Mecanismo de clasificación de páginas basado en el paso de tokens entre TLBs

Albert Esteve, [1] Alberto Ros, [2] Antonio Robles [1] y María E. Gómez [1]

Resumen— **Clasificar los accesos a memoria como datos privados o compartidos se ha convertido en un esquema fundamental para lograr eficiencia y escalabilidad en sistemas multi- y many-core. Puesto que la mayor parte de los accesos a memoria tanto en aplicaciones secuenciales como paralelas son considerados datos privados (accedidos por un único núcleo) o de solo lectura (no se escriben), consagrar el coste del mantenimiento de la coherencia en cada acceso a memoria resulta en un rendimiento sub-óptimo, al mismo tiempo que limita la escalabilidad y eficiencia del sistema. Este trabajo propone TokenTLB, un mecanismo de clasificación de páginas basado en el intercambio y la cuenta de tokens (testigos). La observación principal tras nuestra propuesta es que, a diferencia del mantenimiento de la coherencia, usar tokens para clasificar datos obtiene todos los beneficios de un protocolo basado en tokens sin la carga de un sistema de arbitraje complejo, cuya aplicación ha desalentado la inserción de dichos protocolos en los procesadores actuales. Contar tokens en las TLBs supone una forma natural y eficiente para la clasificación de páginas de memoria. Por un lado, evita el uso de solicitudes persistentes o arbitraje, ya que si dos o más TLBs compiten para acceder a una página, los tokens se distribuyen apropiadamente y la clasifican como compartida. Por otro lado, TokenTLB también favorece la compartición de la traducción entre las TLBs del sistema, lo cual mejora el rendimiento del mismo y reduce gran parte del trafico en comparación con otros mecanismos de clasificación basados en la difusión de mensajes entre TLBs. Esta reducción es debida a que solo las TLBs que poseen tokens adicionales están a cargo de suministrarlos junto con la traducción de la página (una respuesta por fallo de TLB). TokenTLB incrementa de forma efectiva los bloques clasificados como privados hasta un 61.1 %, permitiendo la detección de datos de solo lectura (hasta un 24.4 % de bloques compartidos de solo lectura). Si aplicamos TokenTLB para optimizar la caché de directorio, la energía dinámica consumida por la jerarquía caché se reduce un 27.3 % sobre el sistema base.**

Palabras clave— **Clasificación de datos; Protocolo de tokens; Privado-compartido; Datos de solo lectura**

I. Introducción

Los procesadores multinúcleo (CMPs) están compuestos de una creciente cantidad de núcleos. Esto, a su vez, requiere del uso de jerarquías de memoria multi-nivel por motivos de rendimiento y un modelo de memoria compartida para facilitar la programabilidad. Los modelos de memoria compartida requieren protocolos de coherencia caché para sacar partido a su potencial mejora de rendimiento. Los protocolos basados en directorio son los más adecuados para afrontar los nuevos retos de escalabilidad [1–4]. éstos requieren menor ancho de banda de red comparados con los protocolos *snooping*. Sin embargo, no son capaces de distinguir si los datos accedidos son privados o compartidos. Así, no se saca partido a las potenciales oportunidades para la mejora del rendimiento y la escalabilidad que ofrece esta característica.

Un gran número de propuestas recientes emplean un mecanismo de clasificación de datos y/o accesos en privados o compartidos para así superar las limitaciones en términos de escalabilidad de los CMPs actuales [1, 5–22]. La idea principal en todos estos trabajos es que la naturaleza de los datos privados o compartidos es distinta y, por tanto, las referencias hechas a estos datos se pueden optimizar de acuerdo a dicha naturaleza. La clasificación de datos se ha convertido, por tanto, un mecanismo clave para el diseño de multiprocesadores más escalables y eficientes.

En un escenario ideal, la clasificación debe realizarse con una baja sobrecarga en términos de tráfico, rendimiento o área. Sin embargo, clasificar datos conlleva a menudo grandes requerimientos en términos de almacenamiento para rastrear el estado de compartición. En otras propuestas no se tiene en cuenta las transiciones de compartido a privado, por lo que la precisión en la clasificación se puede ver comprometida, limitando los beneficios potenciales [1,10,12]. Por otro lado, algunos mecanismos de clasificación almacenan su estado de clasificación en las memorias de directorio o las cachés [5, 7, 11, 17, 22], limitando su aplicabilidad. En general, cuanto antes se obtenga la clasificación en un acceso a memoria, mejor. Finalmente, la detección de escritura también ha sido explorada, extendiendo el ámbito del esquema de clasificación y mejorando su eficacia [6, 10], ya que los accesos a datos de solo lectura representan una gran fracción de los accesos a memoria.

Además, la latencia al traducir direcciones se añade al camino crítico para los accesos a memoria y se han dedicado numerosos esfuerzos para reducir dicho impacto. Específicamente, las transferencias de TLB a TLB [9, 18] están indicadas para la clasificación adaptativa de datos aplicados a la desactivación de la coherencia [1, 6], un mecanismo eficiente para mejorar la escalabilidad de los directorios en CMPs. Sin embargo, estas transferencias inundan la red con respuestas tras cada fallo de TLB, muchas de ellas traducciones de página replicadas. Incluso aunque los fallos de TLB son poco frecuentes (solo un 2 % de los accesos a la TLB son fallos), estos mensajes no escalan adecuadamente con el tamaño del sistema. Por último, la reclasificación de compartido a privado no se produce de forma inmediata, sino que es pospues-

[1]Department of Computer Engineering, Universitat Politècnica de València, e-mail: alesgar@gap.upv.es, {megomez,arobles}@disca.upv.es.

[2]Departamento de Ingeniería y Tecnología de Computadores, Universidad de Murcia, e-mail: aros@ditec.um.es..

ta hasta que la página es desalojada de la memoria principal y accedida de nuevo, penalizando por ello la precisión en la clasificación.

