UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Diseño de Controladores de Memoria Eficientes para Futuros Sistemas

Tesina del Máster Universitario en Ingeniería del Software,
Métodos Formales y Sistemas de Información
Curso 2013/2014

Departamento de Sistemas Informáticos y Computación

Paula Navarro Alfonso

Directores:
Julio Sahuquillo Borrás
Javier Oliver Villarroya

Septiembre de 2014

Agradecimientos

Le agradezco a mi director de proyecto, Julio Sahuquillo, su ayuda en estos años y le doy las gracias por que aquella tarde en prácticas me hablase de este proyecto. A mi otro director de proyecto, Javier Oliver, su ayuda y guía para hacer posible la presentación de este trabajo. A los profesores María Engracia y Crispín su paciencia para con mis dudas y mis desvaríos. A mi compañero de trabajo, Vicent Selfa, el apoyo en los malos momentos cuando el trabajo resultaba agobiante y los resultados nefastos.
A mi familia, por todo el apoyo recibido y soportarme durante todos estos años.
A mis compañeros y amigos, por los buenos momentos que hemos compartido.
A todos vosotros, gracias.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Descripción del problema

Actualmente la práctica totalidad de los dispositivos electrónicos basados en microprocesador, de cualquier tamaño ya sea grande o pequeño, están presentes en las distintas facetas de nuestra vida diaria, por ejemplo cuando trabajamos (teléfonos de última generación, tablets, ordenadores de sobremesa, portátiles, etc...), en nuestro tiempo de ocio (consolas, televisiones inteligentes, etc...), cuando viajamos (ordenador de abordo del coche o del avión). De hecho, muchas veces no los percibimos pero dependemos en gran medida de ellos. La inversión en investigación y la evolución de estos ordenadores en los últimos años ha sido imparable lo cual ha hecho posible esta omnipresencia. Todos estos dispositivos incorporan microprocesadores encargados de realizar las tareas de cómputo.

Hace una década, la industria de los microprocesadores evolucionó hacia los procesadores multinúcleo para atacar los importantes problemas de disipación de temperatura y consumo energético de los procesadores del momento, así como para superar los límites inherentes del paralelismo a nivel de instrucción. Inicialmente, se comenzó con la integración únicamente de dos núcleos en un solo chip, pero pronto aparecieron procesadores comerciales con más núcleos (como el AMD Magny-Cours de 12 núcleos y el Intel Xeon Phi de 64 núcleos), y actualmente a medida que la escala de integración disminuye, chips con cientos de núcleos están en el horizonte. Sin embargo, la esperanza de incrementar las prestaciones en los futuros procesadores con tal cantidad de núcleos solo se alcanzará si se consigue superar el reto que supone su escalabilidad en prestaciones y energía.

Los avances en la arquitectura de los núcleos tienen el potencial de continuar in-

crementando las prestaciones de cómputo a un ritmo similar al seguido hasta ahora [JRLCV10]. Sin embargo, otras partes del procesador no avanzan al mismo ritmo. En concreto, la tecnología DRAM con la que se construye la memoria principal en la que se almacenan los datos que utilizan los programas para el cómputo, así como la conexión de los núcleos con dicha memoria, se han mejorado a una velocidad mucho menor, por lo cual el acceso a memoria se ha convertido en el cuello de botella principal de los procesadores actuales y se prevé que está situación se agrave de forma notable en los futuros procesadores con un mayor número de núcleos .

Las altas latencias en el acceso a memoria ya supusieron el mayor cuello de botella en las prestaciones para muchas aplicaciones ejecutadas en los procesadores monolíticos (un único núcleo). En efecto, las peticiones enviadas por el procesador esperan en una o en varias colas del controlador de memoria a que llegue su turno para acceder a memoria. Como tanto el tiempo de espera en las colas como el tiempo de acceso son elevados, la productividad del procesador se ve seriamente afectada, ya que el cómputo se ve detenido a la espera de que lleguen los datos a procesar desde memoria. El problema se agrava todavía más en los procesadores multinúcleo, donde el sistema de memoria es un recurso compartido por los múltiples hilos de los núcleos, los cuales compiten por el acceso a memoria, y más concretamente a los recursos dentro de ella, aumentando los conflictos de obtención de recursos (bus, banco, etc) que se traduce en un gran impacto en el tiempo de acceso. Una aproximación para reducir el problema ha sido incorporar grandes cachés, de una capacidad de decenas de MB, en los procesadores actuales. Sin embargo, esta solución no resuelve el problema completamente y solo es factible para un número de núcleos bajo, ya que está demostrado que este método no funciona para un número de núcleos medio o elevado, es decir, no es escalable.

Las técnicas actuales para ocultar la latencia de memoria se han diseñado centrándose en procesadores monolíticos donde el tráfico entre el procesador y memoria es generado por un solo núcleo. Sin embargo, los inconvenientes citados se magnifican en los procesadores multinúcleo. Estos procesadores suelen ejecutar cargas multiprogramadas o compuestas por varios programas, y el tráfico que llega al controlador es la suma de las peticiones generadas por cada uno de los núcleos individuales. Las peticiones de los distintos programas compiten entre ellas por el acceso a memoria. Esta competencia provoca con frecuencia que el acceso a memoria se convierta en un cuello de botella todavía más acuciante en las prestaciones, ya que se producen altas latencias de espera en las colas del controlador y en la red de interconexión en las cuales las

peticiones aguardan a que les llegue su turno para poder acceder a memoria principal, hecho que perjudica de manera notoria las prestaciones globales del sistema.

Las memorias DRAM se organizan en matrices de bloques distribuidos en filas y columnas conocidos como bancos. Típicamente, un acceso a memoria lee un único bloque, sin embargo, a la DRAM se accede leyendo toda una fila de bloques y luego seleccionando la columna que contiene el bloque buscado. Para reducir las altas latencias de acceso, los bancos de las DRAM actuales sitúan la fila leída en unos amplificadores, llamados búfer fila (RB). Este modo de funcionamiento, conocido como modo *página abierta*, aprovecha la localidad espacial entre las peticiones de memoria, es decir, los subsecuentes accesos a la misma fila pueden obtener los datos del búfer y no necesitan leer la fila del banco otra vez, reduciendo la latencia de acceso. Esta operación se conoce como acierto en el búfer fila (RBH), tarda 3× veces menos tiempo que si la fila no está en el búfer, conocido como fallo en el búfer fila (RBM) [LMNP11]. Sin embargo, el búfer fila no es tan bueno cuando las aplicaciones presentan una localidad baja. En tal caso, almacenar la página entera (fila) en el búfer fila apenas reducirá el tiempo de acceso de los siguientes accesos ya que éstos probablemente no encontrarán su bloque en el búfer fila. Además, si la memoria principal apenas es accedida, mantener el búfer fila habilitado puede producir un aumento de la energía consumida. Para solucionar este problema, algunas propuestas [GMMG12] reducen el tamaño del búfer fila, ahorrando energía. El problema se agrava en los CMPs donde diferentes núcleos/aplicaciones compiten por el mismo búfer fila, por lo tanto, la localidad espacial de las aplicaciones individuales cae con respecto a ejecutarse solamente ella. Algunos investigadores se han dado cuenta recientemente de esta situación, y proponen implementar un búfer fila específico para cada aplicación ejecutándose [HGCT13].

1.2. Objetivos

En la actualidad existen varias técnicas cuyo objetivo es reducir los tiempos de acceso a la memoria. Desde un punto de vista ideal, el acceso a ésta se realizaría de manera instantánea, pero el retraso es dependiente del nivel de caché en el que se encuentre el dato, siendo la memoria principal la más costosa desde un punto de vista temporal y energético. El objetivo de este trabajo es diseñar una organización de memoria y del controlador para reducir este tiempo de acceso con un consumo energético eficiente, aprovechando el paralelismo de acceso a la memoria principal de las diferentes aplicaciones.

El tiempo de acceso al sistema de memoria está directamente relacionado con el hecho de si el acceso encuentra su bloque en el RB o no. Por lo tanto, es importante tratar de maximizar estos aciertos para reducir el tiempo de acceso y por tanto mejorar el rendimiento. En este trabajo se ha propuesto aumentar el número de buffers fila para maximizar los accesos que encuentran su bloque en las filas de los buffers. Para aumentar el número de buffers fila se hace necesario remodelar el sistema de memoria, e introducir la lógica para gestionar estos nuevos recursos.

Asimismo, aunque es mejor tener la mayor cantidad de buffers posibles, no es eficiente usar una gran cantidad de estos a efectos de escalabilidad y costes. Para ello se ha realizado un estudio para encontrar un número óptimo de recursos, donde se han caracterizado el comportamiento de las diferentes aplicaciones usadas cuando se aumenta el número de buffers fila.

Varias aproximaciones se han propuesto para atacar el problema de las altas latencias de acceso explotando la localidad espacial. Una de estas aproximaciones consiste en una organización que sitúe una tabla de filas en los bancos de memoria; mientras que la otra sitúa esta tabla en el lado del controlador de memoria.

Asimismo, la inclusión de más buffers fila tiene un impacto en el consumo energético que se debe cuantificar. Sobre todo en el caso de tener la tabla en el lado del controlador, ya que implica transferir la fila desde el banco al controlador y puede implicar que se alcance un alto consumo de transferencia. Por lo tanto, se han investigado propuestas para solucionar este problema, como es el caso de reducir el tamaño de la fila transferida a una fracción menor de ésta.

Para resumir, el propósito de este proyecto es diseñar y evaluar por medio de simulación una nueva estructura de memoria principal, compuesta por un controlador de memoria, canales, rangos, bancos de memoria y diversos buffers fila, así como las propuestas de mejora basándose es esta nueva estructura. El diseño propuesto se basa en el uso de una tabla de buffers fila situada en dos puntos diferentes: controlador y banco. Con las nuevas estructuras propuestas y gracias al controlador de memoria es posible aplicar prioridades a las peticiones de memoria además de aprovechar mejor su localidad, acelerando su acceso.

1.3. Motivación

Este apartado explora los beneficios potenciales de aumentar el número de RBs. Para ello, se caracteriza el *porcentaje medio de aciertos en los buffers fila* (RBHR) de aplicaciones individuales, ejecutándose solas, variando el número de búfer filas dispo-

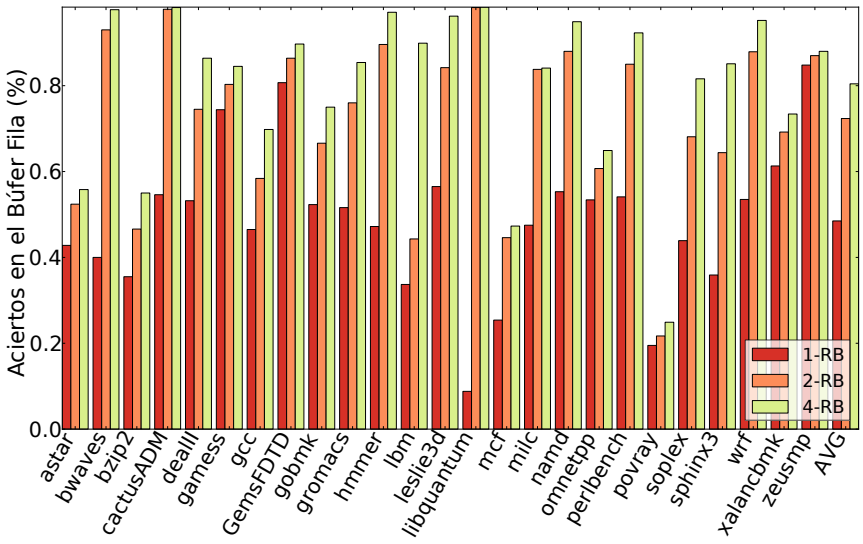


Figura 1.1: Aciertos en los buffers fila variando el número de entradas.

nibles en cada banco de memoria. La Figura 1.1 muestra el resultado de aumentar el número de entradas de 1 a 4 buffers fila. Como se observa, el comportamiento de los aciertos en los RB varia ampliamente entre las aplicaciones estudiadas debido a que cada una tiene un patrón de acceso diferente. De estos resultados se extraen las siguientes observaciones:

- **Observación 1**. Algunas aplicaciones consiguen un RBHR notable (esto es, con 1 RB) mientras algunas otras exhiben una localidad baja en el RB, lo cual corrobora trabajos anteriores [HGCT13].

- **Observación 2**. A pesar del RBHR de la aplicación, éste puede mejorar de forma notoria (por ejemplo sobre 50 %) en la mayoría de aplicaciones con 4 buffers fila.

Teniendo en cuenta estas observaciones, las aplicaciones se han clasificado en cuatro categorías principales presentadas a continuación:

- Categoría 1. Esta categoría incluye aplicaciones que tienen un RBHR alto (más de un 50 % de RBHR con 1 entrada o RB), pero que pueden aumentar significativamente añadiendo más buffers fila. Algunos ejemplos son las aplicaciones `wrf`, `namd`, y `leslie3d`.

- Categoría 2. Estas aplicaciones también tienen un RBHB alto; sin embargo, el porcentaje de aciertos no cambia de forma significativa cuando se añaden más buffers fila. Algunos benchmarks que pertenecen a esta categoría son `zeusmp`, `GemsFDTD`, y `xalancbmx`.

- Categoría 3. Incluye las aplicaciones con un bajo RBHB; sin embargo, añadiendo más entradas se incrementa marcadamente el número de aciertos. Algunos ejemplos son: `libquantum`, `lbm`, y `soplex`.
- Categoría 4. Las aplicaciones en esta categoría tienen poca localidad espacial, así que tienen un RBHR bajo incluso incrementado el número de buffers fila. Algunos benchmarks en esta categoría son `mcf` y `povray`.

El objetivo de este trabajo es diseñar tablas de buffers fila que se aprovechen del comportamiento de los aciertos en los buffers fila de las aplicaciones incluidas en las Categoría 1 y, especialmente, en la Categoría 3, sin afectar negativamente al rendimiento de las aplicaciones de las otras categorías.

1.4. Estructura del trabajo

El resto del presente trabajo se organiza de la siguiente manera. En el capítulo 2 se explicarán los fundamentos teóricos básicos sobre los que se ha desarrollado el trabajo y, por consiguiente, la base del sistema. En el capítulo 3 se hace referencia a trabajos previos y simultáneos relacionados con la propuesta. En el capítulo 4 se describirán detalladamente las propuestas de mejora usando las tablas BRT y CRT. En el capítulo 5 se explicará el entorno de simulación utilizado para el trabajo y los benchmarks utilizados para las simulaciones. En el capítulo 6 se mostrarán y analizarán los resultados de rendimiento y energía obtenidos de las diferentes propuestas. Por último, en el capítulo 7 se resumirán las conclusiones obtenidas y se describirá el trabajo futuro.

Capítulo 2

Estructura básica del sistema modelado

En este capítulo, se describen los aspectos fundamentales de un sistema multinúcleo. Además, para una mejor comprensión de la totalidad del sistema, se describe: la jerarquía de memoria, los núcleos, y la red de conexión que interconecta éstos.

