Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Diseño de Caches L1 Utilizando la Tecnología Emergente Domain Wall Memory

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Hugo Tárrega Sánchez

Tutor: Vicente Jesús Lorente Garcés

Director Experimental: Alejandro Valero Bresó

Curso 2019-2020

Dedicatoria

A mi abuelo Domingo, que al marcharse me dejó todo su cariño al saber.
A mi abuelo Antonio, que me enseñó mediante el ejemplo la virtud del trabajo duro.
A mi madre, que me ha dado toda su fuerza, sin perderla por el camino.
Sin vosotros, estas líneas no existirían.

Agradecimientos

Quiero mostrar mi gratitud a Julio Sahuquillo, por introducirme en el mundo de la investigación y por ofrecerme las prácticas en el Grupo de Arquitecturas Paralelas que han posibilitado este trabajo. Además, su consejo y guía constantes han sido indispensables para la consecución de este proyecto.

Dar las gracias a mi tutor Vicent Lorente, por suponer un gran apoyo y resolver todas mis dudas de carácter teórico. Espero que estas líneas supongan para él como poco una parte del orgullo que ha supuesto contar con él para ayudarme en mi investigación.

Agradecer también a Alejandro Valero, que desde la distancia ha sido (y es) un tutor enérgico con ganas de abrir nuevos caminos en la investigación. Es gracias a gente como él que el tejido joven investigador de este país avanza, pese a las dificultades económicas.

Finalmente agradecer a todo el Grupo de Arquitecturas paralelas por haberme hecho sentir uno más, comodidad que ha contribuido enormemente a la calidad final de este trabajo. Dentro del GAP, me gustaría hacer también una mención especial a José Puche, que me ha ofrecido todos sus conocimientos sobre Multi2Sim para ayudarme a entender el simulador.

Resumen

Las memorias cache de un microprocesador se implementan habitualmente con tecnología *Static Random-Access Memory* (SRAM) puesto que es la tecnología electrónica más rápida. Sin embargo, las caches SRAM ocupan un área significativa del microprocesador y además consumen una gran cantidad de energía estática por corrientes de fuga, lo cual resulta en un problema de diseño importante, ya que este consumo aumenta a medida que el tamaño del transistor se encoge en sucesivos nodos tecnológicos. En este sentido, algunos procesadores comerciales de IBM e Intel incluyen el uso de tecnologías alternativas de bajo consumo como *embedded Dynamic RAM* (eDRAM) en los últimos niveles de cache como L2 o L3. No obstante, eDRAM requiere operaciones de refresco periódicas sobre los datos y además no es tan rápida como SRAM. Estos inconvenientes impiden que eDRAM se pueda utilizar directamente en el primer nivel (L1) de cache. Por otro lado, las tecnologías magnéticas, como la emergente *Domain Wall Memory* (DWM), están generando un interés creciente porque su consumo estático es nulo, no requieren operaciones de refresco y ofrecen una gran densidad y tiempos de acceso competitivos frente a SRAM. Sin embargo, al almacenar los bits en una cinta magnética, DWM requiere operaciones de desplazamiento de la cinta para alinear los cabezales de acceso con los datos requeridos, lo cual afecta al tiempo de acceso de la cache. Algunos trabajos de investigación recientes han explorado diferentes organizaciones de los datos y políticas de manejo de los cabezales para atenuar este problema, concretamente en caches L2 y L3.

En el presente trabajo se explora el uso de la tecnología DWM en caches de datos L1. Para ello, se implementan y validan distintas políticas de manejo de los cabezales del estado-del-arte sobre L1, cuantificando experimentalmente el impacto de cada una de ellas en base a la cantidad de desplazamientos de los datos a través de las cintas. Además, se propone y valida una nueva organización de los datos en la cache que se ajusta a las características y requerimientos de las caches L1. Para ello, se instrumenta un simulador de procesadores ciclo-a-ciclo y se obtienen resultados experimentales mediante la ejecución de un conjunto representativo de aplicaciones científicas.

Los resultados experimentales muestran que, entre las políticas de gestión de cabezales del estado-del-arte, la política que mejor se ajusta a L1 es *Dynamic Lazy* debido a que disminuye el número de operaciones de desplazamiento así como la distancia máxima de desplazamiento en número de bits. Además, la propuesta de organización de los datos en la cache reduce el número de desplazamientos en un 16 % frente a una organización de datos convencional. Finalmente, también se ha comprobado de manera empírica que existe una relación inversa entre la capacidad de la cache y la penalización por desplazamiento.

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

Resum

Les memòries cau d'un microprocessador s'implementen sovint utilitzant la tecnologia *Static Random-Access Memory* (SRAM), ja que és la tecnologia electrònica més ràpida. Tot i això, les memòries cau SRAM ocupen una àrea significativa del microprocessador i a més consumeixen una gran quantitat d'energia estàtica per corrents de fuga, el que resulta en un problema de disseny important perquè aquest consum augmenta a mesura que la mida del transitor es redueix en successius nodes tecnològics. En aquest sentit, alguns processadors comercials d'IBM i Intel inclouen l'ús de tecnologies alternatives de baix consum com *embedded Dynamic RAM* (eDRAM) en els últims nivells de memòria cau com L2 o L3. Això no obstant, eDRAM requereix operacions de refresc periòdiques sobre les dades i a més no és tan ràpida com SRAM. Aquests inconvenients impossibiliten que eDRAM es puga utilitzar directament en el primer nivell (L1) de la memoria cau. Per altra banda, les tecnologies magnètiques, com l'emergent *Domain Wall Memory* (DWM), estan rebent un creixent interés perquè el seu consum estàtic és nul, no requereixen operacions de refresc i ofereixen una gran densitat i accessos competitius front a SRAM. Tot i això, al enmagatzemar els bits en una cinta magnètica, DWM requereix operacions de desplaçament de la cinta per a alinear els capçals d'accés amb les dades requerides, la qual cosa afecta el temps d'accés a la memòria cau. Alguns treballs d'investigació recents han explorat diferents organitzacions de les dades i polítiques de gestió dels capçals per a reduir aquest problema, particularment en les memòries cau L2 i L3.

En el present treball, s'explora l'ús de la tecnologia DWM en memòries cau de dades L1. Per a fer-ho, s'implementen i validen distintes polítiques de gestió dels capçals del estat-del-art sobre L1, quantificant de manera experimental l'impacte de cadascuna d'elles en base a la quantitat de desplaçaments de les dades a través de les cintes. A més, es proposa i valida una nova organització de les dades en la cache que s'ajusta a les característiques i requeriments de les memòries cau L1. Per a això, s'instrumenta un simulador de processadors cicle-a-cicle i s'obtenen resultats experimentals mitjançant l'execució d'un conjunt representatiu d'aplicacions científiques.

Els resultats experimentals mostren que, entre les polítiques de gestió dels capçals del estat-del-art, la política que millor s'ajusta a L1 és *Dynamic Lazy* pel fet que disminueix el nombre d'operacions de desplaçaments així com la distància màxima de desplaçament en nombre de bits. A més, la proposta d'organització de les dades en la memoria cau redueix el nombre de desplaçaments en un 16% front a una organització de dades convencional. Finalment, també s'ha comprovat de manera empírica que existeix una relació inversa entre la capacitat de la memoria cau i la penalització per desplaçament.

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

Abstract

Microprocessor caches are usually implemented with Static Random-Access Memory (SRAM) technology as it is the fastest electronic technology. However, SRAM caches occupy a significant area of the microprocessor and they also consume a large amount of static energy from leakage currents. This results in a major design problem because this consumption increases as the size of the transistor shrinks in successive technology nodes. In this sense, some commercial processors from IBM and Intel include the use of alternative low-power technologies such as embedded Dynamic RAM (eDRAM) in the latest cache levels such as L2 or L3. However, eDRAM requires periodic data refresh operations and is not as fast as SRAM. These drawbacks prevent eDRAM from being used directly on the first-level (L1) cache. On the other hand, magnetic technologies, such as the emerging Domain Wall Memory (DWM), are receiving increasing attention because their static consumption is zero, they do not require refresh operations, and they offer high density and competitive access times compared to SRAM. However, by storing the bits on a magnetic tape, DWM requires tape shifting operations to align the access heads with the required data, which affects the access time of the cache. Recent research has explored different data organizations and head management policies to mitigate this problem, particularly in L2 and L3 caches.

This work explores the use of DWM technology in L1 data caches. For this purpose, different head policies from the state-of-the-art are implemented and validated for L1, experimentally quantifying the impact of each of them based on the amount of data shift operations along the tapes. Besides, a new data organization for the cache is proposed and validated, which fits the characteristics and requirements of L1 caches. In order to achieve that, a cycle-accurate microprocessor simulator is instrumented and experimental results are obtained through the execution of a set of representative scientific applications.

Experimental results show that, among the state-of-the-art head management policies, the policy that works best on L1 is Dynamic Lazy because it decreases the number of shifting operations as well as the maximum shifting distance in number of bits. In addition, the proposed data organization in the cache reduces the number of shifts by 16% compared to a conventional data organization. Finally, it has also been empirically proven that there is an inverse relation between cache capacity and shift penalty.

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

Índice general

Índice general IX
Índice de figuras XI
Índice de tablas XI

1 Introducción **1**
1.1 Contexto y Motivación . . . 1
1.2 Objetivos . . . 2
1.3 Metodología . . . 3
1.4 Estructura de la Memoria . . . 4
2 Tecnologías de Memoria **5**
2.1 Static Random Access Memory . . . 5
2.2 Embedded Dynamic RAM . . . 5
2.3 Non-Volatile Memory . . . 6
2.3.1 Spin-Transfer Torque RAM . . . 7
2.3.2 Domain Wall Memory . . . 8
2.4 Comparativa . . . 9
3 Implementación de DWM en la Cache de Datos L1 **11**
3.1 Racetrack en Anillo . . . 11
3.1.1 Políticas de Desplazamiento . . . 12
3.2 Estructura de una Cache L1 con Tecnología DWM . . . 15
3.3 Organización Entrelazada . . . 16
4 Entorno Experimental **19**
4.1 Simulador Multi2Sim . . . 19
4.1.1 Estructura de Multi2Sim . . . 20
4.1.2 Compilación . . . 20
4.1.3 Ejecución . . . 21
4.1.4 DRAMSim . . . 22
4.2 SPEC CPU 2006 . . . 22
4.2.1 Descripción de Benchmarks SPEC CPU 2006 . . . 22
4.3 GDB . . . 26
4.4 Infraestructura de Simulación . . . 26
4.4.1 Arquitectura del Servidor Eri1 . . . 27
4.4.2 HTCondor . . . 27
5 Resultados Experimentales y Discusión **29**
5.1 Arquitectura del Procesador y Clasificación de Benchmarks . . . 29
5.2 Políticas de Desplazamiento . . . 32
5.3 Comparativa entre Entrelazado y Mapeado Directo . . . 34
5.3.1 Análisis del Aumento del Tramo 9-16 . . . 35
5.4 Impacto de la Capacidad de la Cache en los Tramos . . . 37
5.4.1 Análisis de la Variabilidad en Tramos Intermedios . . . 39
6 Conclusiones y Trabajo Futuro **41**
6.1 Relación del Trabajo Desarrollado con los Estudios Cursados . . . 41

6.2 Conclusiones 42
6.2.1 Objetivos Técnicos 42
6.2.2 Objetivos Académicos 43
6.3 Trabajo Futuro 43
Bibliografía **45**

Apéndice

A Scripts de Tratamiento de Datos **49**
A.1 createDics.py 49
A.2 generarCondorFile.py 51
A.3 missHitsAmontonadoPercnt.py 52
A.4 missHitsAmontonadoAbsoluto.py 56
A.5 mpkisToCsv.py 59
A.6 porcentajesHitsMissesL1.py 60
A.7 ejecucionAmontonados.py 61

Índice de figuras

2.1 Esquema de una celda de memoria 6T SRAM. 6
2.2 Esquema de una celda de memoria 1T1C eDRAM. 7
2.3 Esquema de una celda de memoria STT-RAM. 7
2.4 Esquema con el desplazamiento de dominios sobre una racetrack. 8
2.5 Vista de dominios sobre una racetrack [1]. 9

3.1 Organización convencional de una memoria DWM de 128 bits. 11
3.2 Ejemplo de gestión de cabezales. 14
3.3 Organización convencional de un banco de memoria de 8 KB. 15
3.4 Memoria cache de 32 KB compuesta por 4 bancos. 16
3.5 Comparativa de organización de conjuntos en racetracks de 16 bits. 16

4.1 Estructura de directorios de Multi2Sim. 21

5.1 Esquema del procesador con un solo núcleo. 29
5.2 IPC de las aplicaciones SPEC CPU 2006. 31
5.3 Comparativa de políticas de gestión de cabezales. 33
5.4 Comparativa entre mapeado directo y entrelazado. 35
5.5 Posición de los cabezales tras acceder al conjunto 11. 36
5.6 Posición de los cabezales tras acceder al conjunto 3. 37
5.7 Comparativa de aumento de capacidad de cache. 38
5.8 Posición de los cabezales tras acceder a la dirección 0x040. 40
5.9 Posición de los cabezales tras acceder a la dirección 0x0C0. 40

Índice de tablas

2.1 Comparativa entre diferentes tecnologías de memoria. 9

3.1 Tabla resumen de las políticas de gestión de cabezales. 15

4.1 Características principales de los nodos de Eri1. 27

5.1 Configuración de la jerarquía de memoria del procesador. 30
5.2 Configuración de la arquitectura del procesador. 30

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

«*Recuerdo la dificultad que tuvimos, al principio, para reemplazar los núcleos magnéticos en las memorias y finalmente tuvimos ventajas de coste y rendimiento. Pero no estaba del todo claro al principio*». Esta cita se atribuye a Gordon Moore en referencia al cambio de paradigma que se originó cuando se reemplazaron las memorias magnéticas presentes en el computador por memorias de estado sólido. Este hecho condujo a una revolución en el rendimiento del computador en la década de los años 60.

A medida que se llega al límite físico de miniaturización del transistor y el consecuente fin de la Ley de Moore, resulta necesario investigar nuevas tecnologías que permitan a la industria de los semiconductores seguir ofreciendo procesadores con un mayor ratio rendimiento/Vatio y capacidad de almacenamiento de acuerdo con la continua demanda de cómputo y memoria por parte de las aplicaciones. La mayoría de transistores en un procesador se utilizan para implementar el subsistema de memoria *on-chip* [2]. Por esta razón, gran parte de la investigación se centra en la propuesta de nuevas tecnologías para implementar el subsistema de memoria del procesador.

El subsistema de memoria del procesador se ha implementando tradicionalmente con la tecnología electrónica *Static Random Access Memory* (SRAM) debido a que se trata de la tecnología de memoria más rápida que existe. Sin embargo, SRAM no está exenta de problemas relacionados con el consumo energético, el área y la fiabilidad del sistema. Con el fin del postulado del escalado de Dennard hacia el año 2005, sucesivas reducciones en el tamaño del transistor conllevan a un aumento de potencia por unidad de superficie. Este problema no sólo aumenta la facturación eléctrica de los servidores y centros de datos, así como una reducción de la duración de la batería de los ordenadores portátiles y *smartphones*, sino que además conduce a un aumento de temperatura que puede afectar a las condiciones de operación del dispositivo y comprometer su fiabilidad.

Desde hace algunos años, los avances tecnológicos han permitido apostar por las tecnologías magnéticas para reemplazar a SRAM. Entre ellas, destaca la reciente tecnología *Domain Wall Memory* (DWM). Al contrario que SRAM, esta tecnología ofrece un consumo estático nulo, un consumo dinámico muy bajo y una densidad de almacenamiento muy elevada. Esto se consigue con una implementación y organización disruptiva de las memorias cache basadas en DWM. Los bits en estas caches se distribuyen en una banda

magnética de nanocables y se acceden desplazándolos previamente hasta situarlos debajo de un puerto de acceso. Sin embargo, estas operaciones de desplazamiento se traducen en un consumo dinámico adicional. Además, también provocan un tiempo de acceso a la información variable y más lento frente a SRAM. Por esta razón, algunos trabajos recientes han propuesto el uso de DWM para implementar los niveles inferiores del subsistema de memoria del procesador como los niveles compartidos de L2 y L3 [3, 4, 5].

Este trabajo explora el uso de DWM para implementar caches de datos de primer nivel (L1). Aunque se trata de caches privadas más pequeñas respecto a L2 y L3, y los beneficios que ofrece DWM en cuanto a consumo y área sean a priori menores en L1, en sistemas *multi-core*, y especialmente *many-core*, la capacidad de cache acumulada en L1 puede ser del orden de cientos o miles de kilo-bytes. Nótese también el hecho de que, debido a la alta densidad de DWM, esta tecnología podría implementar caches con una mayor capacidad por unidad de superficie. En este sentido, en los procesadores *Simultaneous Multithreading* (SMT), con varios hilos por núcleo compartiendo la cache L1, se podría dedicar una mayor capacidad de almacenamiento para cada hilo y reducir los efectos de *cache thrashing* entre hilos. Finamente, el uso de DWM en L1 también puede conducir a una reducción significativa del consumo dinámico, el cual es proporcional al número de accesos que se realizan en la cache. En este trabajo se caracterizarán las operaciones de desplazamiento en caches L1 basadas en DWM. Para ello, se implementarán y se evaluarán las políticas de desplazamiento de bits del estado-del-arte para niveles inferiores de cache, y se propondrá una organización de cache novedosa que se ajuste a las características de L1 para reducir el impacto que las operaciones de desplazamiento tendrían tanto en consumo como en tiempo de acceso.

