# Reduciendo el consumo dinámico de energía con Tag Filter Cache

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

*Resumen*—
**El consumo de energía en los actuales multiprocesadores en chip (CMPs) de altas prestaciones se ha convertido en una de las mayores preocupaciones a la hora de realizar nuevos diseños. La tendencia actual del aumento del número de cores no hace sino agravar este problema. Las caches on-chip consumen una fracción significativa del consumo total del chip. La mayoría de las técnicas propuestas para reducir el consumo de energía de estas estructuras de memoria suelen conllevar una pérdida de prestaciones, lo cual puede ser inaceptable para CMPs de altas prestaciones. Estas caches en sistemas multinúcleo se implementan con un alto grado de asociatividad para poder incrementar sus prestaciones. Incluso las caches de primer nivel se están implementando actualmente con ocho vías. El acceso concurrente a todas las vías en el conjunto de la cache es bastante costoso en términos de energía. En este paper proponemos un diseño de cache energéticamente eficiente, la arquitectura Tag Filter Cache (TF-Cache), que filtra algunas de las vías de un conjunto durante los accesos a la cache, permitiendo el acceso únicamente de un subconjunto de ellos sin afectar a las prestaciones. Nuestra cache almacena para cada vía los bits de etiqueta de menor peso en un bit array auxiliar y estos bits se utilizan para filtrar las vías cuyos bits no coincidan con los de la etiqueta del bloque buscado. Los resultados experimentales muestran que, de media, la arquitectura TF-Cache reduce el consumo de energía dinámico hasta un $74.9\,\%$ y $85.9\,\%$ cuando se aplica a una cache L1 y L2, respectivamente, para las aplicaciones evaluadas.**

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

## I. Introducción

CONFORME se avanza en la escala de integración y se aumentan los recursos de silicio, el número de núcleos en las nuevas generaciones de multiprocesadores en chip (CMPs) se va incrementando progresivamente. Estos CMPs aceleran el acceso a memoria introduciendo uno o más niveles de cache, siendo éstas responsables de un porcentaje significativo del área ocupada [1] y de una parte importante del consumo de energía total. Gran parte de este consumo es debido a consumo dinámico (los cambios de nivel de los transistores durante los accesos). Los diseñadores de cache deben ofrecer nuevos diseños en los que se llegue a un compromiso entre prestaciones, coste, tamaño y disipación de energía.

El primer nivel de la jerarquía de memoria es el que domina sobre el consumo dinámico ya que es accedida con mayor frecuencia que el último nivel de cache (LLC), por ejemplo, caches L3, a las cuales se accede bastante menos. Además, este elevado consumo también es debido al hecho de que se accede a etiquetas y datos en paralelo, ya que los primeros niveles de cache tienen un alto grado de influencia sobre las prestaciones del procesador. Esto resulta incluso aún más importante en CMPs que en procesadores monolíticos ya que las caches pueden ser accedidas tanto desde el lado del procesador como desde la red de interconexión (para peticiones de coherencia), aumentando así el número de accesos a la cache.

Debido al interés de mantener unas prestaciones elevadas, estas caches se implementan con un alto grado de asociatividad. En los microprocesadores de altas prestaciones, todas las vías del conjunto se acceden concurrentemente durante un acceso a la cache. Por tanto, el grado de asociatividad define el número de etiquetas que se tienen que mirar en paralelo durante cada acceso. Las caches incluyen un comparador por vías y necesitan realizar tantas comparaciones de etiquetas como número de vías. Como consecuencia directa, la energía dinámica disipada por acceso se ve incrementada con la asociatividad de la cache.

Generalmente, el diseño de caches de bajo consumo dinámico se centran en minimizar la actividad interna de los transistores durante un acceso a la cache. Esta actividad procede de la lectura y comparación de etiquetas en los tag arrays y de la lectura o escritura de datos en los data array. Idealmente, en un acierto en cache, la cache tan solo necesitaría leer y comparar una única etiqueta y acceder a una entrada de datos. Además, en caso de fallo, idealmente una cache no tendría por que necesitar acceder ni al tag array ni al data array. De hecho, para detectar un fallo, la cache no tiene la necesidad de acceder a un campo de etiqueta completo, ya que la diferencia de un único bit es suficiente para detectarlo.

Muchas propuestas para la reducción de energía en las caches se han centrado en procesadores monolíticos (tales como Cache Decay [2], Drowsy Caches [3] y Way Guard [4]). Algunas de ellas, por ejemplo [5], fueron diseñadas originalmente para reducir el tiempo de acceso a la cache, pero investigaciones posteriores han demostrado que estos esquemas también ofrecen importantes ahorros de consumo. No obstante, ya que estos esquemas no se aplican directamente a CMPs o pueden ser mejorados, la investigación actual se centra en el ahorro de energía en CMPs ejecutando cargas paralelas.

