Arquitectura de los servidores escalables*

José M. García y Manuel E. Acacio

Departamento de Ingeniería y Tecnología de Computadores
Facultad de Informática. Universidad de Murcia
Campus de Espinardo - 30080 Murcia (SPAIN)
email: {jmgarcia,meacacio}@ditec.um.es

Resumen En este trabajo presentamos los desarrollos que hemos realizado para tratar de solucionar dos importantes problemas que aparecen en el contexto de los sistemas multiprocesadores escalables de memoria compartida. Es decir, aquellos sistemas basados en una red de interconexión punto-a-punto y un protocolo de coherencia basado en directorio. Estos problemas son la sobrecarga hardware que conlleva el uso del directorio y las grandes latencias en los accesos remotos a memoria. Se ha propuesto una técnica para reducir dicha sobrecarga de memoria debida al directorio a partir del concepto de *agrupamiento multicapa*, proponiendo también una nueva arquitectura para el directorio, que hemos denominado *directorio de dos niveles*, a través de la cual se consigue reducir drásticamente la sobrecarga de memoria a la misma vez que se mantiene el rendimiento proporcionado por un directorio *full-map* no escalable. En segundo lugar, proponemos una clasificación de los fallos de L2 en términos de las acciones que el directorio realiza para servir cada fallo, y presentamos una técnica para reducir la latencia de todos los tipos de fallos de L2 haciendo uso de la posibilidad que hoy día existe de integrar componentes clave dentro del procesador, añadiendo a la arquitectura de directorio de dos niveles un tercer nivel que integramos, junto con una cache para datos compartidos, dentro de chip del procesador.

1. Introducción

La mayoría de los computadores que utilizamos en el día a día (como el PC de casa o de la oficina) disponen de una única CPU. Estas máquinas, a menudo llamadas *máquinas monoprocesador*, ofrecen suficiente capacidad de cálculo para las tareas que realizamos cotidianamente la mayoría de los usuarios. Existen, además, ciertas aplicaciones que tienen necesidad de una mayor potencia de cálculo que la que puede ofrecer un único procesador. Así sucede, por ejemplo, en algunas aplicaciones científicas (predicción meteorológica, plegamiento de proteínas, etc.), comerciales (procesamiento de transacciones en línea, manejo de grandes bases de datos, etc.) o aplicaciones multimedia (simuladores de realidad virtual, etc.), donde se necesita más capacidad de cómputo que la ofrecida por

* Este trabajo ha sido subvencionado en parte por el Ministerio de Ciencia y Tecnología por medio de la ayuda TIC2003-08154-C06-03.

un único procesador. Otro ejemplo son los grandes motores de búsqueda de Internet (como *Google*) que deben ser capaces de satisfacer las peticiones que un gran número de usuarios están lanzando simultáneamente.

Hoy en día, la única alternativa viable para obtener más rendimiento del ofrecido por un único procesador consiste en conectar varios microprocesadores de forma que puedan cooperar en la resolución de la misma tarea. Estas máquinas, conocidas como *multiprocesadores* (ó máquinas de tipo MIMD según la conocida clasificación de Flynn [16]), pueden disponer de unos pocos microprocesadores (como los pequeños SMP con procesadores IA-32, cada vez más de moda) hasta miles de ellos (como el supercomputador BlueGene/L de IBM [45] que contará con hasta 65.536 procesadores para ofrecer un rendimiento pico de 360 TeraFLOPS – billones de operaciones de punto flotante por segundo –).

En general, podemos distinguir dos familias de multiprocesadores: los multiprocesadores de *memoria compartida* y los multiprocesadores de *paso de mensajes*. Los multiprocesadores de memoria compartida ofrecen al programador un espacio de direcciones único que es compartido por todos los procesadores. En estas máquinas, la comunicación entre los distintos procesadores tiene lugar de forma implícita, mediante operaciones de carga y de almacenamiento a posiciones de memoria compartida. Por otro lado, en los multiprocesadores de paso de mensajes cada procesador dispone de su propio espacio privado de direcciones. Dado que un determinado procesador no puede acceder al espacio de direcciones de otro, la comunicación entre procesadores en este tipo de máquinas se realiza mediante el envío de mensajes y debe ser especificada de manera explícita por el programador.

Tradicionalmente, los multiprocesadores de memoria compartida han venido gozando de una mayor popularidad entre los programadores debido al sencillo modelo de programación que ofrecen (memoria compartida), que no es más que una extensión de la forma de programar en monoprocesadores. En este paradigma la comunicación entre los diferentes nodos ocurre de forma implícita como consecuencia de instrucciones convencionales de acceso a memoria (cargas y almacenamientos), lo que hace que estas máquinas sean más sencillas de programar que las basadas en el paradigma de paso de mensajes. Tal es el caso común de los servidores comerciales, encargados de sistemas de administración de bases de datos y programas servidores Web. La mayoría de estos servidores son, desde el punto de vista de la arquitectura, multiprocesadores de memoria compartida.

Los multiprocesadores de memoria compartida adoptan dos tipos de arquitectura básica, en función del tipo de red de interconexión utilizada para conectar los distintos nodos. Si la red de interconexión está basada en un medio compartido por todos los nodos (como podría ser el caso de un bus), tenemos los multiprocesadores simétricos o SMPs, mientras que en el caso de utilizar una red punto-a-punto (como podría ser una malla o un toro) nos da lugar a la arquitectura de memoria compartida-distribuida o DSM, también conocida como cc-NUMA[1]. La utilización de un tipo u otro de arquitectura nos va a afectar

[1] No describiremos otros tipos de arquitectura que podrían aparecer, como máquinas COMA o máquinas sin coherencia de caches, para no alargar esta introducción.

asimismo al tipo de protocolo de coherencia que podemos utilizar. Así, los multiprocesadores SMP utilizarán un protocolo de coherencia basado en fisgoneo (*snooping*) [17], mientras que los multiprocesadores DSMs utilizarán un protocolo de coherencia basado en directorio [11].

Mientras que los multiprocesadores basados en bus común han tenido una amplia aceptación y, hoy en día, son la configuración asumida por la mayoría de los servidores actuales, este tipo de arquitectura no escala más allá de unas decenas de nodos. Precisamente dicho bus, al ser un medio compartido por todos los nodos del sistema, llega a ser un cuello de botella que limita el número de nodos que se pueden poner en esta arquitectura (por ejemplo, la máquina SUN-E10000 [13] ha llegado hasta los 64 nodos).

Para vencer la dificultad de escalar el ancho de banda del bus común a un número cada vez mayor de nodos, algunos diseños basados en *snooping* utilizan un "bus virtual" [13], formado a base de conmutadores (*switches*) y enlaces de alta velocidad, formando una red indirecta con topología de árbol. Aunque de esta forma se consigue ofrecer un ancho de banda mayor que en el caso de los buses, la latencia también es mayor que para un único bus e incluso que para una red directa. Así, y tal y como viene descrito en [33], para sistemas a partir de 16 procesadores el toro 2-D tiene una latencia menor que la anterior configuración. Por otra parte, el uso de conmutadores incrementa el coste total del sistema, siendo ésta otra desventaja con respecto a las redes directas.

Sin lugar a dudas, la solución al problema de la escalabilidad pasa por utilizar un multiprocesador con una red punto-a-punto, la memoria principal distribuida entre los diferentes nodos[2] del sistema (de cara a asegurar que el ancho de banda de la memoria escale con el número de procesadores) y un protocolo de coherencia basado en un directorio. Cada línea de memoria tiene asociada una entrada de directorio que, entre otras cosas, informa sobre las caches que almacenan una copia de la línea y que, por lo tanto, deberán recibir las transacciones de coherencia correspondientes cuando sea preciso. Al igual que en el caso de la memoria, la información de directorio se distribuye entre los distintos nodos del sistema.

Uno de los ejemplo mas conocidos de multiprocesador cc-NUMA disponible comercialmente ha sido el SGI Origin 2000/3000 [31], que puede escalar desde unos pocos procesadores hasta 1024 (512 nodos×2 procesadores/nodo). La Figura 1 muestra la arquitectura de una máquina cc-NUMA típica.

Además, las tendencias tecnológicas favorecen los protocolos de directorio, pues cada vez más en los sistemas actuales y futuros se integran en un único chip el procesador(es), la cache(s), la lógica necesaria para garantizar la coherencia, la lógica dedicada para el encaminamiento y el controlador de memoria. Ejemplos de esto los tenemos en el Alpha 21364 [39] o en el reciente AMD Opteron [27]. Estos nodos permiten una configuración escalable sencilla y rápida por medio de

[2] Entendemos por nodo el elemento constitutivo de la arquitectura del sistema, formado por un elemento de procesamiento (uno o varios procesadores), su correspondiente memoria (tanto la cache de primer y segundo nivel como la parte correspondiente de la memoria principal), el controlador de coherencia, y por último el interfaz de red.

