Universidad Zaragoza

Proyecto Fin de Carrera

Análisis de la efectividad de una memoria cache ACDC en el conjunto de programas de prueba SPEC CPU 2006

Antonio Gallego Padín

Director: Dr. Rubén Gran Tejero

Departamento de Informática e Ingeniería de Sistemas
Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza

Junio 2015

Dedicado a mi familia y amigos.

Agradecimientos

Quiero aprovechar estas líneas para agradecer a todas las personas que me han ayudado a lo largo de estos años en la Universidad de Zaragoza.

En primer lugar agradezco a Rubén Grant, mi director del Proyecto, por todo el trabajo, apoyo y dedicación que ha tenido conmigo a lo largo de estos meses de trabajo. Para mí, lo más importante, es todo lo que me ha enseñado y por eso le estoy muy agradecido.

También quiero agradecer a todos mis amigos y compañeros que he tenido estos años en la Universidad, que me han apoyado en los buenos y malos momentos y que gracias a ellos, hoy puedo estar aquí.

Agradecer a todos los profesores en general de esta Universidad, que de una forma u otra me han ayudado a lograr hacer este Proyecto.

No puedo olvidar mis agradecimientos para mi familia.

Muchas gracias para todos.

Análisis de la efectividad de una memoria ACDC en el conjunto de programas de prueba SPEC CPU 2006

Resumen

Este Proyecto de Fin de Carrera es un Proyecto del Departamento de Informática e Ingeniería de Sistemas de la Universidad de Zaragoza. La idea de este Proyecto surgió a raiz de la creación de una nueva memoria cache ACDC [1] por parte de los profesores del Departamento.

En este proyecto se ha evaluado esta memoria cache ACDC dentro de la jerarquía de memoria de un procesador de altas prestaciones en SPEC CPU 2006 [2]. La memoria ACDC tiene una política de reemplazo basada en instrucciones, a diferencia de las memorias cache convencionales, que se basan en el flujo de instrucciones del programa. Estas instrucciones se obtienen con un profiling o análisis previo.

Esta memoria ACDC tendrá dos organizaciones distintas e independientes dentro de la jerarquía de memoria convencional. Por una parte, estará organizada de forma secuencial. En esta organización la memoria ACDC será la que ocupe el primer nivel de la jerarquía de memoria del procesador. Será a ella a quien el procesador le solicite los datos. La otra organización es de forma paralela con el primer nivel de la jerarquía de memoria. En este caso el procesador solicita los datos a las dos memorias, ACDC y L1, de forma paralela. La latencia de ejecución de esta memoria ACDC es más pequeña que la de la memoria cache L1. Se intenta por tanto, disminuir la latencia de ejecución de las instrucciones.

La finalidad de incluir esta memoria ACDC dentro de la jerarquía convencional de memoria de un procesador de altas prestaciones es intentar que el rendimiento aumente y que el consumo disminuya.

Contenidos

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

El tiempo de ejecución de un programa suele estar limitado por las operaciones de acceso a memoria, tanto de instrucciones como de datos. Tecnológicamente, hay una diferencia muy grande y ***creciente*** entre la capacidad de procesamiento de la CPU y el ancho de banda y latencia de las memorias (figura 1.1). Esto se conoce como *memory wall*. La suma de todo esto nos lleva a severas pérdidas de rendimiento ya que la CPU tiene que esperar a que la memoria le suministre la información para avanzar la ejecución del programa. Una alternativa es introducir una jerarquía de memoria. En la figura 1.1 [4] se puede observar que en los últimos años, el rendimiento de la CPU ha experimentado una mayor velocidad en el procesamiento de las instrucciones en relación a la memoria. Por este motivo, la jerarquía de memoria es tan importante.

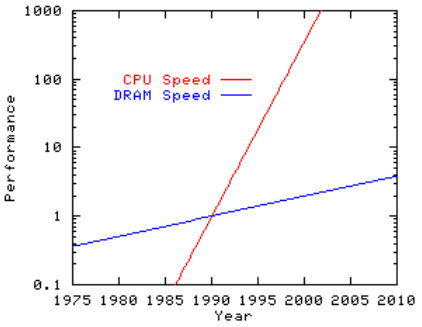


FIGURA 1.1: Diferencia de rendimiento entre la CPU y la memoria [4].

La jerarquía de memoria se organiza en varios niveles de memoria, que son distintos en cuanto a capacidad, latencia y ancho de banda. Cuanto más cerca esté un nivel de memoria del procesador, más pequeña es su capacidad y su latencia. En la figura 1.2 se observa que el nivel más cercano al procesador es el nivel más bajo y se llama nivel 1 (L1). Le siguen el nivel 2 (L2), nivel 3 (L3) y el último nivel de memoria, la memoria principal.

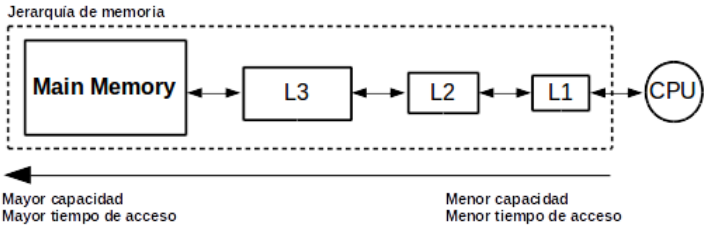


FIGURA 1.2: Jerarquía de memoria con tres niveles y memoria principal.

La intencionalidad de una jerarquía de memoria es que las peticiones por parte del procesador sean servidas por el nivel 1 de la jerarquía de memoria y de esta forma los accesos a memoria experimentan menor latencia que si fuesen servidos desde la memoria principal. Así pues, en presencia de una jerarquía de memoria convencional, el procesador siempre solicita la dirección a la que quiere acceder al nivel 1 de la jerarquía de memoria, con la esperanza que se encuentre allí y así, sea servido con una baja latencia. Si la dirección no se encuentra en dicho nivel, se buscará en el siguiente nivel, a un coste mayor de latencia. De este modo, lo que se busca con esta jerarquía de memoria es que los contenidos más usados estén en los niveles más bajos de la jerarquía para así ocultar el *memory wall*.

Las jerarquías de memoria funcionan bien ya que los programas presentan reuso, tanto a nivel de instrucciones como de datos. Este reuso es generado, por ejemplo, por la presencia de bucles o bien por el acceso reiterado a variables dentro de un programa. Por ejemplo, si hay un bucle de n iteraciones dentro de un programa, se accede a las mismas instrucciones mientras se esté dentro del bucle. Por tanto, existe reuso en las instrucciones del bucle. Por otra parte, por ejemplo, hay reuso temporal, en la variable de control o iterador de dicho bucle. Lo que se busca es que, aquellas instrucciones y variables que presentan más reuso, se encuentren en el nivel más bajo de la jerarquía de memoria, intentando minimizar los accesos a los niveles más altos de la jerarquía de memoria.

En los esquemas convencionales de jerarquía de memoria, el reuso se logra gracias al flujo de referencias de las instrucciones de un programa. Ésta es la forma de gestionar el contenido de los niveles de la jerarquía de memoria y por eso funciona bien.

Otra forma para gestionar el contenido de una jerarquía de memoria es a través de qué instrucción genera la dirección a la variable que presenta reuso. Esta forma de gestión es la que se plantea con la memoria ACDC y es la que utilizaremos en este trabajo.

1.1. Contexto del Proyecto

Este Proyecto de Fin de Carrera es un proyecto del Departamento de Informática e Ingeniería de Sistemas de la EINA de la Universidad de Zaragoza y es una continuación a estudios relacionados con la incorporación de un nuevo nivel de memoria dentro de la jerarquía de memoria.

1.2. Objetivos

El objetivo de este Proyecto Fin de Carrera es evaluar la memoria ACDC en un procesador de propósito general de altas prestaciones, utilizándola como una cache de nivel 0, dentro de la jerarquía de memoria, o bien utilizándola de forma paralela con el nivel L1 de la jerarquía de memoria, en el conjunto de programas de prueba SPEC CPU 2006 [2].

1.3. Estado del arte y trabajos previos

En este Proyecto he extendido el trabajo presentado por Juan Segarra y otros [1] (profesores de la Universidad de Zaragoza). Más concretamente, he evaluado el impacto en energía y rendimiento de la memoria ACDC en un procesador de altas prestaciones, esto es superescalar, con lanzamiento de instrucciones fuera de orden y especulación de latencia de ejecución (ver B.1), para la Benchmark Suite SPEC CPU 2006 [2].

Esta propuesta de memoria busca aprovechar el reuso existente en los algoritmos. La idea principal es que son las instrucciones de acceso a memoria las que generan las referencias a memoria correspondientes a los datos y por lo tanto, se podría identificar aquellas instrucciones de memoria que acceden a los contenidos con más reuso y que fuesen estas instrucciones las que determinasen el contenido de la cache de datos.

El uso de caches de tamaño reducido y muy baja latencia en jerarquías de memoria ha sido muy utilizada [6], [7], [8], [9], [10]. Algunas propuestas buscan reducir el tiempo de acceso y así mejorar el tiempo de ejecución [6], [7]. Otras propuestas, buscan reducir el consumo energético [7], [8], [9], [10]. En cualquier caso, ninguna de las propuestas se basa en controlar el reemplazo de dicha cache de datos a través de las instrucciones que realizan el acceso a cache.

En cuanto a qué información se utiliza para gestionar el contenido de la cache de datos, la gran mayoría de los trabajos utilizan solo el flujo de direcciones generado por el programa para seleccionar qué contenidos permanecen en cache y así aprovechar el reuso. Aquellos trabajos [11] que utilizan la dirección de las instrucciones para gestionar el contenido de cache lo suelen utilizar para predecir cuál va a ser el comportamiento de la cache y no para identificar aquellas instrucciones que tienen reuso como sí lo hace la memoria ACDC [1].

1.4. Organización de la Memoria

Esta Memoria de este Proyecto de Fin de Carrera tiene la siguiente estructura:

- En este capitulo 1 se presenta la problemática asociada al aumento de prestaciones del procesador con respecto a la memoria. También se habla del estado del arte y trabajos previos.
- En el capítulo 2 se describe cómo hemos organizado la memoria ACDC [1] dentro de la jerarquía de memoria de un procesador de altas prestaciones.

- En el capítulo 3 se presentan y analizan los resultados obtenidos con la ejecución de un procesador de altas prestaciones y la carga de trabajo SPEC CPU 2006 [2].
- Finalmente, en el capítulo 4 se muestran las conclusiones de este proyecto y se proponen líneas de trabajo futuro.

Además de estos capítulos, también se incluyen los siguientes apéndices:

- Apéndice A. Carga y desarrollo del Proyecto. En esta parte, se explica la gestión del tiempo del Proyecto, los trabajos realizados, esfuerzo invertido y situaciones problemáticas.
- Apéndice B. Procesadores de altas prestaciones y SPEC CPU2006 [2]. Los procesadores de altas prestaciones son explicados en este apéndice y la problemática de la ejecución especulativa. También se indican las cargas de trabajo utilizadas en el Proyecto [2].
- Apéndice C. Introducción a la memoria ACDC. En este apéndice se describe y explica la memoria ACDC [1].
- Apéndice D. Implementaciones realizadas. Aquí se muestran los cambios hechos en el código del simulador usado en el Proyecto [3] y algunos ficheros de automatización y obtención de datos con el fin de agilizar los trabajos.
- Apéndice E. Resultados, costes energéticos, métricas y análisis previo. En este apéndice se muestran todos los resultados que hemos obtenido a lo largo del proyecto así como las fórmulas necesarias para obtener algunos de ellos. También se describen las métricas usadas y la fase previa de análisis o profiling.

Capítulo 2

Memoria ACDC en procesadores de altas prestaciones

Los procesadores de altas prestaciones deben hacer un uso extensivo de la especulación para conseguir sus objetivos de rendimiento. Un ejemplo de especulación es la predicción de latencia de ejecución de instrucciones de acceso a jerarquía de memoria. En este capitulo voy a presentar mi trabajo de implementación de la memoria ACDC [1] en un procesador para altas prestaciones y más concretamente como afecta esta especulación, evaluándolo sobre SPEC CPU 2006 [2]. Para ello, en la primera parte del capítulo haremos alusión a los procesadores de altas prestaciones y el problema de la ejecución especulativa. Toda la información sobre esto se encuentra en el apéndice B de este documento. Posteriormente presentaremos la idea de la memoria ACDC [1], que se puede consultar en el apéndice C. Finalmente, explicaré cómo he adaptado esta memoria ACDC dentro de la jerarquía de memoria convencional de un procesador de altas prestaciones.

2.1. Los procesadores de altas prestaciones

Los procesadores de altas prestaciones se caracterizan por ser altamente segmentados, superescalares y con ejecución fuera de orden de instrucciones. Además lanzan instrucciones de forma especulativa ya que hay instrucciones cuya latencia de ejecución no es conocida en tiempo de lanzamiento a ejecución. Las instrucciones de acceso a memoria son un ejemplo, ya que tienen una latencia de ejecución que depende del nivel de la jerarquía de memoria en la que se encuentre el dato al que se quiere acceder y este nivel solo es conocido durante el propio acceso a memoria. Todas estas características hacen que los procesadores de altas prestaciones puedan aumentar su rendimiento ya que se mejora su capacidad para explotar el paralelismo a nivel de instrucciones (ILP).

En la figura 2.1 se muestra el segmentando del procesador de altas prestaciones utilizado en este proyecto.

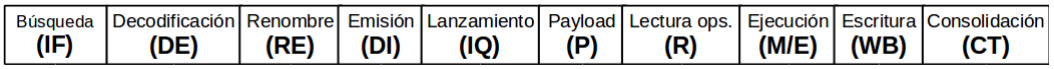


FIGURA 2.1: Segmentado del procesador utilizado en este proyecto.

Para saber más sobre la ejecución especulativa de instrucciones y las implicaciones que tiene sobre los procesadores de altas prestaciones se puede consultar en el apéndice B de este documento.

2.2. La memoria cache ACDC

La memoria cache ACDC [1] es una memoria que ha sido desarrollada por los profesores del Departamento de Informática e Ingeniería de Sistemas de la Universidad de Zaragoza. Se caracteriza por que el contenido de la memoria es controlado por las instrucciones y no por el flujo de direcciones de accesos generados por el programa. Una posible estructura hardware que puede aprovechar el reuso de estas instrucciones es una memoria muy pequeña y completamente asociativa (DC), con una política de reemplazo basada en instrucciones que se encuentran en otra memoria muy pequeña (AC). Por ejemplo, en acierto (hit) en lecturas, se proporcionará el dato. Para saber si hay hit en DC se comprueba si la *addr* del dato se encuentra en alguna línea de la *tag_data*; en acierto en escritura, se actualizará el dato. En fallos, se comprobará si la instrucción que accede al dato (PC) se encuentra en la AC. Si el PC está en la memoria AC, tiene permiso para reemplazar (DRP) en la DC el dato asociado al índice de la instrucción (DC_idx) de la AC. Si no tiene DRP, no se reemplaza. Esta estructura se puede ver en la figura 2.2. Su descripción y funcionamiento se pueden ver en el apéndice C de este documento.

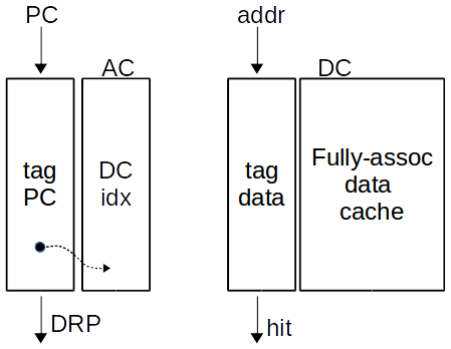


FIGURA 2.2: Estructura hardware que implementa la ACDC.

2.3. Organización de la memoria ACDC dentro de la jerarquía de memoria

En mi trabajo, la memoria ACDC podrá estar colocada en un nivel inferior a la L1, secuencialmente y después del procesador (a un nivel 0, ver figura 2.3) o podrá estar situada al mismo nivel que la L1, de forma paralela (ver figura 2.5). Estos dos tipos de organización, secuencial y paralelo, para las instrucciones de acceso a memoria, se detallan a continuación.

2.3.1. Organización secuencial.

En esta organización de memoria, ver figura 2.3, la memoria ACDC está situada entre el procesador y el primer nivel de memoria de la jerarquía convencional de memoria.

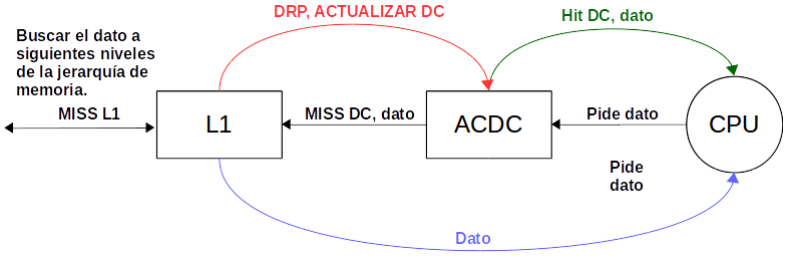


FIGURA 2.3: Organización secuencial de la memoria ACDC con la jerarquía convencional de memoria.

Cuando la CPU solicita un dato lo hace a la ACDC. Si el dato se encuentra en la DC, tendremos un acierto o hit y el dato es suministrado por la memoria DC al procesador. Si el dato no está en la DC, se tendrá un fallo o miss en DC, lo que significa que el dato hay que ir a buscarlo al siguiente nivel de la jerarquía de memoria, el nivel 1. Esto trae consigo que haya una penalización en latencia en comparación a la organización sin el uso de la memoria ACDC, ya que hemos accedido secuencialmente a la DC para buscar el dato y hemos fallado. Así mismo, al causar fallo en DC, preguntamos a la AC si tenemos DRP. Si el dato se encuentra en la L1, será ella, la L1, quien suministre el dato al procesador y si tenemos DRP, la DC se actualizará con el correspondiente dato. Si el dato no está en L1, hay que ir a buscarlo a los siguiente niveles de la jerarquía de memoria con la penalización en latencia que ello supone.

Las situaciones que se pueden dar con este tipo de organización se pueden ver en los cronogramas de la figura 2.4 y se explican a continuación.

El cronograma 2.4(a), representa el funcionamiento de una instrucción de acceso a la jerarquía de memoria sin el uso de la memoria ACDC y hit en L1. En este caso se predijo que la latencia de ejecución para la instrucción de acceso a memoria es la latencia de la memoria L1 y al despertar especulativamente las instrucciones dependientes en el ciclo 5, se ha acertado y hemos obtenido el encadenamiento esperado.

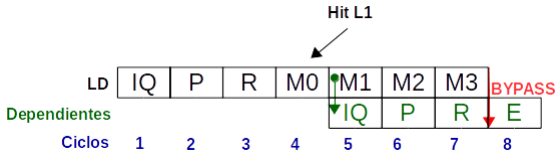
El cronograma 2.4(b), representa el mismo caso que el anterior, pero hay fallo en L1 y el dato hay que ir a buscarlo a los siguientes niveles de la jerarquía de memoria. Aquí, se ha predicho que la latencia de ejecución para el load era la latencia de la L1 y ha habido miss en L1. Por lo tanto, las instrucciones dependientes del LD se anulan en el ciclo 5 y el LD se duerme a la espera del dato. Una vez que el dato ya se encuentre en la L1, el LD se vuelve a lanzar y sus instrucciones dependientes se lanzan en el ciclo 13, pero ya sin especular ya que se predice con latencia de L1 ya que el dato está en L1 y así conseguir que haya encadenamiento.

El cronograma 2.4(c) representa el caso en el que se accede a la DC y se tiene un hit. Las instrucciones que se despertaron especulativamente en la etapa IQ del LD (ciclo 2) se ejecutan haciendo el encadenamiento de instrucciones sin pérdida de rendimiento ya que la ejecución especulativa ha tenido éxito puesto que se predijo que la latencia de ejecución del load sería la de la memoria DC. En relación al cronograma 2.4(a) hemos ganado 3 ciclos en latencia.

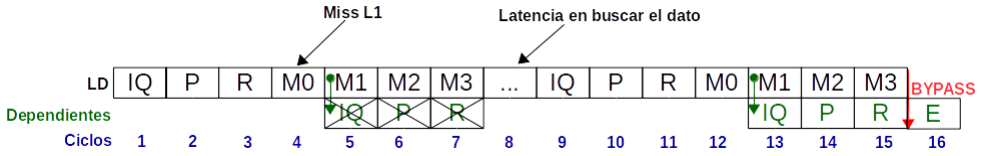
En el cronograma de la figura 2.4(d), se puede ver que tenemos miss en DC y hit en L1. En este caso, las instrucciones dependientes que se despertaron especulativamente en la etapa IQ del LD, ya que hemos predicho que la latencia de ejecución para el load, era la latencia de la memoria DC, se anulan (ciclo 4) y se lanzan dos ciclos más tarde (ciclo 6), ya que tenemos hit en L1 y así conseguir el encadenamiento de las instrucciones. Con respecto al cronograma 2.4(a) hemos perdido 1 ciclo en latencia.

Los cronogramas 2.4(e) y 2.4(f) muestran miss en DC y miss en L1. En estas situaciones, hay que ir a buscar el dato a los siguientes niveles de la jerarquía de memoria. Se anulan las instrucciones dependientes despertadas especulativamente en la etapa IQ del LD (ciclo 4), puesto que hemos predicho al load con latencia de ejecución de la memoria DC y el load se duerme a la espera del dato. Una vez que el dato está disponible en la L1, se relanza el LD y se pueden dar dos situaciones. Si tenemos DRP, como se ve en el cronograma 2.4(e), las instrucciones dependientes se lanzan en la etapa IQ del LD (ciclo 11) para conseguir el encadenamiento de las instrucciones. En este caso y con respecto al cronograma 2.4(b), al tener DRP, hemos conseguido una mejora de 2 ciclos en latencia. Si no tenemos DRP, se anulan las instrucciones dependientes lanzadas en la IQ del LD (ciclo 13) para relanzarlas dos ciclos más tarde (ciclo 15), ya que se ha predicho lanzamiento del load con latencia de la memoria DC (miss en DC) y el dato está en L1. De esta forma se logra el encadenamiento de instrucciones, tal y como se muestra en el cronograma 2.4(f). En este caso con respecto al cronograma de la figura 2.4(b), tenemos una penalización de 2 ciclos en latencia.

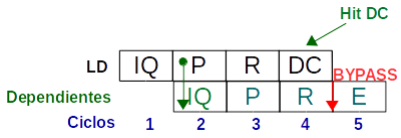
La implementación de esta idea de organización secuencial dentro del simulador que hemos usado [3], se pueden ver en el apéndice D de esta Memoria.



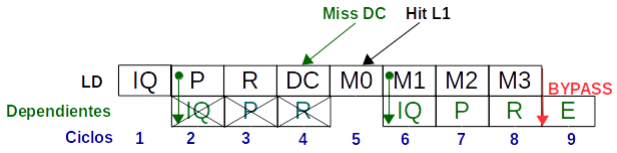
((A)) Sin memoria ACDC. Acierto en L1.



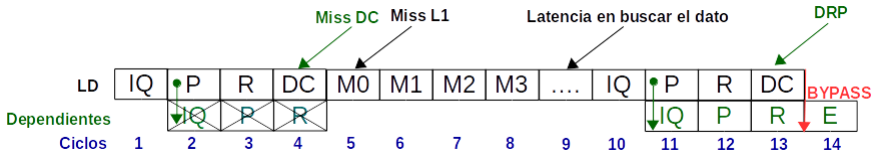
((B)) Sin memoria ACDC. Fallo en L1.



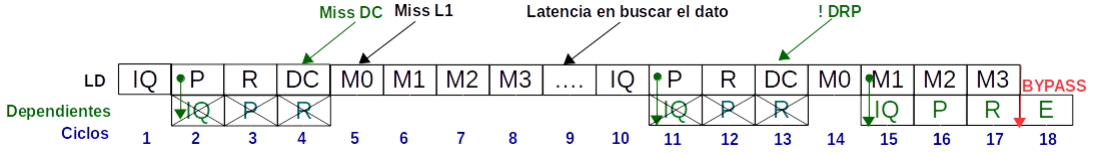
((C)) Acierto en DC.



((D)) Fallo en DC, acierto en L1.



((E)) Fallo en DC, fallo en L1 y con DRP.



((F)) Fallo e DC, fallo el L1, y sin DRP.

FIGURA 2.4: Cronogramas de la organización secuencial de la memoria ACDC con la jerarquía convencional de memoria.

2.3.1.1. Conclusión

El uso de la estrategia de ejecución especulativa de instrucciones de acceso a memoria con una organización secuencial de la memoria ACDC con la jerarquía convencional de memoria puede conseguir un rendimiento superior del procesador si el número de aciertos es DC es grande ya que así disminuimos el número de accesos a L1, disminuyendo la latencia de ejecución en los accesos a memoria. Además, el consumo de energía se reduce ya que el gasto de acceder a la L1 es mayor que el de acceder a la DC. Si por el contrario, el número de aciertos en DC es pequeño, la penalización en latencia es importante lo que conllevará a una disminución del rendimiento.

2.3.2. Organización paralela.

En la figura 2.5 se muestra la organización paralela de la memoria ACDC con la jerarquía convencional de memoria.

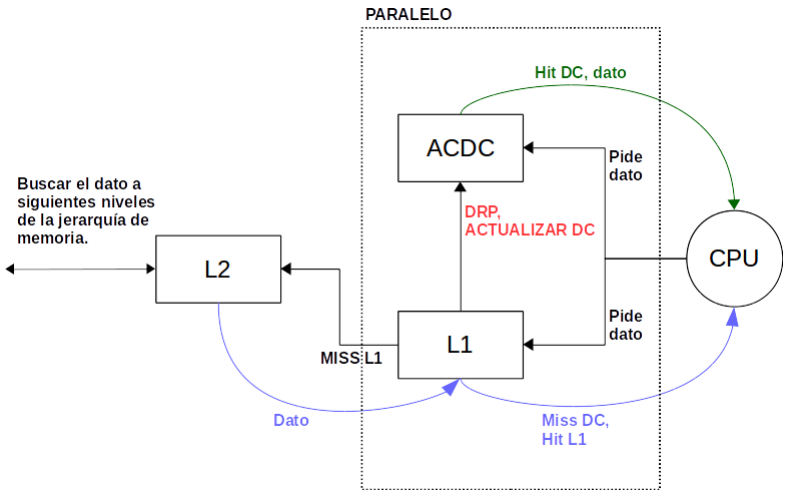


FIGURA 2.5: Funcionamiento paralelo de la memoria ACDC con la jerarquía convencional de memoria.

Cuando la CPU solicita un dato, lo hace de forma paralela tanto a la ACDC como a la L1. Si el dato se encuentra en la DC, tenemos hit en DC y es ésta quien suministra el dato al procesador. De esta forma oculta parte de la latencia de acceso a memoria al ser la DC quien ha proporcionado el dato. La latencia de DC es de 1 ciclo y la de L1 de 4 ciclos (ver tabla 3.1). Si el dato no se encuentra en DC, tenemos un miss en DC y entonces si el dato está en L1, es la L1 quien suministra el dato al procesador. En este caso, miss en DC, se accede a AC para saber si tenemos DRP. En caso afirmativo, el dato se actualiza en la DC. Si el dato no está en la L1, hay que ir a buscarlo a los siguientes niveles de la jerarquía de memoria.

Las situaciones que se pueden dar con este tipo de organización se pueden ver en los cronogramas de la figura 2.6 y se explican a continuación.

El cronograma 2.6(a) representa la ejecución de un LD sin memoria ACDC dentro de la jerarquía de memoria con un hit en L1. Las instrucciones dependientes se lanzan especulativamente en la IQ del LD (ciclo 5) ya que se predice que la latencia de ejecución del load es la latencia de L1 y así conseguir el encadenamiento de instrucciones. Este cronograma es igual que el de la figura 2.4(a) del apartado anterior (2.3.1).

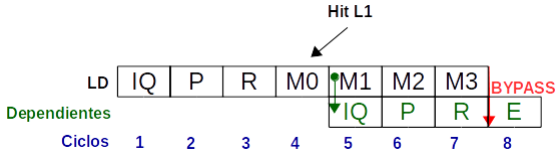
El cronograma de la figura 2.6(b) nos muestra el caso en que hay miss en L1. Se anulan las instrucciones dependientes (ciclo 5) ya que hemos predicho que la latencia de ejecución del load es la latencia de L1 y el LD se duerme a la espera del dato. Cuando el dato esté disponible en L1 se relanza el LD y sus instrucciones dependientes (ciclo 13) y así conseguir encadenamiento de instrucciones. Este cronograma es igual que el de la figura 2.4(b) del apartado anterior (2.3.1).

En el cronograma 2.6(c) se muestra que si un LD en la etapa de memoria, tiene hit en DC, las instrucciones dependientes de ese LD que se despertaron especulativamente en la IQ del LD (ciclo 2), consiguen hacer el encadenamiento de instrucciones, ya que la latencia de ejecución se predijo con la latencia de la memoria DC, como se muestra en la figura 2.6(c). En este caso la ejecución especulativa de instrucciones ha tenido éxito y hemos conseguido aumentar el rendimiento debido a que la latencia de DC es menor que la de L1 y oculta la de la L1. Con respecto al cronograma de la figura 2.6(a) hemos ganado 3 ciclos en latencia.

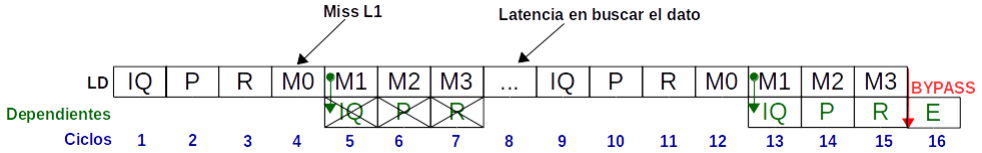
El cronograma 2.6(d) muestra miss en DC y hit en L1. En este caso, anulamos las instrucciones dependientes (ciclo 4) ya que la latencia de ejecución del load se predijo con la latencia de DC, para lanzarlas al ciclo siguiente (ciclo 5) y así conseguir el encadenamiento de instrucciones. En este caso, la ejecución especulativa de instrucciones dependientes ha fallado, pero no hemos tenido penalización en latencia con respecto al cronograma de la figura 2.6(a). Sin embargo hay más relanzamientos.

Finalmente, en los cronogramas 2.6(e) y 2.6(f) se muestran el caso en que hay miss en DC y miss en L1. En este caso, el dato hay que ir a buscarlo a los siguientes niveles de la jerarquía de memoria. Las instrucciones dependientes se anulan (ciclo 4) ya que la latencia de ejecución del load se predijo con la latencia de la memoria DC y el load se duerme a la espera del dato. Una vez que el dato está disponible en la L1, se relanza el LD y se pueden dar dos situaciones. Si tenemos DRP (el dato está en DC), las instrucciones dependientes se despiertan en la IQ del LD (ciclo 10) para así conseguir encadenamiento de instrucciones. Se puede ver esto en el cronograma 2.6(e). Con respecto al cronograma de la figura 2.6(b) hemos obtendido una ganancia en latencia de 3 ciclos. Pero si no tenemos DRP, las instrucciones dependientes se anulan (ciclo 12) y se lanzan al ciclo siguiente (ciclo 13) ya que la latencia de ejecución del load se predijo con la latencia de la memoria DC y así tener encadenamiento de instrucciones, como se ve en el cronograma 2.6(f). De esta manera no hemos obtenido ganancia en latencia en relación al cronograma de la figura 2.6(b) y además ha habido más relanzamientos.

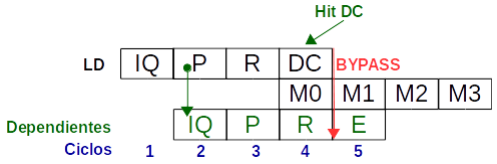
La implementación de esta idea organización paralela dentro del simulador que hemos usado [3], se pueden ver en el apéndice D de esta Memoria.



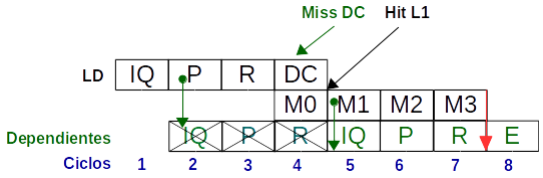
((A)) Sin memoria ACDC. Acierto en L1.



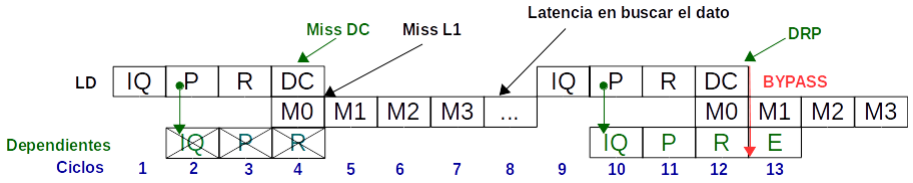
((B)) Sin memoria ACDC. Fallo en L1.



