UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Caché de directorio multinivel escalable para CMP NUCAs

Proyecto Final de Carrera de Ingeniería en Informática

Curso 2011/2012

Escuela Técnica Superior de Ingeniería Informática

Joan Josep Valls Mompó

Directores:

Julio Sahuquillo Borrás

María Engracia Gómez Requena

Julio de 2012

Agradecimientos

A mi familia, por todo el apoyo recibido y soportarme durante todos estos años.

A mis compañeros y amigos, por los buenos momentos que hemos compartido.

A Julio y María Engracia, por haber confiado en mí y haberme dado esta gran oportunidad.

A Alberto, por toda esa ayuda recibida desde el principio y, en general, por estar siempre al otro lado del correo y ayudarme a solucionar todos los problemas que han ido surgiendo.

A todos vosotros, gracias.

Índice general

1. Introducción **7**

1.1. Descripción del problema 7

1.2. Objetivos 8

1.3. Motivación 8

1.4. Estructura del trabajo 12

2. Aspectos fundamentales **13**

2.1. Arquitectura de cache no uniforme (NUCA) 13

2.2. Protocolos de coherencia 14

2.2.1. Protocolo MOESI 14

2.2.2. Protocolos de actualización e invalidación 16

2.2.3. Protocolos snoopy y basados en directorio 18

2.3. Tecnologías de memoria 19

3. Trabajo relacionado **22**

4. Estructura de directorio propuesta **26**

4.1. Arquitectura base 26

4.2. Esquema directorio PS 27

5. Entorno de simulación **31**

5.1. Herramientas de simulación 31

5.1.1. Simics-GEMS 31

5.1.2. CACTI 32

5.1.3. Sistema simulado 32

5.2. Métricas y metodología 34

5.3. Benchmarks 35

5.3.1. Barnes 36

5.3.2. FFT 36

5.3.3. Ocean . . . 37
5.3.4. Radiosity . . . 37
5.3.5. Radix . . . 37
5.3.6. Raytrace . . . 38
5.3.7. Volrend . . . 38
5.3.8. Water-Nsq . . . 38
5.3.9. Blackscholes . . . 39
5.3.10. Swaptions . . . 39

6. Evaluación experimental **40**
6.1. Impacto en el tiempo de ejecución . . . 40
6.2. Análisis de energía y área . . . 42
6.3. Análisis de escalabilidad . . . 44
6.4. Reducción del número de vias . . . 45

7. Conclusiones y trabajo futuro **47**
7.1. Conclusiones . . . 47
7.2. Trabajo futuro . . . 48
7.3. Publicaciones relacionadas con el proyecto . . . 48

Índice de figuras

1.1. Número de accesos a bloques privados y compartido por kiloinstrucciones en una caché de directorio convencional. 10
1.2. Expulsiones de bloques privados y compartidos por kiloinstrucciones en una caché de directorio convencional y su efecto en las prestaciones. . . 11

2.1. Evolución en los tiempos de acceso. 13
2.2. Tiempo de acceso medio en función de la configuración de la memoria. 15
2.3. Diagrama de transición de estados para un protocolo MOESI. 16
2.4. Ejemplo de funcionamiento de un protocolo de actualización. 17
2.5. Ejemplo de funcionamiento de un protocolo de invalidación. 17

4.1. Organización de un tile y de un CMP 4×4. 26
4.2. Organización del Directorio Privado-Compartido. 28
4.3. Diagrama de flujo del controlador de memoria. 29
4.4. Acceso paralelo a la caché Compartida y a la NUCA. La caché Privada solo se accede si hay fallo en la Compartida. 30

6.1. Prestaciones normalizadas respecto a una caché de directorio convencional. 41
6.2. Energía consumida por el directorio normalizado respecto a una caché de directorio convencional. 43
6.3. Análisis de la escalabilidad en función del área. 44
6.4. Prestaciones normalizadas respecto a una caché de directorio convencional. 45

Índice de tablas

2.1. Características de las tecnologías eDRAM y SRAM. 20

5.1. Parámetros del sistema 33
5.2. Latencias del directorio 33

6.1. Área (in $mm^2 * 1000$) de las configuraciones PS para 16 núcleos frente un directorio single caché 1×. 43
6.2. Consumo de energía estática y dinámica de las configuraciones PS para 16 núcleos frente un directorio single caché 1×. 44

Capítulo 1

Introducción

1.1. Descripción del problema

Conforme avanza la tecnología, la escala de integración permite reducir cada vez más el tamaño de los transistores y, gracias a eso, cada vez encontramos una mayor cantidad de transistores en los chips. Este nuevo número de transistores se está dedicando principalmente en aumentar el número de procesadores en un único chip, ya que esto resulta más sencillo y aporta mejores beneficios que intentar mejorar el rendimiento de un único procesador. Es así como surgen los *chip multiprocessors* (CMP). El número de núcleos en estos CMP ha ido creciendo continuamente y se espera que se llegue a varios centenares en los próximos años [BEA08]. La mayoría de los CMPs permiten el modelo de programación de memoria compartida y, por tanto, implementan un protocolo de coherencia para mantener la coherencia de los datos en las cachés privadas de los procesadores.

Los protocolos de coherencia deben diseñarse para poder escalar al aumentar el número de núcleos. Hay sistemas con hasta 80 (Polaris) y 100 núcleos (Tilera) y algunos trabajos académicos ya plantean CMPs con 1000 núcleos [KMP+10, KJLP10]. La solución más utilizada en sistemas con pocos núcleos se basa en procotolos de snooping. En este caso, todos los núcleos deben ver todos los mensajes de coherencia en el mismo orden para garantizar la coherencia de los datos. No obstante, estos protocolos no escalan y hasta ahora la alternativa más escalable son los protocolos basados en directorio. Estos protocolos usan una estructura denominada directorio de coherencia para mantener información sobre las cachés de los procesadores (por ejemplo la L1) que tienen una copia del bloque. Al directorio se accede para realizar las acciones de mantenimiento de coherencia, tales como enviar peticiones de invalidación ante ope-

raciones de escritura, o solicitar una copia del bloque al propietario del mismo (por ejemplo, al último procesador que lo haya escrito).

Una forma más eficiente de organizar el directorio consiste en almacenar la información en una caché. En este diseño, cuando una línea es reemplazada del directorio, las copias del bloque involucrado en las cachés del procesador deben invalidadarse, incluso aunque estén siendo utilizadas por el procesador, aumentando así los llamados fallos de *coverage* [CRG+11]. Este tipo de fallos no se encuentran en un esquema de directorio completo ya que en estos casos se dispone de una entrada de directorio por cada línea de memoria y, como consecuencia, no es necesario reemplazar ningún bloque por falta de espacio. No obstante, una caché de directorio resulta más escalable dado que su número de entradas es más reducido. Con el aumento del número de núcleos, el número de fallos de *coverage* se incrementa considerablemente y nuevos diseños de directorio más eficientes son necesarios.

1.2. Objetivos

El propósito de este proyecto es diseñar y evaluar por medio de simulación una nueva estructura de directorio más escalable que los esquemas de caché de directorio tradicionalmente utilizados, así como otros publicados en el estado del arte. El diseño propuesto se basa en la observación de que un elevado número de bloques tan solo son accedidos por la caché privada de un único procesador. Estos bloques muestran un comportamiento muy diferente al de los bloques compartidos por varios núcleos. Por ello en este trabajo se propone usar 2 estructuras distintas con dos tipos actuales de tecnología para el diseño de la jerarquía de memoria: eDRAM y SRAM. La idea fundamental es separar una caché de directorio única en dos estructuras exclusivas que podremos tratar de forma diferenciada y en la que se mantendrá la coherencia de los bloques privados y compartidos, respectivamente. Dada la naturaleza de los bloques privados, dispondremos de una estructura más compacta ya que para mantener la coherencia de este tipo de bloques no es necesario mantener un vector de presencia.

1.3. Motivación

El hecho de que los bloques privados y compartidos muestren un comportamiento diferente desde el punto de vista del directorio es la base de este trabajo. Esto se puede detallar en cuatro puntos fundamentales:

1. Las entradas del directorio que mantienen la información de los bloques privados no requieren del vector de compartidores.

2. La mayoría de los bloques de datos en cargas paralelas son privados, como se ha comprobado en trabajos recientes [CRG+11].

3. La mayoría de los accesos al directorio son a causa de bloques compartidos.

4. Los bloques privados tan solo se acceden una vez.

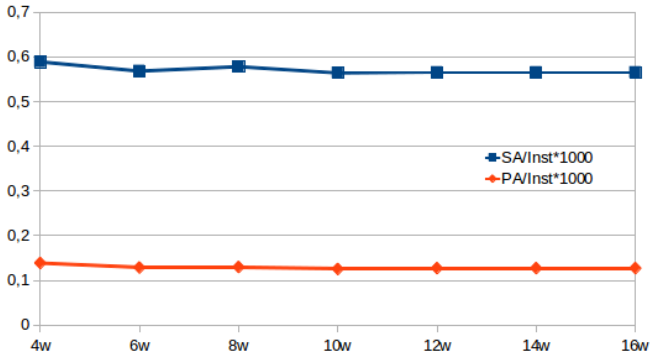
Todo esto apunta al diseño de un directorio en el que se empleen dos estructuras caché independientes, cada una de los cuales está diseñada para mantener la información de los bloques privados y compartidos respectivamente.

Con respecto a los dos primeros puntos, es razonable pensar que la caché privada debería diseñarse más alta (es decir, con más entradas por el elevado número de bloques privados) y con entradas más estrechas (ya que no se necesita el vector de compartidores) que los de la caché compartida. De esta forma podemos obtener importantes ahorros en área , especialmente en sistemas con un elevado número de núcleos. A mayor tamaño de la caché privada en comparación con la caché compartida, mayores ahorros obtendremos debido a la ausencia del campo de compartidores.

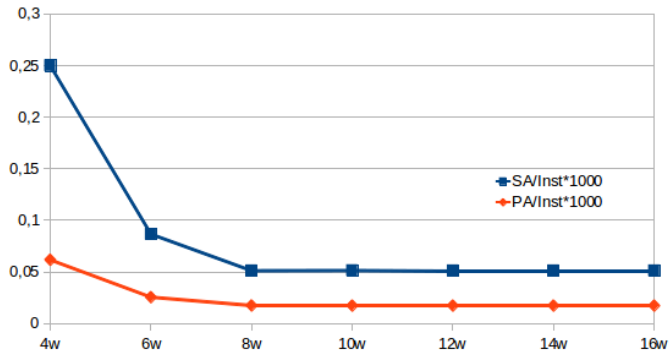
La figura 1.1 muestra el número de accesos a una caché de directorio convencional por kilo instrucciones para un benchmark SPLASH-2 (Barnes) y un PARSEC (Blackscholes), distinguiendo si dichos accesos son a bloques privados o compartidos. En el eje x se va aumentando el número de vías de la caché de directorio mientras que el número de conjuntos se mantiene constante.

Como se puede ver, el número de accesos a bloques compartidos es 5× veces mayor que el número de accesos a bloques privados, tal y como se explicaba en los últimos dos puntos. Las entradas de los bloques privados tan solo son consultadas en caso de que el bloque hubiera sido expulsado de la caché del procesador, por tanto, es normal que los bloques privados apenas sean accedidos a pesar de su gran número.

Los resultados para Blackscholes muestran diferencias menores para un número más elevado de vías, porque la capacidad del directorio aumenta (como ya se ha dicho, el número de conjuntos se mantiene constante), haciendo así que se reduzcan las expul-



(a) Barnes



(b) Blackscholes

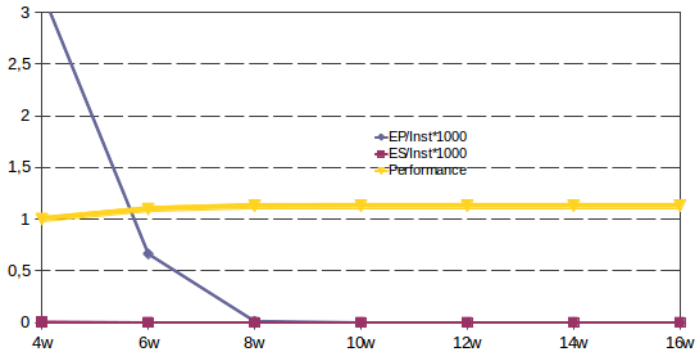
Figura 1.1: Número de accesos a bloques privados y compartido por kiloinstrucciones en una caché de directorio convencional.

siones útiles del directorio.

