Selección de contenidos basada en reuso para caches compartidas en exclusión

J. Díaz[1], T. Monreal[2], V. Viñals[1], P. Ibáñez[1], J.M. LLabería[2]
1) Universidad de Zaragoza, 2) Universitat Politècnica de Catalunya

Resumen— **Publicaciones previas revelan que el flujo de referencias que llega a la cache compartida (SLLC) de un chip multiprocesador muestra poca localidad temporal. Sin embargo, muestra localidad de reuso, es decir, los bloques referenciados varias veces tienen más probabilidad de ser referenciados en un futuro. Esto provoca que, si se realiza una gestión convencional, el uso de la cache es ineficiente. Existe varias propuestas que abordan este problema para caches inclusivas, pero pocas que se centren en caches exclusivas. En este trabajo se propone un nuevo mecanismo de selección de contenidos para caches exclusivas que aprovecha la localidad de reuso que presentan los accesos a la SLLC. Consiste en incluir un Detector de Reuso entre cada cache L2 y la SLLC. Su misión es detectar bloques sin reuso para evitar que sean insertados en la SLLC. Esta propuesta se evalúa con un conjunto de cargas multiprogramadas ejecutando en un simulador detallado de un sistema con 8 procesadores en chip y su jerarquía de memoria. Los resultados muestran que el Detector de Reuso permite incrementar el rendimiento por encima de otras propuestas recientes como CHAR o la Reuse Cache. Por ejemplo, para una configuración del Detector de Reuso balanceada entre coste y prestaciones, se obtiene un 8,5% de reducción de la tasa de fallos de la SLLC y un incremento del IPC de un 2,5%, frente a un sistema base con reemplazo TC-AGE.**

Palabras clave— **Cache compartida de último nivel, exclusión, reuso.**

I. INTRODUCCIÓN

En los últimos años, los sistemas multiprocesador en chip con memoria compartida son mayoritarios en el mercado, tanto en servidores de alto rendimiento como en sistemas de sobremesa, dispositivos móviles y sistemas embebidos. En ellos, el diseño habitual incluye una jerarquía de memoria multinivel, con una cache compartida de último nivel o SLLC (acrónimo de Shared Last Level Cache). Ésta es crítica en términos de coste, prestaciones y consumo. En coste, porque suele ocupar una superficie del chip comparable a la de varios procesadores. En prestaciones y consumo, porque es el último recurso existente antes acceder a la memoria DRAM que, situada fuera del chip, es inferior en prestaciones y consume más energía.

Varios estudios ponen de manifiesto que los diseños convencionales no resultan eficientes en su implementación de la SLLCs, ya que desaprovechan una fracción mayoritaria del espacio de almacenamiento. Esto es así porque almacenan bloques muertos, es decir, bloques que no van a ser accedidos nunca antes de su expulsión. Frecuentemente, los bloques están muertos en cuanto llegan a la SLLC [1] [2] [3]. La razón de que esto ocurra es que las caches L1 y L2 aprovechan la mayor parte de la localidad temporal, por lo que resulta *filtrada* antes de llegar a la SLLC [4] [5]. Con el objetivo de evitar este efecto, e incrementar la tasa de acierto de la SLLC, se han publicado en los últimos años varias propuestas de modificaciones en la política de inserción y reemplazo en la SLLC (ver Sección IV). La mayor parte de los trabajos se refieren a caches inclusivas o no inclusivas, y sólo un grupo reducido [3] [6] se enfoca en una SLLC exclusiva [7].

Hoy en día, hay ya microprocesadores con una SLLC exclusiva o parcialmente exclusiva [8]. A medida que el número de procesadores dentro del chip crece, también lo hace el tamaño de cache en los niveles inferiores y, por lo tanto, la diferencia en rendimiento entre una SLLC exclusiva y una inclusiva. En un futuro que se prevé de muchos procesadores (many-core) dentro del chip, y SLLCs no mucho mayores que las existentes hoy en día [9], utilizar una cache inclusiva será más ineficiente aún. Por lo tanto, es de esperar que, salvo cambios en el diseño básico de la jerarquía de memoria, la utilidad de las SLLC exclusivas crezca en un futuro.

Este trabajo se centra en mejorar la eficiencia y el rendimiento de una SLLC exclusiva en un entorno multiprocesador on-chip. En concreto, se presenta un nuevo mecanismo de selección de contenidos para caches exclusivas que aprovecha la *localidad de reuso* que presentan los accesos a la SLLC [10]. Dicha localidad consiste en que, cuando se referencia un bloque dos veces (se reusa), es probable que dicha dirección se referencie en un futuro cercano. El mecanismo propuesto persigue que sólo se guarden en la SLLC aquellos bloques que tienen reuso en dicho nivel de cache, es decir, aquellos bloques que son solicitados más de una vez desde las L2 privadas. Para ello, un elemento denominado Detector de Reuso detecta qué bloques expulsados de las L2 no presentan reuso, y evita que sean insertados en la SLLC, realizando *bypass* de los mismos.

Se evalúa esta propuesta simulando un sistema con 8 procesadores en un chip que ejecuta un conjunto de cargas multiprogramadas. El Detector de Reuso evita la inserción de bloques poco útiles en la SLLC, facilitando que se mantengan los más reusados. Ello permite incrementar el rendimiento, por encima de otras propuestas recientes.

El trabajo está estructurado como sigue. La Sección II muestra evidencia experimental de la presencia mayoritaria de bloques muertos en una SLLC exclusiva, y de que la localidad de reuso es una propiedad presente en los accesos a la misma. La Sección III explica en detalle la propuesta del Detector de Reuso. La Sección IV analiza los trabajos relacionados, y los compara con esta nueva propuesta. La Sección V detalla la metodología empleada, incluyendo el entorno de experimentación y la configuración de los sistemas simulados. La Sección VI presenta los experimentos y

analiza los resultados, comparándolos con dos propuestas actuales, que son CHAR [6] y una versión en exclusión de la Reuse Cache [11]. Por último, en la Sección VII se extraen conclusiones.

II. MOTIVACIÓN

En esta sección, se analiza el comportamiento de un conjunto de 100 mezclas multiprogramadas, creadas combinando de forma aleatoria los 29 programas de SPEC CPU 2006, ejecutando en un sistema con 8 procesadores, caches privadas y una SLLC exclusiva de 8 MB (ver detalles del entorno en la Sección V). El objetivo es comprobar que la SLLC es poco eficiente en el aprovechamiento de su espacio, y que una selección de los contenidos basada en el reuso de cada bloque puede resultar de utilidad para incrementar su eficiencia.

La Figura 1 muestra, para las 100 mezclas mencionadas, la fracción de bloques contenidos en la SLLC que de media están vivos durante la ejecución de la mezcla. Se entiende que un bloque de la SLLC está vivo en un determinado momento si experimentará al menos un acceso antes de ser expulsado, por lo cual resulta útil mantenerlo en la cache.

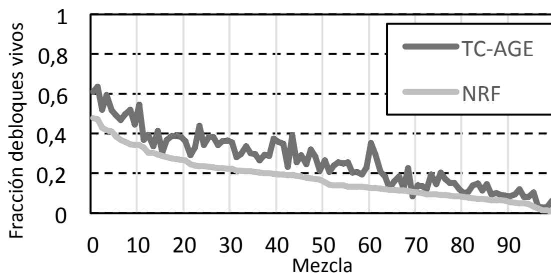

Figura 1: Fracción media de bloques vivos de una SLLC exclusiva de 8 MB, para 100 mezclas multiprogramadas de SPEC CPU2006 (ver Sección V). Se muestran valores para las políticas de reemplazo NRF y TC-AGE. Las mezclas están ordenadas de mayor a menor fracción de bloques vivos con NRF.

Por el contrario, se entiende que un bloque está muerto, en un determinado momento, cuando va a ser expulsado sin haber recibido ningún acceso. En cada mezcla, se toma información de los bloques vivos cada millón de ciclos, y se muestra en el gráfico la media de los valores.

