El directorio PS: Una caché de directorio multinivel escalable para CMPs

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

Resumen— **El constante aumento del número de núcleos en los *chip multiprocessors* (CMP) actuales en los últimos años y futuros requiere nuevas estrategias que permitan la escalabilidad del sistema. Los protocolos de coherencia basados en directorio constituyen la alternativa más escalable. Sin embargo, un elevado porcentaje de los bloques almacenados en el directorio son accedidos por un único núcleo y por tanto no es necesaria la existencia de un vector de presencia.**

Teniendo esto en consideración, proponemos una caché de directorio de dos niveles que mantenga la información de los bloques compartidos en una caché pequeña y rápida de primer nivel y la información de los bloques privados en una de segundo nivel más grande pero sin código de compartición. Dado que no es necesario mantener un vector de presencia para garantizar la coherencia en este segundo nivel, este esquema permite reducir el área de silicio requerida.

Desde la perspectiva de área y velocidad, parece interesante utilizar tecnología eDRAM, mucho más densa pero más lenta que la tecnología SRAM, para el segundo nivel, lo que reducirá el consumo. Resultados experimentales para CMPs de 16 núcleos muestran que, comparado con una caché de directorio convencional, nuestra propuesta mejora el rendimiento en un 14 % además de reducir el área y el consumo en un 33,98 % y 27 %, respectivamente.

I. Introducción

El número de núcleos en los chips ha ido creciendo continuamente y se espera que se alcancen varios centenares de ellos en los próximos años [1]. La mayoría de los CMPs permiten el modelo de programación de memoria compartida e implementan un protocolo de coherencia para mantener la coherencia de los datos en las cachés de los procesadores.

Los protocolos de coherencia deben escalar al aumentar el número de núcleos. La alternativa más escalable son los protocolos basados en directorio. Una forma de organizar el directorio consiste en almacenar la información en una caché. En este diseño, cuando una línea del directorio es reemplazada de la caché, las copias del bloque involucrado en las cachés del procesador deben ser invalidadas, incluso si están siendo utilizadas por el procesador aumentando así los llamados fallos de *coverage* [2].

Las cachés de directorio mantienen la información tanto de bloques compartidos (accedidos por varios núcleos) como privados (accedidos por solo uno). Una observación interesante es que la mayoría de los bloques accedidos son a bloques privados, incluso en cargas paralelas [2], [3]. Para mantener la información de estos bloques no es necesario un vector de bits, con lo que se puede reducir el área requerida por la caché de directorio, especialmente para sistemas con un elevado número de núcleos.

En este artículo proponemos una nueva organización de caché de directorio que discierne entre bloques privados y compartidos, el directorio PS (Private-Shared). El directorio PS está organizado en dos cachés independientes, una para bloques privados y otra para compartidos. La idea principal detrás de esta propuesta es que cada tipo de bloque tiene distintos requisitos de área y latencia. Por ello, cada estructura se ha diseñado acorde a esas necesidades. La caché compartida está diseñada con los mismos campos que una caché de directorio convencional pero con un número mucho menor de entradas, dado que se espera un reducido número de bloques compartidos que requieren baja latencia. En cambio, la caché privada, posee un número más elevado de entradas pero carece de un vector de bits, siendo así más escalable.

Aunque nuestra propuesta puede ser implementada con tecnología SRAM (6T cells), también estudiamos los beneficios del uso de tecnología eDRAM [4] que ya ha sido utilizada para implementar algunas cachés grandes de procesadores comerciales actuales como el IBM Power7 [5]. La tecnología SRAM se usa por su velocidad para implementar una caché compartida rápida y de baja asociatividad, mientras que la eDRAM se usa para una caché privada más grande, ya que su tiempo de acceso no es muy crítico.

Resultados experimentales para un CMP de 16 núcleos, comparados con los de un directorio convencional con el mismo número de entradas, muestran que el directorio PS mejora el rendimiento en un 14 % debido al trato separado de bloques privados y compartidos, además de reducir el área en un 26,35 %. Considerando tecnología eDRAM la reducción se incrementa hasta el 33,98 %. Finalmente, el directorio PS alcanza un ahorro de energía del 27 %.

II. Tecnologías eDRAM y SRAM

La jerarquía de memoria de los CMPs se suele implementar con una tecnología SRAM (6 transistores por celda) que consume una cantidad importante de energía y área. Hace unos pocos años, los avances tecnológicos permitieron el uso de celdas eDRAM en tecnologías CMOS [4]. La Tabla I muestra como estas tecnologías se comportan para los distintos aspectos de diseño estudiados en este artículo. Comparado con SRAM, las celdas eDRAM tienen menos consumo de energía y una mayor densidad, pero menos velocidad. A causa de la reducida velocidad, las celdas eDRAM no se usan para fabricar cachés de procesador de primer nivel y de altas prestaciones.

