Departament d'Informàtica de Sistemes i Computadors

Universitat Politècnica de València

Análisis y evaluación de prestaciones del empleo de técnicas de caché para acelerar el proceso de búsqueda en tablas de páginas

Trabajo Fin de Master

Máster en Ingeniería de Computadores

Autor: Alberto Ramis Fuambuena

Director: Antonio Robles Martínez

4 de septiembre de 2014

Resumen

El proceso de traducción de direcciones virtuales a direcciones físicas llevado a cabo por la MMU de los procesadores requiere varios accesos a memoria para recuperar la entrada de cada uno de los niveles del árbol en que se estructura la tabla de páginas. Las TLBs (*Translation Lookaside Buffers*) evitan estos accesos almacenando las direcciones de páginas que se han traducido más recientemente. Sin embargo, cada vez que se produce un fallo en la TLB, se requiere llevar a cabo el proceso completo de traducción accediendo múltiples veces a la tabla de páginas en memoria. El proceso de traducción estaba inicialmente soportado por software a cargo del SO (manejador de excepción producida con cada fallo de TLB), lo cual ralentizaba enormemente la resolución de los fallos de TLB.

Afortunadamente, los procesadores actuales incorporan soporte *hardware* al proceso de traducción, mediante un autómata denominado Page Table Walker, el cual se encarga de ir accediendo a memoria a los distintos niveles de traducción de la tabla de páginas. No obstante, cada uno de estos accesos a memoria principal entraña una elevada latencia en comparación con los accesos a los niveles de caché, los cuales aparecen cada vez más rápidos en términos relativos, penalizando de manera significativa las prestaciones globales del sistema. Una manera efectiva para acelerar este proceso de traducción es introducir una cache dentro (o cerca) de la MMU para almacenar las traducciones parciales correspondientes con cada uno de los niveles de traducción de la tabla de páginas.

Las propuestas actuales en este sentido ponen de manifiesto las ventajas del uso de la *Page Table Walk Cache* (PTWC) en cada núcleo. En este trabajo, además de analizar dichas ventajas, analiza las que se derivan, en el contexto de un CMP, del empleo de una configuración de PTWC compartida entre los distintos núcleos que componen el CMP, al objeto de aprovechar la compartición de memoria que se puede derivar de la ejecución de aplicaciones paralelas. Dicho análisis trata de mostrar las ventajas del empleo de una PTWC compartida frente a una configuración de PTWC privadas.

Palabras clave: TLB, MMU, PTWC compartida, Caché, Memoria Virtual.

Índice general

Capítulo 1

Introducción

1.1. Motivación

La memoria virtual es una abstracción que permite a los sistemas operativos multitarea asignar direcciones contiguas del espacio de direcciones virtual a cada tarea. El sistema operativo maneja la asignación del espacio físico para cada tarea específica del espacio virtual, almacenando la asignación en unas tablas jerárquicas llamadas Tabla de Páginas (*Page Table*).

En el proceso de traducción, muchas arquitecturas organizan la tabla de páginas en una estructura multinivel que contiene la asignación de direcciones, donde cada nivel contiene pequeñas porciones de la tabla de páginas unidas mediante una tabla de páginas mayor formando un árbol. Cada nivel del árbol tiene un puntero al siguiente nivel, menos el último que contiene la traducción. Las unidades de manejo de memoria (*Memory Management Unit*, o MMU), usan los *buffers* de traducción rápida o TLBs (*Translation Lookaside Buffers*), para almacenar las traducciones, alcanzando un 99 % de éxito.

Cuando hay un error en la TLB, los procesadores actuales hacen un *Page Table Walk* (se recorren los niveles de la tabla de páginas) para obtener la traducción de la dirección virtual a física, lo que suele costar varios accesos a memoria. Para reducir este impacto, AMD e Intel incorporaron unas pequeñas cachés privadas para cada núcleo situadas dentro de la MMU, denominadas *Page Table Walk Cache* (**PTWC**).

Recientemente, en el contexto del proyecto europeo *vIrtical* [1], el cual se centra en el diseño HW/SW de sistemas en chip (SoC) heterogéneos multinúcleo

virtualizados para aplicaciones empotradas, y con el objeto de aprovechar la compartición de memoria que pudiera derivarse de la paralelización de aplicaciones, se ha propuesto una PTWC compartida para todos los núcleos del SoC.

Junto con la propuesta, se realizó una evaluación preliminar de sus ventajas respecto al empleo de una PTWC privada. Dicho estudio estaba enfocado al procesador Cortex A15 de ARM [3] y supeditado a las características específicas del entorno en que se desarrolla dicho proyecto, dejando pendiente la realización de un estudio de carácter más general, y al mismo tiempo de más profundidad, de la PTWC compartida.

1.2. Objetivos

El objetivo fundamental de este trabajo es el análisis en profundidad de las prestaciones que se pueden derivar del empleo de una configuración de PTWC compartida en el seno de un CMP, a través de su comparación con la configuración tradicional de PTWC privada. Con ello se persiguen tres objetivos particulares, a saber:

- Ampliar los estudios preliminares previos [12] llevados a cabo en el contexto del proyecto *vIrtical*, analizando otras arquitecturas de procesador y llevando a cabo una evaluación de carácter más exhaustivo en términos de rangos de variación de los parámetros de configuración del modelo de PTWC analizado.
- Familiarizarse con la instalación, configuración y manejo de herramientas avanzadas de modelado de sistemas multiprocesador y de simulación dirigida por ejecución de los mismos [2] [8].
- Familiarizarse con los procedimientos de evaluación y análisis de prestaciones de los modelos simulados, la discusión de los resultados y su presentación mediante la elaboración de la correspondiente memoria técnica.

Capítulo 2

Memoria Virtual y sus componentes

2.1. Sistema de gestión de memoria

La memoria es un componente básico en cualquier ordenador, y uno de los elementos principales que caracterizan un proceso es la memoria que utiliza. Ésta está lógicamente separada de la de cualquier otro proceso del sistema. Un proceso no debe acceder al espacio de memoria asignado a otro proceso, ya que en caso de hacerlo se comprometería la seguridad y estabilidad del sistema.

Los objetivos del sistema de gestión de memoria consisten en ofrecer a cada proceso un espacio lógico propio, proporcionar protección entre procesos, permitir que los procesos compartan memoria, dar soporte a las distintas regiones del proceso, maximizar el rendimiento del sistema y proporcionar a los procesos mapas de memoria muy grandes.

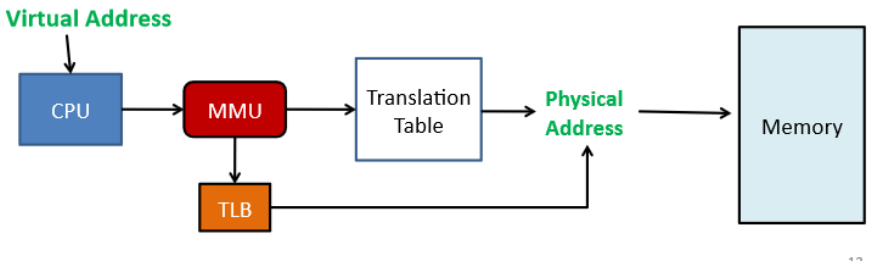


Figura 2.1: Proceso de traducción de dirección lógica a física

El tamaño combinado del programa, datos y pila puede exceder la cantidad de memoria física disponible. El sistema operativo guarda aquellas partes del programa en uso de manera concurrente en memoria principal y el resto en el disco duro. Mientras un programa espera que se cargue en memoria principal desde el disco duro otra parte del mismo, la CPU se puede asignar a otro proceso.

El espacio de direcciones de un proceso es el conjunto de direcciones a las que hace referencia. Los espacios de direcciones involucrados en la gestión de la memoria son de tres tipos:

- *Espacio de direcciones lógicas del procesador.* Es el rango de direcciones al que puede acceder un procesador.
- *Espacio de direcciones lógicas de un proceso.* Es el espacio que ocupa un proceso en ejecución.
- *Espacio de direcciones físicas.* Son aquellas que hacen referencia alguna posición de la memoria física. Se obtienen después de aplicar una transformación por parte de la MMU (*Manegement Memory Unit*).

2.2. Unidad de Manejo de Memoria

La MMU o Unidad de Manejo de Memoria es una parte del procesador cuyas funciones son:

- *Convertir las direcciones lógicas emitidas por los procesos en direcciones físicas*
- *Comprobar que la conversión se puede realizar.* Ya que una dirección lógica podría no tener una dirección física asociada (como en el caso de que la página a la que se quiere acceder haya sido intercambiada con el sistema de almacenamiento secundario)
- *Comprobar que el proceso que intenta acceder a una posición de memoria determinada tenga permisos para hacerlo.*

Para realizar la traducción de direcciones se utiliza la tabla de páginas. Esta tabla contiene la dirección base de cada página en memoria física y es mantenida por el sistema operativo. En la mayoría de sistemas operativos se asigna para cada proceso del sistema una tabla de páginas propias, permitiendo que cada proceso pueda usar el rango completo de direcciones lógicas. La implementación *hardware* puede hacerse de varias maneras. Se puede usar un

conjunto de registros dedicados de muy alta velocidad en el caso mas simple, siempre que las tablas de páginas sean de tamaño razonablemente pequeño (por ejemplo, 256 entradas). Si aumenta tamaño del espacio de direcciones virtual esta implementación resulta inviable, como sucede actualmente.

Para ese caso se utiliza una tabla de páginas multinivel. Por ejemplo, podemos utilizar un registro base de la tabla de páginas (PTBR, *Page Table Base Register*) que apunta a la tabla de páginas. Para cambiar la tabla de páginas solo hace falta cambiar un único registro, reduciendo así el tiempo del cambio de contexto. El problema de este método es el tiempo requerido para acceder a una ubicación de memoria del usuario. Con este esquema se requieren varios accesos a memoria.

Con objeto de evitar que cada acceso a memoria se traduzca en un acceso adicional a la tabla de páginas residente en memoria, lo cual duplicaría los accesos a memoria y haría prácticamente inviable cualquier aplicación, la MMU de los procesadores incluye necesariamente una pequeña memoria cache, a modo de *buffer* de traducción rápida de direcciones, a la que se conoce como *Translation Lookaside Buffer* (TLB), la cual almacena las más recientes traducciones de página que han tenido lugar. Con ello se evita el accesos a la tabla de páginas por parte del sistema operativo, acelerando así el proceso de traducción desde el *hardware*.

Como el objetivo es maximizar la velocidad, la caché que constituye la TLB tiene necesariamente un número muy reducido de entradas a fin de ofrecer bajos tiempos de acceso. El número de entradas se haya comprendido comúnmente entre 64 y 512 entradas. Ello permite implementar TLBs con elevado grado de asociatividad, incluso totalmente asociativas, lo cual suele beneficiar a la tasa de aciertos.

Existen diferentes arquitecturas de TLB, pudiendo consistir éstas de uno o más niveles (como la jerarquía de cache) y tener estos un carácter privado o compartido entre varios núcleos. El acceso a la TLB suele ser previo al acceso al primer nivel de cache (caso de una cache con etiquetas físicas). Al ser habitual en los procesadores actuales un primer nivel de cache segregado (instrucciones y datos), también deben existir TLBs de instrucciones y datos independientes al objeto de poder acceder en paralelo (arquitectura *Harvard*).