En este trabajo proponemos la Tag Filter Cache (TF-Cache), una arquitectura cache que reduce el número de etiquetas y bloques de datos que se comprueban cuando se accede a la jerarquía de memoria. La TF-Cache se puede aplicar a cualquier nivel de la jerarquía de memoria con el objetivo de reducir el consumo de energía dinámico de estas estructuras. Se basa en la utilización de los bits de menor peso (LSB) de la etiqueta para discernir que vías de una cache asociativa por conjuntos puede contener el bloque buscado. Únicamente se accede a aquellas vías que puedan estar conteniendo el bloque referenciado, consiguiendo así ahorrar el consumo de energía necesario para acceder al resto de vías.

La TF-Cache se puede implementar con una complejidad de hardware mínima. Además, se puede acceder en

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

paralelo a los tag y data arrays así que no se origina ninguna degradación de prestaciones, lo cual es una de las mayores preocupaciones en las cache L1. A diferencia de otras propuestas como [6] no es necesario realizar ningún alineamiento de vías.

Los resultados experimentales muestran que la arquitectura TF-Cache puede reducir el consumo de energía dinámico hasta un $74.9\,\%$ y $85.9\,\%$ para las caches L1 y L2, respectivamente, obteniendo así mejores resultados que otros trabajos recientes.

El resto de este trabajo se organiza de la siguiente forma:

La sección II describe las razones principales que motivan esta investigación. La sección III describe el trabajo relacionado. La sección IV presenta la propuesta. La sección V describe el entorno experimental. La sección VI presenta los resultados obtenidos. Finalmente, la sección VII presentará unas conclusioens finales.

## II. Motivación

Las memorias cache, especialmente las caches de primer y segundo nivel, se acceden frecuentemente, dado que las instrucciones de memoria representan un porcentaje significativo de la instrucciones ejecutadas. Una fracción significativa del consumo total de energía es consumida normalmente por las caches on-chip, tal y como sucede en el procesador Niagara2 [7], donde el $44\,\%$ del consumo del chip lo consume la cache L2. La reducción del consumo de energía dinámico en las caches de los CMPs es un problema actual que está siendo investigado [8] [6]. Para dar solución a este problema, este trabajo propone una arquitectura que pretende aprovechar la distribución homogénea de los bits menos significativos de la etiqueta en las vías de un conjunto en una cache asociativa por conjuntos.

Lanzamos varios experimentos para verificar esta hipótesis en las cargas evaluadas. La figura 1 muestra la distribución media de los bloques en una cache L1 de 8 vías y una cache L2 de 16 vías para un sistema CMP con 16 núcleos[1].

Como se puede ver en las figuras 1(a) y 1(b), de media hay 1 y 2 vías en estado invalido, bajo un protocolo de coherencia MOESI, para la L1 y L2, respectivamente. Mientras tanto, el resto de vías mantienen una distribución bastante homogénea considerando los bits de menor peso de los bloques almacenados. No hay ninguna necesidad de acceder a todas las vías de un conjunto si existe un mecanismo que permite filtrar el acceso a un subconjunto de vías que pueda estar conteniéndolo. Una distribución homogénea como la mencionada anteriormente (los bits de menor peso) es, por tanto, un método muy interesante para un mecanismo de filtrado.

El objetivo de este trabajo es ahorrar energía dinámica reduciendo el número de consultas que se hacen durante cada acceso a la cache. Concretamente, la propuesta ahorra energía accediendo únicamente a aquellas vías cuyos bits de menor peso coincidan con los bits de menor peso del bloque solicitado.

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

## III. Trabajo Relacionado

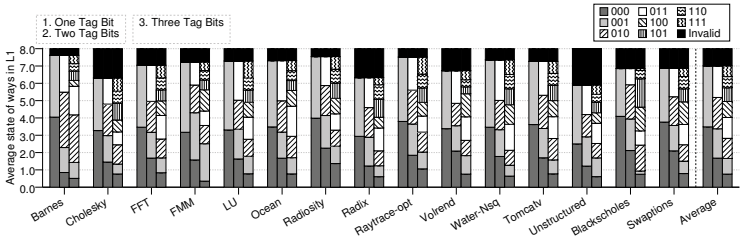
Este trabajo presenta un diseño de cache energéticamente eficiente que utiliza los bits de menor peso de las etiquetas de los bloques referenciados por las aplicaciones para filtrar vías y reducir el consumo. Por tanto, esta sección comentará algunos trabajos relacionados sobre el diseño de caches que pretenden reducir el consumo de energía.

