Evaluación y optimización de políticas de reemplazo caché en entornos PCM

Roberto Alonso Rodríguez Rodríguez

MÁSTER EN INVESTIGACIÓN EN INFORMÁTICA. FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo Fin Máster en Ingeniería de Computadores

Junio 2012

Calificación: Matrícula de Honor (9.5)

Director y colaborador del proyecto:

Luis Piñuel Moreno

Fernando Castro Rodríguez

A mis padres: Roberto y Elisa y a Laura por su cariño, dedicación, comprensión y por estar siempre ahí.

Agradecimientos

Al grupo ArTeCS especialmente a Luis Piñuel y Fernando Castro por permitirme realizar este trabajo y por su apoyo en la realización del mismo, a Dani Chaver por la colaboración que me ha brindado y a los técnicos por su paciencia y pronta respuesta a mis peticiones.

A la Universidad de Costa Rica por el apoyo brindado para mi formación académica en el exterior.

Resumen en castellano

El diferente ritmo de evolución entre el microprocesador y la memoria principal constituye uno de los principales retos que los diseñadores deben afrontar para desarrollar computadores más potentes. A este problema, llamado brecha de memoria, se suma el hecho de que la capacidad de escalado de la tecnología DRAM es ya muy limitada actualmente, lo que conlleva que se consideren nuevas tecnologías de memoria como posibles candidatas a reemplazar a las convencionales DRAM. A día de hoy PCM se postula como la mejor alternativa para ello.

PCM presenta importantes ventajas respecto a DRAM, pero también algunas debilidades que deben ser mitigadas antes de que PCM pueda ser empleada como tecnología de memoria principal. En particular, la vida útil de este tipo de memorias (limitada por el número de ciclos de escritura que se pueden realizar sobre cada celda) es uno de los principales inconvenientes de esta tecnología. En este trabajo se presenta un análisis del comportamiento, en cuanto a número de escrituras sobre memoria principal se refiere, de las políticas convencionales de reemplazo aplicadas a la caché; asimismo, se detallan nuevas propuestas de políticas orientadas a reducir el número de escrituras a memoria principal y de esta forma, sin degradar significativamente el rendimiento del sistema, aumentar la vida útil de la memoria cuando ésta se desarrolla usando la tecnología PCM.

Palabras Clave

PCM, *Phase Change Memory*, Políticas de reemplazo caché, DRRIP, PeLIFO, SHiP, LRU, CLP, Reducción de escrituras a memoria principal

Abstract

The different evolution rate between the microprocessor and the main memory is one of the greatest challenges that current designers face in order to develop more powerful computer systems. In addition to this problem, called memory gap, it is the fact that the scalability of the DRAM technology is very limited currently, leading to consider new memory technologies as possible candidates for the replacement of conventional DRAM. PCM is currently postulated as the best alternative for it.

PCM has significant advantages over DRAM, but also some drawbacks that need to be mitigated before PCM can be used as main memory technology for the next computers generation. In particular, the endurance of this memory (the life time limited by the number of write cycles that can be performed on each cell) is one of the main drawbacks of this technology. This work presents a behavior analysis, in terms of number of writings on main memory, of conventional cache replacement policies. New proposals aimed at reducing the number of writes to main memory are exposed thought for increase the life time of the memory when it is developed using PCM technology, without significantly degrading the system performance.

Keywords

PCM, Phase Change Memory, Memory Hierarchy, Cache replacement Policies, DRRIP, PeLIFO, SHiP, LRU, CLP, Main memory writes reduction

Índice general

Índice I

Índice de Figuras IV

Índice de Cuadros VII

1. Introducción 1

1.1. Jerarquía de memoria 3

1.2. Problemática de las tecnologías actuales 9

1.2.1. Memoria DRAM 9

1.2.2. Memoria *Flash* 11

1.2.3. RAM Ferro-Eléctrica 13

1.2.4. RAM Magnética y de Transferencia de Giro de Torque . 14

1.2.5. RAM Resistiva 14

1.3. PCM como posible solución 15

1.4. Motivación y objetivos 17

1.5. Organización del documento 18

2. Trabajo Relacionado 19

2.1. Técnicas de reducción de escrituras en PCM 19

2.1.1. *Wear Leveling* 19

2.1.2. Escritura a nivel de bloques 20

2.1.3. Filtrado de escrituras de granularidad fina 21

2.1.4. *Flip-N-Write* 22

2.2. Políticas de reemplazo de caché 22

2.2.1. Política Aleatoria (Random) 23
2.2.2. Política de Bélády 23
2.2.3. Política LRU 24
2.2.4. Política peLIFO 25
2.2.5. Política DRRIP 28
2.2.6. Política SHiP 32
2.2.7. Política CLP 34

3. Propuesta de Políticas de Reemplazo en PCM 36

3.1. Políticas Basadas en LRU 37
3.1.1. Política WPP1 37
3.1.2. Política WPP2 38
3.2. Políticas Basadas en DRRIP 39
3.2.1. Política DRRIPW1 39
3.2.2. Política DRRIPW2 39
3.2.3. Política DRRIPW3 40
3.2.4. Política DRRIPW4 40
3.2.5. Política DRRIPW5 40
3.2.6. Política DRRIPW6 41
3.2.7. Política DRRIPW7 41
3.2.8. Política DRRIPW8 41
3.2.9. Política DRRIPW9 41
3.3. Políticas Basadas en peLIFO 42
3.3.1. Política peLIFO1 42
3.3.2. Política peLIFO2 42
3.3.3. Política peLIFO3 42
3.3.4. Política peLIFO4 43
3.3.5. Política peLIFO5 43

3.3.6. Política peLIFO6 43

4. Entorno Experimental **44**

4.1. Pin 44

4.2. MultiCacheSim 46

4.3. *Benchmarks* 47

4.4. Configuración 48

4.5. Consumo de energía y tiempo medio de acceso a memoria 49

5. Evaluación de Resultados **52**

5.1. Políticas Convencionales 52

5.1.1. Análisis de datos de jerarquía no exclusiva sin actualización de estadísticas al haber escritura retardada 53

5.1.2. Análisis de datos de jerarquía no exclusiva con actualización de estadísticas al haber escritura retardada 54

5.2. Políticas Propuestas 60

5.2.1. Políticas basadas en LRU 60

5.2.2. Políticas basadas en DRRIP 60

5.2.3. Políticas basadas en peLIFO 67

6. Conclusiones y Trabajo Futuro **77**

7. Publicaciones realizadas **80**

Bibliografía **81**

Índice de figuras

1.1. Evolución del rendimiento de la memoria SRAM (caché) y DRAM (memoria principal) . . . 2
1.2. Evolución de la capacidad de la memoria SRAM (caché) y DRAM (memoria principal) . . . 2
1.3. Jerarquía de memoria . . . 4
1.4. Jerarquía de memoria con escritura directa . . . 6
1.4. Jerarquía de memoria con escritura retardada . . . 8
1.5. Estructura de una celda DRAM . . . 9
1.6. Memoria dinámica sencilla . . . 10
1.7. Celda de memoria *flash* y sus estados . . . 12
1.8. Celda de memoria ferro-eléctrica . . . 13
1.9. Pulsos de programación de una celda PCM . . . 16
1.10. Estructura de una celda PCM . . . 16

2.1. Ejemplo de *Wear Leveling* . . . 21
2.2. Estado de la caché siguiendo en algoritmo de Bélády . . . 23
2.3. Caché de cuatro bloques que usa la política LRU . . . 25
2.4. Operaciones de LIFO y peLIFO . . . 26
2.5. Caché de ocho bloques que usa la política CLP . . . 35

3.1. Cola LRU para la política WPP1 . . . 38
3.2. Cola LRU para la política WPP2 . . . 39

4.1. Arquitectura de Pin . . . 45
4.2. Arquitectura del *MultiCacheSim* . . . 47

5.1. Escrituras normalizadas a LRU, sin actualización de estadísticas 54
5.2. Tasa de fallos en L_3 normalizada a LRU, sin actualización de estadísticas 55
5.3. TMAM normalizado a LRU, sin actualización de estadísticas . . 56
5.4. Consumo de energía normalizada a LRU, sin actualización de estadísticas 57
5.5. Escrituras normalizadas a LRU, con actualización de estadísticas 57
5.6. Tasa de fallos en L_3 normalizada a LRU, con actualización de estadísticas 58
5.7. TMAM normalizado a LRU, con actualización de estadísticas . 58
5.8. Consumo de energía normalizada a LRU, con actualización de estadísticas 59
5.9. Escrituras normalizadas a LRU y tasa de fallos en L_3 normalizada a LRU. Para las políticas propuestas basadas en LRU. 61
5.10. TMAM normalizado a LRU y consumo de energía normalizado a LRU. Para las políticas propuestas basadas en LRU. 62
5.11. Escrituras normalizadas a LRU para las versiones de SRRIP . . 63
5.12. Escrituras normalizadas a LRU para las políticas basadas en DRRIP 64
5.13. Tasa de fallos en L_3 normalizada a LRU para las políticas basadas en DRRIP 65
5.14. TMAM normalizado a LRU para las políticas basadas en DRRIP 66
5.15. Consumo de energía normalizada a LRU para las políticas basadas en DRRIP 66
5.16. Escrituras normalizadas a LRU para las políticas basadas en pe-LIFO 67

5.17. Tasa de fallos en L_3 normalizada a LRU para las políticas basadas en peLIFO 68
5.18. TMAM normalizado a LRU para las políticas basadas en peLIFO 69
5.19. Consumo de energía normalizada a LRU para las políticas basadas en peLIFO 70
5.20. Escrituras normalizadas a LRU 73
5.21. Tasa de fallos en L_3 normalizada a LRU 74
5.22. TMAM normalizado a LRU 75
5.23. Consumo de energía normalizada a LRU 76

Índice de cuadros

1.1. Estados de una celda de memoria PCM 15
1.2. Comparación de características de las tecnologías emergentes de almacenamiento 17

2.1. Valor del RRPV para cada tipo predicción de acceso 29
2.2. Estado de la memoria caché para diferentes algoritmos de reemplazo 31

4.1. Pruebas del conjunto SPEC2006 48
4.2. Configuración de memoria caché elegida para realizar la evaluación 48
4.3. Latencia y consumo de energía para la configuración de memoria utilizada 50

Capítulo 1

Introducción

El diferente ritmo de evolución existente entre el microprocesador y la memoria principal constituye uno de los principales retos que los diseñadores deben afrontar para implementar sistemas computacionales más potentes. A este problema, llamado brecha de memoria [1, 2], se suma el hecho de que la escalabilidad de la tecnología DRAM (actualmente la empleada mayoritariamente en la memoria principal) es ya muy limitada actualmente, lo que conlleva que se consideren nuevas tecnologías de memoria como posibles candidatas a reemplazar a las convencionales DRAM. A día de hoy, varias tecnologías emergentes de memoria se investigan para convertirse en la sustituta de DRAM, entre las cuales PCM se postula como la mejor alternativa para ello.

Sin embargo, pese al problema de escalabilidad anteriormente mencionado, las memorias DRAM poseen un crecimiento, en cuanto a capacidad de almacenamiento, superior al que presentan las memorias SRAM (caché) cercanas al procesador. Las figuras 1.1 y 1.2 muestran este crecimiento.

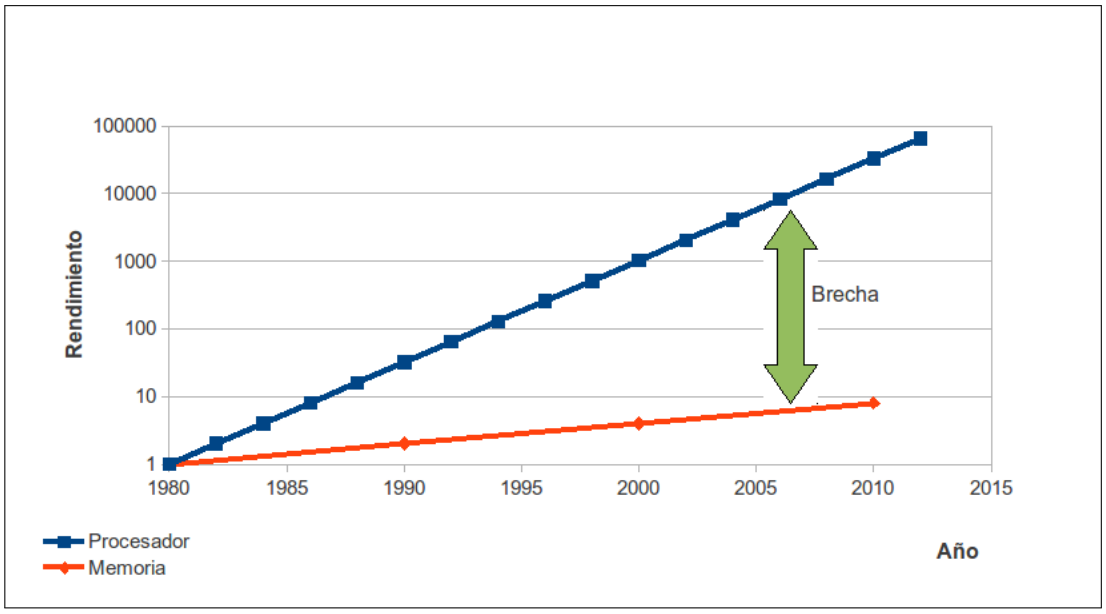


Figura 1.1: Evolución del rendimiento de la memoria SRAM (caché) y DRAM (memoria principal)

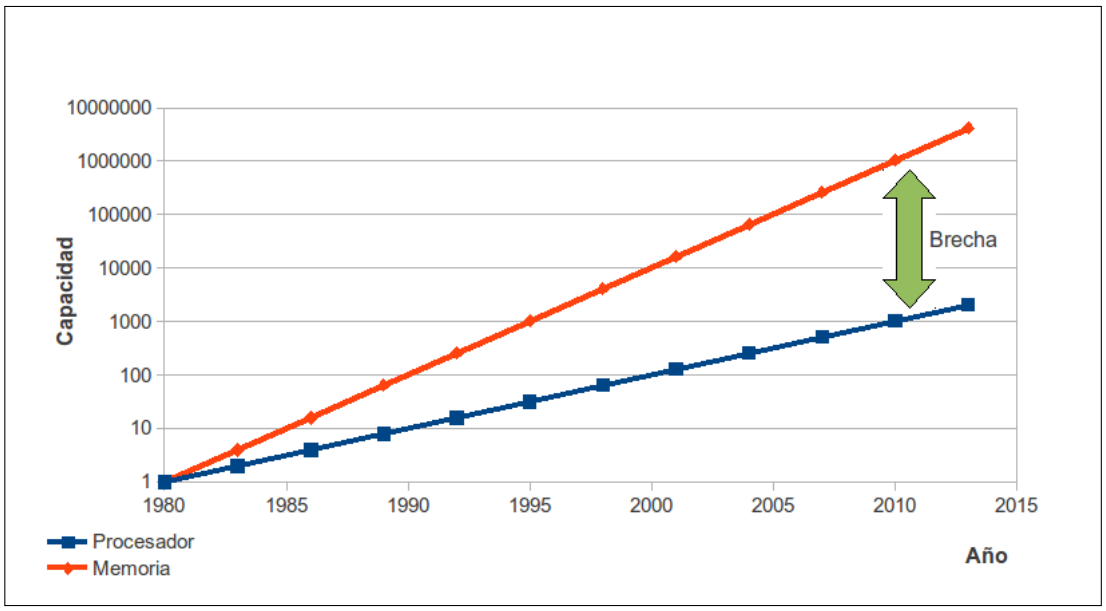


Figura 1.2: Evolución de la capacidad de la memoria SRAM (caché) y DRAM (memoria principal)

1.1. Jerarquía de memoria

En un sistema computador se utiliza una jerarquía de memoria para crear al microprocesador la ilusión de disponer de una memoria ideal, esto es, una memoria con un tiempo de respuesta cercano a cero y con una capacidad de almacenamiento ilimitada.

Para definir una jerarquía de memoria, se establecen diferentes niveles donde cada uno de ellos puede hacer uso de diferentes tecnologías de fabricación así como de distintos algoritmos para entrelazar cada par de niveles.

En el nivel más cercano al microprocesador se tienen los registros, que ofrecen el mayor rendimiento a pesar que se cuenta con un número reducido de ellos. Después se tienen los niveles de memoria caché, los cuales utilizan memorias de tipo SDRAM, éstas se encuentran en el mismo circuito integrado que el procesador y se fabrican haciendo uso de la misma tecnología con la que se construye éste; lo cual hace que los niveles caché de la jerarquía sean los más rápidos y al mismo tiempo, dado el alto costo económico de esta tecnología, los de menor capacidad de almacenamiento.

En el siguiente nivel de memoria (memoria principal), tradicionalmente, se utilizan memorias tipo DRAM. Este nivel posee una capacidad de almacenamiento y una latencia mayor que el nivel anterior. En el nivel de memoria que sigue (memoria secundaria) se utilizan dispositivos de almacenamiento de tipo magnético o de estado sólido e independientemente del tipo de tecnología, este nivel presenta mayor capacidad de almacenamiento y latencia que el nivel anterior. La figura 1.3 muestra un esquema típico de una jerarquía de memoria.

La gestión de los datos que circulan por los diferentes niveles de la jerarquía de memoria se realiza por medio de distintos algoritmos. En el caso del tráfico de datos entre diferentes niveles de memoria caché y entre el último nivel de memoria caché y la memoria principal, la gestión se lleva a cabo mediante

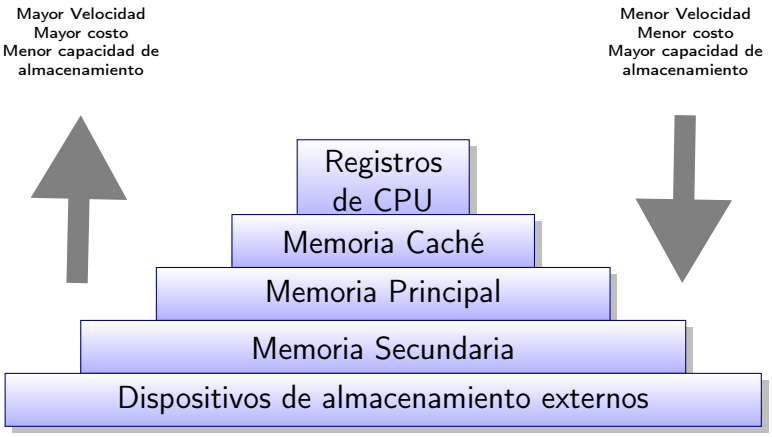


Figura 1.3: Jerarquía de memoria

algoritmos que se encuentran implementados en el controlador de memoria; siendo el sistema operativo el encargado de la gestión de datos entre memoria principal y secundaria.

La memoria caché puede estar formada por varios niveles, estos niveles pueden ser inclusivos (un bloque que está en el nivel más cercano al procesador deber estar en todos los demás), exclusivos (un bloque que está en un nivel de caché no puede estar en ninguno de los demás) o no exclusivos. En una jerarquía no exclusiva, cuando el procesador solicita un dato que no está en el nivel de caché más próximo a él, el bloque que contiene el dato se copia desde el nivel en donde se encuentre a todos los niveles de la jerarquía más cercanos al procesador; sin embargo, la gestión que cada nivel haga del bloque no afecta a los demás niveles, siempre y cuando el bloque no se haya modificado.

Dado que los niveles de la jerarquía que interesan en este trabajo son la caché y memoria principal, de ahora en adelante cuando se utilice el término jerarquía se refiere a estos dos niveles. Cuando un bloque se escribe en algún nivel de la jerarquía, la escritura debe propagarse a los demás niveles empezando desde el nivel más cercano al procesador para mantener así la coherencia de los datos. Dicha escritura puede realizarse mediante dos políticas: escritura directa (*write-*

through) o escritura retardada (*write-back*). En la escritura directa, cuando un dato se escribe en un nivel de la jerarquía, se escribe en todos los demás; en el caso de la escritura retardada el dato solo se escribe entre niveles adyacentes de la jerarquía una vez que va a ser desalojado del correspondiente nivel.

La figura 1.4 muestra una jerarquía de memoria con una política de escritura directa; al darse una escritura, ésta se propaga por toda la jerarquía. La figura 1.4 muestra una jerarquía de memoria con una política de escritura retardada; en el caso de darse una escritura, el nuevo dato se mantiene únicamente en el nivel en que se escribió hasta que el bloque escrito requiera reemplazarse de dicho nivel. En esta figura 1.4 también puede observarse cómo en una jerarquía no exclusiva, al llevarse un bloque a memoria caché (figura 1.4(e)), éste se copia en todos los niveles de la jerarquía, sin embargo, un bloque que está en un nivel cercano al procesador no necesariamente está en uno más lejano a éste. En este caso el bloque que contiene d_1 se desaloja del nivel L_1 y el bloque que contiene e_1 se desaloja del L_2; así el bloque que contiene e_1 permanece en la caché L_1 pero no en la L_2.

En una jerarquía que utiliza una política de escritura retardada, para llevar el control de los bloques que deben escribirse a niveles adyacentes de la jerarquía se agrega, a cada bloque, un bit que indica si el bloque ha sido modificado o no, este bit se llama *dirty bit*. En caso de que el *dirty bit* tenga un valor de uno significa que el bloque ha sido modificado y se dice que esta *sucio*; cuando el *dirty bit* tiene un valor de cero el bloque no ha sido modificado y se dice que está *limpio*.