En este trabajo proponemos TokenTLB, un mecanismo de clasificación novedoso, implementado directamente en la estructura de TLB e inspirado por los protocolos de coherencia token [23–25]. TokenTLB se basa en la observación de que, a diferencia de la coherencia, aplicar los tokens para la clasificación no requiere utilizar peticiones persistentes o mecanismos de arbitraje complejos. Las peticiones persistentes son un tipo especial de solicitud ideada para resolver las carreras de datos que ocurren ocasionalmente cuando muchos núcleos intentan escribir datos al mismo tiempo. Estas peticiones suponen una fuente de complejidad para el protocolo de coherencia, representando a su vez uno de los motivos principales por el cual los protocolos de coherencia basados en tokens no se han implementado ampliamente en los procesadores actuales. Sin embargo, TokenTLB evita estas carreras dedicando los tokens a la clasificación de datos. Si dos o más TLBs compiten al acceder a una página, los tokens se distribuyen entre ellas de forma natural y la página es clasificada como compartida. En otras palabras, la clasificación no requiere un *owner token*. Las principales contribuciones de TokenTLB son:

1. TokenTLB esta diseñado como un mecanismo de clasificación a nivel de página en las TLBs, basado en tokens. Permite la compartición a través de TLBs, lo cual acelera las traducciones de página, favoreciendo una mejora del rendimiento del sistema.
2. TokenTLB reduce el consumo de la red en comparación a otras propuestas similares basadas en TLBs. Solo TLBs que son propietarias de tokens están a cargo de proveerlos junto con la traducción de la página, lo que supone alrededor de una respuesta por fallo de TLB.
3. TokenTLB extiende la caracterización de la clasificación a través de la detección de escrituras a páginas, al mismo tiempo que realiza una clasificación basada en la cuenta y el intercambio de tokens. La clasificación basada en tokens hace posible que se identifique de forma natural e inmediata como una entrada de TLB evoluciona de compartida a privada.
4. TokenTLB introduce un predictor capaz de resolver fallos de TLB a través de mensajes punto a punto, incrementando así su escalabilidad. El predictor se basa en un pequeño buffer llamado Token Predictor Buffer (TPB), el cual se encarga del almacenamiento durante un corto espacio de tiempo de potenciales TLBs propietarias.

Por medio de simulaciones ciclo a ciclo de un sistema CMP de 16 núcleos, incluyendo una gran variedad de cargas de trabajo tanto científicas como comerciales, se muestra que TokenTLB aumenta la cantidad de bloques clasificados como privado en el momento del fallo hasta un 61.1 %, y los bloques compartidos de solo lectura hasta un 24.4 %. Además, si la clasificación se aplica a la desactivación de la coherencia, ésta conlleva una reducción en la utilización del directorio, dejándolo en un 28.8 % de ocupación. También se reduce el consumo de la jerarquía caché hasta un 27.3 % sobre el sistema base. Finalmente, la inclusión de la TPB ha permitido reducir aún más el tráfico de las TLBs prácticamente un 20 % sobre el protocolo TokenTLB base. Específicamente, TokenTLB, combinado con TPB, genera una sola respuesta por fallo de TLB. En el caso de que la traducción sea resuelta en la tabla de páginas, ninguna TLB debe responder. Por tanto, TPB permite, junto con TokenTLB, un mecanismo más escalable.

II. Motivación y Trabajo Relacionado

A. Clasificación de Datos

Los mecanismos de clasificación de datos están ganando interés, ya que permiten un gran número de optimizaciones en el manejo de bloques basándose en su estado de compartición. Hay muchos ejemplos recientes en la literatura para esquemas de clasificación. Específicamente, Kim *et al.* [12] evitan solicitar datos coherentes a través de mensajes de difusión en los protocolos *snooping* al acceder bloques privados, lo cual reduce el tráfico de red generado. Alternativamente, Y. Li *et al.* [15] presentan una pequeña estructura buffer cerca de las TLBs, llamada buffer de compartición parcial (PSB). Cuando una página se hace compartida, estará probablemente alojada en la PSB tras un fallo de TLB. Así, la traducción de la página se puede obtener con baja latencia y menores recursos de almacenamiento. Además, Hardavellas *et al.* [10] y Kim *et al.* [14, 16] mantienen los bloques privados en el banco NUCA local, reduciendo así la latencia de acceso a las caches NUCA. Ros y Kaxiras [26] proponen un protocolo de coherencia simple y eficiente que implementa una política de escritura aplazada para los bloques privados y de escritura inmediata para los compartidos. Finalmente, Cuesta *et al.* [1, 6] proponen evitar el almacenamiento de bloques privados en los directorios, desactivando así el mantenimiento de la coherencia para éstos bloques. La desactivación de la coherencia permite así directorios más pequeños y rápidos.

La mayor parte de las propuestas descritas usan mecanismos de clasificación que se benefician de estructuras del sistema operativo existentes (i.e., las TLBs o la tabla de páginas) para clasificar las páginas y almacenar el estado de las mismas. Así evitan requerir estructuras hardware adicionales. Por otro lado, las propuestas asistidas por el compilador [14, 16] lidian con la dificultad de saber en tiempo de compilación (a) si una variable va o no a ser accedida y (b) en qué núcleos van a ser planificados y replanificados los datos. Además, los mecanismos basados en directorio [5, 7, 11, 17, 22] solo descubren el estado de compartición de los datos tras acceder a la estructura de directorio o caché, limitando por tanto su aplicabilidad a optimizaciones donde el conocimiento a-priori del estado de los datos accedi-

	A-priori	Solo lectura	Adaptativo	Preciso
Directory	✗	✓	✓	✓
TLB	✓	✗	✓	✓
OS	✓	✓	✗	✓
Compiler	✓	✓	✓	✗
TokenTLB	✓	✓	✓	✓

TABLA I: Propiedades de los esquemas de clasificación

dos no sea requerido. Por último, mecanismos basados en las propiedades de los lenguajes de programación [20, 21], a pesar de ser muy precisos, no son aplicables a la mayor parte de los códigos existentes. Por contra, los mecanismos basados en el sistema operativo realizan una clasificación en tiempo de ejecución válido para cualquier código, evitando dichas dificultades.

El principal problema de la clasificación basada en el sistema operativo es que realiza una clasificación no adaptativa. Si una página transita de privada a compartida, permanecerá en ese estado durante el resto del tiempo de ejecución (a no ser que se desaloje la página de la memoria principal). En aplicaciones ejecutándose durante un largo período de tiempo, muchas páginas podrían llegar a ser consideradas como compartidas en algún momento, negando las ventajas que se derivan de la clasificación.

Para realizar una clasificación adaptativa que detecte temporalidad en páginas privadas y la migración de sus hilos, se introdujo la *clasificación basada en TLBs* [9, 18], que se utiliza transferencias entre TLBs para consultar las páginas accedidas desde otros núcleos del sistema y así descubrir de forma natural si los bloques dentro de las mismas están presentes en una caché remota, y por tanto la página es compartida, o si, por el contrario, la página esta actualmente privada. Las transferencias entre TLBs están basadas en la observación de que las comunicaciones entre núcleos del CMPs son mucho más rápidas comparadas a las comunicaciones en procesadores tradicionales. Otros trabajos ya se han beneficiado de dicha observación con diferentes propósitos [27, 28].

Sin embargo, las transferencias entre TLBs generan respuestas replicadas, que a su vez incluyen réplicas de la traducción de cada núcleo del sistema tras cada fallo de TLB, lo cual hace que el consumo de la red se incremente de forma drástica [9]. Por otro lado, enviar difusiones en la red de forma frecuente, generando múltiples respuestas cada vez, no es una propuesta escalable, ya que la cantidad de etapas de paso de mensajes incrementa de forma proporcional al tamaño del sistema. Además, reclasificar páginas a privado requiere que la traducción sea eliminada completamente de las TLBs del sistema para que sea efectiva, limitando por tanto la precisión del mecanismo de clasificación. Finalmente, la detección de escritura no ha sido explorada, limitando así el esquema de clasificación a la clásica dicotomía privado-compartida para un mecanismo adaptativo.