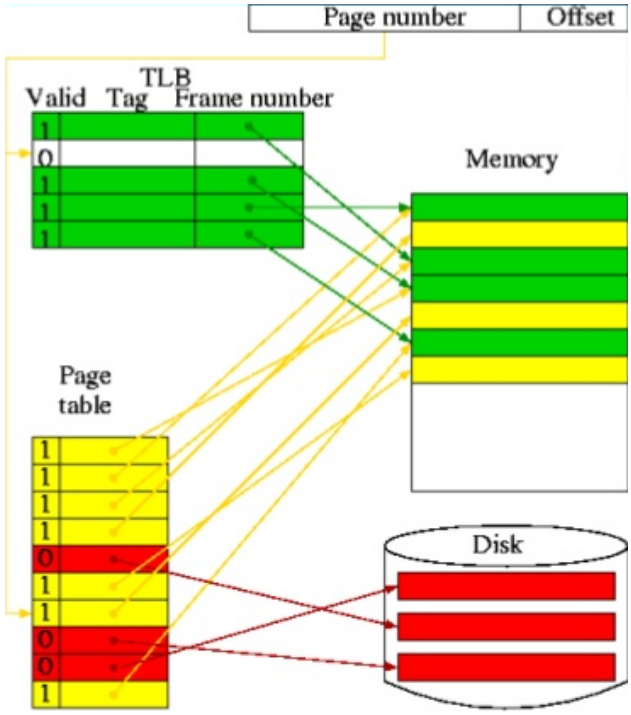


Figura 2.2: TLB en la estructura de la memoria virtual.

La TLB es crucial para el rendimiento del sistema debido a la gran penalización en caso de fallo. Cada vez que se produzca un fallo en esta memoria, se deberá consultar la tabla de páginas, y en caso de que la página que contiene la traducción no se encuentre en memoria principal, la penalización será mucho mayor. En la figura 2.2 se observa con colores. Los aciertos en la TLB (verde) son los que menos latencia tienen, seguidos por la búsqueda en la tabla de páginas en memoria principal (amarillo), que tiene un sobrecoste de un acceso adicional a memoria. La traducción de esta página se colocará en la TLB para que los futuros accesos a esta página se resuelvan directamente por parte de la TLB. Estas entradas serán eliminadas de la TLB dependiendo de el algoritmo de reemplazo utilizado. En el peor de los casos, habrá que buscar la página en memoria secundaria (rojo) y actualizar debidamente las tablas de páginas antes de introducir la correspondiente entrada en la TLB.

La TLB solo mantiene la traducción de páginas correspondiente a un proceso, y durante un cambio de contexto es necesario invalidar su contenido, evitando que se utilicen traducciones correspondientes a otros procesos. El proceso que entra en el procesador encuentra la TLB vacía, llenándose conforme el proceso acceda a sus páginas.

2.3. Proceso de consulta de las tablas de páginas

En caso de fallo en la TLB se ha iniciar un proceso de consulta en la tabla de páginas en búsqueda de la traducción de la dirección lógica. Este proceso de consulta puede ser más o menos complejo, dependiendo de la estructura de las tablas de páginas. Durante varias generaciones de procesadores x86, desde el *Intel 80386* hasta los *Pentium*, la *Page Table* tenía como mucho **dos** niveles. A raíz de ello, en el caso de ocurrir un fallo de TLB solo había en el peor de los casos dos accesos a memoria principal. Sin embargo, conforme el espacio de direcciones físicas y virtuales soportados por los procesadores con arquitectura x86 crecía en tamaño, la profundidad máxima del árbol crecía, primero a **tres** niveles con los *Pentium Pro* para soportar direcciones físicas de 36 bits en una entrada de la tabla de páginas, y mas recientemente **cuatro** niveles en los *AMD Opteron* para soportar un espacio de direcciones virtuales de 48 bits. Esto ha significado que cada década que ha pasado desde la llegada del *80386*, se ha incrementado en un nivel la profundidad de la Page Table. Más recientemente, a consecuencia del soporte a virtualización ofrecido por algunos procesadores, el número de niveles de traducción a llegado hasta 15, como es el caso del Cortex A15 de ARM [3].

El proceso de consulta de la Tabla de Páginas solía correr a cargo del sistema operativo, lo que introducía una sobrecarga importante en el procesador. En la actualidad, la MMU de los procesadores proporciona un soporte *hardware* a la consulta de las tablas a través de un autómata al que se conoce como *Page Table Walker* (PTW), lo que elimina la sobrecarga del procesador. Sin embargo, ello no evita los múltiples accesos a memoria, tanto más numerosos cuanto mayor el número de niveles de traducción ofrecidos por las tablas de páginas.

Se ha demostrado que el impacto de un fallo en la TLB provoca una disminución del rendimiento entre el 5 % y el 14 %. Y tal y como las aplicaciones crecen de tamaño, los fallos en la TLB tienen una importancia mayor. Aunque el uso de páginas largas puede reducir este impacto, con aplicaciones muy grandes disminuye su efectividad.Al objeto de mitigar este sobrecoste y su impacto en el rendimiento, algunos procesadores incluyen una cache en su MMU para almacenar los niveles intermedios de la traducción de direcciones, a la que se denomina *Page TAble Walker Cache* (PTWC), la cual se analizará en el capítulo 3.

En lo que sigue, se describen las estructuras de tablas de páginas, y los pasos seguidos por el PTW en su consulta, correspondientes a dos arquitectura de procesador diferentes, como son la x86 y la de ARM.

2.3.1. Traducción de direcciones en la arquitectura x86

Como se ha comentado anteriormente, todos los procesadores con arquitectura x86 desde el *80386* utilizan una estructura de árbol para almacenar la asignación de direcciones virtuales a físicas cuya raíz, que constituye la tabla de páginas, está apuntada (dirección física) por el registro CR3 de la MMU. A pesar de que la profundidad de este árbol se ha incrementado, el procedimiento de traducción de las direcciones virtuales a direcciones físicas usando este árbol se mantiene básicamente sin cambios.

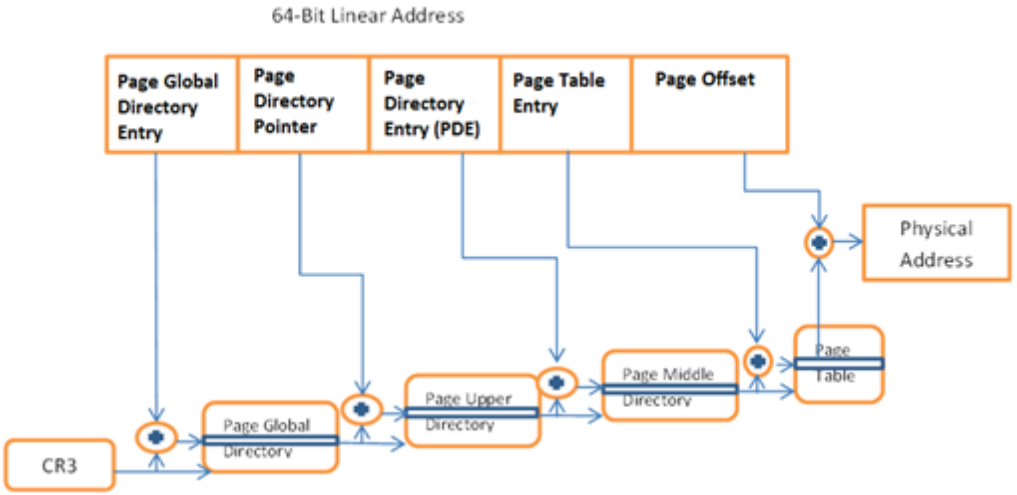


Figura 2.3: Niveles y pasos de traducción de direcciones virtuales a físicas seguidos por el PTW en arquitectura *x86*

Una dirección virtual se divide en un número de página y un *offset* de página. El número de página es dividido además en una secuencia de indices. El primer índice, combinado con el puntero CR3, se usa para seleccionar una entrada desde la raíz del árbol, el cual puede contener de nuevo un puntero al nodo del siguiente nivel mas abajo del árbol. Si la entrada contiene un puntero, el siguiente índice se utilizará para seleccionar una entrada de este nodo, el cual puede contener otra vez un puntero al nodo del siguiente nivel mas abajo del árbol. Estos pasos se repiten hasta que la entrada seleccionada es inválida (básicamente, un puntero a *NULL* indicando que no es una

traducción válida para ese espacio de memoria) o bien la entrada apunta a una página de datos usando la dirección física. En esta caso, el *offset* de la página se añade a la dirección física de esta página de datos para obtener la dirección física completa. En una MMU de diseño simple, este procedimiento requiere un acceso de memoria por cada nivel en el árbol.

En la figura 2.3, se muestra la descomposición de las direcciones virtuales en los procesadores x86-64. Las páginas estándar de esta arquitectura don de 4kB, por lo que hay un *offset* de página de 12 bits. El resto de los 48 bits de las direcciones virtuales se dividen en cuatro índices de 9 bits, los cuales son utilizados para seleccionar entradas en los cuatro niveles de la tabla de páginas (*page table*). Los cuatro niveles de la tabla de páginas se llaman **PML4** (*Page Map Level 4*, o Mapa de Páginas Nivel 4), **PDP** (*Page Directory Pointer*, o Punteros de Directorios de Páginas), **PD** (*Page Directory, o Directorio de Páginas*) y **PT** (*Page Table*, o Tabla de Páginas). A partir de aquí y por simplificar, nos referiremos a estos niveles como:

- **L4 -** PML4.
- **L3 -** PDP.
- **L2 -** PD.
- **L1 -** PT.

Al final, la dirección virtual de 48 bits se extiende a 64 bits para futuras ampliaciones. Como el espacio de direcciones crece se pueden añadir nuevos campos de índices (como podría ser L5), reduciendo el tamaño del campo inicial.

Una entrada de la tabla de páginas tiene 8 bits de tamaño, independientemente de su nivel en el árbol, y el tamaño total de un nodo es siempre 4kB al igual que el tamaño de una página. Por ello los nodos son llamados habitualmente *page table pages* (página de tabla de páginas). El árbol puede ser escasamente poblado de nodos, ya que si en cualquier nivel no hay una dirección virtual válida de un índice para 9 bits particular, el árbol por debajo de este índice no se instanciará. Esto mejora significativamente el ahorro de memoria, con grades porciones de los 256TB del espacio de direcciones virtuales que nunca se asignarán en la mayoría de aplicaciones.