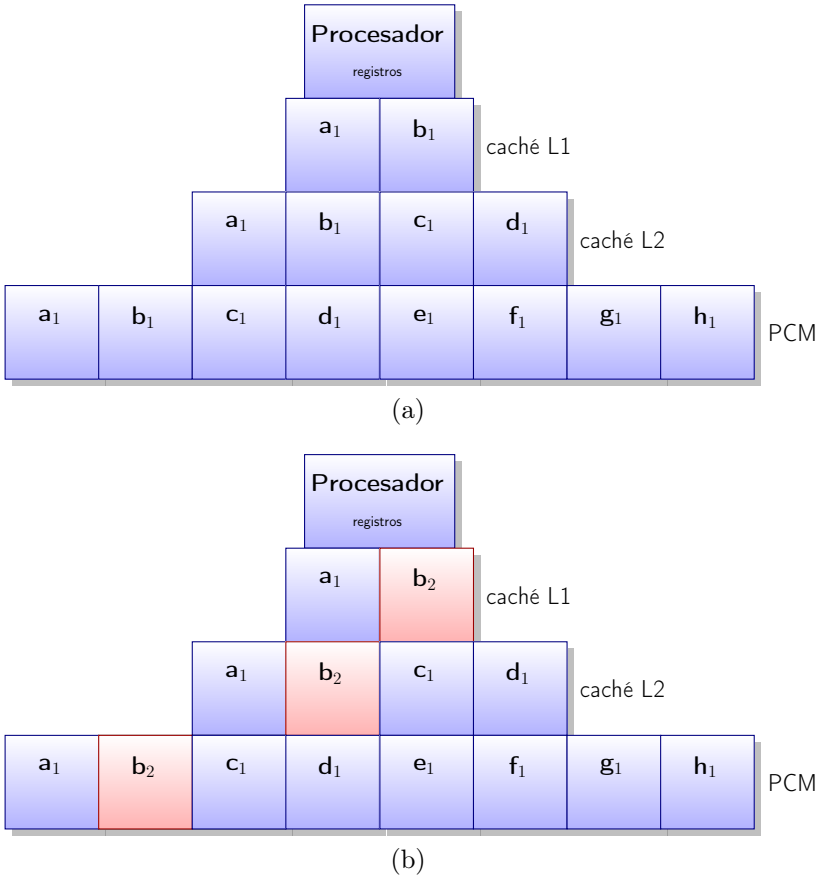
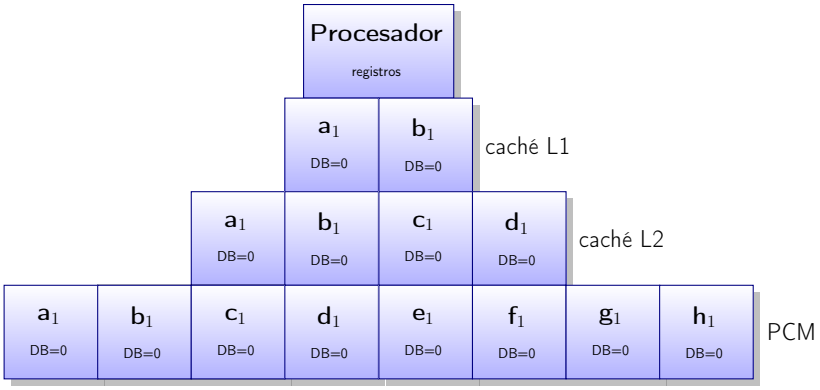
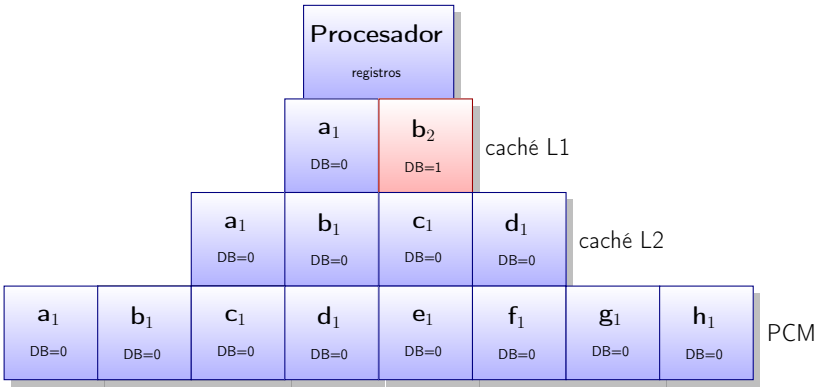


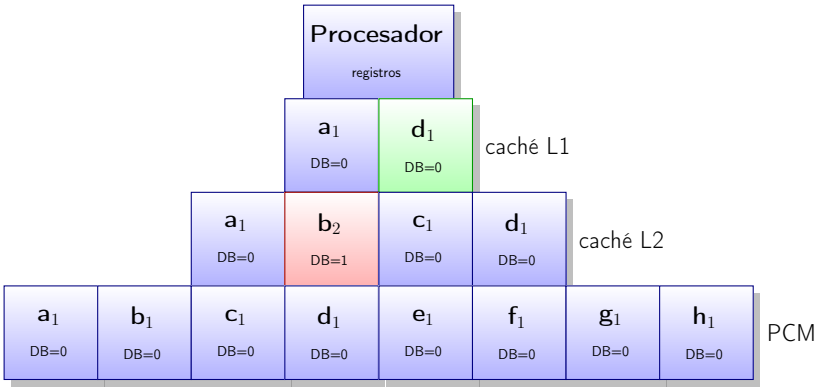
Figura 1.4: Situación inicial de una jerarquía de memoria con escritura directa, los bloques son de tamaño de un dato. La jerarquía consta de dos niveles de memoria caché y uno de memoria principal (a). Al darse una escritura al bloque que contiene el dato b_1 el nuevo dato escrito b_2 se propaga a todos los niveles de la jerarquía (b).


Procesador
registros
a1 DB=0
b1 DB=0
caché L1
a1 DB=0
b1 DB=0
c1 DB=0
d1 DB=0
caché L2
a1 DB=0
b1 DB=0
c1 DB=0
d1 DB=0
e1 DB=0
f1 DB=0
g1 DB=0
h1 DB=0
PCM

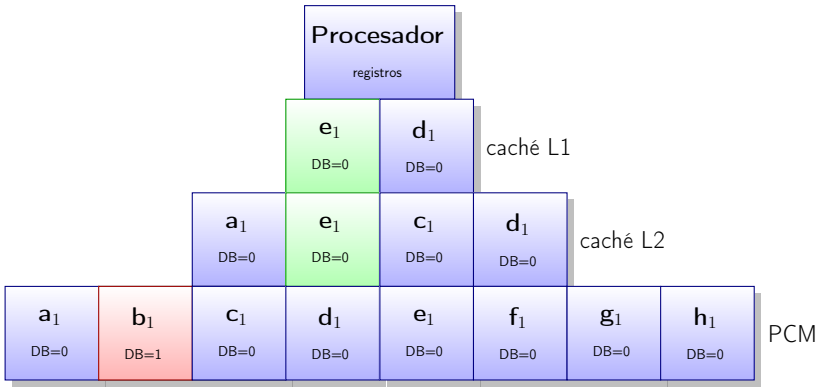
(a)


Procesador
registros
a1 DB=0
b2 DB=1
caché L1
a1 DB=0
b1 DB=0
c1 DB=0
d1 DB=0
caché L2
a1 DB=0
b1 DB=0
c1 DB=0
d1 DB=0
e1 DB=0
f1 DB=0
g1 DB=0
h1 DB=0
PCM

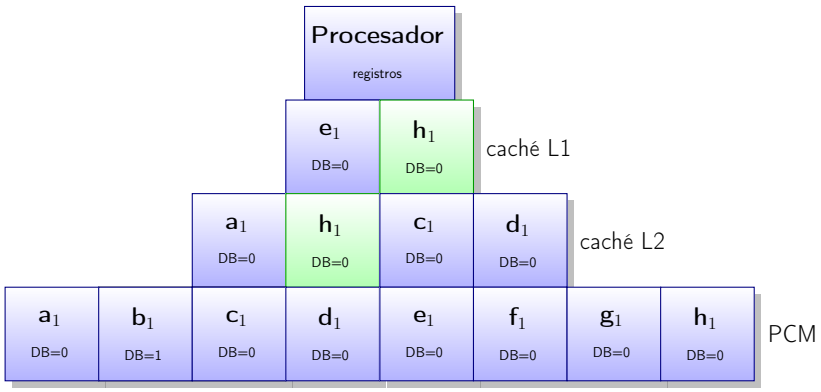
(b)


Procesador
registros
a1 DB=0
d1 DB=0
caché L1
a1 DB=0
b2 DB=1
c1 DB=0
d1 DB=0
caché L2
a1 DB=0
b1 DB=0
c1 DB=0
d1 DB=0
e1 DB=0
f1 DB=0
g1 DB=0
h1 DB=0
PCM

(c)



(d)



(e)

Figura 1.4: Situación inicial de una jerarquía de memoria con escritura retardada, los bloques son de tamaño de un dato. La jerarquía consta de dos niveles de memoria caché y uno de memoria principal (a). Al solicitar el procesador el dato b_1 para ser escrito, el nuevo dato b_2 se escribe solo en la caché L_1 (b). Si a continuación el procesador solicita el bloque que contiene el dato d_1, en el caso de reemplazar el bloque que contiene b_2 en la cache L_1 el dato se escribe en la caché L_2 (c). El procesador solicita el bloque que contiene el dato e_1, en el caso de reemplazar el bloque que contiene a_1 en la caché L_1 y el bloque que contiene b_2 de la caché L_2, el dato (b_2) se escribe en memoria principal (d). El procesador solicita el bloque que contiene el dato h_1, en el caso de reemplazar el bloque que contiene d_1 en la caché L_1 y el bloque que contiene e_1 de la caché L_2, se forma una jerarquía no exclusiva, es este caso los bloques que están en un nivel de caché no tienen porque estar en los más lejanos a éste (e).

1.2. Problemática de las tecnologías actuales

1.2.1. Memoria DRAM

Esta ha sido la memoria que tradicionalmente se ha usado como memoria principal en los sistemas computadores. Actualmente DRAM posee una escalabilidad de hasta 40nm y una vida útil de 10^{15} ciclos de escritura [3, 4]. La figura 1.5 muestra la estructura elemental de una celda de memoria DRAM [5], que consta principalmente de un transistor y un condensador el cual contiene el dato lógico, uno si se encuentra cargado y cero si está descargado.

Figura 1.5: Estructura de una celda DRAM

Debido a las fugas que presenta el condensador debido a su impedancia intrínseca, es necesario que el dato se esté actualizando cada cierto tiempo, para lo que se requiere de una circuitería extra que se encargue de este trabajo. Al mismo tiempo la lectura del dato es destructiva por lo que éste debe ser almacenado de nuevo después de leerlo. En la figura 1.6 se muestra una celda amplificadora (AS), la cual compara el valor de cada celda de memoria y restituye el dato leído. Esto debe hacerse dado que los niveles de tensión de cada celda de memoria son muy pequeños y al leer el dato la carga del condensador varía. Además, gracias a otras señales como R/W y CE (la primera indica si el acceso es una lectura o escritura y la segunda habilita o deshabilita la me-

moria), es posible controlar la dirección del flujo de datos y la actividad del circuito respectivamente.

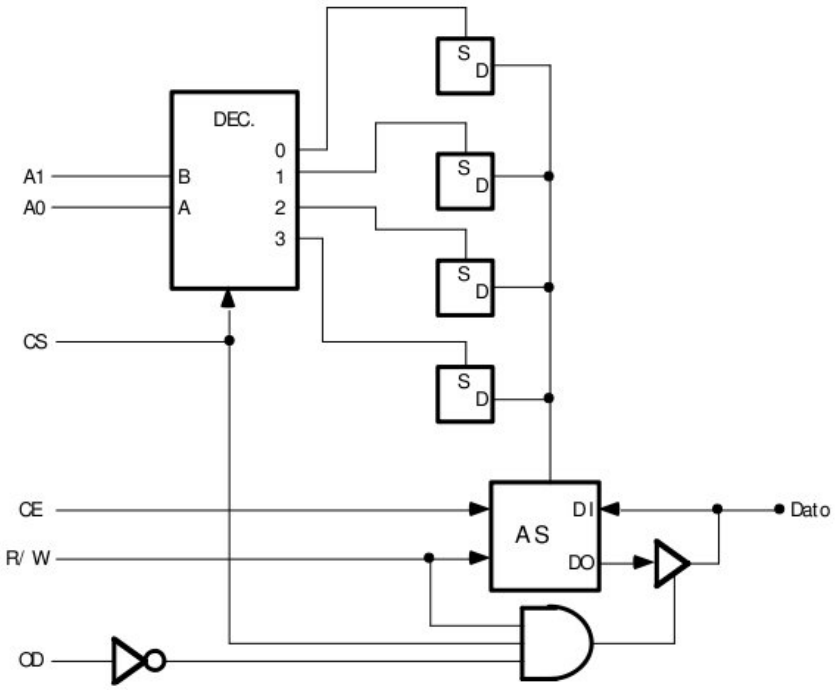


Figura 1.6: Memoria dinámica sencilla

En las celdas DRAM los condensadores deben de ser lo suficientemente grandes, no solo para almacenar una carga que pueda ser detectada por los sensores sino para mitigar las cargas por las corrientes de fuga. Asimismo, los transistores deben tener la capacidad de ejercer un control total en el canal entre el drenador y la fuente, por estas razones no se tienen soluciones por debajo de los 40nm [3, 6].

Sumado al reto que significa la integración de DRAM, surgen otros obstáculos en lo que respecta a esta tecnología, como son los de costo y consumo de energía. En el caso de los servidores de alto rendimiento, el mayor costo económico lo acarrea la memoria, asimismo, el mayor consumo de energía también recae en ésta [7].

1.2.2. Memoria *Flash*

La memoria *flash* [8] se basa en el transistor de efecto de campo o FET, por sus siglas en inglés. Este transistor, tal como se muestra en la figura 1.7(a), posee tres terminales: drenador, puerta y fuente. Cuando los tres terminales están al mismo potencial no existe ninguna corriente entre drenador y fuente.

En el caso del transistor de canal N, cuando se aplica un potencial positivo en el drenador y en la puerta (manteniendo la fuente a 0V y si el voltaje entre puerta y fuente crece más allá de un voltaje umbral, este voltaje umbral está definido por el proceso de fabricación del transistor) se establece un canal entre la fuente y el drenador. Este canal es debido al efecto del campo eléctrico generado por el voltaje aplicado a la puerta y permite el tránsito de electrones entre la fuente y drenador creando una corriente eléctrica. Esta corriente hace que el voltaje de salida en la fuente sea similar al voltaje en el drenador lo cual se interpreta como un uno lógico, figura 1.7(b).

Si el potencial en la puerta se aumenta por encima del voltaje de operación normal del transistor, los electrones del canal ganan más energía cinética de lo normal y se disparan hacia la puerta, fenómeno denominado electrón caliente (*hot electron*). Los electrones no logran llegar hasta la puerta sino que quedan atrapados en la puerta flotante (figura 1.7(c)). Ahora, si como antes se colocan los voltajes de operación de la figura 1.7(b), lo que sucede es que el campo eléctrico generado por el voltaje aplicado a la puerta se cancela con el campo eléctrico generado por los electrones atrapados en la puerta flotante, de modo que no se establece el canal entre el drenador y la fuente. Esto hace que el voltaje de salida en la fuente sea cercano a cero, lo cual se interpreta como un cero lógico, como muestra la figura 1.7(d).

Para eliminar la carga de la puerta flotante y devolver el transistor a su estado original, se aplica un voltaje negativo a la puerta y uno positivo a la

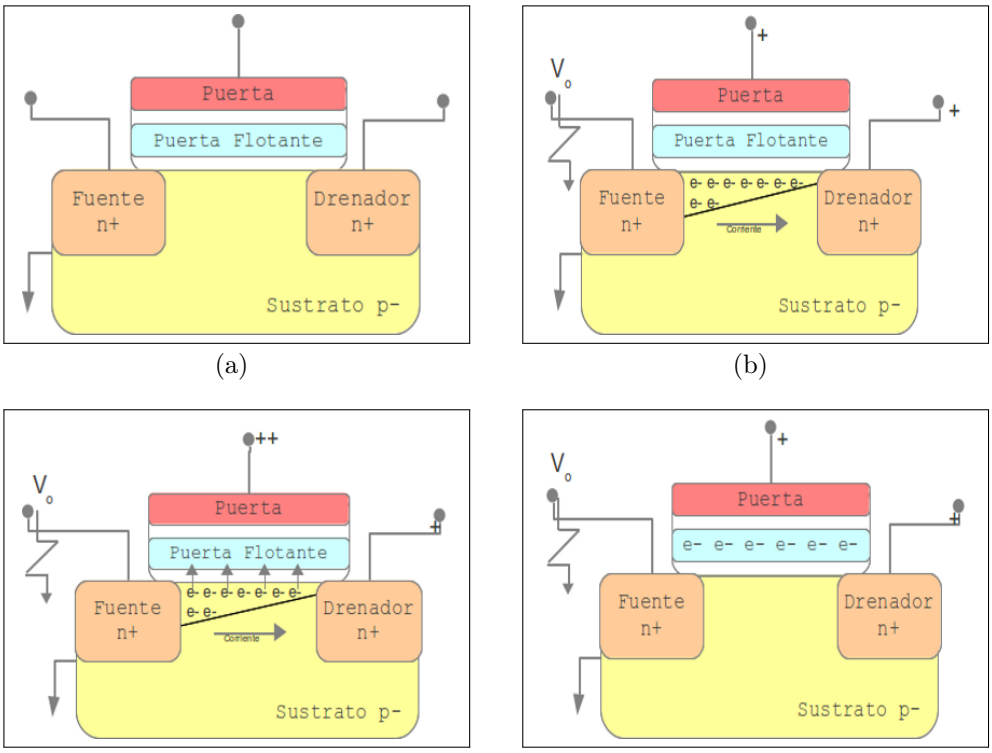


Figura 1.7: Celda de memoria *flash* y sus estados. Celda de memoria *flash* incluyendo una puerta flotante en el transistor FET (a). Funcionamiento normal del transistor, al activar la puerta se lee un uno en la fuente (b). Programación de un cero en el transistor, se aplica un voltaje fuerte en la puerta (c). Lectura de un cero en la fuente, los electrones atrapados en la puerta flotante no permiten la formación del canal de conducción (d).

fuente, haciendo que los electrones se desplacen hacia la fuente. El fenómeno utilizado aquí se denomina túnel *Fowler-Nordheim* (FN). Adicionalmente, el mismo túnel FN se puede usar en lugar del fenómeno del electrón caliente para programar un uno en el transistor.

La meta principal en la fabricación de memorias *flash* [4] ha sido aumentar la capacidad de almacenamiento. Estos esfuerzos han traído como consecuencia adversa un bajo rendimiento en las lecturas y escrituras al compararlas con las tecnologías usadas habitualmente para memoria caché y principal. Sobre todo

por su vida útil y velocidad, esta tecnología no es una opción para sustituir a la memoria DRAM como memoria principal.

1.2.3. RAM Ferro-Eléctrica

En algunos materiales, bajo el efecto de un campo eléctrico externo, las cargas eléctricas se pueden desplazar, agrupándose en un extremo las negativas y en otro las positivas; este fenómeno se llama polarización [4]. En el caso de los materiales dieléctricos la polarización inducida se desvanece al desaparecer el agente externo que produce el campo eléctrico. En otros materiales como son los ferro-eléctricos existen dos niveles estables de polarización, esta polarización llamada remanente no desaparece al eliminar el agente externo que genera el campo eléctrico.

Una celda de memoria ferro-eléctrica (FeRAM) se puede generar al colocar un material ferro-eléctrico entre dos placas conductoras, el circuito finaliza al añadir un transistor de efecto de campo (FET) para prevenir la modificación del dato cuando esto no se desea. La figura 1.8 muestra la celda antes descrita.

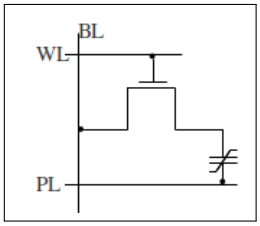


Figura 1.8: Celda de memoria ferro-eléctrica [4]

El mayor atractivo de esta tecnología es que los pulsos de programación pueden ser de 50ns, asimismo, posee una vida útil de 10^{12} ciclos de lectura y escritura. Sus principales desventajas es que la lectura es destructiva y que no presenta una alta escalabilidad, dado que la carga eléctrica que se requiere para ser analizada por un sensor alcanza una escalabilidad de 20nm x 20nm;

esto podría mejorarse con circuitos tridimensionales. Asimismo, se requieren temperaturas de 600 a 700°C para la fabricación del elemento ferro-eléctrico.

1.2.4. RAM Magnética y de Transferencia de Giro de Torque

La estructura de una celda RAM magnética (MRAM) [4] consta de un material no magnético que se coloca entre dos capas materiales ferromagnéticos. La resistencia del sistema varía si los campos magnéticos generados poseen el mismo sentido o sentidos opuestos. Estas estructuras tienen el inconveniente de requerir una densidad de corriente muy alta para ser programadas, del orden de los 10 MA/cm^2.

Las memorias de transferencia de giro de torque (STT-RAM) hacen uso de las corrientes causadas por la interacción con el material ferromagnético para cambiar la orientación del campo magnético y de esta forma programar la memoria. De esta forma se reduce la densidad de corriente a 1 - 5 MA/cm^2.

STT-RAM presenta velocidades de lectura y escritura de 20ns y una vida útil de 10^{12} ciclos de escritura. Actualmente se debe reducir el tamaño de las celdas para poder llegar a ser considerada una memoria de uso universal.

1.2.5. RAM Resistiva

Una celda RAM resistiva [4] (RRAM) se forma al colocar un dieléctrico entre dos electrodos de metal. Normalmente este sistema no conduciría electrones de un electrodo al otro, pero al aplicar un voltaje elevado se forma un filamento o camino de conducción entre los dos extremos, camino que se puede formar por defectos inducidos, como migración de metal entre otros. Aplicando otro voltaje se puede eliminar el filamento y generar de nuevo un camino de alta impedancia entre los dos electrodos.

Esta tecnología aún no está lo suficientemente madura ya que existen una

gran cantidad de materiales que se pueden usar como potenciales celdas de almacenamiento y se continúa analizando el mecanismo que genera la conmutación de los dispositivos así como la capacidad de escalado que tendrá.

1.3. PCM como posible solución

PCM viene de memoria de cambio de fase, por sus siglas en inglés. Esta memoria está formada por un cristal calcógeno que puede tomar dos estados: amorfo, en donde presenta una alta resistividad, y cristalino, en donde posee una resistividad baja. Este material puede conmutar entre estos dos estados un gran número de veces de forma rápida y confiable. La forma de hacerlo cambiar de un estado a otro es mediante pulsos de corriente, que se aplican de forma diferente dependiendo del sentido de la transición. La figura 1.9 muestra el proceso para hacer cambiar una celda PCM entre sus estados: para cristalizar el material se aplica una corriente baja y controlada hasta que el material se caliente sobre su punto de cristalización pero sin llegar a su punto de fusión (esta operación de denomina *set*), para hacer cambiar el material a su estado amorfo se aplica una corriente hasta que el material se caliente por encima de su punto de fusión, (esta operación de denomina *reset*) [7]. El cuadro 1.1 muestra las operaciones de una celda de memoria PCM y su relación con sus estados lógicos y físicos.

Operación	**Valor lógico**	**Impedancia**	**Estado del Material**
Set	1	Baja	Cristalino
Reset	0	Alta	Amorfo

Cuadro 1.1: Estados de una celda de memoria PCM

Una celda de memoria PCM se forma al colocar el material calcógeno entre dos electrodos, con un elemento calefactor en el electrodo inferior que se conecta con este material. La figura 1.10 muestra el esquema de una celda de este tipo.

