Diseño y Evaluación de un Directorio Basado en Distancia en Arquitecturas CMP

Alberto Ros y Manuel E. Acacio [1]

Resumen— **Conforme aumenta el número de núcleos de ejecución en una arquitectura CMP, la organización del directorio empleado para mantener coherentes las cachés privadas de cada uno de los núcleos resulta un aspecto crítico, ya que incide directamente sobre los requerimientos del protocolo de coherencia en cuanto a área y consumo de energía, así como determina su rendimiento. Mientras que una organización de directorio basada en un vector de bits de compartidores resulta viable cuando el número de núcleos es pequeño, no lo es de la misma manera en diseños con muchos núcleos. En estos casos, los códigos de compartición comprimidos pueden reducir significativamente la sobrecarga de memoria introducida por el directorio, y por lo tanto, sus necesidades en cuanto a consumo de energía, si bien también pueden reducir el rendimiento como consecuencia de los mensajes de coherencia innecesarios resultantes. En este trabajo proponemos y evaluamos un nuevo código de compartición comprimido, que hemos denominado Código de Compartición Basado en Distancia (DASC), que puede ser implementado de forma muy eficiente en una arquitectura con muchos núcleos y que tiene unos requisitos muy bajos en cuanto a sobrecarga de almacenamiento se refiere. Dicho código de compartición, que se combina con un esquema dinámico de asignación del nodo origen por parte del SO, se compara con otros que tienen un tamaño similar, obteniendo ventajas significativas en términos de Tamaño (en bits) × Tráfico generado^2 (en bytes) o $S \times T^2$.**

Palabras clave— **Arquitectura CMP, Coherencia de cachés, Directorio, Código de compartición, Mensajes de coherencia innecesarios.**

I. Introducción

LAS arquitecturas multinúcleo que integran varias decenas de núcleos de procesamiento en el mismo chip son ya una realidad comercial. Intel, por ejemplo, comercializa una versión de 60 núcleos de su procesador Intel Xeon Phi [5], y anteriormente, Tilera ya había anunciado su procesador Tile-Gx100 con 100 núcleos de procesamiento [12]. Es de esperar, además, que el número de núcleos por chip siga creciendo al ritmo marcado por las mejoras tecnológicas, y hay quien habla de arquitecturas con varios cientos e incluso miles de núcleos *on-chip* [6].

De cara a organizar convenientemente un número de núcleos tan grande, las arquitecturas basadas en celdas (*tiles*) han sido defendidas como la mejor opción. Una arquitectura multinúcleo basada en celdas se diseña como una colección de bloques idénticos (o muy similares), que se disponen uno al lado del otro siguiendo un patrón regular (formando, por ejemplo, una malla con dos dimensiones). En este tipo de arquitectura, cada celda (*tile*) contiene al menos un núcleo de procesamiento, el cual dispone de varios niveles de caché (algunos privados y un último nivel que suele ser compartido) y una interfaz de red, que permite conectar a dicha celda con sus celdas vecinas en la topología, formando una red de interconexión punto-a-punto escalable. En cuanto al modelo de comunicación, la tendencia actual apunta hacia la utilización del modelo de memoria compartida, con un mecanismo implementado en hardware que asegure la coherencia de los niveles privados de caché de cada núcleo [9].

El protocolo de coherencia de cachés más adecuado para estas arquitecturas multinúcleo es el basado en directorio. A cada línea de memoria se le asigna una celda origen (*home*) que es la encargada de almacenar la información de directorio para la línea y hacia donde se envían los fallos de caché de esa línea desde los distintos núcleos procesamiento. La asignación del nodo origen para cada línea de memoria suele realizarse de forma estática, en hardware, a través de un grupo de bits de la dirección de bloque. Alternativamente, se han propuesto mecanismos de asignación dinámicos más eficientes que pueden implementarse también en hardware [3] o en software, a nivel del sistema operativo [2]. En cuanto a la información de directorio, para cada línea de memoria se almacena en el nodo origen la identidad de los compartidores de la misma en cada momento. Dicha identidad puede almacenarse de forma precisa, haciendo uso, por ejemplo, de un vector de bits con tantos bits como celdas haya en la arquitectura. El gran problema que esta organización tiene es que el tamaño del directorio (número de bits para cada entrada) se incrementa linealmente con el número de celdas, lo que hace que resulte inviable para sistemas grandes. Otras alternativas consiguen reducir la sobrecarga de memoria del directorio y aseguran su escalabilidad a un número de celdas grandes mediante la utilización de códigos de compartición comprimidos, los cuales se basan en codificar un superconjunto del conjunto real de compartidores. El inconveniente que presenta esta última opción es la aparición de mensajes de coherencia innecesarios como consecuencia de la codificación por exceso que se lleva a cabo.

