UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISCA
DEPARTAMENTO DE INFORMÁTICA DE SISTEMAS Y COMPUTADORES

Máster en Ingeniería de Computadores y Redes
Trabajo Fin de Máster

Fusión de los Niveles L1 y L2 de la Jerarquía de Memoria Cache Utilizando DWM.

Autor: Hugo Tárrega Sánchez

Tutor: Vicente Jesús Lorente Garcés

Director Experimental: Alejandro Valero Bresó

2 de julio de 2021

Dedicatoria

Cada nuevo científico desempeñando su labor indagadora en el mundo tiene una deuda impagable con todos los científicos que le precedieron. La Ciencia, cómo corpus de saber, constituye la obra colectiva humana más palpable y sincera, a mi parecer. Cada científico aporta según su capacidad y en base a sus recursos sus pequeños avances, que llevan unos pasos más adelante esta gran obra común. Pese a lo difícil y precario de la tarea del investigador en España (que no se puede si no remarcar y denunciar), esta concepción de gran obra común me hace sentir un elemento no imprescindible, pero a la vez terriblemente necesario. Cómo esa idea de gran obra común me ayuda a levantarme por las mañanas, a seguir adelante, a trabajar e incluso a poner a calentar la cafetera, no puedo si no dedicar este trabajo. Así pues, este trabajo va dedicado a todos los científicos que abrieron camino antes que a mí y que permitieron con sus aportaciones que exista hoy mi investigación, mi pequeño granito de arena esforzado. Y es por eso, que, aunque esta deuda de la que hablaba es impagable, dedico este trabajo a aquellos científicos que vinieron antes que yo. Porque la Ciencia es mucho más grande que todos nosotros, pero también somos todos nosotros.

Agradecimientos

Agradecer primero a Julio Sahuquillo por hacer posible mi investigación. Pese a ser una persona muy ocupada siempre tiene tiempo para buscar opciones para que los investigadores más jóvenes podamos abrirnos camino. Además, su guía a la hora de dirigir mi investigación es imprescindible para que esta llegue a buen puerto. Gracias de corazón. Gracias también a Salva Petit, por ser siempre crítico con el trabajo y siempre proponer nuevas formas de refinar nuestra propuesta.

Congratular a mi tutor Alejandro Valero por la sorprendente exhibición de paciencia que hace día a día conmigo. De él, aparte de un sinfín conocimientos a nivel técnico, estoy aprendiendo la dedicación y la perseverancia. Este trabajo, del que me siento orgulloso, espero que le suponga a él un orgullo también, ya que sin él no existiría. Ojalá la academia reconozca tus méritos y te recompense de forma justa por ellos, porque de verdad lo mereces.

Dar las gracias también a mi tutor Vicent Lorente. Me ha dedicado un tiempo que no tiene y no ceja en el empeño de que afiance mis conocimientos de electrónica. Gracias por el soporte y por el apoyo continuo, por creer en el proyecto desde el principio, por empujarme a leer tanto cómo pueda y no quedarme en la superficie. Finalment, un gràcies grupal als anteriorment citats, són la vostra espenta i passió els que fan possible que en aquest país seguisque sent plausible investigar tot i la carestia de recursos.

Resumen

El presente trabajo aborda la necesidad cada vez mayor por parte de la industria de los semiconductores de contar con memorias cache más densas y con un menor consumo energético que las actuales. Debido a que la tecnología actual más utilizada, SRAM, no puede ofrecer estas mejoras, este trabajo propone el uso de las memorias magnéticas DWM (*Domain Wall Memory*) como tecnología emergente sustitutiva.

El presente trabajo aborda principalmente uno de los mayores inconvenientes de DWM como es la latencia variable por acceso a los datos almacenados en una cinta magnética. Este hecho es especialmente crítico en las caches de primer nivel (L1) al encontrarse en el pipeline del procesador y conllevar un impacto directo en el rendimiento del sistema. Para superar este problema, se propone un diseño de cache de datos L1 ascendente. En primer lugar, se diseña una celda de memoria DWM capaz de almacenar múltiples bits y de reducir el impacto de la latencia variable de acceso mediante el uso de múltiples puertos de acceso sobre la cinta, entre otras características. A continuación, se diseña un módulo de cache que integra múltiples celdas DWM, de manera que los conjuntos se organizan de manera entrelazada entre los puertos, favoreciendo la localidad espacial que exhiben las aplicaciones en L1 y por tanto reduciendo la problemática de la latencia variable de acceso. Finalmente, el uso de módulos DWM permite implementar el vector de datos completo de una memoria cache de datos asociativa por conjuntos de n vías. La alta densidad de DWM permite fusionar los niveles L1 y L2 en un sólo nivel DWM con el objetivo de aumentar el rendimiento frente a un diseño de jerarquía de cache convencional SRAM.

La propuesta de cache DWM se implementa y evalúa en el simulador ciclo-a-ciclo Multi2Sim, ampliamente utilizado tanto en la industria como en la academia. Los resultados experimentales muestran que la cache DWM reduce significativamente la penalización media por acceso a memoria, los fallos por kilo-instrucción y los ciclos de parada en el *reorder buffer* frente a un diseño de cache convencional. Todo ello conlleva a una mejora en el rendimiento del sistema de un 10 % en la media no sólo frente a un diseño convencional basado en SRAM sino también frente al diseño DWM del estado-del-arte, referido como TapeCache e implementado como parte del presente trabajo.

Palabras clave: Cache de datos L1, Domain Wall Memory, mononúcleo, Multi2Sim

Resum

El present treball aborda la necessitat cada volta més apressant per part de la indústria dels semiconductors de trobar memòries cau més denses i amb un menor consum energètic que les actuals. Com que la tecnologia actual més utilitzada, SRAM, no pot oferir aquestes millores, aquest treball propose l'ús de les memòries magnètiques DWM (*Domain Wall Memory* com a tecnologia de substitució.)

Aquest treball tracta principalment un dels majors inconvenients de les DWM, com és la latència variable per accés a les dades emmagatzemades en una cinta magnètica. Aquest fet es especialment crític en les memòries cau de primer nivell (L1) al trovar-se en el *pipeline* del processador i implicar un impacte directe en el rendiment del sistema. Per a superar aquest problema, es propose un disseny de memòria L1 de dades ascendent. En primer lloc, es dissenya una cel·la de memòria DWM capaç d'emmagatzemar múltiples bits i de reduir l'impacte de la latència variable d'accés per mitjà de l'ús de múltiples ports d'accés sobre la cinta, entre altres característiques. A continuació, es dissenya un mòdul de memòria cau que integre múltiples cel·les DWM, de manera que els conjunts s'organitzen de manera entrellaçada entre els ports, afavorint la localitat espacial que exhibeixen les aplicacions en L1 i per tant reduint la problemàtica de la latència variable d'accés. Finalment, l'ús de mòduls DWM permet implementar el vector de dades completes d'una memòria cau de dades associatives per conjunts de n vies. L'alta densitat de DWM permet fusionar els nivells L1 i L2 en un només nivell DWM amb l'objectiu d'augmentar el rendiment enfront d'un disseny de jerarquia de memòria cau convencional SRAM.

La proposta de memòria cau amb DWM s'implementa i avalue en el simulador cicle-a-cicle Multi2Sim, àmpliament utilitzat tant en la indústria com en l'acadèmia. Els resultats experimentals mostren que la memòria cau DWM redueix significativament la penalització mitjana per accés a memòria, les fallades per quilo-instrucció i els cicles de parada en el *reorder buffer* enfront d'un disseny de memòria convencional. Tot això comporta a una millora en el rendiment del sistema d'un 10% en la mitjana no sols enfront d'un disseny convencional basat en SRAM sinó també enfront del disseny DWM de l'estat-del-art, referit com TapeCache i implementat com a part del present treball.

Paraules clau: Cau de dades L1, Domain Wall Memory, mononucli, Multi2Sim

Abstract

The present work addresses the growing need of the semiconductor industry for denser cache memories with lower power consumption than the current ones. Since the most widely used current technology, SRAM, cannot offer these improvements, this work proposes the use of DWM (Domain Wall Memory) magnetic memories as a substitute emerging technology.

This work mainly addresses one of the major drawbacks of DWM, which is the variable latency for accessing data stored on a magnetic tape. This fact is especially critical in first-level (L1) caches as they are located in the processor pipeline and have a direct impact on the system performance. To overcome this problem, a bottom-up L1 data cache design is proposed. First, it is designed a DWM memory cell capable of storing multiple bits and reducing the impact of the variable access latency by using multiple access ports on the tape, among other features. Next, it is designed a cache module that integrates multiple DWM cells, such that the sets are organized in an interleaved structure between ports, favoring the spatial locality exhibited by applications on L1 and thus reducing the variable access latency issue. Finally, the use of DWM modules allows implementing the complete data array of an associative data cache with n-way sets. The high density of DWM allows merging the L1 and L2 levels into a single DWM level with the goal of increasing performance over a conventional SRAM cache hierarchy design.

The proposed DWM cache is implemented and evaluated on the Multi2Sim cycle-accurate simulator, which is widely used in both industry and academia. Experimental results show that the DWM cache significantly reduces the average memory access penalty, misses per kilo-instruction, and stall cycles in the reorder buffer compared to a conventional cache design. This leads to a 10% improvement in the average system performance not only over a conventional SRAM-based design but also over the state-of-the-art DWM design, referred to as TapeCache and implemented as part of this work.

Key words: Domain Wall Memory, L1 data cache, Multi2Sim, single-core

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

Este primer capítulo introduce al lector el contexto, motivación, objetivos, metodología y tareas principales que componen el presente trabajo, así como una descripción de la estructura general de la memoria.

1.1 Contexto y Motivación

« *Al principio la Ley de Moore era sólo una observación, un intento de predecir el abaratamiento de la electrónica... pero la industria lo convirtió en una profecía auto cumplida. Ahora, las hojas de ruta de la industria se basan en ese ritmo continuo de mejora, varios nodos tecnológicos aparecen regularmente para mantenernos en esa curva, por lo que todos los participantes en el negocio reconocen que si no se mueven rápido se quedan atrás en la tecnología, por lo que esencialmente de ser sólo una medida de lo que ha sucedido, se ha convertido en un conductor de lo que va a suceder.*». Esta cita se atribuye a Gordon Moore en una entrevista para la Universidad de Stanford [2].

La industria de los semiconductores se esfuerza por mantener vigente la Ley de Moore año tras año. Pese a ello, se está alcanzando el límite físico de miniaturización del transistor. Es por esta razón que la industria se encuentra en una pugna constante por ofrecer nuevas tecnologías y mejoras en el diseño del procesador que permitan seguir aumentando la capacidad de cómputo y almacenamiento de estos dispositivos.

En los procesadores comerciales actuales, el subsistema de memoria es una estructura clave para el rendimiento del procesador, puesto que reduce la diferencia de velocidad entre los núcleos del procesador y la memoria principal. Además, el subsistema de memoria ocupa la mayor parte del área del procesador [3]. Normalmente, la jerarquía de memoria se organiza en al menos tres niveles. Estos niveles se organizan de menor a mayor capacidad de almacenamiento, debido a la relación inversa entre la capacidad de una cache y su tiempo de acceso en ciclos de procesador.

Las caches de primer nivel (L1) deben ser lo suficientemente pequeñas como para estar contenidas en una etapa del pipeline del procesador y cumplir con sus restricciones temporales. Para implementar las caches L1, se utiliza tradicionalmente la tecnología de memoria estática de acceso aleatorio (SRAM). Esta tecnología limita la capacidad de almacenamiento en L1 a unas decenas de KB para cumplir con estas restricciones. Por el contrario, las caches L2 se diseñan con una mayor capacidad, del orden de centenas de KB, aunque lo suficientemente pequeñas como para ocultar la latencia de la ejecución fuera de orden. Por último, las caches L3 tienen una mayor capacidad, del orden de unidades o decenas de MB, para evitar los costosos accesos a la memoria principal fuera del chip.

Teniendo en cuenta las características de cada nivel de la jerarquía de memoria y debido a los problemas intrínsecos de SRAM, están surgiendo nuevas tecnologías que permiten suplir las carencias de SRAM. Una de las tecnologías recientes más destacada es *Domain Wall Memory* (DWM). Frente a SRAM, esta tecnología ofrece un consumo estático prácticamente nulo, un consumo dinámico muy bajo y una densidad de almacenamiento mucho mayor. Su funcionamiento se basa en una cinta magnetizada de nanocables donde se sitúan los bits en forma de dominios magnéticos. Para acceder a esta información, la cinta incluye unos cabezales especiales, hacia los que se desplaza la información para acceder a ella. Este desplazamiento de la información sobre la cinta hasta que se sitúa debajo de un cabezal supone un tiempo de acceso variable a los datos. Debido a esta problemática, algunos trabajos se han centrado en implementar esta tecnología en los niveles L2 y L3, menos sensibles al tiempo de acceso [4, 5, 6].

El presente trabajo propone el uso de la tecnología DWM para implementar una cache de datos L1 que agrupe en un único nivel una cache L2 y la propia cache de datos L1 (L1D). Teniendo en cuenta la mayor densidad de DWM respecto a SRAM, es posible implementar una cache con la misma capacidad que la suma de capacidades de memoria de L1D y L2, pero aún así ocupando menos área. De esta manera se consigue, además de un ahorro de área en el procesador, reducir el tiempo de acceso a los datos en L2 en número de ciclos respecto a una L2 convencional implementada con SRAM. El diseño de cache que se propone en el presente trabajo explota políticas de desplazamiento de los datos del estado-del-arte conjuntamente con una propuesta novedosa de arquitectura y organización de la cache.

Pese a que el presente trabajo centra la línea de investigación en procesadores mononúcleo con un solo *thread* (hilo), cabe destacar que en procesadores multihilo simultáneos (SMT), cada núcleo tiene varios hilos que comparten la cache L1, por lo que, con la solución propuesta, a cada hilo se le podría asignar más capacidad de almacenamiento y reducir el efecto de *cache thrashing* [7]. Esta línea de trabajo se abordará en el futuro y queda fuera del alcance del presente proyecto.

Este trabajo se encuentra dentro del marco de investigación sobre arquitecturas del computador que existe desde hace varios años en el Grupo de Arquitecturas Paralelas (GAP) del Departamento de Informática de Sistemas y Computadores (DISCA) de la *Universitat Politècnica de València*. Nótese que este trabajo supone la continuación natural de la línea de investigación comenzada en el trabajo final de grado [8], donde se propone una implementación preliminar de una arquitectura de cache DWM exclusiva para L1D y los resultados experimentales se limitan a la evaluación de la latencia por operaciones de desplazamiento. En el presente trabajo se fusionan los niveles L1D y L2, presentando un detallado diseño de cache ascendente. Para ello, resulta necesario refinar el diseño y la organización de la propuesta de cache. Además, los resultados experimentales incluyen nuevas métricas del procesador para una evaluación experimental más detallada, así como una comparativa frente a la propuesta DWM del estado-del-arte referida como TapeCache.

1.2 Objetivos

En este trabajo se presentan dos objetivos principales: i) implementar una memoria cache DWM que agrupe L1D y L2 en una única memoria para aprovechar su densidad y bajo consumo energético y ii) resolver los problemas técnicos derivados de este diseño, proponiendo finalmente una arquitectura que permita superar en rendimiento a una cache SRAM convencional y a la propuesta TapeCache. Estos dos objetivos se pueden dividir a su vez en una serie de hitos. Estos hitos permiten, por un lado, clarificar el trabajo

realizado; y por otro, establecer y planificar una serie de metas concretas como parte del desarrollo de este proyecto.

Los hitos en los cuales se divide el primer objetivo, ordenados de manera cronológica, son los siguientes:

- Instrumentar un simulador ciclo-a-ciclo para modelar memorias basadas en tecnología DWM.
- Implementar la política de desplazamiento del estado-del-arte que mejores resultados en cuanto a rendimiento del sistema ha demostrado (*Dynamic Lazy*).
- Diseñar e implementar una propuesta de distribución entrelazada de conjuntos que permita aprovechar la localidad espacial de los datos.
- Desarrollar una arquitectura de memoria cache que permita aprovechar las virtudes de la tecnología DWM.

A su vez, los hitos en los que se desglosa el segundo objetivo son los siguientes:

- Implementar la arquitectura de cache del estado-del-arte TapeCache [4], con el fin de presentar una evaluación cuantitativa de rendimiento frente a la propuesta del presente trabajo.
- Cuantificar las mejoras conseguidas confrontando la propuesta de este trabajo con un diseño convencional de memoria cache SRAM.
- Evaluar la mejora de la propuesta en base al análisis de diferentes métricas relativas al rendimiento y específicas de la arquitectura de los computadores.

1.3 Metodología

El proyecto emplea una metodología de trabajo basada fundamentalmente en el uso del simulador ciclo-a-ciclo Multi2Sim [9]. Este simulador ofrece soporte a diversas arquitecturas, entre ellas la arquitectura x86. Una vez configurado e instrumentado según las especificaciones de diseño, es posible lanzar experimentos, para luego realizar un análisis de los resultados arrojados por el simulador y refinar la implementación en iteraciones sucesivas. Teniendo esto en cuenta, podemos organizar el presente trabajo en diversas fases o tareas, que corresponderán a los experimentos realizados:

- Modelado del sistema a simular. Este paso consiste en conocer con detalle la arquitectura y el sistema con el que se va a trabajar para introducir sus parámetros mediante los ficheros de configuración que ofrece Multi2Sim.
- Instrumentación del código fuente del simulador y mantenimiento en un repositorio. Resultará indispensable modificar el simulador para modelar las características que Multi2Sim no ofrece en soporte nativo, principalmente, el uso de la tecnología DWM. Dada la naturaleza compleja del simulador, realizar estas modificaciones ha supuesto una dificultad añadida durante el desarrollo del proyecto.
- Lanzamiento masivo de simulaciones. El lanzamiento de las cargas de trabajo se realizará en un clúster de computadores real. En este sentido, resultarán necesarios conocimientos de Shell, Bash y *scripts* para mantener un entorno de simulación y gestionar una cola de trabajos.

- Extracción y tratamiento automático de los datos. Los datos relevantes del resultado de la simulación se deben extraer y tratar mediante el desarrollo de una serie de *scripts* con el objetivo de obtener gráficas comparativas y tablas que permitan visualizar de forma sencilla los resultados. A partir de las tablas y las gráficas se obtendrán las conclusiones sobre los experimentos realizados.

En resumen, para la realización de este trabajo ha sido necesario lidiar con varios problemas que han supuesto un plus de dificultad: i) la complejidad del simulador, cuyo periodo de aprendizaje requiere consultar ampliamente la documentación y el propio código fuente, ii) la gran cantidad de datos volcados por Multi2Sim al acabar cada simulación, que obliga a realizar *scripts* que seleccionen los datos de manera específica, teniendo así que aprender también cómo tratar de forma automatizada grandes volúmenes de datos y iii) realizar los experimentos en un clúster compartido por varios usuarios, lo que obliga al uso de colas de procesos. A pesar de las dificultades expuestas, se han alcanzado los objetivos planteados en el inicio de este trabajo, cuyos resultados y conclusiones más relevantes se presentan en los capítulos finales del mismo.

1.4 Estructura de la Memoria

Sin contar con la presente introducción, el resto de la memoria se organiza en siete capítulos, resumidos a continuación:

- **Capítulo 2, Tecnologías de Memoria:** exposición de los conocimientos necesarios para el completo entendimiento del proyecto, incluyendo referencias al estado-del-arte conforme se detallan las diferentes tecnologías.
- **Capítulo 3, Antecedentes:** en este capítulo se comentan las aportaciones de otros artículos a la propuesta original de cache DWM así como unaa justificación de la elección de los mismos.
- **Capítulo 4, Implementación de DWM Fusionando L1 y L2:** se detalla la propuesta de diseño desglosada en las partes que la componen mediante un diseño de cache ascendente.
- **Capítulo 5, Diseños de Cache DWM del Estado-del-Arte:** este capítulo describe, entre otras, la arquitectura TapeCache, mostrando sus ventajas e inconvenientes. También se detallan los mecanismos que este diseño utiliza para reducir la penalización por desplazamiento de los datos.
- **Capítulo 6, Entorno Experimental:** se recoge el entorno experimental sobre el cual se valida la propuesta y se obtienen los resultados de este trabajo, incluyendo un simulador de procesadores, cargas de evaluación, depuradores e infraestructura de lanzamiento de simulaciones, entre otras consideraciones.
- **Capítulo 7, Resultados Experimentales y Discusión:** se muestran y discuten los resultados experimentales. En primer lugar se abordan las características principales del procesador modelado, para a continuación presentar resultados con la distribución de latencia, fallos por kilo-instrucción, ciclos de parada del ROB y el rendimiento de los sistemas de memoria analizados.
- **Capítulo 8, Conclusiones y Trabajo Futuro:** este capítulo expone una reflexión acerca de cómo las asignaturas del Máster en Arquitectura de Computadores y Redes han posibilitado la elaboración del trabajo, las conclusiones principales y qué

lineas de investigación resultaría interesante explorar en el futuro a partir de los resultados obtenidos.

