Trabajo Fin de Máster

Caracterización en Memoria de la Suite de Benchmarks SPEC CPU2017

Memory Characterization of the SPEC CPU2017 Benchmark Suite

Autor

Autor Navarro Torres, Agustín

Director

Ibáñez Marín, Pablo

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2018

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

D./Dª. Agustín Navarro Torres,

con nº de DNI 25199418C en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Máster, (Título del Trabajo) Caracterización en Memoria de la Suite de Benchmarks SPEC CPU2017 (Memory Characterization of the SPEC CPU2017 Benchmark Suite)

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 21 de Septiembre de 2018

Fdo: Agustín Navarro Torres

Agradecimientos

Sobretodo quiero agradecer a mi director, Pablo, todo lo que me ha enseñado, sus consejos, su paciencia y la ayuda que me ha proporcionado durante el desarrollo del trabajo. También quiero agradecer a Chus todas sus recomendaciones, el tiempo que me ha dedicado, sus consejos y sus aportes a la redacción de la memoria. Además, dar las gracias a todo el grupo de arquitectura de computadores (gaZ) por acogerme como uno más.

Agradecer a mis compañeros de máster por su solidaridad y apoyo durante el curso. Y a Adrián por sus ideas, ayuda y pausas para el *café*.

A mi familia, a Alex y a mis amigos por estar siempre ahí y ser grandes pilares de apoyo.

Resumen

La investigación experimental en arquitectura de computadores se basa en alimentar a una máquina real o a un modelo de simulación con una carga de trabajo (*workload*) que nos permita evaluar ideas. La opción más habitual es usar una carga de trabajo consistente en un conjunto (*suite*) de programas de prueba (*benchmarks*) seleccionados para ser representativos del software contemporáneo o futuro a la fecha de selección. SPEC CPU es una de las *suites* de *benchmarks* más utilizadas en la investigación de arquitectura de computadores. La caracterización de estos programas es una de las primeras tareas a realizar por la comunidad de arquitectura de computadores. Entre los objetivos de los trabajos de caracterización podemos destacar la selección de muestras para simulación y la clasificación de programas según determinadas características. En este trabajo presentamos un análisis detallado del rendimiento de la jerarquía de memoria de un procesador Intel Xeon Skylake SP ejecutando los programas de SPEC CPU2006 y los mono-hilo de CPU2017.

La experimentación se ha basado en varias herramientas disponibles en procesadores de altas prestaciones. Se ha utilizado la herramienta de *profiling Perf* para la lectura de los contadores hardware del procesador. Asimismo, se ha desarrollado la herramienta *Perf++* que amplía las funcionalidades de *Perf*. Este soporte hardware de monitorización permite identificar fases temporales en base a cambios en la velocidad de ejecución de los programas. Se ha utilizado *Intel Resource Directory*, una tecnología presente en los procesadores Intel Xeon que permite limitar la asociatividad de la memoria cache del último nivel (*Last Level Cache*, LLC). De esta forma se puede analizar el comportamiento de los programas ante distintos tamaños del último nivel de la memoria cache. Además, mediante el registro de estado (MSR) se puede habilitar y deshabilitar los distintos prebuscadores hardware lo que permite analizar su influencia en la ejecución de los programas.

Nuestros experimentos muestran que una parte importante de los programas provoca tasas de fallos muy bajas en la LLC, incluso con tamaños reducidos y sin prebúsqueda hardware. Aquellos programas que sí presionan los niveles superiores de cache han sido clasificados según su sensibilidad al tamaño de la LLC y a la prebúsqueda hardware. Hemos observado que aumentar el tamaño de la LLC reduce la tasa de fallos en LLC para muchos de estos programas. Por otra parte, la prebúsqueda hardware es muy eficiente: reduce los fallos en la LLC sin aumentar de forma significativa el ancho de banda utilizado. Por último, el análisis temporal de las programas muestra como la utilización de *Simpoint* no garantiza la identificación de puntos de simulación representativos desde el punto de vista del uso de la jerarquía de memoria.

Índice

Índice de cuadros **XI**

Índice de figuras **XIII**

1. Introducción **1**

1.1. Objetivos 1

1.2. Plan de trabajo 2

1.3. Organización de la memoria 3

2. Estado del arte **5**

2.1. Cargas de trabajo 5

2.2. Caracterización 6

2.3. Selección de puntos de simulación 6

2.3.1. Hierarchical Clustering 7

2.3.2. SimFlex 7

2.3.3. Simpoint 7

3. Metodología **9**

3.1. Métricas 9

3.2. SPEC CPU 9

3.2.1. SPEC CPU2006 10

3.2.2. SPEC CPU2017 10

3.3. Intel Xeon Gold 5120 12

3.3.1. Subsistema de memoria 13

3.3.2. Model-specific registers (MSR) 14

3.4. Herramientas de medición 15

3.4.1. Contadores Hardware 15

3.5. Limitación de vías de asociatividad de la LLC 16

3.5.1. Intel Resource Director 16

3.6. Limitación de conjuntos de la LLC 16

3.6.1. Funciones Hash de asignación de banco y conjunto 17
3.6.2. Coloreo de página en Linux 18
3.6.3. Problemas en el uso de coloreo de páginas 19
3.7. Experimentos realizados 19

4. Resultados de la caracterización **21**
4.1. Caracterización general 21
4.2. Identificación de los programas intensivos en memoria 24
4.3. Sensibilidad al tamaño de LLC y a la prebúsqueda hardware 24
4.3.1. Correlación entre MPKI3 y CPI 29
4.4. Rendimiento de los prebuscadores Hardware 29
4.5. Evolución temporal de los programas 34

5. Conclusiones y líneas abiertas **37**
5.1. Conclusiones técnicas 37
5.2. Problemas encontrados 38
5.3. Líneas abiertas 38
5.4. Conclusiones personales 38

6. Bibliografía **41**

Anexos **44**

A. Jerarquía de Memoria **47**
A.1. Memoria Cache 47
A.1.1. Estructura 48
A.1.2. Algoritmo de Remplazo 51
A.1.3. Prebúsqueda 51

B. Skylake-SP Scales Server Systems **53**
B.1. Características generales 53
B.2. Subsistema de memoria 53

C. Análisis **55**
C.1. SPEC CPU2006 56
C.1.1. Caracterización General 56
C.1.2. Evolución Temporal 57
C.1.3. Pre-búscadores Hardware 68
C.1.4. Tamaño de LLC 79
C.2. SPEC CPU2017 Rate 90

C.2.1. Caracterización General 90
C.2.2. Evolución Temporal 91
C.2.3. Pre-búscadores Hardware 98
C.2.4. Tamaño de LLC 105
C.3. SPEC CPU2017 Speed Single-Thread 113
C.3.1. Evolución Temporal 113
C.3.2. Caracterización General 113
C.3.3. Pre-búscadores Hardware 116
C.3.4. Tamaño de LLC 119

D. Articulos **123**

Índice de cuadros

3.1. Descripción de los SPECint 2006. 10

3.2. Descripción de los SPECfp CPU2006. 11

3.3. Relación entre los nombres de los programas con múltiples entradas de SPEC CPU2006 usados y sus entradas. 11

3.4. Descripción de los SPEC Integer CPU2017 12

3.5. Descripción de los SPEC Float CPU2017. 12

3.6. Relación entre los nombres de los programas con múltiples entradas de SPEC CPU2017 usados y sus entradas. 13

3.7. Características de la jerarquía de memoria del procesador Xeon Gold 5120 . . 13

4.1. Tabla resumen con miles de millones de instrucciones, porcentaje de instrucciones de lectura, porcentaje de instrucciones de escritura, MPKI en L1, L2, y L3, y CPI de los programas de SPEC CPU2006. 22

4.2. Tabla resumen con miles de millones de instrucciones, porcentaje de instrucciones de lectura, porcentaje de instrucciones de escritura, MPKI en L1, L2, y L3, y CPI de los programas de SPEC CPU2017. 23

C.1. Tabla resumen con los millones de instrucciones, porcentaje de instrucciones de lectura, porcentaje de instrucciones de escritura, MPKI en L1, MPKI en L2, MPKI en L3 y CPI de los programas de SPEC CPU2006 56

C.2. Tabla resumen con los millones de instrucciones, porcentaje de instrucciones de lectura, porcentaje de instrucciones de escritura, MPKI en L1, MPKI en L2, MPKI en L3 y CPI de los programas de SPEC CPU2017Rate 90

C.3. Tabla resumen con los millones de instrucciones, porcentaje de instrucciones de lectura, porcentaje de instrucciones de escritura, MPKI en L1, MPKI en L2, MPKI en L3 y CPI de los programas de SPEC CPU2017Speed Single-thread 113

Índice de figuras

1.1. Diagrama de Gantt del desarrollo del proyecto durante 10 meses 3

3.1. Jerarquía cache del procesador utilizado. 14

3.2. Asignación de vías de la LLC por núcleo. Al núcleo 0 le corresponden 4 vías, y los núcleos 1 y 2 tienen asignadas 2 vías de la LLC en cada conjunto. . . . 17

3.3. Descomposición de la dirección de memoria de un bloque para la selección del conjunto y del banco a donde va dicho bloque. 18

3.4. Descomposición de la dirección de memoria de un bloque para la selección del conjunto y del banco a donde va dicho bloque en un procesador Intel. La caja H representa la función de hash para selección de banco. 18

4.1. MPKI1, MPKI2 y MPKI3 para todas las parejas programa-entrada de SPEC CPU2006. En blanco los programas sin interés para la jerarquía de memoria, en gris con interés pero no seleccionados y en negro con interés y seleccionados. 24

4.2. MPKI1, MPKI2 y MPKI3 para todas las parejas programa-entrada mono-hilo de SPEC CPU2017. En blanco los programas sin interés para la jerarquía de memoria, en gris con interés pero no seleccionados y en negro con interés y seleccionados. 25

