PS-Cache: Un diseño energéticamente eficiente para caches en CMPs

Joan J. Valls [1], Alberto Ros [2], Julio Sahuquillo[1], María E. Gómez[1]

Resumen—**El consumo de energía es una de las mayores preocupaciones en los diseños actuales de multiprocesadores en chip de altas prestaciones. Esta problemática se acrecenta conforme aumenta el número de núcleos. Una fracción importante del consumo total es consumido por las estructuras cache en el chip, así pues, se ha hecho mucho enfásis en la investigación para reducir el consumo de energía en las mismas. Para incrementar las prestaciones, las caches en el chip disponen de un grado de asociatividad elevado. Consecuentemente, acceder concurrentemente a todas las vías de un conjunto es costoso en concepto de energía. Este paper presenta la arquitectura PS-Cache, un diseño de cache energéticamente eficiente que reduce el número de vías accedidas sin perjudicar las prestaciones. La PS-Cache utiliza la información de estado privado-compartido del bloque referenciado para acceder exclusivamente a aquellas vías que contengan bloques en ese mismo estado. Los resultados experimentales muestran que, de media, la arquitectura PS-Cache puede reducir el consumo dinámico en las caches L1 y L2 en un $22\,\%$ y $40\,\%$, respectivamente.**

Palabras clave— **Multiprocesadores en chip; Memorias cache; Consumo de energía; Aplicaciones multihilo**

I. Introducción

CONFORME los recursos de silicio aumentan, el número de núcleos en las generaciones de multiprocesadores en chip (CMP) se incremente rápidamente. Estos CMP, normalmente implementan un modelo de programación de memoria compartida y aceleran los accesos a memoria utilizando uno o más niveles de caches privadas por núcleo. Esto se hace transparente al software gracias a al empleo de protocolos de coherencia de cache. Se implementan dos tipos de protocolos de coherencia en los CMPs modernos, protocolos basados en directorio y protocolos snoopy, cada uno orientado a sistemas de distintas escalas.

Las caches en los CMPs actuales están organizados en una jerarquía de memoria de dos o tres niveles, ocupando un prscentaje significativo del área del CMP [1] y consumiendo una parte importante del consumo total. Con respecto al consumo de energía dinámico, predomina en la cache de primer nivel ya que este nivel es el más accedido en los procesadores modernos, mientras que a las caches de último nivel (LLC), e.g. L3, habitualmente se accede poco.

Durante un acceso a la cache en los microprocesadores de altas prestaciones, todas las vías del conjunto correspondiente se acceden concurrentemente, así pues, el grado de asociatividad define el número de etiquetas que se miran en paralelo en cada acceso. Se incluye un comparador por vía y se realizan tantas comparaciones de etiquetas como número de vías haya. Por tanto, la energía dinámica disipada por acceso aumenta junto la asociatividad de la cache.

Muchos estudios de reducción de energía en caches se han centrado en procesadores monolíticos (e.g., Cache Decay [2] o Drowsy Caches [3]). No obstante, ya que estos esquemas no se aplican directamente a CMPs o pueden ser mejorados, trabajos más recientes se han concentrado en los ahorros de energía en CMPs ejecutando cargas paralelas. A diferencia de las cargas multiprogramadas, los bloques accedidos en estas cargas se pueden clasificar en dos categorías diferentes: bloques privados y bloques compartidos. Los primeros son accedidos únicamente por un solo núcleo, mientras que los otros son accedidos por varios. La figura 1 muestra una posible distribución de bloques privados y compartidos en una cache con conjuntos de 8 vías. Algunos bloques pueden estar en un estado inválido, así que no están clasificados ni como privado ni como compartido.

Algunos trabajos de investigación recientes se han centrado en utilizar el comportamiento de los accesos a bloques de los dos tipos para mejorar las prestaciones [4–7] o el consumo de energía [8–10] de diferentes componentes de un CMP. En particular, en [10], los autores particionan el último nivel de cache (LLC) en vías compartidas (situadas a la izquierda de la cache) y vías privadas (situadas a la derecha de la cache) añadiendo una lógica compleja. De esta forma, el número de vías que son accecidas en un acceso a la cache se reduce, ahorrando consumo de energía dinámico. Las LLC se acceden tradicionalmente de forma secuencial, es decir, primero se miran las etiquetas y, en caso de acierto, se accede a la vía correspondiente de los datos. Ya que estas caches se acceden mucho menos que las caches L1, el consumo de energía dinámico no es tan crítico como en las caches L1.

A diferencia de las LLC, tanto las etiquetes como los datos tienen que ser accedidos al mismo tiempo en las caches de primer nivel. Esto es debido a que la resolución de un acceso debe ser rápida ya que éstas tienen un impacto considerable sobre las prestaciones generales del procesor. En resumen, las caches L1 se acceden con frecuencia, y se accede en paralelo tanto a etiquetas como a datos. En consecuencia, el aumento del consumo de energía dinámico en estas estructuras se convierte en un

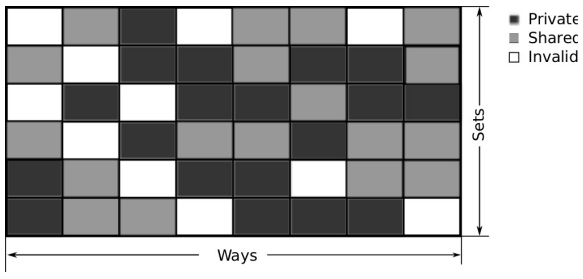


Fig. 1. Ejemplo de distribución en una cache con cjts. de 8 vías.

[1]DISCA, Universitat Politècnica de València, e-mails: joavalmo@fiv.upv.es,{jsahuqui,megomez}@disca.upv.es
[2]DITEC, Universidad de Murcia, e-mail: aros@ditec.um.es

problema de diseño. Este problema se agrava en CMPs, puesto que se accede a las caches tanto desde el lado del procesador, como desde la red de interconexión (NoC) a causa de peticiones de coherencia, aumentando así aún más los accesos a la cache.