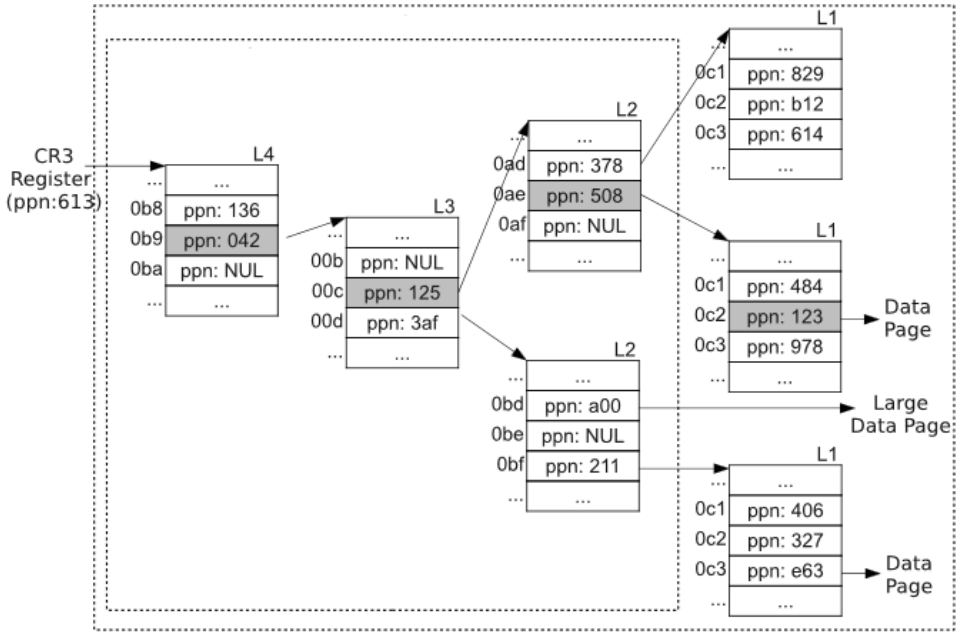


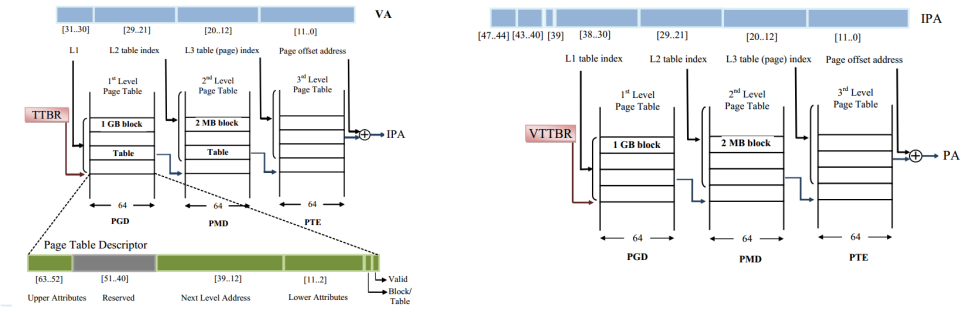
Figura 2.4: Un ejemplo de *page table walk* para la dirección virtual (0xb9, 0x0c, 0xae, 0xc2, 0x16).

El proceso de traducción, tal y como se muestra en la figura 2.4, funciona como se explica a continuación. En primer lugar el *hardware* del *page table walker* debe localizar el nivel mas alto de la página de tabla de páginas, que almacena las entradas de L4. La dirección física de esta página es almacenada en el registro del procesador CR3. A fin de poder traducir la dirección, se extrae el campo índice L4 de 9 bits de la dirección virtual y se añade al número de la dirección física de 40 bits del registro CR3. La dirección de 49 bits resultante se usa para apuntar a la entrada de 8 bits correspondiente del nivel L4 de la tabla (el *offset* 0xb9 en la página de tabla de páginas L4). La entrada L4 puede contener un número de página física o un puntero a la página de tabla de páginas L3 (en este caso 0x42). El proceso continua extrayendo el campo índice L3 de la dirección virtual y añadiéndolo al puntero a la página de tabla de páginas L3 obtenido en la etapa anterior para direccionar la entrada L3 correspondiente. Este proceso se repite hasta que la entrada seleccionada es invalida o se especifica el número de página física en memoria, como se muestra en la figura. Cada entrada de tabla de páginas a lo largo de esta ruta está señalado en gris en la figura 2.4. El *offset* de página desde la correspondiente página virtual es añadida a este número de página física para conseguir la dirección física.

Con esta estructura, lo actuales espacios de direccionamiento de 48 bits *x86-64* requieren cuatro accesos de memoria para caminar (walk) la tabla de páginas de arriba abajo en busca de la traducción de las direcciones virtuales. Como el espacio de direcciones continua creciendo, se añadirán mas niveles a la tabla. Un espacio de direcciones virtual completo de 64 bits requiere seis niveles, consiguiendo seis accesos a memoria por traducción.

2.3.2. Traducción de direcciones en la arquitectura ARM

Los procesadores con arquitectura ARM cuentan con dos tablas, una que denominaremos virtual (figura 2.5(a)) y que gestiona el sistema operativo, y otra que llamaremos física (figura 2.5(b)). Las dos tablas tienen tres niveles con entradas de 64 bits. Con la primera tabla se consigue una dirección intermedia entre la física y la virtual, llamada (*Intermediate Physical Address* o IPA. En la segunda tabla y mediante la IPA, se consigue la dirección física (PA)



(a) Tabla de páginas virtual (De VA a IPA) (b) Tabla de páginas física (De IPA a PA)

Figura 2.5: Estructura de las tablas de páginas con arquitectura ARM.

La traducción se realiza en dos fases (figura 2.6), requiriendo un número total de 15 accesos a memoria para conseguir la dirección física. En la primera fase, para obtener la IPA se realizan 12 accesos, debido a que la tabla virtual con la que conseguimos la IPA tiene tres niveles, y en cada uno de esos niveles se debe traducir su contenido en la tabla física antes de apuntar al siguiente nivel de la tabla virtual, necesitando para ello 4 accesos a memoria. Finalmente, en la segunda fase se realizan los otros tres accesos en la tabla física para obtener la PA.

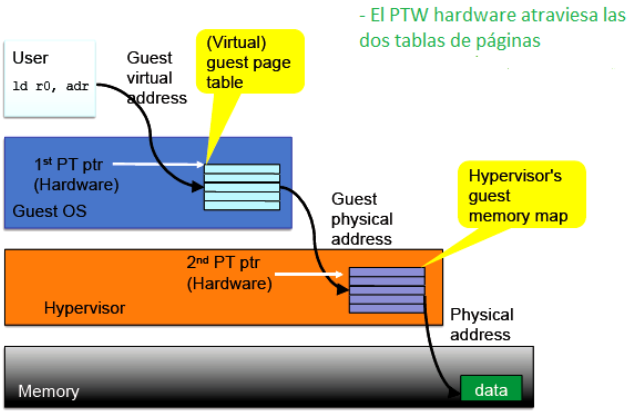


Figura 2.6: Las dos etapas de traducción

Capítulo 3

Aceleración de la búsqueda en tabla de páginas mediante cachés

3.1. Estructuras de caché para tablas de páginas

Las tablas de páginas en árbol requieren varios accesos para traducir una única dirección, sin embargo los accesos a los niveles superiores muestran una localidad temporal significativa. Cuando se accede a páginas consecutivas en el espacio de direcciones virtuales, normalmente se utilizan las mismas entradas de nivel superior, ya que los índices con los que se indexan dichas entradas provienen de los bits de orden superior de la dirección virtual, los cuales terminan siendo compartidos por grandes grupos de páginas consecutivas. Por este motivo, los fabricantes de procesadores, entre ellos Intel y AMD, han incluido unas cachés privadas de muy baja latencia como parte de la MMU con el propósito de almacenar pasos intermedios de traducción de la dirección correspondientes a los niveles superiores de la tabla de páginas. A las referidas cachés se las denomina *Page Table Walk Cache* (PTWC).

Las PTWCs pueden almacenar elementos de la tabla de páginas etiquetadas por su dirección física en la memoria, tal y como hace una cache de datos convencional. Llamaremos a estas PTWCs cachés de tabla de páginas (*page table caches*). Pero las PTWCs podrían ser indexadas también a través de partes de las direcciones virtuales, y en este caso las llamaremos cache de traducción (*translation caches*).

En cualquiera de estos esquemas de etiquetado, los elementos de diferentes niveles de la tabla de páginas se pueden mezclar en una única caché (cache unificada), o situarse en distintas caches (cache dividida). También podemos almacenar en cada entrada de la PTWC todos los niveles intermedios de traducción o ruta completa (caché de ruta).

Cachés de tabla de páginas

En la PTWC de tabla de páginas (*Page Table Cache* o PTC) los elementos son etiquetados con su dirección física en la tabla de páginas. Las entradas correspondientes al nivel L1 no son almacenadas en cache en ninguna de las configuraciones de cache que se describen a continuación.

- **Caché de tabla de páginas unificada (UPTC)** El diseño mas simple dedicado a una PTC es una caché única de solo lectura y alta velocidad para las entradas de tablas de páginas, etiquetada por direcciones físicas en memoria. Las entradas de los distintos niveles de la tabla de páginas se mezclan en una misma caché. Cada una de las entradas de la caché se halla etiquetada por una dirección física (puntero a alguna de las páginas de tabla de páginas), más un índice procedente de la dirección virtual, siendo su contenido una dirección física (puntero a la página de tabla de páginas del siguiente nivel). Como en la TLB, se debe mantener la coherencia de la misma con respecto a la tabla de páginas. Este diseño es el que utiliza las PTWC de AMD.

Base Location	Index	Next Page
125	0ae	508
042	00c	125
613	0b9	042
...	...	...

Figura 3.1: Ejemplo del contenido de una UPTC .

La figura 3.1 muestra un ejemplo de UPTC después de que la MMU recorre la tabla de páginas en usca de la traducción de la dirección virtual <0xb9, 0x0c, 0xae, 0xc2, 0x16>(véase ejemplo mostrado en figura 2.4). Con este tipo de cache se consigue que tres de los cuatro accesos a la tabla de páginas no sean a memoria, sino a una caché privada.

- **Cache de tabla de páginas dividida. (SPTC)** Un diseño alternativo para la PTC separa las entradas de la tabla de páginas de cada uno de los niveles de traducción en cachés diferentes. En la figura 3.2 se puede observar la estructura de la SPTC. En este diseño, cada entrada individual contiene la misma etiqueta y datos que si tratara de una UPTC. La diferencia principal respecto a la UPTC reside en que cada nivel de tabla de página está en una cache privada, de modo que las entradas de niveles diferentes no compiten por un mismo *slot*.

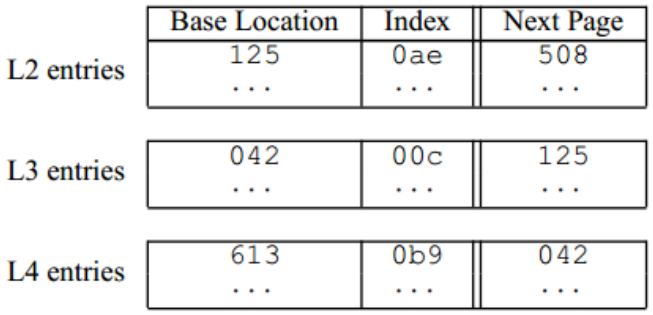


Figura 3.2: Ejemplo del contenido de una SPTC. Cada entrada mantiene la misma etiqueta y los mismos datos que en la UPTC.

Caches de traducción

Como alternativa a etiquetar las entradas de la caché por su dirección física, también puede etiquetarse mediante los campos índice presentes en la dirección virtual. Una entrada correspondiente al nivel de traducción L4 podrá ser etiquetada por el índice L4 de 9 bits de la dirección, y la entrada L3 con los indices de L3 y L4, y la L2 con los índices mencionados anteriormente más los de L2, como se puede apreciar en las figuras 3.3 y 3.4. Llamaremos a esta configuración de caché **caché de traducción**, ya que almacena una traducción parcial de una dirección virtual. Con este esquema de etiquetado, se evita ir encadenando consultas sobre la PTWC en busca de los niveles de traducción siguientes, dado que la propia PTWC proporciona la traducción completa (a excepción del nivel inferior) o parcial hasta cierto nivel. Todas las búsquedas pueden realizarse independientemente de las demás. Al final, la MMU seleccionará la entrada que coincida con el prefijo mas largo de la dirección virtual porque permite al *page walk* saltarse la mayoría de los niveles.

- **Caché de traducción unificada (UTC).** Al igual que las cachés de tabla de páginas pueden configurarse tanto divididas como unificadas, también existe la configuración de una Caché de Traducción Unificada (*Unified Translation Cache* o UTC). Por otra parte, al igual que la UPTC, los UTC mezcla elementos de todos los niveles de la tabla de páginas en una misma cache, como se aprecia en la figura 3.3.

L4 index	L3 index	L2 index	Next Page
0b9	00c	0ae	508
0b9	00c	xx	125
0b9	xx	xx	042
...	...	...	...