- **Apéndice A, Scripts de Tratamiento de Datos:** en este anexo se presentan los *scripts* elaborados con el objetivo de automatizar la extracción de datos. Es importante remarcar que este anexo se encuentra fuera del presente documento, pero se adjunta en el apartado Anexos de la aplicación de gestión de TFM.

CAPÍTULO 2
Tecnologías de Memoria

Las memorias cache son un componente de diseño esencial en los procesadores actuales puesto que permiten ocultar las diferencias de velocidad entre el procesador y la memoria principal. Este capítulo describe las tecnologías utilizadas actualmente en el diseño y fabricación de memorias cache, así como sus características principales.

2.1 Static Random Access Memory

La tecnología *Static Random Access Memory* (SRAM) se ha empleado tradicionalmente para implementar las estructuras de memoria de los procesadores, incluyendo la jerarquía de memoria cache, el banco de registros o las estaciones de reserva, entre otras. Este hecho se debe principalmente a que se trata de la tecnología que ofrece una mayor velocidad de acceso. Esta velocidad de acceso se consigue implementando cada celda de memoria SRAM con hasta 6 transistores (celdas 6T). Debido al gran número de transistores por celda, los dos grandes inconvenientes de la tecnología SRAM son su su baja densidad de integración y su alto consumo energético estático derivado del consumo estático provocado por el biestable y las corrientes de fuga de los transistores. Además, este tipo de consumo tiende a aumentar en sucesivos nodos tecnológicos más pequeños puesto que es proporcional al número de transistores por unidad de superficie. Por estos motivos, y puesto que el tamaño de la jerarquía de memoria aumenta en sucesivas generaciones de procesador, la jerarquía de memoria supone un porcentaje elevado del área y del consumo de los procesadores actuales.

La Figura 2.1 muestra la implementación de una celda 6T. Los cuatro transistores centrales implementan un bucle de inversores que permite almacenar un valor lógico '0' o '1'. El resto de transistores de paso se controlan mediante la señal *wordline* (WL), permitiendo operaciones de lectura y escritura en la celda a través de las señales *bitline* (BL) y su complementaria ($\overline{BL}$). Finalmente, V_{dd} corresponde al punto de suministro de corriente de la celda. Por ejemplo, V_{dd} adopta una tensión nominal de 0,7 V en nodos de TSMC de 10 y 7 nanómetros [10].

2.2 Embedded Dynamic RAM

La tecnología *embedded Dynamic* RAM (eDRAM) compatible con CMOS apareció hace algo más de una década como alternativa a SRAM debido a la problemática asociada con las mismas [11]. A diferencia de las celdas SRAM, las celdas eDRAM se implementan utilizando un condensador y un transistor de paso (celdas 1T1C). Este diseño permite que el consumo estático sea mucho menor y el área en torno a 8 veces menor que SRAM, a

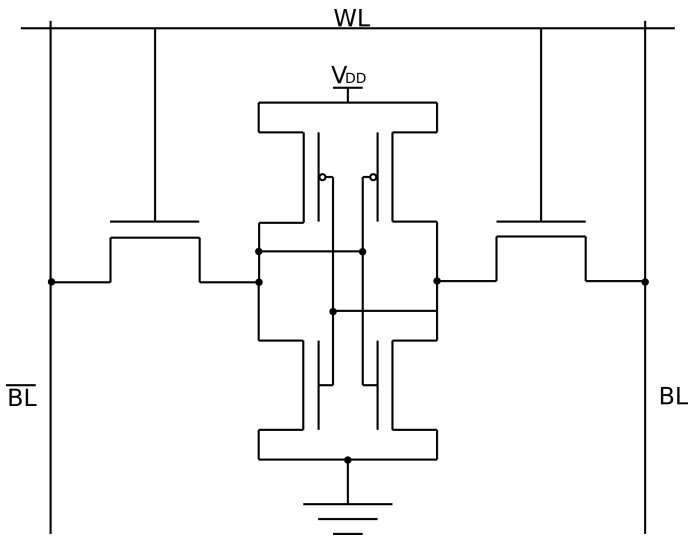


Figura 2.1: Esquema de una celda de memoria 6T SRAM.

costa de una velocidad de acceso no tan elevada frente a SRAM. Además, debido a que los valores lógicos en una celda eDRAM se almacenan como carga en el condensador, el valor lógico almacenado no se puede obtener pasado un periodo de tiempo al cual nos referimos como tiempo de retención. En este sentido, son necesarias operaciones de refresco periódicas y frecuentes que permitan retener el valor almacenado. Las operaciones de refresco compiten con los accesos regulares a la cache por parte del procesador, lo cual puede conllevar no sólo a un consumo de energía adicional sino también a una degradación severa de las prestaciones del sistema. Además, las lecturas en las celdas eDRAM son destructivas, es decir, por cada lectura se precisa de una escritura posterior para mantener el valor lógico almacenado.

Por todos los motivos anteriores, el uso de la tecnología eDRAM en procesadores comerciales se ha restringido al último nivel de cache (L3), donde a diferencia de L1, el tiempo de acceso no tiene un impacto directo sobre las prestaciones del sistema. Dos ejemplos de estos procesadores son IBM POWER [12] e Intel Haswell [13]. Por otro lado, algunos trabajos académicos han propuesto el uso de celdas eDRAM en caches L1, realizando modificaciones en el diseño de la celda 1T1C original para disminuir el impacto en el tiempo de acceso y reducir o eliminar por completo la necesidad de operaciones de refresco [14, 15, 16, 17].

La Figura 2.2 muestra la implementación de una celda 1T1C. La presencia o ausencia de carga en el condensador etiquetado como C se corresponde con un '1' o '0' lógico almacenado, respectivamente. De manera similar al diseño de celda 6T, la señal WL permite realizar operaciones de lectura y escritura a través del transistor de paso y la señal BL.

2.3 Non-Volatile Memory

Las tecnologías *Non-Volatile Memory* (NVM) como la memoria Flash, *Phase Change* RAM (PCRAM), *Resistive* RAM (ReRAM), *Spin-Transfer Torque* RAM (STT-RAM) o *Domain Wall Memory* (DWM) ofrecen múltiples beneficios como una muy alta densidad, consumo estático nulo y la habilidad de retener los valores lógicos almacenados durante largos periodos de tiempo, eliminando la necesidad de operaciones de refresco cómo ocurre con la tecnología eDRAM. Entre estas tecnologías, destacan STT-RAM y DWM

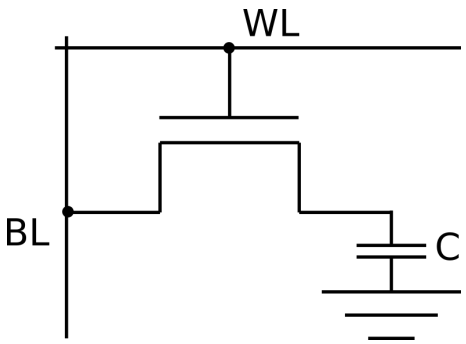


Figura 2.2: Esquema de una celda de memoria 1T1C eDRAM.

por su compatibilidad con CMOS, su resistencia y sus tiempos de lectura competitivos en comparación con los de la tecnología SRAM.

2.3.1. Spin-Transfer Torque RAM

La Figura 2.3 ilustra el diseño de una celda de memoria STT-RAM. Estas celdas están compuestas por un elemento *Magnetic Tunnel Junction* (MTJ) y un transistor de paso. El MTJ es el dispositivo de almacenamiento de la celda. Consiste en dos capas ferromagnéticas (capa libre y de referencia) separadas por una barrera aislante ultra-fina de óxido de magnesio. La orientación magnética de la capa libre (paralela o anti-paralela respecto a la capa con orientación fija de referencia) define el valor lógico almacenado [18].

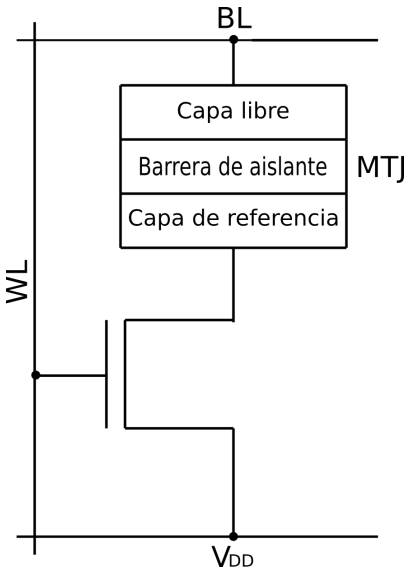


Figura 2.3: Esquema de una celda de memoria STT-RAM.

Comparada con SRAM, la tecnología STT-RAM adolece de un tiempo de acceso lento y un consumo dinámico elevado durante las operaciones de escritura. Debido a esta problemática, gran parte de las investigaciones relacionadas con esta tecnología se han centrado en la cache de último nivel, donde la mayoría de las operaciones de escritura son filtradas por los niveles de cache anteriores [19, 20, 21, 18, 22, 23]. Otras líneas de investigación han empleado STT-RAM en propuestas de diseño de cache L1. Estos diseños disminuyen el tiempo de retención de las celdas STT-RAM, lo cual alivia las ineficiencias de las operaciones de escritura a expensas de incorporar operaciones de re-

fresco [24, 25, 26], o proponen diseños de cache híbrida SRAM/STT-RAM, cuya porción SRAM limita la alta densidad y bajo consumo estático de STT-RAM [27, 28].

2.3.2. Domain Wall Memory

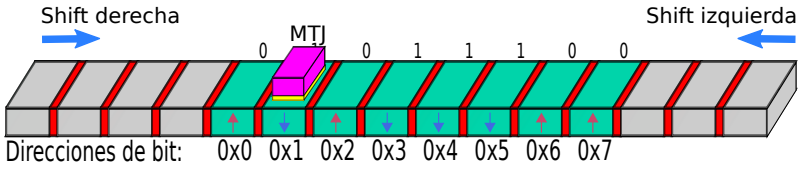
Domain Wall Memory (DWM) es una tecnología NVM emergente que ha atraído un amplio interés debido a su altísima densidad y eficiencia energética respecto a las anteriores tecnologías NVM. Al contrario que STT-RAM, la tecnología DWM, también conocida como *Racetrack Memory* (RTM), permite compartir los costosos puertos de acceso o cabezales (una capa del MTJ y un transistor de paso) con múltiples bits de memoria. Esto se logra mediante el uso de una fina banda magnética de nanocables, conocida como *racetrack* o cinta, y dividida en sucesivos dominios. Un dominio consiste en un campo magnético polarizado en torno a un átomo o un grupo de átomos. Cada dominio representa un bit de información, donde la dirección de magnetización determina el estado binario almacenado. Para poder acceder a ellos, los dominios se desplazan en un sentido u otro de la cinta aplicando una corriente a lo largo de la misma, alineándose debajo de un cabezal, el cual se mantiene estático, y formando un MTJ. Cabe destacar que, cuando no se requiere de una operación de desplazamiento para acceder a la dirección solicitada (los datos requeridos se encuentran debajo del cabezal), el tiempo de acceso de DWM es comparable con el de SRAM. Al integrar múltiples bits en un nanocable, la tecnología DWM ofrece una densidad de almacenamiento aproximadamente 12 y 28 veces más alta en comparación con STT-RAM y SRAM, respectivamente [29].

La Figura 2.4 muestra un esquema con el desplazamiento de los dominios sobre una cinta. Las flechas rojas y azules en cada dominio representan la polarización, es decir, el valor lógico almacenado. El estado inicial (mostrado en la Figura 2.4a) sitúa el dominio correspondiente en la dirección de memoria 0x1 debajo del cabezal. Supongamos que deseamos acceder a la dirección 0x3 (Figura 2.4b). El desplazamiento necesario para llevar a cabo este acceso se produce desplazando previamente la cinta en dos dominios hacia la derecha. Normalmente, cada desplazamiento de un dominio tiene una penalización temporal de un ciclo de procesador [5]. En definitiva, el mecanismo de desplazamiento, ya sea hacia la izquierda o derecha, se implementa mediante una serie de transistores que permiten mediante pulsos desplazar los dominios sobre la cinta hasta que el dominio en cuestión se posiciona debajo del cabezal.

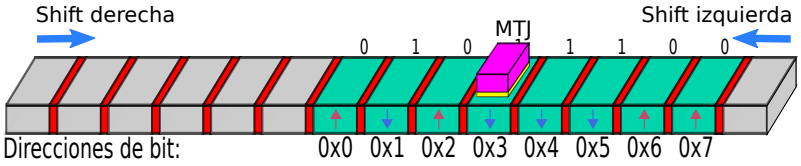
Respecto a la forma en que se realiza la lectura/escritura, depende del método de que elijamos para realizar estas operaciones, ya que existen diversas formas de hacerlo atendiendo a la literatura existente. Es por ello que en el Capítulo 3 se discute y elije la forma de lectura y escritura que se adopta en este trabajo, ya que no se trata de una decisión trivial.

Como se aprecia en la Figura 2.4, el desplazamiento de los bits sobre la cinta requiere dominios adicionales, los cuales no forman parte de la capacidad de almacenamiento en bits, para que el desplazamiento no sea destructivo. De no disponer de estos dominios adicionales, los bits podrían perderse en los extremos de la cinta al realizar desplazamientos. Estos dominios adicionales se muestran en color gris. Al desplazarse por la cinta, los bits ocupan progresivamente los dominios adicionales sin perderse por los extremos. La necesidad de estos dominios implica que una cinta de 16 dominios sólo pueda almacenar 8 bits. Generalizando, se necesita una cinta con una capacidad de 2N dominios para almacenar N bits.

La Figura 2.5 muestra la implementación física de una cinta. Las formas triangulares oscuras que se pueden observar en las cintas de la Figura 2.5a son los dominios. En este ejemplo, el color oscuro de los dominios indica una polarización anti-horaria, lo cual se



(a) Estado inicial



(b) Desplazamiento hacia la derecha en la cinta

Figura 2.4: Esquema con el desplazamiento de dominios sobre una cinta.

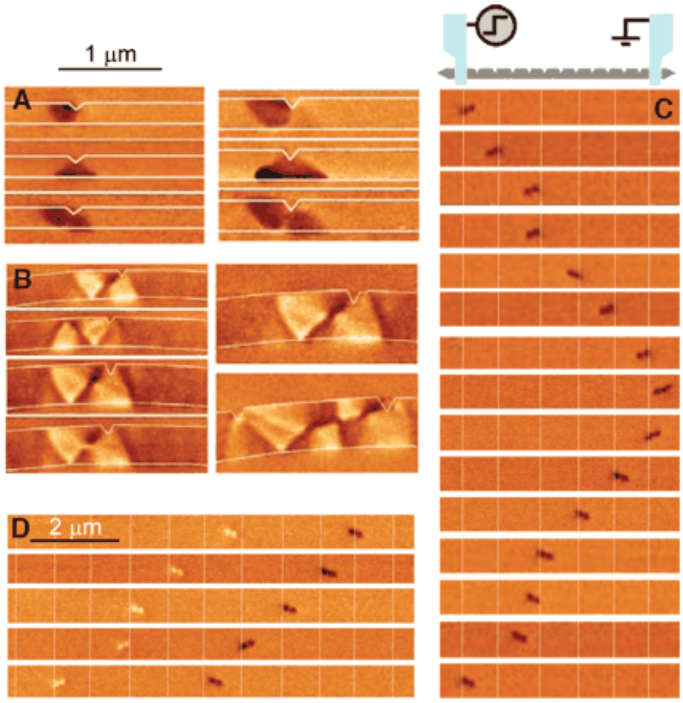


Figura 2.5: Vista de dominios sobre una cinta [1].

interpreta como un '1' lógico. Por el contrario, la Figura 2.5b muestra varios dominios con un color claro, polarizados en sentido horario, lo cual se interpreta como un '0' lógico. La Figura 2.5c muestra cómo se desplazan los dominios sobre la cinta. Concretamente, se puede apreciar el desplazamiento de un '1' lógico hacia el extremo derecho de la cinta, para luego retornar hasta la posición inicial. En la parte superior de esta figura vemos un pequeño dibujo esquemático del inyector de pulsos responsable del desplazamiento de los bits sobre la cinta. Finalmente, en la Figura 2.5d se observa la progresión de posiciones en la cinta para ambos valores lógicos '0' y '1'.

Tecnología	Velocidad	Densidad	Consumo estático	Consumo dinámico	Lectura destructiva	Refresco
SRAM	Alta	Baja	Alto	Medio	No	No
DRAM	Baja	Alta	Bajo	Medio	Sí	Sí
STT-RAM	Media	Alta	Nulo	Alto	No	No
DWM	Variable	Muy alta	Nulo	Bajo	No	No

Tabla 2.1: Comparativa entre diferentes tecnologías de memoria.

2.4 Comparativa

Como se ha comentado en las secciones anteriores, cada tecnología de memoria presenta sus ventajas e inconvenientes. La Tabla 2.1 resume las características principales de cada una de ellas.

Las tecnologías alternativas a SRAM ofrecen en mayor o menor medida una reducción del consumo estático, así como una mayor densidad de integración. Entre las tecnologías de memoria discutidas, DWM ofrece por diseño un consumo estático nulo, un consumo dinámico bajo y una muy alta densidad. Esto se consigue compartiendo los cabezales entre distintos dominios en una cinta. De hecho, el número de transistores en una memoria DWM no depende tanto de la capacidad de almacenamiento en número de bits sino del número de cabezales por cinta y del número de cintas en la memoria, asumiendo que cada cinta tiene asociado un mecanismo independiente para realizar operaciones de desplazamiento.

El principal inconveniente de las tecnologías alternativas a SRAM es su reducción de velocidad en el acceso a los datos. En el caso de DWM, el tiempo de acceso a los datos es variable y depende de la posición de los cabezales con respecto a los datos solicitados. Por esta razón, resulta necesario explorar organizaciones de cache y políticas de desplazamiento de los bits que reduzcan el impacto del tiempo de acceso variable.

CAPÍTULO 3
Antecedentes

La cache que se propone en el presente trabajo tiene como objetivo integrarse en el primer nivel de cache de datos (L1D). Esto implica que esta memoria debe tener una características específicas como son ocupar un área suficientemente pequeña para integrarse en el pipeline del procesador y reducir el tiempo de acceso a un rango adecuado para este nivel de cache (e.g., 1-3 ciclos). También es necesario resolver problemas específicos de esta tecnología, como la necesidad de 2N dominios para almacenar N bits, lo que perjudica la densidad de la propuesta. Además, resulta necesario elegir entre diferentes propuestas de lectura y escritura para esta tecnología, ya que cada una presenta unas características asociadas.

El presente capítulo detalla algunas aportaciones de otros artículos a este trabajo para adecuar el uso de la tecnología DWM en L1D.

3.1 MTJ de Lectura/Escritura

Las operaciones de lectura y escritura en la tecnología DWM se pueden realizar de manera sencilla a través de un MTJ. Esta sección detalla cuales son los componentes principales que intervienen en estas operaciones y sus inconvenientes, en particular en las operaciones de escritura.

La lectura del valor lógico almacenado en un dominio se realiza midiendo la magnetorresistencia del túnel del MTJ, que cambia según la dirección de magnetización del dominio. Esto permite distinguir entre '0' y '1', utilizando para ello un circuito que actúe como *sense amplifier*[1], según la resistencia variable en el MTJ [30]. De esta forma, es posible realizar una lectura del bit situado debajo del MTJ en un único ciclo de reloj y con un circuito relativamente sencillo, lo que permite ahorrar área. Es por estas razones que este método de lectura se mantendrá en todo el trabajo.

En cuanto al método de escritura mediante MTJ, consiste en hacer pasar una gran cantidad de corriente a través del MTJ con el objetivo de cambiar la polarización de la capa libre sobre el que se encuentra. Esta forma de escritura presenta dos problemas principales: por una parte una latencia alta, ya que requiere sostener una gran cantidad de corriente durante varios ciclos y por otra, un gran coste energético derivado de la necesidad de esta gran cantidad de corriente [24, 25, 26].

[1] Circuito electrónico que permite ampliar la corriente de un circuito a valores concretos a partir de rangos de voltaje fijados previamente.

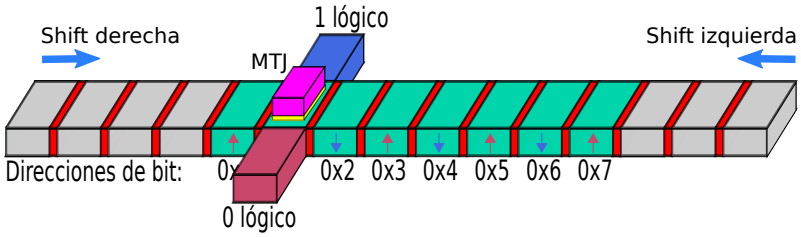


Figura 3.1: Cinta DWM con dos dominios de escritura prefijados a los valores '0' y '1'.