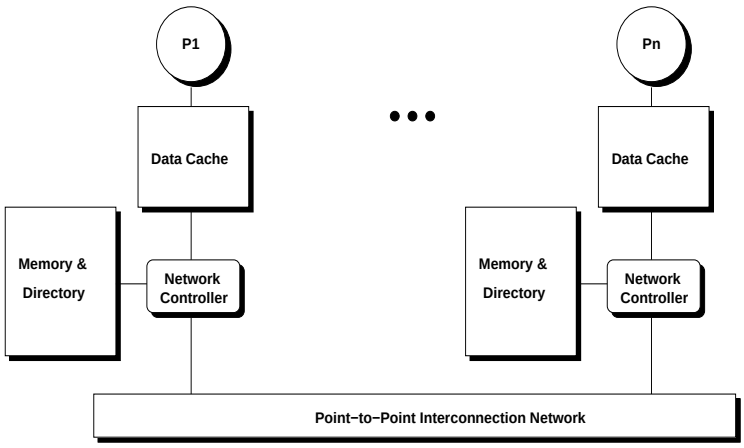


Figura 1. Arquitectura de un multiprocesador de memoria compartida distribuida escalable con coherencia de caches

una interconexión con redes directas, lo que impide la utilización de un protocolo de coherencia basado en fisgoneo dado que la red ya no garantiza la propiedad del ordenamiento en las peticiones, por lo que hay que usar un protocolo basado en directorio.

Aunque ésta ha sido la solución comercial adoptada hasta ahora para configuraciones de mediano y gran tamaño de número de nodos, este tipo de diseños también plantean una serie de problemas, estando los más importantes relacionados precisamente con el uso del directorio. Quizá los dos más importantes son la sobrecarga hardware que conlleva el uso del mismo y las grandes latencias en los accesos remotos a memoria.

El primero de los dos factores anteriores está fundamentalmente causado por la cantidad de memoria necesaria para almacenar la información de directorio, y más concretamente, la parte del directorio en la que se codifican los nodos que en un instante determinado mantienen una copia de cada línea (el *código de compartición*). La Figura 2 muestra la sobrecarga de memoria (calculada como tamaño del código de compartición dividido entre el tamaño de la línea de memoria) introducida por el código de compartición más sencillo, el vector de bits o *full-map*, cuando se utilizan líneas de memoria de 128 bytes. Como puede observarse, conforme el número de nodos en el sistema aumenta, la sobrecarga de memoria debida al directorio también lo hace, llegando a ser del 100 % para una configuración con 1024 nodos.

Por otro lado, las largas latencias de los fallos de L2 que caracterizan a las máquinas cc-NUMA están causadas fundamentalmente por la indirección que supone el uso de los directorios. Como consecuencia de la disparidad cada vez mayor existente entre las velocidades de las memorias y de los procesadores, y debido a que la información de directorio suele estar almacenada en la memoria principal, la componente más importante de la latencia de los fallos de L2, en

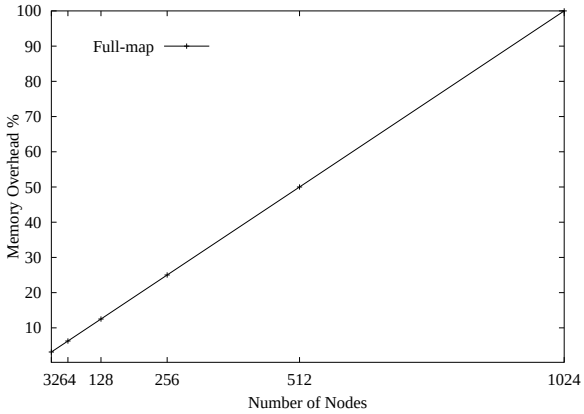


Figura 2. Sobrecarga de memoria para el código de compartición *full-map*

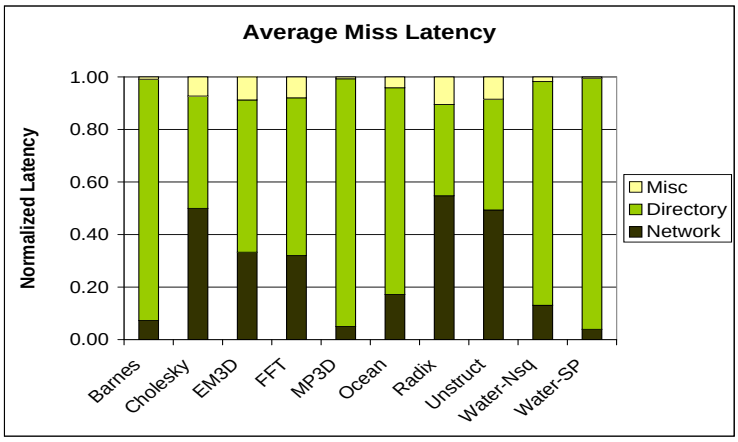


Figura 3. Latencia media de los fallos de L2

la mayoría de los casos, es la correspondiente al directorio, esto es, el número de ciclos que los directorios tardan en resolver cada fallo de L2. La Figura 3 muestra las latencias medias de los fallos de L2 para las aplicaciones utilizadas en el presente trabajo. Como puede observarse, para 7 de las 10 aplicaciones, la mayor parte de los ciclos que en promedio se requieren para resolver los fallos de L2 se consumen en el directorio.

Dichas latencias se manifiestan de forma dramática especialmente en los fallos de transferencia cache-a-cache y en los fallos de actualización [9]. Como ejemplo, una transferencia cache-a-cache tarda 1036 ns en un SGI Origin 2000, comparada a los 742 ns que necesita en una SUN-E10000, a pesar de tener unos tiempos parecidos de acceso a memoria [35].

Este trabajo de investigación presenta los desarrollos que se realizaron para tratar de solucionar los dos problemas que hemos presentado en el contexto de los sistemas multiprocesadores escalables de memoria compartida. Estos desarrollos constituyeron, en parte, la Tesis Doctoral de Manuel Acacio [5]. El resto del artículo se estructura como sigue. En la sección 2 describiremos los trabajos previos que se han desarrollado sobre este tópico. El agrupamiento multicapa y el directorio de dos niveles serán presentados en la sección 3, mientras que en la sección 4 mostraremos la nueva arquitectura de nodo basada en un directorio de tres niveles. A continuación, las propuestas realizadas serán evaluadas y analizadas, finalizando el artículo con unas conclusiones.

2. Trabajos previos relacionados

Los grandes problemas de los sistemas de memoria escalables basados en directorios son la sobrecarga adicional de memoria que la estructura de directorio introduce y las altas latencias en el acceso a memoria en caso de un fallo en L2. Vamos a ver algunos de los principales trabajos realizados en cada uno de estos campos.

2.1. La sobrecarga del directorio

Existen dos alternativas fundamentales a la hora de almacenar la información de directorio con nodos *home* fijos [15]: los esquemas basados en memoria (*flat, memory-based directories*) y los esquemas basados en cache (*flat, cache-based directories*). En los protocolos de coherencia que utilizan el primer tipo de esquemas, el nodo *home* que cada línea de memoria tiene asignado almacena información sobre el estado de la línea así como la lista de nodos que mantienen una copia de la misma. Por el contrario, para los protocolos de coherencia basados en la segunda de las alternativas, el nodo *home* almacena el estado de la línea y un puntero al primero de los compartidores (en lugar de la lista completa de compartidores almacenada en el caso anterior). El resto de nodos que mantienen una copia de la línea de memoria se disponen formando una lista doblemente enlazada mediante el uso de punteros adicionales asociados a cada *línea de cache* en cada nodo.

Desde un punto de vista hardware, para la primera alternativa se han planteado tradicionalmente dos soluciones ortogonales al problema de la sobrecarga del directorio: reducir el *ancho* o el *alto* de la estructura asociada al directorio [15]. La forma para reducir el ancho de la estructura del directorio es usar otra forma de codificación más comprimida distinta a la de *bit-vector*, como por ejemplo *coarse vector* [18], *gray-tristate* o *home* [37]. Otros autores proponen reducir el ancho de las entradas del directorio, poniendo en hardware tan sólo un limitado número de punteros (elegido para que cubra el caso más común) a cada línea de cache [12] y manejando (habitualmente por software) las situaciones en que se produce un desbordamiento en dicha estructura [15]. Ejemplos de tales tipos de sistemas podrían ser FLASH [28] o Alewife [6]. Más recientemente ha sido

propuesta la técnica denominada "directorio de segmentos" [14] como un híbrido entre los esquemas de punteros limitados y un esquema *full-map*.

La forma para reducir el alto del directorio (es decir, el número de entradas necesarias en el directorio) se puede realizar combinando varias entradas de directorio en una sola [44]. A partir de la apreciación de que el número total de líneas de memoria es mucho mayor que el número de líneas de cache, se han desarrollado otras estrategias para aprovechar esta propiedad de que los directorios sean muy dispersos y estén habitualmente vacíos, reduciendo de forma práctica el tamaño (el alto) del directorio, organizando la estructura del directorio como una cache [42].

Otra alternativa a estos diseños consiste en mantener la cuenta de los nodos compartidores por medio de un directorio formado por una lista enlazada, distribuyendo dicha estructura de directorio entre las caches de los diversos nodos que comparten una línea dada. Este protocolo se conoce con el nombre de protocolo de directorio encadenado, constituyendo el estándar SCI de IEEE [43]. Una máquina comercial construida según este protocolo ha sido la Sequent NUMA-Q [32], diseñada específicamente para cargas comerciales del estilo de bases de datos y procesamiento de transacciones comerciles (OLTP). Otras máquinas construidas con el protocolo de directorio SCI fueron también los diversos modelos de Convex Exemplar [46], siendo éstos más dirigidos al cálculo científico.

