Diseño y evaluación de una arquitectura de directorio ligero para multiprocesadores de memoria compartida escalables

Alberto Ros, Manuel E. Acacio y José M. García

Departamento de Ingeniería y Tecnología de Computadores

Facultad de Informática

Universidad de Murcia

30071 Murcia

{a.ros,meacacio,jmgarcia}@ditec.um.es

Resumen

Uno de los problemas más importantes de las arquitecturas cc-NUMA es que la cantidad de memoria necesaria para mantener la coherencia de los bloques en cache (información de directorio) puede llegar a ser excesiva, y lo que es peor, aumentar linealmente con el número de procesadores del sistema. Desafortunadamente, las soluciones actuales reducen el tamaño de esta memoria a cambio de degradar el rendimiento. Este artículo presenta el *directorio ligero*, un nuevo enfoque a la hora de organizar la información de directorio que reduce considerablemente la sobrecarga de memoria y mejora el rendimiento en los sistemas cc-NUMA. Esta arquitectura elimina completamente la estructura de directorio de la memoria principal. Ahora la información de directorio reside en cache, que posee muchas menos entradas que la memoria principal, con lo que se logra una considerable reducción en tamaño, además de un acceso más rápido a dicha información de directorio.

1. Introdución y motivación

En la actualidad, la mejor opción para diseñar multiprocesadores escalables de memoria compartida son las llamadas arquitecturas cc-NUMA. Estas arquitecturas se basan en una red de interconexión punto a punto [7] y la memoria principal físicamente distribuida, para garantizar que el acceso a la memoria escale según el número de procesadores. Además, necesitan una estructura de directorio para mantener la coherencia de los bloques en cache [4].

La forma convencional de organizar la información de directorio consiste en tener una entrada de directorio para cada bloque de la memoria principal. En cada entrada se especifica qué nodos cachean dicho bloque (código de compartición) y cuál es el estado de dicho bloque (bits de estado). Debido al uso de la estructura de directorio, nos aparecen otros dos problemas adicionales que afectan a la escalabilidad de esta arquitectura. El primero de ellos tiene que ver con la sobrecarga de memoria que representa la estructura de directorio, ya que el código de compartición suele usar un bit para cada nodo (*full-map* o *bit-vector* [4]), con lo que aumentará linealmente según el número de nodos del sistema. El otro problema consiste en que al estar el directorio en la memoria principal, las latencias de los fallos de cache son muy elevadas.

En este artículo proponemos la arquitectura de *directorio ligero*, que toma las ventajas de la integración dentro del chip del procesador para diseñar un multiprocesador cc-NUMA escalable. En nuestra propuesta, la información de directorio se asocia únicamente a bloques que estén en alguna cache, obteniendo a la vez una menor necesidad de memoria y un acceso más rápido a la información de directorio.

Esta propuesta está motivada por la observación de que sólo un pequeño conjunto de bloques necesitan ser almacenados en cache en un momento dado (localidad temporal). Por tanto, cuando un nodo acceda a un bloque remoto, es decir, perteneciente a la zona de memoria de otro nodo, es muchas ocasiones, éste ya habrá sido accedido por el nodo local o será accedido por él en breve.

Tal y como hace el protocolo de directorio convencional cuando se produce un fallo de cache, dicho fallo se envía al nodo *home* del bloque con el fin de obtener la información de directorio. En el caso del protocolo de directorio ligero, en la primera referencia (local o remota) al bloque se almacena una entrada en la cache del nodo *home* para mantener la información de directorio a ese nivel. De este modo, las sucesivas referencias a dicho bloque encontrarán la información necesaria en la cache del nodo *home*, con lo que no es necesario acceder a memoria principal, reduciendo así la latencia del fallo.

Sin embargo, el hecho de reservar en la cache entradas con información de directorio puede provocar el reemplazo de un bloque de la cache que podría estar siendo accedido y, por tanto, un aumento en la tasa de fallos de cache. Afortunadamente, la localidad temporal que exhiben la mayoría de las aplicaciones hace que la tasa de fallos se mantenga. En la figura 1 se muestra la relación existente entre el número de los bloques almacenados (suponiendo caches ilimitadas) en la cache más sobrecargada para el protocolo convencional y para el protocolo de directorio ligero. Se puede apreciar que este impacto es mínimo, siendo la aplicación `OCEAN` la que contiene el nodo más perjudicado con tan sólo un 34 % de incremento en el número de bloques que serían usados en cache.