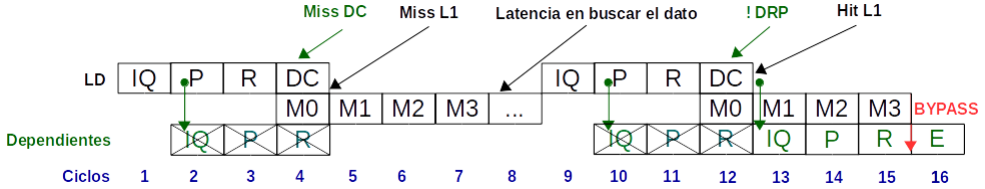
((C)) Acierto en DC.



((D)) Fallo en DC, acierto en L1.



((E)) Fallo en DC, fallo en L1 y con DRP.



((F)) Fallo e DC, fallo el L1, y sin DRP.

FIGURA 2.6: Cronogramas de la organización paralelo de la memoria ACDC con la jerarquía convencional de memoria.

2.3.2.1. Conclusión

La organización paralela de la ACDC con la jerarquía convencional de memoria es una buena opción ya que el rendimiento no puede ser peor con respecto a la jerarquía convencional de memoria, ya que trabajan ambas memorias a la vez (L1 y DC). Si hay hit en DC, la latencia de ejecución de la instrucción de acceso a memoria es menor que la latencia de L1 (ver tabla 3.1) y así el rendimiento aumenta al ocultar la latencia del la L1. En caso contrario, no hay penalización. Se puede decir que es el caso ideal. El inconveniente que tiene esta organización es el aumento del gasto energético ya que estamos accediendo a dos memorias a la vez y el número de instrucciones relanzadas.

2.3.3. Instrucciones de escritura en memoria

Las instrucciones de escritura en memoria son los stores (ST) y estos escriben en memoria en la etapa *commit* (**CT**) del procesador (ver sección B.1) , cuando las instrucciones se retiran dentro del pipeline del procesador. Tanto si estamos en la organización paralela como en la organización secuencial, el funcionamiento es el mismo para ambos casos. Primero preguntamos si el dato está en DC. Si hay hit en DC, lo escribimos. Si hay miss en DC, preguntamos en AC si tenemos DRP. En caso afirmativo, escribimos el dato en DC. En caso contrario, no hacemos nada. Esto se puede ver en la figura 2.7.

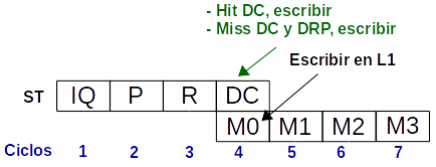


FIGURA 2.7: Escritura de datos en memoria, tanto en DC como en L1.

Al mismo tiempo, el dato se escribe en la memoria L1. El contenido de la DC está en la L1 ya que es en la etapa de commit cuando se escribe tanto en la L1 como en DC, si ésta tiene DRP. Por tanto la DC es inclusiva con la L1.

Capítulo 3

Resumen de resultados

3.1. Introducción

En este capítulo se muestran y se analizan los resultados obtenidos con la ejecución de un procesador de altas prestaciones como el descrito en la sección B.1 y con nuestra propuesta, en la carga de trabajo SPEC CPU 2006 [2] (ver sección B.2). Este capítulo está organizado de la siguiente manera. En la primera parte del capítulo se exponen las configuraciones de las memorias que he utilizado en los experimentos para medir el comportamiento de nuestra propuesta. También se hace mención a la nomenclatura usada en el capitulos respecto a las organizaciones de la memoria dentro de la jerarquía de memoria. Posteriormente, se mostrarán los resultados obtenidos con la ejecución de la suite enteros, dividiendo las explicaciones en rendimiento, gasto energético y tasa de aciertos en la memoria DC. Finalmente se hará una comparativa de datos.

3.2. Configuración de la jerarquía de memoria

La tabla 3.1 nos muestra la configuración de los módulos de memoria que hemos usado en el procesador de altas prestaciones (ver sección B.1) para hacer las pruebas, incluida la memoria ACDC.

Memoria L1	Tamaño	64KB
	Asociatividad	4
	Tamaño bloque	32B
	Latencia	4 ciclos
Memoria L2	Tamaño	256KB
	Asociatividad	8 instrucciones
	Tamaño bloque	128B
	Latencia	14 ciclos
Memoria L3	Tamaño	4MB
	Asociatividad	4
	Tamaño bloque	128B
	Latencia	19 ciclos
Memoria Principal	Latencia	200 ciclos
Memoria ACDC	Tamaño	256B
	Asociatividad	Completamente asociativa
	Tamaño bloque	16B
	Latencia	1 ciclo

Tabla 3.1: Tabla con las configuraciones de la jerarquía de memoria del procesador a utilizar en la ejecución de las simulaciones.

Como se ha dicho en la sección 2.3, hemos incluido la memoria ACDC dentro de la jerarquía de memoria del procesador de altas prestaciones de dos formas distintas: organización secuencial (ver sección 2.3.1) y organización paralela (ver sección 2.3.2). Para poder hacer las pruebas de simulación hemos empleado SimpleScalar [3], ya que nos permite simular el procesador de altas prestaciones que hemos presentado anteriormente (ver sección B.1).

La implementación de estas organizaciones de la memoria ACDC las he realizado sobre el SimpleScalar [3] y pueden verse en la sección D.1. Estas implementaciones han consistido en ampliar las funciones relativas a los accesos de memoria y lanzamiento de instrucciones así como crear otras funciones nuevas.

En este capítulo hemos evaluado tres modelos: A la ejecución del procesador sin el uso de la memoria ACDC, que llamamos *"modo solo"*. A la ejecución del procesador con la organización secuencial, que llamamos *"modo secuencial"* y a la ejecución del procesador con la organización paralela que llamamos *"modo paralelo"*.

3.3. Resultados

En este apartado se muestran y analizan los resultados obtenidos con la ejecución del procesador descrito en la sección B.1 y la carga de trabajo SPEC CPU 2006 [2] (sección B.2). Al analizar los resultados hemos visto que tanto para la suite enteros como para la suite float, los resultados siguen una misma línea o patrón. Para explicar los resultados hemos elegido la suite entera.

3.3.1. Rendimiento

La gráfica de la figura 3.1 muestra los valores obtenidos de IPC. Se puede observar que tanto para el *modo paralelo* como para el *modo secuencial*, los valores obtenidos son mayores que en el *modo solo*. Esto se debe al número de aciertos en la memoria DC (ver columnas "H" y "J" de la tabla 3.3) ya que al ser su latencia de ejecución menor que la de la memoria L1, se mejora el rendimiento.

Además se observa que, en el *modo paralelo*, el rendimiento es mayor que en el *modo secuencial*. Esto es debido a que en el *modo paralelo* no hay penalización en ciclos por fallar en la memoria DC (ver cronogramas de la figura 2.6) y para el *modo secuencial*, sí que hay penalización por fallar en DC (ver cronogramas de la figura 2.4).

El valor medio de ganancia en el *modo paralelo* con respecto al *modo solo*, es del 9.55 %. Para el *modo secuencial*, la ganancia experimentada es del 6.55 %, con respecto al *modo solo*.

Sabiendo que el ancho de instrucciones que se pueden lanzar es de 8 instrucciones y observando la gráfica 3.1, podemos pensar que cuanto más bajo sea el valor de IPC de un benchmark, más ganancia en rendimiento se podrá obtener. El benchmark "Bzip2" tiene un valor de IPC en el *modo solo* de 0.81, y es igual al *modo secuencial* y ligeramente inferior al *modo paralelo* (0.82).

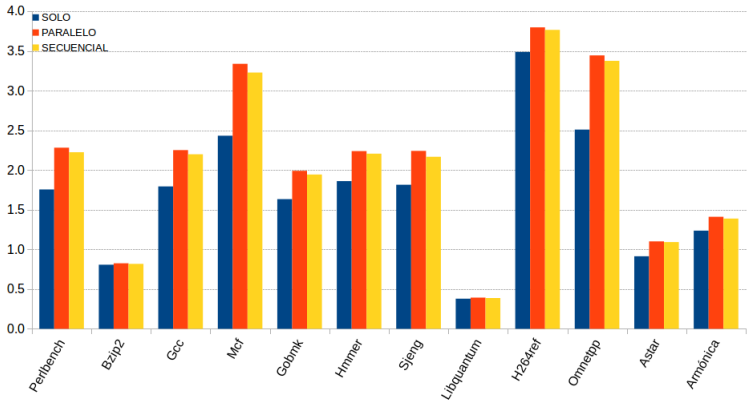


FIGURA 3.1: Gráfica que representa los valores de IPC obtenidos con la ejecución del procesador con las distintas organizaciones de memoria, *modo solo*, *modo paralelo* y *modo secuencial* para la suite enteros.

Este valor de IPC está muy por debajo del ancho de banda de lanzamiento de instrucciones, que es 8. Esto indica que "Bzip2" tiene muy poco ILP. Vemos también que su tasa de aciertos en la memoria DC es del 69.70 %, tanto para el *modo paralelo y modo secuencial* (ver la tabla 3.3). A pesar de que nuestra propuesta acorta la latencia de los loads, no se obtiene mejora en el rendimiento. Esto me lleva a pensar que la latencia del camino crítico de "Bzip2" no depende de estas instrucciones load.

Estas mismas conclusiones son válidas para el benchmark "Libquantum", que le pasa lo mismo que al "Bzip2". Su valor de IPC es prácticamente el mismo en los tres modos de funcionamiento, 0.38, 0.39 y 0.39 para el *modo solo, modo paralelo y modo secuencial* respectivamente y muy por debajo del ancho de lanzamiento de instrucciones y con una tasa de aciertos en la DC es del 47.17 % (ver tabla 3.3).

3.3.2. Gasto energético de la jerarquía de memoria

Para obtener los costes de energía de las memorias usadas en este Proyecto hemos usado la herramienta Cacti en su versión 6.5 [12]. Para ver estos costes, consultar la sección E.1 de este documento.

La figura 3.2 muestra el consumo total de las simulaciones. Se puede ver que el mayor consumo de energía para cada benchmark corresponde con la componente dinámica (ver sección E.1) y supone de media, las tres cuartas partes del consumo total.

Este consumo de la componente dinámica es mayor en el *modo paralelo* que en los modos *solo y secuencial*. Esto se debe a que en el *modo paralelo* estamos accediendo a dos módulos de memoria a la vez cuando el procesador solicita un dato (al módulo de memoria DC y al módulo de memoria L1).

También podemos ver que el *modo secuencial* es el que menos consumo de energía tiene. Esto se debe a que en este modo, cuando el procesador solicita un dato, se accede solamente a la memoria DC. Unicamente cuando fallamos en DC, accedemos a la L1, aumentando el consumo

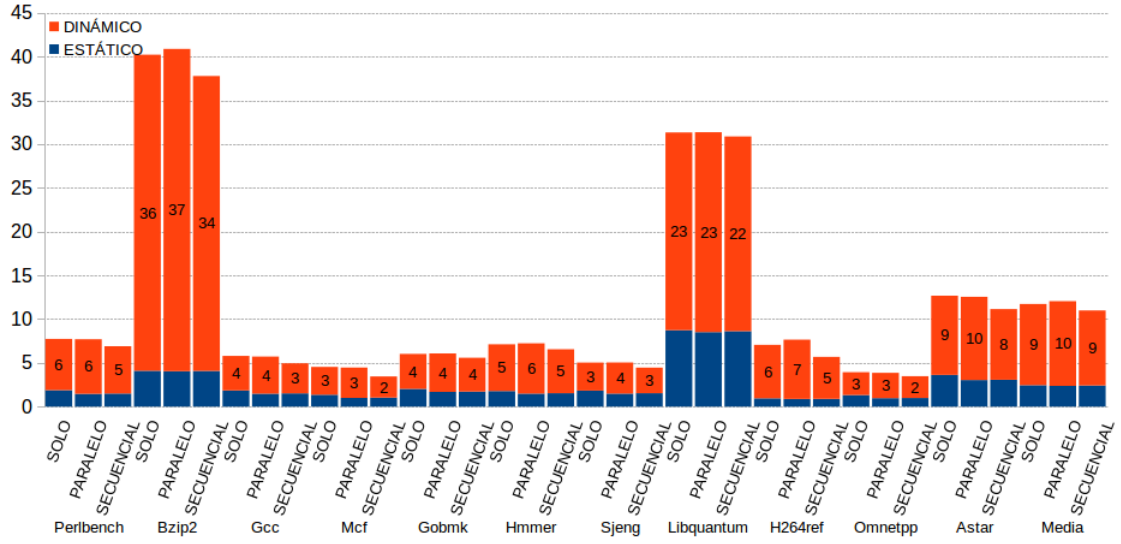


FIGURA 3.2: Gráfica que muestra el consumo de total de energía en las simulaciones expresado en mJ, para los tres modos de funcionamiento en la suite enteros.

de energía. La tabla 3.2 nos muestra la métrica Energy_Delay [5], que relaciona el tiempo de ejecución del programa y su consumo energético. Según esta tabla 3.2, el *modo secuencial* es el más eficiente de los tres modos de funcionamiento.

También se observa en la figura 3.2 que el *modo solo* tiene más consumo estático a pesar de que tiene menos hardware. Esto es por que su rendimiento es peor que en los otros modos (*paralelo y secuencial*). En la tabla 3.2 se puede ver que su valor medio de Energy_Delay es el mayor de los tres modos de funcionamiento, atendiendo a esta métrica.

Nombre	Perlbench	Bzip2	Gcc	Mcf	Gobmk	Hmmer	Sjeng	Libquantum	H264ref	Omnetpp	Astar	Media
SOLO	4.41	49.84	3.23	1.87	3.69	3.83	2.77	82.68	2.02	1.57	13.88	15.44
PARA	3.38	49.54	2.54	1.34	3.05	3.23	2.25	79.88	2.02	1.12	11.40	14.52
SECU	3.11	46.21	2.25	1.07	2.87	2.96	2.06	79.69	1.51	1.03	10.21	13.91

TABLA 3.2: Tabla que muestra la métrica Energy_Delay [5] de las simulaciones para los tres modos de funcionamiento en la suite enteros.

3.3.3. Tasa aciertos en DC

Cuanto mayor sea el número de aciertos en la memoria DC, el rendimiento se espera que sea mejor, ya que la latencia de ejecución de la memoria DC es menor que la latencia de ejecución de la memoria L1 (ver tabla B.1).

En la columna "F" de la tabla 3.3 se muestra la tasa de aciertos esperados en la DC. Esta tasa de aciertos esperados en la DC se obtiene del cociente entre los aciertos que se han obtenido en la DC en la fase de análisis previo (ver E.2) y el número de instrucciones lanzadas. El valor medio

es del 54.35 % y supone una cota superior de aciertos en la DC. La tasa de aciertos obtenidos para el *modo paralelo* es del 40.02 %, valor inferior. Esta tasa de aciertos obtenidos en la DC es el resultado del cociente entre los aciertos obtenidos en la DC y los accesos totales a la DC. El motivo de esta diferencia puede ser por lo siguiente. Hemos dicho anteriormente que en la AC guardamos aquellas instrucciones que aprovechan más el reuso de los datos del programa (ver sección E.2 y sección [1]). Si en una determinada parte del programa se accede a un dato y ese dato debería estar en la memoria DC y no está por que la instrucción que tiene permiso para reemplazar en la DC todavía no se ha lanzado. Entonces el dato todavía no está en DC y tenemos fallo.

Por otra parte , cuando tenemos DRP, el dato se escribe en la memoria DC. Si este dato no está disponible en este ciclo por que no se encuentra en L1, hay que esperar a que el dato esté disponible. Durante esta espera, puede haber accesos a ese dato. Estos accesos causan fallo en la memoria DC.

Estos motivos también son válidos para el *modo secuencial*, cuya tasa de aciertos en la DC es del 41.13 % (ver 3.3.4), valor ligeramente superior al *modo paralelo* debido a que hay más relanzamientos que en el *modo paralelo*.

Otro motivo por el cual la tasa de aciertos esperados en DC es baja (media del 54.35 %, ver tabla 3.3), es debido a que en la fase de análisis previo, se han tenido en cuenta aquellos PC's que han obtenido acierto en el dato y los aciertos de otras instrucciones que también acceden al mismo dato. Es decir, se tomaron PC's que tienen reuso de grupo que ya tienen aciertos [1].

3.3.4. Comparativa ficheros análisis, 1 vs 100

La fase previa de análisis la hacemos para estudiar qué instrucciones aprovechan más el reuso de los datos del programa. Hasta ahora, este análisis lo hemos hecho de un solo tramo, es decir, con el total de las instrucciones del programa. Posteriormente, hemos hecho una división de este análisis en 100 tramos de un millón de instrucciones. La finalidad de esta división en tramos más pequeños es que el análisis del reuso sea más preciso.

La tabla 3.3 muestra una comparativa de rendimiento, consumo y aciertos en la DC, con la ejecución de la simulación con 1 fichero de análisis y con 100 ficheros de análisis.

Las columnas "F" y "G" de la tabla 3.3 nos muestran la tasa de aciertos esperados en la memoria DC y se ha calculado como el cociente entre los aciertos que se han obtenido en la DC en la fase de análisis previo (1 tramos y 100 tramos) y el número de instrucciones lanzadas. Las columnas "H", "I", "J" y "K" muestran las tasa de aciertos obtenidos en la memoria DC, tanto en paralelo como en secuencial, para 1 fichero de análisis y 100 ficheros. Este valor se ha calculado como el cociente de los aciertos que se han obtenido en la DC y los accesos totales a la DC.

Podemos ver en las columnas relativas al rendimiento que no hay diferencia entre 1 fichero y 100 ficheros de análisis, a pesar de que la tasa de aciertos esperados en DC con 100 tramos es mayor que con la de 1 tramo y que la tasa de aciertos obtenidos en DC es mayor. Esto es debido a que estos aciertos en DC son sobre caminos distintos al camino crítico y aunque en estos se

NOMBRE	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
Perlbench	1.76	2.28	2.28	2.22	2.22	52.40%	67.98%	41.15%	45.74%	42.84%	45.50%	3.33	2.74	2.67	2.43	2.33
Bzip2	0.81	0.83	0.83	0.82	0.82	85.16%	83.25%	69.70%	69.70%	69.70%	69.70%	44.76	44.63	27.31	41.21	23.77
Gcc	1.79	2.25	2.25	2.20	2.20	46.48%	67.13%	41.99%	44.46%	43.87%	45.89%	2.20	1.89	2.35	1.56	2.03
Mcf	2.43	3.34	3.34	3.23	3.22	61.99%	94.94%	54.81%	60.57%	57.51%	59.85%	1.31	1.04	1.29	0.75	0.98
Gobmk	1.63	1.99	1.99	1.94	1.94	32.99%	59.56%	32.49%	34.32%	32.96%	35.00%	2.45	2.21	2.88	1.99	2.64
Hmmer	1.86	2.24	2.24	2.21	2.21	39.84%	81.90%	30.23%	33.19%	30.57%	33.60%	2.88	2.56	2.62	2.27	2.28
Sjeng	1.81	2.24	2.24	2.17	2.17	74.21%	96.37%	28.24%	35.67%	31.43%	37.30%	1.77	1.59	2.16	1.35	1.89
Libquantum	0.38	0.39	0.40	0.39	0.39	33.03%	35.72%	47.17%	47.17%	47.17%	47.17%	59.62	58.19	48.61	57.43	47.37
H264ref	3.49	3.80	3.80	3.76	3.76	52.04%	89.61%	25.34%	65.40%	25.38%	65.60%	1.75	1.78	1.92	1.27	1.25
Omnetpp	2.51	3.44	3.44	3.37	3.37	27.69%	81.86%	33.24%	34.59%	34.44%	35.72%	1.04	0.84	1.05	0.74	0.95
Astar	0.91	1.10	1.10	1.09	1.09	92.07%	264.45%	35.86%	47.73%	36.51%	49.18%	9.91	8.64	7.47	7.40	5.83
Media	1.24	1.41	1.41	1.39	1.40	54.35%	92.98%	40.02%	47.14%	41.13%	47.68%	11.91	11.46	9.12	10.76	8.30

A: IPC en modo solo
B: IPC en modo paralelo con 1 tramo
C: IPC en modo paralelo con 100 tramos
D: IPC en modo secuencial con 1 tramo
E: IPC en modo secuencial con 100 tramos
F: Tasa de aciertos esperados en DC con 1 tramo
G: Tasa de aciertos esperados en DC con 100 tramos
H: Tasa de aciertos obtenidos en DC en paralelo con 1 tramos
I: Tasa de aciertos obtenidos en DC en paralelo con 100 tramos
J: Tasa de aciertos obtenidos en DC en secuencial con 1 tramo
K: Tasa de aciertos obtenidos en DC en secuencial con 100 tramos
L: Indicador energy_delay solo
M: Indicador energy_delay en paralelo con 1 tramo
N: Indicador energy_delay en paralelo con 100 tramos
O: Indicador energy_delay en secuencial con 1 tramo
P: Indicador energy_delay en secuencial con 100 tramos

TABLA 3.3: Tabla que muestra la comparativa de los datos de rendimiento, tasas de aciertos en DC y consumo de 1 tramos vs 100 tramos para la suite enteros.

disminuye la latencia, en el camino crítico no. El paralelismo a nivel de instrucción del programa ya está altamente explotado y resulta dificil conseguir mejoría en rendimiento.

Sin embargo, esta superior tasa de aciertos en la DC con 100 tramos (columnas "I" y "K" de la tabla 3.3), supone un consumo de energía menor, con respecto al *modo solo*. Esta reducción experimenta una mejoria del 2.34 % en el *modo paralelo* con respecto al análisis con 1 tramo y de un 2.46 % en el *modo secuencial*, también con respecto al análisis con 1 tramo. Esto es debido a que los tiempos medios de ejecución de los programas han disminuido con el incremento de aciertos en la DC. En el *modo paralelo* se disminuye en un 0.36 % y un 0.70 % en el *modo secuencial* con respecto a los mismos modos con un tramo de análisis (ver tabla E.15).

Todos los resultados, tanto para la suite enteros como para la suite float, se pueden ver en la sección E.4 de este documento.

3.3.5. Otras pruebas

En estas sección hago referencia a otras pruebas que he hecho. Estas pruebas han consistido en la ejecución de las simulaciones en el *modo solo, paralelo y secuencial* con distintas entradas. He usado una versión de las suites enteras y float que han sido modificadas con unos datos distintos. Los resultados se pueden ver en la tabla E.17. Estos resultados son prácticamente iguales a los que hemos obtenido en las primeras simulaciones, lo que demuestra que la memoria ACDC es capaz de aprovechar el reuso del programa.

Capítulo 4

Conclusiones y trabajos futuros.

4.1. Conclusiones

El uso de una nueva memoria como la ACDC, dentro de la jerarquía de memoria en un procesador de altas prestaciones, según las pruebas hechas en este Proyecto, trae consigo una mejoría, tanto en rendimiento como en consumo energético.

	PARALELO			SECUENCIAL		
	IPC	ENERGIA	DC	IPC	ENERGIA	DC
ENTEROS	9.55%	10.63%	40.02%	6.57%	23.28%	41.13%
FLOAT	5.13%	22.57%	47.50%	5.42%	24.09%	47.62%

Tabla 4.1: Resultados finales.

La tabla 4.1 muestra los resultados finales, en valores medios, que hemos obtenido al evaluar la memoria ACDC en un procesador de altas prestaciones en SPEC CPU 2006. Se puede ver que en ambas organizaciones de la memoria ACDC dentro de la jerarquía de memoria (secuencial y paralela), el rendimiento ha aumentado y el consumo de energía se ha reducido a pesar de que el número de aciertos en la DC no ha sido elevado. Con estos datos es aconsejable el uso de la memoria ACDC en un procesador de altas prestaciones como el expuesto en este documento.

Para que el uso de la memoria ACDC tenga éxito, hay que hacer un análisis previo y como se ha visto, los resultados dependen de él. Este análisis previo es el mayor inconveniente de esta propuesta.

4.2. Trabajos futuros.

En este proyecto se ha introducido la memoria ACDC en la jerarquía de memoria de un procesador de altas prestaciones con la idea de mejorar el rendimiento y disminuir su consumo energético en SPEC 2006.

Ya que en este Proyecto se ha conseguido alcanzar los objevos, para continuar con esta idea, y como trabajo futuro, se podría introducir este tipo de memoria en un sistema multiprocesador, con una memoria ACDC privada para cada procesador o bien con una más grande y común a todos los procesadores, en los que haya que tener en cuenta los protocolos de coherencia.

Otro posible trabajo futuro es el uso de esta memoria ACDC, pero sin hacer un profiling. Sería el compilador quien detecte en fase de compilación aquellas instrucciones que más reuso aprovechan del programa. También se podría hacer de forma adaptativa, mediante un hardware que vaya aprendiendo cuales son esas instrucciones.

Un estudio futuro interesante es ver qué pasaría si, al hacer este estudio, en lugar de usar la memoria ACDC se usa una memoria cache convencional del tamaño de la ACDC comparando algunas de las propuestas en el capitulo 1 con nuestro trabajo y ver si hay diferencias en rendimiento.

Apéndice A

Carga y Desarrollo del Proyecto

Este apéndice contiene detalles acerca de la gestión del tiempo y el esfuerzo invertido durante el proyecto, así como algunos problemas encontrados a lo largo de su desarrollo.

A.1. Gestión del tiempo

Comenzé este Proyecto de Fin de Carrera a principios del mes de mayo de 2014 y lo finalizé en mayo de 2015, dedicándome en exclusiva a su realización. En el diagrama de Gantt que se muestra en la figura A.1 se puede observar cómo hemos repartido las tareas a lo largo de la duración del proyecto.

Cuando comenzamos el proyecto, programamos hacer reuniones todos los viernes a las 10:00 horas de la mañana para seguir con la evolución del proyecto. Esto lo cumplimos a la perfección y la duración de las reuniones han variado entre una hora y tres horas. Además de acudir a tutorías cuando se presentaban dudas. La planificación inicial fue dividir el proyecto en tres partes:

- La primera parte consistiría en la instalación y aprendizaje del simulador, ya que era una herramienta completamente nueva para mí.
- La segunda parte la dedicaríamos a implementar las dos organizaciones de la memoria ACDC con la jerarquía de memoria de un procesador de altas prestaciones.
- La tercera parte la dedicaríamos a la realización de la ejecución de las simulaciones con la herramienta de simulación SimpleScalar [3]. De esta forma obtendríamos los resultados, para sacar las conclusiones de este Proyecto.

La primera parte la hicimos según la planificación, pero las otras dos fueron encadenadas, esto es, cada vez que hacíamos unas modificaciones en el código, hacíamos las pruebas necesarios y así estudiar los resultados. Esta parte ha sido una constante a lo largo del proyecto. Durante

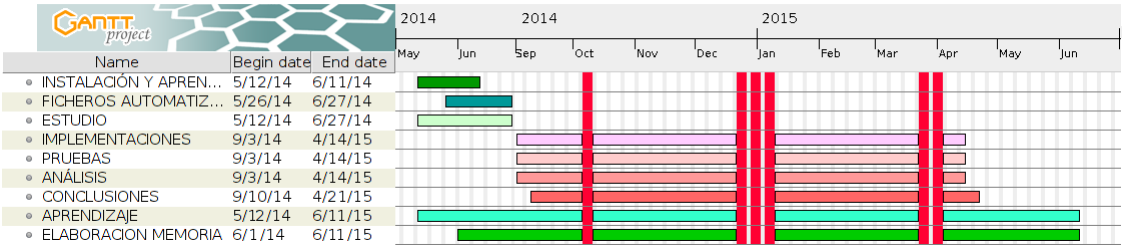


FIGURA A.1: Diagrama de Gantt del proyecto.

esta parte hemos usado algunas herramientas de ayuda como el depurador de C [13], ya que el simulador de SimpleScalar [3] está escrito en lenguaje C.

A continuación se muestra un resumen del trabajo que hemos realizado en cada tarea.

- **Instalación y aprendizaje.** En esta parte del proyecto lo que hicimos fue la instalación del simulador que usamos para las simulaciones [3] y estudiar cómo funciona y sus posibles configuraciones.
- **Scripts de automatización.** En este proyecto íbamos a realizar muchas pruebas, lo que aconsejó que se automatizaran la mayor parte de ellas para minimizar los tiempos de trabajo. En esta parte, lo que hice fueron los ficheros de automatización para lanzar el simulador con cada benchmark (scripts en unix [14]), ficheros para coger la información de los ficheros resultados de la ejecución de las simulaciones con Python [15] y todos aquellos que nos han hecho falta, según íbamos trabajando.
- **Aprendizaje gdb.** Como se ha dicho anteriormente, las moficicaciones se realizaron en el simulador SimpleScalar [3] que está realizado en Lenguaje C. Para la búsqueda de errores, hemos usado la herramienta de depuración de C [13], que ha sido muy importante para el desarrollo del proyecto.
- **Implementaciones.** Las implementaciones realizadas sobre el simulador [3] han durado prácticamente todo el Proyecto.
- **Pruebas.** En la parte de pruebas lanzamos el simulador con la configuración que queremos y esperamos a que termine y nos lance los ficheros de resultados.
- **Análisis.** En esta fase de análisis, estudiamos los ficheros de resultados que hemos obtenido en las pruebas.

  Estas tres tareas anteriores están ligadas entre sí, puesto que se relacionan. Se hacen implementaciones, pruebas y se observan resultados.
- **Conclusiones.** En esta tarea se extraen las conclusiones finales de las tareas anteriores.
- **Elaboración memoria** Esta parte se corresponde con la redacción de la memoria en LaTex.

A.2. Esfuerzo invertido

La duración inical del Proyecto era de seis meses. En el cómputo total de tiempo, he tardado 12 meses en realizarlo, aunque a este tiempo hay que descontarle los periodos de vacaciones, que han sido aproximadamente 3 meses. Por lo tanto, el tiempo que he tardado en realizar este Proyecto ha sido de 9 meses, con una media diaria de trabajo de 5 horas, siendo a partir del mes de septiembre cuando el trabajo comenzó a ponerse serio. En total, unas 900 horas, que en parte han sido por el desconocimiento de nuevas herramientas y sobre todo por iniciarse en el campo de la investigación. Mencionar que a lo largo de este Proyecto, he realizado más de 800 simulaciones, con el tiempo que ello supone.

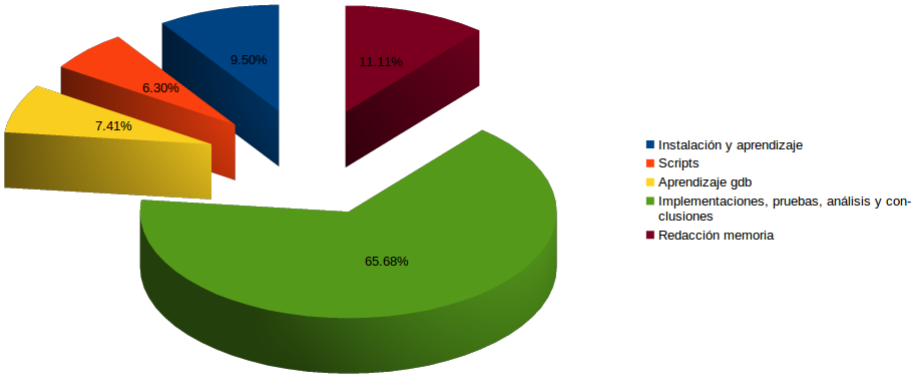


FIGURA A.2: Distribución del tiempo invertido en la realización de este Proyecto de Fin de Carrera.

En la gráfica A.2 se presenta el porcentaje de horas dedicadas a cada tarea. En esta gráfica se observa que casi el 65 % del tiempo se ha estado realizando implementaciones con pruebas y análisis de resultados. El resto del tiempo se ha dedicado al aprendizaje, instalación, realización de scripts y realización de memoria. Una parte importante que no la puedo cuantificar es el tiempo de aprendizaje propio, que ha durado todo el tiempo del proyecto.

A.3. Problemas encontrados

El primer gran problema que he encontrado ha sido el uso de implementaciones y trabajos hechas por otras personas. Esto que puede parecer una ventaja, en general no lo es, ya que en muchas ocasiones hay que estudiar qué hacen las cosas y por qué lo hacen.

Otro inconveniente ha sido el uso de herramientas nuevas, aunque solamente al principio del proyecto ya que una vez aprendidas a usarlas, todo parece más fácil.

Aúnque no es un problema como tal, pero está relacionado con el campo de la investigación, el tiempo empleado en la realización de las pruebas es importante, ya que supone esperar a que terminen para poder sacar conclusiones y a veces ralentiza las ganas de trabajar.

Finalmente, al ser mi primera vez que me acerco al campo de la investigación, en general, al principio se dan palos de ciego. Por ejemplo, cosas que son triviales, no las ves hasta que te enseñan a verlas.

Apéndice B

Procesadores de altas prestaciones y SPEC CPU 2006

B.1. Los procesadores de altas prestaciones

Como se dijo en la sección 2.1, los procesadores de altas prestaciones suelen tener las siguientes características: altamente segmentados, superescalares y con ejecución fuera de orden de instrucciones. Gracias a todo esto, se consigue aumentar su rendimiento ya que se mejora su capacidad para explotar el paralelismo a nivel de instrucciones (ILP).