2.2. El problema de la indirección en los protocolos de coherencia

Un protocolo de coherencia de caches maneja los permisos de lectura y escritura de los datos de las caches para asegurar que todos los procesadores observan un vista de memoria consistente, lo que viene a denominarse el *invariante de coherencia*. Esto habitualmente permite que cada bloque de memoria tenga múltiples copias de sólo-lectura o una única copia de escritura, pero nunca ambas al mismo tiempo. Los protocolos actuales de coherencia fuerzan este invariante por medio de una delicada combinación de acciones locales y restricciones en el ordenamiento de las peticiones realizadas a un mismo bloque por los diversos procesadores de cara a evitar las condiciones de carrera. Por otra parte, el protocolo de coherencia también debe asegurar que el sistema esté libre de inanición *(starvation)*, es decir, que se asegure de que todas las referencias a memoria se puedan completar en algún momento determinado. Al mismo tiempo, un buen protocolo de coherencia debe procurar ofrecer las máximas prestaciones posibles, procurando minimizar los efectos de la ordenación (serialización) de escrituras y de la indirección introducida por el directorio[3].

Muchos trabajos se han realizado tratando de diseñar un protocolo de coherencia que tenga un comportamiento lo más parecido posible a un protocolo *snooping*, mientras mantiene la escalabilidad de los protocolos de directorio. El protocolo *Multicast Snooping* [10] es un protocolo híbrido *snooping*-directorio que permite que los procesadores envíen mensajes *multicast* a aquellos nodos que deben recibir las transacciones de coherencia. En caso de acierto, dicho

[3] Para aquellos protocolos que usen directorio.

protocolo tiene un comportamiento similar a un protocolo *snooping*. En [35] desarrollan *Timestamp Snooping*, un nuevo protocolo que añade el ordenamiento suficiente (mediante el empleo de marcas de tiempo y reordenamiento de peticiones) para soportar el protocolo tradicional de *snooping* sobre redes directas que no garantizan la ordenación de eventos. Posteriormente, los mismos autores han propuesto el protocolo denominado *Bandwidth Adaptive Snooping Protocol* (BASH) [36]. Este es un protocolo híbrido que se comporta como *snooping* (mediante transacciones de coherencia de difusión a todos los nodos) cuando en la red de interconexión hay ancho de banda suficiente, y se comporta como un protocolo de directorio (empleando mensajes uni-destino) cuando el ancho de banda es limitado.

Recientemente, Mark Hill y su grupo han desarrollado una nueva familia de protocolos de coherencia denominada "Coherencia de Token", basada en el concepto de desacoplar el rendimiento del protocolo de la comprobación de su corrección [34]. De esta forma, la parte dedicada al rendimiento se puede centrar en obtener las máximas prestaciones posibles (buscando mejorar sobre todo el caso más frecuente), sin preocuparse de las condiciones de carrera y de garantizar la coherencia, pues para eso ya está el substrato dedicado a asegurar la corrección del protocolo.

Otra línea de trabajo para conseguir el objetivo anteriormente citado ha sido por medio de la predicción. Nosotros mismos hemos desarrollado técnicas para reducir el problema de la indirección por medio de mejorar el protocolo de coherencia para soportar predicción de los nodos que tienen copia de la línea de cache, tanto para fallos de cache-a-cache [3] como para fallos de actualización [4]. La predicción en el contexto de multiprocesadores fue primeramente estudiada en [38] para predecir mensajes de coherencia. Posteriormente, en [29] se modificaron estos predictores para mejorar su efectividad y reducir su tamaño permitiendo, además, ejecutar operaciones de coherencia de forma especulativa basadas en las predicciones realizadas. Concretamente, el esquema propuesto intenta predecir cuándo el productor de una determinada línea de cache puede reenviar dicho dato a los consumidores de dicha línea. En [26] se presenta una clasificación de los esquemas de predicción, desarrollando nuevas propuestas más precisas que las anteriores. Otros autores [25] han propuesto optimizaciones a los patrones de compartición migratorios basadas en predicción. Por último, en [30] se propone el predictor denominado *"Last-Touch Predictor"*, el cual mejora por medio de la predicción una idea desarrollada años antes de que cada procesador se autoinvalide aquellas líneas compartidas que no va a seguir utilizando. De esta forma, se reduce la latencia de los fallos provocados por la actualización de datos.

Aunque inicialmente la idea de una cache de directorio fue propuesta para reducir su tamaño, también se ha usado como un medio para reducir el tiempo de acceso al mismo. En [40] propusieron la integración de caches de directorio dentro de los controladores de coherencia, idea que luego aplicaron al diseño de Everest [41]. Otros autores han aplicado la idea de usar caches para datos remotos para acelerar el acceso a dichos datos [32]. Por último, en [23] desarrollan esta técnica colocando caches para los datos remotos en los conmutadores de la red de

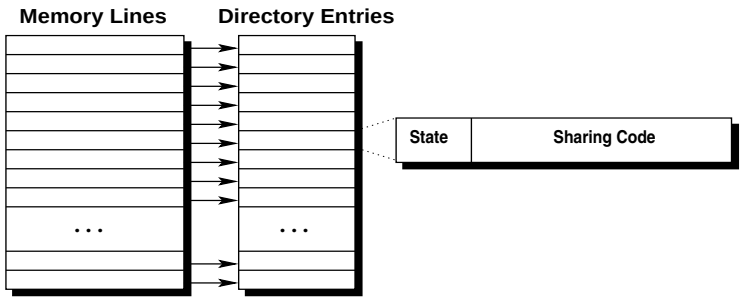


Figura 4. Organización del directorio para la familia de protocolos de coherencia asumidos en este trabajo

interconexión, aplicando en [24] una idea similar para reducir la latencia en los fallos cache-a-cache; en ambos casos, son necesarias topologías de red específicas para mantener coherentes los datos almacenados en los conmutadores.

Por último, citar que para pequeños sistemas el AMD Opteron evita el acceso al directorio realizando, en paralelo con dicha consulta, un *broadcast* a todos los nodos [8].

3. Agrupamiento Multicapa y Directorio de Dos Niveles

En esta sección nos centramos en los protocolos de coherencia pertenecientes a los esquemas basados en memoria. La Figura 4 muestra cómo se organiza el directorio en estos protocolos. Cada línea de memoria tiene asociada una *entrada de directorio* que se almacena en el nodo *home* correspondiente, al lado de la línea (normalmente en memoria principal). La entrada de directorio de una determinada línea de memoria se utiliza para almacenar el *estado* de la misma así como su *código de compartición*, el cual indica los nodos que mantienen una copia de la línea. Este último elemento consume la mayoría de los bits de cada entrada de directorio y su elección determina, de forma directa, la cantidad de memoria requerida para el directorio (por ejemplo, la sobrecarga de memoria introducida por el código de compartición *full-map* para una configuración de 1024 procesadores y para líneas de memoria de 128 bytes es del 100 %).

Con el objetivo de reducir la *anchura* de cada una de las entradas del directorio en configuraciones con un número considerable de nodos y, por lo tanto, la sobrecarga de memoria debida al directorio, se han propuesto los *códigos de compartición comprimidos*. Estos códigos de compartición requieren menos bits debido a que almacenan una representación *en exceso* de los compartidores de una línea de memoria en un instante determinado (se almacena un *superconjunto* del conjunto de los compartidores reales). De esta forma, comparados con códigos de compartición *precisos* como *full-map* ó *punteros limitados* que identifican exactamente qué nodos mantienen una copia de la línea, los códigos de compartición comprimidos introducen una pérdida de precisión que tiene consecuencias negativas en el rendimiento de las aplicaciones paralelas. Existen dos

motivos fundamentales por los que los códigos de compartición comprimidos pueden aumentar el tiempo de ejecución de las aplicaciones:

1. Aparición de *mensajes de coherencia innecesarios*. Es decir, mensajes de coherencia que aparecen como consecuencia de la codificación en exceso que se realiza de los compartidores. Estos mensajes consumen recursos del sistema (congestionan la red y los controladores, tanto de cache como de directorio) y son inútiles, es decir, no serían enviados si se dispusiera de un código de compartición preciso.
2. Se necesitan más mensajes para detectar ciertas condiciones de carrera.

Otra forma de reducir la cantidad de memoria requerida por los directorios consiste en disminuir la *altura* de los mismos. Es decir, en lugar de tener una entrada de directorio por cada línea de memoria, se dispone de una cache que almacena información de directorio para las líneas de memoria más recientemente usadas. El problema de utilizar un directorio de este tipo es la aparición de *invalidaciones innecesarias* como consecuencia de los reemplazos que tienen lugar en este tipo de estructuras. Estas invalidaciones innecesarias pueden aumentar drásticamente el número de fallos de L2 y, por lo tanto, afectar negativamente al rendimiento final.

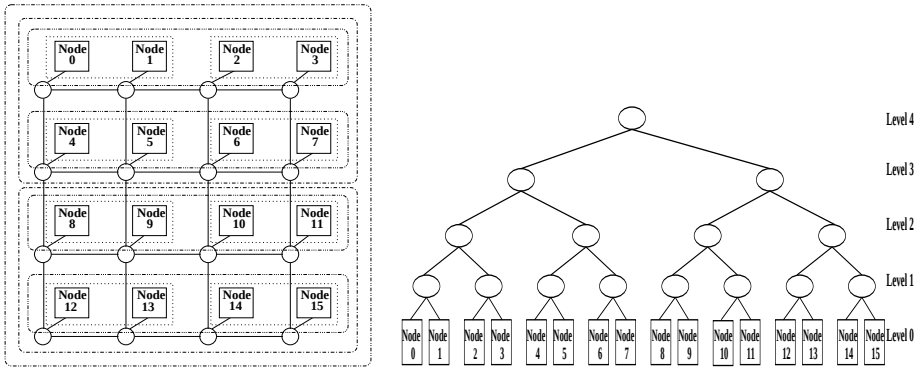
En el trabajo aquí presentado mostramos una nueva arquitectura de directorio, denominada *directorio de dos niveles*, que ofrece las ventajas de ambos tipos de soluciones en un único diseño. Una organización de directorio de dos niveles tiene la capacidad de reducir de forma sustancial los requerimientos de memoria sin afectar negativamente al rendimiento final.

