Gestión de los Reemplazos de Bloques Limpios en Protocolos de Coherencia Basados en Directorio

Ricardo Fernández-Pascual, Alberto Ros y Manuel E. Acacio [1]

Resumen— **Asegurar el mantenimiento de la coherencia de las cachés en arquitecturas con cientos o incluso miles de núcleos no es tarea sencilla. De entre las diversas alternativas propuestas hasta la fecha, los protocolos basados en directorio representan la solución más escalable, y gracias al conocimiento que sobre estos protocolos se ha ido acumulando durante los últimos cuarenta años, a día de hoy existe un consenso general sobre el impacto que la mayoría de sus aspectos de diseño tienen sobre el rendimiento, el consumo de energía y el coste. Sin embargo, a día de hoy existe aún un aspecto de diseño sutil pero importante, para el cual no hay unanimidad en los trabajos de investigación recientes. Más concretamente, mientras que algunos trabajos asumen una política de reemplazos *silenciosos* para bloques limpios expulsados del último nivel de cachés privadas, otros implementan para estos casos justo lo contrario, una política de reemplazos *ruidosos*, e incluso algunos otros trabajos ni siquiera mencionan la manera en la que se gestionan los reemplazos de bloques limpios. En este trabajo demostramos que este aspecto puede tener una influencia significativa sobre el rendimiento y el consumo de energía de un protocolo de coherencia basado en directorio. El utilizar la política de reemplazos *silenciosos* puede ahorrar una gran cantidad de tráfico de red (más de un 25 % en varios casos, 9.6 % de media) en comparación con el uso de la política de reemplazos *ruidosos*. Dado que en las arquitecturas multinúcleo con un gran número de núcleos que están por venir se espera que la red de interconexión dentro del chip represente una fracción importante del consumo de energía total, la utilización de la política de reemplazos *silenciosos* para bloques limpios implicará ahorros en consumo de energía no despreciables. Sin embargo, lo que es más importante es que hemos comprobado que en función de cómo se organice la estructura de directorio, el uso de la política de reemplazos *silenciosos* podría afectar o no al rendimiento, lo que indica que la asunción de la política de reemplazos *ruidosos* no está justificada siempre, ya que incrementaría de forma innecesaria el tráfico de red sin que ello supusiese ninguna ventaja en cuanto a rendimiento se refiere.**

Palabras clave— **Arquitecturas multinúcleo, protocolo de coherencia de cachés, reemplazos, bloques limpios, directorio, rendimiento, tráfico.**

I. Introducción

La práctica totalidad de las arquitecturas multinúcleo de propósito general actuales ofrecen como paradigma de programación de bajo nivel la abstracción de memoria compartida. El paradigma de memoria compartida se ha popularizado gracias a su cercanía con la programación secuencial, la facilidad de mapear determinados tipos de algoritmos y la amplia disponibilidad de aplicaciones basadas en POSIX Threads y OpenMP. Además, ha sido la abstracción usada tradicionalmente en la construcción de los sistemas operativos. De esta forma, a menos que suceda un cambio radial (algo realmente improbable), las arquitecturas multinúcleo que están por venir seguirán empleando el modelo de memoria compartida e implementarán a nivel hardware un mecanismo que asegure la coherencia de los niveles privados de caché de cada núcleo [1]. De esta forma, comunicación y sincronización (esta última implementada normalmente usando posiciones de la memoria compartida) ocurrirán bajo el control del protocolo de coherencia de cachés.

La mejor manera de asegurar la coherencia de las cachés privadas cuando el número de núcleos de procesamiento es grande es a través de un protocolo de coherencia de cachés basado en directorio. En estas soluciones se emplea una estructura de directorio distribuida que almacena información sobre qué cachés privadas mantienen copia de qué bloques de datos. De esta forma, los fallos en la cada caché privada de último nivel son enviados al módulo de directorio correspondiente, el cual estará encargado de realizar las acciones de coherencia oportunas (por ejemplo, la invalidación de todas las copias compartidas del bloque de datos en un fallo de escritura).

Desde que fueron propuestos por primera vez a finales de la década de los 70 [2], los protocolos de coherencia basados en directorio vienen siendo empleados como una solución escalable al problema de la coherencia de cachés tanto en el ámbito académico como diseños comerciales [3], [4]. De esta manera, existe a día de hoy un conocimiento profundo sobre el impacto que la mayoría de sus aspectos de diseño tienen sobre el rendimiento, el consumo de energía y el coste. Sin embargo, a pesar del conocimiento acumulando durante los últimos cuarenta años, existe aún un aspecto de diseño sutil pero importante, para el cual parece no haber unanimidad en los trabajos de investigación recientes y que tiene que ver sobre cómo se han de manejar los reemplazos de los bloques que no han sido modificados localmente (bloques limpios)[1]. Más concretamente, mientras que algunos trabajos asumen una política de reemplazos *silenciosos* para bloques limpios expulsados del último nivel de cachés privadas [5], [6], [7], [8],

[1]Departamento de Ingeniería y Tecnología de Computadores, Universidad de Murcia, e-mail: {rfernandez, aros, meacacio}@ditec.um.es

[1]A partir de una revisión que hemos realizado de la mayoría de los artículos sobre coherencia de caché aparecidos en las cinco últimas ediciones de las conferencias ISCA, HPCA, PACT and MICRO, hemos encontrado que de 36 artículos, la política de reemplazos *ruidosos* se asumía en 14, la de reemplazos *silenciosos* en 8, mientras que en otros 14 artículos no se mencionaba cómo se trataban este tipo de reemplazos.

otros implementan para estos casos justo lo contrario, una política de reemplazos *ruidosos* [9], [10], [11], [12], [13], e incluso algunos otros trabajos ni siquiera mencionan la manera en la que se gestionan los reemplazos de bloques limpios [14], [15].