La idea de combinar las tecnologías ya comentadas ha sido empleada tanto en la industria como el ámbito acadé-

TABLA I
Características de las tecnologías eDRAM y SRAM.

Tecnología	Densidad	Velocidad	Potencia
SRAM	baja	rápida	alta
eDRAM	alta	lenta	lenta

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

mico, pero a diferencia de nuestro trabajo, ellos se centraban en cachés de procesador convencionales. Por ejemplo, en algunos microprocesadores modernos [6], [7], [5] se usa SRAM en las cachés L1 del procesador mientras que se usa eDRAM para permitir grandes capacidades de almacenamiento en las cachés de último nivel. Con respecto al campo académico, algunos trabajos recientes [8], [9] se han publicado mezclando ambas tecnologías en la misma y/o diferentes estructuras de la jerarquía de memoria. En este artículo se combinan ambas tecnologías para implementar la caché de directorio. Se usa SRAM por su velocidad en la caché compartida mientras que se usa eDRAM por su ahorro de área y consumo en la caché privada de mayor tamaño. De esta forma, conseguimos ganar en escalabilidad y rendimiento gracias a un diseño en el que empleamos técnicas estructurales y una elección de tecnología apropiada para cada una de las cachés del directorio.

III. Trabajos relacionados

Los esquemas tradicionales de directorio no escalan con el número de núcleos, lo cual es la tendencía actual en la industria del microprocesador. Las implementaciones del directorio, tanto en el ámbito académico como en la industría, siguen dos aproximaciones principales: etiquetas duplicadas y directorios *sparse*.

Los directorios de etiquetas duplicadas mantienen una copia de las etiquetas de todos los bloques en la caché de nivel inferior. Por tanto, no se aumenta el número de invalidaciones por culpa del directorio. El vector de compartición se obtiene accediendo a una estructura de directorio completamente asociativa. Con esta metodología se han implementado algunos CMPs modernos [10] y son la base de algunos trabajos de investigación recientes [11], [12]. El principal inconveniente de esta aproximación es la asociatividad requerida por la estructura del directorio, que debe ser igual al producto del número de cachés del núcleo por la asociatividad de tales cachés. El elevado consumo producido por estos sistemas, ha hecho que algunos proyectos de investigación se centren en conseguir una alta asociatividad con un menguado número de vías. El directorio Cuckoo [13] utiliza diferentes funciones hash para indexar cada vía del dírectorio, como las cachés *skew-associative*. Los aciertos tan solo requieren de un acceso, pero los reemplazo necesitan de varias funciones hash para obtener varios candidatos, consiguiendo así la ilusión de una caché de mayor asociatividad a costa de un mayor consumo y latencia.

Los directorios *sparse* [14] están organizados como cachés asociativas. Cada entrada en la caché de directorio mantiene una lista de los compartidores asociados al bloque, normalmente usando un vector de bits como código de compartición. En este esquema, el área por núcleo crece linealmente con el número de núcleos, mientras que el directorio aumenta cuadráticamente ya que el tamaño de las estructuras del directorio aumentan con el número de núcleos. Para acortar el tamaño de las entradas algunos utilizan compresión [15], [16], [17]. En [15], [18] se propone una caché de directorio de dos niveles. En el primer nivel se almacena el típico vector de presencia mientras que el segundo utiliza uno comprimido. Usando compresión, ahorramos área a expensas de una representación inexacta del vector de presencia, con lo que perdemos en productividad. A diferencia de los directorios sparse típicos, un esquema reciente [19] utiliza un formato de entrada distinto y de mismo tamaño. Líneas con uno o pocos compartidores utilizan una sola entrada, mientras que las líneas ampliamente compartidas usan varías líneas de la caché (formato multi-tag) usando vectores de bits jerárquicos. Este esquema requiere de una complejidad y de unos accesos extra para mantener los cambios dinámicos (expandir/contraer) en el formato. Finalmente, otras propuestas [2] se centran en reducir el número de entradas implementadas en la caché de directorio en vez de centrarse en el vector de presencia. Mientras que esta aproximación no afecta a la productividad, requiere modificar el OS, la tabla de paginación, las TLBs del procesador y el protocolo de coherencia.

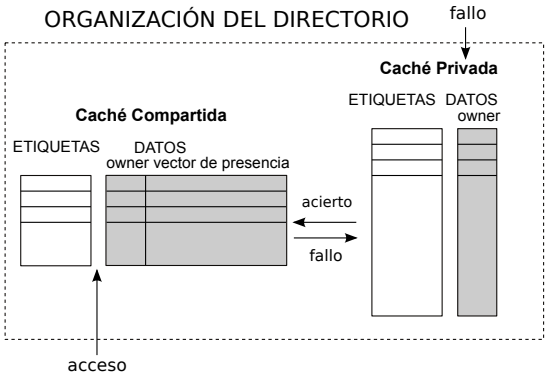


Fig. 1. Organización del Directorio Privado-Compartido.