El presente trabajo se ajusta a las líneas de investigación en arquitecturas de procesadores que lleva realizando durante los últimos años el Grupo de Arquitecturas Paralelas (GAP) del Departamento de Informática de Sistemas y Computadores (DISCA) de la *Universitat Politècnica de València*, profundizando así en el destacado papel innovador del tejido académico de esta universidad. La investigación que se presenta supone una primera aproximación en una línea de investigación que podría tener un gran impacto en los años venideros.

1.2 Objetivos

Este trabajo consta de dos objetivos principales: i) explorar las posibilidades que ofrece implementar una memoria cache L1 con la tecnología DWM, siendo este un objetivo técnico y ii) afianzar y profundizar en los conocimientos sobre arquitectura de computadores obtenidos cursando el Grado de Ingeniería Informática, siendo este un objetivo personal. Estos dos grandes objetivos los podemos dividir en hitos, objetivos más simples que permitan por un lado visualizar el trabajo realizado; y por otro, establecer y planificar una serie de metas concretas durante el desarrollo de este proyecto.

Los hitos relacionados con el objetivo técnico principal son los siguientes:

- Instrumentar un simulador ciclo-a-ciclo para modelar memorias basadas en tecnología DWM.
- Implementar políticas de desplazamiento de datos del estado-del-arte, medir las prestaciones entre ellas y obtener conclusiones.
- Diseñar, implementar, validar y evaluar una propuesta de distribución de conjuntos de cache entrelazados que se ajuste a las características de la localidad espacial y temporal de los datos en L1.

- Evaluar la mejora de prestaciones obtenidas al aumentar la capacidad de almacenamiento de la cache L1 de acuerdo con la mayor densidad que ofrece DWM.

A su vez, los hitos relacionados con el objetivo personal principal son los que siguen:

- Aprender cómo funciona un simulador ciclo-a-ciclo de procesadores utilizado ampliamente en la industria y la academia, para poder posteriormente instrumentarlo y adaptarlo a las necesidades del proyecto.
- Realizar una búsqueda y estudio del estado-del-arte sobre tecnologías no volátiles para implementar memorias cache, profundizando en la tecnología DWM.
- Conocer en profundidad la arquitectura de un procesador x86 y de su jerarquía o subsistema de memoria.
- Desarrollar herramientas que conviertan automáticamente los datos en bruto del simulador en resultados que puedan ser interpretados fácilmente.

1.3 Metodología

El proyecto emplea una metodología de trabajo basada fundamentalmente en el uso del simulador ciclo-a-ciclo Multi2Sim [6]. 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 fraccionar 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 las 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

El resto de la memoria se organiza en cinco capítulos, resumidos a continuación:

- **Capítulo 2, Tecnologías de Memoria:** se exponen los conocimientos necesarios para el completo entendimiento del proyecto, incluyendo referencias al estado-del-arte conforme se detallan las diferentes tecnologías.

- **Capítulo 3, Implementación de DWM en la Cache de Datos L1:** en este capítulo se detalla a nivel teórico las principales propuestas de memorias cache implementadas con tecnología DWM y se propone una distribución de los datos que se ajusta a las características de L1 y minimiza las penalizaciones por desplazamientos.

- **Capítulo 4, Entorno Experimental:** se detalla el conjunto de herramientas utilizadas durante el desarrollo de este trabajo, indicando de manera razonada la motivación de su uso.

- **Capítulo 5, Resultados Experimentales y Discusión:** se realizan una serie de experimentos para verificar las hipótesis teóricas y se discuten los resultados obtenidos con el objetivo de valorar la validez de las propuestas.

- **Capítulo 6, Conclusiones y Trabajo Futuro:** de acuerdo con la discusión del capítulo anterior, se razonan una serie de conclusiones. Además, se detalla el trabajo futuro y/o ampliaciones posibles del proyecto.

- **Apéndice A, Scripts de Tratamiento de Datos:** en este anexo se presentan los *scripts* elaborados con el objetivo de automatizar la extracción de datos.

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 diferentes tecnologías que se utilizan para implementar memorias cache, incluyendo la tecnología DWM utilizada en el presente trabajo.

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 de 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, 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 Voltios en nodos de TSMC de 10 y 7 nanómetros [7].

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 [8]. 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 así como el área ocupada sea mucho menor a costa de una velocidad de acceso no tan elevada frente a SRAM. Además, debido a que los valores lógicos en una

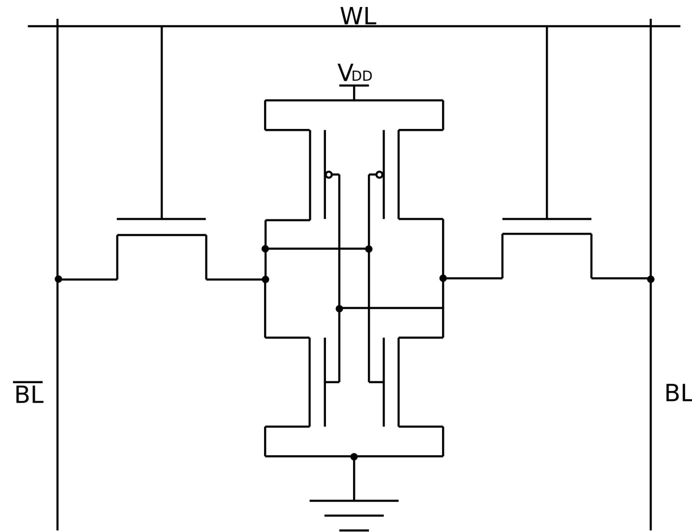


Figura 2.1: Esquema de una celda de memoria 6T SRAM.

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.

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 [9] e Intel Haswell [10]. 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 [11, 12, 13, 14].

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 frente a la tecnología eDRAM. Entre estas tecnologías, destacan STT-RAM y DWM por su compatibilidad con CMOS, su resistencia y sus tiempos de lectura competitivos en comparación con los de la tecnología SRAM.

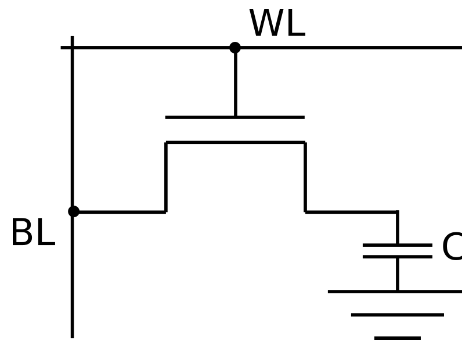


Figura 2.2: Esquema de una celda de memoria 1T1C eDRAM.

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 [15].

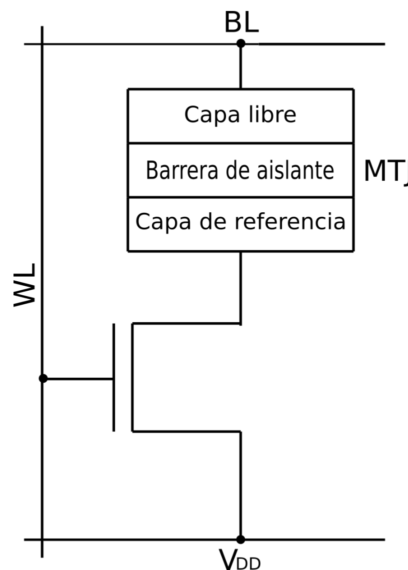


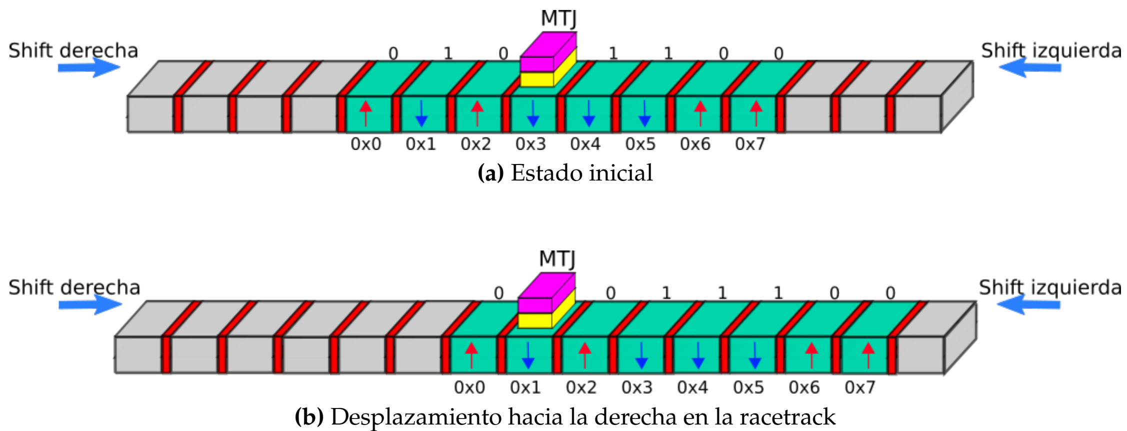
Figura 2.3: Esquema de una celda de memoria STT-RAM.

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 [16, 17, 18, 15, 19, 20]. 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 refresco [21, 22, 23], o proponen diseños de cache híbrida SRAM/STT-RAM, cuya porción SRAM limita la alta densidad y bajo consumo estático de STT-RAM [24, 25].

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*, 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 racetrack 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 [26].

La Figura 2.4 muestra un esquema con el desplazamiento de los dominios sobre una racetrack. 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 a la dirección de memoria 0x3 debajo del cabezal. Supongamos que deseamos acceder a la dirección 0x1 (Figura 2.4b). El desplazamiento necesario para llevar a cabo este acceso se produce a partir del mecanismo de desplazamiento a ambos lados de la racetrack. El mecanismo de desplazamiento se implementa mediante una serie de transistores que permiten desplazar los dominios sobre la racetrack hasta que el dominio en cuestión se posiciona debajo del cabezal.



(a) Estado inicial

(b) Desplazamiento hacia la derecha en la racetrack

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

Como se aprecia en la Figura 2.4, el desplazamiento de los bits sobre la racetrack 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 racetrack al realizar desplazamientos. Estos dominios adicionales se muestran en color gris. Al desplazarse por la racetrack, los bits ocupan progresivamente los dominios adicionales sin perderse por los extremos. La necesidad de estos dominios implica que una racetrack de 16 dominios sólo pueda almacenar 8 bits. Generalizando, se necesita una racetrack con una capacidad de 2N dominios para almacenar N bits.

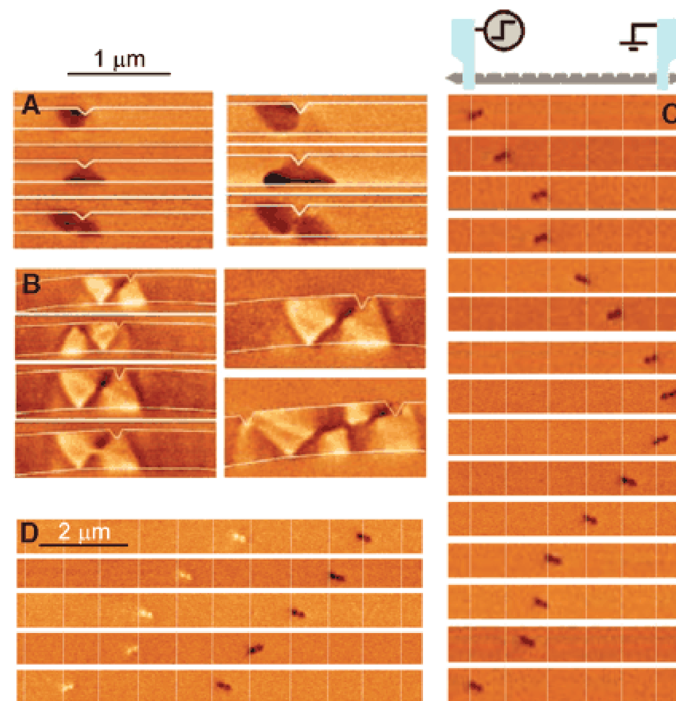


Figura 2.5: Vista de dominios sobre una racetrack [1].

La Figura 2.5 muestra la implementación física de una racetrack. Las formas triangulares oscuras que se pueden observar en las racetracks 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 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 racetrack. Concretamente, se puede apreciar el desplazamiento de un '1' lógico hacia el extremo derecho de la racetrack, 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 racetrack. Finalmente, en la Figura 2.5d se observa la progresión de posiciones en la racetrack para ambos valores lógicos '0' y '1'.

2.4 Comparativa

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.

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 racetrack. 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 racetrack y del número de racetracks en la memoria, asumiendo que cada racetrack 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

Implementación de DWM en la Cache de Datos L1

Este capítulo expone desde un marco teórico las dos principales propuestas presentadas en este trabajo, que son el resultado de un proceso iterativo. Primero se aborda la implementación de una memoria cache de datos L1 basada en DWM para posteriormente proponer una organización entrelazada de los cojuntos de la cache con el objetivo de mejorar el tiempo de acceso a la memoria.

3.1 Racetrack en Anillo

Partiendo del diseño de racetrack comentado en el capítulo anterior, la Figura 3.1 muestra la organización de una memoria de 128 bits (8× 16) de correspondencia directa con un tamaño de bloque de 8 bits (16 bloques o conjuntos) y 8 racetracks (los dominios vacíos no se muestran en la figura). De acuerdo con la organización de los datos, cada racetrack contiene un bit de cada bloque. En color verde se aprecian cuatro grupos de cabezales, con un total de ocho cabezales por cada grupo. El color verde más intenso se refiere al grupo de los cabezales 0, a partir de los cuales se puede realizar un acceso simultáneo a los 8 bits pertenecientes a la dirección de memoria o conjunto 2.

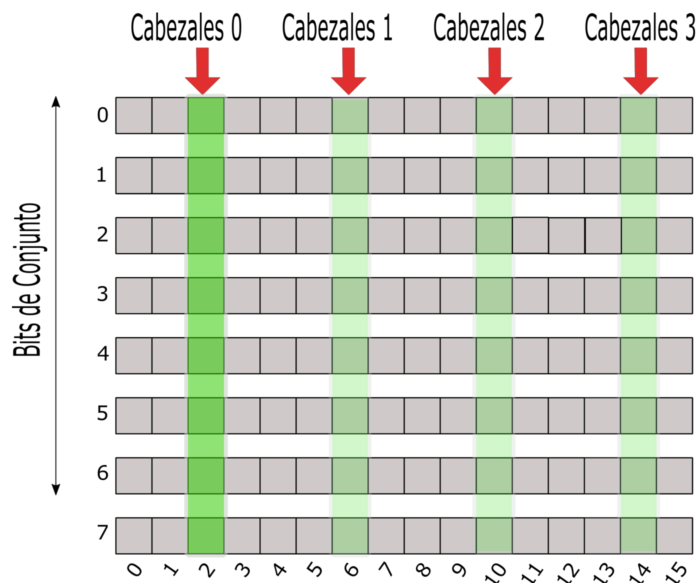


Figura 3.1: Organización convencional de una memoria DWM de 128 bits.

Uno de los inconvenientes principales de este tipo de memorias es que el tiempo de acceso depende del desplazamiento necesario para situarse sobre el dato que se quiere leer/escribir. Dado que cada desplazamiento requiere un ciclo de reloj [3], el objetivo es reducir al máximo el número de ciclos necesarios para situarse sobre cualquier dirección de memoria. Por este motivo se opta por distribuir de manera equidistante los grupos de cabezales a lo largo de las racetracks, ya que esto consigue reducir la distancia máxima entre ellos. Por ejemplo, si solo tuviésemos un cabezal en la cinta, la latencia máxima sería de 16 ciclos. Al colocar 4 cabezales equidistantes conseguimos reducir esta latencia máxima a 4 ciclos de reloj. En otras palabras, la distancia entre el grupo de cabezales 0 y el grupo de cabezales 1 es de 4 dominios. Lo mismo ocurre para el resto, y esta equidistancia se mantendrá, desplazándose los bits por las racetracks en una operación de desplazamiento común para que un grupo de cabezales pueda siempre acceder a los 8 bits de una misma dirección. Por ejemplo, si alineamos el grupo de cabezales 0 una posición hacia a derecha en el conjunto 3 (desplazamiento de bits una posición hacia la izquierda en las racetracks), los grupos de cabezales 1, 2 y 3 se alinean con los conjuntos 7, 11 y 15, respectivamente.

Como se ha comentado en la Sección 2.3.2, es necesario utilizar 2N dominios para almacenar N bits utilizando esta tecnología. Esto supone un inconveniente porque se dobla el área necesaria para almacenar los bits. En este sentido, resulta de interés encontrar una arquitectura capaz de solucionar este problema.

Una posibilidad de evitar el problema de los dominios auxiliares es implementar las racetracks en forma de anillo [27]. Esta técnica consiste en unir por los extremos la racetrack, de manera que los bits se desplazan por la cinta sin que se produzca una destrucción de datos. Los cabezales se sitúan estáticos de igual manera que en la memoria convencional, mientras que los transistores que implementan el mecanismo de desplazamiento se sitúan uno en cada mitad de la racetrack. Esto supone unos cambios mínimos en la implementación, pero se consigue eliminar la necesidad de 2N dominios para almacenar N bits. Estos cambios implican tener en cuenta que al tratarse de una racetrack circular, los datos pueden pasar de un extremo a otro de la racetrack (de la posición 15 a la 0, por ejemplo, solo habría el desplazamiento de un bit).

La siguiente sección parte de este tipo de implementación circular para explicar las políticas de desplazamiento de cabezales.

3.1.1. Políticas de Desplazamiento

Esta sección presenta una clasificación de las diferentes políticas de elección y gestión de cabezales, realizando un estudio para elegir la opción que minimice el tiempo de acceso en racetracks circulares. Para lograr un desplazamiento más eficiente, se presentan tres políticas diferentes de gestión de los cabezales. Estas políticas dependen de la posición actual de la cabeza de la racetrack (dominio 0) y del dominio que se desea acceder. Las políticas de gestión de cabezales, propuestas en la literatura para racetracks convencionales [3], se dividen en dos grupos en base a su objetivo: las que atañen a la selección del cabezal y las que atañen a la actualización del cabezal una vez realizado el acceso a los datos.