Nuestra propuesta, por tanto, aporta dos beneficios fundamentales. En primer lugar, la sobrecarga debida a la información de directorio es reducida en un factor de 1024 o más comparada con el modelo de directorio convencional. En segundo lugar, se reduce el acceso a la información de directorio, obteniendo mejoras de hasta un 20 % en el tiempo de ejecución

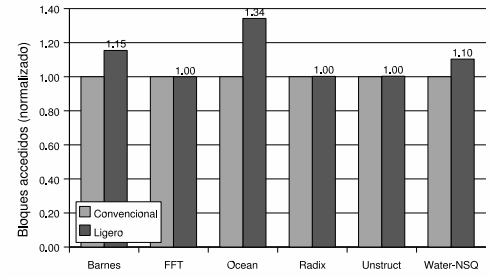

Figura 1: Bloques accedidos por la cache más sobrecargada.

de las aplicaciones.

El resto del artículo se organiza del siguiente modo. La sección 2 ofrece una descripción acerca de los esquemas de reducción del tamaño de directorio existentes. Las secciones 3 y 4 describen la arquitectura diseñada y el protocolo adecuado para ella respectivamente. La sección 5 introduce la metodología empleada en la evaluaciones. La sección 6 muestra los resultados de nuestra propuesta. Finalmente, la sección 7 muestra las conclusiones de este trabajo y sus futuras vías de investigación.

2. Trabajo relacionado

Uno de los principales problemas de los sistemas cc-NUMA es la sobrecarga de memoria que introduce la estructura de directorio. Dicha sobrecarga podría llegar a representar un coste adicional del 100 % de la memoria que necesitaríamos para poder ejecutar las aplicaciones deseadas, dependiendo del número de nodos del sistema y código de compartición elegido [1].

A la hora de reducir el tamaño del directorio se han planteado tradicionalmente dos alternativas hardware: reducción en *anchura* y reducción en *altura* [6].

La forma para reducir el ancho de la estructura del directorio es usar otra forma de codificación más comprimida para el código de compartición distinta a la de *full-map*. Por ejemplo, *Coarse vector* [8] es el esquema usado en el multiprocesador SGI Origin 2000/3000 [11] y se basa en el uso de un único bit del código

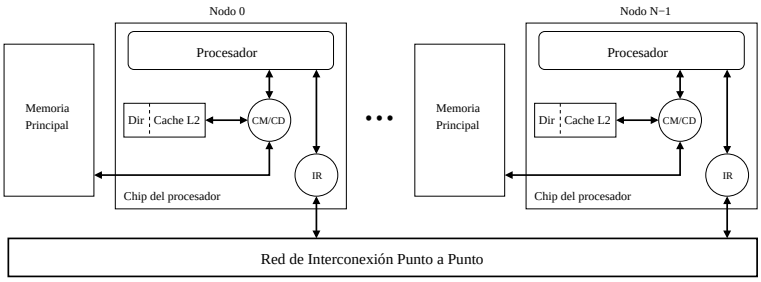

Figura 2: Organización del multiprocesador basado en el directorio ligero.

de compartición para designar a un conjunto de K nodos del sistema. *Tristate* [2] (también llamado *superset scheme*) almacena una palabra de d dígitos donde cada dígito o toma uno de entre tres valores: 0, 1 o *ambos*, denotando todos los compartidores cuyos identificadores coinciden asignando a *ambos* los valores 0 ó 1. *Gray-tristate* [12] mejora *tristate* en algunos casos usando el código *Gray* para numerar los nodos. Por último, una codificación basada en el concepto de agrupamiento multicapa en árboles binarios es introducido en [1] siendo su más elaborada propuesta *binary tree with subtrees* que usa dos árboles binarios para incluir a todos los compartidores. Uno de ellos es calculado desde el nodo *home* y el otro desde un nodo simétrico.

Otros autores proponen reducir el ancho de las entradas del directorio, poniendo en hardware tan sólo un número limitado de punteros (elegido para que cubra el caso más común) a cada bloque de cache y manejando (habitualmente por software) las situaciones en que se produce un desbordamiento en dicha estructura [5].

