Uso de Combinación de Vías para mejorar la Eficiencia de las Cachés de Directorio

Rubén Titos-Gil, Antonio Flores, Ricardo Fernández-Pascual, Alberto Ros y Manuel E. Acacio [1]

Resumen— **El mercado de los procesadores de propósito general pone a nuestra disposición hoy día chips con cientos de núcleos de procesamiento, al tiempo que aparecen los primeros prototipos con miles de núcleos. El mantenimiento de la coherencia ante un número de núcleos tan elevado en dichas arquitecturas *manycore* exige un cuidadoso diseño del directorio, usado para llevar cuenta de la localización de los bloques de memoria en los niveles privados de caché de cada núcleo. En este trabajo proponemos una organización del directorio basada en el concepto de *combinación de vías*. En concreto, nuestra propuesta emplea un puntero por entrada, que es óptimo para el caso común en el que un bloque tiene un único compartidor. Para aquellas direcciones de bloques que requieren más de un puntero, hemos observado que en la mayoría de casos existen vías vacías en el mismo conjunto que pueden ser utilizadas para almacenar más punteros. De esta forma, nuestra propuesta minimiza la sobrecarga de almacenamiento sin perder la flexibilidad para adaptarse a diferentes grados de compartición, y sin la complejidad de otras técnicas previamente propuestas. Mediante simulaciones detalladas de una arquitectura con 128 núcleos, mostramos que el directorio de vías combinadas consigue un rendimiento muy cercano al de un directorio *sparse* que usa de vectores de bits, y supera el de otras propuestas escalables del estado del arte.**

Palabras clave— **Arquitecturas multinúcleo, protocolo de coherencia de cachés, caché de directorio, asociatividad, combinación de vías, rendimiento, energía.**

I. Introducción

La tecnología disponible a día de hoy ya permite la comercialización de arquitecturas multinúcleo con cerca de cien núcleos de procesamiento, como los procesadores con 72 núcleos Knights Landing [1] y Tile GX8072 [2], de Intel y Tilera, respectivamente, e incluso se están realizando prototipos de investigación con miles de núcleos en un único chip, como el chip KiloCore desarrollado en UC Davis [3].

En estas arquitecturas con un gran número de núcleos de procesamiento, los mecanismos a través de los cuales dichos núcleos se comunican y sincronizan constituyen elementos clave del diseño. Si la tendencia actual se mantiene, las arquitecturas multinúcleo que están por venir seguirán empleando el modelo de memoria compartida e implementarán a nivel hardware un mecanismo que asegure la coherencia de los niveles privados de caché de cada núcleo [4]. De esta forma, comunicación y sincronización (esta última implementada normalmente usando posiciones de la memoria compartida) ocurrirán bajo el control del protocolo de coherencia de cachés.

La mejor manera de asegurar la coherencia de las cachés privadas cuando el número de núcleos de procesamiento es grande es a través de un protocolo de coherencia de cachés basado en directorio. Aún así, el mantenimiento de la coherencia en arquitecturas con cientos de núcleos exige un diseño muy cuidadoso de la estructura de directorio empleada para registrar la identidad de las cachés privadas que almacenan copia de cada uno de los bloques de memoria. Las arquitecturas de directorio basadas en etiquetas replicadas (*duplicate tag directories*) que fueron utilizadas en algunos de los procesadores multinúcleo de primera generación [5] son simplemente irrealizables cuando el número de núcleos es grande, dado que el grado de asociatividad requerido crece con el número de núcleos. Es por ello por lo que la mayoría de las propuestas actuales están basadas en lo que se denomina una estructura de directorio dispersa (*sparse directory* [6]). Este tipo de estructuras se organizan como una caché asociativa típica, en la que cada entrada registra la lista de compartidores para el bloque de memoria que tiene asignado, y permite implementaciones más escalables.

Dos son los aspectos que determinan los requisitos de área de un directorio disperso [7]: El número total de entradas que tiene y el tamaño en bits de cada entrada. Lo primero condiciona el número máximo de direcciones que puede almacenar el directorio en un momento dado, y por lo tanto tiene un efecto directo sobre la cantidad de bloques de memoria distintos que pueden estar en las cachés privadas. El término cobertura (*coverage*) suele ser utilizado para hacer referencia al número total de entradas de directorio con respecto al número total de entradas en el último nivel de cachés privadas. Una cobertura del 100 % significa que el directorio dispone de entradas suficientes para almacenar información de compartición para todos los bloques de memoria que en potencia pueden ser almacenados en las cachés privadas. Este grado de cobertura del 100 % es imprescindible para poder aprovechar dichas cachés privadas en presencia de cargas secuenciales, y trabajos anteriores (como por ejemplo [8]) demuestran que suele también ser suficiente para cargas paralelas.