En este trabajo demostramos que este aspecto puede tener una influencia significativa sobre el rendimiento y el consumo de energía de un protocolo de coherencia basado en directorio. El utilizar la política de reemplazos *silenciosos* puede ahorrar una gran cantidad de tráfico de red (más de un 25 % en varios casos, 9.6 % de media) en comparación con el uso de la política de reemplazos *ruidosos*. Dado que en las arquitecturas multinúcleo con un gran número de núcleos que están por venir se espera que la red de interconexión dentro del chip represente una fracción muy importante del consumo de energía total [16], [17], la utilización de la política de reemplazos *silenciosos* para bloques limpios implicará ahorros en consumo de energía significativos. Sin embargo, lo que es más importante es que hemos comprobado que en función de cómo se organice la estructura de directorio, el uso de la política de reemplazos *silenciosos* podría afectar o no al rendimiento, lo que indica que la asunción de la política de reemplazos *sucios* no está justificada siempre, ya que incrementaría de forma innecesaria el tráfico de red sin que ello supusiese ninguna ventaja en cuanto a rendimiento se refiere. Para ahondar en este último aspecto, en este trabajo mostramos dos casos de uso en los que en cada uno es mejor aplicar una política sobre la contraria.

El resto del documento se organiza como sigue. Empezamos discutiendo en la sección II las ventajas e inconvenientes de cada una de las políticas de gestión de reemplazos de bloques limpios comentadas con anterioridad. Después, en la sección III, mostramos el entorno de evaluación que hemos utilizado y los casos de uso evaluados, y a continuación, en la sección IV, proporcionamos resultados detallados en términos de tiempo de ejecución y tráfico de red, demostrando la importancia que tiene el manejo adecuado de los reemplazos de bloques limpios. Por último, en la sección V resumimos las principales conclusiones de este trabajo.

II. Reemplazos de bloques limpios: ¿silenciosos o ruidosos?

Cada vez que se produce un fallo de caché, y una vez resuelto el fallo, el bloque solicitado es llevado a la caché. Si no hay espacio en caché para dicho bloque, otro bloque tiene que ser reemplazado de la caché. Los reemplazos, por tanto, son tan frecuentes como los fallos de cache (asumiendo que la caché esté llena).

El bloque que se reemplaza puede haber sido modificado localmente (bloque *sucio*) o no (bloque *limpio*). Ante un reemplazo de un bloque sucio, el siguiente nivel de caché debe ser actualizado. Sin embargo, ante un reemplazo de un bloque limpio no hay que actualizar nada, y por tanto se puede realizar de forma *silenciosa*. No obstante, en sistemas multiprocesador hay veces que es conveniente informar del reemplazo de bloques limpios para actualizar la información de directorio. En este artículo llamaremos a estos reemplazos *ruidosos*.

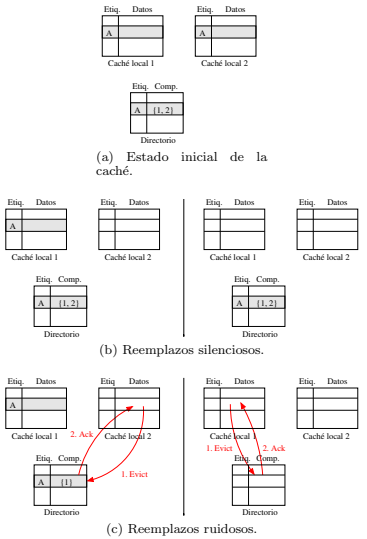


Fig. 1: Ejemplos de reemplazos.

Las siguientes secciones detallan las ventajas e inconvenientes del uso de reemplazos silenciosos y reemplazos ruidosos y analizan cuándo es mejor utilizar cada uno de ellos.

A. Ejemplos de reemplazos silenciosos y ruidosos

La Fig. 1 muestra el comportamiento de los reeplazos silenciosos y ruidosos. En los ejemplos se muestran tan solo dos cachés locales y uno de los bancos de directorio, que guarda información acerca de los bloques cacheados. La situación inicial de la caché es la misma en ambos casos y se muestra en la Fig. 1a: las dos cachés almacenan una copia del bloque *A* y el directorio contiene tal información (en el campo *Comp.*).

En la Fig. 1b los reemplazos se realizan de forma silenciosa. La figura de la izquierda nuestra el reemplazo del bloque *A* en la *caché local 2*. Este reemplazo se realiza sin la necesidad de emitir ningún mensaje de coherencia. El dato es simplemente descartado de la caché local. El directorio, por tanto, mantiene su información intacta, por lo que no será precisa.