Una forma para reducir el alto del directorio, es decir, el número de entradas del mismo, consiste en combinar varias entradas de directorio en una sola [16]. Organizar la estructura de directorio como una cache [14] es otra estrategia para evitar que los directorios sean muy dispersos y estén habitualmente vacíos. En [13] se habla de una cache de directorios que sólo almacena las entradas de un bloque si algún nodo remoto posee una copia de dicho bloque.

3. El directorio ligero

La diferencia fundamental entre una arquitectura de directorio convencional y la arquitectura de directorio ligero es la eliminación del directorio de la memoria principal. Ahora esta información se guardará a nivel de la cache L2, con lo que se obtienen dos ventajas importantes. La primera ventaja es una importante reducción en memoria para la información del directorio. La segunda ventaja consiste en un acceso más rápido al directorio, por estar en cache.

Sin embargo, este rápido acceso a la información de directorio en la cache de segundo nivel no sería posible si el controlador de directorio estuviera fuera del chip del procesador. Afortunadamente, los avances tecnológicos han permitido integrar algunos componentes clave del sistema como el controlador de memoria, el *hardware* de coherencia y el interfaz de red dentro de este chip. Este es el caso del *Compaq Alpha 21364 EV7* [9] y del *AMD Hammer* [3]. Por tanto, en este trabajo asumimos que dichos componentes están dentro del chip del procesador, junto con la cache L2.

La figura 2 muestra la organización propuesta para un multiprocesador de N nodos. Los nodos del sistema están conectados mediante la red de interconexión punto a punto a través del interfaz de red (IR). La función del controlador de memoria y directorio (CM/CD) es asistir a los fallos de cache. Cuando la cache de segundo nivel no puede satisfacer una petición del procesador, el controlador de directorio se encarga de averiguar el nodo *home* del

bloque pedido para enviarle la solicitud de dicho bloque. Una vez en el nodo *home*, se busca una entrada en la cache con la información de directorio necesaria. Si la encuentra ya puede resolver la petición sin necesidad de acceder a memoria principal. Si no la encuentra quiere decir que el bloque no reside en la cache de ningún nodo y, por tanto, es necesario acceder a memoria principal.

Cada bloque de la cache L2 se encuentra dividido en la parte de los *tags* (que contiene, entre otras cosas, la información de directorio) y la de datos. El acceso a ambas partes de la cache se hace en paralelo, de tal modo que al acceder a la cache se conoce la información de directorio y el dato del bloque, si el nodo *home* mantenía una copia. La parte de los *tags* está formada por cuatro campos fundamentales: la etiqueta del bloque, el estado en cache, el estado del directorio y el código de compartición.

El campo del estado en cache puede tomar uno de los cuatro valores del protocolo MESI [6], teniendo un significado adicional en el caso del estado *I* (inválido). En el protocolo MESI la ocupación de dicha entrada en cache en estado *I* por un nuevo bloque no supone un reemplazo. En el protocolo de directorio ligero, puede ser que este bloque posea información de directorio (aunque no datos). En ese caso, el bloque deberá ser reemplazado y todas las copias de dicho bloque invalidadas para mantener la coherencia (ver la sección 4 para más detalles). El campo de estado de directorio puede tomar dos valores (un bit):

- S: *Shared* o compartido, el bloque se encuentra compartido en varias caches, todas ellas con una copia válida. La cache del nodo *home* de ese bloque será la encargada de proporcionarlo, ya que tendrá una copia del bloque aunque no lo haya necesitado.

- O: *Owned* o propietario, el bloque se encuentra en una y sólo una cache y ha podido ser modificado por ésta. La única copia válida la tendrá dicha cache y es ella la que proporcionará el bloque. El nodo *home* siempre tendrá una entrada en su cache para ese bloque (en estado I si no es el propietario).

También existe un estado implícito U (*Uncached*), que se produce cuando el bloque se encuentra únicamente en memoria principal, bien porque no ha sido solicitado por ningún nodo, o porque fue reemplazado por otro bloque.

El código de compartición almacena la identidad de los nodos que mantienen una copia del bloque en sus caches. Aunque la arquitectura de directorio ligero es compatible con cualquier tipo de código de compartición (por ejemplo, los códigos comprimidos), por simplicidad hemos optado por usar el código *full-map*. Este campo constará de un bit por cada nodo del sistema, y estará activado si dicho nodo tiene una copia del bloque.