En este trabajo proponemos un mecanismo para caches en CMPs, que puede aplicarse a cualquier nivel de la jerarquía de memoria, con la intención de reducir el consumo de energía dinámico en estas estructuras. Se basa en clasificar los bloques en cache durante la ejecución en compartidos o privados, y, en el momento de acceder a la cache, solo mirar aquellas vías que mantengan un bloque que comparta tipo con el bloque solicitado. Hasta donde sabemos, no hay ningún trabajo previo sobre este comportamiento en caches de primer nivel en sistemas CMP a pesar de ser el nivel en el que más energía dinámica se consume. El esquema propuesto se puede implementar con una complejidad hardware mínima. El esquema presenta importantes características de diseño: i) la clasificación de bloques se realiza con una complejidad despreciable, tan solo es necesario añadir un bit en cada entrada de la TLB; ii) los array de etiquetas y datos se pueden acceder en paralelo, así que no surgirá ninguna degradación en las prestaciones, lo cual es una de las mayores preocupaciones en caches L1; y iii) a diferencia de otras propuestas como [10] no es necesario realizar ningún alineamiento de las vías.

Los resultados experimentales muestran que la arquitectura PS-Cache puede reducir el consumo dinámico de energía en alrededor de un $22\,\%$ tanto en procolos snoopy como en protocolos de directorio y que puede reducir el consumo de la L2 en un $40\,\%$. Estas reducciones se consiguen sin perjudicar a las prestaciones.

El resto del trabajo se organiza de la siguiente forma:

La sección II describe las razones principales que motivan esta investigación. La sección III comenta el estado del arte. La sección IV presenta la propuesta. La sección V describe el entorno de simulación. La sección VI analiza el comportamiento de ambos tipos de bloque y muestra resultados de energía. Finalmente, la sección VII muestra unas conclusiones finales.

II. Motivación

Hay dos principales razones que impulsan este trabajo. Primero, las instrucciones de referencia a memoria representan un porcentaje significativo de las instrucciones ejecutadas. La segunda, al ejecutar cargas multihilo, además de los accesos a la cache local, se accede a otras caches (*e.g.*, caches remotas) por razones de coherencia.

El primer punto indica que se accede con frecuencia a las caches de memoria. Lanzamos experimentos para cuantificar el porcentaje de instrucciones de referencia a memoria en las aplicaciones estudiadas. La figura 2 muestra el porcentaje de instrucciones de referencia a memoria en cada benchmark ejecutado en un sistema CMP de 16 cores[1]. Este valor es aproximadamente el mismo, sobre un $20\,\%$ en todos los benchmarks.

El segundo punto indica que al ejecutar cargas paralelas, el número de accesos a cache se incrementa en

[1]El entorno experimental, los parámetros del sistema, protocolos y jerarquía de memoria se describen en la sección V.

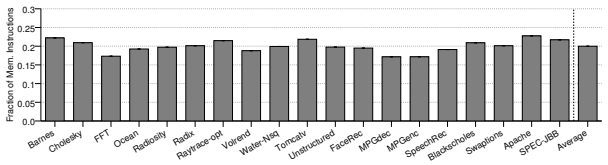


Fig. 2. Fracción de instrucciones de memoria.

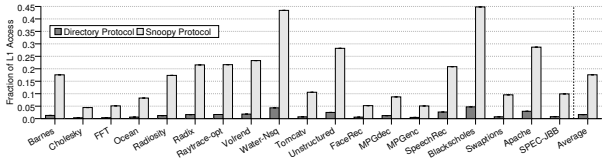


Fig. 3. Búsquedas por coherencia en L1

comparación con procesadores monolíticos a causa de las peticiones de coherencia lanzadas por otros núcleos. En otras palabras, la cache no se accede solo desde el procesador, sino también desde la red de interconexión (NoC), incrementandose así el consumo de energía dinámico. En este contexto, el número de accesos proveniente de la NoC depende del tipo de protocolo de coherencia subyacente: protocolo snoopy o de directorio.

Los protocolos snoopy se basan en mensajes de difusión a todos los núcleos, lo que requieres un elevado ancho de banda y consumo en la red, pero también por parte de las caches, ya que todas las presentes en el sistema se deben de acceder tras una petición de coherencia. En consecuencia, tan solo son apropiados para sistemas de escala reducida [8, 11, 12]. Los protocolos de directorio mantienen la información de coherencia de los bloques almacenados en una estructura de directorio localizada entre los niveles de cache privada y compartida [5, 13, 14]. Esto permite que el procesador identifique con facilidad réplicas de un bloque, minimizando así la comunicación de coherencia. Las peticiones de coherencia se envían únicamente a núcleos que mantengan una copia, así que tan solo se mira en un subconjunto de las caches. Esto hace que sean más asequibles para CMPs de gran escala, ya que reducen la energía y el ancho de banda en comparación con los protocolos snoopy.

La figura 3 muestra la fracción de los accesos a la cache proveniente desde la NoC para ambos tipos de protocolos en nuestro sistema de 16 núcleos. Como se puede observar, este valor es significativo en los protocolos snoopy y representa una quinta parte de los accesos totales. En algunas cargas este valor puede alcanzar el $45\,\%$. En contraste, este valor muestra poco interés en directorios de protocolo.

La discusión previa ilustra la importancia de reducir el consumo de energía dinámico en las caches de sistemas CMP, y en procotolos snoopy también, donde se realizan más peticiones por parte de los controladores de cache. Para afrontar este problema, este trabajo propone una solución arquitectural con el objetivo de hacer uso del comportamiento exhibido en las cargas paralelas.

La propuesta ahorra una cantidad de energía significativa mirando únicamente aquellas vías que contienen blo-

ques del mismo tipo (compartido o privado) que el bloque solicitado tanto por el procesador como por un acceso de coherencia.

III. Estado del arte

Este trabajo presenta un diseño energéticamente eficiente de cache que hace uso de la clasificación de bloques en privados y compartidos. De esta forma, esta sección primero repasa el estado del arte en diseños de cache de eficiencia energética y, luego, se revisa otras optimizaciones basadas en la clasificación de privado-compartido.