La tabla I resume las propiedades principales de los mecanismos de clasificación en el estado del arte comparados con TokenTLB. En primer lugar, conocer la clasificación antes (*a-priori*) de acceder a la caché es crítico para la aplicabilidad del mecanismo de clasificación. Algunas propuestas almacenan el estado de compartición en el directorio y, por tanto, no pueden emplearse para técnicas tales como la *desactivación de la coherencia* [1] o *reactive NUCA* [10], entre otros. Además, la clasificación *adaptativa* conlleva una enorme mejora en la precisión del mecanismo, detectando la migración de los hilos y los accesos a datos en diferentes fases privadas de la página. La clasificación de *solo lectura* es también una propiedad de gran calado, ya que los bloques compartidos de solo lectura pueden llegar a suponer hasta un 48.7 % de todos los bloques accedidos [6]. Sin embargo, ningún mecanismo de clasificación basado en TLBs ha explorado previamente la detección de escritura. Finalmente, los compiladores deben ser conservadores en la clasificación, ya que no poseen información acerca del estado de compartición en tiempo de ejecución, y por tanto, no están obligados a realizar una clasificación *precisa*, ya que la privacidad no se puede garantizar siempre.

B. Coherencia Token

Martin *et al.* presentó TokenB [24], que captura los mejores aspectos de los protocolos de *snoop* y directorio: fallos de caché con baja latencia y la no dependencia en redes de interconexión totalmente ordenadas. Raghavan *et al.* presentó también Token tenure [29], basándose en una caché de directorio para almacenar los tokens.

Los protocolos token garantizan la seguridad de la coherencia a través de la cuenta de los tokens: un procesador puede escribir si y solo si contiene todos los tokens, y puede leer si y solo si contiene al menos un token para dicho bloque. Sin embargo, puesto que las peticiones son enviadas a todos los procesadores a través de mensajes de difusión, se pueden producir condiciones de carrera en el protocolo cuando compiten por un bloque de memoria, y por tanto no resuelven el fallo caché en ninguno de sus intentos. Para poder evitar la inanición y garantizar la resolución de los fallos, los protocolos token invocan *peticiones persistentes* tras diez tiempo de fallo medios no satisfechos.

Las peticiones persistentes causan grandes problemas, ya que requieren arbitraje, añaden una sobrecarga de latencia poco flexible y requieren estructuras adicionales no escalables en la oblea de silicio. Ésta es una de las principales razones por las que los protocolos token no se han impuesto en los procesadores actuales. Por contra, las TLBs no modifican directamente las traducciones en la estructura TLB. Como consecuencia, usar tokens para clasificar y distribuirlos a través de las TLBs puede evitar éstas condiciones de carrera. En el caso de que varias TLBs disputen por la traducción de una misma página, ésta será sencillamente clasificada como compartida.

III. TokenTLB

Nuestro objetivo es lograr todas las propiedades deseables para un mecanismo de clasificación: realizar la clasificación previamente al acceso a la jerarquía caché; implementar un mecanismo completamente adaptativo que sea capaz de realizar una reclasificación de forma precisa; mejorar la caracterización de la clasificación al discernir los accesos de escritura, reconociendo así las páginas de solo lectura; y realizar una clasificación precisa en tiempo de ejecución que sea válida para cualquier código.

Para este fin, este trabajo propone TokenTLB, una técnica de clasificación adaptativa basada en el conteo de tokens. TokenTLB acelera los fallos de TLB a través de un eficiente sistema de resolución de traducciones entre TLBs, al mismo tiempo que copa con la sobrecarga de tráfico que habitualmente conlleva su uso.

A. Clasificar contando tokens: Concepto

TokenTLB asocia un número fijo de tokens a cada entrada de traducción. En un sistema con N núcleos, debe haber N tokens por entrada. Los tokens no se pueden crear o destruir. Los tokens se intercambian a través de mensajes entre TLBs junto con la traducción de la dirección de página. Una entrada de TLB se clasifica de acuerdo con su conteo de tokens: privada si contiene todos los tokens (N), compartida mientras contenga un subconjunto de todos los tokens de la página (de 1 a $N-1$), e invalido si no contiene ningún token. Finalmente, solo entradas de TLB válidas (es decir, que mantengan al menos un token) puede contestar tras una petición de traducción.

Además, para poder discernir si la página ha sido escrita o no, cada entrada de traducción tiene asociado un *flag* de escritura (W, *written*) que es enviado junto con los tokens en transacciones de TLB. El flag de escritura incrementa el alcance de la traducción, añadiendo categorías adicionales: Privada de solo lectura, privada-escrita, compartida de solo lectura y compartida-escrita.

B. Solicitud de tokens tras un fallo de TLB

TokenTLB inicia la resolución de la página en la tabla de páginas tras un fallo de TLB en paralelo con un mensaje de difusión a otras TLBs del sistema. Inicialmente, la tabla de páginas mantiene los N tokens para cada traducción. En consecuencia, tras el primer fallo de TLB para una página de memoria, la tabla de páginas concede todos los tokens a la TLB que lo solicitó. A partir de ahí, los tokens son mantenidos por las TLBs y enviados a través de mensajes como respuesta a otras TLBs solicitándolos, facilitando su intercambio. Cuando una TLB recibe una petición de una traducción de otra TLB, comprueba si posee la entrada de traducción (es decir, si mantiene la traducción junto con dos o más tokens) y si es así, responde a la solicitud con un mensaje, manteniendo un token y enviando el resto. Al recibir la primera respuesta con la traducción y los tokens, la consulta a la tabla de páginas es cancelada, los tokens son anotados de forma privada en la entrada de TLB correspondiente y el acceso de memoria puede proceder. De ésta forma, el trafico de respuesta en las TLBs es contenido, ya que solo se permite que una TLB responda en el caso general (la que ha adquirido la traducción más recientemente). Además, el acceso a la página se desbloquea antes comparado a otros mecanismos similares [9, 18], mejorando por tanto el tiempo de ejecución (como se ve en la Sección VI-A).

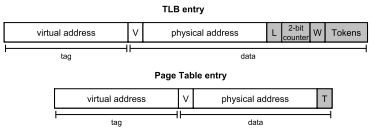

Fig. 1: Formatos de entrada para la TLB y la tabla de páginas. Los campos sombreados representan los campos adicionales que se requieren.