Las dos políticas que hacen referencia a la selección del cabezal a realizar el acceso son las siguientes:

- *Static* o estático. Esta política implica que cada cabezal tiene asignada una zona estática en la racetrack de lectura/escritura. Esta zona se reparte de manera equitativa, haciendo que la racetrack se divida entre el número de cabezales.

- *Dynamic* o dinámico. Con dinámico nos referimos a que la selección del grupo de cabezales que va a realizar el acceso se hace en base a la posición actual de los grupos de cabezales. Es decir, los cabezales no tienen áreas de la racetrack asignadas, sino que es el grupo de cabezales más cercano a la dirección emitida el que realiza el acceso. Para ello desplazamos los dominios solicitados hasta el grupo de cabezales más próximo.

Las políticas que hacen referencia a la actualización de los cabezales una vez finaliza el acceso son las que siguen:

- *Lazy* o vago. *Lazy* implica que una vez ya se ha realizado el acceso, los datos accedidos se mantienen alineados con el cabezal, sin realizar ninguna operación de desplazamiento adicional.

- *Eager* o ansioso. La política *eager* establece que la posición de base de los cabezales es la inicial, con el objetivo de minimizar la latencia máxima. Esto quiere decir que, para una racetrack de 16 bits con 4 cabezales, la posición de partida ideal de los cabezales serán los conjuntos 2, 6, 10 y 15. Después de un acceso que implique un desplazamiento, los cabezales deberán volver a estas posiciones con el objetivo. Esto tiene implicaciones en la penalización por desplazamiento, ya que los datos no sólo han de desplazarse sobre la racetrack cuando se lean o escriban, sino que han de realizar el desplazamiento pertinente para volver a la posición asignada previamente, una vez finalizado el acceso.

De acuerdo con la clasificación anterior, se pueden obtener cuatro combinaciones en lo que respecta a los desplazamientos de las racetracks, ya que resulta pertinente combinar tanto las políticas de selección como de actualización de cabezal. Estas cuatro políticas son: *Static Lazy*, *Dynamic Lazy*, *Static Eager* y *Dynamic Eager*. Se prescindirá de *Dynamic Eager* puesto que obtiene los mismos resultados en cuanto a desplazamientos y selección de cabezal que *Static Eager*. Por tanto, las tres políticas a tratar en este trabajo son:

- *Static Lazy* (de ahora en adelante referida como SL). La principal ventaja de esta política es lo sencillo que resulta calcular el cabezal que debe leer/escribir en cada acceso, lo cual simplifica el diseño hardware necesario para implementar esta política de gestión de los cabezales.

- *Dynamic Lazy* (de ahora en adelante DL). Esta política reduce teóricamente el coste del desplazamiento al elegir el cabezal de la cinta con menor penalización por desplazamiento en relación con SL. Por el contrario, la selección dinámica de los cabezales requiere almacenar una serie de posiciones actuales de los cabezales para calcular cual de ellos es el más cercano al dominio requerido.

- *Static Eager* (de ahora en adelante SE). Para reducir la penalización por desplazamiento, la posición inicial de los cabezales es la parte media del área asignada a cada cabezal de manera estática. De esta manera se reduce la distancia máxima de desplazamiento del cabezal a la mitad de la distancia entre dos grupos de cabezales.

La Figura 3.2 ilustra, de forma simplificada, las tres políticas de gestión de cabezales utilizando una memoria DWM circular con cuatro cabezales. La Figura 3.2a representa el estado inicial para cualquier política, antes de realizar un acceso. El resto de figuras muestran el comportamiento de las tres políticas para leer el conjunto 3.

En las Figuras 3.2b y 3.2d se aprecia, para las políticas estáticas, la zona designada para la lectura/escritura del cabezal 0 en color naranja, la zona designada al cabezal 1 en

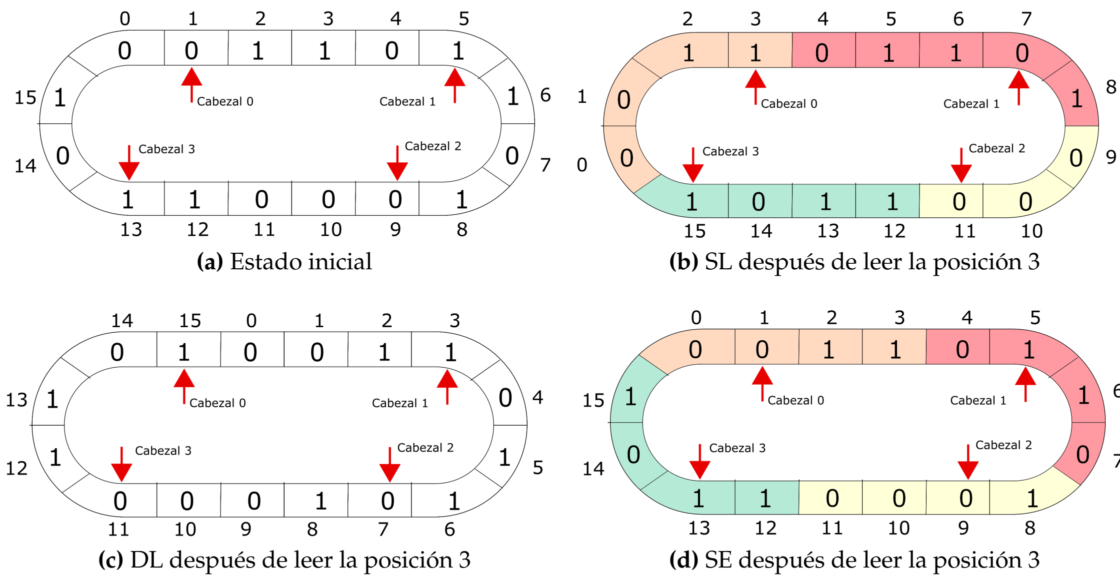


Figura 3.2: Ejemplo de gestión de cabezales.

color rojo, la zona designada para el cabezal 2 en color amarillo y la zona designada para el cabezal 3 en color verde. Nótese que DL (Figura 3.2c) asigna dinámicamente los cabezales y por tanto se desconoce a priori qué cabezal se encargará del acceso. Considerando el desplazamiento realizado para leer la posición 3, la política SL, al disponer de un área designada, hará servir el cabezal 0 realizando un desplazamiento de 2 bits. Por su parte, la política DL usará el cabezal más cercano a la posición, esto es, el cabezal 1 con un desplazamiento de 2 bits. Finalmente, la política SE usará el cabezal 0 que necesita también un desplazamiento de 2 bits, tras el cual volverá al estado inicial. En resumen, DL intenta reducir la distancia de desplazamiento aprovechando siempre el cabezal más cercano, mientras que SE intenta reducir esta distancia empezando siempre desde una posición intermedia, para reducir el desplazamiento máximo a la mitad de las áreas delimitadas estáticamente a cada cabezal.

Las políticas objeto de estudio tienen una serie de requisitos de implementación, que por supuesto afectan también a la elección de la política. Si comparamos *Static* con *Dynamic*, la primera permite saber qué cabezal realizará el acceso en el preciso instante en que se conoce la dirección. En cambio, *Dynamic* requiere una consulta previa de la posición de todos los cabezales, para determinar cual es el más próximo a la dirección. Esto implica necesariamente que *Dynamic* necesite hardware adicional (e.g., registros y sumadores para calcular distancias) respecto a *Static*.

Analizando detenidamente las diferencias entre *Eager* y *Lazy*, puesto que *Eager* regresa siempre a la posición intermedia, no requiere hardware adicional en este sentido, ya que la posición de partida es siempre la misma. En cambio, *Lazy* requiere añadir un registro por cada cabezal para poder almacenar en qué posición se encuentra.

Finalmente, a modo de resumen del razonamiento expuesto, se presenta la Tabla 3.1. En ella se recogen las conclusiones teóricas que serán más tarde comprobadas de forma empírica.

	SL	DL	SE
Aprovechamiento de la localidad	Sí	Sí	No
Hardware adicional	Medio	Complejo	Simple
Elección de cabezal	Simple	Compleja	Simple

Tabla 3.1: Tabla resumen de las políticas de gestión de cabezales.

3.2 Estructura de una Cache L1 con Tecnología DWM

Partiendo del diseño convencional de una memoria de 128 bits mostrado en la Figura 3.1, a continuación se amplia la capacidad hasta situarnos en un banco de 8 KB. Para ello, se asume un tamaño de bloque de 64 bytes y un total de 128 conjuntos con correspondencia directa. La Figura 3.3 un esquema con la organización del banco de memoria. De nuevo en color verde se muestran los 4 grupos de cabezales equidistantes y en verde más oscuro cómo se realiza un acceso sobre la dirección o conjunto 2. Nótese que en este diseño la separación entre los grupos de cabezales aumenta hasta los 32 dominios.

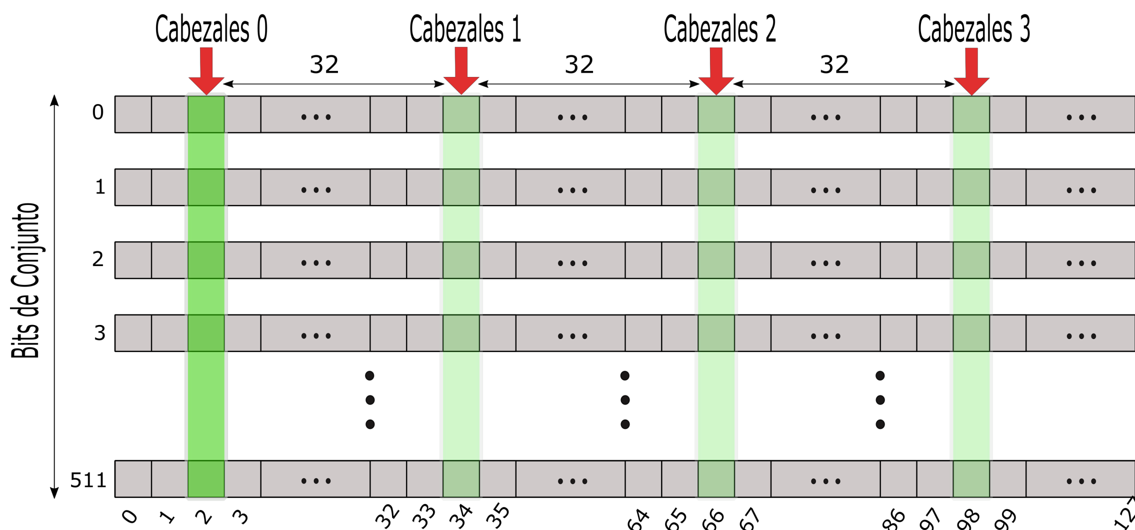


Figura 3.3: Organización convencional de un banco de memoria de 8 KB.

En este trabajo, se considerará un tamaño base de la cache de datos L1 de 32 KB. Para implementar esta capacidad de cache, se consideran 4 bancos de memoria de 8 KB como el descrito anteriormente. La Figura 3.4 muestra cuatro bancos de 8KB superpuestos (bancos 0-3). Esta organización en bancos es habitual en las caches de procesadores comerciales y se utiliza para reducir el tamaño de los decodificadores. Con ello se consigue reducir el tiempo de acceso a la cache. Además, nótese que cada banco implementa una vía distinta en la cache, de manera que utilizando 4 bancos de 8 KB se consigue una memoria cache de 32 KB asociativa por conjuntos de 4 vías. Esta es una organización de cache L1 habitual en procesadores comerciales. Con el objetivo de reducir el tiempo de acceso, se asume que el acceso a etiquetas y datos en la cache se realiza en paralelo. En otras palabras, el acceso a un conjunto de cache supone acceder a todos los bancos (vías del conjunto) al mismo tiempo.

Finalmente, cabe destacar que, aunque las figuras anteriores muestran racetracks lineales a efectos de simplicidad en los esquemas, el resto del trabajo asume la implementación de racetracks en anillo discutida en la Sección 3.1.

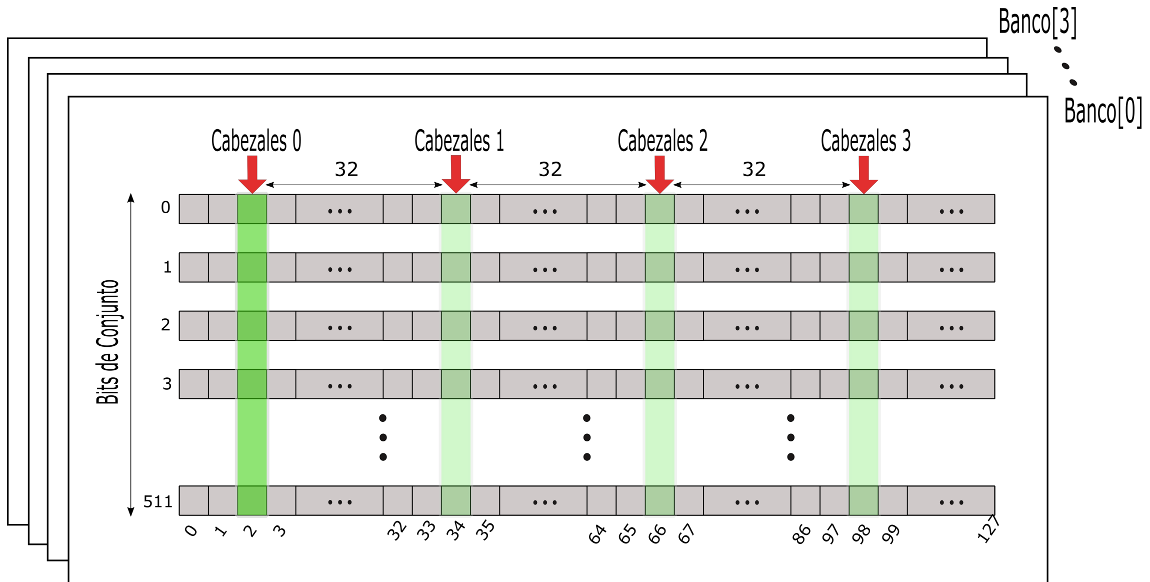
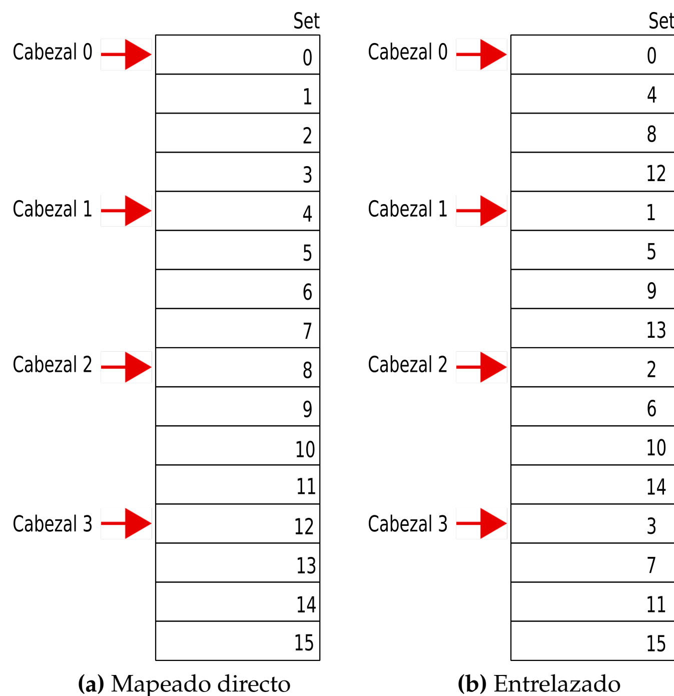


Figura 3.4: Memoria cache de 32 KB compuesta por 4 bancos.

3.3 Organización Entrelazada

En la siguiente iteración de diseño, el objetivo es explotar en la medida de lo posible la localidad espacial en la cache de datos L1 con el objetivo de reducir la distancia de desplazamiento de los cabezales. Para ello, se reorganiza la distribución de conjuntos en las racetracks.

La organización propuesta, a la que nos referiremos como entrelazado de conjuntos, consiste en ordenar los conjuntos de manera que los cabezales estén siempre alineados en conjuntos consecutivos. De esta manera, los conjuntos se intercalan en base a la cantidad de grupos de cabezales que implemente la cache.



(a) Mapeado directo **(b)** Entrelazado

Figura 3.5: Comparativa de organización de conjuntos en racetracks de 16 bits.

La Figura 3.5 muestra las diferencias entre una organización convencional de conjuntos donde estos se distribuyen consecutivamente (a esta organización nos referimos como mapeado directo) y la propuesta de organización entrelazada a partir de un ejemplo de racetrack de 16 bits con 4 cabezales. En la propuesta entrelazada, los conjuntos se reordenan intercalando de cuatro en cuatro los conjuntos de acuerdo con el número de cabezales. De esta manera, se puede acceder a cuatro conjuntos consecutivos sin realizar ningún desplazamiento.

El objetivo de acceder a cuatro conjuntos consecutivos de memoria sin desplazamiento es aprovechar la localidad espacial en la cache, es decir, la tendencia del código a acceder a una dirección cercana con respecto a la actual (localidad de referencias). De esta manera, se debería reducir teóricamente la cantidad total de desplazamientos, consiguiendo mitigar la principal desventaja de las memorias cache basadas en la tecnología DWM.

Para implementar este comportamiento entrelazado en el simulador ciclo-a-ciclo, se ha desarrollado una ecuación para traducir una dirección física entrelazada a una dirección física directa. Esto tiene utilidad a nivel de implementación para considerar los conjuntos en base al entrelazado cuando se realiza el cálculo de distancia de desplazamiento. De esta manera, se puede medir el impacto del entrelazado pero realizando cálculos a partir de la traducción a mapeado directo. A partir del número de cabezales presentes en la racetrack, la cantidad de conjuntos y la dirección a traducir, la Ecuación 3.1 permite ubicar dentro de una racetrack dónde se almacenará una posición de memoria.

