(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 713 579**

(21) Número de solicitud: 201731343

(51) Int. CI.:

G06F 12/0815 (2006.01)

(12) SOLICITUD DE PATENTE A1

(22) Fecha de presentación:

21.11.2017

(43) Fecha de publicación de la solicitud:

22.05.2019

(71) Solicitantes:

UNIVERSIDAD DE CANTABRIA (100.0%)
Avda. de los Castros, s/n
39005 Santander (Cantabria) ES

(72) Inventor/es:

GREGORIO MENEZO, Lucía;
PUENTE VARONA, Valentín y
GREGORIO MONASTERIO, José Ángel

(74) Agente/Representante:

PONS ARIÑO, Ángel

(54) Título: **SISTEMA Y MÉTODO DE MANTENIMIENTO DE COHERENCIA CACHÉ EN ARQUITECTURAS MULTIPROCESADOR Y MULTINÚCLEO**

(57) Resumen:

Sistema y método de mantenimiento de coherencia caché en arquitecturas multiprocesador y multinúcleo. Se describe un sistema y un método que permiten mantener la coherencia caché en arquitecturas multiprocesador y multinúcleo mediante gestión de una serie de metadatos asociados a cada bloque de datos, de forma jerarquizada a nivel de núcleo, chip y sistema; denominados tokens. Para llevar a cabo el objeto de la invención, se implementa, asociado al último nivel de cache compartido en cada chip (LLC) una estructura D|F-LLC compuesta por un directorio y un filtro que contienen información sobre los bloques que están en la caches privadas de ese chip. Asimismo, asociado a cada controlador de memoria de cada chip, se implementa una estructura similar D|F-MEM con información sobre los bloques que están siendo utilizados por los diferentes chips.

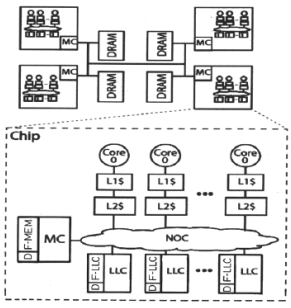

FIG. 1

ES 2 713 579 A1

SISTEMA Y MÉTODO DE MANTENIMIENTO DE COHERENCIA CACHÉ EN ARQUITECTURAS MULTIPROCESADOR Y MULTINÚCLEO

D E S C R I P C I Ó N

OBJETO DE LA INVENCIÓN

El objeto de la presente invención se enmarca en campo técnico de las tecnologías de computación.

Más concretamente, la presente invención va dirigida al campo de los sistemas de mantenimiento de la coherencia de memoria caché, en adelante caché, en los sistemas multiprocesador y multinúcleo (multicore por su terminología en inglés) compuestos por varios chips (*Chips Multi Processor o CMP*).

ANTECEDENTES DE LA INVENCIÓN

Las infraestructuras de computación de alto rendimiento típicamente emplean placas multi-zócalo para incrementar la densidad de computación y la cantidad de recursos. En muchos casos, todos los procesadores pueden acceder todo el espacio de memoria y ello permite que los programadores de cualquier aplicación (HPC, nube, *web*, etc.) puedan emplear un paradigma de memoria compartida lo que suministra transparencia y flexibilidad. Sin embargo, el empleo de este paradigma exige la introducción de mecanismos para mantener la coherencia de las distintas copias alojadas en las caches privadas de los distintos núcleos. El mantenimiento de esta coherencia en los sistemas *multi-socket* o multi-CMP se ha convertido en un auténtico desafío. Debe mantenerse la coherencia dentro de cada CMP y entre los diferentes CMPs y ello genera un acceso no uniforme a memoria que afecta al rendimiento de las aplicaciones. Aunque hay numerosos mecanismos para tratar de minimizar la distancia desde el procesador hasta los datos, el mecanismo de coherencia debe ser transparente a este efecto.

Hay un gran número de protocolos de coherencia para sistemas multi-núcleo. Las primeras soluciones para CMPs estaban basadas en el uso de directorios con suficiente capacidad como para almacenar toda la información sobre los bloques temporalmente almacenados en las caches privadas. Sin embargo, la elevada asociatividad que exige esta solución limita

gravemente su escalabilidad, pero reducir esta escalabilidad daña gravemente el rendimiento de las aplicaciones.

Para reducir el coste del directorio algunas soluciones proponen hacer la codificación de los compartidores más flexible como la propuesta en *D. Sanchez and C. Kozyrakis, "SCD: A scalable coherence directory with flexible sharer set encoding," in International Symposium on High Performance Computer Architecture (HPCA), 2012, pp. 1–12,* eliminar la sobrecarga que suponen los tags de los bloques como se propone en *J. Zebchuk, V. Srinivasan, M. K. M. K. Qureshi, and A. Moshovos, "A tagless coherence directory," in International Symposium on Microarchitecture (MICRO), 2009, pp. 423–434*, emplear la semántica de las aplicaciones como se describe en *B. A. Cuesta, A. Ros, M. E. Gómez, A. Robles, and J. F. Duato, "Increasing the effectiveness of directory caches by deactivating coherence for private memory blocks," in Int. Symposium on Computer Architecture (ISCA), 2011, p. 93*, en *S. Demetriades and S. Cho, "Stash Directory: A Scalable Directory for Many-Core Coherence," in Int. Symp. on High Perf. Computer Architecture (HPCA), 2014.* o en *L. G. Menezo, V. Puente, and J. A. Gregorio, "The case for a scalable coherence protocol for complex on-chip cache hierarchies in many-core systems," in International Conference on Parallel Architectures and Compilation Techniques (PACT), 2013, pp. 279–288.,* o incluso emplear aproximaciones basadas en difusión o broadcast cuando el número de cores no es demasiado alto, como en los procesadores comerciales como en *P. Hammarlund, A. J. Martinez, A. A. Bajwa, D. L. Hill, E. Hallnor, H. Jiang, M. Dixon, M. Derr, M. Hunsaker, R. Kumar, R. B. Osborne, R. Rajwar, R. Singhal, R. D'Sa, R. Chappell, S. Kaushik, S. Chennupaty, S. Jourdan, S. Gunther, T. Piazza, and T. Burton, "Haswell: The Fourth-Generation Intel Core Processor," IEEE Micro, vol. 34, no. 2, pp. 6–20, 2014* o en P. Conway, N. Kalyanasundharam, G. Donley, K. Lepak, and B. Hughes, "Cache Hierarchy and Memory Subsystem of the AMD Opteron Processor," IEEE Micro, vol. 30, no. 2, pp. 16–29, Mar. 2010.

En estas aproximaciones, basadas en técnicas de difusión conocidas como *broadcast*, para minimizar las comunicaciones y número de procesos de determinación de presencia de un determinado dato (conocidos por su denominación anglosajona *snoops)*, algunos trabajos proponen el uso de filtros como es el caso del ya mencionado *P. Conway, N. Kalyanasundharam, G. Donley, K. Lepak, and B. Hughes, "Cache Hierarchy and Memory Subsystem of the AMD Opteron Processor," IEEE Micro, vol. 30, no. 2, pp. 16–29, Mar. 2010* o como ocurre en *A. Moshovos, "RegionScout: Exploiting Coarse Grain Sharing in Snoop-Based Coherence," in International Symposium on Computer Architecture (ISCA), 2005, pp.*

234–245, otros documentos conocidos describen cómo adaptar el comportamiento del protocolo a la anchura de banda disponible o dar soporte, sobre la propia red de interconexión, para el proceso de broadcast o de agrupación (conocido como *gathering*) como es el caso de *E. Jerger, L. S. L.-S. Peh, M. H. M. H. Lipasti, and N. D. Enright Jerger, "Virtual tree coherence: Leveraging regions and in-network multicast trees for scalable cache coherence," Int. Symp. Microarchitecture, pp. 35–46, Nov. 2008* o *T. Krishna, L.-S. Peh, B. M. Beckmann, and S. K. Reinhardt, "Towards the ideal on-chip fabric for 1-to-many and many-to-1 communication," in International Symposium on Microarchitecture (MICRO), 2011, vol. 2, pp. 71–80.*

Sin embargo, el número de trabajos dedicados a los sistemas multi-CMP, que es la estructura sobre la que se le aplica esta invención, es mucho más reducido. Por ejemplo, la metodología conocida como *Token-CMP* detallada en *M. R. Marty, J. D. Bingham, M. D. Hill, A. J. Hu, M. M. K. Martin, and D. A. Wood, "Improving Multiple-CMP Systems Using Token Coherence," in 11th International Symposium on High-Performance Computer Architecture (HPCA), 2005, pp. 328–339* extiende el protocolo de coherencia basado en tokens descrito en *M. M. K. Martin, M. D. D. Hill, and D. A. Wood, "Token Coherence: Decoupling Performance and Correctness," in 30th International Symposium on Computer Architecture (ISCA), 2003, pp. 182–193* a este tipo de estructura, adaptando su versión *TokenB*. Sin embargo, su problema de inanición (denominada en el estado del arte como *starvation*) no solo sigue existiendo, sino que puede empeorar, al necesitar dos mecanismos de requerimiento persistente.