Mientras que el grado de cobertura no depende del número de núcleos, y por tanto, no representa un obstáculo en la escalabilidad de la estructura de directorio, la cantidad de bits por entrada (tamaño de cada entrada de directorio) sí que lo es. El tamaño de cada entrada de la entrada de directorio depende fundamentalmente de la forma en la que se organiza el código de compartición, el cual lleva registro de la lista de compartidores de cada bloque. Un vector de bits es un claro ejemplo de un código de compartición no escalable, ya que su tamaño

[1]Departamento de Ingeniería y Tecnología de Computadores, Universidad de Murcia, e-mail: {rtitos, aflores, rfernandez, aros, meacacio}@ditec.um.es

se incrementa linealmente con el número de núcleos. Representaciones alternativas como el uso de punteros limitados [9], [10] o códigos de compartición comprimidos [6], [11], resuelven el problema a costa de incrementar el número de mensajes de coherencia o la tasa de fallos en las cachés privadas.

Además, es bien sabido que el grado de compartición de una aplicación varía entre bloques de memoria y también a lo largo del tiempo, de manera que no existe una organización del código de compartición óptima para todos los casos. De forma ideal, cada entrada de directorio debería tener suficiente flexibilidad para poder adaptarse a situaciones distintas. Varios trabajos anteriores han demostrado que una fracción significativa de las entradas de directorio (casi el 90 % en algunos casos), tienen asignados bloques privados, para los cuales un único puntero es suficiente. Se ha visto también que para la mayoría de las entradas que tienen asignados bloques compartidos, el número de compartidores es muy pequeño (dos o tres). El resto de entradas (pocas) han de dar soporte a bloques de memoria con un número grande de compartidores, pero el número de estas no se incrementa con el tamaño del sistema [12]. Obviamente, si se trata de una carga secuencial, prácticamente todas las entradas de directorio estarían registrando un único compartidor.

En este trabajo se propone una nueva arquitectura de directorio disperso, denominada *WC-dir*, que ha sido diseñada considerando el caso común, es decir, un puntero por entrada de directorio. De esta forma, WC-dir se amolda de manera perfecta a las necesidades de las cargas secuenciales. Para cargas paralelas, sin embargo, WC-dir hace uso de una observación no explotada hasta ahora y es que cuando se requiere más de un puntero para un bloque, pueden encontrarse entradas libres –y por tanto punteros sin usar– en el mismo conjunto del directorio disperso. Estas entradas libres surgen como consecuencia de la presencia de bloques de memoria compartidos en estas aplicaciones (estos bloques residen en varias cachés privadas y dado que la cobertura es del 100 %, hacen que aparezcan entradas libres en el directorio). En estos casos –pocos, como ya hemos comentado– WC-dir aplica el concepto propuesto en este trabajo de *combinación de vías* a través del cual pueden usarse los punteros disponibles en varias vías para registrar los compartidores de un mismo bloque de memoria. De esta forma, la combinación de vías permite ver cada conjunto de un directorio disperso como un *pool* de entradas, las cuales son repartidas dinámicamente y según se necesite, entre las diferentes direcciones que se mapean a dicho conjunto. El resultado es una estructura de directorio *diseñada para el caso común*, que es capaz de *adaptarse a grados de compartición cambiantes*, *más sencilla* que propuestas adaptativas anteriores como [13], [8], [14], que tiene *menores requisitos de área*, y con un *rendimiento muy cercano* a un directorio disperso no escalable basado en el uso de vectores de bits.

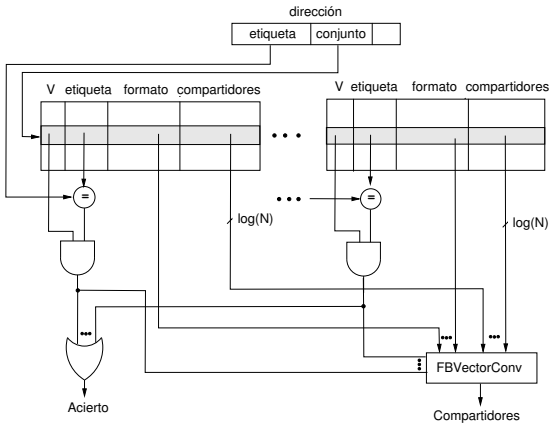

Fig. 1
Implementación del directorio de vías combinadas.

II. WC-dir

El directorio distribuido de vías combinadas (a partir de ahora, *WC-dir*) almacena la información de los bloques compartidos entre los niveles privados de la jerarquía de caché en un multiprocesador. La estructura del WC-dir (Fig. 1) es casi idéntica a la de una caché de directorio asociativa de N-vías. Cada dirección se mapea a un único conjunto de la caché, mientras que la información de compartición, si está presente, puede almacenarse en cualquiera de las N entradas del conjunto. Sin embargo, al contrario que en una caché convencional, WC-dir permite usar *múltiples* entradas del conjunto para la misma dirección, de tal manera que un acceso al WC-dir puede dar como resultado cero, uno o más *aciertos de etiqueta*. En este último caso, la información de compartición almacenada en cada entrada coincidente se *combina* para generar un vector de bits completo de los compartidores de la dirección solicitada. Tal y como se muentra en la Fig. 1, WC-dir reemplaza el multiplexor N:1 normalmente utilizado en una caché asociativa de N-vías (el cual selecciona los datos de la entrada coincidente) por una unidad combinacional denominada *FBVectorConv* cuyo propósito es fundir la información de compartición proveniente de todas las entradas coincidentes.