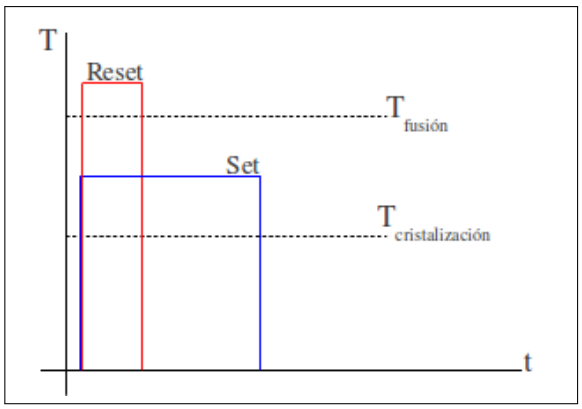


Figura 1.9: Pulsos de programación de una celda PCM, (t=tiempo, T=temperatura)

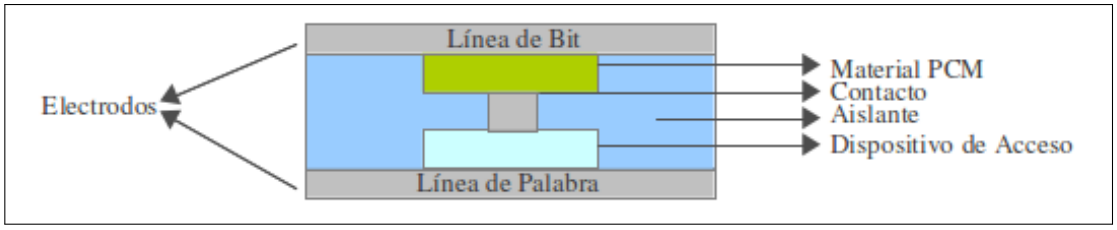


Figura 1.10: Estructura de una celda PCM

PCM logra un nivel de integración mejor que el de DRAM, de hecho se han fabricado prototipos de 3nmx20nm. Asimismo, una celda PCM puede almacenar varios bits y se trata de una memoria no volátil. PCM se presenta así como la mejor opción para sustituir a DRAM, aunque presenta serias deficiencia en lo que respecta a su vida útil. [4, 7, 9–11] ya que su vida útil es de 10^6 a 10^8 ciclos de escritura y borrado mientras que en DRAM se calcula que sobrepasa los 10^{15} ciclos. Esto hace que para tomar en consideración a PCM como un potencial sustituto de DRAM esta deficiencia deba de solventarse. Para ello, recientemente diversos autores plantean modificar ciertos aspectos arquitectónicos y de gestión de la jerarquía de memoria.

El cuadro 1.2 (tomado de [4]) muestra, a modo de resumen, diferentes características para las tecnologías emergentes de memoria antes tratadas, es decir:

Flash, FeRAM, STT-RAM, RRAM y PCM. Por su madurez PCM se presenta como una de las tecnologías más prometedoras para convertirse en la sustituta de DRAM.

Característica	***Flash***	**FeRAM**	**STT-RAM**	**RRAM**	**PCM**
Nodo de tecnología	27 nm	130 nm	54 nm	130 nm	45 nm
Half-pitch	27 nm	225 nm	54 nm	200 nm	52 nm
Área de memoria	0.00375 μm^2	0.252 μm^2	0.041 μm^2	0.168 nm	0.015 μm^2
Tamaño de celda	$5.1F^2/3$	$5F^2$	$14F^2$	$4.2F^2/4$	$5.5F^2$
Capacidad	64 Gb	128 Mb	64 Mb	64 Mb	1 Gb
Velocidad de escritura	7 MBps	83 ns	<15 ns	1-10 ms	100-500 ns
Velocidad de lectura	200 MBps	43 ns	<20 ns	100 ms	85 ns
Vcc	2.7-3.6 V	1.9 V	1.8 V	3-4 V	1.8 V

Cuadro 1.2: Comparación de características de las tecnologías emergentes de memoria.

1.4. Motivación y objetivos

En la actualidad nuevas tecnologías de almacenamiento se hacen necesarias para sustituir a la actual que está agotando su capacidad de mejora. Con estas nuevas tecnologías surgen nuevos retos que pueden llevar a replantearse la forma en que actualmente se realiza la gestión de la memoria. Según los resultados mostrados en el cuadro 1.2, PCM, entre las tercnologías emergentes de memoria, es de las tecnologías más prometedoras para ser la sustituta de DRAM, pero sin duda su vida útil es una desventaja que debe solventarse de alguna manera.

Como consecuencia de que el número de ciclos de escritura en PCM es mucho

menor que en DRAM, se plantea el reenfocar algunos elementos de gestión de memoria, como el de las políticas de reemplazo, a no solo reducir la tasa de fallos y aumentar el rendimiento general del sistema, sino también, a reducir el número de escrituras a memoria principal.

En este trabajo se establecen como objetivos analizar el comportamiento de las políticas de reemplazo de memoria caché clásicas [2] y actuales [12–14] desde el punto de vista del número de escrituras a memoria principal y proponer nuevas políticas o modificaciones de las ya existentes que reduzcan este número de escrituras a memoria principal, sin que generen un detrimento significativo en el rendimiento del sistema.

1.5. Organización del documento

El documento está organizado como sigue:

- El capítulo 2 recopila el trabajo relacionado en el área.
- El capítulo 3 presenta las propuestas realizadas en el trabajo para cumplir con el objetivos.
- El capítulo 4 detalla el entorno experimental utilizado, simuladores y herramientas.
- El capítulo 5 presenta y analiza los resultados experimentales derivados de las propuestas presentadas en el capítulo 3.
- El capítulo 6 lista las conclusiones extraídas del trabajo. Asimismo, en este capítulo se presenta el trabajo futuro que se podría realizar en el área.

Capítulo 2

Trabajo Relacionado

En esta sección se analizan las técnicas propuestas hasta la fecha encaminadas a la reducción de escrituras en memorias PCM para extender su vida útil así como las políticas de reemplazo de caché más representativas (tanto tradicionales como las más recientemente propuestas).

2.1. Técnicas de reducción de escrituras en PCM

Uno de los principales retos que presenta PCM es su reducido número de ciclos de escritura, entre 10^6 y 10^8. Esto hace que se hayan propuesto varias técnicas para aumentar su vida útil, entre ellas destacan:

2.1.1. *Wear Leveling*

Es importante tener un uso uniforme de la memoria, ya que si unas páginas se utilizan más que otras, se provocará que la memoria se desgaste sin haberse aprovechado su potencial vida útil al 100 % dado que algunas celdas agotarán su vida útil mucho antes que otras. En el caso de un uso uniforme se asegura una vida útil de 4 a 20 años para una memoria PCM [9]. Sin embargo, se ha comprobado [4] que los programas no presentan una gran uniformidad en el uso de la memoria, sino por el contrario hacen uso de espacios pequeños de memoria. Para evitar esto, la técnica *wear leveling* propone realizar una

rotación de páginas cada cierto tiempo.

La figura 2.1 muestra el principio de funcionamiento del *wear leveling*, en un caso particular en donde se tienen 8 bloques de memoria principal. Esta técnica añade un bloque extra o bloque de paso, un registro *paso* que apunta al bloque de paso y otro registro *inicio*, que al principio apunta a la primera posición del *array* de bloques de memoria, figura 2.1(a).

Al bloque de paso se mueven los datos del bloque inmediatamente superior y se decrementa el registro *paso* en uno. Esto se muestra en la figura 2.1(b). El proceso continúa, realizando un intercambio cada ψ escrituras a memoria, donde ψ es un parámetro que determina la frecuencia del proceso. Esto se repite hasta que el bloque de paso alcanza la primera posición del *array* de bloques, figura 2.1(c); en este caso la última posición y la primera se intercambian incrementando en uno el registro *inicio*; el estado final del la memoria después de una rotación de todos los bloques se puede ver en la figura 2.1(d). El *Wear Leveling* hace que los bloques se muevan a localidades cercanas, de esta forma asegura el uso uniforme de la memoria.

2.1.2. Escritura a nivel de bloques

Típicamente la memoria principal es accedida a nivel de páginas, sin embargo para reducir el número de escrituras en PCM y de esta forma aumentar la vida útil de la memoria, la escritura a nivel de bloques [7] propone realizar la escritura a una granularidad más fina.

Si las escrituras se pueden manejar en la granularidad del procesador, no a nivel de páginas sino a nivel de bloques, se reduce el número de escrituras innecesarias. Para ello se lleva un registro de los bloques que se han escrito y solo se escriben éstos y no la página entera.

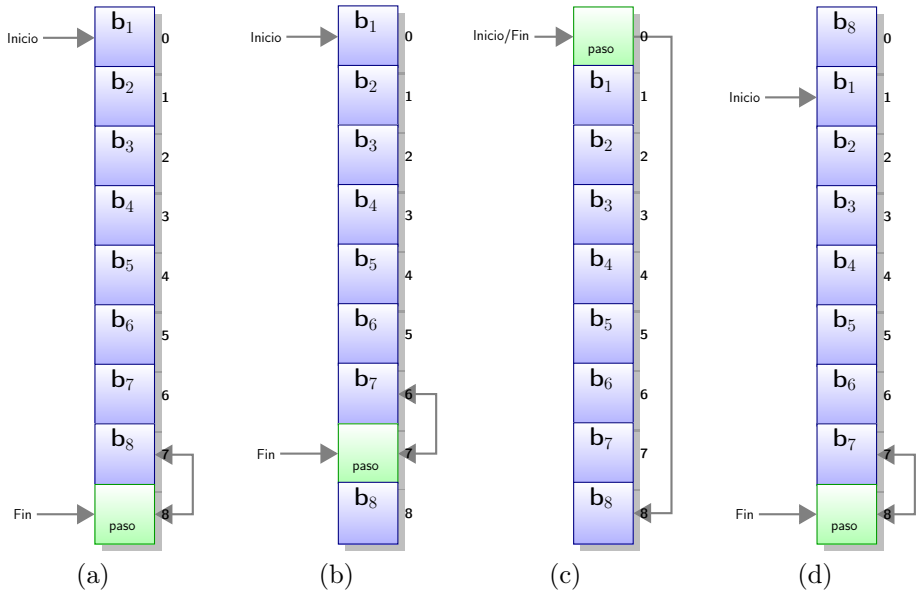


Figura 2.1: Ejemplo de la rotación de los bloques de memoria en *Wear Leveling* [4]

2.1.3. Filtrado de escrituras de granularidad fina

Como parte del proceso para mejorar la vida útil de una memoria PCM el primer paso a tomar sería reducir la frecuencia de las escrituras. En el caso de la memoria DRAM una actualización por escritura escribe toda la página. Sin embargo, se ha visto que muchas de las escrituras que se realizan son redundantes [4, 9] porque muchos bits del bloque no han cambiado.

El filtrado de escrituras de granularidad fina consiste en que a la hora de escribir en memoria, primero se realiza una lectura y se compara lo que se va a escribir con lo que se encuentra en memoria, escribiéndose únicamente aquellos bits que difieren. Esto no resulta muy problemático, ya que en PCM las lecturas son mucho más rápidas que las escrituras y, dado que las escrituras no son por lo general prioritarias, el rendimiento del sistema no se ve significativamente afectado. Con esta técnica se consigue prolongar la vida útil de la memoria hasta 22 años [9].

2.1.4. *Flip-N-Write*

La técnica de *Flip-N-Write* [15] propone para mejorar el tiempo de vida de la memoria PCM reducir el ancho de banda de las escrituras, es decir el número de bits que se escriben a PCM. Para esto se compara el nuevo dato a escribir con el antiguo y se aplica una codificación al nuevo dato tal que reduzca el número de bits a escribir.

Esta codificación consiste en que al escribir un dato se comparan, bit a bit, tanto el dato que se desea escribir como su inverso con el dato presente en memoria; el que presente menos cambios es el que se escribe, escribiendo solo los bits que cambian. Para indicar si se ha almacenado el dato o su inverso lógico, se agrega a cada bloque un bit extra.

2.2. Políticas de reemplazo de caché

Cuando el procesador solicita un dato o instrucción que no se encuentra disponible en memoria caché se produce lo que se denomina un fallo, en este caso el dato solicitado se tiene que ir a buscar al nivel inmediatamente inferior de la jerarquía de memoria, sea otro nivel de memoria caché, memoria principal o secundaria. Cuando el bloque que contiene ese dato se lleva al nivel en que se originó el fallo, si este nivel está lleno o la posición o posiciones de memoria (dependiendo de la asociatividad) en las que el bloque pueden ser almacenado están ocupadas, se debe desplazar un bloque para almacenar el solicitado. La función de determinar qué bloque desplazar la realizan los algoritmos de reemplazo.

A continuación se describen las políticas de reemplazo caché más representativas, empezando por las políticas clásicas para continuar con las más recientes.

2.2.1. Política Aleatoria (Random)

Al usar este algoritmo se selecciona el bloque de cache a reemplazar en forma aleatoria [2], sin atender a ningún tipo de información referente a la temporalidad de las referencias a memoria hechas por el procesador.

2.2.2. Política de Bélády

En 1966 Bélády [16] determinó lo que sería el algoritmo de reemplazo más eficiente, esto es, se debe descartar el bloque que no será requerido por mayor tiempo en el futuro. Dado que no es posible saber cuál es ese bloque, este algoritmo no puede desarrollarse en la práctica, de ahí que se hayan propuesto muchos algoritmos que tratan de aproximar este comportamiento.

Para ilustrar la idea del algoritmo de Bélády, la ecuación 2.1 muestra un patrón de referencias a memoria en el que se referencia el bloque que contiene el dato a_1 una vez al principio y otra vez al final de la secuencia mientras que entre ambas solicitudes se hace referencia n veces una secuencia de bloques que contienen los datos b_1, b_2, b_3, b_4.

$$\{a_1, (b_1, b_2, b_3, b_4)^n, a_1\} \qquad \text{con } n > 1 \tag{2.1}$$

En la figura 2.2 se muestra el estado de una caché de cuatro bloques después de las cuatro primeras referencias. Dado el patrón de acceso de la ecuación 2.1, se tiene que el bloque óptimo a sustituir según Bélády cuando se hace la quinta referencia (b_4), es el bloque que contiene el dato a_1. Esto es debido a que este bloque contiene el dato que no será requerido por el mayor tiempo.

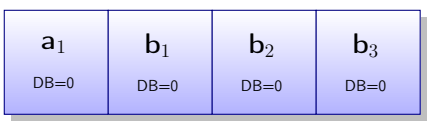


Figura 2.2: Caché después de los primeros 4 accesos de la secuencia 2.1

2.2.3. Política LRU

El algoritmo LRU [2] (menos recientemente utilizado por sus siglas en inglés), trata de reducir la posibilidad de descartar bloques que van a ser utilizados de nuevo en un futuro cercano. Para ello se almacena la información relativa al uso de los bloques de forma que en un reemplazo se descarta el bloque que no ha sido utilizado por más tiempo. LRU se basa en el principio de localidad temporal, según el cual si un bloque ha sido utilizado recientemente es probable que sea utilizado de nuevo en un futuro cercano, de ahí que el mejor candidato para reemplazar sea el bloque que no ha sido utilizado por más tiempo.

La implementación de este algoritmo requiere llevar un registro del uso de cada bloque de modo que la actualización del uso de un bloque implica actualizar el de todos los demás, de ahí que esta implementación tiende a ser muy costosa para caches de gran asociatividad. Este algoritmo funciona como una estructura de datos tipo cola, de ahí que se refiere a ella como cola LRU. El principio de la cola (cabecera), y por ello el candidato a abandonarla primero, es el bloque al que menos recientemente se haya hecho referencia (bloque LRU). Asimismo, el final de la cola (cola) y por ello el candidato a permanecer más tiempo en ella, es el bloque al que más recientemente se haya hecho referencia (bloque MRU, del inglés más recientemente usado).

Dado que este algoritmo se basa en el principio de localidad temporal, presenta un buen desempeño ante cargas con elevada localidad temporal en los datos, pero en el caso de cargas donde las nuevas referencias ocurren en el futuro lejano, LRU presenta un mal rendimiento. Como ejemplo de este último tipo de cargas se tiene el caso en el que el conjunto de trabajo es mayor que la capacidad de la caché o cuando un grupo de datos sin ningún tipo de reuso supone el reemplazo de los datos del conjunto de trabajo [13].

La figura 2.3(a) muestra la cola LRU, en una situación inicial dada. La

figura 2.3(b) presenta el estado de la cola LRU después de un acceso al bloque b_1. Al darse un acierto, el bloque al que se hace referencia pasa a la posición MRU de la cola y los demás bloques se mueven una posición hasta que alguno llega a ocupar el lugar del bloque al que se ha hecho referencia.

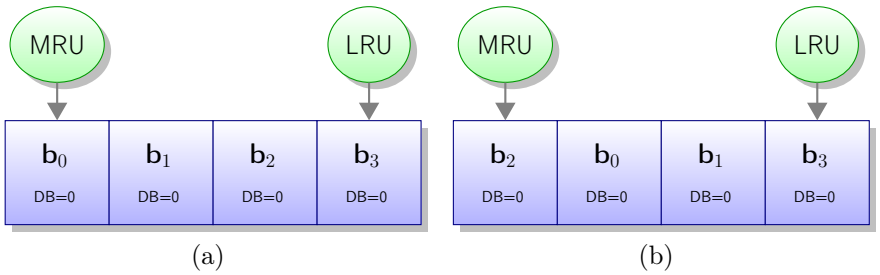


Figura 2.3: Caché de cuatro bloques que usa la política LRU. Estado inicial de la caché (a). Estado de la caché después de una referencia al bloque b_1 (b)

Hasta este punto se han descrito las políticas convencionales y a continuación se presentan las políticas más recientes y que mejor rendimiento presentan en cuanto a tasa de aciertos se refiere.

2.2.4. Política peLIFO

La política peLIFO [12] se basa en el algoritmo LIFO (del inglés el último que entra es el primero que sale). Para la implementación de este último se usa una pila de llenado en la que se almacenan los bloques en el orden en el que ingresan en la caché. El bloque que entra en el último puesto de la pila es el candidato para ser reemplazado. LIFO posee un mal rendimiento ya que al contrario de LRU desaloja de la caché los bloques que han ingresado más recientemente lo cual se opone al principio de localidad temporal. El autor de peLIFO hace varias optimizaciones para mejorar el pésimo rendimiento de la política LIFO.

En el caso de LIFO se tiene una única posición en la pila de llenado en la que

se aloja el candidato a reemplazar, esta posición se denomina punto de salida o de escape. Entre las optimizaciones planteadas en peLIFO está el definir, dinámicamente, varias puntos de salida, en lugar de tener uno fijo como sucede en LIFO. Una comparaiva del modo de operación de ambas políticas se puede apreciar en la figura 2.4. Sin embargo, tal como sucede en LIFO, peLIFO trata de mantener la parte inferior de la pila para el reuso a largo plazo.

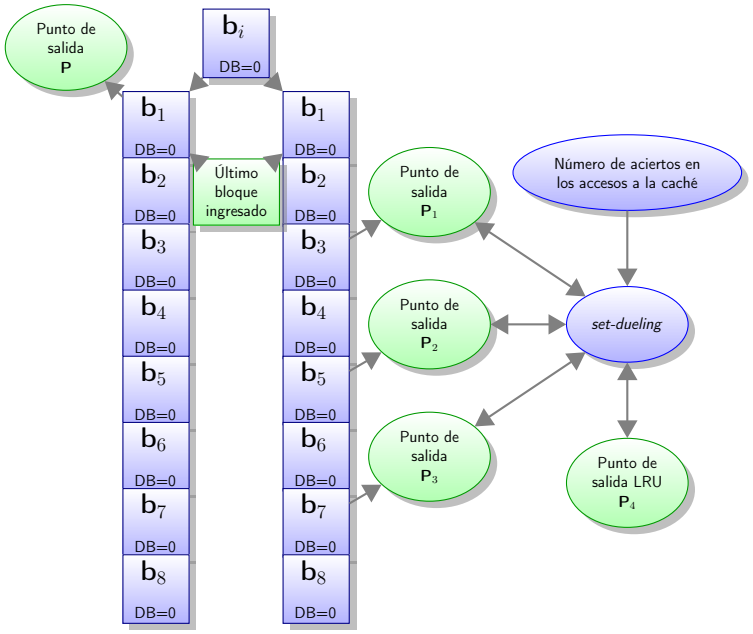


Figura 2.4: Operaciones de LIFO (izquierda) y peLIFO (derecha)

La gestión que realiza la política peLIFO se basa en el uso de dos conceptos:

1. Probabilidad de Escape: probabilidad, para una posición k de la pila, de que los bloques alojados en posiciones mayores de k tengan aciertos. Como se define en [17], «ésta se calcula como el número de bloques de caché que generan al menos un acierto en posiciones de la pila de llenado mayores a k dividido por el número de bloques ingresados en la caché».

2. Punto de Escape: posición en la pila tal que la probabilidad de escape se reduce por debajo de cierto umbral, de modo que los bloques que se

encuentran en esta posición son candidatos para ser reemplazados.

De esta forma se definen dos áreas en la pila de llenado: la parte superior y la parte inferior. La primera contiene los bloques que han ingresado recientemente y se utiliza para realizar la mayoría de los desalojos, siempre que los bloques hayan satisfecho su reuso a corto plazo. Por otro lado, la parte inferior se destina para el reuso a largo plazo [18].

La forma en que funciona esta política es el siguiente:

1. Inicialmente se utiliza la política LRU en cada reemplazo.

2. Se leen un conjunto de contadores los cuales se actualizan de acuerdo con los aciertos que se dan en la pila de llenado. A partir de estos contadores se determinan las probabilidades de escape, esto se hace cada N ciclos, el tiempo entre el cálculo de probabilidades de escape se denomina una *época*.

3. Usando las probabilidades de escape calculadas en el punto 2 se analizan cuatro políticas (en cuanto a número de aciertos se refiere): tres políticas peLIFO basadas en tres puntos de escape asociados (P_1, P_2, P_3) y la política LRU (P_4), este análisis se hace mediante el mecanismo de *Set Dueling* [19]. *Set Dueling* reserva algunos conjuntos de cache para gestionarlos cada uno de ellos con una política fija diferente, basado en el resultado de esta muestra se elige la política que presenta el mejor desempeño para gestionar el resto de la caché.

4. Si la diferencia entre las probabilidades de escape de dos épocas consecutivas es mayor que cierto umbral, se da lo que se denomina un cambio de fase, se recalculan los puntos de escape y se mantiene la política peLIFO hasta que se de un nuevo cambio de fase, en cuyo caso se vuelve al punto

1; en caso contrario el funcionamiento de la política se mantiene entre los puntos 2 a 4.