A. Diseños de cache de eficiencia energética

El consumo de la cache es el resultado tanto del consumo de fuga (o estático) como del dinámico. Con respecto a reducciones del estático, Powell *et al.* [15] propuso una técnica Gated-Vdd que pretende reducir el consumo de fuga de las caches de instrucciones reconfigurando y desconectando aquellas líneas que no están en uso. Kaxiras *et al.* [2] proponen Cache Decay, que reduce el consumo de fuga de las caches de procesador al desconectar aquellas líneas que se predicen como muertas, i.e., que no han sido referenciadas por el procesador antes de haber sido expulsadas. Alternativamente, Flautner *et al.* [3] se vale del hecho de que en un periodo de tiempo particular tan solo se accede a un subconjunto de las líneas para proponer Drowsy Caches. A diferencia de las anteriores propuestas, el voltaje se reduce, pero no se elimina, para aquellas líneas que no están siendo accedidas. De esta forma, el contenido de la línea de cache no se borra.

Mientras que las técnicas de corriente de fuga se centran en reducir (o cortar) el voltaje, las técnicas de energía dinámica intentan minimizar el número de datos leídos y escritos en cada acceso de cache. Por ejemplo, Albonesi [16] propone Selective Cache Ways, un diseño de cache que puede activar tan solo un subconjunto de las vías de la cache cuando la actividad no es alta. La predicción de vías ya fue previamente propuesta por Calder *et al.* [17] para reducir el tiempo de acceso en caches asociativas por conjuntos. Ghosh et al. [18] popone Way Guard, un mecanismo para grandes caches asociativas por conjunto que utilizan bloom filters para reducir la energía dinámica al evitar mirar en aquellas líneas que, según la información del bloom filter, no contienen los datos solicitados.

Otras técnicas se centran en reducir ambos tipos de consumo, por ejemplo, al reducir el área las etiquetas en la cache, como en TLB Index-Based Tagging [19], o realizando particionamiento en tiempo de ejecución, como en el esquema Cooperative Caching [10] or en ReCaC [20].

Nuestra propuesta permite la reducción de consumo estático y dinámico sin la necesidad de llevar a cabo ninguna reconfiguración.

B. Optimizaciones privado-compartido

La detección de datos privados y compartidos también se puede usar para optimizar prestaciones y energía. Kim *et al.* [8] detecta el grado de compartición de las páginas de memoria para reducir la fracción de snoops en un protocolo de tokens. De esta forma, puede sustituir los mensajes de difusión por otro de multicast, reduciendo el consumo de la interconexión. El R-NUCA de Hardavellas *et al.* [4] detecta páginas privadas y de sólo lectura y las mapea eficientemente en una cache NUCA distribuida. Al mapear páginas privadas al banco NUCA más cercano al núcleo que la accede, la latencia de acceso se reduce, al igual que el tráfico generado, lo que impactará en la energía consumida por la red. Cuesta *et al.* [5] desactiva la coherencia de las páginas privadas, evitando así mantener su información en las caches de directorio. Esto permite la reducción del tamaño del directorio hasta ocho veces manteniendo las mismas prestaciones. Las reducciones en el área del directorio también conllevan reducciones en el consumo tanto estáticas como dinámicas.

Algunos trabajos previos se apoyan en el compilador y/o en el memory allocator para clasificar las páginas de memoria para o bien elimianr la coherencia para páginas privadas [21] o mejorar la localización de datos [22, 23]. En [22], el análisis de propiedad de datos de regiones de memoria se realiza en tiempo de compilación. Esta información se transfiere a la tabla de páginas modificando el comportamiento del memory allocator. Esta propuesta se mejora en [23] considerando una nueva clase de datos, parcialmente privados, que son mapeados a la cache NUCA según una política first-touch. En [21], los datos privados no se almacenan en la LLC para así evitar el cache thrashing de los bloques privados. A diferencia de nuestra propuesta, estos trabajos marcan de forma estática los datos como privado bien por el memory allocator o en tiempo de compilación, cuando la privacidad de algunos datos no se puede garantizar.

SWEL [6] es un novedoso protocolo de coherencia hardware que usa la clasificación privado-compartido de los bloques en el directorio para almacenar bloques compartidos de lectura y escritura únicamente en la LLC compartida, evitando así la necesidad de un mantenimiento de coherencia. El principal inconveniente de esta propuesta es la penalización en la latencia al acceder a estos bloques, que deben ser servidos por la LLC. POPS [7] separa la información de datos y coherencia en la LLC compartida para reducir la latencia de acceso a esta información y para mejorar la capacidad de la NUCA agregada. También usa una clasificación privado-compartido de directorio (con la ayuda de una tabla de predicciónes). Spatio-temporal Coherence Tracking [24] también clasifica datos privados y compartidos en el directorio, teniendo en cuenta datos temporalmente privados. Intenta encontrar grandes regiones privadas y fusionarlas en el directorio para ahorrar espacio. Finalmente, Ros y Kaxiras [9] proponen VIPS, un protocolo de coherencia de complejidad eficiente que utiliza caches de write-back para datos privados por razones de eficiencia y caches write-through para datos compartidos para simplicidad del protocolo.

IV. Propuesta

El objetivo principal de nuestra propuesta es utilizar la clasificación entre bloques privados y compartidos para diseñar una arquitectura cache energéticamente eficiente que reduce el número de vías que se miran en cada acceso a la cache. En vez de acceder a todas las vías de un conjunto, tan solo se hace en un subconjunto de ellas.

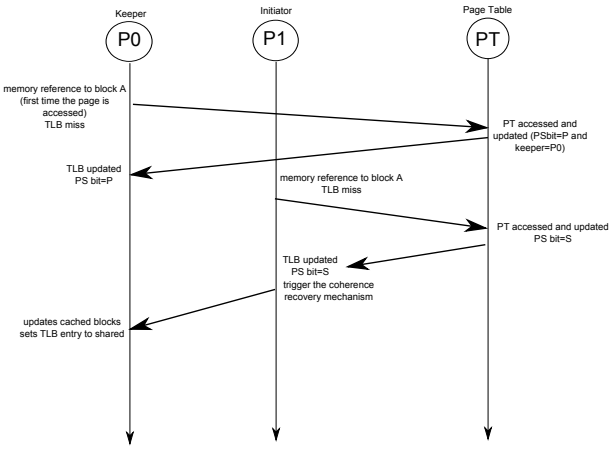


Fig. 4. Flujo del mecanismo de clasificación. P0 y P1 son procesadores y PT es la tabla de páginas en memoria principal.