El consumo de las caches proviene tanto del leakage (o consumo estático) como del dinámico. Con respecto al ahorro del leakage, Powell *et al.* [9] proponen Gated-Vdd, una técnica que pretende reducir el consumo estático de las caches de instrucciones reconfigurándolas y desconectando líneas no utilizadas. Kaxiras *et al.* [2] proponen Cache Decay, una propuesta que reduce el leakage de las caches del procesador apagando aquellas líneas que se predicen como muertas, es decir, aquellas que no van a volver a ser referenciadas por el procesador antes de su expulsión. Alternativamente, Flautner *et al.* [3] explotan el hecho de que durante un periodo de tiempo concreto solo se accede a un subconjunto de las líneas de la cache para proponer las Drowsy Caches. A diferencia de las propuestas anteriores, el voltaje es reducido pero no eliminado para aquellas líneas que no están siendo accedidas. Consecuentemente, no se preserva el contenido de la línea de la cache.

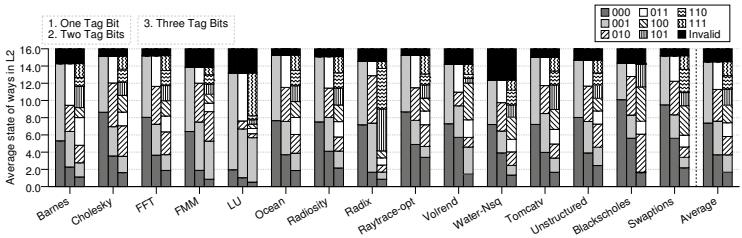
Mientras que las técnicas que intentan reducir el leakage se centran en reducir (o cortar) el voltaje, las técnicas de ahorro de energía dinámica intentan minimizar el número de datos leídos y escritos en cada acceso a la cache. Por ejemplo, Albonesi [10] propone Selective Cache Ways, un diseño de cache que activa tan solo un subconjunto de las vías de la cache cuando la actividad de la cache no es muy elevada. La predicción de vías ya fue previamente propuesta por Calder *et al.* [5] para reducir el tiempo de acceso a caches asociativas por conjunto. Esta propuesta funciona bien en caches L1 con un grado de asociatividad relativamente pequeño ya que presenta unos patrones de acceso muy predecibles. No obstante, como mostramos en la evaluación, este mecanismo presenta unos resultados pobres para niveles inferiores de cache ya que la localidad queda ocultada por los niveles anteriores.

Ghost *et al.* [4] proponen Way Guard, un mecanismo para grandes caches asociativas por conjunto que utilizan bloom filters para reducir el consumo de energía dinámico al evitarse el acceso a aquellas vías que no contengan los datos solicitados a partir de la información del bloom filter. Este esquema requiere añadir un decodificador grande y dos campos por vía: un bloom filter segmentado, propuesto previamente por los mismos autores para filtrar el acceso de toda la cache [11], y otro bloom filter para filtrar el acceso a las vías. Esto puede suponer un sobrecoste excesivo y un incremento notable de la complejidad. Se realiza una comparación cuantitativa con Way Guard en la sección de evaluación.

Valls *et al.* [12] proponen PS-Cache, un mecanismo que filtran las vías que se miran en cada acceso a la cache al clasificar cada bloque como privado o compartido, según la información de la tabla de páginas. En cada acceso a la cache, solo se accede a las vías que contienen




(a) Número medio de vías de cada tipo en un conjunto en la cache L1




(b) Número medio de vías de cada tipo en un conjunto en la cache L2

Fig. 1. Número medio de vías de cada tipo en un conjunto en la jerarquía de cache.

bloques cuyo tipo coinciden con el de la clasificación. También se ha realizado una comparación con esta propuesta.

Finalmente, otras técnicas propuestas recientemente se centran en reducir los dos tipos de consumo, por ejemplo, reduciendo el área de las etiquetas, como en TLB Index-Based Tagging [8], utilizando caches de mapeado directo junto mecanismos para eliminar fallos por conflicto, como en ASCIB [13], o realizando particionamiento en tiempo de ejecución, como en Cooperative Caching [6] o ReCaC [14].