El sistema modelado [JNW07] está basado en un procesador multinúcleo donde los distintos núcleos se modelarán como procesadores superescalares agresivos de manera análoga a los procesadores reales. Los distintos núcleos se encontrarán interconectados mediante una red de interconexión.

2.1. Núcleos

El procesador ha sido tradicionalmente el foco de los diseñadores de sistemas ya que proporciona la cada vez más exigida potencia de cálculo de la máquina. Esto se puede lograr sobre todo siguiendo dos direcciones principales: la mejora de la potencia de cálculo de los núcleos individuales, aumentar el número d núcleos, o combinar ambas acciones en conjunto. La mayoría de los fabricantes de procesadores optan por los procesadores multinúcleo con el objetivo de proporcionar un buen equilibrio entre rendimiento y potencia. Para este fin, incluyen varios núcleos simples. Sin embargo, los recientes avances en la tecnología y las técnicas microarquitecturales de conciencia energética, han permitido a los fabricantes desplegar el poder de la agresividad de las ejecuciones fuera de orden en dispositivos. Desde un punto de vista de alto nivel, dos rasgos principales caracterizan el procesador, la cantidad de núcleos y su agresividad.

2.2. Red de interconexión

La red de interconexión es la encargada de comunicar los elementos del sistema entre si, y posibilitar que intercambien peticiones de acceso a memoria y bloques de datos en respuesta a dichas peticiones. Concretamente, la red conecta las cachés entre si y con el controlador de memoria. Por lo que cada vez que una petición de acceso a memoria falle en una de estas cachés, esta petición se enviará al controlador de memoria a través de la red de interconexión. Por su parte, el controlador de memoria obtendrá el dato pedido de memoria principal y lo enviará de vuelta a la caché que lo pidió a través de la red de interconexión.

La red de interconexión se compone principalmente de dos elementos: conmutadores y enlaces. Los enlaces conectan las cachés con los conmutadores, los conmutadores entre sí y el controlador de memoria con los conmutadores. Por su parte, los conmutadores son los encargados de intercambiar mensajes entre los enlaces a los que está conectado. Por decirlo de alguna forma, los enlaces son las carreteras por las que circulan las peticiones de memoria y los conmutadores son como guardias de tráfico situados en los cruces que deciden qué mensajes cruzan por el cruce y qué camino de salida han de tomar. En esta analogía se ven los tres principales papeles que tiene un conmutador: arbitrar, encaminar y conmutar.

En primer lugar, un conmutador debe decidir cual de todas las peticiones que están intentando cruzarlo es la que va a cruzarlo definitivamente, esto se conoce como arbitraje. Por otro lado, un conmutador debe decidir por qué otro enlace va a enviar la petición que lo va a cruzar, normalmente lo enviará por el enlace que lo acerque a su destino sea éste el controlador de memoria o una caché, esto se conoce como encaminamiento. Finalmente, una vez decidido qué petición será la ganadora del arbitraje y qué enlace tomará, el conmutador permitirá que la petición cruce desde el enlace en el que se ha recibido al enlace por el que se encamina; esto se conoce como conmutación.

La parte del encaminamiento depende enormemente de dos parámetros de diseño de la red de interconexión: la topología y el algoritmo de encaminamiento. La topología define la forma de la red, es decir, qué elemento se conecta con qué otro elemento mediante un enlace. En otras palabras, define los enlaces que habrá entre las cachés, el controlador de memoria y los conmutadores. Por continuar con el símil anterior, la topología define el mapa de carreteras de la red. En cuanto al encaminamiento, el algoritmo de encaminamiento define la ruta que se tiene que seguir para llegar desde

cada posible origen de las peticiones a cada posible destino, lo que significa que define las rutas que seguirán las peticiones a través de la red de interconexión. Ambos parámetros son cruciales ya que las prestaciones de la red dependen fuertemente de ellos.

2.3. Protocolo MOESI

Existen muchas alternativas para diseñar protocolos de coherencia dependiendo de los estados de los bloques almacenados en las cachés privadas. Estas alternativas suelen ser nombradas en función de los estados que utilizan: MOESI, MOSI, MESI, MSI, etc. Cada estado representa unos permisos de lectura y escritura distintos para el bloque almacenado en la caché privada. Para este proyecto se ha tenido en consideración el protocolo MOESI [ea03], que dispone de un mayor número de estados (el resto de protocolos utilizan un subconjunto de estos). Estos estados son:

- M (Modified): Un bloque en estado modificado mantiene la única copia válida de los datos. El núcleo que mantiene esta copia en su caché tiene permisos de lectura y escritura sobre el bloque. Las otras cachés privadas no pueden tener una copia. Las copias en las cachés de niveles inferiores (si están presentes) son obsoletas. Cuando otro núcleo solicita el bloque, la caché con el bloque en estado modificado debe proporcionarlo, y pasará a estado Owner.

- O (Owner): Un bloque en estado propietario mantiene una copia válida de los datos, pero en este caso, otras copias en estado compartido pueden coexistir. Únicamente puede haber una copia de ese bloque en estado propietario. El núcleo que mantiene esta copia en su caché tiene permisos de lectura, pero no puede modificarlo. Cuando este núcleo trata de modificarlo, se requieren acciones de coherencia para invalidar el resto de copias. De esta forma, el estado propietario es similar al compartido. La diferencia reside en el hecho de que la caché que posee el bloque en estado propietario es responsable de proporcionar la copia del bloque ante un fallo de caché, ya que la copia en las cachés compartidas de niveles inferiores (si está presente) es obsoleta. Además, los desalojos de bloques en estado propietario siempre conllevan operaciones de escritura a su nivel superior.

- E (Exclusive): Un bloque en estado exclusivo mantiene una copia válida de los datos. Las otras cachés privadas no pueden tener una copia de este bloque. El núcleo con esta copia tiene permisos de escritura y lectura. Las cachés compartidas de niveles inferiores pueden tener también almacenada una copia válida del

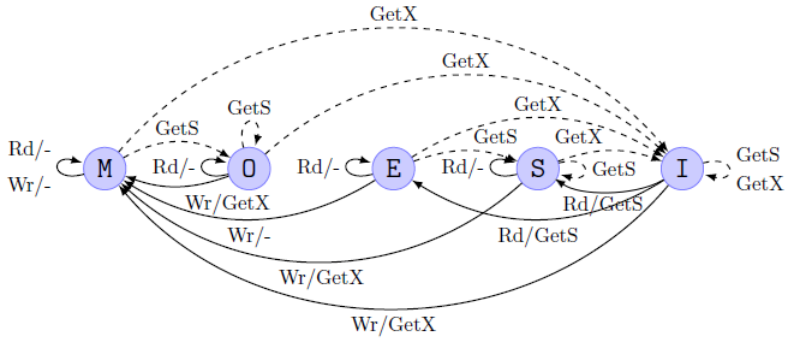


Figura 2.1: Esquema de transiciones entre estados del protocolo MOESI.

bloque.

- S (Shared): Un bloque en estado compartido mantiene una copia válida de los datos. Otros núcleos pueden tener copias del bloque en estado compartido y uno de ellos en estado propietario. Si ninguna caché privada tiene el bloque en estado propietario, las cachés compartidas disponen también de una copia válida del bloque y son responsables de proporcionarlo si fuera solicitado.
- I (Invalid): Un bloque en estado inválido no mantiene ninguna copia válida de los datos. Las copias válidas pueden encontrarse bien en otros niveles de la jerarquía de caché o en otra caché privada del mismo nivel.

Las transiciones entre estados se ilustran en la Figura 2.1. A continuación se explican en detalle.

- Rd/-: Lectura de un bloque local.
- Wd/-: Escritura sobre un bloque local.
- GetX: Se recibe una petición de invalidación (otro núcleo quiere acceso exclusivo al bloque).
- GetS: Se recibe una petición de un bloque por parte de otro núcleo.

2.4. Tecnologías de memoria

Los sistemas CMP deben diseñarse para ajustarse a presupuestos específicos de área y energía. Ambas restricciones tecnológicas representan un problema general, dado que dificultan la escalabilidad de los futuros CMPs con el incremento de núcleos. El

consumo de energía está distribuido entre los núcleos y las grandes cachés de memoria en los diseños de chips actuales. Las cachés ocupan un elevado porcentaje del área del chip para mitigar las altas latencias que corresponden con un acceso a memoria principal. Dando más área de silicio y energía a la jerarquía de memoria y estructuras relacionadas (por ejemplo las cachés de directorio) deja menos espacio y energía para los núcleos, lo que fuerza a los diseños de CMPs a usar núcleos más simples reduciendo la productividad, especialmente para aplicaciones de un solo hilo [MH08].

Ha habido muchos esfuerzos por parte de la industria y el mundo académico para enfrentarse al problema de área y energía en el subsistema de caché, incluyendo las cachés del procesador, cachés fuera del chip y estructuras de directorio. Con respecto a estas últimas estructuras, las cachés de directorio han demostrado ser efectivas para escalar tanto en energía como en área cuando el número de núcleos es medio o bajo. En cualquier caso, estas dificultades de diseño deben afrontarse correctamente para sistemas futuros, ya que la presión de obtener buenas prestaciones incrementa con el número de núcleos. Hay dos formas de aproximarse a estas dificultades: ofrecer soluciones estructurales para conseguir un buen balance entre productividad, área y consumo, y combinar distintas tecnologías. Ambos casos se pueden aplicar de manera independiente o conjunta.

La jerarquía de memoria de los CMPs se suele implementar con una tecnología SRAM (6 transistores por celda) que consume una cantidad importante de energía y área. Hace unos pocos años, los avances tecnológicos permitieron el uso de celdas eDRAM en tecnologías CMOS [MS05]. La Tabla 2.1 muestra cómo estas tecnologías se comportan para los distintos aspectos de diseño estudiados en este trabajo. Comparadas con las celdas SRAM, las celdas eDRAM presentan menos consumo de energía y una mayor densidad, pero menos velocidad. A causa de la reducida velocidad, las celdas eDRAM no se usan para fabricar cachés de procesador de primer nivel y de altas prestaciones.

La idea de combinar las tecnologías descritas ha sido empleada tanto en la indus-

Cuadro 2.1: Características de las tecnologías eDRAM y SRAM.

Tecnología	Densidad	Velocidad	Potencia
SRAM	baja	rápida	alta
eDRAM	alta	lenta	lenta

tria como el ámbito académico, pero a diferencia de nuestro trabajo, ellos se centraban en cachés de procesador convencionales. Por ejemplo, en algunos microprocesadores modernos [TDF+02, SKT+05, KSSF10] se usa SRAM en las cachés L1 del procesador mientras que se usa eDRAM para permitir grandes capacidades de almacenamiento en las cachés de último nivel. Con respecto al campo académico, algunos trabajos recientes [VSP+09, WLZ+09] se han publicado mezclando ambas tecnologías en la misma y/o diferentes estructuras de la jerarquía de memoria.

2.5. Controlador de memoria

La función de un controlador de memoria DRAM es gestionar el flujo de datos de entrada y de salida de los dispositivos DRAM conectados a ese controlador en el sistema de memoria. Sin embargo, debido a la complejidad de los protocolos de acceso a la memoria DRAM, el gran número de parámetros de temporización, las innumerables combinaciones de las organizaciones del sistema de memoria, las diferentes características de carga de trabajo, y los diferentes objetivos de diseño, el espacio de diseño de un controlador de memoria para un dispositivo DRAM dado tiene muchos grados de libertad. Un protocolo de acceso a la DRAM define el protocolo de interfaz entre un controlador de memoria DRAM y el sistema de dispositivos DRAM. En ambos casos, las características de rendimiento reales dependen de las implementaciones microarquitecturales específicas en lugar de la descripción superficial de un modelo de programación o de protocolo de interfaz. Es decir, dos controladores de memoria DRAM que apoyan el mismo protocolo de acceso a la DRAM pueden tener enormes diferencias en latencia y ancho de banda, en función de las respectivas implementaciones microarquitecturales.

El controlador de memoria contiene una serie de colas donde se depositan las peticiones de acceso a memoria a la espera de ser atendidas. Hay dos posibles implementaciones de las colas: usar una única cola para todas las peticiones y usar una cola por banco. En nuestro estudio se ha optado por esta segunda opción ya que es la más realista. Cuando estas colas se llenen al máximo no será posible enviar más peticiones al controlador por lo que el sistema se congestionará.

El controlador de memoria acepta solicitudes de uno o más núcleos y uno o más dispositivos de E/S y proporciona la interfaz de arbitraje para determinar qué solicitud entrará en la de memoria principal. Una vez que una transacción ha sufrido arbitraje se convierte en una secuencia de comandos de DRAM. Para acceder a un bloque el controlador envía diversos comandos, como *precarga*, *activación*, *lectura* y *escritura*.

- El comando de *precarga* borra el contenido previo del RB y lo almacena en el banco si la fila hubiese sido modificada.
- El comando de *activación* dispara la lectura de la fila solicitada, almacenándola en el RB.
- El comando de *lectura* solamente lee una columna dada de la fila previamente activada.
- El comando de *escritura* escribe un bloque en una columna dada previamente activada.

La secuencia de comandos se coloca en una de las colas que existen en el controlador de memoria. Si la cola está completa el controlador devuelve la orden al nivel de memoria anterior, normalmente L2, para que reintente la inserción en el controlador en un momento posterior. Una vez insertada la petición, éstas se atienden según su prioridad.

2.6. Memoria principal

El subsistema de memoria principal ha llegado a ser la principal preocupación del diseño de procesadores multinúcleo debido a dos razones. Primera, éste representa el

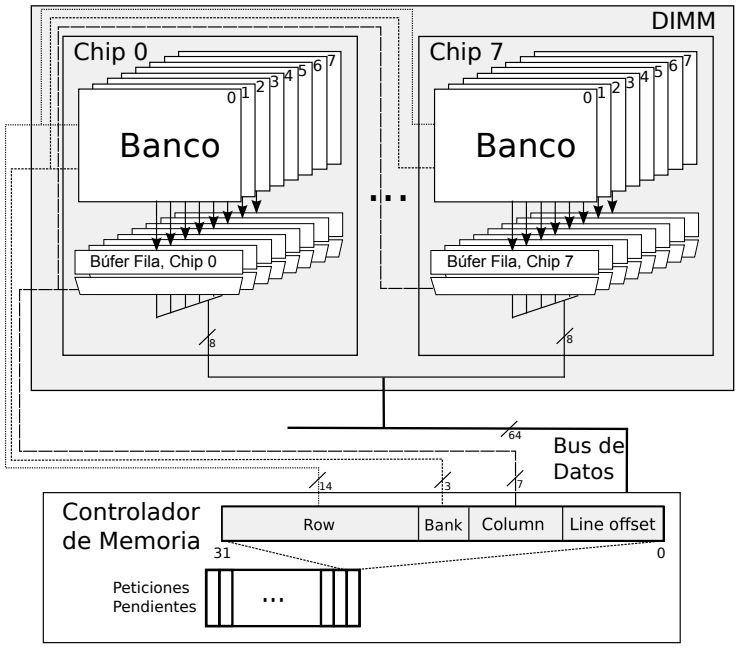


Figura 2.2: Arquitectura DRAM: DIMM, chips, vectores y buffers fila.

mayor cuello de botella, que se agrava al incrementar el número de núcleos, ya que estos compiten por el acceso a un determinado controlador. Segundo, el coste de la memoria principal representa una fracción importante del coste total del sistema [HP12]. La Figura 2.2 muestra el diagrama de esta organización.