Figura 3.3: Ejemplo del contenido de una UTC. "xx"significa que da igual el contenido

- **Caché de traducción dividida (STC).** Como una SPTC, la Caché de Traducción Dividida (*Split Translation Cache* o STC) almacena entradas de diferentes niveles de la tabla de páginas en cachés separadas. La diferencia reside en el hecho de que la STC utiliza una manera diferente de etiquetar las entradas, tal y como muestra la figura 3.2.

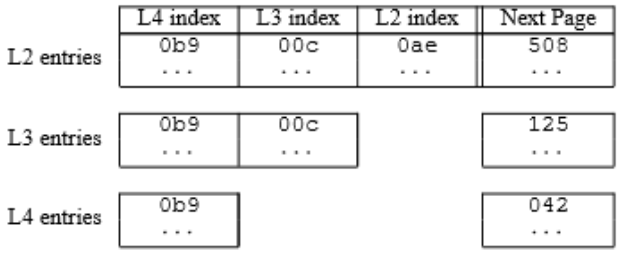


Figura 3.4: Ejemplo del contenido de una STC.

El dato contenido en cada entrada de la caché corresponde a un número de página física de 40 bits correspondiente al siguiente nivel de la tabla de páginas. El acierto en una entrada de la caché L2 requiere que haya coincidencia con los tres indices que etiquetan la entrada. En el caso de una entrada en la caché de nivel L3 la coincidencia deberá ser con solo dos de los índices, mientras que en la de nivel L4 bastará la coincidencia con un único índice. Esta búsqueda puede realizarse en cualquier orden, incluso en paralelo.

Caché de ruta de traducción (*Tranlation-Path Cache*)

Observando la figura 3.3 que ejemplifica una UTC, las etiquetas de las tres entradas representan una sola ruta hacia abajo de la tabla de páginas en la que todas tienen el mismo contenido. Las entradas de L3 y L4 usan menos de la dirección virtual que las entradas de la caché L2, pero los fragmentos de los que hace uso son los mismos. Consecuentemente es posible almacenar todos los tres números de página físicos de ese ejemplo en una sola entrada. En las cachés de ruta de traducción (*Tranlation-Path Cache* o TPC), una única entrada representa una ruta entera, incluyendo todas las entradas intermedias, para un *page table walk* dado, en vez de una única entrada intermedia a lo largo de ese *page table walk*, como se puede apreciar en la figura 3.4.

L4 index	L3 index	L2 index	L3	L2	L1
0b9	00c	0ae	042	125	508
...	...	...	...	...	...

Figura 3.5: Ejemplo del contenido de una TPC.

La TPC mantiene los tres índices de nueve bits como si se tratara de una caché de traducción, pero almacenando los tres números de página física de 40 bits correspondientes a cada uno de los tres niveles de traducción de la tabla de páginas.

3.2. Diseño de la caché de tabla de páginas

La UPTC es el diseño que utilizan los procesadores AMD, mientras que la STC es la que utiliza Intel. Todos los diseños que hemos comentado permiten la aceleración del *page table walk* de páginas almacenando en cache información relativa a los niveles superiores de la tabl de páginas. Sin embargo, estos diseños tienen diferencias en cuanto a la forma de indexar, particionado, cobertura y complejidad.

3.2.1. Forma de indexar

El esquema de indexado determina cómo se realiza la búsqueda en la PTWC. Los indices de la cache pueden ser derivados de las direcciones físicas de las páginas que constituyen los diferentes niveles de la tabla de páginas o pueden ser derivados directamente de la dirección virtual, correspondiendo con los niveles de la tabla de páginas. Las cachés de traducción usan componentes de las direcciones virtuales como indices. Por ejemplo, la TLB es una

caché de traducción L1 que usa el número de página virtual como índice.

3.2.2. Particionado

Las PTWCs pueden ser unificadas o particionadas, o también pueden almacenar la información de la ruta completa. El particionado de la caché indica como las entradas de la caché se almacenaran en los distintos niveles de la tabla de páginas. Esto determina realmente como las entradas de distintos niveles son aisladas las unas de las otras.

El impacto del esquema de particionado depende ampliamente de cómo utilicen las aplicaciones el espacio de direcciones virtuales. Las aplicaciones que hacen uso de manera densa de su memoria principal, suelen hacer uso de unas pocas entradas de L3 y L4, reutilizando mucho, sin embargo, las entradas de nivel L2. Por el contrario, las aplicaciones que utilizan el espacio de memoria virtual de forma más dispersa harán poca reutilización de las entradas de nivel L2, pero utilizarán con mas probabilidad un numero repetido de veces las entradas correspondientes a L3 y L4. La estrategia de particionado y la política de reemplazo determinan cómo estas entradas rellenarán el espacio de almacenamiento ofrecido por la PTWC, lo que puede tener un impacto significativo para la PTWC.

3.2.3. Cobertura

La caracterización de la cobertura de la PTWC, ya sea de tipo traducción (TC) o de tabla de páginas (PTC), no es algo sencillo, ya que primero debe considerarse cuál es su significado. Por ejemplo, supongamos un acierto en la traducción de direcciones correspondiente al nivel L3 de la tabla de páginas, pero un fallo en la traducción del nivel L2. En este caso, la traducción se ha acelerado por la PTWC, pero sin embargo se ha requerido un acceso a memoria para ir a buscar la entrada de tabla de página L2. Por lo tanto es discutible si la PTWC ha proporcionado cobertura o no. Se tomará la posición estricta de que la cobertura significa que la PTWC proporciona pleno de aciertos en la traducción de la dirección correspondiente a los niveles L4, L3 y L2.

En general, con el mismo número de entradas, las PTWC de traducción pueden cubrir un espectro del espacio de direcciones mayor que una PTWC

de tabla de páginas. El motivo es que una caché de traducción puede hacer un uso más eficiente de sus entradas que una caché de tabla de páginas. Para que una caché de tabla de páginas pueda ofrecer cobertura debe mantener simultáneamente una entrada para cada uno de los niveles L2, L3 y L4, mientras que una caché de traducción puede mantener cobertura con una única entrada L2. Esto quiere decir que las cachés de traducción deben almacenar entradas L2 adicionales en lugar de las entradas de L3 y L4, que se requieren en la cachés de tabla de páginas.

Cuando la aplicación es simplemente demasiado grande para la PTWC como para dar una cobertura completa, las caches unificadas con políticas de reemplazo inteligentes, las cachés divididas y las TPC pueden acelerar las traducciones para más amplias zona del espacio de direcciones que las cachés unificadas con políticas de reemplazo convencionales. Esto es debido al hecho de que las entradas de nivel alto de las cachés precedentes, que ofrecen traducción parcial para grandes regiones, poseen menor probabilidad de ser desalojadas en favor de las entradas de nivel mas bajo.

3.2.4. Complejidad

Todas las organizaciones mas efectivas son PTWC totalmente asociativas. Sin embargo, las diferentes formas de organización tienen maneras distintas de etiquetar y distintos tamaños de las entradas, por lo que por norma general se requerirá un número distinto de entradas para lograr un ratio de acierto similar. Estos factores lideraran la complejidad de las implementaciones para las diferentes organizaciones.

3.3. Caché de tabla de páginas compartida

En un entorno con varios núcleos de procesamiento en el que corren aplicaciones que se ejecutan en paralelo, puede ocurrir que varios hilos accedan al mismo espacio de direcciones, y que por lo tanto compartan buena parte de las etapas intermedias de traducción cuando se realiza un "*page walking*". Para aprovechar este hecho se propuso un esquema en el que la PTWC fuera compartida entre los diferentes núcleos, en lugar de ser privativas de cada uno de ellos, a la cual denominaremos SPTWC (Shared PTWC).

La configuración de la SPTWC que se ha propuesto se corresponde con una

UPTC. La SPTWC está situada de manera centralizada respecto a los núcleos, a los que se halla conectada mediante una topología tipo *crossbar*, como se aprecia en la figura 3.5(b), y está asociada al controlador de memoria. El diseño del *Page Table Walker* le permite a la MMU no ser consciente de si hay o no una PTWC por debajo en la jerarquía de memoria, funcionando como cualquier otra caché del procesador.

La SPTWC puede tener diferentes grados de asociatividad, mientras que la política de reemplazo utilizada corresponde a un algoritmo LRU básico. El formato de las entradas de la SPTWC es exactamente el mismo que el de una "privada", incluyendo una dirección base de 40 bits y un índice de 9 bits como etiqueta para recuperar la entrada deseada y el puntero de 40 bits al siguiente nivel de la tabla de páginas.

La SPTWC se accede siempre que hay un fallo en alguna de la TLBs asociadas a los núcleos a los que sirve. Por cada fallo de TLB, la SPTWC se accederá tantas veces como niveles tiene la tabla de páginas (cuatro veces en el caso de la arquitectura x86-64) menos uno, puesto que la entrada de último nivel que contiene la conversión final de la dirección de página lógica a física no se almacena, como ya se comentó con ocasión del diseño de la PTWC tradicional. Además se asume que existe un esquema de SPTWC separado para datos e instrucciones, de acuerdo a la existencia de TLBs para datos e instrucciones independientes.

Capítulo 4

Evaluación de prestaciones

4.1. Herramientas utilizadas

4.1.1. El simulador gem5

El simulador gem5 [8] es la unión de los simuladores M5 [9] y GEMS [10]. M5 provee un *framework* de simulación configurable, múltiples *ISA* y diversos modelos de CPU. GEMS complementa estas funcionalidades con un sistema de memoria flexible y detallado, incluyendo soporte para múltiples protocolos de coherencia de cache y modelos de intercomunicación. Actualmente gem5 soporta los *ISA* mas comerciales (ARM, ALPHA, MIPS y x86), incluyendo el arranque en Linux con tres de ellos (ARM, ALPHA y x86).

Opciones de simulación

- **CPU Model.** El simulador gem5 nos ofrece cuatro modelos diferentes de CPU. *AtomicSimple* es un modelo mínimo de una CPU IPC. *TimingSimple* es igual salvo porque simula las temporizaciones de la memoria. *InOrder* es un modelo por etapas ordenado, y *O3* es un modelo por etapas desordenado.

- **Modos del sistema** Cada simulación puede ser lanzada en dos modos. El modo *System-Call Emulation*, o modo de simulación de llamadas al sistema (SE) evita la necesidad de modelar dispositivos para un sistema operativo emulando muchos servicios del sistema. Por otro lado, tenemos el modo *Full-System* o sistema completo (FS), que ejecuta instrucciones tanto del nivel de usuario como del *kernel*, y modela un sistema completo incluyendo el sistema operativo y los dispositivos. A diferencia del modo SE, el modo FS mejora tanto la precisión como

la variedad de espacios de trabajo que gem5 puede ejecutar. Los espacios de trabajo que requieren muchos servicios del sistema operativo o dispositivos de entrada/salida solo funcionan en FS.

- **Sistema de memoria**. El simulador gem5 incluye modelos de memoria distintos, *Clasic* (clásico) y *Ruby*. El modelo clásico, proveniente de M5, nos ofrece una configuración del sistema de memoria rápida y fácil, mientras que el modelo Ruby, que viene del GEMS, ofrece una infraestructura flexible capaz de simular una amplia variedad de sistemas de coherencia de cache en memoria.

- ***ISA.*** Las arquitecturas simuladas por gem5 son **Alpha, ARM, MIPS, Power, SPARC y x86**.

Las simulaciones almacenan el resultado de ciertos eventos, que pueden especificarse y modificarse, en un fichero para su posterior análisis. También se puede lanzar aplicaciones a partir de un instante dado. De esta manera podemos comparar el resultado de varias simulaciones con mayor precisión.