El procesador modelado en este proyecto tiene un segmentado de instrucción de 10 etapas en el cual se distinguen una parte en la que se lanzan instrucciones en orden, *front-end* y está formada por las etapas de búsqueda, decodificación, renombre y emisión de la instrucción, y otra parte, que corresponde al núcleo de ejecución fuera de orden, *back-end*, conteniendo las etapas de lanzamiento, lectura información relevante para la ejecución, lectura de operandos, ejecución, escritura y finalización de la instrucción. Al conjunto de todas estas etapas se le llama *pipeline* del procesador. Una breve descripción de estas etapas es:

- Búsqueda (**IF**), se busca la instrucción en memoria y carga en el procesador.
- Decodificación (**DE**), decodificación de la instrucción.
- Rename (**RE**), renombre de registros.
- Emisión (**DI**), emisión de la instrucción lista para emitirse.
- Lanzamiento (**IQ**), lanzamiento fuera de orden de las instrucciones.
- Payload (**P**), lectura de información importante para ejecutar instrucciones.
- Lectura ops. (**R**), acceso a registros.

- Ejecución (**E o M0, ..., Mn**), ejecución de la instrucción o acceso a la jerarquía de memoria, siendo n los ciclos de latencia.. Para el caso que sea un acceso a memoria, dentro del primer ciclo de dicha etapa se calcula la dirección de acceso al dato.
- Escritura (**WB**), escribir en registros el resultado.
- Consolidación (**CT**), consolidación y retirada de la instrucción en orden.

B.1.1. Lanzamiento a ejecución de instrucciones especulativamente.

Con objeto de maximizar la cantidad de paralelismo a nivel de instrucción, los procesadores de altas prestaciones, deben encadenar la ejecución de instrucciones dependientes (figura B.1). Es decir, que las etapas de ejecución de dichas instucciones dependientes puedan tener lugar en ciclos consecutivos. Dada la latencia existente entre la etapa de lanzamiento IQ, y la etapa de ejecución E, es necesario que las instrucciones dependientes sean lanzadas a ejecución antes de que sus operandos estén calculados (ver cronograma figura B.1) para conseguir dicho encadenamiento o *back-to-back execution*.

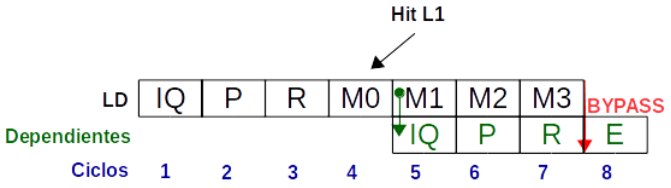


FIGURA B.1: Cronograma en el que se muestra el encadenamiento de instrucciones dependientes cuando hay acierto en la predicción de la latencia de ejecución.

En caso contrario, cuando las instrucciones dependientes no pueden ejecutarse encadenadamente, las perdidas del rendimiento pueden ser de más de 10 %, [16].

Para aquellas instrucciones cuya latencia de ejecución es conocida en su etapa de lanzamiento, es sencillo y seguro calcular cuando deben ser lanzadas sus instrucciones dependientes para que haya encadenamiento.

Por otra parte, hay otras instrucciones, por ejemplo las instrucciones de acceso a memoria, cuya latencia de ejecución no se conoce ya que esta latencia varía dependiendo de en qué nivel se encuentra el contenio accedido. En este escenario y para conseguir la ejecución encadenada de instrucciones, se requiere predecir una latencia de ejecución para la instrucción de acceso a memoria, típicamente latencia de acierto en L1, y así lanzar las instrucciones dependientes de manera especulativa. En caso de acierto en la predicción, las instrucciones dependientes lanzadas especulativamente tendrán su operando fuente disponible a tiempo para la ejecución (ver figura B.1). En caso de fallo de predicción, aquellas instrucciones que fueron lanzadas especulativamente, deben ser anuladas. Más tarde, cuando el dato esté disponible en la L1, se deberá relanzar dichas instrucciones dependientes [17], como se muestra en la figura B.2 para conseguir el encadenamiento.

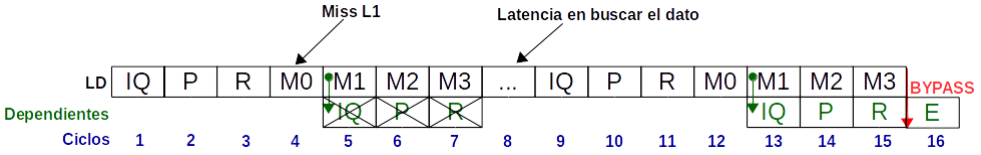


FIGURA B.2: Cronograma en el que se muestra el encadenamiento de instrucciones después de un fallo en la predicción y relanzamiento.

B.1.2. Configuración del procesador de altas prestaciones

En la figura B.1 se puede ver la configuración del procesador de altas prestaciones simulado en este Proyecto.

Ancho de las etapas	Fetch	8 instrucciones
	Decode	
	Rename	
	Issue	
	Commit	
Tamaño RUU		256 instrucciones
Tamaño LDST		128 instrucciones
Lanzamiento inst. int		64 instrucciones
Lanzamiento inst pf		32 instrucciones
Memoria L1	Tamaño	64KB
	Asociatividad	4
	Tamaño bloque	32B
	Latencia	4 ciclos
Memoria L2	Tamaño	256KB
	Asociatividad	8 instrucciones
	Tamaño bloque	128B
	Latencia	14 ciclos
Memoria L3	Tamaño	4MB
	Asociatividad	4
	Tamaño bloque	128B
	Latencia	19 ciclos
Memoria Principal	Latencia	200 ciclos
Memoria ACDC	Tamaño	256B
	Asociatividad	Completamente asociativa
	Tamaño bloque	16B
	Latencia	1 ciclo
Puertos memoria		4
ALU's enteras		8
Multiplicadores/divisores		2
ALU's pf		4
Multiplicadores pf		4

TABLA B.1: Configuración del procesador de altas prestaciones utlizado en este Proyecto para las simulaciones.

B.2. Cargas de trabajo

La Standard Performance Evaluation Corporation (SPEC) [2] es una corporación creada para establecer, mantener y apoyar un conjunto standarizado de puntos de referencia relevantes que se pueden aplicar a las nuevas generaciones de computadoras de alto rendimiento.

La CPU SPEC 2006 es un conjunto de pruebas intensivo que hace hincapié en el procesador y la memoria del sistema. Estos programas se llaman *benchmark* y sirven para evaluar el rendimiento de un computador completo o de uno de sus subsistemas. Un conjunto de benchmarks se denomina *suite*. Este SPEC CPU 2006 tiene dos suites, una para los enteros y otra para los float.

Para la realización de nuestro proyecto hemos usado los benchmarks enteros que se muestran en la tabla B.2, y los benchmars float de la tabla B.3. Para saber más sobre cada benchmark puede leer las descripciones aquí [18].

NOMBRE	ÁMBITO APLICACION
Perlbench	Programming language
Bzip2	Compression
Gcc	C Language optimizing compiler
Mcf	Combinatorial optimization / single-depot vehicle scheluding
Gobmk	Artificial inteligence - game playing
Hmmer	Search a gene sequence database
Sjeng	Artificial inteligence
Libquantum	Physics / Quantum Computing
H264ref	Video compression
Omnetpp	Descrete event simulation
Astar	Computer games. Artificial inteligence. Path finding

Tabla B.2: Benchmarks enteros usados en la ejecución de las simulaciones.

NOMBRE	ÁMBITO APLICACION
Bwaves	Computacional fluid dynamics
Gamess	Quantum chemical computations
Milc	Physics / Quantum chromodynamics (QCD)
Zeusmp	Physics / Magnetohydrodynamics
Gromacs	Chemistry / Molecular dynamics
CactusADM	Physics / General relativity
Leslie3d	Computational fluid dynamics (CFD)
Namd	Scientific structural biology
DealII	Solution of partial differentiall equations
Soplex	Simplex linear program (LP) solver
Povray	Computer visualization
Calculix	Structural mechanics
GemsFDTD	Computational electromagnetics (CEM)
Tonto	Quantum crystallography
Lbm	Computational fluid dynamics, lattice Boltzmann method
Wrf	Weather forecasting
Sphin3	Speech recognition

Tabla B.3: Benchmarks floats usados en la ejecución de las simulaciones.

Apéndice C

Descripción y funcionamiento de la memoria ACDC

C.1. Descripción y funcionamiento de la memoria ACDC

Como ya comentamos en el capitulo 1, en la memoria ACDC [1], el contenido de la memoria es controlado por las instrucciones y no por el flujo de direcciones de accesos generados por el programa [1]. A continuación mostramos un ejemplo para ver cómo algunas instrucciones pueden marcar el reuso del programa.

En la figura C.1 se puede ver un fragmento de programa sobre el cual se va a explicar el reuso

```
1:      int i, b, p1, p2;
        ...
2:      for(i = 0; i < 100; i++)
        {
                ...
3:              b = p2 * 5;             // Reuso escalar temporal
4:              a[i] = a[i] + p1*b;     // Reuso espacial stride 1
                ...
        }
```

FIGURA C.1: Fragmento de código de un programa.

y cómo la memoria ACDC es capaz de aprovechar ese reuso. Este código tiene un bucle con 100 iteraciones, el cual contiene varias instrucciones. La instrucción de la línea 3 de la figura C.1 muestra reuso escalar temporal, puesto que estamos accediendo a la misma variable todas las iteraciones del bucle. En la instrucción de la línea 4 de esa misma figura C.1, tenemos reuso espacial de stride 1 ya que accedemos a la misma posición del vector para leer y escribir, con salto 1. En la figura C.2 nos muestra estas instrucciones escritas en código máquina. En la línea 4 de la figura C.2, el store está accediendo a la misma variable "b" para escribir un dato, tantas veces como iteraciones del bucle. Si solo los bloques accedidos por estas instrucciones fuesen mantenidos en cache, se conseguiría aprovechar siempre todo el reuso que presenta el algoritmo. Por otra parte en la línea 8 de la figura C.2, se lee un dato de un vector en una posición, para

después en la línea 10 de esa misma figura, almacenar en esa misma posición un dato. En estos dos ejemplos se muestra reuso de datos.

```
1:      LD      $1, p2
2:      LD      $2, #5
3:      MUL     $3, $1, $2
4:      ST      b, $3
5:      LD      $4, p1
6:      MUL     $5, $4, $3
7:      LD      $6, i
8:      LD      $7, a[$6]
9:      ADD     $8, $7,$5
10:     ST      a[$6], $8
```

FIGURA C.2: Código máquina de las instrucciones del bucle del ejemplo de la figura C.1.

Una posible estructura hardware que puede aprovechar la idea que acabamos de ver sobre este reuso es una memoria muy pequeña y completamente asociativa (DC), con una política de reemplazo basada en instrucciones. Por ejemplo, en aciertos de lectura, se proporcionaría el dato; en aciertos de escrituras se actualizaría el dato. En fallo de memoria, se comprobará si la instrucción de acceso al dato tiene permiso para reemplazar. Si lo tuviese, la línea asociada de la memoria es reemplazada por el dato traído de memoria principal. En caso contrario no se reemplaza y el dato es proporcionado por la memoria principal.

Esta idea puede ser implementada mediante una tabla que guarda los PC (direcciones de instrucciones) de aquellas instrucciones que pueden reemplazar un línea de memoria y una línea específica que se quiere sustituir (DC icx). Esta tabla, por ejemplo AC, será una memoria pequeña y completamente asociativa donde se guardan dichos PC's y un puntero o índice de la línea de la DC que se quiere reemplazar (figura C.3).

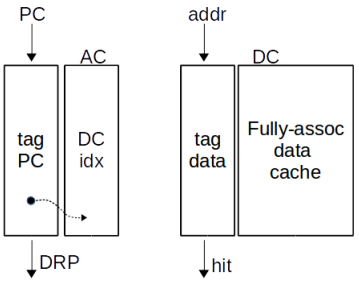


FIGURA C.3: Estructura que implementa la ACDC.

El funcionamiento es el siguiente. Se accede a la DC con la dirección del dato (addr) para ver si el dato requerido se encuentra ahí (C.3). En caso de fallo (miss), cuando el dato de memoria principal esté disponible tenemos que saber si lo podemos guardar en una línea de la DC. Para ello, con esta propuesta de memoria, se accede a la AC con la dirección de la instrucción y comprobamos que se encuentra ahí. Si no se encuentra, el dato no será reemplazado en la DC. Si hay acierto en AC, el dato tiene permiso de reemplazo en la DC (DRP) y la línea de la DC a reemplazar viene dada por el índice asociado a la línea de la AC (DC idx) en que se encuentre el PC. Entonces el dato es reemplazado. En caso de acierto (hit) tanto de lecturas como de escrituras, el proceso termina.

Con esta descripción, en nuestro ejemplo de la figura C.2, si en la AC tuviésemos el PC del load que accede al vector (línea 8 de la figura C.2), la primera vez que se accediese al dato, tendremos

miss en DC, pero al tener DRP, actualizaremos el contenido de la DC asociado al índice del PC_load y en el siguiente acceso al dato en la DC (línea 10 de la figura C.2), tendremos un hit. Hemos aprovechado el reuso espacial de stride 1 para aumentar el rendimiento.

Apéndice D

Implementaciones

En este apéndice se van a mostrar las modificaciones hechas en el simulador usado en este Proyecto [3] y las implementaciones de nuevos métodos usados, así como los ficheros de automatización del trabajo (para intentar disminuir el tiempo de trabajo).

D.1. Modificaciones en simulador [3]

En esta sección se muestran los cambios hechos en el código del simulador [3] para adaptarlo a las distintas organizaciones de la memoria ACDC según lo explicado en el capítulo 2 de este documento. Para nuestro trabajo partimos del este simulador [3] modificado por Jesús Alastruey (profesor de la Universidad de Zaragoza) cuyas modificaciones hacen que su funcionamiento se parezca más a un procesador real. Se pueden leer estas modificaciones en su tesis doctoral [19]. Los cambios se realizan en el archivo *"sim-outorder.base.c"*. La figura D.1 muestra las etapas del simulador como el que se ha explicado en el apéndice B de este documento, en el cual se aprecian las colas usadas en este Proyecto.

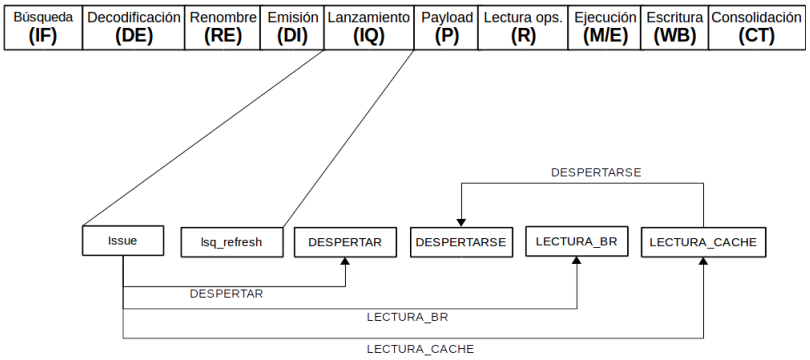


FIGURA D.1: Esquema de las etapas del procesador simulado en este proyecto con las colas asociadas a las etapas.

Los cambios introducidos en el código del simulador afectan a la etapa de ejecución, en concreto, a los accesos a memoria (M) y a la etapa de lanzamiento (IQ).

Para entender mejor el simulador, el núcleo principal de funcionamiento se encuentra en la función sim_main(). Aquí se comienza con una fase de inicialización para posteriormente pasar a un bucle infinito en el que cada iteración correrresponde a un ciclo del procesador. En cada iteración se ejecutan las funciones que implementan las distintas etapas del procesador. Se observa que se recorren al revés estas etapas del pipeline lo que permite que se puedan manejar correctamente los registros de sincronización entre etapas con una sola pasada a lo largo de cada etapa. Para terminar la ejecución se hace una llamada al sistema con exit().

```
ruu_init();

for( ; ;) {
        ruu_commit();               // Modificado
        ruu_release_fu();
        ruu_writeback();
        ruu_actualizar_acdc();      // Nuevo
        ruu_lectura_cache();        // Modificado
        ruu_missDC_hitL1();         // Nuevo
        ruu_lectura_br();
        ruu_despertarse();
        ruu_despertar();
        lsq_refresh();
        ruu_issue();                // Modificado
        ruu_dispatch();
        ruu_rename();
        ruu_decode();
        ruu_fetch();
    }
```

FIGURA D.2: Ejecución de las etapas del simulador que implementa el procesador usado en este Proyecto.

La figura D.2 se muestran las todas las etapas, señalando las que he modificado en este Proyecto o las nuevas que he creado.

Los cambios realizados explicados de una forma general, son los siguientes: En la etapa de lanzamiento, ruu_issue(), he cambiado la predicción de acertar en la cache de nivel 1 a que acierte en la memoria DC. Es decir, lanzamos las instrucciones dependientes pensando que vamos a acertar en la DC. Se observa en la figura D.1 que en etapa de lanzamiento (IQ), está implementada con dos funciones. Una es issue, que manda un evento a la cola DESPERTAR, que almacena aquellas instrucciones que tiene que despertar dependientes. También envía un evento a la cola LECTURA_BR, para aquellas instrucciones que tienen que hacer lectura de registros y finalmente envía un evento a la cola LECTURA_CACHE, que son aquellas instrucciones que tienen que acceder a memoria. Aquí, en lectura_cache, si el dato no se encuentra en la L1, se envía un evento a la cola DESPERTARSE, que almacena los loads que esperan su dato de la jerarquía de memoria. En cada ciclo de ejecución se accede a estas colas y si procede, se van sacando instrucciones de ellas. Y por otra parte está lsq_refresh, que lo que hace es comprobar que las instrucciones tiene las dependencias resueltas.

La función de acceso a memoria, ruu_lectura_cache, la he modificado para introducir las dos organizaciones expuestas a lo largo de este documento (ver capítulo 2). Hay tres modos de funcionamiento, el simulador sin ACDC (acdc == 0), con la organización paralela (acdc == 1) y con la organización secuencial (acdc == 2), que son opciones que he añadido al fichero de configuración que se le pasa al simulador. Para el modo solo, no he realizado modificaciones. Para

el *modo paralelo*, he modificado el código de acuerdo a la idea de organización paralela explicada en el capítulo 2.3.2. Es importante recalcar aquí, que se accede a la vez a ambas memoria (L1 y DC). Y para el *modo secuencial*, las modificaciones están hechas de acuerdo al capítulo 2.3.1, matizando que solo accedemos a la cache L1 cuando fallamos en DC.

Además he realizado dos nuevas funciones, ruu_missDC_hitL1() y ruu_actualizar_acdc(), con el uso de otras dos nuevas colas. La cola FALLODC_ACIERTOL1, en la que meto aquellas instrucciones que fallan en DC y aciertan en L1, para despertarlas cuando corresponda. La función ruu_missDC_hitL1(void), vacía esta cola. Y la cola ACTUALIZAR_ACDC, donde meto aquellas instrucciones que tienen que actualizar la DC, y que la función ruu_actualizar_acdc(void) vacía esta cola. En la figura D.3 se pueden ver todas las colas que he usado para este Proyecto.

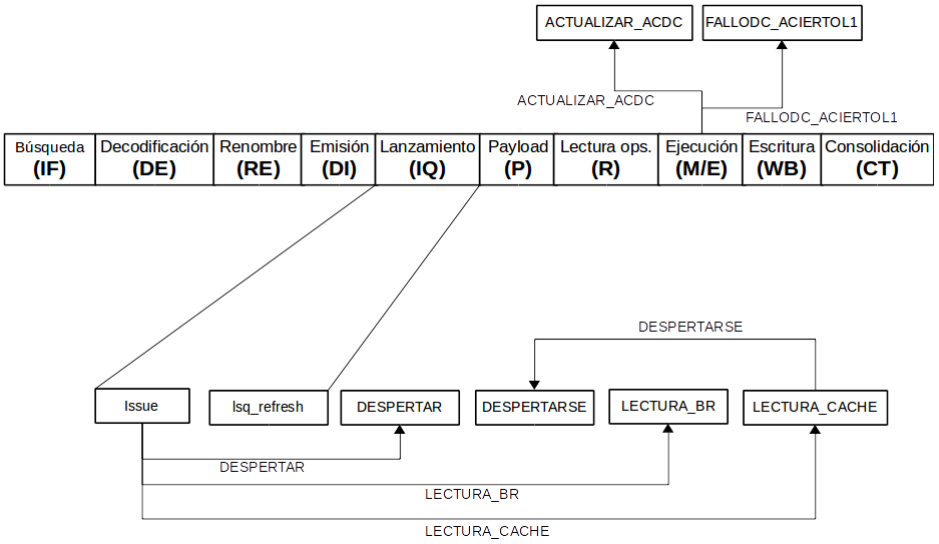


FIGURA D.3: Segmentado del procesador usado en este Proyecto con todas las colas que he usado.

También he realizado cambios en algunas funciones para obtener los resultados que necesitaba y estadísticas, así como en los ficheros .h. Para saber más sobre SimpleScalar, puede leer el tutorial aquí [20].

D.2. Implementaciones

En esta sección se muestra el código de las implementaciones realizadas.

```
void sim_reg_options(struct opt_odb_t *odb) {
...
opt_reg_int(odb, "-uso_acdc", "0 = jerarquia convencional; 1 = paralelo; 2 = secuencial",
            &acdc, /* default */0, /* print */TRUE, /* format */NULL);
opt_reg_int(odb, "-tramos", "Son los tramo para sacar los pc's", &tramos, 0, TRUE, NULL);
opt_reg_int(odb, "-crearficheros", "Son los tramos para sacar los pc's", &crearficheros,
            0, TRUE, NULL);
opt_reg_string(odb, "-cache:acdc", "bloque:vias", &cache_acdc_opt, "acdc:4:4", TRUE, NULL);
opt_reg_int(odb, "-cache:acdcHitLatency", "latencia de la memoria acdc en aciertos",
            &cache_acdc_hit_latency, /* default */ 1, /* print */TRUE, /* format */NULL);
opt_reg_int(odb, "-cache:acdcMissLatency", "latencia de la memoria acdc en fallos",
            &cache_acdc_miss_latency, /* default */ 1, /* print */TRUE, /* format */NULL);
...
}
```

```
void sim_check_options(struct opt_odb_t *odb, int argc, char **argv)
{
...
    if (acdc < 0 || acdc > 2)
        fatal("El valor de ACDC no debe ser mayor de 2 y ha puesto: %d", acdc);
    /* Creamos la memoria cache_ACDC */
    if (!mystricmp(cache_acdc_opt, "none"))
        cache_acdc = NULL;
    else
    {
        if (sscanf(cache_acdc_opt, "%[^:]:%d:%d",name,&bloque, &via) != 3)
            fatal("Los parametros no estan bien. Son: <name>:<bloque>:<via>");
        cache_acdc = newACDC(bloque, via);
    }

    if (cache_acdc_hit_latency < 1)
        fatal("ACDC _hit_latency debe ser mayor que cero");
    if (cache_acdc_miss_latency < 1)
        fatal("ACDC_miss_latency debe ser mayor que cero");
...
}
```

```
void sim_reg_stats(struct stat_sdb_t *sdb)
{
...
    stat_reg_counter(sdb, "sim_total_issued_loads \t","numero de load isuados",
                     &sim_total_issued_loads,
                      sim_total_issued_loads,NULL);
    stat_reg_counter(sdb, "sim_total_issued_stores\t","numero de store isuados",
                     &sim_total_issued_stores,
                      sim_total_issued_stores,   NULL);
    stat_reg_counter(sdb, "sim_total_commited_loads",
                     "numero de load jubilados",
                     &sim_total_commited_loads,
                      sim_total_commited_loads,  NULL);
    stat_reg_counter(sdb, "sim_total_commited_stores",
                     "numero de store jubilados",
                     &sim_total_commited_stores,
                      sim_total_commited_stores, NULL);
...
}
```

```
void sim_aux_stats(FILE *stream)
{
...
    fprintf(stream, "ESTADISTICAS ACDC\n");
    if (acdc != 0)
    {
        fprintf(stream, "aciertosdc %10d # aciertos_dc", aciertos_dc);
        fprintf(stream, "fallosdc %10d # fallos_dc", fallos_dc);
        fprintf(stream, "total_accesos_dc %10d # total_accesos_dc",aciertos_dc+fallos_dc);
        fprintf(stream, "aciertosac %10d # aciertos_ac",aciertos_ac);
        fprintf(stream, "fallosac %10d # fallos_ac",fallos_ac);
        fprintf(stream, "total_accesos_ac %10d # total_accesos_ac",aciertos_ac+fallos_ac);
        fprintf(stream, "aciertos_dc_loads %10d # aciertos_dc_loads",aciertos_dc_loads);
        fprintf(stream, "fallos_dc_loads %10d # fallos_dc_loads",fallos_dc_loads);
        fprintf(stream, "total_dc_loads %10d # accesos totales a dc en loads,aciertos+fallos",
                aciertos_dc_loads+fallos_dc_loads);
        fprintf(stream, "aciertos_ac_loads %10d # aciertos_ac_loads\n", aciertos_ac_loads);
        fprintf(stream, "fallos_ac_loads %10d # fallos_ac_loads\n", fallos_ac_loads);
        fprintf(stream, "total_ac_loads %10d # accesos totales a ac en loads, aciertos +
                fallos",aciertos_ac_loads + fallos_ac_loads);
        fprintf(stream, "aciertos_dc_stores %10d # aciertos_dc_stores\n", aciertos_dc_stores);
        fprintf(stream, "fallos_dc_stores %10d # fallos_dc_stores\n", fallos_dc_stores);
        fprintf(stream, "total_dc_stores %10d # accesos totales a dc en stores, aciertos +
                fallos\n", aciertos_dc_stores + fallos_dc_stores);
        fprintf(stream, "aciertos_ac_stores %10d # aciertos_ac_stores\n", aciertos_ac_stores);
        fprintf(stream, "fallos_ac_stores %10d # fallos_ac_stores\n", fallos_ac_stores);
        fprintf(stream, "total_ac_stores %10d # accesos totales a ac en stores, aciertos +
                fallos\n", aciertos_ac_stores + fallos_ac_stores);
    }
    fprintf(stream, "FIN ESTADISTICAS ACDC\n");
...
}
```

```
// Creación de estructuras y colas
    static struct RS_link *falloDC_aciertoL1; // Para meter en las instrucciones
                                              // que fallan en DC y aciertan en L1
    static struct RS_link *actualizar_acdc;   // Para actualizar la ACDC

    #define FALLODC_ACIERTOL1    4        // Cola para meter los loads que fallan en DC y aciertan en L1
    #define ACTUALIZAR_ACDC      5        // Cola para meter los loads que tienen DRP
```

```
static void cola_init()
{
...
    falloDC_aciertoL1 = NULL;
    actualizar_acdc = NULL;
...
}
```

```
static void cola_queue(int tipo, struct RUU_station *rs, tick_t when)
{
...
    case FALLODC_ACIERTOL1:
        ev = falloDC_aciertoL1;
        break;
    case ACTUALIZAR_ACDC:
        ev = actualizar_acdc;
        break;

        ...
    case FALLODC_ACIERTOL1:
        new_ev->next = falloDC_aciertoL1;
        falloDC_aciertoL1 = new_ev;
        break;
    case ACTUALIZAR_ACDC:
        new_ev->next = actualizar_acdc;
        actualizar_acdc = new_ev;
        break;
...
}
```

```
static struct RUU_station* cola_next(int tipo)
{
...
    case FALLODC_ACIERTOL1:
        ev = falloDC_aciertoL1;
        break;
    case ACTUALIZAR_ACDC:
        ev = actualizar_acdc;
        break;
        ...
    case FALLODC_ACIERTOL1:
        falloDC_aciertoL1 = falloDC_aciertoL1->next;
        break;
    case ACTUALIZAR_ACDC:
        actualizar_acdc = actualizar_acdc->next;
...
}
```

```
static void lectura_cache(struct RUU_station *rs)
{
...
    if (acdc == 1) /* Paralelo */
    {
        if (rs->falloL1D == TRUE) /* Relanzamientos */
        {
            if (rs->drp != -1)
            {
                load_lat = cache_acdc_miss_latency;
            }
            else
            {
                load_lat = cache_dl1_lat;
            }
        }
        else
        {
            int valid_addr = MD_VALID_ADDR(rs->addr);
                if (!spec_mode && !valid_addr)
                {
                sim_invalid_addrs++;
            }
            if (cache_dl1 && cache_acdc && valid_addr)
            {
                lat1 = cache_access(cache_dl1, Read,
                   (rs->addr & ~3),
                    NULL, 4,
                    sim_cycle + sim_cycle_base,
                    NULL, NULL);
                rs->hit = isHit(cache_acdc, rs->addr);
                if (rs->hit != -1) /* Acierto en DC */
```

```
                {
                    aciertos_dc_loads++;
                    aciertos_dc++;
                    lat2 = cache_acdc_hit_latency;
                    load_lat = MIN(lat1, lat2);
                }
                else /* Fallo en DC */
                {
                    fallos_dc_loads++;
                    fallos_dc++;
                    lat2 = cache_acdc_miss_latency;
                    rs->drp = hasDRP(cache_acdc, rs->PC);
                    if (rs->drp != -1) // Tengo DRP
                    {
                        aciertos_ac_loads++;
                        aciertos_ac++;
                        cola_queue(ACTUALIZAR_ACDC, rs, sim_cycle+lat1);
                    }
                    else /* No tengo DRP */
                    {
                        fallos_ac_loads++;
                        fallos_ac++;
                    }
                    load_lat = lat1;
                }
            if (load_lat > cache_dl1_lat)
            {
                events |= PEV_CACHEMISS;
            }
        }
        else
        {
            load_lat = rs->latencia_exe;
        }
    }
} /* fin if(acdc == 1) */

if (acdc == 2) /* Secuencial */
{
    if (rs->falloL1D == TRUE ) /* Relanzamientos de loads que han fallado en L1 */
    {
        if (rs->drp != -1)
        {
            load_lat = cache_acdc_miss_latency;
        }
        else
        {
            load_lat = cache_dl1_lat+cache_acdc_miss_latency;
        }
    }
    else
    {
        rs->hit = isHit(cache_acdc, rs->addr);
        if (rs->hit != -1) // Acierto en DC
        {
            aciertos_dc_loads++;
            aciertos_dc++;
            load_lat = cache_acdc_hit_latency;
        }
        else // Fallo en DC
        {
            int valid_addr = MD_VALID_ADDR(rs->addr);
            if (!spec_mode && !valid_addr)
            {
                sim_invalid_addrs++;
            }
            if (cache_dl1 && valid_addr)
            {
                lat1 = cache_access(cache_dl1, Read,
                    (rs->addr & ~3),
                    NULL, 4,
                    sim_cycle + sim_cycle_base,
                    NULL, NULL);
                    load_lat = lat1 ;
                if (load_lat > cache_dl1_lat)
                {
                    events |= PEV_CACHEMISS;
                }
            }
            else
            {
                load_lat = rs->latencia_exe;
            }
            fallos_dc_loads++;
            fallos_dc++;
            rs->drp = hasDRP(cache_acdc, rs->PC);
            if (rs->drp != -1) // Tengo DRP
```

```
            {
                aciertos_ac_loads++;
                aciertos_ac++;
                cola_queue(ACTUALIZAR_ACDC, rs, sim_cycle+lat1);
            }
            else // No tengo DRP
            {
                fallos_ac_loads++;
                fallos_ac++;
            }
        }
    }
} // fin if(acdc == 2)

...

else // if (acdc == 0)
{
    if (rs->falloL1D == TRUE) /* Relanzamientos de loads que han fallado en L1 */
    {
        if (rs->drp != -1)
        {
            eventq_queue_event(rs, sim_cycle+cache_acdc_hit_latency);
            if (rs->ofisicos[0] != NULO)
                ciclo_wb[rs->ofisicos[0]]=sim_cycle+cache_acdc_hit_latency;
                IQint_saca(rs);
                ptrace_newstage(rs->ptrace_seq,PST_EXECUTE,((rs->ea_comp ? PEV_AGEN : 0)|events));
        }
        else
        {
            if (acdc == 1) /* Paralelo */
            {
                cola_queue(FALLODC_ACIERTOL1, rs, sim_cycle+1); /* Miss en DC, despierto sin
                                                                   especular */
                anula_dependientes(rs);
                eventq_queue_event(rs, sim_cycle+cache_dl1_lat);
                if (rs->ofisicos[0] != NULO)
                    ciclo_wb[rs->ofisicos[0]] = sim_cycle+cache_dl1_lat;
                IQint_saca(rs);
                ptrace_newstage(rs->ptrace_seq,PST_EXECUTE,((rs->ea_comp ? PEV_AGEN : 0)|events));
            }
            if (acdc == 2) // Secuencial
            {
                cola_queue(FALLODC_ACIERTOL1, rs, sim_cycle+2); /* Miss en DC, despierto sin
                                                                   especular */
                anula_dependientes(rs);
                eventq_queue_event(rs, sim_cycle+cache_dl1_lat+cache_acdc_miss_latency);
                if (rs->ofisicos[0] != NULO)
                    ciclo_wb[rs->ofisicos[0]] = sim_cycle+cache_dl1_lat+cache_acdc_miss_latency;
                IQint_saca(rs);
                ptrace_newstage(rs->ptrace_seq,PST_EXECUTE,((rs->ea_comp ? PEV_AGEN : 0)|events));
            }
        }
    }
    if (dtlb && MD_VALID_ADDR(rs->addr) && (rs->falloACDC == FALSE) && (rs->falloL1D == FALSE))
    {
        tlb_lat = cache_access(dtlb, Read,
                (rs->addr & ~3), NULL, 4,
                sim_cycle + sim_cycle_base,
                NULL, NULL);
        if (tlb_lat > 1)
        {
            events |= PEV_TLBMISS;
        }
        load_lat = MAX(tlb_lat, load_lat);
    }
    if (rs->hit != -1)
    {
        eventq_queue_event(rs, sim_cycle+cache_acdc_hit_latency);
        if (rs->ofisicos[0] != NULO)
            ciclo_wb[rs->ofisicos[0]] = sim_cycle+cache_acdc_hit_latency;
        IQint_saca(rs);
        ptrace_newstage(rs->ptrace_seq,PST_EXECUTE,((rs->ea_comp ? PEV_AGEN : 0)|events));
    }
    else // Fallo en DC
    {
        if ((load_lat > cache_dl1_lat) && (rs->falloL1D == FALSE)) // Fallo en L1. Relanzo.
        {
            rs->tag++;  // Para que el evento de esa cola (DESPERTAR) lo tome como no válido.
            rs->issued = FALSE; // Para que la instruccion se vuelva a lanzar
            rs->falloL1D = TRUE;
            rs->falloACDC = TRUE;
            rs->dormida = TRUE;
            cola_queue(DESPERTARSE, rs, sim_cycle+load_lat-3); // Programo el despertar de load dormido
            anula_dependientes(rs);
        }
        else // Acierto en l1
```

```
        {
            rs->falloACDC = TRUE;
            if (rs->falloL1D == FALSE) // Para que no entre cuando relanzo.
            {
                if (acdc == 1) // Paralelo
                {
                    anula_dependientes(rs);
                    cola_queue(FALLODC_ACIERTOL1, rs, sim_cycle+1);
                    eventq_queue_event(rs, sim_cycle + cache_dl1_lat);
                    if (rs->ofisicos[0] != NULO)
                        ciclo_wb[rs->ofisicos[0]] = sim_cycle+cache_dl1_lat;
                }
                else // Secuencial
                {
                    cola_queue(FALLODC_ACIERTOL1, rs, sim_cycle+2);
                    anula_dependientes(rs);
                    eventq_queue_event(rs, sim_cycle + cache_dl1_lat+cache_acdc_miss_latency);
                    if (rs->ofisicos[0] != NULO)
                        ciclo_wb[rs->ofisicos[0]] =sim_cycle+cache_dl1_lat+cache_acdc_miss_latency;
                }
                IQint_saca(rs);
                ptrace_newstage(rs->ptrace_seq,PST_EXECUTE,((rs->ea_comp ? PEV_AGEN : 0)|events));
            }
        }
    }
}
} % fin de lectura_cache
...
}
```

```
static void ruu_commit(void)
{
...
else /* if (acdc == 0) */
{
    if (cache_acdc)
    {
        a = isHit(cache_acdc, rs->addr);
        if (a != -1) /* Acierto en DC */
        {
            cache_acdc->Lineas[a].datatag = rs->addr;
            aciertos_dc_stores++;
            aciertos_dc++;
        }
        else
        {
            fallos_dc_stores++;
            fallos_dc++;
            b = hasDRP(cache_acdc,LSQ[LSQ_head].PC);
            if(b != -1) /* Tengo DRP */
            {
                cache_acdc->Lineas[b].datatag = rs->addr;
                aciertos_ac_stores++;
                aciertos_ac++;
            }
            else
            {
                fallos_ac_stores++;
                fallos_ac++;
            }
        }
    }
}
...
}
```

```
static void ruu_issue(void)
{
...
    sim_total_issued_loads++;
    else /* if (acdc == 0) */
    {
        issue_load_data_1(rs);
    }
...
}
```

```
static void ruu_decode(void)
{
...
    if (contador == fichero)
    {
        nombre = sim_eio_fname;
        if (crearficheros == 1)
```

```
        {
            strcat(nombrefinal, nombre);
            strcat(nombrefinal, ".");
            char aux[15];
            sprintf(aux,"%d",tramos);
            strcat(nombrefinal,aux);
            strcat(nombrefinal, ".");
            char aux2[99];
            sprintf(aux2, "%d", numerofichero);
            sprintf(aux2, "%d", fichero);
            if (fichero == 0)
                ;
            else
            {
                strcat(nombrefinal, aux2);
                char *definitivo = strcat(nombrefinal, ".def");
                prueba = fopen(definitivo,"w");
                perfprintPCCachePFC(prueba, perfdatatrace);
                printf("Fichero: %s creado con exito\n", definitivo);
                fclose(prueba);
            }
            *nombrefinal = 0;
            perfdatatrace = NULL;
            contador++;
            numerofichero++;
            cambio = 0;
        }
        else     // No creamos fichero de análisis
        {
            contador++;
            numerofichero++;
            cambio = 0;
        }
    }
    else
    {
        ...
        if (crearficheros == 1)          // Creamos ficheros de análisis
        {
            if (MD_OP_FLAGS(lsq->op) & F_MEM)
            {
                if ((MD_OP_FLAGS(lsq->op) & (F_MEM|F_LOAD)) == (F_MEM|F_LOAD))
                {
                    perfdatatrace = perfaccessPCCache(perfdatatrace, lsq->PC, 0,
                                    (lsq->addr & ~15), sim_num_refs);
                }
                if ((MD_OP_FLAGS(lsq->op) & (F_MEM|F_STORE)) == (F_MEM|F_STORE))
                {
                    perfdatatrace = perfaccessPCCache(perfdatatrace, lsq->PC, 1,
                                    (lsq->addr & ~15), sim_num_refs);
                }
            }
        }
    ...
    }
}
```

```
void sim_main(void)
{
...
    if (acdc > 0)
    {
        if( (tramos * numerofichero > sim_num_commit) && (cambio == 0) )
        {
            strcat(ficherocarga, sim_eio_fname);
            strcat(ficherocarga, ".");
            char aux1[15];
            sprintf(aux1, "%d", tramos);
            strcat(ficherocarga, aux1);
            strcat(ficherocarga, ".");
            char aux2[99];
            sprintf(aux2, "%d", numerofichero);
            strcat(ficherocarga, aux2);
            strcat(ficherocarga, ".");
            strcat(ficherocarga, "ana");
            puts(ficherocarga);
            file = fopen(ficherocarga, "r");
            for (i = 0; i < cache_acdc->dcways; i++)
            {
                fscanf(file, "%p", &pc);
                fscanf(file, "%d", &repes);
                fscanf(file, "%s", ldst);
                cache_acdc->Lineas[i].pcs.pc = pc;
            }
            fclose(file);
            *ficherocarga = 0;
```

```
            cambio = 1;
        }
    }
    ...

    ruu_actualizar_acdc();
    ...
    ruu_missDC_hitL1();
    ...
}
```