4.3. MPKI3 de las parejas programa-entrada de los CPU2006 seleccionadas para distintos tamaños de LLC. 26

4.4. MPKI3 de las parejas programa-entrada de los CPU2017 seleccionadas para distintos tamaños de LLC. 27

4.5. Speed-up de la prebúsqueda hardware y del tamaño para los SPEC CPU2006 y CPU2017. 28

4.6. Relación entre MPKI3 y CPI de las parejas programa-entrada de CPU2006 seleccionadas. 30

4.7. Relación entre MPKI3 y CPI de las parejas programa-entrada de CPU2017 seleccionadas. 31

4.8. Impacto de los prebuscadores hardware en el CPI y BPKI de las parejas programa-entrada de CPU2006 seleccionadas. 32

4.9. Impacto de los prebuscadores hardware en el CPI y BPKI de las parejas programa-entrada de CPU2017 seleccionadas. 33

4.10. Evolución temporal del MPKI3 y *Simpoints* de las parejas programa-entrada de los CPU2006 seleccionadas. 35

4.11. Evolución temporal del MPKI3 y *Simpoints* de las parejas programa-entrada de los CPU2017 seleccionadas. 36

A.1. Características de la jerarquía según nivel 47

A.2. Estructura de una memoria cache con vector de *TAGs* y de datos 48

A.3. Niveles de cache y velocidad asociada 48

A.4. Inclusividad gráfica de una memoria cache multinivel 48

A.5. Jerarquía de cache con dos cores, dos L1 privada y una L2 compartida 49

A.6. Jerarquía de cache con cuatro cores con L1 y L2 privada, y una L3 compartida dividida en cuatro bancos 49

A.7. Correspondencia de un bloque a la memoria cache según tipo. Las lineas grises representa donde puede ir el bloque y los corchetes los conjuntos de líneas . . 50

A.8. Descomposición de la dirección de memoria de un bloque para la selección del conjunto y del banco a donde va dicho bloque 50

A.9. Diferencia entre ejecución con pre-búsqueda y sin ella 51

B.1. Comparación entre el subsistema de memoria de Skylake y las generaciones anteriores. En *Skylake-SP* LLC es mayormente no inclusiva. Las peticiones de lectura (1) van directamente a memoria y las victimas de L2 van a L3 (2). Los datos compartidos entre múltiples cores pueden mantenerse en L3 (3). 54

Capítulo 1

Introducción

La investigación en arquitectura de computadores requiere el uso de cargas de trabajo, *workloads*, que permiten evaluar de forma realista un sistema. Diversas organizaciones internacionales definen periódicamente conjuntos de programas (*benchmark suites*) orientados a plataformas y objetivos distintos. La caracterización de estos programas es una de las primeras tareas a realizar en la comunidad de arquitectura de computadores. Entre los objetivos de los trabajos de caracterización podemos destacar la selección de muestras para simulación, la clasificación de programas según determinadas características y la detección de comportamientos no óptimos de los sistemas existentes.

La implementación de una nueva idea sobre un sistema real es inviable en la mayoría de los casos debido a su elevado coste. Como alternativa, las nuevas ideas se implementan sobre simuladores que modelan en detalle sistemas tan complejos como un chip multiprocesador formado por varios núcleos, una jerarquía de memoria con varios niveles y una red de interconexión. La ejecución completa de programas reales sobre estos simuladores requiere meses o incluso años. Por tanto, se utilizan técnicas de muestreo para seleccionar pequeños fragmentos de los programas que sean representativos de la ejecución completa.

En muchas publicaciones se utilizan cargas de trabajo con características conocidas a priori para analizar el comportamiento de nuevas propuestas. Por ejemplo, en trabajos sobre algoritmos de remplazo para caches compartidas entre varios núcleos es frecuente utilizar mezclas de programas con distinto grado de presión sobre la jerarquía: unas mezclas están formadas con programas que demandan mucho espacio en cache, otras con programas que demandan poco espacio, otras mixtas, etc. La caracterización es necesaria para clasificar los programas según su comportamiento.

Por otra parte, la ejecución de estos programas sobre sistemas actuales y el análisis de su comportamiento permiten detectar oportunidades de mejora, que serán la base de nuevas propuestas hardware.

1.1. Objetivos

En este trabajo hemos caracterizado los conjuntos de programas SPEC CPU2006 y CPU2017 respecto a su comportamiento en la jerarquía de memoria de un procesador Intel Xeon Skylake SP. La caracterización de SPEC CPU2017 tiene especial interés debido a que se propuso a principios de 2017 y solo se han publicado dos trabajos al respecto. El procesador usado también aporta relevancia a este estudio puesto que la familia Intel Xeon Skylake SP se lanzó en Julio de 2017 e incorpora un cambio significativo en la jerarquía de memoria dentro del chip, ya que la cache compartida de último nivel (*Last Level Cache*, LLC) ya no guarda

inclusión con los niveles inferiores como en todos los procesadores Intel previos. Tampoco se han publicado estudios de comportamiento de esta nueva jerarquía.

Los objetivos concretos del trabajo son los siguientes:

- Caracterización general de todos los programas en cuanto a número de instrucciones ejecutadas, porcentaje de instrucciones de acceso a memoria, y tiempo de ejecución sobre el procesador de estudio.
- Identificación de aquellos programas intensivos en la jerarquía de memoria.
- Caracterización de la sensibilidad de cada uno de los programas respecto al tamaño de LLC disponible. Obtención de tasas de fallos y tiempos de ejecución de cada programa con distintos tamaños de LLC.
- Caracterización del comportamiento de los prebuscadores hardware disponibles en el procesador en términos de influencia sobre la tasa de fallos en LLC, la cantidad de información leída de memoria principal y el tiempo de ejecución de los programas.
- Caracterización de la evolución temporal de los programas con el objetivo de identificar fragmentos relevantes para la simulación. Se obtendrá la evolución, a lo largo de la ejecución completa de cada programa, de métricas relacionadas con la jerarquía de memoria: fallos en la L2 por cada mil instrucciones (MPKI2), fallos en LLC por cada mil instrucciones (MPKI3) y número de ciclos por instrucción (CPI).

Para la obtención de las métricas se usarán los contadores hardware proporcionados por la arquitectura Intel, mediante la herramienta *Perf*. El comportamiento de los distintos prebuscadores se estudiará activando cada uno de ellos de forma individual mediante el uso del registro de control correspondiente. La obtención de datos para distintos tamaños de LLC se ha realizado en trabajos previos mediante simulación, lo que imponía el uso de muestreo e impedía disponer de los datos de la ejecución completa. En este trabajo usaremos una nueva herramienta, *Intel Resource Director*, incorporada recientemente a los procesadores Intel Xeon que permite entre otras cosas, limitar el número de vías de LLC dedicadas a un programa. De esta forma seremos capaces de obtener las métricas para la ejecución completa. También se ha intentado implementar coloreo en la asignación de páginas de memoria virtual para limitar el número de conjuntos de LLC que pueden ser destino de los bloques de un programa y en consecuencia el espacio de la LLC usado por el programa.

1.2. Plan de trabajo

A continuación se enumeran las tareas realizadas para conseguir los objetivos descritos anteriormente y se presenta la planificación temporal de dichas tareas.

1. Estudio de trabajos previos sobre caracterización de cargas de trabajo.
2. Aprendizaje, despliegue y desarrollo de herramientas de *profiling* para la utilización de contadores hardware durante la ejecución de un programa.
3. Instalación, compilación y ejecución de los programas que componen los SPEC CPU2006 y CPU2017.
4. Estudio del procesador utilizado, sus características y las herramientas disponibles para modificar el comportamiento de su jerarquía de memoria.

5. Estudio del algoritmo de asignación de páginas en el sistema de gestión de memoria virtual en Linux.
6. Modificación del kernel de Linux para implementar coloreo de página.
7. Definición de los experimentos de caracterización.
8. Ejecución de los experimentos en la máquina de referencia.
9. Recogida y análisis de los resultados.
10. Redacción de la memoria.

Estas diez tareas se han desarrollado durante los últimos 10 meses, parcialmente en paralelo con la realización de las últimas asignaturas del máster. La figura 1.1 muestra el diagrama de Gantt correspondiente. Los números del eje y representan las tareas nombradas anteriormente.

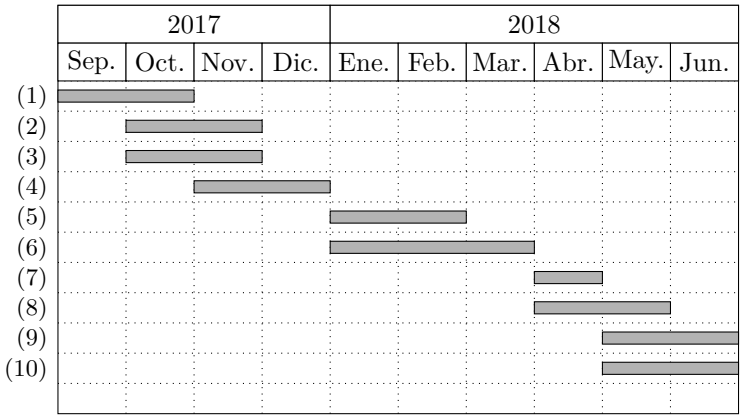


Figura 1.1: Diagrama de Gantt del desarrollo del proyecto durante 10 meses

1.3. Organización de la memoria

El resto del presente documento está organizado del siguiente modo: en el capítulo 2 se introduce el estado del arte en la simulación y las cargas de trabajo; en el capítulo 3 se explica en detalle las cargas de trabajo, el procesador utilizado, aspectos sobre la memoria LLC, contadores hardware y los experimentos realizados; en el capítulo 4 se muestran los resultados de la caracterización; en el capítulo 5 se presentan las conclusiones y posibles lineas abiertas.

Debido a extensión de los resultados obtenidos, las gráficas correspondientes a todos los programas se adjuntan como anexo (C). También se añaden anexos para explicar con mas detalle la jerarquía de memoria de un procesador de altas prestaciones (A) y las características del procesador Intel Xeon Gold 5120 (B). Por último, se incluyen los artículos que han surgido a partir de este trabajo de fin de máster (D).

Capítulo 2

Estado del arte

Las caracterizaciones de nuevas suites de *benchmarks* es un contenido recurrente en la literatura. Estas no aportan grandes innovaciones en la metodología utilizada, pero aportan material de consulta de gran interés para los investigadores.

En este capítulo se presenta el estado del arte en tres ámbitos: los *benchmarks* existentes, y las metodologías más utilizadas para la caracterización y la selección de puntos de simulación.

2.1. Cargas de trabajo

Una carga de trabajo está formada por uno o varios programas de prueba (*benchmarks*) y sirve para evaluar el rendimiento de un sistema. Existen varios conjuntos de programas de prueba (*benchmark suites*) orientados a la evaluación de distintos tipos de sistemas, o de distintas características de dichos sistemas. En la arquitectura de computadores tienen un rol primordial: permiten evaluar el rendimiento de distintas arquitecturas, reales o simuladas, y compararlas. Esto permite corroborar hipótesis de investigación o ayudar a la toma de decisiones para la compra de un determinado sistema.