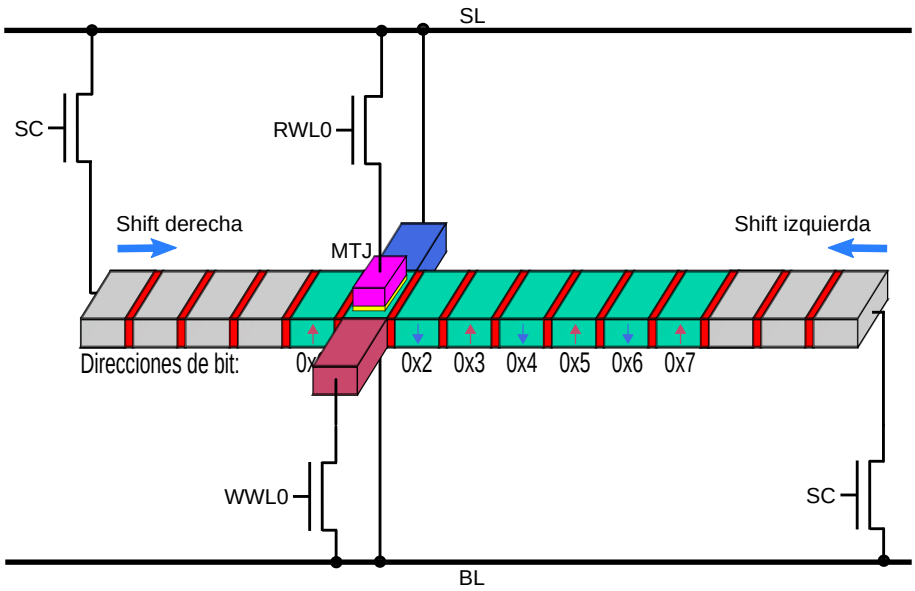


Figura 3.2: Cinta magnética con representación de la electrónica necesaria para acceder a los bits contenidos.

3.2 Escritura Mediante Shift

La escritura mediante shift constituye una alternativa frente a la escritura tradicional mediante un MTJ. Esta alternativa resulta más eficiente en términos de rendimiento y consumo energético. Esta sección detalla el funcionamiento de la escritura mediante shift a partir de la organización de cinta DWM mostrada en la Figura 2.4 del capítulo anterior.

La Figura 3.1 muestra los componentes necesarios añadidos al diseño de cinta DWM para poder realizar escrituras mediante shift. El diseño añade a la cinta dos dominios a los lados del MTJ. Estos dominios tienen una polarización fija. Esta polarización está representada en la imagen en rojo el '0' lógico y en azul el '1' lógico. La escritura se realiza mediante un desplazamiento de los valores lógicos fijos en estos dominios laterales hacia el dominio debajo del MTJ. De esta manera se consigue escribir un '0' o un '1' en un sólo ciclo de reloj (que es lo que cuesta a nivel temporal un shift). Nótese que como la polarización magnética de los dominios laterales es fija, se consigue escribir en el dominio debajo del MTJ sin perder la polarización de los dominios laterales.

La Figura 3.2 muestra la electrónica necesaria para realizar las operaciones de lectura mediante MTJ y de escritura mediante shift. El MTJ puede proporcionar acceso de lectura a diferentes dominios en la misma cinta y está conectado a su correspondiente transistor de paso accionado por la línea de palabra (RWL0), formando un puerto de acceso. Obsérvese que el número máximo de impulsos de corriente necesarios para colocar cualquier

Operación	RWL0	WWL0	SC	BL	SL
Lectura	V_{dd}	Gnd	Gnd	Gnd	V_{dd}
Escribir '0'	Gnd	V_{dd}	Gnd	V_{dd}	Gnd
Escribir '1'	Gnd	V_{dd}	Gnd	Gnd	V_{dd}
Shift derecha	Gnd	Gnd	V_{dd}	V_{dd}	Gnd
Shift izquierda	Gnd	Gnd	V_{dd}	Gnd	V_{dd}
Inactividad	Gnd	Gnd	Gnd	X	X

Tabla 3.1: Tabla con las diferentes operaciones en una celda DWM.

dominio bajo el MTJ es en este caso es 7. Considerando que el MTJ está situado sobre 0x1, serían necesarios 6 ciclos para acceder a 0x7.

Para la lectura, se proporciona un pulso a nivel alto en la línea RWL0 consiguiendo con esto que la corriente pueda transitar desde BL hacia SL, pasando por el MTJ. A su vez, se transmite corriente desde BL a SL. Cuando la corriente pasa a través del MTJ este ofrece una resistencia que varia en función de la dirección en que esté magnetizado el dominio, consiguiendo de esta forma distinguir entre un '0' y un '1'. Una vez esta corriente circula por BL, se conecta a un *sense amplifier*, como se ha comentado anteriormente, que permite conocer si se trata de un '0' o un '1' lógico.

La escritura se realiza de forma diferente según se requiere almacenar un '1' o un '0'. Para escribir un '1' en el dominio sobre el que está el MTJ, SL se conecta a V_{dd} y BL se conecta a tierra (Gnd) para hacer pasar una corriente de SL hacia BL, lo que provocará un desplazamiento vertical hacia abajo, desplazando el '1' del dominio fijo al dominio libre. De esta manera la corriente circula desde SL hacia BL, y si el transistor WWL0 que actúa como interruptor recibe V_{dd}, se consigue un shift de la polarización magnética presente en el dominio azul al dominio sobre el que se sitúa el MTJ. De esta manera, se escribe de manera efectiva un '1' lógico en un sólo ciclo (el ciclo necesario para realizar el shift).

La escritura de un '0' lógico ocurre de manera similar, pero en sentido contrario. De nuevo, la señal WWL0 se conecta a V_{dd} para que el transistor de paso conduzca, actuando como interruptor cerrado, dejando pasar corriente. Esta vez se persigue que la corriente circule desde BL hacia SL, así que BL se conecta a V_{dd} y SL se conecta a Gnd. De esta manera se provoca un shift del dominio en rojo hacia el dominio debajo del MTJ polarizándose igual que este, con el equivalente magnético a un '0' lógico.

Además del puerto de acceso, la celda está conectada a dos puertos de desplazamiento, que consisten en un par de transistores accionados por la línea de desplazamiento SC. Si se conecta tanto SC como BL a V_{dd} y SL a Gnd, se consigue hacer circular la corriente provocando un shift hacia la derecha. Si por el contrario, es BL la que se conecta a Gnd y SL a V_{dd}, se provoca un desplazamiento hacia la izquierda. Estas líneas permiten generar pulsos de corriente para realizar operaciones de desplazamiento en el sentido de las agujas del reloj o en sentido contrario cuando se le suministra corriente a SC, de manera que este permita que circule la corriente.

Nótese que las líneas de bits (BL) y de origen (SL) se ajustan a diferentes tensiones en función de la operación (es decir, lectura, escritura, desplazamiento en el sentido de las agujas del reloj y desplazamiento en sentido contrario a las agujas del reloj) que se va a realizar. La Tabla 3.1 resume las condiciones de tensión para todas las operaciones. Con fines visuales se muestra en rojo la conexión a tierra y en verde la tensión de alimentación. Con una X se representa indiferencia entre ambas.

3.3 Comparativa entre Escritura Mediante MTJ y Mediante Shifting

Teniendo en cuenta lo expuesto en las dos secciones anteriores, la presenta sección justifica el motivo por el cual en este trabajo se elige la escritura mediante shift para implementar la propuesta. A modo de resumen se muestra la Tabla 3.2 donde se compara la forma de escritura mediante MTJ y la que utiliza shift.

	Latencia de escritura	Consumo	Transistores
Convencional	Varios ciclos	Alto	4 por celda (2 shift, 4 cabezales)
Con shift	1 ciclo de reloj	Bajo	6 por celda (2 shift, 4 cabezales)

Tabla 3.2: Tabla comparativa con las dos formas de escritura en la celda DWM.

En definitiva, sacrificando un poco de área (6 transistores para realizar operaciones frente a 4) se consigue una memoria con mucha menor latencia y sin necesidad de tanto consumo dinámico. Al haber dos transistores más, el consumo estático por corrientes de fuga aumenta en la escritura mediante shift, pero este resulta insignificante comparado con el consumo dinámico de la escritura convencional. Teniendo esto en cuenta, en el resto del documento, la escritura asumida será mediante shift.

3.4 Cinta en Anillo

La implementación de DWM en forma de cinta magnética tiene el inconveniente de la necesidad de dominios extra (dominios en color gris en las Figuras 3.1 y 3.2) para asegurar que la lectura de los datos no sea destructiva. Este problema implica que para poder almacenar N bits de forma efectiva son necesarios 2N dominios.

Una posible solución para el inconveniente de los dominios auxiliares es implementar las cintas magnéticas en forma de anillo [30]. Esta solución presente en varios trabajos sobre DWM permite un desplazamiento de los datos sobre los dominios sin necesidad de dominios extra. La Figura 3.3 parte de las anteriormente citadas pero añade un MTJ extra y los dominios en una estructura circular en vez de lineal. En ella se muestra una cinta en forma de anillo con 32 dominios efectivos.

Este trabajo asume la implementación de cintas circulares, ya que utilizando este diseño se consigue representar N bits con N dominios, en vez de 2N como era necesario anteriormente.

3.5 Equilibrio entre Puertos y Dominios

El número de puertos de acceso (MTJ) y los dominios son dos de los principales parámetros que afectan a la latencia, la energía y el área de un diseño de cache basado en la tecnología DWM [31]. La celda DWM original en forma de anillo está adaptada a las caches de último nivel, donde se desea una alta densidad por encima de una latencia de acceso reducida en número de ciclos de desplazamiento. De esta forma, los accesos a un anillo de 128 bits se realizan a través de 4 puertos de acceso.

Por otro lado, trabajos anteriores centrados en caches L1 sacrifican la densidad DWM para eliminar por completo el requisito de desplazamiento [32, 4]. Esto se consigue con un diseño de disposición de celdas que consiste en un mismo número de dominios y puertos

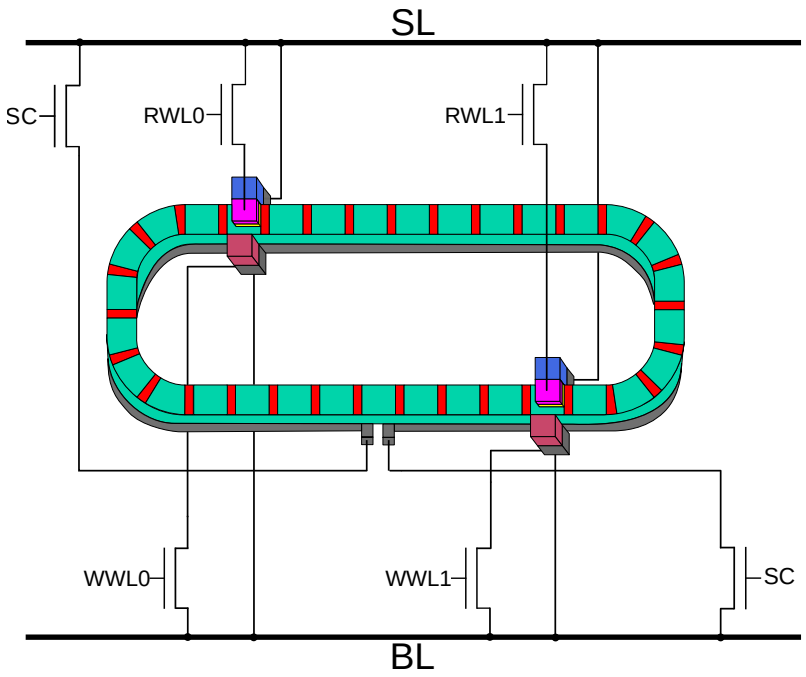


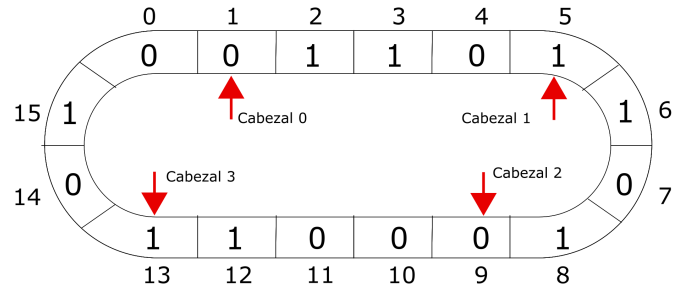
Figura 3.3: Diagrama de una cinta de 32 bits con dos MTJ.

de acceso, donde cada dominio tiene un puerto de acceso dedicado, asemejándose a un diseño de STT-RAM. Este diseño se denomina celda DWM de 1 bit. A diferencia de los trabajos anteriores, en este trabajo se propone un diseño de celda DWM de varios bits que cumple los requisitos de alta densidad y latencia de acceso media reducida deseados en la cache L1.

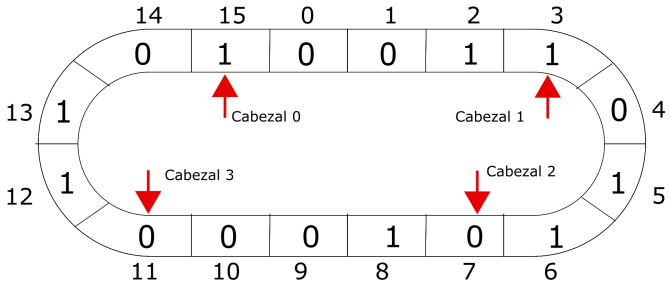
3.6 Política de Desplazamiento de Cabezales

Como ya se ha mencionado, el acceso a un bit en DWM requiere de una operación de desplazamiento previa para alinear el bit solicitado bajo un puerto MTJ. Dado que las operaciones de desplazamiento implican una latencia de acceso variable en un DWM, es imperativo minimizar el número de ciclos de desplazamiento tanto como sea posible. Para ello se han propuesto diversas políticas de desplazamiento.

Venkatesan *et al.* proponen diferentes políticas de gestión de puertos para una memoria DWM multipuerto atendiendo a dos criterios diferentes: selección de puertos y actualización de puertos [4]. La selección de puertos establece cual es el puerto adecuado para acceder a los datos solicitados. Existen dos enfoques, denominados estático y dinámico. El enfoque estático determina de antemano a qué bits se debe acceder a través de un puerto específico, mientras que el enfoque dinámico calcula el puerto más cercano en tiempo de ejecución. Por otra parte, la actualización de puertos define la posición del cabezal con respecto a los puertos tras la finalización de un acceso. Existen dos políticas, denominadas *eager* y *lazy*. La política *eager* restaura la cinta a la posición por defecto después de un acceso, mientras que la política *lazy* mantiene la última posición de la cinta hasta el siguiente acceso. Este trabajo asume una selección dinámica de puertos para minimizar el número de ciclos requeridos en una operación de desplazamiento y una actualización de puertos *lazy* para explotar la localidad espacial presente en las caches de datos L1. La sobrecarga de la lógica requerida para implementar tal política de gestión de puertos es insignificante.



(a) Estado inicial.



(b) DL después de haber leído la posición 3.

Figura 3.4: Desplazamiento de los datos para realizar una lectura utilizando DL.

A modo de ejemplo ilustrativo del funcionamiento de la estrategia dinámica y *lazy*, también referida como *Dynamic Lazy* (DL), la Figura 3.4a muestra el estado inicial de una cinta circular simplificada de 16 bits con 4 cabezales o puertos MTJ. Considerando que se quiere acceder al dominio 3, DL elegirá el cabezal más cercano a este dominio. En este caso concreto, podrían elegirse tanto el cabezal 0 como el 1, pero como se puede observar en la Figura 3.4b se ha elegido el cabezal 1. Ante una situación como esta, este trabajo asume la elección del cabezal con mayor número identificador.

El lector puede consultar [4] para más detalles o [8] donde se ofrece una justificación teórica y experimental acerca de por qué DL es la política que más reduce la penalización por desplazamiento.

CAPÍTULO 4
Implementación de DWM Fusionando L1 y L2

Este capítulo detalla la propuesta de cache de datos L1 fusionada con L2 y adaptada para que resulte competitiva comparada con una cache convencional SRAM. Para ello, se presenta un diseño de cache ascendente. En primer lugar, se diseña específicamente una celda DWM de múltiples bits con forma de anillo para cumplir los requisitos de alta densidad y latencia de acceso media reducida. A continuación, la celda propuesta se utilizará para implementar un módulo DWM compuesto por un número de líneas de cache de diferentes conjuntos. Además, se propondrá una organización de conjuntos para reducir aún más la penalización de la latencia por desplazamiento. Por último, el módulo propuesto se utilizará para implementar las vías y todos los conjuntos de un array de datos completo.

4.1 Celda DWM Multi-Bit en Anillo

La latencia de acceso en una celda DWM está determinada por la corriente que pueden proporcionar los transistores de acceso. Se ha demostrado que un tamaño de transistor de 10F[1] ofrece un buen compromiso en cuanto a rendimiento y ocupación de área de la cache [33, 34]. Por otro lado, la latencia por desplazamiento está influida tanto por el número de dominios (longitud de la cinta magnética) como por el tamaño de los transistores que controlan el desplazamiento. Cintas más largas requieren más corriente para desplazar todos los dominios y, por tanto, transistores de desplazamiento más grandes. Para un tamaño de transistor de desplazamiento dado, para desplazar los datos sobre la cinta más rápidamente, esta debe ser lo más corta posible [30]. Este trabajo limita la longitud de la cinta en forma de anillo a 128 dominios para establecer transistores de desplazamiento con un tamaño de 10F y una latencia de desplazamiento dentro de un ciclo de reloj.

Para optimizar la disposición de los componentes en la celda, tanto los transistores de desplazamiento como los de acceso se sitúan bajo la cinta. El número de transistores de desplazamiento se limita a 2 para garantizar los movimientos de desplazamiento en sentido horario y antihorario. El número de transistores de acceso está determinado por el número de dominios entre dos puertos consecutivos por encima de la cinta. Este trabajo limita dicho número de dominios a 4. De este modo, se necesitan 32 puertos para una cinta de 128 dominios, lo que se traduce en 64 transistores de acceso (se necesitan dos transistores de acceso para implementar un puerto). Obsérvese también que, suponiendo

[1] F responde a *feature size* y se refiere a la distancia mínima en nanómetros entre la fuente y el drenaje de un transistor.

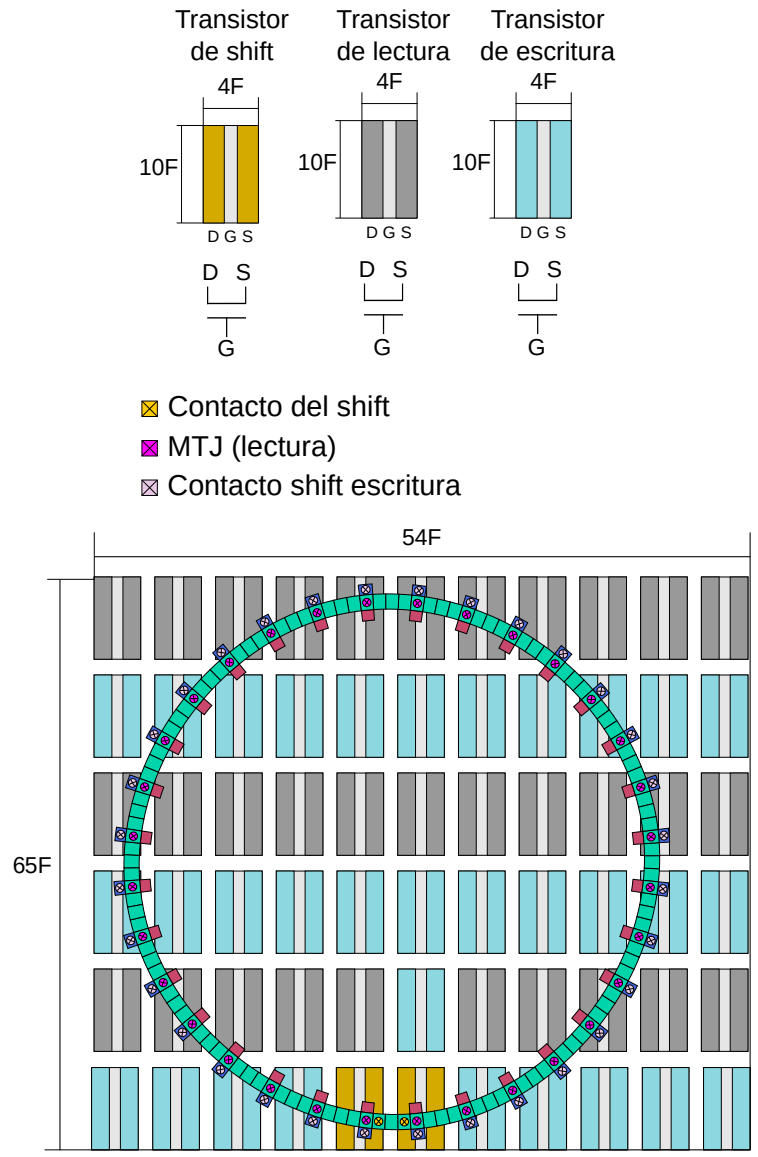


Figura 4.1: Propuesta de celda de memoria DWM de 128 bits.

una gestión *Dynamic Lazy* (DL) de los puertos, una distancia de 4 dominios entre puertos sucesivos define una latencia de desplazamiento máxima de 2 ciclos para acceder al bit más lejano de un puerto.