D.3. Métodos nuevos

En esta sección se muestran los métodos nuevos que he implementado dentro de *"sim-outorder.base.c"*.

```
static void actualizar_ACDC(struct RUU_station *rs)
{
        updateMiss(cache_acdc,rs->drp, rs->PC, rs->addr, sim_cycle);
}
```

```
static void ruu_actualizar_acdc(void)
{
        struct RUU_station *rs;

        while (rs = cola_next(ACTUALIZAR_ACDC))
        {
                actualizarACDC(rs);
        }
}
```

```
static void ruu_missDC_hitL1(void)
{
        struct RUU_station *rs;
        while (rs = cola_next(FALLODC_ACIERTOL1))
        {
                despierta_dep(rs, 0);
                despierta_dep(rs, 1);
        }
}
```

```
static void issue_load_data_1(struct RUU_station *rs)
{
    if (cache_acdc)
    {
        cola_queue(DESPERTAR, rs, sim_cycle + cache_acdc_hit_latency);
    }
    else
        cola_queue(DESPERTAR, rs, sim_cycle + rs->latencia_exe);
    cola_queue(LECTURA_CACHE, rs, sim_cycle);
}
```

D.4. Automatizaciones

En esta sección se muestran algunos scripts de automatización realizados. Estos scripts se han hecho para agilizar todo el proceso de ejecución de las simulaciones. Son ficheros .sh y se lanzan desde la línea de comandos del shell. También se muestran scripts en Phython [15] para la búsqueda de información que nos interesa de los ficheros resultado de la ejecución del simulador. Son los archivos .py. También se muestra los cambios hechos en el archivo que se le pasa como entrada al simulador (antonio.conf).

```
automatizartrazasenteros.sh
./sim-outorder.base -config antonio.conf trazas/400.perlbench-ref.eio 2>400.perlbench-ref.eio.resul
./sim-outorder.base -config antonio.conf trazas/401.bzip2-ref.eio 2>401.bzip2-ref.eio.resul
./sim-outorder.base -config antonio.conf trazas/403.gcc-ref.eio 2>403.gcc-ref.eio.resul
```

```
./sim-outorder.base -config antonio.conf trazas/429.mcf-ref.eio 2>429.mcf-ref.eio.resul
./sim-outorder.base -config antonio.conf trazas/445.gobmk-ref.eio 2>445.gobmk-ref.eio.resul
./sim-outorder.base -config antonio.conf trazas/456.hmmer-ref.eio 2>456.hmmer-ref.eio.resul
./sim-outorder.base -config antonio.conf trazas/458.sjeng-ref.eio 2>458.sjeng-ref.eio.resul
./sim-outorder.base -config antonio.conf trazas/462.libquantum-ref.eio 2>462.libquantum-ref.eio.resul
./sim-outorder.base -config antonio.conf trazas/464.h264ref-ref.eio 2>464.h264ref-ref.eio.resul
./sim-outorder.base -config antonio.conf trazas/471.omnetpp-ref.eio 2>471.omnetpp-ref.eio.resul
./sim-outorder.base -config antonio.conf trazas/473.astar-ref.eio 2>473.astar-ref.eio.resul
```

```
automatizartrazasfloat.sh
./sim-outorder.base -config antonio.conf trazas/410.bwaves-ref.eio      2>410.bwaves-ref.eio.resul
./sim-outorder.base -config antonio.conf trazas/416.gamess-ref.eio      2>416.gamess-ref.eio.resul
./sim-outorder.base -config antonio.conf trazas/433.milc-ref.eio        2>433.milc-ref.eio.resul
./sim-outorder.base -config antonio.conf trazas/434.zeusmp-ref.eio      2>434.zeusmp-ref.eio.resul
./sim-outorder.base -config antonio.conf trazas/435.gromacs-ref.eio     2>435.gromacs-ref.eio.resul
./sim-outorder.base -config antonio.conf trazas/436.cactusADM-ref.eio   2>436.cactusADM-ref.eio.resul
./sim-outorder.base -config antonio.conf trazas/437.leslie3d-ref.eio    2>437.leslie3d-ref.eio.resul
./sim-outorder.base -config antonio.conf trazas/444.namd-ref.eio        2>444.namd-ref.eio.resul
./sim-outorder.base -config antonio.conf trazas/447.dealII-ref.eio      2>447.dealII-ref.eio.resul
./sim-outorder.base -config antonio.conf trazas/450.soplex-ref.eio      2>450.soplex-ref.eio.resul
./sim-outorder.base -config antonio.conf trazas/453.povray-ref.eio      2>453.povray-ref.eio.resul
./sim-outorder.base -config antonio.conf trazas/454.calculix-ref.eio    2>454.calculix-ref.eio.resul
./sim-outorder.base -config antonio.conf trazas/459.GemsFDTD-ref.eio    2>459.GemsFDTD-ref.eio.resul
./sim-outorder.base -config antonio.conf trazas/465.tonto-ref.eio       2>465.tonto-ref.eio.resul
./sim-outorder.base -config antonio.conf trazas/470.lbm-ref.eio         2>470.lbm-ref.eio.resul
./sim-outorder.base -config antonio.conf trazas/481.wrf-ref.eio         2>481.wrf-ref.eio.resul
./sim-outorder.base -config antonio.conf trazas/482.sphinx3-ref.eio     2>482.sphin3-ref.eio.resul
```

```
antonio.conf
// Archivo de configuración que se le pasa al simulador con la configuración deseada
# cache acdc
-uso_acdc                1              # 0 sin acdc, 1 paralelo, 2 secuencial
-crearficheros           0              # 1 creamos ficheros de analisis, 0 no los creamos
-tramos          100.000.000            # num. instrucciones
-cache:acdc      ACDC:4:4               # nombre, bloques, vias

# latencia de la acdc
-cache:acdcHitLatency    1              # latencia en caso de acierto
-cache:acdcMissLatency   1              # latencia en caso de fallo
```

```
leerarchivos.py
# Con este archivo tomamos los ficheros que hemos creado con la extension resul resultantes de la ejecución de la simulación.

#!/usr/bin/python
def leerarchivos():
        import os
        camino = "./"
        directorios = os.listdir(camino)

        # Me creo una lista para guardar todos los archivos que me interesen.
        lista = []

        # Hacemos un listado con listdir()
        directorios = os.listdir(camino)

        # Elegimos que extension vamos a elegir de archivo.
        palabra = ".resul"
        for i in directorios:
                if palabra in i:
                        lista.append(i)
        return lista                                    # Devuelvo la lista
leerarchivos()
```

```
leerficheroenteros.py

# Con este archivo, cogemos de cada fichero de resultados la información que necesitamos (ipc,...).
# En este caso es para los resultados de la suite enteros.
# Análogo a este fichero he hecho otro para recoger los resultados de la suite float.
# Los cambios afectan solamente al fichero donde se guardan los datos (datosFLOAT.txt) y al nombre de fichero (leerficherofloat.py).
# No se incluye el fichero "leerficherofloat.py.en este documento.

#!/usr/bin/python

# Importamos lo que nos hace falta.
from leerarchivos import leerarchivos

# Le paso como argumento la lista obtenida en leerarchivos.py
def leerFicheroEnteros(lista):
```

```
    # Aqui guardamos los datos que nos interesan.
    datos = open("./datosENTEROS.txt","a")

    datos.write("\nNOMBRE," + "Modo," + "tramos," + "bloque-vias," + "sim_total_issued_loads," + "
sim_total_issued_stores," + "sim_total_commited_loads," + "sim_total_commited_stores," + "
sim_num_refs," + "sim_num_loads," + "sim_num_stores," + "sim_elapsed_time," + "sim_total_refs," +
 "sim_total_loads," + "sim_total_stores," + "sim_cycle,"+ "sim_IPC," +"dl1.accesses," + "dl1.hits
," + "dl1.misses," + "dl1.replacements," + "dl1.writebacks," + "ul2.accesses," + "ul2.hits," + "
ul2.misses," + "ul2.replacements," + "ul2.writebacks," + "ul3.accesses," + "ul3.hits," + "ul3.
misses," + "ul3.replacements," + "ul3.writebacks," + "dtlb.accesses," + "dtlb.hits," + "dtlb.
misses," + "dtlb.replacements," + "aciertos_dc," + "fallos_dc," + "total_accesos_dc," + "
aciertos_ac," + "fallos_ac," + "total_accesos_ac," + "aciertos_dc_loads," + "fallos_dc_loads," +
"total_dc_loads," + "aciertos_ac_loads," + "fallos_ac_loads," + "total_ac_loads," + "
aciertos_dc_stores," + "fallos_dc_stores," + "total_dc_stores," +"aciertos_ac_stores," + "
fallos_ac_stores," + "total_ac_stores,\n")

    # Recorro la lista pasada como argumento.
    for archivo in lista:

        # Le damos el camino (path) + el nombre para abrir el archivo.
        camino = "./"
        nombre = camino + "/" + archivo

        # Abrimos el archivo
        fh = open(nombre)

        # Leemos linea a linea
        linea = fh.readlines()

        # Recorremos cada linea y buscamos lo que nos interese
        for l in linea:
            if "loading" in l:          # Para cargar el nombre del fichero de prueba
                x = archivo[:23]# Tomo los 23 primeros caracteres del nombre del archivo
                datos.write("\n" + x)
            if "uso_acdc" in l:
                x = l[l.find("uso_acdc")+16:]
                w = x[:x.find("#")]       # Eliminamos los comentarios
                w = w.lstrip()            # Con lstrip(), alineamos a la izquierda.
                datos.write("," + w )
            if "tramos" in l:
                x = l[l.find("tramos")+6:]
                w = x[:x.find("#")]       # Eliminamos los comentarios
                w = w.lstrip()            # Con lstrip(), alineamos a la izquierda.
                datos.write("," + w )
            if "ACDC:" in l:
                x = l[l.find("ACDC:")+5:]
                w = x[:x.find("#")]
                w = w.lstrip()
                datos.write("," + w )
            if "sim_cycle" in l:
                x = l[l.find("sim_cycle")+9:]
                w = x[:x.find("#")]
                w = w.lstrip()
                datos.write("," + w )
            if "sim_elapsed_time" in l:
                x = l[l.find("sim_elapsed_time")+16:]
                w = x[:x.find("#")]
                w = w.lstrip()
                datos.write("," + w)
            if "sim_IPC" in l:
                x = l[l.find("sim_IPC")+7:]
                w = x[:x.find("#")]
                w = w.lstrip()
                datos.write("," + w )
            if "sim_num_loads" in l:
                x = l[l.find("sim_num_loads")+13:]
                w = x[:x.find("#")]
                w = w.lstrip()
                datos.write("," + w )
            if "sim_num_stores" in l:
                x = l[l.find("sim_num_stores")+14:]
                w = x[:x.find(".")]
                w = w.lstrip()
                datos.write("," + w )
            if "sim_num_refs" in l:
                x = l[l.find("sim_num_refs")+12:]
                w = x[:x.find("#")]
                w = w.lstrip()
                datos.write("," + w )
            if "sim_total_loads" in l:
                x = l[l.find("sim_total_loads")+15:]
                w = x[:x.find("#")]
                w = w.lstrip()
                datos.write("," + w )
            if "sim_total_stores" in l:
                x = l[l.find("sim_total_stores")+16:]
                w = x[:x.find("#")]
```

```
            w = w.lstrip()
            datos.write("," + w )
    if "sim_total_refs" in l:
            x = l[l.find("sim_total_refs")+14:]
            w = x[:x.find("#")]
            w = w.lstrip()
            datos.write("," + w )
    if "dl1.accesses" in l:
            x = l[l.find("dl1.accesses")+12:]
            w = x[:x.find("#")]
            w = w.lstrip()
            datos.write("," + w )
    if "dl1.hits" in l:
            x = l[l.find("dl1.hits")+8:]
            w = x[:x.find("#")]
            w = w.lstrip()
            datos.write("," + w )
    if "dl1.misses" in l:
            x = l[l.find("dl1.misses")+10:]
            w = x[:x.find("#")]
            w = w.lstrip()
            datos.write(","+ w)
    if "dl1.replacements" in l:
            x = l[l.find("dl1.replacements")+16:]
            w = x[:x.find("#")]
            w = w.lstrip()
            datos.write(","+ w)
    if "dl1.writebacks" in l:
            x = l[l.find("dl1.writebacks")+14:]
            w = x[:x.find("#")]
            w = w.lstrip()
            datos.write(","+ w)
    if "ul2.accesses" in l:
            x = l[l.find("ul2.accesses")+12:]
            w = x[:x.find("#")]
            w = w.lstrip()
            datos.write("," + w )
    if "ul2.hits" in l:
            x = l[l.find("ul2.hits")+8:]
            w = x[:x.find("#")]
            #w = w.lstrip()
            datos.write("," + w)
    if "ul2.misses" in l:
            x = l[l.find("ul2.misses")+10:]
            w = x[:x.find("#")]
            w = w.rstrip()
            datos.write("," + w)
    if "ul2.replacements" in l:
            x = l[l.find("ul2.replacements")+16:]
            w = x[:x.find("#")]
            w = w.lstrip()
            datos.write(","+ w)
    if "ul2.writebacks" in l:
            x = l[l.find("ul2.writebacks")+14:]
            w = x[:x.find("#")]
            w = w.lstrip()
            datos.write(","+ w)
    if "ul3.accesses" in l:
            x = l[l.find("ul3.accesses")+12:]
            w = x[:x.find("#")]
            w = w.lstrip()
            datos.write("," + w )
    if "ul3.hits" in l:
            x = l[l.find("ul3.hits")+8:]
            w = x[:x.find("#")]
            #w = w.lstrip()
            datos.write("," + w)
    if "ul3.misses" in l:
            x = l[l.find("ul3.misses")+10:]
            w = x[:x.find("#")]
            w = w.rstrip()
            datos.write("," + w)
    if "ul3.replacements" in l:
            x = l[l.find("ul3.replacements")+16:]
            w = x[:x.find("#")]
            w = w.lstrip()
            datos.write(","+ w)
    if "ul3.writebacks" in l:
            x = l[l.find("ul3.writebacks")+14:]
            w = x[:x.find("#")]
            w = w.lstrip()
            datos.write(","+ w)
    if "dtlb.accesses" in l:
            x = l[l.find("dtlb.accesses")+13:]
            w = x[:x.find("#")]
            w = w.lstrip()
            datos.write("," + w )
```

```
if "dtlb.hits" in l:
        x = l[l.find("dtlb.hits")+9:]
        w = x[:x.find("#")]
        #w = w.lstrip()
        datos.write("," + w)
if "dtlb.misses" in l:
        x = l[l.find("dtlb.misses")+11:]
        w = x[:x.find("#")]
        w = w.rstrip()
        datos.write("," + w)
if "dtlb.replacements" in l:
        x = l[l.find("dtlb.replacements")+17:]
        w = x[:x.find("#")]
        w = w.lstrip()
        datos.write(","+ w)
if "aciertosdc" in l:
        x = l[l.find("aciertosdc")+10:]
        w = x[:x.find("#")]
        w = w.rstrip()
        datos.write("," + w)
if "fallosdc" in l:
        x = l[l.find("fallosdc")+8:]
        w = x[:x.find("#")]
        w = w.lstrip()
        datos.write("," + w)
if "total_accesos_dc" in l:
        x = l[l.find("total_accesos_dc")+16:]
        w = x[:x.find("#")]
        w = w.lstrip()
        datos.write("," + w)
if "aciertosac" in l:
        x = l[l.find("aciertosac")+10:]
        w = x[:x.find("#")]
        w = w.rstrip()
        datos.write("," + w)
if "fallosac" in l:
        x = l[l.find("fallosac")+8:]
        w = x[:x.find("#")]
        w = w.lstrip()
        datos.write("," + w)
if "total_accesos_ac" in l:
        x = l[l.find("total_accesos_ac")+16:]
        w = x[:x.find("#")]
        w = w.lstrip()
        datos.write("," + w)
if "aciertos_dc_loads" in l:
        x = l[l.find("aciertos_dc_loads")+17:]
        w = x[:x.find("#")]
        w = w.rstrip()
        datos.write("," + w)
if "fallos_dc_loads" in l:
        x = l[l.find("fallos_dc_loads")+15:]
        w = x[:x.find("#")]
        w = w.lstrip()
        datos.write("," + w)
if "total_dc_loads" in l:
        x = l[l.find("total_dc_loads")+14:]
        w = x[:x.find("#")]
        w = w.lstrip()
        datos.write("," + w)
if "aciertos_ac_loads" in l:
        x = l[l.find("aciertos_ac_loads")+17:]
        w = x[:x.find("#")]
        w = w.lstrip()
        datos.write("," + w)
if "fallos_ac_loads" in l:
        x = l[l.find("fallos_ac_loads")+15:]
        w = x[:x.find("#")]
        w = w.lstrip()
        datos.write("," + w)
if "total_ac_loads" in l:
        x = l[l.find("total_ac_loads")+14:]
        w = x[:x.find("#")]
        w = w.lstrip()
        datos.write("," + w)
if "aciertos_dc_stores" in l:
        x = l[l.find("aciertos_dc_stores")+18:]
        w = x[:x.find("#")]
        w = w.lstrip()
        datos.write("," + w)
if "fallos_dc_stores" in l:
        x = l[l.find("fallos_dc_stores")+16:]
        w = x[:x.find("#")]
        w = w.lstrip()
        datos.write("," + w)
if "total_dc_stores" in l:
        x = l[l.find("total_dc_stores")+15:]
```

```
                                w = x[:x.find("#")]
                                w = w.lstrip()
                                datos.write("," + w)
                        if "aciertos_ac_stores" in l:
                                x = l[l.find("aciertos_ac_stores")+18:]
                                w = x[:x.find("#")]
                                w = w.lstrip()
                                datos.write("," + w)
                        if "fallos_ac_stores" in l:
                                x = l[l.find("fallos_ac_stores")+16:]
                                w = x[:x.find("#")]
                                w = w.lstrip()
                                datos.write("," + w)
                        if "total_ac_stores" in l:
                                x = l[l.find("total_ac_stores")+15:]
                                w = x[:x.find("#")]
                                w = w.lstrip()
                                datos.write("," + w)
                        if "sim_total_issued_loads" in l:
                                x = l[l.find("sim_total_issued_loads")+22:]
                                w = x[:x.find("#")]
                                w = w.lstrip()
                                datos.write("," + w)
                        if "sim_total_issued_stores" in l:
                                x = l[l.find("sim_total_issued_stores")+23:]
                                w = x[:x.find("#")]
                                w = w.lstrip()
                                datos.write("," + w)
                        if "sim_total_commited_loads" in l:
                                x = l[l.find("sim_total_commited_loads")+24:]
                                w = x[:x.find("#")]
                                w = w.lstrip()
                                datos.write("," + w)
                        if "sim_total_commited_stores" in l:
                                x = l[l.find("sim_total_commited_stores")+25:]
                                w = x[:x.find("#")]
                                w = w.lstrip()
                                datos.write("," + w)
        fh.close()
# Fin def leerFicheroEnteros(lista)

# Comienzo
lista = leerarchivos()
leerFicheroEnteros(lista)
# Fin
```

Apéndice E

Resultados, costes, métricas y análisis previo.

Este apéndice está dedicado a mostrar los costes de energía relacionados con el Proyecto. También se explica la fase de análisis previo o profiling que es indispensable para este Proyecto (ver seccion E.2). Se muestran también las métricas que hemos utilizado para valorar los resultados obtenidos y finalmente se muestran los resultados mediante las tablas que hemos obtenidos así como sus gráficas.

E.1. Costes energéticos

Para obtener los costes de energía que hemos usado en este Proyecto, hemos usado la herramiente Cacti en su versión 6.5 [12]. Esta herramienta sirve para modelar memorias caches. Tiene varias versiones y la elegida para mi trabajo ha sido la última versión que se usa mediante compilación y ejecución en la línea de comandos del shell.

El gasto de energía que experimenta un procesador de altas prestaciones está ligado al uso de la jerarquía de memoria. Por una parte está el consumo estático que es aquel que tienen los transistores que forman los módulos de la memoria cuando no hay transiciones, es decir, cuando su salida no cambia (consumo estático de energía). Cuando hay transiciones en estos transistores, hay consumo de energía y se conoce como consumo dinámico de energía.

La tabla E.1 nos muestra los costes estáticos, que son aquellos que están relacionados con la falta de transición en los transistores que forman los módulos de la jerarquía de memoria. En la tabla E.2 se muestran los valores de los costes de energía dinámicos, que son los que están relacionados con los cambios de estado de los transistores, es decir, cuando hay transiciones hay consumo de energía dinámica.

Para la realización de algunos cálculos hemos supuesto que la frecuencia del procesador es de 2.5GHz.

	Coste
Memoria L1	0.047432
Memoria L2	0.150232
Memoria L3	2.457176
Memoria principal	0.866304
Memoria ACDC	0.004

Tabla E.1: Coste de energía estática para las distintas memorias expresado en nJ.

	Coste leer	Coste escribir
Memoria L1	0.0892139	0.0678675
Memoria L2	0.226472	0.21833
Memoria L3	0.916666	0.964853
Memoria Principal	0.672907	0.693852
Memoria AC	0.0098815628	0.0098815628
Memoria DC	0.0098815628	0.0098815628

Tabla E.2: Coste de energía dinámica para las distintas memorias expresado en nJ.

```
Coste energético =
                    total stores * coste escribir l1          +
                    reemplazos l1 * coste escribir l1         +
                    reemplazos l2 * coste leer l2             +
                    escrituras l2 * coste escribir l2         +
                    reemplazos l2 * coste escribir l2         +
                    reemplazos l3 * coste leer l3             +
                    escrituras l3 * coste escribir l3         +
                    reemplazos l3 * coste escribir l3         +
                    reemplazos MM * coste leer MM             +
                    escrituras MM     * costes escribir MM    +
                    total accesos DC * coste leer DC          +
                    total accesos AC * coste leer AC          +
                    aciertos AC * coste escribir AC           +
                    fallos DC * coste leer l1
```

Figura E.1: Fórmula que muestra cómo se obtiene el coste de energía dinámica en la organización secuencial.

```
Coste energético =
                    total load * coste leer l1                +
                    total stores * coste escribir l1          +
                    reemplazos l1 * coste escribir l1         +
                    reemplazos l2 * coste leer l2             +
                    escrituras l2 * coste escribir l2         +
                    reemplazos l2 * coste escribir l2         +
                    reemplazos l3 * coste leer l3             +
                    escrituras l3 * coste escribir l3         +
                    reemplazos l3 * coste escribir l3         +
                    reemplazos MM * coste leer MM             +
                    escrituras MM     * costes escribir MM    +
                    total accesos DC * coste leer DC          +
                    total accesos AC * coste leer AC          +
                    aciertos AC * coste escribir AC
```

Figura E.2: Fórmula que muestra cómo se obtiene el coste de energía dinámica en la organización paralela.

Para la obtención de los consumos de energía dinámica para el caso de la organización secuencial, se ha aplicado la fórmula de la figura E.1. Para obtener el coste energético de la organización paralela, se ha aplicado la fórmula de la figura E.2. La diferencia entre ambas radica en que en el coste de energía en la organización secuencial, los accesos al módulo de memoria L1 de la jerarquía de memoria son los fallos en la DC.

E.2. Análisis

En esta sección se explica cómo se han obtenido las instrucciones que se cargan en la AC. Este proceso es el que hemos llamado "la fase de análisis previo" o profiling y consiste en buscar aquellas instrucciones que aprovechan más el reuso de los datos dentro del programa (SPEC CPU 2006 [2]). Estas instrucciones son las que más veces acceden a los datos, tanto ellas, como amigas,

entendiendo amigas como aquellas instrucciones que acceden al mismo dato (ver ejemplo C.1 y [1]). Una vez que hemos obtenidos las instrucciones que maś aprovechan el reuso, las ordenamos por el número de aciertos al dato al que acceden y así se podrán cargar en la AC cuando se lance la ejecución de la simulación usando la ACDC. La AC tendrá 16 líneas, que cada línea corresponde con un PC.

Los programas de prueba [2] están preparados para simulaciones con 100.000.000 de instrucciones de fastfwd, es decir, para el calentamiento de las caches y 100.000.000 de instrucciones para la simulación en sí.

Inicialmente, este análisis previo se ha hecho sobre el total de las instrucciones, obteniendo un solo fichero con los PC's que más reuso aprovechan. Posteriormente y para hacer otras pruebas, hicimos un análisis de 100 ficheros con 1.000.000 de instrucciones cada uno, para intentar buscar con más profundidad el reuso. Es decir, dividimos el programa en tramos más pequeños y así intentar localizar las partes del programa donde más reuso hay.

Los resultados del análisis de la suite enteros y de la suite float se pueden ver, tanto para 1 fichero como para 100 en la sección E.8.3.

E.3. Métricas

Las métricas que se han usado para valorar la memoria ACDC [1] en SPEC CPU 2006 [2] son las siguientes:

- IPC, instrucciones por ciclo (rendimiento).
- Energy_Delay. Este indicador tiene en cuenta el tiempo de ejecución y el gasto de energía. Con ello tenemos una métrica más real del ratio rendimiento-consumo.
- Aciertos obtenidos y aciertos esperados en DC. Los aciertos obtenidos son aquellos que han acertado en la memoria DC cuando estamos usando o la organización secuencial (ver sección 2.3.1) o la organización paralela (ver sección 2.3.2) y los aciertos esperados son los que se han obtenido de la fase de análisis previo (ver sección E.2).

E.4. Graficas suite enteros y suite float

En esta sección se muestran las gráficas y las tablas que hemos obtenido con la ejecución de las simulaciones en este Proyecto. En primer lugar se mostrarán las gráficas tanto para la suite enteros como para la suite float. Se comienza por las gráficas de rendimiento y posteriormente siguen las gráficas del consumo (estático, dinámico, total y con la métrica Energy_Delay). Primero se muestran las gráficas con sus valores y luego en porcentajes.