## IV. Tag Filter Cache

El objetivo principal de Tag Filter Cache es reducir el número de etiquetas que se comparan en cada acceso a la cache y también del número de vías que se acceden en paralelo en el data array, de forma que se pueda reducir el consumo de energía dinámico en estas estructuras. En un acceso a cache típico, para comprobar si el bloque se encuentra en la cache, es necesario comparar todas las etiquetas en todas las vías del conjunto y, como mucho, una de esas comparaciones tendrá éxito, mientras que el resto fallarán. En las caches de primer nivel, todas las vías del conjunto en el data array se acceden simultáneamente, antes de saber si el bloque se encuentra finalmente en el conjunto. Este trabajo propone filtrar el acceso de aquellas vías (tanto en etiquetas como en datos) en las que se espera que la comparación de etiquetas vaya a fallar.

La figura 2 muestra un diagrama de bloques de la propuesta para una cache L1. El filtro consiste en comparar únicamente un subconjunto de $X$ bits de los bits de las etiquetas. Con este propósito, el tag array se desacopla en dos estructuras: una de $X_s$ bits y otras de $N - X$ bits. La TF-Cache utiliza los bits de menor peso de la etiqueta, que se encuentran almacenados en la estructura delgada de $X_s$ bits, para reducir el número de vías accedidas.

El mecanismo realiza la comparación de etiquetas en dos fases. En la primera fase, tan solo se comparan los $X_a$ bits de menor peso de la etiqueta del bloque solicitado con los bits de menor peso de todas las vías del conjunto. El reducido número de bits que se usan en el mecanismo permiten que esta primera comparación sea rápida e introduzca una penalización de tiempo despreciable. En la segunda fase, el resto de bits de etiqueta se comparan con los bits correspondientes de la dirección virtual del bloque que se está buscando. Esta segunda comparación, que requiere de un número mayor de bits, solo se realiza en aquellas vías que hayan tenido acierto durante la primera comparación.

De igual forma, la TF-Cache solo accede a aquellas vías en el data array cuya primera comparación hayan tenido éxito. Hay que tener en cuenta, que en una cache convencional, todos los bits de etiqueta y todos los datos se leen en paralelo para todas las vías del conjunto. Gracias al filtrado, y como se demuestra en la sección de evaluación, el mecanismo consigue importantes ahorros

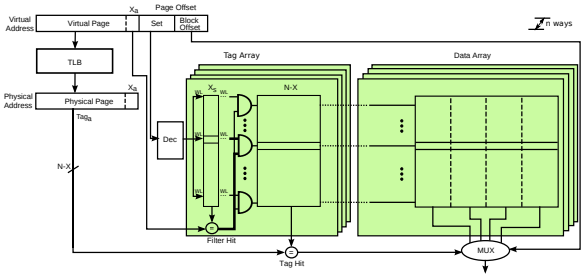



Fig. 2. La arquitectura TF-Cache para caches L1.

de energía.

Para que este mecanismo funcione en caches VIPT (virtually indexed physically-tagged) como las de los procesadores Intel, esta primera comparación se tiene que realizar antes de que la salida de la TLB se conozca. Para ello, asumimos que el sistema operativo es el responsable de asegurar que los $X$ bits de menor peso de la dirección virtual coincidan con aquellos de la dirección física. Esto es una suposición razonable ya que se espera una distribución de direcciones de página uniforme y las capacidades de memoria principal son de cuatro órdenes de magnitud superiores que las del tamaño de páginas (por ejemplo 32GB de memoria principal [15] y un tamaño de página de 4KB), lo que permite que el OS tenga cierta flexibilidad de ubicación. Bajo este supuesto, la primera comparación puede realizarse en cuanto esté el resultado de la decodificación del conjunto, mientras la traducción de la dirección en la TLB está aún siendo procesada. En cuanto la TLB termina la traducción, los $N - X$ bits de las vías que no hayan sido filtradas en la primera comparación se comparan con los $N - X$ que devuelve la TLB.

En cuanto a complejidad, la propuesta requiere un hardware mínimo para adaptarse a las caches actuales. Como se menciona arriba, el tag array se desacopla en dos estructuras independientes. Se necesita una lógica sencilla para dirigir la señal del wordline (WL) tanto a la estructura de etiquetas de $N - X$ bits como al data array. Como se puede apreciar en la figura 2, el wordline activa únicamente aquellas vías que hayan tenido éxito durante la primera comparación. Hay que tener en cuenta que las puertas AND no cortan el suministro de potencia pues esto no preservaría el contenido de los datos.

Este diseño permite reducir el consumo de energía dinámico de forma significativa, ya que, como los resultados experimentales revelan, tan solo una pequeña fracción de las vías se acceden en la mayoría de los accesos.