La Figura 4.1 muestra la organización de la celda para la cache propuesta. La celda consta de 11×6 transistores. Los transistores de desplazamiento están situados en la última fila de arriba a abajo. De acuerdo con el tamaño definido para los transistores y asumiendo un espacio entre transistores adyacentes de 1F, el área de la celda es de $54F \times 65F = 3510F^2$.

Partiendo de las explicaciones de la sección anterior, vamos a mostrar de forma simple la justificación de la gran densidad que presentan las DWM sobre SRAM.

La Figura 4.2 pone de manifiesto la mayor densidad de la tecnología DWM frente a SRAM, mostrando el número de celdas SRAM que puede contener la celda DWM definida. Teniendo en cuenta que una celda SRAM 6T típica ocupa una área de $10F \times 14,6F = 146F^2$ [35], pueden caber hasta 20 celdas SRAM dentro del área que delimita la celda DWM. De este modo, la densidad de la celda propuesta es 6,4 veces superior a SRAM

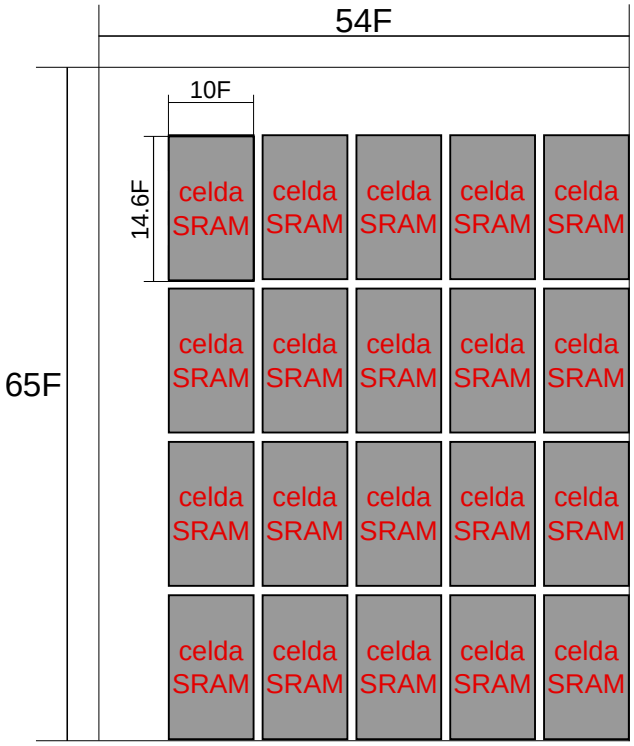


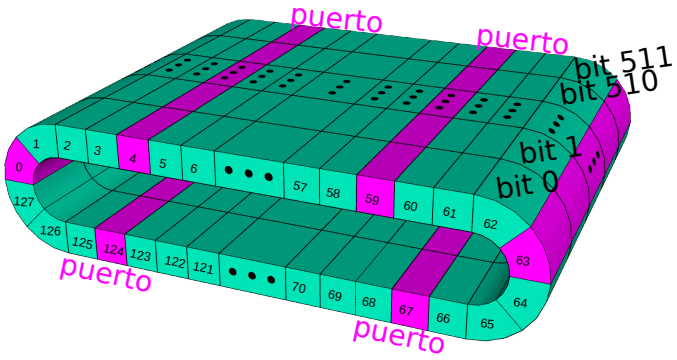
Figura 4.2: Comparación del área ocupada por la celda DWM de 128 bits y celdas SRAM de 1 bit.

(i.e., 128/20). Por el contrario, una celda DWM de 1 bit incluiría sólo 32 bits de acuerdo con el número de puertos definido con anterioridad, lo que limitaría la densidad a 2,67x con respecto a SRAM.

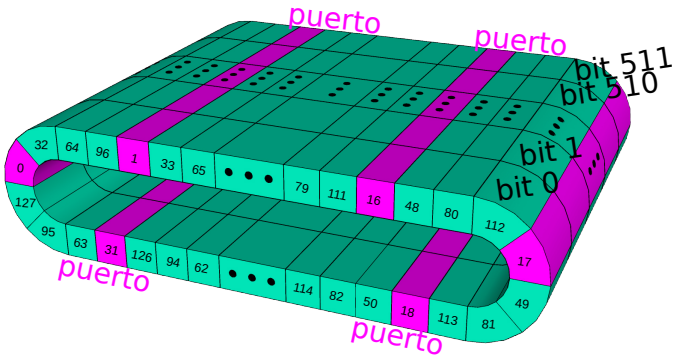
Al igual que en [30], este trabajo asume una longitud de 40 nm para cada dominio y una anchura de pared de dominio (*domain wall*) de 5 nm entre dos dominios adyacentes, por lo que cada unidad de almacenamiento tiene una longitud de 45 nm. Considerando un nodo tecnológico de 40 nm, las dimensiones de altura y anchura de la celda propuesta permitirían hasta 48 y 57 dominios, respectivamente, lo que se ajustaría a una longitud ideal de cinta en forma rectangular de 210 dominios. Sin embargo, como se ha comentado anteriormente, la longitud de la cinta se limita a 128 dominios no sólo para establecer una penalización por desplazamiento de un dominio dentro de un ciclo, sino también para simplificar el direccionamiento asumiendo una potencia de 2.

4.2 Organización de Módulo de Cache

La celda DWM sirve para implementar un módulo de memoria. Suponiendo un tamaño de línea de cache de 64 B, un módulo consta de 512 celdas DWM. Las posiciones del bit *i*-ésimo de cada celda implementan la línea *i*-ésima del módulo. A su vez, la línea *i*-ésima se refiere a una línea de un conjunto con identificador *i*. De este modo, un módulo se refiere a 128 conjuntos, cada uno de los cuales consta de una sola línea de 64 B. Todos los bits que implementan una línea comparten la señal de desplazamiento (SC) y las líneas de palabra (RWL y WWL), lo cual permite desplazar todas las celdas de un módulo al mismo tiempo y acceder en paralelo a todos los bits que conforman una línea.



(a) Organización secuencial de los conjuntos.



(b) Organización entrelazada de los conjuntos.

Figura 4.3: Organización de un módulo DWM en 512 celdas. Un módulo se organiza en 128 conjuntos, cada uno de los cuales contiene una sola línea de 512 bits.

La Figura 4.3a muestra la distribución de las 512 celdas (anillos) para implementar un módulo. El número dentro de un dominio se refiere al identificador de conjunto. A efectos ilustrativos, sólo se destacan algunos conjuntos[2].

En una organización de conjuntos secuencial, los conjuntos están distribuidos consecutivamente a lo largo de un anillo. De esta manera, con una distancia de 4 conjuntos entre puertos consecutivos, los conjuntos 0, 4, 8, 12, 16, y así sucesivamente, se alinean bajo un puerto, representando un estado inicial. Independientemente de la dirección de desplazamiento, se mantiene una distancia de 4 conjuntos entre puertos consecutivos.

Esta distribución de conjuntos no aprovecha la localidad espacial que presentan las caches de datos L1. Por esta razón, este trabajo propone una organización de conjuntos alternativa que coloca los conjuntos de forma intercalada en los puertos, correspondiente a la función *id conjunto* % *num. puertos*. De este modo, los conjuntos consecutivos se alinean con los puertos. La Figura 4.3b muestra un estado en el que los conjuntos de 0 a 31 están alineados con los 32 puertos. De este modo, un acceso a los conjuntos vecinos no implica ninguna operación de desplazamiento. Obsérvese que, en la estrategia propuesta, el acceso a cualquier conjunto del rango 32-63 implicaría sólo una penalización de desplazamiento de un ciclo. Tras este desplazamiento, todos los conjuntos dentro de dicho rango permanecerían alineados bajo un puerto. De esta manera se consigue explotar la localidad espacial para mejorar el tiempo de acceso a los datos.

A efectos de cuantificar la mejora de la estrategia de distribución de conjuntos entrelazada con los puertos, la Figura 4.4 muestra un desglose de la latencia de desplazamiento en número de ciclos para ambas estrategias secuencial (Sec) y propuesta (Ent). Una la-

[2]También cabe remarcar que con propósito ilustrativo y para facilitar su visualización en una figura tridimensional, la cinta tiene forma ovalada en vez de puramente circular

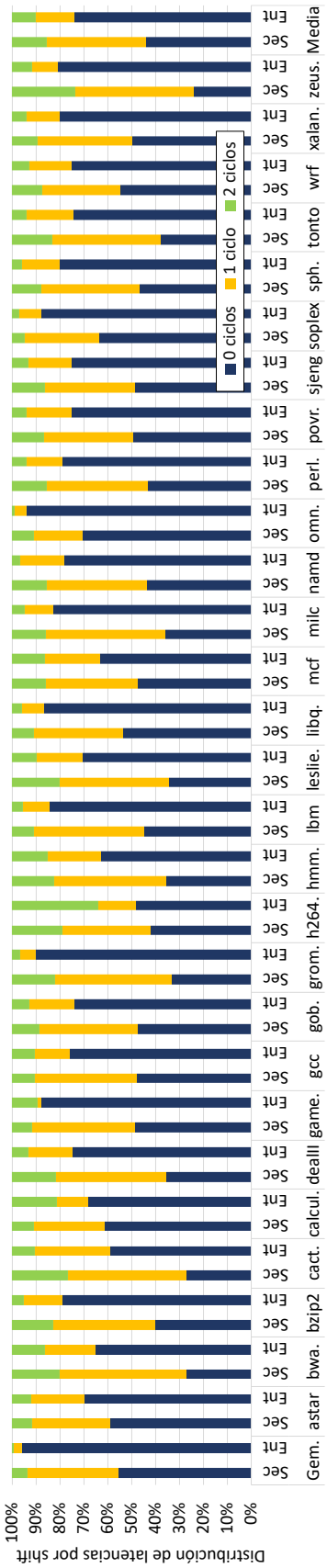


Figura 4.4: Distribución de la latencia de desplazamiento para la organización secuencial de conjuntos (Sec) e intercalada por puertos (Ent) en una cache de datos L1 DWM de 160 KB.

tencia de 0 ciclos significa que no se requiere ninguna operación de desplazamiento, ya que el bloque solicitado se encuentra alineado bajo un puerto. Los resultados se muestran para una cache de datos L1 DWM de 160 KB donde se ejecutan todas las aplicaciones del conjunto de pruebas SPEC CPU 2006 [36][3].

Como se puede observar, al explotar la localidad espacial de los datos, la organización de conjuntos propuesta mejora en gran medida la latencia de desplazamiento respecto a la organización secuencial en todas las aplicaciones estudiadas. El porcentaje de latencia de 0 ciclos mejora especialmente en aplicaciones con una alta localidad espacial como *GemsFDTD*, *gamess*, o *gromacs*, donde dicho porcentaje supera un 80 %. En general, la organización de conjuntos propuesta evita que se produzca una operación de desplazamiento en un 76 % de los accesos a la cache. Este porcentaje se reduce a un 43 % para el enfoque secuencial. En el resto del documento se asume la organización de conjuntos entrelazados con los puertos en los módulos de DWM.

4.3 Organización de Array de Datos de Cache

Un módulo compuesto por 128 conjuntos y una sola línea de 64 B por conjunto implementa un array de datos modesto consistente en 8 KB con mapeo directo. Sin embargo, la alta densidad de la tecnología DWM permite implementar una capacidad de memoria cache mucho mayor con respecto a una cache SRAM convencional en una limitación de área determinada. Esta sección explora la posibilidad de aumentar tanto el número de conjuntos como el de vías bajo la restricción de área definida por el diseño convencional de cache.

Para aumentar el número de conjuntos más allá de 128, este trabajo propone utilizar múltiples módulos DWM para implementar una cache. Cada módulo contiene un subconjunto de 128 conjuntos adyacentes de la cache. Esta distribución de conjuntos en módulos beneficia la localidad espacial, ya que las operaciones de desplazamiento sólo afectan al módulo de destino donde reside el conjunto solicitado. Esto evita que los módulos restantes realicen una operación de desplazamiento, manteniendo el mismo estado de acuerdo a su último acceso. El número total de conjuntos de cache se limita a un múltiplo de 128 para explotar todos los dominios disponibles de un módulo.

Al contrario que las caches de bajo nivel, las caches L1 asociativas por conjuntos de n vías suelen acceder en paralelo a las n etiquetas y a las n vías del conjunto de destino para reducir el tiempo de acceso. Teniendo en cuenta este comportamiento, las diferentes vías de la cache DWM propuesta están físicamente separadas en diferentes bancos implementados con diferentes módulos. Esta decisión de diseño proporciona un tiempo de acceso mínimo porque una misma operación de desplazamiento se lleva a cabo en todas las vías del conjunto de destino al mismo tiempo con la comparación de etiquetas.

De acuerdo con la mayor capacidad de almacenamiento que ofrece la tecnología DWM, este trabajo explora combinar la cache de datos L1 y la cache L2 para implementar una única cache de datos L1. Así, la cache L2 SRAM se elimina por completo de la jerarquía de memoria en chip propuesta. Bajo esta consideración, se exploran dos posibles estrategias de diseño denominadas iso-área e iso-capacidad.

El enfoque iso-área intenta implementar una DWM que ocupe la misma área que la cache de datos L1 original más el área de la cache L2. En este sentido, teniendo en cuenta que la densidad de la celda DWM propuesta es 6,4x con respecto a SRAM (ver Sección 4.1), se podría implementar una cache de aproximadamente 1 MB en lugar de una cache de datos L1 de 32 KB y una cache L2 de 128 KB. Sin embargo, el principal reto

[3] Se remite al lector al Capítulo 6 para más detalles sobre el entorno experimental.

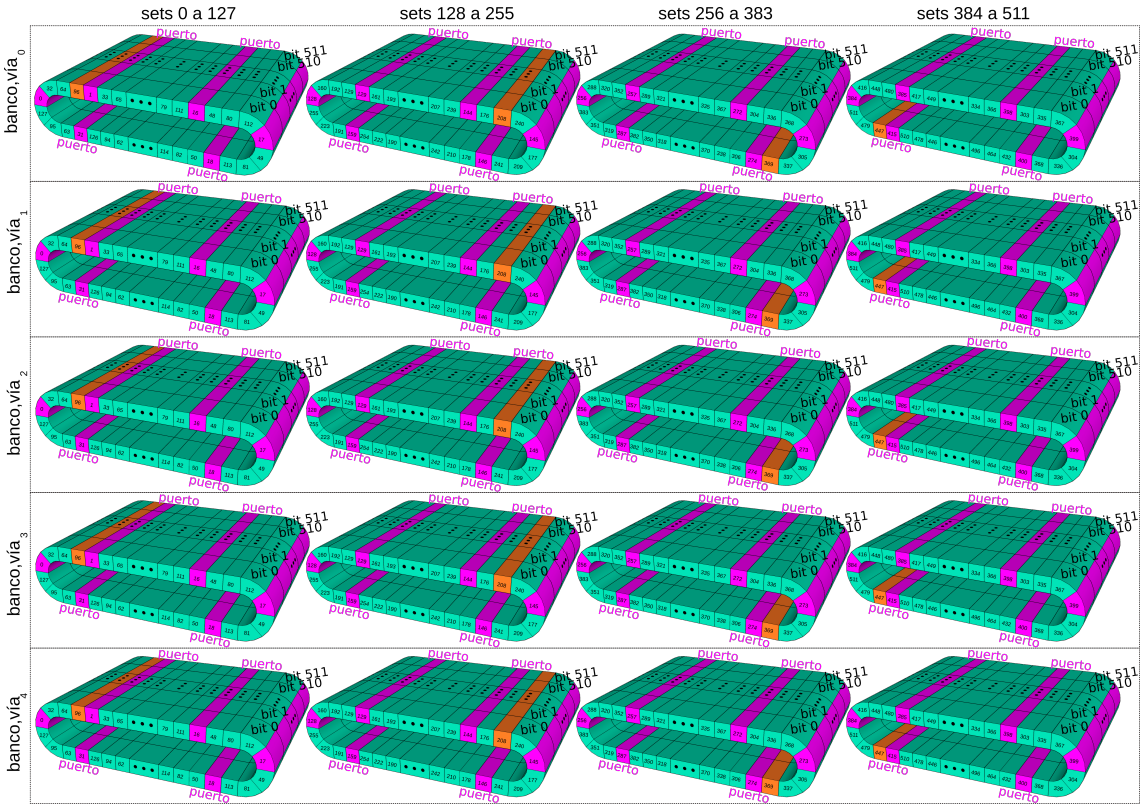


Figura 4.5: Diagrama de una organización de cache DWM con una capacidad de 160 KB, conjuntos de 512 bits y 5 vías. Cada conjunto se expande a 5 anillos, uno por cada vía de cache (filas). Cada columna consta de tantos módulos como vías de cache. Cada vía se expande a 4 columnas (número de conjuntos/128). En color naranja se destaca un par de conjuntos de destino.

del enfoque iso-área es mantener una velocidad de acceso rápida. Esto se debe principalmente a una mayor sobrecarga de tiempo de los circuitos periféricos en una área mucho mayor en comparación con una cache L1 convencional. Por este motivo, este trabajo deja el diseño iso-área como trabajo futuro.

El enfoque iso-capacidad implementa una cache de datos L1 con la misma capacidad que una cache de datos L1 convencional más una cache L2. Por ejemplo, una jerarquía de memoria con dos niveles compuesta por una cache de datos L1 de 32 KB y una cache L2 de 128 KB se traduciría en una única cache de datos L1 de 160 KB. Este enfoque elimina potencialmente el área requerida por la L2 e incluso reduce el área original ocupada por la cache de datos L1, dependiendo del tamaño de la L2. Por ejemplo, una DWM de 160 KB ocupa menos área en comparación con una cache de datos L1 de 32 KB. Respecto a una jerarquía de memoria inclusiva, la idea clave detrás del enfoque de iso-capacidad es mejorar el rendimiento global del sistema aumentando la capacidad de almacenamiento efectiva y manteniendo el tiempo medio de acceso por debajo del de una cache L2 a pesar de la necesidad de operaciones de desplazamiento.

La Figura 4.5 muestra cómo se organizan los conjuntos y las vías en los módulos para una configuración de cache de 160 KB y 5 vías. Obsérvese que cada columna de módulos implementa un número de conjuntos consecutivos, empezando por el rango 0-127 en la primera columna de izquierda a derecha. Por otro lado, cada fila implementa un banco/-vía diferente. De este modo, un acceso a todas las vías de un conjunto puede realizarse en paralelo en todos los módulos de la columna donde se encuentra el conjunto de destino. Por ejemplo, cuando se accede al conjunto naranja mostrado en la primera columna, se accede a todas las vías de la cache (filas en la figura) del conjunto de destino en paralelo,

de forma similar a las caches de alto rendimiento. Por otro lado, dado que los dominios de destino (coloreados en naranja) se encuentran justo al lado de un puerto (color morado), se requiere una operación de desplazamiento de un dominio. Análogamente, un acceso al conjunto naranja indicado en la segunda columna implica una operación de desplazamiento de 2 dominios en todos los módulos de la columna.

Finalmente, cabe destacar que este trabajo asume que el array de etiquetas de la cache propuesta se implementa con tecnología SRAM o DWM de 1 bit, puesto que este array es mucho más pequeño que el homólogo de datos y por tanto se esperan menos beneficios tanto energéticos como de área.

CAPÍTULO 5
Diseños de Cache DWM del Estado-del-Arte

El presente capítulo detalla diseños de cache DWM del estado-del-arte clasificándolos en enfoques que proponen organizaciones de datos y políticas de gestión de datos para caches de último nivel (L2), políticas de gestión de puertos para dichas organizaciones y diseños de cache DWM en el primer nivel de la jerarquía.

5.1 Organizaciones de Cache de Último Nivel

Las organizaciones de cache DWM de último nivel se centran normalmente en el segundo nivel de la jerarquía de memoria on-chip. La presente sección detalla los aspectos de diseño más relevantes de cuatro propuestas diferentes.

5.1.1. TapeCache

La arquitectura de memoria cache referida como TapeCache (TapeC) es una de las propuestas más utilizadas y citadas en la literatura [4]. En el Capítulo 7 se cuantifica el rendimiento de TapeC frente a la propuesta original de este trabajo.