Las peticiones de memoria generadas por los núcleos (por ejemplo, fallos de caché en los últimos niveles) se encolan en el controlador de memoria, y la traduce las peticiones a órdenes de los módulos de memoria.

Los canales de memoria son el medio físico que interconecta el controlador de memoria con los chips de memoria, por lo que un controlador de memoria puede emitir tantas órdenes simultáneamente a la memoria principal como canales de memoria haya.

Después de que una orden es emitida por el controlador de memoria a través del canal, el banco es accedido y el canal es liberado hasta que los datos estén disponibles en el búfer fila (salida de banco). Suponiendo que una transacción por el canal tiene una duración de un ciclo, el controlador de memoria puede emitir dos solicitudes distintas a través del mismo canal en dos ciclos consecutivos de bus (canal) cuando se destinen a diferentes bancos de memoria (es decir, no surge ningún conflicto de banco). Obsérvese que si dos solicitudes se dirigen al mismo banco de memoria, el controlador de memoria detendrá la segunda hasta que se complete la primera, puesto que se produce lo que se denomina conflicto de banco.

La Figura 2.3 representa el diagrama de una arquitectura de memoria moderna que consta de dos canales de memoria, dos rangos por canal y dos bancos por rango. El propósito de la organización en canales, bancos y rangos es mejorar el acceso paralelo a

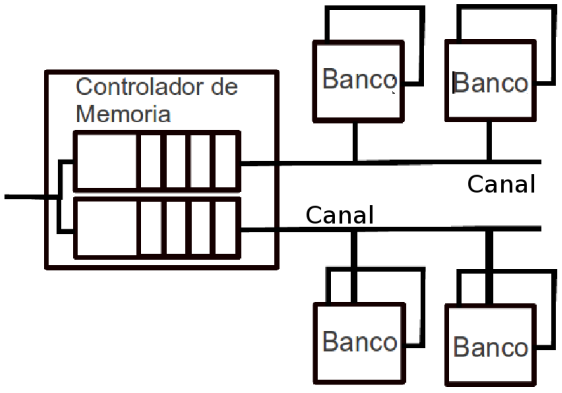


Figura 2.3: Esquema del controlador de memoria.

la memoria con el fin de aliviar la enorme latencia de acceso a memoria que se exacerba con el número de núcleos.

2.6.1. Canales

Los canales comunican físicamente el controlador de memoria con los bancos en última estancia. Constan de un bus de datos, donde se transfieren los bloques de datos desde/hacia memoria, y un bus de comandos, por donde se envían los comandos. Estos buses son compartidos por un grupo de bancos y tienen mecanismos para detectar colisiones, de forma que si alguna petición está accediendo, otra petición que quiera usar el canal tendrá que esperar a que éste se libere.

2.6.2. Rangos

En esencia, un rango es un conjunto de dispositivos DRAM que trabajan al mismo tiempo para responder a una orden dada en el sistema de memoria. Las direcciones y las órdenes se envían a través de buses independientes que están conectados a cada dispositivo DRAM en el sistema de memoria. El ancho del bus de datos se distribuye entre los diferentes dispositivos DRAM que se encuentran conectados al bus. Por ello, los diferentes rangos compiten en el acceso a este bus, también conocido como canal.

2.6.3. Bancos

Los bancos son matrices de bloques dentro en un dispositivo DRAM. Las dispositivos DRAM modernos contienen múltiples bancos para que los diferentes accesos a las matrices DRAM puedan ocurrir en paralelo. En este diseño, cada banco de memoria es una matriz independiente que puede estar en diferentes fases del ciclo de acceso a una fila. Aunque algunos de los recursos deben ser compartidos entre los diferentes bancos, hay una clara mejora resultado de usar múltiples bancos debido a su funcionamiento independiente y simultáneo.

Los módulos actuales tienen dos formas de acceso: página abierta y página cerrada. Básicamente, estas dos políticas se diferencian en el uso de los RB y como resultado, en la secuencia de comandos que el MC envía a los bancos de memoria.

Página Abierta. En este modo, el búfer fila almacena el contenido de la última fila leída/escrita hasta que un nuevo comando de precarga se lance para leer/escribir una nueva fila. Si la petición acierta en el búfer fila (su bloque se encuentra allí), no se lanza el comando de *precarga* ya que posibles futuros accesos también podrían acertar, esto hace que esta operación sea 3× veces más rápida que leer la fila otra vez. Este hecho mejora el rendimiento general para la mayoría de aplicaciones.

Para aprovechar la localidad del búfer fila y mejorar el rendimiento, los controladores de memoria actuales implementan la política FR-FCFS [RDK+00] la cual prioriza las peticiones que están en las colas del MC siguiendo dos pasos. Primero, se seleccionan como candidatas a ser servidas aquellas peticiones que aciertan en el búfer fila y tienen los recursos que necesitan libres (bus, banco, ...). Luego, la petición más vieja (first come first served) de las seleccionadas en el primer paso tiene la máxima prioridad. Cuando ya no existen más peticiones que acierten en el búfer fila o éstas tengan sus recursos ocupados, se seleccionan peticiones que son fallos en el búfer fila y por tanto se requiere vaciar el RB con un comando de *precarga* y cargar la fila en el RB con una *activación*.

Página Cerrada. En este modo el MC envía un comando conjunto de *activación* y de *lectura/escritura* a la memoria en cada acceso ya que el búfer fila no almacena la última fila accedida. Cuando se ha leído/escrito el bloque, se envía el comando de *precarga* para eliminar la fila del búfer. Esta política se elige en algunas tecnologías como Hybrid Memory Cube [JK12] y las próximas generaciones de productos nVidia [Smi] para ahorrar energía, ya que mantener las filas abiertas consume mucha energía.

2.6.4. Búfer Fila

En los dispositivos DRAM, una *fila* es un grupo de celdas de almacenamiento que se activan en paralelo en respuesta a una orden de activación de fila. Cada banco contiene un búfer del tamaño de una fila. Cuando una fila de un banco es accedida, se activa y es transferida a un búfer temporal que contendrá su información evitando activar la fila otra vez si es accedida de nuevo, cosa que consume mucho tiempo. Este búfer, conocido como búfer fila, permite una reducción del tiempo de acceso ya que el almacenamiento temporal de la fila permite accesos consecutivos a dicha fila sin acceder al banco de memoria. Esto permite reducir el tiempo de acceso al banco ya que no es necesario ir al banco, es decir, pasar por el descodificador de fila y activar el *word line* (salida del descodificador que activa toda la fila). Si la fila se cierra (página cerrada) sí que

es que necesario acceder al banco. Por contra, esta política consume mucho menos que mantener la fila abierta.

Capítulo 3

Trabajo relacionado

La estructura del sistema de memoria principal se ha basado en trabajos anteriores [JNW07] donde se definen los elementos principales de la memoria principal: canales, bancos, rangos, búfer fila, etc. Así como la interconexión entre estos elementos y cómo se comportan las peticiones dentro de sistema de memoria.

Las arquitecturas DRAM tradicionales son altamente ineficientes para los servidores de centro de datos y plataformas HPC, por ello necesitan una importante reforma [UMC+10]. Consecuentemente una gran cantidad de trabajos de investigación se ha concentrado en mejorar el desempeño de la memoria principal.

En el pasado, un importante trabajo [RZ97] se centró en reordenar el acceso a memoria principal para mejorar la productividad de la memoria principal. Trabajos recientes ha abordado este problema en procesadores multicore [RZ97, RDK+00, LMNP11, ELMP11], donde la localidad del búfer fila de una aplicación dada varia dependiendo de los *co-runners*. Por lo tanto, estos enfoques permiten emitir fuera de orden peticiones de memoria de varias aplicaciones con el objetivo de maximizar los aciertos en el búfer fila. El orden de acceso se selecciona de acuerdo con una política de planificación que elige las peticiones que próximamente se emitirán.

Algunos trabajos [KPMHB10, MM08, MM07, ZLZZ08] proponen una planificación de memoria inteligente de acceso a memoria por diferentes hilos, para mejorar tanto la localidad del búfer fila como el paralelismo a nivel de banco, sin perjudicar a algún hilo (fairness). Uno de estos trabajos [KPMHB10] propone una nueva política de planificación que mejora tanto la productividad como la igualdad de acceso entre diferentes hilos a los recursos de memoria. Para ello agrupa los hilos en *memory-intensive* y *non-memory-intensive*, y aplica una priorización entre ambos grupos. En otro trabajo [MM07] propone usar heurísticas para estimar qué hilo es perjudicado comparándolo

con su ejecución solitaria y éste se prioriza. Esto permite un acceso más justo, sin embargo, se produce pérdida de productividad. Algunos trabajos [MM08] clasifican los hilos y priorizan las peticiones de hilos *memory-intensive*, que se traduce en largas esperas para hilos *non-memory-intensive*. Para dirigir decisiones de planificación, sin embargo, se realiza un seguimiento de los patrones de acceso al controlador de memoria, el cual puede requerir una complejidad del hardware significante. Se ha de tener en cuenta que las estrategias de planificación son ortogonales a este trabajo.

Recientes aproximaciones se han centrado en nuevas organizaciones de la DRAM [UMC+10, ZLZ+08, YJE11, ALSJ09, GMMG12, HGCT13].

La idea de usar pequeños buffers fila ha sido ampliamente usada en trabajos recientes [ZLZ+08, YJE11, ALSJ09, GMMG12, MLM12, LIMB09]. La idea detrás de estos esquemas es ahorrar energía [GMMG12] mientras se mantiene el rendimiento gestionando apropiadamente pequeños buffers fila en vez de buffers más grandes (del tamaño de la fila). En contraste, nuestra propuesta CRT utiliza pequeños buffers fila en las tablas localizadas en el MC para ahorrar tiempo de transferencia, y en consecuencia reducir el tiempo de espera para conseguir el bus y así aumentar el rendimiento.

En entornos mutinúcleo, el rendimiento aislado y la igualdad en el acceso a recursos es tan importante como la productividad. Algunos autores [HGCT13] se han dado cuenta de que un único búfer fila puede dañar el rendimiento en cargas multiprogramadas y han propuesto un esquema que asigna un búfer fila dedicado a cada hilo. Esto mantiene la localidad de cada hilo, garantizando el rendimiento individual de las aplicaciones. A diferencia de nuestro enfoque, este esquema supone un solo búfer fila fijo para cada hilo ejecutándose, lo cual asume unos requerimientos de recursos iguales, que como se discutió en el apartado 1.3 varían según la aplicación. Además, se demuestra que mantener los buffers fila en el controlador de memoria puede mejorar altamente el rendimiento en las aplicaciones con alta localidad en el RB.

Un enfoque interesante trata de explotar la corta distancia del controlador de memoria en el chip [YKK+13]. Los autores proponen un esquema que rastrea peticiones de prebúsqueda en el controlador de memoria, entonces la circuitería adicional del MC intenta predecir si se necesitarían más prebúsquedas de una fila antes de cerrarla. Si es así, entonces los bloques que se predice que son prebúscados por los núcleos se almacenan en un pequeño búfer en el MC. Hay que tener en cuenta que este enfoque introduce el uso de un componente de almacenamiento en el MC como hacemos en el esquema de la CRT, pero limitado a prebúsquedas predichas por los núcleos.

Capítulo 4

Propuestas de mejora

Como se ha estudiado en el apartado 1.3 un gran conjunto de aplicaciones ejecutándose solas mejoran notablemente su rendimiento con múltiples búfers fila. El objetivo de las aproximaciones propuestas es beneficiarse de esta observación. Aunque estas propuestas pueden aplicarse en cualquier entorno, en este trabajo se ha centrado en CMPs con cargas multiprogramas donde las interferencias entre aplicaciones en el RB pueden ser más importantes.

En este capítulo se introduce el diseño y la gestión de una tabla fila, la cual consiste un pequeño conjunto de búfers fila, donde cada banco tiene su propia tabla fila. Se han diseñado dos esquemas diferentes dependiendo de donde se localizan las tablas. En la primera aproximación, la tabla se encuentra en los módulos de memoria o bancos como una extensión de los actuales búfers fila de la DRAM; esta propuesta se llama BRT (Bank Row Table). La segunda opción, conocida como CRT (memory Controller Row Table), intenta explotar el comportamiento de los RB pero las tablas fila se implementan en el lado del MC. Para realizar esto, la página de memoria es transferida al MC cuando se lee en el banco, de esta forma se aceleran los múltiples accesos a la misma fila.

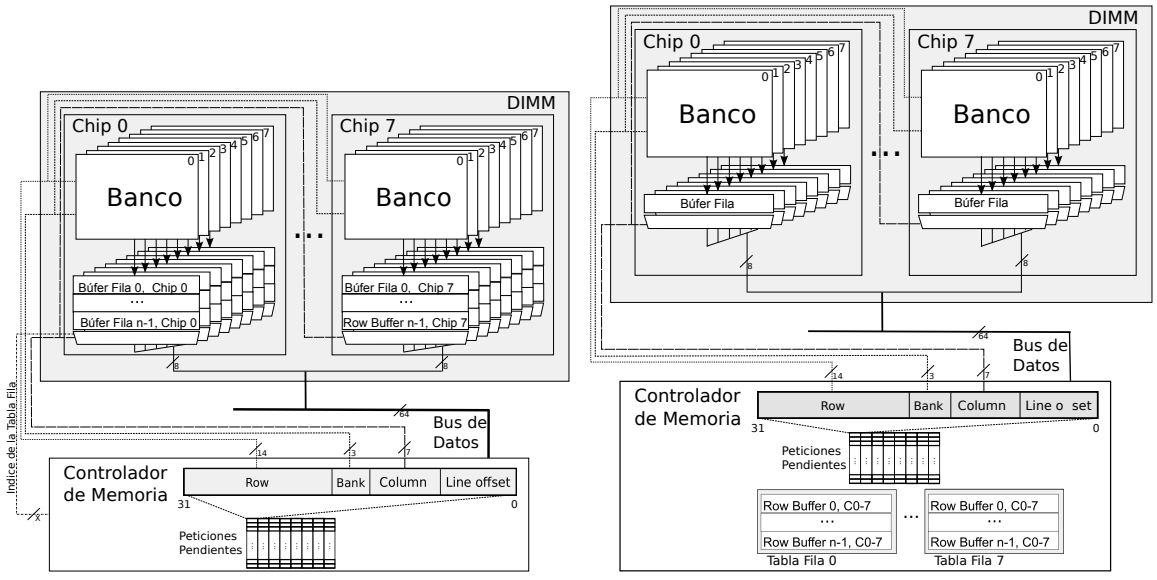
Asimismo, debido al aumento de recursos disponibles para que las peticiones se sirvan, ligado a un mayor número de búfers fila, son necesarios mecanismos que detecten posibles conflictos de concurrencia entre las peticiones, aseguren la independencia entre peticiones, y eviten posibles abrazos mortales entre peticiones o bloqueos infinitos de los recursos, pero intentando maximizar el paralelismo.