Es importante resaltar que no puede ocurrir ningún comportamiento incorrecto debido a esta imprecisión, siempre y cuando el directorio incluya en el conjunto de compartidores a todos los compartidores actuales (puede incluir nodos adicionales). Si el

directorio necesita invalidar todos los compartidores de un bloque, algunas cachés podrán recibir mensajes de invalidación innecesarios que requerirán respuesta, pero esto sólo se traduce en tráfico de coherencia extra, no en un comportamiento incorrecto.

La Fig. 1b (derecha) muestra otro reemplazo silencioso del mismo bloque A, en este caso de la *caché local 1*. Como se puede observar, la entrada del bloque *A* se mantiene ocupada en el directorio a pesar de no haber ninguna caché que almacene una copia de ese bloque.

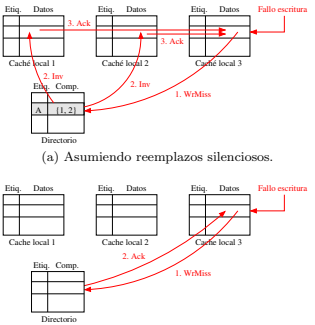
La Fig. 1c muestra como se realizan los reemplazos ruidosos. En este caso, es necesario enviar mensajes de coherencia al directorio. En primer lugar, la caché que reemplaza el bloque envía la notificación al directorio. Cuando el directorio la recibe, actualiza el campo *Comp.*, eliminando la caché que ha reemplazado el dato de la lista de compartidores. Por último, el directorio confirma a la caché el reemplazo y la transacción de coherencia acaba. La Fig. 1c (izquierda) muestra el reemplazo ruidoso del bloque A de la *caché local 2*.

La Fig. 1c (derecha) muestra unç reemplazo del bloque A en la *caché local 1*. El reemplazo ocurre de la misma forma que para la *caché local 2*, pero en este caso el directorio ya no necesita almacenar información acerca de los compartidores, debido a que ninguna caché tiene copia del dato. Por tanto, el directorio puede liberar la entrada del bloque *A*, permitiendo así hacer un mejor uso del mismo.

B. Consecuencias de la política de reemplazo

La primera consecuencia de la política de reemplazo (silenciosa o ruidosa) es la cantidad de *tráfico* generada por el protocolo de coherencia de caché. Informar al directorio sobre el reemplazo de un bloque limpio implica dos mensajes de control, tal y como se muestra en la Fig. 1c. Observe que también serían necesarios dos mensajes de control para invalidar una copia limpia ya reemplazada en el caso en el que los reemplazos fueran silenciosos. Sin embargo estos mensajes sólo se producirían si los bloques sufrieran un fallo de escritura o un reemplazo de la información de directorio, es decir, una invalidación. Por tanto, los bloques de solo lectura o los bloques limpios que se reemplazan y cachean frecuentemente sin ninguna invalidación entre ambos eventos se beneficiarán de una política de reemplazo silenciosa. Como hemos observado que estas situaciones son frecuentes, el tráfico adicional generado por el uso de reemplazos ruidosos puede representar una gran parte del tráfico global generado por el protocolo de coherencia de cachés.

La segunda consecuencia de implementar reemplazos ruidosos para bloques limpios es la *exactitud* de la información de directorio. Esto afecta al protocolo de coherencia de dos formas diferentes. Por un lado, se reduce la cantidad de información que el directorio necesita almacenar. Esto significa que ante un fallo de escritura o un reemplazo en el directorio se emiten menos mensajes de invalidación, y por lo tanto se recibirán también menos mensajes de confirmación. El envío de un menor número de invalidaciones reduce tanto el tráfico como la latencia de fallos de escritura. Informar de los reemplazos de datos limpios puede ser también beneficioso cuando cada entrada de directorio sólo puede almacenar un número limitado de compartidores (por ejemplo, en un esquema de punteros limitados). De esta forma se reutiliza naturalmente el almacenamiento del directorio, reduciendo, por tanto, el número de situaciones de desbordamiento que de lo contrario podrían surgir. Por otro lado, y más importante, la ocupación de directorio se reduce, ya que las entradas de directorio para los bloques que han sido reemplazados de todas las cachés pueden ser eliminadas del directorio. La reducción de la presión en el directorio conduce en efecto a un menor número de reemplazos de directorio y, en consecuencia, a un menor número de invalidaciones, que a su vez da lugar a un menor número de fallos de caché.



(a) Asumiendo reemplazos silenciosos.

(b) Asumiendo reemplazos ruidosos.

Fig. 2: Ejemplo de fallo de escritura dependiendo de la política de reemplazo.

El efecto de tener información de directorio precisa ante un fallo de escritura se muestra en la Fig. 2. Supongamos por ejemplo, que ocurre un fallo de escritura para el bloque *A* después de los dos reemplazos de la Fig. 1. En el caso de haber usado reemplazos silenciosos (Fig. 2a), el directorio tiene que enviar invalidaciones a las L1 cachés a pesar de que ya no tienen una copia del bloque. Esto se debe a que el directorio no fue notificado cuando el bloque se reemplazó. El resultado es que los dos mensajes ahorrados por los reemplazos silenciosos son generados ahora por la invalidación, con un coste extra en latencia en el camino crítico del fallo. Sin embargo, en el caso de reemplazos ruidosos (Figura 2b), el directorio no realiza invalidaciones innecesarias, sino que directamente da permiso de escritura a la caché que generó el fallo.