4.1.2. La *suite* de aplicaciones de prueba PARSEC2.1

PARSEC es un conjunto de *benchmarks* precompilados para varias plataformas y ya preparados para ser usados en gem5. En la siguiente tabla aparecen las aplicaciones usadas en este trabajo, de las 13 que forman esta suite:

Aplicación	Descripción
Blackscholes	Cálculos financieros utilizando la ecuación diferencial parcial *blackscholes*.
Facesim	Simulación del movimiento de un rostro humano para animación.
Fluidanimate	Simulación física de fluidos para animación.
Freqmine	Minería de datos.
Streamcluster	Resuelve el problema de online clustering.
Swaptions	Calcula los precios de una cartera de valores usando el modelo Heath–Jarrow–Morton.
x264	Codificación de vídeo en H.264.

Tabla 4.1: Aplicaciones de PARSEC

PARSEC nos presenta seis tipos de entradas para sus aplicaciones, según queramos mayor carga o mayor rapidez. Son los siguientes:

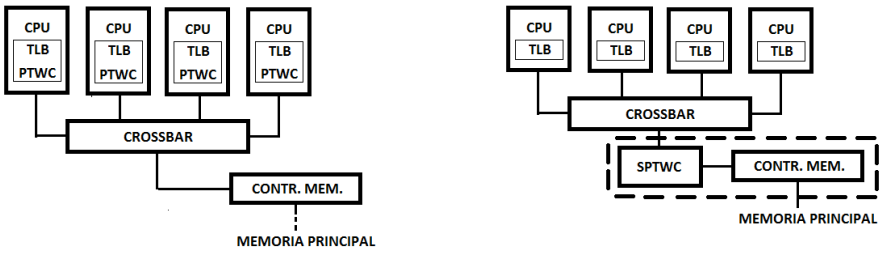
- ***test.*** Una entrada muy pequeña para probar la funcionalidad básica del programa.
- ***simdev.*** Entrada muy pequeña que garantiza un comportamiento del programa similar al real, destinada a la prueba y desarrollo del simulador.
- ***simsmall, simmedium y simlarge.*** Entradas de diferentes tamaños (pequeña, mediana y grande) adecuadas para el estudio de microarquitecturas con simuladores.
- ***native.*** Entrada muy grande destinada a la ejecución nativa. Se considera que se trata de una aproximación a la ejecución con una entrada realista.

4.1.3. CACTI

CACTI, desarrollado por miembros de la empresa *Hewlett Packard*, es una herramienta que permite obtener el tiempo de acceso, tanto en memorias principales como en memorias caché, ademas del tiempo de ciclo, el área que se ocupa, corrientes de fuga y energía dinámica, entre otras medidas, en un solo modelo. La herramienta es de uso generalizado por la comunidad de arquitectos de computadores para medir las prestaciones de las organizaciones de memoria, y la relación existente entre el tiempo, la potencia y el área de las mismas. En este trabajo se ha obtenido mediante CACTI tanto el área como el consumo por acceso de la caché con las distintas configuraciones analizadas.

4.2. Modelo simulado

El sistema sobre el que se ha trabajado es en x86 multinúcleo. La jerarquía de memoria del procesador se compone de cachés privadas L1 separadas para instrucciones y datos y de una caché privada L2 compartida. La PTWC compartida está situada en el mismo nivel que la caché L2 del procesador en la jerarquía de memoria.



(a) PTWC privada para cada núcleo

(b) PTWC compartida para todos los núcleos (SPTWC)

Figura 4.1: Distintas configuraciones de PTWC.

Para los experimentos se ha simulado en modo FS un sistema de un procesador con varios núcleos O3 con las siguientes características:

CPU	**CMP de 1, 2, 4 y 8 núcleos, por etapas y desordenado, con arquitectura x86-64 a 1GHz, con caché L1 de datos de 64KB y de instrucciones de 32kB, las dos con asociatividad por conjuntos de 2 vias, y caché L2 de datos e instrucciones de 2MB con asociatividad por conjuntos de 8 vias**
L1TLB	**Único nivel L1 de TLB separada para datos e instruccones, privada para cada núcleo y totalmente asociativa. Tamaño de 8 entradas** [1].
PTWC	**Dos configuraciones. Una privada (para un núcleo que no necesita acceso al crossbar) y otra compartida (accedida por todos los núcleos de modo centralizado, haciendo uso del crossbar). Cachés separadas para datos e instrucciones, desde 4 hasta 32 entradas, asociatividad variable desde 2 vías hasta totalmente asociativa. Algoritmo de reemplazo LRU (*Last Recently Used,* o menos usada recientemente).**

Tabla 4.2: Parámetros de los componentes

Las aplicaciones mostradas en la sección 4.1.2 han sido usadas con los parámetros *simmedium*, excepto facesim,swaptions y x264, en los que se ha

[1]Tamaño original de 64 entradas. Se ha redimensionado para el tamaño de las aplicaciones

utilizado parámetros *simlarge*. Esta elección presenta un buen compromiso entre tiempo de simulación y resultados obtenidos para este trabajo. Las aplicaciones se han lanzado a partir del mismo instante de tiempo mediante puntos de control, y las medidas han sido realizadas desde el punto en que se inicia la aplicación hasta que esta se acaba. El tamaño tanto de la TLB como de la PTWC se ha adaptado al tamaño de las aplicaciones.

Para el cálculo de la aceleración en el tiempo de acceso a memoria se han tenido los siguientes parámetros, teniendo un sistema con una memoria principal DDR2 a 1333MHz, con la CPU a 1 GHZ:

Crossbar (T_{Cross})	**2 ns**
PTWC (T_{PTWC})	**2 ns**
Controlador de memoria ($T_{CntrMem}$)	**4 ns**
Memoria (T_{Mem})	**35 ns**

Tabla 4.3: Parámetros temporales

4.3. Descripción de los estudios

Los estudios realizados se orientan únicamente hacia la PTWC de datos, en línea con otros estudios publicados [12]. Los cuatro primeros estudios corresponden a medidas recogidas directamente de los resultados ofrecidos por el simulador gem5. En primer lugar analizaremos el llenado de la PTWC en términos de porcentaje de ocupación de las entradas. Dicho análisis muestra además el llenado correspondiente para cada uno de los niveles del *page table walker*. Posteriormente se analiza la tasa de aciertos tanto global como por niveles, expresada en términos de número de aciertos en la PTWC por fallo de TLB. Como se indicó en la sección 3.3, la PTWC alberga sólo tres de los cuatro niveles de traducción presentes en la arquitectura x86, por lo que a lo máximo que se puede aspirar es a lograr tres aciertos por fallo de TLB. A continuación, se analizará la reutilización de las entradas por nivel, expresado en porcentaje de reutilizaciones de cada uno de los niveles sobre el total de reusos (si ha habido 100 reusos en total, y 10 pertenecen a entradas de nivel L2, la reutilización de las entradas de nivel L2 corresponden al 10 % del total).

El cuarto estudio corresponde a la aceleración obtenida en el tiempo de acceso a memoria. Para ello se calcula el cociente entre los tiempos de resolución de los fallos de TLB sin PTWC y con PTWC, respectivamente, de la siguiente manera (considerando los cuatro niveles de traducción):

Como de los cuatro niveles de traducción sólo los tres primeros pueden estar alojados en la PTWC, el tiempo de resolución de un fallo de TLB se descompondrá en dos partes. La primera parte, correspondiente a los tres primeros accesos de traducción, dependerá de un tiempo medio de acceso a memoria resultante de las tasa de aciertos en la PTWC; la segunda, correspondiente al último acceso de traducción, dependerá exclusivamente del tiempo de acceso a memoria. Para ello, primero se realiza el calculo del tiempo que tarda un **acierto en un nivel de la PTWC** (T_H):

$$T_H = T_{Cross} + T_{PTWC}$$

A continuación, se calcula lo que cuesta un fallo en la PTWC, que en realidad equivale al **tiempo de acceso a memoria** (T_F), ya que cada vez que haya un fallo en la PTWC, se accederá a ella:

$$T_F = T_{Cross} + T_{PTWC} + T_{CntrMem} + T_{Mem}$$

Despues calculamos el **tiempo medio de acceso a memoria** (T_{avg}) correspondiente a los tres primero niveles de traducción, el cual dependerá de la **tasa de aciertos** (H) de la PTWC:

$$T_{avg} = H * T_H + (1 - H) * T_F$$

Con todo ello, ya estamos en condiciones de calcular el **tiempo de resolución de un fallo de TLB con PTWC** ($T_{missTLBwithPTWC}$), teniendo en cuenta que los tres primeros accesos de traducción contabilizarán al T_{avg}, mientras que el último lo hará al T_F:

$$T_{missTLBwithPTWC} = 3 * (T_{avg}) + T_F$$

Por su parte, el tiempo de resolución de un **fallo de TLB sin PTWC** ($T_{missTLBwhitoutPTWC}$) viene dado por:

$$T_{missTLBwhitoutPTWC} = 4 * (T_{Cross} + T_{CntrMem} + T_{Mem})$$

Finalmente, la **aceleración del tiempo de acceso a memoria** a:

$$T_{missTLBwhitoutPTWC}/T_{missTLBwithPTWC}$$

Como se aprecia, la aceleración depende fundamentalmente de la tasa de aciertos de la PTWC. En la medida en la que dicha tasa de aciertos se aproxime al 100 % (3 aciertos por fallo de TLB, tal y como se ha definido anteriormente), el tiempo de resolución de los fallos de TLB se aproximará

al tiempo de acceso a la PTWC (para los tres primeros niveles de traducción) más un tiempo de acceso a memoria principal (para el último nivel de traducción, el cual no se almacena en la PTWC). La disminución del tiempo de fallo de TLB reduce el denominador de la relación entre los tiempos de resolución de fallo de TLB son PTWC y con PTWC, propiciando así el incremento de la aceleración.

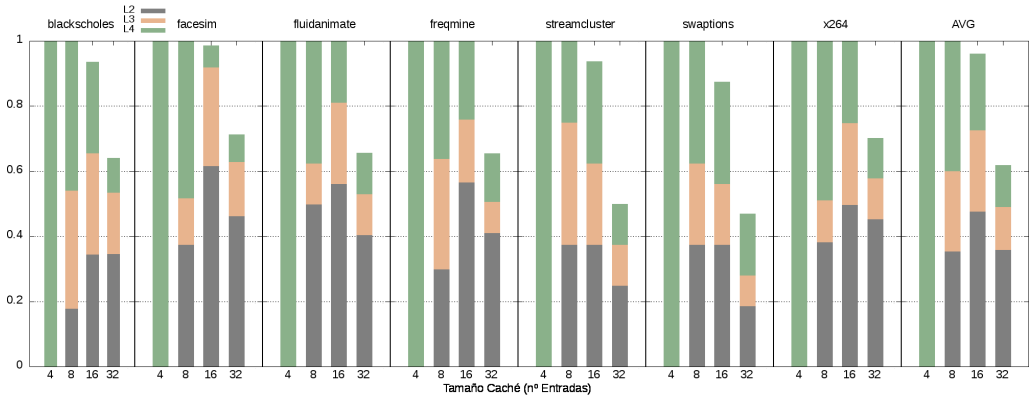
Los últimos estudios realizados corresponden al consumo y área ocupada por la PTWC, para lo que se ha recurrido a la aplicación CACTI 5.3 en su versión de escritorio. Los resultados obtenidos son con tecnología de 32 nm. El consumo está expresado en nanoJulios y el área en milímetros cuadrados

4.4. Análisis de resultados

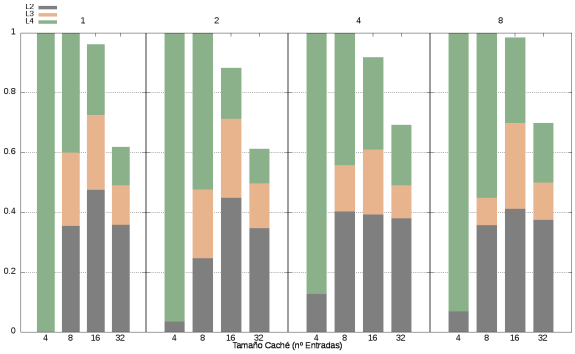
En todos los estudios realizados se analizará la influencia tanto del tamaño de la PTWC (en término de número de entradas) como del grado de asociatividad o número de vías de la misma, considerando para ello un único núcleo. Los resultados de muestran para todas las aplicaciones analizadas, incluyendo el resultado promedio para todas ellas. Cada uno de los estudios se completa con el análisis de la influencia del número de núcleos en el caso de la SPTWC (PTWC compartida). Los análisis en los cuales se mide el comportamiento de la caché en general, se muestra unicamente la influencia del número de núcleos.