La información se representa para dos políticas de reemplazo diferentes, NRF y TC-AGE. La política NRF (acrónimo de Not Recently Filled) es análoga a la NRU (Not Recently Used) en caches inclusivas. NRF utiliza un único bit de reemplazo por bloque, y selecciona como víctima a un bloque aleatorio que no haya sido recientemente insertado, es decir, no tenga el bit de reemplazo a uno. El bit de reemplazo se pone a uno cuando el bloque se inserta en la SLLC, proveniente de una expulsión de una cache L2 privada. Si todos los bits del conjunto están a 1, se cambian todos a 0 salvo el del bloque recién insertado. La política TC-AGE [3] para caches exclusivas es análoga a SRRIP [5] para caches inclusivas, y utiliza 2 bits por línea de cache para almacenar la edad del bloque. TC-AGE selecciona como víctima un bloque aleatorio de entre aquellos del conjunto que tengan la menor edad. La edad se asigna cuando se inserta el bloque en la SLLC. Si el bloque ya ha recibido anteriormente algún acierto en la SLLC, se le asigna la edad 3, y si no se le asigna la edad 1. La información de acierto en la SLLC se guarda en un bit adicional en la cache L2 privada, y se envía a la SLLC junto con el bloque. Cuando, tras un reemplazo, no queda ningún bloque con edad 0 en el conjunto, se resta 1 a la edad de todos los bloques. Es decir, TC-AGE asigna mayor edad, y por lo tanto menor probabilidad de reemplazo, a los bloques que hayan demostrado ser útiles en la SLLC, ya que han sido reusados.

Como puede apreciarse en la figura, la fracción de bloques vivos con NRF varía entre un 1% y un 48%, en función de la mezcla, siendo la media de un 18%. La política TC-AGE demuestra su efectividad incrementando los bloques vivos en 99 de las 100 mezclas, alcanzando una media del 26%. Estos valores demuestran que la SLLC no utiliza eficientemente su espacio de almacenamiento, ya que la mayoría de los bloques están muertos incluso usando las mejoras políticas de reemplazo propuestas en la literatura.

La Figura 2a muestra, para cada programa que participa en la carga de trabajo mencionada, la media de la distribución de bloques reemplazados de la SLLC, en función del número de accesos que cada bloque ha registrado durante su estancia en la cache. Cada bloque se clasifica según haya recibido un solo acceso (U), dos accesos (R, reuso), o más de dos accesos (M, múltiple reuso). Como política de reemplazo de la SLLC se ha usado TC-AGE.

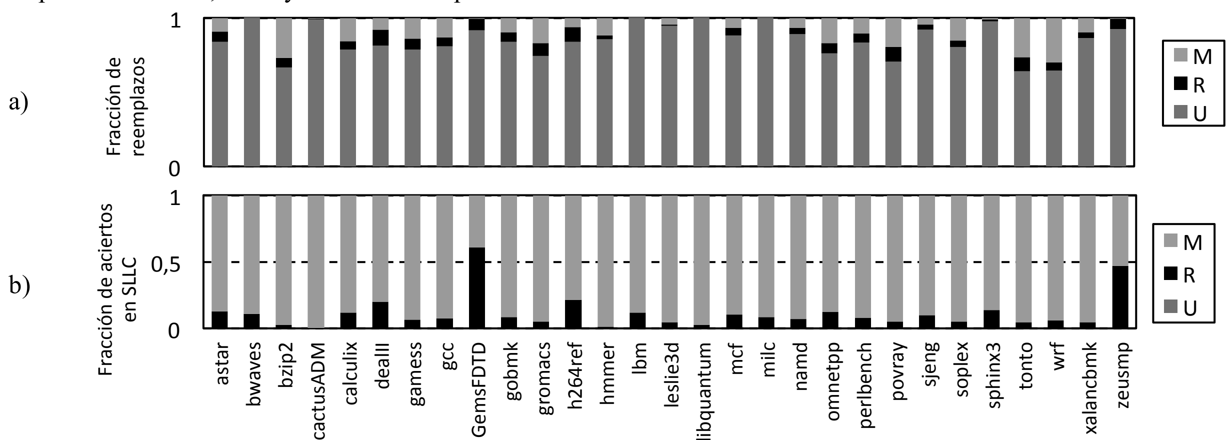

Figura 2: (a) distribución media de reemplazos de bloques de una SLLC exclusiva de 8 MB con TC-AGE, para 100 mezclas multiprogramadas de SPEC CPU2006 (ver Sección V), categorizadas según el número de accesos que recibe el bloque en la SLLC antes de su reemplazo: (U) un uso, (R) reuso – dos usos, (M) múltiple reuso – tres o más usos. (b) distribución media de aciertos en la SLLC según las mismas categorías.

Incluso con la política TC-AGE, la gráfica muestra que, en función del programa, entre un 64% y un 99% de los bloques reemplazados de la SLLC tienen un único uso antes de su reemplazo, con una media del 85%. Este uso es el que les insertó en la cache, por lo que su estancia ha sido inútil. Esto es debido a que, en este nivel de la jerarquía de memoria, la localidad temporal de los programas es escasa, ya que ha sido *filtrada* por las caches privadas. Todos estos bloques son buenos candidatos para no ser siquiera almacenados en la SLLC, es decir, para hacer bypass cuando son expulsados de la cache L2.

La Figura 2b muestra, para cada programa, la media de la distribución de los aciertos en SLLC, en función del tipo de bloque sobre el que se producen (U, R, M). La mayor parte de los aciertos de la SLLC se producen en bloques con múltiple reuso (M), es decir, a nivel de SLLC los programas muestran *localidad de reuso*. En 27 de los 29 programas, entre un 78% y un 99% de los aciertos son en dichos bloques, mientras que en *zeusmp* y *GemsFDTD* son del 59% y del 33% respectivamente.

Estos resultados nos indican que una política de selección de contenidos de la SLLC que sólo almacene los bloques que han demostrado reuso (al menos dos accesos) conseguiría guardar la pequeña porción de bloques con múltiples reusos (M en Figura 2a) que producen la mayoría de los aciertos (M en Figura 2b). Además, esta política impediría la entrada en SLLC de la gran porción de bloques que no se llegan a reusar (U en Figura 2a), lo que disminuiría la probabilidad de que los bloques M fuesen reemplazados.

III. Diseño e implementación de la propuesta

A. *Diseño general*

El diseño de partida es el de una SLLC cuyos contenidos se encuentran en exclusión con los contenidos de las caches privadas de cada procesador. Para mantener la coherencia en la jerarquía de memoria, existe también un directorio que mantiene, para cada bloque presente en dicha jerarquía, tanto su estado como la información precisa de dónde se encuentra, que puede ser uno o varios procesadores y/o la SLLC.

En dicho diseño, los bloques que llegan de memoria se envían directamente a la cache L2 solicitante. Eventualmente, el bloque es expulsado de ella y se envía para su almacenaje en la SLLC. Desde ahí, o bien el bloque es solicitado de nuevo desde alguna cache L2, siendo entonces enviado y desalojado de la SLLC, o bien el bloque es reemplazado por otro que necesita espacio para su inserción.

Sobre este diseño, la propuesta es incluir un elemento intermedio a la salida de cada cache L2, entre cada una de éstas y la SLLC. A este elemento, que llamaremos Detector de Reuso (DR), se dirigen todos los bloques expulsados de la cache L2. Al estar el DR situado fuera del camino de petición de bloques a la SLLC, el retardo que añade no afecta a los tiempos de acierto o fallo de la SLLC. En cambio, afecta al tiempo que necesita un bloque desde su expulsión de la cache L2 a su eventual llegada a la SLLC. La Figura 3 muestra un esquema de este diseño.

Figura 3: Esquema del diseño general con Detector de Reuso.

El DR decide entre enviar el bloque a la SLLC o no hacerlo, es decir, hacer bypass del mismo. La decisión se basa en las categorías mostradas en la sección anterior: Si un bloque expulsado se clasifica en la categoría U, con un único uso, se hace bypass. Si se clasifica en la categoría R o M, con uno o más reusos, no se hace bypass.

