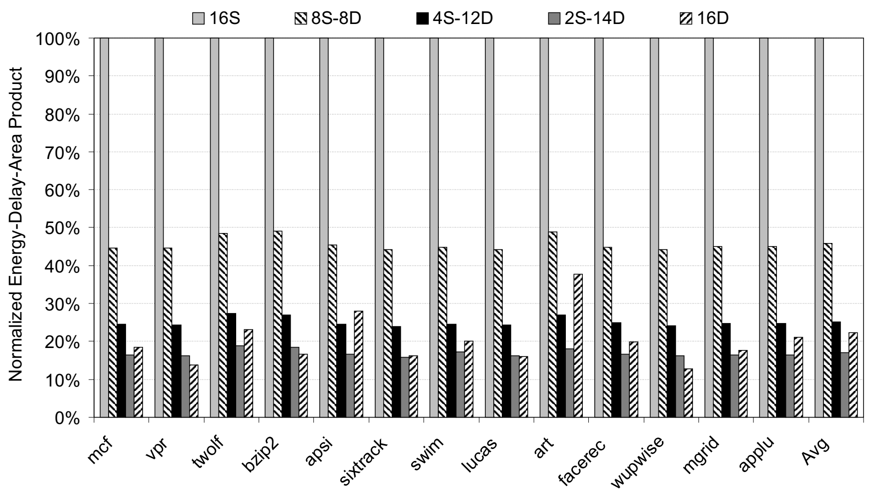
The recently proposed EDAP metric [49] has been used for the sake of completeness in the trade-off analysis. Figure 3.10(a) and Figure 3.10(b) plot the normalized results on the basis of capacity.

Results indicate that all the eDRAM-based caches reduce the EDAP with respect to the 16S design. These reductions are larger than those obtained with the ED^2P metric, confirming that when considering area, which is a major design concern in LLCs, the hybrid approach is even further the best design choice. Similar to the ED^2P analysis, the 2S-14D approach is the best configuration on average. Compared to the pure eDRAM cache, such a hybrid configuration reduces the EDAP in 9 of 13 applications regardless of the cache capacity. These results point out that, on the basis of capacity and taking into account performance, energy, and area, the 2S-14D hybrid design is still the best choice despite the 16D cache provides larger leakage and area savings. The energy-delay-area reduction is on average by 72–75% and 81–83% for 4S-12D and 2S-14D, respectively, depending on the cache organization. These percentages are by 74–78% for the 16D eDRAM cache.

Finally, Figure 3.11 shows the normalized EDAP results on the basis of area. In this case, the reduction is not as large as in the previous study, because the area differences between the eDRAM-based approaches and the conventional SRAM have been relaxed.



a) 512KB-16way



b) 1MB-16way

FIGURE 3.10: Normalized EDAP (%) with respect to the pure SRAM approach on the basis of capacity.

Anyway, the proposed 2S-14D hybrid scheme is the one that most reduces the EDAP. The reductions range on average from 42% to 61% depending on the number of eDRAM ways.

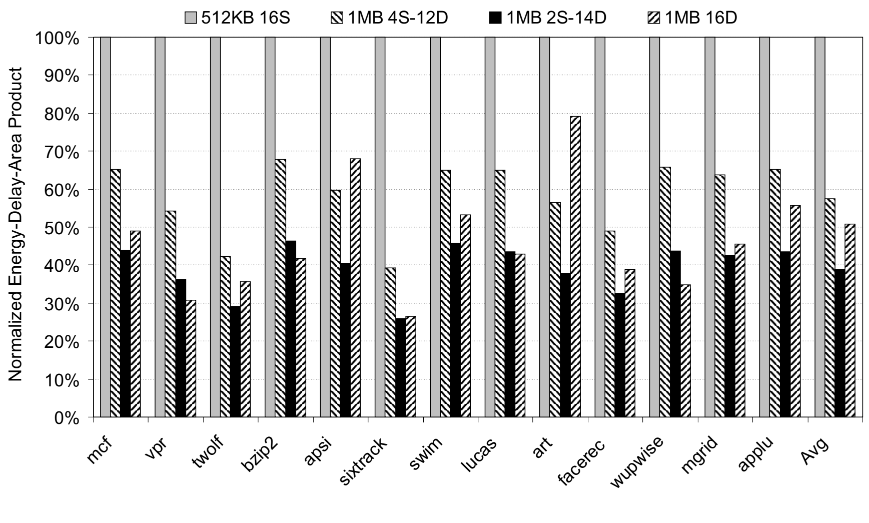


FIGURE 3.11: Normalized EDAP (%) with respect to the pure SRAM cache on the basis of area.

3.4 Contemporary Hybrid Architectures

This section describes recent research works on hybrid architectures that were published during the development of this dissertation.

To take advantage of the properties that each technology offers, the following works have focused on hybrid architectures in different memory structures such as Non-Uniform Cache Architectures (NUCAs), main memories, and multi-threaded register files.

In [16], Wu *et al.* proposed two hybrid NUCA designs: LHCA and RHCA. The former design implements the third-level (L3) cache with eDRAM, MRAM, or PRAM, while both L1 and L2 levels are built with SRAM technology. In the latter design both L2 and L3 caches are flatten into a pair of regions to form a single level. One region is SRAM-based and the other is eDRAM, MRAM, or PRAM-based, whereas the L1 is SRAM-based. The RHCA design requires much hardware complexity than in the proposed hybrid L2 cache to manage data movements between regions, since the design requires not only the LRU stack of all the lines in a set, but also an additional *sticky bit* for the SRAM lines and a 2-bit saturating counter per eDRAM line. Unlike this thesis, there is not a design space exploration varying the size of the SRAM region, which is fixed to 256KB across all the experiments.

Lira *et al.* [50] proposed two different architectures (homogeneous and heterogeneous) for a hybrid eDRAM/SRAM NUCA. In the homogeneous organization, the fast SRAM banks store the frequently accessed blocks and they are placed close to the cores, whereas the eDRAM banks are located in the center of the NUCA. However, this approach is penalized by the shared data, since it is usually located in *slow* eDRAM banks. On the other hand, the heterogeneous architecture balances the number of SRAM and eDRAM banks with the location of them (close to the cores or in the center of the NUCA). Authors argue that the same number of SRAM and eDRAM banks provide the best trade-off among performance, power, and area in this organization.

Qureshi *et al.* [51] proposed a PRAM-based main memory system that includes a DRAM buffer. The requested pages from hard disk and main memory are stored in the DRAM buffer, while the PRAM memory is only written (if required) when the page is evicted from the buffer. The PRAM technology provides higher density with respect to DRAM, while the DRAM buffer allows reduce the number of accesses to the *slow* PRAM memory and the number of write operations to minimize its write endurance problems.

In [52], Yu *et al.* presented an augmented 1-bit SRAM cell with several eDRAM cells, resulting in a multiple-bit eDRAM/SRAM cell to implement register files. The fast SRAM cell is aimed at storing the active context, whereas each pair of eDRAM cells store a dormant context. An additional pair of eDRAM cells is used as a replica of the active context. A dormant context becomes active by transferring the data from the pair of eDRAM cells to the SRAM one. Performance, energy, and area are evaluated using this hybrid cell.

3.5 Summary

In this chapter, both SRAM and eDRAM technologies have been mingled in the LLC, resulting in a hybrid LLC design consisting of SRAM and eDRAM banks. The optimal percentage of SRAM banks has been explored to achieve the best trade-off among performance, energy, and area. The cache controller is enhanced to allow data migration between banks to keep the MRU data in fast SRAM banks. On a cache access, these banks are accessed first to avoid many destructive reads of eDRAM data. This technique is referred to as bank-prediction. A conventional and distributed refresh policy is considered to avoid data losses due to capacitor discharges.

Experimental results have shown that, compared to a conventional SRAM LLC with the same storage capacity and number of ways, performance degradation never exceeds on average 3%, whereas energy and area savings are on average by 69% and 46%, respectively, for a 1MB-16way hybrid cache. Compared to a conventional SRAM cache on the basis of area, the hybrid cache improves performance on average up to 5.9%, while the total energy reduction is by 32%. For a 45nm technology node, the energy-delay squared product confirms that, on average, a hybrid cache is a better design than the conventional SRAM cache regardless of the number of eDRAM banks, and also better than a pure eDRAM cache when the percentage of SRAM banks is 12.5% and 25%. Furthermore, taking into account performance, energy, and area in the trade-off analysis, the hybrid approach with a 12.5% of its banks built with SRAM is still the better design choice.

The work discussed in this chapter has been partially published in [53].

Chapter 4

MRU-Tour Replacement Algorithms

This chapter introduces the family of replacement algorithms for LLCs based on the MRUT concept. First, the shortcomings of the LRU algorithm are discussed, leading to the necessity of alternative algorithms for LLCs. Then, it is presented the MRUT concept and how the baseline MRUT algorithm works. In the experimental section, the performance of the baseline MRUT algorithm is enhanced considering recency of information, adapting the policy to changes in the behavior of the applications, and using a small victim cache for the LLC. In addition, both performance and hardware complexity are compared against a set of the most representative state-of-the-art replacement algorithms.

Bench.	P_{ret}	LRU stack location														
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
swim	24.6	38.2	61.4	0.3	0	0	0	0	0	0	0	0	0	0	0	0
mgrid	69.1	72.6	3.5	0.7	0.9	0.4	0.5	1.0	6.4	0.6	6.8	6.3	0.1	0	0	0
applu	8.4	92.6	4.9	1.3	0.2	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
vpr	59.0	49.4	5.4	0.7	0.5	0.9	2.8	2.5	3.9	3.7	4.9	3.1	3.5	5.2	6.9	7.0
galgel	91.4	85.7	7.2	0.3	0.3	0.3	0.2	0.2	0.2	0.2	0.2	0.1	0.1	0.1	1.2	3.8
art	13.5	9.8	0	0	0	0	0	0	11.1	45.7	7.0	6.0	7.1	5.4	4.7	3.2
mcf	21.2	53.4	16.3	11.6	6.9	3.6	1.9	1.2	0.9	0.8	0.7	0.6	0.6	0.5	0.5	0.5
facerec	47.2	6.7	0.2	0	0	0	0.1	69.2	7.7	3.1	2.5	2.2	1.6	1.9	1.4	3.5
ammp	5.6	4.9	2.4	3.1	3.4	4.5	4.7	5.3	7.0	7.6	7.7	7.9	7.8	9.4	11.1	13.2
lucas	37.5	44.8	15.0	1.6	1.1	1.2	1.7	1.6	2.8	2.9	3.6	3.6	4.7	4.7	5.0	5.8
bzip2	75.1	38.9	13.7	6.8	4.8	4.5	4.5	4.4	4.2	3.8	3.2	2.8	2.4	2.2	2.0	1.9
twolf	84.9	31.1	12.8	8.4	6.8	5.8	5.2	4.6	4.1	3.7	3.4	3.2	3.0	2.8	2.7	2.5
apsi	89.8	37.5	9.1	8.3	1.8	0.4	0.1	0.1	0.1	16.5	20.6	4.7	0.9	0.1	0	0

TABLE 4.1: Probability of a block to be referenced again using the LRU algorithm.

4.1 Weaknesses of LRU

In the LRU algorithm, when a block is not being referenced it descends the LRU stack as other blocks are referenced. During this *walk*, if the block is accessed again, it returns to the MRU position. However, for caches with high associativities, the likelihood of coming back to the MRU position is not uniformly distributed among the locations of the stack. Table 4.1 shows this probability for a 128B-line 1MB-16way L2 (LLC) cache[1]. The P_{ret} column refers to the probability of a block to return to the MRU position. For instance, $P_{ret} = x$ means that a block has a likelihood of $x\%$ to return and $(100 - x)\%$ to be evicted before being referenced again. The other columns indicate the conditional probability of a block residing on a given (non-MRU) stack location given that the block is referenced again (over 100%). Location 0 is the MRU position and it is not represented, while location 15 refers to the bottom of the stack.

As observed, for most applications, when a non-MRU block is referenced it is more likely to stay in locations closer to the MRU (mainly locations 1 and 2). This information is useful to analyze why LRU achieves good or bad performance for a given application, as experimental results will show. For instance, LRU would not work well in applications like *art* or *ammp*, since this probability is higher in lower positions (closer to the bottom) than in the middle or upper positions. Note that blocks located in lower LRU positions are more likely to be evicted and, in such a case, causing misses if they are finally requested.

[1]Some applications have been skipped following the criteria discussed in Section 4.4.

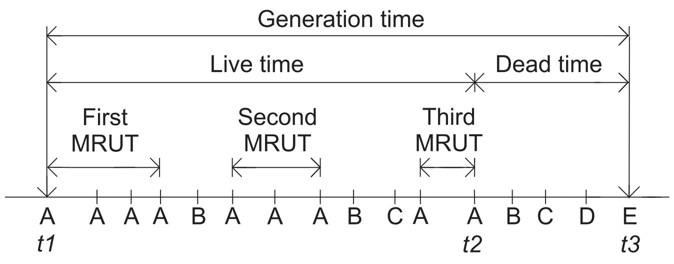


Figure 4.1: Generation time of the cache block A.

Furthermore, these results also show that keeping the whole stack location is not important for most applications in high-associative caches. Nevertheless, the LRU algorithm spends a significant number of bits (e.g, 4 bits in a 16-way cache) to maintain the order of reference for each cache line in the set.

4.2 MRUT Concept and MRUT-based Algorithms

This section presents the MRUT concept and characterizes current benchmarks according to this concept. Then, the family of MRUT-based algorithms is presented, beginning by the simplest one, which will be referred to as baseline.

4.2.1 Overview

The concepts of live and dead times of a block [54] have been widely used in cache research. The generation time of a block defines the elapsed time since the block is fetched into the cache until it is replaced. This amount of time can be divided into live and dead times. The live time refers to the elapsed time since the block is brought until its last access before it is replaced, and the dead time refers to the time from its last access until eviction.

Figure 4.1 shows the concept of MRUT in the context of the live time of the cache block A. Assuming the LRU replacement policy, block A is initially allocated at time *t1* in the MRU position at the top of the LRU stack. The block maintains this position while it is being accessed. Then, the block leaves this position because block B is referenced. At this point, it can be said that block A has finished its first MRUT. After accessing block B, block A is referenced again so returning to the MRU position and starting a

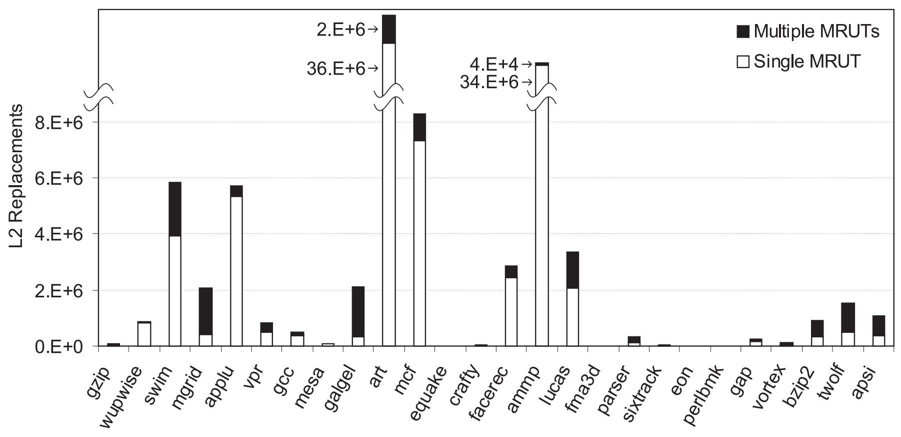


FIGURE 4.2: Number of replacements split into single and multiple MRUTs.

second MRUT. At time *t2*, block *A* finishes its third MRUT, which is the last MRUT of this block before leaving the cache at time *t3*.