4.4.1. Análisis del llenado de la PTWC

En esta sección se analiza en que porcentaje se llena la PTWC, así como con que tipo de entradas lo hace, mostrando el porcentaje de cada tipo respecto al total de entradas totales que hay en caché. En la figura 4.2(a) se observa que manteniendo fija la asociatividad (4 vías) y variando el tamaño de la PTWC, esta se sólo se llena al 100 % con 4 y 8 entradas. Con 32 entradas el llenado apenas supera el 60 %. En cuanto a la distribución de niveles, se observa que conforme aumenta el tamaño también lo hace la ocupación de los niveles bajos. Esto es debido a que las entradas que más se reutilizan son las de nivel superior, y con muy pocas entradas el algoritmo de reemplazo no puede almacenar entradas de los niveles L3 y L2. Esta situación cambia en cuanto se dispone de más espacio.



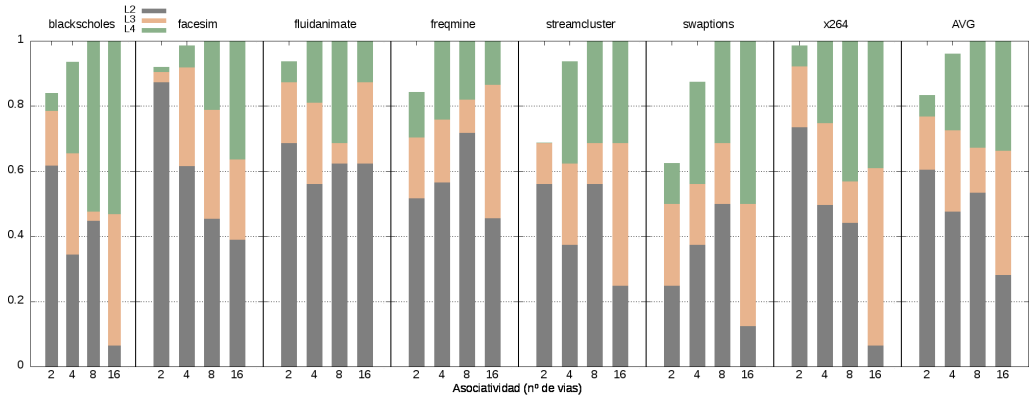
(a) Resultados para PTWC de un núcleo



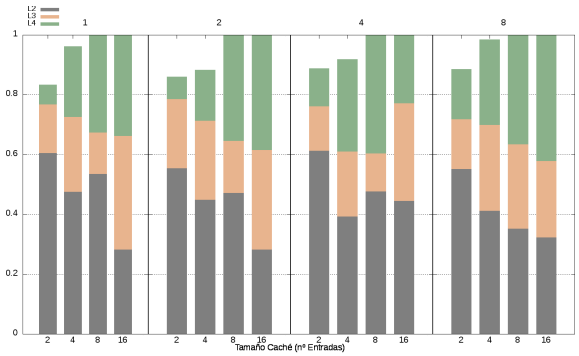
(b) Resultados medios de la SPTWC para todas las aplicaciones para 1, 2, 4 y 8 núcleos

Figura 4.2: Número de entradas por nivel de las aplicaciones obtenidos con asociatividad por conjuntos de 4 vías.

Cuando variamos la asociatividad , asumiendo un tamaño fijo de PTWC de 16 entradas, se aprecia que a mayor número de vías, mayor el porcentaje de la caché que se llena. En particular, observamos que con 4 vías se logra un llenado superior al 95 % y que con 8 vías ya se alcanza el 100 %. Ello muestra que no es necesario que la PTWC sea completamente asociativa para llenarla al 100 %. Respecto al grado de presencia de los distintos niveles L1, se observa que una mayor asociatividad favorece el porcentaje de presencia de entradas correspondientes a niveles superiores. Se puede explicar este hecho con que al haber pocas vías las entradas de nivel L2, que son más numerosas que las de los otros niveles, ocupan toda la caché sin que esta pueda ser aprovechada del todo.



(a) Resultados para PTWC de un núcleo



(b) Resultados medios de la SPTWC para todas las aplicaciones para 1, 2, 4 y 8 núcleos

Figura 4.3: Número de entradas por nivel de las aplicaciones obtenidos con tamaño de la PTWC de 16 entradas.

Respecto al comportamiento con varios núcleos, al variar el número de entradas no se observa ninguna tendencia que permita sacar conclusiones. Donde si se aprecia una tendencia es variando la asociatividad. A mas núcleos, más se llena la caché y mayor porcentaje ocupan las entradas de nivel L4, ya que cuanto mayor sea el número de núcleos, mas estresada estará la caché.

4.4.2. Análisis de la tasa de aciertos de la PTWC

En esta sección se analiza la tasa de aciertos global de la PTWC en términos del número de aciertos por fallo de TLB. Como ya se indicó anteriormente, el máximo al que podemos aspirar es a 3 aciertos por fallo de TLB. Se muestran únicamente los resultados promedio de todas las aplicaciones y para distinto número de núcleos, desde 1 hasta 8. En primer lugar, se analiza la influencia del tamaño de la PTWC sobre la tasa de aciertos promedio

para una caché de 4 vías. Como se puede observar en la figura 4.4(a),la tasa de aciertos aumenta, aproximándose al máximo de 3, a medida que aumenta el tamaño de la PTWC, independientemente del número de núcleos. No obstante, se aprecia que las configuraciones con varios núcleos se ven penalizadas para SPTWCs de bajo número de entradas, debido a la competencia entre los núcleos. Sin embargo, conforme se aumenta el número de entradas el porcentaje de aciertos se iguala, ya que disminuye la competencia entre los distintos núcleos al tener mayor espacio.

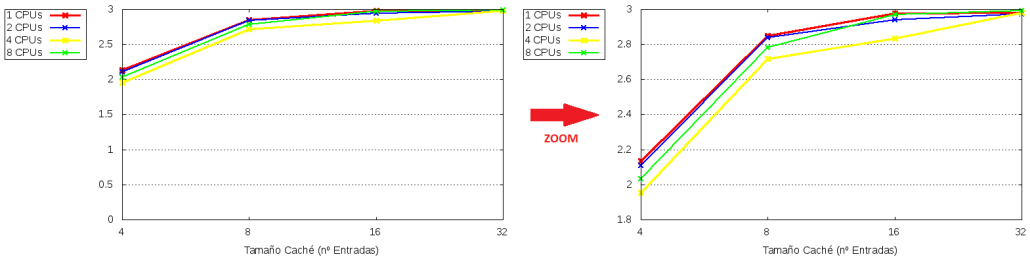


Figura 4.4: Media de aciertos de la PTWC obtenidos con caché de 4 vías por conjunto.

Cuando se analiza la influencia de la asociatividad para un tamaño fijo de PTWC, se aprecia la tasa de aciertos se ve favorecida a medida que aumenta la asociatividad, si bien con 8 vías ya se alcanza prácticamente la máxima tasa de aciertos, como se observa en la figura 4.5. En este caso, un bajo número de vías favorece las configuraciones de varios núcleos. Esta diferencia se mitiga a medida que aumenta el número de vías.

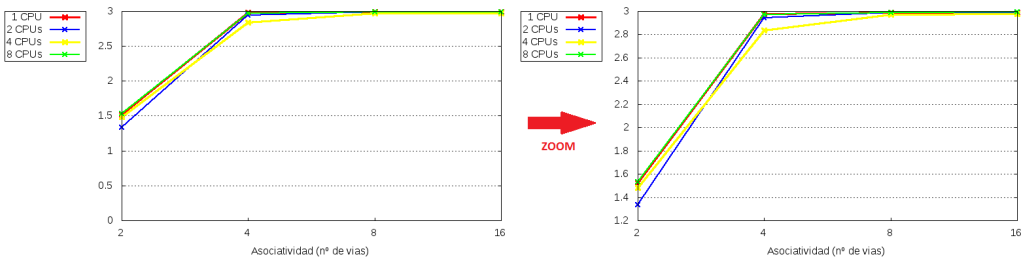
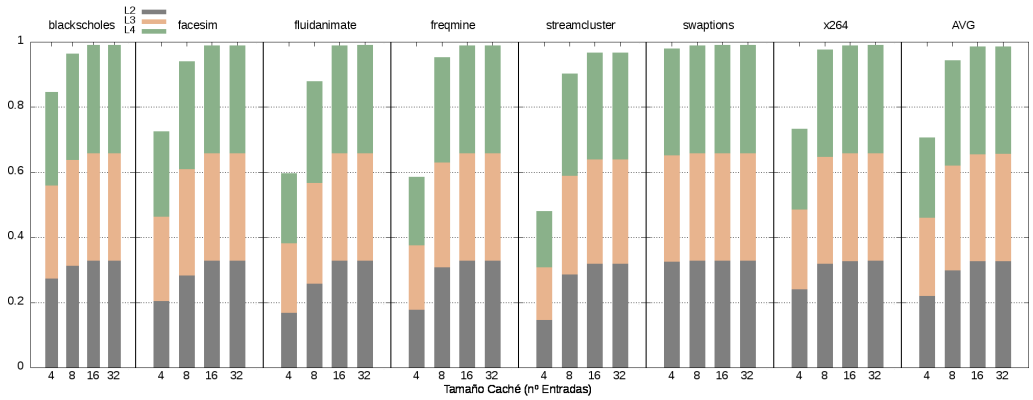


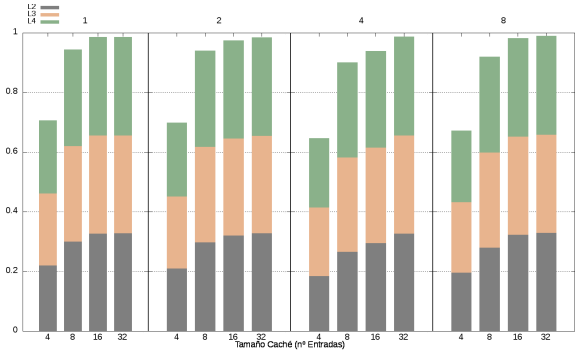
Figura 4.5: Media de aciertos de la PTWC obtenidos con tamaño de PTWC de 16 entradas.

Después de observar la tasa de aciertos de manera global, vamos a proceder a hacer el análisis por nivel. Los resultados de dicho estudio se muestra para todas las aplicaciones analizadas. La distribución del porcentaje de

aciertos por nivel está normalizado a 3 aciertos por cada *page table walk*. Las figuras 4.5(a) y 4.5(b) muestran la influencia del tamaño de la caché, mientras que las figuras 4.6(a) y 4.7(b) hacen lo propio con la influencia del grado de asociatividad.



