Optimización de políticas de reemplazo cache para entornos PCM

Roberto Rodríguez, Rodrigo González-Alberquilla, Fernando Castro, Daniel Chaver, Luis Piñuel y Francisco Tirado[1]

Resumen— **El diferente ritmo de evolución 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, se suma el hecho de que la capacidad de escalado de la memoria 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 desventajas que deben ser mitigadas antes de que PCM pueda ser empleada como tecnología de memoria principal en la próxima generación de computadores. En particular, la vida útil de este tipo de memorias (limitada por el número de escrituras 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 cache, así como se detallan nuevas propuestas de políticas orientadas a reducir el número de escrituras en memoria principal y de esta forma, sin degradar significativamente el rendimiento del sistema, aumentar la vida útil de la misma cuándo ésta se implementa usando la tecnología PCM.

Palabras clave— **PCM, Jerarquía de memoria, Políticas de reemplazo cache.**

I. Introducción

EXISTE una marcada diferencia o brecha [1], [2] entre la velocidad de evolución de los procesadores y la de la memoria principal, de modo que el rendimiento de los procesadores crece a un ritmo mucho más elevado que el de la memoria principal. Esto genera un aumento en la latencia procesador-memoria, que tiende a convertirse en un cuello de botella que reduce el rendimiento total del computador.

En un sistema computador se utiliza una jerarquía de memoria para crear al microprocesador la ilusión de tener una memoria ideal, esto es, una memoria con un tiempo de respuesta cercano a cero y con una capacidad de almacenamiento ilimitada. Para implementar dicha jerarquía de memoria, se establecen diferentes niveles donde cada uno de ellos puede hacer uso de diferentes tecnologías así como de diferentes algoritmos de reemplazo para entrelazar cada nivel. Así, en el nivel más cercano al procesador se utilizan memorias de tipo SDRAM constituyendo el nivel más rápido y al mismo tiempo, dado el alto costo económico de esta tecnología, el de menor capacidad de almacenamiento. Este nivel lo constituye la memoria cache. 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, independiente del tipo de tecnología. Este nivel presenta mayor capacidad de almacenamiento y latencia que el nivel anterior.

En las celdas DRAM los condensadores deben de ser lo suficientemente grandes, no sólo 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 drenaje y la fuente; por estas razones no se tienen soluciones más allá de los 40nm [3], [4]. 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 la búsqueda de diferentes tecnologías que puedan ser eficientemente utilizadas como memoria principal, actualmente destacan: Memoria *Flash*, RAM Ferro-Eléctrica, RAM Magnética y de Transferencia de Giro de Torque, RAM Resistiva y PCM (*Phase Change Memory*) [5]. Entre éstas, considerando el nivel de integración (se han fabricado prototipos de 3nmx20nm), capacidad de almacenamiento que permite y velocidad de escritura y lectura, se tiene que la opción más factible es PCM. Asimismo, una celda en esta tecnología puede almacenar varios bits y, en contraposición a DRAM, se trata de una memoria no volátil.

A pesar de esta posición ventajosa, PCM presenta serias deficiencias en lo que respecta a su vida útil [6], [7], [8], [9], [5], dado que ésta posee un valor de entre 10^6 y 10^8 ciclos de escritura y borrado mientras que en DRAM supera los 10^{15} ciclos. Esto motiva que para considerar realmente a PCM como una alternativa a DRAM esta deficiencia deba solventarse. Para ello, recientemente diversos autores plantean modificar ciertos aspectos arquitectónicos y de gestión de la jerarquía de memoria. Uno de estos aspectos lo constituye la política de reemplazo escogida para los distintos niveles de cache, que en este contexto estará encaminada no sólo a reducir la tasa de fallos y aumentar el rendimiento general del sistema, sino también a reducir el número de escrituras a memoria principal.