Después de las gráficas se muestran las tablas correspondientes a los datos extraídos de las simulaciones. En primer lugar se muestran los datos correspondientes a la suite enteros en el modo solo. Le siguen el modo secuencial y el modo paralelo. Los datos que se muestran hacen referencia al nombre de benchmark, al modo de ejecución (solo, paralelo o secuencial), las instrucciones que se han lanzado (IQ), las instrucciones que han hecho el commit (CT), el tiempo en ciclos de la simulación (sim_cycle), IPC, accesos a los distintos módulos de memoria (contando los aciertos y los fallos) y accesos a la DC y AC (con fallos y aciertos).

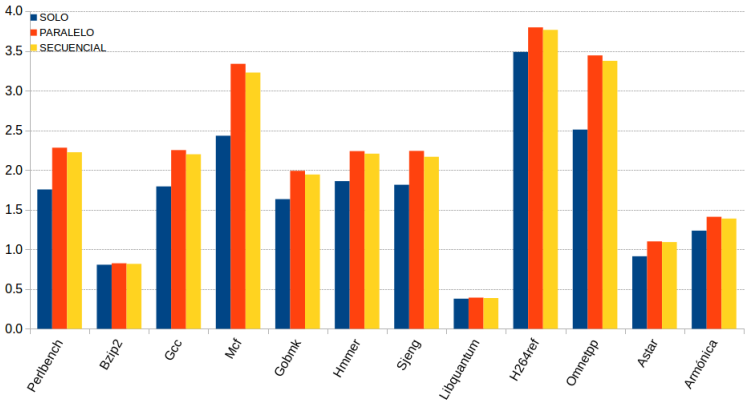


FIGURA E.3: Valores de IPC para la suite enteros.

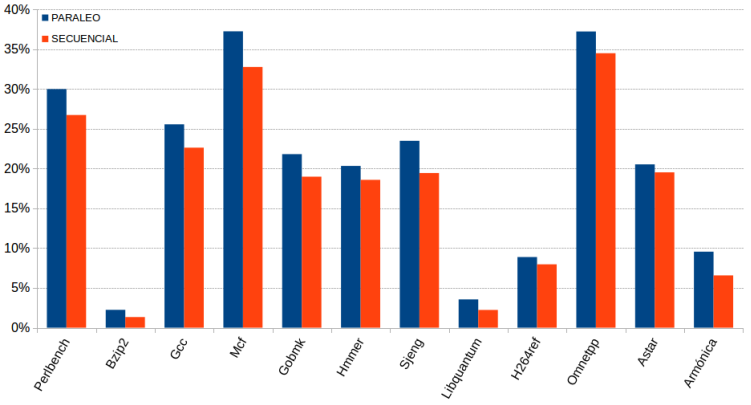


FIGURA E.4: Valores de IPC en porcentajes para la suite enteros.

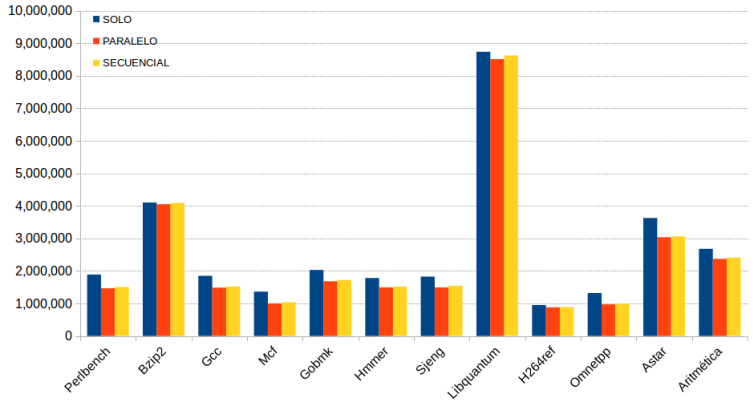


FIGURA E.5: Consumo estático para la suite enteros expresado en nJ.

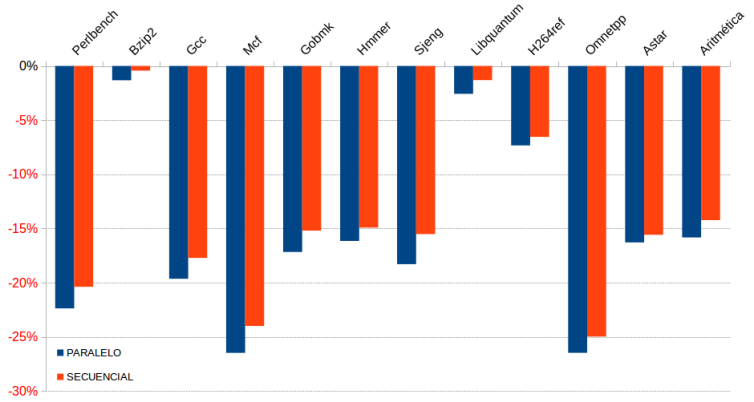


FIGURA E.6: Porcentaje de consumo estático para la suite enteros.

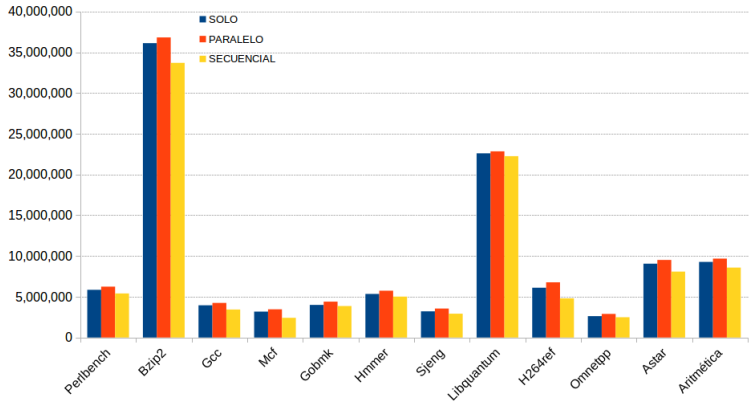


FIGURA E.7: Consumo de energía dinámica expresado en nJ para la suite enteros.

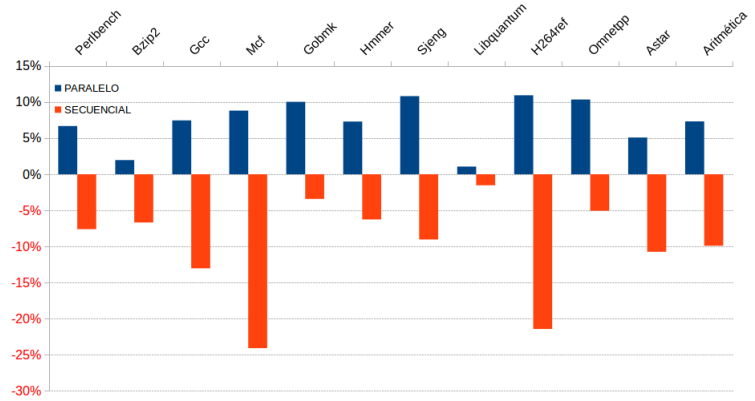


FIGURA E.8: Porcentaje de consumo dinámico para la suite enteros.

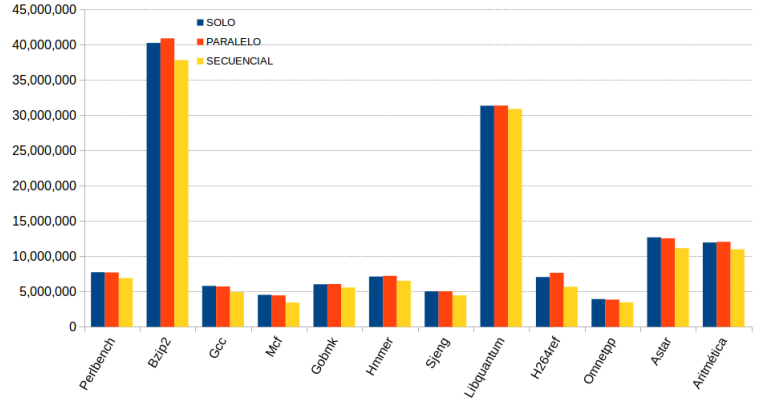


FIGURA E.9: Consumo total expresado en nJ para la suite enteros.

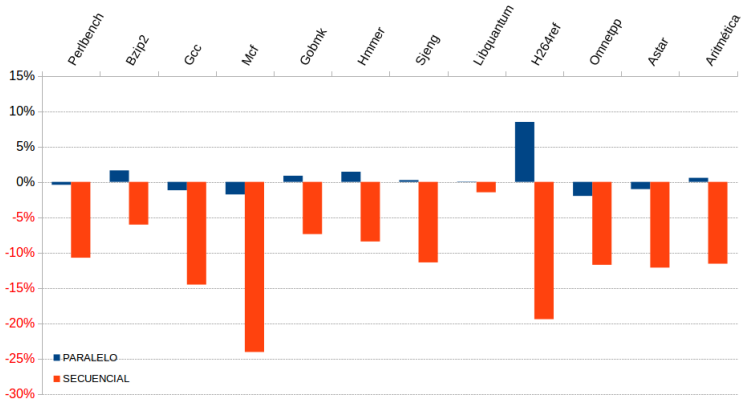


FIGURA E.10: Consumo total en porcentajes para la suite enteros.

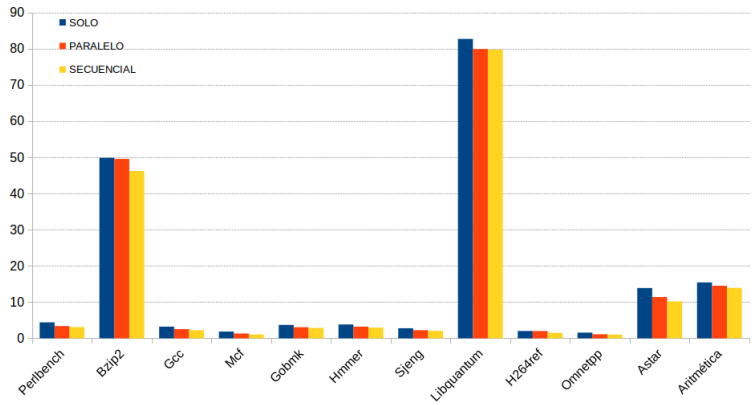


FIGURA E.11: Indicador Energy_Delay del consumo de energía para la suite enteros.

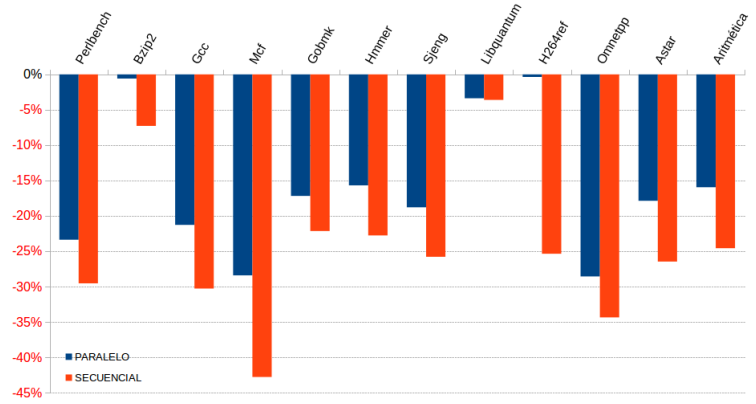


FIGURA E.12: Indicador del consumo Energy_Delay expresado en porcentajes de la suite enteros.

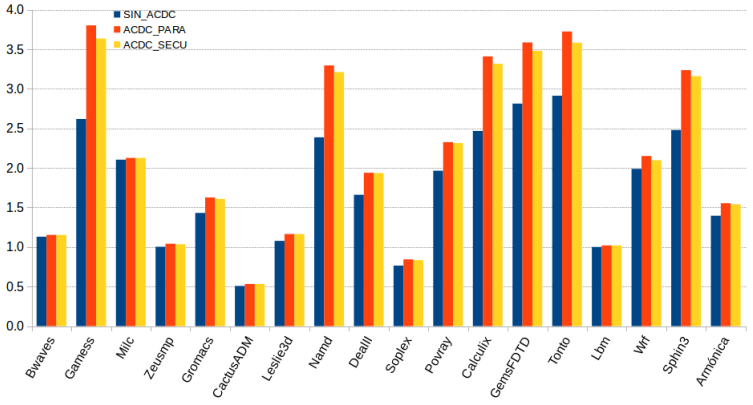


FIGURA E.13: Valores de IPC para la suite float.

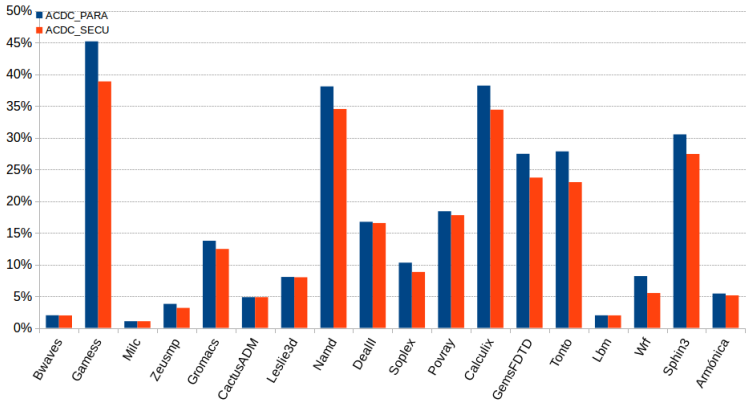


FIGURA E.14: Valores de IPC en porcentajes para la suite float.

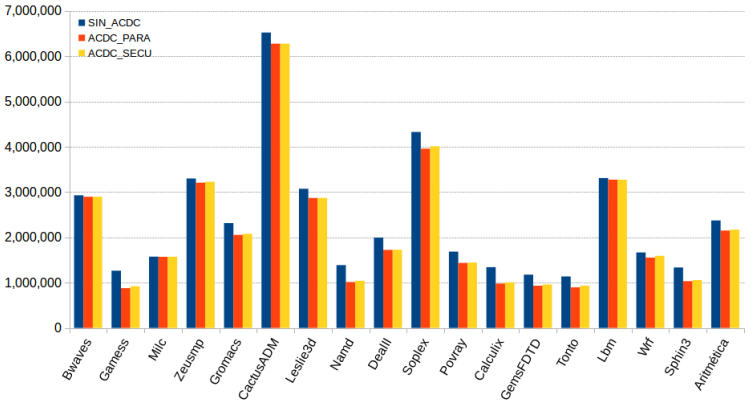


FIGURA E.15: Consumo estático para la suite float expresado en nJ.

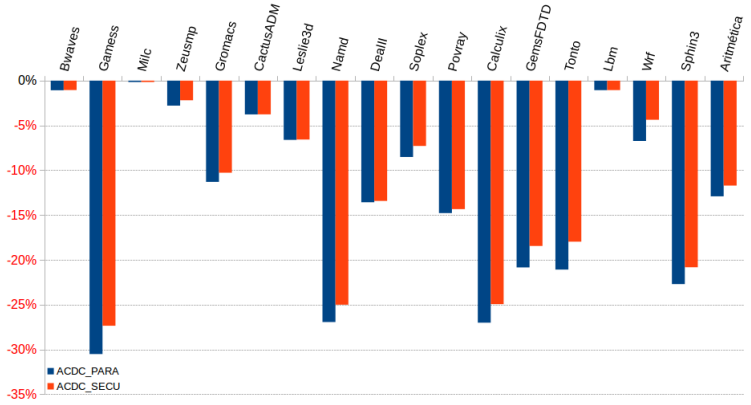


FIGURA E.16: Porcentaje de consumo estático de la suite float.

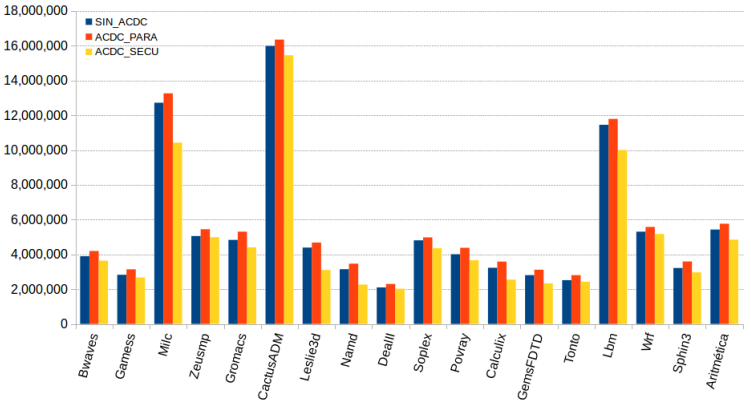


FIGURA E.17: Consumo de energía dinámica expresado en nJ para la suite float.

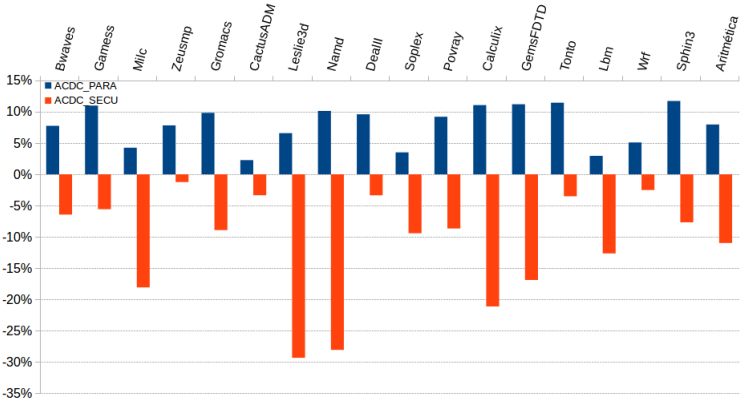


FIGURA E.18: Porcentaje de consumo dinámico para la suite float.

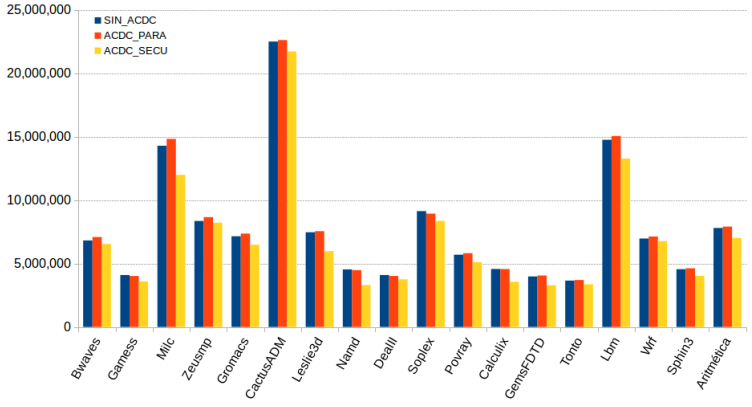


FIGURA E.19: Consumo total expresado en nJ para la suite float.

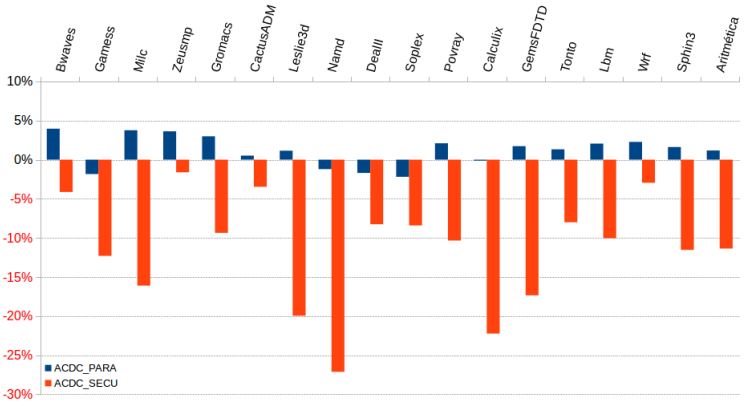


FIGURA E.20: Consumo total en porcentajes de la suite float.

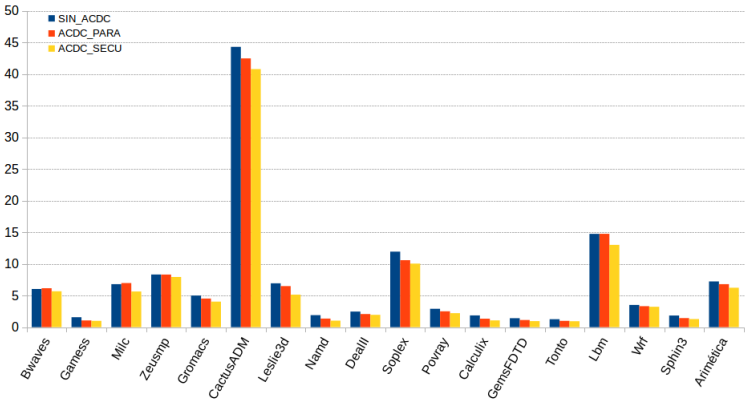


FIGURA E.21: Indicador Energy_Delay del consumo de energía para la suite float.

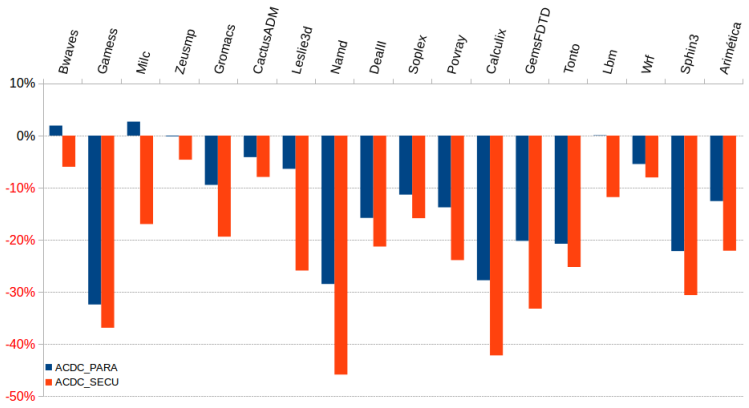


FIGURA E.22: Indicador del consumo Energy_Delay expresado en porcentajes de la suite float.

E.5. Resultados suite enteros

E.5.1. Resultados sin el uso de la memoria ACDC, suite enteros.

Tabla E.3: Resultados para la suite Enteros sin memoria ACDC.

Nombre	total_issued_ld	total_issued_st	total_commited_ld
Perlbench	36,533,844	14,727,176	32,075,587
Bzip2	58,695,597	17,391,281	54,347,500
Gcc	33,181,516	10,982,480	28,515,273
Mcf	27,222,739	9,539,306	25,780,211
Gobmk	33,864,570	11,957,115	27,076,634
Hmmer	39,993,654	9,008,185	32,198,719
Sjeng	30,046,310	6,864,883	24,750,238
Libquantum	29,278,586	6,977,940	18,570,642
H264ref	49,365,710	16,326,366	48,203,947
Omnetpp	24,200,813	6,734,346	21,880,320
Astar	46,148,122	7,263,300	23,853,370

Nombre	total_commited_st	sim_num_refs	sim_num_ld	sim_num_st
Perlbench	12,600,176	44,675,773	32,075,593	12,600,180
Bzip2	17,391,239	71,738,830	54,347,567	17,391,263
Gcc	9,681,488	38,196,768	28,515,279	9,681,489
Mcf	9,083,897	34,864,123	25,780,220	9,083,903
Gobmk	10,450,938	37,527,589	27,076,645	10,450,944
Hmmer	7,310,067	39,508,799	32,198,731	7,310,068
Sjeng	6,208,686	30,958,932	24,750,246	6,208,686
Libquantum	6,977,899	25,548,548	18,570,646	6,977,902
H264ref	16,097,809	64,301,844	48,204,012	16,097,832
Omnetpp	6,468,517	28,348,853	21,880,328	6,468,525
Astar	3,518,020	27,371,393	23,853,373	3,518,020

Nombre	elapsed_time	total_refs	total_ld	total_st	simcycle
Perlbench	246	58,436,869	42,079,106	16,357,763	56,977,368
Bzip2	510	71,739,160	54,347,830	17,391,330	123,915,380
Gcc	232	49,511,424	36,890,640	12,620,784	55,773,994
Mcf	200	37,732,986	28,014,650	9,718,336	41,129,611
Gobmk	252	55,522,511	41,036,898	14,485,613	61,219,900
Hmmer	238	57,978,208	46,057,066	11,921,142	53,757,656
Sjeng	227	43,374,779	35,552,229	7,822,550	55,118,350
Libquantum	612	25,548,812	18,570,829	6,977,983	263,817,483
H264ref	202	68,216,509	51,175,055	17,041,454	28,678,481
Omnetpp	182	33,686,705	26,522,056	7,164,649	39,851,402
Astar	400	70,226,593	61,002,063	9,224,530	109,455,503

Nombre	IPC	dl1.accesses	dl1.hits	dl1.misses	dl1.replace
Perlbench	1.755083	44,913,372	44,330,831	582,541	582,541
Bzip2	0.807002	71,738,806	63,043,175	8,695,631	8,695,631
Gcc	1.792950	38,051,871	37,895,190	156,681	156,681
Mcf	2.431338	33,810,379	33,768,043	42,336	42,336
Gobmk	1.633456	40,496,916	40,388,830	108,086	108,086
Hmmer	1.860200	39,965,790	39,459,110	506,680	506,680
Sjeng	1.814278	33,509,077	33,450,437	58,640	58,640
Libquantum	0.379050	25,548,565	20,033,655	5,514,910	5,514,910
H264ref	3.486935	64,479,040	64,369,908	109,132	109,033
Omnetpp	2.509322	26,210,984	26,128,443	82,541	82,541
Astar	0.913613	44,269,220	42,697,704	1,571,516	1,571,516

Nombre	dl1.writebacks	ul2.accesses	ul2.hits	ul2.misses
Perlbench	127,159	948,537	874,675	73,862
Bzip2	4,347,812	13,043,447	10,869,513	2,173,934
Gcc	119,556	668,634	618,176	50,458
Mcf	42,335	84,674	66,827	17,847
Gobmk	74,553	380,402	339,289	41,113
Hmmer	443,477	950,172	838,567	111,605
Sjeng	40,998	109,565	79,900	29,665
Libquantum	4,878,548	10,393,631	9,014,799	1,378,832
H264ref	65,451	186,374	185,078	1,296
Omnetpp	45,358	161,642	126,296	35,346
Astar	631,882	2,203,399	2,005,132	198,267

Nombre	ul2.replacements	ul2.writebacks	ul3.accesses	ul3.hits
Perlbench	582,541	16,396	90,258	83,769
Bzip2	8,695,631	1,087,032	3,260,966	1,087,032
Gcc	156,681	26,154	76,612	52,178
Mcf	42,336	15,895	33,742	23,758
obmk	108,086	21,535	62,648	46,459
Hmmer	506,680	105,464	217,069	216,339
Sjeng	58,640	17,241	46,906	26,110
Libquantum	5,514,910	1,218,118	2,596,950	1,218,118
H264ref	109,033	248	1,544	1,236
Omnetpp	82,541	13,222	48,568	38,161
Astar	1,571,516	110,218	308,485	308,424

Nombre	ul3.misses	ul3.replacements	ul3.writebacks
Perlbench	6,489	582,541	4,805
Bzip2	2,173,934	8,695,631	1,086,589
Gcc	24,434	156,681	20,668
Mcf	9,984	42,336	8,982
Gobmk	16,189	108,086	15,454
Hmmer	730	506,680	0
Sjeng	20,796	58,640	13,974
Libquantum	1,378,832	5,514,910	1,203,782
H264ref	308	109,033	0
Omnetpp	10,407	82,541	5,062
Astar	61	1,571,516	0

E.5.2. Resultados con la organización secuencial, suite enteros.

Tabla E.4: Resultados para la suite Enteros con la organización Secuencial.

Nombre	Modo	total_issued_ld	total_issued_st
Perlbench	Secuencial	36,063,440	14,621,797
Bzip2	Secuencial	58,695,561	17,391,281
Gcc	Secuencial	32,166,608	10,761,066
Mcf	Secuencial	26,795,751	9,475,726
Gobmk	Secuencial	32,944,010	11,772,486
Hmmer	Secuencial	39,464,351	9,306,874
Sjeng	Secuencial	29,085,729	6,745,863
Libquantum	Secuencial	29,278,543	6,977,939
H264ref	Secuencial	49,339,898	16,392,440
Omnetpp	Secuencial	23,957,653	6,730,453
Astar	Secuencial	43,981,094	7,099,406

Nombre	total_commited_ld	total_commited_st	num_refs
Perlbench	32,075,588	12,600,176	44,675,773
Bzip2	54,347,500	17,391,239	71,738,830
Gcc	28,515,273	9,681,488	38,196,768
Mcf	25,780,211	9,083,897	34,864,123
Gobmk	27,076,634	10,450,938	37,527,589
Hmmer	32,198,719	7,310,067	39,508,799
Sjeng	24,750,238	6,208,686	30,958,933
Libquantum	18,570,642	6,977,899	25,548,547
H264ref	48,203,947	16,097,809	64,301,844
Omnetpp	21,880,320	6,468,517	28,348,853
Astar	23,853,369	3,518,019	27,371,393

Nombre	num_ld	num_st	elapsed_time
Perlbench	32,075,593	12,600,180	253
Bzip2	54,347,567	17,391,263	586
Gcc	28,515,279	9,681,489	236
Mcf	25,780,220	9,083,903	204
Gobmk	27,076,645	10,450,944	243
Hmmer	32,198,731	7,310,068	230
Sjeng	24,750,246	6,208,687	217
Libquantum	18,570,646	6,977,901	622
H264ref	48,204,012	16,097,832	216
Omnetpp	21,880,328	6,468,525	171
Astar	23,853,373	3,518,020	372

Nombre	total_refs	total_ld	total_st	simcycle
Perlbench	57,423,642	41,380,671	16,042,971	44,962,448
Bzip2	71,739,022	54,347,714	17,391,308	122,284,866
Gcc	48,823,287	36,364,761	12,458,526	45,486,503
Mcf	37,554,117	27,813,132	9,740,985	30,982,167
Gobmk	54,804,257	40,495,075	14,309,182	51,455,987
Hmmer	58,555,881	46,615,487	11,940,394	45,334,142
Sjeng	43,042,547	35,258,506	7,784,041	46,151,444
Libquantum	25,548,806	18,570,827	6,977,979	258,063,091
H264ref	68,342,710	51,252,008	17,090,702	26,563,269
Omnetpp	33,467,038	26,342,515	7,124,523	29,632,289
Astar	69,054,880	59,943,456	9,111,424	91,578,623

Nombre	IPC	dl1.access	dl1.hits	dl1.misses	dl1.replaces
Perlbench	2.224078	37,774,068	37,189,952	584,116	584,116
Bzip2	0.817763	39,130,447	30,434,813	8,695,634	8,695,634
Gcc	2.198454	30,437,556	30,279,922	157,634	157,634
Mcf	3.227663	23,184,320	23,141,998	42,322	42,322
Gobmk	1.943408	37,497,456	37,389,143	108,313	108,313
Hmmer	2.205843	34,866,467	34,359,770	506,697	506,697
Sjeng	2.166780	29,136,566	29,077,430	59,136	59,136
Libquantum	0.387502	20,474,671	14,959,760	5,514,911	5,514,911
H264ref	3.764597	64,364,605	64,253,885	110,720	110,621
Omnetpp	3.374697	23,738,468	23,655,316	83,152	83,152
Astar	1.091958	31,588,356	30,016,978	1,571,378	1,571,378

Nombre	dl1.wb	ul2.access	ul2.hits	ul2.misses	ul2.replaces
Perlbench	127,698	944,531	871,195	73,336	584,116
Bzip2	4,347,813	13,043,451	10,869,517	2,173,934	8,695,634
Gcc	119,630	670,148	619,588	50,560	157,634
Mcf	42,315	84,639	66,773	17,866	42,322
Gobmk	74,567	378,235	337,540	40,695	108,313
Hmmer	443,501	950,211	838,580	111,631	506,697
Sjeng	41,259	110,257	80,590	29,667	59,136
Libquantum	4,878,547	10,393,629	9,014,799	1,378,830	5,514,911
H264ref	66,185	188,690	187,368	1,322	110,621
Omnetpp	45,466	161,856	126,468	35,388	83,152
Astar	631,815	2,203,194	2,004,912	198,282	1,571,378

Nombre	ul2.wb	ul3.access	ul3.hits	ul3.misses	ul3.replaces
Perlbench	16,279	89,615	83,898	5,717	584,116
Bzip2	1,087,032	3,260,966	1,087,032	2,173,934	8,695,634
Gcc	26,161	76,721	52,867	23,854	157,634
Mcf	15,895	33,761	24,707	9,054	42,322
Gobmk	21,466	62,161	48,398	13,763	108,313
Hmmer	105,469	217,100	216,369	731	506,697
Sjeng	17,247	46,914	26,760	20,154	59,136
Libquantum	1,218,117	2,596,947	1,218,117	1,378,830	5,514,911
H264ref	248	1,570	1,255	315	110,621
Omnetpp	13,241	48,629	38,299	10,330	83,152
Astar	110,237	308,519	308,458	61	1,571,378

Nombre	ul3.wb	dtlb.access	dtlb.hits	dtlb.misses	hitsDC
Perlbench	5,629	12,600,554	12,599,467	1,087	18,876,151
Bzip2	1,086,589	17,391,703	17,374,713	16,990	49,999,619
Gcc	22,301	9,681,790	9,681,402	388	16,275,515
Mcf	105	9,084,408	9,079,116	5,292	19,084,513
Gobmk	13,698	10,451,222	10,444,460	6,762	13,301,315
Hmmer	0	7,310,453	7,310,443	10	12,132,845
Sjeng	6,724	6,208,972	6,194,034	14,938	10,510,199
Libquantum	1,187,397	6,978,400	6,959,319	19,081	12,051,828
H264ref	0	16,098,081	16,098,079	2	16,420,582
Omnetpp	0	6,468,856	6,468,671	185	9,073,932
Astar	0	3,518,400	3,518,392	8	16,143,500

Nombre	missDC	totAccesosDC	hitsAC	missAC	totAccesosAC
Perlbench	25,182,954	44,059,105	11,939	25,171,015	25,182,954
Bzip2	21,739,208	71,738,827	8,695,565	13,043,643	21,739,208
Gcc	20,820,881	37,096,396	76,639	20,744,242	20,820,881
Mcf	14,100,423	33,184,936	14	14,100,409	14,100,423
Gobmk	27,055,545	40,356,860	107,766	26,947,779	27,055,545
Hmmer	27,556,412	39,689,257	13,324,869	14,231,543	27,556,412
Sjeng	22,927,979	33,438,178	11,334	22,916,645	22,927,979
Libquantum	13,496,775	25,548,603	3,365,643	10,131,132	13,496,775
H264ref	48,267,621	64,688,203	4,864,193	43,403,428	48,267,621
Omnetpp	17,271,352	26,345,284	1,272,405	15,998,947	17,271,352
Astar	28,070,337	44,213,837	8,084,037	19,986,300	28,070,337

Nombre	hitsDC_ld	missDC_ld	totDC_ld	hitsAC_ld	missAC_ld
Perlbench	6,275,975	25,182,954	31,458,929	11,939	25,171,015
Bzip2	32,608,380	21,739,208	54,347,588	8,695,565	13,043,643
Gcc	6,594,027	20,820,881	27,414,908	76,639	20,744,242
Mcf	10,000,616	14,100,423	24,101,039	14	14,100,409
Gobmk	2,850,377	27,055,545	29,905,922	107,766	26,947,779
Hmmer	4,822,778	27,556,412	32,379,190	13,324,869	14,231,543
Sjeng	4,301,513	22,927,979	27,229,492	11,334	22,916,645
Libquantum	5,073,929	13,496,775	18,570,704	3,365,643	10,131,132
H264ref	322,773	48,267,621	48,590,394	4,864,193	43,403,428
Omnetpp	2,605,415	17,271,352	19,876,767	1,272,405	15,998,947
Astar	12,625,481	28,070,337	40,695,818	8,084,037	19,986,300

Nombre	**totAC_ld**	**hitsDC_st**	**missDC_st**	**totDC_st**
Perlbench	25,182,954	12,600,176	0	12,600,176
Bzip2	21,739,208	17,391,239	0	17,391,239
Gcc	20,820,881	9,681,488	0	9,681,488
Mcf	14,100,423	9,083,897	0	9,083,897
Gobmk	27,055,545	10,450,938	0	10,450,938
Hmmer	27,556,412	7,310,067	0	7,310,067
Sjeng	22,927,979	6,208,686	0	6,208,686
Libquantum	13,496,775	6,977,899	0	6,977,899
H264ref	48,267,621	16,097,809	0	16,097,809
Omnetpp	17,271,352	6,468,517	0	6,468,517
Astar	28,070,337	3,518,019	0	3,518,019

E.5.3. Resultados con la organización paralela, suite enteros.