Si nos centramos en la evaluación del rendimiento de los procesadores, podemos destacar tres conjuntos de programas:

- **SPEC CPU** [8]: su objetivo es medir el rendimiento del procesador, la jerarquía de memoria y el compilador. Para ello evalúan la capacidad que tiene un sistema para ejecutar una tarea de forma individual, y el trabajo que puede realizar un sistema por unidad de tiempo. Los programas que lo componen se dividen en dos categorías: cálculo de enteros y cálculo de coma flotante. Actualmente existen seis suites, de las cuales cuatro están retiradas: CPU89, CPU92, CPU95 y CPU2000; y dos están en activo: CPU2006 y CPU2017.

- **SPLASH2** [28]: es una colección de trece programas y kernels paralelos del dominio de la computación de altas prestaciones, *High Performance Computing (HPC)*. Ha sido ampliamente utilizada para la evaluación y diseño de multiprocesadores desde su salida en los años 90. Sin embargo, debido a su antigüedad y por contar únicamente con modelos de paralelización HPC, su uso ha disminuido a favor de nuevas *suites*.

- **PARSEC** [4]: trece programas paralelos, de diversas áreas de conocimiento, cuyo objetivo es la evaluación de multiprocesadores. Incluyen programas que probablemente serán de gran relevancia en el futuro, aunque no lo sean actualmente.

Además, existen otros conjuntos de programas de prueba de uso menos extendido por estar dirigidos a áreas de conocimiento específicas como la biología (*BioParallel*) o la física (*PhysicsBench*).

Nuestro trabajo se centra en SPEC CPU por ser la suite de mayor difusión. Además, el lanzamiento reciente de la versión CPU2017 abre la necesidad de realizar este tipo de trabajos de caracterización para los nuevos programas. Estos trabajos aportarán nuevos datos sobre el comportamiento de los programas actuales.

2.2. Caracterización

En la literatura, los trabajos de caracterización de programas de prueba se han realizado utilizando complejos simuladores o contadores hardware.

La simulación permite modelar distintas configuraciones de los componentes que se desea estudiar como múltiples tamaños de cache o distintos algoritmos de reemplazo. Sin embargo, el tiempo necesario para simular un programa completo es de semanas o meses, por lo que estos trabajos sólo caracterizan una pequeña parte de la ejecución de cada programa.

Los contadores hardware permiten obtener resultados y caracterizar programas en el tiempo de ejecución real de los mismos. Sin embargo, la plataforma sobre la que se realiza dicha caracterización permanece fija, impidiendo experimentar con distintas configuraciones.

Los trabajos dedicados a la caracterización de SPEC CPU2006 son numerosos. Entre los relacionados con nuestro objetivo de estudio, Jaleel et al. caracterizan el comportamiento de los programas de SPEC CPU2006 con distintos tamaños de cache sobre un simulador [11], Korn et al. estudian el rendimiento de la suite según el tamaño de página virtual [14], y Bird et al. analizan el rendimiento de los programas de la *suite* ejecutándolos sobre un procesador Intel Core 2 Duo [25].

Respecto a SPEC CPU2017, sólo hemos encontrado dos trabajos de caracterización, Limaye et al. [15] y Panda et al. [19]. En estos se analizan con contadores hardware el comportamiento de los programas sobre un procesador Intel con la micro-arquitectura Haswell, y ofrecen datos generales sobre número de instrucciones ejecutadas, reparto por tipo de instrucciones, *footprint*, y tasas de fallos en cada nivel de la jerarquía. Terminan presentando una metodología de clasificación de programas que analizamos más adelante. Respecto a la caracterización de la jerarquía de memoria, tienen varias limitaciones como que usan un procesador antiguo, presentan datos de tasa de fallos en lugar de MPKI, y no analizan la respuesta a tamaño de cache ni prebúsqueda.

En nuestro trabajo se han utilizado contadores hardware para la obtención de las métricas deseadas Por otra parte se han usado diversas técnicas para variar algunos parámetros del sistema sobre el que se ejecuta. Los *Model Specific Registers* (MSR) de Intel nos permiten activar y desactivar de forma independiente los distintos prebuscadores hardware disponibles. La nueva tecnología *Intel Resource Director* nos permite variar la capacidad de la LLC dedicada a un programa, modificando el número de vías de la LLC asociadas a dicho programa. Con todo ello conseguimos extraer datos del programa completo para distintas configuraciones de la jerarquía de memoria.

2.3. Selección de puntos de simulación

Como hemos descrito, las *suites* está formadas por un conjunto de programas, algunos de ellos con varias entradas distintas, lo que resulta en decenas de ejecuciones posibles.

Como ejemplo, SPEC CPU2006 nos proporciona 55 parejas programa-entrada. El tiempo de ejecución de un programa completo sobre un simulador es del orden de semanas o meses, lo que hace inviable la simulación del conjunto completo. Para reducir dicho tiempo se usa muestreo a dos niveles. En primer lugar, se selecciona un subgrupo de los programas y entradas. En segundo lugar, se seleccionan fragmentos de la ejecución del programa que sean representativos del comportamiento de la ejecución completa. En la literatura se han propuesto diversas técnicas de muestreo entre las que destacamos *Hierarchical Clustering* [21] para seleccionar programas y las metodologías *SimFlex* [27] y *Simpoint* [20] para la selección de fragmentos.

2.3.1. Hierarchical Clustering

La metodología *Hierarchical Clustering* se aplica en tres pasos: i) ejecución de todos los programas con todas las entradas para obtener 20 métricas mediante contadores hardware, ii) análisis de componentes principales para reducir el número de métricas a 4 variables, y iii) Hierarchical Clustering para agrupar programas similares.

En el primer paso, los autores seleccionan métricas independientes de la organización del sistema, relacionadas con el tipo de instrucciones ejecutadas y su proporción. Por tanto, esta metodología no considera el comportamiento en la jerarquía como parámetro para guiar el muestreo.

En este proyecto analizamos la representatividad del subconjunto seleccionado con esta metodología respecto al comportamiento en la jerarquía de memoria.

2.3.2. SimFlex

La metodología *SimFlex* pretende simular sistemas a gran escala de forma rápida, precisa y flexible. Esta compuesta por:

- **Flexus**: framework de simulación que permite simular un sistema completo, con componentes bien definidos, fáciles de integrar y optimizados en tiempo de compilación.
- **SMART**: teoría de muestreo estadístico para la selección de puntos de simulación.

Está metodología produce numerosos puntos de simulación de pequeño tamaño, que están distribuidos a lo largo de todo el programa, garantizando que sean representativos del mismo.

Sin embargo, *SimFlex* tiene un inconveniente importante para ser utilizada en la investigación en jerarquía de memoria cache, puesto que los puntos de simulación obtenidos no tienen la suficiente extensión para proporcionar datos correctos sin realizar calentamiento previo de las estructuras de datos (*warm-up*). Este calentamiento supone una sobrecarga no asumible cuando las estructuras de datos o el número de puntos de simulación son grandes.

2.3.3. Simpoint

Simpoint es una de las metodologías más utilizadas para la selección de puntos de simulación. *Simpoint* calcula las fases de la ejecución de una pareja programa-entrada, y selecciona el punto más representativo de cada fase (puntos de simulación). El conjunto de los puntos de simulación es representativo de la pareja programa-entrada.

Para la utilización de *Simpoint* primero se realiza un *profiling* del programa. Para ello se utilizan los *Basic Block Vector (BBV)*, que capturan información sobre los cambios de un programa a lo largo de su ejecución. Un *BBV* [22] es un vector unidimensional donde cada elemento corresponde a un bloque básico de código (intervalos) con información asociada como el número de veces que se ejecuta, número de instrucciones, tasa de fallos en cache, tasa de fallos en el predictor de saltos…. El *BBV* de un programa se utiliza como entrada para un algoritmo de clusterización, *k-means*, que agrupa por conjuntos aquellos intervalos con características similares. Una vez se obtienen los distintos conjuntos se seleccionan los intervalos más cercanos a la media de cada conjunto (*centroides*). Estos intervalos son los puntos de simulación, los cuales tienen un peso asociado que indica lo representativo que es el intervalo en el programa. Dicho peso es la división entre el número de instrucciones de la agrupación seleccionada partido por el número de instrucciones totales del programa.

Pese a que *Simpoint* ofrece varios puntos de simulación para cada programa, la mayoría de los trabajos que usan esta herramienta en el ámbito de la jerarquía de memoria sólo utilizan un fragmento, el que la metodología determina como más representativo.

Nuestro objetivo en este proyecto es determinar la representatividad del fragmento seleccionado por *Simpoint* respecto al comportamiento en la jerarquía. Para ello se va a analizar la evolución temporal de métricas como CPI, MPKI2 y MPKI3 a lo largo de toda la ejecución de los programas. El fragmento seleccionado por *Simpoint* se representará sobre las gráficas de evolución temporal.

Capítulo 3

Metodología

En este capitulo se presenta la metodología utilizada para la realización de los experimentos. En la sección 3.1 se definen las métricas usadas para presentar los resultados. En 3.2 se ofrece información sobre las *suites* SPEC CPU2006 y CPU2017. En 3.3 se describe el procesador sobre el que se han realizado los experimentos. En 3.5 se describe cómo se ha modificado la asociatividad de la LLC para disminuir su tamaño. En 3.6 se explica cómo limitar el tamaño de la LLC mediante coloreo de página. Por último, en 3.7 se definen los experimentos realizados para la caracterización.

3.1. Métricas

Las métricas utilizadas para presentar los resultados son las siguientes:

- **MPKIi**: fallos en cada nivel de cache (L1, L2 y L3) por cada mil instrucciones. $MPKIi = \frac{LiMisses}{Instrucciones/1000}$
- **CPI**: ciclos por instrucción. $CPI = \frac{Ciclos}{Instrucciones}$
- **BPKI**: bytes leídos de memoria principal por cada mil instrucciones retiradas por el procesador. $BPKI = \frac{Bytes\ leidos\ en\ memoria\ principal}{Instrucciones/1000}$

3.2. SPEC CPU

"*SPEC*, *Standard Performance Evaluation Corporation* es una organización sin ánimo de lucro cuya función es establecer, mantener y estandarizar *benchmarks* y herramientas que permitan analizar el rendimiento y eficiencia energética de los próximos sistemas computacionales."[1] SPEC oferta múltiples *suites* que simulan cargas de trabajo realistas para distintos tipos de sistema como *cloud*, *cpu*, servidores de correo....

La rama *CPU* está compuesta por cargas de trabajo intensivas en cálculo. Estos *benchmarks* son utilizados por fabricantes e investigadores, siendo uno de los más utilizados en las investigaciones sobre jerarquía de memoria, procesadores y compiladores.