5. Para determinar el bloque a reemplazar se busca en el conjunto el bloque que cumpla lo siguiente:

   a) El bloque no ha generado un acierto en su posición actual en la pila de llenado

   b) Su posición actual en la pila de llenado es mayor o igual que el punto de escape p_i

   c) De no encontrarse un bloque que cumpla lo anterior se usa la política LRU.

2.2.5. Política DRRIP

Debido a lo inviable que resulta el desarrollar la política LRU en sistemas con una alta asociatividad de caché, se utiliza la política NRU (del inglés no usado recientemente) como una aproximación de la misma [20, 21].

La política NRU utiliza un bit extra por cada bloque de caché, este bit se usa para tratar de predecir el reuso del bloque, es decir si se hará referencia a éste en un futuro *cercano* (0) o *distante* (1). El algoritmo funciona de la siguiente forma:

1. Al inicio todos los bit NRU se asignan como predicción de acceso *distante*, es decir todos los bit NRU se ponen a 1.

2. Al insertar un bloque nuevo en la caché o al darse un acierto en un acceso a un bloque existente, el bit NRU correspondiente a este bloque se asigna a cero para indicar una predicción de acceso *cercano*.

3. Al tener que desalojar un bloque se busca el primer bloque (de acuerdo al orden de ingreso) cuyo bit NRU sea para una predicción de acceso *distante*.

4. De no haber ningún bloque con su bit NRU indicando una predicción de acceso *distante*, se cambian los bit NRU de todos los bloques a este estado.

Teniendo únicamente dos estados posibles, la política NRU limita el rendimiento de la memoria caché cuando se encuentra frente a patrones de acceso que son de tipo *búsqueda*, en donde los bloques a los que se hace referencia no serán reutilizados. Es decir, NRU podría asignar una probabilidad de acceso *cercano* a bloques que no van a recibir un nuevo acceso, permitiendo de esta forma que se contamine la caché con bloques que no van a ser reutilizados.

Para solventar el problema de NRU, RRIP [13] (del inglés intervalo de predicción de un reuso), asigna más bits de estado a cada bloque de forma que además de poder predecir un reuso *cercano* y *distante* se pueda predecir uno *lejano*, esto es un intervalo de predicción intermedio pero ligeramente desviado al *distante*. Este *array* de N bits de predicción se denomina RRPV (del inglés valores de predicción de reuso). El cuadro 2.1 muestra el valor del RRPV para cada tipo predicción de acceso.

Predicción de acceso	**Valor para N bits**	**Valor para 2 bits**
cercano	0	0
lejano	$2^N - 2$	2
distante	$2^N - 1$	3

Cuadro 2.1: Valor del RRPV para cada tipo predicción de acceso

El algoritmo RRIP funciona de la siguiente manera:

1. Al inicio para todos los bloques el RRPV se asigna como predicción de

acceso *distante*. Cada conjunto de caché puede verse como un *array* de tantas posiciones como asociatividad tenga la caché, en RRIP estas posiciones siempre se recorren de la misma forma, de izquierda a derecha si se visualiza como un *array* horizontal, siendo la primera posición la de más a la izquierda y la última la de más a la derecha.

2. Al insertar un bloque en la caché se asigna su RRPV a predicción de acceso *lejano*.

3. Al tener que desalojar un bloque se busca el primer bloque en el conjunto cuyo RRPV sea para una predicción de acceso *distante*.

4. De no haber ningún bloque con su RRPV indicando una predicción de acceso *distante*, se incrementa en uno el RRPV de todos los bloques del conjunto hacia predicción de acceso *distante* y se vuelve al punto 3.

5. Cuando se da un acierto, el valor RRPV del bloque correspondiente se debe mover hacia predicción de acceso *cercano*. Esto se puede hacer de dos maneras.

    a) Cambiándolo directamente a predicción de acceso *cercano*, esto se define como RRIP con prioridad de acierto (RRIP-HP)

    b) Modificar en uno el valor RRPV hacia predicción de acceso *cercano* (decrementar el RRPV según cuadro 2.1), esto se define como RRIP con prioridad de frecuencia (RRIP-FP)

Para ilustrar el comportamiento de diferentes políticas vistas hasta ahora, la ecuación 2.2 muestra un patrón de referencias a memoria el cual se desarrolla en el cuadro 2.2 para cuatro políticas diferentes: Béládi, LRU, NRU y RRIP-HP. En dicho cuadro se utiliza una caché de cuatro bloques totalmente asociativa, y en el caso de las políticas NRU y RRIP-FP cada bloque tiene un subíndice

que indica el valor del bit NRU y de RRPV respectivamente, siendo el RRPV utilizado de dos bits.

$$\{a_1^n, (c_1, c_2, c_3, c_4, c_5), a_1^n\} \qquad \text{con } n = 2 \tag{2.2}$$

Ref	Bélády	LRU	NRU	RRIP-HP
a_1	x_0 x_0 x_0 x_0	*MRU* *LRU* x_0 x_0 x_0 x_0	$x_0{}_1$ $x_0{}_1$ $x_0{}_1$ $x_0{}_1$	$x_0{}_3$ $x_0{}_3$ $x_0{}_3$ $x_0{}_3$
a_1	a_1 x_0 x_0 x_0	*MRU* *LRU* a_1 x_0 x_0 x_0	$a_1{}_0$ $x_0{}_1$ $x_0{}_1$ $x_0{}_1$	$a_1{}_2$ $x_0{}_3$ $x_0{}_3$ $x_0{}_3$
c_1	a_1 x_0 x_0 x_0	*MRU* *LRU* a_1 x_0 x_0 x_0	$a_1{}_0$ $x_0{}_1$ $x_0{}_1$ $x_0{}_1$	$a_1{}_0$ $x_0{}_3$ $x_0{}_3$ $x_0{}_3$
c_2	a_1 c_1 x_0 x_0	*MRU* *LRU* c_1 a_1 x_0 x_0	$a_1{}_0$ $c_1{}_0$ $x_0{}_1$ $x_0{}_1$	$a_1{}_0$ $c_1{}_2$ $x_0{}_3$ $x_0{}_3$
c_3	a_1 c_1 c_2 x_0	*MRU* *LRU* c_2 c_1 a_1 x_0	$a_1{}_0$ $c_1{}_0$ $c_2{}_0$ $x_0{}_1$	$a_1{}_0$ $c_1{}_2$ $c_2{}_2$ $x_0{}_3$
c_4	a_1 c_1 c_2 c_3	*MRU* *LRU* c_3 c_2 c_1 a_1	$a_1{}_0$ $c_1{}_0$ $c_2{}_0$ $c_3{}_0$	$a_1{}_0$ $c_1{}_2$ $c_2{}_2$ $c_3{}_2$
c_5	a_1 c_4 c_2 c_3	*MRU* *LRU* c_4 c_3 c_2 c_1	$c_4{}_0$ $c_1{}_1$ $c_2{}_1$ $c_3{}_1$	$a_1{}_1$ $c_4{}_2$ $c_2{}_3$ $c_3{}_3$
a_1	a_1 c_4 c_5 c_3	*MRU* *LRU* c_5 c_4 c_3 c_2	$c_4{}_0$ $c_5{}_0$ $c_2{}_1$ $c_3{}_1$	$a_1{}_1$ $c_4{}_2$ $c_5{}_2$ $c_3{}_3$
a_1	a_1 c_4 c_5 c_3	*MRU* *LRU* a_1 c_4 c_3 c_2	$c_4{}_0$ $c_5{}_0$ $a_1{}_0$ $c_3{}_1$	$a_1{}_0$ $c_4{}_2$ $c_5{}_2$ $c_3{}_3$

Cuadro 2.2: Resultado de utilizar el patrón de acceso de la ecuación 2.2 en una cache de 4 vías con las políticas Bélády, LRU, NRU y RRIP-HP

En el cuadro 2.2 se muestra como RRIP-HP es la política que se acerca más al comportamiento ideal (Bélády). Según los resultados presentados por el autor es justamente la política RRIP-HP la que menor tasa de fallos. Asimismo, el autor se refiere a esta política como RRIP estática (SRRIP-HP).

SRRIP hace un uso ineficiente de la caché cuando entre cada reuso de los bloques hay más referencias a bloques que ocupan el mismo conjunto que vías disponibles. Para corregir esto, los autores definen la política RRIP de dos modos (BRRIP). Esta política asigna el RRPV de un nuevo bloque de dos formas, asignando con una alta probabilidad un RRPV con una predicción de acceso *distante* y asignando de forma ocasional un RRPV con una predicción de acceso *lejano*, es decir mayoritariamente funciona como RRIP-HP con inserción como acceso *distante* y ocasionalmente como RRIP-HP con inserción como acceso *lejano*.

Debido a que BRRIP no hace un uso eficiente de la memoria para patrones distintos para los que fue creado, se plantea el uso de un RRIP dinámico (DRRIP) el cual usa *set-dueling* [19] para determinar la mejor política a usar entre RRIP-HP con inserción como acceso *lejano* y BRRIP.

2.2.6. Política SHiP

Mientras que varios de los algoritmos propuestos hasta el momento predicen, en su mayoría, el mismo intervalo de reuso para la mayor parte de las inserciones (lo que resulta en decisiones de una granularidad gruesa), el algoritmo SHiP [14] (predictor de acierto basado en firma, por sus siglas en inglés), propone realizar una predicción de grano fino categorizando las referencias hechas a los bloques de caché dentro de diferentes grupos, lo cual se hace asociando una firma con cada referencia a caché. La meta es que las referencia de caché que tengan la misma firma lleguen a tener un intervalo de reuso similar.

SHiP es un mecanismo de inserción que se puede usar con cualquier política de reemplazo. Lo que determina SHiP es el reuso que va a tener el bloque que se inserta, estableciendo cual debería ser su predicción de acceso. En el caso particular de SRRIP determinaría si el valor de RRPV debe ser cercano, lejano

o distante.

Para realizar la predicción SHiP hace uso de un *array* de contadores llamado cuadro de contadores de historia de firma (SHCT por su siglas en inglés). En este caso SHCT es un *array* de 16384 posiciones siendo, cada uno de los contadores de dos bits.

Se puede determinar la firma de un bloque de tres formas distintas (aunque solo se usa un método a la vez): firma de región de memoria, de contador de programa y de historia de secuencia de instrucciones. La firma de contador de programa, según los resultados mostrados por los autores, es la muestra los mejores resultados.

Además del *array* SHCT, SHiP agrega a cada bloque de caché dos campos extra, uno para almacenar la firma de ese bloque de caché (*Signature*) y otro para indicar si ese bloque ha sido reutilizado de nuevo (*Outcome*). El campo *Outcome* es de un bit, siendo 0 al inicio y modificado a 1 si el bloque presenta un reuso.

El algoritmo funciona de la siguiente manera:

1. Al inicio todos los campos del *array* SHCT se inicializan a cero.

2. Al insertar un bloque en la cache se determina su firma, ésta se almacena en *Signature* y *Outcome* se pone a 0.

3. Si se da un acierto en un bloque de caché el bit *Outcome* se cambia a uno y el contador de la posición *Signature* del *array* SHCT se incrementa en uno.

4. Cuando se desaloja un bloque de la caché, si su bit *Outcome* es cero se decrementa el contador de la posición *Signature* del *array* SHCT.

5. Al insertarse un nuevo bloque a la caché su bit *Outcome* se pone a 0 y se calcula su *Signature*. Del *array* SHCT se lee el valor del contador

correspondiente a la posición de la *Signature* antes calculada, si este valor es cero se le asigna al nuevo bloque ingresado una predicción de acceso *distante*, si el valor es diferente de cero se establece una predicción de acceso *intermedio*. En el caso particular de utilizar SHiP con RRIP esta predicción de acceso *intermedio* correspondería a una predicción de acceso *lejano*.

2.2.7. Política CLP

La política de preferencia de víctima limpia o CLP [22] por sus siglas en inglés, es el primer algoritmo de reemplazo de caché propuesto explícitamente para reducir el número de escrituras en memoria PCM. Esta política propone hacer una variación del algoritmo LRU y en lugar de reemplazar el bloque menos recientemente usado, reemplazar el menos recientemente usado que no presente escrituras (es decir que no haya sido modificado desde su ingreso en caché).

Dado que esta técnica puede afectar gravemente el rendimiento, debido a que la memoria caché se puede llenar solo de bloques que presenten escrituras (sucios) y que sean poco utilizados, proponen lo que denominan *N-chance*. Esto es, fijan el número de intentos que se van a hacer en búsqueda de un bloque que no presente escrituras. Específicamente, cuando se requiere desalojar un bloque se busca el menos recientemente usado que no presente escrituras, pero esto solo se hace N veces (se limita la búsqueda a N bloques). De no encontrarse ninguno se reemplaza el menos recientemente usado aunque éste esté sucio. Esto previene llenar la caché solo de bloques poco utilizados que presenten escrituras. N es un número entre 1, que equivaldría al algoritmo LRU, y la asociatividad de la caché.

La figura 2.5 muestra una memoria caché de ocho bloques que utiliza el

algoritmo CLP. En la figura además del nombre del bloque (b_x), se detalla el valor del *dirty bit* (DB). En el caso de que se requiera desalojar un bloque según los algoritmos LRU y CLP con *N-chance* igual a uno, el bloque a desalojar sería b_7 en ambos casos. Al variar *N-chance* a dos el bloque a desalojar sería también b_7 dado que el siguiente bloque a desalojar después de b_7 está también sucio. Al incrementar *N-chance* a cinco, el bloque a desalojar sería b_3 ya que se hacen cinco intentos de encontrar un bloque limpio y en este caso b_3 está limpio y corresponde al quinto bloque después del bloque LRU.

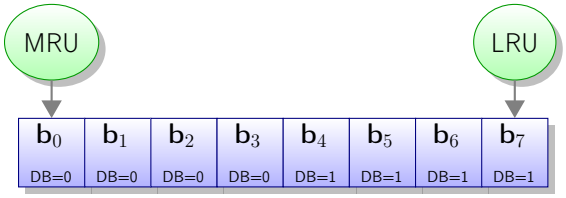


Figura 2.5: Caché de ocho bloques que usa la política CLP

Capítulo 3

Propuesta de Políticas de Reemplazo en PCM

Antes de detallar las políticas de reemplazo propuestas para reducir el número de escrituras a memoria PCM, se resumirán los distintos aspectos generales sobre los que se actúa para diseñar un algoritmo eficiente en este tipo de entornos PCM.

- Cambiar la inserción de nuevos bloques: se hace de forma que a los bloques accedidos para escribir (escrituras) se les asigne una preferencia mayor que a los bloques accedidos para leer (lecturas). Esta preferencia depende del mecanismo utilizado por la política original sobre la que se plantean modificaciones para optimizarla a entornos PCM. Este aspecto trata de asegurar que los bloques que presenten escrituras van a mantenerse más tiempo en la caché que los bloques no las presenten. Esto se hace para darle más oportunidad a los bloques modificados de ser reutilizados.

- Modificar la promoción de los bloques: se refiere a como se trata un bloque sobre el que ha habido un acierto, para que permanezca más tiempo en la caché. La idea de esta técnica es darle preferencia al reuso de bloques que presenten escrituras sobre los bloques que no las presentan, al igual que el caso anterior esta técnica persigue mantener más tiempo en la caché

los bloques modificados.

- Al desalojar un bloque de caché dar preferencia a los bloques que no presenten modificaciones (limpios).
- Hacer que se tome en cuenta el desalojo de bloques que generan una escritura retardada a memoria principal en el mecanismo que elige entre varias políticas que se debaten en el *set-dueling*.
- Utilizar en el *set-dueling* políticas que presenten un buen comportamiento en cuanto a reducción de escrituras a memoria principal.

A continuación se plantean las políticas propuestas para reducción de escrituras en memoria principal. Estas políticas se basan en LRU (por ser la política comúnmente aceptada como referencia sobre la que comparar), y también en DRRIP y peLIFO dado que son las políticas que proporcionan mejores resultados en cuanto a rendimiento se refiere.

3.1. Políticas Basadas en LRU

3.1.1. Política WPP1

Este algoritmo propone ingresar los bloques accedidos para escribir (escrituras) como el bloque más recientemente usado (MRU) y los bloques accedidos para leer (lecturas) en algún otro punto de la cola LRU, por ejemplo en la posición *asociatividad medios* (asociatividad/2).

La figura 3.1 muestra una caché de cuatro vías que usa una cola LRU modificada para la política WPP1, en esta figura se muestra como el punto de ingreso para las lecturas se desplaza a la *asociatividad medios*, mientras que las escrituras siguen insertándose en MRU.

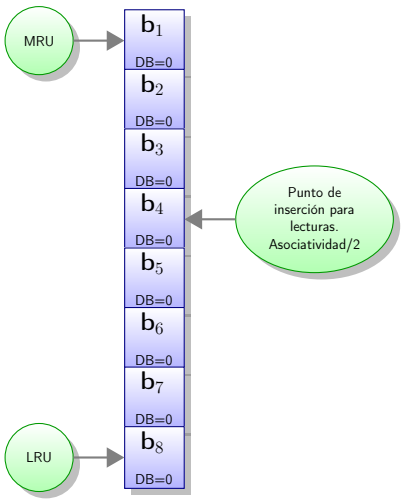


Figura 3.1: Cola LRU para la política WPP1

3.1.2. Política WPP2

Se basa en WPP1, pero además de ingresar las lecturas en otro punto distinto al MRU, propone que los bloques limpios se promuevan hasta el punto de ingreso y no hasta el bloque MRU, mientras que los bloques sucios sí pueden ser promovidos hasta el bloque MRU.

La figura 3.2 muestra una caché de cuatro vías que usa una cola LRU modificada para la política WPP2, en esta figura se muestra cómo el punto de ingreso para las lecturas se desplaza a la *asociatividad medios*; asimismo, este nuevo punto corresponde al máximo punto de promoción para un bloque limpio. Por otra parte, la inserción y promoción de una escritura sigue haciéndose hasta que ésta ocupe la posición MRU.

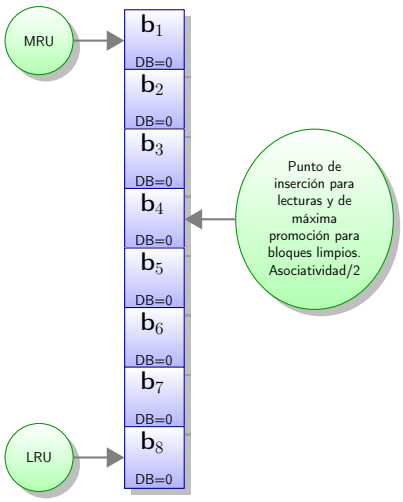


Figura 3.2: Cola LRU para la política WPP2

3.2. Políticas Basadas en DRRIP

3.2.1. Política DRRIPW1

Se propone replantear la promoción de los bloques y en lugar de promover todos a acceso *cercano* (RRIP-HP) se propone tratar los aciertos a bloques limpios como RRIP-FP (al darse un acierto en un acceso a un bloque si su RRPV es mayor que cero se decrementa en uno) y los aciertos a bloques sucios como RRIP-HP (al darse un acierto en un acceso a un bloque se cambia el valor de su RRPV a cero).

3.2.2. Política DRRIPW2

En este caso se cambia la inserción, de forma que la inserción de la política BRRIP se cambia de mayoritariamente acceso *distante* a:

- Al ingresar una escritura se asigna su RRPV para indicar un acceso *lejano*
- Al ingresar una lectura se asigna su RRPV para indicar un acceso *distante*

Asimismo, tanto en la parte ocasional de la política BRRIP como en RRIP-HP con inserción como acceso *lejano*, se modifica la inserción de la siguiente forma:

- Al ingresar una escritura se asigna su RRPV para indicar un acceso *cercano*
- Al ingresar una lectura se asigna su RRPV para indicar un acceso *lejano*

3.2.3. Política DRRIPW3

Se modifican la inserción y la promoción de los bloques realizando los cambios planteados en DDRIPW1 y DRRIPW2. Estos cambios se hacen para favorecer la inserción de escrituras y la promoción de bloques sucios sobre la inserción de lecturas y la promoción de bloques limpios, de forma que se propicie que los bloques que sean candidatos a generar una escritura retardada a memoria principal permanezcan más tiempo en la caché y así aumente su oportunidad de ser reutilizados antes de que sean desalojados.

3.2.4. Política DRRIPW4

En DRRIP la política elegida es la que presenta el mejor desempeño en el *set-dueling*, éste basa la elección en un contador que se incrementa o decrementa según el número de aciertos que se generan en los conjuntos reservados para cada política. Se propone que en lugar de que la política a elegir se determine por los aciertos que genera, se determine en base al menor número de reemplazos que generen una escritura en memoria principal.

3.2.5. Política DRRIPW5

Se propone ampliar el número de bits usados en el RRPV de dos a tres, esto hace que un acceso *lejano* tenga un valor de cinco y uno *distante* de siete, tal como se establece en el cuadro 2.1. Se permite que los bloques limpios sean promovidos hasta un acceso *lejano* y los sucios a uno *cercano*.

3.2.6. Política DRRIPW6

Se mantiene la política fija (RRIP-HP inserción *lejana*) y se cambian las políticas de BRRIP de RRIP-HP inserción mayoritariamente *distante* y ocasionalmente *lejana* a RRIP-FP inserción *distante* y ocasionalmente *lejana*, dado que son las políticas SRRIP que presentan mejores resultados en cuanto a número de escrituras a memoria principal.

3.2.7. Política DRRIPW7

Se realiza una combinación de DRRIPW4 y DRRIPW6. Es decir, se hace que el menor número de reemplazos que generen una escritura retardada en memoria principal sea el factor que utilice el *set-dueling* para determinar la política a utilizar y se modifica la política utilizada en BRRIP a RRIP-FP inserción *distante* y ocasionalmente *lejana*.

3.2.8. Política DRRIPW8

Se combinan DRRIPW4 y DRRIPW1, para ambas políticas del *set-dueling*, es decir en el caso de la promoción de los bloques se tratan los aciertos a bloques limpios como RRIP-FP y los de bloques sucios como RRIP-HP.

3.2.9. Política DRRIPW9

Se combinan DRRIPW4 y DRRIPW1, solo para la política fija del *set-dueling*, es decir RRIP-HP inserción *lejana* manteniendo la política del BRRIP como en DRRIPW6. Esto es, en BRRIP siempre se utiliza RRIP-FP pero en la política fija se tratan los aciertos a bloques limpios como RRIP-FP y los de bloques sucios como RRIP-HP.