Para que la propuesta pueda funcionar necesitamos i) mantener los bloques clasificados como privados o compartidos en la cache, y ii) un mecanismo de clasificación privado-compartido que nos indique el tipo del bloque a buscar. Los bloques se clasifican usando un bit (el bit PS) añadido en cada línea de la cache. Este bit indica el tipo de bloque almacenado. Adicionalmente, aunque nuestra propuesta puede trabajar con cualquier mecanismo de clasificación privado-compartido, en este trabajo asumimos el mecanismo utilizado en [5] y previamente propuesto en [4] para saber el tipo del bloque accedido antes de acceder a los arrays de etiquetas y datos, y acceder, de esta forma, únicamente a aquellas vías cuyo tipo (el valor del bit PS) coincidan con él. En otras palabras, solo se acceden aquellas vías del conjunto cuyo tipo coincida con el bit de la TLB de la página del bloque.

A. *Mecanismo de clasificación de páginas PS*

El mecanismo de clasificación está basado en la asistencia del OS, así pues, la clasificación se realiza con una granularidad de página. Esto significa que todos los bloques de la misma página se clasifican con el mismo tipo. La información de compartición se almacena tanto en la tabla de páginas como en la TLB que contiene la traducción de las páginas más recientemente referenciadas. La información de compartición está formada por el bit PS y el id del *keeper*, que es el primer núcleo que solicitó la traducción de la página. Esta información se almacena en la tabla de páginas, mientras la TLB tan solo almacena el bit PS. Tras una referencia a memoria, el núcleo obtiene el tipo del bloque de la referencia de la TLB cuando es accedida con la intención de obtener la traducción de la dirección. En un fallo de TLB, se accede a la tabla de páginas (como se suele hacer), pero el mecanismo de clasificación también actualiza la información en la tabla de páginas y en la TLB del núcleo.

La figura 4 muestra un ejemplo de como funciona el mecanismo de clasificación. El primer fallo (sufrido por P0 en el ejemplo) inicializa el estado de la página como privado y en el campo del keeper se coloca P0. La página se marca como privada en la TLB de P0. En fallo posteriores, si la página se encuentra como privada (lo que sucede en el segundo acceso por P1 en el ejemplo), es necesario comparar el campo de keeper (P0) con el del núcleo haciendo la petición de acceso (P1) para comprobar si la página pasa a ser compartida o si, en caso contrario, el que está solicitando la traducción es el mismo núcleo que la primera vez. Si la página tiene que convertirse compartida, la entrada de la tabla de páginas (PT) se actualiza y un mecanismo de recuperación de coherencia se activa para mantener la coherencia entre la tabla de páginas y el keeper de la TLB y los bits PS en las caches.

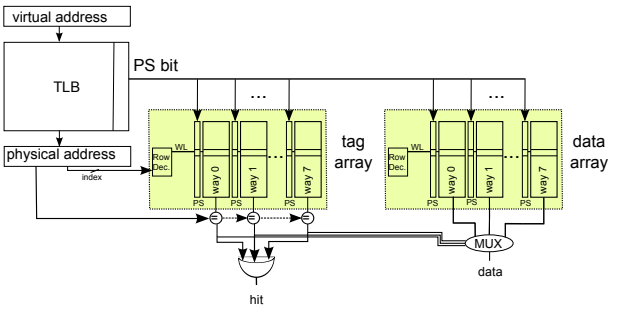


Fig. 5. La arquitectura PS-Cache para caches L1.

El mecanismo de recuperación de coherencia tiene que asegurarse de que todos los bits PS de lo bloques cacheados de la página así como en las TLBs matengan el mismo tipo que de su entrada en la tabla de páginas. Para eso, el núcleo solicitante lanza una *petición de recuperación* al keeper de la página (obtenido de la entrada de la tabla de páginas). Tras la llegada de esa petición, el keeper actualiza tanto el bit PS en la entrada correspondiente en la TLB como de los bloques cacheados pertenecientes a la página. Hay que remarcar que este proceso solo es necesario durante una transición de privado a compartido. De esta forma, mentenemos el bit PS de todos los bloques en cache coherentes con el estado de la página en las TLBs y en la tabla de páginas. Más detalles del mecanismo se pueden encontrar en [5].

Tras solucionar el fallo de la TLB, el bit PS se almacena junto la traducción de la página en la TLB del núcleo que realizó la petición y se usa este bit PS para comprobar el tipo del bloque solicitado (privado o compartido). Como se ha comentado previamente, el bit PS permite discernir el grupo de vías en el que el bloque solicitado puede estar, y, consecucuentemente solo acceder a ellas.

Aunque la clasificación privado-compartido empleada en este trabajo se realiza accediendo a la tabla de páginas en cada fallo de TLB [4, 5], la PS-Cache también puede funcionar con mecanismos de clasificación que utilizan transferencias entre TLBs [25], lo que puede mejorar las prestaciones generales del sistema.

B. *La arquitectura PS-Cache*

Durante la ejecución de una instrucción de referencia a memoria, el controlador de cache mira primero en la TLB para obtener la dirección física[2]. Como se comentaba anteriormente, la TLB incluye un bit por entrada, el bit PS (Private-Shared), que indica como ha sido clasificada la página. El valor de este bit se lee de la entrada de la TLB junto con la dirección física. Con esta información,

[2]Si ocurre un fallo de TLB, tras resolver el fallo la entrada correspondiente está en la TLB.

el controlador de la cache procede a buscar el bloque en la cache. La información de la traducción de la TLB se usa para comparar las etiquetas del conjunto correspondiente, pero como novedad, la propuesta usa el bit PS para evitar alguno de los accesos. El mecanismo se puede aplicar a cualquier nivel de cache. Por razones ilustrativas, la figura 5 muestra la arquitectura cache propuesta para la cache L1.