4. Protocolo de coherencia

La arquitectura descrita anteriormente requiere un protocolo de coherencia de cache similar al protocolo MESI con algunas pequeñas modificaciones. Estas modificaciones se realizan para garantizar que para todos los bloques que estén en alguna cache, su información de directorio debe estar presente en la cache L2 del nodo *home*. Además, cuando un bloque de memoria sea reemplazado de dicha cache, todas las copias del bloque deben ser previamente invalidadas para asegurar la coherencia.

Usaremos el término de fallo local para referirnos a los fallos de L2 que ocurren en el nodo *home*, y el término fallo remoto para el caso contrario. Para fallos locales, el protocolo accede a la información de directorio (parte de los *tags*) y realiza las mismas acciones que el protocolo convencional.

Por otro lado, los fallos remotos son enviados al nodo *home*, donde el controlador de directorio comprueba si la información necesaria para resolver el fallo se encuentra en cache. Si no la encuentra, estamos ante el estado implícito *uncached*. En ese caso, se debe acceder al bloque en memoria principal y enviarlo al peticionario, además de reservar una entrada en la cache del nodo *home* para la información de directorio.

		Estado de Directorio		
		Uncached	**Shared**	**Owned**
Convencional	**Inf. Dir.**	Memoria	Memoria	Memoria
	Datos	Memoria	Memoria	Cache propietaria
Ligero	**Inf. Dir.**	-	Cache *home*	Cache *home*
	Datos	Memoria	Cache *home*	Cache propietaria

Cuadro 1: Dónde se encuentra la información de directorio y los datos cuando se produce un fallo de L2 tanto en el protocolo convencional como en el ligero.

Si el fallo remoto encuentra la información de directorio en la cache del *home* no será necesario acceder a memoria principal. Además, si el estado del directorio es *shared*, dicha cache también posee una copia del bloque y, por tanto, podrá enviársela inmediatamente al peticionario.

En cuanto a los reemplazos, el protocolo se diferencia del MESI sólo en los que se producen en la cache del nodo *home*. Si el estado del bloque reemplazado es *shared*, el controlador de directorio debe buscar la información de los nodos que tienen copia del bloque y enviarles mensajes de invalidación. Cuando todos los nodos respondan confirmando las invalidaciones se procederá al reemplazo. Si el estado del bloque reemplazado es *owned* y la única copia la posee un nodo remoto, también se debe invalidar la copia para después reemplazarlo y actualizar la memoria principal.

El resto de los casos son manejados tal y como ya lo hace el protocolo de directorio convencional. El cuadro 1 resume las ventajas de nuestra propuesta. El directorio ligero evita ir a memoria principal a buscar la información de directorio cuando el estado es *shared* o *owned*. Además, si el estado es *shared* es la cache del nodo *home* la que sirve el bloque. Por último, no necesitamos información de directorio para bloques en estado *uncached* con lo que reducimos considerablemente la memoria necesaria para mantener la coherencia.

5. Entorno de evaluación

Hemos usado una versión modificada del simulador guiado por la ejecución RSIM (*Rice Simulator for ILP Multiprocessors*) [10]. Hemos simulado un sistema cc-NUMA de 32 nodos que implementa el protocolo de directorio ligero. El cuadro 2 muestra los parámetros usados para evaluar la arquitectura propuesta. En las simulaciones se modelan la contención en los accesos a la parte de los *tags* y los datos en las caches para peticiones remotas. Por último, comentar que hemos utilizado una versión optimizada de la consistencia secuencial con ejecución de cargas especulativas.