3.3. Políticas Basadas en peLIFO

3.3.1. Política peLIFO1

Es exactamente igual que peLIFO original, pero se añade una condición más para determinar qué bloque se debe reemplazar en cada desalojo:

1. El bloque no ha generado un acierto en su posición actual en la pila de llenado
2. Su posición actual es la pila de llenado es mayor o igual que el punto de escape p_i
3. El bloque está limpio
4. Si no se encuentra ningún bloque que satisfaga estas condiciones, se escoge el bloque LRU

3.3.2. Política peLIFO2

Exactamente igual que peLIFO1, pero en el caso de no encontrar ningún bloque que satisfaga las condiciones se escoge el bloque LRU que esté limpio (una combinación de peLIFO1 y CLP).

3.3.3. Política peLIFO3

Como peLIFO original, pero en vez de calcular las probabilidades de escape como la probabilidad de que los bloques de la caché que están en posiciones de la pila de llenado mayores que k generen aciertos, que sea la probabilidad de que los bloques de la caché que están en posiciones mayores que k generen aciertos de escritura.

3.3.4. Política peLIFO4

Como peLIFO3, pero a la hora de escoger el punto de escape a utilizar, en vez de hacerlo en base al número de fallos de cada política P_1, P_2, P_3 y P_4, se hace en base al número de aciertos de escritura. La mejor política y por tanto la elegida es la que más aciertos de escritura tiene.

3.3.5. Política peLIFO5

Como peLIFO3, pero a la hora de escoger el punto de escape a utilizar, en vez de hacerlo en base al número de fallos de cada política P_1, P_2, P_3 y P_4, se hace en base al número de fallos que provoquen una escritura en el siguiente nivel de la jerarquía, memoria principal en este caso.

3.3.6. Política peLIFO6

Como peLIFO original, pero a la hora de escoger el punto de escape a utilizar, en vez de hacerlo en base al número de fallos de cada política P_1, P_2, P_3 y P_4, se hace en base al número de fallos que provoquen una escritura en memoria principal. Esta política es similar a peLIFO5, la diferencia está a la hora de calcular las probabilidades de escape: en peLIFO5 se hace según lo establecido en peLIFO3, no según lo establecido en peLIFO original como se hace aquí.

Capítulo 4

Entorno Experimental

En esta sección se describe el entorno de simulación utilizado en la realización de este trabajo, además se describen las herramientas involucradas en el desarrollo del mismo.

En términos generales se utiliza el simulador de caché *MultiCacheSim*. Como entrada al simulador se hace uso de Pin, que realiza instrumentación binaria en tiempo de ejecución de las aplicaciones a ejecutar (SPEC2006). Se usa CACTI y datos de artículos actuales para generar un modelo de consumo de energía y de tiempo medio de acceso a memoria.

4.1. Pin

La instrumentación es una técnica cada vez más común en la que se utiliza código extra para realizar el análisis de programas y obtener información referente a su rendimiento, errores y comportamiento en general. En particular, *Pin* [23] es una herramienta que realiza instrumentación binaria en tiempo de ejecución, que fue desarrollada por *Intel Corporation* y la Universidad de Colorado.

Pin permite analizar el estado arquitectónico de un proceso tal como valores de registros, memoria y control de flujo. *Pin* funciona mediante *Pintools*, que son las rutinas de instrumentación que genera el usuario. El usuario puede

agregar rutinas a la aplicación que desea analalizar para obtener información de ella, existiendo dos tipos de rutinas: de instrumentación y de análisis. Las rutinas de instrumentación son las que se ejecutan al inicio y determinan donde se van a agregar las rutinas de análisis. Ambas rutinas están escritas en C/C++.

En cuanto a la arquitectura de *Pin*, como se muestra en la figura 4.1, está constituida por una máquina virtual, una caché de código y una API de instrumentación que es la utilizada por la *Pintool*. La máquina virtual está formada por un compilador JIT (del inglés justo a tiempo), un emulador y un despachador. El compilador JIT compila e instrumenta el código de la aplicación, que es lanzado por el despachador y almacenado en la caché de código. La interfaz *máquina virtual/caché de código* se encarga de almacenar y restaurar el estado de los registros utilizados por la aplicación. El emulador se encarga de interpretar instrucciones que no pueden ejecutarse directamente (llamadas de sistema), que requieren un manejo especial por parte de la máquina virtual.

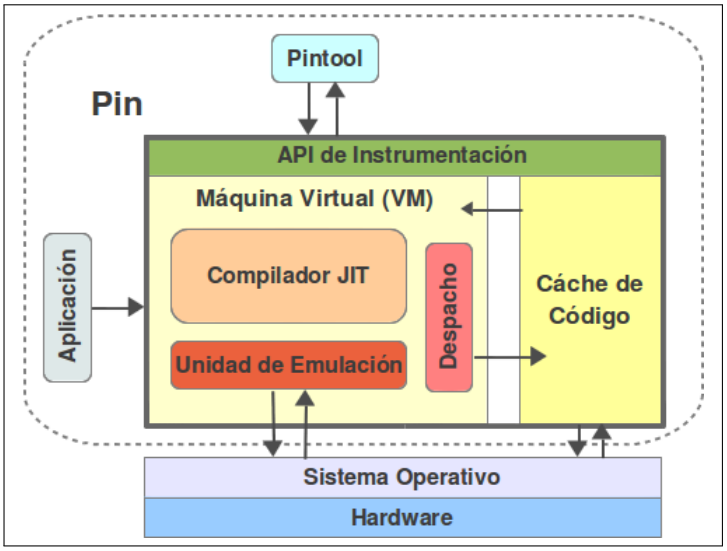


Figura 4.1: Arquitectura de *Pin* [23]

Se escogió el utilizar *Pin* por las siguientes razones:

- Permite un acceso total a cada componente de la aplicación

- Se tiene total control de la herramienta de análisis y dado que no es un simulador cerrado en donde solo se tiene acceso a cierta información, posee una capacidad de crecimiento ilimitada

- Rapidez en la ejecución, lo que permite analizar una aplicación completa y no solo porciones de ella (traza)

4.2. MultiCacheSim

Como simulador se requería uno que fuera lo suficientemente flexible para permitir realizar medidas de diferentes parámetros tales como: número y tipo de accesos, tasa de fallos, dirección y contenido de los datos. Asimismo, el simulador debe tener un buen rendimiento para poder realizar una ejecución en su totalidad de cada programa de prueba.

El *MultiCacheSim* [24] es un simulador desarrollado por Brandon Lucia en la universidad de Washington, basado en la memoria cache desarrollada en el SESC [25] (del inglés simulador SuperESCalar) desarrollado por el grupo I-acoma de la universidad de Illinois.

Este simulador se eligió principalmente por el rendimiento que provee, ya que los otros simuladores, SESC [25], CRC [26], Simics [27] al simular un sistema completo son más lentos.

Se utilizó la versión del simulador modificada en [17]. Se tuvo que modificar éste para agregar las diferentes políticas de reemplazamiento ya que solo contaba con las políticas LRU, RANDOM y peLIFO. Asimismo, se tuvo que desarrollar el manejo de la no exclusividad en el simulador.

La figura 4.2 muestra la arquitectura del simulador utilizado. *MultiCache-Sim* es un simulador escrito en C++, la clase MultiCacheSim sirve como contenedor de todos los niveles de caché que se definan, esta clase es la que se instancia en la *Pintool* y a la que *Pin* le hace llegar todas las instrucciones

de acceso a memoria. Cada uno de los niveles de memoria caché es de tipo *SimpleSMPCache*, que es la clase que utiliza *MultiCacheSim* para el manejo de la coherencia, *WriteBacks* y actualización de las políticas en cuanto a fallos o aciertos en los accesos. Cada una de las memorias definidas pertenecen a la clase *CacheGeneric*, que es la clase que maneja el funcionamiento de la caché y en donde se desarrollan las políticas de reemplazo.

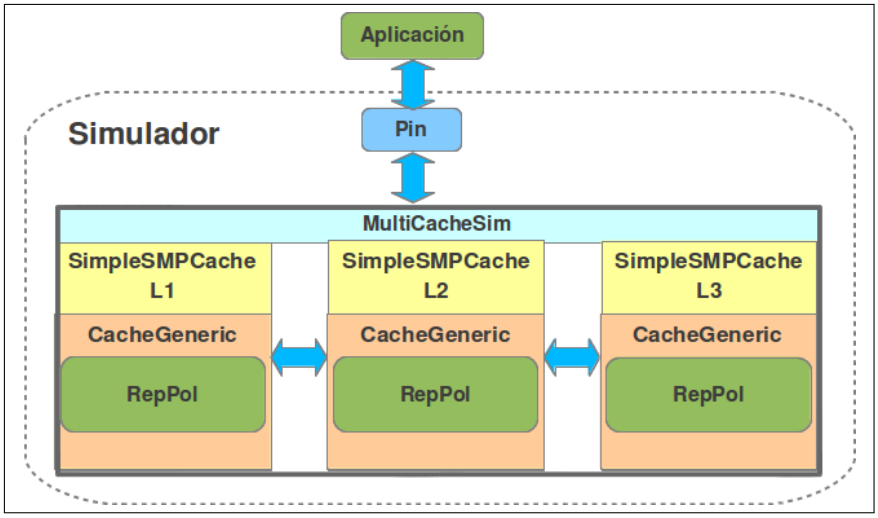


Figura 4.2: Arquitectura del simulador utilizado.

4.3. *Benchmarks*

Para evaluar la eficiencia de las políticas de reemplazo convencionales y propuestas se hizo uso de los SPEC2006 [28], que es la más reciente versión de las pruebas generadas por la SPEC (del inglés Corporación de evaluación de rendimiento normalizado). Estas aplicaciones estresan un sistema a nivel de procesador y memoria de forma que generan un resultado normalizado.

Los SPEC2006 están formados por 29 *benchmarks*, que se ejecutaron completamente para cada política de reemplazo usando las entradas de *train*. Las aplicaciones que provee el conjunto SPEC2006 se muestran en el cuadro 4.1.

400.perlbench	401.bzip2	403.gcc
410.bwaves	416.gamess	429.mcf
433.milc	434.zeusmp	435.gromacs
436.cactusADM	437.leslie3d	444.namd
445.gobmk	447.dealII	450.soplex
453.povray	454.calculix	456.hmmer
458.sjeng	459.GemsFDTD	462.libquantum
464.h264ref	465.tonto	470.lbm
471.omnetpp	473.astar	481.wrf
482.sphinx3	483.xalancbmk	

Cuadro 4.1: Pruebas del conjunto SPEC2006

4.4. Configuración

Para la evaluación de los distintos algoritmos se eligió una configuración de tres niveles de caché, siendo estos tres niveles no exclusivos, tomando como ejemplo la configuración del *Core i7* de Intel [29]. El tamaño de cada nivel de caché así como otros parámetros relevantes se muestra en el cuadro 4.2.

Nivel de caché	**Capacidad**
L1	32KB, 8 vías, 64 bytes/bloque
L2	256KB, 8 vías, 64 bytes/bloque
L3	1MB, 16 vías, 64 bytes/bloque

Cuadro 4.2: Configuración de memoria caché elegida para realizar la evaluación

En todos los casos los niveles L_1 y L_2 usan siempre la política LRU. Las políticas a evaluar se utilizan en el nivel L_3 también llamado, en este caso, LLC (del inglés caché de último nivel).

4.5. Consumo de energía y tiempo medio de acceso a memoria

Además de mostrar datos relativos a la tasa de fallos y al número de escrituras en memoria principal, en la sección 5 se mostrarán también datos del consumo de energía y del tiempo medio de acceso a memoria (TMAM) que conlleva cada uno de los algoritmos evaluados. A pesar de que el TMAM solo considera las latencias y no da cuenta del grado de solapamiento de los accesos a memoria, se acepta como medida de estimación de rendimiento. Aquí se utiliza debido a que no se cuenta con un simulador exacto a nivel de ciclos del procesador.

En cuanto al modelo de energía utilizado, para evaluar los consumos por acceso en los distintos niveles de cache se utiliza CACTI 6.5 [30], mientras que para determinar el consumo por accesos a memoria principal se sigue lo propuesto en [31], de acuerdo a lo cual el consumo por lectura en PCM es del orden de 1J/GB mientras que para una escritura es de 6J/GB.

Para realizar una estimación del rendimiento se utiliza como ya se ha mencionado el tiempo medio de acceso a memoria. Para las latencias de acceso a memoria caché se hace uso de las utilizadas en [13,14], en el caso de las latencias a memoria principal de nuevo se hace uso de lo propuesto en [31]. El cuadro 4.3 resume los valores de consumo de energía y latencias para cada nivel de la jerarquía de memoria.

Las ecuaciones 4.1 y 4.2 muestran los modelos utilizados para el consumo

Nivel de memoria	**Latencia (ciclos)**	**Consumo de energía (nJ) Lectura/Escritura/Consulta al *array* de etiquetas**
L1	1	0.222466/0.211604/0.00174426
L2	10	0.530476/0.542389/0.0055894
L3	30	2.25868/2.50327/0.019461
PCM lectura	200	59.6046447754
PCM escritura	4000	357.6278686523

Cuadro 4.3: Latencia y consumo de energía para la configuración de memoria utilizada

de energía y tiempo medio de acceso a memoria, respectivamente.

$$\begin{aligned}\text{Energía} = {} & RHL_1 * REL_1 + WHL_1 * WEL_1 + (RML_1 + WML_1) * (TEL_1 + WEL_1) + \\ & RHL_2 * REL_2 + WHL_2 * WEL_2 + (RML_2 + WML_2) * (TEL_2 + WEL_2) + \\ & RHL_3 * REL_3 + WHL_3 * WEL_3 + (RML_3 + WML_3) * (TEL_3 + WEL_3) + \\ & RPCM * REPCM + WPCM * WEPCM \end{aligned} \tag{4.1}$$

con:

RHL_x = Número de aciertos de lectura en L_x

WHL_x = Número de aciertos de escritura en L_x

RML_x = Número de fallos de lectura en L_x

WML_x = Número de fallos de escritura en L_x

RPCM = Número de lecturas a PCM

WPCM = Número de escrituras a PCM

REL_x = Consumo de energía de una lectura a L_x

WEL_x = Consumo de energía de una escritura a L_x

TEL_x = Consumo de energía de la consulta del *array* de etiquetas de L_x

REPCM = Consumo de energía de una lectura a PCM

WEPCM = Consumo de energía de una escritura a PCM

TMAM=

$$\frac{(RHL_1 + WHL_1) * LtL_1 + RHL_2 * LtL_2 + RHL_3 * LTL_3 + RPCM * LtRPCM}{AccL_1} \tag{4.2}$$

con:

RHL_x = Número de aciertos de lectura en L_x

WHL_x = Número de aciertos de escritura en L_x

RPCM = Número de lecturas a PCM

$AccL_1$ = Número de accesos a L_1

LtL_x = Latencia de acceso a L_x

LtRPCM = Latencia de acceso a PCM

Capítulo 5

Evaluación de Resultados

A continuación se presentan los resultados obtenidos en la evaluación de las políticas convencionales y las propuestas en este trabajo. Se muestran gráficas de las escrituras a memoria principal (PCM), tasa de fallos, consumo de energía y tiempo medio de acceso a memoria. En particular, las gráficas comparan la media aritmética de la normalización a LRU de las 29 aplicaciones de los SPEC2006.

5.1. Políticas Convencionales

Las políticas convencionales que se analizan a continuación son RANDOM, peLIFO, DRRIP, SHiP y CLP para *N-chance* igual a la asociatividad y para *N-chance* igual a la mitad de la asociatividad. Como se mencionó en la sección 4.4 se simula una jerarquía de memoria caché con tres niveles no exclusivos, se utiliza en los dos niveles más cercanos al procesador la política LRU y la política que se varía es la correspondiente al nivel más cercano a la memoria principal.

Al realizar una escritura retardada el acceso que se hace al nivel de caché para escribir el bloque no corresponde a un acceso propiamente dicho, esto se debe a que este acceso no es una referencia generada en ese momento por el procesador sino una consecuencia del mecanismo utilizado para mantener la coherencia de los datos. Por esta razón el bloque escrito no debería ser promo-

vido en la caché en que se escribe. Por ejemplo, al desalojar un bloque sucio de L_1, éste origina una escritura retardada en L_2, el bloque escrito no debería pasar a la posición MRU de la cola LRU de L_2 o, en otras palabras, no se deberían actualizar las estadísticas de la política ante un acceso por escritura retardada.

Por otra parte, dado que el promover un bloque hace que éste permanezca más tiempo en la caché, podría ser conveniente el actualizar las estadísticas de la política ante una escritura retardada. A continuación se presentan los resultados para ambos escenarios, es importante decir que la actualización de las estadísticas supone una gasto extra en el consumo de energía que aún no se ha evaluado.

5.1.1. Análisis de datos de jerarquía no exclusiva sin actualización de estadísticas al haber escritura retardada

Para los resultados presentados a continuación no se realiza la actualización de las estadísticas de la política cada vez que se da una escritura retardada, ya que como se explicó anteriormente, en principio una escritura retardada no corresponde a una referencia real a memoria.

En la figura 5.1 se muestra el número de escrituras a memoria principal normalizado a LRU para las políticas convencionales estudiadas. Como puede apreciarse, la política que más reduce el número de escrituras respecto a LRU es CLP con *N-chance* igual a la asociatividad, siendo esta reducción de un 17 %; en el caso de DRRIP la reducción alcanzada es del 10.7 % respecto a LRU. La tasa de fallos en L_3 y el TMAM se muestran en las figuras 5.2 y 5.3, en este caso la política DRRIP es la que obtiene el mejor resultado, con una tasa de fallos un 8.65 % menor a la de LRU y un TMAM 4.3 % menor al de LRU. Por último, la figura 5.4 muestra el consumo de energía en la jerarquía de memoria normalizado a LRU, siendo de nuevo DRRIP el que logra la mayor mejora

reduciendo el consumo de energía un 6.3 % respecto a LRU. Es importante resaltar que aunque CLP con *N-chance* igual a la asociatividad es la política que más reduce el número de escrituras respecto a LRU, esta política degrada significativamente el desempeño del sistema ya que incrementa notablemente la tasa de fallos y el TMAM respecto a LRU, concretamente la tasa de fallos en L_3 se incrementa un 45 %.

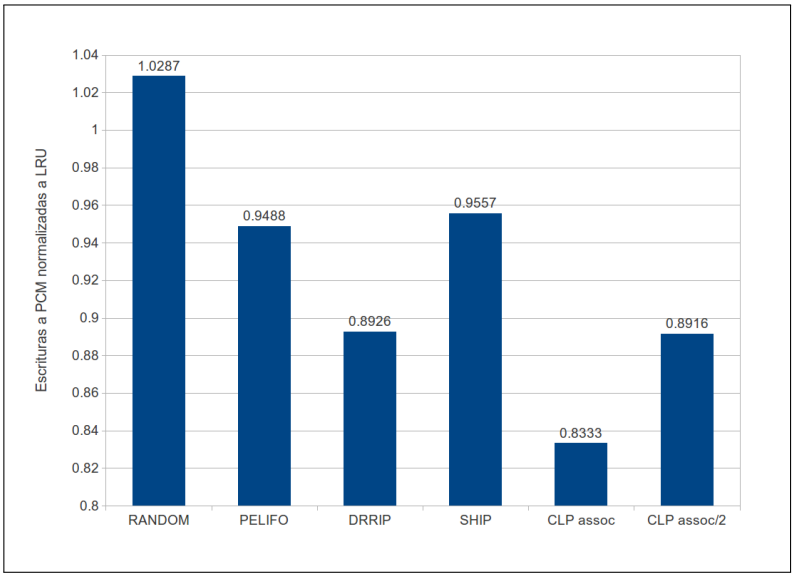


Figura 5.1: Escrituras normalizadas a LRU, sin actualización de estadísticas

5.1.2. Análisis de datos de jerarquía no exclusiva con actualización de estadísticas al haber escritura retardada

Para reducir el número de escrituras a memoria principal se plantea que al generarse una escritura retardada se actualicen las estadísticas del nivel en que se escribe, esto hace que los bloques que hayan sido modificados se mantengan más tiempo en la memoria caché y tengan más oportunidad de ser reutilizados por una nueva referencia a ellos. Además de las políticas convencionales mencionadas en la sección 5.1 se agrega a las gráficas la política LRU_NR que

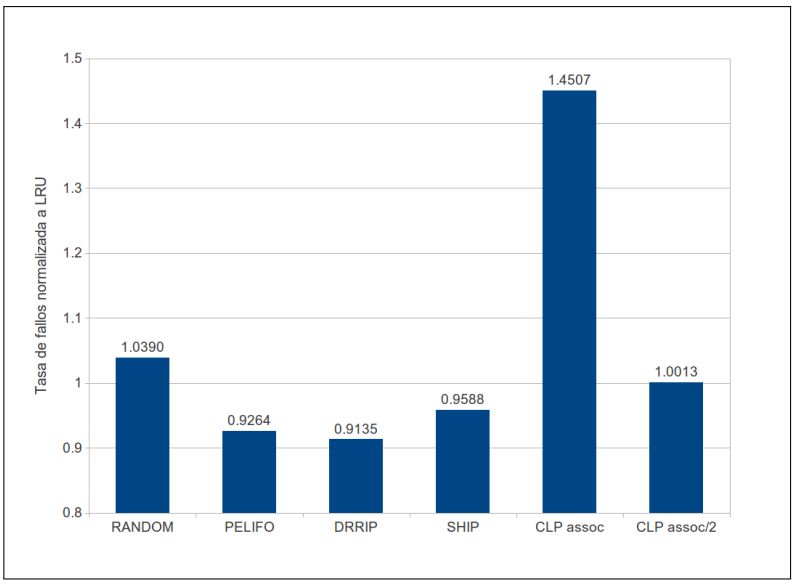


Figura 5.2: Tasa de fallos en L_3 normalizada a LRU, sin actualización de estadísticas

corresponde a la política LRU de la sección 5.1.1 (en donde no se actualizan las estadísticas).

Las figuras 5.5, 5.6, 5.7 y 5.8 muestran el número de escrituras, tasa de fallos en L_3, tiempo medio de acceso a memoria y consumo de energía, respectivamente, para el caso en donde se actualizan las estadísticas de la política después de una escritura retardada. Es importante notar que en todos los casos la política LRU_NR presenta peores resultados respecto a la política LRU actualizando estadísticas.