3.1. Agrupamiento Multicapa

El objetivo del *agrupamiento multicapa* [1] es el de reducir el tamaño del código de compartición. Se realiza una distinción entre el sistema *físico* (tal y como es) y el sistema *lógico* (tal y como es visto por los directorios). La idea es la de formar un sistema lógico en el que todos los nodos se agrupan recursivamente en grupos de igual tamaño (en nuestro caso, 2) hasta que se obtenga un único grupo. La compresión se alcanza especificando el nivel del grupo mínimo en la jerarquía en el que están contenidos todos los compartidores. En nuestro caso este agrupamiento recursivo en grupos produce como resultado un árbol binario lógico en el que los nodos se encuentran en las hojas del árbol.

Por ejemplo, para un multiprocesador de 16 nodos que utiliza una malla como red de interconexión, la Figura 5(a) muestra el sistema físico, mientras que la Figura 5(b) muestra el sistema lógico que se obtiene al aplicar el agrupamiento multicapa. A partir de este sistema lógico derivamos tres nuevos códigos de compartición comprimidos:

1. **Árbol binario (*binary tree* ó *BT*)**. Los nodos que almacenan una copia de una determinada línea de memoria se expresan codificando el nivel de la raíz del subárbol mínimo que contiene a todos los compartidores. Esto puede ser expresado utilizando un total de $\lceil \log_2 (\log_2 N + 1) \rceil$ bits.



(a) Sistema *físico* (b) Sistema *lógico*

Figura 5. Ejemplo de aplicación del agrupamiento multicapa

2. **Árbol binario con nodos simétricos (*binary tree with symmetric nodes* ó *BT-SN*)**. Se introduce el concepto de nodos simétricos de un determinado nodo *home*. Si asumimos un total de 3 nodos simétricos por cada nodo home, éstos son codificados a través de las diferentes combinaciones de los 2 bits más significativos del identificador del nodo *home* (observar que una de las cuatro combinaciones representa al propio nodo *home*). Ahora el proceso de elegir el subárbol mínimo que incluya a todos los compartidores se repite para los nodos simétricos. Se elige entonces el mínimo de los 4 subárboles para representar a los compartidores (el calculado desde el nodo *home* y los calculados desde los 3 simétricos). El tamaño del código de compartición es el mismo que el del caso anterior más los 2 bits necesarios para codificar el simétrico que se usa.
3. **Árbol binario con subárboles (*binary tree with subtrees* ó *BT-SuT*)**. Este esquema resuelve el caso común de tener un único compartidor codificando directamente la identidad del nodo (para lo que hacen falta, al menos, $\log_2 N$ bits). Cuando varios nodos tienen copia de la línea se utiliza una representación alternativa: se emplean 2 subárboles (en lugar de 1) para incluir a todos los compartidores. Uno de ellos se calcula desde el nodo *home*, mientras que el otro se calcula desde uno de los 3 nodos simétricos. Por lo tanto, el tamaño total de este código de compartición es de $\text{máx}\left\{(1+\log_2 N), (1+2+2\lceil\log_2(\log_2 N)\rceil)\right\}$ bits.

3.2. Directorio de Dos Niveles

Aunque los directorios comprimidos (esto es, aquellos que utilizan un código de compartición comprimido) consiguen reducir de forma considerable la cantidad de memoria necesaria para almacenar la información de directorio, las

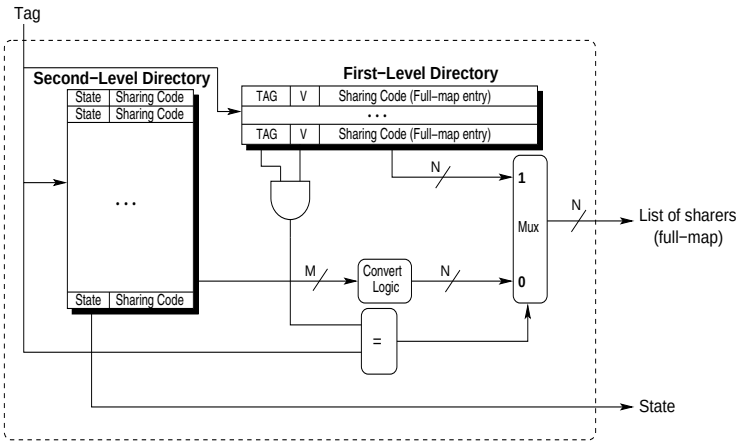


Figura 6. Arquitectura de un directorio de dos niveles

consecuencias negativas sobre el tiempo de ejecución de las aplicaciones que implican reduce la utilidad de los mismos.

En este trabajo presentamos una *arquitectura de directorio de dos niveles* [1] para reducir de manera significativa la cantidad de memoria dedicada al directorio y, a la misma vez, obtener el rendimiento de un directorio *full-map* que por definición no es escalable. Como se muestra en la Figura 6, un directorio de dos niveles consta de:

1. *Directorio de primer nivel*, que es una cache de directorio que mantienen información de directorio precisa para las líneas de memoria que han sido solicitadas más recientemente y que, por lo tanto, serán accedidas en un futuro cercano con más probabilidad. El código de compartición que hemos utilizado para este nivel ha sido *full-map*.
2. *Directorio de segundo nivel*, que es un directorio comprimido en el que se mantiene la información de directorio actualizada para todas las líneas de memoria que tiene asignadas el nodo *home*. Este nivel proporcionará la información de directorio sólo en aquellos casos en los que la entrada correspondiente no se encuentre en el directorio de primer nivel. Para este segundo nivel podría utilizarse cualquiera de los códigos de compartición comprimidos previamente presentados. En concreto, nos hemos decantado por BT-SuT, ya que es el que obtiene la mejor relación entre el costo en términos de requerimientos de memoria y el rendimiento.

4. Una Nueva Arquitectura para Reducir la Latencia de los Fallos de L2

Como ya apuntamos en la Sección 1, las largas latencias de los fallos de L2 en las que normalmente incurren los multiprocesadores cc-NUMA constituyen

otro de los factores que limitan la escalabilidad de este tipo de arquitecturas. El motivo de estas importantes latencias es la indirección que supone el acceso a la información de directorio y, más concretamente, el número de ciclos que el directorio *home* requiere para llevar a cabo el procesamiento de cada fallo de L2 (la *componente de directorio* de la latencia). Esto incluye el tiempo necesario para acceder a memoria principal, lugar en el que normalmente se ubica la información de directorio.

Desafortunadamente, la tendencia hacia microprocesadores cada vez más rápidos no se sigue en igual medida en el caso de la memoria, cuya velocidad crece mucho más lentamente [20]. Este hecho acentúa aún más el problema y tiene especial repercusión sobre aquellos fallos de L2 en los que el directorio *home* no tiene que proporcionar la línea de memoria, sino únicamente enviar mensajes de coherencia a los nodos que mantienen una copia de la línea. Al contrario que para las máquinas cc-NUMA, los multiprocesadores SMP no requieren el acceso a la memoria principal para este tipo de fallos, lo que reduce de manera considerable su latencia y, como consecuencia, aumenta la popularidad de los mismos.

Por otro lado, el número de transistores que pueden proporcionarse en un único chip aumenta cada vez más como consecuencia de las mejoras tecnológicas. Esto permite a los arquitectos de computadores incluir componentes claves dentro del chip del procesador principal. Por ejemplo, el procesador Alpha 21364 EV7 diseñado por Compaq [19] incluye dentro del procesador el controlador de memoria, la circuitería de coherencia así como el interfaz de red y el encaminador.

En la propuesta presentada hacemos uso de esta escala de integración con el objetivo de derivar una *nueva arquitectura* [2] capaz de reducir, de forma considerable, la latencia de los fallos de L2. Nuestra propuesta, cuyo punto de partida es una organización como la del Alpha 21364 EV7, reemplaza el directorio tradicional por un directorio de tres niveles (que es una extensión de la arquitectura de directorio de dos niveles presentada con anterioridad) y la adición de una cache para datos compartidos. El directorio de primer nivel y la cache para datos compartidos se incluyen dentro del chip del procesador, junto con la circuitería de coherencia.

4.1. Clasificación de los Fallos de L2

Considerando las acciones realizadas por el directorio *home* para resolver los fallos de L2, éstos pueden clasificarse en cuatro categorías (suponiendo los estados MESI para las caches):

1. Fallos *$-to-$*: el directorio tiene que re-enviar el fallo al único nodo que mantiene una copia de la línea de memoria.
2. Fallos *Mem*: el directorio tiene que proporcionar directamente la línea de memoria.
3. Fallos *Inv*: el directorio tiene que invalidar todas las copias de la línea de memoria salvo la que el nodo peticionario mantiene.