4.1. Esquema BRT

Este esquema es la solución más sencilla, consiste en replicar los RB (buffers fila) en los módulos de memoria. La Figura 4.1(a) representa un módulo DDR típico. Este conjunto de búfers está compartido entre todos los núcleos (y aplicaciones) del sistema, y se gestionan como un todo, sin tener en cuenta el núcleo que emite la petición de memoria. Los buffers fila se asignan dinámicamente a los núcleos en tiempo de ejecución de acuerdo con las necesidades de las aplicaciones. Se ha optado una política LRU (least recently used) para desasignar y asignar las entradas (filas) de la tabla. Esto significa que una tabla con n entradas mantiene los contenidos de las n filas más recientemente referenciadas del banco asociado.

Las tablas fila añaden una pequeña complejidad. Además del contenido de una página (fila) de memoria (en este caso, 4KB), cada entrada requiere algunos campos extra, tales como un campo para controlar la política de reemplazo LRU (2 bits para una tabla con 4 entradas) y un campo "válido"(1 bit) para controlar si hay alguna petición accediendo a esa entrada en ese instante.

BRT apenas necesita cambios en la lógica del planificador de memoria en el MC, ya que éste debe ser consciente de todas las filas almacenadas en la tabla para distinguir aciertos de fallos y proceder de acuerdo con una adecuada secuencia de comandos de memoria. Teniendo esto en cuenta, en los resultados mostrados en la Figura 1.1, BRT



(a) BRT (b) CRT

Figura 4.1: Las dos implementaciones propuestas de la tabla fila.

puede potencialmente conseguir importantes ganancias en el rendimiento.

Para poder servir una petición es necesario garantizar que están disponibles los recursos necesarios. Por ello, antes de extraer una petición de las colas del controlador de memoria, se ha de garantizar que i) el bus de comandos de memoria esté libre, ii) hay al menos una entrada o RB disponible, y en el caso de que sea un fallo (no esté la fila en la tabla) iii) el banco no debe estar sirviendo otro comando. Posteriormente se deberá solicitar el bus de datos para leer/escribir un bloque, hasta que este recurso no esté garantizado la petición deberá esperar a que se libere el bus.

Observando las posibles combinaciones que pueden ocurrir cuando varias peticiones acceden a un mismo banco, dos peticiones pueden obtener su bloque al mismo tiempo si ambas son aciertos en las tablas fila y acceden a una entrada diferente (filas diferentes), a diferencia de los sistemas con solo un RB, ya que lo obtienen de buffers diferentes. Sin embargo, como ambas tienen que transferir su bloque por el mismo bus de datos, liberan los RB/entradas y se almacenan en un búfer de salida que detecta cuando el bus de datos está vacío y puede transferir los bloques en orden de llegada. Por otro lado, en caso de que haya un fallo y un acierto, las peticiones pueden obtener su bloque concurrentemente siempre que accedan a filas diferentes. Sin embargo, dos fallos al mismo banco no pueden servirse al mismo tiempo ya que ambos necesitan acceder al banco en el mismo instante, es decir, que se procesen diversos comandos a la vez.

Por lo tanto, tener más filas en la tabla incrementa la concurrencia, que crecerá cuantos más aciertos haya.

4.2. Esquema CRT

El tiempo de transferencia (a través del bus de memoria) de múltiples peticiones que aciertan en los buffers fila puede limitar el rendimiento en aplicaciones con alta localidad en el RB ya que, aunque accedan al bloque rápidamente, tienen que esperar a transferir su bloque por el bus. Para solucionar este problema, esta aproximación implementa las tablas fila en el controlador de memoria y envía la página al MC cuando ésta es leída.

La Figura 4.1(b) muestra el diagrama de esta organización. Se puede apreciar que el conjunto de buffers fila se implementa en el MC y los módulos de memoria solo tienen 1 RB, como los dispositivos estándar. Por lo tanto, CRT puede ser usado en los módulos DRAM actuales ya que se implementa completamente en el MC y no es necesario modificar los chips de memoria estándar actuales.

En el esquema CRT, cuando hay una fallo en la tabla fila correspondiente a un

banco DRAM, se accede al banco en el módulo de memoria y su fila es transferida al MC donde se escribe en una entrada de la tabla fila. CRT transfiere el bloque crítico primero, es decir, el bloque que ha causado el fallo es servido primero y se puede enviar directamente al núcleo una vez que es leído del RB del banco. Pero las peticiones siguientes a la misma fila esperan a que la fila esté completamente almacenada en la tabla antes de ser servidas.

Al igual que el esquema BRT, la tabla mantiene las n filas más recientemente accedidas pero en el lado del controlador de memoria, y las entradas se reemplazan siguiendo la política LRU. Consecuentemente, las peticiones de memoria subsecuentes completan su lectura y escritura mucho más rápido ya que la fila ya esta almacenada en el MC, así que no hay necesidad de acceder al módulo de memoria (banco). Comparado con un esquema típico de 1 RB, cuando la aplicación tiene localidad espacial, CRT puede reducir tanto el tiempo de acceso debido a que incrementa el porcentaje de aciertos en el RB, como el tiempo de transferencia ya que la fila se toma del MC y el bloque pedido no tiene que se transferido a través del bus de memoria.

CRT trabaja en modo página cerrada ya que cada vez que una fila se lee siempre es transferida a una tabla fila en el MC, desde donde ésta se toma en peticiones posteriores a esta página.

La complejidad de las tablas fila es similar a la discutida en el esquema BRT. CRT tiene mecanismos similares a BRT; sin embargo, la detección de cuando una petición puede ser lanzada es menos restrictiva. Al contrario que BRT que detecta si el bus de datos está ocupado, CRT no necesita transferir el bloque del banco al MC, y por lo tanto, no necesita que el bus de datos esté libre posteriormente y tanto los aciertos en las tablas como los fallos pueden procesarse a la vez. Múltiples aciertos a un mismo banco se procesan igual que en BRT, ya que es necesario asegurar que cada entrada de las tablas solo sea accedida por una petición a la vez, así que se bloquea (se pone el bit "válido.ª 1) y cuando termina de realizar las operaciones necesarias se desbloquea. Los fallos en las tablas se procesan como en los sistemas clásicos con 1 RB en cada banco, y como trabaja en página cerrada solo puede acceder una petición cada vez al banco. No obstante, es necesario garantizar que cuando se transfiere una fila del banco ésta tenga una entrada en la tabla correspondiente donde alojarse, por ello se bloquea una entrada que se libera después de que el bloque llegue a la tabla y la fila se mantiene en la entrada hasta que es reemplazada por otra fila.

En el caso de las escrituras, para garantizar la integridad de la información, si el bloque se encuentra en una de las tablas fila, éste se actualiza con los nuevos datos y a la vez se escribe en el propio banco. Por el contrario, si el bloque no se encuentra en

las tablas fila, éste se actualiza en el banco pero no se guarda su contenido en la tabla, ya que no hay garantía que siguientes lecturas accedan a él. Por lo tanto, las tablas fila implementan la política *write-through*, es decir, una petición de escritura se escribe tanto en la tabla fila como en el banco de memoria, por lo tanto, la información se mantiene *consistente* en ambos lados.

CRT ayuda a reducir tanto el tiempo de acceso (ya que el porcentaje de aciertos en el RB será alto) y el tiempo de transferencia (ya que la fila es tomada del MC) en aplicaciones con buena localidad espacial en comparación con el típico esquema de 1 RB.

4.3. Variante $CRT_{1/x}$

El esquema CRT presenta algunas ventajas e inconvenientes con respecto a BRT.

CRT transfiere especulativamente la página entera como una ráfaga larga aunque hayan pocas peticiones esperando en el MC para solicitar la misma página. Esto beneficia el rendimiento de las aplicaciones con alta localidad en el búfer fila. El gran tiempo de transferencia, sin embargo, puede incrementar significativamente el tiempo que esperan las peticiones que son falladas en las tablas fila para conseguir el bus, lo cual puede impactar negativamente el rendimiento. Asimismo, se puede producir un incremento importante del consumo de energía (ver apartado 6.4) si la mayoría de los bloques de la página no se piden más tarde.

Para solucionar este problema, se ha propuesto una variante del CRT que consiste en transferir solo una fracción de la página (fila) en vez de la página entera. Se han analizado dos fracciones diferentes: un cuarto (1KB) y un sexto (256B) de la página entera (4KB), conocidas como los esquemas $CRT_{1/4}$ y $CRT_{1/16}$, respectivamente.

La fracción de la página elegida es aquella que contiene el bloque pedido. Para seleccionar esta fracción, se extraen los 2 y 4 bits de mayor peso aplicando una mascara para $CRT_{1/4}$ y $CRT_{1/16}$, respectivamente. Estos bits determinan a que fracción pertenece el bloque, y la lógica del banco los usa para leer la correspondiente *sub-página*. Cabe notar que esta aproximación necesita un campo adicional en las entradas de las tablas fila para identificar la sub-página almacenada.

Finalmente, a diferencia de trabajos anteriores sobre sub-filas que persiguen reducir el consumo energético mientras mantienen el mismo rendimiento o permitir una área más grande de almacenamiento con sub-filas más pequeñas (y así mejorar el rendimiento), el esquema $CRT_{1/x}$ propuesto es el primero que hace uso de las sub-filas para reducir el tiempo de transferencia. Para propósitos de comparación el esquema CRT

usará el mismo número de *sub-filas* como número de filas implementadas en el esquema BRT.

Capítulo 5

Entorno de simulación

En este capítulo se describen las herramientas de simulación utilizadas durante este proyecto, el sistema simulado con todos sus parámetros, la metodología, las métricas y estadísticas empleadas, y finalmente una breve explicación de los benchmarks usados y cómo se han combinado para las pruebas de ejecuciones multinúcleo.

5.1. Herramientas de simulación

5.1.1. Multi2Sim

Multi2Sim [USPL07] es un simulador, basado en C, de un sistema completo capaz de simular distintos tipos de hardware, incluyendo sistemas multiprocesador. Tiene la capacidad de modelar procesadores segmentados superescalares, arquitecturas multihilo y multinúcleo, unidades de procesamiento gráfico (GPUs), y soporte para los benchmarks más comunes en investigación. Se trata de un simulador de tipo *application-only* que permite ejecutar una o más aplicaciones sin tener que arrancar primero un sistema operativo. El paradigma de simulación usado en este simulador se puede dividir en dos módulos: la simulación funcional y la detallada. La funcional es solo una emulación del programa que recibe como entrada y tiene el mismo comportamiento que tendría dicho programa ejecutado nativamente en una máquina x86. El simulador detallado ofrece un modelo de estructuras de CPU como cachés, unidades funcionales, redes de interconexión, etc., y un procesador con seis etapas (fetch, decode, dispatch, issue, writeback y commit), ofreciendo soporte para ejecución especulativa. El número de hilos y el número de núcleos es configurable. Se pueden configurar la jerarquía de memoria con distintos niveles de caché, con cualquier número de cachés en cada nivel (se mantiene la coherencia mediante el protocolo MOESI). Las cachés pueden ser unificadas

o separadas para datos e instrucciones, privadas o compartidas por núcleo, por hilo, o por unidad de cómputo de GPU, y pueden servir rangos específicos de direcciones físicas. El modelo de red de interconexión entre diferentes niveles de la jerarquía de memoria también es ampliamente configurable. Incluye un conjunto de nodos finales, de conmutadores y de enlaces que permiten definir la topología de la red; y una tabla de encaminamiento bidimensional que permite definir cualquier algoritmo de encaminamiento. Por otra parte, se puede obtener un informe detallado de las estadísticas relacionadas con la segmentación del procesador, clasificadas por hilo y por núcleo de ejecución. Hay resultados de simulación genéricos y estadísticas tanto para las estructuras hardware (búfer de reordenación, cola de instrucciones, archivo de registros, búfer de destino de salto), como para cada etapa del pipeline. El informe de estadísticas de jerarquía de memoria contiene una sección por cada caché, módulo de memoria principal y red de interconexión. Además, desde la versión 3.0, Multi2Sim ha aumentado sus capacidades de simulación con un modelo para las GPUs de AMD, que también cuenta con su propio depurador de pipeline. Dicho depurador es una herramienta para analizar la ejecución del código OpenCL en la unidad correspondiente del simulador. Mediante una interfaz gráfica actúa como una ayuda visual en la prueba del código y de la arquitectura del propio simulador. En sus últimas versiones, Multi2Sim usa el formato INI para todos sus archivos de entrada y salida. Finalmente, Multi2Sim se ha adaptado para proporcionar las estadísticas que requiere McPAT como archivos de entrada.

5.1.2. Sistema simulado

Para evaluar las propuestas se ha extendido el entorno de simulación multinúcleo Multi2sim [USPL07] para modelar el sistema propuesto, incluyendo el controlador de memoria y los módulos DRAM.

La configuración de los parámetros principales de los diferentes componentes del sistema (la red, la jerarquía memoria y el núcleo) se muestran en la Tabla 5.1. Los parámetros de memoria principal se han configurado según un dispositivo de memoria reciente MICRON DDR3 [mic11].

5.2. Métricas y metodología

Las propuestas presentadas en este trabajo se evalúan en el capítulo 6 en términos de prestaciones. En las pruebas realizadas, tanto en mezclas como en aplicaciones indi-

Núcleo	
Frecuencia	3GHz
Política de lanzamiento	Fuera de orden
Predictor de salto	bimodal/gshare hybrid: gshare con historia global de 14 bits + 16K contadores de 2 bits, bimodal con 4K contadores de 2 bits y selección con 4K contadores de 2 bits
Ancho de Issue/Commit	3 instrucciones/ciclo
Tamaño ROB	256 entradas
Cola de load/store	64/48 entradas
Jerarquía de cache	
L1 Icache	privada, 32KB, 8 vías, 64B-line, 2 ciclos
L1 Dcache	privada, 32KB, 8 vías, 64B-line, 2 ciclos
L2	privada, 256KB, 16 vías, 64B-line, 11 ciclos 16 MSHR
Red de interconexión	
Topología	Malla
Enrutamiento	X-Y
Tamaño del búfer de entrada/salida	144B
Ancho de banda de los enlaces	72B
Memoria principal	
Dispositivos DRAM	64 Meg x 8 x 8 bancos
Frecuencia del bus de memoria	1066MHz
Anchura del bus de memoria	8B/ciclo
Tamaño de la página/fila	4KB
Longitud del Burst (BL)	8
Canales	1
Controlador de memoria	prioridad FR-FCFS
Memoria total	4GB, 1 rango
Latencia	t_{RP}, t_{RCD}, t_{CL} 13.09ns cada uno

Cuadro 5.1: Configuración del controlador de memoria y la memoria principal.

viduales, cada benchmark ejecuta 500M instrucciones para *warm-up* el sistema, luego cada aplicación ejecuta al menos 300M de instrucciones.