Las caches distribuidas en los sistemas basados en el Haswell-EP se mantienen coherentes empleando un protocolo basado en determinación de presencia de un determinado dato (*snooping)*. En concreto, dos extensiones del protocolo MESIF; este protocolo almacena dos bits de información de directorio por cada línea de cache, en los bits ECC de memoria. Esto es como un filtro básico para limitar el número de *snoops* enviados.

Por ejemplo, el procesador AMD *Opteron* emplea el denominado *HypTransport Assist* (*probe filte*r) que evita un gran número de mensajes de *broadcast* del protocolo. Para ello emplea una parte de significativa de la cache de tercer nivel, como un directorio disperso (*sparse*) para guardar información sobre todos los bloques de datos presentes en las caches privadas del chip correspondiente. Sin embargo, si el directorio se llena, y debe sustituirse alguna entrada, deberá ser invalidada cualquier copia del bloque cuya información se guardaba en la entrada eliminada y ello genera una caída significativa del rendimiento.

El IBM Power 8 emplea un protocolo basado en snooping no bloqueante y los controladores responden en un tiempo fijo, empleando multiplexado en el tiempo. Para evitar pérdida de slots, su sistema de interconexión introduce la posibilidad de suscribirse en exceso (efecto conocido por su nomenclatura anglosajona *oversubscribe*) a la asignación de ancho de banda de cada chip del procesador. Los sistemas construidos con Power 8 pueden tener hasta 192 cores, agrupados en 4 grupos de 4 chips cada uno y conectados como “todos-con-todos”. Cada línea de cache tiene tres ámbitos: chip, grupo y sistema. Así, mediante técnicas de predicción se trata de limitar el ámbito en el que se realizan las operaciones de brodcasts. Cuando se produce un fallo en la predicción, se aumenta el ámbito de búsqueda hasta que se alcanza el bloque requerido. En cualquier caso, este protocolo de coherencia está fuertemente ligado a las características específicas del procesador y su red de interconexión por lo que es difícilmente extensible a otras arquitecturas.

A la vista de lo anterior, se hace necesario solucionar de forma adecuada el problema que plantea el mantenimiento de la coherencia en los sistemas multi-CMP. Los principales problemas existentes actualmente provienen de dos hechos de difícil coexistencia: por un lado, la cantidad de memoria necesaria para almacenar información para la localización de cada bloque presente en cada cache privada de cada núcleo (*comúnmente* denominado por su terminología inglesa *core*) del sistema es demasiado elevada. El problema es que, si se reduce, las soluciones actuales imponen la necesidad de invalidar los bloques correspondientes a la entrada que debe ser eliminada del directorio. Como estos bloques, en general, aún serán empleados por aquellos núcleos en los que se encontraban, se producirán nuevos fallos de cache que no sólo afectarán negativamente al rendimiento, sino que provocarán nuevas necesidades de entradas en el directorio, lo que provocará nuevas invalidaciones.

En el otro extremo se encuentran las soluciones basadas en la búsqueda de los bloques a través de difusión (definida como *broadcast*), preguntando a todas las cachés del sistema sobre la presencia o no de tales bloques. El problema con estas soluciones es que requieren cantidades prohibitivas de anchura de banda de los sistemas de interconexión y un consumo elevado de energía por la gran cantidad de tests de comprobación de la presencia (o no) del bloque de datos buscado (técnica conocida como *snooping/ snoops* como se ha indicado anteriormente).

Los problemas de ambas soluciones se exacerban aún más en los sistemas multi-CMP. La estructura de directorio no solo crece con el número de chips o CMPs, sino que además

debe almacenar información sobre los bloques situados en las memorias compartidas por todos los núcleos de cada CMP y estas memorias suelen tener una capacidad mucho más elevada que las correspondientes a las caches privadas.

Asimismo, en este tipo de sistemas, las soluciones basadas en *broadcast* (difusión) sufren el enorme aumento de la “distancia” a los datos. Así como en el interior del chip es factible una elevada anchura de banda en la red de interconexión, no es el caso con la interconexión entre chips, donde es mucho más costosa.

DESCRIPCIÓN DE LA INVENCIÓN

Más concretamente, la invención se refiere a la arquitectura hardware y los correspondientes procesos para su gestión necesarios para mantener coherentes los datos distribuidos por las cachés privadas de los distintos núcleos (*cores*) de la arquitectura multichip-procesador. Núcleos a los que se encuentran respectivamente asociados una memoria caché de nivel (L1, L2, L3, …..Ln), una memoria caché de último nivel (LLC), y un controlador de memoria (MC).

El sistema objeto de la invención se basa en dos estructuras, D|F-LLC y D|F-MEM respextivamente asociadas al último nivel de cache y al controlador de memoria (MC), respectivamente. Cada una de estas estructuras D|F-LLC y D|F-MEM está dividida en dos partes: una estructura directorio (D) y una estructura de filtro (F). Así un directorio de cache de último nivel (DLLC) de la D|F-LLC indica dónde en el sistema se encuentra un determinado bloque de datos, si nadie ha eliminado su entrada correspondiente (eliminado por otro que coincide en la misma entrada cuando comienza a ser compartido y va en la misma entrada que otro que ya estaba siendo compartido). Un filtro asociado a la caché de último nivel (F-LLC) se encarga de verificar la presencia (o no) de un bloque en el interior del chip del CMP

De manera similar, la invención añade una estructura asociada al controlador de memoria (D|F-MEM) compuesta por un directorio (D-MEM) que indica qué chips comparten un bloque da datos (también si nadie ha eliminado su entrada) y un filtro (F-MEM) que indica si ese dato ha salido de la memoria DRAM del sistema hacia algún chip.

En cualquier multi-CMP, un bloque puede ser privado a un chip o compartido entre varios chips. Al mismo tiempo, dentro de cada chip, ese bloque puede ser privado a un núcleo o

compartido entre otros. Esta diferencia privada y compartida se usa en para optimizar las estructuras que respectivamente comprenden el directorio y filtro de caché de último nivel (D|F-LLC) y el directorio y filtro del controlador de memoria (D|F-MEM).

En la Figura 2 se muestra el esquema básico que representa ambas estructuras. Lo que se denomina en dicha figura “parte compartida”, es lo que se está denominando directorio D, y lo que se denomina “parte privada”, es lo que corresponde a la estructura de filtro F.
La parte de directorio, en ambas estructuras, estarán dedicadas a la localización de algunos de los bloques compartidos en el sistema. La parte de filtro, en ambas estructuras, contendrá información sobre la presencia de los bloques que están en las caches privadas, pero no se están compartiendo, es decir, bloques privados. En ambas estructuras, estas dos partes funcionan de la misma manera, aunque la información que guardan se refiere a los chips en la D|F-MEM o a los núcleos dentro del chip en el D|F-LLC.