Se utilizan las abreviaturas *dir ent* para dirección entrelazada y *nro* para número. La ecuación se obtiene a partir de un planteamiento matricial de dos niveles, siendo el número de cabezales las filas y el número de conjuntos las columnas de la matriz. La primera parte de la fórmula (hasta el operador de suma) sirve para calcular el cabezal y la segunda parte (a partir del operador de suma) sirve para calcular el conjunto. A partir de la suma de ambas partes se obtiene la traducción completa.

$$(dir\ ent \,\%nro\ cabezales) \times (\frac{nro\ sets}{nro\ cabezales}) + (\frac{dir\ ent}{nro\ cabezales})\,\%(\frac{nro\ sets}{nro\ cabezales}) \qquad (3.1)$$

CAPÍTULO 4

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.

4.1 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 [28, 6]. 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.

4.1.1. 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 4.1 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.

4.1.2. Compilación

Para compilar Multi2Sim será necesario ejecutar el siguiente comando:

```
1 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

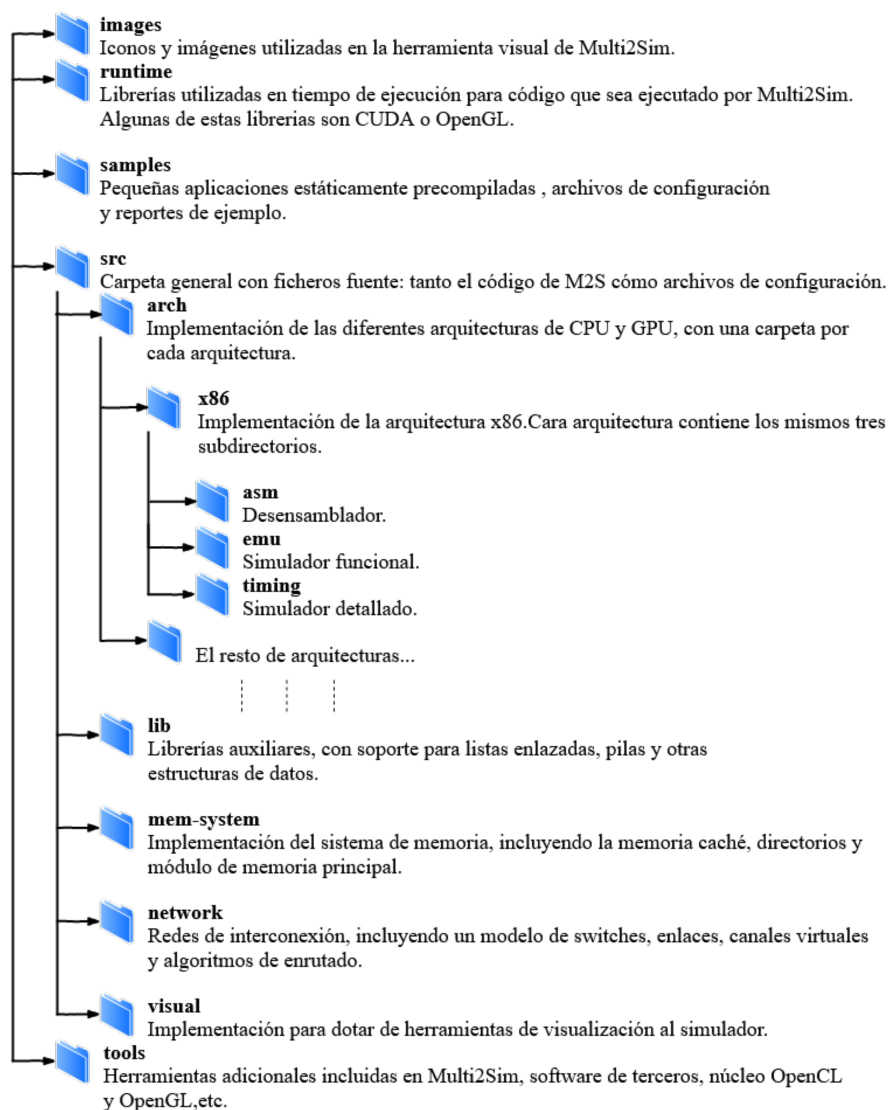


Figura 4.1: Estructura de directorios de Multi2Sim.

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.

4.1.3. Ejecución

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

```
./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 4.2 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.

4.1.4. 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 [29] para más detalles acerca de este simulador.

4.2 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 [30]. 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 4.2.1 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 [31].

4.2.1. 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 [30].

- 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.

4.3 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 [32]. 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.

4.4 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.

4.4.1. 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 4.1 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 4.1: 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.

4.4.2. HTCondor

HTCondor es un software de código abierto de alto rendimiento para la paralelización distribuida de tareas de gran intensidad computacional [33]. 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 5

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 del impacto de las operaciones de desplazamiento comparando políticas de desplazamiento del estado-del-arte aplicadas sobre L1, el impacto de la propuesta de organización de conjuntos entrelazada y finalmente un estudio del impacto de la capacidad de la cache.

5.1 Arquitectura del Procesador y Clasificación de Benchmarks

La arquitectura en chip con la que hemos trabajado consta de un núcleo[1], una cache L1 con arquitectura Harvard, caches L2 y L3 unificadas y una memoria principal (MP). La Figura 5.1 muestra un diagrama con la estructura comentada.

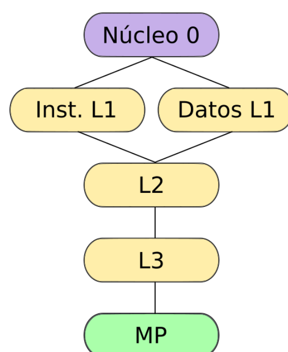


Figura 5.1: Esquema del procesador con un solo núcleo.

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 el subapartado correspondiente. La Tabla 5.1 detalla los parámetros de la jerarquía de memoria correspondientes al fichero *–mem-config* suministrado durante la simulación (véase la Sección 4.1.3). 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.

[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.

Cache L1 (datos e instrucciones)	
Tamaño	32KB
Asociatividad	Por conjuntos de 4 vías
Tamaño de bloque	64B
Latencia	1 ciclo
Política	LRU
MSHR	4
Cache L2	
Tipo	Unificada
Tamaño	512KB
Asociatividad	Por conjuntos de 16 vías
Tamaño de bloque	64B
Latencia	6 ciclos
Política	LRU
MSHR	4
Cache L3	
Tipo	Unificada
Tamaño	8MB
Tamaño de bloque	64B
Asociatividad	Por conjuntos de 16 vías
Latencia	12 ciclos
Política	LRU
MSHR	4
Memoria principal	
Tipo	DDR4 Micron
Tamaño	8 GB
Latencia	Generada por DRAMSim

Tabla 5.1: Configuración de la jerarquía de memoria del procesador.

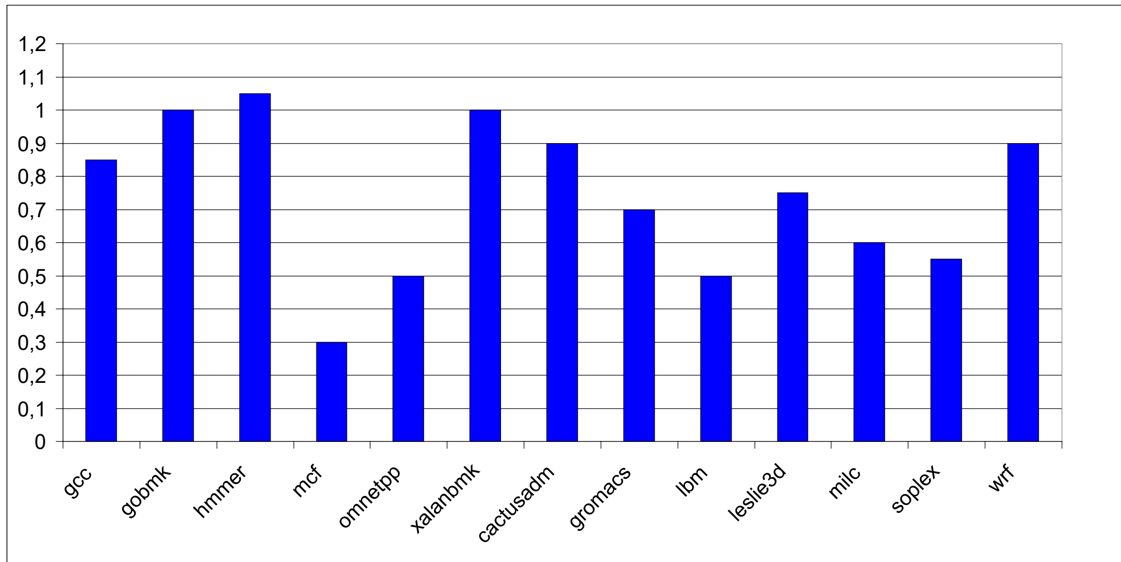
Arquitectura x86	
Fast-forward	500 millones de instrucciones
Colas y registros	
Tamaño del ROB	144
Tamaño de RfInt	288
Tamaño de RfFp	288

Tabla 5.2: Configuración de la arquitectura del procesador.

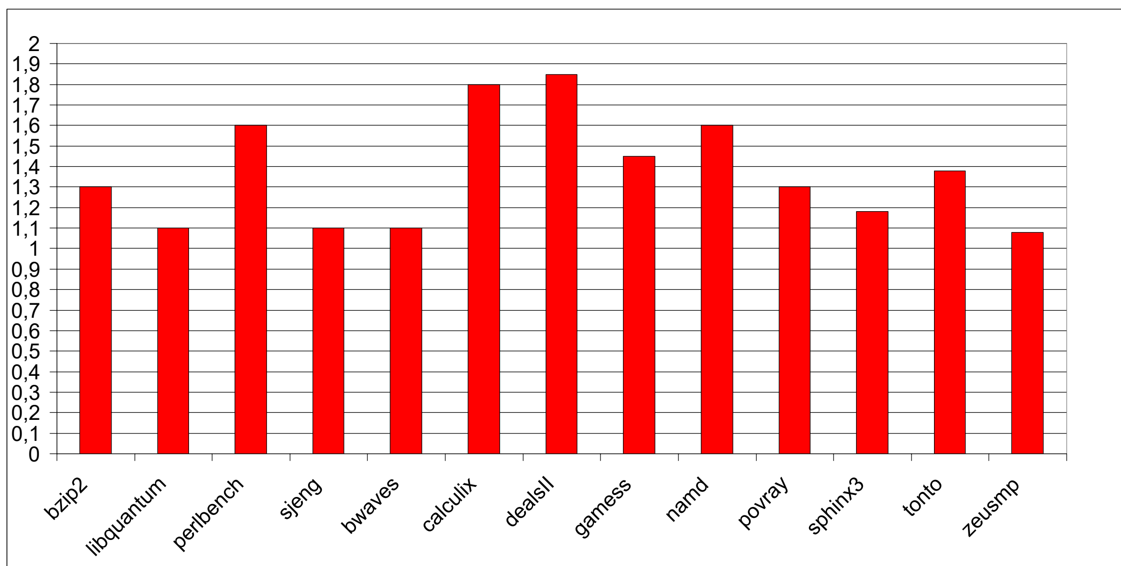
La Tabla 5.2 muestra los valores de la arquitectura x86 que se indican en el fichero *–x86-config*. Por cada benchmark, se ha establecido un *fast-forward* constante de 500 millones de instrucciones, siendo este número un valor habitual en multitud de trabajos de investigación que hacen uso de SPEC CPU 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.

Dado que la suite de SPEC CPU 2006 consiste en más de 20 aplicaciones, a efectos ilustrativos, a lo largo de la memoria los benchmarks se diferenciarán entre aquellos con un valor de Instrucciones por Ciclo (IPC) alto y bajo. Definimos un IPC alto a aquel mayor

que 1,05 e IPC bajo al resto. La Figura 5.2 muestra los resultados, los cuales son similares a resultados de IPC reportados en trabajos previos [34].



(a) IPCs bajos.



(b) IPCs altos.

Figura 5.2: IPC de las aplicaciones SPEC CPU 2006.

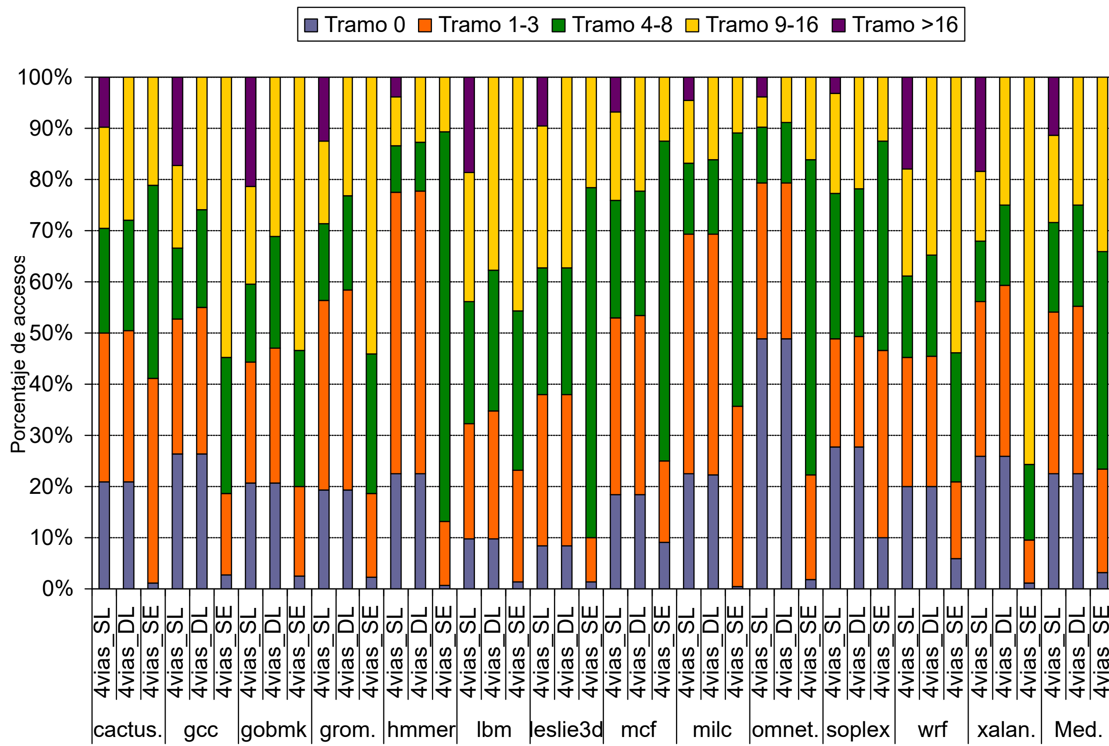
5.2 Políticas de Desplazamiento

En este trabajo se asume que la penalización por el desplazamiento de un bit en la racetrack es un ciclo de reloj [3]. Este coste es alto para una cache L1, motivo por el cual resulta indispensable estudiar cómo se comportan las políticas de desplazamiento del estado-del-arte SE, DL y SL sobre una cache L1. La Figura 5.3 muestra una distribución porcentual de desplazamientos (en número de bits desplazados) generados por los accesos a la cache. Un acceso puede realizar una operación de lectura o escritura sobre los datos solicitados, y también puede tratarse de un acierto o un fallo en la cache. Para una mayor claridad en la muestra de los resultados, los desplazamientos se agrupan en cinco tramos diferenciados. Un acceso que produce un desplazamiento 0 se interpreta como un acceso a datos que ya se encuentran debajo del cabezal y por tanto no es necesario realizar ningún desplazamiento. En este sentido, el tiempo de acceso es un ciclo y coincide con el de una cache convencional basada en tecnología SRAM. El tramo 1-3 corresponde a accesos que requieren un desplazamiento entre 1 y 3 bits, etcétera, antes de acceder a los datos ya situados debajo del cabezal en un ciclo adicional.