En este trabajo presentamos DASC, un nuevo código de compartición que denominamos *basado en distancia* (*Distance Aware Sharing Code*), y que evaluamos en el contexto de un esquema dinámico de asignación de líneas de memoria a nodos origen, también basado en distancia, a través del sistema operativo previamente propuesto [13]. Los resultados obtenidos mediante un simulador de sistema completo (GEMS) demuestran que este código de compartición es el que mejores valores ofrece desde el

[1]Dpto. de Ingeniería y Tecnología de Computadores, Univ. Murcia, e-mail: {aros, meacacio}@um.es

punto de vista Tamaño (en bits) × Tráfico generado (en bytes)^2 o $S \times T^2$. Para ello comparamos nuestra propuesta con distintos códigos de compartición comprimidos que tienen un tamaño similar.

El resto del documento se organiza como sigue. En la Sección II incluimos un breve resumen del mecanismo de asignación de líneas de memoria a nodos origen ya propuesto sobre el que se construye DASC. Posteriormente, en la Sección III presentamos el código de compartición basado en distancia (DASC). La Sección IV discute el entorno de evaluación que hemos empleado en nuestras pruebas. La Sección V muestra los resultados de la evaluación, y finalmente, la Sección VI presenta las principales conclusiones que pueden extraerse de este trabajo.

II. DARR: Asignación Dinámica de Nodos Origen Basada en Distancia

Un problema que aparece en las arquitecturas de gran escala basadas en celdas es la alta latencia de acceso al nivel de caché compartido, en nuestro caso, la caché L2. Dado que esta caché se encuentra físicamente distribuida entre las diferentes celdas del sistema (caché NUCA), su tiempo de acceso dependerá de lo cerca que se encuentre la celda origen (*home*) del núcleo que solicita la línea de memoria.

Cuando se usa una política de mapeo de cachés en las que los bits menos significativos de la dirección de bloque (sin contar el *offset* de bloque) definen el nodo *home* de dicha línea, las líneas se reparten entre los diferentes bancos de caché de un modo cíclico (*round-robin*), tal y como se lleva a cabo tanto en CMPs comerciales [7], [14] como en la literatura reciente [4], [16]. Esta distribución de las líneas de memoria no se preocupa en absoluto de la distancia entre los bancos de caché donde se mapean las mismas y los núcleos que más frecuentemente acceden a ellos.

Otra forma de distribuir las líneas de memoria en las cachés, propuesta por Cho y otros [2], consiste en definir el nodo *home* usando bits más significativos de la dirección de bloque, en concreto, un subconjunto de los bits que definen el número de página física. De este modo, cuando el sistema operativo realiza la traducción de página virtual a física, se puede elegir un número de página física tal que el mapeo de dicha página corresponda al mismo nodo que solicitó por primera vez la línea de memoria. Esta política es conocida como *first-touch*. La mejor granularidad de asignación para esta política es la de tamaño de página, ya que es la más fina de las que posibilita. Por tanto, se deben coger para la elección del nodo *home* los bits menos significativos sin contar con el *offset* de página. El problema de esta política consiste en que si la carga de trabajo de cada procesador no está bien balanceada, habrá unos bancos de caché con muchas más páginas asignadas que otros, con lo que se limitaría la capacidad total de la caché compartida y se incrementaría la tasa de fallos de dicha caché, que normalmente es la última en los niveles de la jerarquía de memoria *on-chip* y causaría accesos costosos. Esto ocurre muy a menudo en los servidores que ejecutan varias instancias de aplicaciones con requerimientos de memoria completamente distintos en un mismo CMP.

La solución a este problema es una política de mapeo de cachés controlada por el sistema operativo y sensible a la distancia y a la tasa de fallos [13]. Esta política, llamada *DARR* (*Distance-Aware Round-Robin*), intenta mapear páginas a los bancos de caché que son locales al procesador que realiza la primera petición para una línea de esa página, pero a la vez intenta balancear la distribución de las páginas en los diferentes bancos de caché. De este modo se reduce la latencia de acceso a la caché sin incrementar excesivamente la tasa de fallos. Además, otra ventaja importante de este mecanismo es que no necesita ningún hardware extra para ser implementado, al ser llevado a cabo por el sistema operativo.