La diferencia clave es que la PS-Cache solo accede a aquellas vías que compartan el mismo tipo que el indicado por la TLB, eliminando así la energía consumida por el acceso a las demás vías. Como se puede observar, cada línea de la cache tiene adjuntada un bit PS (Private-Shared) que indica el tipo de cada bloque (según la tabla de páginas y las otras TLBs). El bit PS provisto por la TLB se compara con los bits PS de todas las vías del conjunto. Una lógica sencilla se incluye para seleccionar el wordline (WL) de aquellas vías cuyos bits PS coincidan con el valor de la TLB para la página de la actual referencia a memoria. Esto implica que, en el array de etiqueta, tan solo un subconjunto de las etiquetes se leen y se comparan con la etiqueta de la dirección física y, en el array de datos, tan solo se leen un subcojunto de los bloques de datos. En un acierto, el multiplexor colocado en el array de datos selecciona el bloque de datos correctos de entre los accedidos. Esto permite que la propuesta reduzca significativamente el consumo de energía dinámico a lo largo de todos los accesos a memoria, ya que, por lo general y como los resultados experimentales muestran, tan solo es necesario acceder a una pequeña fracción de las vías en gran parte de las operaciones de memoria.

La propuesta también reduce el consumo de energía dinámico al acceder a la cache desde la NoC (*i.e.*, peticiones de coherence). En este caso, tan solo se mira en las vías compartidas, ya que el mecanismo de clasificación nos asegura de que antes de que llegue una petición de coherencia, el bloque sea compartido o se haya reclasificado como tal por el mecanismo de actualización.

Adicionalmente, para reducir el consumo estático, el mecanismo propuesto también utiliza el bit de inválido. La potencia de todas las vías en estado inválido se desconecta. Esto permite no solo reducir el número de vías posibles para el bloque (cuan más bajo el número, menor el consumo dinámico), sino que también se reduce el consumo de energía estático ya que la fuente de energía a estas vías permanece desconectado mientras pemanecen en este estado.

En el caso de acceder a caches L2 o L3, el bit PS del bloque (ya obtenido de la TLB) se transporta en la petición del fallo. Por el otro lado, los bits PS en la entradas de la cache se actualizan acorde al protocolo de coherencia de la cache, así que el bit PS de una petición y el bit PS del bloque solicitado son siempre coherentes.

Con respecto a la complejidad hardware, la propuesta requiere una complejidad mínima. Por un lado, no se añade ninguna información adicional con la excepción de un único bit (el bit PS) por entrada. Señalar que este bit también puede usarse para optimizar el protocolo de coherencia de cache como en [5]. Por el otro lado, la propuesta se puede adaptar facilmente a las caches actuales. De hecho, usar un único wordline para todas las vías del conjunto presenta varios problemas debido a que, entre otros, muchos transistores estan conectados a los wordlines de las filas y los bitlines de las columnas aumentan la capacitancia total, resultando en un incremento en el retardo y en la disipación de potencia. Para enfrentarse a este problema, los diseños de cache SRAM actuales utilizan un wordline dividido (DWL), que divide el wordline en un número fijo de bloques, por ejemplo, un WL por vía de la cache [26]. Hay que tener en cuenta, que nuestra propuesta se beneficia de este esquema de wordline ya en funcionamiento en caches actuales. Debido al bajo overhead de nuestro esquema, la latencia de acceso de la PS-Cache no se ve afectada. En la L2, el bit PS se conoce antes de acceder a la cache y, por tanto, no afecta al tiempo de ejecución. Incluso considerando el primer nivel de cache, la propuesta se puede integrar en la mayoría de los procesadores deep pipelined, ya que el acceso al primer nivel de cache suele requerir de varias etapas; e.g., un acierto en una L1 necesita 3 ciclos en el AMD Opteron X4 2356 (Barcelona) [27].

La discusión previa se centra en la caches físicamente etiquetadas y físicamente indexadas. No obstante, la propuesta también podría aplicarse a otro tipos de caches; por ejemplo, virtualmente indexadas y físicamente etiquetadas. En estas caches, el array de etiquetas se accede en paralelo con la TLB, y luego la dirección física se usa para comparar solo las etiquetas cuyos tipos coincidan con el del bloque solicitado.

V. Entorno de simulación

Evaluamos nuestra propuesta con una simulación de sistema completo usando Virtutech Simics [28] y el toolset Wisconsin GEMS [29], que permite una simulación dellatada de sistema multiprocesador. GARNET [30], un simulador detallado de redes incluido en el toolset GEMS, modela la red de interconexión.

La tabla I muestra los valores de los parámetros del sistema principal que se corresponde con nuestro sistema base, que es una arquitectura CMP de 16 tiles. Usamos la herramienta CACTI 6.5 [31] para estimar los tiempos de acceso, requisitos de área y consumo de energía de las distintas estructuras cache para una tecnología de 32nm y transistores de altas prestaciones.

Nuestra evaluación analiza dos protocolos de coherencia de cache diferentes: un protocolo de directorio y un protocolo snoopy. Ambos protocolos almacenan los bloques es caches privadas consdierando estados MOESI e implementan una política no inclusiva entre la L1 y la LLC. Además, los reconocimientos de invalidación se envían directamente al solicitante. La mayor diferencia entre ambos protocolos es el compromiso entre área y energía. El protocolo de directorio implementa una cache de directorio en el chip, lo que aumenta el overhead en área, mientras el protocolo snoopy realiza difusiones tras cada escritura o tras cada lectura en caso de que la información no se encuentre en la LLC. Como se analizaba en la sección II, los protocolos snoopy inducen un número más elevado de peticiones de coherencia a las caches L1, por tanto, se beneficiará más de una reducción del número de vías accedidas durante esas peticiones.