IV. Esquema propuesto: Directorio PS

La idea principal de la propuesta es ofrecer escalabilidad manteniendo la información de coherencia de bloques compartidos y privados en dos cachés de directorio independientes con diferente organización y capacidad de almacenamiento. De esta forma el directorio PS aprovecha las ventajas que exhiben los distintos comportamientos de los bloques privados y compartidos. La Figura 1 muestra la organización propuesta compuesta de una caché Privada y una caché Compartida, además de los campos requeridos para cada estructura. Como se puede ver, la altura (número de entradas) de ambas caché de directorio y la anchura de las entradas difieren. Ambas cachés tienen diferente altura porque la mayoría de los bloques solo son accedidos por un único núcleo como se ha demostrado en publicaciones recientes [20], [2], así que la caché Privada se ha diseñado con un número de entradas mucho mayor. La anchura de las entradas es diferente a causa del vector de presencia, que no escala con el número de núcleos y que solo se implementa en la caché Compartida, mientras que la caché Privada almacena exclusivamente el propietario del bloque. Por tanto el vector de presencia está tan solo presente en una pequeña fracción de las entradas del directorio.

Una observación interesante concerniente a la caché Privada es que los bloques privados acceden al directorio únicamente una vez (ante un fallo del directorio). En otras palabras, cuando un acceso a un bloque privado falla en la caché del procesador, se busca en el directorio para el mantenimiento de coherencia. Luego, si el acceso resulta en un fallo, el bloque es proporcionado por un banco de la

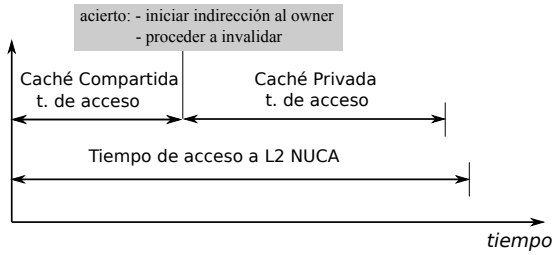


Fig. 2. Acceso paralelo a la caché Compartida y a la NUCA. La caché Privada solo se accede si hay fallo en la Compartida.

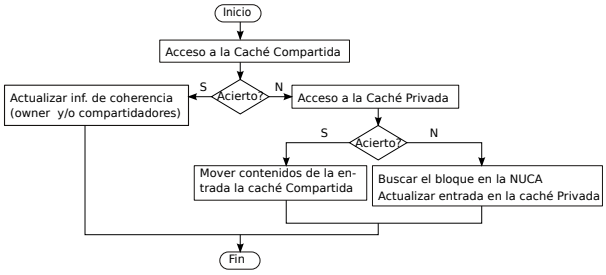


Fig. 3. Diagrama de flujo del controlador de memoria.

NUCA (o por memoria principal) a la caché del procesador y una entrada se coloca en el directorio para mantener la información de ese bloque. La propuesta considera que la caché Privada es la caché por defecto, es decir, ante un fallo de directorio el bloque se asume que es privado y se coloca en la caché Privada. En sucesivos accesos al bloque privado, el procesador lo encontrará en su caché L1, así que no se realizarán más accesos al directorio. Por otro lado, cuando un bloque privado se elimina de la caché del procesador, y asumiendo un protocolo MOESI, el protocolo invalida la entrada asociada en el directorio que se encontrará en estado exclusivo (E) o modificado (M). El siguiente acceso a ese bloque resultaría en un fallo de la caché del directorio. Esto implica que el tiempo de acceso a la caché Privada no afecta al rendimientos de los bloques privados dado que estos bloques van directamente a los núcleos desde el <banco> de la NUCA o memoria principal. Si un bloque de la caché Privada es accedido por un núcleo diferente al del propietario, significa que el bloque pasa a ser compartido y es reubicado en la caché Compartida actualizando el vector de presencia adecuadamente. Desde entonces y hasta su expulsión, la coherencia de ese bloque se mantiene en la caché Compartida. Esta propuesta permite movimientos unidireccionales desde la caché privada a la compartida. Se ha estudiado la transferencia bidireccional de las entradas entre ambas cachés, pero el coste extra del hardware no justifica los escasos beneficios. En cuanto a temporización, las cachés de directorio normalmente son accedidas en paralelo con la caché NUCA y, ante un acierto en el directorio, si los datos del bloque deben ser proporcionados por alguna caché de procesador, el acceso a la NUCA es cancelado. De manera similar, proponemos acceder a la caché Compartida en paralelo con el banco de la NUCA. La Figura 2 muestra esta decisión de diseño. Dependiendo del protocolo, acciones de coherencia específicas pueden comenzar tan pronto como haya un acierto en la caché Compartida, por ejemplo, una petición de coherencia solicitando un bloque puede ser redireccionado al propietario o solicitudes de invalidación para un bloque dado pueden ser enviadas. Ante un fallo en la caché Compartida, se accede a la caché Privada. Este acceso también podría realizarse en paralelo con la caché Compartida, pero a expensas de un consumo adicional que aporta pocos beneficios en la productividad.