C. Cuándo usar reemplazos silenciosos o ruidosos

A partir de los ejemplos anteriores se puede deducir que los dos tipos de políticas de reemplazo tie-

nen ventajas y desventajas. Los reemplazos silenciosos pueden reducir el tráfico de coherencia debido a los reemplazos, mientras que los reemplazos ruidosos pueden reducir la latencia de los fallos de escritura y mejorar la eficiencia de directorio. La mejor política dependerá tanto de las características de la aplicación como del protocolo de coherencia de caché empleado.

Los inconvenientes de los reemplazos silenciosos son las invalidaciones adicionales al resolver tanto los fallos de escritura como los reemplazos de directorio. Los datos de sólo lectura, por lo tanto, se beneficiarían de los reemplazos silenciosos si con frecuencia son reemplazados y solicitados de nuevo sin que el directorio reemplace la entrada correspondiente. Del mismo modo, los datos de lectura y escritura se verían favorecidos por los reemplazos silenciosos si los fallos de lectura dominan claramente a los fallos de escritura. De lo contrario, los remplazos ruidosos pueden ser preferibles debido a una menor latencia de las escrituras.

Las propiedades del protocolo de coherencia de caché empleado tienen aún más importancia a la hora de tomar una decisión sobre la implementación de la política de reemplazo. Por ejemplo, un protocolo basado en listas enlazadas [18], [19], [20] no podría emplear reemplazos silenciosos ya que la lista de compartidores debe ser actualizada ante un reemplazo [21]. Del mismo modo, un protocolo basado en tokens [22], [23] no puede usar reemplazos silenciosos ya que los tokens que acompañan al bloque en caché deben transferirse al directorio o a otro compartidor.

Esta decisión es especialmente relevante para protocolos que emplean directorios donde el número de entradas utilizadas por bloque de memoria depende del número de compartidores (por ejemplo, asignación dinámica de punteros [24] o SCD [13]). En estos casos, una mejor precisión proporcionada por el uso de reemplazos ruidosos puede permitir una mejor utilización del directorio y, en consecuencia, reducir el número de reemplazos de directorio. Esto es menos importante en directorios tradicionales, ya que, en ese caso, las entradas de directorio solo se eliminan cuando el número de compartidores llega a cero.

III. Entorno de evaluación

Hemos utilizado los simuladores PIN [25] y GEMS 2.1 [26], conectándolos de forma similar a como se propone en [27]. Para modelar la red de interconexión, usamos SiCoSys [28]. La arquitectura simulada corresponde a un multiprocesador en un único chip (*tiled*-CMP) con 64 núcleos. Los principales parámetros de evaluación se muestran en la Tabla I.

Evaluamos el impacto que tiene la política de reemplazos de bloques limpios en dos configuraciones CMP de 64 núcleos. La primera configuración utiliza un directorio disperso que utiliza un vector de bits no escalable para codificar los compartidores de cada entrada (la primera barra de todas las gráficas representa a esta configuración con reemplazos silenciosos,

TABLA I: Parámetros del sistema.

Parámetros de la memoria	
Tamaño de bloque	64 bytes
Caché L1 de datos e instr.	32 KiB, 4 vías
Latencia de acceso a L1	1 ciclo
Caché L2 compartida	256 KiB/celda, 16 vías
Latencia de acceso a L2	6 ciclos más latencia de red
Organización L1 y L2	Inclusiva
Información de directorio	Incluida en L2
Tiempo de acceso a memoria	160 ciclos
Parámetros de la red	
Topología	Malla 2-D de 8×8 nodos
Técnica de conmutación	Wormhole
Técnica de enrutamiento	Determinista X-Y
Multicast/Broadcast	No soportado
Tamaño de mensajes	4 flits (datos), 1 flit (control)
Tiempo de enrutamiento	1 ciclo
Tiempo de conmutador	1 ciclo
Tiempo de enlace	2 ciclos
Tamaño del buffer	6 flits
Ancho de banda	1 flit por ciclo

mientras que la segunda usa reemplazos ruidosos). La segunda configuración representa a la qciéntemente propuesta arquitectura de directorio SCD [13] (que aparece usando reemplazos silenciosos en la tercera barra de la gráficas y usando reemplazos ruidosos en la cuarta).

Para nuestras imulaciones, hemos usado una gran variedad de aplicaciones de los conjuntos de aplicaciones SPLASH-2 [29] y PARSEC 2.1 [30]. *Barnes*, *Cholesky*, *FFT*, *Ocean*, *Radix*, *Raytrace*, *Volrend* y *Water-NSQ* pertenecen a SPLASH-2 y emplean los mismos tamaño de entrada recomendados en el artículo [29]. *Bodytrack*, *Canneal*, *Streamcluster* y *Swaptions* pertenecen a PARSEC 2.1 y usan tamaño de entrada *simmedium*. Hemos tenido en cuenta la variabilidad en las aplicaciones paralelas tal y como se describe en [31]. Todos los resultados mostrados en este trabajo corresponden a la parte paralela de las aplicaciones evaluadas.