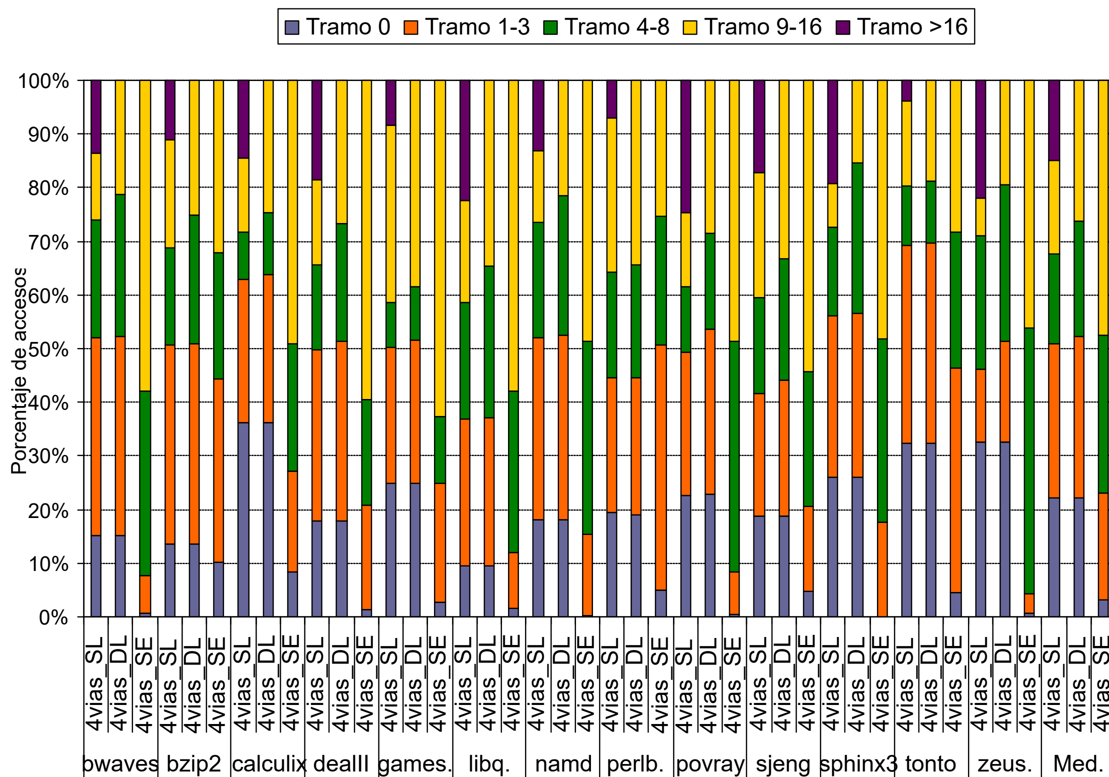
Para el tramo 0 se puede observar que tanto para SL como para DL los resultados son muy similares por cada benchmark, variando desde un 9 % (*leslie3d* y *libquantum*) hasta un 49 % (*omnetpp*). Esto se debe a que estas dos políticas aprovechan la alta localidad temporal en L1. Por el contrario, debido a que SE desplaza la racetrack hasta la posición media, no aprovecha la localidad temporal, reduciendo drásticamente el porcentaje de tramo 0 respecto a las políticas anteriores. Atendiendo a la media aritmética global, la diferencia entre SE y las otras dos políticas es de aproximadamente un 20 %. Uno de los objetivos principales de este trabajo es aumentar el tramo 0 para que el impacto del tiempo de acceso variable con DWM sea el menor posible. A partir de estos resultados, podemos concluir que la política SE no es adecuada para L1.

Respecto al tramo 1-3, en DL la media es ligeramente superior a la de SL. Esta mejora se debe principalmente al aprovechamiento que hace DL de los desplazamientos. Es decir, como la política DL siempre realiza un desplazamiento de bits hacia el cabezal más cercano al bit solicitado, esto se traduce en un acortamiento del tramo 1-3 frente a SL. En cambio, como SE tiende a favorecer desplazamientos medios, este tramo es en promedio menor con respecto a DL y SL. De hecho, partir de una posición intermedia generando desplazamientos medios implica que la política SE aumente el porcentaje de los tramos 4-8 y 9-16. Si comparamos SL con DL, vemos que en promedio lo que corresponde al tramo >16 en SL se reparte entre los tramos 1-3, 4-8 y 9-16, ya que DL carece de tramo >16. Esta ausencia de tramo >16 en DL se debe a que en esta política la distancia máxima que se puede recorrer (en bits) es la mitad de la distancia entre cabezales (en bits), es decir, 16 en la actual organización de cache. Algo similar ocurre con SE, que también tiene una distancia máxima fijada en la mitad de la distancia entre cabezales, ya que la posición más alejada de cualquier otra posición es la posición intermedia entre dos cabezales, que es donde a vuelve la racetrack tras un acceso.

En resumen, DL es la política que ofrece mejores prestaciones debido a i) su ausencia de tramo >16, a diferencia de SL y ii) el tamaño de su tramo 0 respecto al de SE, que es en promedio un 20 % más grande. Otro aspecto a tener en cuenta en SE es que, el hecho de tener que volver a desplazar la racetrack a su posición inicial puede demorar los accesos posteriores a la cache hasta que el desplazamiento de vuelta finalice, lo cual tendría un impacto adicional en el tiempo de ejecución frente a SL y DL.



(a) IPCs bajos



(b) IPCs altos

Figura 5.3: Comparativa de políticas de gestión de cabezales.

5.3 Comparativa entre Entrelazado y Mapeado Directo

Teniendo en cuenta las conclusiones derivadas del estudio de las políticas del estado-del-arte en la sección anterior, para el resto de resultados asumiremos la política DL. En el presente estudio se compara la organización de conjuntos entrelazada propuesta en este trabajo frente a una organización de conjuntos convencional y consecutivos en la racetrack, a la cual nos referiremos como mapeado directo de conjuntos.

La Figura 5.4 muestra los resultados. Por cada benchmark, la primera barra hace referencia a una política DL con mapeado directo, mientras que la segunda barra se refiere a una política DL con una organización de conjuntos entrelazada en la racetrack. Nótese que el tramo >16 desaparece al tratar exclusivamente con DL. A la vista de los resultados, tanto para aplicaciones con IPC alto y bajo, el tramo 0 aumenta en torno a un 15 % en la media para DL entrelazado respecto a DL directo. En otras palabras, con DL entrelazado, un 16 % de accesos no tendrá una penalización en el tiempo de acceso por desplazamiento, siendo el tiempo de acceso idéntico al de una cache convencional SRAM. Esta mejora del tramo 0 se debe a que con la organización entrelazada se consigue explotar la localidad espacial de los datos en L1, ya que ahora los cabezales se sitúan en conjuntos consecutivos. Además, cabe destacar que este aumento del tramo 0 se observa para todas las aplicaciones, lo que indica que todas ellas muestran localidad espacial y se benefician del entrelazado. Sin embargo, los porcentajes de mejora varían entre las aplicaciones, desde un 0,5 % (*libquantum*) hasta un 29 % (*tonto*). El resultado de *libquantum* se explica porque, a diferencia del resto de benchmarks, esta aplicación hace muy poco uso de cache de datos L1.

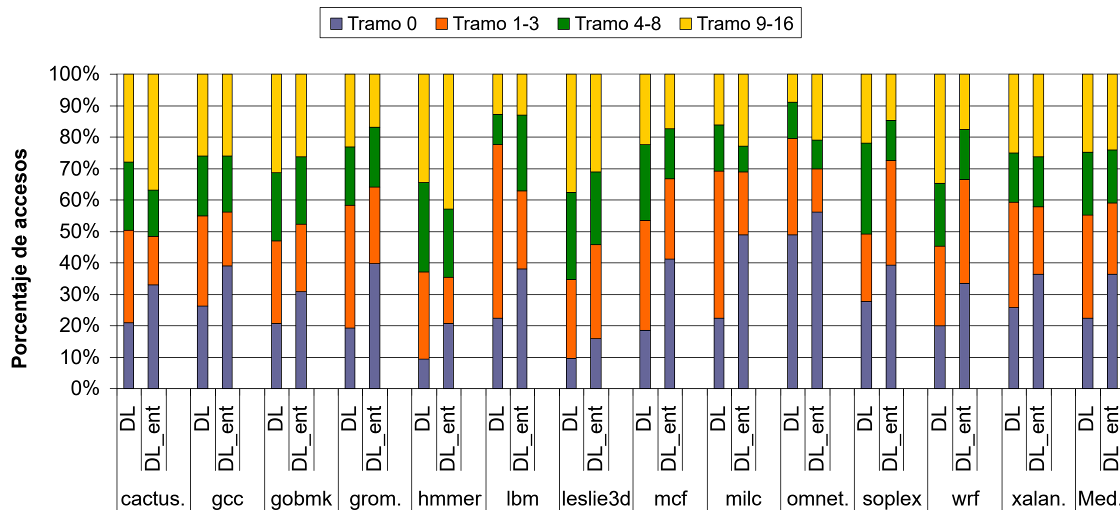
En cuanto al tramo 1-3, este disminuye en promedio en torno a un 10 y un 6 % para IPCs bajos y altos, respectivamente, si comparamos el mapeado entrelazado frente al directo. Esta disminución corresponde a una parte del tramo 1-3 del mapeado directo que transita al tramo 0 en entrelazado gracias al aprovechamiento de los cabezales situados en conjuntos consecutivos. Si nos fijamos en los benchmarks individualmente, vemos que en *zeusmp*, *soplex*, *wrf* y *leslie3d* este tramo aumenta aproximadamente un 23, 12, 8 y 5 %, respectivamente. Este aumento es debido a que la disminución relativa del tramo 1-3 en estos benchmarks es menor que el porcentaje de los tramos 4-8 y 9-16 que transita al tramo 1-3 gracias al entrelazado. Para el resto de aplicaciones se mantiene la tendencia de aumentar el tramo 1-3 gracias a la parte de tramo 4-8 y 9-16 que transita a este tramo. Este aumento del tramo 1-3 frente a los tramos 4-8 y 9-16 es positivo a efectos de minimizar la penalización por desplazamiento.

En el tramo 4-8 ocurre algo similar al anterior: en promedio disminuye sobre un 3 y un 5 % para IPCs bajo y altos, respectivamente, debido a que estos porcentajes promedios ha transitado al tramo 0. En las aplicaciones *calculix*, *dealII* y *gamess*, sin embargo, se aprecia un aumento marginal del tramo del 2 ,3 y 1 %, respectivamente.

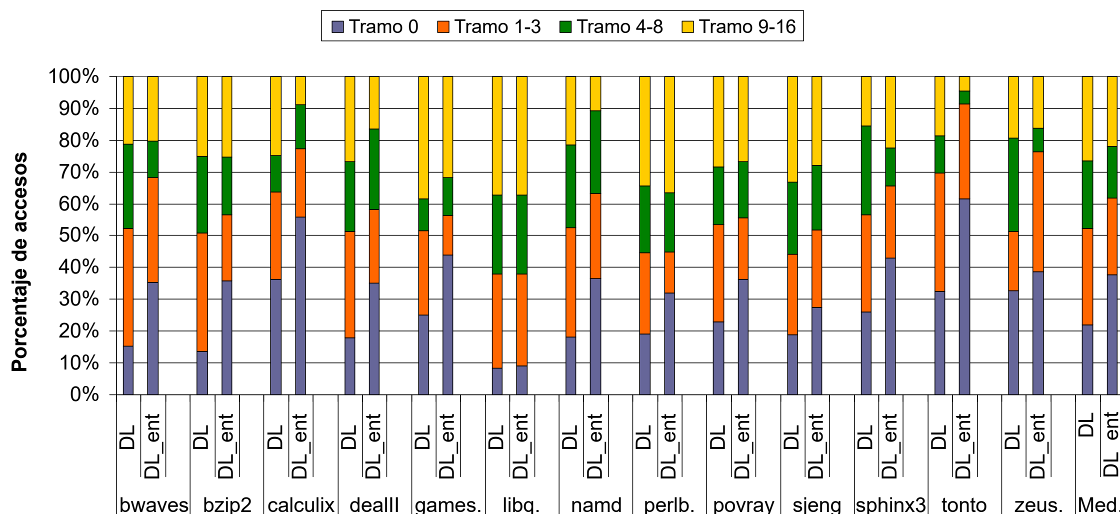
Finalmente, en el tramo 9-16 la reducción media supone un 1 y 5 % para IPCs bajo y altos, respectivamente. Esta tendencia se observa en todas las aplicaciones a excepción de *cactusADM*, *hmmer*, *milc*, *omnetpp*, *bzip2*, *perlbench* y *sphinx3*. Esta tendencia contraria se justifica debido a que el mapeado entrelazado genera cambios significativos en el movimiento de la racetrack que hacen que para accesos al mismo conjunto sean diferentes cabezales los que se utilicen para acceder a los datos en la organización entrelazada y directa. Este efecto se explica de forma detallada en la Sección 5.3.1.

En resumen, los resultados muestran la efectividad de la organización entrelazada al aumentar significativamente el tramo 0 en todas las aplicaciones. Además, en el resto de tramos las tendencias son en general hacia la reducción de tramos, lo que nos lleva a

afirmar que el entrelazado debe implementarse en memorias cache basadas en DWM ya que aprovecha la localidad espacial de las aplicaciones.



(a) IPCs bajos



(b) IPCs altos

Figura 5.4: Comparativa entre mapeado directo y entrelazado.

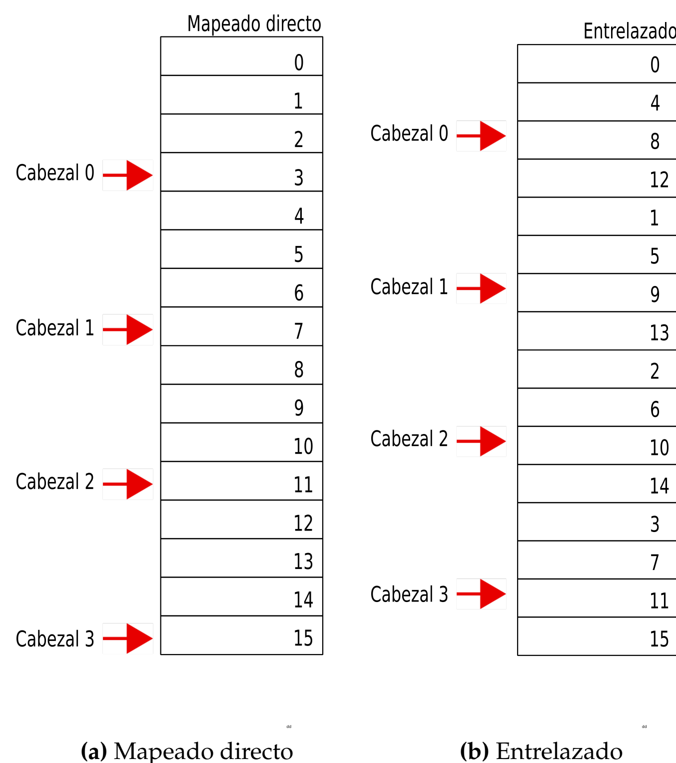
5.3.1. Análisis del Aumento del Tramo 9-16

Esta sección tiene como objetivo explicar de forma detallada el aumento del tramo 9-16 observado en la sección anterior. El aumento medio en este tramo es del 7 % para los benchmarks donde se da esta situación, lo que supone un porcentaje significativo y requiere un análisis específico. Para explicar la causa de este efecto nos valdremos de un ejemplo ilustrativo, para posteriormente ofrecer una posible solución.

Partimos del caso mostrado en la Figura 5.5, donde diferenciamos entre dos situaciones iniciales para el mapeado directo y el entrelazado, ambos asumiendo una política DL para el desplazamiento. A efectos ilustrativos, se consideran racetracks con 16 bits y 4 cabezales. El estado inicial en ambas implementaciones supone mantener uno de los cabezales sobre el conjunto 11 accedido anteriormente.

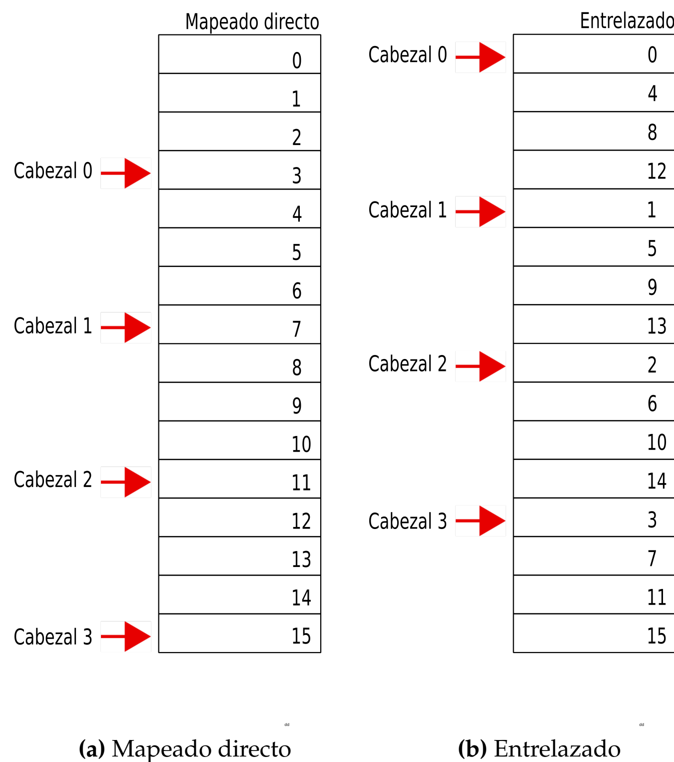
Asumamos un acceso al conjunto 3 en ambos casos. Tras este acceso, la posición de la racetrack con respecto a los cabezales se ilustra en la Figura 5.6. En el caso del mapeado directo, no resulta necesario un desplazamiento, puesto que partiendo de la situación anterior, el cabezal 0 ya se encuentra sobre el conjunto 3. Sin embargo, en la organización de conjuntos entrelazada, el acceso al conjunto 3 supone un desplazamiento previo de 2 bits para que el cabezal 3 pueda situarse encima de este conjunto.

Estos ejemplos cumplen el propósito de mostrar cómo a partir de dos accesos consecutivos (conjunto 11 y 3) se dan casos donde el mapeado directo no necesita desplazamientos, mientras que el entrelazado debe realizar el máximo desplazamiento posible (2 bits en este ejemplo). Este fenómeno ocurre debido a que, por un lado, de forma casual, el cabezal en el mapeado directo se encuentra alienado de antemano en el conjunto a acceder; por otro lado, también a la larga distancia entre dos accesos consecutivos a la memoria, es decir, en ausencia de localidad espacial. Este ejemplo se puede extrapolar a nuestro caso, ya que difiere únicamente en el tamaño de la racetrack. Una posible solución para solventar este problema sería simplemente mandar a la cache L2 aquellos accesos sin localidad espacial y que fueran a requerir un desplazamiento extenso. De esta manera, la racetrack quedaría sin movimiento en L1 y podría beneficiar a siguientes accesos que sí mostrasen localidad espacial.



(a) Mapeado directo **(b)** Entrelazado

Figura 5.5: Posición de los cabezales tras acceder al conjunto 11.



(a) Mapeado directo **(b)** Entrelazado

Figura 5.6: Posición de los cabezales tras acceder al conjunto 3.