Sistema de 32 Nodos - Protocolo Ligero	
Parámetros del procesador ILP	
Velocidad del procesador	1 GHz
Tasa máxima *fetch/retire*	4
Ventana de instrucciones	128
Predictor de saltos:	2 bit agree
* Contadores	2048
* *Shadow mappers*	8
Parámetros de la cache	
Tamaño de la línea de cache	64 bytes
Cache L1 dividida en I & D:	*write-through*
* Tamaño, asociatividad	16KB, mapeo directo
* Tiempo de acierto	2 ciclos
Cache L2 unificada:	*write-back*
* Tamaño, asociatividad	64KB, 4-vías
* Tiempo de acierto	15 ciclos (6 + 9)
Parámetros del directorio	
Controlador de directorio	1 ciclo (on-chip)
Información de directorio	6 ciclos (*tag* L2)
Creación de paquetes:	
* Primer mensaje	4 ciclos
* Siguientes mensajes	2 ciclos
Parámetros de memoria	
Tiempo de acceso a memoria	80 ciclos
Intercalado de memoria	4 vías
Parámetros del Bus	
Ancho de banda	8 bytes
Ciclos de bus	1 ciclo
Parámetros de Red	
Topología	Malla bidimensional
Tamaño del *flit*	8 bytes
Tamaño de mensaje sin datos	2 *flits*
Velocidad del *router*	250 MHz
Retardo de Arbitraje	4 ciclos del *router*
Velocidad del canal	500 MHz
Ancho del canal	32 bits

Cuadro 2: Parámetros básicos del sistema.

Las aplicaciones utilizadas en las simulaciones cubren una amplia gama de patrones de comunicación y computación. `Barnes` (8192 cuerpos, 4 pasos), `FFT` (256K complejos),

`Ocean` (258x258 celdas), `Radix` (1M claves, radio 1024), y `Water-NSQ` (512 moléculas, 4 pasos) son del conjunto de aplicaciones del SPLASH-2 [17] y `Unstructured` (malla 2K, 5 pasos) es una aplicación de computación de fluidos dinámicos. Todos los resultados mostrados en este artículo corresponden a la fase paralela de las aplicaciones. Los tamaños de problema de las aplicaciones han sido escogidos de tal modo que los conjuntos de trabajo sean mayores que la capacidad de las caches.

6. Resultados y análisis

En esta sección comparamos la memoria necesaria para almacenar la información de directorio tanto para la arquitectura propuesta como para otros esquemas de compresión del código de compartición. Además, mostramos la mejora en tiempo de ejecución obtenida con respecto a la organización convencional.

6.1. Reducción en memoria

Existen dos factores que impiden la escalabilidad del directorio. En primer lugar, si se usa *full-map* el número de bits usados para el código de compartición tiende a incrementar linealmente conforme aumenta el número de nodos del sistema. En segundo lugar, se necesitan tantas entradas de directorio como bloques en memoria principal tiene el sistema.

El objetivo de este artículo es mejorar la escalabilidad del directorio reduciendo el número de entradas. Sin embargo, la organización propuesta se puede combinar con técnicas de reducción del código de compartición. La figura 3 muestra la evolución de la sobrecarga de memoria en función del número de nodos. Mostramos sólo la parte correspondiente a la sobrecarga de hasta un 10 % para poder distinguir bien las diferencias entre los tres últimos esquemas. Hemos considerado un multiprocesador con $2GB$ de memoria principal, $2MB$ de cache L2 y tamaño de bloque 128 bytes, para los esquemas *full-map*, *coarse vector* (asumiendo $K = 4$), *gray-tristate*, *binary tree with subtrees* (asumiendo 4 nodos simétricos) y nuestra propuesta.

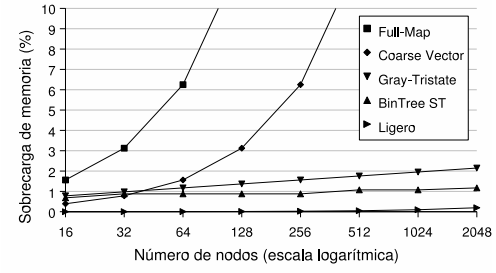

Figura 3: Sobrecarga de memoria de directorio según los nodos del sistema.

El esquema *full-map* tiene una escalabilidad limitada y para un sistema de 1024 nodos el tamaño del código de compartición iguala al de un bloque de memoria (sobrecarga del 100 %). *Coarse vector* reduce la sobrecarga de memoria, pero no soluciona el problema de escalabilidad. *Gray-tristate* y *binary tree ST* presentan una mejor escalabilidad aumentando de forma logarítmica con el número de nodos. Finalmente, el directorio ligero logra una mayor reducción que los esquemas anteriores y escala perfectamente para un sistema de 2048 nodos. La mejora obtenida dependerá de la relación de tamaño entre cache y memoria principal, por lo que nuestra propuesta reducirá aún más la sobrecarga en las futuras arquitecturas cc-NUMA.