El presente trabajo analiza el comportamiento de

[1]Grupo ArTeCS, Dpto. de Arquitectura de Computadores y Atomática, Universidad Complutense de Madrid, e-mail: {robalrr,rgalberquilla,fcastror,dani02,lpinuel,ptirado}@pdi.ucm.es.

las políticas clásicas de reemplazo cache [2] y también actuales [10], [11], [12] desde el punto de vista del número de escrituras a memoria principal, así como propone nuevas políticas o modificaciones de las ya existentes encaminadas a reducir este número sin que se genere una degradación significativa en el rendimiento del sistema.

El documento está organizado como sigue: el capítulo 2 describe el trabajo relacionado en el área. El capítulos 3 presenta los algoritmos de reemplazo propuestos para aumentar la vida útil de las memoria PCM. Los capítulos 4 y 5 detallan el entorno experimental utilizado y los resultados experimentales obtenidos respectivamente. Finalmente, el capítulo 6 establece las conclusiones derivadas de este estudio.

II. Trabajo relacionado

En el caso de PCM se han propuesto varias técnicas para aumentar su vida útil, entre ellas están:

A. Técnicas de reducción de escrituras en PCM

- Filtrado de escritura de granularidad fina [5], [6]: Se ha comprobado que muchas de las escrituras que se realizan son redundantes, por lo que esta propuesta consiste en comparar el dato a escribir con el dato ya presente en memoria y escribir únicamente si ambos difieren.
- *Flip-N-Write* [13]: Al escribir se comparan tanto el dato que se desea escribir como su inverso bit a bit, de modo que el que presente menos cambios es el que se escribe, escribiendo efectivamente sólo los bits que cambian. Se debe agregar un bit extra que indica si se tiene almacenado el dato o su inverso lógico.
- *Wear Leveling* [5]: Se propone realizar una rotación de páginas periódicamente, a fin de garantizar que las escrituras a memoria principal se distribuyan uniformemente y evitar así que alguna página muy accedida provoque que se alcance prematuramente el tiempo de vida útil de una celda con el consiguiente fracaso de todo el sistema.
- Escritura a nivel de líneas [9]: Se propone realizar la escritura a una granularidad más fina, no a nivel de páginas sino a nivel de líneas, tal y como son manejadas por el procesador. De esta forma se reduce el número de escrituras innecesarias.

B. Políticas de reemplazo cache

Cuando el procesador solicita un dato que no se encuentra disponible en memoria cache (fallo), éste se debe buscar en el nivel siguiente de la jerarquía de memoria, sea otro nivel de memoria cache, memoria principal o secundaria. Cuando llevar el dato al nivel en que se originó el fallo implica el desalojo de un bloque, el bloque escogido para ello obedece al algoritmo de reemplazo empleado.

De entre los algoritmos clásicos y modernos en este trabajo se analizan los siguientes:

- Aleatorio [2]: Se selecciona la línea de cache a reemplazar de forma aleatoria.
- LRU [2]: Este algoritmo trata de reducir la posibilidad de descartar información que va a ser utilizada en un futuro cercano. Para ello se almacena información realtiva al acceso a los bloques de forma que se descarta el bloque que no ha sido utilizado durante más tiempo.
- PeLIFO [10]: Esta política se basa en el algoritmo LIFO. Utiliza una pila de llenado para registrar el orden de ingreso de los bloques en la cache, alojándose en último puesto de la misma el candidato a ser reemplazado. El autor de PeLIFO introduce varias optimizaciones para mejorar el pésimo rendimiento de la política LIFO, consistentes en definir, dinámicamente, varias puntos de escape desde donde reemplazar, en lugar de tener uno fijo como sucede en LIFO. Específicamente se definen cuatro puntos asociados a cuatro políticas P_1, P_2, P_3 y P_4 (los tres primeros pueden variar dinámicamente y el cuarto es LRU). La filosofía subyacente detrás de PeLIFO es tratar de mantener la parte inferior de la pila para reusos a largo plazo.
- RRIP [11]: Esta política se basa en el algortimo NRU [14], [15], en donde a cada línea de cache se le asocia un bit para indicar si se ha producido un acierto sobre dicha línea. En un reemplazo se elige una línea cuyo bit no muestre aciertos; de no existir se cambian todos los bits de acierto a no acierto y se reemplaza la primera línea. En RRIP lo que se hace es agregar más bits de acierto de forma que no sólo existan dos posibles estados. Estos bits (2) definen el RRPV (valor de predicción de reuso por sus siglas en inglés), en el que se pueden tener 4 posibles estados. De esta forma se habla de accesos *cercanos, lejanos* y *distantes*. Cuando se requiere eliminar una línea se escoge la que tenga su RRPV como *distante*. Cada vez que se da un acierto el RRPV se promueve a *cercano*. Esto se puede hacer de dos formas: al darse un acierto en un acceso se decrementa en uno el RRPV, o bien se cambia de una vez a acceso *cercano*. El primer método se llama RRIP de prioridad de frecuencia (RRIP-FP) y el segundo RRIP de prioridad de acierto (RRIP-HP). Para hacer RRIP inmune a los patrones de datos de poco reusos se propone DRRIP, en donde se usa *set dueling* [16]; en concreto se usan dos versiones de RRIP: una en donde los datos se insertan como accesos *distantes* y otra en donde se insertan como accesos *lejanos*, de modo que dinámicamente se elige la mejor política.
- Ship [12]: Este algoritmo propone realizar una predicción de grano fino categorizando las referencias hechas a los bloques de cache dentro de diferentes grupos mediante la asociación de una firma con cada referencia a cache. El objetivo es que las referencias de cache 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; de esta forma establece cuál debería ser su predicción de acceso, si se habla de RRIP determinaría si el valor de RRPV debe ser *cercano*, *lejano* o *distante*.

- CLP [17]: Este algortimo se propuso específicamente para reducir las escrituras en memoria principal y así aumentar el tiempo de vida de las memorias PCM. Supone una variación de LRU en la que en lugar de reemplazar la línea menos recientemente usada se reemplaza la menos recientemente usada que no haya sido modificada. Dado que esto puede afectar gravemente el rendimiento, proponen lo que denominan *N-chance*. Esto es, se define el número de veces que se va a tratar de buscar una línea limpia: cuando se requiere reemplazar una línea se busca la menos recientemente usada que esté limpia, pero ésto solo se intenta para las N (N es un número entre 1, que sería LRU, y la asocitividad de la cache) menos recientemente utilizadas. De no encontrarse ninguna se reemplaza la menos recientemente usada aunque esté sucia.

III. Políticas Propuestas

- WPP: Este algoritmo, basándose en LRU, propone ingresar las escrituras en la posición MRU (más recientemente usada) y las lecturas en algún otro punto, por ejemplo en la posición *asociatividad/2*.
- WPP2: Se basa en WPP pero además de ingresar las lecturas en otro punto, propone que las líneas limpias puedan promoverse hasta el punto de ingreso y no hasta la cima de la cola LRU, mientras que las líneas sucias sí pueden promoverse hasta la cima.
- RRIPW: Esta política, basada en RRIP, propone que las escrituras ingresen con un valor RRPV de acceso *cercano* y las lecturas como acceso *lejano*. Asimismo, los aciertos a líneas limpias se tratan como RRIP-FP y los de líneas sucias como RRIP-HP.
- RRIPW2: Esta política se basa en RRIPW. Se agrega un bit extra al RRPV de forma que puede tomar valores de cero a siete, siendo cero un acceso *cercano*, cuatro uno *lejano* y siete uno *distante*. Las escrituras se agregan como un acceso *cercano* y las lecturas como uno *lejano*. Al darse un acierto, si se trata de una línea sucia se promueve a acceso *cercano* (RRIP-HP); si se trata de una línea limpia se decrementa su RRPV en uno (RRIP-FP). Además, las líneas limpias se promueven a lo sumo a acceso *lejano*, es decir, al darse un acierto en una lectura se decrementa su RRPV en uno siempre y cuando sea mayor que *lejano*.
- DRRIPW: Esta política se basa en DRRIP. Lo que se hace es variar las políticas de inserción que se evalúan en el *set-dueling*; en este caso se elige entre dos políticas: una que inserta las líneas limpias como acceso *distante* y las sucias como acceso *lejano*, y otra que inserta las líneas limpias como acceso *lejano* y las sucias como acceso *cercano*. Asimismo, al igual que en RRIPW1, los aciertos a líneas limpias se tratan como RRIP-FP y los de líneas sucias como RRIP-HP.
- PeLIFO1: Es exactamente igual que PeLIFO original, pero se añade una condición más (la tercera que se detalla a continuación) para determinar qué bloque se debe reemplazar:
  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