En concreto, ante un fallo de página, el sistema operativo necesita asignar una dirección física a la página accedida. El algoritmo de asignación elige una dirección física de tal modo que mapee al banco de caché local al procesador que está accediendo a la línea de memoria que provocó el fallo de página, tal y como se realizaría en una política *first-touch*. El sistema operativo posee un contador para cada banco de caché que le informa del número de páginas mapeadas a cada una de ellas. Al mapear una nueva página a un banco de caché, su contador se incrementa.

Para garantizar la distribución uniforme de páginas a bancos de caché, se define un umbral que limita la máxima diferencia del número de páginas mapeadas entre dos bancos cualquiera. Cuando un banco alcanza este umbral, ya no puede alojar más páginas y tiene que delegar el mapeo a un banco vecino, es decir, a un banco que está a un salto según la topología de la red de interconexión. Si los bancos vecinos también han alcanzado el valor umbral, la página se intentará mapear a un banco que esté a dos saltos, y así sucesivamente hasta encontrar un banco que pueda alojar la página. Cuando el contador de todos los bancos es mayor que cero, se decrementa en una unidad cada uno de los contadores con el fin de que nunca lleguen todos los bancos al umbral. Este umbral define el comportamiento del algoritmo. Si es muy bajo, su comportamiento será similar a una política *round-robin*. Si por el contrario es muy alto, el algoritmo funcionará de manera similar a una política *first-touch*. En concreto en este trabajo usamos un valor umbral de 128, como el recomendado en [13].

III. DASC: Código de Compartición Basado en Distancia

En esta sección presentamos DASC, un nuevo código de compartición *basado en distancia* (*Distance Aware Sharing Code*) que ha sido desarrollado teniendo en mente la política dinámica de asignación de líneas de memoria a nodos origen propuesta con anterioridad en [13] y descrita en el apartado anterior. Al igual que otros códigos de compartición

ya propuestos, DASC es un código de compartición comprimido. Es decir, en lugar de almacenar la identidad exacta de los compartidores (como por ejemplo se haría con un código de compartición de vector de bits), DASC codifica una representación *por exceso* del conjunto de compartidores. Esto hace que en ocasiones se tenga que enviar un número más grande de mensajes por evento de coherencia del que realmente sería necesario (el código de compartición incluye más compartidores de los que realmente existen), con la consiguiente pérdida de prestaciones (la latencia del fallo de caché se vería incrementada en este caso) y desperdicio de energía que esto conllevaría. Afortunadamente, y a diferencia de otros códigos de compartición comprimidos, DASC ha sido diseñado para ser utilizado conjuntamente con la política dinámica de asignación de líneas de memoria a nodos origen DARR (en general, para una política de asignación dinámica basada en asignar cada línea de memoria a la celda que la referencia por primera vez), lo que permite reducir el número de mensajes de coherencia innecesarios a la vez que se consigue un tamaño de código de compartición muy reducido.

En DASC, el conjunto de compartidores para una determinada línea de memoria es representado desde el nodo origen. En concreto, DASC codifica la distancia (medida como número de enlaces) entre la celda origen y el compartidor más lejano en la topología. Todas las celdas que están dentro de dicho valor de distancia son consideradas como compartidores potenciales. Afortunadamente, el hecho de que con la política dinámica de asignación DARR cada línea de memoria tienda a estar cerca de su celda origen, hace que el número de compartidores *falsos* sea pequeño. A modo de ejemplo, la Figura 1 muestra la distancia (número de enlaces) entre la celda origen 0 y el resto de celdas en una arquitectura multinúcleo con 16 núcleos de procesamiento y que emplea una malla bidimensional como red de interconexión. En la Tabla I se detallan las celdas que serían incluidas en cada caso para los distintos valores que podría tomar un código de compartición DASC de 2 bits y otro de 3 bits. Por ejemplo, si suponemos que en un instante determinado las cachés privadas de las celdas 1, 4 y 5 mantienen una copia de solo lectura de una determinada línea de memoria cuya celda origen es 0, el valor que se codificaría con DASC sería 2. Obviamente este valor incluye a dichas celdas, pero también a las celdas 0, 2 y 8 que no tienen copia de dicha línea de memoria y por lo tanto podrían recibir mensajes de coherencia innecesarios (por ejemplo, en el caso de una escritura a dicha línea).