La primera parte es la habitual de cualquier directorio en la que cada entrada incluye la etiqueta de dirección y un bit para cada uno de los compartidores que tienen una copia de ese bloque de datos (los participantes serán núcleos en el D-LLC y chips en el D-MEM). No obstante hay dos características poco habituales en este tipo de estructura:

- No se utilizará ninguna de estas entradas para los bloques privados. Es decir, el directorio solo se empleará para guardar información sobre los bloques compartidos
- Otro aspecto importante es que estas estructuras no son inclusivas en relación con el contenido de caches. Esto significa que, si alguna de las entradas tiene que desalojarse del directorio, debido a un conflicto, esto puede hacerse sin tener que invalidar todas las copias a las que hace referencia la entrada, es decir, los bloques de datos presentes en las memorias cache privadas.

El costo a pagar por esta optimización es que se hace necesario usar un mensaje de difusión cuando estas estructuras no tienen información sobre un bloque de datos solicitado de nuevo. A cambio, el tamaño del directorio se puede reducir para que solo contenga los bloques que están siendo altamente compartidos (entre los núcleos en el caso de la D|FLLC y entre los chips en D|MEM.

La segunda parte de cada estructura, la parte F, está dedicada a almacenar información sobre la presencia o no de un bloque de datos en cualquier núcleo (F-LLC) o en cualquier chip (F-MEM). Esto persigue dos objetivos: reducir considerablemente el espacio de almacenamiento necesario, pero filtrando aquellos broadcasts innecesarios que se enviarían

sin encontrar ningún bloque de datos (intra-chip en el D-LLC e inter-chip en el D-MEM). Para detectar la presencia de bloques, se puede usar una estructura d-left Counting Bloom Filter (dlCBF), que usa filtros de conteo de Bloom modificados (CBF) capaces de duplicar la eficiencia de los *Counter Bloom Filters* convencionales y con un costo de implementación similar. Cualquier otro tipo de filtro eficiente es compatible con la invención. El filtro empleado como ejemplo funciona de la siguiente manera: la tabla se divide en d sub-tablas. Cada sub-tabla está dividida en b *buckets* (recipientes) y cada *bucket* está, a su vez, dividido en múltiples celdas. Estas celdas incluyen unos pocos bits para almacenar una firma de la dirección, llamada resto, *r*, y un pequeño contador de colisiones *c*. Este contador sólo se utiliza para el caso inusual en el que direcciones diferentes dan lugar a la misma firma (en la práctica 3 o 4 bits bastan).

Para una determinada dirección, para calcular el *bucket* Bi y el resto r, aplicamos una función hash convencional $H(\cdot)$ a esa dirección, obteniendo $log2b + r\ bits$ y a continuación aplicamos d permutaciones a este valor $\Pi i(H(x))$. Cada una de estas permutaciones dará como resultado un par (Bi, ri) correspondiente a la sub-tabla i. Después de calcular el *bucket* para cada sub-tabla se asigna una entrada en el *bucket* con menos elementos o, si tienen el mismo número de entradas, el situado más a la izquierda (de ahí el nombre: *d-left*).

Con esta estructura de filtrado podemos rastrear todas las etiquetas (*tags*) que se mapean en sus entradas. En el caso de F-LLC, cada vez que un bloque llega por primera vez a una cache privada del chip, es necesario calcular la correspondiente firma o resto del dlCBF. Cuando el bloque de datos deja de estar presente en todas las caches privadas del chip, el dlCBF debe ser actualizado. Debe indicarse que, si un bloque sólo está presente en la LLC, el dlCBF no contendrá su información.

Este tipo de estructura se incluye, tanto en F-LLC como en F-MEM. En ambos casos, su función es determinar la presencia, o no, de un bloque de datos dentro del chip empleando la estructura de F-LLC o fuera de la memoria DRAM empleando la estructura de F-MEM.

Esta división en dos partes de cada una de las estructuras (D|F-LLC y D|F-MEM) es una de las partes esenciales de la aportación porque permiten incrementar la escalabilidad del sistema, al poder emplear con efectividad un directorio muy pequeño que crece menos que O(N) con el número de núcleos N, y con un número de *broadcasts* que crece menos que exponencialmente con N. Estas dos partes, como ser verá más adelante, permiten jerarquizar el protocolo de coherencia, desde el nivel de núcleo, chip y sistema.

Como se ha señalado, una parte de cada una de las estructuras D|F-LLC y D|F-MEM corresponde a directorios tradicionales. En el D-LLC necesitaremos espacio para almacenar el *tag* (etiqueta) y un bit para cada uno de los núcleos dentro del CMP. Cualquier otro tipo de representación de los compartidores es compatible con la presente propuesta. Como se sabe, la asociatividad necesaria del directorio para evitar expulsiones debido a los conflictos no escala con el número de núcleos del sistema. La solución adoptada en nuestra propuesta es, por un lado, limitar la información únicamente a la de los bloques compartidos y por otro, eliminar su propiedad de inclusividad (expulsar una entrada del directorio no implica la invalidación de los bloques presentes en las caches privadas). Ambas decisiones permiten que el directorio únicamente necesite espacio para almacenar información de los bloques altamente compartidos.

Este aspecto es aún más importante para el caso de la estructura D|MEM porque, aunque el número de bits necesarios para mantener información de compartición entre chips es, en general, menor que entre núcleos (hay menos chips que cores), en un chip esta estructura debe incluir también los bloques que han salido de memoria y se encuentran almacenados en las caches de ultimo nivel (LLC) de cada chip y cuya capacidad normalmente es mucho mayor que L1 y L2. Por ello, en esta estructura es aún más importante limitar la información de, únicamente, los bloques altamente compartidos entre varios chips.

Por otro lado, el tamaño utilizado para cada dlCBF (*d-left Counting Bloom Filter*) en el F-LLC y en el F-MEM será lo más pequeño posible, pero capaz de manejar un porcentaje de falsos positivos (el sistema indica que un bloque está presente y no es cierto) que no dañe el rendimiento significativamente. Para una estructura del dlCBF, la probabilidad de falsos positivos, con una configuración de d subtablas y b *buckets* cada uno, r bits para el resto y n bloques privados mapeados en una entrada. Por ejemplo, suponiendo 4 subtablas, 8 celdas por *bucket* y un uso de 3/4 por *bucket*, para lograr una probabilidad de falso positivo del 5%, el número de bits para la firma o resto es 9. Considerando un contador de 3 bits para colisiones de firmas y el uso de cada *bucket* se necesitan aproximadamente 16 bits por elemento rastreado. Esto es menos de la mitad del número requerido por el filtro de Bloom convencional. No obstante, salvo en el espacio requerido para la implementación física, la invención es completamente compatible con cualquier otra forma de filtro capaz de detectar la presencia o no de bloques de datos en un chip o en el sistema.

Para poder facilitar la gestión y mantenimiento jerárquico del invariante “un solo escritor, múltiples lectores” la invención asocia, a cada copia de cada bloque de datos del sistema, tres tipos de “testigos” (del inglés *tokens*) que habilitan o deshabilitan a los núcleos para realizar ciertas operaciones sobre dichos bloques de datos. Los tres tipos de testigos son:

- Testigo de oro. Por cada bloque de datos, solo puede existir una copia del bloque en todo el sistema, que tenga asociado o posea este tipo de testigo. Por ello, este es un tipo de testigo a nivel de sistema.
- Testigos de plata. Por cada bloque de datos, pueden existir hasta n copias del bloque, donde n es el número de chips, que contengan este tipo de testigo. Por ello, este es un tipo de testigo a nivel de de chip.
- Testigo de bronce. Por cada bloque de datos, pueden existir hasta m copias del bloque, donde m es el número de núcleos, que contengan este tipo de testigo. Por ello, este es un tipo de testigo a nivel de núcleo.

La implementación física de los tres tipos de testigos se realiza mediante el añadido de tres conjuntos de bits a los ya habitualmente existentes en la representación software de los bloques de datos en caches, tales como etiquetas (tags), permisos, etc.