Ni la SLLC ni el directorio requieren modificaciones estructurales para adaptarse al mecanismo del DR. Sí que requieren cambios en su protocolo de coherencia y lógica de control, para tener en cuenta el posible bypass. Si bien en este trabajo se utiliza TC-AGE como política de reemplazo en la SLLC, es posible implementar el DR con cualquier política de reemplazo.

B. *El Detector de Reuso*

El DR está compuesto por un buffer que almacena direcciones de bloques y por su lógica de gestión. La misión del buffer es almacenar las direcciones de los bloques que llegan al detector, con el fin de identificar si es la primera vez que dicho bloque se expulsa de la cache L2 o si ya ha sido expulsado anteriormente. Una primera expulsión implica que no hay reuso por parte de la cache L2, mientras que las siguientes implican que sí lo hay.

El buffer está organizado de forma asociativa por conjuntos, y sus características (tamaño, asociatividad, política de reemplazo) son variables de diseño. De entre ellas, la fundamental es el tamaño. Para que el DR sea efectivo, éste ha de ser lo suficientemente grande como para almacenar una parte significativa de las direcciones de los bloques entre su uso y su reuso. Se define la *cobertura de expulsión* como el espacio de memoria ocupado en conjunto por los bloques expulsados para los que el DR puede hacer seguimiento en un determinado momento. Por ejemplo, si el DR puede hacer seguimiento de 1024 bloques de memoria, y cada bloque es de 64 B, su cobertura de expulsión es de 64 KB.

El DR utiliza también, para detectar el reuso, el dato de si el bloque fue enviado a la cache L2 desde memoria o proviene de un acierto en la SLLC. La proveniencia de la SLLC indica también reuso. Esta información se almacena en la cache L2, en un bit añadido a cada línea. Cuando el bloque es expulsado de la cache L2, este bit se incluye en el mensaje que se dirige al DR. Esta misma información es utilizada por la política de reemplazo TC-AGE, por lo que no comporta necesidades de espacio adicionales si el DR se implementa sobre un sistema que ya esté utilizando TC-AGE.

C. *Funcionamiento del Detector de Reuso*

La Figura 4 muestra un esquema del funcionamiento del DR. Sobre esta figura, se detalla a continuación las operaciones que realiza.

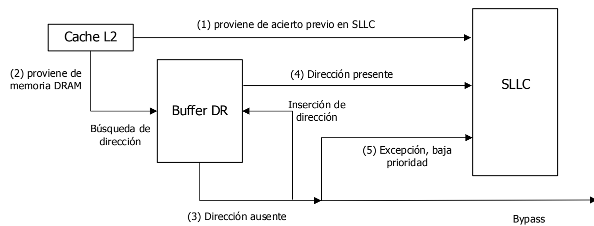

Figura 4: Esquema de funcionamiento del Detector de Reuso.

Cuando un bloque expulsado de la cache L2 llega al DR, se inspecciona primero el bit que indica si el bloque provenía originalmente de la SLLC. Si es así, esto indica que el bloque tiene reuso a ese nivel, por lo que se marca para su almacenamiento en la SLLC (1).

Si el bloque provenía de memoria DRAM (2), se busca su dirección en el buffer. Si la dirección del bloque no está presente, esto indica que es la primera vez que se expulsa de la L2, por lo que no muestra reuso. El bloque se marca para bypass, y su dirección se inserta en el buffer (3). Si el bloque está ya presente en el buffer, esto indica que es la segunda o sucesivas veces que ha estado en la cache L2, por lo que muestra reuso. Por lo tanto, el bloque se marca para su almacenamiento en la SLLC (4).

Como excepción, una fracción predeterminada de los bloques marcados para bypass se marca para su almacenamiento con baja prioridad en la SLLC, revirtiendo la decisión de hacer bypass (5). Al recibir un bloque con esta marca, la SLLC lo almacenará sólo si hay una vía libre en el conjunto correspondiente. Se insertará además con la prioridad más baja (probabilidad de reemplazo más alta) de la política de reemplazo. Esto se hace así porque, al ser la SLLC exclusiva, se genera un nuevo hueco en un conjunto cada vez que se produce un acierto. Si no hay bypassing, este hueco se rellena con relativa rapidez, pero no ocurre así si el nivel de bypass es alto, con lo que se desaprovecha espacio en la SLLC. Al rellenar estos huecos con bloques no reusados, ese espacio se utiliza, si bien no de forma tan eficiente. Las simulaciones realizadas muestran que es suficiente con transformar uno de cada 32 bloques marcados para bypass para obtener la mayor parte del beneficio en rendimiento.

Finalmente, si el bloque está marcado para su almacenamiento en la SLLC, se envía a la misma. Si está marcado para bypass, el funcionamiento depende de si el bloque está sucio o no. Es decir, si ha sido modificado o no durante su estancia en la cache L2. Si está sucio, se envía para su escritura directa en memoria y, si está limpio, se envía un mensaje de control para la actualización del directorio. El directorio se actualiza también con el resto de mensajes.

Si se utiliza TC-AGE como política de reemplazo en la SLLC, como es el caso del entorno de simulación utilizado en este trabajo, el bit que indica la proveniencia o no del bloque de la SLLC se envía también en caso de no realizar bypass. Esta información se utilizará en TC-AGE como en el algoritmo original, si bien ahora presenta un significado ligeramente diferente. La primera vez que no se hace bypass está a cero, pero el DR ya ha detectado reuso para el bloque, con al menos dos accesos. Cuando en sucesivas ocasiones está a uno, el bloque ha recibido ya al menos tres accesos. Por lo tanto, TC-AGE otorgará menos probabilidad de reemplazo en la SLLC a bloques con múltiple reuso que a bloques que han sido reusados una única vez.

D. Detalles de implementación

En una primera aproximación, el buffer del DR puede implementarse de forma asociativa por conjuntos, conteniendo la etiqueta asociada a la dirección del bloque, un bit de validez, y la información para la política de reemplazo. Cada línea de la cache del DR almacenaría una dirección de bloque.

Aunque esta implementación es sencilla y efectiva, presenta el inconveniente de requerir mucho espacio. Por ello proponemos utilizar dos técnicas para reducir el espacio necesario: Almacenar las etiquetas por dirección de sector en vez de por dirección de bloque y comprimir las etiquetas a guardar.

Se entiende por sector a un conjunto de bloques de memoria consecutivos y alineados al tamaño de sector. Almacenar en el DR las etiquetas por dirección de sector permite guardar la información de varias direcciones de bloques consecutivos de memoria en cada línea del buffer del DR. Para cada bloque es necesario mantener un bit de presencia. Por ejemplo, con un tamaño de sector de 4 bloques, una línea se compondrá de una etiqueta (calculada a partir de la dirección del sector), un bit de validez, los bits de reemplazo y 4 bits de presencia.

La compresión de las etiquetas a guardar busca almacenar menos bits para la etiqueta presente en cada línea, manteniendo una buena capacidad de discriminación entre sectores. El proceso que se realiza es como sigue: si llamamos t al número de bits de la etiqueta completa, y c al número de bits de la etiqueta comprimida, se toman los t bits y se separan en varios trozos de tamaño c. El último trozo se rellena con bits a "0" hasta ese tamaño. Después, se realiza la operación XOR de todos ellos, obteniéndose un único valor de c bits, que es el que se almacena.

La utilización de etiquetas comprimidas puede provocar falsos positivos, ya que son varios los sectores de cache L2 que comparten el mismo valor de etiqueta comprimido. Esto provoca la detección de falsos reusos, y el envío de bloques realmente no reusados a la SLLC. Estos bloques no provocan problemas funcionales, pero sí pueden llegar a degradar del rendimiento. Para limitar esta degradación es necesario mantener un número de bits suficiente. El valor concreto depende del tamaño del buffer y de su asociatividad.

El buffer del DR utiliza FIFO de 1 bit como política de reemplazo. Las simulaciones realizadas indican que utilizar FIFO de 1 bit mantiene un rendimiento del DR muy similar a otras que requieren de más espacio. La política FIFO implica que la información de antigüedad se actualiza sólo durante la inserción de una dirección, y no durante aciertos posteriores. Ello es coherente con la misión prioritaria del buffer, que es detectar el primer reuso de un bloque. Para los siguientes reusos, la información de la proveniencia del bloque de la SLLC es la fundamental, si bien se utiliza también la presencia en el buffer de forma complementaria por si el bloque ha sido expulsado de la SLLC.