To explore the potential benefits of taking into account the MRUT concept on the replacement algorithm, the accessed blocks have been split depending on whether they exhibit a single or multiple MRUTs at the time they are evicted. Figure 4.2 depicts the results for the 1MB-16way L2 cache under the LRU algorithm. As observed, blocks having a single MRUT dominate those having multiple MRUTs, especially in those applications having a high number of replacements.

4.2.2 MRUT-based Algorithms

MRUT-based replacement algorithms exploit reuse information by being aware of the number of MRUTs (one or multiple) for each cache block. Figure 4.3 shows the implementation of the baseline algorithm. This scheme uses one bit attached to each cache block, referred to as MRUT-bit, to indicate if the block has experienced one or multiple MRUTs. This control bit is updated each time the block reaches the MRU position during its live time. The algorithm works as follows. Each time a block is fetched into the cache, its associated MRUT-bit is reset to indicate that the first MRUT has started. When the block leaves the MRU position for the first time, it can potentially be replaced. Then, if the same block is referenced again, it returns to the MRU position and a new MRUT starts. This is indicated by setting the MRUT-bit to one. That is, an MRUT-bit value of one indicates that the block has experienced multiple MRUTs, but no additional bits are included to record how many.

```
Algorithm: Baseline MRUT
Cache hit in block x:
    if (x is not in the MRU position)
       set the MRUT-bit of x to 1
Cache miss:
    a) select the block to be replaced
       if (there are candidates with MRUT-bit=0)
          randomly among candidates (except the MRU)
       else
          randomly among all the blocks (except the MRU)
    b) set the MRUT-bit of the incoming block to 0
End Algorithm
```

Figure 4.3: Baseline MRUT algorithm.

This simple algorithm aims to avoid that those blocks with only one MRUT stay in cache, resulting in a negative impact on performance, by choosing them as candidates for eviction. If a block exhibits good locality, it will come back to the MRU position, so this block will not be considered for eviction. In this way, this approach acts as a catalyst to expel those blocks accessed during only one MRUT. To make hardware simple, the victim block is randomly selected among the candidate blocks except the MRU block. If there is no block with its associated MRUT-bit cleared, all the blocks (except the MRU one) are considered as candidates and the victim is randomly selected among them.

On the other hand, as shown in Table 4.1, storing the order of the last referenced blocks may be also important for performance improvements in most applications. The MRUT-x family extends the baseline algorithm to exploit both the MRUT behavior and also recency of information. The latter is exploited by storing the order of the last x referenced blocks. These x blocks will not be considered as candidates for eviction. For instance, MRUT-2 will not consider as candidates the current MRU block and the following one, while MRUT-1 refers to the baseline MRUT algorithm. Notice that complexity is largely reduced with respect to the LRU policy which keeps the order of all the blocks in the stack.

4.3 MRUT Patterns

In order to improve the performance of the MRUT algorithm, the MRUT patterns of the blocks should be gathered, and it has to be analyzed whether the devised scheme will be able to capture such behavior.

Pattern	Generation time								
	1st	2nd	3rd	4th	5th	6th	7th	8th	9th
Regular-1	1	1	1	1	1	1	1	1	1
Regular-n	3	3	3	3	3	3	3	3	3
Irregular-n	4	4	4	4	3	3	4	4	4
Irregular-1-n	1	1	1	8	1	1	8	1	1

Table 4.2: MRUT patterns. Numbers in the cells indicate the number of MRUTs exhibited during the generation times.

Table 4.2 shows a sample of different patterns that have been found in the applications. Patterns show the number of MRUTs exhibited during nine consecutive generation times. Longer patterns (30 generation times) were analyzed, but only a subset is shown for representative purposes. Patterns' values were gathered with the LRU algorithm. Four different patterns have been identified:

- **Regular-1**. Blocks exhibiting this pattern only have a single MRUT during their live time.
- **Regular-n**. This pattern is presented by those blocks having a constant number of MRUTs but greater than one.
- **Irregular-n**. Blocks showing this pattern have a different number of MRUTs during their live times (cyclic or not) but all of them are greater than one.
- **Irregular-1-n**. This pattern is exhibited by blocks that present a variable number of MRUTs (cyclic or not), some of them equal to one, and some others greater than one.

The proposed algorithm, by design, is able to capture the three first patterns; but it is unlikely that catches the Irregular-1-n pattern. More precisely, if some or most blocks in a given set present an Irregular-1-n pattern, blocks with a potential number of MRUTs might be discarded. In such a case, these blocks would be referenced again soon so incurring in performance losses. Notice that regardless of the patterns' mix, the LRU algorithm would work well if the blocks are referenced again before reaching the bottom of the stack. That is, the LRU performance does not depend on the pattern mix but in the number of active blocks that are being referenced by the program.

Nevertheless, the potential performance loss due to the random component of our proposal can be solved by simply adding a small victim cache for the LLC. In this way,

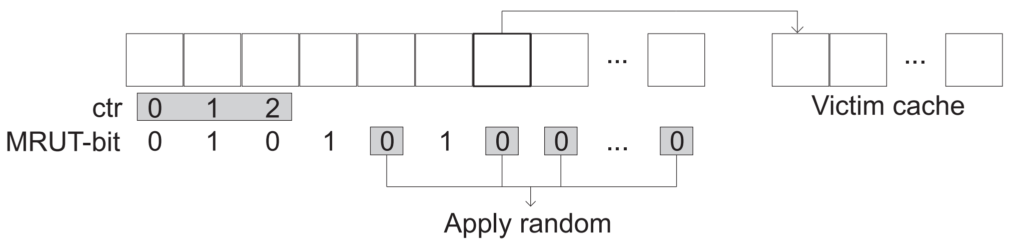


Figure 4.4: Block diagram of the MRUT-3 policy working with a victim cache.

evicted blocks are placed close to the LLC and, if the random strategy fails, these blocks can be quickly returned to the LLC with the MRUT-bit set to one. Figure 4.4 shows a block diagram of the MRUT-3 variant working with a victim cache.

4.4 Experimental Evaluation

This section presents the experimental results of the proposed family of replacement algorithms. The Misses Per Kilo-Instruction (MPKI) metric has been used to quantify the performance of the proposed policies. Results include the dynamic energy consumption and the speedup of the best performing variant of the family when using the victim cache. In addition, this policy is compared against some recently proposed algorithms in terms of performance and hardware complexity. The complexity has been estimated by obtaining the required number of control bits for each policy.

Benchmarks that do not stress the L2 cache have been removed for evaluation purposes; that is, those benchmarks where negligible performance benefits can come from any replacement policy. To this end, the percentage of compulsory misses for each benchmark has been obtained. Figure 4.5 shows the results for the 1MB-16way L2 cache. Applications showing an MPKI value less than one or a percentage of compulsory misses higher than 75% were skipped for this study. The MPKI differences shown among all the replacement algorithms analyzed in this work are less than 0.4 for the skipped benchmarks.

4.4.1 Performance of the Baseline MRUT

This section evaluates the performance of the proposed baseline MRUT algorithm. To this end, its performance is compared against the LRU algorithm. Figure 4.6(a) and

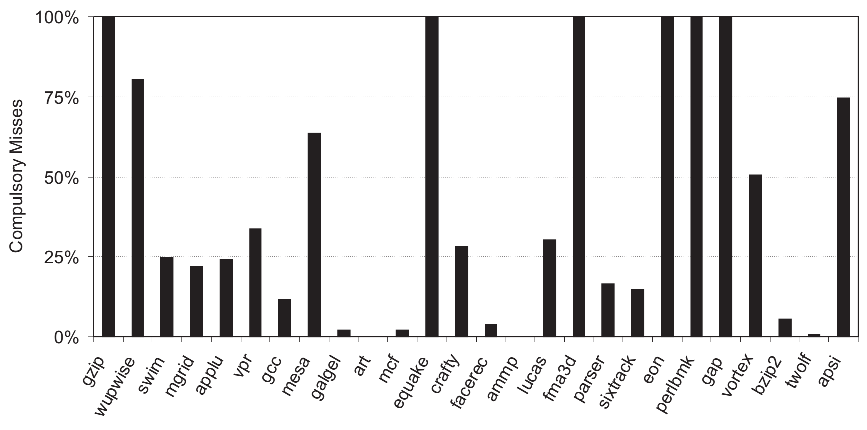


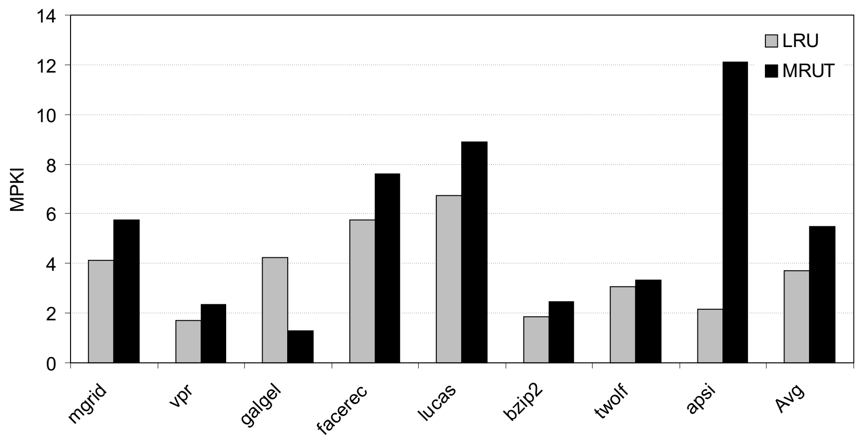
FIGURE 4.5: Percentage of compulsory misses.

Figure 4.6(b) show the MPKI results split into benchmarks with MPKI lower and higher than 10, respectively. The MRUT algorithm shows, on average, the best results and reduces MPKI by 15% compared to LRU. As observed, MRUT performs much better than LRU in applications showing a high MPKI (see average for MPKI>10), while it performs a bit worse in applications achieving low MPKI (see average for MPKI≤10).

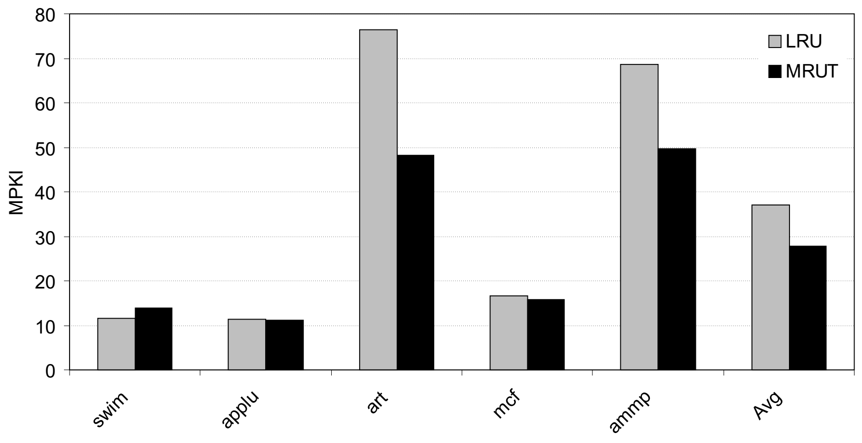
Two cases are worth to be analyzed in detail. The MPKI in *ammp* is around 68.7 and 49.7 in LRU and MRUT, respectively. The poor performance of LRU in this case can be explained by looking at the results presented in Table 4.1. For instance, in the case of *ammp*, blocks have a high likelihood to be referenced again when they are on the lower positions of the stack (e.g., from 8 to 15), where they have a higher probability of being evicted by the LRU algorithm. The behavior of *art* is similar. In this case, the baseline MRUT policy reduces the MPKI by 37% compared to LRU.

4.4.2 Enhancing MRUT Performance

Although the MRUT algorithm works well for some benchmarks, the MPKI is a bit worse than the achieved by LRU in some others (e.g., *facerec*). There are several reasons that could explain this behavior. First, baseline MRUT selects the victim among all the blocks having only one MRUT, regardless of recency of information. Second, blocks change dynamically their behavior with time. For example, a block can have several MRUTs at a given period of time when it exhibits good temporal locality. In such a case, the block will not be considered as candidate for eviction. However, if its locality



a) Benchmarks with MPKI lower than 10

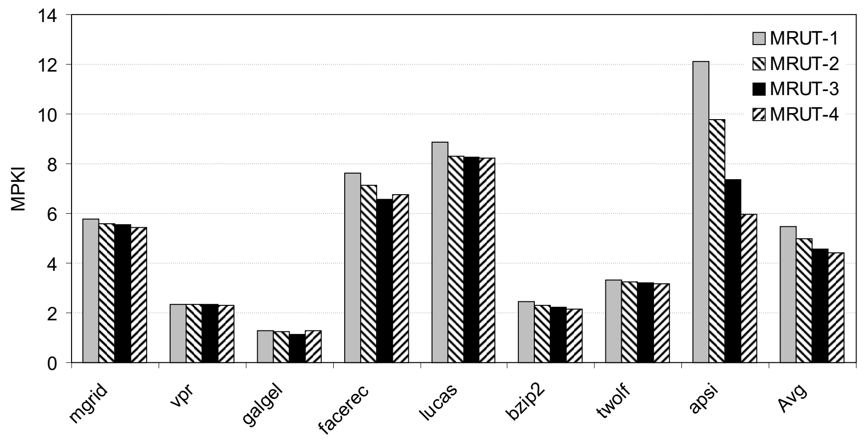


b) Benchmarks with MPKI higher than 10

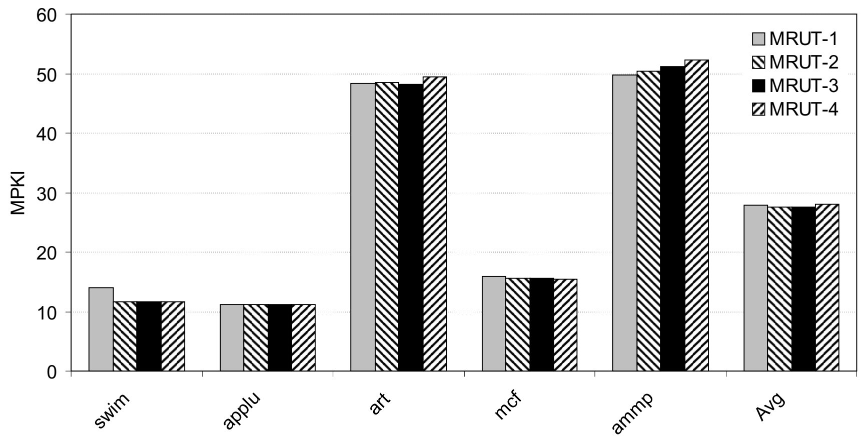
FIGURE 4.6: MPKI of the baseline MRUT and LRU algorithms.

changes or expires with time, the block will not be considered for eviction (unless all the blocks in the set are in the same conditions). Despite this fact, the block is occupying a cache line although it does not exhibit locality any more.