DESCRIPCIÓN DE LOS DIBUJOS

Para complementar la descripción que se está realizando y con objeto de ayudar a una mejor comprensión de las características de la invención, de acuerdo con un ejemplo preferente de realización práctica de la misma, se acompaña como parte integrante de dicha descripción, un juego de dibujos en donde con carácter ilustrativo y no limitativo, se ha representado lo siguiente:

Figura 1.- Muestra un diagrama donde se aprecia la arquitectura básica del sistema sobre el que se aplica la invención. Varios chips multiprocesadores (CMPs) interconectados. Cada uno compuesto por varios núcleos interconectados a través de una red de interconexión (NOC) y cada uno de ellos con dos niveles de cache L1 y L2, y un tercer nivel denominado Last Level Cache (LLC), dividido en bancos para mejorar el rendimiento y asociados con ella se encuentra la estructura D|F-LLC descrita en el texto. Finalmente, cada CMP tiene uno o varios controladores de memoria y asociado a ellos se encuentra la estructura D|F-MEM.

Figura 2.- Muestra un diagrama donde se aprecia un esquema general de las estructuras correspondientes a cualquiera de las D|F-LLC o D|F-MEM. La parte compartida (*shared part*)

corresponde a un directorio convencional donde cada una de las entradas almacena información sobre la dirección (*tag*) de un bloque datos, los compartidores e información sobre los *testigos* asociados a ese bloque. Estas entradas únicamente existen para aquellos bloques que están siendo compartidos por dos o más núcleos (o dos o más chips si la estructura es D-MEM). Es decir, aquí no se almacena información sobre los bloques presentes en un solo núcleo. La otra parte, la parte privada (*Private part*) almacena la presencia o no de bloques de datos mediante el filtro *D-left* descrito en el texto. En ella, básicamente se almacena una firma o resto de cada uno de los bloques que se encuentran dentro del chip (o han salido de DRAM si la estructura es F-MEM). Esta parte de la estructura evita la necesidad de la mayor parte de las difusiones para encontrar un bloque de datos.

Figura 3.- Muestra una representación de alto nivel de la estructura del filtro que puede ser usado para implementar el objeto de la invención en una posible realización del mismo, siendo este un dlCBF (*D-Left Counting Bloom Filter*).

Figura 4.- Muestra una representación de la cronología temporal de las acciones que tienen lugar ante un fallo de cache de lectura, donde los intervalos no están a escala temporal.

Figura 5.- Muestra una representación de la cronología temporal de las acciones que tienen lugar ante un fallo de cache de escritura, donde los intervalos no están a escala temporal.

REALIZACIÓN PREFERENTE DE LA INVENCIÓN

En una realización preferente del objeto de la invención se tiene un sistema como el representado en la figura 1 donde se observa una arquitectura multiprocesador y multinúcleo, núcleos a los que se encuentran respectivamente asociados una memoria caché de nivel (L1, L2), una memoria caché de último nivel (LLC), y un controlador de memoria (MC), a los que se encuentran asociados dos estructuras: una estructura asociada a la caché de último nivel (D|F-LLC) y una estructura de memoria (D|F-MEM), respectivamente. La estructura asociada a la cache de último nivel (D|F-LLC), comprende un directorio (D-LLC), configurado para determinar qué núcleos, dentro del chip, comparten un determinado dato, y un primer filtro (F-LLC), configurado para determinar la presencia, o no, dentro del chip correspondiente, de un determinado bloque de datos; mientras que la estructura de memoria (D|F-MEM), que está asociada a cada controlador de memoria (MC), comprende: un directorio de memoria (D-MEM) configurado para determinar qué chips comparten un bloque de datos, y un segundo filtro (F-MEM) configurado para determinar si

ese bloque dato ha salido, o no, de la memoria principal o memoria del sistema, por ejemplo DRAM, hacia algún chip.

En una posible realización del objeto de la invención, se opta por filtros basados en una variación de un filtro tipo *Bloom*. Observando la figura 3 se aprecia el funcionamiento del filtro viendo que el filtro se divide en subtablas donde, a su vez, cada subtabla se divide en b recipientes y cada recipiente en varias celdas; estando cada celda compuesta por dos conjuntos de bits. Uno de los conjuntos de bits es una firma de la dirección, r, que se obtiene aplicando una función *hash* convencional a la dirección del bloque de datos correspondiente y quedándose con un número de bits igual log2b+r- es decir, en una posible realización en la que el número de recipientes es 4, pues el número de bits de la firma es 2+r, siendo r el valor necesario para obtener un valor medio de falsos positivos de un determinado valor. A esos valores obtenidos se le aplican tantas permutaciones como subtablas obteniendo como resultado un par que comprende: (*bucket*, resto). De ese par solo se almacena el resto en el recipiente que haya resultado el par. Como hay el mismo recipiente en todas las subtablas (por ejemplo, si resulta el *bucket* 2, pues puede ser el 2 de la subtabla 1 de la 2, etc..). Se escoge el *bucket* menos lleno situado más a la izquierda para guardar ese resto.

En cualquier multi-CMP, un bloque puede ser privado a un chip o compartido entre varios chips. Al mismo tiempo, dentro de cada chip, ese bloque puede ser privado a un núcleo o compartido entre otros. Esta diferencia privada y compartida se usa en nuestro protocolo para optimizar las estructuras dividiendo cada una de ellas en dos partes diferentes, como se muestra en la Figura 2. Una de estas partes estará dedicada a la localización de algunos de los bloques compartidos en el sistema. La otra contendrá información sobre los bloques que están en las caches privadas, pero no se están compartiendo, es decir, bloques privados. En ambas estructuras, de caché de último nivel y de controlador de memoria, estas dos partes funcionan de la misma manera, aunque la información que guardan se refiere a los chips en la estructura asociada al controlador de memoria (MC) o a los núcleos dentro del chip en la asociada al último nivel de cache (LLC).

Puede darse el caso en el que con otra dirección (de otro bloque) al aplicar el *hash* y las permutaciones den lugar al mismo resto (estamos usando menos bits). Esto es muy poco probable, pero posible y debe ser evitado porque ello significaría que cuando sale un bloque de las cachés privadas y se elimine esa dirección, lo haríamos con los dos bloques cuyas direcciones colisionan. Para ello se pueden añadir bits a como de contador de colisión, *c*. Este proceso se realiza con cada bloque entra en las cachés privadas (o sale de la memoria

del sistema, depende del filtro) de algún núcleo. Así que cuando otro núcleo requiere un bloque no está en sus cachés privadas, a partir de su dirección se comprueba si coinciden los restos, tal y como se representa en la figura 2. Finalmente, si un bloque sale de las cachés privadas (o vuelve a la memoria del sistema) esa entrada debe ser eliminada del filtro. Se aplica el mismo procedimiento, se comprueba en qué *bucket* está, se disminuye uno el contador de colisión *c* y si éste es cero, se procede a eliminar esa entrada.

El objeto de la invención prevé asociar a cada uno de bloques de datos tres tipos de metadatos a los que denominamos indistintamente testigos o su terminlogía inglesa, *tokens,* siendo el *token* un número o valor, en forma de metadato, asociado a cada bloque de datos representado ciertos derechos-obligaciones de quien los posee; usando para distinguirlos una nomenclatura tal que identifique cada tipo de *token;* como por ejemplo: oro, plata y bronce (por denominación jerárquica se ha empleado una escala basada en metales preciosos a modo de ejemplo ilustrativo). De esta manera se tiene que un testigo denominado *token* “bronce” se refiere a números asociados a cada copia, de cada bloque de datos, cuya suma correspondiente a todos las copias existentes en un momento dado, es un valor constante, normalmente igual al número de núcleos del sistema. Si un núcleo posee un bloque de datos con ese número cuyo valor es > = 1, significa que puede leer ese dato. Un segundo testigo que denominamos *token* “plata” es otro número, también asociado a cada bloque de datos, que identifica el elemento encargado de repartir los *tokens* de bronce dentro del chip, hay uno por cada chip o lo que es lo mismo, la suma de estos es igual al número de chips. Y un tercer tipo de testigo es el denominado *token* “oro”, que identifica un elemento del sistema que reparte los testigos anteriores (token “plata” y “bronce”) teniendo de manera preferente un solo *token* “oro” por bloque de datos en todo el sistema. Para poder modificar un bloque de datos (por ejemplo mediante un comando de almacenamiento *store*), antes es imprescindible recopilar todos los *tokens*. Eso asegura que no hay ninguna copia con permiso de lectura .