Los tokens se almacenan en una campo adicional de la TLB llamado *Tokens*, como se ven en la Figura 1. Ese campo añade $log_2(N)$ bits a cada entrada (por ejemplo, solo 4 bits adicionales serían necesarios en un CMP de 16 núcleos) en la TLB. Cuando todos los tokens son cedidos, es el bit de válido/invalido (V) de la propia entrada de TLB el encargado de rastrear su estado. En el caso de la tabla de páginas, no se requiere hardware dedicado adicional, sino tan solo un bit adicional por cada entrada (T), indicando si posee o no todos los tokens. Éste bit adicional se puede tomar de los bits reservados de la entrada de la tabla de páginas.

Los fallos de TLB alojan una entrada en el Miss Status Holding Register (MSHR), el cual se desaloja tras adquirir la entrada de traducción de la página y al menos un token para la misma. Por tanto, si el fallo se resuelve en la tabla de páginas sin responder con tokens (éstos están actualmente almacenados en otras TLBs del sistema) debemos esperar a la primera respuesta que contenga tokens. El acceso a la página no se puede desbloquear sin la información de compartición (es decir, los tokens).

Cuando una página es escrita por primera vez (es decir, el bit W no esta activo), éste bit necesita activarse en todas las copias de la traducción almacenadas en otras TLBs. Éste bit permanece activo hasta que el *tiempo de generación global* (el tiempo que pasa desde que la página es accedida por primera vez en una TLB hasta el momento en el que es desalojada de la última TLB del sistema) [9] termina para dicha página. Si la escritura acontece en una página privada no se requieren acciones adicionales. En caso contrario, un mensaje de difusión es enviado para actualizar el bit W en todas las TLBs que contengan tokens, lo cual hace que la página transite a compartida-escrita (SW). La información de escritura se envía junto con los tokens como parte de la información de compartición de la página en cada respuesta a fallos de TLB.

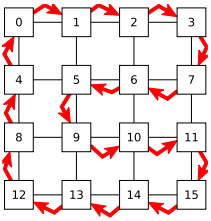

Fig. 2: Camino que sigue los mensajes con tokens tras desalojos en la TLB para una malla 2D de 16 núcleos.

C. Cesión de tokens para la Corrección

Los tokens no se crean ni se destruyen, se transfieren. Esto implica que el sistema debe garantizar en todo momento la existencia de N tokens para cualquier traducción de página. La tabla de páginas contiene o bien todos los N tokens o ninguno.

Al desalojar una entrada de TLB que contiene un subconjunto de tokens, se envía un mensaje buscando un nuevo propietario. Por contra, si una entrada de TLB contiene todos los N tokens, éstos se envían de vuelta a la tabla de páginas. En consecuencia, los desalojos en la TLB deben ser no silenciosos, lo cual añade consumo de red adicional. Sin embargo, los desalojos no silenciosos no dañan las prestaciones del sistema, ya que están fuera del camino crítico para los accesos de memoria. De hecho, los desalojos no silenciosos garantizan una clasificación más dinámica y son la clave para una pronta reclasificación a privado.

Como se ha indicado previamente, un subconjunto de tokens en una entrada de TLB en desalojo deben ser transferidas a un nuevo propietario. A tal fin, se envía un mensaje a otra TLB (*token_evict*) contiendo tokens únicamente. En una situación ideal, una TLB que reciba un mensaje *token_evict* puede solo aceptar los tokens si ya almacena una traducción para dicha página o si tiene aloja una entrada en el MSHR debido a un fallo TLB (Sección III-B). En cualquier otro caso, el mensaje se envía al siguiente TLB designado. Cuando el mensaje *token_evict* encuentra un nuevo propietario, los tokens se anotan en la nueva TLB, la cual responde con un ACK al emisor original.

Para poder evitar potenciales interbloqueos y minimizar el tráfico, la búsqueda de un nuevo propietario requiere una exploración completa de la red. Para ello añadimos el concepto de un anillo lógico, es decir, un camino que cubre todos los nodos de la red, minimizando el número de enlaces a atravesar. El camino seguido es dependiente de la topología. La Figura 2 ilustra un anillo de ejemplo para una malla 4x4. Éste camino represente una de las rutas circulares mínimas que recorren todos los nodos de la red de interconexión.

Sin embargo, los interbloqueos aún puede ocurrir si todas las TLBs que contienen una página (y todos sus tokens) intentan desalojar simultáneamente la entrada asociada. Para solucionar dicho problema, los tokens contenidos en un mensaje *token_evict* pueden ser almacenados en la correspondiente entrada del MSHR de un núcleo cuya TLB este involucrada en un proceso de desalojo, siempre y cuando su ID (identificador) sea mayor que el ID del núcleo que inició el envío.

Cabe observar que la condición clave para evitar la inanición en un escenario en el que múltiples páginas de TLB que son desalojadas ceden indefinidamente los tokens es la comparación entre IDs de núcleo. Si todas las TLBs desalojan simultáneamente la misma página, todos los N tokens serán finalmente alojados en el MSHR de la TLB del núcleo con el ID mayor, el cual enviará todos los tokens de vuelta a la tabla de páginas.

En resumen, una TLB desalojando tokens puede simultáneamente almacenarlos. Por tanto, cuando una TLB recibe una confirmación indicando que los tokens han sido adquiridos por otra TLB, debe comprobar su propio MSHR de nuevo. Si éste contiene un subconjunto de tokens para dicha página, envía un nuevo mensaje buscando un nuevo propietario. En el caso de que el MSHR no contenga más tokens, el proceso de desalojo finaliza.

Por último, una solicitud entre TLBs puede fallar a adquirir tokens tras un fallo si todas las TLBs que sean propietarias de la traducción (entradas que contienen más de un token) han sido desalojadas poco antes de recibir la solicitud, y por tanto sus tokens están actualmente en vuelo. Dicha situación es prevenida a través de un tiempo de vencimiento que se configura tras un fallo de TLB, el cual reenvía el mensaje de difusión si no se consigue adquirir tokens tras el vencimiento. Éste evento no ocurre con frecuencia y no causa un incremento notable en el tráfico de red, como se muestra en la Sección VI-A.

D. Token Predictor Buffer

En la búsqueda de un mecanismo de clasificación más escalable, TokenTLB reduce el tráfico de respuesta de TLB y la duplicación de traducciones. Sin embargo, un mensaje de difusión es enviado aún tras cada fallo de traducción. Esto es debido a que las TLBs no tienen información previa de posibles propietarios en el momento del fallo, por tanto dichos fallos deben ser resueltos inundando la red. No obstante, algunos fallos de TLB ocurren poco después de una invalidación, y por tanto se podría predecir un potencial propietario. Por tanto, el tráfico TLB se puede reducir aún más, contribuyendo a un mecanismo de clasificación más escalable. Para dicho fin, presentamos un predictor que se encarga de identificar otras TLBs como potenciales propietarios. Acertar en el predictor tras un fallo de TLB envía un mensaje punto a punto, por tanto reduce el tráfico de petición. Este mecanismo basado en la historia reciente es similar al propuesto por Martin *et al.* [30]. Otros trabajos también se benefician de ésta observación con diferentes objetivos [31–33].