  Si no se encuentra ningún bloque que satisfaga estas condiciones, se escoge el bloque LRU.
- PeLIFO2: Exactamente igual que PeLIFO1, pero en el caso de que no se encuentre ningún bloque que satisfaga las condiciones, se escoge el bloque LRU que esté limpio (combinación de PeLIFO1 y CLP).
- PeLIFO3: Como PeLIFO original, pero en vez de calcular las probabilidades de escape como probabilidad de que los bloques de la cache que están en posiciones de la pila de llenado mayores que una dada (k) generen aciertos, se calculan como la probabilidad de que los bloques de la cache que están en posiciones mayores que k generen aciertos de escritura.
- 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 política mejor es la que más aciertos de escritura tiene.
- PeLIFO5: Como PeLIFO3, pero en vez de escoger el punto de escape a utilizar en base al número de fallos de cada política, se hace en base al número de fallos que provoquen una escritura en el siguiente nivel de la jerarquía.
- 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, se hace en base al número de fallos que provoquen una escritura en el siguiente nivel de la jerarquía.

IV. Entorno Experimental

El *MultiCacheSim* [18] es un simulador basado en la memoria cache utilizada en el simulador SESC [19]. Como entrada al simulador se hace uso de Pin [20], que es una herramienta que realiza instrumentación binaria en tiempo de ejecución.

El simulador funciona del siguiente modo: se toman aplicaciones reales las cuales se instrumentan con Pin, de esta forma los accesos a memoria se

envian al simulador, éste simula la jerarquía de cache deseada y registra los accesos y escrituras a memoria principal. La figura 1 muestra un diagrama de bloques del sistema de simulación utilizado.

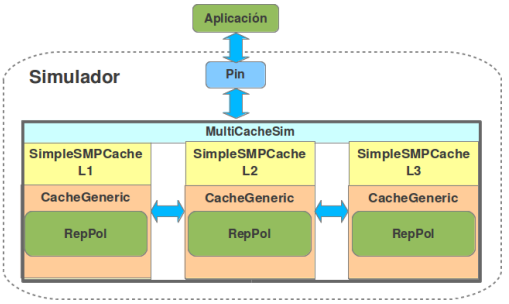

Fig. 1
Arquitectura del simulador utilizado.

Para realizar las pruebas de evaluación de las políticas de reemplazo se hizo uso de los *benchmarks* suministrados en la distribución SPEC CPU2006 [21]. Se registraron datos para la totalidad de las 29 aplicaciones que componen esta distribución, ejecutándose éstas completamente para cada una de las políticas de reemplazo y utilizando entradas de *train* en todos los casos.

La jerarquía de cache elegida, basada en el sistema *Core i7* de Intel [22], incluye tres niveles no exclusivos. El tamaño de cada nivel de cache, así como otros parámetros asociados, se muestra en la tabla I.

TABLA I
Configuración de memoria cache

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

TABLA II
Latencia (L) y consumo de energía (E)

Nivel	L (ciclos)	E (R/W/Tag) (J)
L1	1	0.222/0.211/0.001
L2	10	0.530/0.542/0.005
L3	30	2.258/2.503/0.019
PCM lectura	200	59.604
PCM escritura	4000	357.627

V. Resultados experimentales

Se inicia esta sección realizando una evaluación, en cuanto a número de escrituras a memoria principal se refiere, de los distintos algoritmos clásicos así como de los más recientes. Es importante destacar que los algoritmos de reemplazo evaluados se aplican al último nivel de cache (L3 en este caso) mientras que tanto L1 como L2 utilizan en todos los casos LRU.