Para evaluar las prestaciones, se mide el número de ciclos totales que necesita la ejecución de cada aplicación así como las instrucciones y se obtiene el IPC de cada ejecución.

El IPC no es el único dato de estudio importante, teniendo en cuenta el punto de vista de la memoria principal puede darse el caso de que una mejora en la latencia de acceso a memoria principal no ofrezca como resultado una mejora del IPC global. Por tanto, es importante distinguir las mejoras a nivel de aplicación y las mejoras a nivel de memoria principal.

Para estudiar las mejoras en memoria principal se usan estadísticas temporales

como el tiempo transcurrido desde que llega una petición al MC hasta que es servida y retorna al procesador. Pero éste es un tiempo muy general, por lo que es interesante descomponerlo en partes: tiempo de espera antes de acceder, tiempo de acceso, tiempo de transferencia de la información, tiempo de uso de la red, etc. Todos estos parámetros permiten conocer cuales son los puntos mejorados y los nuevos puntos débiles que se trasformarán en los nuevos cuellos de botella y con ello se puede aislar el punto donde centrar los esfuerzos de mejora.

Asimismo, teniendo el cuenta que el objetivo es dotar al sistema de un mayor paralelismo y aprovecharlo para reducir la latencia, es importante conocer si este mayor paralelismo se aprovecha correctamente. Para ello se usará el porcentaje de aciertos en el búfer fila, es decir, del total de peticiones que acceden al búfer fila, los cuales encontraron el bloque en la fila actual y no tuvieron que acceder al banco a por ella.

También es de sumo interés estudiar la energía consumida por el sistema. Para calcular este consumo se utiliza la hoja de cálculo Micron PowerCalculator [mic] la cual modela la potencia de una configuración de una DRAM a través de la entrada de parámetros de la DRAM por parte del usuario y los valores de utilización del sistema. Se ha combinado esta potencia estimada [1] junto con el tiempo de ejecución de las mezclas para estimar el consumo energético dividido en tres componentes:

- Energía de activación: cuenta la energía consumida activando las filas del banco para las posteriores lecturas y escrituras, la cual está directamente relacionada con el porcentaje de aciertos en el RB.

- Energía de *background*, precarga y refresco: se refiere a la energía consumida en *background* para mantener los dispositivos encendidos, además de la energía consumida debido a la precarga de las líneas de bits, y la energía requerida para evitar que lo condensadores pierdan el valor almacenado en el tiempo.

- Energía de lectura/escritura, conocida como Term. Esta energía es consumida cuando los datos están siendo transferidos por el bus de memoria en las operaciones de lectura y escritura.

[1] al contrario que el IPC y las peticiones de memoria que son recogidas cuando el benchmark finaliza 300 millones de instrucciones, el consumo energético se estima al final de la ejecución de la mezcla para mayor simplificación.

5.3. Estadísticas de evaluación

En esta sección se van a comentar las diferentes estadísticas que se han utilizado para calcular la eficiencia del sistema implementado. Estas estadísticas son extraídas desde dos puntos de vista: el sistema completo y el sistema de memoria principal.

5.3.1. Estadísticas del sistema en general

Las estadísticas incluidas en esta sección se encargan de resumir el comportamiento percibido desde una visión externa del sistema, es decir, evalúan como se comporta todo el sistema trabajando conjuntamente: red, núcleos, memoria, etc.

Dentro de este tipo de estadísticas, una de las más usadas y la que hemos elegido nosotros, es el IPC. Éste calcula las instrucciones que se han completado por ciclo de procesador, por lo tanto es interesante maximizar esta estadística.

Para calcular el IPC en cargas multiprogramadas, es decir, varias aplicaciones trabajando concurrentemente, se calcula la media harmónica de los IPC de las distintas aplicaciones. La media harmónica de una lista de valores tiende fuertemente a los valores más pequeños de la lista. Por lo tanto, comparada con la media aritmética, ésta tiende a mitigar el impacto de los IPC muy altos y agravar el impacto de los pequeños IPC. Este hecho puede utilizarse para reflejar que algunos benchmarks de la mezcla están obteniendo malos resultados, aunque la suma de los IPC haya aumentado como resultado de haber incrementado mucho un IPC que se ha beneficiado a costa de perjudicar a otras aplicaciones, e introduce una ruda estimación de la justicia en el acceso a los recursos en la evaluación del rendimiento.

$$WS_{hm} = \frac{n}{\sum_{i=1}^{n} \frac{1}{IS_i}} \tag{5.1}$$

5.3.2. Estadísticas del sistema de memoria principal

En este apartado se incluyen las estadísticas que evalúan el sistema desde un punto de vista interno, concretamente desde el punto de vista de la memoria principal, por lo tanto permiten evaluar la eficiencia de este subsistema.

Porcentaje de aciertos en el búfer fila. Porcentaje de peticiones que encuentran su bloque en la fila que está dentro del búfer fila, y por tanto no tiene que acceden

internamente al banco:

$$RBHR = \frac{Num.\ Aciertos\ RB}{Num.\ Peticiones} \tag{5.2}$$

Tiempo medio de espera en la cola del controlador. Se trata del tiempo medio que una petición está esperando a ser atendida y se encuentra dentro de la cola de espera de su banco. Este tiempo crecerá a medida que aumente el número de peticiones en la cola. Se calcula como la suma del tiempo que espera cada petición en las colas dividido el número de peticiones que han accedido al controlador:

$$T.\ Medio\ Espera = \frac{T.\ Espera\ Total}{Num.\ Peticiones} = \frac{\sum_{i=0}^{N-1} T.\ Espera(i)}{N} \tag{5.3}$$

Tiempo medio de acceso al banco. Este tiempo está relacionado con la latencia consumida al enviar los comandos de una petición al banco a través del bus de comandos y acceder a la línea del banco que contiene el bloque demandado. Este tiempo está relacionado directamente con el porcentaje de aciertos en el búfer fila (RBHR), es decir, si la fila está contenida o no en éste. En el caso de que haya muchos fallos en término medio, este tiempo se aproximará al tiempo que consume un fallo (40+40+40 ciclos) y si hay pocos se aproximará al tiempo de acierto (40 ciclos). A continuación se muestra su cálculo:

$$T.\ Medio\ Acceso = \frac{\sum_{i=0}^{N-1} T.\ Acceso(i)}{N} = RBHR\ *\ N \tag{5.4}$$

Tiempo medio de transferencia. Este tiempo computa el tiempo desde que una petición obtiene su bloque en el banco hasta que accede a la red para enviarlo a un nivel superior. Este tiempo depende del ancho de bus del canal y el tamaño del bloque.

$$T.\ Medio\ Transferencia = \frac{\sum_{i=0}^{N-1} \frac{Tamaño\ Bloque(i)}{Ancho\ Bus}}{N} \tag{5.5}$$

Tiempo de medio de acceso a memoria principal(MM). Este tiempo mide la latencia total desde que una petición entra en el controlador de memoria, hasta que sale de él. Por lo tanto, este tiempo incluye a todos los comentados anteriormente:

$$T.\ Medio\ Acceso\ MM = \frac{T.\ Espera\ Total + T.\ Acceso\ Total + T.\ Trans.\ Total}{N} \tag{5.6}$$

5.4. Benchmarks

Para los experimentos se ha usado una amplia gama de aplicaciones científicas (benchmarks) incluidas en el paquete SPEC CPU 2006 [Hen06].

Astar. Astar se deriva de una librería 2D de búsquedas de rutas que se utiliza en la IA del juego. Esta biblioteca implementa tres algoritmos de búsqueda de ruta diferentes: el primero es el bien conocido algoritmo A* para mapas con tipos de terrenos transitables y no transitables. El segundo, es una modificación del algoritmo de búsqueda de camino A* para la búsqueda de mapas con diferentes tipos de terreno y diferente velocidad de movimiento. El tercero es una implementación del algoritmo A* para los gráficos. Está formado por las regiones del mapa con relación de vecindad.

Bwaves. Bwaves simula numéricamente las ondas de choque en tres flujos viscosos de dimensiones transitorias transónicas.

La configuración inicial del problema de las ondas de choque consiste en una región de alta presión y densidad en el centro de una celda cúbica de una red periódica con baja presión y la densidad en otro lugar. Condiciones de contorno periódicas se aplican a la matriz de celdas cúbicas que forman una red infinita. Inicialmente, el volumen de alta presión comienza a expandirse en dirección radial como las ondas de choque clásicas. Al mismo tiempo, las ondas de expansión se mueven para llenar el vacío en el centro de la celda cúbica. Cuando el flujo alcanza la expansión de los límites, choca con las imágenes periódicas de otras células, creando así una estructura compleja de ondas de interferencia no lineal. Estos procesos crean un sistema periódico amortiguado no lineal con energía que está siendo disipada en el tiempo. Por último, el sistema llegará a un equilibrio.

Bzip2. Bzip2 se basa en la versión 1.0.3 de Julian Seward. La única diferencia entre bzip2 1.0.3 y el benchmark bzip2 es que la versión SPEC de bzip2 no actúa sobre ningún archivo de E/S que no sea la lectura de la entrada. Toda compresión y descompresión ocurre completamente en la memoria con el fin de ayudar a aislar el trabajo realizado solo a la CPU y a la memoria.

CactusADM. CactusADM es una combinación de Cactus, un problema de código abierto, y BenchADM, un representante del núcleo computacional de muchas aplicaciones en la relatividad numérica. CactusADM resuelve las ecuaciones de evolución de Einstein, que describen cómo las curvas espacio-tiempo son la respuesta a su contenido

de materia, y son un conjunto de diez ecuaciones diferenciales parciales no lineales acopladas.

DealII. DealII es un programa que utiliza deal.II, una biblioteca de programación en C++ cuyo objetivo son los elementos finitos adaptativos y la estimación de error. La biblioteca utiliza técnicas de programación del estado del arte del lenguaje de programación C++, incluyendo la biblioteca Boost.

El objetivo principal de deal.II es permitir el desarrollo de algoritmos de elementos finitos modernos, utilizando entre otros, sofisticados estimadores de error y mallas adaptativas. Escribir este tipo de programas es una tarea no trivial, y programas exitosos tienden a ser muy grandes y complejos.

Este benchmark resuelve una ecuación de tipo Helmholtz con coeficientes no constantes que está en el corazón de las soluciones para una amplia variedad de aplicaciones. El código utiliza métodos adaptativos modernos basados en la estimación de la dualidad de error ponderado para generar mallas óptimas.

Gamess. Una amplia gama de cálculos químicos cuánticos son posibles con GAMESS. Éste hace referencia a los siguientes cálculos: Cálculo del Campo autoconsistente (SCF) de la molécula de citosina mediante el método directo SCF. Cálculo SCF de agua y Cu2+ utilizando el método directo SCF. Cálculo SCF de triazolio iónico utilizando el método directo SCF.

Gcc. Gcc está basado en la versión 3.2 de gcc. Este benchmark genera código para un procesador AMD Opteron. El índice de referencia se ejecuta como un compilador con muchos de sus parámetros de optimización habilitados.

GemsFDT. GemsFDTD resuelve las ecuaciones de Maxwell en 3D en el dominio del tiempo utilizando el método de dominio de tiempo en diferencias finitas (FDTD). Se calcula la sección transversal del radar (RCS) de un objeto perfectamente conductor (PEC). El núcleo del método FDTD son las aproximaciones precisas de segundo orden a las leyes de Faraday y de Ampere.

Gobmk. El programa juega al Go y ejecuta un conjunto de comandos para analizar las posiciones del Go.

Gromacs. Gromacs se deriva de GROMACS, un paquete versátil que realiza la dinámica molecular, es decir, la simulación de las ecuaciones de Newton del movimiento para

sistemas con cientos de millones de partículas. Este benchmark lleva a cabo una simulación de la proteína lisozima en una solución de agua y iones. Mediante la simulación de los movimientos atómicos de estas estructuras se puede obtener una comprensión significativa de la dinámica de las proteínas y la función, y en algunos casos, incluso podría ser posible predecir la estructura de las nuevas proteínas.

H264ref. H264ref es una implementación de referencia de H.264/AVC (Advanced Video Coding), el último estándar de compresión de vídeo del estado del arte. El estándar ha sido desarrollado por la VCEG (Video Coding Experts Group) de la ITU (Unión Internacional de Telecomunicaciones) y MPEG (Moving Pictures Experts Group) de la ISO/IEC (Organización Internacional de Normalización). Esta norma sustituye a la norma MPEG-2 actualmente muy utilizado, y se está aplicando para aplicaciones tales como los DVD de próxima generación (Blu-ray y HD DVD) y la radiodifusión de vídeo.

Hmmer. Profile Hidden Markov Models son modelos estadísticos de múltiples alineamientos de secuencia, que se utilizan en la biología computacional para buscar patrones en las secuencias de ADN. La técnica se utiliza para hacer una búsqueda de bases de datos sensible, usando descripciones estadísticas de una secuencia de familia. Se utiliza para el análisis de secuencia de la proteína.

Lbm. Este programa implementa el denominado Método de Lattice Boltzmann (LBM) para simular los fluidos incompresibles en 3D. Es la parte computacionalmente más importante de un código que se utiliza en el campo de la ciencia de los materiales para simular el comportamiento de los fluidos con superficies libres, en particular la formación y el movimiento de burbujas de gas en el metal.

Leslie3d. Leslie3d se deriva de LESlie3d (Large-Eddy Simulations with Linear-Eddy Model in 3D). Es la solución primaria utilizada para investigar una amplia variedad de fenómenos de turbulencia tales como mezclado, la combustión, la acústica y la mecánica de fluidos generales.

Para CPU2006, el programa se ha creado una para resolver un problema de prueba que representa un subconjunto de estos flujos. Este tipo de flujo se produce en las regiones de mezcla de todas las cámaras de combustión que emplean inyección de combustible.

LESlie3d utiliza un algoritmo fuertemente conservador, de volúmenes finitos con el esquema de integración temporal Predictor-Corrector MacCormack.

Libquantum. Libquantum es una biblioteca para la simulación de un ordenador cuántico. Los ordenadores cuánticos se basan en los principios de la mecánica cuántica y pueden resolver ciertas tareas computacionalmente difíciles en tiempo polinomial.

Libquantum proporciona una estructura para la representación de un registro cuántico y algunas puertas elementales. Además, libquantum ofrece la simulación de decoherencia, el obstáculo más importante en la construcción de ordenadores cuánticos prácticos. Por lo tanto, no solo es posible simular cualquier algoritmo cuántico, sino también desarrollar algoritmos de corrección de errores cuánticos.

Mcf. Mcf es un benchmark que se deriva de MCF, un programa de programación de vehículos en el transporte público de masas. El programa está escrito en C. La versión de referencia utiliza casi exclusivamente aritmética de enteros.

El programa está diseñado para la solución de los problemas sobre un único depósito de programación de vehículos que se producen en el proceso de planificación de las empresas de transporte público.