La asociatividad utilizada para el buffer del DR es 16. Las simulaciones realizadas indican que mantiene un rendimiento similar a otras asociatividades más altas.

E. Coste de hardware

En esta sección, se calcula el número total de bits de almacenamiento que es necesario añadir para implementar el DR sobre el sistema base descrito en la Sección V.B. Aparte de este almacenamiento, el DR requiere también su lógica de gestión.

La configuración del DR considerada tiene una cobertura de expulsión de 2 MB, con 1.024 conjuntos de asociatividad 16, un tamaño de sector de 2 bloques, y utiliza etiquetas de 10 bits. Esta configuración es la utilizada en la Sección VI para realizar la comparativa con otras propuestas, tras el trabajo de selección de una configuración balanceada considerando prestaciones frente a coste.

Cada entrada del DR requiere 14 bits (10 de etiqueta, 2 de presencia de bloque, 1 de reemplazo y 1 de validez). El número de entradas por DR es de 16 K, lo que implica 28 KB por procesador. El coste total para los 8 procesadores es de 224 KB, un 2,6% del tamaño de la SLLC.

IV. TRABAJOS RELACIONADOS

La localidad de reuso ha sido estudiada en varias propuestas publicadas. Inicialmente se identifica y aprovecha en el ámbito de caches de discos, donde Karedla et al. proponen segmentar la pila de reemplazo LRU para separar, en dos listas, los bloques referenciados una vez de los referenciados varias veces (reusados) [12]. Esta misma estrategia se ha aplicado recientemente como política de reemplazo para SLLCs. Albericio et al. identifican que el flujo de accesos a la SLLC presenta localidad de reuso, y priorizan con su política NRR la expulsión de bloques de la lista de no (recientemente) reusados, manteniendo un coste equivalente a NRU [10]. Qureshi et al. limitan con su política LIP el tamaño de la lista no referenciada a un único elemento, y proponen un mecanismo de competición entre conjuntos con distinta política *(set dueling)* para decidir si utilizar LIP o LRU [2]. Gao et al. utilizan la segmentación de la pila LRU para la división de cada conjunto de la SLLC en dos listas de tamaño variable, aplicando otras optimizaciones como la utilización de bypassing si resulta rentable según los resultados de la competición entre conjuntos [13]. Este mismo mecanismo es utilizado por Khan et al. para limitar de forma dinámica el tamaño de las listas, y opcionalmente realizar bypass de bloques, adaptándose al comportamiento reciente de los accesos a la SLLC [14]. Al contrario que en estas propuestas, el DR evita la presencia de bloques no reusados en la SLLC, y utiliza un buffer aparte para que la detección del reuso no esté limitada por la capacidad de almacenamiento de la SLLC o las anteriores decisiones de selección de contenidos realizadas.

Otras propuestas buscan predecir el comportamiento de reuso de los bloques, y utilizan esta predicción para modificar la política de inserción y reemplazo en la SLLC. Jaleel et al. realizan una predicción del intervalo de re-referencia de cada bloque que, en su versión estática (SRRIP), asigna un intervalo intermedio a los bloques recién insertados, y un intervalo mínimo a los que son reusados, priorizando el reemplazo de los bloques con intervalo más largo. En su versión bimodal (BIP), algunos bloques son aleatoriamente insertados con menor intervalo de re-referencia y, en su versión dinámica (DRRIP), se selecciona entre los dos anteriores utilizando competición entre conjuntos [5]. Wu et al. presentan una evolución de SRRIP donde se busca mejorar su predicción estática, correlándola con otros valores de referencia como el contador de programa (PC), la región de memoria y la reciente secuencia de instrucciones ejecutada [15]. Gaur et al. adaptan SRRIP a su uso en caches exclusivas dándole el nombre de TC-AGE, y proponen nuevos mecanismos de predicción para dicho tipo de caches basándose en el número de veces que un bloque viaja de la SLLC a la cache L2 y en el número de accesos que presenta en la cache L2 [3]. El mecanismo del DR es compatible con TC-AGE, siendo de hecho la política de reemplazo seleccionada para la SLLC a la hora de mostrar resultados en la Sección VI.

En la misma línea predictiva, Li et al. realizan un seguimiento de los pares de bloques víctima y entrantes y, al detectar el primer reuso de los dos, aproximan el comportamiento que tendría un algoritmo de bypass óptimo, y predicen que aquellos bloques que se accederán desde el mismo contador de programa se comportarán igual, guiando su decisión de bypass [16]. Seshadri et al. hacen un seguimiento global a través de un filtro Bloom de las direcciones de los últimos bloques expulsados de la SLLC y, si un bloque expulsado se vuelve a acceder pronto, predicen que es un bloque con alta probabilidad de reuso, al que se asigna la menor probabilidad de reemplazo [17]. Chaudhuri et al. proponen un mecanismo (CHAR) que registra el patrón de acceso que han tenido los bloques en todos los niveles de la jerarquía de memoria, y los clasifica en cuatro clases en función del mismo. Para cada clase, se observa de forma dinámica si sus bloques muestran reuso o no, y se predice que el comportamiento futuro de un bloque será el observado para su clase. Esta decisión guía un mecanismo de bypass [6]. Todas estas propuestas son compatibles con el Detector de Reuso, y podrían utilizarse para cambiar la decisión fija que hace el DR de realizar bypass de bloques que no han demostrado previamente reuso.

Argumentando que ninguna de las alternativas existentes resuelve de forma satisfactoria la presencia mayoritaria de bloques muertos en la SLLC, Albericio et al. proponen con la Reuse Cache separar en la SLLC la matriz de etiquetas de la matriz de datos, y reducir el tamaño de esta última sin perder rendimiento, seleccionando como contenido sólo bloques que hayan demostrado reuso en la matriz de etiquetas [11]. El Detector de Reuso utiliza este mismo criterio de selección, pero el resto del diseño es diferente.

En este trabajo se ha seleccionado CHAR y la Reuse Cache como propuestas que representan el estado del arte. Hay dos razones para ello: por un lado, ambas presentan en los artículos publicados rendimientos superiores a otras de las analizadas; por otro, existe una versión de CHAR para caches exclusivas, y es posible la adaptación de la Reuse Cache a su uso en exclusión. El

rendimiento del DR se compara con ellas en la Sección VI, donde también se dan más detalles de su funcionamiento.

V. METODOLOGÍA

Esta sección detalla el entorno de experimentación y la configuración del sistema base que se han utilizado para la evaluación de la propuesta y la obtención de los resultados expuestos en la Sección VI.

A. Entorno de experimentación

Como motor de simulación se utiliza Simics [18], un simulador de ejecución de sistemas completos. También se utilizan los plug-ins Ruby y Opal de Multifacet GEMS [19]. Se usa Ruby para modelar la jerarquía de memoria con un alto grado de detalle: caches, directorio, protocolo de coherencia, red on-chip, buffers, contención, etc., añadiendo además un modelo detallado de una DRAM DDR3. Se utiliza Opal (también conocido como TFSim) para modelar de forma detallada un procesador superescalar con ejecución fuera de orden.

Se ejecuta sobre Solaris 10 para SPARC una carga de trabajo multiprogramada compuesta por aplicaciones de la suite SPEC CPU 2006 [20]. Para localizar el final de la fase de inicialización de cada programa, utilizamos contadores hardware en una máquina real, y ejecutamos todos los binarios SPARC con las entradas de referencia hasta su finalización. Para nuestro sistema con 8 procesadores hemos producido una serie de 100 mezclas, combinaciones aleatorias de 8 programas cada una, tomados de entre los 29 programas que componen SPEC CPU 2006. Cada programa aparece entre 18 y 41 veces, siendo el número medio de apariciones de 27,6 y la desviación típica de 6,1. En cada mezcla, nos aseguramos de que ninguna aplicación está en su fase de inicialización, avanzando la simulación hasta que todas las fases de inicialización están terminadas. Comenzando en este punto, en cada simulación se ejecutan 300 millones de ciclos de calentamiento del sistema de memoria, y luego recolectamos estadísticas para los siguientes 700 millones de ciclos.