5.4 Impacto de la Capacidad de la Cache en los Tramos

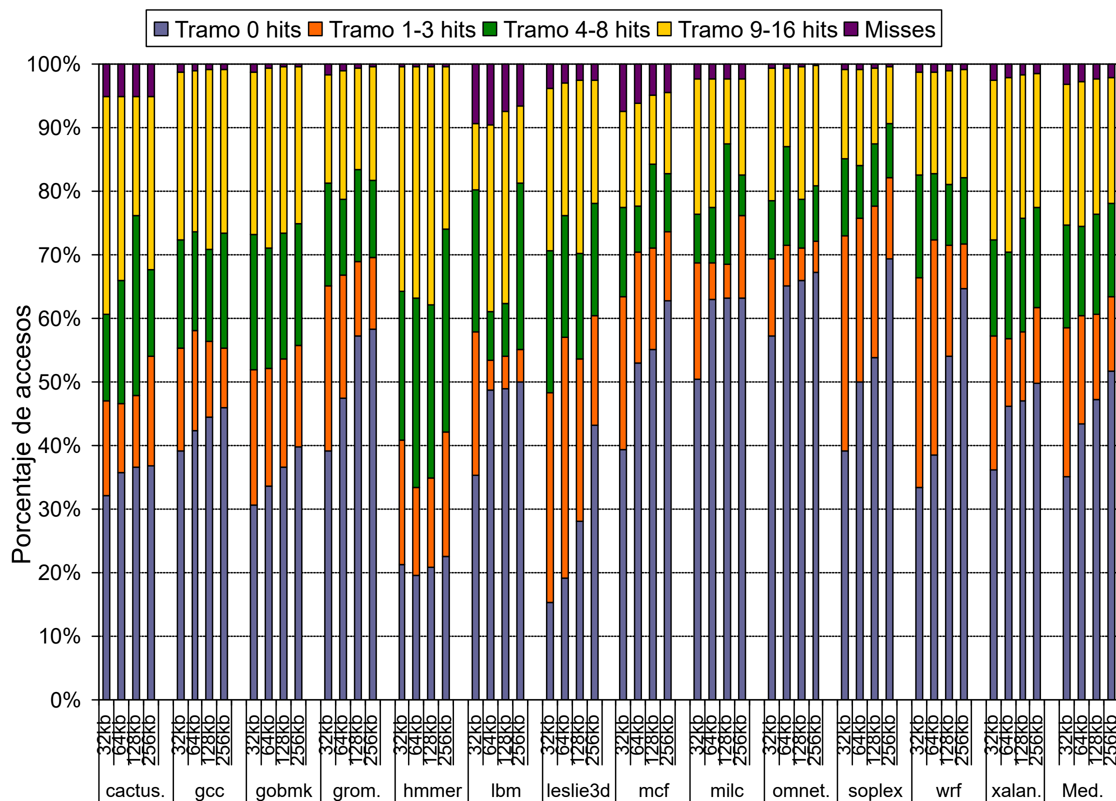
La alta densidad de las memorias cache DWM permite aumentar la capacidad de almacenamiento por unidad de superficie respecto a caches convencionales SRAM. En este estudio se analiza el efecto que tiene el aumento de la capacidad de la cache sobre la distribución de accesos en los tramos de desplazamiento, manteniendo constante el número de vías en 4 y la política de desplazamiento DL en la racetrack.

Las capacidades de cache consideradas son 64, 128 y 256 KB. A estas capacidades asociamos 8, 16 y 32 cabezales, respectivamente, para mantener siempre una distancia constante de 32 bits entre cabezales consecutivos. A estas configuraciones, cabe añadir la configuración base de 32 KB y 4 cabezales. La decisión de mantener constante la distancia entre cabezales consecutivos se debe al objetivo de cuantificar el impacto en los tramos exclusivamente por el hecho de aumentar el número de conjuntos de cache.

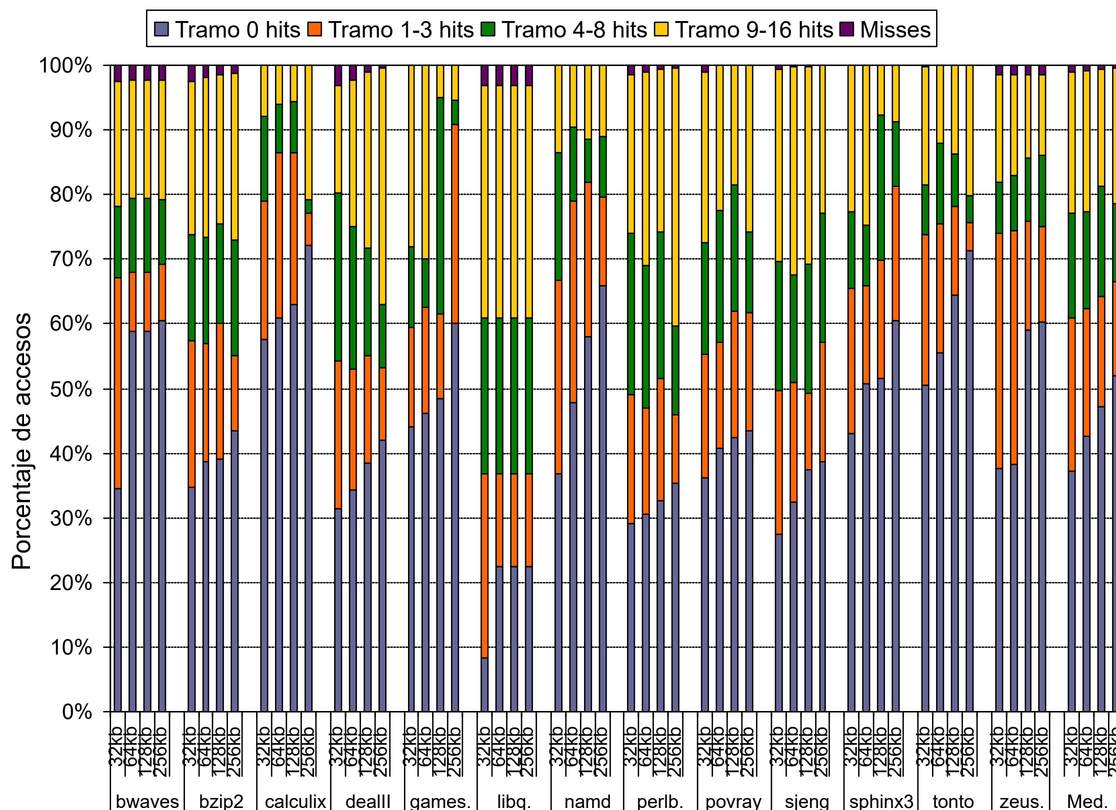
La Figura 5.7 muestra los resultados. A diferencia de figuras anteriores, en este estudio se diferencia entre aciertos y fallos para visualizar el impacto del aumento del tamaño de cache en la tasa de aciertos. Los fallos de cache también generan desplazamientos en la racetrack al acceder en paralelo a los arrays de etiquetas y datos; sin embargo, no se han categorizado en la figura por razones de claridad en la muestra de resultados.

El tramo 0 tiende a aumentar a medida que aumenta la capacidad de cache. Esto se debe principalmente a que un mayor número de cabezales consiguen alinear a un mayor número de conjuntos consecutivos. Si comparamos las capacidades de 256 frente a 32 KB, el tramo 0 aumenta en promedio hasta un 17 % y 15 % para IPCs bajos y altos, respectivamente.

En lo que concierne a los tramos 1-3, 4-8 y 9-16, en promedio disminuyen conforme aumenta la capacidad de cache. Sin embargo, en algunos benchmarks encontramos resultados inesperados donde, a pesar de aumentar la capacidad de cache, estos tramos lejos



(a) IPCs bajos



(b) IPCs altos

Figura 5.7: Comparativa de aumento de capacidad de cache.

de disminuir, aumentan. Este comportamiento se observa en las aplicaciones *leslie3d*, *gcc*, *perlbench* y *gamess*. Esto se debe al efecto que provoca la distribución enlazada de conjuntos junto con la decodificación del conjunto en las direcciones al aumentar la capacidad de cache (el número de conjuntos). Este fenómeno se explica de manera detallada en la Sección 5.4.1.

En cuanto al tramo de fallos, como cabía esperar, se reduce conforme aumenta la capacidad de cache al eliminar los fallos de capacidad. Esto se aprecia claramente en aquellas aplicaciones con una menor tasa de acierto con 32 KB como *lbm*, *mcf* y *dealII*. La tasa de fallos media es 3, 2,5, 2 y 1 % para las capacidades de 32, 64, 128 y 256 KB, respectivamente.

En resumen, el aumento de la capacidad de cache permite, por un lado, aumentar el porcentaje de accesos con tramo 0, reduciendo así el número de operaciones de desplazamiento, y por otro lado, reducir el número de fallos de capacidad, lo cual también repercute favorablemente en las prestaciones del sistema.

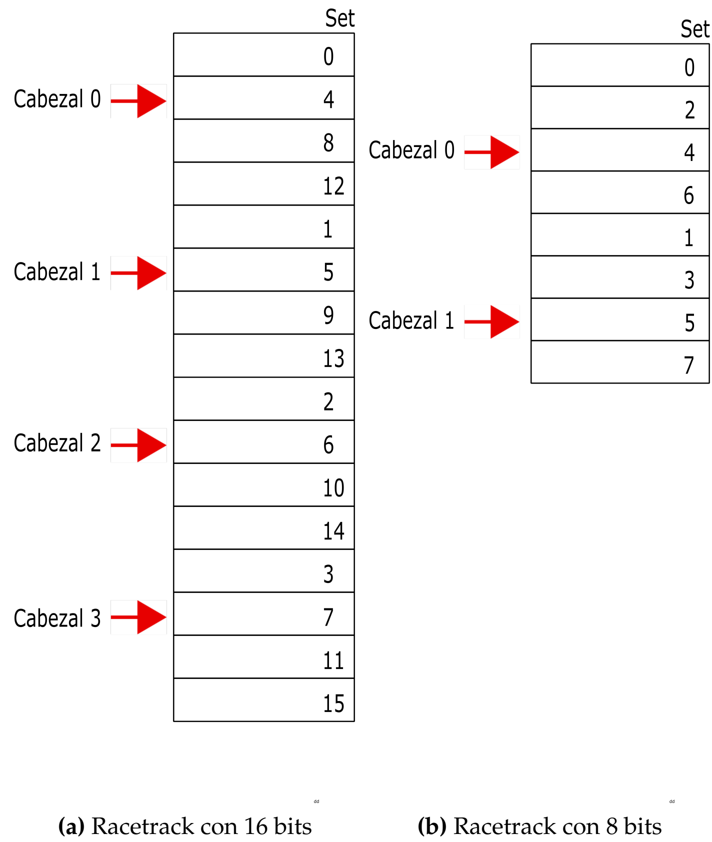
5.4.1. Análisis de la Variabilidad en Tramos Intermedios

Esta sección explica mediante un ejemplo sencillo qué situaciones generan un aumento en los tramos 1-3, 4-8 y 9-16 cuando aumenta la capacidad de cache. La Figura 5.8 muestra dos escenarios con racetracks gestionadas mediante una política de desplazamiento DL y organización de conjuntos entrelazadas. Una racetrack cuenta con 16 bits y 4 cabezales y la otra cuenta con 8 bits y 2 cabezales, manteniendo en ambas una distancia entre cabezales de 4 bits.

Supongamos que trabajamos con una cache indexada a nivel de byte, un tamaño de bloque de 16 bytes y realizamos un acceso a la dirección de memoria 0x040 en ambas racetracks. El campo de conjunto en la dirección estará formado por 4 y 3 bits en la racetrack de 16 y 8 conjuntos, respectivamente, pero en cualquier caso, en ambas racetracks accederíamos al conjunto 4. De esta manera, los cabezales 0 en cada racetrack se encuentran sobre el conjunto 4 cuando finaliza el acceso.

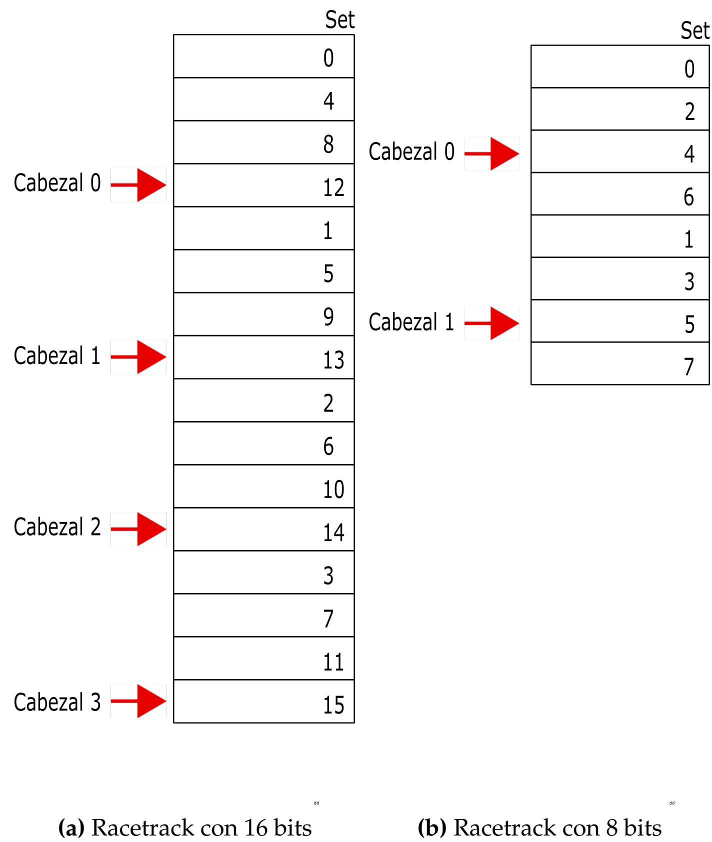
Partiendo de la situación anterior, supongamos ahora un acceso a la dirección 0x0C0. Una solicitud a esta dirección genera un acceso al conjunto 12 en la racetrack con 16 bits (1100) y un acceso al conjunto 4 en la racetrack con 8 bits (100). Por consiguiente, en la racetrack con 16 bits debemos realizar un desplazamiento de 2 bits hasta que el cabezal se encuentre encima del conjunto 12, mientras que en la racetrack con 8 bits no es necesario realizar tal desplazamiento al estar el cabezal 0 ya alineado con el conjunto 4. La Figura 5.9 ilustra el estado de los cabezales una vez se ha accedido al conjunto correspondiente.

El ejemplo ilustra cómo, al doblar cada vez la capacidad de cache en número de conjuntos, cambia la decodificación de las direcciones. Estos cambios pueden provocan aumentos de tramo de forma arbitraria.



(a) Racetrack con 16 bits **(b)** Racetrack con 8 bits

Figura 5.8: Posición de los cabezales tras acceder a la dirección 0x040.



(a) Racetrack con 16 bits **(b)** Racetrack con 8 bits

Figura 5.9: Posición de los cabezales tras acceder a la dirección 0x0C0.

CAPÍTULO 6
Conclusiones y Trabajo Futuro

Este capitulo expone una reflexión acerca de cómo las asignaturas del Grado de Informática han posibilitado la elaboración del trabajo, las conclusiones principales y qué tareas sería interesante realizar en el futuro a partir de los resultados obtenidos.

6.1 Relación del Trabajo Desarrollado con los Estudios Cursados

Para implementar las propuestas, ha sido necesario modificar y adaptar el simulador Multi2Sim para poder emular la tecnología DWM, lo que ha requerido la comprensión en profundidad del simulador, en especial la arquitectura x86 y su jerarquía de memoria, a la vez que se profundizaba en los conocimientos en lenguaje C adquiridos durante el Grado. Además, para el tratamiento de datos ha sido necesario recurrir a la realización de scripts para la automatización del proceso, escritos en Python3. En el caso de Python 3, que ya había sido estudiado en el Grado, ha sido necesario cambiar el enfoque hacia el tratamiento de grandes volúmenes de datos de manera automatizada. A su vez se han utilizado diversas herramientas nuevas que no habían sido estudiadas en el Grado como el depurador de código GDB o el conjunto de aplicaciones SPEC CPU 2006. Gracias a las habilidades adquiridas durante el Grado se ha visto facilitado el aprendizaje de las mismas.

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 en diversas asignaturas, siendo las más destacables Fundamentos de Computadores (FCO), Estructura de Computadores (ETC), Arquitectura e Ingeniería de Computadores (AIC) y Arquitecturas Avanzadas (AAV).

En relación a las competencias transversales trabajadas durante el Grado, 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:

- **Análisis y resolución de problemas.** Esta competencia ha sido esencial ya que tanto los resultados como el trabajo realizado han requerido un análisis constante, sumado a la resolución de los problemas que revelaban estos análisis.

- **Innovación, creatividad y emprendimiento.** Este trabajo tiene un carácter innovador puesto que abre una línea de investigación prometedora. La creatividad ha sido importante a la hora de ofrecer soluciones nuevas que no se habían planteado hasta entonces. En cuanto al emprendimiento, como trabajo futuro, si las propuestas tienen una buena acogida entre la comunidad científica, podrían establecerse contactos con empresas de cara a fabricar el producto.

- **Conocimiento de problemas contemporáneos.** La primera parte de este proyecto consistió en una documentación sobre los problemas que presentan las memorias cache de los procesadores actuales, lo que requirió conocer los problemas contemporáneos en el área concreta de la arquitectura y tecnología de computadores.

- **Instrumental específica.** Se han utilizado herramientas específicas del área como el simulador Multi2Sim, o generales como el depurador GDB o el entorno de programación Pycharm.

- **Planificación y gestión del tiempo.** Debido al tiempo asociado a cada simulación, ha sido imprescindible planificar tanto los lanzamientos como los tiempos de obtención y procesamiento de resultados. Además, se han marcado unos hitos con unas fechas asociadas para encauzar el trabajo.