The first problem is attacked by MRUT-x algorithms, where the last x referenced blocks are not candidates to be replaced regardless of their MRUT-bit value. Notice that values of x have been evaluated ranging from 1 to 4 (MRUT-1 refers to the baseline MRUT algorithm). Figure 4.7(a) and Figure 4.7(b) plot the MPKI split into applications with MPKI lower and higher than 10, respectively. MRUT-3 shows on average a slightly better MPKI than the others. Again, the reason can be explained by looking at the



a) Benchmarks with MPKI lower than 10



b) Benchmarks with MPKI higher than 10

FIGURE 4.7: MPKI of MRUT-x algorithms varying x from 1 to 4.

results presented in Table 4.1. Most applications show a high probability to reference blocks near the MRU position (e.g., *swim* and *mcf*). Compared to the baseline and LRU, the MRUT-3 algorithm reduces MPKI on average by 4% and 19%, respectively. Nevertheless, notice that increasing x does not always improve performance. This is the case of *art* and *ammp*. The reason is that the likelihood of accessing one of the last x referenced blocks is quite poor (about 10% for $x = 3$). Thus, if they are not considered for eviction, they can hurt the performance because a block with higher probability of reuse could be wrongly replaced. As opposite, increasing x slightly improves performance on average in those applications showing a low MPKI (e.g., *apsi*). To sum up, MRUT-3 improves on average both MRUT-2 and MRUT-4 replacement algorithms.

Benchmarks	Reset interval				
	32K	256K	2M	8M	∞
mgrid	4.9	4.8	5.3	5.5	5.5
vpr	1.8	1.8	1.9	2.2	2.3
galgel	1.2	1.2	1.1	1.1	1.1
facerec	6.0	5.9	6.3	6.8	6.6
lucas	8.2	8.2	8.2	8.2	8.3
bzip2	2.0	2.0	2.1	2.2	2.2
twolf	3.5	3.2	3.1	3.2	3.2
apsi	2.3	2.4	5.4	6.7	7.3

a) Benchmarks with MPKI lower than 10

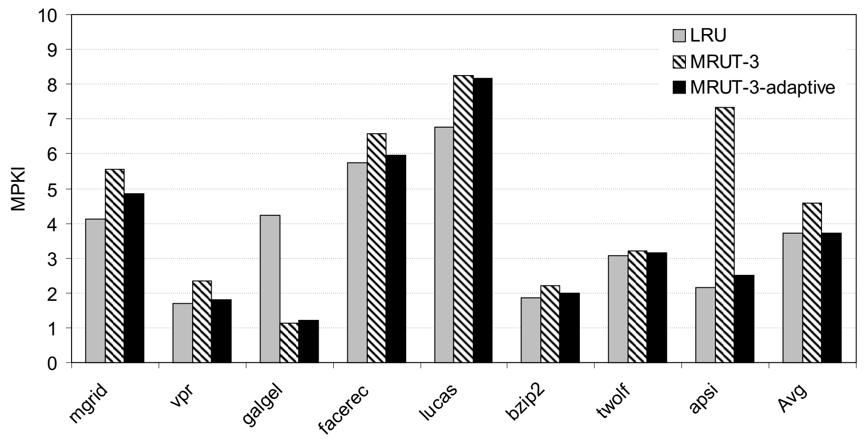
Benchmarks	Reset interval				
	32K	256K	2M	8M	∞
swim	11.6	11.6	11.6	11.6	11.6
applu	11.5	11.4	11.2	11.2	11.2
art	66.8	59.2	50.5	49.2	48.3
mcf	16.5	16.0	15.6	15.6	15.5
ammp	61.5	59.9	58.9	58.1	51.1

b) Benchmarks with MPKI higher than 10

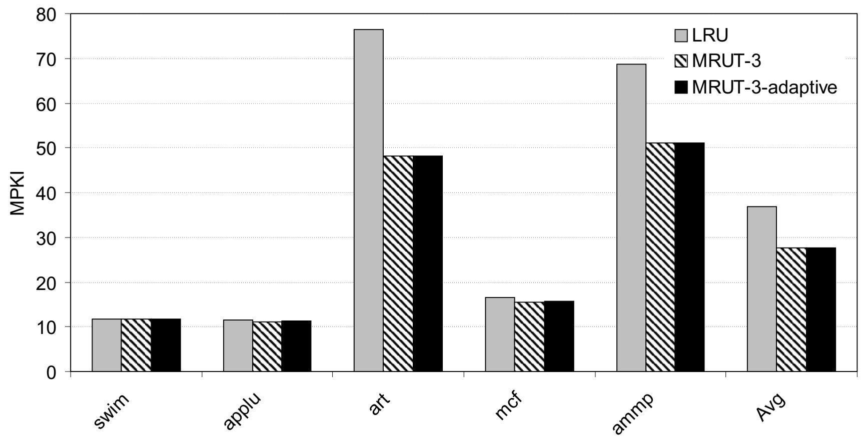
Table 4.3: MPKI of MRUT-3 varying the reset interval.

To deal with the second problem (i.e., block behavior changing with time), the MRUT-bit of each block should be updated at run-time according to its behavior. With this aim, the MRUT-3 scheme has been enhanced to address two main issues: i) reset the MRUT-bit at regular intervals and ii) choose the best interval length at run-time for the next execution period.

The first enhancement pursues to clear the MRUT-bit of a block when its temporal locality expires. Different intervals have been evaluated ranging from 32K to 8M number of committed instructions. Table 4.3(a) and Table 4.3(b) show the MPKI results lower and higher than 10, respectively, achieved by MRUT-3 for each regular interval length. The column labeled as ∞ means that no reset is performed. As observed, 256K instructions is the interval that provides, in general, the best performance for those applications showing an MPKI less than 10. On the other hand, for the remaining applications, the performance increases with the interval length.



a) Benchmarks with MPKI lower than 10



b) Benchmarks with MPKI higher than 10

FIGURE 4.8: MPKI achieved by MRUT-3-adaptive, MRUT-3, and LRU algorithms.

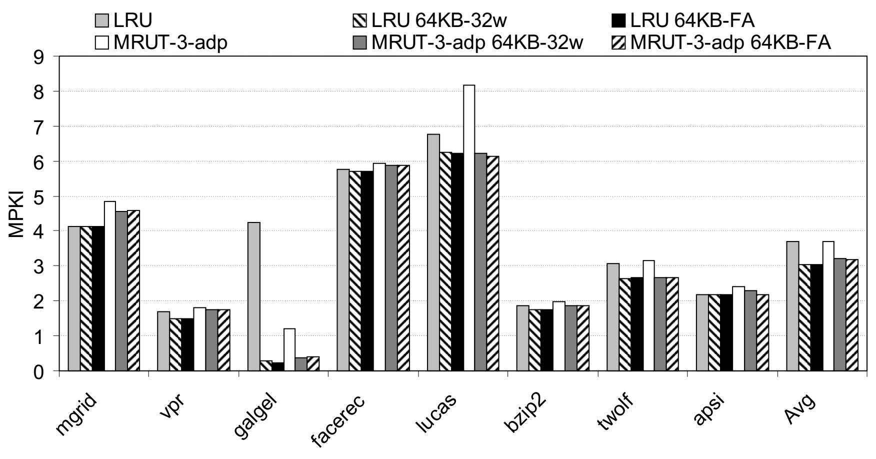
Based on this empirical information, the *reset* algorithm has been extended to be adaptive as follows. The policy resets the MRUT-bits every 256K committed instructions in those workloads with an MPKI less than 10, while MRUT-bits are never reset for the remaining applications. This new policy is referred to as MRUT-3-adaptive. Notice that the number of committed instructions is usually available in the set of performance counters of processors. Thus, storing the LLC misses during a given interval is enough to distinguish if the MPKI is higher or lower than 10. Figure 4.8(a) and Figure 4.8(b) show the obtained results. MPKI values of both MRUT-3 and LRU algorithms have been plotted for comparison purposes. MRUT-3-adaptive shows on average an MPKI reduction by about 3% and 22% compared to MRUT-3 and LRU, respectively.

Notice that this variant of the proposal provides noticeable MPKI improvements on applications with low MPKI like *mgrid*, *vpr*, *facerec* or *apsi*. Results are on average roughly the same as those of LRU for these applications, while they are much better in those benchmarks exhibiting a higher MPKI.

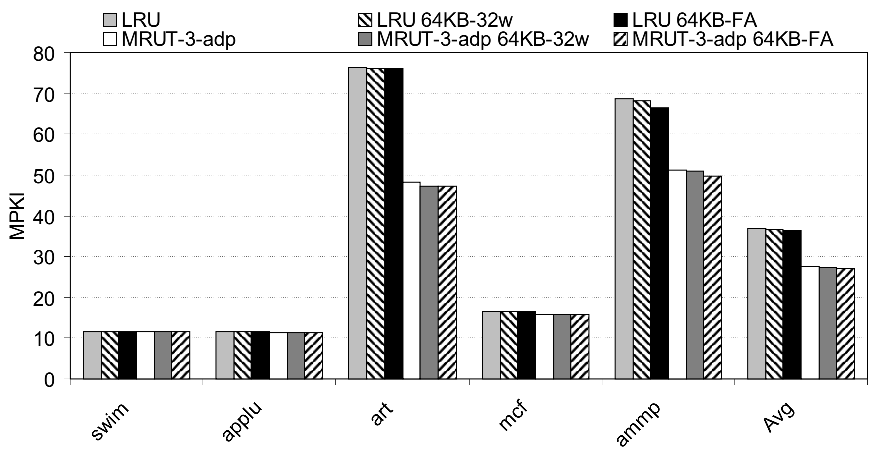
4.4.3 Introducing the Victim Cache

Although the adaptive MRUT algorithm significantly improves on average the performance with respect to LRU, there are still a few applications such as *mgrid* and *lucas* where LRU performs better. To mitigate these performance differences, both policies are evaluated under a victim cache of L2. In this way, evicted blocks due to inaccuracy in the random strategy will have another chance to return to L2 without accessing to main memory. It has been assumed a fixed size of 64KB for the victim cache, which is much smaller (sixteen times) than the size of L2 (1MB in this testbed). Both a full-associative and a 32-way victim caches have been tested. Results of MPKI, hit ratio of the victim cache, speedup, and dynamic energy consumption are presented.

Figure 4.9 shows the MPKI results split into benchmarks with MPKI lower and higher than 10. Remark that, to these MPKI values, the corresponding number of hits in the victim cache have been subtracted. This is done to analyze the impact of the victim cache on the MPKI of L2. As can be seen, the use of a victim cache mitigates the impact of evicting some blocks that are likely to be referenced sooner. Most of these blocks are those exhibiting an Irregular-1-n pattern (see Section 4.3). In particular, MRUT with a victim cache reduces the MPKI of those benchmarks where LRU obtained better results. This is the case of *lucas*, *twolf* and *apsi*, where the MPKI values obtained by MRUT with the victim cache are better than those obtained by LRU with the additional victim cache. In some other cases, although the use of a victim cache does not allow MRUT to outperform LRU, the differences are reduced. This is the case of *mgrid* and *bzip2*. On the other hand, as expected, the use of the victim cache also helps improve the performance of LRU in some benchmarks (*vpr* and *galgel*). Finally, it can be appreciated that it is not necessary a fully-associative victim cache, instead a 32-way cache reaches similar performance with simpler complexity. From now on it is assumed such a victim cache organization.



a) Benchmarks with MPKI lower than 10



b) Benchmarks with MPKI higher than 10

FIGURE 4.9: MPKI of MRUT-3-adaptive working with victim cache.

Figure 4.10 shows the hit ratio of the victim cache (VC) when applying both replacement algorithms in L2. Of course, the applications in which the MPKI is strongly reduced with a victim cache present a high hit ratio. For instance, the best hit ratio for MRUT is achieved by *lucas* (more than 10%), which obtains a reduction in MPKI (see Figure 4.9(a)) from more than 8 to 6. For LRU, the best hit ratio corresponds to *galgel*, which improves MPKI from 4 to less than 0.5. LRU and MRUT-3-adaptive achieve a hit ratio on average by 0.8% and 1.2%, respectively.

Figure 4.11 shows the relative execution time of the analyzed schemes compared to strict LRU (the higher is the better). As observed, although the addition of the victim cache

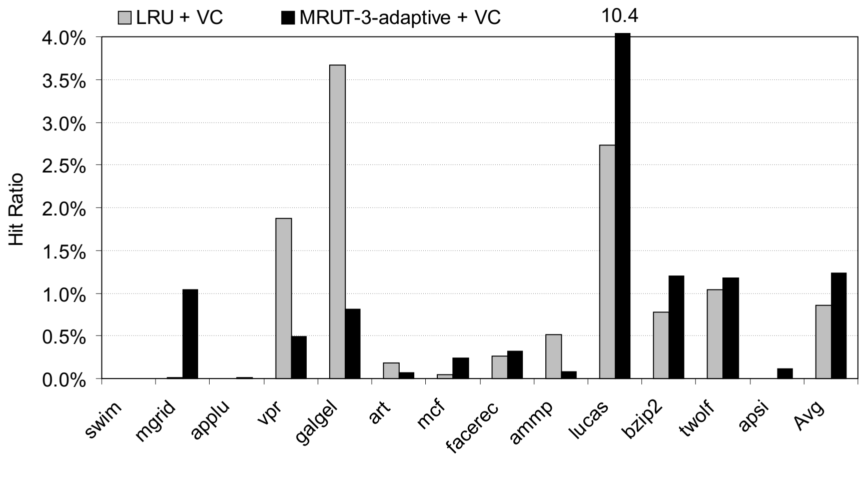


FIGURE 4.10: Hit ratio (%) in the victim cache.

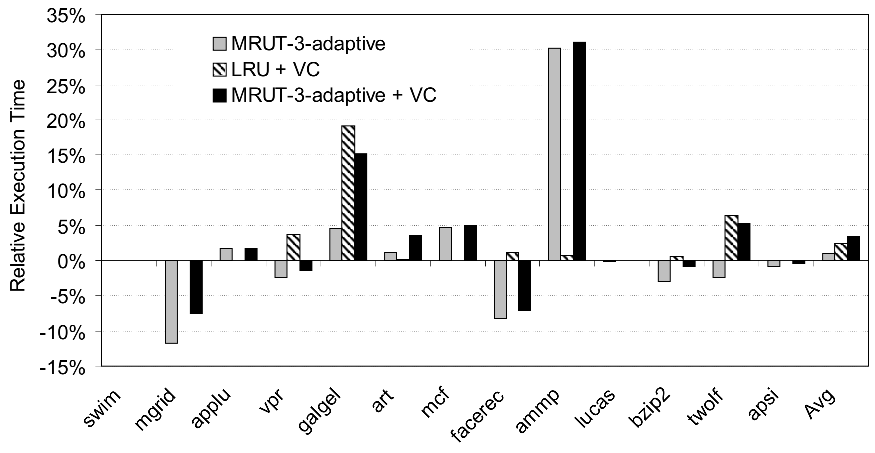


FIGURE 4.11: Impact (%) of adding a victim cache in the relative execution time with respect to strict LRU.

allows improve LRU, this effect is more noticeable in MRUT, which, on average, obtains the best performance. Taking into account that the MRUT replacement algorithm is simpler to implement (see Section 4.5), these results demonstrate that MRUT-3-adaptive replacement algorithm combined with a victim cache is an efficient replacement for LRU in LLCs.