Como puede apreciarse de la figura 5.5, la política que más reduce el número de escrituras a PCM respecto a LRU es CLP con *N-chance* igual a la asociatividad, siendo esta reducción de un 15.9 %. En el caso de DRRIP la reducción alcanzada es del 12.7 % respecto a LRU. En las figuras 5.6 y 5.7 puede apreciarse como la política DRRIP es la que obtiene el mejor resultado en términos de tasa de fallos y TMAM, siendo estos un 7.4 % y un 3.3 % menor respecto a LRU. Por último de la figura 5.8 puede concluirse de nuevo que DRRIP logra

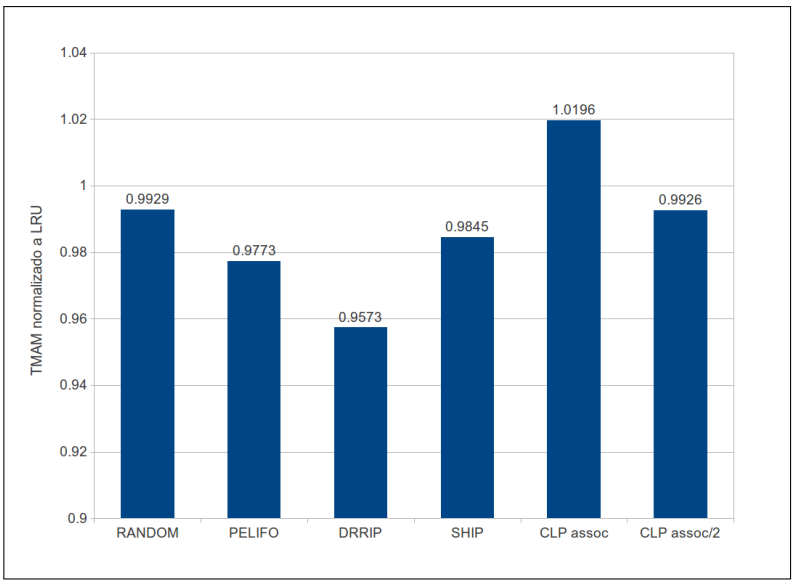


Figura 5.3: TMAM normalizado a LRU, sin actualización de estadísticas

la mayor mejora reduciendo el consumo de energía un 8.2 % respecto a LRU. Es importante resaltar que, al igual que en el caso en donde no se actualizan las estadísticas, CLP con *N-chance* igual a la asociatividad es la política que más reduce el número de escrituras respecto a LRU. Sin embargo, es la política que más degrada el desempeño del sistema ya que es la que más aumenta tasa de fallos y TMAM respecto a LRU.

Observamos que en la reducción del número de escrituras a PCM se logra una mejora en el caso donde se actualizan las estadísticas respecto al que no lo hace, por lo que en el resto del trabajo, para el análisis de las nuevas propuestas de reducción de escrituras a memoria principal, se hará la actualización de las estadísticas después de una escritura retardada.

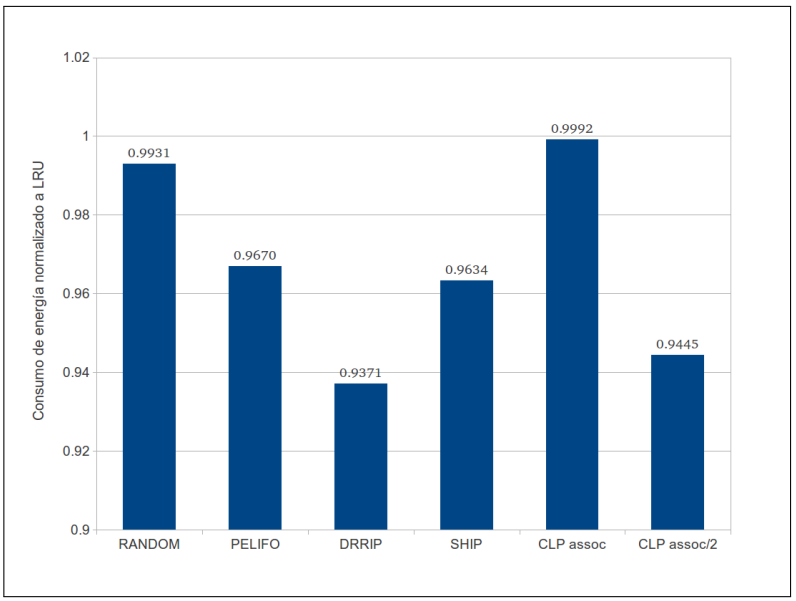


Figura 5.4: Consumo de energía normalizada a LRU, sin actualización de estadísticas

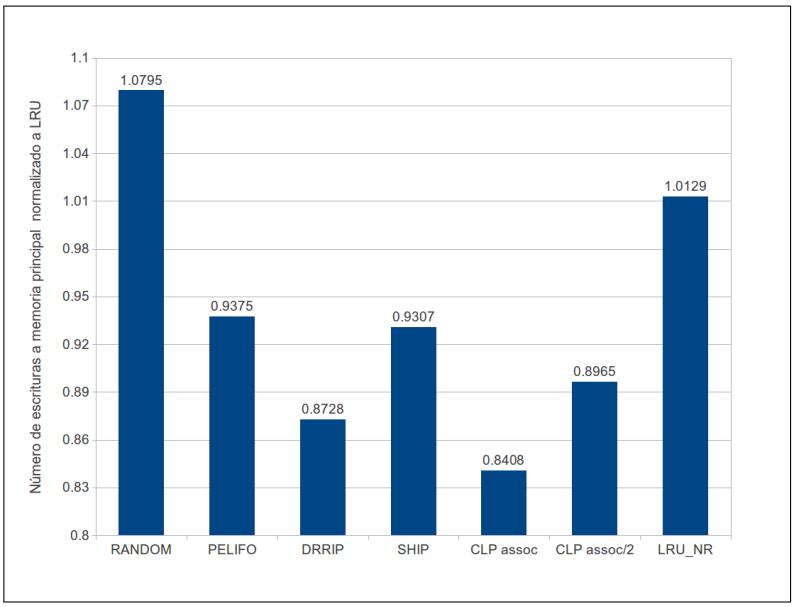


Figura 5.5: Escrituras normalizadas a LRU, con actualización de estadísticas

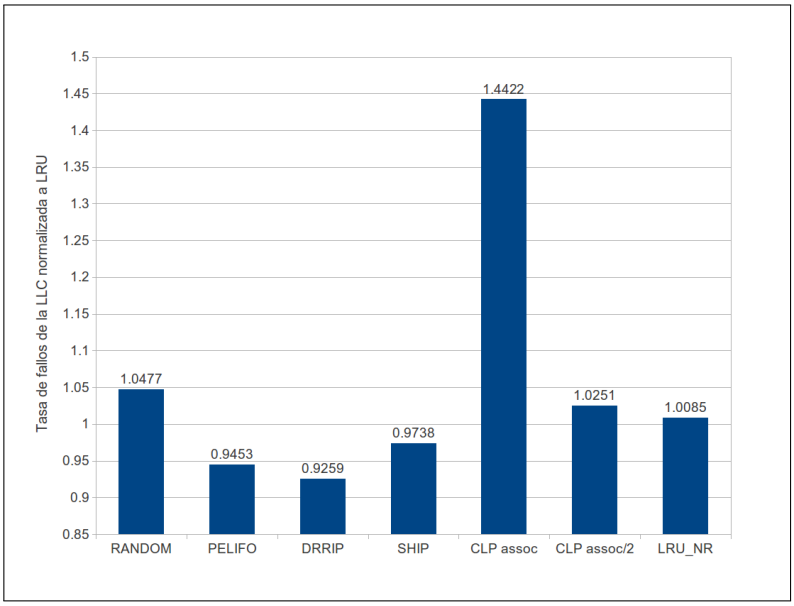


Figura 5.6: Tasa de fallos en L_3 normalizada a LRU, con actualización de estadísticas

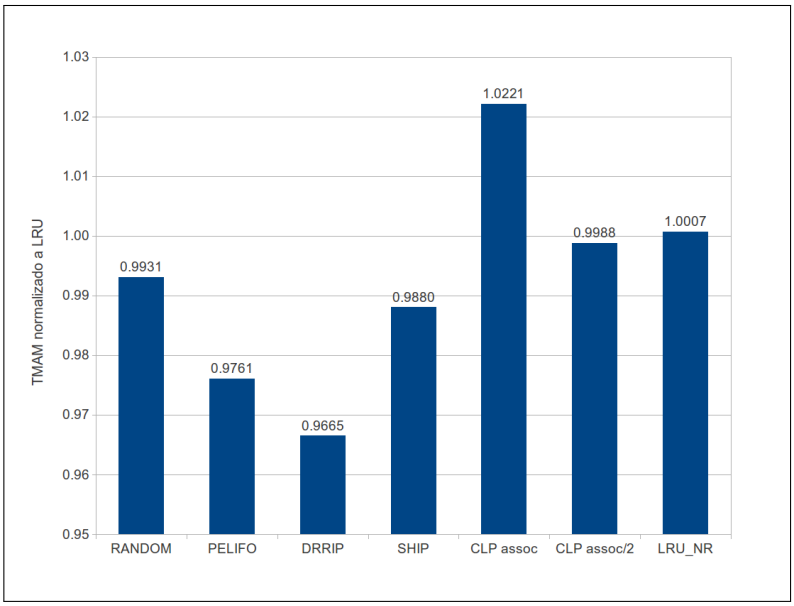


Figura 5.7: TMAM normalizado a LRU, con actualización de estadísticas

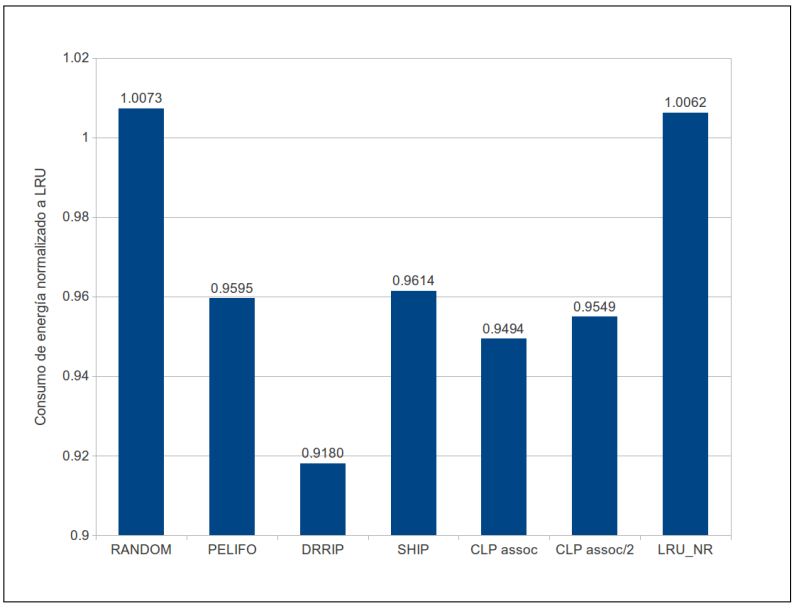


Figura 5.8: Consumo de energía normalizada a LRU, con actualización de estadísticas

5.2. Políticas Propuestas

A continuación se presentan los resultados derivados de la evaluación de las políticas propuestas para la reducción de escrituras a memoria principal. Estos algoritmos se dividen en los tres tipos de políticas mencionadas en la sección 3, basadas en LRU, basadas en DRRIP y basadas en peLIFO.

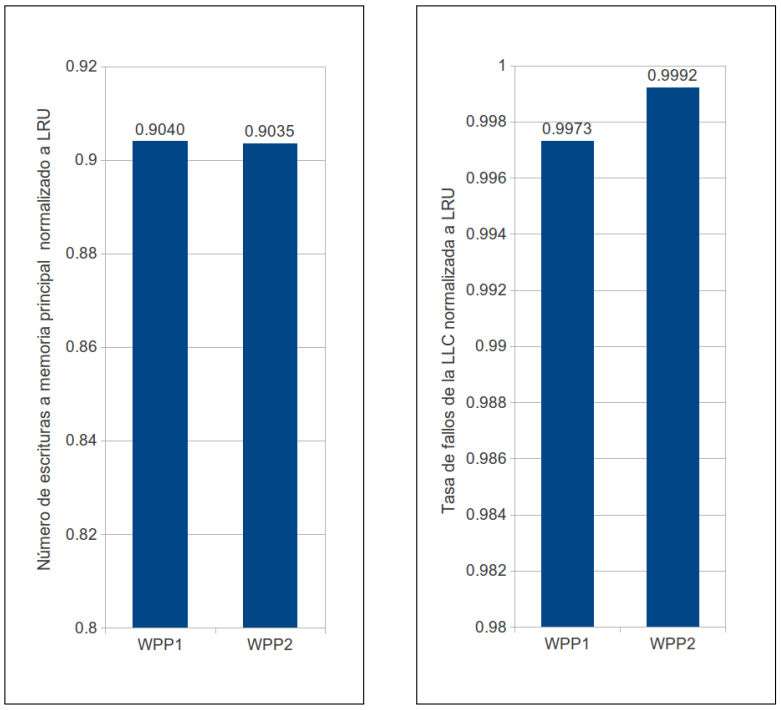
5.2.1. Políticas basadas en LRU

Como se explicó en la sección 3.1 se proponen dos políticas basadas en LRU, WPP1 y WPP2. En la figura 5.9(a) se puede apreciar como ambas políticas son capaces de reducir el número de escrituras a memoria principal respecto a LRU, siendo en ambos casos la reducción cercana al 10 %. Sin embargo, como se puede apreciar en la figura 5.9(b), WPP1 presenta una tasa de fallos ligeramente inferior a la de WPP2. En lo que respecta a TMAM (figura 5.10(a)) y consumo de energía (figura 5.10(b)) ambas políticas presentan resultados muy similares y en ambos casos mejores que LRU.

Aunque ninguna de las políticas propuestas basadas en LRU presenta el nivel de mejora en reducción de número de escrituras a memoria principal que alcanzan DRRIP o CLP con *N-chance* igual a la asociatividad, ambas propuestas logran superar a las demás políticas convencionales. Cabe resaltar que en el caso de WPP1 la propuesta es muy simple, ya que como se menciona en la sección 3.1.1, solo se modifica la inserción de los bloques, mientras que la promoción de los bloques sigue funcionando como en LRU convencional.

5.2.2. Políticas basadas en DRRIP

Antes de analizar los resultados de las políticas basadas en DRRIP, vale la pena analizar el comportamiento de las seis combinaciones de SRRIP en cuanto a reducción del número de escrituras. Estas seis combinaciones corresponden a

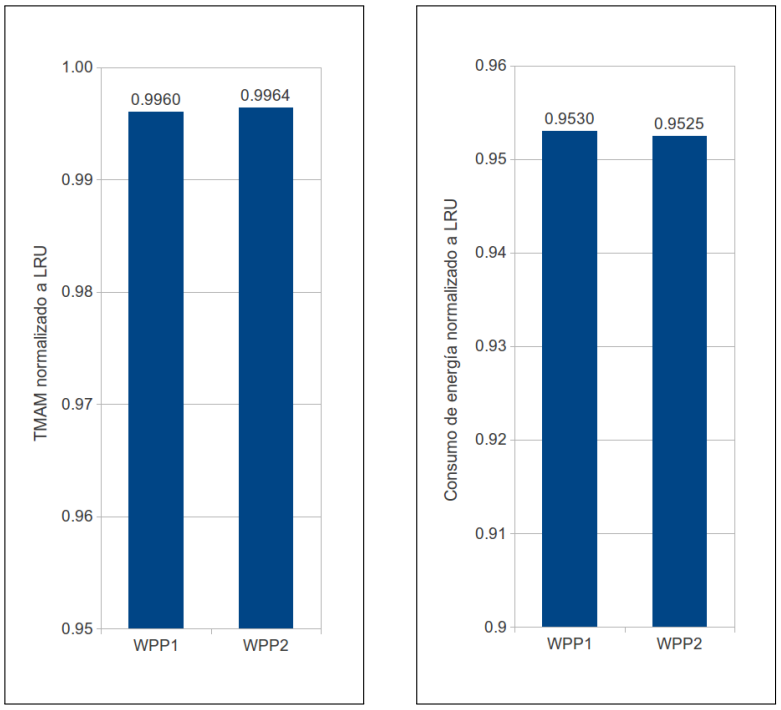


(a) (b)

Figura 5.9: Escrituras normalizadas a LRU (a). Tasa de fallos normalizada a LRU (b).

RRIP-HP y RRIP-FP ambos con inserciones como acceso *lejano* (L), *cercano* (C) y *distante* (D). De aquí que las seis combinaciones se denotarán como: RRIP-HP-D, RRIP-HP-L, RRIP-HP-C, RRIP-FP-D, RRIP-FP-L y RRIP-FP-C.

El algortimo DRRIP original, como se detalla en [13], está formada por dos políticas, una que permita a los bloques que van a ser reutilizados durante un largo periodo de tiempo permanecer en la caché hasta que se haga referencia a ellos de nuevo y otra que no permita que se contamine la caché con bloques a los que solo se va a hacer referencia una vez. Para hacer esto se utiliza una política que inserte los bloques como acceso *lejano*, y otra que lo haga como acceso *distante* y se elige dinámicamente la política a usar mediante la técnica de *set-dueling*.



(a) (b)

Figura 5.10: TMAM normalizado a LRU (a). Consumo de energía normalizado a LRU (b).

La figura 5.11 muestra el comportamiento de las variantes de SRRIP en cuanto a número de escrituras a memoria principal. Como se estableció en la sección 3.2.6, en DRRIPW6 se propone cambiar las políticas que utiliza DRRIP original; basándose en los resultados de la figura 5.11 se mantiene RRIP-HP con inserción *lejana* como una de las políticas y dado que presenta una ligera mejora se elige utilizar RRIP-FP inserción *distante* como parte de la política BRRIP. Es importante resaltar que los resultados para RRIP-FP-D y RRIP-HP-D son exageradamente desfavorables debido al comportamiento de una de las aplicaciones (462.libquantum); al eliminar esta aplicación se obtiene una media aritmética de 1.0142 y 1.0684 respectivamente. De ahí que RRIP-FP-D sea una buena elección para utilizarla en DRRIPW6.

La figura 5.12 muestra los resultados relativos al número de escrituras

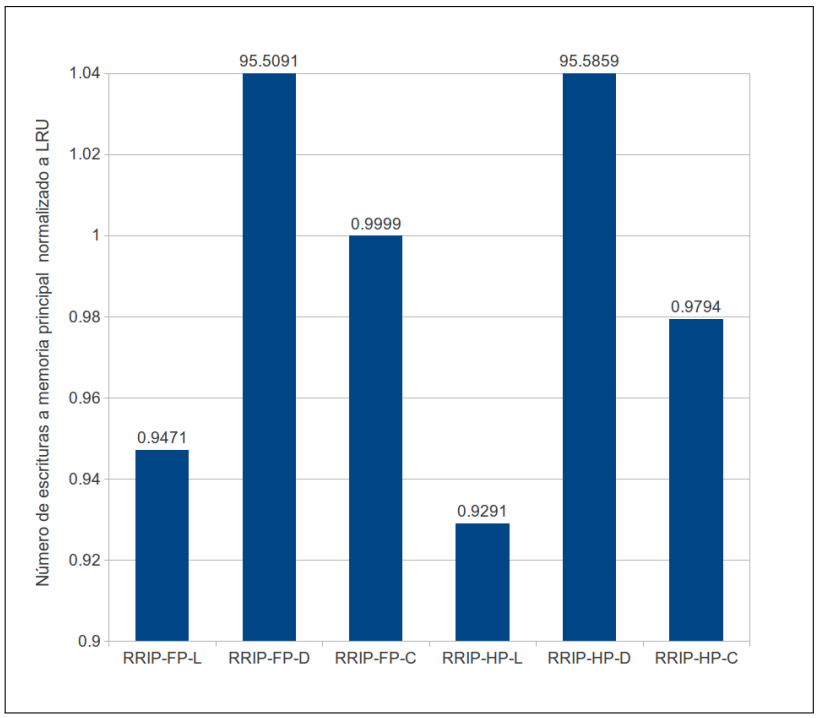


Figura 5.11: Escrituras normalizadas a LRU para las versiones de SRRIP

a memoria principal de las políticas basadas en DRRIP presentadas en la sección 3.2. Como puede observarse, cambiar la promoción de los aciertos (DRRIPW1) da un mejor resultado que modificar la inserción de los nuevos bloques (DRRIPW2), mientras que al combinar ambas políticas (DRRIPW3) los resultados no mejoran. De estas tres políticas solo DRRIPW1 presenta una mejora respecto a DRRIP original. En el caso de DRRIPW4 puede verse que al modificar el parámetro del *set-dueling* que determina la política a utilizar se logra una gran mejora, prueba de ello es que DRRIPW4 logra reducir las escrituras a memoria principal casi al nivel de CLP con *N-chance* igual a la asociatividad. El aumentar el número de bits del RRPV (DRRIPW5) no muestra una mejora sobre DRRIP. El cambiar las políticas que se debaten en el *set-dueling* (DRRIPW6) logra mejorar ligeramente el número de escrituras a memoria principal respecto a DRRIP. Al combinar DRRIPW4 y DRRIPW6

(DRRIPW7) se logra una gran reducción en el número de escrituras a memoria principal, reducción que se incrementa en DRRIPW9 al cambiar la promoción de los aciertos de la política fija de DRRIPW7. La máxima reducción de escrituras a memoria principal respecto a LRU la presenta la combinación de DRRIPW4 y DRRIPW1 (DRRIPW8), seguida de DRRIPW9 y por último DRRIPW7, estas reducciones son de un 16.10 %, 16.06 % y 16 % respectivamente.

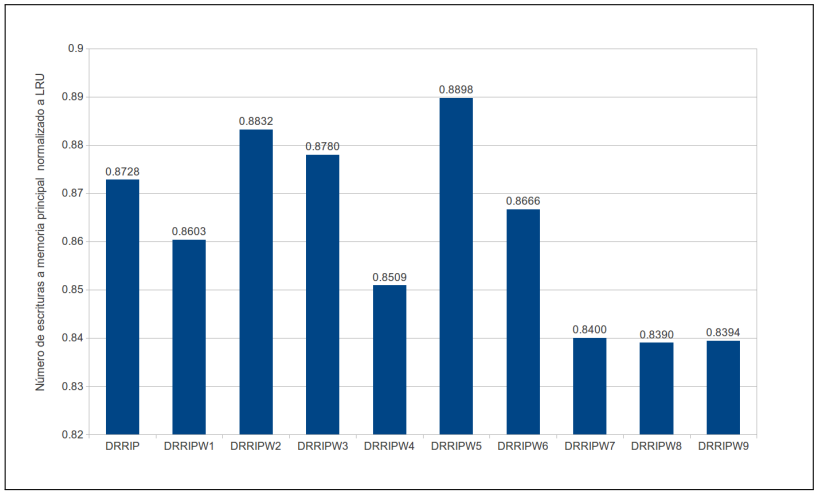


Figura 5.12: Escrituras normalizadas a LRU para las políticas basadas en DRRIP

Dado que los mejores resultados de las políticas propuestas basadas en DRRIP, en lo que respecta a reducción de escrituras a memoria principal, las presentan DRRIPW7, DRRIPW8 y DRRIPW9, el resto del análisis se basará en estas tres políticas.