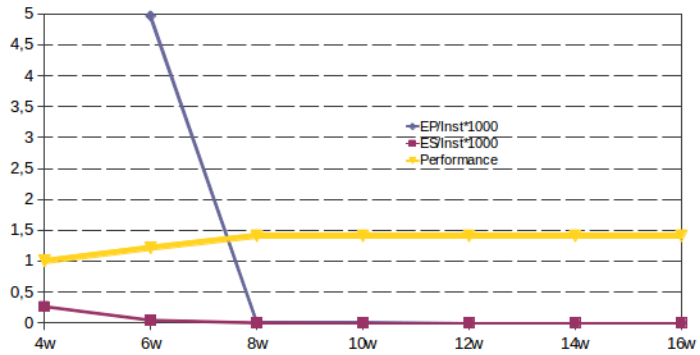
Estos resultados sugieren que los bloques compartidos deberían tener un tiempo de acceso reducido, mientras que la latencia no resulta especialmente crítica para los privados.

Una vez que se ha definido la caché privada más alta y estrecha, se debe investigar el grado de asociatividad adecuado para cada caché. Con este fin, se ha medido en una caché de directorio unificada común las líneas expulsadas que han provocado fallos de *coverage*, clasificándolos acorde con el tipo de bloque del que se estaba manteniendo la información, y su efecto en las prestaciones.

La figura 1.2 muestra este comportamiento para un benchmark SPLASH-2 (Barnes) y un PARSEC (Blackscholes). Los resultados se muestran por kilo instrucciones



(a) Barnes



(b) Blackscholes

Figura 1.2: Expulsiones de bloques privados y compartidos por kiloinstrucciones en una caché de directorio convencional y su efecto en las prestaciones.

lanzadas. Al igual que antes, en el eje x se va aumentando el número de vías de la caché de directorio mientras que el número de conjuntos se mantiene constante.

Se puede observar que el número de bloques compartidos que provocan fallos de coverage apenas varía con el aumento de las vías, mientras que el número de bloques privados que causan este tipo de fallos decrece exponencialmente. El número de bloques privados expulsados es realmente elevado para un grado de asociatividad bajo, lo que se traduce en una degradación significante de las prestaciones. Asumiendo una política de reemplazo LRU típica y sabiendo que un bloque privado no volverá a ser referenciado en el directorio, su tiempo de vida depende exclusivamente del número de vías que tenga disponible. Si es demasiado bajo, es altamente probable que el bloque deba ser expulsado de la caché de procesador que lo tuviera (para poder mantener la coherencia del sistema) aunque aún estuviera siendo usado, aumentando así los fallos de *coverage*.

Por el otro lado, con un número de vías mayor le estamos dando más oportunidades al bloque antes de que tenga que ser expulsado. Se puede apreciar que 8 vías son suficientes para estabilizar el número de expulsiones útiles de bloques privados. Por ese motivo, nuestra caché privada podría diseñarse con ese número de vías, mientras que nuestra caché compartida dispondría de un número menor de ellas (2 o 4, por ejemplo), ya que los bloques privados no se ven beneficiados de una complejidad tan alta.

1.4. Estructura del trabajo

El resto del presente trabajo se organiza de la siguiente manera. En el capítulo 2 se explicarán los temas fundamentales en los que se centra el trabajo: NUCAs, protocolos de coherencia, protocolos basados en directorio y tecnologías de memoria. En el capítulo 3 se presentarán algunos trabajos recientes relacionados. En el capítulo 4 se describirá detalladamente la estructura base desde la que se parte y la propuesta. En el capítulo 5 se explicará el entorno de simulación utilizado para el trabajo y los benchmarks utilizados para las simulaciones. En el capítulo 6 se mostrarán y analizarán los resultados obtenidos en los experimentos lanzados. Por último, en el capítulo 7, se resumirán las conclusiones obtenidas y se describirá el trabajo futuro.

Capítulo 2

Aspectos fundamentales

2.1. Arquitectura de cache no uniforme (NUCA)

El aumento en la escala de integración ha conducido a un aumento en la capacidad de las memorias caché y a una jerarquía de memoría de mayor profundidad (L1, L2, L3, etc.), que en general ocupan un alto porcentaje del área del chip. La problemática surge en que un mayor tamaño de la memoria conlleva una mayor latencia de acceso. En el diseño convencional se diseña teniendo en cuenta el peor caso posible y, en este caso, asume que la latencia viene definida por el tiempo requerido para acceder a la posición de memoria más alejada. Cuando el tamaño no era demasiado grande esto no suponía un problema grave, pero actualmente supone una penalización considerable en los accesos de memoria. En la figura 2.1 se presenta un ejemplo.

Una solución para este problema es el uso de cachés de tipo NUCA (Non Uniform Cache Architecture), en los que una memoria de gran tamaño se divide en varios bancos de tamaño más reducido, cada uno de ellos con una latencia de acceso diferente en función de su distancia. En otras palabras, la organización de una NUCA va en función

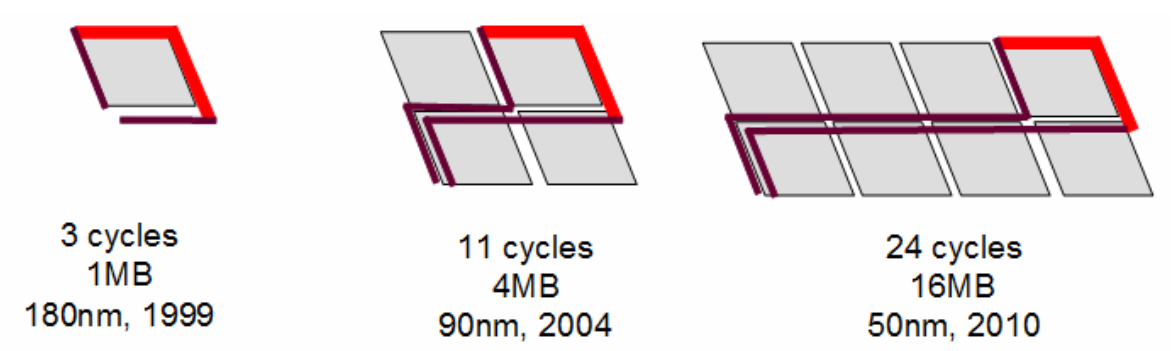


Figura 2.1: Evolución en los tiempos de acceso.

de su número de columnas, su número de bancos y la capacidad individual de cada uno de ellos. Obtenemos así, para m columnas y n bancos de capacidad c, una capacidad total $C = m \times n \times c$.

En la figura 2.2 podemos observar distintas configuraciones y sus respectivos tiempos medios de acceso. En primer lugar (parte superior izquierda) se presenta una UCA tradicional junto a una UCA multinivel inclusiva. Con ese nivel de jerarquía de memoria adicional se consigue reducir las altas latencias de la UCA. Después se presentan distintas variantes para el diseño de una NUCA. En la S-NUCA-1 (Static-NUCA-1) se utilizan los bits de menor paso para decidir en que banco se encuentra o se debe almacenar el bloque, cada uno de ellos con una latencia de acceso diferente. Todos los bancos están conectados mediante buses de datos y de direcciones, lo que impone un *wire overhead* de un 20.9 %, lo cual conlleva un problema de prestaciones considerable. En la S-NUCA-2 se disminuye dicho overhead hasta un 5.9 % gracias al uso de una red de inteconexión 2D. Por último tenemos la D-NUCA (Dynamic-NUCA) que parte de un diseño similar a la S-NUCA-2. En esta configuración un bloque no tiene un banco fijo asignado, sino que en función de su utilización se encontrará en los bancos más cercanos o los más alejados. Es decir, se permite la migración de los datos a lo largo de los bancos de la caché dinámicamente. Se consigue con esto mejorar la latencia de acceso media.

2.2. Protocolos de coherencia

Para poder mantener la coherencia entre todos los procesadores de un CMP se requieren protocolos de coherencia de caché. En esta sección se hará un breve repaso a las opciones de diseño para estos protocolos.

2.2.1. Protocolo MOESI

Existen muchas alternativas para diseñar protocolos de coherencia dependiendo de los estados de los bloques almacenados en las cachés privadas (típicamente L1). Estas alternativas suelen ser nombradas en función de los estados que utilizan: MOESI, MOSI, MESI, MSI, etc. Cada estado representa unos permisos de lectura y escritura distintos para el bloque almacenado en la caché privada. Para este proyecto se ha tenido en consideración el protocolo MOESI, que dispone de un mayor número de estados (el resto de protocolos utilizan un subconjunto de estos), y cuyos estados son los siguientes:

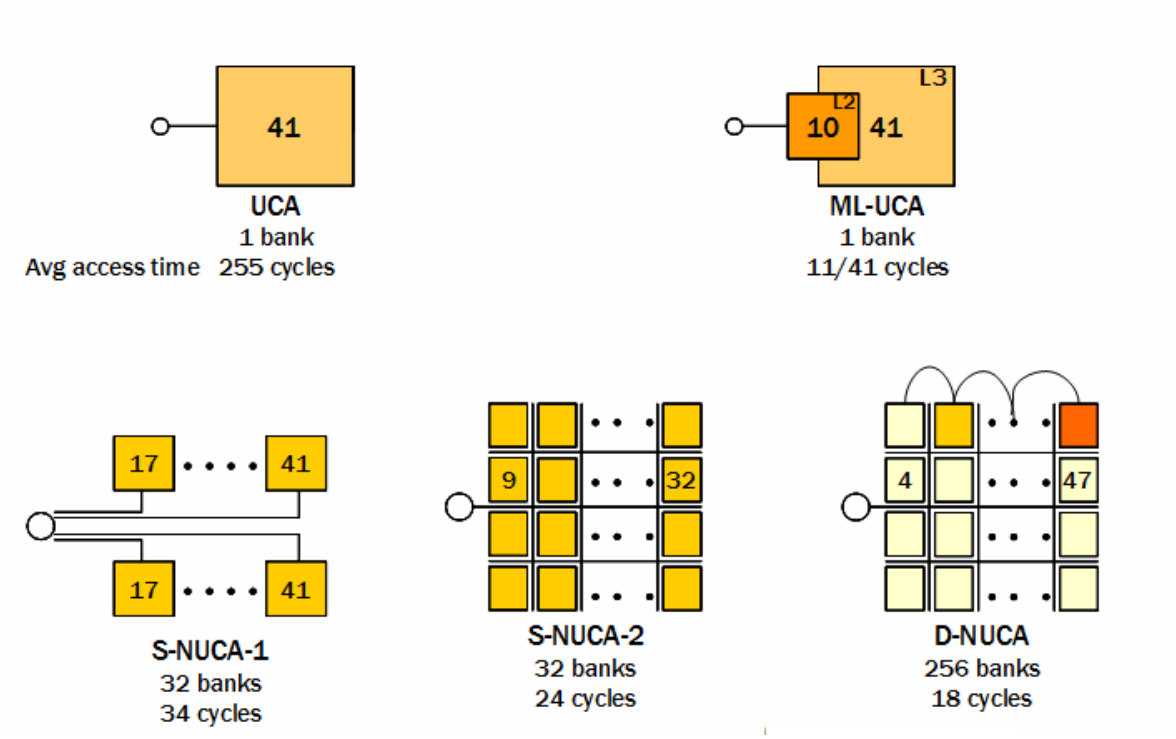


Figura 2.2: Tiempo de acceso medio en función de la configuración de la memoria.