En la figura 2 se muestra el promedio de escrituras (para las distintas políticas) normalizado al caso del algoritmo LRU. Como puede observarse, DRRIP es, entre los algoritmos ya propuestos, el que menor número de escrituras en memoria principal conlleva, lo que se traduce en una mayor vida útil para una memoria principal tipo PCM. Específicamente y de acuerdo con los datos mostrados, DRRIP reduce el número promedio de escrituras considerando las 29 aplicaciones alrededor de un 13,5 % respecto a LRU. Entre los algoritmos propuestos en este trabajo el que mejor resultados reporta es DRRIPW, que consigue reducir este número en un 15 % sobre LRU y en alrededor de un 1,3 % sobre DRRIPW, seguido de PeLIFO6 que prácticamente iguala a DRRIP reduciendo el número de escrituras en casi un 7 % respecto a PeLIFO original.

Se evalúa a continuación el impacto de los distintos algoritmos sobre la tasa de fallos en L3. De acuerdo con la figura 3, DRRIP es la que menor tasa de fallos reporta, mientras que para DRRIPW este porcentaje es un 4 % superior. En el caso de PeLIFO6, aún siendo un 3 % superior a DRRIP, se mejora la tasa de fallos respecto a PeLIFO original.

Finalmente, se incluye un estudio inicial acerca del rendimiento y el consumo de energía de los distintos algoritmos para evaluar más fehacientemente el potencial de los algoritmos propuestos en este trabajo.

En cuanto al modelo de energía utilizado, para evaluar los consumos por acceso en los distintos niveles de cache se utiliza CACTI 6.5 [23], mientras que para determinar el consumo por accesos a memoria principal se sigue lo propuesto en [24], de acuerdo a lo cual el consumo por lectura en PCM es de 1J/GB mientras que para una escritura es de 6J/GB. Los datos referentes a latencia y consumo de energía se muestran en la tabla II. De acuerdo con este modelo, el consumo de energía asociado a cada política, normalizado al de LRU, se muestra en la figura 4. Aquí se muestra como el algoritmo que posee un menor consumo de energía es DRRIP, sin embargo es importante resaltar como PeLIFO6 reduce el consumo respecto a PeLIFO original y el incremento en el caso de DDRIPW respecto a DRRIP es inferior al 1 %.

Por último, para hacer una estimación del impacto en el rendimiento de cada uno de los algoritmos en un entorno PCM, se calcula el *Tiempo medio de acceso a memoria* (TMAM) a partir del número de aciertos y fallos acaecidos en cada nivel y de las latencias asociadas a los mismos. La figura 5 muestra los valores de TMAM obtenidos normalizados a LRU. Se puede comprobar que DRRIP es nuevamente la que mejor resultado reporta, pero tanto DRRIPW como PeLIFO6, reduciendo significativamente el número de escrituras a memoria principal, no sufren como contrapartida un incremento apreciable del TMAM.

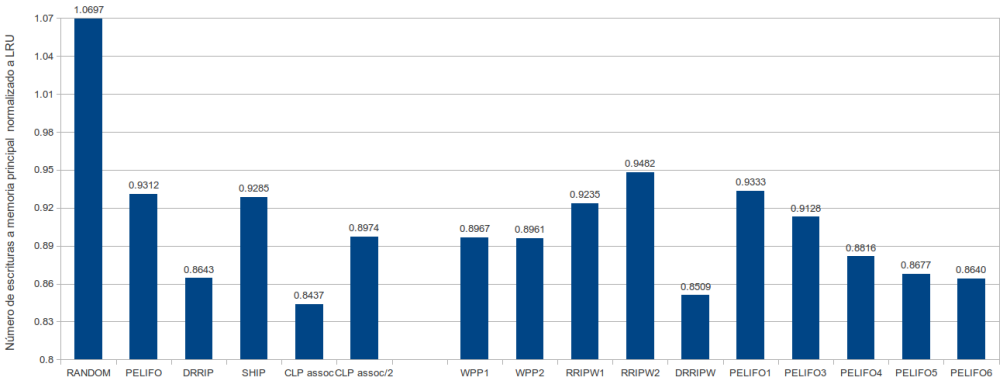

Fig. 2

Escrituras a memoria principal normalizadas a LRU.