Los *tokens* anteriores sirven para mantener la coherencia de cualquier bloque de cache en todo el sistema mediante gestión de las estructuras (D|F-LLC, D|F-MEM) y empleando las características de los testigos metadatos o *tokens*. Cada uno de los *tokens* tiene un ámbito de aplicación que es el que se controla mediante el método de la invención cuyos aspectos se detallan a continuación y que se pueden dividir en dos sub-funciones dependiendo de las acciones que se quieran llevar a cabo con los datos (lectura o escritura).

Si lo que se desea realizar es una operación de lectura de un determinado dato, el método de la invención comprende asegurar que ese dato (el bloque que lo contiene) tiene asociado, al menos un *token* bronce. Si lo tiene, puede realizarse la acción de lectura con garantía de que ningún otro núcleo del sistema está modificándolo mediante una escritura. Si no lo tiene, debe leer una entrada correspondiente en el directorio de la caché de ultimo nivel del chip (D-LLC) en el que se encuentre el núcleo que trata de leer el dato, y comprobar si ese bloque de datos está siendo compartido. Si esa entrada existe, indicará, de entre los núcleos compartidores, cuál es el que tiene el *token* de plata asociado, es decir el propietario, y por tanto al que hay que requerirle, al menos un *token* bronce para adquirir el permiso de lectura. Si la entrada no existe en el directorio (D-LLC), el protocolo debe encargarse de comprobar si ese bloque de datos está presente en el chip mediante el empleo del segundo filtro (F-LLC) asociado. Si lo está, se encargará de hacer un *broadcast* para que el propietario del *token* plata le envíe un *token* bronce. Si no lo está, el protocolo debe ir al controlador de memoria (MC) *home* del bloque de dato para realizar las mismas comprobaciones, pero a nivel de chip. Si no estuviese, ni el directorio de memoria (D-MEM) del controlador de memoria ni el primer filtro (F-MEM), deberá hacer un requerimiento a la memoria del sistema.

Por tanto, un fallo de lectura seguiría la cronología que se aprecia en la figura 4 asociada al proceso:

if shared_data *then*
    //entry in D-LLC
    request to the silver-token owner;
else if data_in_LLC *then*
    send data+tokens to the requestor;
else if block present in chip *then*
    //present in F-LLC
    broadcast;
    creation of new entry in D-LLC;
else
    // data out of the chip
    read-request to the D|F-MEM home;

Si lo que se desea es realizar una operación de escritura el método de la invención comprende recopilar todos los *tokens* (oro, plata y bronce). Para ello irá testeando, como en

el caso anterior, por los diferentes ámbitos, LLC, chip y por último sistema completo. Únicamente que ahora no requiere un *token* bronce, como en el caso de la lectura, sino todos los *tokens* que deben ser enviados al núcleo que ha requerido permiso para una escritura del dato. Es decir, se debe asegurar que el que hace el requerimiento de escritura posee todos los *tokens* antes de escribir, para que no haya ningún otro núcleo en el sistema haciendo una operación, ni de lectura ni de escritura con ese dato.

Un fallo de escritura seguiría la cronología que se aprecia en la figura 5 asociada al proceso:

```
if shared_data then
    //entry in D-LLC
    multicast to chip-sharers to invalidate;
    if all_tokens_are_present then
        complete the request;
    else
        send request to D|F-MEM home;
else if data_in_LLC_with all tokens then
    send data to requestor
else if block present in F-LLC then
    broadcast;
    creation of new entry in D-LLC;
    complete request;
else
  //data out of the chip
   write-request to the D|F-MEM home;
```

En definitiva, se realiza una gestión de los testigos (*tokens)*, de cada tipo dentro del ámbito correspondiente, para mantener el invariante: un solo escritor múltiples lectores (SWMR).

Puede considerarse también, como parte del método aquí descrito, la ordenación de la actualización de las entradas en cada una de las estructuras (D|F-MEM, D|F-LLC). Es decir: directorio asociado a la cache de último nivel (D-LLC), y el primer filtro (F-LLC) ; directorio de memoria (D-MEM) asociado al controlador de memoria (MC) de cada chip y el correspondiente segundo filtro (F-MEM); aunque estas son acciones convencionales de modificación de tablas y valores. Es decir, es el encargado de ordenar la acción de introducción de una entrada en la estructura correspondiente. Por ejemplo, cuando un dato

sale de la memoria del sistema, debe apuntarse, en el primer filtro (F-MEM) asociado al controlador de memoria (MC), que el bloque de datos ha salido de la memoria del sistema y se encuentran algún chip. Si ese bloque de datos vuelve a ser requerido por otro chip, se dirige la acción de introducir una nueva entrada en el directorio de memoria (D-MEM) de ese controlador de memoria (MC) indicando los chips que lo comparten. De la misma forma, se encargará de ordenar las acciones de inicialización y mantenimiento de las estructuras asociadas al último nivel de cache (D|F-LLC).

R E I V I N D I C A C I O N E S

1. Sistema de mantenimiento de coherencia cache en arquitecturas multiprocesador y multinúcleo, núcleos a los que se encuentran respectivamente asociados una memoria caché multi-nivel (L1, L2,… LN), una memoria caché de último nivel (LLC) compartida por todos los núcleos de cada chip, una memoria de sistema distribuida entre todos los chips y, al menos, un controlador de dicha memoria (MC) por cada chip, estando el sistema caracterizado porque comprende una estructura asociada a la caché de último nivel (D|F-LLC) y una estructura de memoria (D|F-MEM) respectivamente asociadas al último nivel de cache (LLC) y al controlador de memoria (MC), donde:
   - la estructura asociada a la caché de último nivel (D|F-LLC) comprende:
     i. un directorio asociado a la cache de último nivel (D-LLC), que comprende una serie de entradas, con etiqueta de dirección y un identificador de cada uno de los núcleos compartidores que tienen una copia del correspondiente bloque de datos, y
     ii. un primer filtro (F-LLC) destinado a indicar si dicho bloque de datos está o no dentro de alguna de las cachés privadas de algún núcleo en el interior del chip, y
   - la estructura de memoria (D|F-MEM) comprende:
     i. un directorio de memoria (D-MEM) asociado a cada controlador de memoria (MC) de cada chip, configurado para determinar qué chips comparten un bloque de datos, y
     ii. un segundo filtro (F-MEM) configurado para determinar si dicho bloque de datos ha salido, o no, desde la memoria del sistema hacia algún chip.

2. Sistema de mantenimiento de coherencia cache en arquitecturas multiprocesador y multinúcleo, según reivindicación 1 caracterizado porque el filtro es una estructura configurada para determinar la presencia de un bloque de datos en las memorias privadas de los núcleos de un chip, de manera preferente un *d-left Counting Bloom Filter* (dlCBF), que comprende filtros de basados en conteo de Bloom (CBF).

3. Método de mantenimiento de coherencia cache en arquitecturas multiprocesador y multinúcleo, que hace uso del sistema descrito en una cualquiera de las reivindicaciones 1 ó 2; estando el método caracterizado porque comprende asociar a

cada bloque de datos unos testigos correspondientes a tres tipos de metadatos jerarquizados a nivel de: sistema, chip y núcleo; de tal manera que:

- se asocia un metadato de nivel de sistema a cada bloque de datos, de tal manera, que solamente puede haber un único núcleo, en todo el sistema, que posea una copia de ese bloque de datos con ese testigo,
- se asocia, a cada bloque de datos, tantos metadatos de nivel de chip como chips componen el sistema, de tal manera que, como máximo, puede haber ese número de copias de un bloque de datos que contengan ese testigo, y
- se asocia, a cada bloque de datos, tantos metadatos de nivel de núcleo como núcleos haya en el sistema completo, de tal manera que todos los núcleos pueden tener una copia de cada bloque de datos.