No se han hechos esfuerzos por distinguir las aplicaciones por su tipología ni sus patrones o estadísticas de acceso a memoria. Se muestra en la Tabla 1 el número medio de fallos por kilo-instrucción (MPKI) de cada aplicación en todas las mezclas donde aparece, en los tres niveles de la jerarquía de memoria, cuando las ocho aplicaciones de cada mezcla se ejecutan conjuntamente sobre el sistema base.

B. Configuración del sistema base

Modelamos un sistema base con 8 procesadores superescalares con ejecución especulativa y fuera de orden. Cada procesador consta de 4 vías, una segmentación de 18 etapas y 10 unidades funcionales. El predictor de saltos es de tipo YAGS [21] con un PHT (*Pattern History Table*) de 4K entradas. La Tabla 2 muestra más información del modelo de procesador simulado.

Arquitectura base	**SPARC v9**
Procesadores	8, superescalares de 4 vías, 2.66 GHz
Segmentación	18 etapas: 4 fetch, 4 decode, 4 dispatch/read, 1 (o más) execute, 3 memory, 2 commit
Buffers	Buffer de reordenación de 128 entradas
Bancos de registros	Enteros: 160 (lógicos) + 128 (renombre) Punto flotante: 64 (lógicos) + 128 (renombre)
Unidades funcionales	4 Enteros, 4 punto flotante, 2 load/store
Predictor de saltos	Tipo YAGS PHT: 4.096 entradas

TABLA 2: ESPECIFICACIONES DEL PROCESADOR

Cada procesador tiene 2 niveles de cache privados y todos los procesadores comparten la cache exclusiva de tercer nivel. La SLLC utiliza TC-AGE como política de reemplazo, y tiene 8 MB de capacidad total, con cuatro bancos entrelazados a nivel de línea de cache (64B). Una red de tipo crossbar conecta los procesadores y dichos bancos. Hay dos canales de memoria DDR3 a 667 MHz. La Tabla 3 muestra más información de la jerarquía de memoria simulada. El cálculo de los tiempos de acceso de las caches se ha realizado mediante la herramienta CACTI [22], con una tecnología de 45 nm.

Cache privada L1 I/D	32 KB, 4 vías, reemplazo LRU, tamaño de bloque 64 B, latencia de acceso de 3 ciclos
Cache privada unificada L2	256 KB inclusiva de L1, 8 vías, reemplazo LRU, tamaño de bloque 64 B, latencia de acceso de 7 ciclos
Interconexión	Red tipo crossbar, ancho de bus de 80 bits, latencia de 5 ciclos
Cache compartida L3 (SLLC)	8 MB exclusiva (4 bancos de 2 MB cada uno), entrelazado por bloques, tamaño de bloque 64 B. Cada banco: 16 vías, reemplazo TC-AGE de 2 bits, latencia de acceso de 10 ciclos, 32 MSHR
DRAM	2 rangos, 8 bancos, 4 KB de tamaño de página, Double Data Rate (DDR3 1333 MHz). 92 ciclos de latencia de acceso bruta
Bus DRAM	2 canales a 667 MHz, cada uno con bus de 8 B, 4 ciclos de DRAM por línea, 16 ciclos de procesador por línea

TABLA 3: ESPECIFICACIONES DE LA JERARQUÍA DE MEMORIA

Aplicación	**L1**	**L2**	**LLC**	**Aplicación**	**L1**	**L2**	**LLC**	**Aplicación**	**L1**	**L2**	**LLC**
astar	7,5	1,1	0,7	gromacs	11,7	3	1,2	perlbench	10,2	1,8	0,8
bwaves	24,5	21,1	20,1	hmmer	3,3	2,4	0,2	povray	11,5	0,2	0,1
bzip2	8,4	3,9	0,9	h264ref	4,2	1,4	0,7	sjeng	6,9	0,8	0,5
cactusADM	20,8	11,4	4,9	lbm	65,4	38,6	36,7	soplex	8,9	7,1	3,1
calculix	8,5	4,3	1,5	leslie3d	40,4	23,2	17,9	sphinx3	18,8	14,3	11,7
dealII	1,6	0,5	0,3	libquantum	45,8	33,2	32,2	tonto	6,7	1,3	0,5
gamess	6,7	1	0,6	mcf	64,9	36	18,9	wrf	14,3	8,9	1,5
gcc	22	6,4	2,1	milc	24,6	23,5	22	xalancbmk	15,1	8,7	2,8
gemsFDTD	42,7	29,7	22,8	namd	1,7	0,2	0,2	zeusmp	32,3	8,7	7,2
gobmk	13,2	1,1	0,3	omnetpp	12,6	9,2	2,2				

TABLA 1: MPKI MEDIO PARA CADA NIVEL DE CACHE DEL SISTEMA BASE (SLLC EXCLUSIVA DE 8 MB CON TC-AGE)

VI. RESULTADOS

En esta sección se evalúa el mecanismo propuesto utilizando la metodología expuesta en la sección anterior. Se utilizan dos métricas de rendimiento: el número de instrucciones por ciclo normalizado al del sistema base (IPC normalizado) y la reducción en fallos por instrucción frente al sistema base. Los valores mostrados son la media de los resultados obtenidos para cada una de las 100 mezclas. Para cada mezcla, el IPC normalizado obtenido para una propuesta "PROP" se calcula como $\frac{\sum_t IPC_t^{PROP}}{\sum_t IPC_t^{TC-AGE}}$, donde IPC_t es el IPC obtenido para el procesador *t*. La reducción en número de fallos por instrucción se calcula como

$$1 - \frac{\sum_t F_t^{PROP}}{\sum_t F_t^{TC-AGE}} \cdot \frac{\sum_t I_t^{TC-AGE}}{\sum_t I_t^{PROP}}$$

, donde F_T es el número de fallos de SLLC medidos a lo largo de la ejecución para el procesador *t,* e I_T es el número de instrucciones ejecutadas para el procesador *t*.

Los primeros tres apartados evalúan la influencia en rendimiento y coste de la variación en el DR del tamaño del buffer, el tamaño de sector y el tamaño de la etiqueta almacenada. Se obtiene así una configuración balanceada entre coste y prestaciones. A continuación, se hace un análisis del funcionamiento del DR, explicando de qué manera reduce la tasa de fallos de la SLLC. Por último, se realiza una comparativa con otras propuestas.

A. Influencia del tamaño del buffer del DR

En esta sección estudiaremos cómo varían los resultados en función del tamaño del buffer del DR. La Figura 5 muestra el IPC y la tasa de fallos obtenidos de media en las 100 mezclas descritas en la Sección V, normalizados respecto del sistema base con TC-AGE, en función de la cobertura de expulsión de cada DR. El tamaño de sector del DR es de 1 bloque, y se almacenan etiquetas completas.

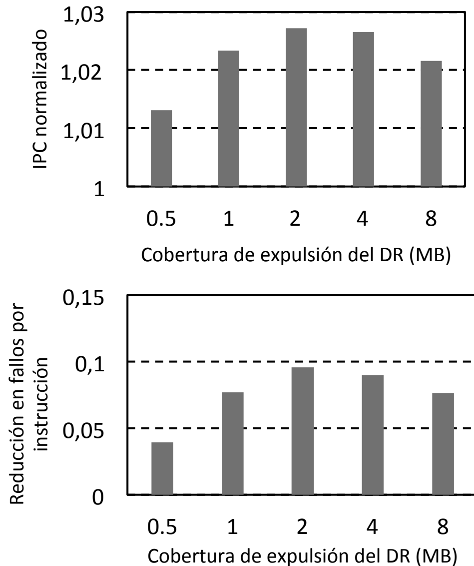

Figura 5: IPC normalizado (arriba) y reducción en fallos de SLLC por instrucción (abajo) frente al sistema base con TC-AGE, en función de la cobertura de expulsión del DR en cada procesador.