La propuesta se ha evaluado usando una amplia ga-

TABLA I
PARÁMETROS DEL SISTEMA

Memory Parameters	
Cache hierarchy	Non-inclusive
Cache block size	64 bytes
Split L1 I & D caches	64KB, 8-way
L1 cache access time	2 cycles
Shared single L2 cache	512KB/tile, 16-way
L2 cache access time	6 cycles (2 if only tag)
Memory access time	160 cycles

Network Parameters	
Topology	2-dimensional mesh (4x4)
Routing technique	Deterministic X-Y
Flit size	16 bytes
Data and control message size	5 flits and 1 flit
Routing, switch, and link time	2, 2, and 2 cycles

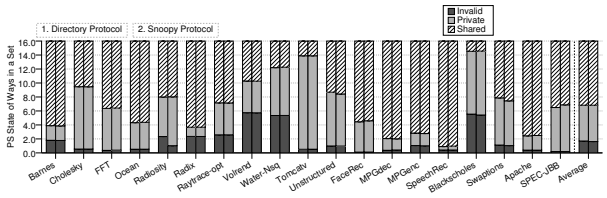


Fig. 6. Número medio de vías en un conjunto de cada tipo en la cache L2 para los dos protocolos estudiados.

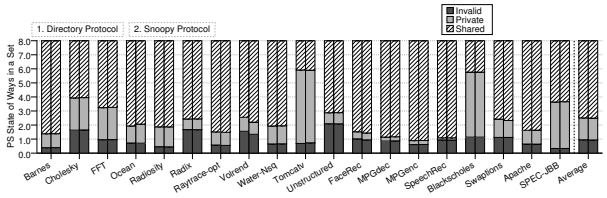


Fig. 7. Número medio de vías en un conjunto de cada tipo en la cache L1 para los dos protocolos estudiados.

ma de aplicaciones científicas de la suite de benchmarks SPLASH-2 [32], la suite ALPBench [33], la suite PARSEC [34], aplicaciones científicas y cargas comerciales. La tabla II muestra una lista de las aplicaciones consideradas en el estudio. Los resultados experimentales mostrados en este trabajo se corresponde con la fase paralela de los benchmarks evaluados.

VI. EVALUACIÓN EXPERIMENTAL

A. Análisis del comportamiento de los bloques

Los beneficios energéticos de la propuesta dependen del número medio de vías a las que se accede durante el acceso a la cache. Este número cambia dependiendo de la cache a la que se está accediendo (L1 o L2), y del tipo de bloque que se está buscando.

El estudio empieza con la cache L2 ya que implementa un número mayor de vías, por tanto, la propuesta potencialmente puede obtener mayores ahorros de energía en esta cache.

TABLA II
APLICACIONES SIMULADAS

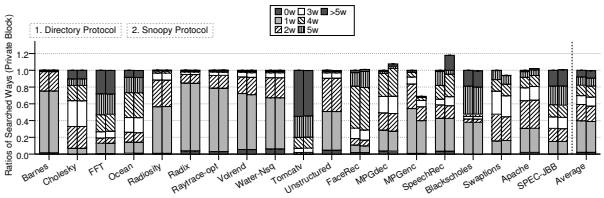
SPLASH-2 benchmark suite	
Name	**Parameters**
Barnes	16K particles
Cholesky	tk15
FFT	64K complex doubles
Ocean	514×514 ocean
Radiosity	room, -ae 5000.0 -en 0.050 -bf 0.10
Radix	512K keys, 1024 radix
Raytrace	teapot - optimized
Volrend	head
Water-Nsq	512 molecules

PARSEC suite	
Name	**Parameters**
Blackscholes	simmedium
Swaptions	simmedium

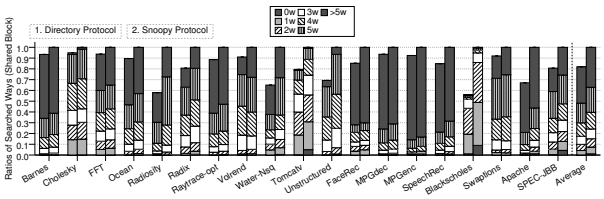
ALPBench suite	
Name	**Parameters**
FaceRec	script
MPGdec	525_tens_040.m2v
MPGenc	output of MPGdec
SpeechRec	script

Scientific benchmarks	
Name	**Parameters**
Tomcatv	256 points
Unstructured	Mesh.2K

Commercial workloads	
Name	**Parameters**
Apache	4000 transactions
SPEC-JBB	4000 transactions



(a) Número de vías accedidas cuando se busca un bloque privado.



(b) Número de vías accedidas cuando se busca un bloque compartido.

Fig. 8. Distribución del número de vías accedidas en la cache L1 normalizadas con un protocolo snoopy.

La figura 6 muestra el número medio de bloques de cada tipo en una cache L2, asociativa por conjuntos, de 16 vías. Los resultados que se muestran se corresponden a un protocolo snoopy y a uno de directorio MOESI. Como se puede apreciar, de media, hay unos cinco bloques privados en un conjunto, cuyo acceso supondría un ahorro de

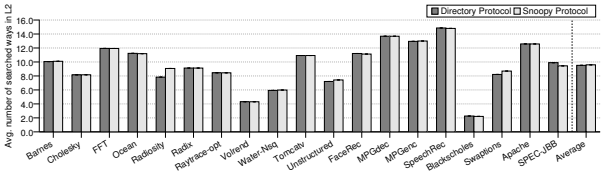


Fig. 9. Número medio de vías accedidas en la cache L2 para los protocolo estudiados.

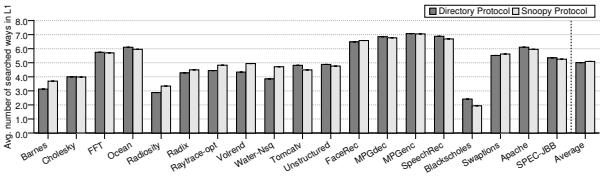


Fig. 10. Número medio de vías accedidas en la cache L1 para los protocolo estudiados.

energía considerable. Aún así, este número depende especialmente de la aplicación. Hay algunas aplicaciones con más de doce bloques privados en un conjunto de medía (e.g. *tomcatv*), pero como se puede comprobar, la mayoría de las aplicaciones almacenan bloques compartidos en la mayoría de sus vías.