El diseño de la TapeC parte de un módulo DWM multi-bit constituido por 512 cintas en forma de tira con 64 dominios por cinta. Sin embargo, tan sólo la mitad de estos dominios almacenan información útil, mientras que e resto de dominios aseguran que el desplazamiento de los datos sobre la cinta no sea destructivo. La Figura 5.1 muestra los dominios útiles de un módulo de la TapeC. La capacidad del módulo es de 2 KB (32 bits $\times$ 512 cintas). En otras palabras, el módulo definido almacena 32 líneas de cache, cada una de las cuales está constituida por 512 bits. Nótese además que el módulo consta de cuatro puertos de acceso, estableciendo una distancia de 8 dominios entre puertos consecutivos.

El módulo TapeC se utiliza para implementar una cache L2 de 128 KB y 8 vías. De esta manera, cada vía constituye un total de 16 KB. De acuerdo con esta capacidad, cada vía requiere un total de 8 módulos. La Figura 5.2 ilustra la organización de la TapeC en módulos. Obsérvese como las vías se clasifican en vías rápidas y densas. Las vías densas, 7 de las 8 vías de la cache, se implementan mediante los módulos definidos en la Figura 5.1. Por tanto, estas vías tienen una latencia de acceso variable de acuerdo con la necesidad de desplazamiento. Por el contrario, la vía definida como rápida se implementa mediante tecnología DWM de un bit, es decir, las cintas contienen tantos puertos como dominios útiles. Al contrario que las vías densas, la vía rápida permite acceder a los datos contenidos con una latencia constante y sin necesidad de realizar desplazamientos. El

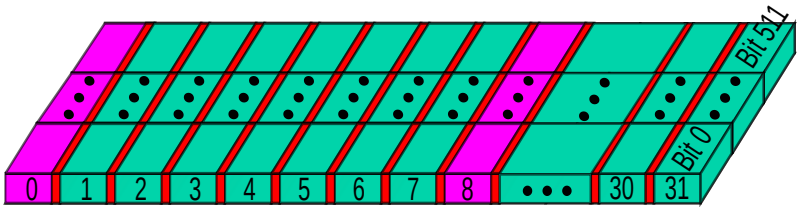


Figura 5.1: Módulo TapeC de 2 KB agrupando 512 cintas con 32 bits cada una. El color cian muestra la ubicación de los puertos de acceso.

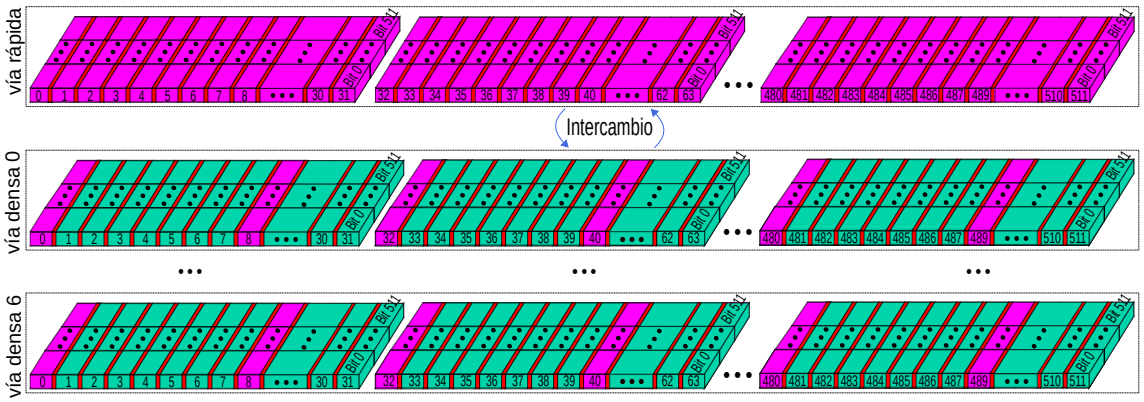


Figura 5.2: Memoria cache TapeC con 128 KB y 8 vías.

número de vías rápidas se limita a tan sólo una por su mayor sobrecoste en área respecto a la implementación de las vías densas.

Operaciones de Intercambio

El diseño de la TapeC permite realizar operaciones de intercambio de datos entre la vía rápida y el resto de vías densas. El objetivo de estas operaciones es tratar de concentrar la mayoría de los accesos a la cache en la vía rápida para evitar la latencia por desplazamiento requerida en las vías densas.

Para cumplir con este objetivo, las operaciones de intercambio se llevan a cabo cuando, en un conjunto determinado, se realizan dos accesos consecutivos a una vía densa. Bajo esta situación, los datos requeridos por segunda vez y contenidos en una vía densa se intercambian físicamente con los de la vía rápida. De esta manera, la operación mantiene los datos más recientemente utilizados en la vía rápida. Esta técnica se implementa mediante un array auxiliar que registra sobre qué vía densa se ha realizado el último acceso a un conjunto.

El intercambio de datos entre la vía rápida y una vía densa conlleva un sobrecoste de 2 ciclos de reloj, sin tener en cuenta la penalización por acceso a los datos implicados en cada vía y los potenciales desplazamientos en la vía densa.

En definitiva, el intercambio de datos persigue aprovechar la vía rápida de manera dinámica, cambiando su contenido en tiempo de ejecución y aprovechando la localidad de los datos de los programas.

Umbral de Desplazamiento y Desplazamiento Especulativo (*Preshift*)

Al contrario de lo que ocurre en un circuito en forma de anillo, la gestión de los puertos en una cinta en forma de tira puede provocar una alineación sesgada de los mismos, es decir, algunos dominios pueden quedar excesivamente alejados de un puerto de acceso. Para hacer frente a este problema, durante los periodos de inactividad de la TapeC, se restablece la cinta a una posición predefinida cuando los puertos están sesgados más allá de un umbral de 4 dominios. Nótese que la distancia entre puertos consecutivos de 8 dominios y el umbral de 4 dominios define una latencia por desplazamiento máxima de hasta 11 dominios.

Con el objetivo de minimizar aún más la penalización por desplazamiento, TapeC implementa una política de pre-desplazamiento o *preshift*. Esta política tiene en cuenta la localidad espacial de las aplicaciones y predice que el siguiente acceso a la cache solicitará un bloque del siguiente conjunto sucesivo con respecto al último acceso. De acuerdo con esta predicción, tras finalizar el último acceso y antes de comenzar el siguiente acceso, el puerto más cercano se alinea especulativamente bajo dicho conjunto predicho antes de que se realice el siguiente acceso.

Nótese que una política de *preshift* para caches L1 probablemente competiría con los accesos regulares e impactaría negativamente en el rendimiento del sistema.

5.1.2. Otras Propuestas

Sun *et al.* proponen una organización de cache DWM multi-bit en la que se implementa un conjunto con múltiples cintas de lado a lado sobre el área definida por los puertos de acceso [37]. La longitud de las cintas determina el número de vías en un conjunto. Los conjuntos se almacenan secuencialmente en las tiras, mientras que el número de cintas dentro de un conjunto está determinado por el tamaño del bloque. Los bloques de instrucciones y datos se asignan estáticamente a conjuntos específicos. Se accede a los conjuntos de instrucciones y datos a través de puertos de sólo lectura y de lectura/escritura, respectivamente. Además, el número de vías activas de un determinado conjunto de datos se ajusta dinámicamente en función de la demanda de la aplicación en tiempo de ejecución, reduciendo así el número medio de operaciones de desplazamiento.

De forma similar al trabajo anterior, el diseño de la arquitectura jerárquica y densa para cintas (HDART) despliega cintas de lado a lado sobre el área de los puertos de acceso [5]. Una cinta i-ésima es lo suficientemente grande como para almacenar los bits i-ésimos de un subconjunto de vías consecutivas de un número dado de conjuntos sucesivos. De acuerdo con una distribución diagonal de los puertos de acceso bajo las cintas, un subconjunto de las mismas vías no consecutivas de múltiples conjuntos se alinean bajo los puertos y se puede acceder a ellas en paralelo. Además, los autores proponen un mecanismo de gestión de datos que identifica los bloques con un acceso intensivo. Gracias a operaciones de intercambio, dichos bloques se sitúan más cerca de los puertos de acceso para aliviar la penalización por desplazamiento.

La arquitectura de cache SKM propone una organización intercalada por bits en la que los subconjuntos de cintas que contienen un bloque completo se disponen en bancos [38]. Además, la cinta i-ésima de un banco almacena los bits i-ésimos de conjuntos consecutivos. De este modo, se puede acceder a todas las vías de un conjunto dado en paralelo. Este trabajo trata con bancos de cache con un tiempo de acceso diferente como consecuencia de las variaciones en el proceso de fabricación del chip. Se diseña una política de migración de datos para que los bloques con un acceso intensivo se mantengan en los bancos identificados como rápidos.

5.2 Políticas de Gestión de Puertos

Las políticas de gestión de puertos del estado-del-arte se basan en la arquitectura TapeC. La arquitectura de cache denominada como Preshift (no confundir con la técnica específica anterior para la TapeC) introduce una política de prebúsqueda por hardware para predecir la siguiente operación de desplazamiento en caches de bajo nivel [39]. La información utilizada para la predicción es la distancia en número de dominios entre accesos consecutivos. El hardware propuesto almacena en una tabla diferentes secuencias de desplazamiento pasadas y una predicción de desplazamiento para cada secuencia. Para el próximo acceso, se realiza una predicción si la secuencia actual coincide con alguna secuencia pasada.

El enfoque Multilane Racetrack Caches añade diferentes esquemas de compresión de bloques de datos a la TapeC [40]. De este modo, la gestión de los puertos se realiza con una granularidad más fina de 16 bits en lugar de un bloque entero, lo que permite un desplazamiento independiente para los bloques comprimidos, mejorando el rendimiento del sistema.

DyReCTape es una cache DWM reconfigurable dinámicamente [41]. Este enfoque reduce la penalización por desplazamiento ajustando el número de bits activos por pista en función de la demanda de la aplicación en tiempo de ejecución. Además, la parte no utilizada de la cache se aprovecha como cache víctima.

5.3 Organizaciones de Cache de Primer Nivel

Los trabajos centrados en las caches L1 suelen explotar la tecnología DWM de un sólo bit, en la que se incluyen tantos puertos de acceso como el número de bits almacenados para eliminar completamente la latencia de desplazamiento a costa de sacrificar la densidad [32, 4].

En cuanto a los diseños DWM de varios bits para caches L1, el enfoque Preshift comentado anteriormente también se evalúa en caches L1 DWM de varios bits. Sin embargo, incluso con la predicción de desplazamiento habilitada, asumir una organización TapeC L2 para la cache L1 conduce a una sobrecarga de desplazamiento que perjudica significativamente el rendimiento del sistema. En consecuencia, los autores descartan la aplicación de DWM multi-bit en L1.

FusedCache integra los niveles de cache L1 y L2 en una única cache L1 implementada con tecnología DWM multi-bit [42]. En la cache DWM propuesta, los autores diferencian los bloques de datos que originalmente pertenecerían a los niveles L1 y L2. El diseño fuerza a los bloques L1 a alinearse bajo los puertos, mientras que los bloques L2 son los que requieren una operación de desplazamiento previa a su acceso. Esto implica un direccionamiento e indexación de la cache alternativos, así como operaciones de intercambio entre los bloques L2 a los que se accede y los bloques L1 menos utilizados recientemente para preservar dicha diferenciación. Las operaciones de intercambio son especialmente costosas cuando la asociatividad de L1 y L2 difiere, lo cual suele ser habitual en los procesadores comerciales. A diferencia de este diseño, el trabajo presentado en este documento no distingue entre bloques L1 y L2. En lugar de operaciones de intercambio, se propone el uso de múltiples puertos de acceso por cinta y una organización de conjuntos intercalados por puertos para reducir el número de operaciones de desplazamiento.

CAPÍTULO 6
Entorno Experimental

Este capítulo recoge el entorno experimental sobre el cual se valida la propuesta y se obtienen los resultados de este trabajo, incluyendo un simulador de procesadores, cargas de evaluación, depuradores e infraestructura de lanzamiento de simulaciones, entre otros.

6.1 Simulador Multi2Sim

Multi2Sim es una aplicación software de código libre, escrita en C y para sistemas operativos Linux que permite modelar y validar con detalle (ciclo a ciclo) la simulación de diferentes arquitecturas de procesador, incluyendo CPU y GPU [43, 9]. Multi2Sim permite ejecutar a su vez cualquier aplicación compilada para un modelo de procesador soportado y obtener con ello estadísticas de rendimiento del procesador. Además, Multi2Sim es capaz de simular desde procesadores simples mononúcleo (como MIPS) hasta procesadores multinúcleo y multihilo (como x86) y sistemas heterogéneos compuestos de CPU y GPU. Este software se utiliza ampliamente tanto en la industria como en la academia para modelar y validar nuevos diseños de procesador.

El proyecto Multi2Sim se creó y empezó a desarrollar en el seno del Grupo de Arquitecturas Paralelas de la *Universitat Politècnica de València*. Actualmente, este proyecto se mantiene y se sigue desarrollando en el grupo de investigación *Northeastern University Computer Architecture* (NUCAR) en Boston, Massachusetts.

La siguiente lista presenta las características más destacables del simulador:

- Soporte multiprocesador. Multi2Sim soporta las siguientes arquitecturas de procesador CPU: x86, MIPS-32 y ARM.
- Multihilo. Multi2Sim permite la simulación de arquitecturas multihilo, permitiendo además tres paradigmas diferentes para la gestión de los hilos: multihilo simultaneo (SMT, *simultaneous multithreading*), *coarse-grain* y *fine-grain*.
- CPU multinúcleo. El simulador es capaz de modelar procesadores con varios núcleos, los cuales se comunican a través de una red de interconexión.
- Entorno superescalar. El simulador emula el flujo de instrucciones de un procesador superescalar, incluyendo las etapas de *fetch*, *decode*, *issue*, *writeback* y *commit* de un pipeline en el flujo de instrucciones del procesador. Posee también estructuras adicionales como un *reorder buffer* (ROB), colas para *loads* y *stores*, ejecución especulativa, fuera de orden y predicciones de rama.
- Simulación de tarjetas gráficas o GPU. La infraestructura de simulación es capaz de emular también las siguientes arquitecturas de GPU: AMD Evergreen, AMD

Southern Islands, NVIDIA Fermi y NVIDIA Kepler. Además también es posible ejecutar aplicaciones de OpenCL y CUDA con Multi2Sim.

- Jerarquía de memoria. Es posible modelar la jerarquía de memoria con tantos niveles cómo se desee. Las cache se pueden configurar en base a latencia, geometría, unificadas o separadas en datos e instrucciones, privadas o compartidas entre núcleos, etcétera. Asimismo, incluye la implementación del protocolo MOESI para gestionar la coherencia de los datos.
- Red de interconexión. Es posible especificar la topología, el ancho de banda, los algoritmos de encaminamiento, el número de canales virtuales, etcétera.

Las siguientes secciones detallan brevemente cómo se organiza Multi2Sim en directorios y cómo se realiza la compilación y ejecución de la aplicación.

6.1.1. Estructura de Multi2Sim

El código de Multi2Sim puede descargarse desde su repositorio[1] o desde su página Web[2]. Para la realización de este trabajo, se ha utilizado la arquitectura x86 contenida en la versión 4.1 del simulador. En concreto, se ha instrumentado el simulador para modelar con detalle el comportamiento de las memorias cache DWM.

La Figura 6.1 ilustra parte del árbol de directorios de Multi2Sim. Cabe destacar el directorio *src*, el cual contiene código fuente con la implementación de todas las arquitecturas en *arch*, incluyendo x86, y el directorio *mem-system* donde se puede encontrar la implementación de la jerarquía de memoria *on-chip* común a todas las arquitecturas. En este directorio es donde se realiza la instrumentación pertinente para modelar caches con tecnología DWM. A su vez, por cada arquitectura podemos encontrar el código organizado en los directorios *asm*, *emu* y *timing*, los cuales contienen la implementación del repertorio de instrucciones de la arquitectura, así como la simulación funcional y detallada a través del pipeline de la ejecución de las instrucciones de una carga.

En cuanto a las modificaciones realizadas durante el presente trabajo al simulador, éstas pueden encontrarse en el repositorio de github de este proyecto[3]. Concretamente en la rama de entrelazado de dicho proyecto es la que contiene la versión final del software, ya que la rama master se ha utilizado como copia de seguridad de versión original de Multi2Sim y en hugo_dev se ha realizado parte del desarrollo temprano pero ha sido empleado también para realizar pruebas.

6.1.2. Compilación

Para compilar Multi2Sim será necesario ejecutar el siguiente comando:

```
libtoolize && aclocal && autoconf && automake --add-missing && ./configure --
    enable-debug --disable-opengl && make
```

Después de la ejecución del comando se generará el directorio *bin*, que a su vez contendrá el binario *m2s*. Cabe mencionar en la orden las opciones *–enable-debug*, activada para que Multi2Sim genere ficheros para poder realizar una traza de la ejecución y facilitar la depuración del código y la desactivación de *opengl* (*–disable-opengl*) ya que para simular un procesador x86 no es necesario el módulo del simulador capaz de ejecutar

[1] www.github.com/Multi2Sim/multi2sim
[2] www.multi2sim.org/
[3] https://github.com/hugots363/Multi2Sim/tree/entrelazado

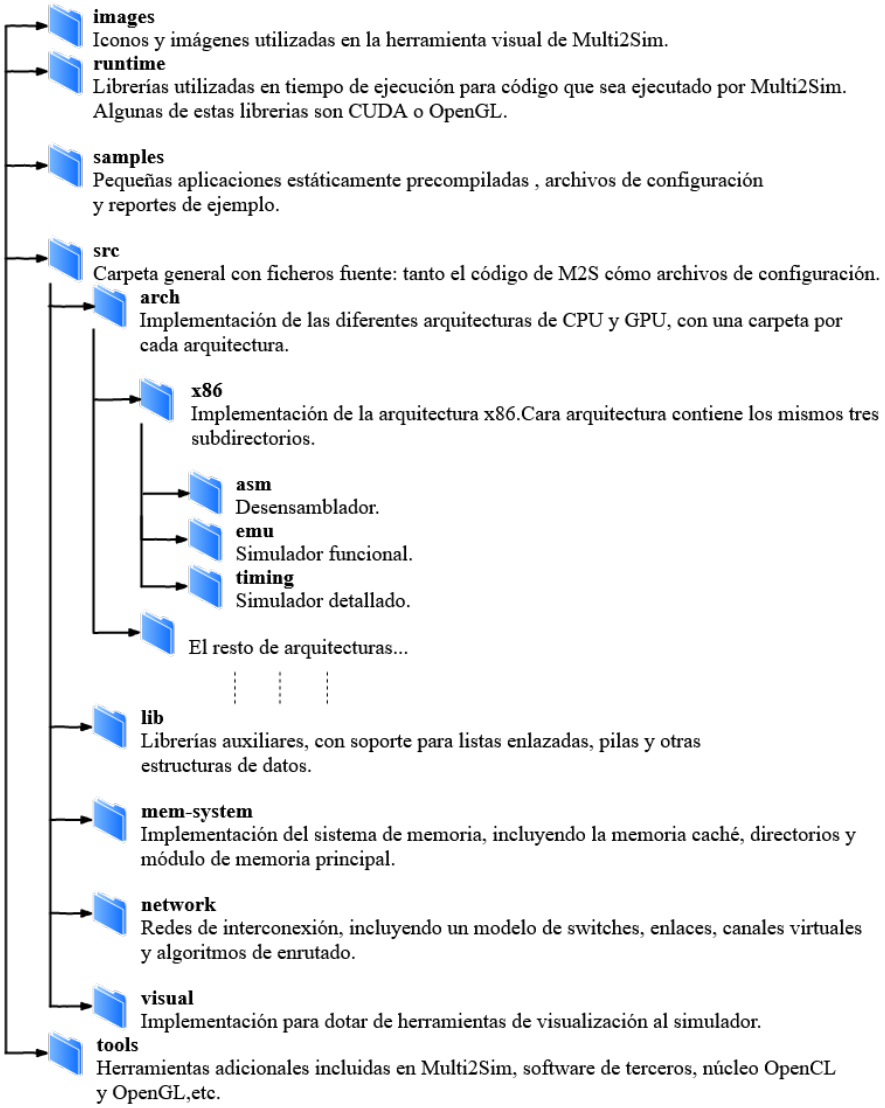


Figura 6.1: Estructura de directorios de Multi2Sim.

OpenCL. Una vez tenemos el ejecutable *m2s* dentro de la carpeta *bin* podemos pasar a la configuración de los parámetros para el lanzamiento de las aplicaciones.

6.1.3. Ejecución

Las simulaciones de Multi2Sim se pueden lanzar desde la línea de comandos. El siguiente ejemplo muestra cómo lanzar una instancia de simulación:

```
1 ./bin/m2s --x86-sim detailed --x86-config cpuconfig --mem-config cacheconf- -
    net-config net_config --ctx-config ctxconfig.benchmark --x86-min-inst-per-
    ctx 500000000 --epoch-length 12000000 --reports-dir carpeta-benchmark-
    reportes
```

Como parámetros de la orden, encontramos *–x86-sim detailed*, el cual indica al simulador el lanzamiento de una simulación detallada del procesador x86. Los siguientes cuatro parámetros corresponden respectivamente a ficheros de entrada con detalles de configuración de la CPU a modelar, la jerarquía de memoria, las redes y la carga de trabajo así como sus respectivos parámetros de entrada (véase la Sección 6.2 para más detalles).