Nuestro diseño se basa en la observación de que la mayoría de los bloques sólo presentan unos pocos compartidores, a menudo solamente uno. La prevalencia de entradas con un único compartidor (i.e., datos privados) es intuitivo en cargas multiprogramadas, pero incluso en aplicaciones paralelas la mayor parte de las entradas del directorio también contienen bloques privados. Además, el caso común para bloques compartidos es que la mayoría de ellos sólo lo sean por dos o tres compartidores. En este escenario, los directorios distribuidos tradicionales con vectores de bits que se usan para codificar a los compartidores presentan una pobre utilización del área dedicada a almacenar la información de compartición.

Otro factor importante para entender nuestro diseño es que cuando dos o más cachés privadas mantienen una copia de un bloque, sólo se necesita una

entrada en el directorio. Esto significa que, para cada dirección presente en el nivel privado de la caché, en un directorio con una cobertura del 100 % debe de haber una entrada libre por cada compartidor diferente del primero. WC-dir aprovecha dichas entradas libres cuando las mismas están en el mismo conjunto para aquellas direcciones cuya información de compartición no cabe en una única entrada.

Para aprovechar estas observaciones en un diseño simple, WC-dir permite que las entradas de un mismo conjunto sean combinadas (i.e., usadas por la misma dirección). La información de compartición de cada dirección se codifica usando una o mas entradas mediante el uso de punteros o de vectores de bits de grano grueso (*coarse vector*). En la representación *coarse vector* [6], cada bit indica si el nodo o los nodos que representa mantienen una copia de la dirección (bit a 1) o si ninguno de ellos guardan en dicho instante el bloque indicado por la dirección en sus cachés privadas (bit a 0). Por tanto, esta representación da lugar a pérdida de precisión ya que, normalmente, el número de compartidores será superior al real.

Las entradas de WC-dir cambian del formato por defecto basado en punteros al formato *coarse vector* solamente cuando los recursos del directorio son insuficientes para mantener una información de compartición exacta. En general, WC-dir cambia dinámicamente la cantidad de almacenamiento dedicada a cada código de compatición en un intento de maximizar la utilización del directorio y la precisión de la información almacenada al mismo tiempo que mantiene bajo el sobrecoste en área y la complejidad de operación.

La posibilidad de WC-dir de combinar entradas del mismo conjunto es independiente de la representación empleada para almacenar el conjunto de compartidores de una dirección dada. De hecho, el formato en el que se almacena el código de compartición de una dirección puede cambiar con el tiempo, dependiendo del número de entradas asignadas a dicha dirección, tal y como se indicó anteriormente. El formato de cada entrada se codifica mediante un bit extra, denominado *bit de formato*. Las entrads en formato puntero contienen un puntero a un compartidor, mientras que las entradas en formato *coarse vector* contienen una porción del vector de compartidores. Esta información de compartición está almacena de forma conjunta en las entradas combinadas y puede ser decodificada usando un circuito combinacional.

La Fig. 2 ilustra el compartimiento de WC-dir para el caso de un conjunto de 4 vías y 128 nodos. Así, cada entrada del conjunto contiene un puntero de 7 bits de tal manera que, cuando se combinan las 4 entradas, se puede seguir el rastro de hasta cuatro compartidores usando el formato de puntero, estando disponibles hasta 28 bits para formar un vector de grano grueso.

Cada vez que una nueva dirección se inserta en el directorio, el formato de la misma se establece a puntero. La Fig. 2 (a) muestra un conjunto que contiene dos entradas recién insertadas: *dirA* y *dirB*, ambas en formato puntero y con un único compartidor. Para añadir nuevos compartidores a una entrada existente se utiliza alguna de las entradas disponibles en el conjunto, combinándolas para almacenar el código de compartición, tal y como se muestra en la Fig. 2 (b). Cuando todas las entradas de un conjunto están ocupadas (por una o más direcciones) y en formato puntero, no podemos insertar más compartidores en el directorio sin primero hacer espacio en el conjunto. Dado que expulsar una dirección del directorio da lugar a invalidaciones en la cachés privadas que pueden perjudicar al rendimiento al causar fallos adicionales de caché, WC-dir siempre intenta minimizar los reemplazos de direcciones a costa de reducir la precisión del código de compartición. Así, sólo ocurren reemplazos de direcciones cuando se inserta una nueva dirección en un conjunto lleno en donde cada entrada está ocupada por una dirección diferente. En este caso se utiliza un algoritmo de reemplazo LRU para seleccionar la víctima. Nótese que este comportamiento asegura que las direcciones almacenadas en el WC-dir son las mismas que las que serían si hubiéramos utilizado un directorio con vector de bits.