Además de reducir el número de mensajes de coherencia innecesarios en comparación con otros códigos de compartición con tamaño similar, DASC resulta también mucho más sencillo de implementar. En concreto, para calcular la distancia desde la celda peticionaria hasta la celda origen, proponemos incluir en cada mensaje de petición (mensaje generado en cada fallo de caché) un campo de bits (de

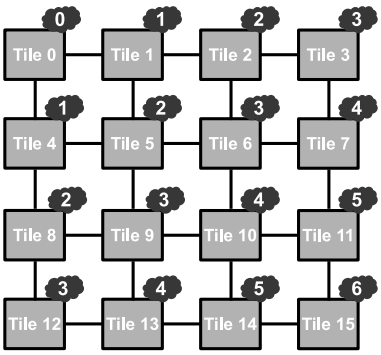

Fig. 1. Distancia desde la celda (*tile*) 0 al resto en una arquitectura con 16 núcleos.

la misma longitud que el código DASC implementado por el directorio) que es gestionado a modo de contador saturado. El valor más grande de dicho contador lo reservamos para representar la situación en la que todas las celdas deben ser incluidas. Cada vez que ocurre un fallo de caché, y por tanto, se genera un nuevo mensaje de petición, el valor de dicho campo se pone a 0. Cada vez que un mensaje de petición ha de atravesar uno de los canales externos de un *router*, el valor de dicho contador es incrementado. Finalmente, cuando el mensaje de petición llega a la celda origen, el valor almacenado en el directorio ha de ser cambiado por el que trae el mensaje de petición si este último es mayor. Otros códigos de compartición de tamaño similar a DASC propuestos previamente (por ejemplo BT y BT-SN, que serán evaluados también en este trabajo [1]) requieren un hardware relativamente complejo en cada celda origen para calcular el valor del código de compartición conforme nuevos compartidores van pidiendo el bloque de datos. Obviamente, esta implementación de DASC asume una estrategia de encaminamiento de ruta mínima (en nuestro caso asumimos un esquema de encaminamiento determinista X-Y).

De igual manera, otra importante ventaja que se deriva de la utilización de DASC es que el envío de los mensajes de coherencia puede ser realizado de manera muy eficiente. Concretamente, en cada evento de coherencia (envío de invalidaciones u órdenes de transferencia caché-a-caché), el controlador de directorio de la celda origen crearía un solo mensaje de coherencia, el cual se propagaría a través de todos los puertos de cada *router* hasta que el número de enlaces por recorrer fuese cero. Para ello se emplearía un campo en los mensajes de coherencia idéntico al añadido para los mensajes de petición. Dicho campo sería inicializado por el controlador de directorio de la celda origen con el valor DASC almacenado en el directorio. Obviamente, si el directorio inicializa dicho campo con su valor máximo, el mensaje se iría propagando hasta asegurar que todas las celdas son cubiertas (se realizaría una difusión completa del mensaje). Por otro lado, si el valor de dicho campo es distinto de cero, se reduce en una unidad y el mensaje se copia a través de todos los puertos del *router*. Por último cuando el valor de dicho campo es cero, el mensaje se envía únicamente

TABLA I
EJEMPLOS DE CODIFICACIÓN CON DASC EN UNA ARQUITECTURA CON 16 NÚCLEOS QUE EMPLEA UNA MALLA BIDIMENSIONAL, ASUMIENDO QUE 0 ES LA CELDA ORIGEN

2 bits		**3 bits**	
Valor	Celdas incluidas	Valor	Celdas incluidas
0	{0}	0	{0}
1	{0,1,4}	1	{0,1,4}
2	{0,1,2,4,5,8}	2	{0,1,2,4,5,8}
3	Todas las celdas	3	{0,1,2,3,4,5,6,8,9,12}
		4	{0,1,2,3,4,5,6,7,8,9,10,12,13}
		5	{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14}
		6–7	Todas las celdas

al controlador de L1 local. En definitiva, los mensajes de coherencia podrían enviarse con una implementación *multicast* sencilla realizada en hardware. Otros códigos de compartición comprimidos, por el contrario, necesitarían una lógica hardware en cada celda origen para calcular la identidad de los compartidores a partir del valor del código de compartición (por ejemplo, en BT y BT-SN). Esta lógica, que no es necesaria con DASC, incrementaría el tiempo de procesamiento de las peticiones por parte del directorio (y por lo tanto el tiempo que otras peticiones han de esperar a ser atendidas) e incrementaría el consumo energético. Por otro lado, mientras que en DASC se generaría un único mensaje de coherencia que iría propagándose conforme fuese necesario, BT y BT-SN requerirían la creación de tantos mensajes de coherencia como destinos fuese necesario alcanzar, con la consiguiente penalización en tiempo (creación de varios mensajes), consumo de recursos de red y, obviamente, mayor carga energética.