Al aumentar la cobertura del DR se puede hacer seguimiento de líneas que han sido expulsadas de la cache L2 hace más tiempo, es decir, detectar reusos más lejanos. La configuración óptima se consigue con una cobertura de expulsión de 2 MB, donde presenta un incremento del IPC de un 2,7%, y una reducción de los fallos por instrucción en la SLLC del 9,6%.

B. Influencia del tamaño de sector del DR

La Figura 6 muestra cómo varía el rendimiento cuando se incrementa el tamaño de sector en el buffer del DR. Dentro de una misma cobertura de expulsión del DR, al doblar el tamaño de sector se reduce a la mitad el número de conjuntos.

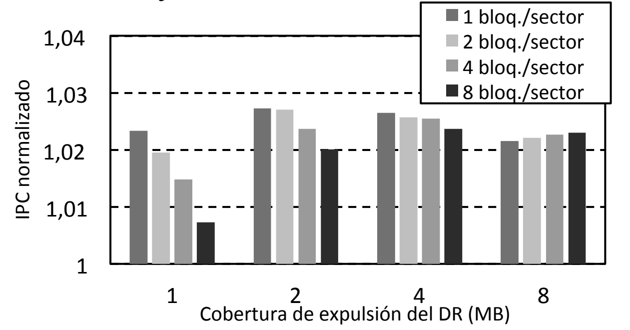

Figura 6: IPC normalizado frente al sistema base con TC-AGE, en función de la cobertura de expulsión del DR, y para diferentes tamaños de sector del DR.

Dentro de una misma cobertura, al incrementarse el tamaño de sector la capacidad efectiva es menor, puesto que en algún caso la falta de localidad espacial hace que alguno de los bloques del sector no sea referenciado. Esto hace que se detecte menos reuso, lo que produce una degradación del rendimiento para todos los tamaños de DR salvo para 8 MB, donde ya se detectan más bloques con reuso de los que caben en la SLLC.

A cambio, la cantidad de bits de memoria requeridos es menor. La Figura 7 muestra la cantidad de memoria requerida por cada DR en función de la cobertura de expulsión y del tamaño de sector, utilizando una etiqueta de 10 bits.

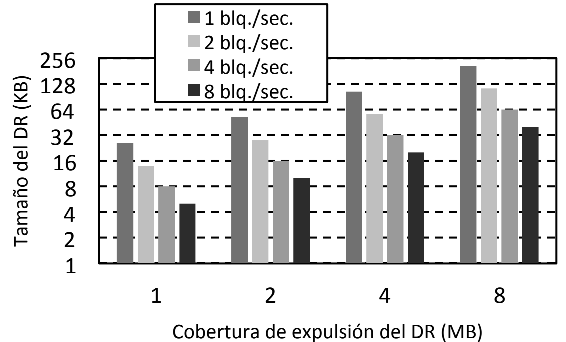

Figura 7: Tamaño del buffer de cada DR en KB, en función de la cobertura de expulsión del mismo, y para diferentes tamaños de sector del DR. El tamaño de sector está expresado en el número de bloques que sigue cada línea. Los valores están calculados considerando una etiqueta de 10 bits.

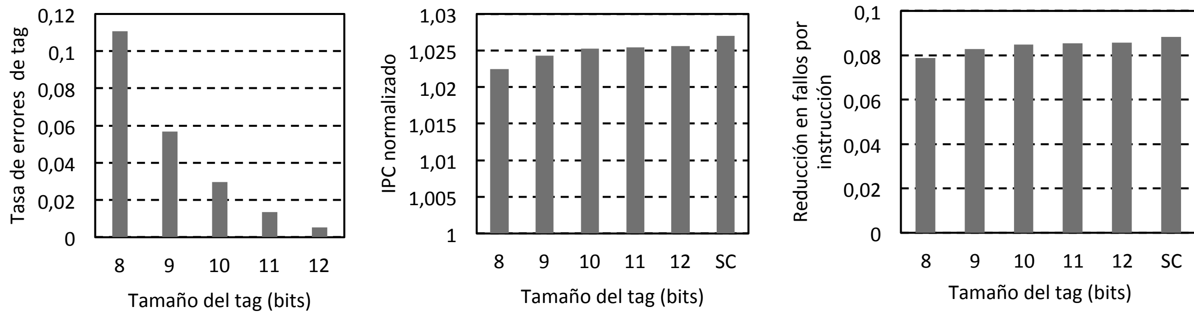

Figura 8: Izquierda: Tasa media de errores en la comprobación de la etiqueta debidos a la compresión de la misma. Centro: IPC normalizado frente al sistema base con TC-AGE. Derecha: Reducción en fallos de SLLC por instrucción frente al sistema base con TC-AGE. Incluye la etiqueta sin compresión ("SC").

Para una misma cobertura, el almacenamiento requerido disminuye al aumentar el tamaño de sector. Esto es debido a que el ahorro de espacio por la reducción del número de entradas es mayor que el incremento por añadir bits de presencia de bloque a la línea. La configuración con mejor rendimiento, con 2 MB de cobertura, precisa de un buffer de 48 KB en cada DR. Otras configuraciones presentan mejores relaciones entre prestaciones y coste, como por ejemplo aquella con 2 MB de cobertura y sectores de 2 bloques, con un rendimiento un 0,02% menor y un buffer de 28 KB, un 42% menor. Esta última es la configuración seleccionada para el resto de experimentos.

C. *Influencia del tamaño de la etiqueta del DR*

Para reducir la cantidad de memoria requerida, en el DR pueden almacenarse etiquetas comprimidas, como se ha explicado en la Sección III.D. La Figura 8 muestra a la izquierda, en función del tamaño de la etiqueta almacenada, la tasa media de errores en la comprobación de la etiqueta debidos a esta compresión. Estos errores son falsos positivos, en los cuales se detecta un falso reuso porque la etiqueta comprimida del sector recibido coincide con la de otro sector diferente, anteriormente registrado. El envío a la SLLC de bloques no reusados reduce la efectividad del mecanismo.

La Figura 8 muestra, en el centro y a la derecha, el IPC y la tasa de fallos normalizados obtenidos para la configuración de 2 MB de cobertura y un sector de 2 bloques, en función del tamaño de la etiqueta en el DR. La pérdida de rendimiento es muy baja para tamaños de etiqueta como el de 10 bits, donde se empeora un 0,16% el IPC y un 0,34% los fallos frente a la configuración sin compresión de la etiqueta. Ello justifica la compresión con el fin de reducir la cantidad de memoria requerida.

En adelante, salvo indicación de lo contrario, esta configuración con un DR de 2 MB de cobertura de expulsión, un tamaño de sector de 2 bloques, y 10 bits de etiqueta es la utilizada en los resultados mostrados.

D. *Análisis del funcionamiento del DR*

En esta sección analizamos cómo consigue el DR reducir la tasa de fallos de la SLLC. La Figura 9 muestra, para los programas de una mezcla de ejemplo, la distribución de los bloques recibidos desde el punto de vista del DR, en cinco categorías: (U) primer uso, (R) primer reuso, (MD) múltiple reuso detectado sólo por el DR, (MC) múltiple reuso detectado sólo porque el bloque proviene de la SLLC, y (MA) múltiple reuso detectado por ambos mecanismos. Una expulsión categorizada como U provoca el bypass del bloque, mientras que el resto envían el bloque para alojarlo en la SLLC. Las expulsiones de tipos U, R o MD indican que originalmente el bloque viene de la memoria, o sea, provienen de un fallo en la SLLC, mientras que los tipos MC y MA indican un acierto previo en la SLLC. El hardware propuesto del DR no puede distinguir entre los casos R y MD, pero se han separado en la gráfica para ejemplificar la complementariedad de los mecanismos de detección de reuso.

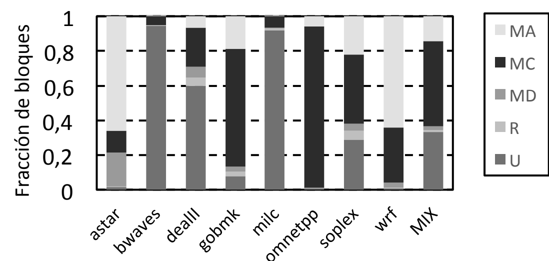

Figura 9: Fracción de expulsiones de bloques de cada programa de una mezcla ejemplo, categorizadas desde el punto de vista del DR según su tipología de reuso en: (U) primer uso, (R) primer reuso, (MD) múltiple reuso detectado sólo por el DR, (MC) múltiple reuso detectado sólo porque el bloque proviene de la SLLC, y (MA) múltiple reuso detectado por ambos mecanismos.