El predictor consiste en un pequeño buffer de da-

Parámetros de memoria			
Frecuencia del procesador	2.8GHz	Jerarquía TLB	Exclusive
L1 TLBs separadas instr & datos	8 conjuntos, 4 vías	L1 TLB tiempo de acierto	1 ciclo
L2 TLB combinada	128 conjuntos, 4 vías	L2 TLB tiempo de acierto	2 ciclos
Timeout adquisición de tokens	1200 ciclos	TPB	32 conjuntos, 4 vías
TPB tiempo de acierto	1 ciclo	Tamaño de página	4KB (64 bloques)
Jerarquía caché	No inclusiva	Tamaño de bloque caché	64 bytes
L1 cachés separadas instr & data	64KB, 4 vías	L2 cache combinada compartida	1MB/tile, 8 vías
L1 cache tiempo de acierto	1 (tag) y 2 (tag+datos) ciclos	L2 cache tiempo de acierto	2 (tag) y 6 (tag+datos) ciclos
Caché de directorio	256 sets, 4 ways	Directorio tiempo de acierto	1 ciclo
Tiempo de acceso a memoria	160 ciclos		
Parámetros de red			
Topología	Malla 2D (4x4)	Mecanismo de encaminamiento	Determinístico X-Y
Tamaño de flit	16 bytes		
Tiempo de encaminamiento, switch y enlace	2, 2 y 2 ciclos	Tamaño de mensajes de datos y control	5 flits y 1 flit

TABLA II: Parámetros del sistema base.

tos situado en paralelo con la L2 TLB llamado Token Predictor Buffer (TPB) que almacena la ID del proceso, la dirección virtual y la ID del núcleo. Tras ceder tokens en desalojos no silenciosos, el receptor se convierte en un propietario potencialmente conocido y es almacenado en la TPB. Si ocurre un fallo en la L1 TLB, se consultan en paralelo la L2 TLB y la TPB. Si la L2 TLB falla y la TPB mantiene información de un potencial propietario para la misma página, se envía un mensaje punto a punto y la entrada de TPB es desalojada. Si la TLB que recibe la solicitud es aún propietaria, responde con la traducción y la información de compartición. En caso contrario, si la TLB consultada no es ya propietaria, responde negativamente y el mecanismo habitual es invocado, inundando la red de interconexión y consultando en paralelo la tabla de páginas.

IV. Desactivación de la Coherencia con TokenTLB

Para probar los beneficios del esquema de clasificación que proporciona TokenTLB, lo aplicamos al mecanismo de *Desactivacion de la Coherencia*, el cual ha demostrado una mejora en el uso del directorio bajo un esquema de clasificación no adaptativo [6]. De acuerdo con este esquema, tan solo los bloques pertenecientes a página compartidas y escritas (SW) requieren que se mantenga la coherencia, lo cual incrementa dramáticamente lo accesos a datos cuya coherencia no es necesaria.

Como se ha indicado previamente, la clasificación en TokenTLB puede evolucionar naturalmente de compartido a privado y viceversa múltiples veces en cada tiempo de generación de cada página en cada TLB. A diferencia de otros mecanismos de clasificación, TokenTLB detecta transiciones a privado inmediatamente (tan pronto como la TLB obtiene todos los N tokens), lo cual permite detectar eficientemente periodos privados de vidas globales.

Sin embargo, al aplicar TokenTLB a la desactivación de la coherencia, hay que tener especial cuidado con sus implicaciones. Incluso en un estado coherente (SW), la clasificación podría transitar de nuevo a no coherente durante el mismo tiempo de generación de la página, siempre que una entrada recupere todos sus tokens y se convierta de nuevo en privada (PW).

Por tanto, en una reclasificación de privado o solo lectura a compartido y escrito, todas las copias no coherentes de dicha página deben ser desalojadas de las caches privadas de todos los núcleos compartidores. Por contra, al reclasificar a no coherente, no se requieren acciones especiales.

V. Entorno de Simulación

Nuestra propuesta esta evaluada a través de simulación de sistema completo usando Virtutech Simics [34] junto con el conjunto de herramientas de Wisconsin GEMS [35], lo que habilita la simulación detallada de sistemas multiprocesador. La red de interconexión se ha modelado mediante el simulador GARNET [36]. La simulación consiste en una arquitectura CMP de 16 núcleos implementando un mecanismo de coherencia caché basado en directorio con los parámetros mostrados en la Tabla II, en la arquitectura considerada base para la evaluación. La latencia de la L2 TLB asume cuatro referencias a memoria para atravesar la tabla de páginas, como ocurre con el espacio de direcciones virtual de 28 bits del x86-64. Las latencias de las cachés y la TLB, así como su consumo, se han calculado usando la herramienta CACTI [37] asumiendo tecnología de 32nm. A través de la experimentación se ha determinado un tiempo de 1200 ciclos para la adquisición de tokens, lo cual ofrece un buen equilibrio entre prestaciones y tráfico de red.

Para la evaluación se han utilizado una amplia variedad de cargas de trabajo paralelas de varias suites, cubriendo así diferentes patrones y grados de compartición. *Barnes* (8192 bodies, 4 time steps), *Cholesky* (tk15.O), *FFT* (64K complex doubles), *Ocean* (258 × 258 ocean), *Radiosity* (room, -ae 5000.0 -en 0.050 -bf 0.10), *Raytrace-opt* (teapot), *Volrend* (head), and *Water-NSQ* (512 molecules, 4 time steps) son de la SPLASH-2 benchmark suite [38]. *Tomcatv* (256 points, 5 time steps) y *Unstructured* (Mesh.2K, 5 time steps) son dos benchmarks científicos. *FaceRec* (script), and *SpeechRec* (script) pertenecen al ALPBenchs suite [39]. *Blackscholes* (simmedium), *Swaptions* (simmedium) y *x264* (simsmall) pertenecen a PARSEC [40]. Finalmente, *Apache* (1000 HTTP transactions) y *SPEC-JBB* (1600 transactions) son dos cargas de trabajo comerciales [41]. Raytrace-opt optimiza la aplicación Raytrace original al eliminar la adquisición del cierre de exclusión para un ID de ray que no esta en uso actualmente. Todos los resultados experimentales mostra-

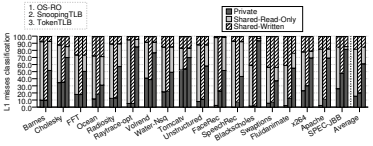

Fig. 3: Proporción de fallos de L1 de datos clasificados.