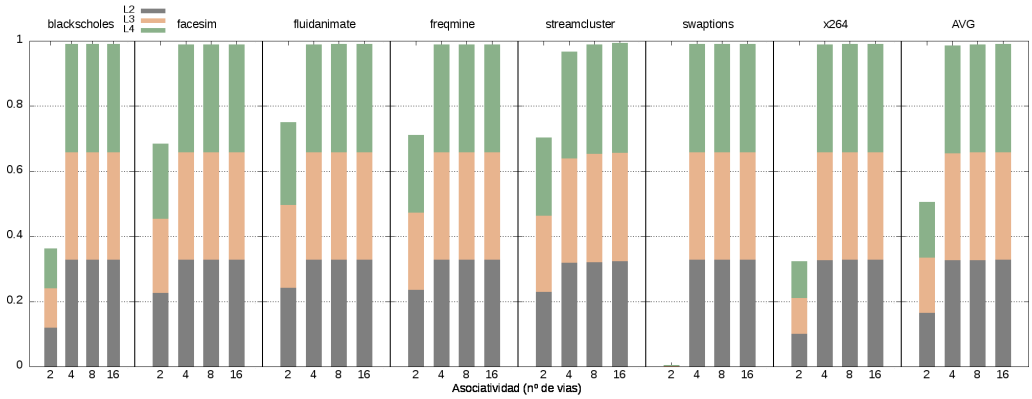
(a) Resultados para PTWC de un núcleo



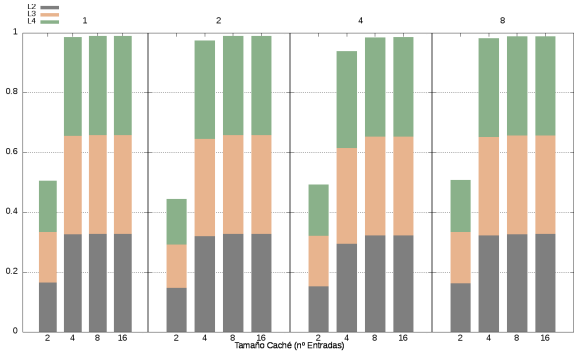
(b) Resultados medios de la SPTWC para todas las aplicaciones para 1, 2, 4 y 8 núcleos

Figura 4.6: Distribución del porcentaje de aciertos por nivel de las aplicaciones obtenidos con asociatividad por conjuntos de 4 vías.

En todos los casos se aprecia cómo el porcentaje de aciertos se distribuye de forma prácticamente equilibrada entre los tres niveles, lo que explica que anteriormente, en los casos más favorables, se llegaran a alcanzar prácticamente los 3 aciertos por fallo de TLB.



(a) Resultados para PTWC de un núcleo



(b) Resultados medios de la SPTWC para todas las aplicaciones para 1, 2, 4 y 8 núcleos

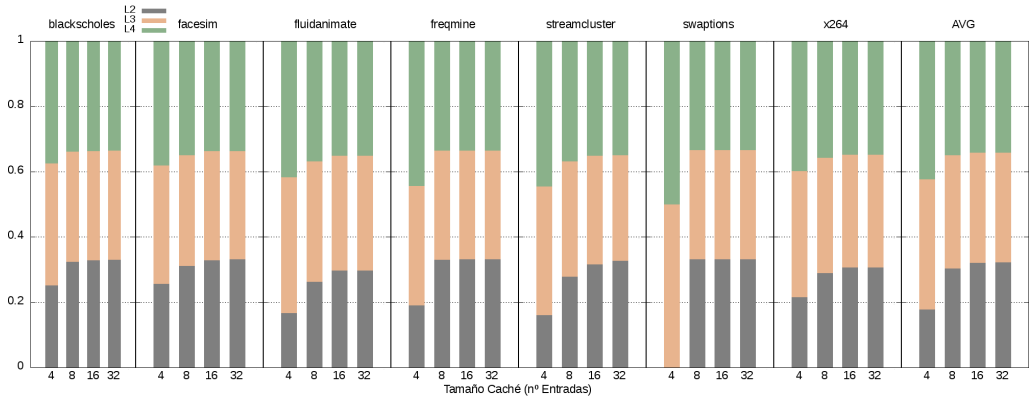
Figura 4.7: Distribución del porcentaje de aciertos por nivel de las aplicaciones obtenidos con tamaño de la PTWC de 16 entradas.

Asimismo, se observa cómo aún en los casos en que la tasa de aciertos global queda por debajo (tamaños de caché pequeños o baja asociatividad), la distribución de los aciertos continua estando equilibrada entre los distintos niveles. Todo ello demuestra que pese a que haya pocas entradas de nivel superior, estas son mas reutilizadas que las de nivel inferior, que suelen poblar mayoritariamente la caché.

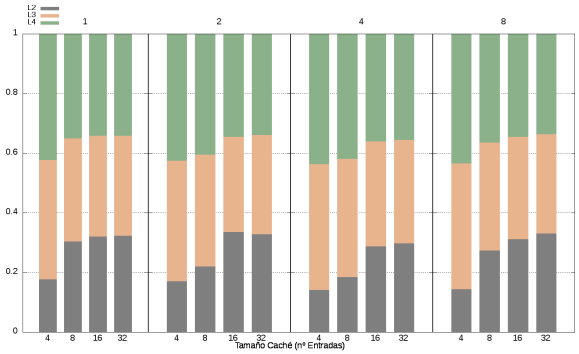
4.4.3. Análisis de la reutilización de las entradas de la PTWC

En esta sección se analiza el grado de reutilización de las entradas correspondientes a cada nivel. La figura 4.8. muestra como varía el grado de reutilización de los distintos niveles a medida que aumenta el tamaño de la

PTWC. Como se puede observar, conforme se dispone de mas espacio, las entradas del niveles mas bajo son mas veces reutilizadas.



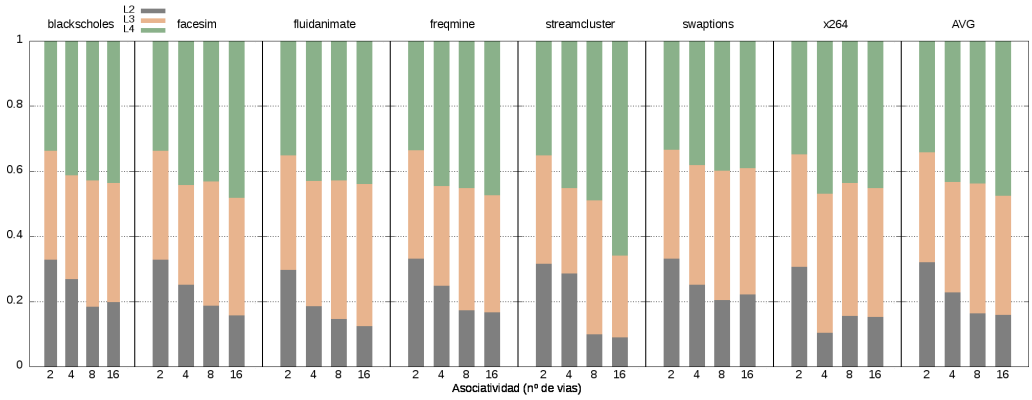
(a) Resultados para PTWC de un núcleo



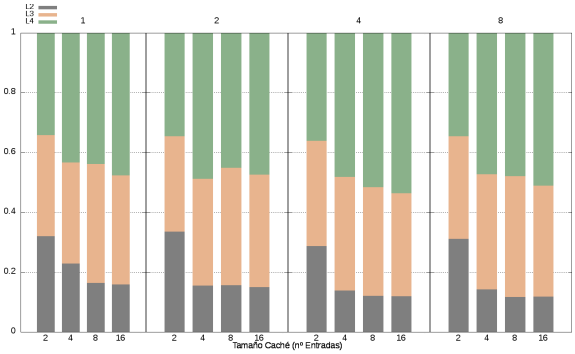
(b) Resultados medios de la SPTWC para todas las aplicaciones para 1, 2, 4 y 8 núcleos

Figura 4.8: Distribución de la reutilización de las entradas por nivel obtenidos con asociatividad por conjuntos de 2 vías.

Por su parte, la figura 4.9. muestra la influencia de la asociatividad en el grado de reutilización de los distintos niveles. Como se puede apreciar, con un bajo número de vías, la mayor población de entradas de nivel L2 produce que estas sean muy reutilizadas, en detrimento de las de nivel superior L3 y L4, hecho que se corrige aumentando el número de vias.



(a) Resultados para PTWC de un núcleo



(b) Resultados medios de la SPTWC para todas las aplicaciones para 1, 2, 4 y 8 núcleos

Figura 4.9: Distribución de la reutilización de las entradas por nivel obtenidos con tamaño de la PTWC de 16 entradas.

Al aumentar el número de núcleos, al aumentar el tamaño de la PTWC, la reutilización de las entradas de nivel L2 disminuye. Variando el número de vias, sucede lo mismo que al aumentar el tamaño. Esto se puede explicar debido a que

4.4.4. Análisis de la aceleración en el tiempo de acceso a memoria

En esta sección analizamos la aceleración o incremento de velocidad en el acceso a memoria, tal y como se definió en la sección 4.3. Como se puede apreciar en la figura 4.10, en la que se muestra el resultado promedio de todas las aplicaciones analizadas, la aceleración aumenta conforme aumenta el tamaño de la PTWC, aproximándose a un valor de 3 como consecuencia del

incremento de la tasa de aciertos en la PTWC. El comportamiento relativo de las curvas para diferente número de núcleos es similar al observado con ocasión del análisis de la tasa de aciertos.

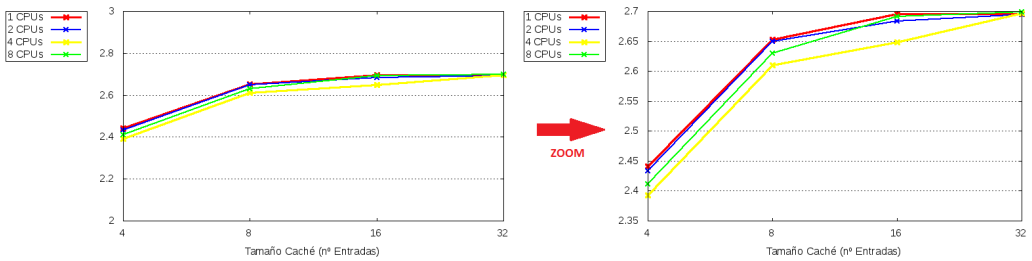


Figura 4.10: Mejora de rendimiento media obtenida con asociatividad por conjuntos de 4 vias.

La figura 4.11. muestra la la evolución de la aceleración respecto a la variación del número de vías. Dicha evolución es similar al observada anteriormente para la tasa de aciertos. Como se observa, a medida que se incrementa la asociatividad, se maximiza la aceleración y se aprecia una convergencia en el comportamiento de las configuraciones para diferentes núcleos.

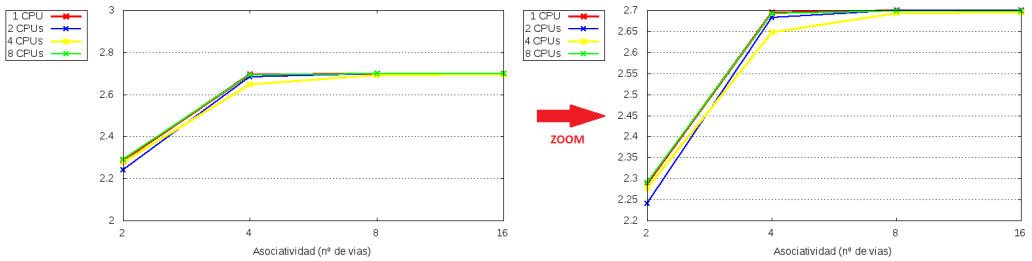


Figura 4.11: Mejora de rendimiento media obtenida con tamaño de PTWC de 16 entradas.