Finally, it has been quantified the dynamic energy consumed by the analyzed replacement algorithms. The CACTI tool was used to compute the dynamic energy per access for each memory structure. It has been assumed a 1MB-16way SRAM LLC, a 64KB-32way SRAM victim cache, and a commodity 1GB DRAM main memory. Main memory energy has been taken into account to estimate the energy costs of satisfying a miss in

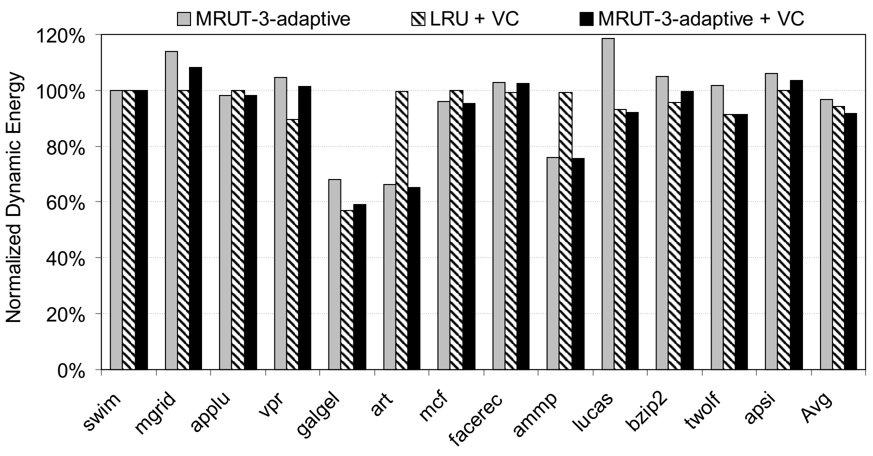


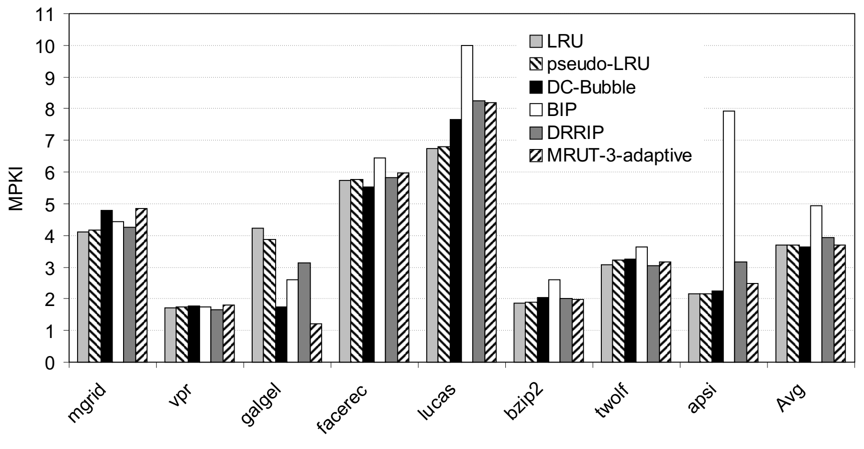
FIGURE 4.12: Normalized dynamic energy (%) with respect to strict LRU.

both the LCC and the victim cache. Results provided by CACTI were used with the number and type of memory operations measured during the benchmark execution to calculate the total dynamic energy of each algorithm.

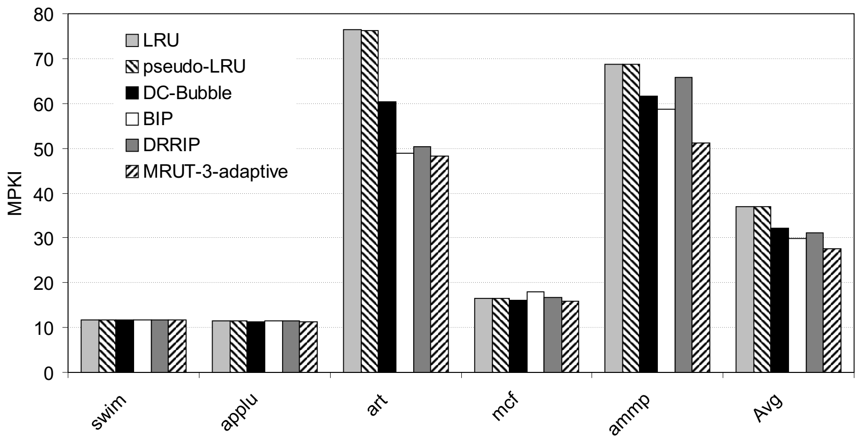
Figure 4.12 depicts the normalized dynamic energy for the analyzed algorithms with respect to strict LRU without victim cache. Regarding the victim cache exempt policies, a miss in the LLC means accessing the power-hungry main memory. Thus, the MRUT-3-adaptive policy achieves more energy savings than LRU in those applications where LLC misses are reduced. The proposed policy reduces dynamic energy by 3% compared to LRU. On the other hand, the schemes with victim cache consume less energy than those without victim cache since a hit in the victim prevents main memory from being accessed. The LRU policy using a victim cache obtains energy savings by 6% on average, whereas this reduction is by 8% in the proposal.

4.4.4 Comparison Against Other Recent Approaches

This section evaluates the performance of MRUT-3-adaptive without victim cache against a set of the most representative state-of-the-art approaches such as Bubble [25], adaptive insertion policies [26], and RRIP [27] (see Section 1.1.2.1). Results for both strict and pseudo-LRU algorithms are also shown in this study. A binary tree-based variant of the pseudo-LRU algorithm has been implemented. Since most recent approaches usually



a) Benchmarks with MPKI lower than 10



b) Benchmarks with MPKI higher than 10

FIGURE 4.13: MPKI achieved by MRUT-3-adaptive, pseudo-LRU, LRU, and other recent policies (DRRIP, BIP, and DC-Bubble).

propose several adaptive variants, the algorithms have been adjusted for this testbed and chosen, for comparison purposes, the best performing variant of each proposal[2].

Figure 4.13 shows the results. As observed, LRU performs slightly better than pseudo-LRU for some benchmarks such as *facerec*, *lucas*, and *twolf*. However, both policies achieve on average almost the same MPKI. On the other hand, the other recently proposed approaches perform better than LRU on average and, like the MRUT-based algorithms, the major benefits are achieved in those applications showing a high MPKI,

[2]The Bubble algorithm uses the DC technique and 4-block groups; the best adaptive insertion policy is BIP with $\epsilon = \frac{1}{32}$; and the 3-bit DRRIP HP with 32-entry SDMs, 10-bit PSEL counter, and $\epsilon = \frac{1}{32}$ is the best cache replacement with RRIP.

whereas the MPKI is similar on average for applications showing an MPKI less than 10. Nevertheless, the MRUT-3-adaptive is the policy achieving more MPKI reduction on average. In particular, it reduces MPKI by 10%, 11%, and 11% compared to DRRIP, BIP, and DC-Bubble, respectively.

4.5 Hardware Complexity

This section analyzes the hardware complexity in terms of area for the studied policies. In addition, results for both counter-based Access Interval Predictor (AIP) and Live-time Predictor (LvP) approaches proposed in [22] are presented (please refer to Section 4.6). It has been assumed that area overhead is mainly dominated by the control bits and additional hardware structures (e.g., tables and counters) required to implement the replacement/placement strategies.

The strict LRU replacement algorithm requires $log_2(n)$ bits per cache block (LRU counters) to maintain the order of the blocks of the LRU stack in an n-way set-associative cache. This results in $log_2(n) \times n$ control bits per cache set. For instance, in a 16-way cache, up to 4 bits per block and 64 bits per set are required. In addition, keeping the order of all the stack requires a circuitry to update the counters, which is expensive in terms of area. In contrast, the pseudo-LRU scheme implemented with a binary tree only uses $n - 1$ bits per set, which is the studied algorithm requiring the least number of control bits.

Concerning the counter-based cache replacement algorithms, the LvP approach saves area with respect to AIP, since the former requires 17 control bits per cache block instead of 21 that uses the latter scheme. However, both approaches have a hardware overhead larger than the required by strict LRU. In addition, both policies use a prediction table to store the counter values of the victimized blocks. This table has 256×256 entries, each one storing 5 bits, resulting in an area overhead of 40KB.

Regarding the Bubble algorithm, it requires as many control bits as LRU, since the entire stack order must be maintained. In addition, a bit per set is required to indicate whether the previous access to the set resulted in a hit or in a miss. Dividing the set in groups helps Bubble to reduce the number of control bits. DC-Bubble requires $log_2(n/g)$ control bits per block, where g is the number of groups in each set. For example, in a

16-way cache, each 4-block group requires 2 control bits per block. Like Bubble, it also uses the aforementioned bit per set to record the previous access result.

In the adaptive insertion policies, the number of control bits of each policy is at least equal to the number required by LRU since they are based on it. LIP does not require more control bits since the only difference with respect to LRU is that the incoming block is placed in the LRU position and then promoted to the MRU. The BIP policy requires as many control bits as LIP, plus a 5-bit counter that is increased on each cache miss to indicate which incoming block is inserted in the MRU position. Finally, the DIP algorithm requires the same amount of bits as BIP, plus a 10-bit saturating counter to implement the *duel* between the competing algorithms.

In contrast to the adaptive insertion policies, the y-bit SRRIP policy uses y control bits per block regardless of the number of ways. On the other hand, the y-bit BRRIP algorithm requires as many control bits as y-bit SRRIP and, similar to BIP, a 5-bit counter. Finally, the y-bit DRRIP policy requires as many control bits as y-bit BRRIP and, analogous to DIP, a 10-bit counter to choose between SRRIP and BRRIP. For example, when $y = 2$, SRRIP, BRRIP, and DRRIP require 32, 37, and 47 control bits per set, respectively, whereas they require 48, 53, and 63 bits, respectively, when $y = 3$.

The baseline MRUT algorithm reduces the hardware complexity compared to the policies analyzed above, since it only requires 2 bits per block regardless of the cache associativity: the MRUT-bit and an extra bit that indicates the MRU block (MRU-bit). In addition, by randomly selecting the victim among the candidate blocks, circuit complexity is largely simplified, not only compared to the LRU policy but also, to the best of our knowledge, with any recent proposal. On the other hand, the MRUT-x family of replacement algorithms maintains the order of the last x referenced blocks using z bits. A special combination of these bits is also used to register that the block order is not kept for the remaining $n - x$ blocks. Thus, $z = \lceil log_2(x+1) \rceil$ MRU-bits per block are required. For instance, MRUT-2 and MRUT-3-adaptive policies need 3 control bits per block (2 bits for the block order and the MRUT-bit).

Table 4.4 summarizes the area overhead (in number of control bits) per cache set for the analyzed policies in an n-way cache. Results for the 128B-line 1MB-16way cache account the total control bits (in Kbits) for the whole cache. As in the previous section, only results for the best variant of each proposal in terms of performance are shown.

Algorithm	n-way cache	1MB-16way cache
LRU	$log_2(n) \times n$	32Kb
pseudo-LRU	$n - 1$	7.5Kb
AIP	$21 \times n$	488Kb
LvP	$17 \times n$	456Kb
DC-Bubble	$log_2(n/g) \times n + 1$	16.5Kb
BIP	$log_2(n) \times n$	32Kb + 5b
3-bit DRRIP	$3 \times n$	24Kb + 15b
Baseline MRUT	$2 \times n$	16Kb
MRUT-3-adaptive	$3 \times n$	24Kb

Table 4.4: Area overhead (in number of control bits) in an n-way cache (per set) and a 1MB-16way cache for the studied algorithms.

As mentioned above, the binary tree implementation permits pseudo-LRU to be the cheapest algorithm in terms of area. However, the performance of this algorithm might be even worse than strict LRU in LLCs. The DC-Bubble is the policy, among the recent proposals, which most reduces the number of control bits since it applies the DC technique that considers each set divided into several groups (i.e., only the order in the group needs to be maintained). Nevertheless, the baseline MRUT algorithm obtains even better reduction in area in spite of considering the entire cache sets. Finally, only keeping recency of information for three blocks is quite enough to occupy smaller area than LRU, AIP, LvP, BIP, and DRRIP policies.

4.6 Reuse Information and Other Replacement Algorithms

There is an impressive amount of research works that attempt to improve the cache performance. This section first introduces some proposals that leverage reuse information. Then, some other recent replacement algorithms are summarized.

Reuse information has been intensively investigated in the past to enhance the cache performance, especially in L1 data caches. Tyson *et al.* proposed the Cacheable/Non-Allocatable (C/NA) cache [55], where the data block placement depends on the program counter of the accessing memory instruction. Data blocks are marked as cacheable or not depending on the reuse behavior of previously accessed blocks sharing the same instruction. Blocks with multiple references are marked as cacheable, whereas blocks referenced only once before eviction are allowed to bypass the cache. In [56], Rivers and Davidson proposed the Non-Temporal Streaming (NTS) scheme, where a block is classified as temporal or non-temporal based on the reuse behavior during its previous generation time. Blocks that are not accessed during a generation time are considered as non-temporal and stored in a separate cache. In this scheme, the data placement depends on the effective address of the requested block. The Memory Address Table (MAT) cache scheme [57], proposed by Johnson and Hwu, also makes use of the effective address for data placement purposes. However, it classifies cache blocks into two groups: frequently and infrequently accessed blocks. The granularity for grouping blocks is augmented with respect to the other proposals, and it is defined as a contiguous group of blocks showing the same usage pattern.

In [23], Lin and Reinhardt propose a couple of hardware approaches that predict when to evict a block before it reaches the bottom of the LRU stack. The first approach is referred to as sequence-based prediction. This approach records and predicts the sequence of memory events leading up to the *last touch* of a block. The second one is the time-based approach, which tracks a line's timing to predict when a line's last touch will likely occur.

The counter-based L2 cache replacement [22] also predicts when to replace a block before it occupies the bottom of the stack. Two approaches were presented: the Access Interval Predictor (AIP) and the Live-time Predictor (LvP). The former bases its predictions by using counters to keep track of the number of accesses to the same set during a given access interval of a cache block. If the counter reaches a threshold value (learned from the previous behavior of the block), the associated block can be selected for replacement. The latter differs from the first one in that it counts the number of accesses to each block instead of to the same set.

The cache burst-based prediction [58] is applied in L1 caches and predicts the number of bursts that a block will exhibit during its live time in L1. Authors argue that such a predictor does not work in L2 caches, since most cache accesses are filtered by the L1 cache and are not seen by L2. As a result of this observation, they propose another kind of predictor (i.e., reference counter-based) for L2 caches. Contrary to this work, in this dissertation it is claimed that the MRUT concept can contribute to performance in L2 caches since it has been found that most blocks have a single MRUT in L2. In addition, it can be done with simple hardware and without help of any assistant predictor.