## V. Entorno de simulación

Evaluamos nuestra propuesta con un simulador de sistema completo utilizando Virtutech Simics [16] y el toolset Wisconsin GEMS [17], que permite una simulación detallada de sistemas multiprocesador. Para modelar la red de interconexión se utiliza GARNET [18], un simulador detallado de redes incluido en el toolset de GEMS.

TABLA I
Parámetros del Sistema

| **Parámetros de Memoria** | |
|---|---|
| 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 accessed) |
| Directory cache | 256 sets, 4 ways (same as L1) |
| Directory cache hit time | 2 cycles |
| Memory access time | 160 cycles |
| **Paramétros de Red** | |
| 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 |

La tabla I muestra los valores de los parámetros del sistema principal correspondientes a nuestro sistema base, que es una arquitectura CMP de 16 tiles. Utilizamos la herraminenta CACTI 6.5 [19] para estimar los tiempos de acceso, los requisitos de área y el consumo de energía de las diferentes estructuras cache para una tecnología de 32nm y transistores de altas prestaciones.

Nuestra evaluación analiza una jerarquía cache con L1 privadas para cada núcleo y una L2 NUCA distribuida entre todos los tiles. Se utiliza un protocolo de coherencia basado en directorio para mantener la coherencia de los datos almacenados en las caches privadas.

Se evalua la propuesta con una amplia gama de aplicaciones científicas. *Barnes* (16K particles), *Cholesky* (tk15), *FFT* (64K complex doubles), *FMM* (16K particles), *LU* (512×512 matrix), *Ocean* (514×514 ocean), *Radiosity* (room, -ae 5000.0 -en 0.050 -bf 0.10), *Radix* (512K keys, 1024 radix), *Raytrace* (teapot –optimizada–), *Volrend* (head), y *Water-Nsq* (512 molecules) pertenecen al benchmark suite SPLASH-2 [20]. *Tomcatv* (256 points) y *Unstructured* (Mesh.2K) son dos benchmarks

científicos. *Blackscholes* (simmedium) y *Swaptions* (simmedium) pertenecen al suite PARSEC [21]. Los resultados experimentales se corresponden con la fase paralela de los benchmarks evaluados.

## VI. Evaluación Experimental

En esta sección se explican los esquemas empleados para la evaluación comparativa y se analizan los resultados experimentales obtenidos.

### A. Esquemas Comparados

Comparamos el esquema propuesto con otras propuestas que también reducen el consumo dinámico de energía accediendo a un subconjunto de las vías en lugar de a todas ellas. Estos esquemas son Way Prediction y dos propuestas más recientes: Way Guard y PS-Cache.

Las técnicas de predicción de vías [5, 10] predicen que vía tiene que se accedida por adelantado, habitualmente la vía que contiene el bloque más recientemente usado (MRU), y solo accede a esa vía en primer lugar. El problema surge cuando la predicción falla, en cuyo caso, el resto de las vías deben buscarse en una segunda fase para intentar dar con el bloque referenciado. Esto significa que tras cada fallo de predicción se está malgastando energía y se está incrementando la latencia, ya que se precisan ciclos adicionales para resolver la petición de memoria.

Way Guard [4] ha demostrado que tiene un funcionamiento eficiente en caches altamente asociativas. Este mecanismo implementa un counting bloom filter asociado a cada vía. Funciona de la siguiente forma: primero, se aplica una función hash a un subconjunto de los bits de la dirección del bloque. La salida de la función hash es un indíce de $m$ bits que se decodifica para acceder al vector de bloom filters de $2^m - 1$ entradas. Si el bit está activo a 1 entonces se accede a la vía de la cache asociada (tanto etiquetas como datos). En caso contrario, no se mira en esa vía. Cada entrada en el bloom filter tiene asociado un contador ascendente-descendente (de 3 bits en el trabajo original), que se decrementa cada vez que se expulsa una línea de la cache cuya dirección se mapea a esa posición y se incrementa cuando un bloque se almacena en la cache. En el trabajo original, se muestran los resultados para un valor de $m$ igual a cuatro veces el número de bloques en una cache. Esta propuesta requiere de un decodificador con 4 veces más salidas que las que ya vienen implementadas en la cache para indexar el conjunto.

La PS-Cache [12] marca los bloques en tiempo de ejecución como compartido o privado siguiendo un sencillo mecanismo de clasificación basado en la información de la tabla de páginas. Durante un acceso a la PS-Cache, tan solo se miran aquellas vías cuyo tipo coincide con el del bloque requerido.