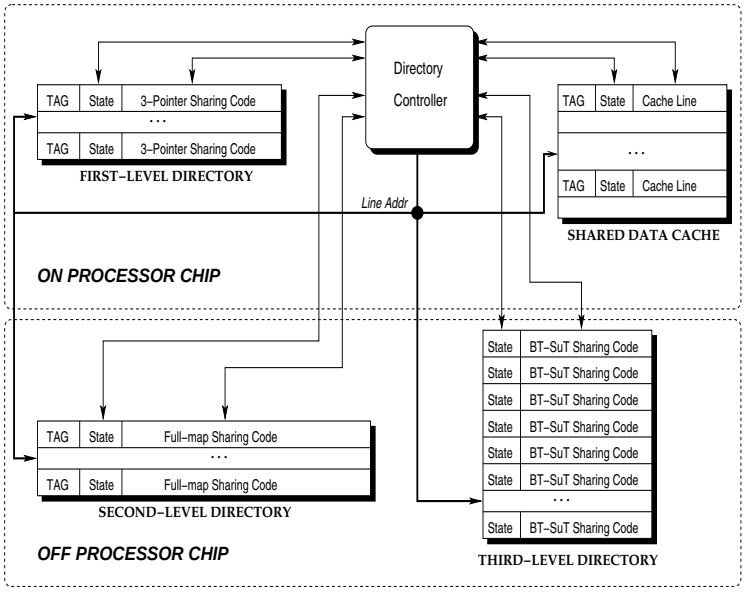


Figura 7. Arquitectura propuesta para los nodos de un multiprocesador cc-NUMA

4. Fallos *Inv+Mem*: el directorio tiene que invalidar primero todas las copias de la línea de memoria para enviar, a continuación, una copia de la misma al nodo peticionario.

4.2. Una Nueva Arquitectura de Nodo

Partiendo de una arquitectura como la del Alpha 21364 EV7, la arquitectura de nodo que presentamos y que mostramos en la Figura 7, añade los siguientes elementos:

- **Directorio de tres niveles**. Este directorio reemplaza al directorio tradicional que se ubica en memoria principal. El directorio de tres niveles que proponemos constituye una extensión de la arquitectura de directorio de dos niveles que presentamos con anterioridad. En particular, distinguimos los siguientes niveles de directorio:
  - *Directorio de primer nivel.* Es una pequeña cache de directorio que ubicamos dentro del chip del procesador. Cada entrada en este directorio dispone de un número limitado de punteros como código de compartición. En concreto, hemos observado que disponer de 3 punteros por cada entrada de directorio es suficiente para la mayoría de los casos y, por lo tanto, cada entrada de directorio en este nivel puede almacenar la identidad de hasta 3 compartidores.
  - *Directorio de segundo nivel.* Este nivel constituye una *cache de víctimas* del primero. Es decir, en este nivel vamos a almacenar todas las entradas

de directorio que fueron desalojadas del primer nivel, bien como consecuencia de un reemplazo, bien debido a que el número de compartidores en un instante determinado supera el límite de los de los 3 que pueden almacenarse en cada entrada del directorio de primer nivel. Este directorio de segundo nivel, que usa como código de compartición *full-map*, lo implementamos haciendo uso de una cache de directorio y lo ubicamos fuera del chip del procesador. Es importante reseñar que la información de directorio para una determinada línea de memoria sólo puede residir en uno de los dos primeros niveles, nunca en ambos.

- *Directorio de tercer nivel.* Es un directorio comprimido que se ubica al lado de la memoria principal. El directorio de tercer nivel mantiene información de compartición actualizada para todas las líneas de memoria y es accedido cuando la entrada de directorio que se busca no está disponible en los dos niveles inferiores. Este nivel de directorio emplea *BT-SuT* como código de compartición comprimido.

Mientras que a través del directorio de primer nivel se consiguen reducciones sustanciales de la latencia de los fallos de L2, muchos de los cuales pueden ser satisfechos rápidamente desde el propio chip del procesador, el cometido fundamental de los dos niveles restantes es el de reducir la sobrecarga de memoria debida al directorio. Como puede observarse, la arquitectura de directorio de tres niveles que acabamos de describir se deriva de la arquitectura de dos niveles que propusimos con anterioridad, añadiendo únicamente lo que ahora denominamos el primer nivel.

- **Caché para datos compartidos**. Esta estructura es una pequeña cache para datos que ubicamos dentro del chip del procesador, junto al directorio de primer nivel. El propósito de la cache para datos compartidos es el de almacenar aquellas líneas de memoria que el directorio *home* probablemente tendrá que proporcionar y que, de otra forma, tendrían que ser buscadas en memoria principal.

A través del diseño que acabamos de presentar, podemos reducir de forma significativa el número de ciclos que el directorio *home* requiere para cada una de las categorías de los fallos de L2 anteriormente presentadas. Para los fallos *$-to-$* (también conocidos como fallos *3-hop*), el directorio *home* puede re-enviar rápidamente la petición al propietario actual de la línea de memoria cuando la entrada de directorio correspondiente se encuentra en el directorio de primer nivel y, en menor medida, cuando lo hace en el directorio de segundo nivel. De manera similar, el proceso de invalidación para los fallos *Inv* podrá comenzar antes cuando el directorio de primer o segundo nivel proporciona la información de compartición. Como comentamos con anterioridad, para los fallos *Mem* el directorio debe proporcionar la línea de memoria y, por lo tanto, los fallos pertenecientes a esta categoría serán acelerados cuando ésta se encuentre en la cache para datos compartidos. Finalmente, hemos observado cómo el directorio emplea la mayor parte de los ciclos necesarios para resolver un fallo *Inv+Mem* realizando el proceso de invalidación de los compartidores. De ésta forma, el encontrar la línea de memoria en la cache para datos compartidos no supone

ningún beneficio en cuanto a reducción de la latencia de estos fallos, ya que la línea no es requerida antes de que el proceso de invalidación haya concluido. Igualmente, el no hallar la línea en la cache para datos compartidos no supone perjuicio alguno, debido a que el acceso a memoria principal ocurriría en paralelo con la invalidación de los compartidores. Como para el caso *Inv*, los beneficios que este tipo de fallos pueden obtener de la arquitectura propuesta radican en que el proceso de invalidación puede comenzar antes cuando la información de directorio se encuentra en uno de los dos primeros niveles.

5. Evaluación y Análisis

5.1. Entorno de Evaluación

A la hora de llevar a cabo la evaluación de las propuestas presentadas en en este trabajo, decidimos hacer uso del simulador RSIM v.1.0 (***R**ice **S**imulator for **I**LP **M**ultiprocessors*), el cual está disponible de forma pública [22]. RSIM es un simulador guiado por la ejecución (*execution-driven*) que permite llevar a cabo la simulación de un multiprocesador cc-NUMA actual con un alto grado de detalle, y ofrece la posibilidad de seleccionar los valores de muchos de los parámetros de la arquitectura. El simulador modela un procesador *superescalar* en cada uno de los nodos (muy parecido a un procesador R10000 [48]), la jerarquía de memoria (2 niveles de memoria cache, el bus del sistema y el directorio, que se encuentra en memoria principal) así como una red de interconexión escalable con topología de malla 2-D. La Tabla 1 resume los valores de los parámetros que hemos utilizado en nuestras simulaciones. Estos valores son representativos de los multiprocesadores cc-NUMA actuales. Hemos elegido la consistencia secuencial como modelo de memoria siguiendo las recomendaciones dadas por Hill [21].

El simulador RSIM ha de utilizarse conjuntamente con algunos programas de prueba (*benchmarks*) que son ejecutados por el mismo. En nuestro caso particular, hemos seleccionado diez programas de prueba: BARNES-HUT, CHOLESKY, FFT, OCEAN, RADIX, WATER-SP y WATER-NSQ, de la *suite* SPLASH-2 [47], EM3D, que es una implementación para memoria compartida de la versión Split-C, MP3D, de la *suite* SPLASH y, finalmente, UNSTRUCTURED, que es una aplicación para dinámica de fluidos. La Tabla 2 muestra los programas de evaluación que hemos utilizado en los desarrollos aquí presentados, junto con los tamaños de problema usados en cada caso. Estos tamaños han sido seleccionados en función del número máximo de procesadores disponibles en cada caso y que se muestra en la tercera columna de la tabla.

5.2. Evaluación del agrupamiento multicapa

De los tres códigos de compartición comprimidos que aquí proponemos, *BT-SuT* es el que obtiene los mejores resultados en términos de número de mensajes de coherencia innecesarios introducidos. Los otros dos códigos de compartición,

ILP Processor	
Processor Speed	1 GHz
Max. fetch/retire rate	4
Active List	64
Functional Units	2 integer arithmetic
	2 floating point
	2 address generation
Branch Predictor	2-bit history predictor
Counters in the Branch Predictor Buffer	512
Shadow Mappers	8
Memory queue size	32 entries
Memory Hierarchy Parameters	
Cache line size	64 bytes
L1 cache (on-chip, WT)	Direct mapped, 32KB
L1 request ports	2
L1 hit time	2 cycles
L2 cache (on-chip, WB)	4-way associative, 512KB
L2 request ports	1
L2 hit time	15 cycles, pipelined
Number of MSHRs	8 per cache
Memory access time	70 cycles (70 ns)
Memory interleaving	4-way
Cache-coherence protocol	MESI
Consistency model	Sequential consistency
Directory Cycle	10 cycles
First coherence message creation time	4 directory cycles
Next coherence messages creation time	2 directory cycles
Bus Speed	1 GHz
Bus width	8 bytes
Network Parameters	
Router speed	250 MHz
Channel width	32 bits
Channel speed	500 MHz
Number of channels	1
Flit size	8 bytes
Non-data message size	16 bytes
Router's internal bus width	64 bits
Arbitration delay	1 router cycle

Tabla 1. Valores de los parámetros usados en las simulaciones

BT y *BT-SN*, son demasiado agresivos (es decir, comprimen en demasía la información de compartición) y, como resultado, incrementan en gran medida el tiempo de ejecución de las aplicaciones.