IV. Resultados

El objetivo de nuestra evaluación es mostrar el impacto de los reemplazos silenciosos frente a los ruidosos. Comenzamos nuestro análisis observando la frecuencia de los reemplazos de L1, clasificándolos según el tipo del bloque expulsado. Para cada aplicación, la figura 3 muestra el número de reemplazos por instrucción ejecutada, clasificando los reemplazos en cuatro catagorías: *Sucio*, *LimpioExclusivo*, *LimpioCompartidoRuidoso* y *LimpioCompartidoSilencioso*. Los reemplazos Sucios se corresponden con los de bloques en estado M (modificado) y los LimpioExclusivo con bloques en estado E. Los reemplazos de bloques en estado S se han clasificado como LimpioCompartidoRuidoso cuando el protocolo utliza la política de reemplazos ruidosos y como LimpioCompartidoSilencioso si usa reemplazos silenciosos.

En promedio, los bloques en estado E no están involucrados en una parte significativa de los reemplazos, aunque hay excepciones en aplicaciones como *Canneal* y *FFT*. La mayoría de los reemplazos de caché L1 (67 % en SCD y 73 % en el directorio basado en vector de bits, de media) corresponden a

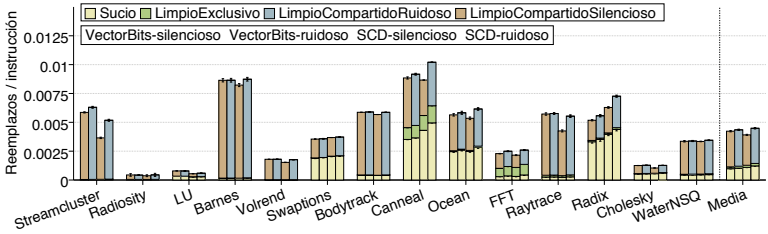


Fig. 3: Reemplazos de L1 por instrucción, categorizados.

bloques en estado S. Este es el primer indicio de la importancia que tiene la implementación eficiente de este tipo de reemplazos.

Los reemplazos ruidosos incrementan la precisión de la información de directorio al eliminar del conjunto de compartidores apuntado en el directorio aquellos nodos que expulsan el bloque de su caché local. En un directorio disperso tradicional, las entradas de directorio se liberan únicamente cuando el número de compartidores llega a 0. Sin embargo, en directorios en los que el código de compartición se distribuye entre varias entradas, tales como SCD, el uso de reemplazos ruidosos ayuda a reducir la presión en el directorio porque permite liberar algiunas entradas tan pronto como dejan de ser necesarias, lo que reduce la ocupación del directorio. Con el objetivo de analizar este efecto, la Figura 4 muestra el número de reemplazos de directorio por instrucción ejecutada. De media, cuando se usan reemplazos ruidosos, el número de reemplazos de directorio se reduce en un 1.3 % en el caso del vector de bits y en un 30 % en el caso de SCD.

Otro efecto esperado del uso de los reemplazos silenciosos en una reducción en la latencia de los fallos de escritura debido a que se necesita enviar menos invalidaciones. La Figura 5 muestra la latencia de los fallos de escritura para cada protocolo. Cada barra está dividida en cinco partes: tiempo accediendo a la L1 (*En_L1*); tiempo hasta llegar a la L2 (*Hasta_L2*), tiempo de espera en la L2 hasta que ésta puede atender la petición, debido principalmente a que el controlador está ocupado atendiendo a toras peticiones (*En_L2*); tiempo de acceso a la memoria (*Memoria*) y tiempo desde que el fallo deja la L2 hasta que la L1 recibe la respuesta y se resuelve (*Hasta_L1*) el fallo. En realidad, sólo se aprecian muy pequeñas variaciones en la latencia: se reduce muy ligeramente para algunos benchamrks en el caso del vector de bits o se incrementa también muy ligeramente para otros en el caso de SCD. Esto ocurre así porque en la práctica el número de invalidaciones por fallo es similar y, lo que es más, todas las invalidaciones se envían en paralelo y sólo el procesamiento de los mensajes de reconocimiento se beneficia de la reducción, ya que esto tiene que ser realizado secuencialmente por el peticionario. Por otro lado, el ligero incremento en tráfico y reemplazos de directorio puede explicar el ligero incremento en latencia.

Por otro lado, el efecto en la latencia de los fallos de lectura se puede ver en la Figura 6. Los fallos de lectura se manejan exactamente igual tanto en el caso de los reemplazos ruidosos como en el de los silenciosos, por lo cualquier diferencia se debe explicar principalmente por la diferente cantidad de tráfico que viaja por la red y por el diferente número de reemplazos de directorio debido a la mayor precisión en la información de compartidores proporcionada por los reemplazos ruidosos. Por ejemplo, la latencia de *Streamcluster* con SCD se reduce cuando se usan reemplazos ruidosos en lugar de silenciosos. Esto es porque el tiempo *En_L2* es menor, lo que es consecuencia de que hay menos reemplzos de directorio manteniendo ocupado al controlador. Igual que en caso de las escrituras, vemos que las diferencias en promedio son mínimas, con un incremento casi insignificante de la latencia cuando se usan reemplazos ruidosos.