La figura 7 muestra el número medio de bloques de cada tipo en una cache L1, también asociativa por conjuntos, de 8 vías. A diferencia de las caches L2, la diferencia entre vías privadas y compartidas es mayor. Alrededor de dos vías almacenan bloques privados y otras cinco almacenan bloques compartidos.

Los resultados confirman que el número final de vías accedidas varía acorde al tipo de bloque que se busca y del comportamiento de la aplicación. El número medio de bloques que se buscan a la llegada de una petición de un bloque privado puede diferir bastante de los que se buscan a la llegada de una petición de un bloque compartido.

Para poder hacerse una idea de cuanto ahorro de energía se puede conseguir con al propuesta, la figura 8(a) y la figura 8(b) muestran la distribución del número de vías accedidas en cada acceso a la cache L1 diferenciando entre accesos por un bloque privado y accesos por un bloque compartido. Los datos en la primera figura están normalizado al número de accesos a memoria usando un protocolo de directorio y PS-Cache, mientras que la segunda figura está normalizada al número de accesos a memoria cuando se usa un protocolo snoopy y PS-Cache. Como se esperaba a causa de la figura 7, la mayoría de las aplicaciones buscan más vías cuando acceden a bloques compartidos que cuando acceden a bloques privados, con tan solo algunas excepciones como *Tomcatv* y *Blackscholes*. Una apreciación interesante es que al buscar bloques privados, la mayoría de las veces (más del $60\,\%$ de los accesos) tan solo se mira en una o dos vías. Los beneficios son menores cuando se busca un bloque compartido, pero incluso en este caso, sobre el $60\,\%$ de las veces, se miran 5 vías o menos, lo que también supone un ahorro energético. En cuanto a los protocolos, se pueden extraer dos observaciones. Primero, se puede apreciar el ratio compartido-privado es bastante similar en ambos protocolos independientemente de si se solicita un bloque privado o compartido. Segundo, las diferencias más grandes entre los protocolos aparecen principalmente cuando se busca un bloque compartido, con la excepción de *SpeechRec* cuando se busca un bloque privado. En esta caso, el protocolo de directorio reduce el número de búsquedas cerca de un $20\,\%$ de media con respecto al protocolo snoopy. Además, esta reducción puede ser incluso de hasta un $70\,\%$ en *Water-Nsq* cuando se busca un bloque compartido.

La figura 9 muestra el número medio de vías accedidas por acceso en la L2. De media, una cache de segundo nivel implementando la arquitectura PS-Cache tan solo accede a 10 de sus 16 vías, aunque hay algunos casos (i.e. *BlackScholes*) en los que este valor puede ser tan bajo como tan solo 2 vías por acceso.

Como se comentaba anteriormente, la propuesta se puede aplicar a cualquier nivel de la jerarquía de cache, así que está sección explora los beneficios de la cache L1 en el sistema evaluado.

La figura 10 presenta cuantas vías se miran de media en los diferentes benchmarks en la cache de primer nivel propuesta de 8 vías. Los resultados apenas señalan una diferencia significativa en ambos tipos de protocolos de coherencia, siendo que ambos acceden de media a 5 vías. En algunas aplicaciones, la PS-Cache consigue reducir notablemente el número de vías que se mirán (*e.g.*, tan solo 2 en *Blackscholes*), mientras que en otros como *MPGenc* que presenta un número elevado de vías compartidas, el impacto no es tan notable.

B. Consumo de energía

Esta sección analiza el impacto de la propuesta en el consumo de energía de las caches.

La figura 11 muestra el consumo de energía dinámico de la cache L2 para los protocolos de directorio y snoopy considerados en este trabajo. Se han incluido los protocolos y caches convencionales (baseline) con fines comparativos. Los resultados de la arquitectura PS se han etiquetado junto al nombre del protocolo implementado en el sistema. En esta cache, no hay demasiado diferencia entre ambos protocolos de coherencia. Como se puede comprobar, los ahorros energéticos varían entre los distintos benchmarks. De media, la PS-Cache consigue una reducción de energía del $40\,\%$ en ambos protocolos de coherencia. No obstante, hay que señalar que en el agunos casos el consumo de energía de la PS-Cache es tan solo del $12\,\%$ del sistema convencional (*BlackScholes*), e incluso en el peor caso, estos beneficios siempre superan el $8\,\%$.

La figura 12 muestra los resultados para la cache L1. De media, se obtienen resultados similares de ahorro de energía (i.e. sobre un $22\,\%$) por los dos protocolos en este primer nivel de cache. Un punto a señalar es que un protocolo snoopy aplicando la arquitectura PS-Cache puede consumir menos que un protocolo de directorio convencional. Esto simplemente se consigue mediante la búsqueda selectiva de vías dentro de un conjunto gracias al bit PS.

Como se sugirió en la sección anterior, aplicaciones con un gran número de búsquedas de bloques privados,

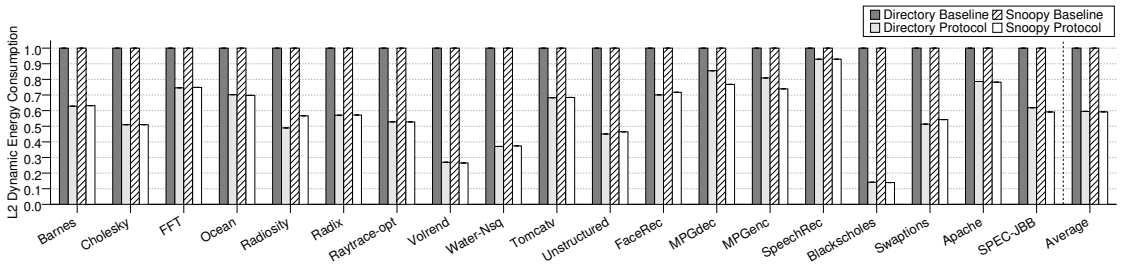


Fig. 11. Reducción del consumo dinámico de enegía en la L2.