Milc. El Código MILC es un conjunto de códigos escritos en C desarrollado por MILC para hacer simulaciones de cuatro dimensiones en máquinas paralelas MIMD. Milc en Spec CPU2006 utiliza la versión en serie del programa su3imp. La versión de un solo procesador de esta aplicación es importante y relevante, ya que el rendimiento paralelo depende de un buen rendimiento solo procesador.

Namd. Namd se deriva de la disposición de los datos y el bucle interno de NAMD, un programa paralelo para la simulación de sistemas biomoleculares grandes.

Omnetpp. Este benchmark realiza la simulación de eventos discretos de una red Ethernet de gran tamaño. La simulación se basa en la OMNeT++ sistema de simulación de eventos discretos, un marco genérico de simulación y abierto. El área de aplicación principal de OMNeT++ es la simulación de redes de comunicación, pero su arquitectura genérica y flexible permite su uso en otras áreas tales como la simulación de sistemas de colas, redes, arquitecturas de hardware o procesos de negocios.

Perlbench. Perlbench es una versión reducida de Perl v5.8.7, el lenguaje de programación. La versión de SPEC de Perl ha omitido la mayor parte de las características específicas del sistema operativo.

Povray. POV-Ray es un trazador de rayos. El trazado de rayos es una técnica de representación que calcula una imagen de una escena simulando la forma en que los rayos de luz viajan en el mundo real, pero lo hace al revés. En el mundo real, los rayos de luz son emitidos por una fuente de luz e iluminan los objetos. La luz se refleja en los objetos o pasa a través de los objetos transparentes. Esta luz reflejada golpea el ojo humano o un lente de la cámara. Como la gran mayoría de los rayos nunca golpea a un observador, tomaría una eternidad trazar una escena. Por lo tanto, ray-trazadores como POV-Ray comienzan con su cámara simulada y trazan los rayos hacia atrás en la escena. El usuario especifica la ubicación de la cámara, fuentes de luz, y los objetos, así como las texturas de la superficie y de los interiores.

Para cada píxel del rayo es disparado desde la cámara a la escena para ver si se cruza con cualquiera de los objetos en la escena. Se calcula cada vez que un objeto es golpeado, el color de la superficie en ese punto. Con este fin los rayos se envían a cada fuente de luz para determinar la cantidad de luz que viene de él o si el objeto está en la sombra.

Soplex. Soplex se basa en SoPlex Versión 1.2.1., resuelve un programa lineal utilizando el algoritmo Simplex.

El LP se administra como una $m \times n$matriz A, junto con un vector b de dimensión m, y un vector coeficiente c que es la función objetivo, de dimensión n. En general, el problema es encontrar el vector x para: minimizar c'x sujeto a Ax <= b con x>= 0.

En la práctica, x puede tener límites superiores y las limitaciones de A (i,.) X <= b (i) podrían ser restricciones mayores-que o iguales a, o restricciones de igualdad.

SoPlex, como la mayoría de las implementaciones del algoritmo simplex, emplea algoritmos de álgebra lineal dispersa, en particular, una escasa LU-factorización y las rutinas adecuadas para resolver los sistemas de ecuaciones triangulares resultantes.

Sphinx3. Sphinx-3 es un sistema de reconocimiento de voz muy conocido en la Universidad Carnegie Mellon.

Wrf. WRF se basa en el modelo de Investigación del tiempo y Prospectiva, que es un sistema de predicción numérica del tiempo destinado a servir a la predicción operativa y las necesidades de investigación atmosférica. WRF es adecuado para un amplio espectro de aplicaciones a través de escalas que van de metros a miles de kilómetros.

Xalancbmk. Este programa es una versión modificada de Xalan-C++, un procesador XSLT escrito en un subconjunto portátil de C++. Xalan-C++ versión 1.8 es una aplicación robusta de las Recomendaciones W3C para XSL Transformations (XSLT) y el Lenguaje de rutas XML (XPath). Se utiliza el lenguaje XSLT para redactar hojas de estilo XSL. Una hoja de estilo XSL contiene instrucciones para la transformación de documentos XML a partir de un tipo de documento a otro tipo de documento (XML, HTML, u otros). En términos estructurales, una hoja de estilo XSL especifica la transformación de un árbol de nodos (la entrada XML) en otro árbol de nodos (el resultado de la salida o transformación).

Zeusmp. Zeusmp se basa en ZEUS-MP, un código de dinámico de fluidos computacionales desarrollado en el Laboratorio de Astrofísica Computacional (NCSA de la Universidad de Illinois) para la simulación de fenómenos astrofísicos. ZEUS-MP resuelve problemas en tres dimensiones espaciales con una amplia variedad de condiciones de contorno.

El programa resuelve las ecuaciones de ideales (sin resistencia), no relativistas, hidrodinámica y magnetohidrodinámica, incluyendo campos gravitacionales aplicados externamente y auto-gravedad. El gas puede ser adiabático o isotérmico, y la presión térmica es isotrópica. Las condiciones de contorno se pueden especificar como un reflejo, periódico, entrada o salida.

El problema resuelto en ESPEC CPU2006 es un 3-D Blastwave simulado con la presencia de un campo magnético uniforme a lo largo de la dirección x.

Mezcla	Benchmark (categoría)			
m1	hmmer(3)	mcf(4)	soplex(3)	sphinx3(3)
m2	gobmk(1)	sphinx3(3)	wrf(1)	zeusmp(1)
m3	gromacs(1)	hmmer(3)	povray(4)	soplex(3)
m4	GemsFDTD(2)	gromacs(1)	sphinx3(3)	wrf(1)
m5	GemsFDTD(2)	leslie3d(1)	libquantum(3)	zeusmp(1)
m6	gromacs(1)	povray(4)	sphinx3(3)	zeusmp(1)
m7	astar(4)	cactusADM(1)	hmmer(3)	povray(4)
m8	cactusADM(1)	libquantum(3)	gobmk(1)	wrf(1)
m9	sphinx3(3)	zeusmp(1)	soplex(3)	povray(4)

Cuadro 5.2: Mezclas de 4 núcleos.

5.5. Mezclas

El diseño de las pruebas multinúcleo se ha basado en la caracterización presentada en el apartado 1.3. Se ha diseñado un conjunto de mezclas de 4 núcleos usando una malla 2x2 para estudiar el comportamiento de las propuestas en diferentes escenarios. La Tabla 5.2 muestra las mezclas de los 4 núcleos. Cada mezcla contiene aplicaciones de diferentes categorías (la categoría de cada benchmark está entre paréntesis) para analizar las internaciones entre diferentes tipos en el acceso a memoria principal.

Capítulo 6

Evaluación experimental

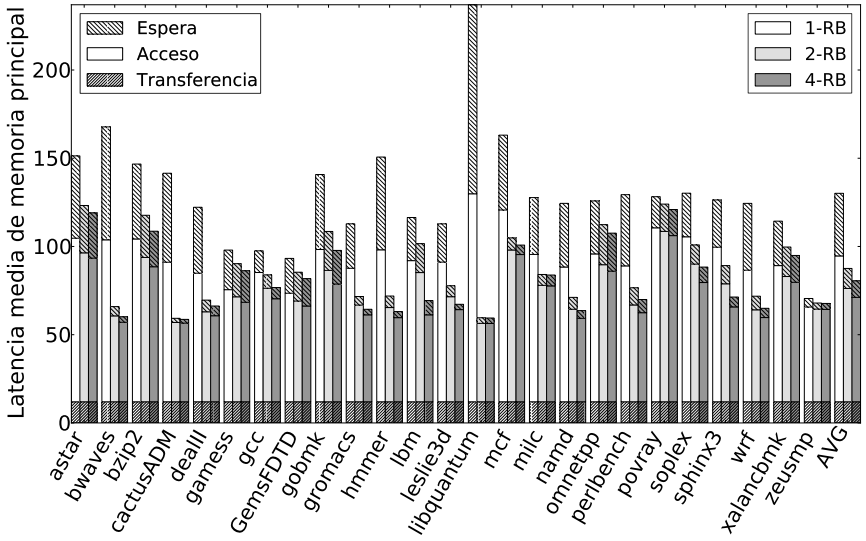
En este capítulo se evalúa y se compara el rendimiento de los dos esquemas de tablas fila propuestos con un modelo del estado del arte. Las propuestas estudiadas consideran 1 y 2 entradas por núcleo en cada una de las tablas fila del sistema, independientemente si las entradas son consideradas buffers fila o sub-buffers fila. Por consiguiente, el esquema $\text{CRT}_{1/x}$ está en clara desventaja con respecto al área, es decir, tiene un menor tamaño de almacenamiento. Sin embargo, a pesar de este hecho, los resultados muestran que bajo ciertas circunstancias se pueden alcanzar beneficios en el rendimiento. Esta asunción nos lleva a no comparar resultados de rendimiento contra esquemas del estado del arte basados en sub-filas, ya que estos claramente serían peores que los esquemas basados en filas cuando trabajan con el mismo número de entradas.

Por lo tanto, por propósitos de comparación, se ha considerado un modelo del estado del arte [HGCT13], conocido como RBC (row buffers per core). Este esquema asigna un número fijo de filas o entradas de cada tabla a cada hilo. Este número de filas se estableció a 1 en su trabajo original. Sin embargo, para realizar una comparación justa, este esquema ha sido modificado para asignar múltiples entradas de las tablas filas (por ejemplo 1 y 2) para cada núcleo. De esta forma, el número total de entradas en las tablas fila es igual al de los esquemas propuestos en este trabajo. Esta propuesta principalmente se diferencia de BRT en que BRT no preasigna buffers a los núcleos, sino que las entradas se asignan dinámicamente a los núcleos de acuerdo con las necesidades de cada hilo en tiempo de ejecución. Esta política sigue la observación discutida en el apartado 1.3 que afirma que cada benchmark requiere un número diferente de RBs para lograr su mejor rendimiento.

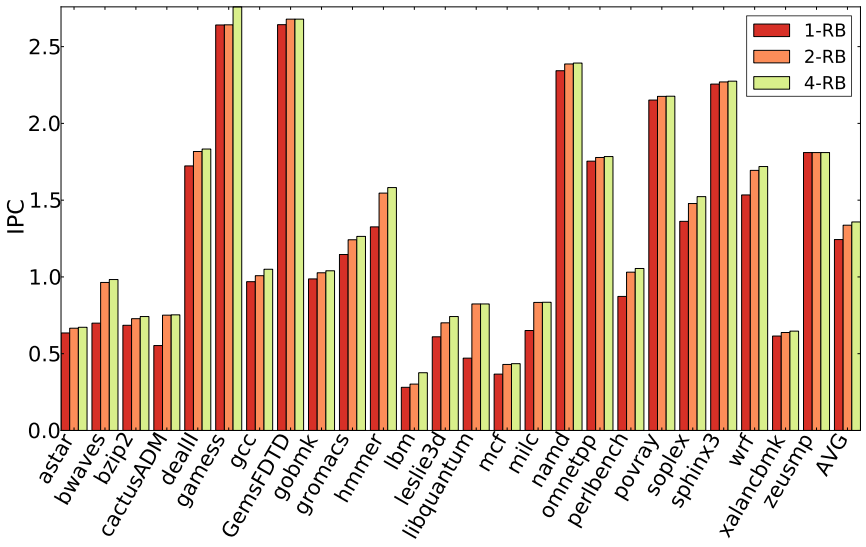
6.1. Efecto del tamaño de las tablas fila en aplicaciones individuales

En el apartado 1.3 se analizó el rendimiento del búfer fila y se probo que incrementar el número de buffers ayuda a mejorar el porcentaje de aciertos en los RBs o tablas fila. Este apartado analiza este efecto en el rendimiento (IPC) de aplicaciones individuales bajo el esquema BRT.

Como se muestra en la Figura 6.1(a), el incremento del porcentaje de aciertos en las tablas fila se convierte en un ahorro de latencia de memoria. En término medio, la



(a) Latencia



(b) IPC

Figura 6.1: Efecto del tamaño de las tablas de filas en aplicaciones individuales

latencia de memoria se reduce un 32 % cuando se pasa de 1 a 2 entradas por tabla, y por consiguiente por banco, y un 37 % cuando se pasa a 8 entradas. Para entender mejor de donde vienen estos beneficios, cada barra de cada aplicación se divide en tres segmentos de acuerdo a los tres componentes de la latencia de memoria (ver apartado 5.3).

- *Tiempo de espera.* Representa el tiempo medio que permanece una petición en las colas del MC debido a conflictos para acceder al banco o canal, es decir, el tiempo que esperan las peticiones para poder servirse debido a que requieren un banco o canal que esta siendo usado por una petición previa.

- *Tiempo de acceso.* Tiempo real transcurrido accediendo al banco de memoria y trasladando la fila objetivo a un búfer fila.

- *Tiempo de transferencia.* Cuenta el tiempo requerido para transferir los datos (bloque) desde los módulos de memoria al MC.

BRT reduce de forma significativa el tiempo de acceso a medida que aumenta el número de entradas en las tablas ya que se incrementa el número de aciertos en estas tablas, los cuales son servidos de forma más rápida ya que no necesitan acceder al módulo de memoria. Como era de esperar, esta reducción es más aparente en aplicaciones de las categorías 1 y 3. Por ejemplo, en *lbm* al pasar de 1 entrada a 8, la latencia se reduce un 41 %.

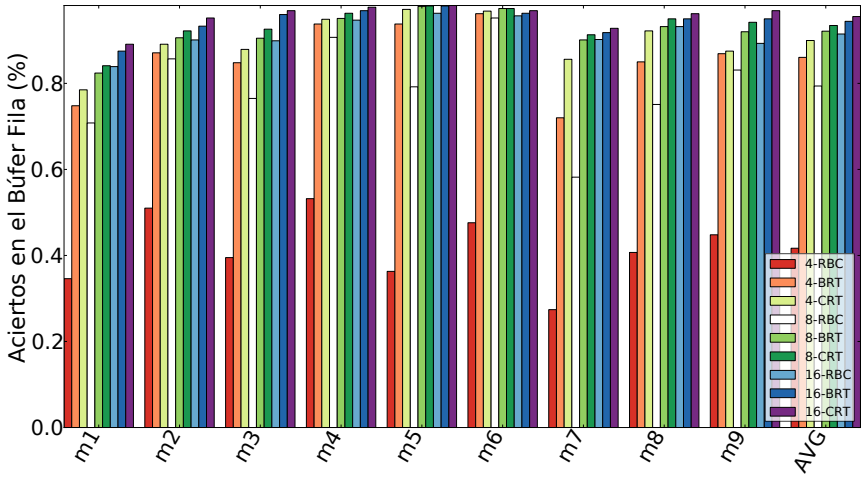
Estos ahorros de latencia se convierten en mejoras del IPC como muestran los resultados de la Figura 6.1(b). Como se observa, el IPC aumenta en todas las aplicaciones a medida que aumenta número de entradas de las tablas fila. En término medio, el IPC se incrementa un 7.5 % con 2 entradas y un 9.2 % con 4 entradas. Por otro lado, algunas aplicaciones como `bwaves` y `libquantum` usando 4 buffers se pueden mejorar un 40 % y 70 % respectivamente. Incluso aplicaciones de categorías que no son sensibles el número de buffers como `mcf` muestran un incremento del IPC del 18 %.