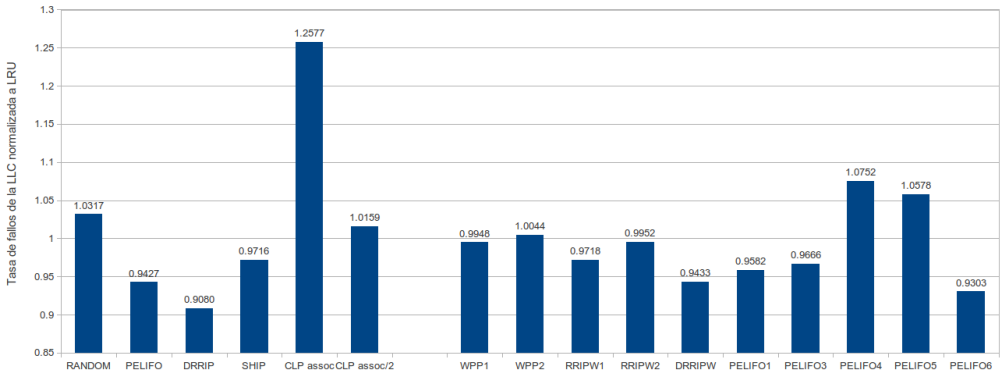

Fig. 3

Tasas de fallos en el LLC normalizadas a LRU.

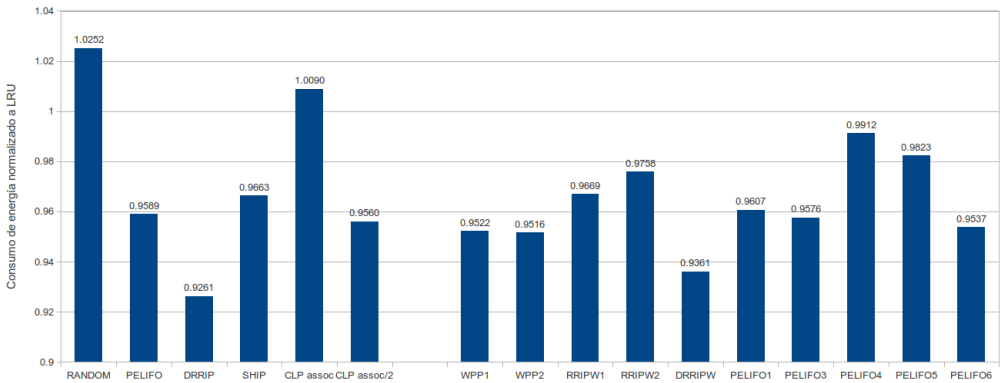

Fig. 4

Consumo de energía normalizado a LRU.

VI. Conclusiones

En este trabajo se aborda el estudio del comportamiento de distintos algoritmos de reemplazo clásicos en un entorno PCM y se proponen nuevas políticas para reducir el número de escrituras a memoria principal en este tipo de escenarios, incrementando así el tiempo de vida útil de esta clase de memorias. De acuerdo con los resultados obtenidos, DRRIP es, dentro de los algoritmos clásicos, el que mejor se comporta en cuanto a número de escrituras a PCM se