La figura 5.13 muestra la tasa de fallos en L_3 para las políticas basadas en DRRIP, en este caso ninguna de las políticas logra una tasa de fallo menor a la original. En el caso de DRRIPW7 el incremento respecto a DRRIP es de solo 0.2 %, mientras que DRRIPW8 incrementa un 3.78 % y DRRIPW9 un 1.65 %. En los tres casos la tasa de fallos es menor a la de LRU y de las polí-

ticas clásicas en el caso de DRRIPW7 solo DRRIP presenta un mejor resultado.

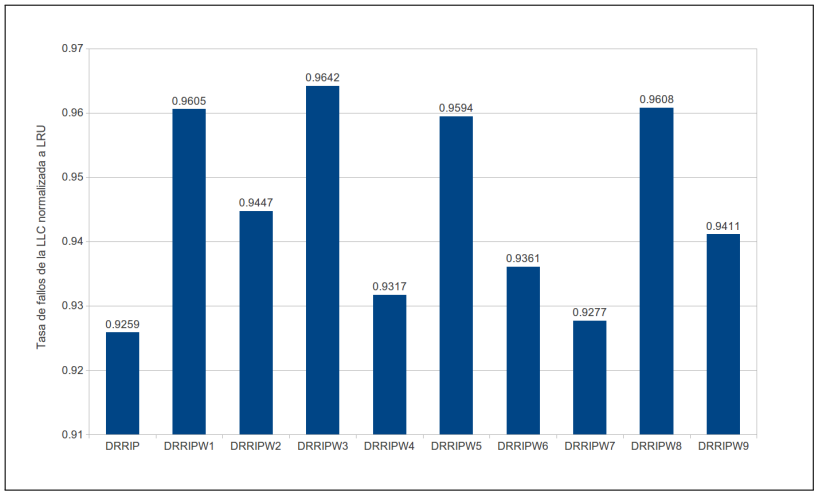


Figura 5.13: Tasa de fallos en L_3 normalizada a LRU para las políticas basadas en DRRIP

Los resultados para el TMAM se muestran en la figura 5.14. Para DRRIPW7, DRRIPW8 y DRRIPW9 el TMAM se incrementa respecto a DRRIP, siendo este incremento de 0.23 %, 0.26 % y 0.28 % respectivamente. En los tres casos el TMAM se mantiene menor que el de LRU y que el de todas las políticas convencionales excepto DRRIP original.

La figura 5.15 presenta los resultados para el consumo de energía, en donde DRRIPW7, DRRIPW8 y DRRIPW9 muestran una reducción del mismo de un 1.72 %, 1.84 % y 1.80 % respecto a DRRIP original que es la política convencional que muestra el menor consumo de energía.

De las tres propuestas que presentan los mejores resultados, DRRIPW7 es la óptima a implementar ya que, además del parámetro considerado para determinar mediante el *set-dueling* la política a utilizar, incluye un único cambio adicional respecto a DRRIP original. Este cambio supone variar la política

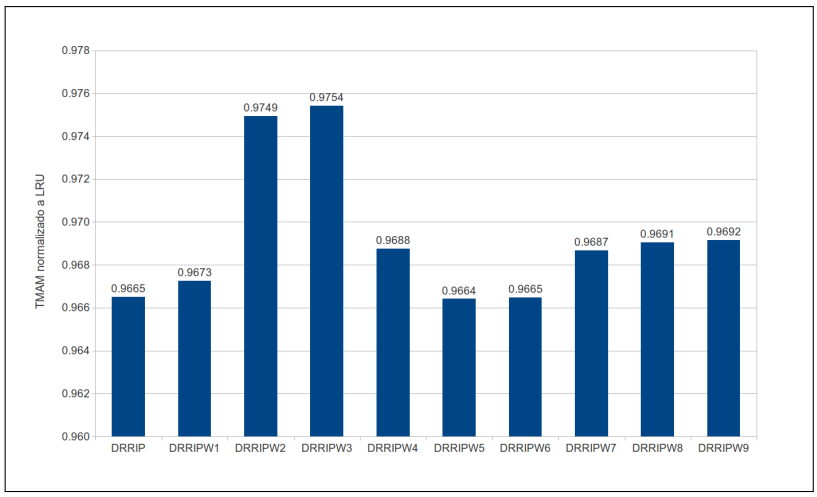


Figura 5.14: TMAM normalizado a LRU para las políticas basadas en DRRIP

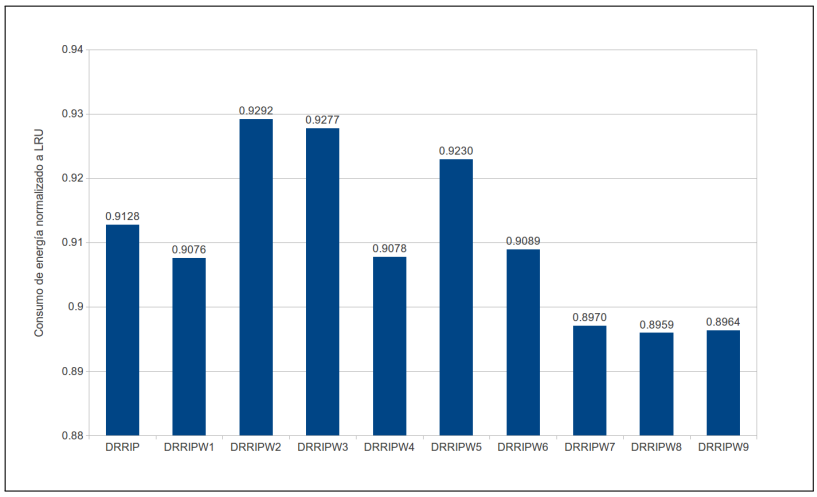


Figura 5.15: Consumo de energía normalizada a LRU para las políticas basadas en DRRIP

utilizada por BRRIP cuando se da un acierto en un bloque (en lugar de cambiar el valor del RRPV a cero se decrementa su valor en uno). De aquí que no sería complejo la implementación de esta política.

5.2.3. Políticas basadas en peLIFO

A pesar de que peLIFO no muestra un comportamiento espectacular en los resultados mostrados en la sección 5.1.2, se decidió plantear modificaciones a esta política dada la referencia que se hace de la misma en la literatura y la experiencia que se tiene de esta política en el grupo de trabajo. Las modificaciones propuestas para peLIFO se detallaron en la sección 3.3.

La figura 5.16 muestra los resultados relativos al número de escrituras a memoria principal. De acuerdo con esta figura peLIFO1, peLIFO3, peLIFO4, peLIFO5 y peLIFO6 logran reducir el número de escrituras a memoria principal respecto a LRU más que peLIFO original, siendo esta reducción de un 6.7 %, 8.05 %, 11.57 %, 13.04 % y 13.85 % respectivamente.

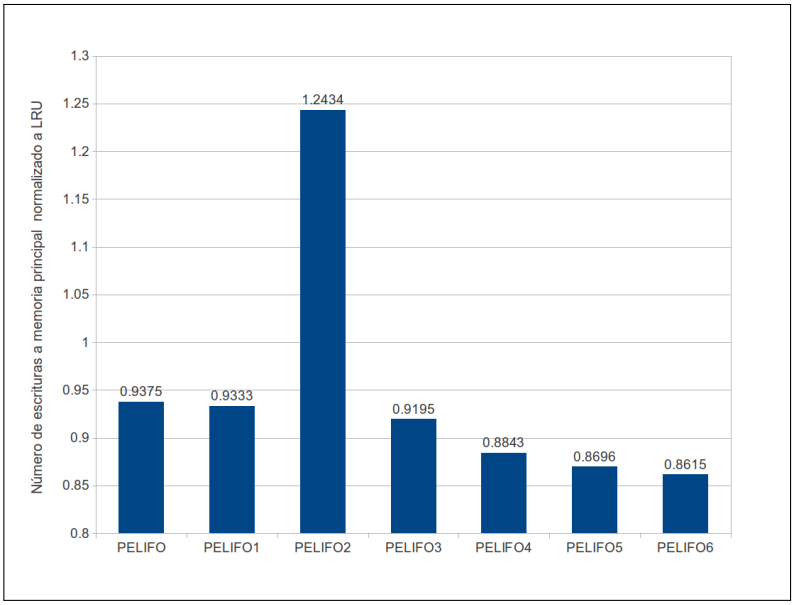


Figura 5.16: Escrituras normalizadas a LRU para las políticas basadas en peLIFO

La tasa de fallos en L_3 se muestra en la figura 5.17, en donde se observa que solo peLIFO6 mejora a peLIFO original reduciendo la tasa de fallos un 6.17 % respecto a LRU. De la figura 5.18 se tiene que peLIFO1, peLIFO2,

peLIFO3 y peLIFO6 logran superar a peLIFO original en lo que se refiere al TMAM, reduciéndolo respecto a LRU en un 2.67 %, 0.47 %, 1.28 % y 2.53 % respectivamente. Por último, como se detalla en la figura 5.19, en lo que respecta al consumo de energía se obtiene una reducción respecto a LRU de 4.49 %, 4.18 %, 5.07 %, 5.53 % y 7.7 % para peLIFO1, peLIFO3, peLIFO4, peLIFO5 y peLIFO6 respectivamente, mejorando a peLIFO original en todos los casos.

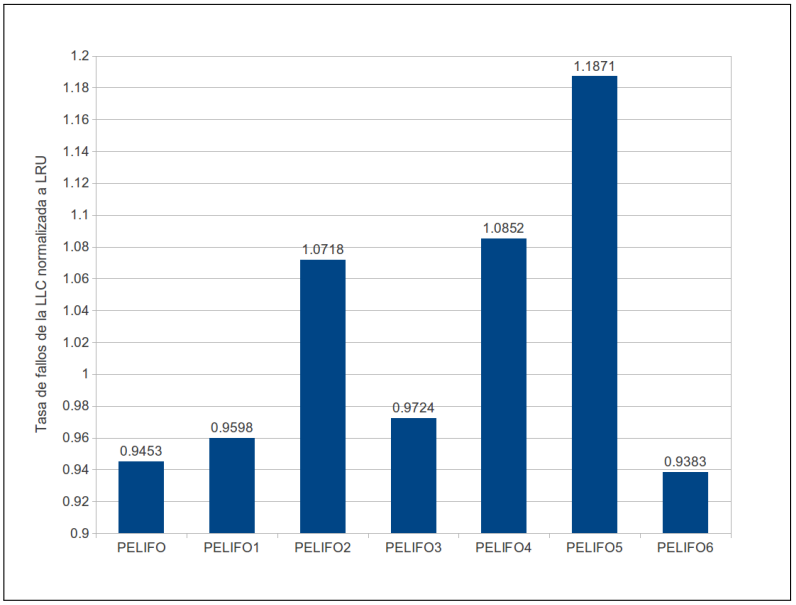


Figura 5.17: Tasa de fallos en L_3 normalizada a LRU para las políticas basadas en peLIFO

De los resultados obtenidos se concluye que peLIFO6 se presenta como una mejora atractiva respecto a peLIFO original ya que no solo reduce el número de escrituras a memoria principal sino que también reduce la tasa de fallos, TMAM y consumo de energía. Se recuerda que la diferencia con peLIFO original radica en contar los reemplazos del LLC que originan una escritura retardada a memoria principal, en lugar de los aciertos, para determinar mediante el *set-dueling* la mejor de las políticas a utilizar.

Hasta este punto se han mostrado resultados para la media aritmética de las

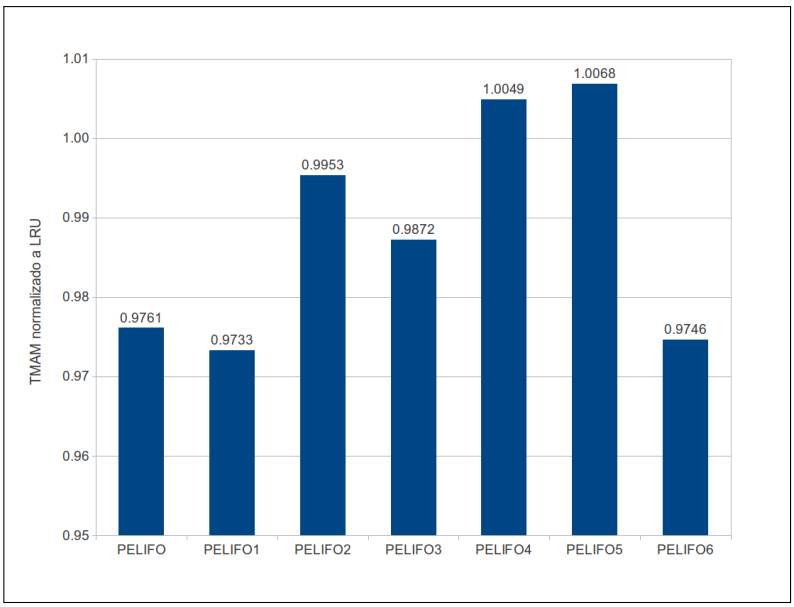


Figura 5.18: TMAM normalizado a LRU para las políticas basadas en peLIFO

normalizaciones a LRU; sin embargo, a continuación se va a ilustrar, para un análisis más completo, el comportamiento individual por aplicación. Específicamente se presentan los resultados relativos a las variables evaluadas (número de escrituras en PCM, tasa de fallos, consumo de energía y TMAM) para las 29 aplicaciones utilizadas en el análisis de las políticas de reemplazo, en particular se presentan los resultados para las políticas CLP con *N-chance* igual a la asociatividad, DRRIP, peLIFO, DRRIPW7, DRRIPW8, DRRIPW9 y peLIFO6.

En la figura 5.20 se muestran los resultados para el número de escrituras normalizado a LRU para las 29 aplicaciones utilizadas. En particular puede apreciarse como la aplicación *482.sphinx3* es para la que se logra la mayor reducción, llegando a alcanzarse un valor respecto a LRU del 97.3 % por parte de la política CLP con *N-chance* igual a la asociatividad y de un 92.1 % por parte de peLIFO6. Por contra en el caso de la aplicación *462.libquantum*, ninguna de las políticas propuestas es capaz de reducir el número de escrituras respecto a LRU, siendo las políticas basadas en DRRIP las que presentan los peores

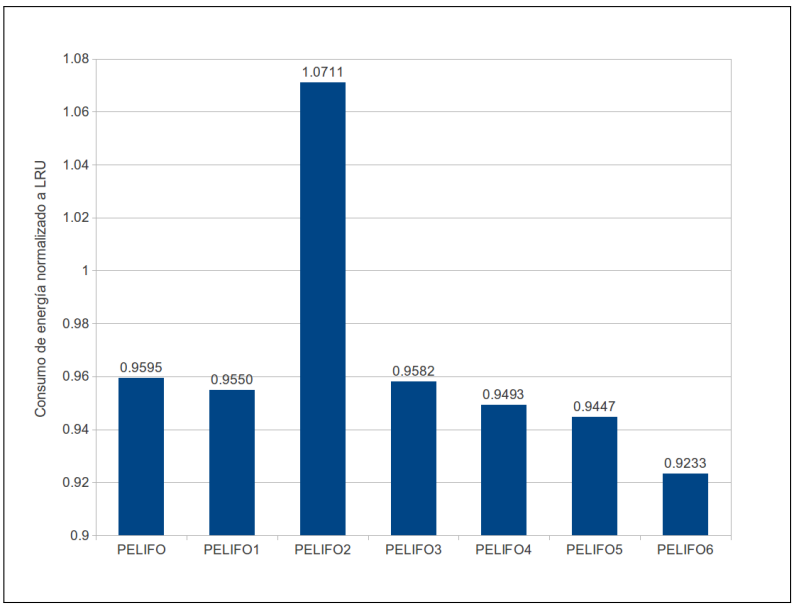


Figura 5.19: Consumo de energía normalizada a LRU para las políticas basadas en peLIFO

resultados. Ésta es la única aplicación en donde las políticas basadas en DRRIP no reducen el número de escrituras respecto a LRU. Para peLIFO6 se tiene que, con excepción de 5 aplicaciones, esta política reduce el número de escrituras en PCM respecto a LRU, manteniéndose el incremento para dichas aplicaciones por debajo de un 3 % respecto a LRU.

La figura 5.21 muestra la tasa de fallos en L_3 de cada aplicación normalizada a LRU. En el caso de CLP con *N-chance* igual a la asociatividad en 18 de las 29 aplicaciones posee una tasa de fallos superior a LRU, llegando a ser este incremento de hasta el 655 % y 461 % en el caso de las aplicaciones *416.gamess* y *453.povray* respectivamente. Para las políticas basadas en DRRIP, en 11 de las 29 aplicaciones la tasa de fallos es mayor a LRU, aunque esta diferencia se mantiene por debajo de un 8 % excepto para la aplicación *462.libquantum* para la cual la tasa de fallos es un 19 % mayor que la de LRU. En el caso de peLIFO6, al igual que para DRRIP, esta política aumenta la tasa de fallos respecto a LRU

en 11 aplicaciones, manteniendo el aumento por debajo de un 3.5 % excepto en dos aplicaciones en donde el incremento es del 6.2 % y 26.8 %, estas aplicaciones son *434.zeusmp* y *465.tonto* respectivamente.

La figura 5.22 muestra el TMAM normalizado a LRU para las 29 aplicaciones, de nuevo es CLP con *N-chance* igual a la asociatividad la que obtiene los peores resultados, llegando a tener un TMAM superior hasta en un 30 % al de LRU como ocurre en la aplicación *401.bzip2*. En el caso de las aplicaciones basadas en DRRIP, en 11 aplicaciones el TMAM es mayor que el de LRU, pero en todos los casos este incremento es menor a un 1 % excepto para *465.tonto* en donde alcanza el 3 % sobre LRU. En las aplicaciones en donde el TMAM es menor que LRU, éste llega a decrementarse hasta en un 14 %. Para peLIFO6 se tiene que en 10 aplicaciones posee un TMAM superior a LRU, manteniéndose este incremento en todos los casos por debajo del 1 % excepto para *434.zeusmp* en donde el incremento es del 4 % por encima de LRU. Para las aplicaciones en donde el TMAM es menor que LRU esta diferencia llega a ser hasta del 4 %.

La figura 5.23 muestra el consumo de energía normalizado a LRU para todas las aplicaciones consideradas. Como se observa el consumo de energía se mantiene bastante uniforme, siendo pocas las políticas que generan en algún caso un consumo de energía mayor que LRU. En particular destacan CLP con *N-chance* igual a la asociatividad para las aplicaciones *483.xalancbmk* y *445.gobmk* y peLIFO para *481.wrf* en donde el consumo de energía es superior a LRU en un 3.1 %, 3.3 % y 1.9 % respectivamente. Las políticas basadas en DRRIP conllevan un consumo mayor que LRU en cinco aplicaciones, manteniéndose este incremento por debajo del 0.5 % en todos los casos, mientras que por otra parte logran reducir el consumo de energía hasta en un 35 % en alguna aplicación. Para peLIFO6 el incremento en el consumo de energía por encima de LRU se da para 7 aplicaciones, pero en todos los casos este incremento se mantiene por

debajo del 1 %, llegando esta política a reducir el consumo por debajo de LRU hasta en un 30 %.