El impacto más importante de la política de reemplazos de datos limpios está en el tráfico de la red de interconexión. La Figura 7 muestra el tráfico de coherencia medido en flits y normalizado respecto al directorio basado en vector de bits con reemplazos silenciosos. Se ha dividido el tráfico en las siguientes categorías: mesajes de datos debidos a fallos de caché (*Data*), mensajes de datos debidos a reemplazos (*WBData*), mensajes de control debidos a fallos de caché (*Control*), y mensajes de control debidos a reemplazos (*WBControl*). Vemos que, aunque un protocolo con reemplazos silenciosos genera en algunas ocasiones un poco más de tráfico de mensajes de control debidos a fallos de caché, un protocolo con reemplazos ruidosos genera siempre una cantidad significativamente mayor de mensajes de control debidos a reemplazos. En el caso del directorio vector de bits, los reemplazos ruidosos casi siempre incrementas el tráfico total (excepto en *FTT* y *Radix*), pero el incremento es menos pronunciado en el caso de SCD y hay más exceptciones (*FTT* y *Radix* de nuevo, pero también *Canneal*, *Cholesky*, *LU* y *Streamcluster*). Esto ocurre porque los reemplazos ruidosos pueden

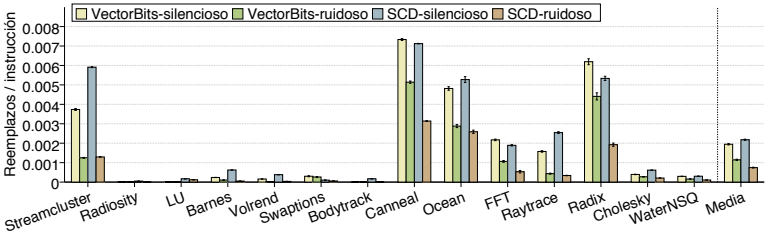


Fig. 4: Reemplazos de directorio por instrucción.

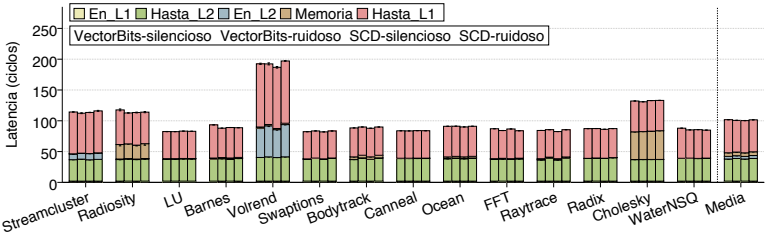


Fig. 5: Latencia de los fallos de escritura en L1.

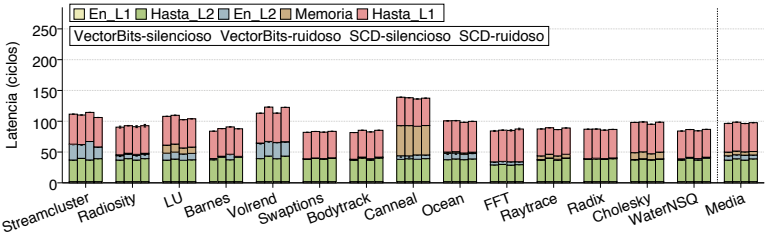


Fig. 6: Latencia de los fallos en L1.

reducir el númerode reemplazos de directorio (como se muestra en la figura 4). Estos reemplazos de directorio requieren la invalidación de cualquier copia del bloque de datos que estuviera en alguna caché local, por lo que su reducción reduce también el número de fallos que, a su vez, reduce la cantidad de mensajes de control y, lo que es más importante, la cantidad de mensajes de datos (que son más grandes). De media, los reemplazos ruidosos incrementa el tráfico total de la red en un 9.6 % para un directorio basado en vector de bits y en un 4.1 % para el directorio SCD.

Para terminar la evaluación, mostramos también el impacto en el tiempo de ejecución de la política de reemplazos en la Figura 8. Todos los resultados se han normalizado respecto a un directorio basado en vector de bits que usa reemplazos silenciosos. De media, el uso de reemplazos ruidosos no afecta en nada al tiempo de ejecución de un protocolo de directorio basado en vector de bits respecto al uso de reemplazos silenciosos. Dado que los reemplazos ruidosos incrementan el tráfico en un 9.6 %, podemos concluir que cuando se usa un directorio basado en vector de bits, es mejor utilizar reemplazos silenciosos. El efecto en el tiempo medio de ejecución del directorio SCD es muy pequeño: los reemplazos ruidosos reducen el tiempo de ejecución en un 1.5 % a

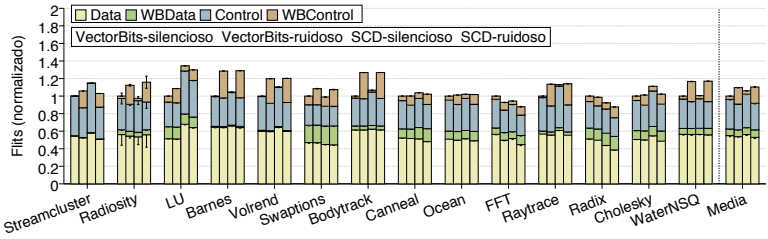


Fig. 7: Tráfico en la red de interconexión.

cambio de un incremento del 4.1 % en tráfico. Por tanto, el uso de reeemplazos ruidosos en un protocolo como SCD puede estar justificado si el diseñador quiere conseguir el máximo rendimiento sin que importe el coste energético.