Tabla E.5: Resultados para la suite Enteros con la organización paralela.

Nombre	Organización	total_issued_ld	total_issued_st
Perlbench	Paralela	36,138,911	14,649,682
Bzip2	Paralela	58,695,571	17,391,278
Gcc	Paralela	32,362,570	10,688,680
Mcf	Paralela	26,880,949	9,474,315
Gobmk	Paralela	33,091,062	11,836,970
Hmmer	Paralela	39,002,977	9,201,474
Sjeng	Paralela	29,268,328	6,760,013
Libquantum	Paralela	29,278,541	6,977,941
H264ref	Paralela	49,406,362	16,396,949
Omnetpp	Paralela	24,055,791	6,741,887
Astar	Paralela	44,406,262	7,148,681

Nombre	total_commited_ld	total_commited_st	num_refs
Perlbench	32,075,588	12,600,176	44,675,773
Bzip2	54,347,500	17,391,239	71,738,830
Gcc	28,515,273	9,681,488	38,196,768
Mcf	25,780,211	9,083,896	34,864,125
Gobmk	27,076,634	10,450,937	37,527,588
Hmmer	32,198,719	7,310,067	39,508,800
Sjeng	24,750,238	6,208,686	30,958,933
Libquantum	18,570,642	6,977,899	25,548,548
H264ref	48,203,947	16,097,809	64,301,844
Omnetpp	21,880,320	6,468,517	28,348,853
Astar	23,853,369	3,518,019	27,371,393

Nombre	num_ld	num_st	elapsed_time
Perlbench	32,075,593	12,600,180	235
Bzip2	54,347,567	17,391,263	501
Gcc	28,515,279	9,681,489	208
Mcf	25,780,222	9,083,903	170
Gobmk	27,076,644	10,450,944	230
Hmmer	32,198,731	7,310,069	218
Sjeng	24,750,246	6,208,687	204
Libquantum	18,570,646	6,977,902	596
H264ref	48,204,012	16,097,832	207
Omnetpp	21,880,328	6,468,525	162
Astar	23,853,373	3,518,020	358

Nombre	total_refs	total_ld	total_st	simcycle
Perlbench	58,162,860	41,928,160	16,234,700	43,836,085
Bzip2	71,739,050	54,347,732	17,391,318	121,197,901
Gcc	49,120,300	36,585,172	12,535,128	44,424,141
Mcf	37,594,200	27,866,769	9,727,431	29,969,716
Gobmk	55,344,803	40,907,674	14,437,129	50,258,443
Hmmer	59,060,906	47,091,880	11,969,026	44,676,855
Sjeng	43,517,279	35,675,719	7,841,560	44,637,907
Libquantum	25,548,810	18,570,827	6,977,983	254,755,438
H264ref	68,606,115	51,474,836	17,131,279	26,340,662
Omnetpp	33,682,897	26,514,237	7,168,660	29,041,978
Astar	69,914,134	60,713,140	9,200,994	90,815,335

Nombre	IPC	dl1.access	dl1.hits	dl1.misses	dl1.replaces
Perlbench	2.281226	44,346,800	43,763,511	583,289	583,289
Bzip2	0.825097	71,738,831	63,043,198	8,695,633	8,695,633
Gcc	2.251028	37,206,785	37,049,698	157,087	157,087
Mcf	3.336702	33,330,471	33,288,137	42,334	42,334
Gobmk	1.989715	40,505,627	40,397,212	108,415	108,415
Hmmer	2.238295	39,919,064	39,412,388	506,676	506,676
Sjeng	2.240248	33,690,541	33,631,048	59,493	59,493
Libquantum	0.392533	25,548,601	20,033,690	5,514,911	5,514,911
H264ref	3.796412	64,732,808	64,621,972	110,836	110,737
Omnetpp	3.443292	26,366,227	26,283,568	82,659	82,659
Astar	1.101136	44,480,868	42,908,523	1,572,345	1,572,345

Nombre	dl1.wb	ul2.access	ul2.hits	ul2.misses	ul2.replaces
Perlbench	127,446	945,832	872,402	73,430	583,289
Bzip2	4,347,812	13,043,449	10,869,515	2,173,934	8,695,633
Gcc	119,611	667,031	616,545	50,486	157,087
Mcf	42,333	84,669	66,822	17,847	42,334
Gobmk	74,602	379,073	338,294	40,779	108,415
Hmmer	443,480	950,169	838,527	111,642	506,676
Sjeng	41,481	110,789	81,123	29,666	59,493
Libquantum	4,878,548	10,393,630	9,014,799	1,378,831	5,514,911
H264ref	66,281	188,908	187,592	1,316	110,737
Omnetpp	45,374	161,547	126,260	35,287	82,659
Astar	631,940	2,204,286	2,006,060	198,226	1,572,345

Nombre	ul2.wb	ul3.access	ul3.hits	ul3.misses	ul3.replaces
Perlbench	16,359	89,789	84,069	5,720	583,289
Bzip2	1,087,032	3,260,966	1,087,032	2,173,934	8,695,633
Gcc	26,145	76,631	52,773	23,858	157,087
Mcf	15,892	33,739	24,685	9,054	42,334
Gobmk	21,467	62,246	48,404	13,842	108,415
Hmmer	105,474	217,116	216,386	730	506,676
Sjeng	17,251	46,917	26,763	20,154	59,493
Libquantum	1,218,118	2,596,949	1,218,118	1,378,831	5,514,911
H264ref	248	1,564	1,250	314	110,737
Omnetpp	13,225	48,512	38,180	10,332	82,659
Astar	110,202	308,428	308,367	61	1,572,345

Nombre	ul3.wb	dtlb.access	dtlb.hits	dtlb.misses	aciertosDC
Perlbench	5,632	12,600,551	12,599,464	1,087	18,247,964
Bzip2	1,086,589	17,391,703	17,374,713	16,990	49,999,619
Gcc	22,304	9,681,788	9,681,400	388	15,623,194
Mcf	105	9,084,362	9,079,070	5,292	18,270,092
Gobmk	13,779	10,451,218	10,444,456	6,762	13,158,271
Hmmer	0	7,310,453	7,310,443	10	12,069,335
Sjeng	6,724	6,208,964	6,194,026	14,938	9,512,894
Libquantum	1,187,398	6,978,400	6,959,319	19,081	12,051,828
H264ref	0	16,098,081	16,098,079	2	16,405,225
Omnetpp	0	6,468,857	6,468,672	185	8,764,830
Astar	0	3,518,403	3,518,395	8	15,949,764

Nombre	missDC	totAccesosDC	hitsAC	missAC	totAccesosAC
Perlbench	26,098,836	44,346,800	18,290	26,080,546	26,098,836
Bzip2	21,739,212	71,738,831	8,695,565	13,043,647	21,739,212
Gcc	21,583,591	37,206,785	64,564	21,519,027	21,583,591
Mcf	15,060,379	33,330,471	50,525	15,009,854	15,060,379
Gobmk	27,347,356	40,505,627	128,218	27,219,138	27,347,356
Hmmer	27,849,729	39,919,064	13,218,866	14,630,863	27,849,729
Sjeng	24,177,647	33,690,541	14,112	24,163,535	24,177,647
Libquantum	13,496,773	25,548,601	3,365,643	10,131,130	13,496,773
H264ref	48,327,583	64,732,808	4,867,141	43,460,442	48,327,583
Omnetpp	17,601,397	26,366,227	1,309,733	16,291,664	17,601,397
Astar	28,531,104	44,480,868	8,317,713	20,213,391	28,531,104

Nombre	hitsDC_ld	missDC_ld	totDC_ld	hitsAC_ld	missAC_ld
Perlbench	5,647,788	26,098,836	31,746,624	18,290	26,080,546
Bzip2	32,608,380	21,739,212	54,347,592	8,695,565	13,043,647
Gcc	5,941,706	21,583,591	27,525,297	64,564	21,519,027
Mcf	9,186,196	15,060,379	24,246,575	50,525	15,009,854
Gobmk	2,707,334	27,347,356	30,054,690	128,218	27,219,138
Hmmer	4,759,268	27,849,729	32,608,997	13,218,866	14,630,863
Sjeng	3,304,208	24,177,647	27,481,855	14,112	24,163,535
Libquantum	5,073,929	13,496,773	18,570,702	3,365,643	10,131,130
H264ref	307,416	48,327,583	48,634,999	4,867,141	43,460,442
Omnetpp	2,296,313	17,601,397	19,897,710	1,309,733	16,291,664
Astar	12,431,745	28,531,104	40,962,849	8,317,713	20,213,391

Nombre	**totAC_ld**	**hitsDC_st**	**missDC_st**	**totDC_st**
Perlbench	26,098,836	12,600,176	0	12,600,176
Bzip2	21,739,212	17,391,239	0	17,391,239
Gcc	21,583,591	9,681,488	0	9,681,488
Mcf	15,060,379	9,083,896	0	9,083,896
Gobmk	27,347,356	10,450,937	0	10,450,937
Hmmer	27,849,729	7,310,067	0	7,310,067
Sjeng	24,177,647	6,208,686	0	6,208,686
Libquantum	13,496,773	6,977,899	0	6,977,899
H264ref	48,327,583	16,097,809	0	16,097,809
Omnetpp	17,601,397	6,468,517	0	6,468,517
Astar	28,531,104	3,518,019	0	3,518,019

E.6. Resultados suite float

E.6.1. Resultados sin el uso de la memoria ACDC, suite float.

Tabla E.6: Resultados para la suite Float sin memoria ACDC.

Nombre	total_issued_ld	total_issued_st	total_commited_ld
Bwaves	30,944,798	6,965,494	29,923,965
Gamess	25,319,343	13,063,955	24,345,014
Milc	35,923,940	18,754,328	35,913,822
Zeusmp	29,631,475	11,763,219	28,070,343
Gromacs	44,835,144	9,037,666	36,938,169
CactusADM	38,314,366	7,744,184	36,129,566
Leslie3d	27,216,538	2,603,275	26,234,970
Namd	29,060,723	9,438,455	27,218,280
DealII	18,166,447	2,993,322	16,337,412
Soplex	19,597,625	105,525	15,800,342
Povray	39,063,949	11,617,954	35,534,327
Calculix	26,053,320	14,887,899	24,496,664
GemsFDTD	22,975,413	11,570,977	22,330,291
Tonto	21,760,633	8,676,269	21,063,627
Lbm	34,188,518	5,375,783	28,757,967
Wrf	27,283,832	10,634,638	26,309,667
Sphin3	28,623,333	10,572,972	26,846,836

Nombre	total_commited_st	num_refs	num_ld	num_st
Bwaves	6,965,171	36,889,164	29,923,990	6,965,174
Gamess	12,332,225	36,677,263	24,345,030	12,332,233
Milc	18,754,080	54,668,036	35,913,909	18,754,127
Zeusmp	11,760,980	39,831,329	28,070,347	11,760,982
Gromacs	8,654,623	45,592,814	36,938,183	8,654,631
CactusADM	7,738,762	43,868,356	36,129,588	7,738,768
Leslie3d	2,603,273	28,838,274	26,234,997	2,603,277
Namd	8,884,583	36,102,879	27,218,289	8,884,590
DealII	2,955,000	19,292,420	16,337,420	2,955,000
Soplex	102,375	15,902,720	15,800,345	102,375
Povray	10,392,424	45,926,784	35,534,350	10,392,434
Calculix	13,723,752	38,220,426	24,496,668	13,723,758
GemsFDTD	11,197,269	33,527,566	22,330,297	11,197,269
Tonto	8,463,067	29,526,701	21,063,634	8,463,067
Lbm	5,375,259	34,133,239	28,757,980	5,375,259
Wrfo	10,299,486	36,609,173	26,309,677	10,299,496
Sphin3	9,871,312	36,718,173	26,846,842	9,871,331

Nombre	elapsed_time	total_refs	total_ld	total_st	simcycle
Bwaves	305	36,948,622	29,963,130	6,985,492	88,428,627
Gamess	190	40,561,116	26,707,916	13,853,200	38,173,252
Milc	246	54,703,362	35,948,457	18,754,905	47,495,539
Zeusmp	300	39,940,363	28,174,538	11,765,825	99,618,466
Gromacs	281	57,149,815	47,580,305	9,569,510	69,881,323
CactusADM	603	44,020,043	36,248,761	7,771,282	196,892,878
Leslie3d	279	29,018,573	26,414,922	2,603,651	92,778,148
Namd	292	39,539,354	29,942,586	9,596,768	41,879,602
DealII	413	21,294,196	18,104,436	3,189,760	60,188,848
Soplex	373	17,980,799	17,871,755	109,044	130,643,708
Povray	257	56,934,714	44,011,139	12,923,575	50,874,121
Calculix	210	44,253,239	28,160,014	16,093,225	40,515,597
GemsFDTD	192	35,821,408	23,785,254	12,036,154	35,528,758
Tonto	185	32,107,487	22,814,509	9,292,978	34,312,635
Lbm	605	34,208,632	28,832,157	5,376,475	99,906,761
Wrf	220	38,323,768	27,385,293	10,938,475	50,275,529
Sphin3	202	40,925,588	29,822,959	11,102,629	40,313,223

Nombre	IPC	dl1.access	dl1.hits	dl1.misses	dl1.replaces
Bwaves	1.130856	33,568,079	32,886,160	681,919	681,919
Gamess	2.619635	31,707,172	31,707,111	61	60
Milc	2.105461	54,667,514	50,022,901	4,644,613	4,644,613
Zeusmp	1.003830	39,829,079	38,218,213	1,610,866	1,610,866
Gromacs	1.430998	48,276,234	47,220,559	1,055,675	1,055,675
CactusADM	0.507890	30,363,738	27,731,075	2,632,663	2,632,663
Leslie3d	1.077840	28,839,092	27,417,962	1,421,130	1,421,130
Namd	2.387797	35,108,249	35,094,118	14,131	14,131
DealII	1.661437	18,810,793	17,479,584	1,331,209	1,331,209
Soplex	0.765441	16,018,216	13,481,640	2,536,576	2,536,573
Povray	1.965636	44,524,283	44,411,366	112,917	112,731
Calculix	2.468185	36,193,512	36,193,151	361	358
GemsFDTD	2.814621	31,462,245	31,462,237	8	0
Tonto	2.914378	28,208,488	28,207,672	816	816
Lbm	1.000933	34,195,983	28,818,481	5,377,502	5,377,502
Wrf	1.989039	32,266,333	31,060,748	1,205,585	1,205,585
Sphin3	2.480576	35,843,363	35,821,428	21,935	21,935

Nombre	dl1.wb	ul2.access	ul2.hits	ul2.misses
Bwaves	323,065	1,005,161	834,657	170,504
Gamess	40	177	151	26
Milc	4,644,299	9,288,972	8,127,582	1,161,390
Zeusmp	87,787	1,700,020	1,306,245	393,775
Gromacs	210,221	1,266,173	1,178,010	88,163
CactusADM	2,579,534	5,212,197	2,344,075	2,868,122
Leslie3d	560,534	1,981,714	1,621,680	360,034
Namd	14,131	28,272	24,731	3,541
DealII	37,045	1,368,254	1,359,246	9,008
Soplex	10,670	2,547,248	238,860	2,308,388
Povray	18,492	137,597	137,544	53
Calculix	152	611	573	38
GemsFDTD	0	46,140	46,131	9
Tonto	267	30,565	30,542	23
Lbm	3,962,904	9,340,406	7,925,807	1,414,599
Wrf	1,143,473	2,351,042	1,859,007	492,035
Sphin3	10,462	32,429	26,946	5,483

Nombre	ul2.replaces	ul2.wb	ul3.access	ul3.hits	ul3.misses
Bwaves	170,504	79,298	249,802	79,346	170,456
Gamess	8	1	27	1	26
Milc	1,161,390	1,161,177	2,322,567	1,161,515	1,161,052
Zeusmp	393,775	22,798	416,573	102,857	313,716
Gromacs	88,163	29,715	117,878	95,316	22,562
CactusADM	2,868,122	2,579,492	5,447,614	2,864,858	2,582,756
Leslie3d	360,034	143,126	503,160	143,194	359,966
Namd	3,541	3,532	7,073	3,537	3,536
DealII	9,008	7,803	16,811	16,811	0
Soplex	2,308,388	5,471	2,313,859	2,313,649	1,930
Povray	0	0	53	0	53
Calculix	9	0	38	0	38
GemsFDTD	0	0	9	0	9
Tonto	16	5	28	11	17
Lbm	1,414,599	1,060,908	2,475,507	1,060,908	1,414,599
Wrf	492,035	392,715	884,750	534,116	350,634
Sphin3	5,483	2,722	8,205	2,827	5,378

Nombre	ul3.replaces	ul3.wb	dtlb.access	dtlb.hits	dtlb.misses
Bwaves	170,456	78,978	36,889,780	36,887,113	2,667
Gamess	0	0	37,319,028	37,319,028	0
Milc	1,161,052	1,161,002	54,676,682	54,621,010	55,672
Zeusmp	313,716	26,230	39,836,380	39,831,209	5,171
Gromacs	13,352	6,556	50,526,649	50,524,435	2,214
CactusADM	2,582,756	2,579,481	43,911,648	41,485,058	2,426,590
Leslie3d	359,966	147,844	28,839,110	28,833,283	5,827
Namd	0	0	37,459,227	37,459,172	55
DealII	0	0	19,383,732	19,383,730	2
Soplex	1,765	51	16,046,657	16,019,275	27,382
Povray	0	0	48,629,824	48,629,821	3
Calculix	0	0	39,151,970	39,151,970	0
GemsFDTD	0	0	33,819,623	33,819,623	0
Tonto	0	0	29,824,402	29,824,402	0
Lbm	1,414,599	1,060,907	34,195,983	34,173,882	22,101
Wrf	350,634	338,295	37,260,887	37,252,688	8,199
Sphin3	8	0	38,010,782	38,010,697	85

E.6.2. Resultados con la organización secuencial, suite float.

Tabla E.7: Resultados para la suite float con la organización Secuencial.

NOMBRE	Modo	total_issued_ld	total_issued_st	total_commited_ld
Bwaves	Secuencial	30,944,332	6,965,314	29,923,965
Gamess	Secuencial	25,204,354	12,938,941	24,345,014
Milc	Secuencial	35,923,578	18,754,345	35,913,822
Zeusmp	Secuencial	29,627,113	11,762,794	28,070,343
Gromacs	Secuencial	44,557,242	9,080,031	36,938,169
CactusADM	Secuencial	38,266,599	7,744,186	36,129,566
Leslie3d	Secuencial	27,216,449	2,603,279	26,234,970
Namd	Secuencial	28,518,241	9,302,840	27,218,279
DealII	Secuencial	18,131,708	2,998,183	16,337,412
Soplex	Secuencial	19,588,781	105,225	15,800,342
Povray	Secuencial	38,439,834	11,488,788	35,534,327
Calculix	Secuencial	25,606,071	14,504,747	24,496,664
GemsFDTD	Secuencial	22,979,116	11,531,290	22,330,291
Tonto	Secuencial	21,678,396	8,651,850	21,063,627
Lbm	Secuencial	34,188,852	5,375,918	28,757,967
Wrf	Secuencial	27,283,914	10,637,762	26,309,667
Sphin3	Secuencial	28,636,127	10,553,354	26,846,836

NOMBRE	total_commited_st	num_refs	num_ld	num_st	elapsed_time
Bwaves	6,965,171	36,889,164	29,923,990	6,965,174	312
Gamess	12,332,225	36,677,263	24,345,030	12,332,233	173
Milc	18,754,080	54,668,036	35,913,909	18,754,127	263
Zeusmp	11,760,980	39,831,329	28,070,347	11,760,982	306
Gromacs	8,654,623	45,592,814	36,938,183	8,654,631	272
CactusADM	7,738,762	43,868,356	36,129,588	7,738,768	584
Leslie3d	2,603,273	28,838,274	26,234,997	2,603,277	269
Namd	8,884,581	36,102,879	27,218,289	8,884,590	179
DealII	2,955,000	19,292,420	16,337,420	2,955,000	201
Soplex	102,375	15,902,720	15,800,345	102,375	357
Povray	10,392,424	45,926,784	35,534,350	10,392,434	239
Calculix	13,723,752	38,220,426	24,496,668	13,723,758	188
GemsFDTD	11,197,269	33,527,566	22,330,297	11,197,269	177
Tonto	8,463,067	29,526,701	21,063,634	8,463,067	171
Lbm	5,375,259	34,133,239	28,757,980	5,375,259	327
Wrf	10,299,486	36,609,171	26,309,676	10,299,495	215
Sphin3	9,871,314	36,718,177	26,846,845	9,871,332	186

NOMBRE	total_refs	total_ld	total_st	simcycle	IPC
Bwaves	36,947,825	29,963,134	6,984,691	86,719,993	1.153137
Gamess	40,464,290	26,555,967	13,908,323	27,487,470	3.638021
Milc	54,703,306	35,948,417	18,754,889	47,001,904	2.127574
Zeusmp	39,938,892	28,173,474	11,765,418	96,572,388	1.035493
Gromacs	56,464,617	46,926,614	9,538,003	62,145,903	1.609117
CactusADM	43,993,173	36,227,265	7,765,908	187,794,222	0.532498
Leslie3d	29,021,628	26,417,731	2,603,897	85,915,223	1.163938
Namd	39,516,675	29,919,896	9,596,779	31,130,367	3.212298
DealII	20,915,512	17,767,114	3,148,398	51,648,405	1.936168
Soplex	17,941,832	17,833,195	108,637	120,048,707	0.832995
Povray	55,200,236	42,415,988	12,784,248	43,193,685	2.315153
Calculix	43,725,270	27,814,836	15,910,434	30,140,362	3.317810
GemsFDTD	35,842,123	23,795,172	12,046,951	28,717,489	3.482199
Tonto	32,065,279	22,808,158	9,257,121	27,898,224	3.584458
Lbm	34,208,755	28,832,242	5,376,513	97,969,987	1.020721
Wrf	38,108,783	27,277,995	10,830,788	47,652,360	2.098532
Sphin3	41,600,419	30,452,905	11,147,514	31,634,418	3.161114

NOMBRE	dl1.access	dl1.hits	dl1.miss	dl1.replaces	dl1.wb
Bwaves	29,571,285	28,889,365	681,920	681,920	323,065
Gamess	28,197,306	28,197,243	63	62	42
Milc	23,358,801	18,714,169	4,644,632	4,644,632	4,644,315
Zeusmp	37,546,317	35,935,669	1,610,648	1,610,648	87,787
Gromacs	41,849,924	40,748,825	1,101,099	1,101,099	211,403
CactusADM	22,565,292	19,932,790	2,632,502	2,632,502	2,579,533
Leslie3d	12,240,087	10,818,959	1,421,128	1,421,128	560,532
Namd	22,934,945	22,920,814	14,131	14,131	14,128
DealII	17,523,726	16,192,514	1,331,212	1,331,212	37,046
Soplex	10,252,824	7,704,820	2,548,004	2,548,004	10,971
Povray	38,964,479	38,851,561	112,918	112,733	18,493
Calculix	25,919,460	25,919,100	360	358	152
GemsFDTD	24,060,756	24,060,748	8	0	0
Tonto	26,036,575	26,035,755	820	820	269
Lbm	15,970,948	10,595,685	5,375,263	5,375,263	3,960,719
Wrf	29,368,049	28,162,298	1,205,751	1,205,751	1,143,473
Sphin3	31,477,891	31,455,978	21,913	21,913	10,459

NOMBRE	ul2.access	ul2.hits	ul2.miss	ul2.replaces	ul2.wb
Bwaves	1,005,162	834,658	170,504	170,504	79,298
Gamess	169	145	24	8	2
Milc	9,289,006	8,127,616	1,161,390	1,161,390	1,161,177
Zeusmp	1,699,796	1,306,020	393,776	393,776	22,798
Gromacs	1,312,778	1,218,547	94,231	94,231	29,985
CactusADM	5,212,035	2,342,723	2,869,312	2,869,312	2,579,491
Leslie3d	1,981,710	1,621,677	360,033	360,033	143,126
Namd	28,268	24,728	3,540	3,540	3,532
DealII	1,368,258	1,359,249	9,009	9,009	7,800
Soplex	2,558,977	248,136	2,310,841	2,310,841	5,330
Povray	137,600	137,546	54	0	0
Calculix	603	567	36	8	0
GemsFDTD	46,136	46,128	8	0	0
Tonto	30,050	30,034	16	12	3
Lbm	9,335,982	7,921,439	1,414,543	1,414,543	1,060,907
Wrf	2,351,200	1,859,052	492,148	492,148	392,760
Sphin3	32,400	26,917	5,483	5,483	2,724

NOMBRE	ul3.access	ul3.hits	ul3.miss	ul3.replaces	ul3.wb
Bwaves	249,802	79,563	170,239	170,239	80,510
Gamess	26	2	24	0	0
Milc	2,322,567	1,161,516	1,161,051	1,161,051	1,161,002
Zeusmp	416,574	103,260	313,314	313,314	30,635
Gromacs	124,216	106,473	17,743	4	4
CactusADM	5,448,803	2,866,399	2,582,404	2,582,404	2,579,481
Leslie3d	503,159	143,226	359,933	359,933	153,496
Namd	7,072	3,536	3,536	0	0
DealII	16,809	16,809	0	0	0
Soplex	2,316,171	2,315,830	341	0	0
Povray	54	0	54	0	0
Calculix	36	0	36	0	0
GemsFDTD	8	0	8	0	0
Tonto	19	10	9	0	0
Lbm	2,475,450	1,060,907	1,414,543	1,414,543	1,060,907
Wrf	884,908	534,559	350,349	350,349	338,174
Sphin3	8,207	2,831	5,376	0	0

NOMBRE	dtlb.access	dtlb.hits	dtlb.miss	hitDC	missDC
Bwaves	6,965,530	6,964,260	1,270	10,650,914	22,606,114
Gamess	12,332,542	12,332,542	0	15,737,229	15,865,095
Milc	34,651,009	34,595,356	55,653	50,062,861	4,604,722
Zeusmp	11,761,252	11,760,876	376	14,034,911	25,785,402
Gromacs	8,654,901	8,654,590	311	15,003,076	33,195,311
CactusADM	7,751,449	5,371,331	2,380,118	16,176,765	14,826,530
Leslie3d	2,606,792	2,604,489	2,303	19,203,339	9,636,814
Namd	8,885,095	8,885,040	55	20,549,272	14,050,364
DealII	2,955,375	2,955,373	2	4,498,737	14,568,798
Soplex	102,746	102,720	26	5,939,540	10,150,512
Povray	10,392,785	10,392,784	1	15,039,162	28,653,612
Calculix	13,724,117	13,724,117	0	23,771,689	12,195,718
GemsFDTD	11,197,631	11,197,631	0	18,865,581	12,868,612
Tonto	8,463,420	8,463,420	0	10,691,152	17,573,569
Lbm	5,375,735	5,359,159	16,576	18,226,247	15,969,788
Wrf	10,300,063	10,292,424	7,639	12,713,966	19,068,580
Sphin3	9,871,674	9,871,633	41	14,421,731	21,612,600

NOMBRE	totAcceDC	hitAC	missAC	totAccAC	hitDC_ld
Bwaves	33,257,028	17	22,606,097	22,606,114	3,685,743
Gamess	31,602,324	58,370	15,806,725	15,865,095	3,405,004
Milc	54,667,583	32	4,604,690	4,604,722	31,308,781
Zeusmp	39,820,313	57	25,785,345	25,785,402	2,273,931
Gromacs	48,198,387	12	33,195,299	33,195,311	6,348,453
CactusADM	31,003,295	622,718	14,203,812	14,826,530	8,438,003
Leslie3d	28,840,153	1,599,051	8,037,763	9,636,814	16,600,066
Namd	34,599,636	12	14,050,352	14,050,364	11,664,691
DealII	19,067,535	9	14,568,789	14,568,798	1,543,737
Soplex	16,090,052	551,644	9,598,868	10,150,512	5,837,165
Povray	43,692,774	58,549	28,595,063	28,653,612	4,646,738
Calculix	35,967,407	81,999	12,113,719	12,195,718	10,047,937
GemsFDTD	31,734,193	25,625	12,842,987	12,868,612	7,668,312
Tonto	28,264,721	18,932	17,554,637	17,573,569	2,228,085
Lbm	34,196,035	3,317,181	12,652,607	15,969,788	18,225,087
Wrf	31,782,546	6	19,068,574	19,068,580	2,414,480
Sphin3	36,034,331	23,998	21,588,602	21,612,600	4,550,417

NOMBRE	missDC_ld	totDC_ld	hitAC_ld	missAC_ld	totAC_ld
Bwaves	22,606,114	26,291,857	17	22,606,097	22,606,114
Gamess	15,865,095	19,270,099	58,370	15,806,725	15,865,095
Milc	4,604,722	35,913,503	32	4,604,690	4,604,722
Zeusmp	25,785,402	28,059,333	57	25,785,345	25,785,402
Gromacs	33,195,311	39,543,764	12	33,195,299	33,195,311
CactusADM	14,826,530	23,264,533	622,718	14,203,812	14,826,530
Leslie3d	9,636,814	26,236,880	1,599,051	8,037,763	9,636,814
Namd	14,050,364	25,715,055	12	14,050,352	14,050,364
DealII	14,568,798	16,112,535	9	14,568,789	14,568,798
Soplex	10,150,512	15,987,677	551,644	9,598,868	10,150,512
Povray	28,653,612	33,300,350	58,549	28,595,063	28,653,612
Calculix	12,195,718	22,243,655	81,999	12,113,719	12,195,718
GemsFDTD	12,868,612	20,536,924	25,625	12,842,987	12,868,612
Tonto	17,573,569	19,801,654	18,932	17,554,637	17,573,569
Lbm	10,595,689	28,820,776	3,317,181	7,278,508	10,595,689
Wrf	19,068,580	21,483,060	6	19,068,574	19,068,580
Sphin3	21,612,600	26,163,017	23,998	21,588,602	21,612,600

NOMBRE	hitDC_st	missDC_st	totDC_st
Bwaves	6,965,171	0	6,965,171
Gamess	12,332,225	0	12,332,225
Milc	18,754,080	0	18,754,080
Zeusmp	11,760,980	0	11,760,980
Gromacs	8,654,623	0	8,654,623
CactusADM	7,738,762	0	7,738,762
Leslie3d	2,603,273	0	2,603,273
Namd	8,884,581	0	8,884,581
DealII	2,955,000	0	2,955,000
Soplex	102,375	0	102,375
Povray	10,392,424	0	10,392,424
Calculix	13,723,752	0	13,723,752
GemsFDTD	11,197,269	0	11,197,269
Tonto	8,463,067	0	8,463,067
Lbm	1,160	5,374,099	5,375,259
Wrf	10,299,486	0	10,299,486
Sphin3	9,871,314	0	9,871,314

E.6.3. Resultados con la organización paralela, suite float.