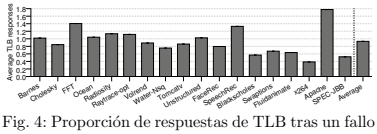

Fig. 4: Proporción de respuestas de TLB tras un fallo en la L2 TLB.

dos corresponden a las fases paralelas de las aplicaciones.

VI. Resultados de la Evaluación

Primeramente los resultados muestran cómo el mecanismo de clasificación TokenTLB se comporta comparado a anteriores mecanismos, incluyendo un estudio de sensibilidad para el Token Predictor Buffer y hasta cuatro tamaños distintos. Seguidamente, se introduce la desactivación de la coherencia en el análisis para comparar los beneficios que se obtienen al aplicar los diversos mecanismos del estudio.

A. Clasificación completamente Adaptativa

Ésta sección muestra la precisión y eficiencia de la clasificación con TokenTLB comparada con propuestas previas.

Datos Privados y de solo Lectura. El porcentaje de páginas privadas y compartidas es una buena métrica general para medir la bondad de un mecanismo de clasificación. Sin embargo, ésta métrica es injusta para los mecanismos de clasificación adaptativos, donde las páginas se reclasifican frecuentemente a privado, ya que dicha reclasificación no se muestra en las gráficas. Además, esta situación se ve favorecida por el hecho de que TokenTLB desbloquee el acceso a la página tras la primera respuesta de otra TLB, lo cual acelera la resolución de fallos de TLB, pero que puede resultar en accesos compartidos a páginas con tokens en vuelo.

A pesar de que un mecanismo adaptativo puede reclasificar páginas libremente, los bloques no son reclasificados individualmente durante su tiempo de generación local. Por tanto, cuanto más tiempo se mantiene una página como privada o de solo lectura, más fallos en la caché L1 serán clasificados como tal. La Figura 3 muestra la clasificación de los accesos a la caché L1 de datos. Se puede observar como los fallos en la caché de datos considerados como privados se incrementa ámpliamente al usar *TokenTLB*, ya que, a diferencia de *SnoopingTLB*, la reclasificación ocurre de forma natural durante el tiempo de generación de la página. En concreto, *TokenTLB* es capaz de clasificar un 61.1 % de fallos en la caché L1 de datos de media como privado, hasta un 40.8 % más que *SnoopingTLB*. También es importante observar que, a diferencia de *SnoopingTLB*, *TokenTLB* y *OS-RO* pueden reconocer páginas de solo lectura, lo cual representa un 24.4 % de los fallos de la caché L1 en el caso de *TokenTLB*, mejorando enormemente la precisión de la clasificación.

Intercambio de tokens entre TLBs. Uno de los beneficios clave de TokenTLB sobre mecanismos de clasificación previos es cómo maneja las transferencias entre TLBs, de forma que obtiene sus beneficios al mismo tiempo que limita el número de respuestas requeridas. Como resultado, el tráfico TLB se reduce y se evita el bloqueo del sistema esperando respuestas. La Figura 4 representa el número de respuestas enviadas tras cada fallo de TLB al usar *TokenTLB*. Mientras que los mecanismos de resolución entre TLBs como *SnoopingTLB* requieren $N-1$ respuestas invariablemente, *TokenTLB* requiere de media solamente 0.93 respuestas por fallo de TLB. Dicha media es menor a uno debido al hecho de que, al usar TokenTLB, no se esperan respuestas si los tokens están en la tabla de páginas. En algunos casos, como *Apache* o *SpeechRec* la media supera ámpliamente a una respuesta por fallo. TokenTLB permite que exista más de un propietario de la traducción en otras TLBs, ya que los desalojos pueden ser aceptados por la primera TLB válida en el anillo de desalojo, y por tanto en estos casos, dos o más respuestas puede enviarse como consecuencia de un fallo TLB.

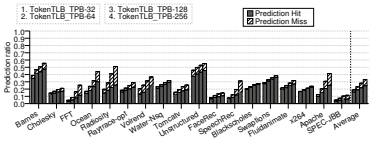

Fig. 5: Ratio de acierto para predicciones de la TPB.

Efectividad del predictor de tokens. Esta sección analiza brevemente el Token Predictor Buffer (TPB), que es concebido para evitar recurrir a mensajes de difusión tras fallos de TLB, siempre que haya un potencial propietario conocido. Por tanto, el TPB tiene un impacto sobre el tráfico y el consumo de la red, pero no en el tiempo de ejecución. La TPB es sencillamente un pequeño buffer de 4 vías, evaluado con hasta cuatro tamaños distintos (32, 64, 128 y 256 entradas).

La Figura 5 muestra la proporción de predicciones exitosas y fallidas (es decir, la cantidad de TLBs remotas que son propietarias de tokens -o no- en el momento de ser solicitadas en una petición punto a punto) con respecto al total de fallos de TLB. Se puede observar que al incrementar el tamaño de la TPB tiene un efecto positivo en la precisión de las predicciones. En concreto, una TPB de 256 entradas evita un 24.7 % de difusiones de un total de casi 33 % de intentos de predicción de media. En algunos casos, como *Barnes* o *Unstructured*, alrededor de un 45 % de difusiones de TLB son prevenidas al usar la TPB.

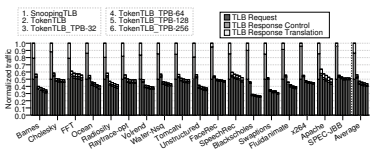

Fig. 6: Tráfico de red TLB.

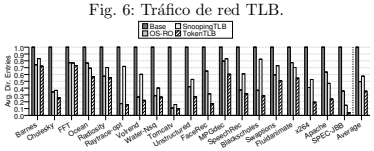

Fig. 7: Utilización del Directorio por ciclo.

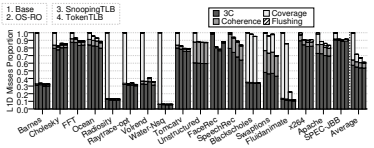

Fig. 8: Fallos en la caché de datos L1 según su causa.

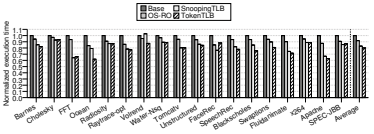

Fig. 9: Tiempo de ejecución.

La Figura 6 detalla el tráfico TLB comparado al base *SnoopingTLB*. Se puede observar como *TokenTLB* incrementa ligeramente el tráfico de solicitud TLB comparado con *SnoopingTLB* debido a los desalojos no silenciosos. Sin embargo, es superado ámpliamente debido a la reducción en el tráfico de respuesta y traducción en la TLB. Se reduce así el tráfico de TLB en general hasta un 44 %. Además, el uso del TPB reduce el tráfico TLB un poco más. En concreto, la mayor TPB en consideración (256 entradas) reduce el tráfico TLB de petición prácticamente en un 20 % con respecto a *TokenTLB*, para una reducción total de 56.3 %.