Los parámetros *–x86-min-inst-per-ctx* y *–epoch-length* corresponden al número de instrucciones que debe ejecutar el procesador y a los tramos en los que se imprimen resultados parciales de la ejecución. Finalmente, *–reports-dir* corresponde al directorio donde se guardarán los resultados volcados por Multi2Sim.

6.1.4. DRAMSim

Aunque Multi2Sim se encarga de simular el comportamiento de la memoria principal, con el objetivo de dotar al simulador de un mayor detalle en el modelado de este componente, se puede sustituir el modelo de memoria principal original por un módulo alternativo que se encargue de modelar el comportamiento de este componente y suministrar la información pertinente a Multi2Sim. En este sentido, se ha elegido para la integración el módulo DRAMSim. El software DRAMSim se ha obtenido a partir de su repositorio[4] y se ha integrado dentro del directorio raíz de Multi2Sim. Se refiere al lector a [44] para más detalles acerca de este simulador.

6.2 SPEC CPU 2006

The Standard Performance Evaluation Corporation (SPEC)[5] es una organización en cuyo consorcio se encuentran empresas informáticas de ámbito global como AMD, Dell o Intel, entre otras, y cuyo objetivo es el de desarrollar, mantener y promocionar herramientas para la evaluación del rendimiento y la eficiencia energética de los nuevos equipos informáticos. De esta manera, SPEC asegura un estándar común para la medición de prestaciones a nivel industrial.

Entre las *suites* de aplicaciones que ofrece SPEC, en este trabajo se hace uso de SPEC CPU 2006. Esta suite está formada por un conjunto de 29 aplicaciones (a partir de ahora *benchmarks*) que realizan un uso intensivo de las estructuras principales de un procesador, entre ellas el subsistema de memoria, con el fin de evaluar el rendimiento del procesador [36]. Los benchmarks de SPEC CPU 2006 se pueden dividir en dos grandes grupos: benchmarks que utilizan principalmente números enteros (CINT2006) y benchmarks que utilizan principalmente aritmética de coma flotante (CFP2006). Todos los benchmarks se distribuyen en código fuente, por lo que para utilizarlos conjuntamente con Multi2Sim ha sido necesario compilarlos para la arquitectura x86. Se refiere al lector al Apéndice 6.2.1 para una descripción de cada benchmark.

Finalmente, cabe destacar que actualmente existe una versión de SPEC CPU más reciente del año 2017. Sin embargo, en este trabajo se ha decidido utilizar la versión del año 2006 debido a que el estrés que ejercen los benchmarks de 2006 sobre el subsistema de memoria es mayor frente a los benchmarks de la versión de 2017 [45].

6.2.1. Descripción de Benchmarks SPEC CPU 2006

En la siguiente lista se desglosa una breve descripción de cada benchmark, así como su área de aplicación, el lenguaje en que está escrito y si pertenece al grupo de aritmética entera o de coma flotante [36].

- 400.perlbench. Área de aplicación: lenguaje de programación. Grupo: aritmética entera. Escrito en ANSI C, se trata de una versión reducida del lenguaje de programa-

[4] www.github.com/umd-memsys/DRAMSim2/
[5] https://www.spec.org/

ción Perl derivada de la versión 5.8.7. Su carga de trabajo es la ejecución de los programas *SpamAssasin*, *MHonARc* y *Specdiff*, escritos en Perl.

- 401.bzip2. Área de aplicación: compresión de archivos. Grupo: aritmética entera. Se trata de una versión modificada de la herramienta de compresión/decompresión de archivos bzip2 escrita en ANSI C. Fue modificado de manera que toda la compresión/decompresión se realice en memoria, para facilitar el análisis de la CPU y su subsistema de memoria. Su carga de trabajo se compone de dos imágenes en JPEG, el binario de un programa, el código fuente de otro programa en un archivo *tar*, un archivo HTML y un archivo combinado de mayor dificultad de compresión.

- 403.gcc. Área de aplicación: compilación del lenguaje C. Grupo: aritmética entera. Se trata de un compilador para lenguaje C que traduce a lenguaje máquina y optimiza código para el procesador AMD Opteron. Está configurado para realizar mucho trabajo tratando de optimizar el código, con el objetivo de generar un mayor uso de memoria. Su carga de trabajo se compone de 9 programas escritos en C, que compilaremos usando esta versión de gcc.

- 429.mcf. Área de aplicación: optimización de rutas mediante combinatoria. Grupo: aritmética entera. Escrito en ANSI C, es un programa cuya finalidad es el cálculo y optimización de rutas de transporte público. Se genera una ruta optimizada utilizando los horarios y los trayectos de los transportes disponibles. El programa se basa en MCF y su carga de trabajo son una serie de tablas de transportes y horarios, a partir de las que calcula las rutas.

- 445.gobmk. Área de aplicación: inteligencia artificial, juegos de estrategia. Grupo: aritmética entera. Se trata de un programa escrito en C nativo, pensado para jugar al Go[6]. Partiendo de un archivo *.sgf* (formato común para la representación de partidas de Go) calcula la secuencia de movimientos más óptima.

- 456.hmmer. Área de aplicación: inteligencia artificial, búsqueda de secuencias genómicas. Grupo: aritmética entera. Escrito en C, este programa utiliza modelos ocultos de Markov (HMMs), para buscar coincidencias en las cadenas genéticas presentes en la base de datos, volcando los resultados (*matchings*) en un fichero. Como carga de trabajo utiliza una base de datos y un fichero compuesto por modelos de Markov ocultos, a partir de los cuales se buscan las coincidencias.

- 458.sjeng. Área de aplicación: inteligencia artificial, búsqueda en árbol, reconocimiento de patrones. Grupo: aritmética entera. Escrito en ANSI C, es un programa que mediante inteligencia artificial calcula las mejores jugadas de diversas variantes del ajedrez, a partir de posiciones iniciales dadas. El programa realiza una búsqueda en árbol de las jugadas más óptimas, descartando las ramas con peores jugadas para optimizar la búsqueda. Como carga de trabajo utiliza un fichero de texto de tipo FEN con varias posiciones iniciales, siendo este un estándar de representación utilizado en el mundo del ajedrez. El programa devuelve un fichero con los movimientos óptimos y una serie de parámetros de valoración, así como su profundidad en el árbol.

- 462.libquantum. Área de aplicación: computación cuántica, simulación. Grupo: aritmética entera. Está escrito en ISO/IEC 9899:1999 (C). Este benchmark simula un computador cuántico que utiliza el algoritmo descubierto por Peter Shor para factorizar números en tiempo polinomial. El algoritmo es dependiente de un ordenador cuántico que simula Libquantum, proporcionando herramientas para simular

[6]Go es un juego de mesa de estrategia de origen asiático con 2^{10} posiciones posibles estimadas.

registros cuánticos y algunas puertas lógicas elementales. El programa recibe un número como parámetro y devuelve unos registros del proceso de factorización del número mientras realiza el proceso.

- 471.omnetpp. Área de aplicación: simulación de eventos, redes. Grupo: aritmética entera. Simula una gran red Ethernet (Campus *backbone*) utilizando OMNeT++. La red representada está disponible de forma pública y comprende cerca de 8000 hosts y 900 switches y hubs, tanto Fast Ethernet como Gigabit Ethernet, en modos *halfy full duplex*. Como carga de trabajo recibe un fichero en lenguaje de descripción de redes de OMNeT++ (tipo *.ned*) que contiene la topología completa de la red. El programa, al igual que el simulador original, está escrito en C++.

- 483.xalancbmk. Área de aplicación: transformación de documentos XML. Grupo: aritmética entera. Es una versión modificada de XalanC++ (escrita en Fortran 77), un procesador XSLT que sigue las normas del W3C (*World Wide Web Consortium*) para la transformación XSL, convirtiendo documentos en formato XML a HTML, texto plano u otros tipos de ficheros XML. Sus datos de entrada son un documento XML y una hoja de estilos XSL (formato específico para XML). Devuelve como resultado de su ejecución un documento HTML generado a partir del XML inicial.

- 410.bwaves. Área de aplicación: dinámica de fluidos, simulación. Grupo: aritmética de coma flotante. Este benchmark, escrito en Fortran 77, realiza una simulación numérica de ondas de choque en un fluido laminar tridimensional. Para crear la simulación se utiliza el algoritmo BiCGstab, utilizado para resolver sistemas de ecuaciones lineales no simétricas de forma iterativa. Como entrada recibe parámetros numéricos como el tamaño de la malla o la cantidad de instantes de tiempo medidos.

- 416.gamess. Área de aplicación: química cuántica. Grupo: aritmética de coma flotante. Gamess es una herramienta de cálculos de química cuántica escrita en Fortran. El programa es capaz de realizar cálculos de moléculas simples como el cobre, de compuestos como el agua e incluso sobre moléculas orgánicas como la citosina. Para suministrarle la información de entrada posee sus propios ficheros de configuración, *input.txt*, *intro.txt* y *prog.txt*, que también servirán como salida donde volcar los resultados.

- 433.milc. Área de aplicación: física, cromodinámica cuántica. Grupo: aritmética de coma flotante. Este benchmark, escrito en C, simula un campo de Gauge. Esta simulación sirve para realizar experimentos sobre la teoría del calibre de celosía y suele requerir de computadores paralelos de tipo MIMD (*Multiple Instruction, Multiple Data*), aunque en este programa concreto se utiliza para medir el rendimiento en un solo núcleo.

- 434.zeusmp. Área de aplicación: física, magneto-hidrodinámica. Grupo: aritmética de coma flotante. Basado en ZEUS-MP, un código de fluido-dinámica computacional para la simulación de fenómenos astrofísicos. Resuelve ecuaciones de hidrodinámica y magneto-hidrodinámica, incluyendo campos gravitacionales. Escrito en Fortran 77, recibe como carga de trabajo el fichero *zmp_inp* con datos de carácter físico y simula una onda expansiva tridimensional.

- 435.gromacs. Área de aplicación: química, dinámicas molecular. Grupo: aritmética de coma flotante. Escrito en una combinación de C y Fortran, se trata de una versión reducida de GROMACS, un software de simulación de dinámicas moleculares a partir de ecuaciones newtonianas de movimiento. El programa recibe el fichero

gromacs.tpr y simula a partir de los datos del fichero el comportamiento de la proteína lisozima en una solución de suero salino.

- 436.cactusADM. Área de aplicación: física, relatividad general. Grupo: aritmética de coma flotante. Escrito en Fortran90 y ANSI C, es una combinación de Cactus y BenchADM. Cactus es un entorno de resolución de problemas de código abierto y ADM es el núcleo común de muchas aplicaciones de computo de relatividad general. Este benchmark resuelve las ecuaciones de evolución de Eisntein que describen la curvatura del espacio-tiempo en base a su contenido en masa y energía.

- 437.leslie3d. Área de aplicación: física, dinámica de fluidos. Grupo: aritmética de coma flotante. Deriva de LESlie3d, un simulador de dinámica de fluidos utilizado en procesos acústicos, de combustión y mezclas. Escrito en Fortran90, este programa simula corrientes de flujo en un proceso de combustión.

- 444.namd. Área de aplicación: biología, simulación, dinámica molecular. Grupo: aritmética de coma flotante. Este benchmark es una versión simplificada de NAMD pensada para ejecutarse en una CPU mononúcleo, desarrollado en C++, programa utilizado para simular grandes sistemas biomoleculares. Concretamente, la carga de trabajo de referencia simula un complejo sistema biológico de proteínas y devuelve como salida varias sumas que sirven como comprobación de los cálculos realizados.

- 447.dealII. Área de aplicación: ecuaciones diferenciales. Grupo: aritmética de coma flotante. Este programa, escrito en C++, utiliza la librería dealII para desarrollar modelos algorítmicos finitos. Con esta carga de trabajo se resuelve una ecuación de tipo Helmholtz, que es un tipo de ecuación muy utilizada para resolver problemas de física electromagnética y de campos estacionarios.

- 450.soplex. Área de aplicación: matemáticas, ecuaciones lineales. Grupo: aritmética de coma flotante. Soplex está basado en el software de resolución de ecuaciones lineales *Soplex*, que es una implementación del algoritmo *Simplex* para la resolución de ecuaciones. Escrito en ANSI C++, con la carga de trabajo dada resuelve un sistema de inecuaciones de n filas y m columnas, utilizando factorización. La ejecución devuelve el conjunto solución o indica que no lo hay en caso de que no exista uno.

- 453.povray. Área de aplicación: simulación, mecánica estructural. Grupo: aritmética de coma flotante. Este benchmark utiliza POV-ray, que es un simulador del trazado implementado en ISO C++ de los rayos de luz en una imagen de carácter tridimensional. Para realizar esta simulación, los rayos generados van desde el observador hacia los objetos simulados y cuando intersectan con estos se realiza el cálculo de color e iluminación en ese punto, además de la refracción y reflexión producida. Con la carga de referencia se simula un tablero de ajedrez con figuras que utilizan todos los objetos geométricos que posee POV-ray y se devuelve esta imagen con la simulación lumínica modelada.

- 454.calculix. Área de aplicación: mecánica estructural. Grupo: aritmética de coma flotante. Escrito en Fortran y C, se trata de un programa para realizar cálculos sobre estructuras tridimensionales como puentes o edificios y la resistencia de las estructuras a fenómenos como terremotos o corrimiento de tierras. La carga de trabajo de esta aplicación simula la deformación de un disco compresor por el efecto de una fuerza centrífuga. Devuelve un fichero con una serie de variables que indican la integridad de las piezas del compresor y su deformación.

- 465.tonto. Área de aplicación: química cuántica, cristalografía. Grupo: aritmética de coma flotante. Escrito en Fortran 95, tonto simula estructuras cristalinas realizando para ello un gran número de integrales. A partir de datos como la estructura de un cristal o datos de refracción, este programa devuelve la función de onda y los datos de difracción de rayos X calculados a partir de los datos de entrada.

- 470.lbm. Área de aplicación: física, dinámica de fluidos. Grupo: aritmética de coma flotante. Lbm, que proviene de Método de Laticce-Boltzmann, es una aplicación que simula el comportamiento de fluidos incompresibles utilizando dicho método. Escrito en ANSI C, el programa genera un vector de velocidades en tres dimensiones a partir de los datos físicos del fluido y diversos parámetros de carácter físico.

- 481.wrf. Área de aplicación: previsión meteorológica. Grupo: aritmética de coma flotante. Desarrollado en Fortran 90 y C, el programa fue diseñado para predecir el tiempo a partir de modelos tridimensionales variables y para investigación sobre la dinámica de fenómenos atmosféricos. Con la carga por defecto se realiza la predicción de temperaturas durante todo un día, en franjas horarias de tres horas en una superficie circular de 10 km de diámetro.

- 482.sphinx3. Área de aplicación: reconocimiento del habla. Grupo: aritmética de coma flotante. Implementado en C, este benchmark deriva del software para reconocimiento del habla *Sphinx-3* desarrollado por la Universidad de Carnegie Mellon. A partir de archivos de audio RAW de la base de datos CMU o AN4, grabada por la Carnegie Mellon University en 1991, sphinx3 genera un fichero ASCII con la transcripción del audio.

6.3 GDB

A la hora de instrumentar aplicaciones, especialmente aplicaciones complejas como Multi2Sim, resulta conveniente disponer de herramientas que permitan depurar y verificar modificaciones en el código. Para este tipo de propósito contamos con los depuradores o *debuggers*. Debido a que en el entorno de experimentación del presente trabajo contamos con un sistema operativo Linux y el lenguaje en el que está escrito el simulador es C, el depurador elegido ha sido el Depurador del Proyecto GNU (GDB).

GDB es un depurador multi-lenguaje con una variedad de herramientas muy amplias [46]. Algunas de las herramientas más utilizadas en el proyecto han sido la función de rastreo de fallos en memoria principal, los puntos de ruptura en el código, la ejecución controlada a nivel de instrucción y la traducción de código C a ensamblador. Los puntos de ruptura han resultado especialmente útiles, puesto que han permitido parar y conocer el estado de las variables en puntos concretos de la ejecución sin tener que mostrarlo por salida estándar mediante la instrumentación adicional del código.

6.4 Infraestructura de Simulación

Para el correcto desarrollo de este trabajo, resulta indispensable explotar una infraestructura de simulación capaz de encolar gran cantidad de experimentos, además de contar con un servidor con varios núcleos para paralelizar lanzamientos y reducir el tiempo de espera entre simulaciones. Los elementos descritos a continuación tienen como finalidad la consecución de estos objetivos.

6.4.1. Arquitectura del Servidor Eri1

Todos los experimentos han sido ejecutados en el servidor *Eri1*. Este servidor es propiedad del Grupo de Arquitecturas Paralelas. La máquina cuenta con un sistema operativo Fedora Linux, concretamente la versión Cambridge, y está compuesto por 18 nodos en formato *blade*. La Tabla 6.1 muestra las características principales de cada nodo.

Procesador	2 hexacore Intel Xeon a 2.4 GHz
Memoria RAM	48 GB DDR3 a 1333 MHz
Disco Duro	SATA de 256 GB

Tabla 6.1: Características principales de los nodos de Eri1.

Teniendo en cuenta lo expuesto en tabla anterior, la capacidad de cómputo total es de 216 núcleos y la cantidad total de memoria RAM de 854 GB.

Puesto que Eri1 es un servidor compartido, resulta necesario gestionar una cola de trabajos. El sistema de colas presente en el servidor es *Condor*, descrito en la siguiente sección. A la hora de trabajar con Eri1, accedemos vía SSH al *frontend* del servidor, desde donde se lanzan mediante los comandos de *Condor* los conjuntos de cargas de trabajo pertinentes.

6.4.2. HTCondor

HTCondor es un software de código abierto de alto rendimiento para la paralelización distribuida de tareas de gran intensidad computacional [47]. HTCondor funciona para una amplia variedad de sistemas operativos, entre ellos Linux. HTCondor puede integrar tanto recursos dedicados (clusters montados en rack) como máquinas de escritorio no dedicadas (*cycle scavenging*) en un solo entorno informático. A partir de la versión 7.1.1, HTCondor puede hibernar y despertar máquinas basadas en políticas especificadas por el usuario, lo que lo convierte en un recurso muy valioso para la gestión responsable de servidores.

CAPÍTULO 7
Resultados Experimentales y Discusión

Este capítulo muestra y discute los resultados experimentales. En primer lugar se abordan las características principales del procesador modelado, para a continuación presentar resultados de rendimiento como es el caso de la distribución de la latencia de acceso, los fallos por kilo-instrucción, los ciclos de parada del ROB y el speedup de la propuesta de cache DWM frente a un diseño convencional SRAM.

7.1 Arquitectura de los Procesadores a Simular

En este trabajo se toma en consideración dos jerarquías de memoria cache on-chip diferenciadas. La primera jerarquía se referirá como convencional, y se tomará en consideración tanto para la propuesta convencional de diseño consistente en caches SRAM como para la propuesta del estado-del-arte TapeC. Esta jerarquía se asocia a un sólo núcleo[1] y consta de una cache L1 con arquitectura Harvard, caches L2 y L3 unificadas y una memoria principal (MP). La Figura 7.1 muestra un diagrama con la estructura comentada.

La segunda jerarquía de memoria contiene la cache DWM propuesta en el presente trabajo, y fusionará por tanto la cache de datos L1 y la cache L2 en un sólo nivel con una capacidad total siendo la suma de las dos caches anteriores. El resto de la jerarquía se mantiene como en el caso anterior, quedando como se aprecia en la Figura 7.2.

A continuación se expone de manera sucinta los parámetros principales del simulador y comunes a todos los experimentos. En caso de que algún parámetro cambie para algún experimento concreto, se mencionará en la sección correspondiente. La Tabla 7.1 detalla los parámetros de la jerarquía de memoria correspondientes al fichero *–mem-config* suministrado durante la simulación (véase la Sección 6.1.3). Las modificaciones de diseño se realizan exclusivamente en la cache de datos L1, por lo que los parámetros del resto de niveles de cache se mantienen constantes.

La Tabla 7.2 muestra los valores de la arquitectura x86 que se indican en el fichero *–x86-config*. Por cada benchmark, se ha establecido un *fast-forward* constante de 500 millones de instrucciones, siendo este número un valor habitual en multitud de trabajos de investigación que hacen uso de SPEC CPU 2006, para posteriormente recoger estadísticas de ejecución de los siguientes 500 millones de instrucciones. *RfInt* y *RfFp* hacen referencia a los bancos de registros para aritmética entera y coma flotante.

[1] Por razones de simplicidad a la hora de simular la arquitectura y debido a que el trabajo se centra en la cache de datos L1, tan sólo se modela un único núcleo.