- M (Modified): Un bloque en estado modificado mantiene la única copia válida de los datos. El núcleo que mantiene esta copia en su caché tiene permisos de lectura y escritura sobre el bloque. Las otras cachés privadas no pueden tener una copia. La copia en la caché L2 compartida (si está presente) es obsoleta. Cuando otro núcleo solicita el bloque, la caché con el bloque en estado modificado debe proporcionarlo.

- O (Owner): Un bloque en estado propietario mantiene una copia válida de los datos, pero en este caso, otras copias en estado compartido pueden coexistir. Únicamente puede haber una copia de ese bloque en estado propietario. El núcleo manteniendo esta copia en su caché tiene permisos de lectura, pero no puede modificarlo. Cuando este núcleo trata de modificarlo, se requieren acciones de coherencia para invalidar el resto de copias. De esta forma, el estado propietario es similar al compartido. La diferencia reside en el hecho de que el estado propietario es responsable de proporcionar la copia del bloque ante un fallo de caché, ya que la copia en la caché L2 compartida (si está presenta) es obsoleta. Además, los desalojos de bloques en estado propietario siempre conllevan operaciones de writeback.

- E (Exclusive): Un bloque en estado exclusivo mantiene una copia válida de los

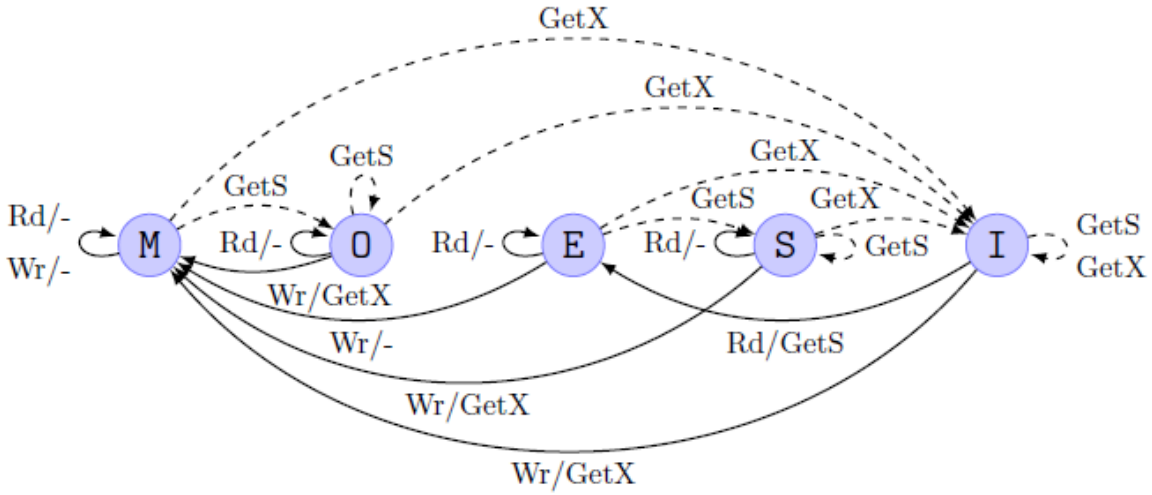


Figura 2.3: Diagrama de transición de estados para un protocolo MOESI.

datos. Las otras cachés privadas no pueden tener una copia de este bloque. El núcleo con esta copia tiene permisos de escritura y lectura. La caché L2 compartida puede tener también almacenada una copia válida del bloque.

- S (Shared): Un bloque en estado compartido mantiene una copia válida de los datos. Otros núcleos pueden tener copias del bloque en estado compartido y uno de ellos en estado propietario. Si ninguna caché privada tiene el bloque en estado propietario, la caché L2 compartida dispone también de una copia válida del bloque y es responsable de proporcionarlo si fuera solicitado.
- I (Invalid): Un bloque en estado inválido no mantiene ninguna copia válida de los datos. Las copias válidas pueden encontrarse bien en la caché L2 compartida o en otra caché privada.

2.2.2. Protocolos de actualización e invalidación

Cuando se produce una escritura en una caché de un bloque que ya se encontraba almacenado en la caché de otro núcleo, es necesario realizar acciones específicas para poder mantener la coherencia en el sistema de memoria. Existen dos principales opciones: protocolos de actualización y protocolos de invalidación.

En un protocolo de actualización, cuando un núcleo escribe sobre un bloque, se actualizan todas las demás copias del resto de cachés del CMP. Los accesos posteriores a este bloque obtendrán una copia actualizada del mismo. Para poder llevar esto a

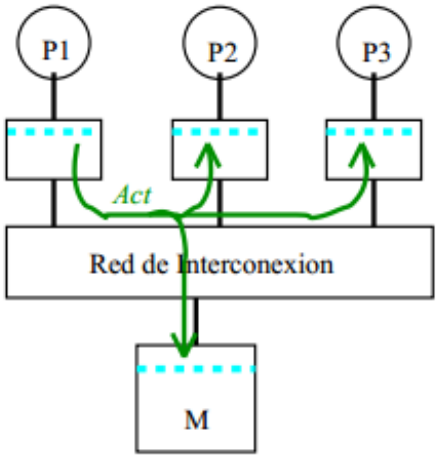


Figura 2.4: Ejemplo de funcionamiento de un protocolo de actualización.

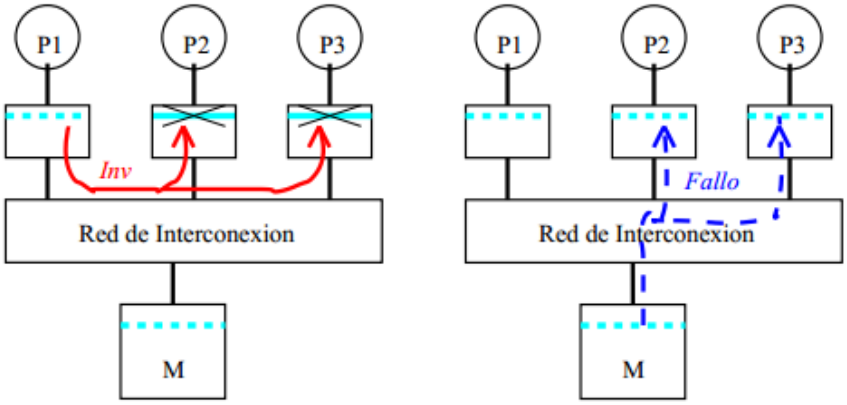


Figura 2.5: Ejemplo de funcionamiento de un protocolo de invalidación.

cabo, es necesario que se avise de esta modificación al resto de cachés, indicando a su vez el número de bloque implicado y el valor actualizado. Una vez actualizado un bloque, las operaciones de lectura por parte de otros procesadores no originan fallos de bloque. Esto es especialmente bueno cuando hay escrituras por parte de un procesador seguidas por una secuencia de lecturas de otros procesadores.

Por el otro lado tenemos los protocolos de invalidación, en el que cuando un núcleo escribe sobre un bloque, todas las copias del resto de cachés son invalidadas. En este caso, los accesos posteriores originarán un fallo de bloque. Tras la gestión de este fallo, la caché obtendra una copia actualizada. Mediante este método, una vez invalidado un bloque, nuevas modificaciones por parte del mismo núcleo no originarán nuevas invalidaciones. Por tanto, el caso en el que mejor funciona un protocolo de invalidación es aquel en el que se realizan múltiples escrituras consecutivas por parte de un mismo núcleo.

Para este trabajo se ha escogido este tipo de protocolos dado que es más rápido y sencillo de implementar, pues tan solo hay que indicar el número del bloque al resto de cachés, que uno de actualización.

2.2.3. Protocolos snoopy y basados en directorio

A la hora de implementar los mecanismos de coherencia, por ejemplo las órdenes de invalidación o actualización ya explicadas en el apartado anterior, en las cachés del sistema disponemos de dos opciones: protocolos basados en *snooping* y protocolos de directorio.

En los protocolos basados en snooping estas órdenes deben enviarse a todas las cachés de todos los núcleos, tengan o no copia del bloque, y son ellas las que deben descubrir por si mismas si la orden les afecta o no. Para que esto sea viable se requiere de una de red de interconexión que permita realizar la operación de difusión con facilidad (MPs. de memoria compartida centralizada, bus común). Además el sistema debe imlementar un mecanismo de monitorización continua del bus para interceptar las órdenes de invalidación/actualización y dotar al bus de líneas específicas para dar soporte al protocolo en concreto que se este utilizando. La principal desventaja de este tipo de protocolos es que cuan mayor es el número de núcleos en el CMP mayor es el tráfico inducido en la red.

Los protocolos de directorio pretenden evitar las limitaciones de escalabilidad e interconexión de los protocolos basados en *snooping*. Los sistemas que emplean estos protocolos son los más recomendados cuando la escalabilidad (en el número de procesadores) es un factor de diseño importante.

Uno de los de los objetivos de los protocolos basados en directorio es evitar el uso del broadcast. En su lugar, las comunicaciones se realizan solo entre aquellos procesadores que posiblemente tengan los datos en sus cachés. Para ello, necesitamos una estructura que almacene si una línea de memoria está en alguna caché privada, qué procesadores tienen una copia y si dicha línea está limpia o sucia. En un esquema de directorio completo se mantiene información sobre todas las líneas de memoria. Por ejemplo, para un sistema de n núcleos, cada uno con su caché privada, se necesita un vector booleano de tamaño n+1. Si un bit i (i=1,...,n) está a cierto, indica que el pro-

cesador i-ésimo tiene una copia de la línea. El bit (i=0) indica si la línea está limpia o sucia. Un vector completamente a 0 indica que la línea se encuentra exclusivamente en memoria principal. Si el bit (i=0) está activo, la línea está sucia y tan solo uno más de los otros bits puede estar activo también. Con esto conseguimos que el tráfico de la red crezca linealmente con el número de procesadores, mientras que con los protocolos basados en snooping este crecimiento es cuadrático. Desde este punto de vista se consigue una mejor escalabilidad.

No obstante, almacenar la información de todas y cada una de las líneas de memoria en un directorio completo requiere bastante área, incurriendo de nuevo en restricciones de escalabilidad. Por eso se utilizan cachés de directorio que funcionan de forma similar a un directorio completo. La mayor diferencia entre ambos es que cuando la caché de directorio se ve obligada a expulsar una línea por falta de espacio (cosa que no puede suceder en un directorio completo), los bloques de las cachés privadas deben invalidadarse, aunque estén siendo utilizados para poder mantener la coherencia. De esta manera se obtiene un nuevo tipo de fallo que se suma a los inherentes a las cachés privadas (arranque, conflicto y capacidad) y a la de los protocolos en sistemas de memoria compartida (coherencia). Son los llamados fallos de coverage, fallos que se producen al acceder a un bloque que ha tenido que expulsarse por falta de espacio en la caché de directorio.

En este proyecto se realizará un estudio sobre esta estructura, planteando una nueva partiendo de ésta como base que pretende mejorar el consumo y área, pero manteniendo las prestaciones. Para ello se apoyará en el comportamiento diferenciado presente entre bloques privados y compartidos.

2.3. Tecnologías de memoria

Los sistemas CMP deben diseñarse para ajustarse a presupuestos específicos de área y energía. Ambas restricciones tecnológicas representan un problema general, dado que dificultan la escalabilidad de los futuros CMPs con el incremento de núcleos. El consumo de energía está distribuido entre los núcleos y las grandes cachés de memoria en los diseños de chips actuales. Las cachés ocupan un elevado porcentaje del área del chip para mitigar las altas latencias que corresponden con un acceso a memoria principal. Dando más área de silicio y energía a la jerarquía de memoria y estructuras relacionadas (por ejemplo las cachés de directorio) deja menos espacio y energía para

los núcleos, lo que fuerza a los diseños de CMPs a usar núcleos más simples reduciendo la productividad, especialmente para aplicaciones de un solo hilo [MH08].