Se han definido seis *suites* (CPU89, CPU92, CPU95, CPU2000, CPU2006 y CPU2017), de las cuales hay cuatro retiradas (CPU89, CPU92, CPU95 Y CPU2000) y las dos últimas están en activo (CPU2006 y CPU2017).

[1] Definición de: `https://www.spec.org/`

3.2.1. SPEC CPU2006

La versión CPU2006 se divide en *SPECint* y *SPECfp*. Los *SPECint*, ver tabla 3.1, son doce programas de calculo de enteros, y los *SPECfp*, ver tabla 3.2, son diecinueve programas de calculo en coma flotante.

El rendimiento de un sistema que utiliza la *suite* se cuantifica mediante dos métricas:

- *Speed*: cuantifica la capacidad de un sistema para completar la ejecución de un programa de forma individual.
- *Rate*: cuantifica cuantas tareas simultaneas puede completar un sistema por unidad de tiempo (*throughput*).

Ambas métricas se calculan ejecutando los mismos programas y con las mismas entradas. Para Speed, se ejecuta una instancia del programa, mientras para Rate se ejecutan varias instancias del mismo programa de forma simultánea. Dado que nuestros objetivos de caracterización se centran en el comportamiento del programa, la diferenciación Speed/Rate no tiene significado en nuestro trabajo. En la tabla 3.3 se muestra la relación entre las parejas programa-entrada y el alias utilizado en el documento, cuando un programa tiene múltiples entradas.

Benchmark	**Lenguaje**	**Descripción**
400.perlbench	C	Interprete de Perl
401.bzip2	C	Compresión
403.gcc	C	Compilador de C
429.mcf	C	Optimización de combinatorio
445.gobmk	C	Inteligencia Artificial: Go
456.hmmer	C	Búsqueda secuencial de genomas
458.sjeng	C	Inteligencia Artificial: Ajedrez
462.libquantum	C	Física / Computación cuántica
464.h264ref	C	Compresión de vídeo
471.omnetpp	C++	Simulación de eventos discretos
473.astar	C++	Algoritmos de caminos
483.xalancbmk	C++	Procesado de XML

Cuadro 3.1: Descripción de los SPECint 2006.

Los programas de la *suite* CPU2006 se han compilado según las recomendaciones de SPEC: *GCC* 4.4.4 y flags *-O3 -fno-stric-aliasing*.

3.2.2. SPEC CPU2017

Debido al reciente lanzamiento de la versión CPU2017, julio de 2017, solamente hay dos trabajos de caracterización sobre ella. Esta consiste en 43 programas divididos en dos categorías:

- *Speed*: veintitrés programas de cálculo de enteros y de coma flotante que miden el rendimiento del sistema para tareas individuales. Los programas de coma flotante y el programa `657.xz_s` pueden ser ejecutadas en multi-hilo. Los programas de enteros, excepto `657.xz_s`, son ejecutados en mono-hilo.
- *Rate*: veinte programas de cálculo de enteros y de coma flotante que miden el trabajo que puede realizar un sistema por unidad de tiempo. Todas ellas son versiones mono-hilo, de las que se ejecutan varias instancias simultáneamente.

Benchmark	Lenguaje	Descripción
410.bwaves	Fortran	Dinámica de fluidos
416.gamess	Fortran	Química Cuántica
433.milc	C	Física / Dinámica cuántica
434.zeusmap	Fortran	Física / CFD
435.gromacs	C, Fortran	Física / Dinámica de moleculas
436.cactusADM	C, Fortran	Física / Relatividad general
437.leslie3d	Fortran	Dinámica de fluidos
444.namd	C++	Biología / Dinámica molecular
447.dealII	C++	Análisis de elementos finitos
450.soplex	C++	Programación lineal, optimización
453.povray	C++	Renderización de imágenes
454.calculix	C, Fortran	Mecánica estructural
459.GemsFDTD	Fortran	Electromagnetismo computacional
465.tonto	Fortran	Química cuántica
470.lbm	C	Dinámica de fluidos
481.wrf	C, Fortran	Predicción meteorológica
482.sphinx3	C	Reconocimiento del habla

Cuadro 3.2: Descripción de los SPECfp CPU2006.

Programa	Entrada	Programa	Entrada
400.perlbench.1	checkspam.pl	416.gamess.1	cytosine.2.config
400.perlbench.2	diffmail.pl	416.gamess.2	h2ocu2+.gradient.config
400.perlbench.3	splitmail.pl	416.gamess.3	triazolium.config
401.bzip2.1	input.source	445.gobmk.1	13x13.tst
401.bzip2.2	chicken.jpg	445.gobmk.2	nngs.tst
401.bzip2.3	liberty.jpg	445.gobmk.3	score2.tst
401.bzip2.4	input.program	445.gobmk.4	trevorc.tst
401.bzip2.5	text.html	445.gobmk.5	trevord.tst
401.bzip2.6	input.combined	450.soplex.1	pds-50.mps
403.gcc.1	166.in	450.soplex.2	ref.mps
403.gcc.2	200.in	456.hmmer.1	nph3.hmm
403.gcc.3	c-typeck.in	456.hmmer.2	retro.hmm
403.gcc.4	cp-decl.in	464.h264ref.1	foreman_ref_encoder_baseline.cfg
403.gcc.5	expr.in	464.h264ref.2	foreman_ref_encoder_main.cfg
403.gcc.6	expr2.in	464.h264ref.3	sss_encoder_main.cfg
403.gcc.7	g23.in	473.astar.1	BigLakes2048.cfg
403.gcc.8	s04.in	473.astar.2	rivers.cfg
403.gcc.9	scilab.in		

Cuadro 3.3: Relación entre los nombres de los programas con múltiples entradas de SPEC CPU2006 usados y sus entradas.

Algunos programas se incluyen tanto en *Speed* como en *Rate*. Si son multi-hilo, en el experimento se usa la versión mono-hilo. La descripción de los programas, según su división en enteros o coma flotante, puede verse en las tablas 3.4 y 3.5 respectivamente. Para cada programa se indica si se usa en *Speed* y/o *Rate* y en cada caso si es multi-hilo (M) o mono-hilo (S). En la tabla 3.6 se muestra la relación entre las parejas programa-entrada y el alias utilizado en el documento, cuando un programa tiene multiples entradas.

Benchmark	**Rate**	**Speed**	**Lenguaje**	**Descripción**
Perlbench	S	S	C	Interprete de Perl
Gcc	S	S	C	Compilador de C
Mcf	S	S	C	Planificador de rutas
Omnetpp	S	S	C++	Simulador discreto de eventos
Xalancbmk	S	S	C++	Conversor de XML to HTML
X265	S	S	C	Compresión de video
Deepsjeng	S	S	C++	Inteligencia Artificial: Ajedrez
Leela	S	S	C++	Inteligencia Artificial: Go
Exchange2	S	S	Fortran	Inteligencia Artificial: Sudoku
Xz	S	M	C	Compresión de datos

Cuadro 3.4: Descripción de los SPEC Integer CPU2017

Rate	**Rate**	**Speed**	**Lenguaje**	**Descripción**
Bwaves	S	M	Fortran	Dinámica de fluidos
CactuBSSN	S	M	C++, C, Fortran	Física
Namd	S	—	C++	Dinámica molecular
Parest	S	—	C++	Imagen Biomédica
Povray	S	—	C++, C	Trazador de rayos
Lbm	S	M	C	Dinámica de fluidos
Wrf	S	M	Fortran, C	Previsión meteorológica
Blender	S	—	C++, C	Renderizado 3D y animación
Cam4	S	M	Fortran, C	Modelado atmosférico
Pop	—	M	Fortran, C	Modelado oceánico a gran escala
Imagick	S	M	C	Manipulación de imágenes
Nab	S	M	C	Dinámica molecular
Fotonik3d	S	M	Fortran	Electromagnetismo
Roms	S	M	Fortran	Modelado oceánico regional

Cuadro 3.5: Descripción de los SPEC Float CPU2017.

Los programas de la *suite* CPU2017 han sido compilados con las opciones recomendadas por la fundación *SPEC* utilizando *GCC 6.1.0*:

- Para los *Speed*: *-g -O3 -march=native -fno-unsafe-math-optimizations -no-tree-loop-vectorize -fopenmp -DSPEC_OPENMP*.
- Para los *Rate*: *-g -O3 -march=native -fno-unsafe-math-optimizations -fno-tree-loop-vectorize*.

3.3. Intel Xeon Gold 5120

El procesador sobre el que se han realizado los experimentos es el *Xeon Gold 5120*, que pertenece a la familia *Skylake SP* de Intel. Esta familia de procesadores fue presentada en Julio de 2017.

Programa	Entrada	Programa	Entrada
500.perlbench_r.1	checkspam.pl	525.x264_r.2	x264_r –pass 2
500.perlbench_r.2	diffmail.pl	525.x264_r.3	x264_r –seek 500
500.perlbench_r.3	splitmail.pl	557.xz_r.1	cld.tar.xz
502.gcc_r.1	gcc-pp.c	557.xz_r.2	cpu2006docs.tar.xz
502.gcc_r.2	gcc-pp.c	557.xz_r.3	input.combined.xz
502.gcc_r.3	gcc-smaller.c	503.bwaves_r.1	bwaves_1.in
502.gcc_r.4	ref32.c	503.bwaves_r.2	bwaves_2.in
502.gcc_r.5	ref32.c	503.bwaves_r.3	bwaves_3.in
525.x264_r.1	x264_r –pass 1	503.bwaves_r.4	bwaves_4.in
600.perlbench_s.1	checkspam.pl	625.x264_s.3	–pass 2
600.perlbench_s.2	diffmail.pl	625.x264_s.4	–seek 500
600.perlbench_s.3	splitmail.pl	657.xz_s.1	cpu2006docs.tar.xz
602.gcc_s.1	-fipa-pta	657.xz_s.2	cld.tar.xz
602.gcc_s.2	-finline-limit=1000	603.bwaves_s.1	bwaves_1.in
602.gcc_s.3	-finline-limit=24000	603.bwaves_s.2	bwaves_2.in
625.x264_s.2	–pass 1		

Cuadro 3.6: Relación entre los nombres de los programas con múltiples entradas de SPEC CPU2017 usados y sus entradas.

El *Xeon Gold 5120*, a partir de ahora *XG5120*, es un procesador multinúcleo de alto rendimiento construido en 14 nm. Dispone de 14 núcleos con ejecución fuera de orden y superescalar de cuatro instrucciones por ciclo. Su frecuencia nominal es de 2.2 GHz con capacidad de aumentar hasta 3.2 GHz (turbo, 1 núcleo).