El nivel de bypass varía de un programa a otro, ajustándose bloque a bloque al patrón de reuso que éste muestra. En *bwaves* y *milc*, en más del 92% de las expulsiones se ha detectado un solo uso, y no se envían a la SLLC. Esto es coherente con las medidas mostradas en la Tabla 1, que indican que la SLLC apenas reduce el número de fallos por instrucción de estos programas. En el extremo opuesto, en *astar*, *omnetpp* y *wrf* menos del 3% de los bloques expulsados de las caches privadas no muestran reuso, por lo que hay escaso bypass. El resto de programas, *dealII*, *gobmk* y *soplex*, presentan valores intermedios, con niveles de bypass del 30%, 8% y 24% respectivamente.

La cantidad de bloques que se envían a la SLLC tras detectar el primer reuso (tipo R) varía entre un 0,2% de *omnetpp* y un 5% de *dealII*, con una media del 1,5%. Estos pocos bloques que muestran un primer reuso son accedidos después múltiples veces (tipos MD, MC y MA), siendo la proporción media de 45 detecciones de múltiple reuso por cada primer reuso que se identifica.

En ocasiones, el bloque ya ha sido reemplazado de la SLLC, y el DR lo inserta de nuevo (MD). Esto ocurre de media en un 4% de las veces que se detecta múltiple reuso.

La eliminación del envío de bloques de poca utilidad en *bwaves, milc, dealII* y *soplex,* va a permitir a la SLLC conservar mejor los bloques útiles de esos mismos programas, y los del resto de los programas de la mezcla. La Figura 10 muestra arriba la fracción de la SLLC ocupada por los bloques de cada programa, en el sistema base sin bypass y en el sistema con DR. Tanto *bwaves* como *milc* ocupan en este último mucho menos espacio en la SLLC, el cual se reparte entre el resto de programas. Como puede verse en la Figura 10 abajo, la buena selección de bloques permite a *bwaves* y *milc* mantener una tasa de fallos similar (un 0,4% peor en el caso de *bwaves*). En el resto de programas, a la selección de bloques se une una expulsión menos frecuente de los mismos y una mayor cantidad de espacio disponible, por lo que su número de fallos por instrucción en la SLLC mejora entre un 4,7% de *dealII* y un 77,3% de *omnetpp*. La reducción en fallos para toda la mezcla es del 11,8%, siendo el IPC normalizado de 1,030. La mezcla ocupa la posición 32 dentro de las 100 si se ordenan de mayor a menor incremento de IPC.

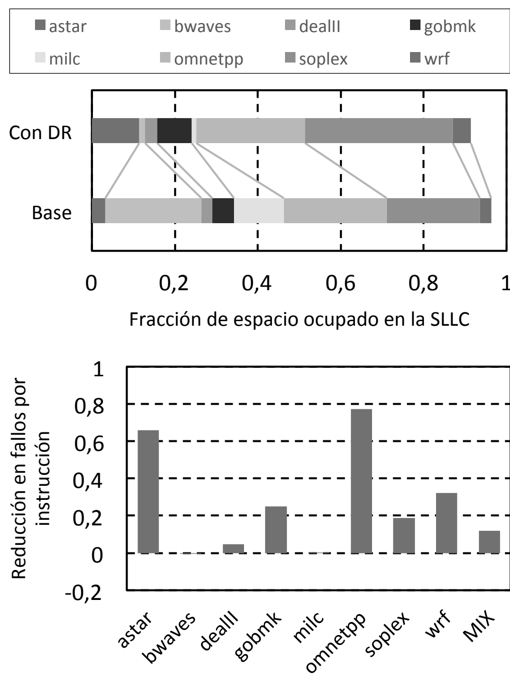

Figura 10: Arriba: Fracción media del espacio ocupado por los bloques de cada programa de la mezcla de ejemplo, en el sistema base y en el sistema con DR. Se toman datos cada millón de ciclos de ejecución, y se calcula la media. Abajo: Reducción en fallos por instrucción en el sistema con DR, normalizada a la del sistema base con TC-AGE.

E. Comparativa con otras propuestas

En esta sección se compara el rendimiento de nuestro mecanismo con otras dos propuestas recientes de políticas de inserción y reemplazo basadas en bypass.

Comparación con CHAR: La política CHAR [6] (acrónimo de "cache hierarchy-aware replacement") para SLLC exclusivas es una propuesta de selección de contenidos que basa la decisión de bypass en el patrón de acceso que han tenido los bloques en todos los niveles de la jerarquía de memoria. La decisión se toma sobre los bloques expulsados de las caches L2. Los bloques son categorizados en cuatro clases. Para la clase que muestra reuso a nivel de SLLC nunca se realiza bypass. Para las otras tres clases, existe una lógica que decide si es provechoso realizar el bypass para el conjunto de la clase o no, actuando en consecuencia. Para ello, se monitoriza a través de contadores la tasa de acierto de algunos conjuntos de la SLLC para los que se mantiene la política base TC-AGE, y se compara con la de otros conjuntos que implementan CHAR.

La Figura 11 muestra el IPC y la reducción en fallos por instrucción, ambos frente al sistema base con TC-AGE, obtenidos para sistemas con DR y CHAR, con 8 MB de SLLC. Como puede apreciarse, el mecanismo del DR mejora en media a CHAR tanto en reducción de fallos (8,5% frente a 5,3%) como en IPC normalizado (2,5% frente a 2,0%). Los resultados en cuanto a reducción de fallos de CHAR son coherentes con la publicación original, pero el IPC normalizado es menor que en dicho artículo. Ello puede deberse a la diferente metodología, ya que los modelos de procesador y jerarquía de memoria utilizados son diferentes en ambos trabajos.

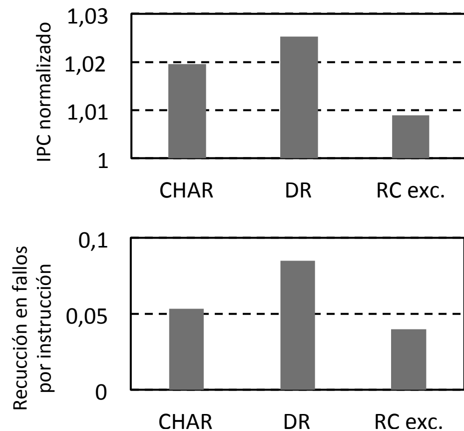

Figura 11: IPC normalizado (arriba) y reducción en fallos de SLLC por instrucción (abajo) frente al sistema base con TC-AGE, para un sistema con un DR de 2 MB de cobertura de expulsión y un tamaño de sector de 2 bloques, CHAR (en exclusión) y una Reuse Cache exclusiva RC-32/8 con NRR en la matriz de etiquetas y TC-AGE en la matriz de datos.

Comparación con una Reuse Cache: La Reuse Cache [11] es una SLLC cuyas estructuras de etiquetas y datos están desacopladas, y que almacena sólo los datos de las líneas que han mostrado reuso. Para realizar la comparación en igualdad de condiciones, se ha desarrollado una versión de la Reuse Cache donde la matriz de datos funciona en exclusión con las L2 privadas. Esta versión funciona como sigue: Cualquier bloque enviado a una cache L2 incluye un bit que indica si debe ser devuelto a la SLLC al ser expulsado (bypass o no bypass). En un primer acceso, se envía el bloque desde memoria a la cache L2 con indicación de bypass, y se inserta la etiqueta en la matriz de etiquetas de la SLLC. Esto permite detectar un reuso posterior. En un segundo acceso, con acierto en la matriz de etiquetas de la SLLC y fallo en la de datos, se envía el bloque desde memoria a la L2 privada con indicación de no bypass, sin almacenarlo en la matriz de datos. La etiqueta se mantiene en la matriz de etiquetas. Cuando el bloque se expulsa de la L2, se almacena en la matriz de datos de la

SLLC. Posteriores accesos, con acierto tanto en la matriz de datos como en la de etiquetas, envían el bloque a la L2 privada con indicación de no bypass, y lo expulsan de la matriz de datos de la SLLC.