La Figura 3 resume las acciones realizadas por el controlador del directorio ante un acceso de coherencia. El directorio propuesto funciona de la siguiente manera:

- Cuando una solicitud de coherencia llega al directorio, el controlador del directorio comprueba primero la caché Compartida ya que es más probable que el resultado de este acceso sea un acierto en esta caché dado que las entradas que almacenan los bloques privados no se vuelven a acceder, salvo si el bloque pasa a ser compartido. Ante un acierto, el controlador actualiza (si es necesario) el vector de presencia, realiza las acciones de coherencia asociadas y cancela el acceso a la NUCA (dependiendo del estado del bloque). Ante un fallo en la caché Compartida, el controlar mira en la caché Privada. Este comportamiento secuencial tendrá, de media, un impacto despreciable en la productividad ya que la mayoría de los accesos al directorio son debidos a bloques compartidos.
- Un acierto en la caché Privada implica que el bloque es compartido, ya que otro núcleo está accediendo a él. El procesador que accedió a él por primera vez no accede al directorio porque el bloque está almacenado en su caché en un estado privado (por ejemplo M). Así que en este caso, el bloque se mueve a la caché Compartida. Esta forma de proceder asegura que los bloques privados se queden en la caché Privada mientras que los bloques compartidos son filtrados y transportados a la caché Compartida.
- Ante un fallo en el directorio, una entrada es ubicada en la caché Privada para mantener la información del bloque fallido. Como no hay información de coherencia almacenada para ese bloque en ninguna caché de directorio, el bloque no se encuentra actualmente en las cachés del procesador y se asume que es privado. La información del propietario (el procesador solicitante) es actualizado.
- En la propuesta implementada, cuando una entrada de bloque es reemplazada de cualquiera de las dos cachés de directorio se marcha y no se permite el movimiento hacia la otra caché.

La propuesta reduce el área de diseño con respecto a las cachés convencionales con un mismo número de entradas usando la misma tecnología ya que las entradas en la caché Privada son mucho más estrechas. Adicionalmente el consumo también se reduce ya que se realizan accesos secuenciales a una estructura caché de menor tamaño. En cualquier caso, el uso de dos organizaciones independientes con diferentes objetivos de diseño: velocidad para la Shared y capacidad para la Private, sugiere que usando tecnologías específicas que se centren en estos objetivos de diseño, se podrían mejorar aún más los ahorros de energía y área. Tecnologías de bajo leakage o transistores de mayor tamaño con bajas corrientes de leakage podrían usarse en la caché Privada, que tiene un mayor número de entradas y cuyo tiempo de acceso no es crítico para la productividad. En este trabajo exploramos

TABLA II
PARÁMETROS DEL SISTEMA

Parámetros de memoria	
Jerarquía de caché	No inclusiva
Tamaño del bloque	64 bytes
Cachés L1 datos e instrucciones	64KB, 4 vias (256 cjtos)
Tiempo de acierto en caché L1	2 ciclos
Caché L2 compartida	512KB/banco, 8 vías (1024 cjtos)
Tiempo de acierto en caché L2	2 (etiq.) y 6 (total) ciclos
Caché de directorio	256 cjtos, 4 vías (igual que L1)
Tiempo de acierto en caché de directorio	2 ciclos
Tiempo de acceso a memoria	160 ciclos
Parámetros de la red	
Topología de la red	Malla de 2 dimensiones (4x4)
Técnica de enrutamiento	Determinista X-Y
Tamaño de flit	16 bytes
Tamaño de mensajes	5 flits (datos) y 1 flit (control)
Tiempo de enrutamiento, switch y enlace	2, 2 y 2 ciclos

el uso de la tecnología eDRAM en esta caché que alcanza, como los resultados experimentales mostrarán, unos ahorros considerables de área y leakage.