4. Método de mantenimiento de coherencia cache en arquitecturas multiprocesador y multinúcleo, según reivindicación 3 caracterizado porque comprende que para que un núcleo lleve a cabo una operación de lectura de un bloque de datos, éste debe contener, al menos, un testigo de nivel de núcleo.

5. Método de mantenimiento de coherencia cache en arquitecturas multiprocesador y multinúcleo, según reivindicación 3 caracterizado porque comprende, para que un núcleo lleve a cabo una modificación mediante una operación de escritura de un bloque de datos, dotar a ese bloque de datos de todos los testigos correspondientes a los metadatos de los tres tipos asociados al bloque de datos.

6. Método de mantenimiento de coherencia cache en arquitecturas multiprocesador y multinúcleo según una cualquiera de las reivindicaciones 3 a 5, donde tras un fallo de acceso a caches privadas, estando el método caracterizado porque comprende determinar el núcleo o núcleos responsables de transferir el metadato o metadatos necesarios mediante las estructuras (D|F-LLC, D|F-MEM), cuando cada núcleo del sistema requiera una operación de lectura o de modificación de un bloque de datos .

7. Método de mantenimiento de coherencia cache en arquitecturas multiprocesador y multinúcleo según reivindicación 3, donde un bloque de datos no se encuentra en ninguna de las memorias caché de ninguno de los núcleos y es requerido por algún núcleo del sistema, estando el método caracterizado porque comprende:

- recibir en uno de los núcleos los testigos desde el controlador de memoria (MC) del chip correspondiente al *home* del bloque de datos, y

- enviar, desde el núcleo que ha recibido los testigos, una copia del bloque de datos con los testigos a aquellas partes del sistema que lo soliciten.

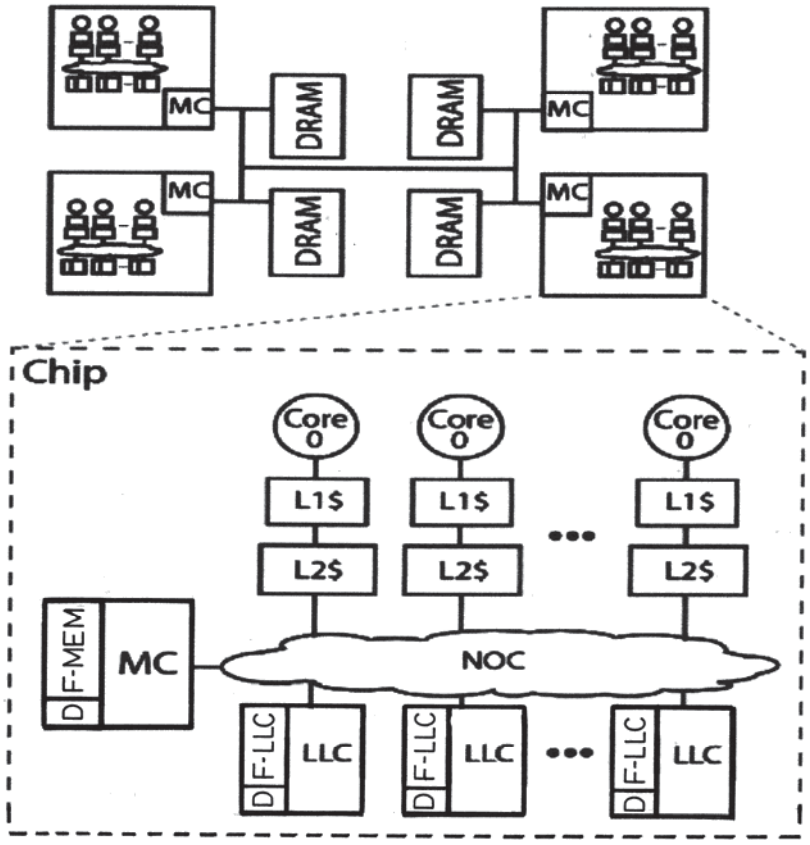
MC
DRAM
DRAM
MC
MC
DRAM
DRAM
MC
Chip
Core 0
Core 0
Core 0
L1$
L1$
L1$
L2$
L2$
L2$
D F-MEM
MC
NOC
D F-LLC
LLC
D F-LLC
LLC
D F-LLC
LLC

FIG. 1

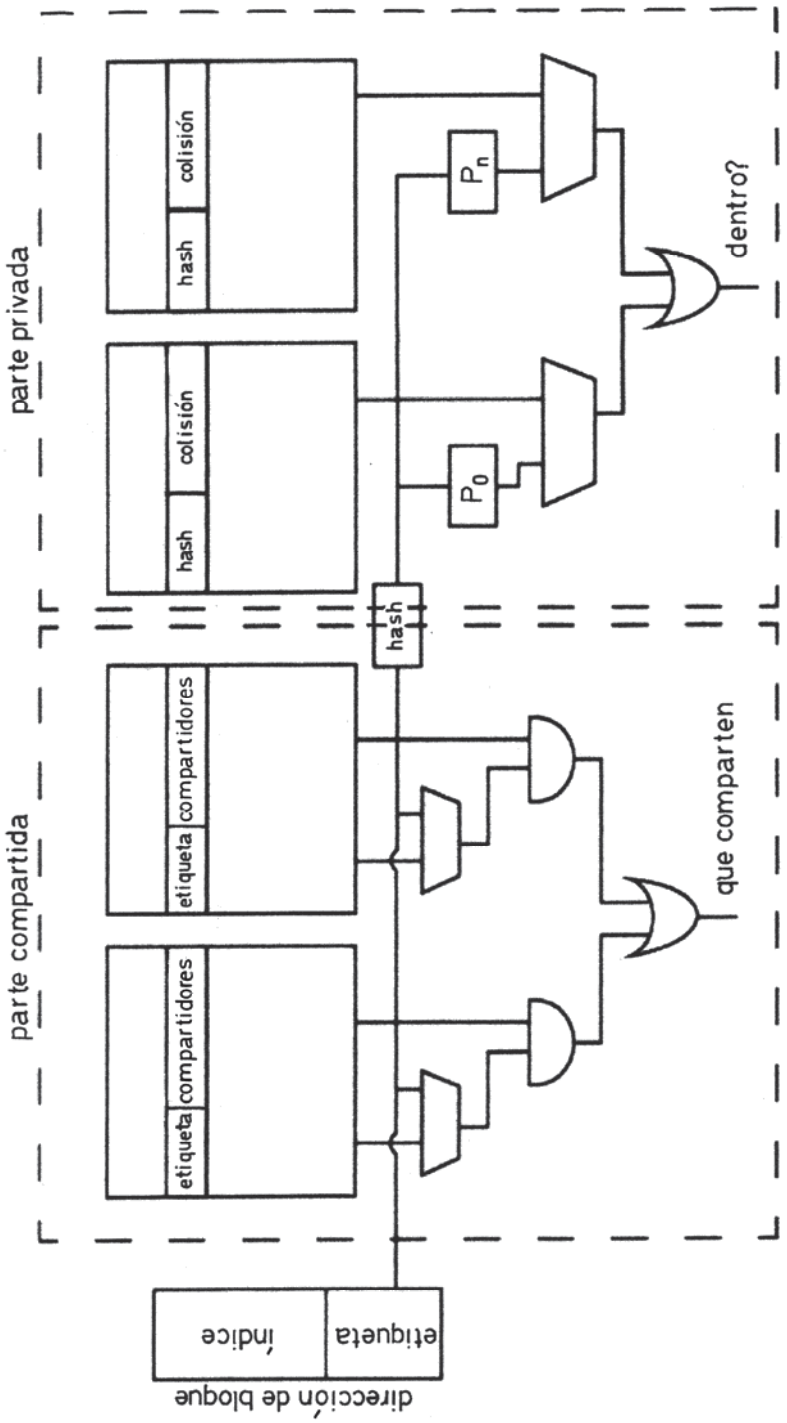
parte compartida
parte privada
etiqueta
compartidores
etiqueta
compartidores
hash
colisión
hash
colisión
hash
P_0
P_n
dentro?
que comparten
dirección de bloque
índice
etiqueta