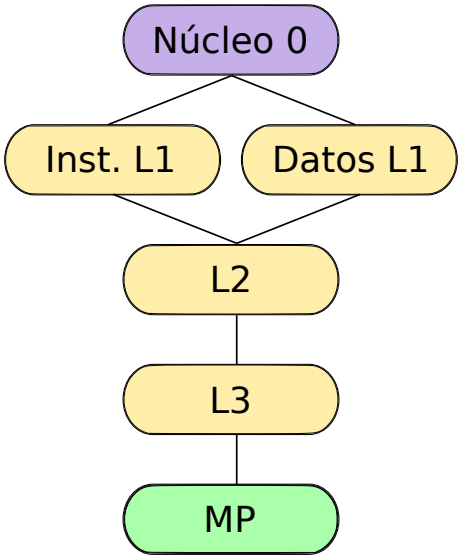


Figura 7.1: Jerarquía de cache convencional utilizada tanto para el diseño convencional SRAM como para TapeC.

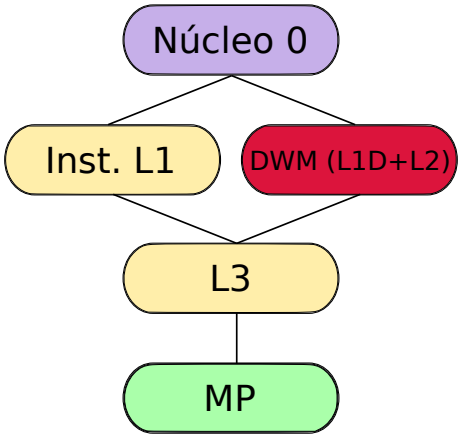


Figura 7.2: Jerarquía de cache propuesta. La cache coloreada en rojo supone la fusión de las caches de datos L1 y L2 originales.

7.2 Distribución de Latencia de Acceso

La Figura 7.3 representa la distribución de la latencia de acceso en la jerarquía de cache desde el 45 % hasta el 100 %. Los resultados se muestran para el subsistema de memoria convencional y el propuesto. Recuérdese que el esquema convencional tiene una latencia L1 constante de 2 ciclos, mientras que la cache L1 DWM tiene una latencia variable de 2 a 4 ciclos y elimina la cache L2 original de 5 ciclos.

Como era de esperar, algunas aplicaciones (10 de 29) presentan una distribución de latencia en L1 superior al 98 % en la jerarquía convencional. Sin embargo, algunos otros benchmarks con una localidad temporal relativamente baja, como *bwaves*, *gcc* y *mcf*, presentan un menor número de aciertos en la cache L1. Por otro lado, al combinar las caches L1 y L2 en un único nivel de cache, el enfoque DWM sustituye la penalización de la latencia L2 de 5 ciclos por una latencia de 4 ciclos (como máximo, en función de la operación de desplazamiento requerida) y alivia la presión en la jerarquía de cache ya que filtra un mayor número de accesos al nivel de cache inferior.

Núcleo del procesador	
Política de issue	Fuera de orden
Predictor de saltos	Hybrid gshare/bimodal
Ancho de fetch, issue y commit	4 instructiones/ciclo
Tamaño ROB (entradas)	244
# ALUs enteros/reales	4/4
Jerarquía de memoria convencional on chip	
Todas las caches	64 B/línea, LRU
Cache inst. L1 SRAM privada	32 KB, 4 vías, 2 ciclos
Cache datos L1 SRAM privada	32 KB, 4 vías, 2 ciclos
Cache L2 SRAM privada	128 KB, 8 vías, 5 ciclos
Cache L3 SRAM compartida	8 MB, 16 vías, 20 ciclos
Jerarquía de memoria on chip propuesta	
Todas las caches	64 B/línea, LRU
Cache inst. L1 SRAM privada	32 KB, 4 vías, 2 ciclos
Cache datos L1 DWM privada	160 KB, 5 vías, 2 ciclos + desplaz.
Cache L2 SRAM compartida	8 MB, 16 vías, 20 ciclos
Memoria principal DRAM	120 ciclos

Tabla 7.1: Parámetros principales del procesador modelado donde se detallan las jerarquías de memoria y que corresponden a la mem-configuration (archivo de configuración de memoria) del simulador.

Arquitectura x86	
Fast-forward	500 millones de instrucciones
Warm-up	200 millones de instrucciones
Colas y registros	
Tamaño de RfInt	288
Tamaño de RfFp	288
Tamaño de IqSize	224

Tabla 7.2: Detalles adicionales del procesador modelado y ventana de ejecución de las aplicaciones, correspondiente a la cpu-configuration (archivo de configuración del procesador) del simulador.

En algunas aplicaciones (10 de 29) como *gromacs*, *lbm* y *mcf*, el enfoque DWM reduce el porcentaje de aciertos en la cache L3 (en realidad la cache L2 en la jerarquía propuesta) con respecto al esquema convencional. En otras palabras, la propuesta con DWM obtiene un mayor porcentaje de aciertos en comparación con el porcentaje de aciertos combinado de las caches L1 y L2 del diseño convencional. Esto se debe principalmente a dos razones. En primer lugar, un bloque de datos puede estar presente en las caches L1 y L2 en el enfoque convencional, lo que reduce la capacidad efectiva de almacenamiento con respecto al diseño propuesto. En segundo lugar, DWM tiene un mayor número de conjuntos de cache (es decir, 512 conjuntos) con respecto a la combinación de las caches L1 y L2 convencionales (es decir, $128 + 256 = 384$ conjuntos). Esto significa que DWM reduce los fallos de capacidad respecto al esquema convencional.

Por otro lado, en algunos otros benchmarks como *bwaves* y *gamess*, DWM muestra un mayor número de aciertos en L3. Esto se debe principalmente a que la organización de cache propuesta tiene menos vías con respecto a las caches L1 y L2 convencionales, lo que implica un mayor número de fallos de conflicto.

En general, el 73 % de los aciertos no requieren ninguna operación de desplazamiento para acceder a los datos solicitados en la cache DWM.

7.3 Fallos por Kilo Instrucción (MPKI)

El estudio anterior no muestra explícitamente si el enfoque propuesto reduce el ratio de fallos en la cache de datos L1 con respecto a los ratios combinados de fallos en la cache de datos L1 y la cache L2 del esquema convencional. Para ello, esta sección evalúa los fallos por kilo-instrucción (MPKI por sus siglas en inglés).

La Figura 7.4 ilustra los resultados de MPKI. El diseño convencional distingue entre MPKI en la cache de datos L1 y en la cache L2. Los resultados muestran que DWM mitiga en gran medida el número de fallos en aquellas aplicaciones con valores de MPKI medios y altos. Esto se debe principalmente a que la capacidad de almacenamiento efectiva del enfoque propuesto es mayor en comparación con el esquema convencional con bloques duplicados en la cache de datos L1 y en la cache L2. Este efecto se aprecia notablemente en las aplicaciones limitadas por la memoria, como *astar* y *bwaves*.

Otras aplicaciones con requisitos de memoria mucho menores, como *gobmk* y *perlbench*, muestran un MPKI bastante bajo, pero incluso en estos casos la cache DWM propuesta reduce el número de fallos. Por último, el conjunto de trabajo de unos pocos benchmarks como *GemsFDTD*, *gamess* y *namd* es lo suficientemente pequeño como para estar contenido en su mayor parte en la cache L1 y los valores de MPKI obtenidos en ambos subsistemas de memoria son insignificantes.

En general, la cache DWM propuesta reduce el MPKI medio en un 66 % con respecto al diseño convencional.

7.4 Paradas en el ROB por Accesos a Memoria

Los bloqueos del procesador pueden clasificarse en tres categorías principales en función del origen del bloqueo: error de especulación, *front-end* y *back-end*. Estos efectos pueden llenar todo el ROB (buffer de reordenación o *reorder buffer*) y producir ciclos de bloqueo en esta estructura. A su vez, las paradas del ROB comprometen el rendimiento del sistema, ya que no permite que el procesador reciba nuevas instrucciones hasta que no cuente con espacio libre. En esta sección se evalúa el número de ciclos que el ROB se atasca debido a la latencia de la memoria en los subsistemas de memoria convencional y DWM.

La Figura 7.5 muestra los ciclos de bloqueo del ROB debidos a los accesos a memoria tanto para el diseño convencional como para el enfoque DWM. Al reducir el MPKI (véase la sección anterior), la cache DWM propuesta minimiza significativamente los ciclos de detención del ROB con respecto al esquema convencional. Además, en aquellas aplicaciones en las que DWM obtiene el mismo MPKI que Conv, como *gcc* y *milc*, el enfoque DWM acorta el número de ciclos de bloqueo del ROB, ya que se omite un acceso a la L2 en el subsistema de memoria DWM propuesto, y los ciclos de desplazamiento necesarios en la cache DWM compensan en gran medida la penalización de la L2.

Obsérvese también que en aplicaciones como *h264ref* y *povray*, en las que el MPKI ya es bajo en el enfoque convencional e inexistente en el diseño DWM, el número de paradas de memoria ROB desaparece por completo.

En general, DWM reduce el número medio de ciclos de bloqueo del ROB en un 43,5 % con respecto al enfoque convencional.

7.5 Rendimiento del Sistema

El análisis de los resultados de la distribución de accesos, el MPKI y las paradas del ROB permite entender la mejora en el rendimiento del sistema de la propuesta DWM frente a un diseño convencional. Esta sección cuantifica la mejora del rendimiento.

El rendimiento se mide utilizando el IPC (instrucciones por ciclo). La Figura 7.6 muestra el *speedup* de la propuesta DWM frente al diseño convencional. A efectos de comparación, también se muestra el speedup alcanzado por la TapeC. Recuérdese que la propuesta DWM cuenta con una L1D de 160KB, mientras que en TapeC se contempla una jerarquía de cache convencional, es decir, una cache de datos L1 SRAM de 32KB y una L2 TapeC de 128KB.

Para un análisis más detallado del speedup, se van a clasificar las aplicaciones en tres grupos: aquellas donde el speedup aumenta un 10 % o más, aquellas donde el speedup aumenta pero menos de un 10 % y aquellas en las que se produce un *slowdown* frente al diseño convencional.

El grupo cuyo speedup aumenta más de un 10 % es el más numeroso y se compone de 20 de 29 aplicaciones, entre ellas *bwaves*, *bzip2* y *gromacs*. Las ganancias de IPC tan pronunciadas se explican debido a tres factores. El primero es la disminución significativa de MPKI. En estas aplicaciones se reducen considerablemente los fallos debido a que el tamaño efectivo de la memoria aumenta. Al haber menos fallos, los accesos son más rápidos, lo que aumenta el rendimeinto del sistema. El segundo factor determinante es la disminución de los ciclos de parada del ROB. Todas las aplicaciones de este grupo presentan una reducción significativa respecto a SRAM, lo que implica menos ciclos en que el procesador está ocioso debido a que el ROB está lleno. El tercer factor es la disminución media de la latencia de acceso. Este apartado es el más variable, ya que dependiendo de la aplicación se consigue mayor o menor mejora. A mayor reducción de los tramos con mayor penalización por desplazamiento, más ganancia se obtiene en el rendimiento.

El grupo cuyo aumento de speedup es menor al 10 % se compone de *calculix*, *hmmer*, *povray* y *sjeng*. El razonamiento es similar al expuesto anteriormente, con la diferencia de que estas aplicaciones adolecen de una disminución más moderada del MPKI y de las paradas del ROB. Además, si se atiende a la latencia de acceso se puede observar que esta aumenta en algunos casos, lo cual limita las mejoras en el rendimiento.

En cuanto al último grupo, el de las aplicaciones con una pérdida de rendimiento frente al diseño convenional se encuentran *GemsFDTD*, *gamess*, *h264ref*, *namd* y *sphinx3*. En el caso de estas aplicaciones, tanto los ciclos de parada del ROB como el MPKI se mantiene o aumenta ligeramente. Además, sufren un aumento del tiempo de acceso a memoria, siendo un caso de esto muy marcado *h264ref*, que pasa de un 99 % de accesos con 2 ciclos de penalización a un 50 % de más de 2 ciclos. Esto hace que el rendimiento del sistema se vea afectado.

Todo lo comentado anteriormente conduce a que, en promedio, la propuesta DWM aumente un 10 % el el speedup frente al diseño SRAM. En la mayoría de casos (24 de 29 aplicaciones), las prestaciones mejoran con la propuesta DWM, ahorrando además tanto área como consumo energético.

Por el contrario, en rendimiento de TapeC es similar al diseño SRAM. En algunas aplicaciones el rendimiento aumenta ligeramente (entre un 1 % y un 2 %) y en otras disminuye en torno a los mismos porcentajes. Esto se debe principalmente a que TapeC se centra únicamente en L2. Aquellas aplicaciones con una alta localidad y patrones de acceso predecibles en L2, como es el caso de *bwaves*, *gobmk* y *sjeng*, muestran una ligera mejora

en el rendimiento. Por el contrario, las aplicaciones que exhiben una baja localidad (e.g., *povray* y *h264ref*) afectan al rendimiento.

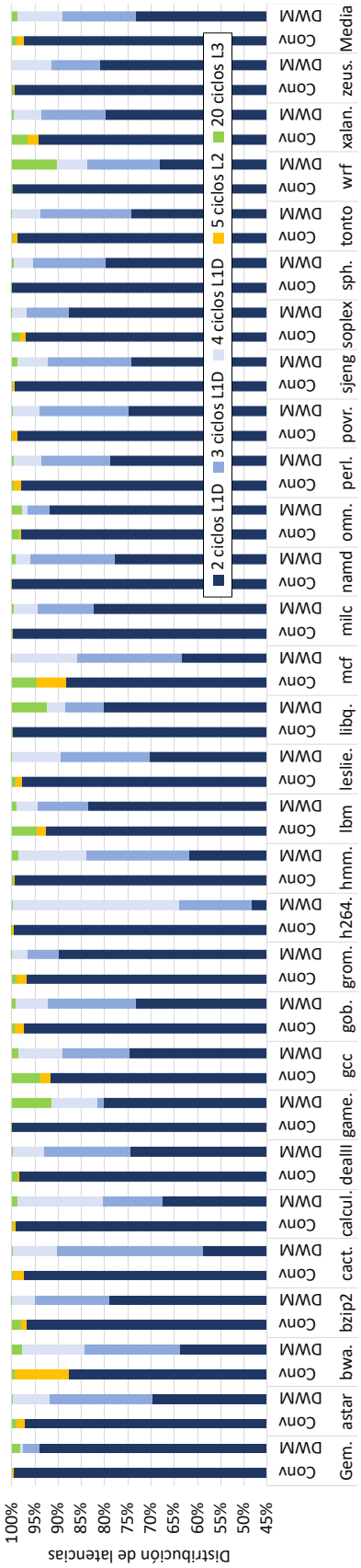


Figura 7.3: Distribución de la latencia de los aciertos en la jerarquía de cache (L1, L2 o L3) para los subsistemas de memoria convencional y el propuesto con tecnología DWM. Los accesos a la memoria principal (no mostrados) representan de media menos del 1 %; sólo 3 de los 29 benchmarks superan el 4 %.

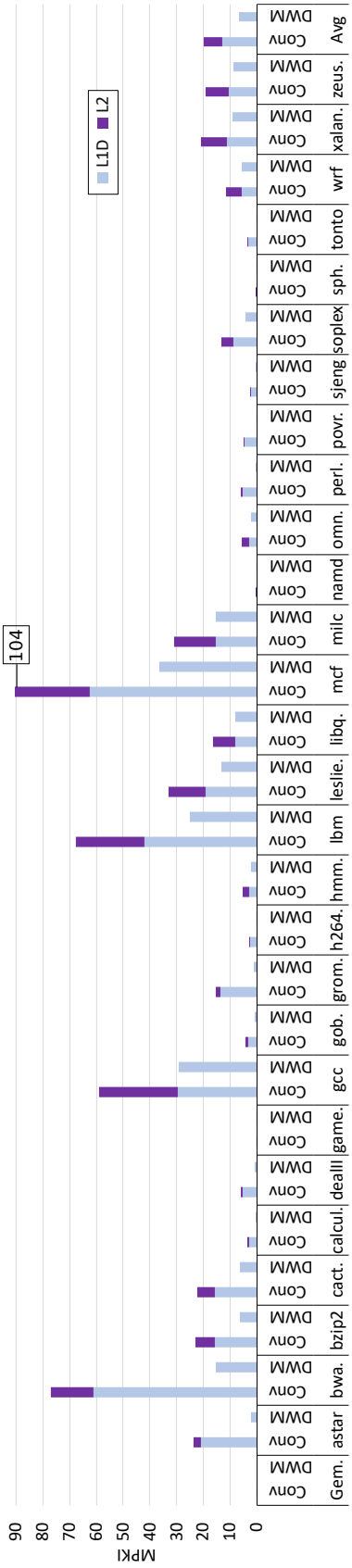


Figura 7.4: Fallos por kilo-instruction (MPKI) en los enfoques convencional y propuesto DWM. El esquema convencional distingue entre MPKI en la cache de datos L1 y en la cache L2.

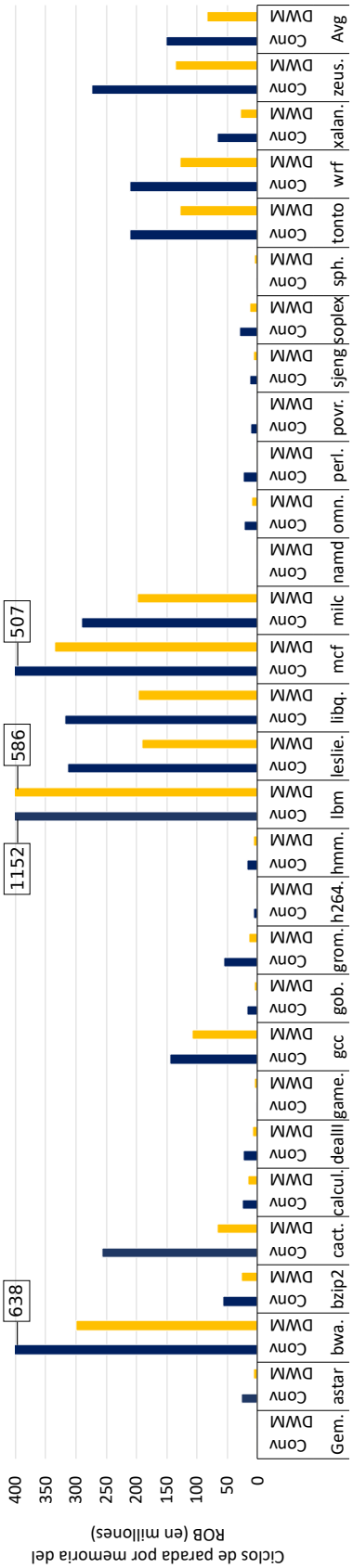


Figura 7.5: Millones de ciclos de parada del ROB para los enfoques convencional y DWM.

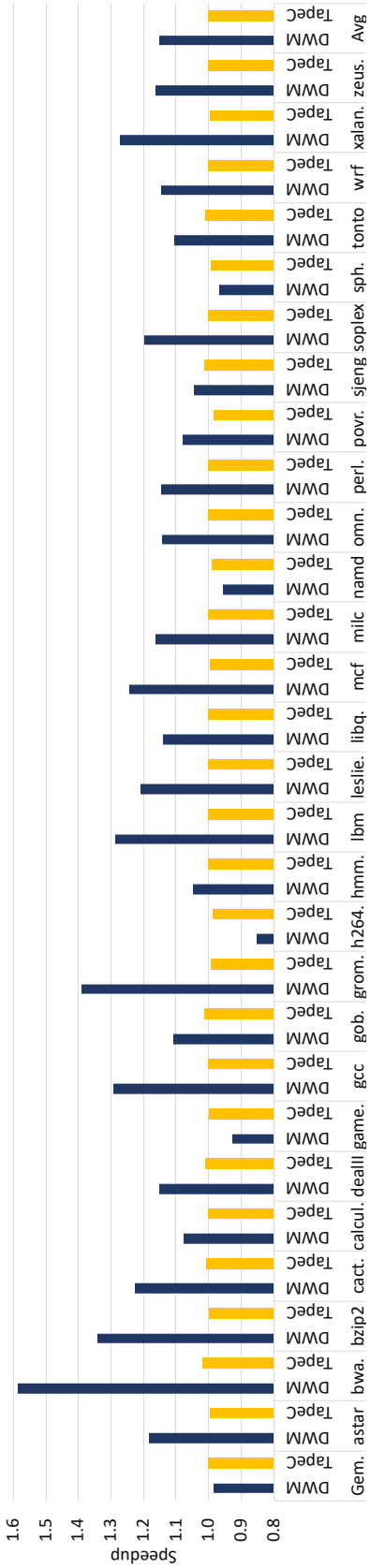


Figura 7.6: Aceleración del diseño de caché DWM propuesto y del TapeCache con respecto al enfoque convencional (cuanto más alto, mejor).