Han habido muchos esfuerzos por parte de la industria y el mundo académico para enfrentarse al problema de área y energía en el subsistema de caché, incluyendo las cachés del procesador, cachés fuera del chip y estructuras de directorio. Con respecto a estas últimas estructuras, las cachés de directorio han demostrado ser efectivas para escalar tanto en energía como en área cuando el número de núcleos es medio o bajo. En cualquier caso, estas dificultades de diseño deben afrontadarse correctamente para sistemas futuros, ya que la presión de obtener buenas prestaciones incrementa con el número de núcleos. Hay dos formas de aproximarse a estas dificultades: ofrecer soluciones estructurales para conseguir un buen balance entre productividad, área y consumo, y combinar distintas tecnologías. Ambos casos se pueden aplicar de manera independiente o conjunta.

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 [MS05]. La Tabla 2.1 muestra como estas tecnologías se comportan para los distintos aspectos de diseño estudiados en este artículo. Comparadas con las celdas SRAM, las celdas eDRAM presentan 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 descritas ha sido empleada tanto en la industria como el ámbito académico, pero a diferencia de nuestro trabajo, ellos se centraban en cachés de procesador convencionales. Por ejemplo, en algunos microprocesadores modernos [TDF+02, SKT+05, KSSF10] 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

Cuadro 2.1: Características de las tecnologías eDRAM y SRAM.

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

[VSP+09, WLZ+09] se han publicado mezclando ambas tecnologías en la misma y/o diferentes estructuras de la jerarquía de memoria.

Uno de los puntos que se estudiará en este proyecto es el uso de las distintas tecnologías a la hora de implementar la estructura propuesta.

Capítulo 3

Trabajo relacionado

Como se ha mencionado previamente, el constante aumento en el número de los núcleos que podemos encontrar en los chips multicore actuales, ha creado la necesidad de encontrar nuevos mecanismos de coherencia escalables que permitan seguir este ritmo. Las implementaciones del directorio, tanto en el ámbito académico como en la industria, que intentan abordar esta problemática 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 este modelo se han implementado algunos CMPs modernos [SBB07] y son la base de algunos trabajos de investigación recientes [RAG10, ZSQM09]. El principal inconveniente de esta aproximación es el grado de asociatividad requerido 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 [FLKBF11] utiliza diferentes funciones hash para indexar cada vía del directorio, como las cachés *skew-associative*. Los aciertos tan solo requieren de un acceso, pero los reemplazos 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. Su idea fundamental se origina en el hecho de que en una indexación de caché tradicional, si un bloque A tiene un conflicto con un

bloque B y este bloque B tiene un conflicto con un bloque C, entonces el bloque A también tiene un conflicto con el bloque C. En caso de disponer únicamente de una estructura con dos vías, si B y C están ya presentes en dicha estructura, para poder insertar A tendremos que reemplazar B o C. Para intentar romper esta relación transitiva hace uso de las funciones hash y de reubicación de bloques para limitar el número de expulsiones, evitando de esta forma la pérdida de prestaciones.

Los directorios *sparse* [GWM90] 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 aumenta con el número de núcleos.

Para acortar el tamaño de las entradas algunas propuestas utilizan compresión [AGGD01, AGGD05, Che93, ON90]. En [AGGD01] también 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 [SK12] utiliza un formato de entrada distinto y del 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.

Enright *et al.* proponen el *Virtual Tree Coherence* (VTC) [EJPL08]. Este mecanismo utiliza un seguimiento de coherencia de grano grueso [CSL+06] y los compartidores de una región de memoria están conectados mediante un árbol virtual. Dado que la raíz del árbol virtual sirve como punto de ordenación en lugar del *home tile* y el tile raíz es uno de los compartidores de la región, la indirección se puede evitar en algunos fallos. En comparación, los protocolos de coherencia directa mantienen la información de coherencia con una granularidad de bloque y el punto de ordenación siempre tiene una copia válida del bloque, lo que conduce a un menor tráfico de la red y menor nivel

de indirección.

Huh *et al.* proponen permitir la replicación en una caché NUCA para reducir el tiempo de acceso en una caché compartida con múltiples bancos [HKS+05]. Por el mismo camino, Zhang *et al.* proponen una replicación de víctima [ZA05]. una técnica que permite que algunos bloques expulsados de la caché L1 sean almacenados en el banco L2 local. De esta forma, el siguiente fallo de caché para este bloque lo encontrará en su tile local, reduciendo así la latencia de fallo. Más recientemente, Beckemann *et al.* [BMW06] presentar el ASR (Adaptative Selective Replication) que replica los bloques de caché únicamente cuando se estima que el beneficio de esta replicación (una menor latencia de acierto en la L2) excede su coste (más fallos en la L2). Estas técnicas podrían también implementarse junto a protocolos con coherencia directa.

Chang y Sohi proponen el *Cooperative Caching* [CS06], un conjunto de técnicas que reducen el número de accesos fuera del chip en un CMP con una organización de caché privada para el último nivel de caché (las cachés L2 en este caso). A diferencia de otros trabajos, ellos asumen una organización privada, en el que los bloques son inherentemente replicados en las cachés L2 permitiendo unos accesos rápidos a la L2, e intentan eliminar las copias de bloques replicados para poder mejorar la tasa de acierto de la caché L2. De nuevo, estas técnicas se pueden implementar junto a protocolos de coherencia directa, ya que pueden emplearse en configuraciones privadas y compartidas.

Martin *et al.* presentan una técnica que permite a los protocolos basados en snooping utilizar redes desordenadas añadiendo una sincronización lógica a las peticiones de coherencia y reordenándolas en su destino para establecer el orden total [MSA00]. De la misma forma, Agarwal *et al.* proponen *In-Network Snoop Ordering* (INSO) [APJ09] para permitir el snooping en las redes desordenadas. Ya que los protocolos de coherencia directa no se basan en peticiones de difusión, generan menos tráfico y, por tanto, un menor consumo de energía comparados con los protocolos basados en snooping.

Martin *et al.* proponen usar una predicción del conjunto destino para reducir el ancho de banda necesario por un protocolo basado en snooping [MHS+03]. Esta propuesta está basada en una interconexión completamente ordenada (un switch crossbar), que no escala con el número de núcleos. La predicción de conjunto de destino también es usada por *Token-M* en multiprocesadores de memoria compartida con redes desordenadas [Mar03]. No obstante, ante fallos en la predicción, las peticiones son resueltas

mediante difusión tras un periodo de time-out.

Cheng *et al.* [CMR+06] adaptan protocolos de coherencia ya existentes para reducir el consumo de energía y el tiempo de ejecución en CMPs con redes heterogéneas. En particular asumen una red heterogénea comprendida por varios conjuntos de cableados, cada uno de ellos con una latencia, ancho de banda y características de energía distintas. Proponen enviar cada mensaje de coherencia a través de un conjunto de cableado en particular dependiendo de sus requisitos de latencia y ancho de banda.

Finalmente, otras propuestas [CRG+11] se centran en reducir el número de entradas implementadas en la caché de directorio en vez de centrarse en el vector de presencia. Para ello se aprovechan del elevado número de bloques privados en las aplicaciones desactivando las acciones de coherencia para este tipo de bloques. El tipo del bloque se define a nivel de página, es decir, cuando se trae una página a memoria, todos sus bloques son considerados privados y, por tanto, no se accede en ningún momento al directorio. No se mantiene ninguna información del procesador que mantiene el bloque y, por tanto, tampoco van circulando peticiones de escritura, lectura o de invalidación sobre los mismos. Cuando un bloque de esa misma página es referenciado por otro núcleo, entonces todos los bloques de esa página pasan a consideradarse compartidos y el sistema debe restaurar en ese instante la coherencia. Mientras que esta aproximación no afecta a la productividad, requiere modificar el SO, la tabla de paginación, las TLBs del procesador y el protocolo de coherencia.

Capítulo 4

Estructura de directorio propuesta

4.1. Arquitectura base

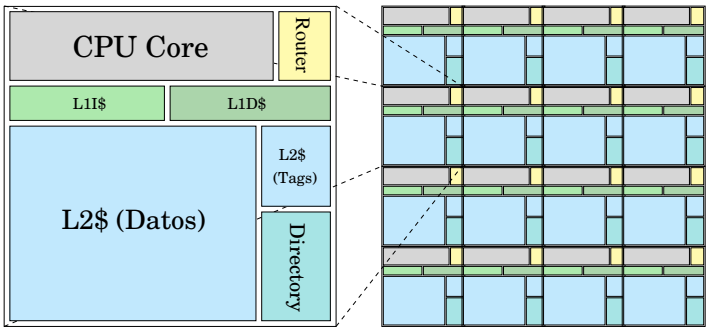


Figura 4.1: Organización de un tile y de un CMP 4×4.

La arquitectura de CMP estudiada en este trabajo consiste en un número de tiles replicados y conectados mediante una red de interconexión directa. Hay diferentes organizaciones distintas pero en este proyecto se asume que cada tile contiene un núcleo de procesamiento con sus cachés prvadas (instrucciones y datos), un fragmento de la caché L2 y una interfaz con la red de interconexión del chip. La coherencia se mantiene a nivel de L1. En concreto, se usa un protocolo de coherencia basado en directorio donde la información de coherencia se almacena en una caché de directorio. Tanto la caché L2 como la del directorio están compartidas por los distintos núcleos, pero se encuentran distribuidas físicamente entre los tiles. Por tanto, una fracción de los accesos a la caché L2 se enviará al slice local, mientras que el resto lo harán a slices remotos (arquitectura L2 NUCA [KBK02]). Además, las cachés L1 y L2 son no inclusivas, lo cual implica que los bloques almacenados en las cachés L1 no tienen una entrada en la caché L2 (pero si en el directorio). La figura 4.1 muestra la organización de un tile

(izquierda) y el de un CMP de 16 tiles (derecha).

4.2. Esquema 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 de los distintos comportamientos que exhiben los bloques privados y compartidos.

La Figura 4.2 muestra la organización propuesta compuesta por 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 trabajos recientes [HFFA09, CRG+11], así que la caché Privada se ha diseñado con un número de entradas mucho mayor. La anchura de las entradas es diferente a causa de que vector de presencia (que no escala con el número de núcleos) 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 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 (fallo en el 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 la coherencia. Luego, si el acceso resulta en un fallo, el bloque es proporcionado por un banco de la NUCA (o por memoria principal) a la caché del procesador y se asigna una entrada 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 reserva una entrada 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 ex-

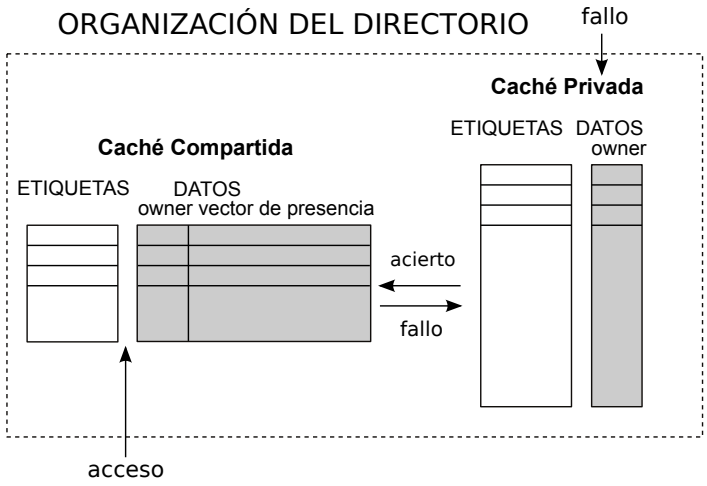


Figura 4.2: Organización del Directorio Privado-Compartido.

clusivo (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 rendimiento 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 dicho momento 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.

En cuanto a temporización, las cachés de directorio normalmente se acceden en paralelo con la caché NUCA y, ante un acierto en el directorio, en caso de que los datos del bloque deban proporcionarse por alguna caché de procesador, se cancela el acceso a la NUCA. De manera similar, se propone acceder a la caché Compartida en paralelo con el banco de la NUCA. La Figura 4.4 muestra esta decisión de diseño. Dependiendo del protocolo, las 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 redireccionarse al propietario del blpque 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 cuanto a productividad.