Por último, es importante notar que DASC podría ser implementado sobre cualquier topología de red y no requiere modificaciones en el hardware para que pueda ser adaptado a otras topologías. Otros códigos de compartición, por el contrario, podrían estar condicionados por la topología de red para la cual se diseñaron.

IV. Entorno de Evaluación

La evaluación del código de compartición propuesto en la sección anterior se ha realizado mediante el simulador funcional Virtutech Simics [8] extendido con Multifacet GEMS 2.1 [10]. A través de GEMS se modela la jerarquía de memoria y se ofrece a Simics la latencia de acceso a memoria de cada petición del procesador. La red de interconexión se ha modelado con el simulador SiCoSys [11]. La arquitectura simulada corresponde a un *tiled* CMP con 32 núcleos. Los principales parámetros de evaluación se muestran en la Tabla II.

Para la evaluación hemos usado aplicaciones de la suite SPLASH-2 [15] y la aplicación *Unstructured*: *Barnes* (16K partículas), *Cholesky* (tk15.O), *Ocean* (océano de 258×258), *Radix* (1M claves, radio 1024), *Raytrace* (teapot –optimizada tras eliminar cerrojos innecesarios–), *Water-Nsq* (512 moléculas) y *Unstructured* (Mesh.2K). Hemos evaluado tanto aplicaciones ejecutándose solas en el chip como cargas multiprogramadas donde varias aplicaciones comparten

TABLA II
PARÁMETROS DEL SISTEMA

Parámetros de la memoria	
Tamaño de bloque	64 bytes
Caché L1 de datos e instr.	32KB, 4 vías
Latencia de acceso a L1	2 ciclos
Caché L2 compartida	256KB/celda, 8 vías
Latencia de acceso a L2	10 ciclos
Información de directorio	Incluida en L2
Tiempo de acceso a memoria	300 ciclos
Tamaño de página	4KB
Parámetros de la red	
Topología	Malla 2-D (4×8)
Técnica de switching	Wormhole
Técnica de enrutamiento	Determinista X-Y
Tamaño de mensajes	4 flits (datos), 1 flit (control)
Tiempo de switch y enlace	2 y 2 ciclos
Ancho de banda	1 flit por ciclo

los núcleos del chip: en *Mix4*, *Ocean*, *Raytrace*, *Water-Nsq* y *Unstructured* son ejecutadas con 8 hilos cada una; en *Mix8* dos instancias de cada una de ellas son ejecutadas con 4 hilos; *Radix4* ejecuta cuatro instancias de *Radix* con ocho hilos cada una; y *Ocean8* ejecuta ocho instancias de *Ocean* con cuatro hilos cada una. Todos los resultados mostrados en este trabajo corresponden a la parte paralela de las aplicaciones evaluadas.

Comparamos DASC con dos códigos de compartición comprimidos, de tamaño similar, previamente propuestos en la literatura: BT y BT-SN [1].

Para evitar introducir en la comparativa cualquier aspecto que dependa de una implementación concreta y que pudiese desvirtuar los resultados en favor de DASC, no hemos considerado las ventajas que se derivan de la sencillez de implementación del mismo. De esta forma, nuestras simulaciones no consideran la sobrecarga que introduce la lógica necesaria para calcular los valores de BT y BT-SN, y determinar los destinos de los mensajes de coherencia en cada evento de coherencia. Dicha lógica habría de ser implementada en las celdas origen e incrementaría el tiempo necesario para resolver los fallos de caché (así como la energía consumida). Asimismo, evaluamos ambas implementaciones asumiendo que la red de interconexión proporciona soporte *multicast*, con lo que tampoco sacamos partido del envío más eficiente de los mensajes de coherencia que permite DASC.