6.2. Mejora en tiempo de ejecución

El directorio ligero obtiene también beneficios en el tiempo de ejecución debido a la menor latencia de acceso a la información de directorio pero, por otro lado, puede acarrear un mayor número de reemplazos degradando así el rendimiento. La figura 4 muestra el tiempo de ejecución para una arquitectura convencional (similar al *SGI Origin* 2000/3000 [11]), para una arquitectura de directorio ligero ideal, en la que sólo se tiene en cuenta el beneficio del menor tiempo de acceso al directorio, y para la arquitectura de directorio ligero real. Se observa que las mejoras del caso ideal van del 6 % al 29 %. Por otro lado, el caso real obtiene mejoras con respecto al caso

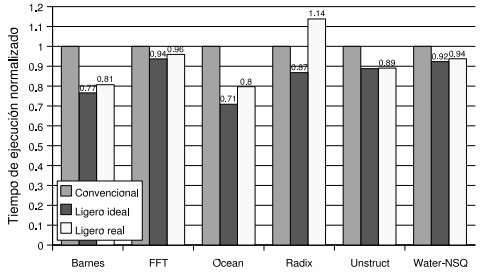

Figura 4: Tiempo de ejecución normalizado para el directorio convencional, ligero ideal y ligero real.

	Conv.	Ligero		
Aplicación	Total	Total	Cache	Memoria
Barnes	0.16	0.17	0.15	0.02
FFT	0.04	0.04	0.02	0.02
Ocean	0.16	0.16	0.04	0.12
Radix	0.11	0.13	0.01	0.12
Unstruct	0.38	0.39	0.38	0.01
Water-NSQ	0.20	0.20	0.08	0.12

Cuadro 3: Tasa de fallos para ambos protocolos, especificando para el ligero cuando el fallo es resuelto en cache.

convencional para todas las aplicaciones salvo `Radix` (14 % de degradación). Particularmente, `Ocean` y `Barnes` obtienen las mejores prestaciones, con una reducción del 20 % y del 19 % respectivamente, mientras que para las otras aplicaciones la reducción oscila entre 4 % y 11 %.

El cuadro 3 ayuda a comprender las diferencias entre el caso ideal y el real. Nuestra propuesta no afecta prácticamente al total de los fallos de cache, como suponíamos en la introducción, ya que no se produce un incremento significativo de los reemplazos en cache. La aplicación `Unstructured` es la que más se asemeja al caso ideal, debido a que logra resolver casi todos los fallos sin acceder a memoria principal. En cambio, el rendimiento de `Ocean` se aleja del ideal debido a que resuelve tres de cada cuatro fallos de cache en memoria. El echo de mantener la tasa de fallos total constante se traduce en la reducción del 20 % con respecto a la arquitectura convencional. Finalmente, en `Radix` aumenta la tasa de fallos y la mayoría tienen que ir a memoria a buscar la información de directorio. Esto hace que el caso real obtenga mucho peor rendimiento que el ideal, y lo que es peor, la tasa de accesos a memoria es mayor que en el caso convencional, por lo que degrada su rendimiento con respecto a dicha arquitectura. En la figura 1 se observaba que, en el caso de `Radix`, la mayoría de los bloques almacenados en cache para mantener la información de directorio habían sido accedidos antes por el nodo *home* o serán accedidos después. El problema por tanto radica en el orden en el que son almacenados, lo cual se podría solucionar con un algoritmo de almacenamiento o reemplazo en cache distinto al usado. Una visión más detallada de los resultados puede verse en [15].

7. Conclusiones y trabajo futuro

En este artículo hemos presentado la arquitectura de directorio ligero como un protocolo de directorios escalable que elimina el directorio de la memoria principal. Hemos introducido el controlador y la información de directorio en el chip del procesador. De este modo, los fallos de cache evitan ir a memoria principal y son resueltos con menor latencia.

Hemos descrito la arquitectura y el protocolo de coherencia adecuado a las particularidades del diseño propuesto. Hemos simulado un conjunto de aplicaciones científicas con el simulador RSIM adaptado a nuestra arquitectura, obteniendo resultados de hasta el 20 % de mejora en tiempo de ejecución. Además, hemos reducido el tamaño de la información de directorio de tal modo que nuestra propuesta escala perfectamente para un multiprocesador de 2048 nodos.