Tabla E.8: Resultados para la suite float con la organización Paralela.

NOMBRE	Modo	total_issued_ld	total_issued_st	total_commited_ld
Bwaves	Paralelo	30,944,343	6,965,314	29,923,965
Gamess	Paralelo	25,233,153	12,996,461	24,345,014
Milc	Paralelo	35,923,468	18,754,314	35,913,820
Zeusmp	Paralelo	29,627,222	11,762,865	28,070,343
Gromacs	Paralelo	44,686,974	9,087,555	36,938,169
CactusADM	Paralelo	38,267,133	7,744,186	36,129,566
Leslie3d	Paralelo	27,216,449	2,603,279	26,234,970
Namd	Paralelo	28,642,538	9,314,123	27,218,280
DealII	Paralelo	18,107,651	2,998,062	16,337,412
Soplex	Paralelo	19,599,241	105,525	15,800,342
Povray	Paralelo	38,481,386	11,503,461	35,534,327
Calculix	Paralelo	25,704,151	14,513,601	24,496,664
GemsFDTD	Paralelo	22,968,295	11,525,310	22,330,291
Tonto	Paralelo	21,736,011	8,670,452	21,063,627
Lbm	Paralelo	34,188,638	5,375,895	28,757,967
Wrf	Paralelo	27,290,106	10,637,724	26,309,667
Sphin3	Paralelo	28,698,196	10,657,819	26,846,836

NOMBRE	total_commited_st	num_refs	num_ld	num_st	elapsed_time
Bwaves	6,965,171	36,889,164	29,923,990	6,965,174	298
Gamess	12,332,225	36,677,262	24,345,029	12,332,233	162
Milc	18,754,079	54,668,038	35,913,910	18,754,128	253
Zeusmp	11,760,980	39,831,329	28,070,347	11,760,982	289
Gromacs	8,654,623	45,592,814	36,938,183	8,654,631	259
CactusADM	7,738,762	43,868,356	36,129,588	7,738,768	557
Leslie3d	2,603,273	28,838,274	26,234,997	2,603,277	257
Namd	8,884,583	36,102,879	27,218,289	8,884,590	168
DealII	2,955,000	19,292,420	16,337,420	2,955,000	194
Soplex	102,375	15,902,720	15,800,345	102,375	337
Povray	10,392,424	45,926,784	35,534,350	10,392,434	224
Calculix	13,723,752	38,220,426	24,496,668	13,723,758	177
GemsFDTD	11,197,269	33,527,566	22,330,297	11,197,269	167
Tonto	8,463,067	29,526,700	21,063,633	8,463,067	163
Lbm	5,375,259	34,133,239	28,757,980	5,375,259	312
Wrf	10,299,486	36,609,171	26,309,676	10,299,495	206
Sphin3	9,871,312	36,718,169	26,846,839	9,871,330	175

NOMBRE	total_refs	total_ld	total_st	simcycle	IPC
Bwaves	36,947,827	29,963,136	6,984,691	86,700,811	1.153392
Gamess	40,444,103	26,537,879	13,906,224	26,291,278	3.803543
Milc	54,703,457	35,948,503	18,754,954	47,001,590	2.127588
Zeusmp	39,938,876	28,173,424	11,765,452	95,980,739	1.041876
Gromacs	57,176,272	47,614,409	9,561,863	61,439,855	1.627608
CactusADM	43,993,173	36,227,265	7,765,908	187,793,984	0.532498
Leslie3d	29,021,438	26,417,910	2,603,528	85,877,021	1.164456
Namd	39,471,411	29,863,373	9,608,038	30,327,680	3.297318
DealII	20,860,588	17,725,531	3,135,057	51,558,609	1.939540
Soplex	17,982,147	17,872,795	109,352	118,454,035	0.844209
Povray	55,452,946	42,575,670	12,877,276	42,969,733	2.327220
Calculix	43,709,981	27,816,223	15,893,758	29,312,176	3.411552
GemsFDTD	35,707,168	23,712,389	11,994,779	27,873,035	3.587697
Tonto	32,227,028	22,914,567	9,312,461	26,839,654	3.725830
Lbm	34,208,652	28,832,177	5,376,475	97,967,439	1.020747
Wrf	38,308,328	27,377,769	10,930,559	46,477,229	2.151591
Sphin3	42,158,087	30,656,549	11,501,538	30,884,736	3.237845

NOMBRE	dl1.access	dl1.hits	dl1.miss	dl1.replaces	dl1.wb
Bwaves	33,257,045	32,575,126	681,919	681,919	323,065
Gamess	31,681,780	31,681,717	63	62	41
Milc	54,667,558	50,022,943	4,644,615	4,644,615	4,644,298
Zeusmp	39,820,700	38,209,976	1,610,724	1,610,724	87,787
Gromacs	48,307,926	47,204,046	1,103,880	1,103,880	211,463
CactusADM	31,003,295	28,370,615	2,632,680	2,632,680	2,579,534
Leslie3d	28,840,153	27,419,023	1,421,130	1,421,130	560,534
Namd	34,825,726	34,811,595	14,131	14,131	14,131
DealII	18,965,798	17,634,577	1,331,221	1,331,221	37,045
Soplex	16,089,083	13,541,463	2,547,620	2,547,620	10,678
Povray	43,805,709	43,692,791	112,918	112,733	18,493
Calculix	36,187,638	36,187,277	361	358	152
GemsFDTD	31,495,966	31,495,956	10	0	0
Tonto	28,301,334	28,300,514	820	820	269
Lbm	34,195,868	28,818,366	5,377,502	5,377,502	3,962,904
Wrf	31,782,461	30,576,711	1,205,750	1,205,750	1,143,473
Sphin3	36,071,633	36,049,698	21,935	21,935	10,462

NOMBRE	ul2.access	ul2.hits	ul2.miss	ul2.replaces	ul2.wb	ul3.access
Bwaves	1,005,161	834,657	170,504	170,504	79,298	249,802
Gamess	169	145	24	8	2	26
Milc	9,288,972	8,127,581	1,161,391	1,161,391	1,161,177	2,322,568
Zeusmp	1,699,867	1,306,091	393,776	393,776	22,798	416,574
Gromacs	1,315,619	1,222,033	93,586	93,586	29,974	123,560
CactusADM	5,212,214	2,344,085	2,868,129	2,868,129	2,579,492	5,447,621
Leslie3d	1,981,714	1,621,680	360,034	360,034	143,126	503,160
Namd	28,271	24,731	3,540	3,540	3,532	7,072
DealII	1,368,266	1,359,245	9,021	9,021	7,808	16,829
Soplex	2,558,301	247,336	2,310,965	2,310,965	5,470	2,316,435
Povray	137,599	137,545	54	0	0	54
Calculix	604	567	37	8	0	37
GemsFDTD	46,140	46,131	9	0	0	9
Tonto	30,671	30,647	24	18	5	29
Lbm	9,340,406	7,925,807	1,414,599	1,414,599	1,060,908	2,475,507
Wrf	2,351,198	1,859,051	492,147	492,147	392,760	884,907
Sphin3	32,425	26,942	5,483	5,483	2,722	8,205

NOMBRE	ul3.hits	ul3.miss	ul3.replaces	ul3.wb	dtlb.access	dtlb.hits
Bwaves	79,563	170,239	170,239	80,510	6,965,530	6,964,260
Gamess	2	24	0	0	12,332,531	12,332,531
Milc	1,161,517	1,161,051	1,161,051	1,161,002	34,654,318	34,598,665
Zeusmp	103,260	313,314	313,314	30,635	11,761,252	11,760,876
Gromacs	105,819	17,741	4	4	8,654,900	8,654,589
CactusADM	2,865,218	2,582,403	2,582,403	2,579,482	7,751,449	5,371,331
Leslie3d	143,226	359,934	359,934	153,496	2,606,223	2,603,920
Namd	3,536	3,536	0	0	8,884,944	8,884,889
DealII	16,829	0	0	0	2,955,378	2,955,376
Soplex	2,316,094	341	0	0	102,746	102,720
Povray	0	54	0	0	10,392,776	10,392,775
Calculix	0	37	0	0	13,724,086	13,724,086
GemsFDTD	0	9	0	0	11,197,621	11,197,621
Tonto	11	18	0	0	8,463,357	8,463,357
Lbm	1,060,909	1,414,598	1,414,598	1,060,907	5,375,735	5,359,159
Wrf	534,559	350,348	350,348	338,174	10,300,059	10,292,420
Sphin3	2,826	5,379	0	0	9,871,639	9,871,598

NOMBRE	dtlb.miss	hitsDC	missDC	totAccesosDC	hitsAC	missAC
Bwaves	1,270	10,650,914	22,606,131	33,257,045	17	22,606,114
Gamess	0	15,739,807	15,941,973	31,681,780	70,077	15,871,896
Milc	55,653	50,054,800	4,612,758	54,667,558	34	4,612,724
Zeusmp	376	13,123,247	26,697,453	39,820,700	53	26,697,400
Gromacs	311	15,027,392	33,280,534	48,307,926	81	33,280,453
CactusADM	2,380,118	16,176,765	14,826,530	31,003,295	622,718	14,203,812
Leslie3d	2,303	17,635,760	11,204,393	28,840,153	2,537,499	8,666,894
Namd	55	19,441,486	15,384,240	34,825,726	203,474	15,180,766
DealII	2	4,490,948	14,474,850	18,965,798	11	14,474,839
Soplex	26	7,987,094	8,101,989	16,089,083	552,379	7,549,610
Povray	1	14,117,946	29,687,763	43,805,709	85,097	29,602,666
Calculix	0	23,583,937	12,603,701	36,187,638	86,184	12,517,517
GemsFDTD	0	19,214,286	12,281,680	31,495,966	58,068	12,223,612
Tonto	0	10,228,690	18,072,644	28,301,334	20,647	18,051,997
Lbm	16,576	18,227,861	15,968,007	34,195,868	3,317,176	12,650,831
Wrf	7,639	12,713,962	19,068,499	31,782,461	7	19,068,492
Sphin3	41	14,400,650	21,670,983	36,071,633	24,707	21,646,276

NOMBRE	totAccAC	hitDC_ld	missDC_ld	totDC_ld	hitAC_ld	missAC_ld
Bwaves	22,606,131	3,685,743	22,606,131	26,291,874	17	22,606,114
Gamess	15,941,973	3,407,582	15,941,973	19,349,555	70,077	15,871,896
Milc	4,612,758	31,300,721	4,612,758	35,913,479	34	4,612,724
Zeusmp	26,697,453	1,362,267	26,697,453	28,059,720	53	26,697,400
Gromacs	33,280,534	6,372,769	33,280,534	39,653,303	81	33,280,453
CactusADM	14,826,530	8,438,003	14,826,530	23,264,533	622,718	14,203,812
Leslie3d	11,204,393	15,032,487	11,204,393	26,236,880	2,537,499	8,666,894
Namd	15,384,240	10,556,903	15,384,240	25,941,143	203,474	15,180,766
DealII	14,474,850	1,535,948	14,474,850	16,010,798	11	14,474,839
Soplex	8,101,989	7,884,719	8,101,989	15,986,708	552,379	7,549,610
Povray	29,687,763	3,725,522	29,687,763	33,413,285	85,097	29,602,666
Calculix	12,603,701	9,860,185	12,603,701	22,463,886	86,184	12,517,517
GemsFDTD	12,281,680	8,017,017	12,281,680	20,298,697	58,068	12,223,612
Tonto	18,072,644	1,765,623	18,072,644	19,838,267	20,647	18,051,997
Lbm	15,968,007	18,224,814	10,595,795	28,820,609	3,317,176	7,278,619
Wrf	19,068,499	2,414,476	19,068,499	21,482,975	7	19,068,492
Sphin3	21,670,983	4,529,338	21,670,983	26,200,321	24,707	21,646,276

NOMBRE	totAC_ld	hitDC_st	missDC_st	totDC_st
Bwaves	22,606,131	6,965,171	0	6,965,171
Gamess	15,941,973	12,332,225	0	12,332,225
Milc	4,612,758	18,754,079	0	18,754,079
Zeusmp	26,697,453	11,760,980	0	11,760,980
Gromacs	33,280,534	8,654,623	0	8,654,623
CactusADM	14,826,530	7,738,762	0	7,738,762
Leslie3d	11,204,393	2,603,273	0	2,603,273
Namd	15,384,240	8,884,583	0	8,884,583
DealII	14,474,850	2,955,000	0	2,955,000
Soplex	8,101,989	102,375	0	102,375
Povray	29,687,763	10,392,424	0	10,392,424
Calculix	12,603,701	13,723,752	0	13,723,752
GemsFDTD	12,281,680	11,197,269	0	11,197,269
Tonto	18,072,644	8,463,067	0	8,463,067
Lbm	10,595,795	3,047	5,372,212	5,375,259
Wrf	19,068,499	10,299,486	0	10,299,486
Sphin3	21,670,983	9,871,312	0	9,871,312

E.7. Resultados con 100 ficheros de análisis. Suite enteros.

En esta sección se muestran los resultados obtenidos de las simulaciones con 100 ficheros de análisis previo. En primer lugar se muestran la suite enteros en los modos paralelo y secuencial. Después se muestran la suite float, también en los dos modos paralelo y secuencial.

E.7.1. Paralelo, 100 tramos.

Tabla E.9: Tabla con resultados de la suite entera y con 100 ficheros de análisis previo.

NOMBRE	Modo	sim_total_issued_loads	sim_total_issued_stores
Perlbench	Paralelo	36,139,529	14,650,152
Bzip2	Paralelo	58,695,571	17,391,278
Gcc	Paralelo	32,363,095	10,690,593
Mcf	Paralelo	26,881,008	9,523,322
Gobmk	Paralelo	33,094,987	11,837,007
Hmmer	Paralelo	39,002,937	9,201,961
Sjeng	Paralelo	29,269,515	6,760,976
Libquantum	Paralelo	29,278,525	6,977,941
H264ref	Paralelo	49,411,963	16,396,711
Omnetpp	Paralelo	24,055,618	6,741,925
Astar	Paralelo	44,431,447	7,152,459

NOMBRE	sim_total_commited_loads	sim_total_commited_stores	sim_num_refs
Perlbench	32,075,588	12,600,176	44,675,773
Bzip2	54,347,500	17,391,239	71,738,830
Gcc	28,515,273	9,681,488	38,196,768
Mcf	25,780,211	9,083,896	34,864,125
Gobmk	27,076,634	10,450,937	37,527,588
Hmmer	32,198,719	7,310,067	39,508,800
Sjeng	24,750,238	6,208,686	30,958,933
Libquantum	18,570,642	6,977,899	25,548,548
H264ref	48,203,947	16,097,809	64,301,844
Omnetpp	21,880,320	6,468,517	28,348,853
Astar	23,853,369	3,518,019	27,371,393

NOMBRE	sim_num_loads	sim_num_stores	sim_elapsed_time
Perlbench	32,075,593	12,600,180	212
Bzip2	54,347,567	17,391,263	482
Gcc	28,515,279	9,681,489	199
Mcf	25,780,222	9,083,903	164
Gobmk	27,076,644	10,450,944	220
Hmmer	32,198,731	7,310,069	213
Sjeng	24,750,246	6,208,687	196
Libquantum	18,570,646	6,977,902	571
H264ref	48,204,012	16,097,832	200
Omnetpp	21,880,328	6,468,525	158
Astar	23,853,373	3,518,020	343

NOMBRE	sim_total_refs	sim_total_loads	sim_total_stores	sim_cycle
Perlbench	58,163,179	41,928,628	16,234,551	43,836,458
Bzip2	71,739,050	54,347,732	17,391,318	121,197,896
Gcc	49,123,074	36,587,633	12,535,441	44,423,434
Mcf	37,594,199	27,866,768	9,727,431	29,969,724
Gobmk	55,352,406	40,913,037	14,439,369	50,260,912
Hmmer	59,062,329	47,092,940	11,969,389	44,672,560
Sjeng	43,516,036	35,672,167	7,843,869	44,635,374
Libquantum	25,548,810	18,570,827	6,977,983	252,352,879
H264ref	68,617,689	51,478,655	17,139,034	26,310,865
Omnetpp	33,682,931	26,514,540	7,168,391	29,040,286
Astar	69,972,835	60,763,530	9,209,305	90,799,135

NOMBRE	sim_IPC	dl1.accesses	dl1.hits	dl1.misses	dl1.replacements
Perlbench	2.281206	44,347,833	43,764,537	583,296	583,296
Bzip2	0.825097	71,738,831	63,043,198	8,695,633	8,695,633
Gcc	2.251064	37,218,007	37,060,919	157,088	157,088
Mcf	3.336701	33,330,472	33,288,138	42,334	42,334
Gobmk	1.989618	40,507,523	40,399,099	108,424	108,424
Hmmer	2.238511	39,919,052	39,412,376	506,676	506,676
Sjeng	2.240376	33,692,200	33,632,730	59,470	59,470
Libquantum	0.39627	25,548,603	20,033,691	5,514,912	5,514,912
H264ref	3.800711	64,734,562	64,623,708	110,854	110,755
Omnetpp	3.443492	26,366,276	26,283,617	82,659	82,659
Astar	1.101332	44,523,105	42,950,790	1,572,315	1,572,315

NOMBRE	dl1.writebacks	ul2.accesses	ul2.hits	ul2.misses	ul2.replacements
Perlbench	127,446	945,826	872,389	73,437	73,437
Bzip2	4,347,812	13,043,449	10,869,515	2,173,934	2,173,934
Gcc	119,611	667,044	616,542	50,502	50,502
Mcf	42,333	84,669	66,822	17,847	17,847
Gobmk	74,607	378,984	338,209	40,775	40,775
Hmmer	443,480	950,169	838,527	111,642	111,642
Sjeng	41,473	110,758	81,101	29,657	29,657
Libquantum	4,878,549	10,393,632	9,014,801	1,378,831	1,378,831
H264ref	66,285	188,921	187,603	1,318	1,155
Omnetpp	45,374	161,546	126,257	35,289	35,289
Astar	631,928	2,204,244	2,006,038	198,206	198,206

NOMBRE	ul2.writebacks	ul3.accesses	ul3.hits	ul3.misses	ul3.replacements
Perlbench	16,358	89,795	84,075	5,720	5,715
Bzip2	1,087,032	3,260,966	1,087,032	2,173,934	2,173,934
Gcc	26,148	76,650	52,792	23,858	23,858
Mcf	15,892	33,739	24,685	9,054	121
Gobmk	21,465	62,240	48,401	13,839	13,839
Hmmer	105,474	217,116	216,386	730	0
Sjeng	17,251	46,908	26,754	20,154	11,028
Libquantum	1,218,118	2,596,949	1,218,118	1,378,831	1,378,831
H264ref	249	1,567	1,253	314	0
Omnetpp	13,225	48,514	38,182	10,332	0
Astar	110,189	308,395	308,332	63	0

NOMBRE	ul3.writebacks	aciertos_dc	fallos_dc	total_accesos_dc
Perlbench	5,632	20,285,884	24,061,949	44,347,833
Bzip2	1,086,589	49,999,619	21,739,212	71,738,831
Gcc	22,304	16,548,143	20,669,864	37,218,007
Mcf	105	20,187,231	13,143,241	33,330,472
Gobmk	13,776	13,902,534	26,604,989	40,507,523
Hmmer	0	13,249,594	26,669,458	39,919,052
Sjeng	6,724	12,016,603	21,675,597	33,692,200
Libquantum	1,187,398	12,051,843	13,496,760	25,548,603
H264ref	0	42,337,923	22,396,639	64,734,562
Omnetpp	0	9,120,981	17,245,295	26,366,276
Astar	0	21,252,655	23,270,450	44,523,105

NOMBRE	aciertos_ac	fallos_ac	total_accesos_ac
Perlbench	22,556	24,039,393	24,061,949
Bzip2	8,695,565	13,043,647	21,739,212
Gcc	106,681	20,563,183	20,669,864
Mcf	22,136	13,121,105	13,143,241
Gobmk	587,285	26,017,704	26,604,989
Hmmer	15,633,966	11,035,492	26,669,458
Sjeng	37,138	21,638,459	21,675,597
Libquantum	10,267,726	3,229,034	13,496,760
H264ref	6,216,449	16,180,190	22,396,639
Omnetpp	1,847,476	15,397,819	17,245,295
Astar	10,761,183	12,509,267	23,270,450

E.7.2. Secuencial, 100 tramos.

Tabla E.10: Tabla que muestra los resultados correspondientes a la suite float, en modo secuencial y con 100 tramos de ficheros de análisis.

NOMBRE	Modo	sim_total_issued_loads	sim_total_issued_stores
Perlbench	Secuencial	36,062,348	14,620,282
Bzip2	Secuencial	58,695,561	17,391,281
Gcc	Secuencial	32,133,691	10,736,938
Mcf	Secuencial	26,809,435	9,489,531
Gobmk	Secuencial	32,944,660	11,773,585
Hmmer	Secuencial	39,469,486	9,310,333
Sjeng	Secuencial	29,081,646	6,745,081
Libquantum	Secuencial	29,278,530	6,977,941
H264ref	Secuencial	49,339,691	16,390,765
Omnetpp	Secuencial	23,954,618	6,727,273
Astar	Secuencial	44,158,152	7,126,296

NOMBRE	sim_total_commited_loads	sim_total_commited_stores	sim_num_refs
Perlbench	32,075,588	12,600,176	44,675,773
Bzip2	54,347,500	17,391,239	71,738,830
Gcc	28,515,273	9,681,488	38,196,768
Mcf	25,780,211	9,083,897	34,864,123
Gobmk	27,076,634	10,450,938	37,527,589
Hmmer	32,198,719	7,310,067	39,508,799
Sjeng	24,750,238	6,208,686	30,958,933
Libquantum	18,570,642	6,977,899	25,548,548
H264ref	48,203,947	16,097,809	64,301,844
Omnetpp	21,880,320	6,468,517	28,348,853
Astar	23,853,369	3,518,019	27,371,393

NOMBRE	sim_num_loads	sim_num_stores	sim_elapsed_time
Perlbench	32,075,593	12,600,180	214
Bzip2	54,347,567	17,391,263	485
Gcc	28,515,279	9,681,489	200
Mcf	25,780,220	9,083,903	165
Gobmk	27,076,645	10,450,944	221
Hmmer	32,198,731	7,310,068	212
Sjeng	24,750,246	6,208,687	198
Libquantum	18,570,646	6,977,902	565
H264ref	48,204,012	16,097,832	198
Omnetpp	21,880,328	6,468,525	155
Astar	23,853,373	3,518,020	339

NOMBRE	sim_total_refs	sim_total_loads	sim_total_stores	sim_cycle
Perlbench	57,466,108	41,412,883	16,053,225	44,975,271
Bzip2	71,739,022	54,347,714	17,391,308	122,284,861
Gcc	48,744,701	36,347,802	12,396,899	45,490,895
Mcf	37,561,769	27,817,245	9,744,524	31,017,969
Gobmk	54,813,865	40,500,741	14,313,124	51,450,303
Hmmer	58,551,670	46,631,132	11,920,538	45,309,241
Sjeng	43,043,544	35,259,346	7,784,198	46,123,141
Libquantum	25,548,807	18,570,827	6,977,980	253,258,110
H264ref	68,356,147	51,262,177	17,093,970	26,564,298
Omnetpp	33,472,135	26,347,039	7,125,096	29,642,386
Astar	69,152,811	60,029,822	9,122,989	91,444,168

NOMBRE	sim_IPC	dl1.accesses	dl1.hits	dl1.misses	dl1.replacements
Perlbench	2.22344	36,614,152	36,029,887	584,265	584,265
Bzip2	0.81776	39,130,447	30,434,813	8,695,634	8,695,634
Gcc	2.19824	29,690,547	29,532,952	157,595	157,595
Mcf	3.22394	21,683,454	21,641,130	42,324	42,324
Gobmk	1.94362	36,113,838	36,005,531	108,307	108,307
Hmmer	2.20706	33,316,629	32,809,939	506,690	506,690
Sjeng	2.16811	27,172,522	27,113,454	59,068	59,068
Libquantum	0.39485	20,474,660	14,959,747	5,514,913	5,514,913
H264ref	3.76445	38,340,746	38,230,830	109,916	109,817
Omnetpp	3.37355	23,403,569	23,320,400	83,169	83,169
Astar	1.09356	26,046,093	24,474,546	1,571,547	1,571,547

NOMBRE	dl1.writebacks	ul2.accesses	ul2.hits	ul2.misses	ul2.replacements
Perlbench	127,712	944,631	871,276	73,355	73,355
Bzip2	4,347,813	13,043,451	10,869,517	2,173,934	2,173,934
Gcc	119,630	670,331	619,764	50,567	50,567
Mcf	42,316	84,642	66,770	17,872	17,872
Gobmk	74,578	378,386	337,717	40,669	40,669
Hmmer	443,502	950,205	838,566	111,639	111,639
Sjeng	41,228	110,172	80,517	29,655	29,655
Libquantum	4,878,549	10,393,633	9,014,802	1,378,831	1,378,831
H264ref	65,399	187,101	185,780	1,321	1,158
Omnetpp	45,472	161,890	126,485	35,405	35,405
Astar	631,745	2,203,293	2,004,902	198,391	198,391

NOMBRE	ul2.writebacks	ul3.accesses	ul3.hits	ul3.misses	ul3.replacements
Perlbench	16,276	89,631	83,914	5,717	5,712
Bzip2	1,087,032	3,260,966	1,087,032	2,173,934	2,173,934
Gcc	26,169	76,736	52,882	23,854	23,854
Mcf	15,896	33,768	24,714	9,054	121
Gobmk	21,467	62,136	48,377	13,759	13,759
Hmmer	105,469	217,108	216,377	731	0
Sjeng	17,242	46,897	26,744	20,153	11,028
Libquantum	1,218,118	2,596,949	1,218,118	1,378,831	1,378,831
H264ref	249	1,570	1,256	314	0
Omnetpp	13,249	48,654	38,324	10,330	0
Astar	110,243	308,634	308,573	61	0

NOMBRE	ul3.writebacks	aciertos_dc	fallos_dc	total_accesos_dc
Perlbench	5,629	20,055,129	24,026,402	44,081,531
Bzip2	1,086,589	49,999,619	21,739,208	71,738,827
Gcc	22,301	17,028,352	20,075,056	37,103,408
Mcf	105	19,869,771	13,326,886	33,196,657
Gobmk	13,694	14,121,953	26,229,613	40,351,566
Hmmer	0	13,337,123	26,355,590	39,692,713
Sjeng	6,723	12,471,200	20,967,085	33,438,285
Libquantum	1,187,398	12,051,842	13,496,764	25,548,606
H264ref	0	42,434,714	22,256,488	64,691,202
Omnetpp	0	9,410,268	16,937,451	26,347,719
Astar	0	21,839,837	22,564,396	44,404,233

NOMBRE	aciertos_ac	fallos_ac	total_accesos_ac
Perlbench	22,372	24,004,030	24,026,402
Bzip2	8,695,565	13,043,643	21,739,208
Gcc	107,412	19,967,644	20,075,056
Mcf	295	13,326,591	13,326,886
Gobmk	555,602	25,674,011	26,229,613
Hmmer	15,743,068	10,612,522	26,355,590
Sjeng	30,652	20,936,433	20,967,085
Libquantum	10,267,727	3,229,037	13,496,764
H264ref	6,211,764	16,044,724	22,256,488
Omnetpp	1,827,344	15,110,107	16,937,451
Astar	10,412,392	12,152,004	22,564,396

E.8. Resultados con 100 ficheros de análisis. Suite float.

E.8.1. Paralelo, 100 tramos.

Tabla E.11: Tabla que muestra los resultados de la suite float para el modo paralelo y con 100 tramos de fichero de análisis.