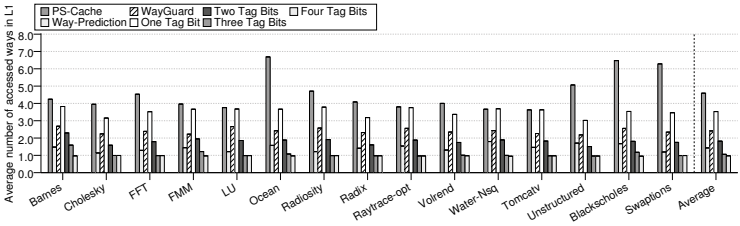
### B. Resultados Experimentales

Los beneficios de esta propuesta dependen del número medio de vías que se acceden en cada acceso a la cache. Este número varía bastante en función del nivel de cache al que se accede (L1 o L2) y del comportamiento de las aplicaciones.

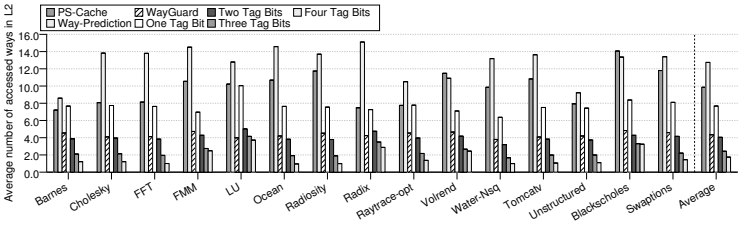
La figura 3(a) muestra el número medio de vías accedidas en una cache L1 de 8 vías para las distintas técnicas estudiadas. Como se puede observar, a mayor número de bits (de 1 a 4) que se utilice en el bit-array para filtrar las vías, menor el número de vías accedidas. De media, para una cache de 8 vías, se accede a $3.53$ vías con un solo bit de etiqueta, $1.82$ con dos bits, $1.06$ con tres bits y $0.98$ con cuatro. Esto significa que los accesos siguen una distribución de acceso uniforme considerando los bits de menor peso. Como era de esperar, el uso de tres bits es suficiente para limitar el número de vías accedidas necesarias a una sola, dado que nuestro primer nivel de cache tiene 8 bías. Por tanto, esto permite que el consumo de una cache asociativa por conjuntos sea similar a la de una cache de mapeado directo. No hay diferencias significativas en los resultados obtenidos para las distintas aplicaciones para un mismo número de bits de etiqueta para el filtrado. Puede darse el caso de tener un número medio de vías accedidas inferior a 1, ya que es posible que ninguno de los bits de menor peso de la etiqueta tengan una coincidencia en el bit-array. En este caso, no se accede a ninguna vía y el fallo de cache se dispara un poco antes de lo habitual.

Comparando con la PS-Cache, la propuesta ya consigue obtener mejores resultados con un único bit de etiqueta. La PS-Cache accede de media a $4.6$ vías, aunque los resultados varían considerablemente entre las distintas aplicaciones. En algunos casos como *Ocean* apenas se consigue una reducción de vías accedidas, mientras que en otras (como *Tomcatv*) puede llegar a reducirlos aproximadamente en un $50\,\%$. El patrón de acceso a bloques privados y compartidos es muy distinto de una aplicación a otro, de ahí estos resultados. Way Guard y Way-Prediction acceden de media a $2.41$ y $1.43$ vías, respectivamente, lo cual se mantiene bastante constante en todas las aplicaciones. Obtienen mejores resultados que la propuesta con un único bit. Dos bits son suficientes para superar a Way Guard y se requiere de un tercero para mejorar a Way-Prediction. Utilizar la predicción de la vía más recientemente usada obtiene buenos resultados debido a su alta tasa de aciertos.

La figura 3(b) muestra el número medio de vías buscadas en una cache L2 de 16 vías. El número medio de vías accedidas en este nivel de la jerarquía de memoria es de $7.68$, $4.04$, $2.43$ y $1.74$ para un valor de $X_s$ de 1, 2, 3 y 4 bits respectivamente. Como sucedía en las caches de primer nivel, descritas anteriormente, la reducción se distribuye de forma similar para todas las aplicaciones. La tendencia muestra que aún hay margen de mejora, pero hay un límite en cuan grande puede ser la pequeña estructura de filtrado. En compración, la PS-Cache, Way-Prediction y Way Guard acceden a $9.85$, $12.7$ y $4.34$, respectivamente. Way-Prediction, que funcionaba realmente bien en caches L1, funciona de forma más pobre en niveles inferiores de la jerarquía cache, ya que la L1 filtra la mayoría de los accesos del procesador, por tanto, la localidad de la aplicación empieza a quedar ofuscada conforme se va descendiendo en la jerarquía. Cuando la predicción acierta, solo se accede a una vía y cuando falla se tienen que acceder al resto. La figura muestra una tasa de acierto de LLC bastante baja. Es interesante mencionar, que un fallo de la predicción también supone ciclos adicionales para poder obtener la información de los datos.