Como futuras vías de investigación proponemos estudiar la política de almacenamiento de bloques en la cache L2, de tal modo que podamos decidir si es conveniente o no almacenar el bloque en la cache. Además, podríamos estudiar el impacto de tener caches separadas para bloques con y sin información de directorio. De esta forma, una de las caches necesitaría menos memoria. Finalmente, para reducir más la sobrecarga del directorio pensamos evaluar el uso de punteros limitados o códigos de compartición comprimidos en lugar de *full-map*.

Referencias

[1] M.E. Acacio, J. González, J.M. García, and J. Duato. "A Two-Level Directory Architecture for Highly Scalable cc-NUMA Multiprocessors". *IEEE Transactions on Parallel and Distributed Systems*, 16(1):67–79, January 2005.

[2] A. Agarwal, R. Simoni, J. Hennessy, and M. Horowitz. "An Evaluation of Directory Schemes for Cache Coherence". *Proc. of the 15th Int'l. Symposium on Computer Architecture (ISCA'88)*, pages 280–289, May 1988.

[3] A. Ahmed, P. Conway, B. Hughes, and F. Weber. "AMD Opteron^{TM} Shared Memory MP Systems". *Proc. 14th HotChips Symposium*, August 2002.

[4] L. Censier and P. Feautrier. "A New Solution to Coherence Problems in Multicache Systems". *IEEE Transactions on Computers*, 27(12):1112–1118, December 1978.

[5] D. Chaiken, J. Kubiatowicz, and A. Agarwal. "LimitLESS Directories: A Scalable Cache Coherence Scheme". *Proc. of Int'l Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS IV)*, pages 224–234, April 1991.

[6] D.E. Culler, J.P. Singh, and A. Gupta. *"Parallel Computer Architecture: A Hardware/Software Approach"*. Morgan Kaufmann Publishers, Inc., 1999.

[7] J. Duato, S. Yalamanchili, and L. Ni. *"Interconnection Networks: An Engineering Approach"*. Morgan Kaufmann Publishers, Inc., 2002.

[8] A. Gupta, W. Weber, and T. Mowry. "Reducing Memory Traffic Requirements for Scalable Directory-Based Cache Coherence Schemes". *Proc. Int'l Conference on Parallel Processing (ICPP'90)*, pages 312–321, August 1990.

[9] L. Gwennap. "Alpha 21364 to Ease Memory Bottleneck". *Microprocessor Report*, 12(14):12–15, October 1998.

[10] C. Hughes, V. Pai, P. Ranganathan, and S. Adve. "RSIM: Simulating Shared-Memory Multiprocessors with ILP Processors". *IEEE Computer*, 35(2), February 2002.

[11] J. Laudon and D. Lenosky. "The SGI Origin: A cc-NUMA Highly Scalable Server". *Proc. of the 24th Int'l Symposium on Computer Architecture (ISCA'97)*, pages 241–251, June 1997.

[12] S.S. Mukherjee and M.D. Hill. "An Evaluation of Directory Protocols for Medium-Scale Shared-Memory Multiprocessors". *Proc. of the 8th Int'l Conference on Supercomputing (ICS'94)*, pages 64–74, July 1994.

[13] A.K. Nanda, A. Nguyen, M.M. Michael, and D.J. Joseph. "High-Throughout Coherence Control and Hardware Messaging in Everest". *IBM Journal of Research and Development*, 45(2):229–244, March 2001.

[14] B. O'Krafka and A. Newton. "An Empirical Evaluation of Two Memory-Efficient Directory Methods". *Proc. of the 17th Int. Symposium on Computer Architecture (ISCA'90)*, pages 138–147, June 1990.

[15] A. Ros, M.E. Acacio, and J.M. García. "A Novel Lightweight Directory Architecture for Scalable Shared-Memory Multiprocessors". *To appear in Proc. of Euro-Par*, August 2005.

[16] R. Simoni. *"Cache Coherence Directories for Scalable Multiprocessors"*. PhD thesis, Stanford University, 1992.

[17] S.C. Woo, M. Ohara, E. Torrie, J.P. Singh, and A. Gupta. "The SPLASH-2 Programs: Characterization and Methodological Considerations". *Proc. of the 22nd Int'l Symposium on Computer Architecture (ISCA'95)*, pages 24–36, June 1995.