En este trabajo utilizamos para la comparación una Reuse Cache Exclusiva con 32MB equivalentes de etiquetas y 8MB de datos. Esta relación entre etiquetas y datos es la que mejor rendimiento ofrece de entre las que disponen de 8MB de datos, tanto en el artículo original como en nuestras simulaciones. En la matriz de datos se emplea TC-AGE como política de reemplazo.

Como puede apreciarse en la Figura 11, el mecanismo del DR también mejora en media a la Reuse Cache Exclusiva tanto en reducción de fallos (8,5% frente a 4,0%) como en IPC normalizado (2,5% frente a 0,9%).

VII. CONCLUSIONES

En un sistema multiprocesador on-chip, el flujo de referencias que llega a la SLLC muestra poca localidad temporal. Sin embargo, muestra localidad de reuso. Esto provoca que una gestión convencional de la cache, basada en la localidad temporal, sea ineficiente. Existe un número importante de propuestas que tratan este problema para caches inclusivas, pero pocas que se centran en caches exclusivas.

En este trabajo se propone un nuevo mecanismo de selección de contenidos para caches exclusivas que aprovecha la localidad de reuso que presentan los accesos a la SLLC. Consiste en incluir un Detector de Reuso entre cada cache L2 y la SLLC, que detecta qué bloques expulsados de las L2 no han demostrado reuso y evita que sean insertados en la SLLC, realizando *bypass* de los mismos.

Se evalúa esta propuesta simulando un sistema con 8 procesadores en un chip que ejecuta una serie de cargas multiprogramadas. Configurado adecuadamente, el Detector de Reuso evita la inserción de bloques poco útiles en la SLLC, facilitando que se mantengan los más reusados. Los resultados muestran que ello permite incrementar el rendimiento, por encima de otras propuestas recientes como CHAR o la Reuse Cache.

AGRADECIMIENTOS

Este trabajo ha sido financiado en parte por los proyectos TIN2013-46957-C2-1-P, TIN2012-34557 y Consolider NoE TIN2014-52608-REDC (Gobierno de España), gaZ: grupo de investigación T48 (Gobierno de Aragón y Fondo Social Europeo), y la red de excelencia europea HiPEAC-3 (European FET FP7/ICT 287759).

REFERENCIAS

[1] S. Khan, Y. Tian y D. A. Jiménez, «Sampling Dead Block Prediction for Last-Level Caches,» Proc. of 43rd Ann. Int. Symp. on Microarchitecture, pp. 175-186, 2010.

[2] M. Qureshi, A. Jaleel, Y. Patt, S. Steely y J. Emer, «Adaptive insertion policies for high performance caching,» Proc. of 34th Ann. Int. Symp. on Computer Architecture, pp. 381-391, 2007.

[3] J. Gaur, M. Chaudhuri y S. Subramoney, «Bypass and Insertion Algorithms for Exclusive Last-level Caches,» Proc. of 38th Int. Symp. on Computer Architecture, pp. 81-92, Junio 2011.

[4] A. Jaleel, E. Borch, M. Bhandaru, S. C. Steely Jr. y J. Emer, «Achieving Non-Inclusive Cache Performance with Inclusive Caches. Temporal Locality Aware (TLA) Cache Management Policies,» Proc. of the 2010 43rd Ann. Int. Symp. on Microarchitecture, pp. 151-162.

[5] A. Jaleel, K. B. Theobald, S. C. Steely Jr. y J. Emer, «High Performance Cache Replacement Using Re-Reference Interval Prediction (RRIP),» Proc. of 37th Int. Symp. on Computer Architecture, pp. 60-71, Junio 2010.

[6] M. Chaudhuri, J. Gaur, N. Bashyam, S. Subramoney y J. Nuzman, «Introducing hierarchy-awareness in replacement and bypass algorithms for last-level caches,» Proc. of the 21st Int. conference on Parallel architectures and compilation techniques, pp. 293-304, 2012.

[7] N. P. Jouppi y S. J. E. Wilton, «Tradeoffs in Two-Level On-Chip Caching,» Proc. the 21st Ann. Int. Symp. on Computer Architecture, pp. 34-45, 1994.

[8] P. Conway, N. Kalyanasundharam, G. Donley, K. Lepak y B. Hughes, «Cache hierarchy and memory subsystem of the AMD Opteron processor,» IEEE micro, nº 30(2), pp. 16-29, 2010.

[9] P. Lotfi-Kamran, B. Grot, M. Ferdman, S. Volos, O. Kocberber, J. Picorel, A. Adileh, D. Jevdjic, S. Idgunji, E. Ozer y B. Falsafi, «Scale-Out Processors,» ACM SIGARCH Computer Architecture News, vol. 40, nº 3, pp. 500-511, 2012.

[10] J. Albericio, P. Ibáñez, V. Viñals y J. M. Llabería, «Exploiting reuse locality on inclusive shared last-level caches,» ACM Trans. on Architecture and Code Optimization, vol. 9, nº 4, p. 38, 2013.

[11] J. Albericio, P. Ibáñez, V. Viñals y J. Llabería, «The reuse cache: downsizing the shared last-level cache,» Proc. of the 46th Ann. Int. Symp. on Microarchitecture, pp. 310-321, 2013.

[12] R. Karedla, J. Love y B. Wherry, «Caching strategies to improve disk system performance,» Computer, nº 27(3), pp. 38-46, 1994.

[13] H. Gao y C. Wilkerson, «A dueling segmented LRU replacement algorithm with adaptive bypassing,» Proc. of the 1st JILP Workshop on Computer Architecture Competitions, 2010.

[14] S. Khan, Z. Wang y D. Jimenez, «Decoupled dynamic cache segmentation,» Proc. IEEE 18th Int. Symp. High Performance Computer Architecture HPCA, pp. 1-12, 2012.

[15] C. J. Wu, A. Jaleel, W. Hasenplaugh, M. Martonosi, S. C. Steely Jr y J. Emer, «SHiP: signature-based hit predictor for high performance caching,» Proc. of the 44th Ann. Int. Symp. on Microarchitecture, pp. 430-441, 2011.

[16] L. Li, D. Tong, Z. Xie, J. Lu y X. Cheng, «Optimal bypass monitor for high performance last-level caches,» Proc. of the 21st Int. conference on parallel architectures and compilation techniques, pp. 315-324, 2012.

[17] V. Seshadri, O. Mutlu, M. A. Kozuch y T. C. Mowry, «The evicted-address filter: a unified mechanism to address both cache pollution and thrashing,» Proc. of the 21st Int. conference on Parallel architectures and compilation techniques, pp. 355-366, 2012.

[18] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt y B. Werner, «Simics: A full system simulation platform,» Computer, nº 35(2), pp. 50-58, 2002.

[19] M. Martin, D. Sorin, B. Beckmann, M. Marty, M. Xu, A. Alameldeen, K. Moore, M. Hill y D. Wood, «Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset,» Computer Architecture News, nº 33(4), pp. 92-99, 2005.

[20] J. L. Henning, «SPEC CPU2006 benchmark descriptions,» ACM SIGARCH Computer Architecture News, vol. 34, nº 4, pp. 1-17, 2006.

[21] A. N. Eden y T. Mudge, «The YAGS branch prediction scheme,» Proc. of the 31st Ann. ACM/IEEE Int. Symp. on Microarchitecture, pp. 69-77, 1998.

[22] S. Thoziyoor, J. H. Ahn, M. Monchiero, J. B. Brockman y N. P. Jouppi, «A comprehensive memory modeling tool and its application to the design and analysis of future memory hierarchies,» 35th Int. Symp. on Computer Architecture, pp. 51-62, 2008.

[23] L. Kohn, G. Maturana, M. Tremblay, A. Prabhu y G. Zyner, «The visual instruction set (VIS) in UltraSPARC,» Computer Conference, IEEE Int., pp. 462-462, 1995.