3.3.1. Subsistema de memoria

El *XG5120* cuenta con una L2 privada a cada núcleo de 1 MiB y una LLC compartida de 19.25 MiB dividida en catorce bancos, uno por núcleo. Las características del subsistema de memoria se encuentran en el cuadro 3.7 y un diagrama del mismo en la figura 3.1.

La familia *Skylake* de procesadores para servidores introduce importantes cambios en la jerarquía de memoria respecto a las familias anteriores. La *LLC* actúa como *victim cache* [12] de la cache *L2* y por tanto pasa a ser no-inclusiva para la mayoría de los bloques. Esto quiere decir que, salvo algunas excepciones, los bloques se guardan bien en *LLC* o bien en *L2*, pero no en ambas simultáneamente. Además, la *LLC* utiliza un patrón de reuso para la selección de contenido y para el algoritmo de reemplazo [2]. Como consecuencia, Intel aumenta la capacidad de la cache *L2* de 256 KiB a 1 MiB por núcleo y el tamaño de la LLC ha disminuido de 2.5 MiB a 1.375 MiB por núcleo.

Nivel	Tamaño	Asociatividad	Tipo	Nº Bancos
L1	32 KiB	8	Privada / Inclusiva	—
L2	1 MiB	16	Privada / Inclusiva	—
L3	19.25 MiB	11	Compartida / Generalmente No-Inclusiva	14x1.375 MiB

Cuadro 3.7: Características de la jerarquía de memoria del procesador Xeon Gold 5120

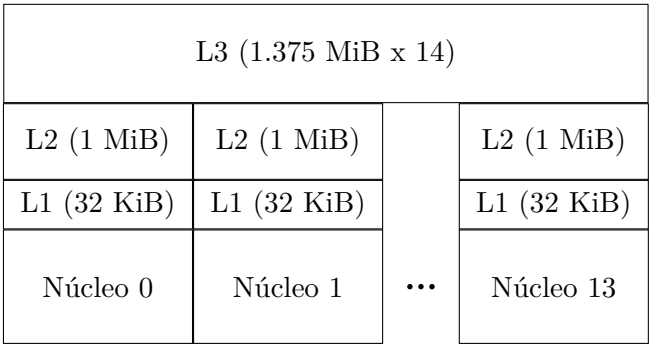


Figura 3.1: Jerarquía cache del procesador utilizado.

Prebuscadores

La familia *Skylake* de procesadores Intel cuenta con cuatro prebuscadores hardware [1]. Estos intentan predecir de forma automática los datos que van a ser utilizados por el programa en ejecución y acercarlos al procesador. Todos ellos limitan su espacio de prebúsqueda al tamaño de una página de memoria. Es decir, nunca lanzan una petición que cruce la frontera de la página en la que han detectado el patrón de acceso.

Los prebuscadores según el nivel de cache sobre el cual actúan son los siguientes:

- La **L1 Data Cache** cuenta con dos prebuscadores:
  - **Data cache unit Prefetcher** (DCU): es un prebuscador secuencial que detecta accesos sobre datos cargados recientemente en L1 con patrón secuencial ascendente y prebusca el siguiente bloque para cargarlo en L1. Por ejemplo: si el procesador realiza accesos con direcciones ascendentes dentro del bloque *N*, el prebuscador traerá a L1 el bloque *N+1*.
  - **Instruction pointer based stride prefetcher** (IP): monitoriza las instrucciones de lectura individualmente. Si detecta un patrón de acceso con distancia regular entre cada par de direcciones (*stride*), traerá la siguiente dirección del patrón. Por ejemplo: un procesador realiza lecturas a L1 con el patrón: *A*, *A+S*, *A+2S*, el prebuscador lanzará petición sobre la dirección *A+3S*.
- La **L2 Data Cache** cuenta con los siguientes prebuscadores hardware:
  - **Spatial Prefetcher** (L2A): asume que hay localidad espacial y prebusca siempre los bloques vecinos a los traídos de memoria. Para cada bloque cargado en L2 (64 bytes), el prebuscador lanza petición sobre su pareja para completar un fragmento alineado de 128 bytes.
  - **Streamer** (L2P): monitoriza las peticiones de lectura de bloque a L2. Sí detecta un patrón secuencial ascendente o descendente, prebusca las siguientes direcciones siguiendo el patrón. Puede generar varias direcciones hasta colocarse 20 posiciones en la secuencia por delante del procesador.

Los prebuscadores pueden ser activados/desactivados de forma independiente mediante un *model-specific register*. Por defecto están todos activados.

3.3.2. Model-specific registers (MSR)

Los *MSR* son varios registros de control incluidos en los procesador de arquitectura x86 y AMD64, que son utilizados para tareas de depuración, monitorización de rendimiento y

activación o desactivación de ciertas características del procesador.

En el proyecto se han usado estos registros para monitorizar el rendimiento del procesador durante la ejecución de los programas, activar y desactivar prebuscadores hardware y utilizar la herramienta *Intel CAT*.

3.4. Herramientas de medición

En este trabajo se han utilizado los contadores hardware como herramienta base para recabar información sobre la ejecución de un programa. Estos contadores pueden ser usados directamente o a través de otras herramientas software ya desarrolladas. En nuestro caso se ha usado *Perf* y se ha desarrollado una nueva herramienta, *Perf++*, que amplia la funcionalidad de *Perf*.

3.4.1. Contadores Hardware

Los contadores hardware son un conjunto de contadores de propósito especial incluidos en los procesadores Intel [18]. Dichos contadores son accesibles a través de registros MSR y almacenan información sobre la actividad del procesador, por ejemplo cuentan eventos como instrucciones ejecutadas, ciclos del procesador, accesos y fallos en cada nivel de cache, número de saltos ejecutados, ….

Los contadores hardware son dependientes del procesador. No todos están disponibles para todos los procesadores o miden los mismos eventos. A veces, dos o más eventos comparten el contador o el registro necesario para su obtención, por lo que no podrán ser medidos de forma simultánea.

El uso de contadores hardware añade una pequeña sobrecarga temporal ya que provoca interrupciones cada cierto tiempo. Además, su naturaleza estadística introduce cierto error de medición en algunos casos. Sin embargo, son una herramienta muy potente ya que permiten obtener información de muy bajo nivel y con mucho detalle.

Perf

Performance Counter for Linux (*Perf*) [6] es una herramienta para el análisis del rendimiento de un programa sobre Linux. Permite instrumentar contadores hardware y obtener información sobre la ejecución de determinadas funciones (*tracepoints*). Es utilizado por empresas y desarrolladores para analizar el rendimiento de sus programas, encontrar sus puntos críticos y optimizarlos.

En este trabajo, Perf se ha utilizado para leer los contadores hardware durante la ejecución de los programas que componen los *SPEC CPU2006* y *CPU2017*. De esta forma se ha obtenido información sobre su ejecución, como el número de instrucciones y ciclos ejecutados.

Perf++

Perf presenta una limitación a la hora de obtener cierto tipo de métricas, puesto que no permite medir el número de ocurrencias de un evento mediante muestreo con un periodo determinado por el número de ocurrencias de otro evento. Como ejemplo, para realizar la caracterización temporal de los programas necesitamos obtener el número de fallos en LLC cada un millón de instrucciones.

Por ello se ha desarrollado una aplicación, **Perf++**, que utilizando llamadas al sistema permite contar las ocurrencias de eventos hardware cada x ocurrencias de otro evento. La herramienta desarrollada permite medir eventos hardware definidos por el sistema operativo o por el procesador.

3.5. Limitación de vías de asociatividad de la LLC

Para ejecutar los programas con múltiples tamaños de LLC se ha utilizado *Intel Cache Allocation Technology*. Esta herramienta nos permite limitar el número de vías de asociatividad por conjunto disponibles para un programa. Por ejemplo, si un programa solo puede usar la mitad de vías, el tamaño de LLC disponible para ella será la mitad. La limitación de vías únicamente afecta a la LLC, por lo que no se modifica el número de accesos a la misma.

3.5.1. Intel Resource Director

Intel Resource Directory Technology (*IRDT*) [9] es un conjunto de herramientas desarrolladas por Intel para sus procesadores Xeon que permiten monitorizar y controlar sus recursos, como el ancho de banda o el tamaño de la LLC. *IRDT* cuenta con cuatro herramientas:

- **Cache Monitoring Technology (CMT)**: monitoriza la cantidad de LLC utilizada por cada núcleo o programa.
- **Memory Bandwidth Monitoring (MBM)**: monitoriza el ancho de banda utilizado por cada núcleo o programa.
- **Cache Allocation Technology (CAT)**: permite limitar la cantidad de memoria de la LLC asignada a cada núcleo o programa.
- **Code and Data Priorization (CDP)**: extensión especializada de la CAT. Ofrece control independiente sobre la colocación del código y los datos de los programas en la LLC.

Cache Allocation Technology

La *Cache Allocation Technology* (*CAT*) [10] permite definir y asignar porciones de la LLC a cada núcleo o programa. Para ello les asigna las vías de memoria que pueden utilizar, ver fig. 3.2. Por ejemplo si tenemos un procesador con 32 MiB de LLC con asociatividad 8 y queremos que el core 0 disponga de 16 MiB, y el 1 y 2 de 8 MiB; asignaremos 4 vías al núcleo 0 y 2 vías al núcleo 1 y 2. Por lo tanto, asignar regiones implica una disminución de memoria y de asociatividad.

3.6. Limitación de conjuntos de la LLC

Otra forma de limitar el tamaño de LLC dedicado a una aplicación consiste en limitar el número de conjuntos de cache disponibles para dicha aplicación mediante la técnica de coloreo de páginas. Esta técnica consiste en clasificar las páginas de memoria según su dirección. En nuestro caso se usará para crear una clase que, por su rango de direcciones, se aloje en un subconjunto de los conjuntos de cache. Esto es posible debido a que algunos bits de la

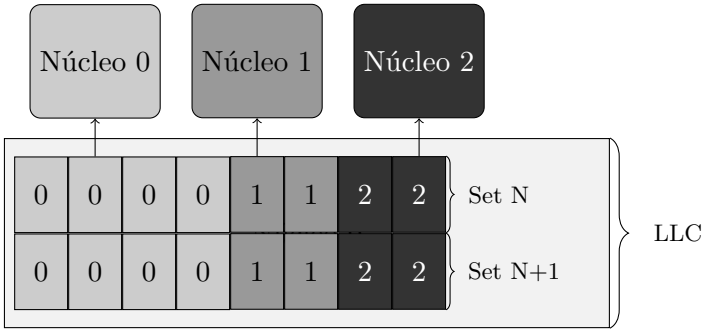


Figura 3.2: Asignación de vías de la LLC por núcleo. Al núcleo 0 le corresponden 4 vías, y los núcleos 1 y 2 tienen asignadas 2 vías de la LLC en cada conjunto.

dirección de bloque forman parte al mismo tiempo del número de página física y de los bits que deciden el conjunto de cache.