Nombre	Modo	total_issued_loads	total_issued_stores	total_commited_loads
Bwaves	Paralelo	30,944,343	6,965,314	29,923,965
Gamess	Paralelo	25,233,153	12,996,461	24,345,014
Milc	Paralelo	35,923,441	18,754,312	35,913,822
Zeusmp	Paralelo	29,627,444	11,762,865	28,070,343
Gromacs	Paralelo	44,689,403	9,087,487	36,938,169
CactusADM	Paralelo	38,266,936	7,744,186	36,129,566
Leslie3d	Paralelo	27,216,449	2,603,279	26,234,970
Namd	Paralelo	28,642,538	9,379,226	27,218,280
DealII	Paralelo	18,107,665	2,998,063	16,337,412
Soplex	Paralelo	19,599,529	105,524	15,800,342
Povray	Paralelo	38,481,383	11,503,461	35,534,327
Calculix	Paralelo	25,710,696	14,513,593	24,496,664
GemsFDTD	Paralelo	22,969,694	11,525,313	22,330,291
Tonto	Paralelo	21,737,353	8,671,290	21,063,627
Lbm	Paralelo	34,188,448	5,375,895	28,757,967
Wrf	Paralelo	27,290,071	10,637,614	26,309,667
Sphin3	Paralelo	28,698,221	10,657,829	26,846,836

Nombre	total_commited_stores	num_refs	num_loads	num_stores
Bwaves	6,965,171	36,889,164	29,923,990	6,965,174
Gamess	12,332,225	36,677,262	24,345,029	12,332,233
Milc	18,754,080	54,668,038	35,913,910	18,754,128
Zeusmp	11,760,980	39,831,329	28,070,347	11,760,982
Gromacs	8,654,623	45,592,814	36,938,183	8,654,631
CactusADM	7,738,762	43,868,356	36,129,588	7,738,768
Leslie3d	2,603,273	28,838,274	26,234,997	2,603,277
Namd	8,884,583	36,102,879	27,218,289	8,884,590
DealII	2,955,000	19,292,420	16,337,420	2,955,000
Soplex	102,375	15,902,720	15,800,345	102,375
Povray	10,392,424	45,926,784	35,534,350	10,392,434
Calculix	13,723,752	38,220,425	24,496,667	13,723,758
GemsFDTD	11,197,269	33,527,566	22,330,297	11,197,269
Tonto	8,463,067	29,526,700	21,063,633	8,463,067
Lbm	5,375,259	34,133,239	28,757,980	5,375,259
Wrf	10,299,486	36,609,171	26,309,676	10,299,495
Sphin3	9,871,312	36,718,169	26,846,839	9,871,330

Nombre	elapsed_time	total_refs	total_loads	total_stores
Bwaves	284	36,947,826	29,963,135	6,984,691
Gamess	154	40,444,103	26,537,881	13,906,222
Milc	242	54,703,421	35,948,480	18,754,941
Zeusmp	276	39,938,876	28,173,424	11,765,452
Gromacs	246	57,176,311	47,612,380	9,563,931
CactusADM	530	43,993,173	36,227,265	7,765,908
Leslie3d	247	29,022,536	26,419,008	2,603,528
Namd	161	39,471,411	29,863,373	9,608,038
DealII	182	20,860,617	17,725,548	3,135,069
Soplex	318	17,990,332	17,880,976	109,356
Povray	213	55,452,934	42,575,661	12,877,273
Calculix	170	43,707,487	27,814,023	15,893,464
GemsFDTD	160	35,707,177	23,712,402	11,994,775
Tonto	154	32,223,172	22,911,521	9,311,651
Lbm	309	34,208,652	28,832,177	5,376,475
Wrf	193	38,309,755	27,376,857	10,932,898
Sphin3	167	42,158,111	30,656,574	11,501,537

Nombre	sim_cycle	IPC	dl1.accesses	dl1.hits
Bwaves	86,700,811	1.1534	33,257,046	32,575,127
Gamess	26,291,309	3.8035	31,681,780	31,681,717
Milc	46,985,707	2.1283	54,667,547	50,022,930
Zeusmp	95,979,498	1.0419	39,820,610	38,209,886
Gromacs	61,396,757	1.6288	48,290,231	47,191,353
CactusADM	187,793,984	0.5325	31,003,295	28,370,615
Leslie3d	85,877,158	1.1645	28,840,153	27,419,023
Namd	30,327,688	3.2973	34,825,725	34,811,594
DealII	51,538,480	1.9403	18,965,810	17,634,589
Soplex	117,730,671	0.8494	16,089,207	13,541,579
Povray	42,969,735	2.3272	43,805,716	43,692,798
Calculix	29,327,984	3.4097	36,187,919	36,187,558
GemsFDTD	27,873,050	3.5877	31,495,808	31,495,798
Tonto	26,841,677	3.7256	28,300,503	28,299,683
Lbm	97,970,594	1.0207	34,195,868	28,818,366
Wrf	46,461,733	2.1523	31,780,708	30,574,947
Sphin3	30,884,764	3.2378	36,071,649	36,049,714

Nombre	dl1.misses	dl1.replacements	dl1.writebacks	ul2.accesses
Bwaves	681,919	681,919	323,065	1,005,161
Gamess	63	62	41	169
Milc	4,644,617	4,644,617	4,644,299	9,288,975
Zeusmp	1,610,724	1,610,724	87,787	1,699,867
Gromacs	1,098,878	1,098,878	211,319	1,310,473
CactusADM	2,632,680	2,632,680	2,579,534	5,212,214
Leslie3d	1,421,130	1,421,130	560,534	1,981,714
Namd	14,131	14,131	14,131	28,271
DealII	1,331,221	1,331,221	37,045	1,368,266
Soplex	2,547,628	2,547,628	10,678	2,558,309
Povray	112,918	112,733	18,493	137,599
Calculix	361	358	152	604
GemsFDTD	10	0	0	46,140
Tonto	820	820	269	30,669
Lbm	5,377,502	5,377,502	3,962,904	9,340,406
Wrf	1,205,761	1,205,761	1,143,473	2,351,209
Sphin3	21,935	21,935	10,462	32,425

Nombre	ul2.hits	ul2.misses	ul2.replacements	ul2.writebacks
Bwaves	834,657	170,504	170,504	79,298
Gamess	145	24	8	2
Milc	8,127,584	1,161,391	1,161,391	1,161,177
Zeusmp	1,306,091	393,776	393,776	22,798
Gromacs	1,217,345	93,128	93,128	29,969
CactusADM	2,344,085	2,868,129	2,868,129	2,579,492
Leslie3d	1,621,680	360,034	360,034	143,126
Namd	24,731	3,540	3,540	3,532
DealII	1,359,245	9,021	9,021	7,808
Soplex	247,342	2,310,967	2,310,967	5,470
Povray	137,545	54	0	0
Calculix	567	37	8	0
GemsFDTD	46,131	9	0	0
Tonto	30,645	24	18	5
Lbm	7,925,807	1,414,599	1,414,599	1,060,908
Wrf	1,859,024	492,185	492,185	392,735
Sphin3	26,942	5,483	5,483	2,722

Nombre	ul3.accesses	ul3.hits	ul3.misses	ul3.replacements	ul3.writebacks
Bwaves	249,802	79,563	170,239	170,239	80,510
Gamess	26	2	24	0	0
Milc	2,322,568	1,161,517	1,161,051	1,161,051	1,161,002
Zeusmp	416,574	103,260	313,314	313,314	30,635
Gromacs	123,097	105,354	17,743	4	4
CactusADM	5,447,621	2,865,218	2,582,403	2,582,403	2,579,482
Leslie3d	503,160	143,226	359,934	359,934	153,496
Namd	7,072	3,536	3,536	0	0
DealII	16,829	16,829	0	0	0
Soplex	2,316,437	2,316,096	341	0	0
Povray	54	0	54	0	0
Calculix	37	0	37	0	0
GemsFDTD	9	0	9	0	0
Tonto	29	11	18	0	0
Lbm	2,475,507	1,060,909	1,414,598	1,414,598	1,060,907
Wrf	884,920	534,574	350,346	350,346	338,174
Sphin3	8,205	2,826	5,379	0	0

Nombre	aciertos_dc	fallos_dc	total_accesos_dc	aciertos_ac	fallos_ac	total_accesos_ac
Bwaves	19,377,489	13,879,557	33,257,046	198	13,879,359	13,879,557
Gamess	22,854,454	8,827,326	31,681,780	569,534	8,257,792	8,827,326
Milc	52,933,879	1,733,668	54,667,547	315,445	1,418,223	1,733,668
Zeusmp	16,922,239	22,898,371	39,820,610	1,680,859	21,217,512	22,898,371
Gromacs	20,638,535	27,651,696	48,290,231	863,346	26,788,350	27,651,696
CactusADM	18,067,756	12,935,539	31,003,295	852,349	12,083,190	12,935,539
Leslie3d	20,316,646	8,523,507	28,840,153	4,049,278	4,474,229	8,523,507
Namd	22,208,303	12,617,422	34,825,725	53,420	12,564,002	12,617,422
DealII	4,767,859	14,197,951	18,965,810	5,044,640	9,153,311	14,197,951
Soplex	8,617,641	7,471,566	16,089,207	1,384,385	6,087,181	7,471,566
Povray	14,511,437	29,294,279	43,805,716	68,603	29,225,676	29,294,279
Calculix	23,684,050	12,503,869	36,187,919	230,774	12,273,095	12,503,869
GemsFDTD	19,197,651	12,298,157	31,495,808	28,337	12,269,820	12,298,157
Tonto	12,781,726	15,518,777	28,300,503	76,800	15,441,977	15,518,777
Lbm	18,040,772	16,155,096	34,195,868	3,877,742	12,277,354	16,155,096
Wrf	25,119,321	6,661,387	31,780,708	466,353	6,195,034	6,661,387
Sphin3	14,413,742	21,657,907	36,071,649	13,962	21,643,945	21,657,907

E.8.2. Secuencial, 100 tramos.

Tabla E.12: Tabla que muestra los resultados de la suite float, en el modo secuencial y con 100 ficheros de análisis.

Nombre	Modo	total_issued_loads	total_issued_stores	total_commited_loads
Bwaves	Secuencial	30,944,355	6,965,314	29,923,965
Gamess	Secuencial	25,210,880	12,962,599	24,345,014
Milc	Secuencial	35,923,530	18,754,325	35,913,822
Zeusmp	Secuencial	29,627,375	11,762,792	28,070,343
Gromacs	Secuencial	44,602,956	9,080,001	36,938,169
CactusADM	Secuencial	38,266,404	7,744,186	36,129,566
Leslie3d	Secuencial	27,216,697	2,603,279	26,234,970
Namd	Secuencial	28,529,301	9,310,977	27,218,279
DealII	Secuencial	18,136,468	2,998,291	16,337,412
Soplex	Secuencial	19,593,722	105,219	15,800,342
Povray	Secuencial	38,439,829	11,488,788	35,534,327
Calculix	Secuencial	25,606,492	14,504,350	24,496,664
GemsFDTD	Secuencial	22,976,186	11,515,180	22,330,291
Tonto	Secuencial	21,682,693	8,653,675	21,063,627
Lbm	Secuencial	34,188,773	5,375,918	28,757,967
Wrf	Secuencial	27,283,718	10,637,598	26,309,667
Sphin3	Secuencial	28,650,021	10,553,476	26,846,836

Nombre	total_commited_stores	num_refs	num_loads	num_stores
Bwaves	6,965,171	36,889,164	29,923,990	6,965,174
Gamess	12,332,225	36,677,260	24,345,027	12,332,233
Milc	18,754,080	54,668,038	35,913,910	18,754,128
Zeusmp	11,760,980	39,831,329	28,070,347	11,760,982
Gromacs	8,654,623	45,592,814	36,938,183	8,654,631
CactusADM	7,738,762	43,868,356	36,129,588	7,738,768
Leslie3d	2,603,273	28,838,274	26,234,997	2,603,277
Namd	8,884,581	36,102,879	27,218,289	8,884,590
DealII	2,955,000	19,292,420	16,337,420	2,955,000
Soplex	102,375	15,902,720	15,800,345	102,375
Povray	10,392,424	45,926,784	35,534,350	10,392,434
Calculix	13,723,752	38,220,425	24,496,667	13,723,758
GemsFDTD	11,197,269	33,527,566	22,330,297	11,197,269
Tonto	8,463,067	29,526,701	21,063,634	8,463,067
Lbm	5,375,259	34,133,239	28,757,980	5,375,259
Wrf	10,299,486	36,609,171	26,309,676	10,299,495
Sphin3	9,871,314	36,718,177	26,846,845	9,871,332

Nombre	elapsed_time	total_refs	total_loads	total_stores
Bwaves	286	36,947,824	29,963,133	6,984,691
Gamess	154	40,483,366	26,574,886	13,908,480
Milc	241	54,703,328	35,948,427	18,754,901
Zeusmp	277	39,938,935	28,173,474	11,765,461
Gromacs	246	56,648,481	47,108,585	9,539,896
CactusADM	531	43,993,173	36,227,265	7,765,908
Leslie3d	246	29,023,782	26,419,885	2,603,897
Namd	162	39,516,660	29,919,886	9,596,774
DealII	185	20,924,570	17,774,713	3,149,857
Soplex	319	17,967,578	17,858,944	108,634
Povray	215	55,208,374	42,422,494	12,785,880
Calculix	171	43,741,079	27,830,655	15,910,424
GemsFDTD	162	35,819,405	23,778,747	12,040,658
Tonto	156	32,051,683	22,799,337	9,252,346
Lbm	310	34,208,755	28,832,242	5,376,513
Wrf	193	38,281,643	27,362,067	10,919,576
Sphin3	169	41,600,431	30,452,917	11,147,514

Nombre	sim_cycle	IPC	dl1.accesses	dl1.hits
Bwaves	86,720,409	1.1531	20,844,721	20,162,787
Gamess	26,676,515	3.7486	20,291,705	20,291,642
Milc	46,983,743	2.1284	20,349,087	15,704,454
Zeusmp	96,550,087	1.0357	32,378,202	30,767,471
Gromacs	62,015,431	1.6125	35,983,743	34,884,973
CactusADM	187,794,374	0.5325	19,560,954	16,928,450
Leslie3d	85,917,194	1.1639	9,707,311	8,285,966
Namd	31,119,527	3.2134	21,116,430	21,102,299
DealII	51,591,120	1.9383	17,246,424	15,914,887
Soplex	118,513,587	0.8438	7,677,338	5,129,138
Povray	43,190,432	2.3153	38,633,214	38,520,296
Calculix	30,178,943	3.3136	26,170,122	26,169,762
GemsFDTD	28,705,879	3.4836	22,308,246	22,308,238
Tonto	27,706,756	3.6092	22,858,636	22,857,816
Lbm	97,973,478	1.0207	16,158,090	10,782,014
Wrf	46,615,073	2.1452	16,912,836	15,707,086
Sphin3	31,610,970	3.1635	31,465,724	31,443,811

Nombre	dl1.misses	dl1.replacements	dl1.writebacks	ul2.accesses
Bwaves	681,934	323,065	1,005,176	834,662
Gamess	63	42	171	146
Milc	4,644,633	4,644,315	9,289,007	8,127,616
Zeusmp	1,610,731	87,793	1,699,885	1,306,102
Gromacs	1,098,770	211,474	1,310,520	1,216,476
CactusADM	2,632,504	2,579,532	5,212,036	2,342,049
Leslie3d	1,421,345	560,553	1,981,948	1,621,767
Namd	14,131	14,128	28,268	24,728
DealII	1,331,537	37,047	1,368,584	1,359,570
Soplex	2,548,200	10,954	2,559,156	248,290
Povray	112,918	18,493	137,600	137,546
Calculix	360	152	603	567
GemsFDTD	8	0	46,136	46,128
Tonto	820	269	30,050	30,034
Lbm	5,376,076	3,961,484	9,337,560	7,922,968
Wrf	1,205,750	1,143,483	2,351,209	1,859,026
Sphin3	21,913	10,459	32,400	26,917

Nombre	ul2.hits	ul2.misses	ul2.replacements	ul2.writebacks
Bwaves	834,657	170,514	79,298	249,812
Gamess	145	8	2	27
Milc	8,127,584	1,161,391	1,161,178	2,322,569
Zeusmp	1,306,091	393,783	22,805	416,588
Gromacs	1,217,345	94,044	29,988	124,032
CactusADM	2,344,085	2,869,987	2,579,493	5,449,480
Leslie3d	1,621,680	360,181	143,145	503,326
Namd	24,731	3,540	3,532	7,072
DealII	1,359,245	9,014	7,802	16,816
Soplex	247,342	2,310,866	5,324	2,316,190
Povray	137,545	0	0	54
Calculix	567	8	0	36
GemsFDTD	46,131	0	0	8
Tonto	30,645	12	3	19
Lbm	7,925,807	1,414,592	1,060,908	2,475,500
Wrf	1,859,024	492,183	392,743	884,926
Sphin3	26,942	5,483	2,724	8,207

Nombre	ul3.accesses	ul3.hits	ul3.misses	ul3.replacements
Bwaves	79,571	170,241	170,241	80,510
Gamess	2	25	0	0
Milc	1,161,518	1,161,051	1,161,051	1,161,002
Zeusmp	103,274	313,314	313,314	30,635
Gromacs	106,288	17,744	4	4
CactusADM	2,867,074	2,582,406	2,582,406	2,579,486
Leslie3d	143,387	359,939	359,939	153,496
Namd	3,536	3,536	0	0
DealII	16,816	0	0	0
Soplex	2,315,849	341	0	0
Povray	0	54	0	0
Calculix	0	36	0	0
GemsFDTD	0	8	0	0
Tonto	10	9	0	0
Lbm	1,060,938	1,414,562	1,414,562	1,060,907
Wrf	534,579	350,347	350,347	338,174
Sphin3	2,831	5,376	0	0

Nombre	aciertos_dc	fallos_dc	total_accesos_dc	aciertos_ac	fallos_ac	total_accesos_ac
Bwaves	19,377,478	13,879,550	33,257,028	207	13,879,343	13,879,550
Gamess	23,649,721	7,959,494	31,609,215	112,198	7,847,296	7,959,494
Milc	53,072,565	1,595,008	54,667,573	87	1,594,921	1,595,008
Zeusmp	16,927,156	22,893,322	39,820,478	1,680,833	21,212,489	22,893,322
Gromacs	20,716,230	27,474,584	48,190,814	852,552	26,622,032	27,474,584
CactusADM	19,181,103	11,822,192	31,003,295	620,320	11,201,872	11,822,192
Leslie3d	21,671,604	7,168,550	28,840,154	3,433,498	3,735,052	7,168,550
Namd	21,424,384	13,298,451	34,722,835	299	13,298,152	13,298,451
DealII	4,779,963	14,291,498	19,071,461	5,044,028	9,247,470	14,291,498
Soplex	8,515,597	7,575,025	16,090,622	1,384,585	6,190,440	7,575,025
Povray	15,370,434	28,322,347	43,692,781	49,677	28,272,670	28,322,347
Calculix	23,523,844	12,447,542	35,971,386	223,374	12,224,168	12,447,542
GemsFDTD	20,544,971	11,116,102	31,661,073	16,649	11,099,453	11,116,102
Tonto	13,871,652	14,395,646	28,267,298	9,493	14,386,153	14,395,646
Lbm	18,039,105	16,156,930	34,196,035	3,877,735	12,279,195	16,156,930
Wrf	25,167,281	6,613,369	31,780,650	422,371	6,190,998	6,613,369
Sphin3	14,434,589	21,600,433	36,035,022	13,249	21,587,184	21,600,433

E.8.3. Comparativas 1 vs 100 tramos

En esta sección se muestran las tablas comparativas con las simulaciones hechas con 1 tramo vs 100 tramos de análisis. También se muestran las comparación de los ciclos de ejecución de cada benchmark.

NOMBRE	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
Perlbench	1.76	2.28	2.28	2.22	2.22	52.40%	67.98%	41.15%	45.74%	42.84%	45.50%	3.33	2.74	2.67	2.43	2.33
Bzip2	0.81	0.83	0.83	0.82	0.82	85.16%	83.25%	69.70%	69.70%	69.70%	69.70%	44.76	44.63	27.31	41.21	23.77
Gcc	1.79	2.25	2.25	2.20	2.20	46.48%	67.13%	41.99%	44.46%	43.87%	45.89%	2.20	1.89	2.35	1.56	2.03
Mcf	2.43	3.34	3.34	3.23	3.22	61.99%	94.94%	54.81%	60.57%	57.51%	59.85%	1.31	1.04	1.29	0.75	0.98
Gobmk	1.63	1.99	1.99	1.94	1.94	32.99%	59.56%	32.49%	34.32%	32.96%	35.00%	2.45	2.21	2.88	1.99	2.64
Hmmer	1.86	2.24	2.24	2.21	2.21	39.84%	81.90%	30.23%	33.19%	30.57%	33.60%	2.88	2.56	2.62	2.27	2.28
Sjeng	1.81	2.24	2.24	2.17	2.17	74.21%	96.37%	28.24%	35.67%	31.43%	37.30%	1.77	1.59	2.16	1.35	1.89
Libquantum	0.38	0.39	0.40	0.39	0.39	33.03%	35.72%	47.17%	47.17%	47.17%	47.17%	59.62	58.19	48.61	57.43	47.37
H264ref	3.49	3.80	3.80	3.76	3.76	52.04%	89.61%	25.34%	65.40%	25.38%	65.60%	1.75	1.78	1.92	1.27	1.25
Omnetpp	2.51	3.44	3.44	3.37	3.37	27.69%	81.86%	33.24%	34.59%	34.44%	35.72%	1.04	0.84	1.05	0.74	0.95
Astar	0.91	1.10	1.10	1.09	1.09	92.07%	264.45%	35.86%	47.73%	36.51%	49.18%	9.91	8.64	7.47	7.40	5.83
Media	1.24	1.41	1.41	1.39	1.40	54.35%	92.98%	40.02%	47.14%	41.13%	47.68%	11.91	11.46	9.12	10.76	8.30

TABLA E.13: Tabla que muestra la comparativa de los datos de rendimiento, tasas de aciertos en DC y consumo de 1 tramos vs 100 tramos para la suite enteros.

NOMBRE	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
Bwaves	1.13	1.15	1.15	1.15	1.15	62.17%	97.99%	32.03%	58.27%	32.03%	58.27%	6.03	6.15	5.67	2.69	2.69
Gamess	2.62	3.80	3.80	3.64	3.75	88.89%	98.48%	49.68%	72.14%	49.80%	74.82%	1.56	1.06	0.99	0.25	0.26
Milc	2.11	2.13	2.13	2.13	2.13	81.36%	90.65%	91.56%	96.83%	91.58%	97.08%	6.79	6.97	5.64	0.84	0.84
Zeusmp	1.00	1.04	1.04	1.04	1.04	72.92%	99.54%	32.96%	42.50%	35.25%	42.51%	8.32	8.31	7.94	3.30	3.34
Gromacs	1.43	1.63	1.63	1.61	1.61	38.09%	79.30%	31.11%	42.74%	31.13%	42.99%	5.00	4.52	4.03	1.36	1.39
CactusADM	0.51	0.53	0.53	0.53	0.53	88.25%	93.53%	52.18%	58.28%	52.18%	61.87%	44.32	42.49	40.80	12.76	12.76
Leslie3d	1.08	1.16	1.16	1.16	1.16	72.98%	94.85%	61.15%	70.45%	66.59%	75.14%	6.92	6.48	5.13	2.64	2.64
Namd	2.39	3.30	3.30	3.21	3.21	68.61%	99.89%	55.83%	63.77%	59.39%	61.70%	1.90	1.36	1.03	0.33	0.35
DealII	1.66	1.94	1.94	1.94	1.94	36.10%	78.53%	23.68%	25.14%	23.59%	25.06%	2.46	2.07	1.94	0.95	0.95
Soplex	0.77	0.84	0.85	0.83	0.84	95.77%	90.49%	49.64%	53.56%	36.91%	52.92%	11.93	10.58	10.04	4.95	5.01
Povray	1.97	2.33	2.33	2.32	2.32	68.54%	92.42%	32.23%	33.13%	34.42%	35.18%	2.90	2.50	2.21	0.67	0.68
Calculix	2.47	3.41	3.41	3.32	3.31	49.22%	96.44%	65.17%	65.45%	66.09%	65.40%	1.85	1.34	1.07	0.31	0.33
GemsFDTD	2.81	3.59	3.59	3.48	3.48	78.19%	88.03%	61.01%	60.95%	59.45%	64.89%	1.42	1.13	0.95	0.28	0.30
Tonto	2.91	3.73	3.73	3.58	3.61	61.06%	97.84%	36.14%	45.16%	37.83%	49.07%	1.25	0.99	0.94	0.26	0.28
Lbm	1.00	1.02	1.02	1.02	1.02	77.43%	96.99%	53.30%	52.76%	53.30%	52.75%	14.75	14.76	13.01	3.50	3.50
Wrf	1.99	2.15	2.15	2.10	2.15	51.84%	3.91%	40.00%	79.04%	40.00%	79.19%	3.51	3.31	3.22	0.79	0.79
Sphin3	2.48	3.24	3.24	3.16	3.16	57.56%	85.05%	39.92%	39.96%	40.02%	40.06%	1.83	1.43	1.27	0.35	0.36
Media	1.40	1.55	1.56	1.54	1.55	67.59%	87.29%	47.50%	56.48%	47.62%	57.58%	7.22	6.79	6.23	2.13	2.15

TABLA E.14: Tabla que muestra la comparativa de los datos de rendimiento, tasas de aciertos en DC y consumo de 1 tramos vs 100 tramos para la suite float.

A: IPC en modo solo
B: IPC en modo paralelo con 1 tramo
C: IPC en modo paralelo con 100 tramos
D: IPC en modo secuencial con 1 tramo
E: IPC en modo secuencial con 100 tramos
F: Tasa de aciertos esperados en DC con 1 tramo
G: Tasa de aciertos esperados en DC con 100 tramos
H: Tasa de aciertos obtenidos en DC en paralelo con 1 tramos
I: Tasa de aciertos obtenidos en DC en paralelo con 100 tramos
J: Tasa de aciertos obtenidos en DC en secuencial con 1 tramo
K: Tasa de aciertos obtenidos en DC en secuencial con 100 tramos
L: Indicador energy_delay solo
M: Indicador energy_delay en paralelo con 1 tramo
N: Indicador energy_delay en paralelo con 100 tramos
O: Indicador energy_delay en secuencial con 1 tramo
P: Indicador energy_delay en secuencial con 100 tramos

Las columnas “F“ y “G“ de estas tablas muestran las tasas de aciertos esperados en la memoria DC. Estas tasa son el resultado del cociente entre los aciertos esperados en la DC obtenidos de la fase de análisis previo y el número de instrucciones lanzadas (para 1 tramo y para 100 tramos). Las columnas “H“, “I“, “J“ y “K“ muestran las tasas de aciertos que hemos obtenido en la memoria DC tanto en el modo secuencial como en el modo paralelo y para 1 fichero de análisis y 100 ficheros. Este valor es el cociente entre los aciertos obtenidos en la DC y el número de accesos a la memoria DC.

NOMBRE	SOLO	PARALELO	PARALELO100	SECUENCIAL	SECUENCIAL100
Perlbench	56,977,368	43,836,085	43,836,458	44,962,448	44,975,271
Bzip2	123,915,380	121,197,901	121,197,896	122,284,866	122,284,861
Gcc	55,773,994	44,424,141	44,423,434	45,486,503	45,490,895
Mcf	41,129,611	29,969,716	29,969,724	30,982,167	31,017,969
Gobmk	61,219,900	50,258,443	50,260,912	51,455,987	51,450,303
Hmmer	53,757,656	44,676,855	44,672,560	45,334,142	45,309,241
Sjeng	55,118,350	44,637,907	44,635,374	46,151,444	46,123,141
Libquantum	263,817,483	254,755,438	252,352,879	258,063,091	253,258,110
H264ref	28,678,481	26,340,662	26,310,865	26,563,269	26,564,298
Omnetpp	39,851,402	29,041,978	29,040,286	29,632,289	29,642,386
Astar	109,455,503	90,815,335	90,799,135	91,578,623	91,444,168
Media	80,881,375	70,904,951	70,681,775	72,044,984	71,596,422
Mejora		14.07%	14.43%	12.27%	12.97%
Diferencia			0.36%		0.70%

TABLA E.15: Tabla que muestra los ciclos de ejecución de la suite enteros en los tres modos de funcionamiento y con 1 vs 100 ficheros de análisis.

NOMBRE	SOLO	PARALELO	PARALELO100	SECUENCIAL	SECUENCIAL100
Bwaves	88,428,627	86,700,811	86,700,811	86,719,993	86,720,409
Gamess	38,173,252	26,291,278	26,291,309	27,487,470	26,676,515
Milc	47,495,539	47,001,590	46,985,707	47,001,904	46,983,743
Zeusmp	99,618,466	95,980,739	95,979,498	96,572,388	96,550,087
Gromacs	69,881,323	61,439,855	61,396,757	62,145,903	62,015,431
CactusADM	196,892,878	187,793,984	187,793,984	187,794,222	187,794,374
Leslie3d	92,778,148	85,877,021	85,877,158	85,915,223	85,917,194
Namd	41,879,602	30,327,680	30,327,688	31,130,367	31,119,527
DealII	60,188,848	51,558,609	51,538,480	51,648,405	51,591,120
Soplex	130,643,708	118,454,035	117,730,671	120,048,707	118,513,587
Povray	50,874,121	42,969,733	42,969,735	43,193,685	43,190,432
Calculix	40,515,597	29,312,176	29,327,984	30,140,362	30,178,943
GemsFDTD	35,528,758	27,873,035	27,873,050	28,717,489	28,705,879
Tonto	34,312,635	26,839,654	26,841,677	27,898,224	27,706,756
Lbm	99,906,761	97,967,439	97,970,594	97,969,987	97,973,478
Wrf	50,275,529	46,477,229	46,461,733	47,652,360	46,615,073
Sphin3	40,313,223	30,884,736	30,884,764	31,634,418	31,610,970
Media	71,629,824	64,338,212	64,291,271	64,921,830	64,697,854
Mejora		11.33%	11.41%	10.33%	10.71%
Diferencia			0.08%		0.38%

TABLA E.16: Tabla que muestra los ciclos de ejecución de la suite float en los tres modos de funcionamiento y con 1 vs 100 ficheros de análisis.

E.8.4. Tablas con otros resultados

En esta sección se muestran las tablas con los resultados obtenidos de las simulaciones con otras entradas (benchmarks modificados).

NOMBRE	SOLO	SOLO_M	DIFERENCIA	PARALELO	PARALELO_M	DIFERENCIA	SECUENCIAL	SECUENCIAL_M	DIFERENCIA
Sjeng	1.81	1.82	0.13%	2.24	2.22	-1.11%	2.17	2.17	0.00%
Perlbench	1.76	1.75	-0.56%	2.28	2.27	-0.64%	2.22	2.22	0.00%
H264ref	3.49	3.49	-0.01%	3.80	3.79	-0.27%	3.76	3.76	-0.01%
Libquantum	0.38	0.38	0.00%	0.39	0.39	-0.64%	0.39	0.39	0.00%
Mcf	2.43	2.44	0.34%	3.34	3.30	-1.04%	3.23	3.23	0.00%
Gobmk	1.63	1.64	0.41%	1.99	1.97	-0.78%	1.94	1.94	0.00%
Gcc	1.79	1.79	-0.10%	2.25	2.24	-0.57%	2.20	2.20	0.00%
Bzip2	0.81	0.81	0.00%	0.83	0.82	-0.45%	0.82	0.82	0.00%
Omnetpp	2.51	2.51	-0.15%	3.44	3.43	-0.26%	3.37	3.37	0.00%

TABLA E.17: Tabla que muestra la diferencia de rendimiento para la suite enteros con una entrada distinta para los tres modos de funcionamiento.

NOMBRE	SOLO	SOLO_M	DIFERENCIA	PARALELO	PARALELO_M	DIFERENCIA	SECUENCIAL	SECUENCIAL_M	DIFERENCIA
Bwaves	1.13	1.13	0.04%	1.15	1.15	-0.01%	1.15	1.15	0.00%
Gamess	2.62	2.62	0.00%	3.80	3.79	-0.24%	3.64	3.64	0.00%
Zeusmp	1.00	1.00	0.04%	1.04	1.04	-0.30%	1.04	1.04	0.00%
Gromacs	1.43	1.44	0.39%	1.63	1.62	-0.41%	1.61	1.61	0.00%
Namd	2.39	2.39	0.00%	3.30	3.28	-0.67%	3.21	3.21	0.00%
DealII	1.66	1.66	-0.01%	1.94	1.94	-0.07%	1.94	1.94	0.00%
Povray	1.97	1.97	0.00%	2.33	2.32	-0.18%	2.32	2.32	0.00%
Calculix	2.47	2.47	0.00%	3.41	3.39	-0.57%	3.32	3.32	0.00%
GemsFDTD	2.81	2.81	0.00%	3.59	3.57	-0.60%	3.48	3.48	0.00%
Tonto	2.91	2.91	0.00%	3.73	3.73	0.02%	3.58	3.58	0.00%
Lbm	1.00	1.00	0.00%	1.02	1.02	0.00%	1.02	1.02	0.00%
Wrf	1.99	1.99	0.03%	2.15	2.14	-0.46%	2.10	2.10	0.00%
Sphin3	2.48	2.48	0.00%	3.24	3.25	0.32%	3.16	3.16	0.00%

TABLA E.18: Tabla que muestra la diferencia de rendimiento para la suite float con una entrada distinta para los tres modos de funcionamiento.

Las columnas de las tablas E.17 y E.18 que terminan en _M muestra los valores que hemos obtenido en las simulaciones con distinta entrada.

Bibliografía

[1] J. Segarra, C. Rodriguez, R. Gran, L.C. Aparicio, and V. Vinals. A small and effective data cache for real-time multitasking systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2012 IEEE 18th*, pages 45–54, April 2012.

[2] SPEC CPU2006. Standard performance evaluation corporation, 2006. https://www.spec.org/cpu2006/.

[3] Todd Austin. Simplescalar llc, Junio 1997. http://www.simplescalar.com.

[4] Stream: Sustainable memory bandwidth in high performance computers. https://www.cs.virginia.edu/stream/.

[5] R. Gonzalez and M. Horowitz. Energy dissipation in general purpose processors. In *Low Power Electronics, 1995., IEEE Symposium on*, 1995.

[6] J. Lee and S. Kim. Filter data cache: An energy-efficient small l0 data cache architecture driven by miss cost reduction. *Computers, IEEE Transactions on*, PP(99):1–1, 2014.

[7] Ju Hee Choi, Jong Wook Kwak, Seong Tae Jhang, and Chu Shik Jhon. Data filter cache with word selection cache for low power embedded processor. In *Proceedings of the 2013 Research in Adaptive and Convergent Systems*, RACS '13, pages 422–427, New York, NY, USA, 2013. ACM.

[8] Young Jin Park, Hong Jun Choi, Cheol Hong Kim, and Jong-Myon Kim. Energy-aware filter cache architecture for multicore processors. In *Electronic Design, Test and Application, 2010. DELTA '10. Fifth IEEE International Symposium on*, pages 58–62, Jan 2010.

[9] S. Hines, D. Whalley, and G. Tyson. Guaranteeing hits to improve the efficiency of a small instruction cache. In *Microarchitecture, 2007. MICRO 2007. 40th Annual IEEE/ACM International Symposium on*, pages 433–444, Dec 2007.

[10] J. Kin, M. Gupta, and W.H. Mangione-Smith. Filtering memory references to increase energy efficiency. *Computers, IEEE Transactions on*, 49(1):1–15, Jan 2000.

[11] Mitchell Hayenga, Andrew Nere, and Mikko Lipasti. MadCache: A PC-aware Cache Insertion Policy. In Joel Emer, editor, *JWAC 2010 - 1st JILP Worshop on Computer Architecture Competitions: cache replacement Championship*, Saint Malo, France, June 2010.

[12] Cacti, an integrated cache and memory access time, cycle time, area, leakage, and dynamic power model. http://www.hpl.hp.com/research/cacti/.

[13] Gdb: The gnu project debugger. http://www.gnu.org/software/gdb/.

[14] The unix® system. http://www.unix.org/.

[15] Python's standard documentation. https://www.python.org/.

[16] Mary D. Brown, Jared Stark, and Yale N. Patt. Select-free instruction scheduling logic. In *Proceedings of the 34th Annual ACM/IEEE International Symposium on Microarchitecture*, MICRO 34, pages 204–213, Washington, DC, USA, 2001. IEEE Computer Society.

[17] I. Kim and M.H. Lipasti. Understanding scheduling replay schemes. In *Software, IEE Proceedings-*, pages 198–209, Feb 2004.

[18] John L. Henning. Spec cpu2006 benchmark descriptions. volume 34, pages 1–17, New York, NY, USA, September 2006. ACM.

[19] Jesús Alastruey Benedé. Renombre de registros especulativo, Octubre 2009. http://webdiis.unizar.es/gaz/biblio/pdfs/2009-tesis-alastruey.pdf.

[20] http://www.simplescalar.com/docs/simple$_t$*utorial*$_v$2.*pdf*.