6.2. Evaluación de las propuestas en cargas multiprogramadas

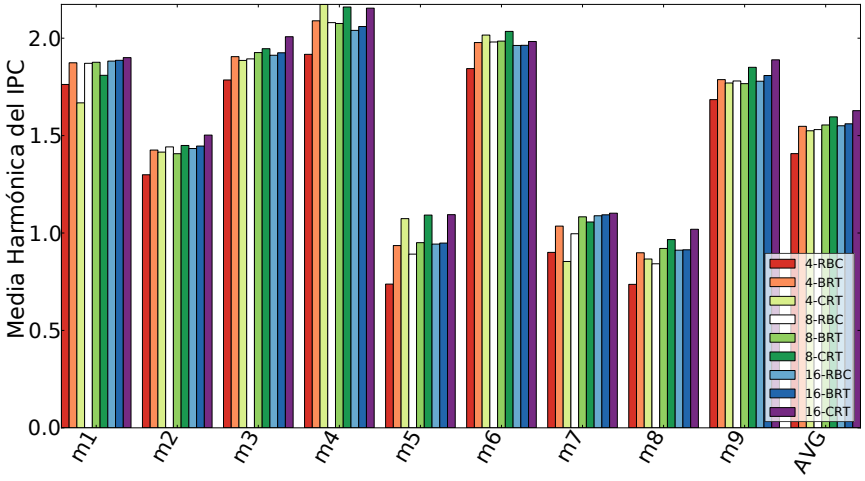
En este apartado se evalúan los esquemas estudiados en CMPs de 4 núcleos. Las cargas utilizadas para evaluar las propuestas se pueden consultar en el apartado 5.5.

La Figura 6.2(a) muestra el porcentaje de aciertos en las tablas fila de los diferentes modelos variando en número de entradas por tabla (banco) de 4 a 16. Éste ese el tamaño máximo que se asume que son 4 entradas por aplicación y por tabla. Como se puede ver, el porcentaje de aciertos en las tablas del esquema RBC para tablas con 4 entradas es más del doble peor que el de las propuestas diseñadas.

Por otro lado, estos esquemas con 4 entradas por tabla mejoran el rendimiento sobre RBC con 8 entradas. Cabe señalar que esta observación se mantiene verdadera cuando se pasa de 8 a 16 entradas. En otras palabras, CRT y BRT con la mitad de entradas en las tablas presentan un mejor porcentaje de aciertos en las tablas que RBC. En término medio, BRT y CRT con 4 entradas mejoran RBC con 8 entradas sobre un 8.4 % y 13.3 %, respectivamente. Esto corrobora que el alojamiento dinámico de las entradas de las tablas (como se ha hecho en los esquemas propuestos) juega



(a) Porcentaje de aciertos en las tablas



(b) Media harmónica del IPC

Figura 6.2: Resultados de las propuestas variando el número de entradas.

un rol importante en el rendimiento. Los esquemas propuestos muestran un similar porcentaje de aciertos en las tablas, aunque de media CRT presenta mejores valores.

La media harmónica del IPC de las mezclas estudiadas se muestra en la Figura 6.2(b), ésta cuantifica cuan justa es la obtención de recursos y el rendimiento. Como se puede observar, el modelo BRT tiene una repartición de los recursos más justa que BRC ya que las entradas de las tablas se asignan dinámicamente de acuerdo con los requerimientos de las aplicaciones, y CRT es el esquema más justo. En cuanto al rendimiento, a primera vista, se puede apreciar cierta relación entre el porcentaje de aciertos en las tablas y el IPC obtenido. En el caso de la mezcla m5 con 4 entradas por tabla, CRT mejora BRT con un 15 % y RBC con un 45 %, mientras que en la mezcla m8, BRT mejora RBC y CRT en un 22 % y 3.7 %, respectivamente.

Sin embargo, es importante darse cuenta que la planificación FR-FCFS reordena el acceso de las peticiones de memoria en el controlador, es decir, la peticiones pueden servirse en orden diferente al que han llegado al controlador de memoria. Esto significa que aún teniendo un gran número de entradas no siempre se puede garantizar que el IPC mejorará, incluso si su porcentaje de aciertos en las tablas mejora. La principal causa de este problema que la peticiones críticas que no son aciertos deben esperar grandes cantidades a ser servidas.

Por otro lado, los esquemas estudiados no trabajan igual; por ejemplo, el esquema CRT transfiere una página entera de memoria (fila) cada vez que una nueva página/fila es accedida, por lo que el tiempo de transferencia puede impactar significativamente en el tiempo de espera (el tiempo media que espera una petición de memoria en el controlador para acceder a un determinado módulo de memoria) de este esquema, lo cual puede reducir el rendimiento. La Figura 6.3[1] muestra la cantidad de datos transferidos desde/hacia controlador de memoria para cada modelo, la cual es proporcional al tiempo en que el bus de datos de memoria está ocupado.

Por lo tanto, los valores del IPC deben ser analizados considerando tanto los aciertos en las tablas como el tiempo de transferencia. Por ejemplo, en la mezcla m1 se puede observar que el porcentaje de aciertos en las tablas de CRT con 4 entradas es mucho mejor que el de RBC con 4 entradas, sin embargo, el IPC es peor. Este empeoramiento se puede explicar mirando los bytes transferidos, donde CRT tiene valores 10× más grandes. Por otro lado, CRT puede alcanzar un mejor rendimiento que RBC en aquellas mezclas que exhiben una gran localidad espacial (por ejemplo, la mezcla m5). Para un gran número de entradas, el comportamiento de CRT es mejor que RBC ya que mejora

[1] los valores para 8 y 16 entradas en las tablas del BRC y BRT no se han incluido ya que sus valores son similares a los de 4 entradas por tabla, ya que el número de bytes transferidos es el mismo.

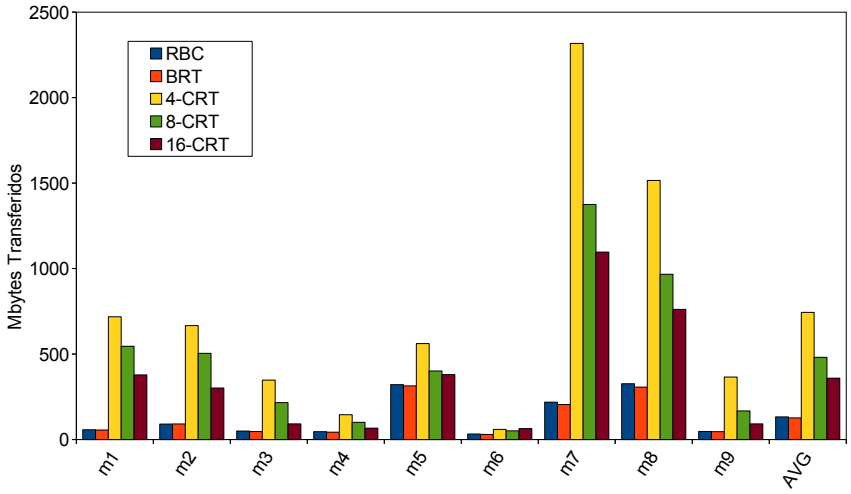


Figura 6.3: Mbytes transferidos desde la DRAM al MC con RBC, BRT y CRT.

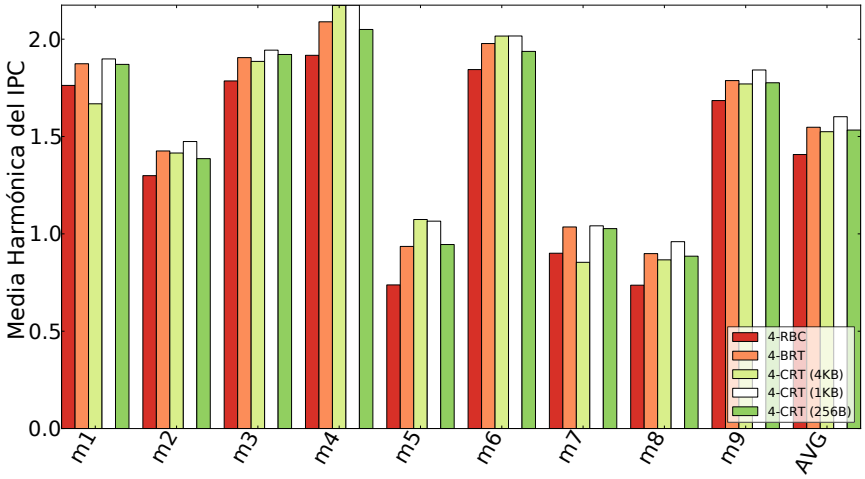
la media harmónica del IPC aumenta un 5 % mientras que BRT apenas mejora el IPC con 16 entradas. Este hecho se puede explicar por el incremento del número de aciertos en CRT como se observa en la Figura 6.2(a).

6.3. Sensibilidad de CRT respecto al tamaño de transferencia

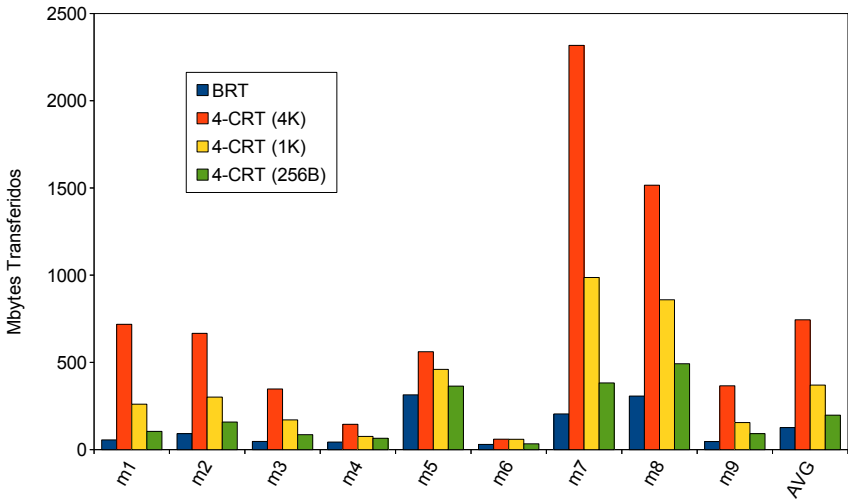
Como se ha demostrado en el apartado anterior, transferir la fila completa desde el banco al MC en BRT puede incrementar de forma notoria los tiempos de espera, lo cual se convierte en importantes pérdidas en el rendimiento y malgaste energético (ver apartado 6.4) debido a transferencias innecesarias, es decir, a transferir filas a las tablas que no son usadas en accesos posteriores.

Para solucionar este problema, este apartado evalúa las variantes $\text{CRT}_{1/4}$ y $\text{CRT}_{1/16}$, las cuales consideran un cuarto (1 KB) y un sexto (256B) de la página o fila. Las variantes CRT trabajan con el mismo número de entradas (sub-páginas) que el número de entradas del CRT original.

La Figura 6.4(a) muestra la media harmónica del IPC para CRT (4KB) y las variantes CRT (1KB y 256B) variando el número de entradas de las tablas. Como se observa, transferir solo una fracción del tamaño de la fila soluciona el problema mencionado, gracias a reducir el tiempo de transferencia como se muestra en la Figura 6.4(b). Por ejemplo, esto se puede apreciar en la mezcla m7 comparando CRT con sus variantes, donde el IPC de $\text{CRT}_{1/4}$ y $\text{CRT}_{1/16}$ aumenta un 22 % y 20 %, respectivamente, debido a la reducción de los bytes transferidos lo cual es un 57 % y 83 %, respectivamente. En



(a) Media harmónica del IPC



(b) Mbytes transferidos desde la DRAM al MC

Figura 6.4: Resultados de la variante $CRT_{1/x}$ con valores de x 1, 4 y 16

general, comparado con CRT, la variante $CRT_{1/4}$ incrementa el IPC un 5 % mientras reduce la cantidad de bytes transferidos en un 50 %, y $CRT_{1/16}$ tiene un IPC similar al CRT original pero reduce los bytes transferidos un 73 %.

Una observación importante es que de media $CRT_{1/16}$ logra una cantidad de bytes transferidos similar a BRT. Por lo tanto, se concluye que 1/16 es la fracción más óptima.

6.4. Consumo energético

En este apartado se analiza el consumo energético de la memoria principal para los esquemas estudiados. Para ello se ha usado la hoja de cálculo Micron Power Energy [mic], que obtiene la potencia de las diferentes partes del sistema de memoria. Se ha

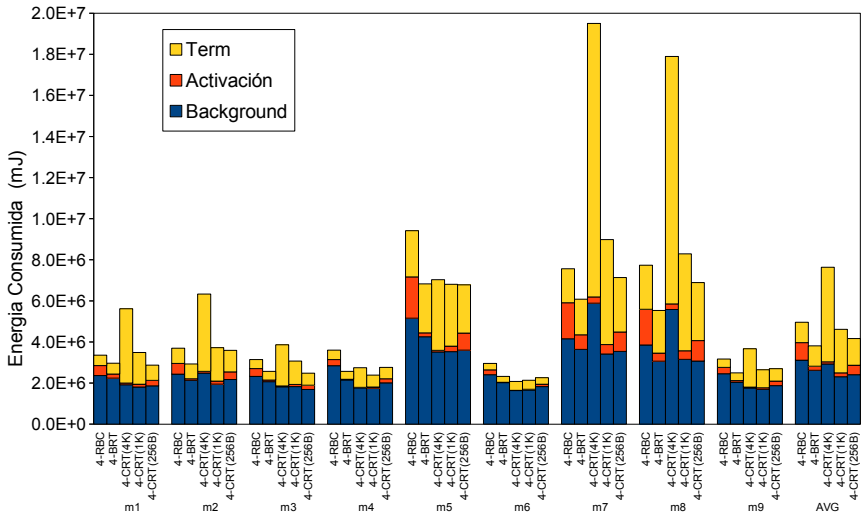


Figura 6.5: Energía consumida por la propuestas analizadas.

combinado esta potencia estimada junto con el tiempo de ejecución de las mezclas para estimar el consumo energético.

La Figura 6.5 presenta los resultados de energía divididos en tres componentes dependiendo de la actividad de memoria que consume la energía:

- Energía de Activación.
- Energía de *Background*, Precarga y Refresco.
- Energía de Term (lectura/escritura).

Como se puede observar, los esquemas RBC y BRT tienen un mayor consumo de energía de *Activación* que el original CRT, principalmente debido a que éstos trabajan en modo de página abierta, manteniendo el contenido de la fila en el búfer, y los transistores deben permanecer trabajando para mantener la información disponible para las futuras peticiones.