V. Resultados

El código de compartición propuesto en este trabajo codifica la información de los compartidores basándose en la distancia de ellos a la celda origen. Por tanto, cuanto más cerca estén los compar-

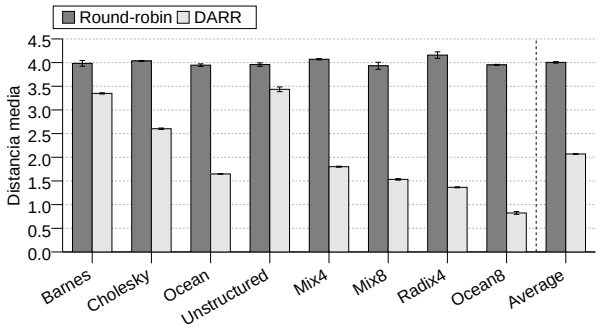

Fig. 2. Distancia de los compartidores a la celda origen.

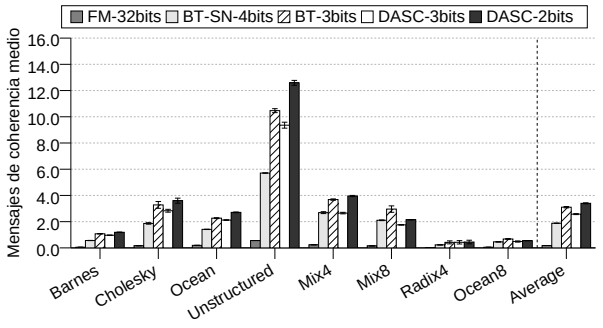

Fig. 3. Número de mensajes de coherencia por evento de coherencia.

tidores de la misma mejor se comportará el código de compartición propuesto, es decir, menos mensajes de coherencia innecesarios serán introducidos. La Figura 2 muestra la distancia media de los compartidores con la política de mapeo DARR sobre la que se construye DASC. Podemos observar que aunque la distancia media está en torno a los 2 saltos, para las cargas multiprogramadas esta distancia es mucho más pequeña que para las aplicaciones que se ejecutan usando la totalidad de los núcleos de que dispone la arquitectura evaluada (32). Por contra, esta distancia es menor que en una organización donde se usa una política de asignación estática como es el caso de la política *round-robin*, donde la distancia media está alrededor de 4 enlaces de red. Por tanto, como puede apreciarse, la organización DARR será la ideal para el código de evaluación propuesto.

Cuanto mayor sea esta distancia media mayor será en número de mensajes de coherencia (invalidaciones u órdenes de transferencia de caché-a-caché) que sería necesario enviar por evento de coherencia cuando se emplea el código de compartición DASC. Sin embargo, esto no siempre se traslada de forma directa sobre los resultados en términos de rendimiento, ya que es posible que la distancia media sea baja pero haya un compartidor que esté muy alejado de la celda origen y que, por tanto, cause el envío de un gran número de mensajes de coherencia innecesarios. La Figura 3 muestra el número medio de mensajes de coherencia (o *probes*) por evento de coherencia.

Observamos que los códigos de compartición comprimidos aumentan el número de mensajes de coherencia con respecto a un código exacto de vector de bits (*bit-vector* o *full-map* –FM–), que en nuestro sistema tendría una longitud de 32 bits. Sin embargo, podemos observar cómo el código de compartición DASC de 3 bits genera menos mensajes de coherencia que BT, el cual emplea el mismo número de bits. Además, el código DASC con solo 2 bits obtiene resultados similares a BT de 3 bits.

La reducción en el número de mensajes de coherencia por evento de invalidación se traduce en una reducción en el tráfico de control, como se muestra en la Figura 4. Podemos ver que de media, el código DASC de 3 bits reduce el tráfico con respecto al código BT del mismo tamaño, y que el código DASC de 2 bits obtiene resultados similares al BT. La sobre-

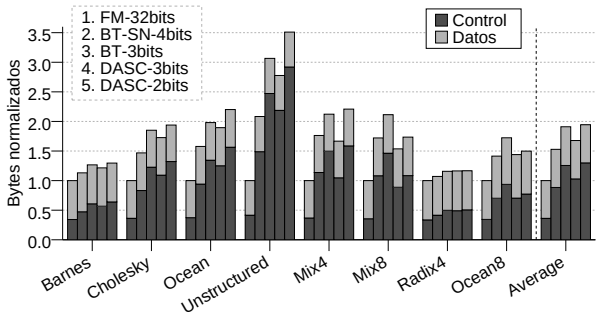

Fig. 4. Tráfico de red normalizado (en *flits*).

carga en tráfico en la red con respecto a un código de vector de bits (FM) es del 67% cuando se emplea el DASC de 3 bits y del 94% cuando se utiliza la versión de 2 bits, pero a cambio de obtener una escalabilidad en el área requerida por el código de compartición.