6.2 Conclusiones

Las conclusiones se dividen en dos grandes bloques, por un lado las conclusiones de carácter técnico y por otro lado las conclusiones académicas relacionadas con el trabajo. Estas conclusiones se exponen a continuación a través de los objetivos del trabajo.

6.2.1. Objetivos Técnicos

El primer objetivo, instrumentar el simulador Multi2Sim para modelar memorias basadas en tecnología DWM, se ha superado ampliamente. Se han implementado no sólo los detalles de diseño para modelar DWM en el simulador, sino también parámetros de entrada que permiten seleccionar cómoda y eficientemente diversas políticas de gestión de cabezales, organización de conjuntos directa o entrelazada, cantidad de cabezales por racetrack, entre otros.

El segundo objetivo de carácter técnico supone implementar las tres políticas de desplazamiento presentes en la literatura (SL, DL y SE) y medir sus prestaciones en la cache L1. Se ha concluido a partir de los resultados obtenidos que DL es la política que ofrece mejores prestaciones debido a que reduce el número de desplazamientos con mayor penalización a la vez que hace aumentar el número de accesos sin necesidad de desplazamiento (tramo 0).

El tercer objetivo es diseñar, implementar, validar y evaluar experimentalmente una propuesta de distribución de conjuntos de cache entrelazados que se ajuste a las características de la localidad espacial y temporal de los datos en L1. Asumiendo una política de desplazamiento DL, los resultados han mostrado que la propuesta de organización entrelazada aumenta significativamente el tramo 0 frente a una organización convencional (mapeado directo). Por tanto, se puede afirmar que, para el caso de caches L1 basadas en DWM, el entrelazado a nivel de conjuntos en base al número de cabezales mejora las prestaciones respecto al mapeado directo.

El último objetivo de carácter técnico supone evaluar el impacto del aumento de la capacidad de almacenamiento sobre los desplazamientos en la racetrack de acuerdo con

la mayor densidad que ofrece la tecnología DWM frente a SRAM. Se concluye a partir de este objetivo que existe una relación entre el aumento de la capacidad de cache con el aumento del tramo nulo de desplazamiento. Por tanto, aumentar la capacidad de cache también beneficia a las prestaciones del sistema, no sólo por la reducción de fallos de capacidad en la cache sino también por la reducción del impacto de los desplazamientos.

6.2.2. Objetivos Académicos

En cuanto a los objetivos de carácter académico, el primero de ellos es aprender y profundizar en un simulador ciclo-a-ciclo de procesadores utilizado en la industria y en la academia. Este objetivo se ha cumplido ampliamente ya que para desarrollar este proyecto ha sido indispensable profundizar tanto en la documentación del simulador como en el código para posteriormente modificarlo.

El segundo objetivo supone la consulta y estudio del estado-del-arte sobre tecnologías de memoria no volátiles, profundizando en DWM. El capítulo 2 es la materialización de este objetivo. En él se describen las principales tecnologías de memoria, incluyendo un análisis detallado de DWM.

El tercer objetivo es estudiar en profundidad la arquitectura x86, especialmente la jerarquía de memoria. Este apartado ha requerido un estudio teórico previo, que se refleja en la originalidad de las propuestas. Además, la instrumentación del simulador para dar soporte a caches basadas en DWM y la implementación de las propuestas ha sido posible gracias a la consecución de este objetivo.

El cuarto objetivo es desarrollar herramientas que conviertan automáticamente los datos en bruto del simulador en resultados que puedan ser interpretados con facilidad. Mediante *scripting* de Python se han realizado *scripts* específicos en base a la información que se necesita extraer. Esto se aprecia no sólo con las gráficas y datos presentes en este trabajo, sino también en el Anexo A.

6.3 Trabajo Futuro

Dada la naturaleza de este trabajo y el tiempo limitado, existen diversas extensiones del trabajo que no han podido concretarse. Esta sección expone el rumbo que seguirá la presente línea de investigación una vez concluido este trabajo.

La organización de los datos en la memoria cache se podría revisar para seguir aumentando el tramo nulo de desplazamiento. En este sentido, se podría proponer una organización que no se limitara a dividir físicamente las vías en bancos de memoria distintos. De esta manera, se podría cubrir un mayor número de conjuntos consecutivos sin necesidad de realizar ningún desplazamiento.

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 [35] y CACTI [36].

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.

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] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

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

[31] 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.

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

[33] 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.

[34] T. K. Prakash and L. Peng, "Performance Characterization of SPEC CPU 2006 Benchmarks on Intel Core 2 Duo Processor," in *Proceedings of the International Conference on Parallel Processing*, 2008, pp. 1–6.

[35] 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.

[36] 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.

APÉNDICE A
Scripts de Tratamiento de Datos

En el presente anexo se explica de forma concisa los *scripts* realizados para la extracción de datos en Python. El objetivo es poder consultar el código de extracción de datos de forma sencilla si se requiriese. De esta manera se adjuntan los ejecutables que tienen como objetivo el tratamiento de los datos para obtener resultados interpretables rápidamente y sin esfuerzo.

A.1 createDics.py

Esta función se encarga de recorrer el directorio donde se encuentran las aplicaciones de SPEC CPU 2006 y genera un diccionario de aplicaciones, que a su vez contiene una estructura matricial donde se guardan los datos a tratar. Esta función se utiliza posteriormente por todos los *scripts* para generar matrices de datos de las aplicaciones.

```
from __future__ import print_function
import pandas as pd
import os

#Def dictionarios anidados
def create_dic(path):
    pid_dictionary = dict()

    for bench in sorted(os.listdir(path)):
        int_reps_dir = path + bench + "/interval_reports/x86_ctx/"


        if len(os.listdir(path)) < 1:
            print("Benchmark " + bench + " err: no input files available")
            continue

        pid_dict = dict()
        for pid in os.listdir(int_reps_dir):
            if pid.endswith("~") == False:
                pidid = pid.split('.')[0]
                data_frame = pd.read_csv(int_reps_dir + pid, sep=',',
                header=0, usecols=None)
                pid_dict[pidid] = data_frame
```

```
        pid_dictionary[bench] = pid_dict

    return pid_dictionary



def create_dic_mod(path):
    pid_dictionary = dict()

    for bench in sorted(os.listdir(path)):
        int_reps_dir = path + bench + "/interval_reports/mod/"

        if len(os.listdir(path)) < 1:
            print("Benchmark " + bench + " err: no input files available")
            continue

        pid_dict = dict()
        for pid in os.listdir(int_reps_dir):
            if (pid == "dl1-0.intrep.csv"):
                if pid.endswith("~") == False:
                    pidid = pid.split('.')[0]
                    data_frame = pd.read_csv(int_reps_dir + pid, sep=',',
                                             header=0, usecols=None)
                    pid_dict[pidid] = data_frame

        pid_dictionary[bench] = pid_dict

    return pid_dictionary

def create_dic_p(path):
    pid_dictionary = dict()

    for bench in sorted(os.listdir(path)):
        # MOD AQUI
        int_reps_dir = path + bench + "/interval_reports/mod/"

        if len(os.listdir(path)) < 1:
            print("Benchmark " + bench + " err: no input files available")
            continue

        pid_dict = dict()
        for pid in os.listdir(int_reps_dir):
            if (pid == "dl1-0.intrep.csv"):
                if pid.endswith("~") == False:
                    pidid = pid.split('.')[0]
                    data_frame = pd.read_csv(int_reps_dir + pid, sep=',',
                                             header=0, usecols=None)
                    pid_dict[pidid] = data_frame

        pid_dictionary[bench] = pid_dict
```

```
    return pid_dictionary
```

A.2 generarCondorFile.py

Este *script* devuelve como resultado un fichero de texto para lanzarlo a la cola de Condor mediante el comando *condor_submit*. Lo genera a partir de una serie de parámetros como la ubicación de los ficheros de configuración de Multi2Sim, la cantidad de instrucciones a ejecutar, los periodos de volcado de datos, etcétera.

```
from __future__ import print_function
from sys import argv

import sys
import os

def files( path ):
        for file in os.listdir(path):
                if os.path.isfile(os.path.join(path, file)):
                        yield file

def create_file_header( condor_file, m2s_path ):
        condor = open(condor_file, "w")
        print ("+GPBatchJob = true", file=condor, end="\n")
        print ("+LongRunningJob = true", file=condor, end="\n")
        print ("Rank = -LoadAvg", file=condor, end="\n")
        print ("Universe = vanilla", file=condor, end="\n")
        print ("Executable = " + m2s_path + "/bin/m2s", file=condor,
         ↪  end="\n")
        print ("Environment = LD_LIBRARY_PATH=$ENV(LD_LIBRARY_PATH)",
         ↪  file=condor, end="\n\n")


if len(argv) < 4:
        print ("Usage: " + argv[0] + " <ctx_dir> <m2s_root_path>
         ↪  <condor_fname> <extra_args>", end="\n")
        exit()

script, path, m2s_root_path, out_file, extra_args = argv

src_path =  m2s_root_path + "src/"
sim_type = "--x86-sim detailed "
cpu_config = "--x86-config " + src_path +
 ↪  "configuraciones-sample/baseline_DDR4/4vias/cpuconfig_baseline "
mem_config = "--mem-config " + src_path +
 ↪  "configuraciones-sample/baseline_DDR4/4vias/cacheconf_4cores_512KB
 ↪  _2MB_baseline "
net_config = "--net-config " + src_path +
 ↪  "configuraciones-sample/baseline_DDR4/4vias/net_4core_baseline "
max_inst = "--x86-min-inst-per-ctx " + str(1000000000) + " "
ep_length = "--epoch-length " + str(12000000) + " "
```

```
rep_dir = "--reports-dir
↪ /nfs/alumnos/hutarsan/reports/baseline_DDR4/4vias/"

i = 0

create_file_header(out_file, m2s_root_path)

for name in files(path):
        if not name.find("ctxconfig.") < 0:
                print ("File: " + name, end="\n")
                ctx_config = "--ctx-config " + path + name + " "
                condor = open(out_file, "a")
                print ("Arguments = " + sim_type + cpu_config + mem_config
                ↪ + net_config + ctx_config + max_inst + ep_length +
                ↪ rep_dir + name.split(".")[1] + extra_args,
                ↪ file=condor, end="\n\n")
                print ("Log =
                ↪ /nfs/alumnos/hutarsan/outputs/baseline_DDR4/4vias/" +
                ↪ name.split(".")[1] + ".log", file=condor, end="\n")
                print ("Output =
                ↪ /nfs/alumnos/hutarsan/outputs/baseline_DDR4/4vias/" +
                ↪ name.split(".")[1] + ".out", file=condor, end="\n")
                print ("Error =
                ↪ /nfs/alumnos/hutarsan/outputs/baseline_DDR4/4vias/" +
                ↪ name.split(".")[1]  + ".err", file=condor, end="\n")

                print ("Queue", file=condor, end="\n\n")
                i = i +1
```

A.3 missHitsAmontonadoPercnt.py

Este *script* genera una matriz con el porcentaje de desplazamientos dividido en los tramos 0, 1-3, 4-8 y 9-16 de la cache de datos L1. Además, divide este porcentaje entre aciertos y fallos. Recibe como parámetro la ubicación de los resultados de los benchmarks, el tamaño del conjunto y la cantidad de cabezales.

```
import pandas as pd
from sys import argv
import re
import createDics as cd


if len(argv) < 2:
    print("Usage: <Script> <Benchmark_path> <Set size> <Cabezales>\n")
    exit()
benchmark = argv[1].split("/")
print(argv)
longitud = len(benchmark)
benchmark = benchmark[longitud - 2]
rango =int(argv[3]) -1
```

```
aux = int(argv[4])
cabezales = int(aux)
# Var declaration
cond = True
i = 0
via = 0
ciclos_p = 0
sets = 0
num = 0
sets = 0
max = 0
vias = re.findall(r"(\d+)vias", argv[1])
vias = vias[0]
vias = int(vias)
total = 0
names = []
benchs = cd.create_dic(argv[1])
# df = pd.DataFrame(argv[1]+"interval_reports/mod/dl1-0.intrep.csv")
accesos = 0
hits = 0
misses = 0

names = []
t1 = []
t2 = []
t3 = []
t4 = []
t5 = []
miss1 = []
miss2 = []
miss3 = []
miss4 = []
miss5 = []
totalx = []
dic = dict()
acumulada = 0
#Variables para la media con 4 tramos
#mt0 = 0;
 #mt1 = 0;
#mt2 = 0;
#mt3 = 0;

# npy.zeros((n,n))
# Main

for benchmark,arc_name in benchs.items():
    #print(benchmark)
    #rint("\n")
    for arc_n,df in arc_name.items():
        df = pd.read_csv(argv[1]+benchmark+
         ↪  "/interval_reports/mod/dl1-0.intrep.csv")
    #Geting the name df
```

```
lrow = len(df.index) -1
names.append(benchmark)
total = 0
acumulada = 0
tope = int((rango+1)/cabezales)
for x in range(vias):
        for y in range(0,tope):
                total = total + df.at[lrow,
                 ↪  "dl1-0-c4t1-via-"+str(x)+"-ciclos-penalizacion-"
                 ↪  +str(y)]

totalx.append(total)
# print("[DEBUG] "+str(benchmark)+ ": " +str(media_t))
for x in range(0,vias):
        hits = hits+  df.at[lrow,
         ↪  "dl1-0-c4t1-via-"+str(x)+"-ciclos-penalizacion hits-0"]
        misses= misses + df.at[lrow,
         ↪  "dl1-0-c4t1-via-"+str(x)+"-ciclos-penalizacion misses-0"]
t1.append((hits/total)*100)
miss1.append((misses/total)*100)
acumulada+= (hits/total)*100 + (misses/total)*100
hits = 0
misses = 0

for x in range(0,vias):
        for y in range(1,4):
            hits = hits + df.at[lrow, "dl1-0-c4t1-via-" + str(x) +
             ↪  "-ciclos-penalizacion hits-"+str(y)]
            misses = misses + df.at[lrow, "dl1-0-c4t1-via-" + str(x) +
             ↪  "-ciclos-penalizacion misses-"+str(y)]
t2.append((hits/total)*100)
miss2.append((misses/total)*100)
acumulada += (hits / total) * 100 + (misses / total) * 100
hits = 0
misses = 0

for x in range(0,vias):
        for y in range(4,9):
            hits = hits + df.at[lrow, "dl1-0-c4t1-via-" + str(x) +
             ↪  "-ciclos-penalizacion hits-"+str(y)]
            misses = misses + df.at[lrow, "dl1-0-c4t1-via-" + str(x) +
             ↪  "-ciclos-penalizacion misses-"+str(y)]
t3.append((hits/total)*100)
miss3.append((misses/total)*100)
acumulada += (hits / total) * 100 + (misses / total) * 100
hits = 0
misses = 0

for x in range(0,vias):
        for y in range(9,17):
            hits = hits + df.at[lrow, "dl1-0-c4t1-via-" + str(x) +
             ↪  "-ciclos-penalizacion hits-"+str(y)]
```

```
                misses = misses + df.at[lrow, "dl1-0-c4t1-via-" + str(x) +
                 ↪  "-ciclos-penalizacion misses-"+str(y)]
    t4.append((hits/total)*100)
    miss4.append((misses/total)*100)
    acumulada += (hits / total) * 100 + (misses / total) * 100
    hits = 0
    misses = 0

    for x in range(0,vias):
            for y in range(17,tope):
                hits = hits + df.at[lrow, "dl1-0-c4t1-via-" + str(x) +
                 ↪  "-ciclos-penalizacion hits-"+str(y)]
                misses = misses + df.at[lrow, "dl1-0-c4t1-via-" + str(x) +
                 ↪  "-ciclos-penalizacion misses-"+str(y)]
    t5.append((hits/total)*100)
    miss5.append((misses/total)*100)
    acumulada += (hits / total) * 100 + (misses / total) * 100
    hits = 0
    misses = 0

#names.append("Media global")
#t1.append((mt0/(mt0+mt1+mt2+mt3))*100)
#t2.append((mt1/(mt0+mt1+mt2+mt3))*100)
#t3.append((mt2/(mt0+mt1+mt2+mt3))*100)
#t4.append((mt3/(mt0+mt1+mt2+mt3))*100)



#[DEBUG]
#print(len(t1))
#print(len(t2))
#print(len(t3))
#print(len(t4))




dic["Benchmark"] = names
dic["Tramo 0 hits %"] = t1
dic["Tramo 1-3 hits %"] = t2
dic["Tramo 4-8 hits %"] = t3
dic["Tramo 9-16 hits %"] = t4
dic["Tramo +16 hits %"] = t5
dic["Tramo 0 misses %"] = miss1
dic["Tramo 1-3 misses %"] = miss2
dic["Tramo 4-8 misses %"] = miss3
dic["Tramo 9-16 misses %"] = miss4
dic["Tramo +16 misses %"] = miss5
dic["Acumulada"] = acumulada


dff = pd.DataFrame(dic)
```

```
print(dff)
dff.to_csv(argv[2], sep= ",",index=False )
```