V. ENTORNO DE SIMULACIÓN

La propuesta ha sido evaluada con una simulación completa del sistema utilizando Virtutech Simics [21] junto con el toolset Wisconsin GEMS [22], que permite una simulación detallada de los sistemas multiprocesador. La red de interconexión se ha modelado usando GARNET [23], un simulador detallado de redes incluido en el toolset de GEMS. Hemos simulado una arquitectura CMP de 16 tiles, aunque también mostramos los valores de escalabilidad para área y consumo hasta para 1024 núcleos. Los valores de los parámetros principales pueden verse en la Tabla II. Usamos el CACTI 6.5 [24] para estimar los tiempos de acceso, requisitos de área y consumo de energía de las diferentes estructuras caché para unos nodos con tecnología de 32nm.

Diferentes configuraciones del directorio PS han sido evaluadas, y sus resultados comparados con los obtenidos por un directorio convencional (configuración single caché) con un ratio de coverage de $1\times$. El ratio de coverage indica el número de entradas del directorio por entrada en la caché del procesador. Para la configuración base el directorio tiene el mismo número de entradas que una caché L1 de procesador ($1\times$). Los directorios PS evaluados varían tantos en este ratio de coverage (desde $1\times$ hasta $0{,}125\times$) como en el ratio entre la caché Compartida y a la caché Privada (1:3 y 1:7). Es decir, el número de entradas en la caché privada es tres y siete veces mayor que el número de entradas en la caché compartida respectivamente. La Tabla III muestra el tiempo de acceso y las características para cada estructura de directorio. Los valores para la caché Privada han sido calculados tanto para la tecnología SRAM como eDRAM. CACTI nos da las latencias en ns y por tanto hemos redondeado estos valores para obtener ciclos de procesador. La caché L2 asumimos que es de 6 ciclos, y el resto de tiempos de accesos se han escalado acorde con esto. Es de notar que las latencias eDRAM son mayores que las latencias SRAM. El número de vías es independiente al ratio de coverage, pero el número de conjuntos se disminuye a la mitad cada vez que reducimos a la mitad el ratio de coverage.

Evaluamos nuestra propuesta con una amplia gama de aplicaciones científicas: Barnes (16K particles), FFT (64K comples doubles), Ocean (514×514 ocean), Radiosity (room, -ae5 5000.0 -en 0-050 -bf 0.10), Radix

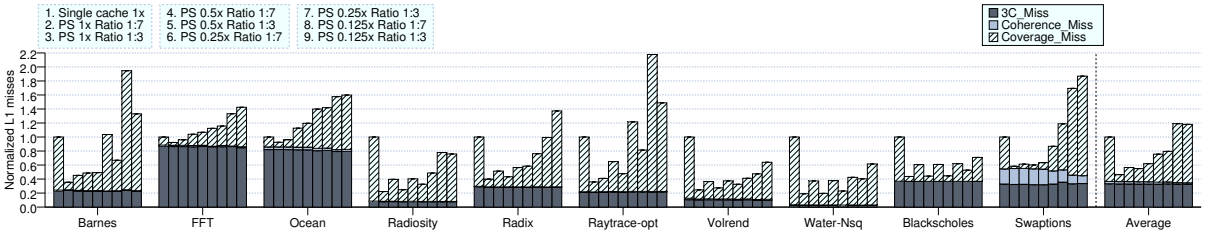
TABLA III
LATENCIAS DEL DIRECTORIO

Caché de directorio	# Vías	1× # Cjtos	Ratio de Coverage 1×	0,5×	0,25×	0,125×
Caché única	4	256	2	2	2	-
Caché compartida 1:3	4	64	2	2	2	2
Caché privada SRAM 1:3	6	128	2	2	2	2
Caché privada eDRAM 1:3	10	128	4	4	3	3
Caché comparida 1:7	4	32	2	2	2	2
Caché privada SRAM 1:7	7	128	2	2	2	2
Caché privada eDRAM 1:7	11	128	4	4	3	3

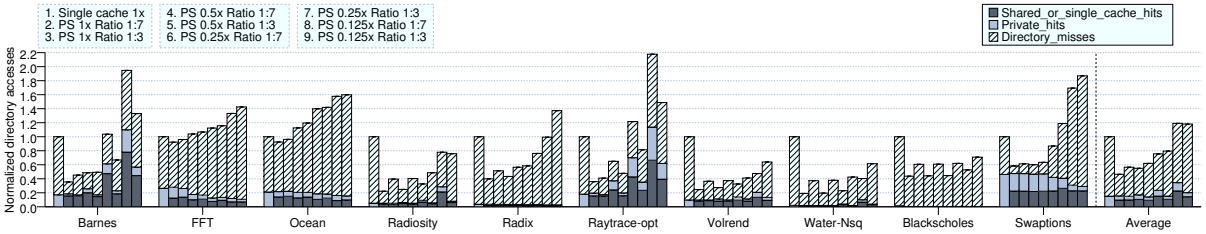
(512 keys, 1024 radix), Raytrace (teapot), Volrend (head), and Water-Nsq (512 molecules) del suite de benchmarks SPLASH-2 [32]. También usamos Blackscholes (simmedium) y Swaptions (simmedium) que pertenecen a las PARSEC [33]. Todos los resultados experimentales recogidos en este trabajo se corresponden con la fase paralela de estos benchmarks.