La Figura 8 compara *BT-SuT* con *coarse vector* [18], *gray-tristate* [37] y Dir_1B [7], en términos del tiempo de ejecución obtenido para las aplicaciones usadas en este trabajo. El tiempo de ejecución obtenido en cada caso ha sido

Benchmark	Problem Size	Max Processors	Memory lines	Dominant Sharing Patterns
BARNES-HUT	8192 bodies	64	22.32 K	Wide sharing
CHOLESKY	tk15.O	64	249.74 K	Migratory
EM3D	38400 nodes	64	120.55 K	Producer-consumer
FFT	256K complex doubles	64	200.87 K	Producer-consumer
MP3D	48000 nodes	64	30.99 K	Migratory
OCEAN	258x258 ocean	32	248.48 K	Producer-consumer
RADIX	2M keys	64	276.30 K	Producer-consumer
UNSTRUCTURED	Mesh.2K	32	13.54 K	Producer-consumer and migratory
WATER-NSQ	512 molecules	64	71.81 K	Migratory
WATER-SP	512 molecules	64	5.85 K	Wide sharing

Tabla 2. Resumen de los tamaños de problema, número máximo de procesadores que pueden usarse, número total de líneas de memoria compartida y patrones de compartición principales, para las aplicaciones mostradas en este trabajo

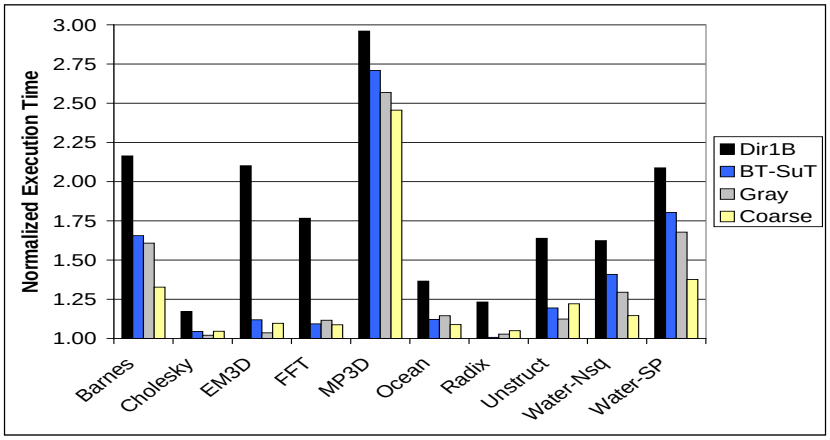


Figura 8. Tiempos de ejecución normalizados para los códigos de compartición *coarse vector*, *gray-tristate*, *BT-SuT* y Dir_1B

normalizado con respecto al que se conseguiría con un directorio *full-map*. Los dos primeros códigos de compartición comprimidos introducen una sobrecarga de memoria mayor, mientras que la inducida por Dir_1B es ligeramente inferior.

Cuando comparamos nuestro esquema más elaborado con Dir_1B, podemos observar en la Figura 8 cómo *BT-SuT* obtiene tiempos de ejecución inferiores para todas las aplicaciones. A pesar de que ambos códigos de compartición tienen un tamaño similar, *BT-SuT* realiza un uso más eficiente de los bits que tiene disponibles. Por otro lado, para las aplicaciones CHOLESKY, EM3D, FFT, OCEAN, RADIX and UNSTRUCTURED, los resultados obtenidos por *coarse vector*,

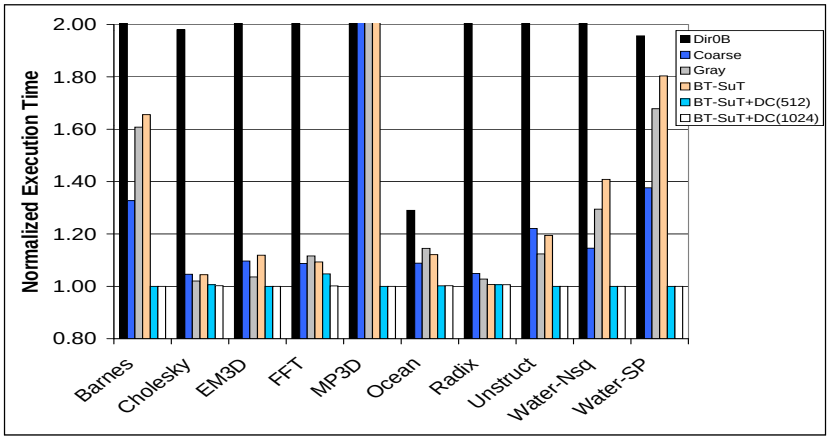


Figura 9. Tiempos de ejecución normalizados para la configuración *BT-SuT* y directorio de primer nivel(*FM*)

gray-tristate y *BT-SuT* son muy parecidos, incluso *BT-SuT* consigue mejores resultados que estos dos códigos de compartición en algunos casos. Para el resto de las aplicaciones, *BT-SuT* logra tiempos de ejecución similares a los obtenidos con *gray-tristate* aunque los mejores resultados se obtienen al utilizar *coarse vector* como código de compartición. Es importante hacer notar que las diferencias, sin embargo, no son excesivamente importantes. El código de compartición *BT-SuT* requiere aproximadamente la mitad de los bits que son necesarios para *gray-tristate* y no tiene los problemas de escalabilidad que *coarse vector* presenta. Por lo tanto, podemos concluir que *BT-SuT* logra la mejor relación entre la sobrecarga de memoria introducida y la degradación del rendimiento experimentada como consecuencia de la compresión de la información de directorio.

5.3. Evaluación del directorio de dos niveles

La Figura 9 muestra los resultados que se obtienen para la arquitectura de directorio de dos niveles previamente explicada cuando el directorio de primer nivel dispone de 512 y 1024 entradas (barras *BT-SuT+DC(512)* y *BT-SuT+DC(1024)*, respectivamente). Se muestran, además, los tiempos de ejecución obtenidos con los distintos códigos de compartición comprimidos. De nuevo, los tiempos de ejecución han sido normalizados con respecto a los obtenidos con un directorio *full-map*.

Podemos observar cómo el rendimiento de un directorio *full-map* puede alcanzarse cuando cada uno de los nodos del multiprocesador incluye un directorio de dos niveles en el que se combinan un pequeño directorio de primer nivel (4 KB para la configuración de 512 entradas) y un directorio completo (una entrada por cada línea de memoria) y comprimido (utilizando *BT-SuT* como código de compartición).

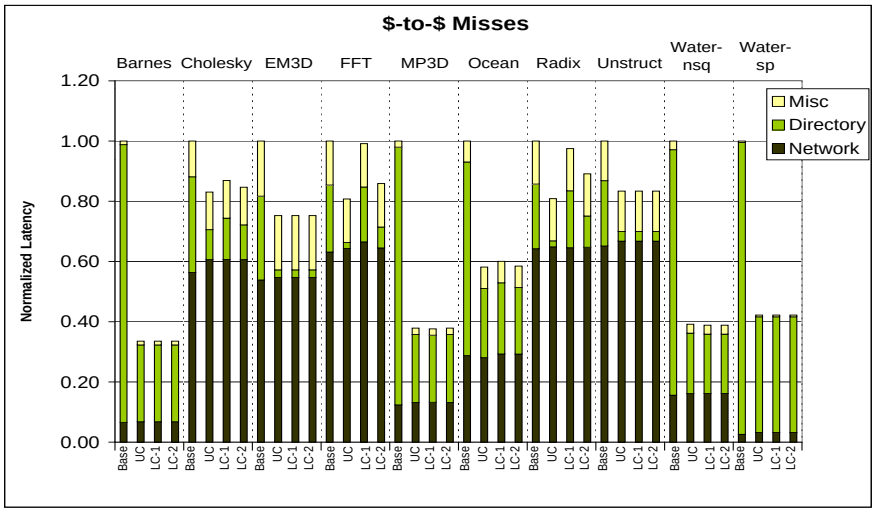


Figura 10. Latencia media de los fallos *$-to-$*

5.4. Evaluación de la nueva arquitectura de nodo

Las Figuras 10, 11, 12 y 13 reflejan cuantitativamente los beneficios (en términos de reducción de las latencias medias normalizadas) conseguidos para cada una de las categorías de los fallos de L2. En todos los casos se comparan el sistema base (*Base*), dos configuraciones basadas en la arquitectura presentada con tamaños diferentes para los directorios de primer y segundo nivel y para la cache para datos compartidos (*LC-1* y *LC-2*)[4] y, por último, una configuración que nos proporciona una aproximación al potencial de nuestra propuesta mediante la utilización de un número ilimitado de entradas en las caches de directorio y en la de datos compartidos (*UC*). Las latencias medias normalizadas mostradas en cada una de las figuras aparecen divididas en tres componentes: latencia de red, latencia de directorio y latencia miscelánea (ciclos empleados en las caches, el bus...).

La Figura 10 muestra la latencia media normalizada de los fallos pertenecientes a la primera de las cuatro categorías anteriores (es decir, los fallos *$-to-$*). Como puede observarse, las reducciones de latencia que se obtienen cuando se usa la configuración *LC-2* van desde un 11 % para RADIX hasta un 66 % para BARNES. Estas reducciones están muy próximas a las que se consiguen con caches ilimitadas para los directorios de primer y segundo nivel y para la cache para datos compartidos (configuración *UC*).