The instruction-based reuse-distance prediction [59] attempts to relate the reuse distance of a cache line to memory-access instructions in LLCs. These instructions normally access lines that exhibit predictable reuse behavior due to program locality. The predictor uses a history table indexed by the Program Counter (PC) of the instruction, and each entry contains a reuse-distance value and a confidence value for the prediction. To update the predictor at run-time, the approach employs two sampler structures. The replacement policy selects the line to be evicted from those having a long reuse distance prediction (they will be referenced again far into the future) or those that were not accessed in the cache.

In the context of victim caches, the Scavenger [60] scheme consists of an LLC architecture that divides the cache organization in two exclusive parts: a traditional LLC and a victim file. The latter part aims to retain the blocks that most frequently missed the LLC structure. On a miss in the LLC structure but a hit in the victim file, the block is transferred to the LLC structure. On a miss in both LLC and victim file, a Bloom filter is used to keep track of the frequency of misses to the target block address. The replaced block from the LLC is allocated in the victim file depending on its frequency value and the lowest frequency value of the blocks residing in the victim. A pipelined priority heap is used in the victim file to maintain the priority values of all the blocks.

Other state-of-the-art approaches improve the performance of LRU by using modified LRU [61] [62] [63] or a pseudo-LIFO stack [24].

4.7 Summary

This chapter has presented the devised MRUT replacement algorithms for LLCs. The number of MRUTs of a block has been defined as the number of times that the block occupies the MRU position during its live time. Based on the fact that most blocks have a single MRUT when they are replaced using the LRU policy, the proposed baseline policy randomly selects as candidates for eviction those blocks with only one MRUT. Variants of this policy leveraging recency of information for a few blocks and adapting to changes in the working set of the applications have been also evaluated.

Experimental results have shown that the best performing variant achieves MPKI reductions on average by 10–11% and 22% compared to a set of recently proposed algorithms and traditional LRU, respectively. The addition of a small victim cache helps mitigate the negative effect of the random component of the algorithm, and allows the MRUT algorithm to reduce the dynamic energy consumption on average by 8% with respect to LRU without victim cache. Speedup improvements reached by this scheme are on average by 4% and 2% compared to LRU and LRU with victim cache, respectively. Finally, hardware complexity is largely reduced since the stack order is just kept for a few blocks.

The work discussed in this chapter has been published in [64] [65] [66].

Chapter 5

Selective Refresh

This chapter presents the distributed and selective refresh mechanism devised for on-chip caches. This mechanism relies on the MRUT algorithm proposed in the previous chapter, and it is applied in an *energy-aware* eDRAM L2 cache and the hybrid L2 architecture proposed in Chapter 3. First, the hit distribution across the cache ways is analyzed again taking into account the MRUT replacement algorithm. Then, both architectures are introduced together with the proposed selective refresh mechanism. Experimental results cover performance and dynamic energy consumption.

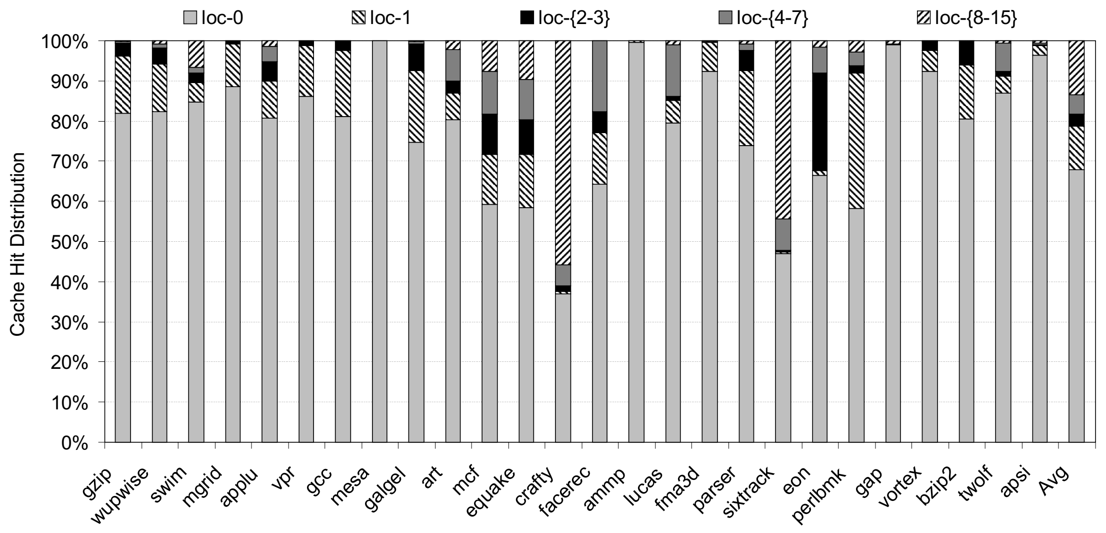


Figure 5.1: Percentage of cache hits across the ways of a 2MB-16way L2 cache using the MRUT policy.

5.1 Energy-Aware eDRAM and Hybrid Cache Architectures

5.1.1 Cache Hit Distribution and Bank-Prediction

This chapter assumes that both cache architectures implement the analyzed MRUT replacement algorithm. Since this policy might affect the percentage of hits across the cache ways, a new study is performed to discern how many ways (or banks) should be predicted to be accessed at the first stage of the bank-prediction technique.

Figure 5.1 plots the results for a 128B-line 2MB-16way L2 cache. Recall that the MRUT policy does not keep all the LRU stack order; however, the LRU counters have been considered to perform this study. Analogous to Figure 3.1, labels *loc-0* and *loc-1* refer to the locations of the LRU stack storing the MRU block and the following MRU block, respectively. Label *loc-15* refers to the position holding the LRU block. Positions from 2 to 3, 4 to 7, and 8 to 15 have been grouped together for illustrative purposes.

Results indicate that, on average, by 68% of cache hits concentrate on *loc-0*. This percentage rises up to 79% when considering an additional position, that is, both *loc-0* and *loc-1*. In such a case, this percentage is above 90% in 16 of 26 benchmarks, while only 6 benchmarks fulfill this condition if only *loc-0* is considered. It has been analyzed the benefits of adding more than a pair of positions to be predicted but, as can be seen in the graph, each additional location provides a small marginal benefit. For instance, including both *loc-2* and *loc-3* only increases the hit ratio on average by 3%. Thus, accessing both *loc-0* and *loc-1*, which correspond to the MRU way and the second MRU way, respectively, during the first stage of the bank-prediction technique is enough to achieve good performance in L2 caches using the MRUT policy. Notice that these results are similar to those presented in Chapter 3, where it was shown that a single SRAM cache bank storing both MRU and second MRU ways (from now on MRU bank) is the best design option in hybrid eDRAM/SRAM L2 caches. In this study it is assumed this cache configuration for the hybrid approach, and the MRU bank will be accessed first during the first stage of the bank-prediction. The only difference between the energy-aware eDRAM and the hybrid cache is that the MRU bank is implemented with eDRAM technology in the former. This cache is proposed with the aim to be compared against the hybrid approach. The tag array of both cache architectures is assumed to be built with SRAM technology.

5.1.2 Coupling the MRUT Algorithm with the Proposed Caches

The results analyzed in the previous section were obtained using the MRUT-2-adaptive policy, which achieves roughly the same performance as the best version of the MRUT family (i.e., MRUT-3-adaptive). The MRUT-2-adaptive has been chosen instead of MRUT-3-adaptive because, for the devised bank distribution with a bank containing the two MRU blocks, the former simplifies the replacement and refresh logic. This proposed implementation works as follows.

The stack order is maintained just for the blocks stored in the MRU bank, and they will not be selected for replacement to leverage recency of information. The remaining blocks are considered as candidates for eviction if they have a single MRUT. Complexity is largely reduced with respect to the traditional LRU algorithm, since blocks in the MRU bank only require the MRU and MRUT control bits (see Section 4.2.2), whereas blocks stored in non-MRU banks do not require any status bit apart from the MRUT-bit. Figure 5.2 depicts a block diagram of the proposed eDRAM cache with a possible set of values of the control bits used by the MRUT policy.

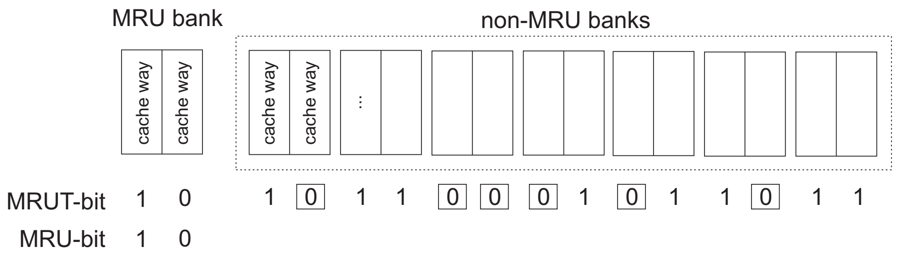


FIGURE 5.2: Diagram of the data array of the energy-aware eDRAM cache with the control bits of the MRUT algorithm.

As done in Chapter 3, the cache controller is enhanced to allow swap operations between blocks to maintain the MRU blocks in the MRU bank. On a cache hit in a non-MRU bank, the target block is swapped with the LRU block of the MRU bank (second MRU block). That is, the target block becomes the MRU and moves to the MRU bank, while the LRU block of the MRU bank is transferred to the non-MRU bank that contained the target block. On a cache miss, the requested block is fetched from main memory and stored in the MRU bank. In such a case, the LRU block of the MRU bank moves to the non-MRU bank which has the victim block according to the MRUT replacement policy. Of course, on a hit in the MRU bank, no data movements between ways or banks are performed; instead the MRU-bit of the MRU bank is updated. Remember that tags are not swapped, meaning that 4 control bits per tag are needed to keep the correspondence between tags and ways in the 16-way cache.

5.1.3 Selective Refresh Policies

Three different refresh policies are evaluated in both energy-aware eDRAM and hybrid caches. The first one, hereafter referred to as *Always*, is the conventional and distributed refresh policy presented in Section 3.2.3. This policy will be considered as the baseline since it avoids capacitor discharges. The two remaining devised policies exploit the MRUT concept for saving energy. These selective refresh policies are referred to as *Conditional* and *Adaptive*. The three policies work as follows:

- **Always**. This policy always refreshes the target block regardless of the value of the control bits.
- **Conditional**. Refresh is only applied if the following condition is satisfied: the target block is stored in the MRU bank or its MRUT-bit='1' (multiple MRUTs).

Otherwise the block is marked as *invalid* and written back to main memory if dirty (*early* writeback).

- **Adaptive**. This policy dynamically adapts between *Always* and *Conditional* at run-time.

The *Conditional* policy aims to reduce refresh energy wasting with respect to the baseline without refreshing all the cache blocks. It allows data losses in those blocks less likely to be useful. Since this policy is speculative, it can yield to performance degradation on misspeculation because of, in such a case, the block must be fetched from main memory if it is requested after capacitors discharge. The *Conditional* mechanism uses both the MRU-bit and the MRUT-bit to decide whether the refresh should be performed or not. In the case of a high prediction accuracy, it would achieve substantial energy savings with minimal performance loss. However, if prediction accuracy is low, this policy would yield to severe performance drops.

The *Adaptive* policy dynamically selects which of the previous policies should be applied depending on the cache behavior. This policy requires a pair of counters to track the number of standard cache misses x as well as the number of cache misses y that hit in the tag array but the associated data line has lost its contents. The latter occur when the requested block has been previously invalidated using the *Conditional* refresh mechanism.

Both counters are initially reset to zero when a sampling period starts. A sampling period finishes when the x counter reaches a given value (e.g., 128). At that point, if the y counter exceeds a given threshold (e.g., 8), the *Always* refresh policy is applied during the next period. Otherwise the *Conditional* policy is selected. Then, both counters are reset and a new period starts. Threshold values of x and y counters are defined as a power of two to make hardware simple. That is, it is enough to check just a single bit of each counter to decide which policy has to be applied during the next period.

Notice that, for the hybrid cache, the refresh period is longer than that of the energy-aware eDRAM, since blocks located in the MRU bank do not have to be refreshed, so reducing bank contention. In addition, contention due to restoring the contents after destructive read accesses is also reduced.

Finally, in both eDRAM and hybrid proposals, the contents of the MRU bank are never lost, since in the former approach they are periodically refreshed (independently of the refresh policy) while in the latter design these contents are stored in SRAM technology.

5.2 Experimental Evaluation

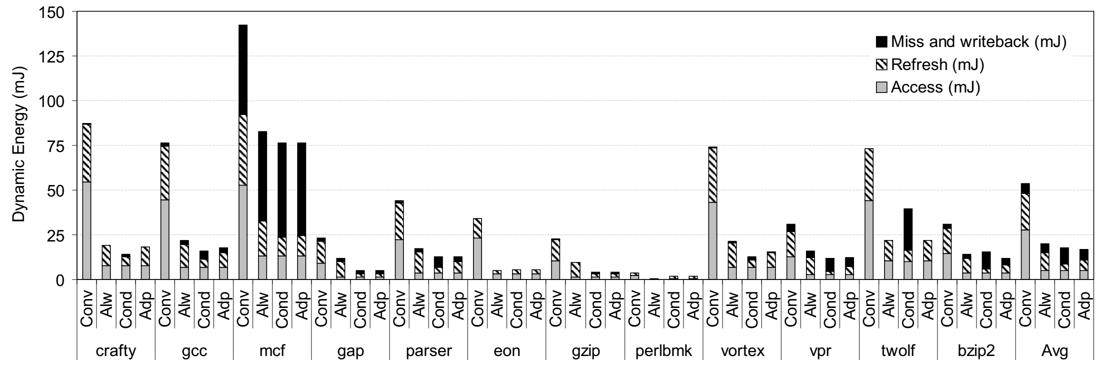
This section evaluates the performance and energy consumption of the proposed selective refresh policies when they are applied in both energy-aware eDRAM and hybrid L2 caches.

Figure 1.3 showed (see Chapter 1) how the refresh energy overhead increases with the cache capacity. Because of this reason, it has been considered a 2MB-16way L2 cache for evaluation purposes. According to CACTI, the access time of such a cache organization is 2, 8, and 12 cycles for the tag array, SRAM banks, and eDRAM banks, respectively.

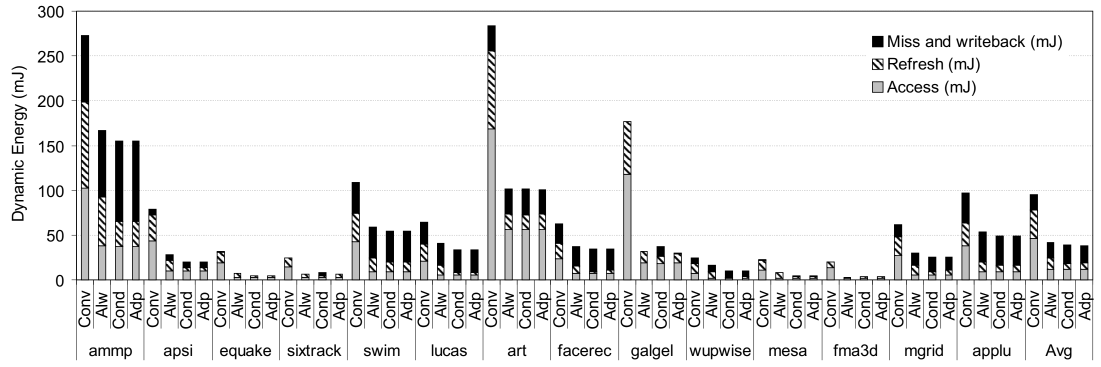
For comparison purposes, both conventional SRAM and eDRAM cache schemes were also modeled. The latter applies the *Always* distributed refresh policy. That is, no mechanism to improve refresh energy is considered. These caches access in parallel the tag array and all the cache ways of the data array. Like in both energy-aware and hybrid proposals, the tag array is assumed to be implemented with SRAM cells.