FIG. 2

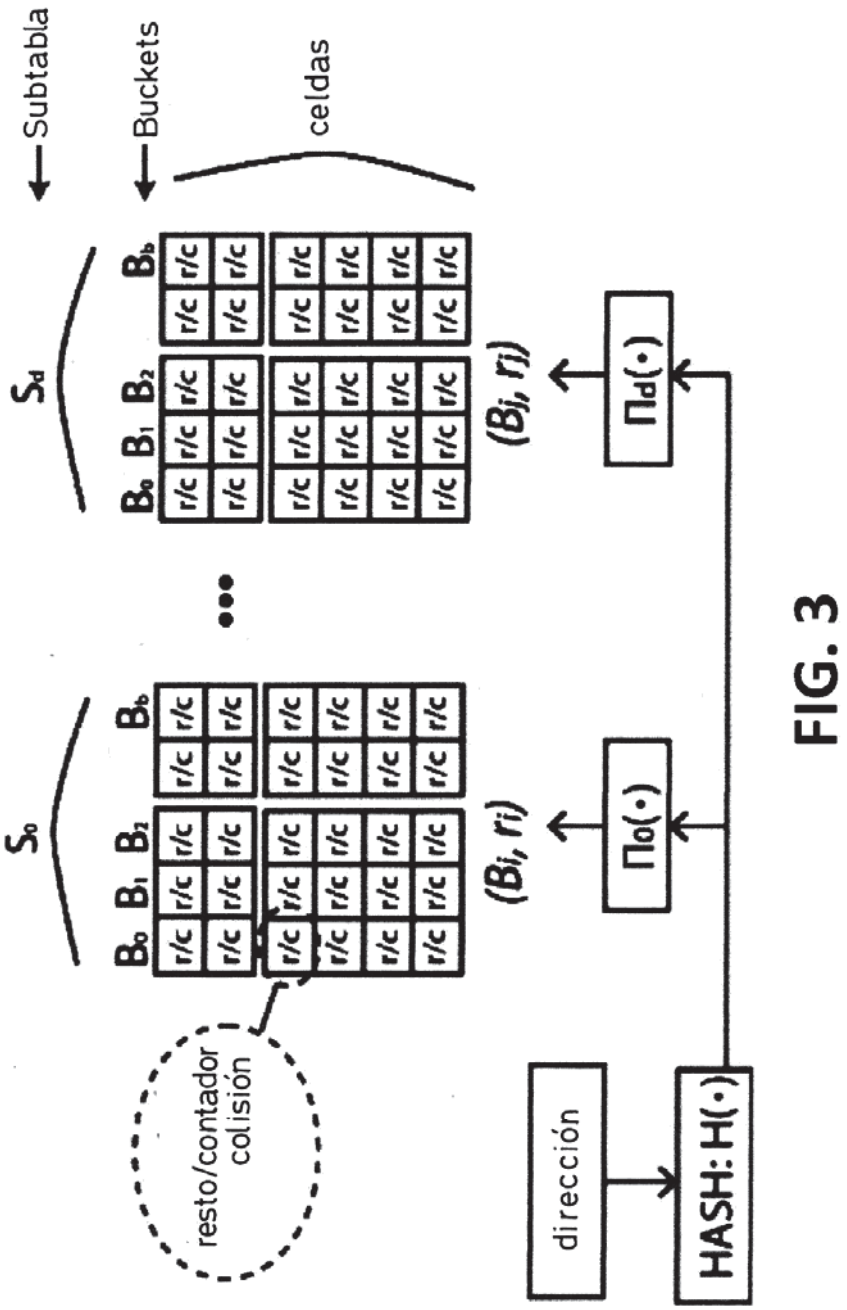
Subtabla
Buckets
celdas
S_0
S_d
B_0 B_1 B_2 B_b
r/c
(B_i, r_i)
$\Pi_0(\cdot)$
$\Pi_d(\cdot)$
resto/contador colisión
dirección
HASH: H(·)

FIG. 3

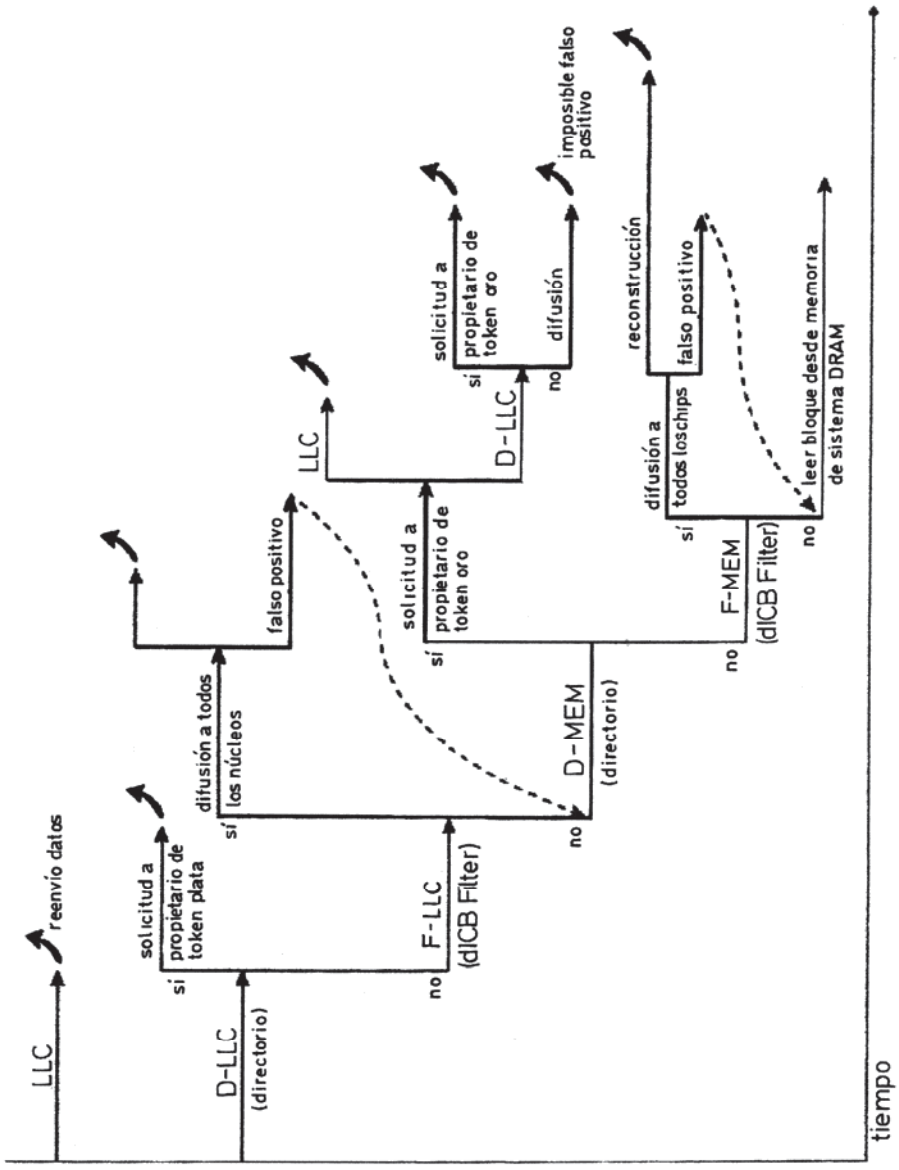
LLC
reenvío datos
D-LLC
(directorio)
solicitud a
propietario de
token plata
sí
no
F-LLC
(dICB Filter)
difusión a todos
los núcleos
sí
no
D-MEM
(directorio)
falso positivo
LLC
solicitud a
propietario de
token oro
sí
no
D-LLC
difusión
imposible falso
positivo
solicitud a
propietario de
token oro
sí
no
F-MEM
(dICB Filter)
difusión a
todos los chips
sí
no
falso positivo
reconstrucción
leer bloque desde memoria
de sistema DRAM
tiempo