La Figura 4.3 resume las acciones realizadas por el controlador del directorio ante un acceso de coherencia. El directorio propuesto funciona de la siguiente manera:

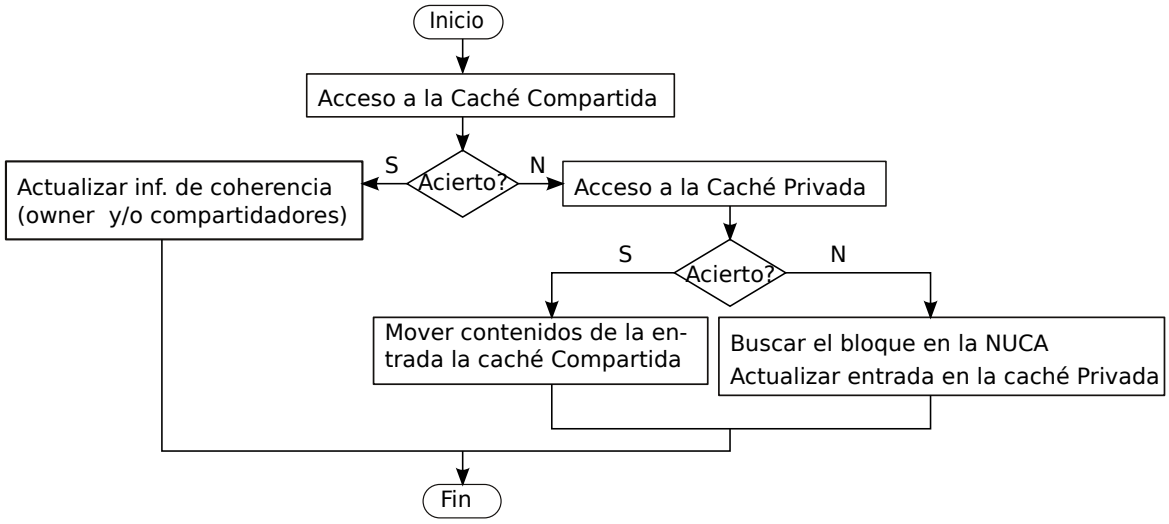


Figura 4.3: Diagrama de flujo del controlador de memoria.

- 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 resulte en acierto en esta caché dado que las entradas referentes a 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 controlador busca 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 núcleo 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, la información contenida en la entrada referente al bloque se mueve a la caché Compartida. Esta forma de proceder asegura que las entradas referentes a los bloques privados se queden en la caché Privada mientras que las referentes a los bloques compartidos son filtradas y transportadas a la caché Compartida.

- Ante un fallo en el directorio, una entrada se ubica en la caché Privada para poder mantener la información del bloque causante del fallo. 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 core solicitante) es actualizada.

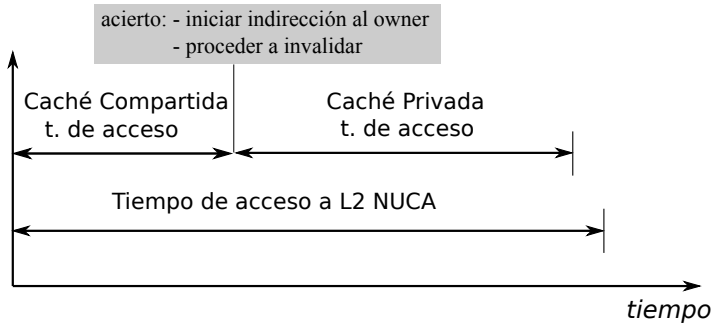


Figura 4.4: Acceso paralelo a la caché Compartida y a la NUCA. La caché Privada solo se accede si hay fallo en la Compartida.

- En la propuesta implementada, cuando una entrada de bloque se reemplaza de cualquiera de las dos cachés de directorio se elimina su información sin más y no se realiza ningún 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. Además, 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. Aunque esta propuesta puede implementarse exclusivamente con tecnología SRAM (6T cells), también estudiamos los beneficios del uso de tecnología eDRAM [MS05] que ya ha sido utilizada para implementar cachés grandes de algunos procesadores comerciales actuales como el IBM Power7 [KSSF10]. 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. De esta forma,se consigue ganar en escalabilidad y rendimiento gracias a un diseño en el que se emplean técnicas estructurales y una elección de tecnología apropiada para cada una de las cachés del directorio.

Capítulo 5

Entorno de simulación

5.1. Herramientas de simulación

En esta sección se describirán las herramientas de simulación utilizadas durante este proyecto. Los datos y estadísticas de las cargas de trabajo se obtendrán utilizando el simulador Simics-GEMS. Los requisitos de área, el consumo y las latencias de acceso a las cachés usadas en las simulaciones de las propuestas de este proyecto han sido calculadas utilizando CACTI.

5.1.1. Simics-GEMS

Simics [MCE02] es un simulador de sistema completo capaz de simular distintos tipos de hardware, incluyendo sistemas multiprocesador. La simulación de un sistema completo nos permite evaluar nuestras ideas ejecutando cargas reales en sistemas operativos actuales. De esta forma también se simula el comportamiento del sistema operativo. A diferencia de los simuladores guiados por trazas, Simics permite un cambio dinámico de las instrucciones a ejecutar en función de los distintos datos de entrada.

GEMS (General Execution-dircen Multiprocessor Simulator) [MSB05]es un entorno de simulación que extiende Virtutech Simics- GEMS está compuesto por un conjunto de módulos implementados en C++ que se añaden a Simics y le otorgan al simulador capacidades de temporización. GEMS ofrece varios módulos para modelar distintos aspectos de la arquitectura. Por ejemplo, Ruby modela la jerarquía de memoria, Opal modela la temporización de un procesador SPARC fuera de orden. y Tourmaline es un simulador de memoria transaccional.

Ruby ofrece un framework dirigido por eventos para simular la jerarquía de memoria que es capaz de medir los efectos en los cambios de los protocolos de coherencia. Particularmente, Ruby incluye un lenguaje específico para especificar los protocolos de coherencia denominado SLICC (Specification Language for Implementing Cache Coherence). SLICC permite desarrollar fácilmente diferentes protocolos de coherencia y se ha utilizado para implementar los protocolos evaluados en este trabajo.

El modelo de memoria de Ruby está compuesto por un número de componentes que modelan las cachés L1, las cachés L2, los controladores de memoria y los controladores del directorio. Estos componentes modelan el tiempo calculando la diferencia entre que se recibe una petición hasta que se genera una respuesta que es inyectada en la red. Todos los componentes están conectados mediante un modelo de red simple que calcula el tiempo necesario para transportar el mensaje de un componente al siguiente.

5.1.2. CACTI

CACTI (Cache Access and Cycle Time Information) [MBJ09] dispone de modelos de tiempo de acceso a memoria, tiempo de ciclo, área, leakage y consumo dinámico. Integrando todos estos modelos, los usuarios pueden tener la certeza de que las compensaciones entre tiempo, consumo y área están basadas en los mismos supuestos y, por tanto, son consistentes entre sí. CACTI es constantemente actualizado debido a las incesantes mejoras en las tecnologías de semiconductores. Se ha utilizado la versión 6.5 para estimar los tiempos de acceso, requisitos de área y consumo de energía de las diferentes estructuras caché para nodos con tecnología de 32nm.

5.1.3. Sistema simulado

Se ha simulado una arquitectura CMP de 16 tiles, aunque también se muestran los valores de escalabilidad para área y consumo hasta 1024 núcleos. Los valores de los parámetros principales pueden verse en la Tabla 5.1.

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 cobertura de 1×. El ratio de cobertura indica el número de entradas del directorio por entrada en la caché del núcleo. Para la configuración base el directorio tiene el mismo número de entradas que una caché L1 de procesador (1×). Los directorios PS evaluados varían tanto en este ratio de cobertura (desde 1× hasta 0,125×)

Cuadro 5.1: 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

Cuadro 5.2: Latencias del directorio

		1×	**Ratio de Coverage**			
Caché de directorio	# Vías	# Cjtos	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

como en el ratio entre la caché Compartida y 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 5.2 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 da las latencias en ns y por tanto se han redondeado estos valores para obtener ciclos de procesador. La caché L2 se asume que es de 6 ciclos, y el resto de tiempos de acceso se han escalado de manera proporcional. El número de vías es independiente al ratio de cobertura, pero el número de conjuntos disminuye a la mitad cada vez que se reduce a la mitad el ratio de cobertura.

5.2. Métricas y metodología

La propuesta presentada en este trabajo se evalúa en el capítulo 6 en términos de prestaciones, área requerida en el chip y consumo. Para evaluar las prestaciones se mide el número de ciclos totales que ha necesitado la ejecución de cada aplicación durante su fase paralela, es decir, el tiempo de ejecución de la fase paralela. Aunque el IPC (instrucciones por ciclo) constituye una medida común para evaluar las mejoras en las prestaciones, no es apropiado para aplicaciones multihilo lanzadas en sistema multiprocesador [AW06]. Esto es debido a las tareas realizadas durante la fase de sincronización de los distintos hilos. Por ejemplo, un hilo puede estar comprobando constantemente el valor de un *lock* hasta que este disponible, lo que incrementa el número de instrucciones completadas (y seguramente también el IPC), pero a nivel efectivo el programa no está realizando ningun tipo de progreso.

Para poder analizar los motivos de la mejora del tiempo de ejecución de la propuesta, se miden también los fallos de caché L1, ya que un fallo de caché L1 se corresponde con un acceso a la estructura de directorio, el elemento de la jerarquía de memoria sobre el cual se centra todo este trabajo. Estos fallos, a su vez, se desglosan en tres tipos de fallo (3C, coherencia y cobertura) para poder saber concretamente cómo ataca el directorio PS a cada uno de ellos. De especial interés son los fallos de cobertura, dado que uno de los puntos principales del diseño del directorio propuesto es reducirlos. Finalmente, estos fallos de L1 también se han desglosado en aciertos y fallos al directorio, de nuevo para poder hacer un estudio de porque mejoramos o empeoramos en el tiempo de ejecución. Un fallo en el directorio, suponiendo que ya esté completo, va a suponer el reemplazo de una entrada en el directorio y, como consecuencia, es posible que se tenga que expulsar ese bloque de una o varias cachés L1, que en caso de que vuelva a ser referenciado impactará negativamente en las prestaciones. Por todo lo mencionado anteriormente se consideran estas métricas importantes para el estudio.

El consumo de potencia es una preocupación de diseño en los actuales CMPs, ya que influye en los costes de encapsulado (package) y refrigeración. Por este motivo, los diseños de procesadores deben ajustarse a unas restricciones de potencia específicas (target power budget). Esto significa que cuando se diseña un procesador, no pueden evaluarse sus prestaciones de manera aislada sino que deben considerarse el coste energético.

La energía disipada tiene dos componentes principales: energía dinámica y energía estática o de leakage. La energía dinámica es disipada debido a los cambios de nivel en los transistores, mientras que la energía estática está siendo continuamente disipada, incluso cuando el transistor está inactivo. La energía estática es proporcional al número de transistores, es decir al área en las estructuras regulares como las memorias cache, por tanto reduciendo el tamaño de una determinada estructura se reduce su consumo. En este trabajo se evaluarán las prestaciones del directorio evaluando también tanto su consumo energético como el área ocupada.

Todos los protocolos de coherencia de caché evaluados en este trabajo se han implementado utilizando el lenguaje SLICC incluido en GEMS. Todos los protocolos implementados se han comprobado exhaustivamente usando el programa de test que ofrece GEMS. El programa de test estresa el protocolo de coherencia realizando muchas peticiones que simulan accesos frecuentes a unos pocos bloques de memoria, consiguiendo así mostrar cualquier tipo de incoherencia o defecto en la implementación.

Todos los resultados experimentales recogidos en este proyecto se corresponden con la fase paralela de estos benchmarks. Para cada benchmarks se han creado puntos de control en los que cada aplicación ya ha sido previamente ejecutada para asegurarse de que la memoria se ha calentado, evitando así los fallos de paginación. Luego ejecutamos cada aplicación otra vez hasta la fase paralela en la que cada hilo ya ha sido asignado a un núcleo. Entonces se ejecuta la aplicación con todo detalle durante la inicialización de cada hilo antes de empezar a tomar las medidas. De esta forma calentaos las cachés para evitar fallos de arranque.