(a) Número medio de vías accedidas en una cache L1 de 8 vías.




(b) Número medio de vías accedidas en una cache L2 de 16 vías

Fig. 3. Número medio de vías accedidas en la jerarquía de memoria.

Way-Prediction es por tanto perjudicial cuando se aplica a este nivel. Tanto Way-Prediction como la PS-Cache obtienen peores resultados que la propuesta incluso con un único bit, mientras que Way Guard obtiene resultados similares a TF-Cache cuando se utiliza un $X_s$ de 2 bits.

La figura 4(a) muestra el consumo de energía dinámico de la cache de primer nivel evaluada en este trabajo. Los resultados se han normalizado con respecto a los de una cache asociativa por conjuntos en la que se accede a todas las vías. La TF-Cache es capaz de reducir el consumo de energía dinámico en un $48.1\,\%$, $65.8\,\%$, $73.2\,\%$, y $74.9\,\%$ para un filtrado de 1, 2, 3 y 4 bits, respectivamente. Se pueden apreciar como los beneficios de los bits adicionales van menguando conforme vamos aumentando su número. Podemos asumir que los resultados con un filtro de 5 bits no serían muy diferentes a los de uno de 4. Como era de esperar por los resultados anteriores, Way-Prediction consigue los mejores resultados, ya que es capaz de reducir el consumo hasta en un $82.1\,\%$. Mientrastanto, la PS-Cache obtiene los peores resultados, ya que es el esquema que accede a un mayor número de vías.

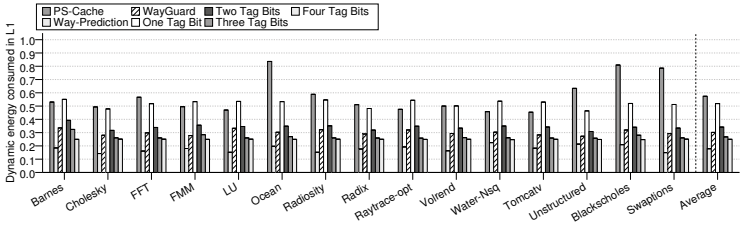
De forma análoga, la figura 4(b) muestra los mismos resultados pero para la cache L2. La TF-Cache es capaz de reducir el consumo en $51.8\,\%$, $72.2\,\%$, $81.1\,\%$ y $85.9\,\%$ para un filtrado de etiquetas de 1, 2, 3 y 4 bits, respectivamente. De nuevo se puede apreciar como la diferencia entre el ahorro en consumo va menguando conforme aumentamos el tamaño del filtro. Way Guard obtiene unas reducciones similares a una TF-Cache de 2 bits, mientras que PS-Cache y Way-Prediction no muestran esas mejoras al compararlas con la arquitectura propuesta, reduciendo el consumo en tan solo un $38.4\,\%$ y $20.4\,\%$, respectivamente.

En resumen, la Tag Filter Cache obtiene unas ganancias de energía significativas, a mayor número de bits de etiqueta mejores, que consiguen superar a otras propuestas recientes como Way Guard, Way-Prediction y PS-Cache. Además, esta idea se puede aplicar a cualquier nivel de la jerarquía de memoria sin incurrir en una degradación de las prestaciones, como le sucede a Way-Prediction en niveles inferiores de cache.

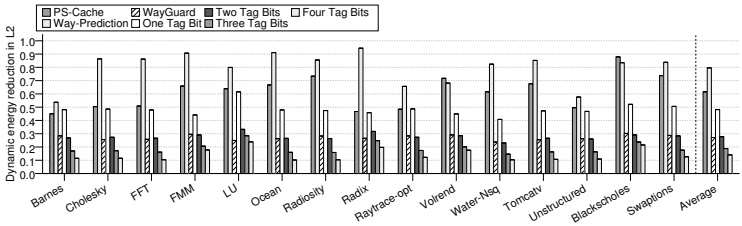
## VII. Conclusiones

Una de las mayores preocupaciones en el diseño de los multiprocesadores en chip de altas prestaciones es el consumo de energía, el cual va en aumento conforme se incrementa el número de núcleos. Las caches on-chip consumen una parte importante del consumo total, y en consecuencia, son un foco importante de investigación. Muchas de las técnicas propuestas consiguen reducir el consumo en estas estructuras de memoria, pero a costa de una pérdida de prestaciones.