5.2.1 Energy Consumption

This section evaluates the dynamic energy savings of the proposed refresh policies for a 45nm technology node. The dynamic energy has been broken down into expenses related to three major components: i) accessing the L2 cache (*Access* consumption), ii) refreshing the L2 stored data (*Refresh* consumption), and iii) fetching blocks to L2 from main memory and L2 writebacks to main memory (*Miss and writeback* consumption). The first component covers the energy spent in the access to the L2 cache, including swap operations between banks. The L2 refresh consumption takes into account the expenses due to refreshing the contents after read accesses and the periodic refresh operations. Finally, the latter takes into account the energy consumed by a commodity 1GB DRAM main memory due to L2 misses and L2 writebacks. The number of these L2 memory



a) Int benchmarks



b) FP benchmarks

Figure 5.3: Dynamic energy (in mJ) of the conventional eDRAM cache and the energy-aware eDRAM cache for the studied refresh policies.

events differs among the studied refresh policies since they present different prediction accuracy.

Figure 5.3 shows the dynamic energy (in mJ) consumed, for Int and FP benchmarks, by the energy-aware eDRAM cache working with the proposed refresh policies: *Always* (labeled as *Alw* in the graph), *Conditional* (*Cond*), and *Adaptive* (*Adp*). The energy consumption of the conventional eDRAM cache using *Always* refresh (*Conv*) is also included for comparison purposes.

As observed, for a given refresh policy, results widely differ across benchmarks due to two main reasons. First, both *Access* and *Miss and writeback* consumption depend on the number of accesses to L2 and main memory, respectively. The higher the number of accesses the higher the energy consumption. Second, applications achieve different

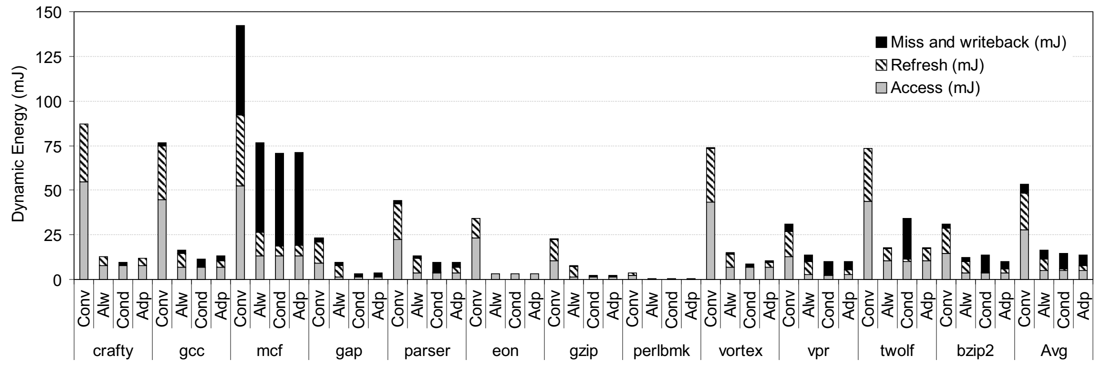
IPCs so they have different execution times. The lower the IPC (longer execution time) the higher the energy dissipated by refresh operations.

Compared to *Conv*, the consumption of the three policies is largely reduced. Note that for most benchmarks, only the *Access* consumption of the conventional eDRAM cache exceeds the total dynamic energy consumption of the proposed eDRAM approach. This means that, although the proposed cache wastes energy in swap operations, this consumption is minimal compared to the benefits brought by the bank-prediction mechanism. These benefits come from two sides: i) only the MRU bank is accessed first and ii) less accesses and refresh operations are carried out in non-MRU banks since only the target non-MRU bank is accessed after the tag comparison.

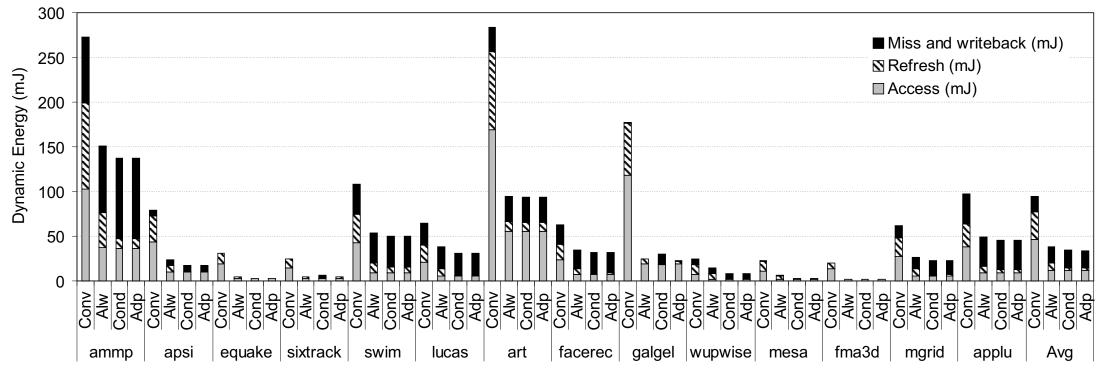
Conv and *Alw* refresh policies present the same consumption in the *Miss and writeback* component since they always refresh the cache blocks. On the other hand, the *Cond* policy significantly increases the consumption of this component due to misses and *early* writebacks caused by non-refreshed blocks. This increase does not compensate on average the refresh energy savings of *Cond*; thus, this policy increases the total energy consumption with respect to the *Adp* refresh policy, meaning that *Adp* is the most energy-efficient refresh method.

The refresh energy savings of *Alw* and *Adp* are on average by 57% and 75%, respectively, compared to *Conv.* The refresh reduction of *Alw* comes from the bank-prediction mechanism, while the savings of *Adp* are given both by bank-prediction and selective refresh. Taking into account the three components, the overall energy savings of *Alw* are on average by 58% with respect to *Conv*, while the *Adp* method reduces the overall energy by 63%. This percentage can be as high as 84% in some applications.

Different threshold values for both x and y counters were analyzed for the *Adp* policy. The presented results were obtained with a threshold of 128 and 8 for x and y, respectively, since it was found that this pair is the most energy-efficient. Increasing the y threshold reduces the *Refresh* consumption, but the *Miss and writeback* component increases much more due to additional induced misses and writebacks. In contrast, a smaller threshold value makes the behavior of *Adp* closer to that of *Alw.* For the x threshold, larger values allow the *Cond* refresh to be applied for a long period, which



a) Int benchmarks



b) FP benchmarks

FIGURE 5.4: Dynamic energy (in mJ) of the conventional eDRAM cache and the hybrid cache for the proposed refresh policies.

may yield to severe performance drops. On the contrary, lower values shorten the sampling period in such a way that the mechanism does not have enough information to decide which is the most appropriate policy for the next period.

Figure 5.4 shows the dynamic energy consumed by the hybrid cache. Compared to the results presented in Figure 5.3, the *Access* and *Miss and writeback* expenses are quite similar in both proposed eDRAM and hybrid caches for a given refresh method. This is because both approaches implement the devised bank-prediction and the swap mechanism. Minor differences appear because the MRU bank is built with distinct technologies.

On the other hand, the hybrid cache reduces the refresh consumption. While the refresh savings of *Alw* applied in the energy-aware eDRAM cache are on average by 57% with

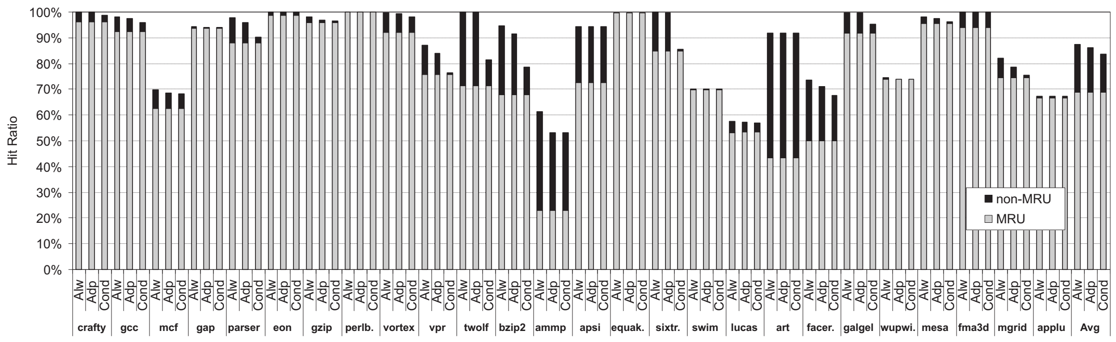


FIGURE 5.5: Hit ratio (%) split into hits in the MRU and non-MRU banks of the energy-aware eDRAM cache for the studied refresh policies.

respect to *Conv*, this percentage is up to 72% for the hybrid approach. This is due to the SRAM-based MRU bank prevents the stored blocks from being refreshed, either by a read access or by the periodic refresh mechanism. The *Refresh* consumption is further reduced by *Adp* and *Cond*. Compared to *Conv*, *Adp* and *Cond* achieve refresh energy savings by 89% and 94%, respectively.

Taking into account the whole dynamic energy consumption, *Alw* saves on average by 63% the consumption of the *Conv* policy. Similar to the energy-aware eDRAM proposal, *Adp* is the most energy-efficient refresh method. This policy improves the overall energy savings by 68%, while this percentage is up to 92% in some benchmarks.

5.2.2 Performance

Figure 5.5 presents the cache hit ratio of the energy-aware eDRAM design, splitting it into hit ratio in MRU and non-MRU banks. Notice that the hit ratio of *Alw* matches the hit ratio of *Conv*, since both refresh methods avoid capacitor discharges.

As observed, on average, the MRU hit ratio is around 70%. This percentage is above 80% in half of the applications, so confirming the effectiveness of the bank-prediction technique and the swap mechanism. In other words, most cache accesses hit the MRU bank at the first stage.

An interesting observation is that the MRU hit ratio remains constant for each benchmark regardless of the refresh policy. This is due to the placement/replacement strategies are the same in the studied refresh methods and blocks stored in the MRU bank are always refreshed.

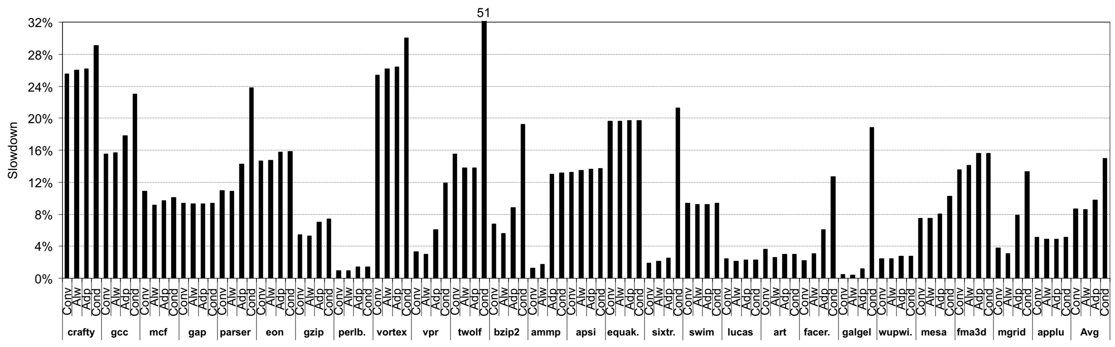


FIGURE 5.6: Slowdown (%) of the conventional eDRAM cache and the energy-aware eDRAM cache for the proposed refresh policies with respect to the conventional SRAM cache.

As expected, the non-MRU hit ratio of the *Cond* policy is lower than that of the *Alw* policy. This is because a significant amount of non-MRU blocks are not periodically refreshed in *Cond*. This situation yields to performance degradation if non-refreshed blocks are required later since, in such a case, their contents must be retrieved from main memory. For this reason, noticeable non-MRU hit ratio differences appear in some benchmarks (9 of 26) like *parser* or *galgel*, which yield to unacceptable performance as shown in Figure 5.6. In contrast, the non-MRU hit ratio differences between *Alw* and *Adp* are rather low with the only exception of *ammp* (by 8.1%).

Moreover, in some benchmarks such as *apsi* and *art*, the non-MRU hit ratio remains almost constant regardless of the refresh policy, which means that most non-MRU blocks in these applications exhibit good locality (i.e., multiple MRUTs) and they are periodically refreshed.

Figure 5.6 shows the slowdown of the energy-aware eDRAM scheme with respect to a fast conventional SRAM cache that accesses the tag array and all the ways of the data array in parallel. In addition, performance of the conventional eDRAM cache applying *Always* refresh (*Conv*) is also shown for comparison purposes. As can be seen, on average, the performance loss of *Conv* (by 8.7%) is roughly the same as that of the proposed eDRAM cache with *Alw* refresh (by 8.6%). However, differences appear in individual benchmarks. For example, in some applications like *crafty*, *Conv* slightly outperforms *Alw*. This is mainly due to the latter has to wait for the tag comparison before accessing the target non-MRU bank. On the other hand, in some other benchmarks such as *mcf*, *Alw* obtains better results than the conventional cache. This is because the latter presents higher bank contention since on each cache access, all the cache banks are accessed in parallel.

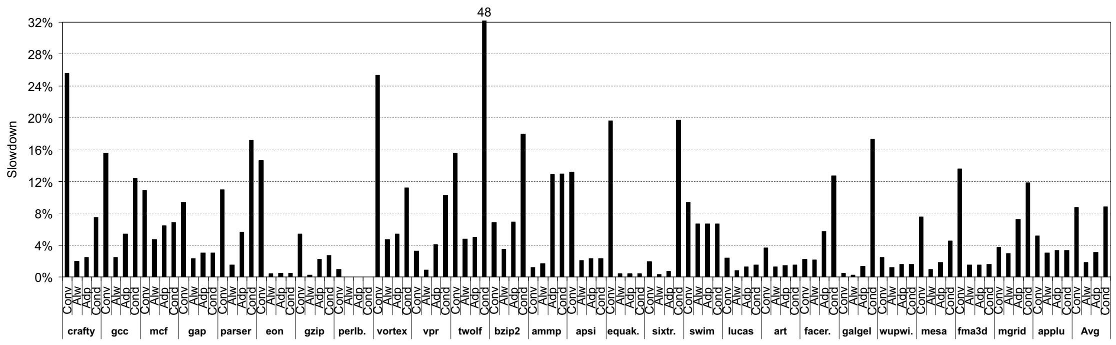


FIGURE 5.7: Slowdown (%) of the conventional eDRAM cache and the hybrid cache for the proposed refresh policies with respect to the conventional SRAM cache.

Thus, a given memory request may be stalled by a previous access, independently of the target bank.