En cambio, la degradación en tiempo de ejecución al usar los códigos de compartición comprimidos no es tan significativa, tal y como puede apreciarse en la Figura 5. En particular los códigos basados en árbol binario (BT y BT-SN) y los códigos DASC obtienen prestaciones similares, siendo estos últimos más escalables en área. Con respecto a un código de compartición de vector de bits (FM) la degradación de rendimiento introducida por DASC es de solo 4.3% para la versión de 3 bits y de 5.6% para la de 2 bits.

Para tener una valoración del incremento de tráfico al reducir el tamaño del código de compartición, utilizaremos una métrica que hemos llamado ST^2 (Size $\times$ Traffic^2), es decir, el número de bytes transmitidos por la red al cuadrado multiplicado por el tamaño del código de compartición en bits. El resultado de aplicar esta métrica para cada código de compartición se puede ver en la Figura 6. Podemos observar que siguiendo esta métrica el código de compartición BT es menos eficiente que el que usa nodos simétricos (BT-SN). Por otro lado, el código de compartición basado en la distancia de 2 bits es el más eficiente de todos, obteniendo unos resultados muy parecidos a los que ofrece su la versión de 3 bits. Cabe destacar que es en el caso de las cargas multiprogramadas, como las que cabe esperar en chips con tantos núcleos, en donde mejor se comporta el código propuesto.

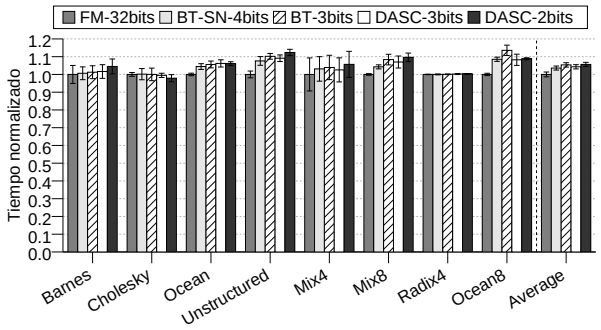

Fig. 5. Tiempo de ejecución normalizado.

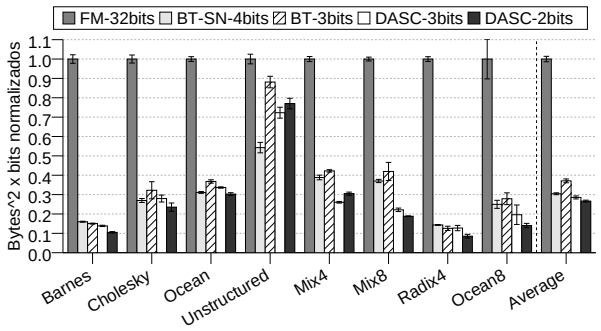

Fig. 6. Métrica ST^2: bytes transmitidos al cuadrado por tamaño en bits del código de compartición.

VI. Conclusiones

La organización del directorio en las futuras arquitecturas multinúcleo con un número grande de núcleos de procesamiento es un aspecto clave, ya que determina en gran medida el rendimiento y afecta al consumo energético del diseño. En este trabajo hemos presentado DASC, un código de compartición basado en distancia que se construye asumiendo una política dinámica de asignación de líneas de memoria a nodos origen también basada en distancia llamada DARR [13]. DASC se caracteriza por tener un tamaño muy pequeño, que no se incrementa linealmente con el número de núcleos del sistema, lo que da lugar a una estructura de directorio mucho más escalable que otras propuestas. Además, tanto el cálculo del valor DASC para cada línea de memoria como el envío de mensajes de coherencia a los compartidores a partir del valor DASC, puede realizarse de manera muy sencilla y sin necesidad de añadir lógica extra en los controladores de directorio (a diferencia de otros códigos de compartición comprimidos ya propuestos). Concretamente, el valor DASC para cada línea de memoria se va calculando conforme el mensaje de petición avanza hacia la celda origen. Así mismo, el envío de los mensajes de coherencia en DASC se puede realizar de forma más eficiente que con otros códigos comprimidos, ya que se llevaría a cabo como si de un esquema *multicast* implementado en hardware se tratase.