FIG. 4

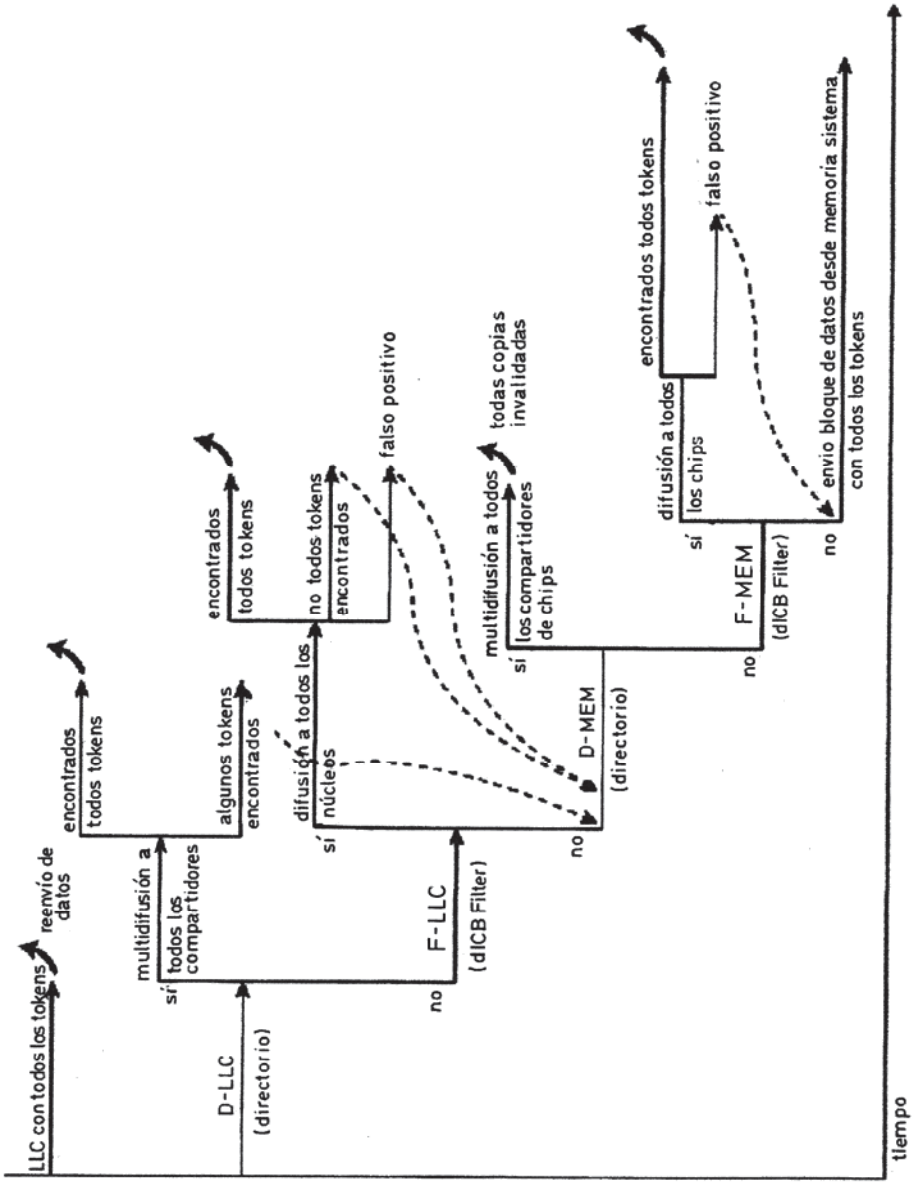
LLC con todos los tokens
reenvío de datos
D-LLC
(directorio)
multidifusión a
sí
todos los compartidores
encontrados
todos tokens
algunos tokens
encontrados
F-LLC
(dICB Filter)
no
difusión a todos los
sí
núcleos
encontrados
todos tokens
no todos tokens
encontrados
falso positivo
D-MEM
(directorio)
no
multidifusión a todos
sí
los compartidores
de chips
todas copias
invalidadas
F-MEM
(dICB Filter)
no
difusión a todos
sí
los chips
encontrados todos tokens
falso positivo
no
envio bloque de datos desde memoria sistema
con todos los tokens
tiempo

FIG. 5

OFICINA ESPAÑOLA
DE PATENTES Y MARCAS

ESPAÑA

(21) N.º solicitud: 201731343

(22) Fecha de presentación de la solicitud: 21.11.2017

(32) Fecha de prioridad:

INFORME SOBRE EL ESTADO DE LA TECNICA

(51) **Int. Cl. :** ***G06F12/0815*** (2016.01)

DOCUMENTOS RELEVANTES

Categoría	**(56) Documentos citados**	**Reivindicaciones afectadas**
Y	MARTY M R et al. Improving multiple-CMP systems using token coherence. Proceedings. 11th International Symposium on High-Performance Computer Architecture 2005 IEEE (Comput. Soc.) Los Alamitos, CA, USA. 30/11/2004, Páginas 328 - 339 [en línea][recuperado el 10/05/2018]. Recuperado de Internet <URL: http://research.cs.wisc.edu/multifacet/papers/hpca05_cmp_token.pdf>, ISSN ISBN 0-7695-2275-0. pág 1, col 1, líneas 6 - 10; col 2, líneas 36 - 37; pág 2, col 1, líneas 10 – 17, 32 - 35; col 2, líneas 34 - 35; pág 3, col 1, línea 1, 7 - 18; col 2, líneas 8 – 9, 13 – 15, 28 - 32; página 6, col 1, líneas 41 – 45, 49 - 50; figura 1,	1-7
Y	G. MENEZO et al. Flask Coherence: A Morphable hybrid coherence protocol to balance energy, performance and scalability. 2015 [en línea][recuperado el 10/05/2018]. Recuperado de Internet <URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7056033>, <DOI: 10.1109/HPCA.2015.7056033>. resumen; sección IV, A, col 2, líneas 21 - 25; sección III, A, 1, líneas 3 - 8; sección III, B, 1, líneas 26 - 33; párrafo [1]; sección III, B, 2, líneas 5 - 9;	1-2
Y	Scalable Cache Coherence. [en línea][Recuperado el 10/05/2018]. Recuperado de Internet <URL: http://web.archive.org/web/20100701000000*/https://people.engr.ncsu.edu/efg/506/s01/lectures/notes/lec18.pdf>. página 2, párrafo 2; página 3, párrafos 1 - 2; página 11, párrafo 4; página 12, párrafos 1 - 2;	3-7

Categoría de los documentos citados

X: de particular relevancia
Y: de particular relevancia combinado con otro/s de la misma categoría
A: refleja el estado de la técnica
O: referido a divulgación no escrita
P: publicado entre la fecha de prioridad y la de presentación de la solicitud
E: documento anterior, pero publicado después de la fecha de presentación de la solicitud

El presente informe ha sido realizado

☒ para todas las reivindicaciones ☐ para las reivindicaciones nº:

Fecha de realización del informe	**Examinador**	**Página**
10.05.2018	A. Oropesa García	1/2

Documentación mínima buscada (sistema de clasificación seguido de los símbolos de clasificación)

G06F

Bases de datos electrónicas consultadas durante la búsqueda (nombre de la base de datos y, si es posible, términos de búsqueda utilizados)

INVENES, EPODOC, internet