V. Conclusiones

En este trabajo ponemos el foco sobre un aspecto de diseño de los protocolos de coherencia basados en directorio que creemos que está siendo subestimado en la actualidad: cómo conviene gestionar los reemplazos de los bloques limpios en el último nivel de cachés privadas. Discutimos las dos alternativas que son posibles (tener reemplazos *silenciosos* frente a reemplazos *ruidosos*), enfatizando las ventajas e inconvenientes de cada una de ellas. A continuación consideramos dos estructuras alternativas para el directorio y mostramos, teniendo en cuenta el rendimiento (tiempo de ejecución) y la cantidad de tráfico de red, que para cada una de ellas es preferible usar una política distinta.

Más concretamente, nuestros resultados demuestran que la utilización de la política de reemplazos *ruidosos* no está justificada cuando se emplea un directorio que utiliza vectores de bits como código de compartición, ya que se incrementaría el tráfico en más de un 25 % en varios casos (9.6 % de media) y no se obtendría ningún beneficio desde el punto de vista del tiempo de ejecución. El hecho de que todos los compartidores tengan que haber reemplazado sus copias del bloque de datos para que la entrada de directorio pueda ser liberada, limita mucho el número de ocasiones en las que la entrada podría ser reasignada a otra dirección distinta como consecuencia de las notificaciones de reemplazo. Además, hemos observado también que hay ocasiones en las que un bloque limpio se reemplaza varias veces antes de que sea invalidado (como consecuencia de un fallo de escritura o del reemplazo de la entrada de directorio asignada a la dirección del bloque), lo que ocasiona bastante tráfico extra cuando se usa la política de reemplazos *ruidosos*. Una conclusión importante, por lo tanto, que se podría extraer de este trabajo es que cuando se asume como sistema base en una comparativa un directorio que utiliza vectores de bits, este debería utilizar la política de reemplazos *silenciosos*, ya que de otra manera se estaría realizando la comparativa partiendo de una configuración base subóptima.

Por otro lado, también hemos constatado que la habilidad que tiene la política de reemplazos *ruidosos* de mejorar la precisión de la información de directorio puede tener efectos balsámicos en algunas aplicaciones cuando el directorio utiliza un código de compartición que se distribuye entre distintas entradas, tal y como sucede en SCD. En estos casos, la utilización de reemplazos *ruidosos* ayuda a reducir la presión sobre el directorio, ya que permite liberar algunas entradas en el momento en el que los pocos compartidores que codificaban han reemplazado su copia del bloque de datos. Esto al final conlleva reducciones en la ocupación del directorio y, gracias a este efecto, el tiempo de ejecución puede ser reducido (hasta un 9.8 %, 2.7 % de media) si se emplea la política de reemplazos *ruidosos*.

Agradecimientos

Este trabajo ha sido financiado por la Fundación Séneca-Agencia de Ciencia y Tecnología de la Región de Murcia mediante el proyecto “19295/PI/14”.

Referencias

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

[2] Lucien M. Censier and Paul Feautrier, “A new solution to coherence problems in multicache systems,” *IEEE Transactions on Computers (TC)*, vol. 27, no. 12, pp. 1112–1118, Dec. 1978.

[3] Daniel J. Sorin, Mark D. Hill, and David A. Wood, *A Primer on Memory Consistency and Cache Coherence*, Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2011.

[4] David E. Culler, Jaswinder P. Singh, and Anoop Gupta, *Parallel Computer Architecture: A Hardware/Software Approach*, Morgan Kaufmann Publishers, Inc., 1999.

[5] Dana Vantrease, Mikko H. Lipasti, and Nathan Binkert, “Atomic coherence: Leveraging nanophotonics to build race-free cache coherence protocols,” in *17th Int’l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2011, pp. 132–143.

[6] Blas Cuesta, Alberto Ros, María E. Gómez, Antonio Robles, and José Duato, “Increasing the effectiveness of directory caches by deactivating coherence for private memory blocks,” in *38th Int’l Symp. on Computer Architecture (ISCA)*, June 2011, pp. 93–103.

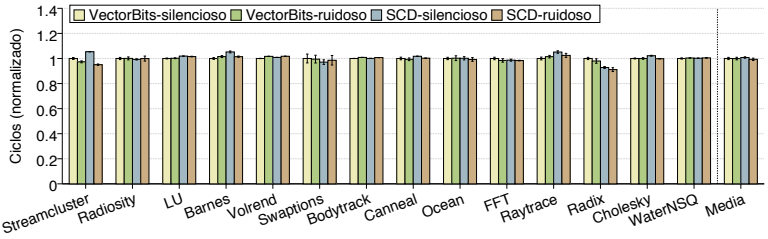


Fig. 8: Tiempo de ejecución.