[4] *LC1* limita a 512 el número de entradas de estas estructuras, lo que representa un tamaño de 2KB, 3KB y 32KB para el directorio de primer nivel, segundo nivel y cache de datos compartidos, respectivamente. *LC2* fija el número de entradas a 1024, representando unos tamaños de 3KB, 10KB y 64KB.

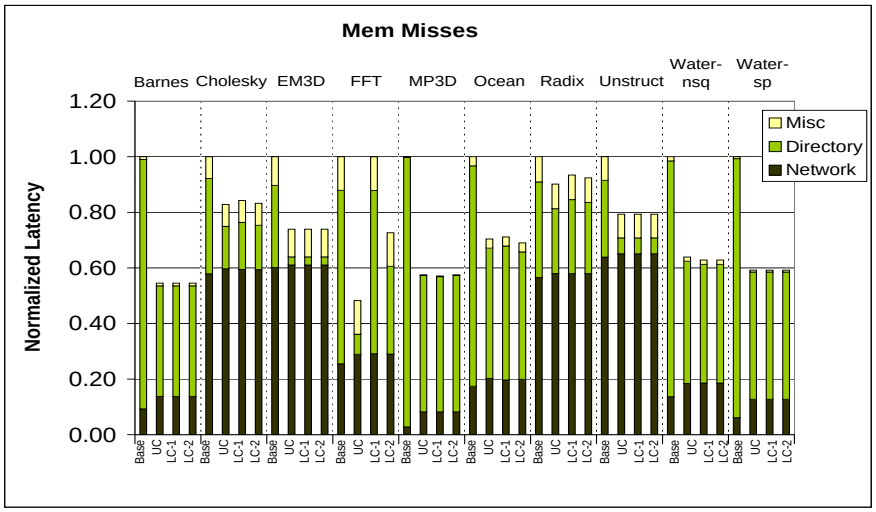


Figura 11. Latencia media de los fallos *Mem*

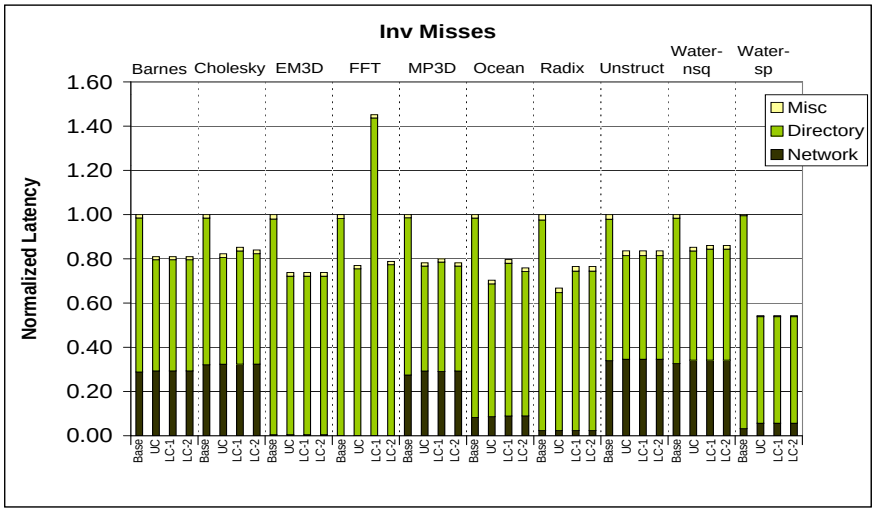


Figura 12. Latencia media de los fallos *Inv*

La Figura 11 presenta las latencias medias normalizadas para los fallos *Mem*. En este caso, las reducciones de latencia van desde un 10 % para RADIX hasta un 45 % para BARNES cuando utilizamos la configuración *LC-2*. Para todas las aplicaciones salvo para FFT, los resultados que obtiene esta configuración coinciden prácticamente con los mostrados para la configuración *UC*.

Como se presenta en la Figura 12, las reducciones en términos de latencia media normalizada para los fallos *Inv* van desde un 15 % para CHOLESKY y

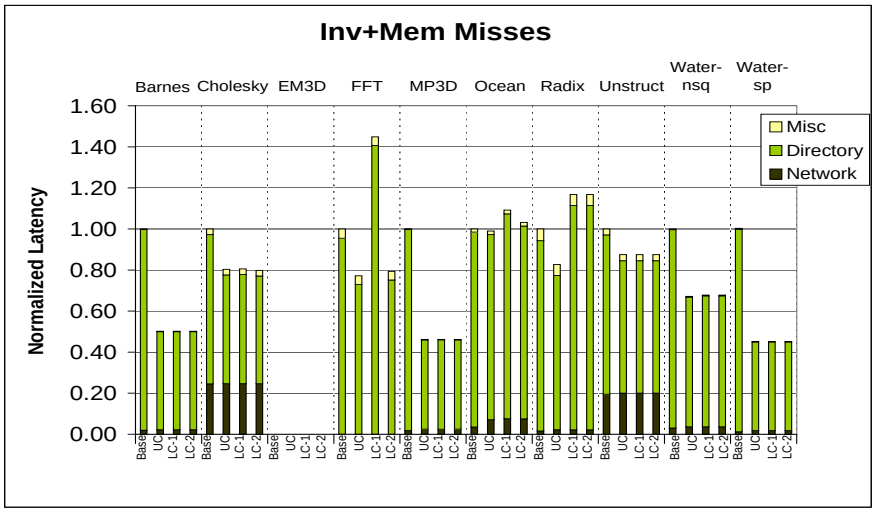


Figura 13. Latencia media de los fallos *Inv+Mem*

WATER-NSQ, hasta el 45 % observado para WATER-SP, sobre todo cuando usamos la configuración *LC-2*. Es importante reseñar que la degradación encontrada en FFT para la configuración *LC-1* es consecuencia del gran número de fallos *Inv* para los que se necesita acceder al directorio comprimido de tercer nivel. Esto es debido a que la entrada de directorio correspondiente no se encuentra en los dos primeros niveles y, como puede verse, desaparece completamente cuando pasamos a utilizar la configuración *LC-2*. Podemos ver también cómo los resultados conseguidos por la configuración *LC-2* están muy próximos a los de la configuración *UC*.

Por último, la Figura 13 presenta las latencias medias normalizadas para todas las aplicaciones en las que ocurren fallos *Inv+Mem* (es decir, todas salvo EM3D). Las reducciones obtenidas en este caso cuando se emplea la configuración *LC-2* van desde un 13 % para UNSTRUCTURED hasta un 55 % para WATER-SP. Como puede observarse, para algunas aplicaciones como OCEAN y especialmente RADIX esta configuración no sólo no es capaz de reducir la latencia de este tipo de fallos, sino que la aumenta ligeramente debido a que para muchos de estos fallos es preciso llegar al directorio de tercer nivel para encontrar la información de directorio.

6. Conclusiones

A pesar de que a través del uso de protocolos de coherencia de cache basados en directorio podemos llevar a cabo la realización de multiprocesadores de memoria compartida escalables y de alto rendimiento, más allá de los límites de los protocolos basados en *fisgoneo*, existen una serie de factores que limitan el

número máximo de nodos que una máquina cc-NUMA puede ofrecer con una buena relación coste/redimiento.

Dos de estos factores son la sobrecarga de circuitería que supone el uso de los directorios, principalmente la cantidad de memoria necesaria para almacenar esta información, y las largas latencias de los fallos de L2 que caracterizan a los multiprocesadores cc-NUMA actuales. Las propuestas aquí mostradas han estado dirigidas a paliar estos dos problemas.

En primer lugar presentamos una nueva organización para el directorio, que hemos denominado *directorio de dos niveles*, a través de la cual se consigue reducir de forma significativa la sobrecarga de memoria introducida por la información de directorio sin que ello suponga una pérdida de rendimiento. Además, presentamos el concepto de *agrupamiento multicapa* y derivamos tres nuevos códigos de compartición comprimidos con menores requerimientos en cuanto a memoria se refiere que los propuestos con anterioridad.

A continuación añadimos a la arquitectura de directorio de dos niveles un tercer nivel que integramos, junto con una cache para datos compartidos, dentro de chip del procesador como una forma de acelerar significativamente los fallos de L2. Esta nueva *arquitectura de nodo* que derivamos a partir de una *clasificación de los fallos de L2* en función del trabajo realizado por el directorio para resolverlos, consigue obtener importantes reducciones en la componente de la latencia debida al directorio para todas las categorías de la clasificación.

Referencias

1. Manuel E. Acacio, José González, José M. García, and José Duato. A new scalable directory architecture for large-scale multiprocessors. In *High-Performance Computer Architecture (HPCA)*, pages 97–106, Monterrey, México, January 2001. IEEE CS Press.
2. Manuel E. Acacio, José González, José M. García, and José Duato. A novel approach to reduce l2 miss latency in shared-memory multiprocessors. In *16th Int'l Parallel and Distributed Processing Symposium (IPDPS'02)*, Fort Laudedale (Florida), USA, April 2002. IEEE Computer Society Press.
3. Manuel E. Acacio, José González, José M. García, and José Duato. Owner prediction for accelerating cache-to-cache transfer misses in cc-numa multiprocessors. In *SC2002 High Performance Networking and Computing*, Baltimore (Maryland), USA, November 2002. IEEE Computer Society Press.
4. Manuel E. Acacio, José González, José M. García, and José Duato. The use of prediction for accelerating upgrade misses in cc-numa multiprocessors. In *11th Int'l Conference on Parallel Architectures and Compilation Techniques (PACT 2002)*, pages 155–164, Charlottesville (Virginia), USA, September 2002. IEEE Computer Society Press.
5. Manuel Eugenio Acacio. *Improving the Performance and Scalability of Directory-based Shared-Memory Multiprocessors.* PhD thesis, Facultad de Informática. Universidad de Murcia, April 2003.
6. Anant Agarwal, Ricardo Bianchini, David Chaiken, Kirk L. Johnson, David Kranz, John Kubiatowicz, Beng-Hong Lim, Ken Mackenzie, and Donald Yeung. The mit alewife machine: Architecture and performance. *Proc. of the 22nd Int'l Symposium on Computer Architecture (ISCA'95)*, pages 2–13, May/June 1995.