A.4 missHitsAmontonadoAbsoluto.py

Este *script* genera una matriz con el porcentaje de desplazamientos dividido en los tramos 0, 1-3, 4-8 y 9-16 de la cache de datos L1. Además, divide estos valores entre número absoluto de aciertos y fallos. Recibe como parámetro la ubicación de los resultados de los benchmarks, el tamaño del conjunto y la cantidad de cabezales.

```
import pandas as pd
from sys import argv
import re
import createDics as cd


if len(argv) < 2:
    print("Usage: <Script> <Benchmark_path> <Set size> <Headers>\n")
    exit()
benchmark = argv[1].split("/")
print(argv)
longitud = len(benchmark)
benchmark = benchmark[longitud - 2]
rango =int(argv[3]) -1
cabezales = int(argv[4])
# Var declaration
cond = True
i = 0
via = 0
ciclos_p = 0
sets = 0
num = 0
sets = 0
max = 0
vias = re.findall(r"(\d+)vias", argv[1])
vias = vias[0]
vias = int(vias)
total = 0
names = []
benchs = cd.create_dic(argv[1])
# df = pd.DataFrame(argv[1]+"interval_reports/mod/dl1-0.intrep.csv")
accesos = 0
hits = 0
misses = 0

names = []
t1 = []
t2 = []
t3 = []
t4 = []
t5 = []
```

```
miss1 = []
miss2 = []
miss3 = []
miss4 = []
miss5 = []
totalx = []
dic = dict()
#Variables para la media con 4 tramos
#mt0 = 0;
 #mt1 = 0;
#mt2 = 0;
#mt3 = 0;

# npy.zeros((n,n))
# Main

for benchmark,arc_name in benchs.items():
    #print(benchmark)
    #rint("\n")
    for arc_n,df in arc_name.items():
        df = pd.read_csv(argv[1]+benchmark+
         ↪ "/interval_reports/mod/dl1-0.intrep.csv")
    lrow = len(df.index) -1
    names.append(benchmark)
    total = 0
    tope = int((rango + 1) / cabezales)
    #print(vias)
    #print(benchmark)
    for x in range(0,vias):
            for y in range(0,tope):
                    total = total + df.at[lrow, "dl1-0-c4t1-via-"+str(x)+
                     ↪ "-ciclos-penalizacion-"+str(y)]

    totalx.append(total)
    # print("[DEBUG] "+str(benchmark)+ ": " +str(media_t))
    for x in range(0,vias):
            hits = hits+  df.at[lrow, "dl1-0-c4t1-via-"+str(x)+
             ↪ "-ciclos-penalizacion hits-0"]
            misses= misses + df.at[lrow, "dl1-0-c4t1-via-"+str(x)+
             ↪ "-ciclos-penalizacion misses-0"]
    t1.append(hits)
    miss1.append(misses)
    hits = 0
    misses = 0

    for x in range(0,vias):
            for y in range(1,4):
                hits = hits + df.at[lrow, "dl1-0-c4t1-via-" + str(x) +
                 ↪ "-ciclos-penalizacion hits-"+str(y)]
                misses = misses + df.at[lrow, "dl1-0-c4t1-via-" + str(x) +
                 ↪ "-ciclos-penalizacion misses-"+str(y)]
    t2.append(hits)
```

```
    miss2.append(misses)
    hits = 0
    misses = 0

    for x in range(0,vias):
            for y in range(4,9):
                hits = hits + df.at[lrow, "dl1-0-c4t1-via-" + str(x) +
                 ↪  "-ciclos-penalizacion hits-"+str(y)]
                misses = misses + df.at[lrow, "dl1-0-c4t1-via-" + str(x) +
                 ↪  "-ciclos-penalizacion misses-"+str(y)]
    t3.append(hits)
    miss3.append(misses)
    hits = 0
    misses = 0

    for x in range(0,vias):
            for y in range(9,17):
                hits = hits + df.at[lrow, "dl1-0-c4t1-via-" + str(x) +
                 ↪  "-ciclos-penalizacion hits-"+str(y)]
                misses = misses + df.at[lrow, "dl1-0-c4t1-via-" + str(x) +
                 ↪  "-ciclos-penalizacion misses-"+str(y)]
    t4.append(hits )
    miss4.append(misses )
    hits = 0
    misses = 0
    for x in range(0,vias):
            for y in range(17,tope):
                hits = hits + df.at[lrow, "dl1-0-c4t1-via-" + str(x) +
                 ↪  "-ciclos-penalizacion hits-"+str(y)]
                misses = misses + df.at[lrow, "dl1-0-c4t1-via-" + str(x) +
                 ↪  "-ciclos-penalizacion misses-"+str(y)]
    t5.append(hits )
    miss5.append(misses)
    hits = 0
    misses = 0

#names.append("Media global")
#t1.append((mt0/(mt0+mt1+mt2+mt3))*100)
#t2.append((mt1/(mt0+mt1+mt2+mt3))*100)
#t3.append((mt2/(mt0+mt1+mt2+mt3))*100)
#t4.append((mt3/(mt0+mt1+mt2+mt3))*100)



#[DEBUG]
#print(len(t1))
#print(len(t2))
#print(len(t3))
#print(len(t4))
```

```
dic["Benchmark"] = names
dic["Tramo 0 hits"] = t1
dic["Tramo 1-3 hits"] = t2
dic["Tramo 4-8 hits"] = t3
dic["Tramo 9-16 hits"] = t4
dic["Tramo +16 hits"] = t5
dic["Tramo 0 misses"] = miss1
dic["Tramo 1-3 misses"] = miss2
dic["Tramo 4-8 misses"] = miss3
dic["Tramo 9-16 misses"] = miss4
dic["Tramo +16 misses"] = miss5
dic["Total"] = totalx


dff = pd.DataFrame(dic)
print(tope)
print(dff)
dff.to_csv(argv[2], sep= ",",index=False )
```

A.5 mpkisToCsv.py

Este ejecutable obtiene los fallos de predicción por kilo-instrucción. Divide el MPKI entre MPKI de cache L1, L2 y L3 y lo genera a partir de las estructuras matriciales de datos generadas en createDics.py

```
import createDics as cd
import pandas as pd
from sys import argv

if len(argv) < 3:
    print("Usage: <Script> <ReportsDir> <nameCSV>\n")
    exit()

dic = dict()
benchs = cd.create_dic(argv[1])
aux1 = []
aux2 = []
aux3=[]
aux4= []


num = 0
for benchmark,arc_name in benchs.items():
    #print(benchmark)
    #rint("\n")
    for arc_n,df in arc_name.items():
        # print(benchmark)
        # print(df["pid100-l1-misses-int"])
        #L1 MISSES
        misses= sum(df["pid100-l1-misses-int"])
```

```
        #print(benchmark + " misses-> " + str(misses))
        lrow = len(df.index) -1
        inst = df.at[lrow, "pid100-uinsts"]
        #print(benchmark + "inst-> " + str(inst))
        aux1.append(benchmark)
        aux2.append(misses/(inst/1000))
        #print(benchmark + " MPKI_L1-> " + str(inst))
        #print(misses)
        #print(inst) #bien
        #L2 MISSES
        misses = sum(df["pid100-l2-misses-int"])
        inst = df.at[lrow, "pid100-uinsts"]
        aux3.append(1000 * misses / inst)
        #L3 MISSES
        misses = sum(df["pid100-l3-misses-int"])
        inst = df.at[lrow, "pid100-uinsts"]
        aux4.append(1000 * misses / inst)

dic["Benchmark"] = aux1
dic["mpki_L1"] = aux2
dic["mpki_L2"] = aux3
dic["mpki_L3"] = aux4



dff = pd.DataFrame(dic)
print(dff)
dff.to_csv(argv[2], sep= ",",index=False )
```

A.6 porcentajesHitsMissesL1.py

Este ejecutable devuelve el porcentaje de fallos y aciertos totales en la cache de datos L1. También devuelve el número de accesos a esta cache y el número absoluto de fallos y aciertos.

```
import createDics as cd
import pandas as pd
from sys import argv

if len(argv) < 3:
    print("Usage: <Script> <ReportsDir> <nameCSV>\n")
    exit()



dic = dict()
accesos_L1= 0
accesos_totales=0


benchs = cd.create_dic_mod(argv[1])
```

```
aux1= []
aux2= []
aux3= []
aux4= []
aux5=[]
aux6=[]


for benchmark,arc_name in benchs.items():
    for arc_n,df in arc_name.items():
        #MRU_hits_totales.append(MRU_hits)
        accesos_L1 = sum(df["dl1-0-hits-int"]) +
        ↪  sum(df["dl1-0-misses-int"])
        aciertos_L1 = sum(df["dl1-0-hits-int"])
        fallos_L1 = sum(df["dl1-0-misses-int"])
        lrow = len(df.index) - 1


        aux1.append(benchmark)
        aux2.append(accesos_L1)
        aux3.append(aciertos_L1)
        aux4.append(fallos_L1)
        aux5.append((aciertos_L1/accesos_L1)*100)
        aux6.append((fallos_L1/accesos_L1)*100)



dic[" Benchmarks"] = aux1
dic["Accesos L1-d"] = aux2
dic["Aciertos L1-d"] = aux3
dic[" Fallos L1-d"] = aux4
dic["Porcentaje aciertos L1-d"] = aux5
dic["Porcentaje fallos L1-d"] = aux6

dff = pd.DataFrame(dic)
print(dff)
dff.to_csv(argv[2], sep= ",",index=False )
```

A.7 ejecucionAmontonados.py

Este *script* utiliza missHitsAmontonadoAbsoluto.py y missHitsAmontonadoAbsolutoPercnt.py para automatizar la obtención de datos. Devuelve los resultados para 32 KB con 4 cabezales, 64 KB con 8 cabezales, 128 KB con 16 cabezales y 256 KB con 32 cabezales con 4 y 8 vías. Estos resultados por tramos muestran tanto la representación absoluta como la representación porcentual.

```
import sys

print("EMPEZANDO AMONTONADO ABSOLUTO")
script_descriptor = open("miss_hits_amontonado_absoluto.py")
miss_hits_amontonado_absoluto = script_descriptor.read()
```

```
sys.argv = ["miss_hits_amontonado_absoluto.py",
↪ "/home/hutarsan/Documentos/
↪ racetrack/reports/ent_comparativa/DL/4vias/256kb_32cap/",
↪ "/home/hutarsan/Documentos/racetrack/scripts
↪ /256kb_32cap_amontonado_4v.csv","1024","32" ]
exec(miss_hits_amontonado_absoluto)
script_descriptor.close()


script_descriptor = open("miss_hits_amontonado_absoluto.py")
miss_hits_amontonado_absoluto = script_descriptor.read()
sys.argv = ["miss_hits_amontonado_absoluto.py",
↪ "/home/hutarsan/Documentos/
↪ racetrack/reports/ent_comparativa/DL/4vias/128kb_16cap/",
↪ "/home/hutarsan/Documentos/racetrack/scripts
↪ /128kb_16cap_amontonado_4v.csv","512","16"]
exec(miss_hits_amontonado_absoluto)
script_descriptor.close()


script_descriptor = open("miss_hits_amontonado_absoluto.py")
miss_hits_amontonado_absoluto = script_descriptor.read()
sys.argv = ["miss_hits_amontonado_absoluto.py",
↪ "/home/hutarsan/Documentos/
↪ racetrack/reports/ent_comparativa/DL/4vias/64kb_8cap/",
↪ "/home/hutarsan/Documentos/racetrack/scripts
↪ /64kb_8cap_amontonado_4v.csv","256","8"]
exec(miss_hits_amontonado_absoluto)
script_descriptor.close()


script_descriptor = open("miss_hits_amontonado_absoluto.py")
miss_hits_amontonado_absoluto = script_descriptor.read()
sys.argv = ["miss_hits_amontonado_absoluto.py",
↪ "/home/hutarsan/Documentos/
↪ racetrack/reports/ent_comparativa/DL/4vias/32kb_4cap/",
↪ "/home/hutarsan/Documentos/racetrack/scripts
↪ /32kb_4cap_amontonado_4v.csv","128","4"]
exec(miss_hits_amontonado_absoluto)
script_descriptor.close()


script_descriptor = open("miss_hits_amontonado_absoluto.py")
miss_hits_amontonado_absoluto = script_descriptor.read()
sys.argv = ["miss_hits_amontonado_absoluto.py",
↪ "/home/hutarsan/Documentos/
↪ racetrack/reports/ent_comparativa/DL/8vias/256kb_32cap/",
↪ "/home/hutarsan/Documentos/racetrack/scripts
↪ /256kb_32cap_amontonado_8v.csv","512","32"]
exec(miss_hits_amontonado_absoluto)
script_descriptor.close()
```

```
script_descriptor = open("miss_hits_amontonado_absoluto.py")
miss_hits_amontonado_absoluto = script_descriptor.read()
sys.argv = ["miss_hits_amontonado_absoluto.py",
↪ "/home/hutarsan/Documentos/
↪ racetrack/reports/ent_comparativa/DL/8vias/128kb_16cap/",
↪ "/home/hutarsan/Documentos/racetrack/scripts
↪ /128kb_16cap_amontonado_8v.csv","256","16"]
exec(miss_hits_amontonado_absoluto)
script_descriptor.close()


script_descriptor = open("miss_hits_amontonado_absoluto.py")
miss_hits_amontonado_absoluto = script_descriptor.read()
sys.argv = ["miss_hits_amontonado_absoluto.py",
↪ "/home/hutarsan/Documentos/
↪ racetrack/reports/ent_comparativa/DL/8vias/64kb_8cap/",
↪ "/home/hutarsan/Documentos/racetrack/scripts
↪ /64kb_8cap_amontonado_8v.csv","128","8"]
exec(miss_hits_amontonado_absoluto)
script_descriptor.close()


script_descriptor = open("miss_hits_amontonado_absoluto.py")
miss_hits_amontonado_absoluto = script_descriptor.read()
sys.argv = ["miss_hits_amontonado_absoluto.py",
↪ "/home/hutarsan/Documentos/
↪ racetrack/reports/ent_comparativa/DL/8vias/32kb_4cap/",
↪ "/home/hutarsan/Documentos/racetrack/scripts
↪ /32kb_4cap_amontonado_8v.csv","64","4"]
exec(miss_hits_amontonado_absoluto)
script_descriptor.close()

print("ACABADO AMONTONADO ABSOLUTO")


script_descriptor = open("miss_hits_%_amontonado.py")
miss_hits_porcentaje_amontonado = script_descriptor.read()
sys.argv = ["miss_hits_%_amontonado.py", "/home/hutarsan/Documentos/
↪ racetrack/reports/ent_comparativa/DL/8vias/32kb_4cap/",
↪ "/home/hutarsan/Documentos/racetrack/scripts
↪ /32kb_4cap_amontonado_8v_%.csv","64","4"]
exec(miss_hits_porcentaje_amontonado)
script_descriptor.close()

script_descriptor = open("miss_hits_%_amontonado.py")
miss_hits_porcentaje_amontonado = script_descriptor.read()
sys.argv = ["miss_hits_%_amontonado.py", "/home/hutarsan/Documentos/
↪ racetrack/reports/ent_comparativa/DL/8vias/64kb_8cap/",
↪ "/home/hutarsan/Documentos/racetrack/scripts
↪ /64kb_8cap_amontonado_8v_%.csv","128","8"]
exec(miss_hits_porcentaje_amontonado)
```

```
script_descriptor.close()

script_descriptor = open("miss_hits_%_amontonado.py")
miss_hits_porcentaje_amontonado = script_descriptor.read()
sys.argv = ["miss_hits_%_amontonado.py", "/home/hutarsan/Documentos/
↪ racetrack/reports/ent_comparativa/DL/8vias/128kb_16cap/",
↪ "/home/hutarsan/Documentos/racetrack/scripts
↪ /128kb_16cap_amontonado_8v_%.csv","256","16"]
exec(miss_hits_porcentaje_amontonado)
script_descriptor.close()

script_descriptor = open("miss_hits_%_amontonado.py")
miss_hits_porcentaje_amontonado = script_descriptor.read()
sys.argv = ["miss_hits_%_amontonado.py", "/home/hutarsan/Documentos/
↪ racetrack/reports/ent_comparativa/DL/8vias/256kb_32cap/",
↪ "/home/hutarsan/Documentos/racetrack/scripts
↪ /256kb_32cap_amontonado_8v_%.csv","512","32"]
exec(miss_hits_porcentaje_amontonado)
script_descriptor.close()

script_descriptor = open("miss_hits_%_amontonado.py")
miss_hits_porcentaje_amontonado = script_descriptor.read()
sys.argv = ["miss_hits_%_amontonado.py", "/home/hutarsan/Documentos/
↪ racetrack/reports/ent_comparativa/DL/4vias/256kb_32cap/",
↪ "/home/hutarsan/Documentos/racetrack/scripts
↪ /256kb_32cap_amontonado_4v_%.csv","1024","32"]
exec(miss_hits_porcentaje_amontonado)
script_descriptor.close()

script_descriptor = open("miss_hits_%_amontonado.py")
miss_hits_porcentaje_amontonado = script_descriptor.read()
sys.argv = ["miss_hits_%_amontonado.py", "/home/hutarsan/Documentos/
↪ racetrack/reports/ent_comparativa/DL/4vias/128kb_16cap/",
↪ "/home/hutarsan/Documentos/racetrack/scripts
↪ /128kb_16cap_amontonado_4v_%.csv","512","16"]
exec(miss_hits_porcentaje_amontonado)
script_descriptor.close()

script_descriptor = open("miss_hits_%_amontonado.py")
miss_hits_porcentaje_amontonado = script_descriptor.read()
sys.argv = ["miss_hits_%_amontonado.py", "/home/hutarsan/Documentos/
↪ racetrack/reports/ent_comparativa/DL/4vias/64kb_8cap/",
↪ "/home/hutarsan/Documentos/racetrack/scripts
↪ /64kb_8cap_amontonado_4v_%.csv","256","8"]
exec(miss_hits_porcentaje_amontonado)
script_descriptor.close()

script_descriptor = open("miss_hits_%_amontonado.py")
miss_hits_porcentaje_amontonado = script_descriptor.read()
```

```
sys.argv = ["miss_hits_%_amontonado.py", "/home/hutarsan/Documentos/
↪   racetrack/reports/ent_comparativa/DL/4vias/32kb_4cap/",
↪   "/home/hutarsan/Documentos/racetrack/scripts
↪   /32kb_4cap_amontonado_4v_%.csv","128","4"]
exec(miss_hits_porcentaje_amontonado)
script_descriptor.close()


print("ACABADO AMONTONADO %")
```