VI. EVALUACIÓN EXPERIMENTAL: IMPACTO EN EL TIEMPO DE EJECUCIÓN

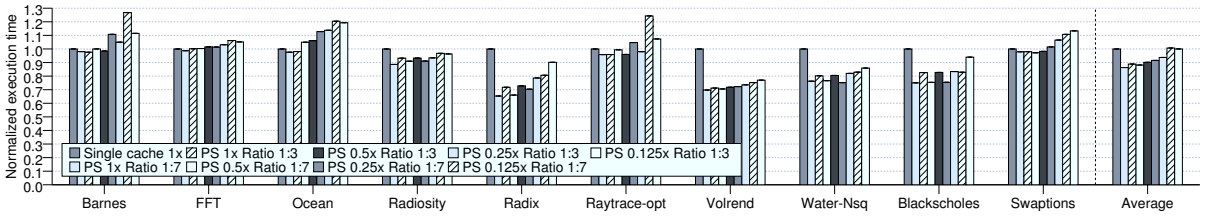
Esta sección analiza los resultados del directorio PS propuesto comparado con un directorio single caché tradicional. Cada vez que una entrada del directorio es expulsada, los mensajes de invalidación son enviados a las correspondientes cachés del procesador para poder seguir manteniendo la coherencia a nivel de caché. Estas invalidaciones causarán fallos de coverage ante siguientes peticiones de memoria a esos bloques, impactando negativamente sobre la productividad final. La Figura 4(a) muestra los fallos de la caché L1 clasificándolos en 3C (*cold* o *compulsory*, *capacity* y *conflict*), Coherencia y Coverage. Una organización eficiente del directorio puede eliminar la gran mayoría de los fallos de coverage como se puede apreciar en la Figura 4(a). Ya que la mayoría de los bloques accedidos por las aplicaciones son privados y la caché de directorio privada tiene una asociatividad adicional sobre el directorio convencional, el directorio PS, evita fallos de conflicto de directorio causados por bloques privados (84,2 % y 68,2 % para los ratios 1:7 y 1:3 respectivamente) con el mismo ratio de coverage. Por otro lado, podemos optar por reducir el tamaño del directorio PS y obtener un número de fallos de coverage similar al de una caché individual. Los resultados muestran que podemos reducir el directorio PS hasta $4\times$ para, incluso así, tener menos fallos de coverage que una caché de directorio convencional. La productividad en una caché de directorio multinivel se puede definir como el número de peticiones de coherencia que encuentran la información de coherencia requerida en el directorio, es decir, el número de aciertos en el directorio. La Figura 4(b) presenta el ratio de aciertos clasificado por cada organización de caché. Como era de esperar, la caché Privada muestra presenta un ratio de aciertos bastante pobre a pesar de que su número de entradas es mucho mayor ($3\times$ y $7\times$ veces las entradas de la caché Compartida), y la mayoría de los aciertos son en la caché Compartida, que se corresponde con la estructura de directorio más pequeña y rápida. Hay que recordar que cada acierto se corresponde con un bloque privado que pasa a ser compartido. Aunque en el ratio 1:7 podría parecer que la caché Compartida es demasiado pequeña, especialmente para ratios de coverage bajos, consigue de media unos resultados mejores que los del



(a) Fallos de caché L1 normalizados.



(b) Accesos al directorio normalizados.



(c) Tiempo de ejecución normalizado.

Fig. 4. Prestaciones normalizadas respecto a una caché convencional.

TABLA IV

ÁREA (IN $mm^2 * 1000$) DE LAS DISTINTAS CONFIGURACIONES PS PARA 16 NÚCLEOS FRENTE UN DIRECTORIO $1\times$ SINGLE CACHÉ.

Coverage	Directorio	Privada	Compartida	Privada	Total	Área (%)
$1\times$	Single		19,51	–	19,51	100,00 %
	PS 1:3	SRAM	6,33	9,50	15,83	81,15 %
	PS 1:7	SRAM	3,28	11,08	14,37	73,65 %
	PS 1:3	eDRAM	6,33	8,22	14,56	74,61 %
	PS 1:7	eDRAM	3,28	9,60	12,88	66,02 %
$0{,}5\times$	PS 1:3	eDRAM	3,28	4,80	8,09	41,47 %
	PS 1:7	eDRAM	1,74	4,80	6,55	33,60 %
$0{,}25\times$	PS 1:3	eDRAM	1,74	3,01	4,76	24,39 %
	PS 1:7	eDRAM	0,84	3,01	3,85	19,76 %

TABLA V

ENERGÍA ESTÁTICA Y DINÁMICA DE LAS CONFIGURACIONES PS PARA 16 NÚCLEOS FRENTE UNA $1\times$ CACHÉ ÚNICA DE DIRECTORIO.