Al aplicar esta técnica se han encontrado problemas que han impedido finalmente su uso para la obtención de resultados de caracterización. Sin embargo, se ha considerado conveniente incluir en esta memoria una breve descripción del trabajo realizado y de los problemas encontrados.

En esta sección se presentan conceptos como la función Hash de asignación de conjuntos y la gestión de memoria virtual en Linux. Posteriormente se explica como se ha implementado el coloreo de páginas en el kernel de Linux.

3.6.1. Funciones Hash de asignación de banco y conjunto

En gran parte de la literatura la selección de banco y conjunto de LLC asociados a un bloque se realiza de forma transparente mediante unos pocos bits de la dirección de bloque, ver fig. 3.3. Por ejemplo para una LLC con k conjuntos, m vías, n bancos y c bytes de tamaño de bloque:

- Los $\log_2(c)$ bits de menor peso para el desplazamiento dentro del bloque.
- Los $\log_2(n)$ bits siguientes para la selección del banco.
- Los $\log_2(k)$ bits posteriores para la selección del conjunto.
- El resto de bits como etiqueta del bloque.
- La vía la selecciona el algoritmo de remplazo.

Sin embargo, para evitar ciertos tipos de ataques informáticos, los procesadores Intel emplean una función de Hash no documentada para la asignación de banco (ver fig. 3.4), que tiene como entrada casi todos los bits de la dirección de bloque, incluyendo los campos de Tag, Set y banco de la figura 3.3. El conjunto y desplazamiento dentro del bloque se realiza como se ha explicado en la sección anterior. En la literatura se han propuesto diversas metodologías para descifrar mediante ingeniería inversa dicha función [3, 29, 26, 17]. Sin embargo, todos estos trabajos se basan en procesadores con un número de bancos potencia de dos. El *XG5120* cuenta con catorce bancos en LLC, lo que convierte dicho problema en un NP-Completo.

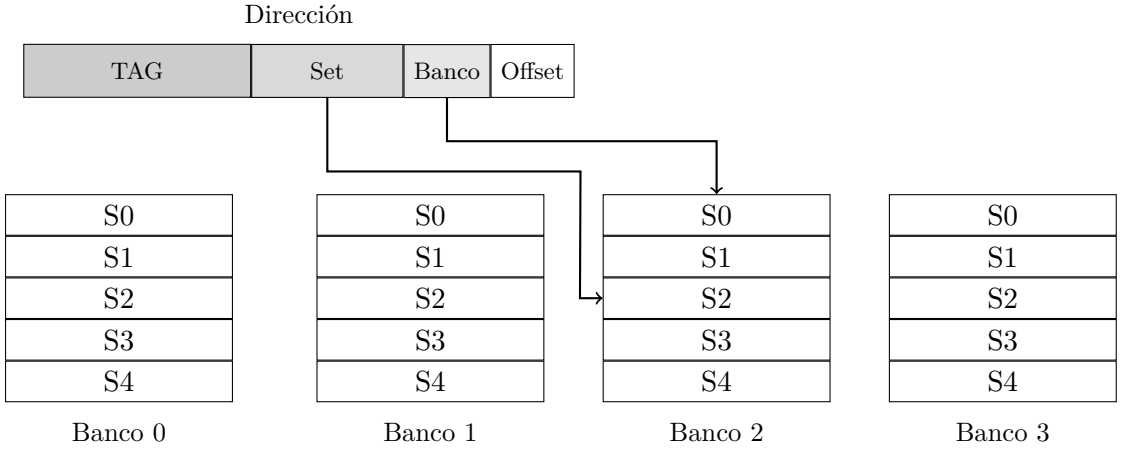


Figura 3.3: Descomposición de la dirección de memoria de un bloque para la selección del conjunto y del banco a donde va dicho bloque.

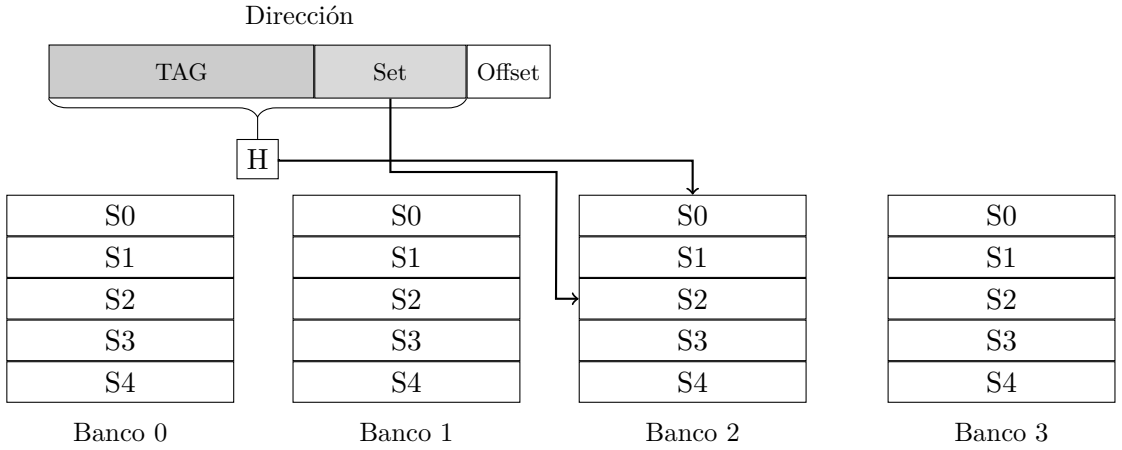


Figura 3.4: Descomposición de la dirección de memoria de un bloque para la selección del conjunto y del banco a donde va dicho bloque en un procesador Intel. La caja *H* representa la función de hash para selección de banco.

3.6.2. Coloreo de página en Linux

La memoria virtual [23] es una técnica que abstrae el manejo de la memoria, creando la ilusión de que un programa tiene a su disposición toda la memoria del sistema. El esquema más utilizado para su manejo es la paginación. La memoria física y el espacio virtual del proceso se dividen en bloques del mismo tamaño llamados páginas (4 KiB, 2 MiB o 1 GiB en Intel). El sistema operativo asigna páginas de memoria física a las páginas virtuales de cada proceso.

El coloreo de página, o coloreo de cache, modifica la asignación de páginas físicas a páginas virtuales. Para ello se asignan páginas dependiendo de los conjuntos de la cache en los cual se van a *mapear*.

Linux no cuenta con coloreo de página. Para su implementación, se ha modificado la asignación de páginas libres a procesos en el kernel. Antes de asignarse una página se hace una operación *and* entre la dirección física y una máscara previamente definida. Si el resultado es cero la página se asigna, en caso contrario se desecha y se pasa a la siguiente. También se ha desarrollado una nueva llamada al sistema que permite modificar la máscara.

3.6.3. Problemas en el uso de coloreo de páginas

El uso de coloreo de páginas para conseguir nuestro objetivo requiere conocer con precisión las funciones de asignación de banco de cache y conjunto. Como hemos descrito, Intel usa una función *hash* compleja y oculta para la asignación de banco, que además interacciona con los bits de selección de conjunto. Esto dificulta enormemente la implementación de coloreo de páginas para nuestros propósitos.

Por otra parte, es necesario que los bits usados para implementar el coloreo no influyan en la asignación de conjunto en los demás niveles de cache. En caso contrario, al limitar el tamaño de LLC también se limita el tamaño de los otros niveles y los resultados observados están influidos por esta limitación. En el caso de nuestro procesador, los bits de selección de conjunto en el nivel L2 de cache son un subconjunto de los bits de selección en LLC.

Por tanto, tras implementar coloreo de páginas y analizar resultados se descartó su uso para la caracterización de prestaciones en función del tamaño de LLC.

3.7. Experimentos realizados

Se han realizado cinco experimentos para la caracterización de los *SPEC CPU2006* y *CPU2017*.

Caracterización general. Resume sus aspectos más importantes (número de instrucciones, CPI y MPKI). Se han ejecutado todos los programas sobre el procesador de referencia sin modificar, toda la LLC disponible y todos los prebuscadores activados.

Identificación de los programas intensivos en memoria. Se han ejecutados todos los programas en un contexto de recursos limitados: prebúsqueda hardware desactivada y el tamaño mínimo de LLC que se puede asignar a un programa (1.75 MiB / 1 vía de asociatividad).

Sensibilidad al tamaño de LLC y prebúsqueda hardware. Usando la herramienta *Intel CAT* se han establecido cincos tamaños de LLC mediante la limitación del número de vías disponibles para cada programa a 11, 8, 4, 2 y 1. Con los registros de estado (MSR) se ha activado y desactivado la prebúsqueda hardware.

Rendimiento de los prebuscadores hardware. Para realizar la caracterización se han ejecutado los programas en el procesador de referencia, con toda la LLC disponible, y con distintas configuraciones de prebúsqueda hardware. Estas configuraciones se han realizado habilitando o deshabilitando los prebuscadores hardware realizando escrituras en el registro *MSR* correspondiente.

Evolución temporal de los programas. Usando la herramienta *Perf++* se han medido los ciclos ejecutados, los accesos y fallos a la LLC cada un millón de instrucciones

Todos los experimentos se han realizado sobre una máquina con dos procesadores *Intel Xeon Gold 5120*, 96 GiB de memoria RAM compartida, y Sistema operativo *Centos 7*. Los programas se han ejecutado de forma individual sobre un procesador garantizando la no compartición de recursos entre ellos.

En todos los experimentos se ha utilizado la herramienta *Perf* para medir los contadores hardware necesarios. Los contadores hardware tienen naturaleza estadística, por tanto sus mediciones contienen cierto grado de error. Con el fin de minimizar dicho error se ha repetido diez veces cada experimento. Los datos presentados corresponden a la media obtenida en cada caso. La desviación estándar media en los experimentos es de 1.3 % y la máxima de 3.2 %.

Capítulo 4

Resultados de la caracterización

En este capítulo se muestra la caracterización de las suites *SPEC CPU2006* y *CPU2017*. El capítulo está divido en cuatro secciones. En la sección 4.1 se ofrecen información general sobre los programas, en 4.2 se identifican aquellos programas intensivos en memoria, en 4.3 se estudia su sensibilidad al tamaño del último nivel de cache y la prebúsqueda hardware, en 4.4 se analiza el impacto de los distintos prebuscadores hardware en detalle y en 4.5 la evolución temporal de los programas.