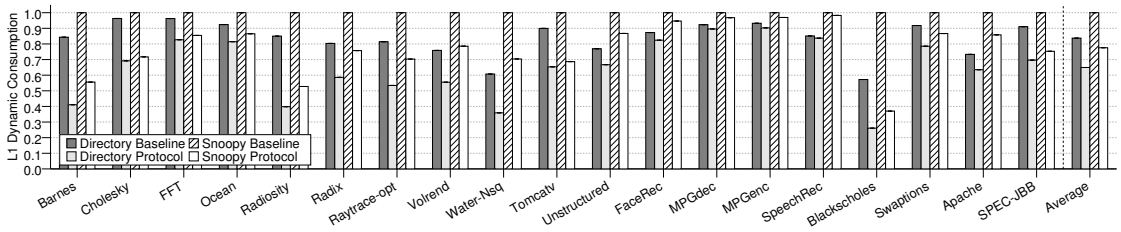


Fig. 12. Reducción del consumo dinámico de enegía en la L1.

obtienen reducciones de energía mayores. Por ejemplo, *Barnes* reduce el consumo de energía dinámica en un $44\,\%$ y un $51\,\%$ para un protocolo snoopy y uno de directorio, respectivamente; y *Radiosity* en un $47\,\%$ y un $53\,\%$, respectivamente. Por el otro lado, aplicaciones con un número de búsquedas de bloques privados bajo no cosechan tantos beneficios. El mejor ejemplo de este caso es el benchmark *SpeechRec*, en el que solo se reduce el consumo de energía en un $3\,\%$.

Los resultados muestran que la reducción de consumo de energía dinámico es mayor en caches L2 que en caches L1, aún más si se consideran chaches L1 virtualmente indexadas y físicamente etiquetadas en vez de físicamente indexadas y físicamente etiquetadas. Así pues, se puede concluir que cuanto más se profundiza en la jerarquía de memoria, más beneficios se pueden obtener de esta técnica.

VII. Conclusiones

Una de las mayores preocupaciones en los diseños de multiprocesadores en chip de altas prestaciones actuales es el consumo de energía, que se incrementa conforme lo hace el número de núcleos. Las caches on-chip normalmente consumen una parte significativa del consumo total del sistema, y mucho trabajo de investigación se ha centrado en reducir el consumo de energía de estas estructuras de memoria a cambio de una penalización en las prestaciones.

En este trabajo, proponemos un diseño de cache energéticamente eficiente que solo accede a un subconjunto de las vías de un conjunto sin afectar a las prestaciones. Los bloques se clasifican a nivel de página como compartidos o privados según la información de la TLB. Además, la propuesta añade un único bit a cada línea de la cache, que activa únicamente el word line de los bloques cuyo tipo coincidan con el obtenido de la TLB. De esta forma se puede ahorrar energía dinámica de forma notable. Las peticiones de coherencia a una cache remota tan solo requieren acceder al subconjunto de vías del conjunto con bloques compartidos. Este diseño de cache se puede implementar en todos los niveles de la jerarquía de memoria.

Los resultados muestran que en sistemas CMP que implementan tanto protocolos de directorio como snoopy, la PS-Cache puede obtener ahorros de energía importantes y que estos ahorros (cuantificados en porcentaje) son similares en todos los protocolos de coherencia probados.

Agradecimientos

Este trabajo se ha realizado conjuntamente con el apoyo de MINECO y la Comisión Europea (FEDER funds) bajo el proyecto TIN2012-38341-C04-01/03 y por la *Fundación Seneca-Agencia de Ciencia y Tecnologia de la Región de Murcia* bajo el proyecto *Jóvenes Líderes en Investigación* 18956/JLI/13.

Referencias

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

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

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

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

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

[6] Seth H. Pugsley, Josef B. Spjut, David W. Nellans, and Rejeev Balasubramonian, "SWEL: Hardware cache coherence protocols to map shared data onto shared caches," in *19th Int'l Conf. on*

Parallel Architectures and Compilation Techniques (PACT), Sept. 2010, pp. 465–476.

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

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

[9] Alberto Ros and Stefanos Kaxiras, "Complexity-effective multicore coherence," in *21st Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Sept. 2012, pp. 241–252.

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

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

[12] Jason F. Cantin, James E. Smith, Mikko H. Lipasti, Andreas Moshovos, and Babak Falsafi, "Coarse-grain coherence tracking: RegionScout and region coherence arrays," *IEEE Micro*, vol. 26, no. 1, pp. 70–79, Jan. 2006.

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

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

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

[16] David H. Albonesi, "Selective cache ways: On-demand cache resource allocation," in *32nd IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, Dec. 1999, pp. 248–259.

[17] Brad Calder and Dirk Grunwald, "Predictive sequential associative cache," in *2nd Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 1996, pp. 244–253.

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

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

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

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

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

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

[24] Mohammad Alisafaee, "Spatiotemporal coherence tracking," in *45th IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, Dec. 2012, pp. 341–350.

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

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

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

[28] Peter S. Magnusson, Magnus Christensson, Jesper Eskilson, Daniel Forsgren, Gustav Hallberg, Johan Hogberg, Fredrik Larsson, Andreas Moestedt, and Bengt Werner, "Simics: A full system simulation platform," *IEEE Computer*, vol. 35, no. 2, pp. 50–58, Feb. 2002.

[29] Milo M.K. Martin, Daniel J. Sorin, Bradford M. Beckmann, Michael R. Marty, Min Xu, Alaa R. Alameldeen, Kevin E. Moore, Mark D. Hill, and David A. Wood, "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset," *Computer Architecture News*, vol. 33, no. 4, pp. 92–99, Sept. 2005.

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

[31] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P. Jouppi, "Cacti 6.0," Tech. Rep. HPL-2009-85, HP Labs, Apr. 2009.

[32] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *22nd Int'l Symp. on Computer Architecture (ISCA)*, June 1995, pp. 24–36.

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

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