Configuraciones			Consumo estático (mW)			Energía lectura (pJ)		
Coverage	Directorio	Privada	Compartida	Privada	Total	Compartida	Privada	Total
$1\times$	Single		4,2346	–	4,2346	0,0048	–	0,0048
	PS 1:3	SRAM	1,1877	2,2572	3,4450	0,0027	0,0028	0,0055
	PS 1:7	SRAM	0,6404	2,6334	3,2739	0,0016	0,0032	0,0049
	PS 1:3	eDRAM	1,1877	0,5123	1,7001	0,0027	0,0067	0,0094
	PS 1:7	eDRAM	0,6404	0,5977	1,2382	0,0016	0,0078	0,0094
$0{,}5\times$	PS 1:3	eDRAM	0,6404	0,4114	1,0518	0,0016	0,0035	0,0052
	PS 1:7	eDRAM	0,3650	0,4799	0,8450	0,0010	0,0041	0,0052
$0{,}25\times$	PS 1:3	eDRAM	0,3650	0,3276	0,6927	0,0010	0,0027	0,0037
	PS 1:7	eDRAM	0,2181	0,3822	0,6003	0,0007	0,0032	0,0039

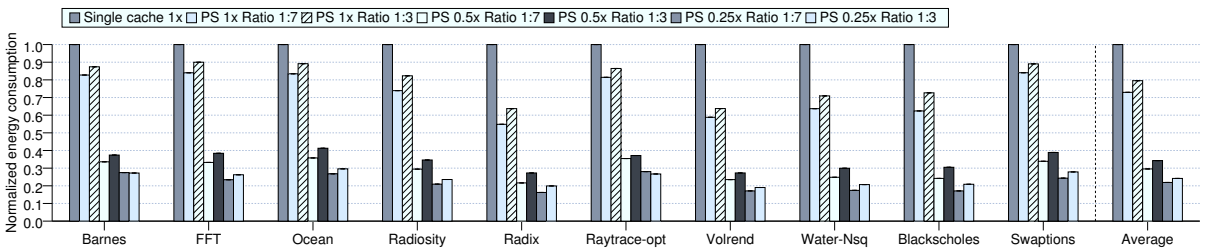


Fig. 5. Energía consumida por el directorio normalizado con un single-caché directory.

ratio 1:3, con la única excepción de un *coverage* 0,125×.

Reducciones en el número de fallos de coverage y en las latencias de acceso al directorio se traducen en mejoras en el tiempo de ejecución. La Figura 5(c) muestra estos resultados. Como se puede ver, el directorio PS con un ratio de coverage de 1× (el mismo que una caché de directorio individual) reduce el tiempo de ejecución en una media de 13,6 % y 11,1 % para los ratios 1:7 y 1:3 respectivamente. Alternativamente, si la reducción del área de silicio es el objetivo del diseño, uno puede optar por reducir el overhead del área del directorio sin perder prestaciones con respecto a un directorio convencional. Este hecho se puede apreciar en la Figura 4(c), donde el directorio PS consigue las mismas prestaciones que una caché de directorio convencional con 8 veces menos entradas.

VII. Conclusiones

El creciente número de núcleos en los CMPs futuros requiere nuevas estructuras de coherencia que puedan escalar en área y energía. Los directorios sparse son la opción preferida ya que cumplen con ambas condiciones para un número bajo o medio de núcleos. Desafortunadamente, el consumo y área en este tipo de directorios, principalmente a causa del vector de presencia, crece cuadráticamente con el número de núcleos haciendo esta elección de diseño prohibitiva. En este trabajo proponemos el directorio PS, que utiliza dos estructuras de caché de directorio diseñadas para satisfacer los requisitos de los bloques que almacenan: la caché Compartida centrada en almacenar los bloques compartidos es pequeña y rápida, y la caché Privada considerablemente más grande centrada en almacenar los bloques privados, que son los que más abundan en las cargas actuales, y que no almacena el vector de presencia. Esta caché Privada actúa como un filtro para los bloques compartidos a los que se les permite trasladarse hasta la caché Compartida mientras que los bloques privados permanecen en esta estructura. Los resultados experimental muestran que, comparado con una caché de directorio individual con el mismo número de entradas, el directorio PS mejora las prestaciones en un 14 % para 16 núcleos a raiz de este trato diferente para los bloques privados y compartidos, mientras que se reduce el área en un 26,35 % principalmente porque el vector de presencia no se almacena para los bloques privados. Adicionalmente, cuando se considera la tecnología eDRAM esta reducción aumenta hasta un 33,98 %. En cuanto al consumo de energía, se consiguen reducciones de un 27 %, que aumenta significativamente con el número de núcleos. Finalmente, la propuesta PS permite reducir hasta 8 veces el número de entradas del directorio (ratio de coverage 0.125x) mientras que se mantienen las prestaciones de una caché de directorio convencional.