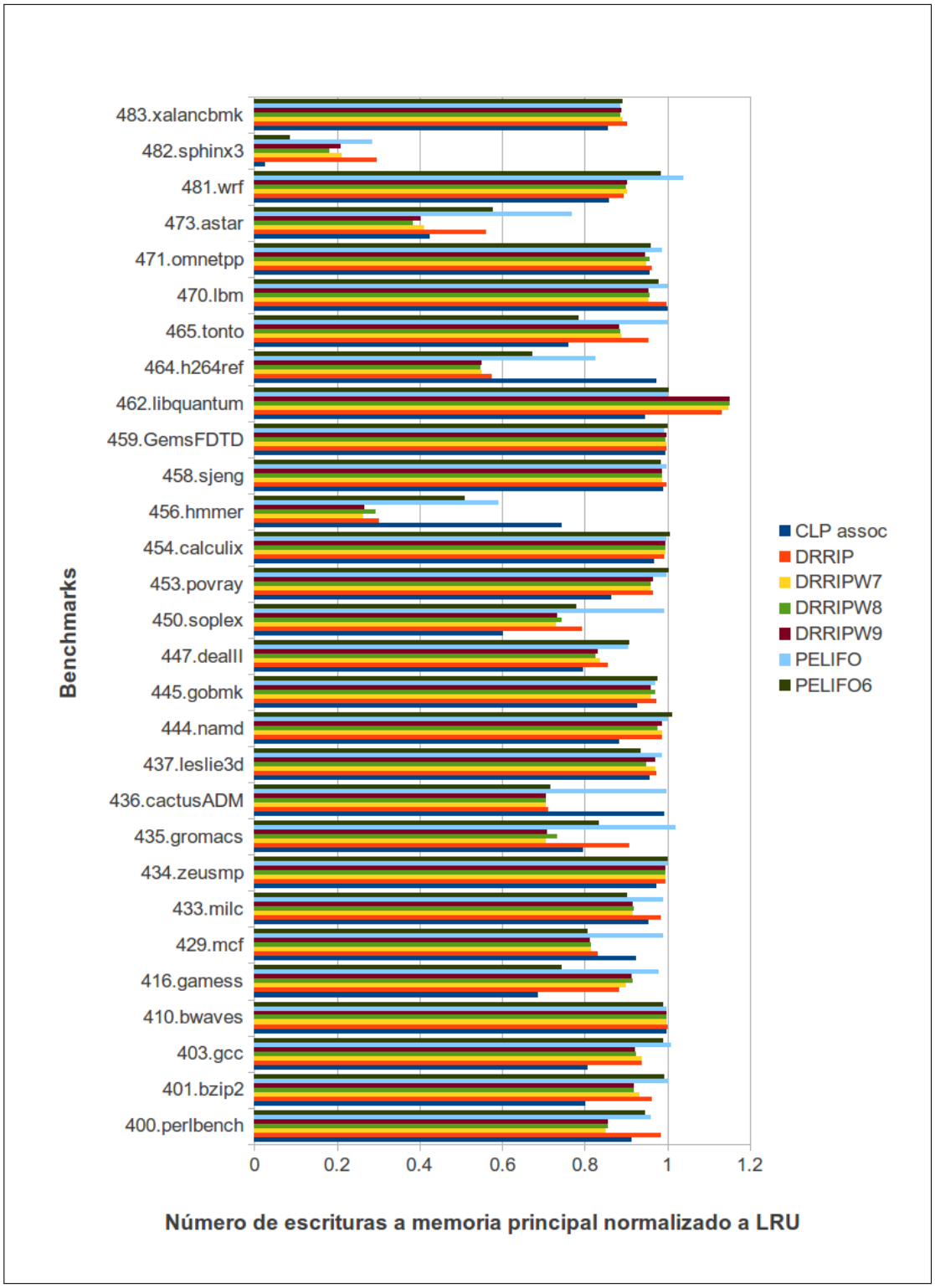


Figura 5.20: Escrituras normalizadas a LRU

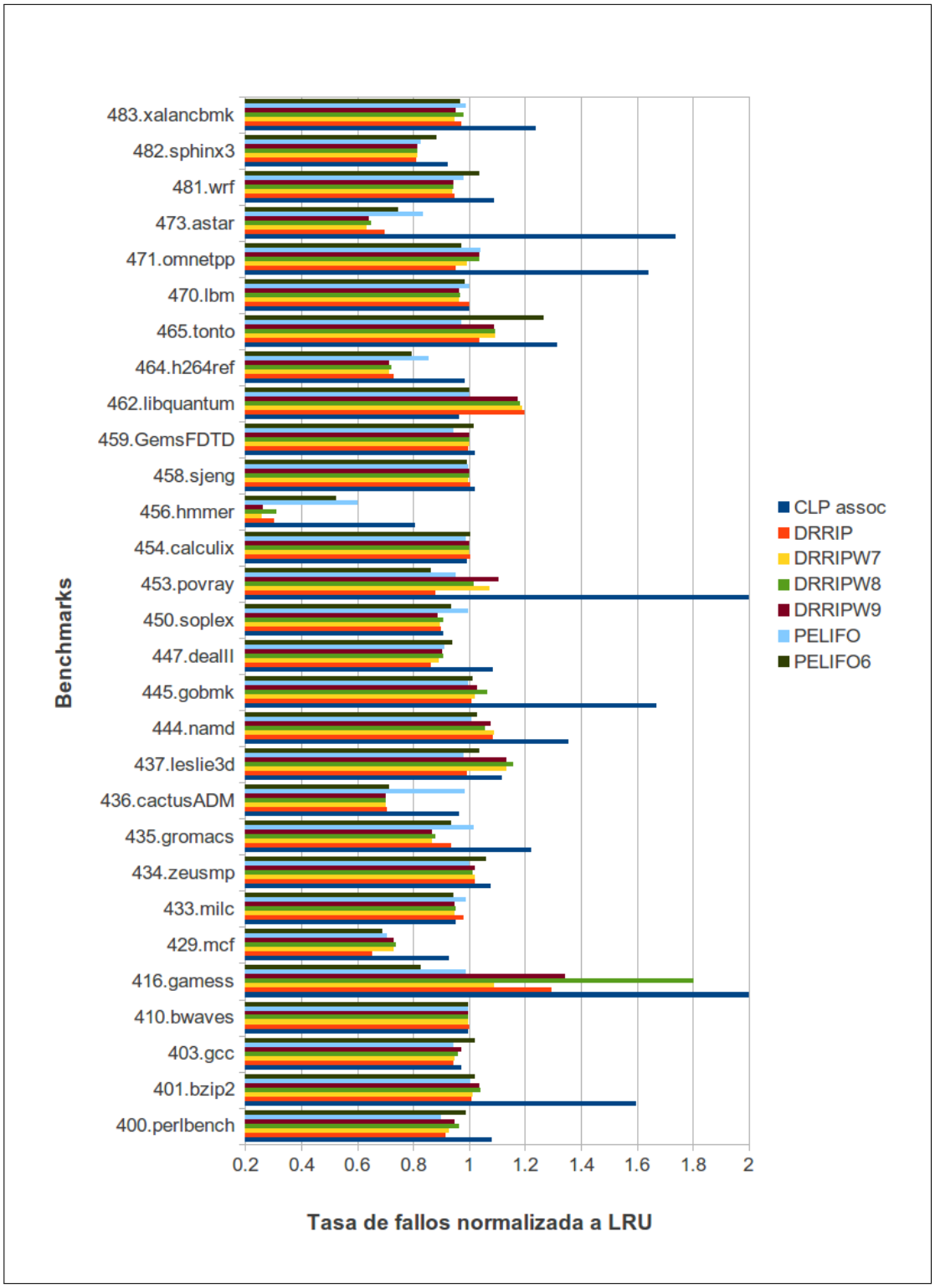


Figura 5.21: Tasa de fallos en L_3 normalizada a LRU

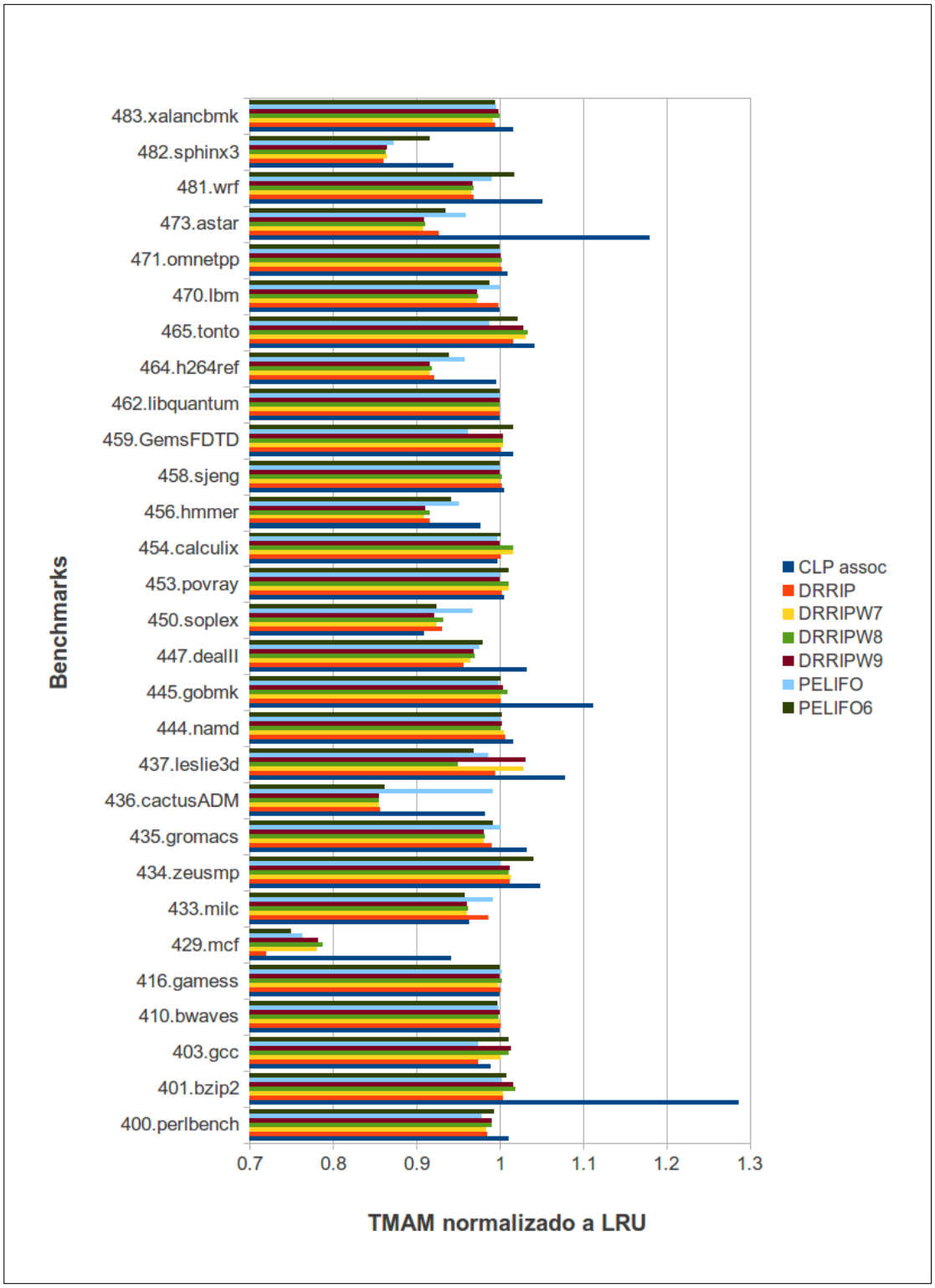


Figura 5.22: TMAM normalizado a LRU

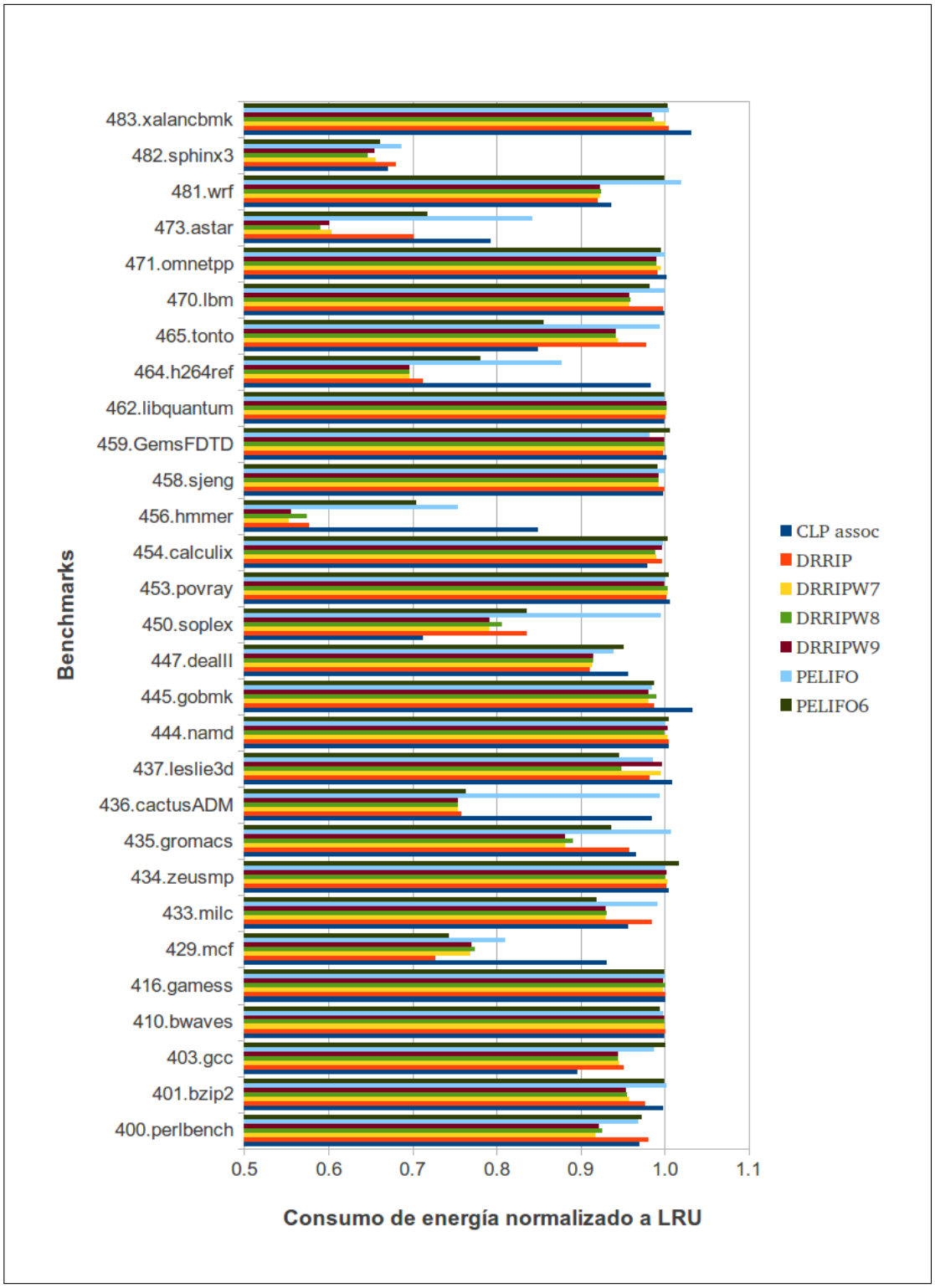


Figura 5.23: Consumo de energía normalizada a LRU

Capítulo 6

Conclusiones y Trabajo Futuro

Debido a varios factores como el consumo de energía y la escalabilidad, la sustitución de la memoria principal basada en tecnología DRAM por otra tecnología que subsane estos inconvenientes se hace inminente. PCM es la tecnología más prometedora para convertirse en la sustituta de DRAM. PCM presenta como gran desventaja una vida útil, medida en número de ciclos de escritura, mucho menor que DRAM. Para extender la vida útil de una memoria principal basada en PCM, en este trabajo se han planteado propuestas orientadas a reducir el número de escrituras a PCM centrándose en los algortimos de reemplazo caché. Estas propuestas están basadas en tres políticas convencionales: LRU (por ser la política comúnmente aceptada como referencia sobre la que comparar) y en DRRIP y peLIFO dado que son las políticas más utilizadas o que mejores resultados reportan en cuanto a rendimiento y reducción de tasa de fallos en el LLC se refiere.

Para estudiar el comportamiento de las políticas de reemplazo caché en entornos PCM se ha analizado experimentalmente la respuesta de los distintos algoritmos de reemplazo, los convencionales y los aquí propuestos, ante las entradas de prueba usadas como norma de comparación (los SPEC2006). De acuerdo con los resultados obtenidos, CLP con *N-chance* igual a la asociatividad es, dentro de los algoritmos propuestos con anterioridad a este trabajo, el

que mejor se comporta en cuanto a reducción de número de escrituras a PCM se refiere. Sin embargo esta política degrada significativamente el rendimiento del sistema. En ese sentido se podría decir que DRRIP presenta el mejor comportamiento entre los algoritmos convencionales ya que logra un balance entre la reducción del número de escrituras a memoria principal y el rendimiento del sistema.

Algunas de las propuestas presentadas en este trabajo consiguen mejorar a DRRIP en términos de escrituras a PCM, como es el caso de DRRIPW7, DRRIPW8 y DRRIPW9, que lo logran sin degradar apenas el rendimiento del sistema; por el contrario logran reducir el consumo de energía y superan incluso a CLP con *N-chance* igual a la asociatividad en reducir el número de escrituras a memoria principal. Particularmente, DRRIPW7 logra reducir hasta en un 16 % el número de escrituras a memoria principal respecto a LRU. Esto se hace a costa de un ligero incremento en la tasa de fallos de L_3. Sin embargo, el consumo y el rendimiento de esta propuesta, precisamente por la disminución en el número de escrituras a PCM, es muy parejo al de DRRIP, por lo que se puede considerar como una propuesta eficiente en un entorno PCM. Además, la implementación de DDRIPW7 no implica un cambio muy drástico respecto a DRRIP original, por lo que podría implementarse fácilmente.

En lo que respecta a las políticas basadas en peLIFO, peLIFO6 constituye una mejora a peLIFO original en cuanto a la reducción del número de escrituras a PCM, tasa de fallos, consumo de energía y TMAM se refiere. La técnica usada en peLIFO6 se basa en una de las utilizadas por DRRIPW7, esto es, el factor a considerar en el *set-dueling* para decidir entre una política u otra no es el número de aciertos sino que se elige la política que haya reemplazado menos bloques que generen una escritura retardada.

En cuanto a posibles líneas futuras de investigación con las que continuar

este trabajo, queda extender a un entorno *multicore* el trabajo realizado, dado que este es el escenario predominante en la actualidad. Para hacer esto se debe preparar el entorno de simulación de modo que soporte varios procesadores. Asimismo, es posible plantear políticas a nivel de compilador o sistema operativo que contribuyan a reducir el número de escrituras a PCM evitando aquellas escrituras de datos que no sean estrictamente necesarias.

Capítulo 7

Publicaciones realizadas

- R. Rodríguez, R. González-Alberquilla, F. Castro, D. Chaver, L. Piñuel, and F. Tirado, "Optimización de políticas de reemplazo cache para entornos PCM,". *XXIII Jornadas de Paralelismo, Jornadas Sarteco*, 2012.

Bibliografía

[1] M. V. Wilkes, "The memory gap and the future of high performance memories," *ACM SIGARCH Computer Architecture News*, 2001.

[2] D. Hennessy, J.L. and Patterson, *Computer architecture: a quantitative approach*. Morgan Kaufmann Pub, 2011.

[3] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable dram alternative," *ACM SIGARCH Computer Architecture News*, vol. 37, p. 2, June 2009.

[4] M. Qureshi and S. Gurumurthi, "Phase Change Memory: From Devices to Systems," *Synthesis Lectures on*, 2011.

[5] H. Messmer, *The Indispensable PC Hardware Book*. Addison-Wesley Professional, 4 edition ed., 2001.

[6] S. I. Association, "International Technology Roadmap for Semiconductors (ITRS)," *, CA: Semiconductor*, 2010.

[7] M. K. Qureshi, V. Srinivasan, and J. a. Rivers, "Scalable high performance main memory system using phase-change memory technology," *ACM SIGARCH Computer Architecture News*, vol. 37, p. 24, June 2009.

[8] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison Wesley, 2010.

[9] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," *ACM SIGARCH Computer Architecture News*, vol. 37, p. 14, June 2009.

[10] W. Zhang and T. Li, "Characterizing and mitigating the impact of process variations on phase change based memory systems," *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture - Micro-42*, p. 2, 2009.

[11] M. Qureshi and M. Franceschini, "Morphable memory system: a robust architecture for exploiting multi-level phase change memories," *Computer Architecture*, 2010.

[12] C. Kim, "LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies," *IEEE Transactions on Computers*, vol. 50, no. 12, pp. 1352–1361, 2001.

[13] A. Jaleel, K. Theobald, and S. S. Jr, "High performance cache replacement using re-reference interval prediction (RRIP)," *ACM SIGARCH Computer*, 2010.

[14] C. Wu, A. Jaleel, and W. Hasenplaugh, "SHiP: Signature-based hit predictor for high performance caching," *Proceedings of the*, 2011.

[15] S. Cho, "Flip-n-write: a simple deterministic technique to improve pram write performance, energy and endurance," *, 2009. MICRO-42. 42nd Annual IEEE/ACM*, 2009.

[16] L. a. Belady, "A study of replacement algorithms for a virtual-storage computer," *IBM Systems Journal*, vol. 5, no. 2, pp. 78–101, 1966.

[17] S. Sepúlveda, *Extensión de la Política de Reemplazamiento peLifo a Entornos Multicore*. Proyecto de fin de máster, Universidad Complutense de Madrid, 2010.

[18] E. S. Algarabel, *Técnicas de simplificación de la política de reemplazamiento caché Probabilistic Escape LIFO*. Proyecto de fin de máster, Universidad Complutense de Madrid, 2010.

[19] M. Qureshi, A. Jaleel, Y. Patt, and S. Steely, "Adaptive insertion policies for high performance caching," *ACM SIGARCH*, 2007.

[20] HP, "Inside the Intel Itanium 2 Processor," Tech. Rep. July, HP Technical White Paper, 2002.

[21] S. Microsystems, "UltraSPARC T2 ™ Supplement to the UltraSPARC architecture," tech. rep., Sun Microsystems, Santa Clara, 2007.

[22] A. P. Ferreira, M. Zhou, S. Bock, B. Childers, R. Melhem, and D. Mossé, "Increasing pcm main memory lifetime," *Proceedings of the*, 2010.

[23] C. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, S. Wallace, V. Janapa, and G. Lowney, "Pin: Building Customized Program Analysis Tools," *Design*, 2005.

[24] B. Lucia, "https://github.com/blucia0a/MultiCacheSim."

[25] I-acoma/University of Illinois at Urbana-Champaign, "http://iacoma.cs.uiuc.edu/~paulsack/sescdoc/."

[26] A. R. Alameldeen and A. Jaleel, "http://www.jilp.org/jwac-1/."

[27] W. River, "http://www.windriver.com/products/simics/."

[28] SPEC, "http://www.spec.org/cpu2006/."

[29] Intel, "http://www.intel.com/content/www/us/en/processors/core/core-i7-processor.html."

[30] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "CACTI 6.0: A Tool to Model Large Caches CACTI 6.0: A Tool to Model Large Caches," tech. rep., 2009.

[31] M. Zhou, Y. Du, B. Childers, R. Melhem, and D. Mossé, "Writeback-aware partitioning and replacement for last-level caches in phase change main memory systems," *ACM Transactions on Architecture and Code Optimization*, vol. 8, pp. 1–21, Jan. 2012.

Índice alfabético

Algoritmo
    NRU, 28
    peLIFO, 27
    RRIP, 29
    SHiP, 33

CACTI, 49
Celda de Memoria *Flash*, 12
Celda de memoria ferro-eléctrica, 13
core i7, 48

DRRIPW1, 39
DRRIPW2, 39
DRRIPW3, 40
DRRIPW4, 40
DRRIPW5, 40
DRRIPW6, 41
DRRIPW7, 41
DRRIPW8, 41
DRRIPW9, 41

Electrón caliente, 11
Escritura a nivel de bloques, 20
Escritura directa, 5
Escritura retardada, 5

FeRAM, 13
FET, 11, 12
Filtrado de escritura de granularidad fina, 21
Flip-N-Write, 22

Hot electron, 11

instrumentación, 44

Jerarquía de Memoria, 3

Brecha de Memoria, 1
Memoria de transferencia de giro de torque, 14
Memoria ferro-eléctrica, 13
Memoria Flash, 11
Memoria Principal, 3
Memoria RAM Magnética, 14
Memoria RAM Resistiva, 14
Memoria Secundaria, 3
Memory Gap, 1
Modelo de Consumo de Energía, 49
MRAM, 14
MultiCacheSim, 46
Arquitectura del MultiCacheSim, 46

no exclusividad, 4

Outcome, 33

peLIFO1, 42
peLIFO2, 42
peLIFO3, 42
peLIFO4, 43
peLIFO5, 43
peLIFO6, 43
Pin, 44
Arquitectura de Pin, 45
Pintools, 44
Políticas
    Aleatorio, 23
    Bélády, 23
    BRRIP, 32
    CLP, 34
    DRRIP, 32
    LRU, 24
    NRU, 28
    peLIFO, 25
    RRIP, 29
    SHiP, 32
polarización, 13
Probabilidad de escape, 26
Punto de Escape, 26

reemplazo, Políticas de, 22
RRAM, 14
RRPV, 29

SESC, 46
SHCT, 33
Signature, 33
SPEC, 47
SPEC2006, 47
STT-RAM, 14

Túnel Fowler-Nordheim, 12

Wear Leveling, 20
WPP1, 37
WPP2, 38
write-back, 5
write-through, 5