5.3. Benchmarks

La propuesta se ha evaluado 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 (512 keys, 1024 radix), Raytrace (teapot), Volrend (head), and Water-Nsq (512 molecules) del suite de benchmarks SPLASH-2 [32]. También se han empleado Blackscholes (simmedium) y Swaptions (simmedium) que pertenecen a las PARSEC [33]. Como ya se ha dicho, todos los resultados experimentales recogidos en este proyecto se corresponden con la fase paralela de estos benchmarks.

5.3.1. Barnes

La aplicación *Barnes* simula la interacción de un sistema de cuerpos (galaxias o particulas, por ejemplo) en tres dimensiones a lo largo de unos intervalos de tiempo, utilizando el método jerárquico N-cuerpos de Barnes-Hut. Cada cuerpo se modela como un punto con masa que ejerce una fuerza a todos los otros cuerpos del sistema. Para acelerar el cálculo de la interacción de fuerzas, los grupos de cuerpos que están suficientemente separados se abstraen también como un único punto con masa. Para facilitar este agrupamiento, el espacio físico se divide recursivamente, formando un *octree*. Esta representación en árbol del espacio debe ser atravesada una vez por cada cuerpo y reconstruida en cada intervalo de tiempo para tener en cuenta el movimiento de los cuerpos.

La estructura de datos principal en Barnes es el árbol en si mismo, que se implementa como una matriz de cuerpos y una matriz de celdas de espacio que están unidas. Los cuerpos son asignados a los núcleos al principio de cada intervalo de tiempo en una fase de particionamiento. Cada núcleo calcula las fuerzas ejercidas en su propio subconjunto de cuerpos. Los cuerpos se mueven entonces bajo la influencia de estas fuerzas. Finalmente, el árbol es regenerado para la siguiente iteración. Hay varias barreras que separan las diferentes fases de computación y los intervalos de tiempo sucesivos. Algunas fases requieren acceso exclusivo a las celdas del árbol y un conjunto de locks se usan para este propósito. Los patrones de comunicación son dependientes de la distribución de las partículas y es bastante irregular. No se hace ningún intento inteligente en la distribución de los datos de los cuerpos en memoria principal, ya que es difícil a nivel de página y no es demasiado importante para la productividad.

5.3.2. FFT

El kernel *FFT* es una versión compleja unidimensional del algoritmo FFT radix-$\sqrt{n}$ de seis pasos que está optimizado para minimizar la comunicación entre procesadores. El conjunto de datos consiste en los n puntos de datos complejos a ser transformados, y otros n puntos de datos complejos a los que nos referimos como raíces n-ésimas de la unidad. Ambos conjuntos de datos se organizan como matrices particionadas de $\sqrt{n}$ x$\sqrt{n}$ de tal forma que a cada núcleo se le asigna un conjunto de filas contiguas que se ubican en su memoria local. La sincronización en esta aplicación se consigue mediante el uso de barreras.

5.3.3. Ocean

La aplicación *Ocean* estudia los movimientos a gran escala del océano basados en remolinos y corrientes limítrofes. El algoritmo simula un recipiente cuboidal usando un modelo de circulación discreto que tiene en consideración el viento de efectos atmosféricos y la fricción del océano con suelos y paredes. El algoritmo realiza la simulación durante varios intervalos de tiempo hasta que los remolinos y el flujo medio del océano consiguen un equilibrio mútuo. El trabajo realizado en cada intervalo requiere esencialmente del planteamiento y resolución de un conjunto de ecuaciones parciales diferenciales espaciales. Para este fin, el algoritmo discretiza las funciones contínuas mediando diferenciación finita de segundo orden, coloca la ecuaciones diferenciales resultantes en grids bidimensionales de tamaño fijo que representan las intersecciones horizontales del recipiente del océano, y resuelve estas ecuaciones usando el solucionador de ecuaciones multigrid de Gauss-Seidel. Cada tarea realiza pasos computacionales en la sección de los grids de la que es propietario, comunicandose regularmente con otros procesos. La sincronización se realiza utilizando tanto locks como barreras.

5.3.4. Radiosity

Esta aplicación computa el equilibrio de la distribución de la luz en una escena usando el método jerárquico diffuse radiosity. Una escena se modela inicialmente como un elevado número de polígonos. Se computa la interacción en el transporte de la luz a lo largo de todos estos polígonos, luego se subdividen jerárquicamente como sea necesario para mejorar la precisión. En cada paso, el algoritmo itera sobre la lista de interacciones actuales, las subdivide recursivamente, y modifica la lista de interacción como sea necesario. Al final de cada paso, se unen todos los conjuntos para comprobar si hay convergencia. La estructura de computación y los patrones de acceso son altamente irregulares. Se consigue el paralelismo mediantes colas de tareas distribuidas, una por núcleo, con robo de tareas para el equilibrio de la carga. No se realiza ningún intento para hacer una distribución inteligente de los datos.

5.3.5. Radix

El programa *Radix* ordena una serie de enteros, llamados *claves*, usando el popular método de ordenamiento radix. Este algoritmo es iterativo, realizando una iteración para cada dígito r radix de las claves. En cada iteración, un núcleo repasa sus claves asignadas y genera un histograma local. Los histogramas locales se acumulan entonces

en un histograma global. Finalmente, cada núcleo usa el histograma local para permutar sus claves en un nuevo array para la siguiente iteración. Esta permutación require de comunicación todos con todos. La permutación es inherentemente determinada por el emisor, asi que las claves son comunicades mediante escrituras antes que lecturas. La sincronización se consigue mediante barreras.

5.3.6. Raytrace

Esta aplicación renderiza una escena 3-dimensional usando ray tracing. Se emplea un grid jerárquico y uniforme para representar la escena. Se sigue un rayo a través de cada pixel en el plano de imagen y produce otros rayos conforme colisiona con los objetos de la escena, resultando en un árbol de rayos por pixel. La imagen se particiona entre los núcleos en bloques contiguos de grupos de pixeles. Se usan colas de tareas distrbuidas con robo de tareas. Los accesos a los datos son altamente impredecibles en esta aplicación. La sincronización en Raytrace se consigue con el uso de locks. Este benchmark se caracteriza por tener secciones críticas de muy corta duración y una alta contención. No se usan barreras para la aplicación *Raytrace*.

5.3.7. Volrend

La aplicación *Volrend* renderiza un volumen 3-dimensional usando una técnica de ray casting. El volumen se representa como un cubo de voxels (elementos de volumen), y se emplea una estructura de datos octree para desplazarnos por el volumen rápidamente. El programa renderiza varios frames desde distintos puntos de vista. Un rayo es disparado a través de cada pixel en cada trama, pero los rayos no se reflejan. En cambio, los rayos se muestrean a lo largo de sus rutas lineales usando interpolación para computar el color correspondiente al pixel. El particionamiento y las colas de tareas son muy similares a las de Raytrace. Los accesos a los datos son dependientes a la entrada e irregulares y, por tanto, no se hace ningun intento para optimizar la distribución de datos. La sincronización en esta aplicación se consigue principalmente con el uso de locks, pero también se incluyen algunas barreras.

5.3.8. Water-Nsq

La aplicación *Water-Nsq* realiza una simulación de la dinámica molecular de N-cuerpos de las fuerzas y potenciales de un sistema de moléculas de agua. Se utiliza

para predecir algunas de las propiedades fisicas del agua en su estado líquido.

Las moléculas se distribuyen estáticamente entre los núcleos y la estructura principal de datos en Water-Nsq es una matriz, considerablemente grande de registros que se usan para almacenar el estado de cada molécula. En cada intervalo de tiempo, los núcleos calculan la interacción de los átomos dentro de cada molécula y la interacción que hay entre las moléculas en sí. Para cada molécula, el núcleo propietario calcula las interacciones con tan solo la mitad de las moléculas que tiene por delante en la matriz. Como las fuerzas entre las moléculas son simétricas, cada par de interacciones entre moléculas tan solo es considerada una única vez. El estado asociado a las moléculas es entonces actualizado.

Aunque algunas partes del estado de la molécula se modifican en cada interacción, otras tan solo se modifican entre los intervalos de tiempo. La gran parte de la sincronización se consigue usando barreras, aunque hay varias variables que contienen propiedades globales que son actualizadas constantemente y, por tanto, están protegidas mediante locks.

5.3.9. Blackscholes

El kernel *Blackscholes* está basado en un algoritmo de modelado financiero que utiliza ecuaciones diferenciales parciales para calcular los precios de de las opciones de mercado europeos. La idea fundamental es que el valor de la opción fluctúa a lo largo del tiempo con el valor actual del mercado. Se hace una computación intensiva en coma flotante y requiere del cálculo de logaritmos, exponenciales y raíces cuadradas.

5.3.10. Swaptions

La aplicación *Swaptions* es una carga de Intel RMS que utiliza el entorno de trabajo Heath-Jarrow-Morton (HJM) para poner precio a un portfolio de swaptions. El programa almacena el portfolio en la matriz swaptions. Cada entrada corresponde a una derivada. Swaptions particiona la matriz en un número de bloques igual al número de hilos y asigna un bloque a cada hilo. Cada hilo itera sobre todos los swaptions en la unidad de trabajo que le fue asignado y computa el precio.

Capítulo 6

Evaluación experimental

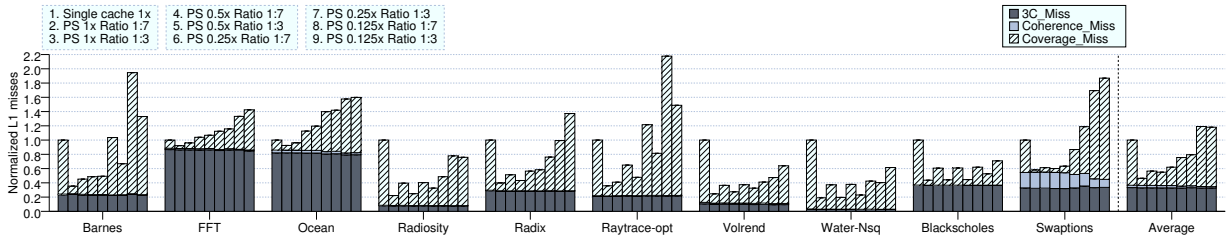
En esta sección se evalúa el directorio PS. Se analizan los resultados de prestaciones, área y consumo del directorio propuesto comparandolos con los de un directorio single caché tradicional.

6.1. Impacto en el tiempo de ejecución

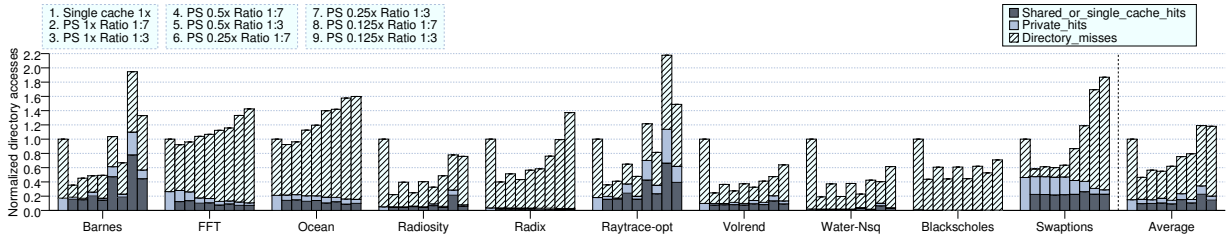
En esta subsección se evalúan las prestaciones del directorio PS. Una medida muy útil para esta evaluación son los fallos de *coverage* o cobertura. Cada vez que una entrada del directorio es expulsada, se envían mensajes de invalidación a las correspondientes cachés de los núcleos para poder seguir manteniendo la coherencia a nivel de caché. Estas invalidaciones causarán fallos de cobertura ante siguientes peticiones de memoria a esos bloques, impactando negativamente sobre la productividad final, pudiendo degradar significativamente las prestaciones.