Como conclusión, *TokenTLB* detecta la condición de escritura en granularidad de página, al mismo tiempo que logra una detección de privados similar a otros mecanismos adaptativos. Aparentemente, sin embargo, pierde precisión en la clasificación. Sin embargo, el punto fuerte de *TokenTLB* es el beneficio obtenido de la reclasificación inmediata de compartido a privado (adaptabilidad completa), incrementando dramáticamente el número de bloques clasificados como privado o de solo lectura en el momento del fallo sobre cualquier otro mecanismo. Finalmente, el tráfico de TLB se ve enormemente reducido al usar nuestro mecanismo. La inclusión de la TPB reduce el tráfico total de la TLB más de la mitad, favoreciendo la escalabilidad del sistema.

B. Desactivación de la Coherencia

El mecanismo de desactivación de la coherencia se ve beneficiado por la precisión de la clasificación. Se puede beneficiar también tanto de la dicotomía privado-compartido como de la detección de escrita. Ésta sección muestra los beneficios de la aplicación de TokenTLB a la desactivación de la coherencia frente a otros mecanismos de clasificación.

El mantenimiento de la coherencia se desactiva si un bloque es considerado como no coherente por el mecanismo de clasificación. En el caso de *SnoopingTLB*, el cual solo caracteriza los accesos a memoria como privados o compartidos, todas las páginas compartidas son coherentes. Por otro lado, tanto *OS-RO* como *TokenTLB* también distinguen páginas escritas, así que solo las páginas compartidas y escritas son consideradas coherentes. Por último, *SnoopingTLB* y *TokenTLB* son mecanismos adaptativos, transitando de privado a compartido y viceversa. La Figura 7 muestra el número medio de entradas de directorio requeridas por ciclo para la misma configuración pero sin desactivar la coherencia. Reduciendo la utilización del directorio es el objetivo principal de la desactivación de la coherencia y es muy dependiente de la precisión de la clasificación. Se puede observar como *OS-RO* mejora la utilización del directorio un 52.8 %, reduciendo incluso más que *SnoopingTLB* debido a la detección de escritura. Por último, *TokenTLB* mejora enormemente la utilización del directorio, ya que requiere tan solo un 34.1 % de las entradas de directorio por ciclo en comparación con *Base*.

Clasificar más páginas como no coherentes y mejorar el uso del directorio reduce los fallos de la caché L1, especialmente los fallos de cobertura. La Figura 8 clasifica los fallos de datos basándose en su causa, y por tanto muestra en qué grado se reducen los fallos de cobertura de la L1 según el mecanismo de clasificación al aplicarlo a la desactivación de la coherencia. Sin embargo, *TokenTLB* es capaz de eliminar prácticamente todos los fallos de cobertura (Coverage) en todos los casos. Por otro lado, los fallos de cobertura empleando *OS-RO* representa aún un 13.4 % de los fallos de caché L1. Finalmente, *SnoopingTLB* esta en un punto intermedio, ya que aún sufre un 8.7 % de fallos de cobertura.

Reducir los fallos de caché y acelerar la traducción de páginas a través de transferencias de TLB a TLB tiene un impacto positivo en el tiempo de ejecución, el cual se mejora hasta un 20 % al usar *TokenTLB* comparado con *Base*, como se muestra en la Figurae 9. Además, el tiempo de ejecución se reduce un 3 % comparado con *SnoopingTLB*, ya que *TokenTLB* desbloquea el acceso a la página antes tras un fallo de TLB y los fallos de caché son reducir en un mayor grado. Puesto que *OS-RO* no se beneficia de la resolución de fallos entre TLBs sus ganancias se producen solamente debidas a la desactivación de la coherencia, reduciendo el tiempo de ejecución un 8.8 %.

Nuestra propuesta conlleva también una reducción

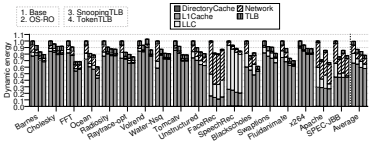

Fig. 10: Consumo dinámico de energía normalizado al sistema base.

en el consumo de energía dinámica de la caché, como se muestra en la Figura 10. *TokenTLB* reduce el consumo global hasta un 21.9 % comparado al sistema base, particularmente el consumo de la L1 caché, ya que la presión se reduce a través de la desactivación de la coherencia. El consumo de la red también se reduce significativamente, aunque proporcionalmente al consumo total el impacto de la misma se diluye. En comparación, *TokenTLB* reduce el consumo dinámica en un 5.7 % con respecto a *SnoopingTLB* y un 9 % con respecto a *OS-RO*.

En resumen, aplicar TokenTLB a la desactivación de la coherencia mejora la escalabilidad del sistema. Reduce enormemente la utilización del directorio debido a sus propiedades. Como consecuencia, TokenTLB permite estructuras de directorio más pequeñas y eficientes, reduciendo los fallos en la caché de L1, el tráfico de la red, el consumo dinámico y mejora también el tiempo de ejecución sobre cualquier otro mecanismo de clasificación previo.

VII. Conclusiones

Este trabajo propone *TokenTLB*, un mecanismo de clasificación novedoso basado en las TLBs y en la cuenta e intercambio de tokens a través de mensajes entre TLBs, en los que solo las TLBs que son propietarias de la traducción deben contestar. Al usar los tokens para la clasificación en lugar de usarlos para la coherencia se evita la necesidad de peticiones persistentes. Además, la cuenta de tokens es muy eficiente al realizar una clasificación adaptativa en un esquema privado-compartido, detectando de forma natural fases privadas de páginas compartidas durante su tiempo de generación. Finalmente, la dicotomía clásica de la clasificación es extendida al permitir la detección de escritura por vez primera en un mecanismo adaptativo. Al predecir los propietarios de los tokens y enviar mensajes punto a punto se mejora la escalabilidad del sistema. TokenTLB presenta todas las características deseables para un esquema de clasificación: ésta es conocida antes del acceso a la caché (*a-priori*), detecta accesos de solo lectura (*read-only*), es *adaptativa* y *precisa*.

Agradecimientos

Este trabajo ha sido financiado por MINECO y la Comisión Europea (fondos FEDER) bajo el proyecto TIN2015-66972-C5-1-R/3-R y la *Fundación Séneca, Agencia de Ciencia y Tecnología de la Región de Murcia* bajo el proyecto *Jóvenes Líderes en Investigación* 18956/JLI/13.

Referencias

[1] 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.

[2] Michael Ferdman, Pejman Lotfi-Kamran, Ken Balet, and Babak Falsafi, "Cuckoo directory: A scalable directory for many-core systems," in *17th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2011, pp. 169–180.

[3] 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.

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

[5] Mohammad Alisafaee, "Spatiotemporal coherence tracking," in *45th IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, Dec. 2012, pp. 341–350.