Si ya existe una entrada combinada cuando se inserta una nueva dirección en un conjunto lleno, WC-dir hace espacio reduciendo el número de entradas asignadas a una de las direcciones con entradas combinadas, reduciendo de esta manera la precisión de su información de compartición. La Fig. 2 (c) muestra como antes de insertar un nuevo compartidor de *dirB*, *dirA* debe cambiar de formato puntero usando tres entradas a formato *coarse vector* usando dos entradas, liberando una de sus entradas (obsérvese que no se expulsa ninguna dirección). Aunque en este ejemplo sólo existe un cadidato, en la práctica se podría emplear varías heurísticas para seleccionar a la víctima entre las direcciones candidatas. WC-dir opta simplemente por LRU, pero priorizando aquellos candidatos en formato de vector de grano grueso sobre los que están en formato puntero para así mantener la precisión de los códigos de compartición en tantas direcciones como sea posible (tal y como se observa en la Fig. 2 (d)). Una vez seleccionada, si la dirección de la víctima ya está en formato *coarse vector* y tiene asignadas K entradas combinadas, el código de compartición combinado se recalcula de acuerdo con su nuevo tamaño ($K-1$ entradas) para obtener una entrada libre para la nueva dirección. Si v todavía usa punteros, se cambia al formato *coarse vector*, calculado a partir de los K punteros iniciales, que es almacenado en las $K-1$ entradas restantes.

Como se comentó anteriormente, insertar un nuevo compartidor no causa expulsiones de direcciones, incluso en el caso en que el conjunto esté lleno. Si la dirección ya se encuentra en formato de vector de grano grueso, ponemos a uno el bit correspondiente, tal y como se muestra en la Fig. 2 (e). En caso de encontrarse en formato puntero con K entradas, se cambia de formato, codificándose los punteros exis-

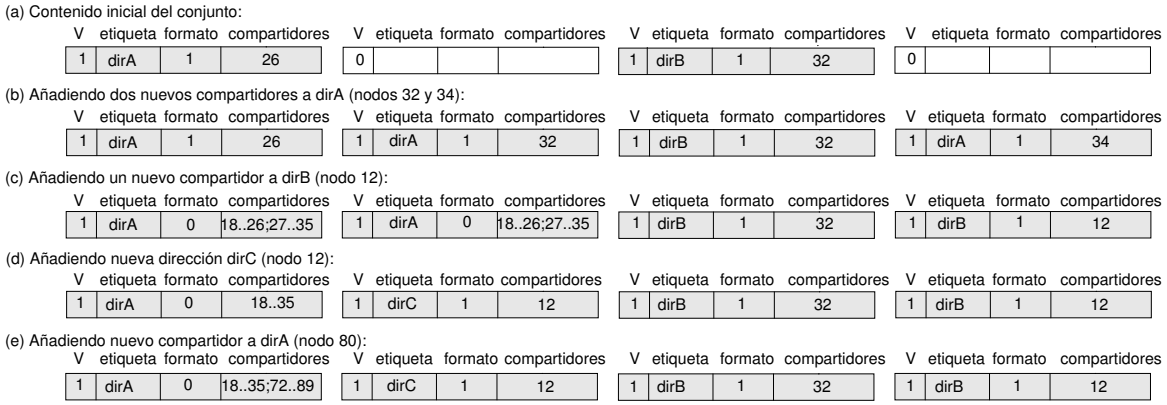

Fig. 2
WC-dir: Ejemplo de operación.

TABLA I
Parámetros del sistema.

Parámetros de la memoria	
Tamaño de bloque	64 bytes
Caché L1 (datos e instr.)	32 KiB, 4 vías
Latencia de acceso a L1	1 ciclo
Caché L2 (datos e instr.)	128 KiB, 8 vías
Latencia de acceso a L2	10 ciclos
Caché L3 (compartida)	1024 KiB/celda, 32 vías
Latencia de acceso a L3	20 ciclos
Organización de las cachés	L2 inclusiva, L3 no-inclusiva
Tamaño del directorio (SCD75)	1536 entradas, 3 vías (cobertura 75 %)
Tamaño del directorio (SCD)	2048 entradas, 4 vías (cobertura 100 %)
Tamaño del directorio (resto)	2048 entradas, 8 vías (cobertura 100 %)
Latencia del directorio	5 ciclos
Tamaño de la dirección física	48 bits
Tiempo de acceso a memoria	200 ciclos
Parámetros de la red	
Topología y Encaminamiento	Malla 2-D (16×8), X-Y
Tamaño de flit	16 bytes
Tamaño de mensaje	5 flits (datos), 1 flit (control)
Latency de enlace	2 ciclos
Ancho de banda	1 flit por ciclo

tentes más el compartidor y almacenándose el resultado en las mismas K entradas.

III. Entorno de evaluación

La evaluación de los distintos esquemas para el directorio de coherencia considerados en este trabajo se ha realizado mediante los simuladores PIN [15] y GEMS 2.1 [16], conectándolos de forma similar a como se propone en [17]. PIN obtiene todos los accesos a los datos accedidos por las aplicaciones, así como la sincronización entre los hilos de la application. GEMS modela la jerarquía de memoria y calcula la latencia de acceso a memoria de cada petición del procesador. La red de interconexión se ha modelado con el simulador Garnet [18]. La arquitectura simulada corresponde a un procesador multinúcleo (*tiled*-CMP) con 128 núcleos (uno por celda). Todas las configuraciones evaluadas implementan estados MESI en las cachés privadas. Los principales parámetros de simulación se muestran en la tabla I.