La Figura 6.1(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 cobertura como se puede apreciar en la Figura 6.1(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 cobertura.

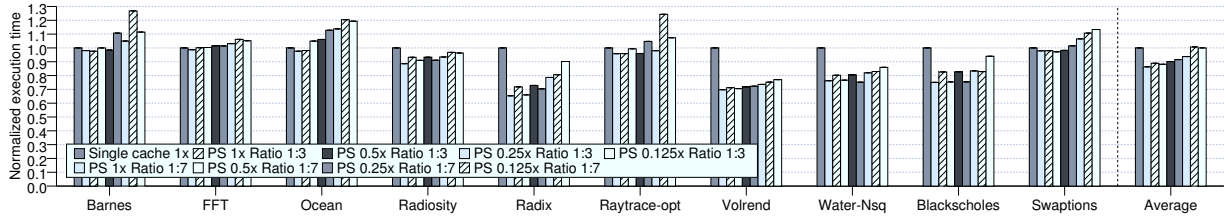
Por otro lado, podemos optar por reducir el tamaño del directorio PS y obtener un número de fallos de cobertura similar al de una caché individual. Los resultados



(a) Fallos de caché L1 normalizados.



(b) Accesos al directorio normalizados.



(c) Tiempo de ejecución normalizado.

Figura 6.1: Prestaciones normalizadas respecto a una caché de directorio convencional.

muestran que podemos reducir el directorio PS hasta 4× para, incluso así, tener menos fallos de cobertura 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 6.1(b) presenta la tasa de acierto clasificada por cada organización de caché. Como era de esperar, la caché Privada presenta una tasa de acierto bastante pobre a pesar de que su número de entradas es mucho mayor (3× y 7× 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 cobertura bajos, consigue de media unos resultados mejores que los del ratio 1:3, con la única excepción de una *cobertura* 0,125×.

Consiguiendo una reducción en el número de fallos de cobertura y en las latencias de acceso al directorio conseguimos mejoras en el tiempo de ejecución. La Figura 5(c) muestra estos resultados. Como se puede ver, el directorio PS con un ratio de cobertura 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, se puede optar por reducir la sobrecarga del área del directorio sin perder prestaciones con respecto a un directorio convencional. Este hecho se puede apreciar en la Figura 6.1(c), donde el directorio PS consigue las mismas prestaciones que una caché de directorio convencional con 8 veces menos entradas.

6.2. Análisis de energía y área

Esta sección muestra como el directorio PS es capaz de reducir el área y el consumo de energía, mientras aumenta la productividad.
En la Tabla 6.1 se puede ver el área requerida para los distintos esquemas PS y de la caché de directorio individual. Como es de esperar, todas las configuraciones PS son capaces de reducir el área, incluso aquellas con el mismo número de entradas (1x) que la caché de directorio convencional. Esto es a raíz de que la caché Privada no implementa el campo del vector de presencia. Primero, las configuraciones 1x PS que usan tecnología SRAM para la caché Privada ahorran un 18,85 % y un 26,35 % del área, para 1:3 y 1:7 respectivamente, en comparación con la caché individual. Como ya se ha mencionado, estos ahorros son a causa de no almacenar el vector de compartidores en la caché Privada. Por otra parte, cuando se considera eDRAM, estas reducciones aumentan hasta un 25,39 % y 33,98 % para los ratios 1:3 y 1:7 respectivamente. Además, cuando se reduce el ratio de cobertura (0.5x y 0.25x), la ganancia de área aumenta significativamente hasta un 80,24 % para la configuración 0.25x 1:7, consiguiendo incluso así mejorar los tiempos de ejecución como se muestra en la sección 6.1. Comparando los resultados de ambos ratios entre los dos niveles del directorio, podemos ver que las configuraciones con ratio 1:7 son más eficientes en área ya que son capaces de reducir el área desde un 12 % hasta un 25 % (dependiendo del ratio de cobertura del directorio) sobre las configuraciones con ratio 1:3, manteniendo aún así unas prestaciones similares.

El directorio PS también ataca al problema del consumo de energía, especialmente leakage, ya que se usan dos pequeñas estructuras complejas con menor capacidad que

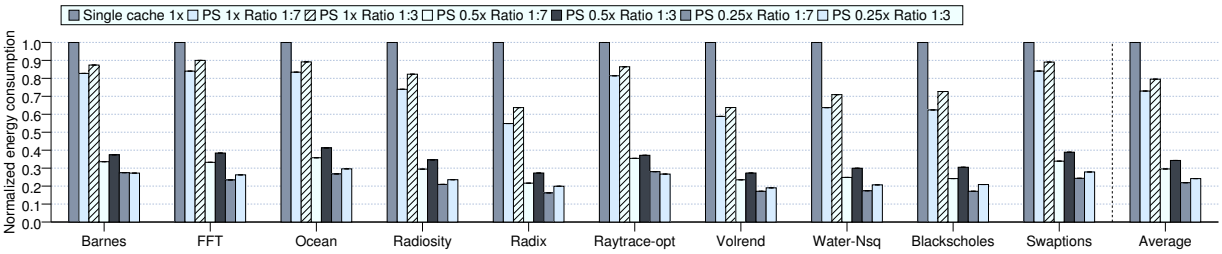


Figura 6.2: Energía consumida por el directorio normalizado respecto a una caché de directorio convencional.

Cuadro 6.1: Área (in $mm^2 * 1000$) de las configuraciones PS para 16 núcleos frente un directorio single caché 1×.

Cobertura	Directory	Private	Shared	Private	Total	Area (%)
1×	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×	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×	PS 1:3	eDRAM	1,74	3,01	4,76	24,39 %
	PS 1:7	eDRAM	0,84	3,01	3,85	19,76 %

una caché de directorio convencional. La Tabla 6.2 muestra la energía (dinámica y estática) consumida por las distintas configuraciones y comparadas con una 1x caché de directorio individual. Como se puede apreciar, las configuraciones 1x y 0.5x PS consumen más energía dinámica por acceso que una caché convencional, pero esto queda eclipsado por el significativo descenso en el consumo de leakage de las configuraciones PS, que se ve reducido incluso cuando se usa tecnología SRAM en la caché Privada. El leakage se reduce desde un 19 % para la configuración SRAM 1x 1:3 hasta un 86 % para la configuración eDRAM 0.25x 1:7. Comparando los ratios 1:3 y 1:7, podemos ver que las configuraciones 1:7 consiguen reducir el consumo de leakage desde un 5 % hasta un 15 % en comparación a las configuraciones 1:3. Teniendo en cuenta estos valores, la Figura 6.2 muestra la energía consumida por una caché de directorio individual. Podemos observar que el directorio PS con un mismo número de entradas (ratio de cobertura 1x) puede ahorrar sobre un 27 % y 2,5 % del consumo de una caché de directorio convencional para los ratios 1:7 y 1:3 respectivamente. Ratios de cobertura menores conllevan un consumo de energía menor a costa de una degradación de las prestaciones.

Cuadro 6.2: Consumo de energía estática y dinámica de las configuraciones PS para 16 núcleos frente un directorio single caché 1×.

Configurations			Pleakage (mW)			Eread (pJ)		
Coverage	Directory	Private	Shared	Private	Total	Shared	Private	Total
1×	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×	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×	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

6.3. Análisis de escalabilidad

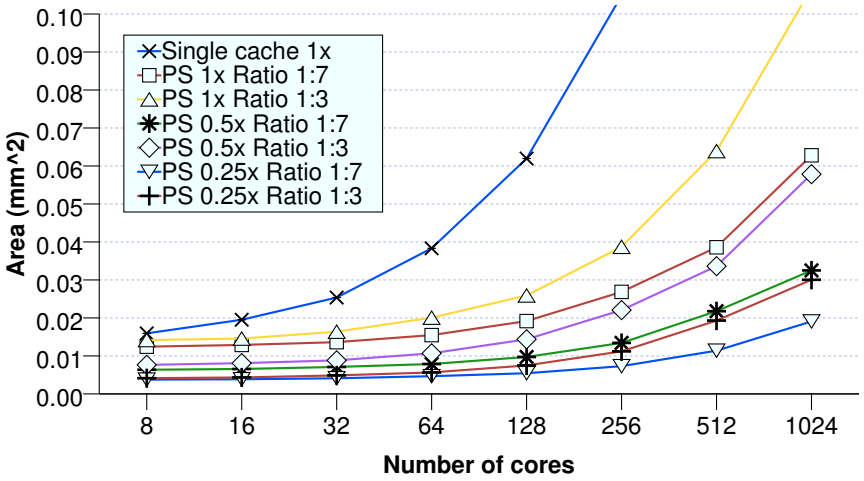
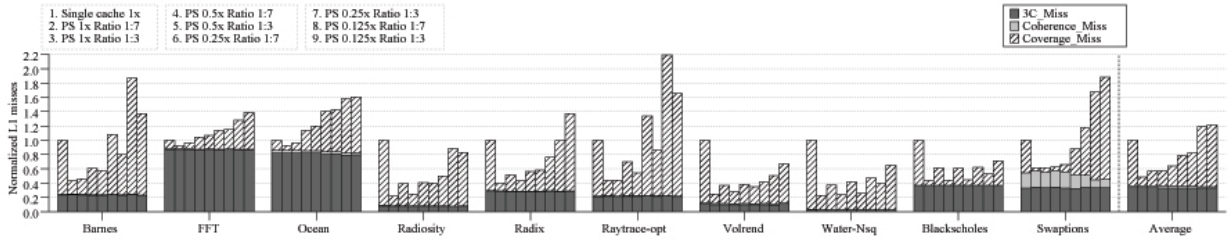
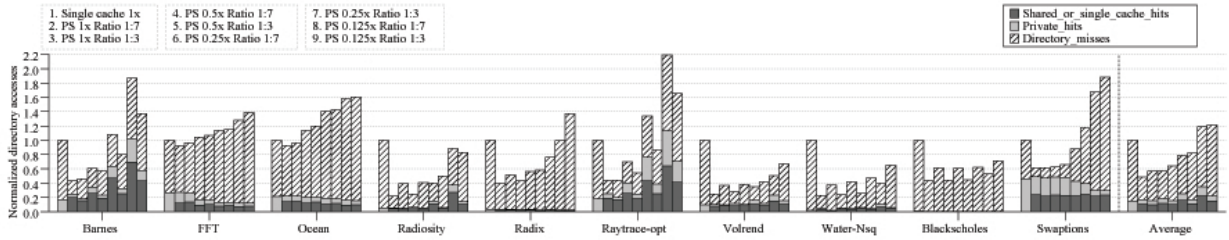


Figura 6.3: Análisis de la escalabilidad en función del área.

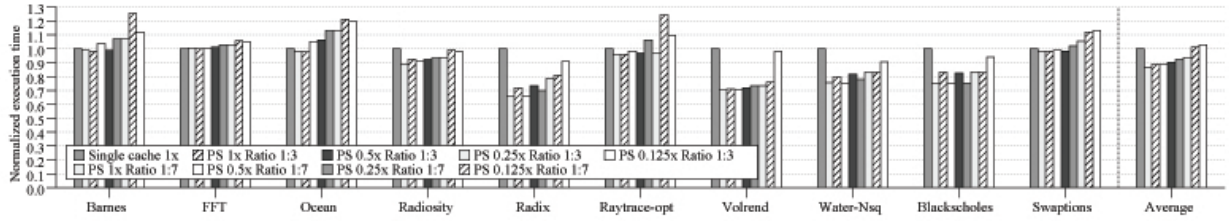
La Figura 6.3 muestra el área de silicio requerida por núcleo para las configuraciones de directorio estudiadas. Como se puede observar, el directorio convencional exhibe un comportamiento muy pobre ya que necesita para 128 núcleos más área que todas configuraciones PS, con la única excepción del PS 1x 1:3. Para una configuración de cobertura 1x, el directorio PS es capaz de reducir un 26,71 % (ratio 1:7) y 15,71 % (ratio 1:3) el área requerida por una caché de directorio convencional en un sistema con 1024 núcleos teniendo ambos el mismo número de entradas. Por supuesto, el área se reduce aún más usando ratios de cobertura menores. Particularmente, para las configuraciones 0.5x PS, el directorio PS es capaz de necesitar tan solo un 14,147 % (ratio 1:3) y 8,13 % (ratio 1:7) del área requerida por una caché de directorio individual, y para las configuraciones 0.25x PS tan solo 7,52 % (ratio 1:3) y 4.77 % (ratio 1:7).