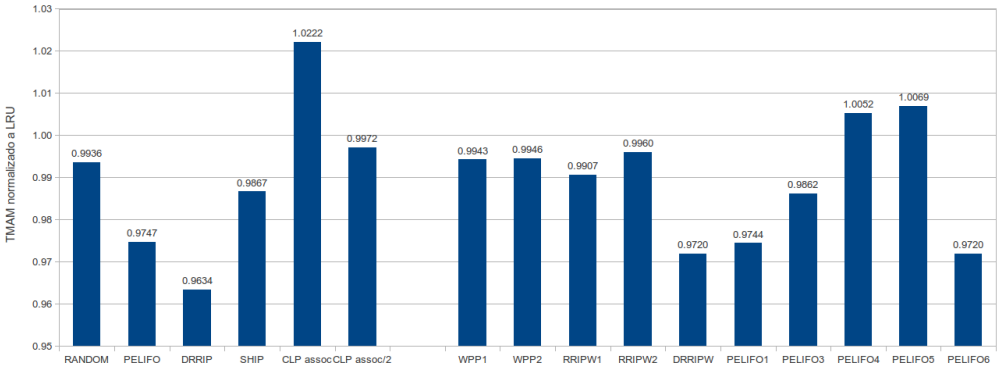

Fig. 5
TMAM normalizado a LRU.

refiere. Sin embargo algunas de nuestras propuestas consiguen mejorar a DRRIP, y en el caso de PeLIFO6 observamos una mejora respecto a PeLIFO original tanto en número de escrituras a PCM como en consumo de energía y TMAM. La mejor de las propuestas planteadas reduce el número de escrituras en un 15 % respecto a LRU. Esto se hace a costa de un ligero incremento en la tasa de fallos del LLC. Sin embargo, el consumo y el rendimiento de estas propuestas, precisamente por la disminución en el número de escrituras a PCM, es muy parejo al de DRRIP, por lo que se puede considerar como propuestas eficientes en un entorno PCM.

Agradecimientos

El presente trabajo ha sido financiado por el Gobierno de España a través del proyecto de investigación CICYT TIN 2008/508, Consolider Ingenio-2010 CSD2007-0050, y la red de Excelencia Europea HIPEAC-2.

Referencias

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

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

[3] Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger, "Architecting phase change memory as a scalable dram alternative," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3, pp. 2, June 2009.

[4] Semiconductor Industry Association, "International Technology Roadmap for Semiconductors (ITRS)," , *CA: Semiconductor*, 2010.

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

[6] Ping Zhou, Bo Zhao, Jun Yang, and Youtao Zhang, "A durable and energy efficient main memory using phase change memory technology," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3, pp. 14, June 2009.

[7] Wangyuan Zhang and Tao 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.

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

[9] Moinuddin K. Qureshi, Vijayalakshmi Srinivasan, and Jude a. Rivers, "Scalable high performance main memory system using phase-change memory technology," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3, pp. 24, June 2009.

[10] CS 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.

[11] Aamer Jaleel, KB Theobald, and SC Steely Jr, "High performance cache replacement using re-reference interval prediction (RRIP)," *ACM SIGARCH Computer*, 2010.

[12] Carole-Jean Wu, Aamer Jaleel, and Will Hasenplaugh, "SHiP: Signature-based hit predictor for high performance caching," *Proceedings of the*, 2011.

[13] 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.

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

[15] Sun Microsystems, "UltraSPARC T2 Supplement to the UltraSPARC architecture," Tech. Rep., Sun Microsystems, Santa Clara, 2007.

[16] MK Qureshi, Aamer Jaleel, YN Patt, and SC Steely, "Adaptive insertion policies for high performance caching," *ACM SIGARCH*, 2007.

[17] Alexandre P. Ferreira, Miao Zhou, Santiago Bock, Bruce Childers, Rami Melhem, and Daniel Mossé, "Increasing pcm main memory lifetime," *Proceedings of the*, 2010.

[18] Brandon Lucia, "https://github.com/blucia0a/ MultiCacheSim," .

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

[20] Chi-keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Steven Wallace, Vijay Janapa, and Geoff Lowney, "Pin: Building Customized Program Analysis Tools," *Design*, 2005.

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

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

[23] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P Jouppi, "CACTI 6.0: A Tool to Model Large Caches," Tech. Rep., 2009.

[24] Miao Zhou, Yu Du, Bruce Childers, Rami Melhem, and Daniel 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, no. 4, pp. 1–21, Jan. 2012.