Debido a la gran cantidad de gráficas de resultados obtenidas, en las secciones 4.3, 4.4 y 4.5 sólo se muestran algunos ejemplos y el análisis de los datos obtenidos. Los resultados completos se incluyen en el anexo C

4.1. Caracterización general

Se ha realizado una caracterización general de todos los programas de las dos *suites* para el obtener número de instrucciones ejecutadas, el porcentaje de instrucciones de lectura y escritura en memoria, los accesos y fallos en LLC y los ciclos por instrucción de los programas. Estos datos pueden verse en las tablas C.1 y 4.2

Los programas de la suite CPU2006 ejecutan una media de 934 mil millones de instrucciones (934 GI). La diferencia entre programas es muy grande siendo `454.calculix.1` el que más instrucciones ejecuta con 4249 GI, y `445.gobmak.1` el que menos con 23 GI.

Los programas de las suites CPU2017 ejecutan en general un mayor número de instrucciones que los de la suite CPU2006, con una media de 1476 GI. La variación entre programas también es muy grande con un máximo y un mínimo de 4204 GI y 186 GI para `521.wrf_r.1` y `502.gcc_r.4` respectivamente.

Podemos observar que el número de accesos a memoria no varía demasiado entre suites. En general es un número alto debido a que los Intel Xeon son procesadores CISC donde la gran mayoría de instrucciones pueden acceder a memoria. El porcentaje de lecturas es del 28 % en CPU2006 y del 31 % en CPU2017. El porcentaje de escrituras es del 11 % y 12 % para CPU2006 y CPU2017 respectivamente.

La variación entre programas es grande. Podemos destacar programas con muchos accesos a memoria como `436.cactusADM` en CPU2006 y a `548.exchange2` en CPU2017, con más del 75 % de accesos a memoria por instrucción. En el lado contrario, sólo `462.libquantum` de CPU2006 tiene menos del 25 %.

Las tablas incluyen valores de CPI y MPKI en los distintos niveles de cache ejecutado los

programas sobre el procesador de referencia sin modificar: toda la LLC disponible y todos los prebuscadores activados. El análisis detallado de estas métricas se realiza en las siguientes secciones.

Benchmark	**Instr**	**Lect**	**Escr**	**MPKI1**	**MPKI2**	**MPKI3**	**CPI**
400.perlbench.1	1052	26 %	12 %	7,52	0,58	0,01	0,38
400.perlbench.2	360	29 %	16 %	13,35	0,03	0,01	0,35
400.perlbench.3	656	28 %	9 %	2,70	0,22	0,02	0,31
401.bzip2.1	409	30 %	11 %	14,30	1,23	0,02	0,66
401.bzip2.2	174	24 %	11 %	12,18	1,67	0,01	0,47
401.bzip2.3	291	24 %	7 %	18,69	3,69	0,01	0,46
401.bzip2.4	532	27 %	12 %	8,30	0,60	0,02	0,51
401.bzip2.5	583	34 %	6 %	14,14	1,69	0,02	0,54
401.bzip2.6	327	29 %	11 %	14,18	1,35	0,02	0,63
403.gcc.1	69	23 %	15 %	34,49	5,02	0,38	0,70
403.gcc.2	140	24 %	13 %	21,62	2,64	0,44	0,69
403.gcc.3	123	18 %	20 %	43,88	2,30	0,31	0,61
403.gcc.4	91	20 %	17 %	33,66	2,61	0,30	0,58
403.gcc.5	100	20 %	17 %	38,17	3,05	0,48	0,60
403.gcc.6	137	19 %	17 %	38,57	3,20	0,53	0,64
403.gcc.7	168	22 %	14 %	36,08	2,94	1,07	0,70
403.gcc.8	149	19 %	19 %	39,04	3,51	0,60	0,70
403.gcc.9	54	24 %	13 %	19,05	1,72	0,24	0,67
410.bwaves.1	2554	37 %	7 %	19,61	0,16	0,14	0,44
416.gamess.1	109	32 %	11 %	8,14	0,00	0,00	0,6
416.gamess.2	85	33 %	9 %	10,32	0,00	0,00	0,33
416.gamess.3	371	31 %	8 %	5,47	0,01	0,00	0,30
429.mcf.1	315	31 %	9 %	123,23	50,33	17,48	2,20
433.milc.1	96	29 %	11 %	26,40	10,03	9,79	1,40
434.zeusmp.1	1915	21 %	8 %	22,08	1,30	1,07	0,52
435.gromacs.1	1948	36 %	13 %	11,75	0,03	0,00	0,45
436.cactusADM.1	2725	56 %	19 %	8,06	2,54	0,64	0,63
437.leslie3d.1	178	30 %	7 %	29,47	0,34	0,16	0,38
444.namd.1	2284	23 %	6 %	10,37	0,02	0,00	0,48
445.gobmk.1	23	25 %	13 %	12,65	0,27	0,02	0,76
445.gobmk.2	61	25 %	14 %	11,39	0,26	0,02	0,73
445.gobmk.3	31	24 %	10 %	11,50	0,12	0,01	0,69
445.gobmk.4	23	25 %	14 %	12,52	0,22	0,02	0,75
445.gobmk.5	33	25 %	14 %	10,77	0,17	0,02	0,71
447.dealII.1	1645	35 %	9 %	17,77	2,75	0,18	0,47
450.soplex.1	344	21 %	7 %	53,59	14,12	1,01	0,93
450.soplex.2	350	29 %	4 %	28,97	4,39	2,03	0,75
453.povray.1	940	33 %	15 %	24,95	0,00	0,00	0,41
454.calculix.1	4248	20 %	2 %	5,59	0,06	0,02	0,50
456.hmmer.1	860	42 %	15 %	6,80	0,03	0,00	0,36
456.hmmer.2	1823	41 %	15 %	4,73	0,01	0,00	0,37
458.sjeng.1	2260	22 %	8 %	2,67	0,38	0,26	0,56
459.GemsFDTD.1	1723	40 %	9 %	29,55	3,05	2,78	0,64
462.libquantum.1	1647	13 %	4 %	21,01	2,63	2,45	0,57
464.h264ref.1	495	41 %	11 %	4,08	0,12	0,00	0,33
464.h264ref.2	322	40 %	16 %	14,24	0,08	0,00	0,40
464.h264ref.3	2887	42 %	15 %	15,81	0,10	0,01	0,38
465.tonto.1	3443	24 %	10 %	10,38	0,16	0,01	0,43
470.lbm.1	1235	19 %	8 %	50,38	0,04	0,02	0,60
471.omnetpp.1	541	27 %	16 %	37,19	19,40	4,62	1,41
473.astar.1	359	32 %	9 %	31,22	6,31	0,32	1,02
473.astar.2	752	29 %	8 %	20,77	1,73	0,01	0,90
481.wrf.1	2931	24 %	6 %	11,92	0,22	0,14	0,41
482.sphinx3.1	3384	24 %	2 %	17,36	1,54	0,00	0,44
483.xalancbmk.1	994	29 %	6 %	27,81	3,82	0,28	0,54
AVG	933	28 %	11 %	21,46	2,99	0,87	0,61

Cuadro 4.1: Tabla resumen con miles de millones de instrucciones, porcentaje de instrucciones de lectura, porcentaje de instrucciones de escritura, MPKI en L1, L2, y L3, y CPI de los programas de SPEC CPU2006.

Benchmark	Instr	Lect	Escr	MPKI1	MPKI2	MPKI3	CPI
500.perlbench_r.1	1218	28 %	19 %	7.22	0.49	0.01	0.38
500.perlbench_r.2	699	31 %	17 %	17.07	0.04	0.02	0.39
500.perlbench_r.3	667	31 %	18 %	6.16	1.53	0.11	0.43
502.gcc_r.1	195	28 %	13 %	25.10	2.28	0.31	0.70
502.gcc_r.2	231	29 %	14 %	24.32	3.02	0.45	0.72
502.gcc_r.3	233	30 %	11 %	27.32	2.49	0.29	0.70
502.gcc_r.4	186	29 %	14 %	26.15	4.49	1.99	0.84
502.gcc_r.5	258	26 %	21 %	42.34	3.82	1.49	0.96
503.bwaves_r.1	1300	34 %	6 %	12.28	0.10	0.08	0.43
503.bwaves_r.2	1570	36 %	6 %	16.03	0.13	0.11	0.48
503.bwaves_r.3	1403	35 %	6 %	14.25	0.12	0.10	0.45
503.bwaves_r.4	1828	34 %	6 %	13.07	0.11	0.09	0.44
505.mcf_r.1	924	33 %	13 %	59.27	16.98	6.04	1.30
507.cactuBSSN_r.1	1113	50 %	11 %	118.76	5.47	1.82	0.80
508.namd_r.1	1777	30 %	7 %	17.31	0.04	0.01	0.39
510.parest_r.1	3390	38 %	4 %	33.36	1.25	0.04	0.50
511.povray_r.1	3307	37 %	16 %	26.31	0.00	0.00	0.43
519.lbm_r.1	1277	21 %	11 %	49.11	0.08	0.04	0.64
520.omnetpp_r.1	1093	30 %	17 %	33.98	11.90	3.49	1.24
521.wrf_r.1	4204	28 %	7 %	11.10	0.31	0.09	0.88
523.xalancbmk_r.1	1274	30 %	7 %	45.03	2.35	0.12	0.66
525.x264_r.1	516	25 %	8 %	4.03	0.11	0.06	0.34
525.x264_r.2	1963	24 %	6 %	4.46	0.04	0.02	0.32
525.x264_r.3	1985	24 %	6 %	3.97	0.04	0.02	0.32
526.blender_r.1	1725	31 %	5 %	7.77	0.91	0.13	0.59
527.cam4_r.1	2685	24 %	11 %	18.79	0.94	0.08	0.59
531.deepsjeng_r.1	1869	25 %	12 %	3.61	0.42	0.25	0.59
538.imagick_r.1	4589	23 %	8 %	6.10	0.02	0.00	0.31
541.leela_r.1	2111	26 %	9 %	4.51	0.10	0.00	0.82
544.nab_r.1	2085	33 %	10 %	9.35	0.02	0.01	0.69
548.exchange2_r.1	2908	51 %	30 %	0.09	0.00	0.00	0.54
549.fotonik3d_r.1	1947	44 %	14 %	34.13	4.30	3.78	0.69
554.roms_r.1	2709	34 %	9 %	27.32	0.92	0.47	0.47
557.xz_r.1	403	23 %	8 %	13.73	4.86	1.19	0.99
557.xz_r.2	1044	23 %	3 %	9.58	0.71	0.10	0.43
557.xz_r.3	567	23 %	6 %	8.05	1.74	0.36	0.65
600.perlbench_s.1	1218	28 %	19 %	7,26	0,50	0,01	0,38
600.perlbench_s.2	699	31 %	17 %	17,78	0,04	0,02	0,39
600.perlbench_s.3	667	31 %	18 %	6,69	1,53	0,11	0,44
602.gcc_s.1	1212	33 %	6 %	42,81	1,59	0,36	0,73
602.gcc_s.2	517	28 %	14 %	28,25	2,04	0,20	0,69
602.gcc_s.3	502	28 %	13 %	24,64	2,32	0,26	0,68
605.mcf_s.1	1653	34 %	10 %	74,58	30,47	7,37	1,41
620.omnetpp_s.1	1056	31 %	18 %	32,35	11,28	3,50	1,15
623.xalancbmk_s.1	1274	30 %	7 %	45,11	2,34	0,11	0,66
625.x264_s.1	516	25 %	8 %	3,98	0,11	0,05	0,34
625.x264_s.2	1963	24 %	6 %	4,34	0,04	0,02	0,32
625.x264_s.3	1985	24 %	6 %	3,87	0,04	0,02	0,32
631.deepsjeng_s.1	2179	25 %	12 %	3,50	0,27	0,25	0,58
641.leela_s.1	2111	26 %	9 %	4,47	0,10	0,00	0,82
648.exchange2_s.1	2908	51 %	30 %	0,10	0,00	0,00	0,54
AVG	1476	30 %	12 %	20.84	2.75	0.73	0.62