Agradecimientos

El presente trabajo ha sido financiado mediante el proyecto CICYT TIN2009-14475-C04.

Referencias

[1] Shane Bell, Bruce Edwards, and John Amann, et al, "TILE64™ processor: A 64-core SoC with mesh interconnect," in *IEEE Int'l Solid-State Circuits Conference (ISSCC)*, Jan. 2008, pp. 88–598.

[2] 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.

[3] Hongzhou Zhao, Arrvindh Shriraman, Sandhya Dwarkadas, and Vijayalakshmi Srinivasan, "SPATL: Honey, i shrunk the coherence directory," in *20th Int'l Conference on Parallel Architectures and Compilation Techniques (PACT)*, Sept. 2011, pp. 148–157.

[4] Richard E. Matick and Stanley E. Schuster, "Logic-based eDRAM: Origins and rationale for use," *IBM Journal of Research and Development*, vol. 49, no. 1, pp. 145–165, 2005.

[5] Ron Kalla, Balaram Sinharoy, William J. Starke, and Michael Floyd, "Power7: IBM's Next-Generation Server Processor," *IEEE Micro*, vol. 30, pp. 7–15, 2010.

[6] J M. Tendler, J S. Dodson, J S. Fields, H. Le, and B. Sinharoy, "POWER4 System Microarchitecture," *IBM Journal of Research and Development*, vol. 46, no. 1, pp. 5–25, 2002.

[7] B. Sinharoy, R N. Kalla, J M. Tendler, R J. Eickemeyer, and J B. Joyner, "POWER5 System Microarchitecture," *IBM Journal of Research and Development*, vol. 49, no. 4/5, pp. 505–521, 2005.

[8] Alejandro Valero, Julio Sahuquillo, Salvador Petit, Vicente Lorente, Ramon Canal, Pedro López, and José Duato, "An Hybrid eDRAM/SRAM Macrocell to Implement First-Level Data Caches," in *Proceedings of the 42th Annual IEEE/ACM International Symposium on Microarchitecture*, New York, NY, USA, 2009, pp. 213–221, ACM.

[9] Xiaoxia Wu, Jian Li, Lixin Zhang, Evan Speight, Ram Rajamony, and Yuan Xie, "Hybrid Cache Architecture with Disparate Memory Technologies," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, New York, NY, USA, 2009, pp. 34–45, ACM.

[10] Manish Shah, Jama Barreh, and Jeff Brooks, et al, "UltraSPARC T2: A highly-threaded, power-efficient, SPARC SoC," in *IEEE Asian Solid-State Circuits Conference*, Nov. 2007, pp. 22–25.

[11] Alberto Ros, Manuel E. Acacio, and José M. García, "A scalable organization for distributed directories," *Journal of Systems Architecture (JSA)*, vol. 56, no. 2-3, pp. 77–87, Feb. 2010.

[12] 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.

[13] M. Ferdman, P. Lotfi-Kamran, K. Balet, and B. Falsafi, "Cuckoo directory: A scalable directory for many-core systems," pp. 169 –180, feb. 2011.

[14] Anoop Gupta, Wolf-Dietrich Weber, and Todd C. Mowry, "Reducing memory traffic requirements for scalable directory-based cache coherence schemes," in *Int'l Conference on Parallel Processing (ICPP)*, Aug. 1990, pp. 312–321.

[15] Manuel E. Acacio, José González, José M. García, and José Duato, "A new scalable directory architecture for large-scale multiprocessors," in *7th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Jan. 2001, pp. 97–106.

[16] Guoying Chen, "Slid - a cost-effective and scalable limited-directory scheme for cache coherence," in *5th Int'l Conference on Parallel Architectures and Languages Europe (PARLE)*, June 1993, pp. 341–352.

[17] Brian W. O'Krafka and A. Richard Newton, "An empirical evaluation of two memory-efficient directory methods," in *17th Int'l Symp. on Computer Architecture (ISCA)*, June 1990, pp. 138–147.

[18] Manuel E. Acacio, José González, José M. García, and José Duato, "A two-level directory architecture for highly scalable cc-NUMA multiprocessors," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 16, no. 1, pp. 67–79, Jan. 2005.

[19] Daniel Sanchez and Christos Kozyrakis, "Scd: A scalable coherence directory with flexible sharer set encoding," in *18th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2012, pp. 129–140.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

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