As explained above, performance differences between *Alw* and *Cond* refresh policies are mostly caused by L2 misses due to misspeculation in *Cond*. These performance drops are highly reduced by the *Adp* policy. Nevertheless, in some benchmarks, the slowdown is too high even for the *Alw* policy.

The proposed hybrid design tackles the performance problems of the energy-aware eDRAM cache by implementing the MRU bank with fast SRAM technology. In addition, the period between two consecutive refresh operations increases, so alleviating bank contention. Figure 5.7 presents the slowdown of the studied policies with respect to the conventional SRAM cache.

As expected, *Alw* is the refresh method with the lowest slowdown. Its performance loss is lower than when the same refresh policy is applied in the energy-aware eDRAM cache and *Conv* (see Figure 5.6). On average, it is by 1.8%, and never exceeds 6.6% (reached by *swim*) in any application. These results confirm that a hybrid cache built with a single SRAM bank composed of a couple of cache ways is enough to achieve *near-zero* slowdown as well as that the swap operation does not substantially impact on performance.

As studied in the energy-aware eDRAM approach, both *Cond* and *Adp* policies increase the slowdown with respect to the *Alw* policy. Nevertheless, the slowdown of *Adp* in the hybrid design is on average only by 3.1%.

5.3 Refresh Mechanisms in Off-Chip Memories

Prior research works that have attacked the refresh energy consumption in off-chip DRAM devices can be classified into three main categories: i) those works that exploit the property that normal accesses implicitly refresh the contents after a destructive read, ii) those works that adapt the refresh period according to the variability of the retention time, and iii) those works that do not refresh useless data.

The Smart Refresh [28] proposed by Ghosh and Lee falls into the first category. This scheme employs a time-out counter for each DRAM row, which counts from its maximum value down to zero within the refresh period of the DRAM row. Whenever the row is accessed or refreshed, the counter is set to its maximum value since the charge has been fully restored, and every time that the counter reaches zero the refresh is performed.

Some works fall into the second category [12] [30] [31] [29]. Liu *et al.* [12] proposed a refresh mechanism, referred to as Retention-Aware Intelligent DRAM Refresh (RAIDR). This mechanism is based on the observation that most DRAM cells are able to hold their data for more than 64ms, which is the common refresh interval in DRAM. The cache controller employs Bloom filters to group DRAM rows having the same retention time. A different refresh rate is applied to each group of rows.

In [30], Venkatesan *et al.* proposed the Retention-Aware Placement In DRAM (RAPID) mechanism. Three variants of this mechanism were presented. The simplest approach, namely RAPID-1, discards memory pages with a short retention time from being populated. This yields to a longer overall refresh period for the populated pages at the expense of performance degradation. The second approach, namely RAPID-2, separates the memory into different regions (bins) depending on the associated retention time. Bins with long retention times are filled first. When all bins have been occupied, freeing bins with short retention times is a matter of chance, since it depends on the application. Finally, the third approach, referred to as RAPID-3, allows data migration among bins. That is, pages stored in *shorter bins* are transferred to *longer bins* when space is available.

Kim and Papaefthymiou [31] use multiple refresh periods for each DRAM array according to the retention time of the associated DRAM cells. ECC is used to restore corrupted cells and thus allowing extended refresh periods. Authors propose an algorithm that calculates optimal refresh periods for a given DRAM array.

Ohsawa *et al.* [29] proposed a pair of DRAM refresh architectures, namely Variable Refresh Architecture (VRA) and Selective RA (SRA), which belong to the second and third category of approaches, respectively. The VRA approach determines the refresh period according to the retention time of the DRAM rows. To reduce hardware complexity, this strategy is applied in regions of several rows, each one having a different refresh period. The SRA approach requires a control bit per DRAM row to indicate which ones should not be refreshed. These rows are those containing useless data. Authors propose SRA for a writeback cache memory system, where rows in off-chip DRAM memory are marked as useless if the same data block stored in cache is dirty, which means that this data block has to be written back to main memory.

5.4 Summary

This chapter has introduced the distributed and selective refresh mechanism for on-chip caches. This mechanism exploits reuse information by using the MRUT replacement policy. The periodic refresh operations of the distributed mechanism are skipped in those blocks candidates to be evicted by the MRUT policy. That is, the proposed selective mechanism allows data losses in those blocks exhibiting poor locality (i.e., they have a single MRUT). This leads to dynamic energy savings at the expense of performance degradation if these blocks are requested later.

The proposed refresh mechanism was applied in both an energy-aware eDRAM and hybrid eDRAM/SRAM L2 caches. The latter approach was built with eDRAM banks except the MRU bank which is SRAM-based. Like the hybrid cache, the former eDRAM cache also implements the bank-prediction technique and keeps the MRU data in a single bank by performing swap operations. In both approaches, blocks stored in the MRU bank are never selected as candidates to be replaced by the MRUT policy. As the selective refresh can be quite aggressive given the high number of blocks with a single MRUT, it is enhanced to adapt between the selective and a conventional mechanism. The selective or conventional refresh policies are selected at run-time whenever the number of cache misses due to data losses exceeds a given threshold.

Experimental results have shown that, compared to a conventional eDRAM cache, the proposed energy-aware and hybrid caches reduce the refresh energy on average by 75% and 89%, respectively, when the devised selective refresh is applied. These percentages are by 63% and 68%, respectively, when the overall dynamic consumption is considered. Compared to a conventional SRAM cache, the performance loss of the hybrid cache is only on average by 1.8% when the conventional refresh is applied. For the proposed refresh policy, this percentage is by 3.1%, which is much lower than the performance degradation of both conventional eDRAM (by 8.7%) and energy-aware eDRAM (by 8.6%) using the conventional refresh method.

The work discussed in this chapter has been partially published in [67].

Chapter 6

Conclusions

This thesis has proposed hybrid eDRAM/SRAM cache architectures as alternatives to conventional SRAM and eDRAM designs in both L1 and L2 caches (LLCs) of the memory hierarchy. In addition, this dissertation has presented a family of replacement algorithms for LLCs based on the MRUT concept that improve the performance of traditional LRU with much lower hardware complexity. Finally, the hybrid L2 cache architecture and the MRUT policy have been combined to reduce the number of refresh operations in on-chip eDRAM-based caches. All these architectured solutions have been implemented and evaluated on top of extensively used simulation tools. 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

In Chapter 2, the macrocell device consisting of one SRAM cell and *n-1* eDRAM cells has been presented. The macrocell has been used to implement n-way set-associative L1 data caches having one SRAM way and *n-1* eDRAM ways. Such a memory design largely reduces both leakage and area thanks to the use of eDRAM technology. Architectural mechanisms were considered to sustain the performance of these memories and to avoid costly refresh logic. In this way, by accessing just the SRAM way first, many destructive reads of eDRAM data are prevented. Swap operations between ways ensure that the MRU blocks are always stored in the fast SRAM way. The performance degradation with respect to a conventional SRAM cache is minimal given the confirmed high data locality of L1 caches. The hybrid design does not include any refresh mechanism, instead, eDRAM blocks are regularly checked to be written back and invalidated before their retention time expires. Experimental results have also shown that a relatively low capacitance is enough to guarantee the maximum performance.

Chapter 3 has introduced the hybrid L2 architecture. In this case, both technologies have been mingled at bank level since the macrocell could be expensive to implement in high-associative caches like LLC memories. A design space exploration varying the percentage of SRAM and eDRAM banks has been considered given that data locality in the LLC is filtered by the L1 cache. As in Chapter 2, the hybrid LLC scheme performs swap operations to keep the MRU data in SRAM banks, which are accessed first. Unlike hybrid L1 caches, refresh operations are not skipped because data in the LLC can be accessed after very long periods of time. Experimental results have shown that a hybrid cache with an eighth of its banks built with SRAM is enough to achieve the best trade-off among performance, energy, and area.

Chapter 4 has presented a family of replacement algorithms for LLCs based on the MRUT concept, which is related to the number of times that a block occupies the MRU position during its live time. Based on the observation that most blocks have a single MRUT when they are evicted by the LRU policy, the baseline MRUT policy randomly select as candidates to be replaced those blocks with one MRUT. This policy has been refined by exploiting recency of information for a few blocks and by adapting it to changes in the working set of the applications. In addition, it has been analyzed the effect of adding a victim cache of the LLC. Results have shown that the MRUT policies, with simpler hardware complexity, perform better than LRU and a set of the most representative replacement policies for LLCs.

Finally, in Chapter 5, a refresh policy for LLCs has been discussed. This policy has been applied in both the hybrid L2 cache and an energy-aware eDRAM L2 cache that differs from the former in that all its banks are built with eDRAM technology. The proposed refresh policy leverages the MRUT concept to reduce the number of refresh operations in these caches. In this way, those blocks candidates to be evicted by the MRUT replacement algorithm are not periodically refreshed. Combining the cache architectures and the refresh policy, results have shown that, compared to a conventional eDRAM cache, the refresh energy savings can be largely reduced. For the hybrid cache, these benefits come at cost of minimal performance loss with respect to a conventional SRAM cache.

6.2 Future Directions

As for future work, it is planned to apply the four main contributions of this thesis in multithreaded and multicore processors. The energy and area benefits brought by the hybrid caches can potentially increase, since these processors are normally implemented with much larger caches. However, the interferences among threads and cores sharing the cache can significantly change both the cache usage and data behavior of the applications. In this context, data locality is less predictable in shared caches.

In L1 data caches for multithreaded processors, a new distribution of the cache hits across the locations of the LRU stack must be carried out to check whether the macrocell should be implemented with just a single SRAM cell or multiple ones to preserve performance for individual threads. The macrocell design must be revisited. For instance, leakage and area savings brought by the eDRAM cells would not be as high as those presented for superscalar processors. In addition, the optimal capacitance of the macrocell that ensures the maximum performance must be estimated again. In such a system with interferences among co-runners, capacitor constraints could limit the capacitance and the refresh logic could not be removed. Like in L1 caches, conclusions will also change in hybrid LLCs of multithread and multicore processors.

The potential of the MRUT concept must be also evaluated again in such processors, since the cache coherence protocol could impact on the number of MRUTs of the blocks. For instance, the shared data could cause that more blocks exhibit multiple MRUTs, so reducing the benefits brought by both replacement and refresh policies.

On the other hand, leveraging reuse information brought by other replacement algorithms to reduce refresh energy in LLCs is also planned as for future work. For example, the information used by the RRIP policies [27] can be useful to predict which blocks should not be refreshed. These blocks would be those with a distance prediction set to be in the distant future.

Finally, the benefits of alternative technologies such as PRAM and MRAM to build cache memories is also left as for future work. These memory technologies are being seen as promising alternatives that can replace SRAM technology in the near future, since they address leakage currents by design and refresh logic is not required. Aside from manufacturing constraints, a hybrid memory architecture could hide the problems associated with these technologies, such as the write endurance and the slow access time for writes. For example, by combining SRAM and PRAM to form a single cache, architecture mechanisms could ensure that most writes are performed in the SRAM cells.

6.3 Publications

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

Journals:

- A. Valero, J. Sahuquillo, V. Lorente, S. Petit, P. López, and J. Duato. Impact on Performance and Energy of the Retention Time and Processor Frequency in L1 Macrocell-Based Data Caches. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (TVLSI), volume 20, issue 6, pages 1108-1117, 2012.
- A. Valero, S. Petit, J. Sahuquillo, P. López, and J. Duato. Design, Performance, and Energy Consumption of eDRAM/SRAM Macrocells for L1 Data Caches.

IEEE Transactions on Computers (TC), volume 61, issue 9, pages 1231-1242, 2012.

- A. Valero, J. Sahuquillo, S. Petit, P. López, and J. Duato. Combining Recency of Information with Selective Random and a Victim Cache in Last-Level Caches. *ACM Transactions on Architecture and Code Optimization* (TACO), volume 9, issue 3, pages 16:1-16:20, 2012. Invited paper presentation at the *8th International Conference on High-Performance and Embedded Architectures and Compilers* (HiPEAC), Berlin, Germany, 2013.

Conferences:

- A. Valero, J. Sahuquillo, S. Petit, V. Lorente, R. Canal, P. López, and J. Duato. An Hybrid eDRAM/SRAM Macrocell to Implement First-Level Data Caches. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture* (MICRO), pages 213-221, New York, NY, USA, 2009. This publication received a HiPEAC Paper Award.

- A. Valero, J. Sahuquillo, S. Petit, P. López, and J. Duato. Improving Last-Level Cache Performance by Exploiting the Concept of MRU-Tour. In *Proceedings of the 20th International Conference on Parallel Architectures and Compilation Techniques* (PACT), page 214, Galveston Island, TX, USA, 2011.

- A. Valero, J. Sahuquillo, S. Petit, P. López, and J. Duato. MRU-Tour-based Replacement Algorithms for Last-Level Caches. In *Proceedings of the 23rd International Symposium on Computer Architecture and High Performance Computing* (SBAC-PAD), pages 112-119, Vitória, ES, Brazil, 2011.

- A. Valero, J. Sahuquillo, S. Petit, P. López, and J. Duato. Analyzing the Optimal Ratio of SRAM Banks in Hybrid Caches. In *Proceedings of the 30th IEEE International Conference on Computer Design* (ICCD), pages 297-302, Montréal, QC, Canada, 2012.

- A. Valero, J. Sahuquillo, S. Petit, and J. Duato. Exploiting Reuse Information to Reduce Refresh Energy in On-Chip eDRAM Caches. In *Proceedings of the 27th International Conference on Supercomputing* (ICS), pages 491-492, Eugene, OR, USA, 2013. This publication received the 1st Place Graduate ACM Student Research Competition (SRC) Award.

- A. Valero, J. Sahuquillo, S. Petit, P. López, and J. Duato. Combining Technologies to Reduce Energy in L1 Data Caches. In *Proceedings of the 7th International Summer School on Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems* (ACACES), pages 51-54, Fiuggi, Italy, 2011.

- A. Valero, J. Sahuquillo, S. Petit, P. López, and J. Duato. MRU-Tour: A Concept to be Applied in Last-Level Cache Replacement. In *Proceedings of the 8th International Summer School on Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems* (ACACES), pages 119-122, Fiuggi, Italy, 2012.

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

- A. Valero, V. Lorente, J. Sahuquillo, S. Petit, P. López, and J. Duato. Memoria dinámica en cachés de datos de primer nivel sin necesidad de refresco. In *Actas de las XX Jornadas de Paralelismo* (JP), pages 265-270, A Coruña, Spain, 2009.

- A. Valero, J. Sahuquillo, S. Petit, P. López, and J. Duato. Impacto en las prestaciones y energía de una cache de datos de primer nivel basada en macroceldas. In *Actas de las XXI Jornadas de Paralelismo* (JP), pages 281-288, València, Spain, 2010.

- A. Valero, J. Sahuquillo, S. Petit, P. López, and J. Duato. Algoritmo de reemplazo para cache de último nivel basado en periodos MRU. In *Actas de las XXII Jornadas de Paralelismo* (JP), pages 557-562, La Laguna, Spain, 2011.

- A. Valero, J. Sahuquillo, S. Petit, P. López, and J. Duato. Incremento en las prestaciones del algoritmo pMRU mediante el uso de una cache víctima. In *Actas de las XXIII Jornadas de Paralelismo* (JP), pages 455-460, Elx, Spain, 2012.