Para la evaluación de este artículo hemos implementado en GEMS cinco configuraciones para el directorio de coherencia que llamamos BV, LP1, SCD, SCD75 y WC1. BV representa un directorio disperso que utiliza como código de compartición el vector de bits, que como ya hemos comentado no es escalable. LP1 es una implementación de Dir_iCV [6]. En este caso cada entrada dispone de 1 puntero que se usa para almacenar la identidad del único compartidor en bloques privados, o que alternativamente se utiliza como un vector de bits de grano grueso (coarse bit-vector) para bloques con más de un compartidor. SCD es una implementación del directorio SCD [8] que emplea una z-cache de 4 vías que explora 3 niveles a la hora de buscar candidatos para reemplazar (equivale aproximadamente a una caché asociativa de 52 vías). SCD75 es otra configuración de SCD que proporciona una cobertura del 75 % pero cuyos requisitos de área están cercanos a los de LP1 y WC1. Para ello, SCD75 utiliza una z-cache de 3 vías que explora 4 niveles (equivale aproximadamente a una caché asociativa de 45 vías). Por último, WC1 es una implementación de WC-dir que utiliza entradas con un único puntero. BV y LP1 emplean reemplazos silenciosos para bloques compartidos (no se envía notificación al directorio cada vez que se reemplaza un bloque en estado compartido) mientras que WC1, SCD y SCD75 usan reemplazos ruidosos en estos casos[1].

Además, hemos usado una gran variedad de programas de prueba de los conjuntos de aplicaciones PARSEC 3.0 [20] and SPLASH-3 [21]. En particular, hemos incluido todas las aplicaciones de estas *suites* que escalan hasta 128 hilos en nuestro sistema simulado. Para poder alcanzar unos tiempos de simulación razonables, para cada aplicación hemos usado el tamaño de problema más pequeño (*simsmall* o *simmedium*) para el cual el tiempo de ejecución con 128 hilos es menor que el alcanzado con 64 hilos. Freqmine utilizas OpenMP y no pudo ser portada a nuestra infraestructura de simulación. Las aplicaciones incluidas son Barnes, Cholesky, Fft, Fmm, Lu_cb, Ocean_cp, Ocean_ncp, Radix, Water_nsqared y Water_spatial de SPLASH-3, y Blackscholes, Bodytrack,

[1]Para cada configuración hemos evaluado ambas opciones y hemos seleccionado la mejor política para cada caso en función del tiempo de ejecución obtenido [19].

Canneal, Dedup y Vips de PARSEC 3.0. Para todas se ha utilizado el tamaño de entrada *simsmall* salvo para Canneal, Dedup and Vips que usan *simmedium*. El conjunto de programas de prueba resultante contiene aplicaciones con comportamientos distintos y patrones de compartición diferentes. Hemos tenido en cuenta la variabilidad en las aplicaciones paralelas tal y como se describe en [22]. Todos los resultados mostrados en este trabajo corresponden a la parte paralela de las aplicaciones evaluadas.

IV. Resultados

En la tabla II se muestra la cantidad de memoria necesaria para implementar cada una de las estructuras de directorio consideradas en este trabajo. No se incluyen los datos de LP1 puesto que son iguales a los de WC1. Además de para 128 núcleos, se muestran los datos para sistemas mayores y menores para que se aprecie la escalabilidad de cada propuesta. En cada celda, el directorio BV necesita más de 39 KiB para soportar cachés privadas de 128 KiB, mientras que WC1 y LP1 requieren solo $9,3$ KiB gracias al menor tamaño de los códigos de compartición. SCD con la misma cobertura que los demás requiere más área que WC1 y LP1 debido tanto a que el código de compartición es más grande como a que las etiquetas requeridas por la z-cache son mayores. Incluso si se reduce la cobertura de SCD al 75 %, sigue necesitando más memoria que WC1 y LP1 para 128 núcleos. Y si miramos cómo varía la sobrecarga con respecto al número de núcleos en cada caso, comprobamos que solo LP1 y WC1 lo mantienen constante. Esto ocurre porque el tamaño del código de compartición crece al mismo ritmo que se reduce el tamaño de las etiquetas (logarítmicamente).

Cada tipo de directorio hace uso de manera diferente de los recursos de memoria para almacenar la información de compartición de las direcciones presentes en el directorio, lo cual determina el coste para acceder a la información de compartición y la precisión de la misma. En algunos casos, el directorio reducirá la precisión de esta información (codificando siempre un superconjunto del conjunto real de compartidores) a cambio de un mayor coste en cuanto a tráfico de red debido a las invalidaciones. La Fig. 3 muestra la precisión media por dirección a lo largo de toda la ejecución. Tanto BV como SCD tienen una precisión perfecta, aunque SCD lo consigue con menos recursos. Las precisiones alcanzadas por LP1 y WC1 son menores, pero podemos ver como la combinación de vías permite a WC1 alcanzar una precición significativamente mejor que LP1 con los mismos recursos. La mejora de la precisión es mayor en aquellos benchmarks que tienen menos direcciones almacenadas por conjunto