CAPÍTULO 8
Conclusiones y Trabajo Futuro

Este capítulo expone una reflexión acerca de cómo las asignaturas del Máster en Arquitectura de Computadores y Redes han posibilitado la elaboración del trabajo, las conclusiones principales y qué tareas sería interesante realizar en el futuro a partir de los resultados obtenidos en el presente trabajo.

8.1 Relación del Trabajo Desarrollado con los Estudios Cursados

En este trabajo ha sido esencial modificar y adaptar el simulador Multi2Sim para poder emular tanto la propuesta DWM como la TapeC utilizada a efectos de comparación. Para ello, el simulador ha tenido que ser modificado, teniendo en cuenta que su código fuente está escrito en C. El estudio de C en Redes para Aplicaciones de Control (RAC) me ha servido para afianzar mis conocimientos en este lenguaje de programación. En cuanto a los conocimientos necesarios sobre arquitectura del computador y jerarquía de memoria cache ha sido especialmente útil la asignatura Procesadores Multinúcleo Modernos: Un Enfoque Práctico Sobre Arquitectura y Prestaciones (ATP). Me ha servido tanto para saber cómo modificar el simulador para emular las características reales del sistema como para conocer y utilizar las métricas que me han permitido evaluar y justificar las mejoras de la propuesta respecto al resto con las que se ha comparado.

Para realizar el tratamiento de los datos masivos que vuelca el simulador, se ha desarrollado una serie de ejecutables en Python para procesarlos. Una vez procesados, se depositaban en csv con los datos ordenados. Este lenguaje se ha tratado también en el Máster, concretamente en Sistemas Basados en Redes Móviles (SRM).

A su vez se han utilizado diversas herramientas nuevas que no habían sido estudiadas en el Master de forma directa como el depurador de código GDB o el conjunto de aplicaciones SPEC CPU 2006. Gracias a las habilidades adquiridas durante el Máster su aprendizaje ha resultado más ágil.

Focalizando en las propuestas teóricas del trabajo, han sido imprescindibles conocimientos en arquitectura y tecnología de computadores, haciendo especial énfasis en conocimientos acerca de la arquitectura x86, jerarquía de memoria, decodificación de instrucciones, etapas del procesador y análisis de prestaciones basado en la comparativa de máquinas reales. Estos conocimientos han sido adquiridos y afianzados principalmente en ATP, ya que esta es la asignatura que trata con mayor profundidad la arquitectura del procesador.

En cuanto al uso de un clúster de computadores, en este trabajo se ha hecho un uso extensivo de un clúster utilizando CondorHTC para la gestión de las colas de trabajo. Para entender en profundidad los clústers y cómo se estructuran, así como profundizar en la generación de scripts de lanzamiento avanzados con CondorHTC, ha resultado muy útil Configuración, Administración y Utilización de Clústers de Computadores (CAC).

En relación a las competencias transversales trabajadas durante el Máster, han sido esenciales para poder aplicarlas en el presente trabajo. Si bien todas ellas han sido aplicadas en mayor o menor medida, las más importantes y que requieren una mención especial por su importancia en este proyecto son:

- **Pensamiento crítico.** Ha sido esencial esta competencia ya que leer artículos y valorarlos ha sido una parte crítica de este trabajo. De los artículos se extraen ideas útiles, sirven para compararse con este trabajo e incluso sirven para ver la estructura de trabajo de otros investigadores.
- **Análisis y resolución de problemas.** El trabajo ha consistido en analizar la propuesta y resolver los problemas que esta planteaba hasta llegar a una solución funcional. La principal dificultad en este sentido ha sido identificar qué parte concreta de la arquitectura del computador era la que generaba los problemas para posteriormente proponer una solución.
- **Comunicación efectiva.** El presente documento supone un ejercicio continuo de transmisión de ideas. Esta transmisión se intenta sustentar en todo momento en una comunicación efectiva para intentar que el contenido sea lo más comprensible posible.
- **Conocimiento de problemas contemporáneos.** Este trabajo ha requerido un análisis exhaustivo del estado-del-arte así como de los problemas que existen actualmente en la jerarquía de memoria cache de los prosadores actuales.
- **Comprensión e integración.** Una vez comprendidos y trabajados los conceptos de este trabajo, ha sido importante la capacidad de síntesis. Se ha intentado a la vez que se resumía llevar la información a un discurso propio generado a partir de la comprensión de los conceptos tratados.

8.2 Conclusiones

Las conclusiones para este trabajo se van a articular en torno a los dos objetivos principales del mismo, que a su vez se analizarán en base a sus hitos. De esta manera, se pueden elaborar de forma detallada las conclusiones a partir de los objetivos y los resultados experimentales.

8.2.1. Implementación de DWM

Este objetivo tiene como primer hito la instrumentación de un simulador ciclo-a-ciclo para modelar memorias basadas en la tecnología DWM. En este caso se ha modelado (ya que M2S por defecto no da soporte más que a SRAM) la propuesta DWM para su posterior evaluación. Se puede afirmar que se ha cumplido el objetivo puesto que todos los resultados experimentales mostrados parten de los datos obtenidos y procesados del simulador. Esto se aprecia no sólo con las gráficas y datos presentes en este trabajo, sino también en el Anexo A (adjunto al trabajo), con los scripts de extracción de datos.

En cuanto a la política de desplazamiento DL, después de comprobar en la literatura que era la política de desplazamiento que más reducía la penalización por desplazamiento, esta se ha integrado en el simulador, dentro tanto de la parte de software correspondiente a DWM como de TapeC. Se ha comprobado además su implementación correcta mediante unas ejecuciones de prueba especialmente desarrolladas con este objetivo.

La propuesta original de conjuntos entrelazados ha sido el siguiente punto de este desarrollo. Este desarrollo ha permitido elegir tanto el número de conjuntos como con qué módulo estaban entrelazados desde los ficheros de configuración del simulador.

Finalmente, el último hito que se a cumplido es el desarrollo de una arquitectura específica DWM, que en este caso ha agrupado L1D y L2, utilizando DL y módulos con conjuntos entrelazados.

8.2.2. Resolución de los Problemas Técnicos de la Propuesta

Se ha implementado en el simulador la arquitectura TapeC a partir de las especificaciones del artículo donde se presenta. Se ha partido de la implementación de DWM previa, para adaptarla a las particularidades de TapeC. Posteriormente se ha comprobado su correcto funcionamiento a partir de ejecuciones de test.

En cuanto a los resultados sobre la comparativa con SRAM, existen tres factores principales que explican el aumento de rendimiento de DWM respecto a SRAM. En primer lugar, cabe destacar la reducción de la latencia de acceso media. Esto a vez conlleva a una reducción del MPKI y de las paradas del ROB. Ambos efectos combinados resultan en una mejora significativa del IPC en la mayoría de las aplicaciones analizadas. Además de este aumento de rendimiento, conviene recordar tanto la ganancia en densidad (2,67 veces más densidad) respecto a SRAM como una disminución de consumo dinámico. Esto convierte a la propuesta basada en DWM en una sustituta viable para la tecnología SRAM, capaz de dar solución a problemas inherentes de esta tecnología.

En cuanto a las mejoras respecto a la propuesta del estado-del-arte TapeC se obtiene asimismo una ganancia significativa de IPC. Esto se debe principalmente a la mayor capacidad efectiva de DWM así como a la propuesta de distribución de conjuntos entrelazada que limita el número de operaciones y distancia a recorrer por el desplazamiento.

8.3 Trabajo Futuro

Dada la naturaleza de este trabajo y el tiempo limitado, existen diversas extensiones del mismo que se podrían explorar partiendo del estado actual de desarrollo de la línea de investigación. Esta sección expone el trabajo futuro a raíz de la conclusión del presente trabajo.

Otra de las propuestas arquitectónicas que se exploraron inicialmente fue el enfoque iso-área (véase la Sección 4.3). Se podría intentar superar el reto de mantener una velocidad de acceso rápida pese al gran tamaño y la sobrecarga de los circuitos periféricos.

El consumo energético es un aspecto de diseño fundamental en los microprocesadores actuales. En este sentido, cabe realizar un estudio que comparase el consumo energético entre caches L1 basadas en DWM y SRAM. A partir de los resultados obtenidos, se podrían mejorar las propuestas originales teniendo en cuenta este aspecto de diseño. Los principales simuladores para obtener el consumo energético con el uso de tecnologías DWM y SRAM son DESTINY [33] y CACTI [35].

Otra extensión prometedora sería realizar un análisis detallado de las posibilidades de la tecnología DWM en procesadores SMT, donde los hilos que comparten un núcleo ejercen una gran presión sobre la cache privada de L1 y pueden *molestarse* entre ellos, reemplazándose continuamente bloques de cache entre ellos y afectando a las prestaciones del sistema. En este sentido, la densidad de DWM podría ser crucial para aumentar la capacidad de la cache por unidad de superficie y minimizar estos efectos, otorgando una porción de cache a cada hilo de manera estática o dinámica según los requerimientos de las aplicaciones.

Finalmente, también resultaría de interés evaluar las presentes propuestas en otros conjuntos de aplicaciones científicas más allá de SPEC CPU 2006. Por ejemplo, podrían considerarse las aplicaciones de SPEC CPU 2017.

Bibliografía

[1] S. S. P. Parkin, M. Hayashi, and L. Thomas, "Magnetic Domain-Wall Racetrack Memory," *Science*, vol. 320, no. 5873, pp. 190–194, 2008.

[2] S. University, "Excerpts from A Conversation with Gordon Moore: Moore's Law," http://large.stanford.edu/courses/2012/ph250/lee1/docs/Excepts_A_Conversation_with_Gordon_Moore.pdf, 2012.

[3] B. Hall, P. Bergner, A. S. Housfater, M. Kandasamy, T. Magno, A. Mericas, S. Munroe, M. Oliveira, B. Schmidt, W. Schmidt, B. K. Smith, J. Wang, S. Warrier, and D. Wendt, *Performance Optimization and Tuning Techniques for IBM Power Systems Processors Including IBM POWER8*. IBM Redbooks, 2015.

[4] R. Venkatesan, V. J. Kozhikkottu, M. Sharad, C. Augustine, A. Raychowdhury, K. Roy, and A. Raghunathan, "Cache Design with Domain Wall Memory," *IEEE Transactions on Computers*, vol. 65, no. 4, pp. 1010–1024, 2016.

[5] Z. Sun, X. Bi, W. Wu, S. Yoo, and H. Li, "Array Organization and Data Management Exploration in Racetrack Memory," *IEEE Transactions on Computers*, vol. 65, no. 4, pp. 1041–1054, 2016.

[6] A. A. Khan, F. Hameed, R. Bläsing, S. S. P. Parkin, and J. Castrillon, "ShiftsReduce: Minimizing Shifts in Racetrack Memory 4.0," *ACM Transactions on Architecture and Code Optimization*, vol. 16, no. 4, 2019.

[7] J. Meng and K. Skadron, "Avoiding Cache Thrashing due to Private Data Placement in Last-Level Cache for Manycore Scaling," in *Proceedings of the IEEE International Conference on Computer Design*, 2009, pp. 282–288.

[8] H. Tárrega Sánchez, "Diseño de caches l1 utilizando la tecnología emergente domain wall memory," Ph.D. dissertation, 2020.

[9] R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli, "Multi2Sim: A Simulation Framework for CPU-GPU Computing," in *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*, 2012, pp. 335–344.

[10] M. Lapedus, "5nm vs. 3nm," https://semiengineering.com/5nm-vs-3nm/, 2019.

[11] R. E. Matick and S. E. Schuster, "Logic-Based EDRAM: Origins and Rationale for Use," *IBM Journal of Research and Development*, vol. 49, no. 1, pp. 145–165, 2005.

[12] J. Barth, D. Plass, E. Nelson, C. Hwang, G. Fredeman, M. A. Sperling, A. Mathews, T. Kirihata, W. R. Reohr, K. Nair, and N. Caon, "A 45 nm SOI Embedded DRAM Macro for the POWER™ Processor 32 MByte On-Chip L3 Cache," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 1, pp. 64–75, 2011.

[13] N. Kurd, M. Chowdhury, E. Burton, T. P. Thomas, C. Mozak, B. Boswell, P. Mosalikanti, M. Neidengard, A. Deval, A. Khanna, N. Chowdhury, R. Rajwar, T. M. Wilson, and R. Kumar, "Haswell: A Family of IA 22 nm Processors," *IEEE Journal of Solid-State Circuits*, vol. 50, no. 1, pp. 49–58, 2015.

[14] X. Liang, R. Canal, G. Wei, and D. Brooks, "Process Variation Tolerant 3T1D-Based Cache Architectures," in *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, 2007, pp. 15–26.

[15] A. Valero, J. Sahuquillo, S. Petit, V. Lorente, R. Canal, P. López, and J. Duato, "An Hybrid eDRAM/SRAM Macrocell to Implement First-Level Data Caches," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2009, pp. 213–221.

[16] A. Valero, S. Petit, J. Sahuquillo, P. López, and J. Duato, "Design, Performance, and Energy Consumption of eDRAM/SRAM Macrocells for L1 Data Caches," *IEEE Transactions on Computers*, vol. 61, no. 9, pp. 1231–1242, 2012.

[17] A. Valero, J. Sahuquillo, V. Lorente, S. Petit, P. Lopez, and J. Duato, "Impact on Performance and Energy of the Retention Time and Processor Frequency in L1 Macrocell-Based Data Caches," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 6, pp. 1108–1117, 2012.

[18] M. Zabihi, Z. I. Chowdhury, Z. Zhao, U. R. Karpuzcu, J. Wang, and S. S. Sapatnekar, "In-Memory Processing on the Spintronic CRAM: From Hardware Design to Application Mapping," *IEEE Transactions on Computers*, vol. 68, no. 8, pp. 1159–1173, 2019.

[19] C. W. Smullen, V. Mohan, A. Nigam, S. Gurumurthi, and M. R. Stan, "Relaxing Non-Volatility for Fast and Energy-Efficient STT-RAM Caches," in *Proceedings of the IEEE 17th International Symposium on High Performance Computer Architecture*, 2011, pp. 50–61.

[20] J. Ahn, S. Yoo, and K. Choi, "DASCA: Dead Write Prediction Assisted STT-RAM Cache Architecture," in *Proceedings of the IEEE 20th International Symposium on High Performance Computer Architecture*, 2014, pp. 25–36.

[21] A. Jog, A. K. Mishra, C. Xu, Y. Xie, V. Narayanan, R. Iyer, and C. R. Das, "Cache Revive: Architecting Volatile STT-RAM Caches for Enhanced Performance in CMPs," in *Proceedings of the Design Automation Conference*, 2012, pp. 243–252.

[22] W. Xu, H. Sun, X. Wang, Y. Chen, and T. Zhang, "Design of Last-Level On-Chip Cache Using Spin-Torque Transfer RAM (STT RAM)," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 3, pp. 483–493, 2011.

[23] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "Energy Reduction for STT-RAM Using Early Write Termination," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design - Digest of Technical Papers*, 2009, pp. 264–268.

[24] K. Kuan and T. Adegbija, "MirrorCache: An Energy-Efficient Relaxed Retention L1 STTRAM Cache," in *Proceedings of the Great Lakes Symposium on VLSI*, 2019, pp. 299–302.

[25] K. Kuan and T. Adegbija, "Energy-Efficient Runtime Adaptable L1 STT-RAM Cache Design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 6, pp. 1328–1339, 2020.

[26] J. Yao, J. Ma, T. Chen, and T. Hu, "An Energy-Efficient Scheme for STT-RAM L1 Cache," in *Proceedings of the IEEE 10th International Conference on High Performance Computing and Communications and the IEEE International Conference on Embedded and Ubiquitous Computing*, 2013, pp. 1345–1350.

[27] M. Imani, S. Patil, and T. Rosing, "Low Power Data-Aware STT-RAM Based Hybrid Cache Architecture," in *Proceedings of the 17th International Symposium on Quality Electronic Design*, 2016, pp. 88–94.

[28] J. Zhang, M. Jung, and M. Kandemir, "FUSE: Fusing STT-MRAM into GPUs to Alleviate Off-Chip Memory Access Overheads," in *Proceedings of the IEEE International Symposium on High Performance Computer Architecture*, 2019, pp. 426–439.

[29] C. Zhang, G. Sun, W. Zhang, F. Mi, H. Li, and W. Zhao, "Quantitative Modeling of Racetrack Memory, A Tradeoff among Area, Performance, and Power," in *Proceedings of the The 20th Asia and South Pacific Design Automation Conference*, 2015, pp. 100–105.

[30] G. Wang, Y. Zhang, B. Zhang, B. Wu, J. Nan, X. Zhang, Z. Zhang, J. Klein, D. Ravelosona, Z. Wang, Y. Zhang, and W. Zhao, "Ultra-Dense Ring-Shaped Racetrack Memory Cache Design," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 1, pp. 215–225, 2019.

[31] Chao Zhang, Guangyu Sun, Weiqi Zhang, Fan Mi, Hai Li, and W. Zhao, "Quantitative Modeling of Racetrack Memory, a Tradeoff among Area, Performance, and Power," in *The 20th Asia and South Pacific Design Automation Conference*, 2015, pp. 100–105.

[32] R. Venkatesan, M. Sharad, K. Roy, and A. Raghunathan, "Dwm-tapestri - an energy efficient all-spin cache using domain wall shift based writes," in *Proceedings of the Design, Automation, and Test in Europe Conference Exhibition*, 2013, pp. 1825–1830.

[33] S. Mittal, R. Wang, and J. Vetter, "DESTINY: A Comprehensive Tool with 3D and Multi-Level Cell Memory Modeling Capability," *Journal of Low Power Electronics and Applications*, vol. 7, no. 3, pp. 1–24, 2017.

[34] S. Motaman, A. S. Iyengar, and S. Ghosh, "Domain wall memory-layout, circuit and synergistic systems," *IEEE Transactions on Nanotechnology*, vol. 14, no. 2, pp. 282–291, 2015.

[35] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi, "CACTI 5.1," *HP Development Company, Palo Alto, CA, USA. Technical Report HPL-2008-20*, 2008.

[36] J. L. Henning, "SPEC CPU2006 Benchmark Descriptions," *ACM SIGARCH newsletter, Computer Architecture News*, vol. 34, no. 4, pp. 1–17, 2006.

[37] Z. Sun, X. Bi, A. K. Jones, and H. Li, "Design Exploration of Racetrack Lower-Level Caches," in *Proceedings of the IEEE/ACM International Symposium on Low Power Electronics and Design*, 2014, pp. 263–266.

[38] F. Chen, Z. Li, W. Kang, W. Zhao, H. Li, and Y. Chen, "Process Variation Aware Data Management for Magnetic Skyrmions Racetrack Memory," in *Proceedings of the 23rd Asia and South Pacific Design Automation Conference*, 2018, pp. 221–226.

[39] A. Colaso, P. Prieto, P. Abad, J. A. Gregorio, and V. Puente, "Architecting Racetrack Memory Preshift through Pattern-Based Prediction Mechanisms," in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium*, 2019, pp. 273–282.

[40] Haifeng Xu, Yong Li, R. Melhem, and A. K. Jones, "Multilane Racetrack Caches: Improving Efficiency Through Compression and Independent Shifting," in *Proceedings of the 20th Asia and South Pacific Design Automation Conference*, 2015, pp. 417–422.

[41] A. Ranjan, S. G. Ramasubramanian, R. Venkatesan, V. Pai, K. Roy, and A. Raghunathan, "DyReCTape: A Dynamically Reconfigurable Cache using Domain Wall Memory Tapes," in *Proceedings of the Design, Automation, and Test in Europe Conference Exhibition*, 2015, pp. 181–186.

[42] H. Xu, Y. Alkabani, R. Melhem, and A. K. Jones, "FusedCache: A Naturally Inclusive, Racetrack Memory, Dual-Level Private Cache," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 2, no. 2, pp. 69–82, 2016.

[43] R. Ubal, J. Sahuquillo, S. Petit, and P. López, "Multi2Sim: A Simulation Framework to Evaluate Multicore-Multithreaded Processors," *Proceedings of the 19th International Symposium on Computer Architecture and High Performance Computing*, pp. 62–68, 2007.

[44] J. Cheng, Z. Ji, and B. Zhang, "An Optimized Method of Memory Simulation Accuracy in Multicore Multithread Processor," in *Proceedings of the International Conference on Automatic Control and Artificial Intelligence*, 2012, pp. 477–480.

[45] A. Limaye and T. Adegbija, "A Workload Characterization of the SPEC CPU2017 Benchmark Suite," in *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software*, 2018, pp. 149–158.

[46] R. M. Stallman, R. Pesch, and S. Shebs, *Debugging with GDB*. Free Software Foundation, 2002.

[47] D. Thain, T. Tannenbaum, and M. Livny, "Distributed Computing in Practice: the Condor Experience," *Wiley Concurrency and Computation: Practice and Experience*, vol. 17, no. 2–4, pp. 323–356, 2005.