- A. Valero, J. Sahuquillo, S. Petit, P. López, and J. Duato. Prestaciones y consumo de caches híbridas variando la proporción de bancos SRAM. To appear in *Actas de las XXIV Jornadas de Paralelismo* (JP), Madrid, Spain, 2013.

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.

The research developed in this thesis has been recognized with an Intel Doctoral Student Award, received by the Ph.D. candidate in 2012.

Finally, the following papers indirectly related to this dissertation have been accepted for publication:

- V. Lorente, A. Valero, J. Sahuquillo, S. Petit, R. Canal, P. López, and J. Duato. Combining RAM technologies for hard-error recovery in L1 data caches working at very-low power modes. In *Proceedings of the Conference on Design, Automation, and Test in Europe* (DATE), pages 83-88, Grenoble, France, 2013.
- V. Lorente, A. Valero, and R. Canal. Enhancing Performance and Energy Consumption of HER Caches by Adding Associativity. To appear in *Lecture Notes in Computer Science –Europar-2013: Parallel Processing Workshops–* (LNCS), Aachen, Germany, 2013.

References

[1] A. J. Smith. Cache Memories. *ACM Computing Surveys*, 14(3):473–530, 1982.

[2] *Semiconductor Industries Association, "International Technology Roadmap for Semiconductors", 2007, available online at http://www.itrs.net/.*

[3] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy Caches: Simple Techniques for Reducing Leakage Power. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, pages 148–157, 2002.

[4] S. Petit, J. Sahuquillo, J. M. Such, and D. Kaeli. Exploiting Temporal Locality in Drowsy Cache Policies. In *Proceedings of the 2nd Conference on Computing Frontiers*, pages 371–377, 2005.

[5] S. Kaxiras, Z. Hu, and M. Martonosi. Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, pages 240–251, 2001.

[6] M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar. Gated-Vdd: A Circuit Technique to Reduce Leakage in Deep-Submicron Cache Memories. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 90–95, 2000.

[7] R. E. Matick and S. E. Schuster. Logic-Based eDRAM: Origins and Rationale for Use. *IBM Journal of Research and Development*, 49(1):145–165, 2005.

[8] 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, 2002.

[9] 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, 2005.

[10] H. Q. Le, W. J. Starke, J. S. Fields, F. P. O'Connell, D. Q. Nguyen, B. J. Ronchetti, W. M. Sauer, E. M. Schwarz, and M. T. Vaden. IBM POWER6 microarchitecture. *IBM Journal of Research and Development*, 51(6):639–662, 2007.

[11] B. Sinharoy, R. Kalla, W. J. Starke, H. Q. Le, R. Cargnoni, J. A. Van-Norstrand, B. J. Ronchetti, J. Stuecheli, J. Leenstra, G. L. Guthrie, D. Q. Nguyen, B. Blaner, C. F. Marino, E. Retter, and P. Williams. IBM POWER7 multicore server processor. *IBM Journal of Research and Development*, 55(3):1–29, 2011.

[12] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu. RAIDR: Retention-Aware Intelligent DRAM Refresh. In *Proceedings of the 39th International Symposium on Computer Architecture*, pages 1–12, 2012.

[13] *http://www.uniramtech.com/embedded_dram.php.*

[14] X. Dong, X. Wu, G. Sun, Y. Xie, H. Li, and Y. Chen. Circuit and Microarchitecture Evaluation of 3D Stacking Magnetic RAM (MRAM) as a Universal Memory Replacement. In *Proceedings of the 45th IEEE/ACM Design Automation Conference*, pages 554–559, 2008.

[15] X. Wu, J. Li, L. Zhang, E. Speight, and Y. Xie. Power and Performance of Read-Write Aware Hybrid Caches with Non-volatile Memories. In *Proceedings of the Conference on Design, Automation, and Test in Europe*, pages 737–742, 2009.

[16] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, and Y. Xie. Hybrid Cache Architecture with Disparate Memory Technologies. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, pages 34–45, 2009.

[17] X. Liang, R. Canal, G.-Y. Wei, and D. Brooks. Process Variation Tolerant 3T1D-Based Cache Architectures. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 15–26, 2007.

[18] Z. Hu, P. Juang, P. Diodato, S. Kaxiras, K. Skadron, M. Martonosi, and D. W. Clark. Managing Leakage for Transient Data: Decay and Quasi-Static 4T Memory Cells. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 52–55, 2002.

[19] B. Stackhouse, S. Bhimji, C. Bostak, D. Bradley, B. Cherkauer, J. Desai, E. Francom, M. Gowan, P. Gronowski, D. Krueger, C. Morganti, and S. Troyer. A 65 nm

2-Billion Transistor Quad-Core Itanium Processor. *IEEE Journal of Solid-State Circuits*, 44(1):18–31, 2009.

[20] J.-L. Baer. *Microprocessor Architecture: From Simple Pipelines to Chip Multiprocessors*. Cambridge University Press, 2010.

[21] L. A. Belady. A Study of Replacement Algorithms for Virtual-Storage Computer. *IBM Systems Journal*, 5(2):78–101, 1966.

[22] M. Kharbutli and Y. Solihin. Counter-Based Cache Replacement and Bypassing Algorithms. *IEEE Transactions on Computers*, 57:433–447, 2008.

[23] W.-F. Lin and S. K. Reinhardt. Predicting Last-Touch References under Optimal Replacement. *Technical Report CSE-TR-447-02, University of Michigan*, 2002.

[24] M. Chaudhuri. Pseudo-LIFO: The Foundation of a New Family of Replacement Policies for Last-level Caches. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 401–412, 2009.

[25] C. Zhang and B. Xue. Divide-and-Conquer: A Bubble Replacement for Low Level Caches. In *Proceedings of the 23rd International Conference on Supercomputing*, pages 80–89, 2009.

[26] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, and J. Emer. Adaptive Insertion Policies for High Performance Caching. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, pages 381–391, 2007.

[27] A. Jaleel, K. B. Theobald, S. C. Steely Jr., and J. Emer. High Performance Cache Replacement Using Re-Reference Interval Prediction (RRIP). In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, pages 60–71, 2010.

[28] M. Ghosh and H.-H. S. Lee. Smart Refresh: An Enhanced Memory Controller Design for Reducing Energy in Conventional and 3D Die-Stacked DRAMs. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 134–145, 2007.

[29] T. Ohsawa, K. Kai, and K. Murakami. Optimizing the DRAM Refresh Count for Merged DRAM/Logic LSIs. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 82–87, 1998.

[30] R. K. Venkatesan, S. Herr, and E. Rotenberg. Retention-Aware Placement in DRAM (RAPID): Software Methods for Quasi-Non-Volatile DRAM. In *Proceedings of the 12th International Symposium on High-Performance Computer Architecture*, pages 155–165, 2006.

[31] J. Kim and M. C. Papaefthymiou. Dynamic Memory Design for Low Data-Retention Power. In *Proceedings of the 10th International Workshop on Integrated Circuit Design, Power and Timing Modeling, Optimization and Simulation*, pages 207–216, 2000.

[32] C. Wilkerson, A. R. Alameldeen, Z. Chishti, W. Wu, D. Somasekhar, and S.-L. Lu. Reducing Cache Power with Low-Cost, Multi-bit Error-Correcting Codes. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, pages 83–93, 2010.

[33] P. G. Emma, W. R. Reohr, and M. Meterelliyoz. Rethinking Refresh: Increasing Availability and Reducing Power in DRAM for Cache Applications. *IEEE Micro*, 28(6):47–56, 2008.

[34] A. Agrawal, P. Jain, A. Ansari, and J. Torrellas. Refrint: Intelligent Refresh to Minimize Power in On-Chip Multiprocessor Cache Hierarchies. In *Proceedings of the 19th International Symposium on High Performance Computer Architecture*, pages 400–411, 2013.

[35] W. Zhao and Y. Cao. Predictive Technology Model for Nano-CMOS Design Exploration. *Journal on Emerging Technologies in Computing Systems*, 3(1):1–17, 2007.

[36] N. H. E. Weste, D. Harris, and A. Banerjee. *CMOS VLSI Design: A Circuits and Systems Perspective*. Pearson/Addison-Wesley, 2005.

[37] B. Calder, D. Grunwald, and J. Emer. Predictive Sequential Associative Cache. In *Proceedings of the 2nd International Symposium on High-Performance Computer Architecture*, pages 244–253, 1996.

[38] K. Inoue, T. Ishihara, and K. Murakami. Way-Predicting Set-Associative Cache for High Performance and Low Energy Consumption. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 273–275, 1999.

[39] D. Burger and T. M. Austin. The simplescalar tool set, version 2.0. *ACM SIGARCH Computer Architecture News*, 25(3):13–25, 1997.

[40] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi. CACTI 5.1. *Hewlett-Packard Laboratories, Palo Alto, Technical Report*, 2008.

[41] S. Thoziyoor, J. H. Ahn, M. Monchiero, J. B. Brockman, and N. P. Jouppi. A Comprehensive Memory Modeling Tool and its Application to the Design and Analysis of Future Memory Hierarchies. In *Proceedings of the 35th Annual International Symposium on Computer Architecture*, pages 51–62, 2008.

[42] *Standard Performance Evaluation Corporation, available online at http://www.spec.org/cpu2000/.*

[43] T. Kirihata, P. Parries, D. R. Hanson, H. Kim, J. Golz, G. Fredeman, R. Rajeevakumar, J. Griesemer, N. Robson, A. Cestero, B. A. Khan, G. Wang, M. Wordeman, and S. S. Iyer. An 800-MHz Embedded DRAM With a Concurrent Refresh Mode. *IEEE Journal of Solid-State Circuits*, 40(6):1377–1387, 2005.

[44] B. Keeth, R. J. Baker, B. Johnson, and F. Lin. *DRAM Circuit Design. Fundamental and High-Speed Topics.* John Wiley and Sons, Inc., 2008.

[45] A. Valero, J. Sahuquillo, S. Petit, V. Lorente, R. Canal, P. López, and J. Duato. An Hybrid eDRAM/SRAM Macrocell to Implement First-Level Data Caches. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 213–221, 2009.

[46] A. Valero, S. Petit, J. Sahuquillo, P. López, and J. Duato. Design, Performance, and Energy Consumption of eDRAM/SRAM Macrocells for L1 Data Caches. *IEEE Transactions on Computers*, 61(9):1231–1242, 2012.

[47] A. Valero, J. Sahuquillo, V. Lorente, S. Petit, P. López, and J. Duato. Impact on Performance and Energy of the Retention Time and Processor Frequency in L1 Macrocell-Based Data Caches. *IEEE Transactions on Very Large Scale Integration Systems*, 20(6):1108–1117, 2012.

[48] Various Methods of DRAM Refresh. *Micron Technology Inc., Technical Note N-04-30*, pages 1–4, 1999.

[49] Omid Azizi, Aqeel Mahesri, Benjamin C. Lee, Sanjay J. Patel, and Mark Horowitz. Energy-Performance Tradeoffs in Processor Architecture and Circuit Design: A Marginal Cost Analysis. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, pages 26–36, 2010.

[50] J. Lira, C. Molina, D. Brooks, and A. González. Implementing a hybrid SRAM / eDRAM NUCA architecture. In *Proceedings of the 18th International Conference on High Performance Computing*, pages 1–10, 2011.

[51] M. K. Qureshi, V. Srinivasan, and J. A. Rivers. Scalable High Performance Main Memory System Using Phase-Change Memory Technology. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, pages 24–33, 2009.

[52] W. k. S. Yu, R. Huang, S. Q. Xu, S.-E. Wang, E. Kan, and G. E. Suh. SRAM-DRAM Hybrid Memory with Applications to Efficient Register Files in Fine-Grained Multi-Threading. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, pages 247–258, 2011.

[53] A. Valero, J. Sahuquillo, S. Petit, P. López, and J. Duato. Analyzing the Optimal Percentage of SRAM Banks in Hybrid Caches. In *Proceedings of the 30th IEEE International Conference on Computer Design*, pages 297–302, 2012.

[54] D. Wood, M. D. Hill, and R. E. Kessler. A Model for Estimating Trace-Sample Miss Ratios. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 79–89, 1991.

[55] G. S. Tyson, M. Farrens, J. Matthews, and A. R. Pleszkun. A Modified Approach to Data Cache Management. In *Proceedings of the 28th Annual International Symposium on Microarchitecture*, pages 93–103, 1995.

[56] J. A. Rivers and E. S. Davidson. Reducing Conflicts in Direct-Mapped Caches with a Temporality-based Design. In *Proceedings of the 25th International Conference on Parallel Processing*, pages 154–163, 1996.

[57] T. L. Johnson and W. m. W. Hwu. Run-time Adaptive Cache Hierarchy Management via Reference Analysis. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, pages 315–326, 1997.

[58] H. Liu, M. Ferdman, J. Huh, and D. Burger. Cache Bursts: A New Approach for Eliminating Dead Blocks and Increasing Cache Efficiency. In *Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture*, pages 222–233, 2008.

[59] P. Petoumenos, G. Keramidas, and S. Kaxiras. Instruction-based Reuse-Distance Prediction for Effective Cache Management. In *Proceedings of the 9th International Symposium on Systems, Architectures, Modeling, and Simulation*, pages 49–58, 2009.

[60] A. Basu, N. Kirman, M. Kirman, M. Chaudhuri, and J. Martínez. Scavenger: A New Last Level Cache Architecture with Global Block Priority. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 421–432, 2007.

[61] H. Dybdahl, P. Stenström, and L. Natvig. An LRU-based Replacement Algorithm Augmented with Frequency of Access in Shared Chip-Multiprocessor Caches. *ACM SIGARCH Computer Architecture News*, 35:45–52, 2007.

[62] S. Jiang and X. Zhang. LIRS: An Efficient Low Inter-reference Recency Set Replacement Policy to Improve Buffer Cache Performance. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 31–42, 2002.

[63] W. A. Wong and J.-L. Baer. Modified LRU Policies for Improving Second-Level Cache Behavior. In *Proceedings of the 6th International Symposium on High Performance Computer Architecture*, pages 49–60, 2000.

[64] A. Valero, J. Sahuquillo, S. Petit, P. López, and J. Duato. Improving Last-Level Cache Performance by Exploiting the Concept of MRU-Tour. In *Proceedings of the 20th International Conference on Parallel Architectures and Compilation Techniques*, page 214, 2011.

[65] A. Valero, J. Sahuquillo, S. Petit, P. López, and J. Duato. MRU-Tour-based Replacement Algorithms for Last-Level Caches. In *Proceedings of the 23rd International Symposium on Computer Architecture and High Performance Computing*, pages 112–119, 2011.

[66] A. Valero, J. Sahuquillo, S. Petit, P. López, and J. Duato. Combining Recency of Information with Selective Random and a Victim Cache in Last-Level Caches. *ACM Transactions on Architecture and Code Optimization*, 9(3):16:1–16:20, 2012.

[67] A. Valero, J. Sahuquillo, S. Petit, and J. Duato. Exploiting Reuse Information to Reduce Refresh Energy in On-Chip eDRAM Caches. In *Proceedings of the 27th International Conference on Supercomputing*, pages 491–492, 2013.