El efecto más directo de la falta de precición de la información almacenada en un directorio es que se envían mensajes de invalidación innecesarios cuando se producen escrituras en líneas compartidas o reemplazos de directorio. Estos mensajes adicionales pueden tener en algunos casos un efecto significativo en el tráfico de la red, como se muestra en la Fig. 4, lo cual afectará al consumo dinámico de energía de la red y a la latencia de los fallos. Aquí vemos que la precisión extra de WC1 con respecto a LP1 proporcionada por la combinación de vías permite que el tráfico de WC1 sea significativamente menor que el de LP1, aunque mayor que el de SCD y BV. Para la mayoría de los benchmarks, el incremento en tráfico no tiene un efecto importante en la latencia de los fallos y, por tanto, no afecta significativamente al tiempo de ejecución. Resulta interesante destacar que aunque la precisión de SCD es perfecta, la diferencia media en cuanto a tráfico es de solo el 10 %, a pesar de que SCD necesita más área. Esto es devido a que SCD sufre algunos reemplazos de directorio más que BV, LP1 o WC1 porque algunas direcciones necesitan ocupar más de una entrada de directorio.

La Fig. 5 muestra el incremento relativo en el tiempo de ejecución normalizado para cada estructura de directorio. En primer lugar, se comprueba que reducir la cobertura de SCD al 75 % para que sus requerimientos de área sean similares a los de LP1 y WC1 tiene un efecto muy negativo en muchos benchmarks (como Canneal o Ocean_cp), de forma que el tiempo medio de ejecución de SCD75 es peor que el de LP1. SCD con cobertura completa obtiene un tiempo de ejecución que es menos de un 5 % peor que BV e incluso obtiene un tiempo de ejecución mejor en algún caso (como FFt y Radix). Esto último es debido a la mayor asociatividad efectiva proporcionada por la z-cache de SCD que elimina algunos fallos por conflicto del directorio que aparecen en esos benchmarks. Por último, el incremento en el tiempo de ejecución de WC1 respecto a BV es de solo el 2 % de media, por lo que es la configuración que más se aproxima al rendimiento del directorio no escalable BV.

V. Conclusiones

Este trabajo propone *WC-dir*, una arquitectura de directorio novedosa cuyo diseño pone el énfasis en el caso común, cuando sólo un puntero por entrada es suficiente para llevar cuenta de los compartidores. De esta forma, WC-dir cubre a la perfección las necesidades de las cargas de trabajo secuenciales. En el caso de cargas paralelas, en las que un puntero no es suficiente, nuestra propuesta se beneficia del hecho de que varias entradas permanecen libres en la mayoría de conjuntos del directorio –una observación hasta ahora no explotada– y aplica el concepto de *combinación de vías* con el fin de proporcionar más espacio para información de compartición a las pocas direcciones del conjunto que así lo requieran. De esta forma, la combinación de vías permite ver cada conjunto de un directorio como un *pool* de entradas que se reservan dinámicamente bajo demanda de las direcciones mapeadas al conjunto, minimizando la sobrecarga de almacenamiento sin necesidad de sacrificar la flexbilidad de adaptación a diferentes grados de compartición.

Nuestro diseño puede derivarse con mínimos cambios a partir de un directorio disperso que use el

TABLA II

Tamaño del directorio y sobrecarga para diferentes configuraciones (los tamaños de LP1 son idénticos a los de WC1).

Nodos	64				128				256				512				1024			
Directorio	BV	SCD	SCD75	WC1	BV	SCD	SCD75	WC1	BV	SCD	SCD75	WC1	BV	SCD	SCD75	WC1	BV	SCD	SCD75	WC1
Etiqueta (bits)	28	36	36	28	27	35	35	27	26	34	34	26	25	33	33	25	24	32	32	24
Código de compartición (bits)	64	11	11	7	128	16	16	8	256	20	20	9	512	28	28	10	1024	37	37	11
Tamaño / Celda (KiB)	23.5	12.3	9.2	9.3	39.3	13.3	9.9	9.3	71.0	14.0	10.5	9.3	134.8	15.8	11.8	9.3	262.5	17.8	13.3	9.3
% sobre L2	17.2	8.9	6.7	6.8	28.6	9.7	7.3	6.8	51.8	10.2	7.7	6.8	98.4	11.5	8.6	6.8	191.6	13.0	9.7	6.8

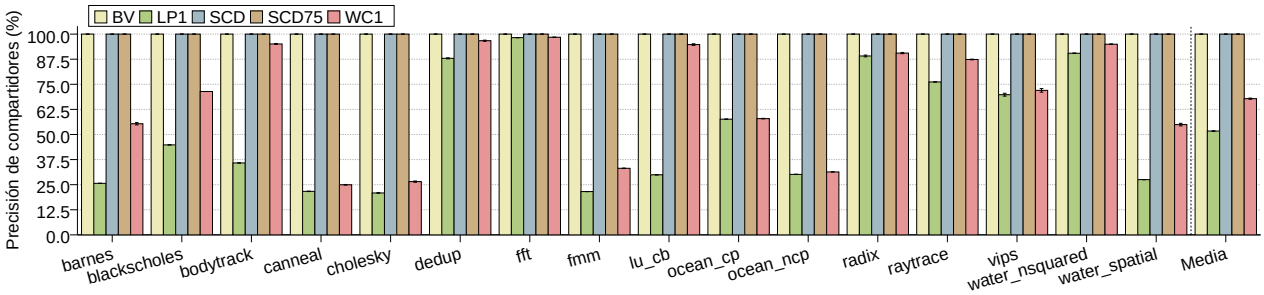

Fig. 3

Precisión por dirección medida como la media para cada dirección del ratio entre el número real de compartidores y el número de compartidores codificados en el directorio. El directorio se muestrea cada 100000 ciclos.