Como era de esperar de la Figura 6.4(b), la energía de *Term* en CRT es la mayor, porque este modelo es la que mayor cantidad de bytes transfiere. Mientras que BRT y BRC solo envían los bloques pedidos, CRT siempre transfiere la fila entera. Cabe destacar que las variantes de CRT, transfieren una fracción de la fila que reduce significativamente la energía de *Term*. En consecuencia, habrán más fallos en las tablas, lo que incrementa la energía de *Activación* ya que los bancos tienen que abrir sus filas para enviar la fracción de la fila correspondiente.

En las mezclas donde la energía de CRT es relativamente baja comparada con RBC y BRT, $\text{CRT}_{1/4}$ y $\text{CRT}_{1/16}$ tiene resultados similares a la CRT original. Por ejemplo,

en la mezcla m5 la energía consumida permanece igual para todas los esquemas CRT porque la energía de *Term* se reduce en sus variantes, pero la energía de *Activación* se incrementa en las variantes CRT debido a un peor porcentaje de aciertos en las tablas como resultado de tener una área más pequeña (el mismo número de entradas pero de menor tamaño). Por el contrario, cuando la energía de CRT es alta (lo que significa que hay una alto porcentaje de fallos en las tablas), las diferencias entre enviar la fila entera o una fracción están más acentuadas porque, aunque el número de transferencias se incrementan, estas cuestan menos; por ejemplo, en la mezcla m8 se reduce su energía consumida más de la mitad.

En término medio, BRT reduce el consumo energético un 23 % con respecto a RBC, ya que BRT tiene un mayor porcentaje de aciertos en las tablas fila, lo cual reduce la energía de *Activación*. Las variantes $\text{CRT}_{1/4}$ y $\text{CRT}_{1/16}$ reducen la energía del CRT original un 40 % y 45 %, respectivamente, principalmente debido a que se ha reducido la energía de *Term*.

Capítulo 7

Conclusiones y trabajo futuro

7.1. Conclusiones

Los procesadores multinúcleo se encuentran en la mayoría de los productos electrónicos de uso cotidiano, que cada día demandan más funcionalidades. Esto significa un mayor número de núcleos por lo que es necesario centrarse en el diseño de técnicas que lo permitan.

Este trabajo ha mostrado que el rendimiento del búfer fila de un conjunto de aplicaciones individuales puede mejorar significativamente (por ejemplo, más del doble) añadiendo buffers adicionales en los módulos de memoria. Esta observación tiene un especial interés en sistemas multinúcleo con cargas multiprogramadas, porque los módulos de memoria actuales implementan una única fila que almacena la última página leída por la última aplicación que accede a memoria principal.

Basándose en esta observación y su impacto en el rendimiento, este trabajo ha presentado el concepto de *tablas fila* como un conjunto de buffers fila compartidos por todas las aplicaciones que compiten por el acceso a un banco de la DRAM. Las entradas de las tablas son asignadas dinámicamente a las aplicaciones basándose en sus requerimientos de memoria. Se han diseñado y evaluado principalmente dos alternativas, BRT y CRT, las cuales implementan las tablas en el módulo de memoria y en el controlador de memoria, respectivamente.

Comparadas con un esquema del estado del arte, ambos diseños duplican el porcentaje de aciertos en el RB y mejoran de media (media harmónica) el IPC un 6 %. La implementación de las tablas en el MC puede conllevar beneficios en el rendimiento tan altos como un 30 % en algunas mezclas. Asimismo, el acceso a los buffers fila y los recursos de la memoria principal es más justo, es decir, los núcleos se reparten los recursos según su demanda pero todos tienen la oportunidad de ser servidos ya que

se mantienen recursos libres. Un ejemplo de este comportamiento sería si una mezcla tuviese varias aplicaciones que accediesen mucho a memoria principal y una aplicación con poco acceso; si no se mantiene una política de acceso justo, las aplicaciones con mayor número de accesos ocuparían todos los recursos y cuando la aplicación con pocos accesos quisiese acceder tendría que esperar una gran cantidad de tiempo.

Un beneficio interesante de CRT es que puede trabajar con los módulos de memoria actuales donde solamente hay un búfer fila en cada banco. Con lo que se pueden conseguir beneficios en el rendimiento con unos cambios mínimos en la lógica del controlador de memoria en los procesadores multinúcleo actuales. El mayor problema de CRT es la gran cantidad de bytes transferidos con respecto a BRT en las aplicaciones con bajo porcentaje de aciertos en las tablas. Esto no solo incrementa el tiempo de espera para conseguir el bus de datos sino que es mucho más costoso en términos de memoria, afectando a la energía relacionada en la transferencia de datos, y aportando escasos beneficios en cuanto a la localidad se refiere debido a que la mayoría de datos no se pedirán más tarde. La variante $\mathrm{CRT}_{1/X}$ soluciona este problema aplicando técnicas de sub-filas para reducir el tiempo de transferencia.

Además del rendimiento, las aproximaciones propuestas reducen la energía consumida ya que un mayor porcentaje de aciertos en las tablas significa que menos filas son leídas. Se ha evaluado la variante $\mathrm{CRT}_{1/x}$ con el mismo número de entradas (sub-filas) que la aproximación CRT original, de forma que los beneficios en el rendimiento provienen de evitar tráfico inútil (el tiempo de espera para conseguir el bus de datos se reduce) ya que esta aproximación tiene tiene 1/x veces menos área de almacenamiento. Los resultados experimentales muestran que para un sistema de 4 núcleos, por término medio, los mecanismos BRT y $\mathrm{CRT}_{1/x}$ ahorran energía un 23 % y 7 %-16 % (dependiendo del valor de la X) y mejoran el IPC un 10 % y 9 %-14 %, respectivamente.

7.2. Trabajo futuro

Un futuro trabajo incluirá un estudio de las aproximaciones combinadas con mecanismos de prebúsqueda lo cual se espera que obtenga mejores resultados debido a que la prebúsqueda incrementa la localidad espacial.

Por otro lado, se deja como trabajo futuro explorar una alternativa que elija entre activar BRT o CRT dinámicamente en tiempo de ejecución, de forma que en algunos momentos de la ejecución es mejor tener activado BRT, cuando hay pocos aciertos en las tablas, y en otras CRT. Asimismo se puede aplicar este esquema adaptativo para elegir el tamaño de la fila que se transfiere en cada momento de la ejecución.

Debido a la importancia del planificador que elige que petición servir en cada momento, resulta interesante explorar nuevas políticas, que teniendo en cuenta las características del nuevo sistema, aceleren el acceso a memoria principal para las peticiones críticas.

7.3. Publicaciones relacionadas con el proyecto

Entre las publicaciones relacionadas con el proyecto cabe destacar:

P. Navarro, V. Selfa, C. Gómez, M. Gómez, J. Sahuquillo, "Row Tables: Design Choices to Exploit Bank Locality in Multiprogram Workloads", *23rd Annual Euromicro International Conference on Parallel, Distributed and Network-based Processing* (PDP), Turku, Finland, March 4-7, 2015, Submitted.

V. Selfa, P. Navarro, C. Gómez, M. Gómez, J. Sahuquillo, "Methodologies and performance metrics to evaluate multiprogram workloads", *23rd Annual Euromicro International Conference on Parallel, Distributed and Network-based Processing* (PDP), Turku, Finland, March 4-7, 2015, Submitted.

V. Selfa, P. Navarro, C. Gómez, M. Gómez, J. Sahuquillo, "Diseño de mecanismos de prebúsqueda adaptativa bajo gestión eficiente de memoria para procesadores multinúcleo", In *XXIV Jornadas de Paralelismo* (JP2013), pages 43-48, Madrid, Spain, 2013.

Bibliografía

[ALSJ09] Jung-Ho Ahn, J. Leverich, R.S. Schreiber, and N.P. Jouppi. Multicore dimm: an energy efficient memory module with independently controlled drams. *Computer Architecture Letters*, 8(1):5–8, Jan 2009. 26

[ea03] M. M. K. Martin et al. *Protocol Specifications and Tables for Four Comparable MOESI Coherence Protocols: Token Coherence, Snooping, Directory, and Hammer*. http://www.cs.wisc.edu/, 2003. 16

[ELMP11] Eiman Ebrahimi, Chang Joo Lee, Onur Mutlu, and Yale N. Patt. Prefetch-aware shared resource management for multi-core systems. In *Proceedings of the annual international symposium on Computer architecture*, 2011. 25

[GMMG12] Nagendra Dwarakanath Gulur, R. Manikantan, Mahesh Mehendale, and R. Govindarajan. Multiple sub-row buffers in dram: Unlocking performance and energy improvement opportunities. In *Proceedings of the 26th ACM International Conference on Supercomputing*, ICS '12, pages 257–266, New York, NY, USA, 2012. ACM. 10, 26

[Hen06] John L. Henning. Spec cpu2006 benchmark descriptions. *SIGARCH Comput. Archit. News*, 34(4):1–17, September 2006. 39

[HGCT13] Enric Herrero, Jose Gonzalez, Ramon Canal, and Dean Tullsen. Thread row buffers: Improving memory performance isolation and throughput in multiprogrammed environments. *IEEE Trans. Comput.*, 62(9), 2013. 10, 12, 26, 46

[HP12] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5 edition, Appendix E by T. Conte, 2012. 21

[JK12] J. Jeddeloh and B. Keeth. Hybrid memory cube new dram architecture increases density and performance. In *Symp. on VLSI Technology*, 2012. 23

[JNW07] Bruce Jacob, Spencer Ng, and David Wang. *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007. 14, 25

[JRLCV10] Vijay Janapa Reddi, Benjamin C. Lee, Trishul Chilimbi, and Kushagra Vaid. Web search using mobile cores: quantifying and mitigating the price of efficiency. In *Proceedings of the 37th annual international symposium on Computer architecture*, ISCA '10, pages 314–325, New York, NY, USA, 2010. ACM. 9

[KPMHB10] Yoongu Kim, Michael Papamichael, Onur Mutlu, and Mor Harchol-Balter. Thread cluster memory scheduling: Exploiting differences in memory access behavior. In *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture*, 2010. 25

[KSSF10] Ron Kalla, Balaram Sinharoy, William J. Starke, and Michael Floyd. Power7: IBM's Next-Generation Server Processor. *IEEE Micro*, 30:7–15, 2010. 19

[LIMB09] Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger. Architecting phase change memory as a scalable dram alternative. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, pages 2–13, New York, NY, USA, 2009. ACM. 26

[LMNP11] Chang Joo Lee, Onur Mutlu, Veynu Narasiman, and Yale N. Patt. Prefetch-aware memory controllers. *IEEE Transactions on Computers*, 60(10), 2011. 10, 25

[MH08] Michael R. Marty and Mark D. Hill. Virtual hierarchies. *IEEE Micro*, 28(1):99–109, 2008. 18

[mic] Micron technology inc. tn-41-01: calculating memory system power for ddr3. `http://download.micron.com`. 36, 52

[mic11] Micron technology, 4gb: x4, x8, x16 ddr3 sdram, 2011. 34

[MLM12] Justin Meza, Jing Li, and Onur Mutlu. A case for small row buffers in non-volatile main memories. In *ICCD*, pages 484–485. IEEE Computer Society, 2012. 26

[MM07] Onur Mutlu and Thomas Moscibroda. Stall-time fair memory access scheduling for chip multiprocessors. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 146–160, 2007. 25

[MM08] Onur Mutlu and Thomas Moscibroda. Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared dram systems. In *Proceedings of the Annual International Symposium on Computer Architecture*, 2008. 25, 26

[MS05] Richard E. Matick and Stanley E. Schuster. Logic-based eDRAM: Origins and rationale for use. *IBM Journal of Research and Development*, 49(1):145–165, January 2005. 18

[RDK+00] Scott Rixner, William J. Dally, Ujval J. Kapasi, Peter Mattson, and John D. Owens. Memory access scheduling. In *Proceedings of the Annual International Symposium on Computer Architecture*, 2000. 23, 25

[RZ97] T. Robinson and W.K. Zuravleff. Controller for a synchronous dram that maximizes throughput by allowing memory requests and commands to be issued out of order, 1997. US Patent 5,630,096. 25

[SKT+05] B. Sinharoy, R N. Kalla, J M. Tendler, R J. Eickemeyer, and J B. Joyner. POWER5 System Microarchitecture. *IBM Journal of Research and Development*, 49(4/5):505–521, 2005. 19

[Smi] Ryan Smith. Nvidia updates gpu roadmap; announces volta family for beyond 2014, http://www.anandtech.com/show/6846/nvidia-updates-gpu-roadmap-announces-volta-family-for-beyond-2014. 23

[TDF+02] J M. Tendler, J S. Dodson, J S. Fields, H. Le, and B. Sinharoy. POWER4 System Microarchitecture. *IBM Journal of Research and Development*, 46(1):5–25, 2002. 19

[UMC+10] Aniruddha N. Udipi, Naveen Muralimanohar, Niladrish Chatterjee, Rajeev Balasubramonian, Al Davis, and Norman P. Jouppi. Rethinking

dram design and organization for energy-constrained multi-cores. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, pages 175–186, New York, NY, USA, 2010. ACM. 25, 26

[USPL07] Rafael Ubal, Julio Sahuquillo, Salvador Petit, and Pedro Lopez. Multi2Sim: A Simulation Framework to Evaluate Multicore-Multithreaded Processors. In *Computer Architecture and High Performance Computing, 2007. SBAC-PAD 2007. 19th International Symposium on*, pages 62 –68, oct. 2007. 33, 34

[VSP+09] Alejandro Valero, Julio Sahuquillo, Salvador Petit, Vicente Lorente, Ramon Canal, Pedro López, and José Duato. An Hybrid eDRAM/SRAM Macrocell to Implement First-Level Data Caches. In *Proceedings of the 42th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 213–221, New York, NY, USA, 2009. ACM. 19

[WLZ+09] Xiaoxia Wu, Jian Li, Lixin Zhang, Evan Speight, Ram Rajamony, and Yuan Xie. Hybrid Cache Architecture with Disparate Memory Technologies. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, pages 34–45, New York, NY, USA, 2009. ACM. 19

[YJE11] Doe Hyun Yoon, Min Kyu Jeong, and Mattan Erez. Adaptive granularity memory systems: A tradeoff between storage efficiency and throughput. *SIGARCH Comput. Archit. News*, 39(3):295–306, June 2011. 26

[YKK+13] P. Yedlapalli, J. Kotra, E. Kultursay, M. Kandemir, C.R. Das, and A. Sivasubramaniam. Meeting midway: Improving cmp performance with memory-side prefetching. In *International Conference on Parallel Architectures and Compilation Techniques*, 2013. 26

[ZLZ+08] Hongzhong Zheng, Jiang Lin, Zhao Zhang, E. Gorbatov, H. David, and Zhichun Zhu. Mini-rank: Adaptive dram architecture for improving memory power efficiency. In *41st IEEE/ACM International Symposium on Microarchitecture, MICRO-41.*, pages 210–221, Nov 2008. 26

[ZLZZ08] Hongzhong Zheng, Jiang Lin, Zhao Zhang, and Zhichun Zhu. Memory access scheduling schemes for systems with multi-core processors. In *Proceedings of the International Conference on Parallel Processing*, 2008. 25