7. Anant Agarwal, Richard Simoni, John Hennessy, and Mark 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.
8. Ardsher Ahmed, Pat Conway, Bill Hughes, and Fred Weber. Amd opteron^{TM} shared memory mp systems. *Proc. of the 14th HotChips Symposium*, August 2002.
9. Luiz A. Barroso, Kourosh Gharachorloo, and Edouard Bugnion. Memory system characterization of commercial workloads. *In Proc. of the 25th Int'l Symposium on Computer Architecture (ISCA'98)*, pages 3–14, June 1998.
10. E. Ender Bilir, Ross M. Dickson, Ying Hu, Manoj Plakal, Daniel J. Sorin, Mark D. Hill, and David A. Wood. Multicast snooping: A new coherence method using a multicast address network. *Proc. of the 26th Int'l Symposium on Computer Architecture (ISCA'99)*, pages 294–304, May 1999.
11. L.M. Censier and P. Feautrier. A new solution to coherence problems in multicache systems. *IEEE Transactions on Computers*, 27(12):1112–1118, December 1978.
12. D. Chaiken, J. Kubiatowicz, and A. Agarwal. Limitless directories: A scalable cache coherence scheme. *Proc. of International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS IV)*, pages 224–234, April 1991.
13. A. Charlesworth. Extending the smp envelope. *IEEE Micro*, 18(1):39–49, Jan/Feb 1998.
14. Jong H. Choi and Kyu Ho Park. Segment directory enhancing the limited directory cache coherence schemes. *Proc. of the 13th Int'l Parallel and Distributed Processing Symposium (IPDPS'99)*, pages 258–267, April 1999.
15. David E. Culler, Jaswinder P. Singh, and Anoop Gupta. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann Publishers, Inc., 1999.
16. Michael Flynn. Very high-speed computing systems. *Proceedings of the IEEE*, 54:1901–1909, December 1966.
17. J. Goodman. Using cache memories to reduce processor-memory traffic. *Proc. of the Int'l Symposium on Computer Architecture (ISCA'83)*, June 1983.
18. A. Gupta, W.-D. Weber, and T. Mowry. Reducing memory and traffic requirements for scalable directory-based cache coherence schemes. *Proc. Int'l Conference on Parallel Processing (ICPP'90)*, pages 312–321, August 1990.
19. L. Gwennap. Alpha 21364 to ease memory bottleneck. *Microprocessor Report*, 12(14):12–15, October 1998.
20. Haldun Hadimioglu, David Kaeli, and Fabrizio Lombardi. Introduction to the special issue on high performance memory systems. *IEEE Transactions on Computers*, 50(11):1103–1105, November 2001.
21. Mark D. Hill. Multiprocessors should support simple memory-consistency models. *IEEE Computer*, 31(8):28–34, August 1998.
22. Christopher J. Hughes, Vijay S. Pai, Parthasarathan Ranganathan, and Sarita V. Adve. Rsim: Simulating shared-memory multiprocessors with ilp processors. *IEEE Computer*, 35(2):40–49, February 2002.
23. Ravi Iyer and Laxmi N. Bhuyan. Switch cache: A framework for improving the remote memory access latency of cc-numa multiprocessors. *Proc. of the 5th Int'l Symposium on High Performance Computer Architecture (HPCA-5)*, pages 152–160, January 1999.
24. Ravi Iyer, Laxmi N. Bhuyan, and Ashwini Nanda. Using switch directories to speed up cache-to-cache transfers in cc-numa multiprocessors. *Proc. of the 14th Int'l Parallel and Distributed Processing Symposium (IPDPS'00)*, pages 721–728, May 2000.

25. Stefanos Kaxiras and James R. Goodman. Improving cc-numa performance using instruction-based prediction. *Proc. of the 5th Int'l Symposium on High Performance Computer Architecture (HPCA-5)*, pages 161–170, January 1999.
26. Stefanos Kaxiras and Cliff Young. Coherence communication prediction in shared-memory multiprocessors. *Proc. of the 6th Int'l Symposium on High Performance Computer Architecture (HPCA-6)*, pages 156–167, January 2000.
27. C. Keltcher, Pat Conway, Bill Hughes, and Fred Weber. The amd opteron processor for multiprocessor servers. *IEEE Micro*, 23(2):66–76, Mar-Apr 2003.
28. Jeffrey Kuskin, David Ofelt, Mark Heinrich, John Heinlein, Richard Simoni, Kourosh Gharachorloo, John Chapin, David Nakahira, Joel Baxter, Mark Horowitz, Anoop Gupta, Mendel Rosenblum, and John Hennessy. The stanford flash multiprocessor. *Proc. of the 21st Int'l Symposium on Computer Architecture (ISCA'94)*, pages 302–313, April 1994.
29. An C. Lai and Babak Falsafi. Memory sharing predictor: The key to a speculative dsm. *Proc. of the 26th Int'l Symposium on Computer Architecture (ISCA'99)*, pages 172–183, May 1999.
30. An-Chow Lai and Babak Falsafi. Selective, accurate, and timely self-invalidation using last-touch prediction. *Proc. of the 27th Int'l Symposium on Computer Architecture (ISCA'00)*, pages 139–148, May 2000.
31. James Laudon and Daniel Lenoski. The SGI origin: A ccNUMA highly scalable server. *Proc. of the 24th Int'l Symposium on Computer Architecture (ISCA'97)*, pages 241–251, June 1997.
32. T. Lovett and R. Clapp. STiNG: A CC-NUMA computer system for the commercial marketplace. In *Proc. of the 23rd Annual Int'l Symp. on Computer Architecture (ISCA'96)*, pages 308–317, 1996.
33. Milo M. Martin, Mark D. Hill, and David A. Wood. Token coherence: A new framework for shared-memory multiprocessors. *IEEE Micro*, 23(2):108–115, Nov-Dec 2003.
34. Milo M. Martin, Mark D. Hill, and David A. Wood. Token coherence: Decoupling performance and correctness. *Proc. of the 30th Int'l Symposium Computer Architecture (ISCA'03)*, June 2003.
35. Milo M. Martin, Daniel J. Sorin, Anastassia Ailamaki, Alaa R. Alameldeen, Ross M. Dickson, Carl J. Mauer, Kevin E. Moore, Manoj Plakal, Mark D. Hill, and David A. Wood. Timestamp snooping: An approach for extending smps. *Proc. of International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS IX)*, pages 25–36, November 2000.
36. Milo M. Martin, Daniel J. Sorin, Mark D. Hill, and David A. Wood. Bandwidth adaptive snooping. *Proc. of the 8th Int'l Symposium on High Performance Computer Architecture (HPCA-8)*, pages 251–262, February 2002.
37. Shubhendu S. Mukherjee and Mark 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.
38. Shubhendu S. Mukherjee and Mark D. Hill. Using prediction to accelerate coherence protocols. *Proc. of the 25th Int'l Symposium on Computer Architecture (ISCA'98)*, pages 179–190, July 1998.
39. Shubhendu S. Mukherjee, Shamik D. Sharma, Mark D. Hill, James R. Larus, Anne Rogers, and Joel Saltz. The alpha 21364 network architecture. In *Proc. of the 9th Symposium on High-Performance Interconnects (HOTI'01)*, pages 113–118. IEEE CS Press, 2001.

40. Ashwini K. Nanda, Anthony-Trung Nguyen, Maged M. Michael, and Douglas J. Joseph. High-throughput coherence controllers. *Proc. of the 6th Int'l Symposium on High Performance Computer Architecture (HPCA-6)*, pages 145–155, January 2000.
41. Ashwini K. Nanda, Anthony-Trung Nguyen, Maged M. Michael, and Douglas J. Joseph. High-throughput coherence control and hardware messaging in everest. *IBM Journal of Research and Development*, 45(2):229–244, March 2001.
42. B. O'Krafka and A. Newton. An empirical evaluation of two memory-efficient directory methods. In *Proc. of the 17th Int. Symposium on Computer Architecture*, pages 138–147. IEEE/ACM, June 1990.
43. SCI. Ieee standard for scalable coherent interface (sci). IEEE Standard 1596-1992, August 1993.
44. R. Simoni. *Cache Coherence Directories for Scalable Multiprocessors*. PhD thesis, 1992.
45. The BlueGene/L Team. An overview of the bluegene/l supercomputer. *Proc. of the Int'l SC2002 High Performance Networking and Computing*, November 2002.
46. Radhika Thekkath, Amit Pal Singh, Jaswinder Pal Singh, Susan John, and John L. Hennessy. An evaluation of a commercial cc-numa architecture - the convex exemplar spp1200. In *Proc. of the IPPS*, pages 8–17, June 1997.
47. Steven C. Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder P. Singh, and Anoop 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.
48. Kenneth C. Yeager. The mips r10000 superscalar microprocessor. *IEEE Micro*, 16(2):28–40, April 1996.