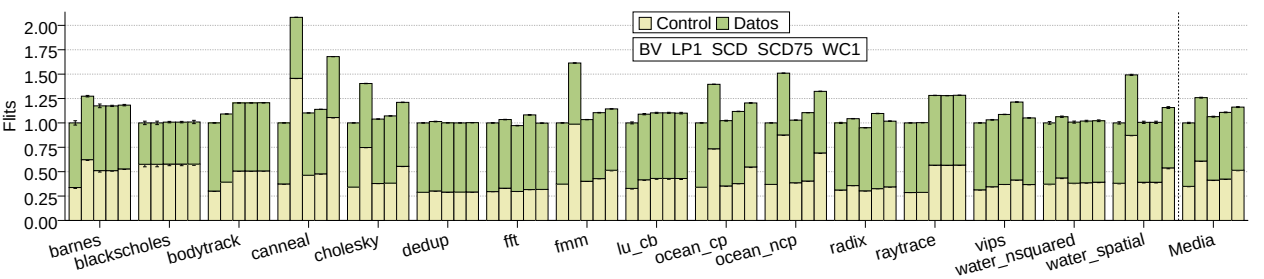

Fig. 4

Tráfico total de la red normalizado.

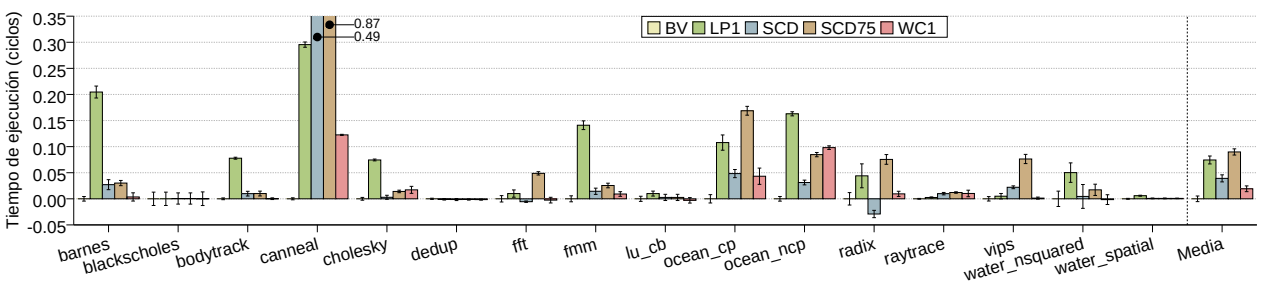

Fig. 5

Incremento del tiempo de ejecución normalizado con respecto a BV.

conocido código de compartición Dir_1CV. Al igual que otras propuestas como SCD, WC-dir puede mantener la lista de compartidores mediante múltiples formatos, desde la representación mediante punteros limitados, hasta *coarse vector* en aquellos casos en los que no quedan entradas libres en el conjunto. Sin embargo, al contrario que SCD, WC-dir consigue dicha flexibilidad sin la notable complejidad adicional que entraña una z-cache, evitando igualmente las re-inserciones iterativas que mantienen al controlador de directorio ocupado durante mayor tiempo. Además, el hecho de que el diseño de WC-dir es similar al de un directorio disperso tradicional permite el uso de algoritmos de reemplazo sencillos y simplifica las operaciones del directorio.

Por medio de simulaciones detalladas de una arquitectura de 128 núcleos, usando un conjunto de benchmarks que exhiben diferentes patrones de compartición, hemos mostrado que WC-dir reduce el tiempo de ejecución promedio comparado con SCD, y prácti-

camente puede igualar el rendimiento obtenido por el directorio no escalable basado en vectores de bits (tan sólo se observa un 2 % de sobrecarga media). En lo referido al área, hemos mostrado que la sobrecarga introducida por WC-dir con respecto a las cachés privadas es menor que la introducida por SCD para 128 núcleos, y lo que es más importante, permanece constante conforme incrementamos el número de núcleos, mientras que en SCD crece aunque lentamente. El único inconveniente que hemos observado es algo de tráfico extra en la red. En particular, WC-dir lo incrementa en un 6 % de media comparado con un SCD con un tamaño (área) similar, y un 10 % comparado con un SCD de igual número de entradas, el cual requiere un 25 % más area. Obsérvese, sin embargo, que el diseño WC1 evaluado en este trabajo pone énfasis en minimizar la sobrecarga en área al tiempo que mantiene el tiempo de ejecución. El tráfico en WC-dir podría reducirse ostensiblemente a cambio de incrementar los requisitos de área, por ejemplo, al duplicar el número de bits por entrada (y por tanto el número inicial de punteros y el tamaño de los vectores de bits de grano grueso), aún conservando las ventajas sobre SCD (menor tiempo de ejecución, menor área –aunque en menor medida– y una implementación más sencilla).