[6] Blas Cuesta, Alberto Ros, María E. Gómez, Antonio Robles, and José Duato, "Increasing the effectiveness of directory caches by avoiding the tracking of non-coherent memory blocks," *IEEE Transactions on Computers (TC)*, vol. 62, no. 3, pp. 482–495, Mar. 2013.

[7] Mahdad Davari, Alberto Ros, Erik Hagersten, and Stefanos Kaxiras, "An efficient, self-contained, on-chip, directory: DIR$_1$-SISD," in *24th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Oct. 2015, pp. 317–330.

[8] 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.

[9] Albert Esteve, Alberto Ros, María E. Gómez, Antonio Robles, and José Duato, "Efficient tlb-based detection of private pages in chip multiprocessors," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 27, no. 3, pp. 748–761, Mar. 2016.

[10] Nikos Hardavellas, Michael Ferdman, Babak Falsafi, and Anastasia Ailamaki, "Reactive NUCA: Near-optimal block placement and replication in distributed caches," in *36th Int'l Symp. on Computer Architecture (ISCA)*, June 2009, pp. 184–195.

[11] Hemayet Hossain, Sandhya Dwarkadas, and Michael C. Huang, "POPS: Coherence protocol optimization for both private and shared data," in *20th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Oct. 2011, pp. 45–55.

[12] Daehoon Kim, Jeongseob Ahn, Jaehong Kim, and Jaehyuk Huh, "Subspace snooping: Filtering snoops with operating system support," in *19th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Sept. 2010, pp. 111–122.

[13] Konstantinos Koukos, Alberto Ros, Erik Hagersten, and Stefanos Kaxiras, "Building heterogeneous unified virtual memories (uvms) without the overhead," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 13, no. 1, pp. 1:1–1:22, Mar. 2016.

[14] Yong Li, Ahmed Abousamra, Rami Melhem, and Alex K. Jones, "Compiler-assisted data distribution for chip multiprocessors," in *19th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Sept. 2010, pp. 501–512.

[15] Yong Li, Rami Melhem, and Alex K. Jones, "PS-TLB: Leveraging page classification information for fast, scalable and efficient translation for future cmps," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 9, no. 4, pp. 28:1–25:21, Jan. 2013.

[16] Yong Li, Rami G. Melhem, and Alex K. Jones, "Practically private: Enabling high performance cmps through compiler-assisted data classification," in *21st Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Sept. 2012, pp. 231–240.

[17] Seth H. Pugsley, Josef B. Spjut, David W. Nellans, and Rajeev Balasubramonian, "SWEL: Hardware cache coherence protocols to map shared data onto shared caches," in *19th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Sept. 2010, pp. 465–476.

[18] Alberto Ros, Blas Cuesta, María E. Gómez, Antonio Robles, and José Duato, "Temporal-aware mechanism to

detect private data in chip multiprocessors," in *42nd Int'l Conf. on Parallel Processing (ICPP)*, Oct. 2013, pp. 562–571.

[19] Alberto Ros, Mahdad Davari, and Stefanos Kaxiras, "Hierarchical private/shared classification: the key to simple and efficient coherence for clustered cache hierarchies," in *21th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2015, pp. 186–197.

[20] Alberto Ros and Alexandra Jimborean, "A dual-consistency cache coherence protocol," in *29th Int'l Parallel and Distributed Processing Symp. (IPDPS)*, May 2015, pp. 1119–1128.

[21] Alberto Ros and Alexandra Jimborean, "A hybrid static-dynamic classification for dual-consistency cache coherence," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. PP, no. 99, Feb. 2016.

[22] 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.

[23] Milo M.K. Martin, *Token Coherence*, Ph.D. thesis, University of Wisconsin-Madison, Dec. 2003.

[24] 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.

[25] 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.

[26] Alberto Ros and Stefanos Kaxiras, "Complexity-effective multicore coherence," in *21st Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Sept. 2012, pp. 241–252.

[27] Bogdan F. Romanescu, Alvin R. Lebeck, Daniel J. Sorin, and Anne Bracy, "UNified instruction/translation/data (UNITD) coherence: One protocol to rule them all," in *16th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2010, pp. 1–12.

[28] Shekhar Srikantaiah and Mahmut Kandemir, "Synergistic tlbs for high performance address translation in chip multiprocessors," in *43rd IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, Dec. 2010, pp. 313–324.

[29] Arun Raghavan, Colin Blundell, and Milo M.K. Martin, "Token tenure: PATCHing token counting using directory-based cache coherence," in *41th IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, Nov. 2008, pp. 47–58.

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

[31] 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 *ACM/IEEE Conf. on Supercomputing (SC)*, Nov. 2002, pp. 1–12.

[32] Alberto Ros, Manuel E. Acacio, and José M. García, "DiCo-CMP: Efficient cache coherency in tiled CMP architectures," in *22nd Int'l Parallel and Distributed Processing Symp. (IPDPS)*, Apr. 2008, pp. 1–11.

[33] Alberto Ros, Manuel E. Acacio, and José M. García, "Dealing with traffic-area trade-off in direct coherence protocols for many-core cmps," in *8th Int'l Conf. on Advanced Parallel Processing Technologies (APPT)*, Aug. 2009, pp. 11–27.

[34] Peter S. Magnusson, Magnus Christensson, Jesper Eskilson, Daniel Forsgren, Gustav Hallberg, Johan Hogberg, Fredrik Larsson, Andreas Moestedt, and Bengt Werner, "Simics: A full system simulation platform," *IEEE Computer*, vol. 35, no. 2, pp. 50–58, Feb. 2002.

[35] 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.

[36] Niket Agarwal, Tushar Krishna, Li-Shiuan Peh, and Niraj K. Jha, "GARNET: A detailed on-chip network model inside a full-system simulator," in *IEEE Int'l Symp. on Performance Analysis of Systems and Software (ISPASS)*, Apr. 2009, pp. 33–42.

[37] Shyamkumar Thoziyoor, Naveen Muralimanohar, Jung Ho Ahn, and Norman P. Jouppi, "Cacti 5.1," Tech. Rep. HPL-2008-20, HP Labs, Apr. 2008.

[38] 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.

[39] Man-Lap Li, Ruchira Sasanka, Sarita V. Adve, Yen-Kuang Chen, and Eric Debes, "The ALPBench benchmark suite for complex multimedia applications," in *Int'l Symp. on Workload Characterization (IISWC)*, Oct. 2005, pp. 34–45.

[40] 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.

[41] Alaa R. Alameldeen, Carl J. Mauer, Min Xu, Pacia J. Harper, Milo M.K. Martin, Daniel J. Sorin, Mark D. Hill, and David A. Wood, "Evaluating non-deterministic multi-threaded commercial workloads," in *5th Workshop On Computer Architecture Evaluation using Commercial Workloads (CAECW)*, Feb. 2002, pp. 30–38.