El valor obtenido para la aceleración parece bastante razonable teniendo en cuenta que sólo se incrementa la velocidad de tres de los 4 accesos a memoria requeridos para resolver los fallos de TLB. El valor de aceleración igual a 3 se vería incrementado en la medida en que aumentase el tiempo de acceso a memoria en relación a la PTWC. Asimismo, dicho valor de la aceleración experimentaría un significativo incremento, superando el valor de 10, si sólo se tomasen como referente los tres primeros ciclos de traducción para definir la aceleración.

4.4.5. Análisis del consumo y del área

Para finalizar analizaremos el rendimiento energético y el área ocupada con todas las configuraciones de caché que han sido objeto de prueba en este trabajo. En el caso del consumo, observando la figura 4.12 podemos decir que con pocas entradas el número de estas afecta en mayor grado que el incremento de la asociatividad. Por el contrario, con un número de entradas alto el consumo se ve mas penalizado por el número de vías, llegándose a multiplicar por cinco con 32 entradas al pasar de 2 a 32 vías. A tenor de lo visto en los análisis del aumento de velocidad, no es recomendable ni el uso de cachés totalmente asociativas ni de tamaños mayores de 16 entradas.

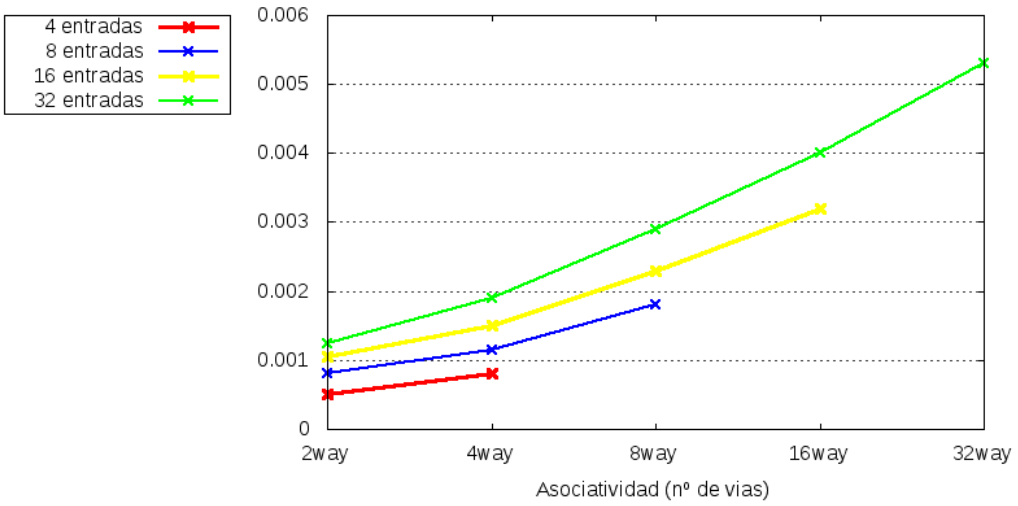


Figura 4.12: Consumo de energía por acceso calculado mediante CACTI para las distintas configuraciones de SPTWC analizadas (en nJ).

Respecto al área (figura 4.13) sucede algo similar al consumo. Sin embargo, el incremento del área cuando se aumenta el número de entradas es similar al aumento del número de vías. A pesar de este comportamiento mas uniforme, las conclusiones pueden ser las mismas para los dos análisis.

Aunque desde el punto de vista del comportamiento la mejor opción siempre es una cache con alta asociatividad, como conclusiones de este análisis se puede decir desde el punto de vista tanto energético como de área, y analizando los resultados obtenidos anteriormente, el mejor compromiso con el tamaño de aplicaciones que se tiene es de 4 vías y 16 entradas.

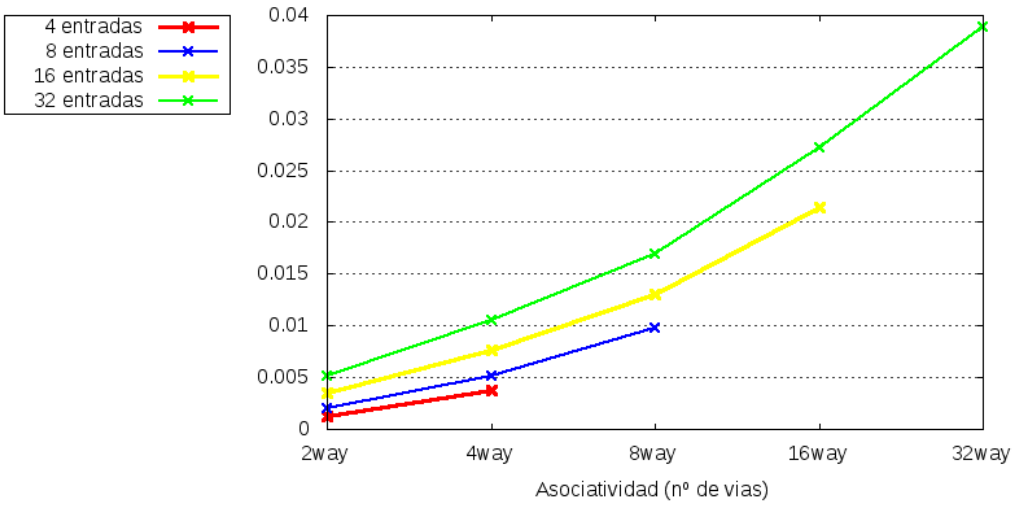


Figura 4.13: Area calculada mediante CACTI para las distintas configuraciones de SPTWC analizadas (en mm2).

Capítulo 5

Conclusiones

En trabajos anteriores [12], se demostró la mejora en el rendimiento que producía el hacer uso de una PTWC compartida (SPTWC)entre varios núcleos cuando se ejecutan aplicaciones paralelas, gracias a poder explotar el patrón de compartición exhibido por dichas aplicaciones. En este trabajo se ha completado dicho análisis, evaluando el comportamiento de la SPTWC ante la variación de diferentes parámetros estructurales de configuración del modelo, como puedan ser el tamaño y asociatividad de la caché.

Cuanto mayor sea el tamaño de la SPTWC y mayor su grado de asociatividad, mayor es el rendimiento del sistema en términos de reducción del tiempo de resolución de los fallos de TLB, como consecuencia de la reducción en la penalización en el acceso a memoria que va ligado al aumento en el porcentaje de aciertos. En particular, teniendo en cuenta las aplicaciones analizadas y los tamaños de problema considerados, se ha mostrado cómo una caché de 16 entradas y 8 vías es capaz de alcanzar prácticamente la mejora máxima de rendimiento que teóricamente cabría esperar con las configuraciones de PTWC/SPTWC objeto de análisis. Por contra, se obtiene una penalización significativa en el tamaño y el consumo a partir de 16 entradas y 4 vías en comparación con las mejoras de rendimiento resultante para el tamaño de las aplicaciones usado.

En términos relativos, la evolución del comportamiento de la SPTWC respecto a la variación de los parámetros estructural analizados es muy similar al observado en la PTWC. Si bien la PTWC (cache privada) exhibe mejor comportamiento que la SPTWC para pequeños tamaños de cache y baja asociatividad, se observa una convergencia en dicho comportamiento a medida que aumenta el número de entradas de la caché y crece la asociatividad. Este resultado es muy importante, en el sentido de que pone de manifiesto cómo

es posible reducir el espacio total de almacenamiento de la caché al pasar de una estructura de caché privada (PTWC) a otra compartida (SPTWC) sin penalizar prestaciones. En particular, para una configuración de 8 núcleos el estudio muestra cómo una SPTWC de 16 entradas y 8 vías puede reemplazar a 8 PTWC del mismo tamaño y asociatividad, y todo ello sin penalización alguna en las prestaciones.

Una gran parte del tiempo consumido en la realización del presente trabajo se ha invertido en portar el código modificado para modelar la SPTWC a una versión más actual (Septiembre 2013) del programa Gem5. Esto ha sido útil como aprendizaje del manejo de *software* complejo para modelado de sistemas multiprocesador.

Sin embargo, ese tiempo invertido en el conocimiento del manejo de las herramientas, y su posterior modificación para el propósito de este trabajo, ha provocado que no haya sido posible realizar varios de los estudios que estaban previstos en principio. Ejemplos de ampliaciones que se pueden realizar son aumentar la cantidad de aplicaciones de prueba, o utilizar otras *suite* de este tipo de aplicaciones, aumentar el tamaño del problema que define cada una de las aplicaciones, analizar cargas mixtas y/o sintéticas, analizar el comportamiento ante cambios de contextos o cambios de máquina virtual, analizar el comportamiento de la caché de instrucciones o probar con otras arquitecturas, como ARM.

Pese a lo comentado en el párrafo anterior, se consideran cumplidos los objetivos, y se deja para trabajos posteriores aquellas ampliaciones del trabajo original que no se pudieron realizar.

Bibliografía

[1] Página *web* del proyecto vIrtical. SW/HW extensions for heterogenous multicore platforms. <www.virtical.eu>(Accedida en Agosto de 2014).

[2] Página *web* del programa Simics. <http://www.windriver.com/simics/>(Accedida en Agosto de 2014).

[3] ARM. *ARM Cortex A15 MPCore Processor. Revision: r4p0. Technical Reference Manual*. 2011-2013.

[4] Red Hat Enterprise Linux 4, *Introducción a la administración de sistemas, Capítulo 4: Memória física y virtual*, Red Hat, Inc. , 2005.

[5] Departamento de Lenguajes y Computación de la Universidad de Almeria, *Diseño de Sistemas Operativos, Tema 3: Gestión de memoria*, Almeria.

[6] A. Silberschatz, P. B. Galvin y G. Gagne, *Fundamentos de sistemas operativos, septima edición. Mc GrawHill*, 2005.

[7] A. Bhattacharjee, D. Lustig y M. Martonosi, *Shared Last-Level TLBs for Chip Multiprocessors*, 2011.

[8] N. Binkert, B. Beckman, G. Black, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoabib, N. Vaish, M. D. Hill y D. A. Wood. *The gem5 Simulator*. <http://gem5.org>

[9] N. Binkert, R. G. Dreslinki, L. R. Hsu, K. T. Lim, A. G. Saidi y S. K. Reinhardt. University of Michigan. *The M5 simulator: Modeling networked systems*, 2006.

[10] M. M. K. Martin, D. J. Sorin, B. M. Beckman, M. D. Hill, M. Xu, M. R. Marty, A. R. Alameldeen, K. E. Moore y D.

A. Wood. University of Pennsylvania, Duke Univ., Univ. of Wisconsin-Madison. *Multifacet's General Execution-driven Multiprocesor Simulator (GEMS) Toolset*, 2005.

[11] T. W. Barr, A. L. Cox y S. Rixner. Rice University Houston, TX. *Translation Caching: Skip, Don't Walk (the page table)*

[12] A. Esteve, M. E. Gómez y A. Robles.Universitat Politècnica de València. *Exploiting Parallelization on Address Translation: Shared Page Walk Cache*, 2013.

[13] M. Ortín, P. Gratz y S. W. Kecker. Escuela de Ingenieria y Arquitectura de la Universidad de Zaragoza. Directores M. Villaroya, D. Suarez.*Caracterización del comportamiento de la suite PARSEC en la jerarquía de memoria del procesador*, 2011.

[14] Página *web* de la aplicación CACTI. <http://www.hpl.hp.com/research/cacti/>(Accedida en Agosto de 2014).

Índice de figuras