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Economía y Competitividad (MINECO) y la Comisión Europea FEDER mediante el proyecto "TIN2015-66972-C5-3-R", y por la Fundación Séneca-Agencia de Ciencia y Tecnología de la Región de Murcia a través del proyecto "19295/PI/14".

Referencias

[1] Avinash Sodani, Roger Gramunt, Jesus Corbal, Ho-Seop Kim, Krishna Vinod, Sundaram Chinthamani, Steven Hutsell, Rajat Agarwal, and Yen-Chen Liu, "Knights landing: Second-generation intel xeon phi product," *IEEE Micro*, vol. 36, no. 2, pp. 34–46, Mar. 2016.

[2] Matthew Mattina, "Architecture and performance of the tilera TILE-Gx8072 manycore processor," Invited presentation at 21st HotInterconnects Symp., 2013.

[3] Brent Bohnenstiehl, Aaron Stillmaker, Jon Pimentel, Timothy Andreas, Bin Liu, Anh Tran, Emmanuel Adeagbo, and Bevan Baas, "A 5.8 pj/op 115 billion ops/sec, to 1.78 trillion ops/sec 32nm 1000-processor array," in *2016 Symposium on VLSI Technology and Circuits*, June 2016, pp. 1–2.

[4] Milo M. K. Martin, Mark D. Hill, and Daniel J. Sorin, "Why on-chip cache coherence is here to stay," *Communications of the ACM*, vol. 55, no. 7, pp. 78–89, July 2012.

[5] Luiz A. Barroso, Kourosh Gharachorloo, Robert McNamara, Andreas Nowatzyk, Shaz Qadeer, Barton Sano, Scott Smith, Robert Stets, and Ben Verghese, "Piranha: A scalable architecture based on single-chip multiprocessing," in *27th Int'l Symp. on Computer Architecture (ISCA)*, June 2000, pp. 12–14.

[6] Anoop Gupta, Wolf-Dietrich Weber, and Todd C. Mowry, "Reducing memory traffic requirements for scalable directory-based cache coherence schemes," in *19th Int'l Conf. on Parallel Processing (ICPP)*, Aug. 1990, pp. 312–321.

[7] David E. Culler, Jaswinder P. Singh, and Anoop Gupta, *Parallel Computer Architecture: A Hardware/Software Approach*, Morgan Kaufmann Publishers, Inc., 1999.

[8] 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.

[9] Anant Agarwal, Richard Simoni, John L. Hennessy, and Mark A. Horowitz, "An evaluation of directory schemes for cache coherence," in *15th Int'l Symp. on Computer Architecture (ISCA)*, May 1988, pp. 280–289.

[10] Jong H. Choi and Kyu H. Park, "Segment directory enhancing the limited directory cache coherence schemes," in *13th Int'l Parallel and Distributed Processing Symp. (IPDPS)*, Apr. 1999, pp. 258–267.

[11] Shubhendu S. Mukherjee and Mark D. Hill, "An evaluation of directory protocols for medium-scale shared-memory multiprocessors," in *8th Int'l Conf. on Supercomputing (ICS)*, July 1994, pp. 64–74.

[12] Minshu Zhao and Donald Yeung, "Studying the impact of multicore processor scaling on directory techniques via reuse distance analysis," in *21th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2015, pp. 590–602.

[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] Sudhanshu Shukla and Mainak Chaudhuri, "Pool directory: Efficient coherence tracking with dynamic direcory allocation in many-core systems," in *33rd Int'l Conf. on Computer Design (ICCD)*, Oct. 2015, pp. 557–564.

[15] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood, "Pin: Building customized program analysis tools with dynamic instrumentation," in *2005 ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI)*, June 2005, pp. 190–200.

[16] 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.

[17] Matteo Monchiero, Jung Ho Ahn, Ayose Falcón, Daniel Ortega, and Paolo Faraboschi, "How to simulate 1000 cores," *Computer Architecture News*, vol. 37, no. 2, pp. 10–19, July 2009.

[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] Ricardo Fernández-Pascual, Alberto Ros, and Manuel E. Acacio, "To be silent or not: On the impact of evictions of clean data in cache-coherent multicores," *Journal of Supercomputing (SUPE)*, Mar. 2017.

[20] Christian Bienia, *Benchmarking Modern Multiprocessors*, Ph.D. thesis, Princeton University, Jan. 2011.

[21] Christos Sakalis, Carl Leonardsson, Stefanos Kaxiras, and Alberto Ros, "Splash-3: A properly synchronized benchmark suite for contemporary research," in *IEEE Int'l Symp. on Performance Analysis of Systems and Software (ISPASS)*, Apr. 2016, pp. 101–111.

[22] Alaa R. Alameldeen and David A. Wood, "Variability in architectural simulations of multi-threaded workloads," in *9th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2003, pp. 7–18.