[7] Marco Elver and Vijay Nagarajan, "TSO-CC: Consistency directed cache coherence for tso," in *20th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2014, pp. 165–176.

[8] Meng Zhang, Jesse D. Bingham, John Erickson, and Daniel J. Sorin, "PVCoherence: Designing flat coherence protocols for scalable verification," in *20th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2014, pp. 392–403.

[9] Jason Zebchuk, Babak Falsafi, and Andreas Moshovos, "Multi-grain coherence directories," in *46th IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, Dec. 2013, pp. 359–370.

[10] Socrates Demetriades and Sangyeun Cho, "Stash directory: A scalable directory for many-core coherence," in *20th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2014, pp. 177–188.

[11] Lucia G. Menezo, Valentin Puente, and José Ángel Gregorio, "Flask coherence: A morphable hybrid coherence protocol to balance energy, performance and scalability," in *21th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2015, pp. 198–209.

[12] Minshu Zhao and Donald Yeung, "Studying the impact of multicore processor scaling on directory techniques via reuse distance analysis," in *21th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2015, pp. 590–602.

[13] Daniel Sanchez and Christos Kozyrakis, "SCD: A scalable coherence directory with flexible sharer set encoding," in *18th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2012, pp. 129–140.

[14] Guowei Zhang, Webb Horn, and Daniel Sanchez, "Exploiting commutativity to reduce the cost of updates to shared data in cache-coherent systems," in *48th IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, Dec. 2015, pp. 13–25.

[15] Yaosheng Fu, Tri M. Nguyen, and David Wentzlaff, "Coherence domain restriction on large scale systems," in *48th IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, Dec. 2015, pp. 686–698.

[16] Thomas Moscibroda and Onur Mutlu, "A case for bufferless routing in on-chip networks," in *36th Int'l Symp. on Computer Architecture (ISCA)*, June 2009, pp. 196–207.

[17] Shekhar Borkar, "Thousand core chips: a technology perspective," in *44th Design Automation Conference (DAC)*, June 2007, pp. 746–749.

[18] David V. James, Anthony T. Laundrie, Stein Gjessing, and Gurindar S. Sohi, "Scalable coherent interface," *IEEE Computer*, vol. 23, no. 6, pp. 74–77, June 1990.

[19] Tom Lovett and Russell Clapp, "STiNG: A cc-NUMA computer system for the commercial marketplace," in *23rd Int'l Symp. on Computer Architecture (ISCA)*, June 1996, pp. 308–317.

[20] Radhika Thekkath, Amit P. Singh, Jaswinder P. Singh, Susan John, and John L. Hennessy, "An evaluation of a commercial cc-NUMA architecture: The CONVEX Exemplar SPP1200," in *11th Int'l Symp. on Parallel Processing (IPPS)*, Apr. 1997, pp. 8–17.

[21] Ricardo Fernández-Pascual, Alberto Ros, and Manuel E. Acacio, "Optimization of a linked cache coherence protocol for scalable manycore coherence," in *Proc. of 29th International Conference on Architecture of Computing Systems (ARCS 2016)*, 2016, pp. 100–112.

[22] Milo M.K. Martin, Mark D. Hill, and David A. Wood, "Token coherence: Decoupling performance and correctness," in *30th Int'l Symp. on Computer Architecture (ISCA)*, June 2003, pp. 182–193.

[23] Michael R. Marty, Jesse D. Bingham, Mark D. Hill, Alan J. Hu, Milo M.K. Martin, and David A. Wood, "Improving multiple-CMP systems using token coherence," in *11th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2005, pp. 328–339.

[24] Richard Simoni and Mark A. Horowitz, "Dynamic pointer allocation for scalable cache coherence directories," in *Int'l Symp. on Shared Memory Multiprocessing*, Apr. 1991, pp. 72–81.

[25] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood, "Pin: Building customized program analysis tools with dynamic instrumentation," in *2005 ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI)*, June 2005, pp. 190–200.

[26] Milo M.K. Martin, Daniel J. Sorin, Bradford M. Beckmann, Michael R. Marty, Min Xu, Alaa R. Alameldeen, Kevin E. Moore, Mark D. Hill, and David A. Wood, "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset," *Computer Architecture News*, vol. 33, no. 4, pp. 92–99, Sept. 2005.

[27] Matteo Monchiero, Jung Ho Ahn, Ayose Falcón, Daniel Ortega, and Paolo Faraboschi, "How to simulate 1000 cores," *Computer Architecture News*, vol. 37, no. 2, pp. 10–19, July 2009.

[28] Valentín Puente, José A. Gregorio, and Ramón Beivide, "SICOSYS: An integrated framework for studying interconnection network in multiprocessor systems," in *10th Euromicro Workshop on Parallel, Distributed and Network-based Processing*, Jan. 2002, pp. 15–22.

[29] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *22nd Int'l Symp. on Computer Architecture (ISCA)*, June 1995, pp. 24–36.

[30] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *17th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Oct. 2008, pp. 72–81.

[31] Alaa R. Alameldeen and David A. Wood, "Variability in architectural simulations of multi-threaded workloads," in *9th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2003, pp. 7–18.