(a) Fallos de caché L1 normalizados (caché compartida de 2 vías).



(b) Accesos al directorio normalizados (caché compartida de 2 vías).



(c) Tiempo de ejecución normalizado (caché compartida de 2 vías).

Figura 6.4: Prestaciones normalizadas respecto a una caché de directorio convencional.

6.4. Reducción del número de vias

Tal y como ya se había comentado en la sección 1.3, la caché compartida no se ve tan beneficiada por el número de vías como la caché privada. Por esa razón se volvieron a lanzar todas las cargas reduciendo a 2 el número de vías de la caché compartida, pero manteniendo el ratio de cobertura, es decir, se duplica el número de conjuntos. De esta forma se pretende comparar cuales son las diferencias en las prestaciones entre ambas configuraciones.

En la figura 6.4 se pueden observar las mismas gráficas que ya se han presentado, pero mostrando los resultados bajo esta nueva configuración. Como era de esperar la diferencia entre ambas es practicamente nula. El número de fallos de cobertura aumenta ligeramente, aunque nada excesivamente importante. El tiempo de ejecución, por tanto, también se ve ligeramente perjudicado. Con esto termina de confirmarse que se puede utilizar una estructura de menor complejidad para el diseño de la caché compartida sin

que se obtengan penalizaciones severas.

Capítulo 7

Conclusiones y trabajo futuro

7.1. 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 cuando el número de núcleos es bajo o medio. Desafortunadamente, el consumo y área en este tipo de directorios crece cuadráticamente con el número de núcleos, principalmente a causa del vector de presencia, haciendo que esta elección de diseño sea prohibitiva cuando empezamos a disponer de un número más elevado de núcleos.

En este proyecto se propone el directorio PS, que utiliza dos estructuras de caché de directorio diseñadas para satisfacer los requisitos de los bloques que almacenan: la caché Compartida, una caché pequeña y de rápido acceso centrada en almacenar los bloques compartidos, y la caché Privada, una caché considerablemente más grande centrada en almacenar los bloques privados y que no almacena el vector de presencia, dado que los bloques privados no lo necesitan. La gran cantidad de bloques privados en las aplicaciones (incluidas las cargas paralelas) es la que ha conducido a tomar la decisión de utilizar este tamaño para la caché Privada. 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 experimentales muestran que, comparado con una caché de directorio convencional con el mismo número de entradas, el directorio PS mejora las prestaciones en un 14 % para 16 núcleos a raíz 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 cobertura 0.125x) mientras que se mantienen las prestaciones de una caché de directorio convencional.

7.2. Trabajo futuro

En cuanto a trabajo futuro queda pendiente estimar el área y energía requeridos para obtener las mismas prestaciones con el directorio PS que los que tendríamos utilizando un diseño de etiquetas duplicadas como el que se explica en el capítulo 3.

Además, podemos aprovecharnos de esta clasificación en bloques privados y compartidos, en la que se ha centrado gran parte del presente trabajo, para reducir el consumo de energía en la caché de primer nivel del procesador. Añadiendo, por ejemplo, un bit adicional que especifique si un bloque se encuentra en estado privado o compartido, podemos evitar el consumo ocasionado por la comparación en un gran número de vías (ya que con ese bit hacemos un primer filtrado de las líneas que no nos interesan buscar). En el caso de los bloques compartidos la ganancia obtenida será mayor, pues su número es más reducido y, por tanto, el número de vías que será necesaria comprar también lo será. No obstante, sería necesario realizar algún tipo de protocolo de predicción de vías para poder llevar esto a cabo ya que a priori el procesador no puede saber el estado del bloque al que quiere acceder (no sin haber accedido previamente a él).

De la misma forma, podemos reducir el consumo de las acciones de coherencia inducidas por el directorio, puesto que no será necesario explorar todas las posibles vías. En este caso, además, podremos saber con certeza el tipo de bloque sobre el que se quiere realizar alguna acción de coherencia, y en consecuencia, no hace falta ningún tipo de predicción, podemos acceder directamente a las vías privadas o compartidas según sea necesario.

7.3. Publicaciones relacionadas con el proyecto

Joan J. Valls, Alberto Ros, Julio Sahuquillo and María E. Gómez. El directorio PS: Una caché de directorio multinivel escalable para CMPs. In *XXIII edición Jornadas de*

Paralelismo SARTECO, 19-21 September 2012.

Joan J. Valls, Alberto Ros, Julio Sahuquillo, María E. Gómez and José Duato. PS-Dir: A Scalable Two-Level Directory Cache. In *21st International Conference on Parallel Architectures and Compilation Techniques (PACT-2012)*, 19-23 September 2012.

Joan J. Valls, Alberto Ros, Julio Sahuquillo, María E. Gómez and José Duato. PS Directory: a Scalable Multilevel Directory Cache for CMP NUCAs. In *19th IEEE International Symposium on High-Performance Computer Architecture (HPCA-19)*, submitted, February 2013.

Bibliografía

[AGGD01] 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)*, pages 97–106, January 2001.

[AGGD05] 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)*, 16(1):67–79, January 2005.

[APJ09] Niket Agarwal, Li-Shiuan Peh, and Niraj K. Jha. In-Network Snoop Ordering (INSO): Snoopy coherence on unordered interconnects. In *15th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, pages 67–78, February 2009.

[AW06] Alaa R. Alameldeen and David A. Wood. Ipc considered harmful for multiprocessor workloads. *IEEE Micro*, 26(4):8–17, July 2006.

[BEA08] Shane Bell, Bruce Edwards, and John Amann, et al. TILE64^{TM} processor: A 64-core SoC with mesh interconnect. In *IEEE Int'l Solid-State Circuits Conference (ISSCC)*, pages 88–598, January 2008.

[BMW06] Bradford M. Beckmann, Michael R. Marty, and David A. Wood. ASR: Adaptive selective replication for CMP caches. In *39th IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, pages 443–454, December 2006.

[Che93] 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)*, pages 341–352, June 1993.

[CMR+06] Liqun Cheng, Naveen Muralimanohar, Karthik Ramani, Rajeev Balasubramonian, and John B. Carter. Interconnect-aware coherence protocols

for chip multiprocessors. In *33rd Int'l Symp. on Computer Architecture (ISCA)*, pages 339–351, June 2006.

[CRG+11] 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)*, pages 93–103, June 2011.

[CS06] Jichuan Chang and Gurindar S. Sohi. Cooperative caching for chip multiprocessors. In *33rd Int'l Symp. on Computer Architecture (ISCA)*, pages 264–276, June 2006.

[CSL+06] Jason F. Cantin, James E. Smith, Mikko H. Lipasti, Andreas Moshovos, and Babak Falsafi. Coarse-grain coherence tracking: Regionscout and region coherence arrays. *IEEE Micro*, 26(1):70–79, January 2006.

[EJPL08] Natalie D. Enright Jerger, Li-Shiuan Peh, and Mikko H. Lipasti. Virtual tree coherence: Leveraging regions and in-network multicast trees for scalable cache coherence. In *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 41, pages 35–46, Washington, DC, USA, 2008. IEEE Computer Society.

[FLKBF11] M. Ferdman, P. Lotfi-Kamran, K. Balet, and B. Falsafi. Cuckoo directory: A scalable directory for many-core systems. pages 169 –180, feb. 2011.

[GWM90] 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)*, pages 312–321, August 1990.

[HFFA09] 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)*, pages 184–195, June 2009.

[HKS+05] Jaehyuk Huh, Changkyu Kim, Hazim Shafi, Lixin Zhang, Doug Burger, and Stephen W. Keckler. A nuca substrate for flexible cmp cache sharing. In *Proceedings of the 19th annual international conference on Supercomputing*, ICS '05, pages 31–40, New York, NY, USA, 2005. ACM.

[KBK02] Changkyu Kim, Doug Burger, and Stephen W. Keckler. An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches. In *10th Int'l Conf. on Architectural Support for Programming Language and Operating Systems (ASPLOS)*, pages 211–222, October 2002.

[KJLP10] John H. Kelm, Matthew R. Johnson, Steven S. Lumettta, and Sanjay J. Patel. Waypoint: scaling coherence to thousand-core architectures. In *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, PACT '10, pages 99–110, New York, NY, USA, 2010. ACM.

[KMP+10] George Kurian, Jason E. Miller, James Psota, Jonathan Eastep, Jifeng Liu, Jurgen Michel, Lionel C. Kimerling, and Anant Agarwal. Atac: a 1000-core cache-coherent processor with on-chip optical network. In *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, PACT '10, pages 477–488, New York, NY, USA, 2010. ACM.

[KSSF10] Ron Kalla, Balaram Sinharoy, William J. Starke, and Michael Floyd. Power7: IBM's Next-Generation Server Processor. *IEEE Micro*, 30:7–15, 2010.

[Mar03] Milo M.K. Martin. *Token Coherence*. PhD thesis, University of Wisconsin-Madison, December 2003.

[MBJ09] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P. Jouppi. Cacti 6.0. Technical Report HPL-2009-85, HP Labs, April 2009.

[MCE02] Peter S. Magnusson, Magnus Christensson, and Jesper Eskilson, et al. Simics: A full system simulation platform. *IEEE Computer*, 35(2):50–58, February 2002.

[MH08] Michael R. Marty and Mark D. Hill. Virtual hierarchies. *IEEE Micro*, 28(1):99–109, 2008.

[MHS+03] Milo M.K. Martin, Pacia J. Harper, Daniel J. Sorin, Mark D. Hill, and David A. Wood. Using destination-set prediction to improve the latency/-bandwidth tradeoff in shared-memory multiprocessors. In *30th Int'l Symp. on Computer Architecture (ISCA)*, pages 206–217, June 2003.

[MS05] Richard E. Matick and Stanley E. Schuster. Logic-based eDRAM: Origins and rationale for use. *IBM Journal of Research and Development*, 49(1):145–165, 2005.

[MSA00] Milo M.K. Martin, Daniel J. Sorin, and Anatassia Ailamaki, et al. Timestamp snooping: An approach for extending SMPs. In *9th Int'l Conf. on Architectural Support for Programming Language and Operating Systems (ASPLOS)*, pages 25–36, November 2000.

[MSB05] 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*, 33(4):92–99, September 2005.

[ON90] 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)*, pages 138–147, June 1990.

[RAG10] Alberto Ros, Manuel E. Acacio, and José M. García. A scalable organization for distributed directories. *Journal of Systems Architecture (JSA)*, 56(2-3):77–87, February 2010.

[SBB07] 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*, pages 22–25, November 2007.

[SK12] 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)*, pages 129–140, February 2012.

[SKT+05] B. Sinharoy, R N. Kalla, J M. Tendler, R J. Eickemeyer, and J B. Joyner. POWER5 System Microarchitecture. *IBM Journal of Research and Development*, 49(4/5):505–521, 2005.

[TDF+02] J M. Tendler, J S. Dodson, J S. Fields, H. Le, and B. Sinharoy. POWER4 System Microarchitecture. *IBM Journal of Research and Development*, 46(1):5–25, 2002.

[VSP+09] 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*, pages 213–221, New York, NY, USA, 2009. ACM.

[WLZ+09] 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*, pages 34–45, New York, NY, USA, 2009. ACM.

[ZA05] Michael Zhang and Krste Asanović. Victim replication: Maximizing capacity while hiding wire delay in tiled chip multiprocessors. In *32nd Int'l Symp. on Computer Architecture (ISCA)*, pages 336–345, June 2005.

[ZSQM09] Jason Zebchuk, Vijayalakshmi Srinivasan, Moinuddin K. Qureshi, and Andreas Moshovos. A tagless coherence directory. In *42nd IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, pages 423–434, December 2009.