Hemos evaluado DASC y lo hemos comparado con otros códigos de compartición comprimidos con tamaño similar, confirmando que DASC es el que mejores resultados obtiene considerando como métrica Tamaño (en bits) × Tráfico generado^2 (en bytes).

Agradecimientos

This work has been supported by the Spanish MINECO, as well as European Commission FEDER funds, under grant “TIN2012-38341-C04-03”

Referencias

[1] M. E. Acacio, J. González, J. M. García, and J. 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.

[2] S. Cho and L. Jin, “Managing distributed, shared L2 caches through OS-level page allocation,” in *39th IEEE/ACM Int’l Symp. on Microarchitecture (MICRO)*, Dec. 2006, pp. 455–465.

[3] M. Hammoud, S. Cho, and R. Melhem, “Acm: An efficient approach for managing shared caches in chip multiprocessors,” in *4th Int’l Conference on High Performance and Embedded Architectures and Compilers (HiPEAC)*, Jan. 2009, pp. 355–372.

[4] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, and S. W. Keckler, “A NUCA substrate for flexible CMP cache sharing,” in *19th Int’l Conf. on Supercomputing (ICS)*, Jun. 2005, pp. 31–40.

[5] “Intel Xeon Phi Coprocessor,” http://software.intel.com/en-us/mic-developer, 2013. [Online]. Available: http://software.intel.com/en-us/mic-developer

[6] G. Kurian, J. E. Miller, J. Psota, J. Eastep, J. Liu, J. Michel, L. C. Kimerling, and A. Agarwal, “Atac: A 1000-core cache-coherent processor with on-chip optical network,” in *19th Int’l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Sep. 2010, pp. 477–488.

[7] H. Q. Le, W. J. Starke, J. S. Fields, F. P. O’Connell, D. Q. Nguyen, B. J. Ronchetti, W. M. Sauer, E. M. Schwarz, and M. T. Vaden, “IBM POWER6 microarchitecture,” *IBM Journal of Research and Development*, vol. 51, no. 6, pp. 639–662, Nov. 2007.

[8] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, “Simics: A full system simulation platform,” *IEEE Computer*, vol. 35, no. 2, pp. 50–58, Feb. 2002.

[9] M. M. K. Martin, M. D. Hill, and D. Sorin, “Why on-chip cache coherence is here to stay,” *Communications of the ACM*, vol. 55, no. 7, pp. 78–89, Jul. 2012.

[10] M. M. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, “Multifacet’s general execution-driven multiprocessor simulator (GEMS) toolset,” *Computer Architecture News*, vol. 33, no. 4, pp. 92–99, Sep. 2005.

[11] V. Puente, J. A. Gregorio, and R. Beivide, “SICOSYS: An integrated framework for studying interconnection network in multiprocessor systems,” in *10th Euromicro Workshop on Parallel, Distributed and Network-based Processing*, Jan. 2002, pp. 15–22.

[12] C. Ramey, “TILE-Gx100 manycore processor: Acceleration interfaces and architecture,” in *Hot Chips 23*, Aug. 2011.

[13] A. Ros, M. Cintra, M. E. Acacio, and J. M. García, “Distance-aware round-robin mapping for large NUCA caches,” in *16th Int’l Conf. on High Performance Computing (HiPC)*, Dec. 2009, pp. 79–88.

[14] M. Shah, J. Barreh, J. Brooks, R. Golla, G. Grohoski, N. Gura, R. Hetherington, P. Jordan, M. Luttrell, C. Olson, B. Saha, D. Sheahan, L. Spracklen, and A. Wynn, “UltraSPARC T2: A highly-threaded, power-efficient, SPARC SoC,” in *IEEE Asian Solid-State Circuits Conference*, Nov. 2007, pp. 22–25.

[15] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, “The SPLASH-2 programs: Characterization and methodological considerations,” in *22nd Int’l Symp. on Computer Architecture (ISCA)*, Jun. 1995, pp. 24–36.

[16] M. Zhang and K. Asanović, “Victim replication: Maximizing capacity while hiding wire delay in tiled chip multiprocessors,” in *32nd Int’l Symp. on Computer Architecture (ISCA)*, Jun. 2005, pp. 336–345.