En este trabajo se propone TF-Cache, un diseño de cache energéticamente eficiente que solo accede a un subconjunto de las vías del conjunto sin ningún tipo de degradación de prestaciones. La propuesta divide el tag array en dos estructuras diferentes. Una de ellas con uno pocos de los bits de menor peso y otra con el resto de los




(a) Energía dinámica consumida en una cache L1 de 8 vías normalizada con una cache convencional.




(b) Energía dinámica consumida en una cache L2 de 16 vías normalizada con una cache convencional.

Fig. 4. Energía dinámica consumida en la jerarquía de cache.

bits de la etiqueta. Para realizar el filtrado de las vías del conjunto, se realizan dos comparaciones. En la primera comparación, los bits de mentor peso de la etiqueta del bloque buscado se comparan con los bits de menor peso de la estructura reducida del tag array. Esta comparación se puede hacer de manera rápida y sin necesidad de esperar a la salida de la TLB. Una vez se tiene el resultado de la comparación, se realiza una segunda comparando el resto de bits de etiqueta, aunque únicamente en aquellas vías en las que haya habido un acierto en la comparación previa. En el caso de que accedamos al data array en paralelo con el tag array, tan solo se buscaría en esas vías que han tenido un acierto en la primera comparación. De esta forma filtramos los accesos de la cache y se obtienen unas reducciones importantes del consumo dinámico de energía. Esta elección de filtrado es apropiado, ya que como los resultados muestran, hay una distribución homogénea a lo largo de las distintas vías de los bits de menor peso. Este diseño de cache puede implementarse en cualquier nivel de la jerarquía de cache, aunque aquellos que se acceden con mayor frecuencia obtendrán los mejores ahorros de energía. Además, a mayor asociatividad implementada en la cache, mayores son los beneficios potencial que puede aportar la TF-Cache.

Los resultados han mostrado que la TF-Cache puede reducir hasta un $87.75\,\%$ y $89.13\,\%$ el número medio de vías que se buscan cuando se aplica a una cache L1 y una L2, respectivamente. Esto se traduce en ahorros de energía. Concretamente, la arquitectura TF-Cache reduce el consumo dinámico en un $74.9\,\%$ y $85.9\,\%$ cuando se aplica a la cache L1 y L2, respectivamente. Comparada con otros esquemas del estado del arte, la TF-Cache obtiene mejores resultados que las arquitecturas estudiadas, con la única excepción de Way-Prediction en caches de primer nivel por un pequeño margen. Por desgracia, Way-Prediction ha demostrado ser completamente inefectivo cuando se aplica a otros niveles de la jerarquía de cache, mientras que la propuesta funciona adecuadamente en cualquier nivel.

## AGRADECIMIENTOS

Este trabajo se ha realizado conjuntamente con el apoyo de MINECO y la Comisión Europea (FEDER funds) bajo el proyecto TIN2015-66972-C5-1-R 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 Mudge, Stefanos 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] 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.

[5] 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.

[6] 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.

[7] Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi, "Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *42nd IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, Dec. 2009, pp. 469–480.

[8] 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.

[9] 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.

[10] 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.

[11] Mrinmoy Ghosh, Emre Özer, Stuart Biles, and Hsien-Hsin S. Lee, "Efficient system-on-chip energy management with a segmented bloom filter," in *19th Int'l Conf. on Architecture of Computing Systems (ARCS)*, Mar. 2006, pp. 283–297.

[12] Joan J. Valls, Alberto Ros, Julio Sahuquillo, and María Engracia Gómez, "PS-cache: An energy-efficient cache design for chip multiprocessors," in *22nd Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Sept. 2013, pp. 407–408.

[13] Alberto Ros, Polychronis Xekalakis, Marcelo Cintra, Manuel E. Acacio, and José M. García, "Ascib: Adaptive selection of cache indexing bits for reducing conflict misses," in *Int'l Symp. on Low Power Electronics and Design (ISLPED)*, July 2012, pp. 51–56.

[14] 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.

[15] "27-inch imac, technical specifications, available online (nov, 2014) at http://www.apple.com/imac/specs/," .

[16] Peter S. Magnusson, Magnus Christensson, and Jesper Eskilson, et al, "Simics: A full system simulation platform," *IEEE Computer*, vol. 35, no. 2, pp. 50–58, Feb. 2002.

[17] Milo M.K. Martin, Daniel J. Sorin, and Bradford M. Beckmann, et al, "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset," *Computer Architecture News*, vol. 33, no. 4, pp. 92–99, Sept. 2005.

[18] 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.

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

[20] 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.

[21] 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.