Cuadro 4.2: Tabla resumen con miles de millones de instrucciones, porcentaje de instrucciones de lectura, porcentaje de instrucciones de escritura, MPKI en L1, L2, y L3, y CPI de los programas de SPEC CPU2017.

4.2. Identificación de los programas intensivos en memoria

Todas las parejas programa-entrada de SPEC CPU2006 (29 programas, 55 parejas) y las mono-hilo que componen SPEC CPU2017 (23 programas, 51 parejas) han sido ejecutadas en un contexto de recursos limitados: prebúsqueda hardware desactivada y el tamaño mínimo de LLC que se puede asignar a un programa (1.75 MiB). Para cada carga de trabajo hemos medido sus fallos cada mil instrucciones en los tres niveles de la jerarquía de memoria cache (MPKI1, MPKI2 y MPKI3). Estas métricas se muestran en las Figuras 4.1 y 4.2. En ellas se encuentran indicadas en blanco las parejas programa-entrada que tienen ratios MPKI2 y MPKI3 muy bajas. En total son 20 de las 55 en SPEC 2006, y 20 de las 50 consideradas en SPEC 2017. Estas cargas de trabajo carecen de interés para el estudio de la jerarquía de memoria. De las restantes, hemos seleccionado una entrada de cada programa para mostrar el resultado de su caracterización en lo que queda de esta sección (20 en CPU2006 y 16 en CPU2017, señaladas en negro en la figura).

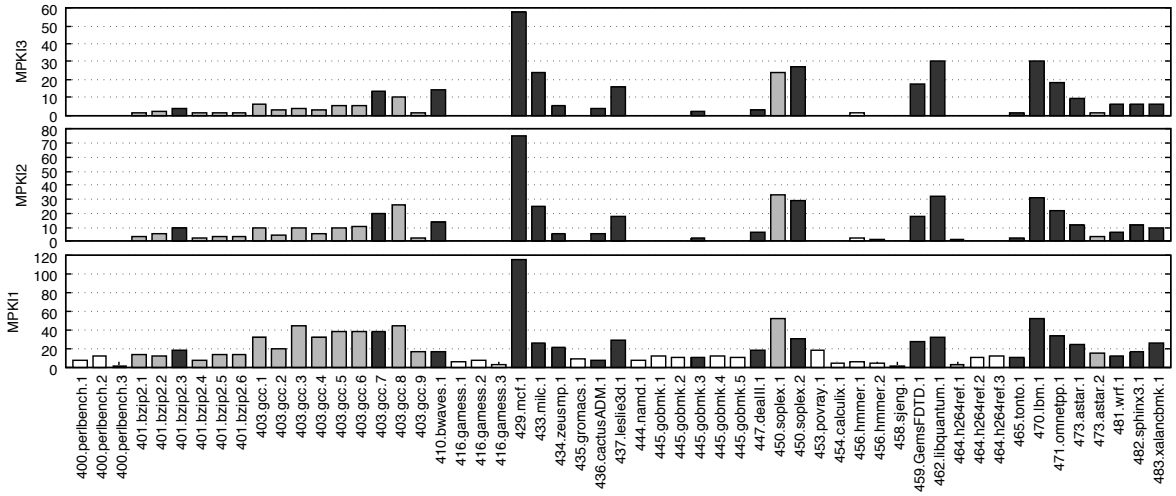


Figura 4.1: MPKI1, MPKI2 y MPKI3 para todas las parejas programa-entrada de SPEC CPU2006. En blanco los programas sin interés para la jerarquía de memoria, en gris con interés pero no seleccionados y en negro con interés y seleccionados.

Esta primera caracterización da información relevante para la selección de aplicaciones representativas según su comportamiento en la jerarquía. Limaye et al. [15] utilizan *Hierarchical clustering* para agrupar aplicaciones con comportamiento similar y seleccionar un grupo representativo. Sin embargo, su clasificación considera parámetros ajenos a la jerarquía y por tanto los grupos que obtiene no son homogéneos. Por ejemplo, este trabajo agrupa `508.namd_r.1` y `549.fotonik3d_r.1`. Según nuestra caracterización, estos dos programas muestran grandes diferencias en sus comportamientos en memoria: el programa `508.namd_r.1` sólo usa el primer nivel de cache mientras que `549.fotonik3d_r.1` es uno de los programas más intensivos en memoria.

4.3. Sensibilidad al tamaño de LLC y a la prebúsqueda hardware

En este experimento estudiamos la sensibilidad de los programas intensivos en memoria al tamaño de la LLC y a la prebúsqueda hardware. Las cargas de trabajo seleccionadas en la sección anterior han sido ejecutadas con cinco tamaños de LLC. Cada una de estas ejecuciones ha sido realizada con todos los prebuscadores activados y todos desactivados.

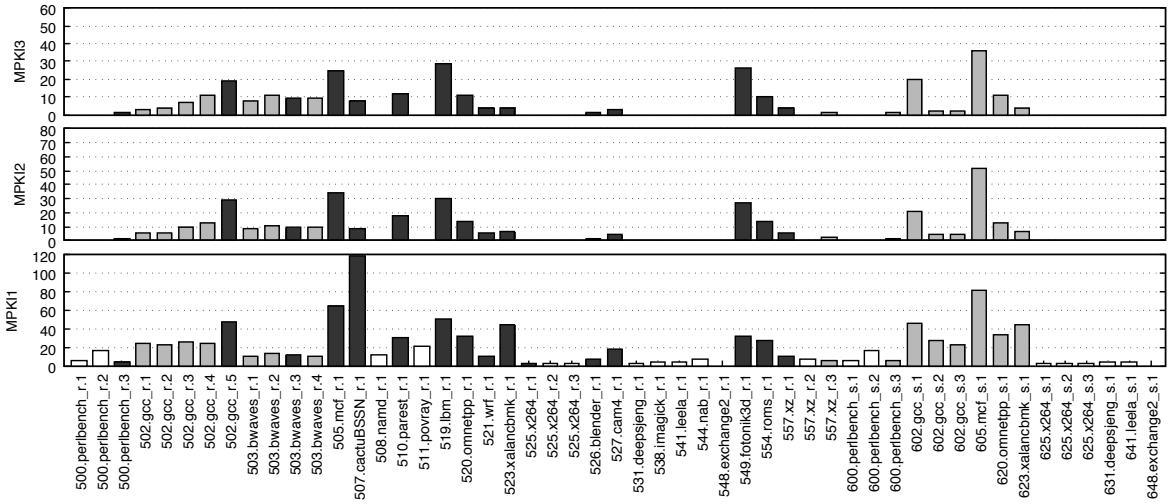


Figura 4.2: MPKI1, MPKI2 y MPKI3 para todas las parejas programa-entrada mono-hilo de SPEC CPU2017. En blanco los programas sin interés para la jerarquía de memoria, en gris con interés pero no seleccionados y en negro con interés y seleccionados.

Los tamaños de LLC utilizados han sido: 1.75 MiB, 3.5 MiB, 7 MiB, 14 MiB y 19.25 MiB, cuyas asociatividades son 1, 2, 4, 8 y 11, respectivamente.

Las figuras 4.3 y 4.4 muestran el MPKI3 de los *benchmarks* seleccionados para los distintos tamaños de LLC con y sin prebúsqueda. La figura se encuentra dividida en dos secciones, superior CPU2006 e inferior CPU2017, dentro de cada sección se ordenan de menor a mayor MPKI3 (izquierda — derecha).

Con la prebúsqueda hardware desactivada, el incremento de tamaño en LLC se traduce en una reducción de MPKI3 en todos los programas en ambas suites, a excepción de `410.bwaves`, `434.zeusmp`, `459.GemsFDTD` en CPU2006 y `503.bwaves` en CPU2017.

Cuando se activa la prebúsqueda, la mejora en MPKI conseguida al aumentar el tamaño de la LLC se reduce de forma considerable para 6 programas de CPU2006 (`433.milc`, `437.leslie3d`, `447.dealII`, `450.soplex.2`, `462.libquantum` y `481.wrf`), y para 5 de CPU2017 (`510.parest`, `519.lbm`, `521.wrf`, `549.fotonik3d` y `554.roms`).

La prebúsqueda hardware es muy efectiva para reducir MPKI3 para todos los tamaños de LLC en 14 programas de CPU2006 y en 10 de CPU2017. También obtiene buenos resultados para tamaños pequeños de LLC en otros 5 programas de CPU2006 (`401.bzip2`, `465.tonto`, `473.astar`, `482.sphinx3` y `483.xalancbmk`) y en 2 de CPU2017 (`510.parest` y `523.xalancbmk`). Sólo para `omnetpp`, presente en las dos suites, la prebúsqueda no reduce MPKI3 en ningún caso, e incluso la incrementa ligeramente con el mayor tamaño de LLC.

A modo de resumen, la figura 4.5 muestra los *speed-ups* conseguidos para cada programa respecto al sistema con tamaño mínimo de LLC y sin prebúsqueda (origen), al aplicar prebúsqueda (eje X), y al aumentar el tamaño de LLC hasta los 19.25 MiB (eje Y). La figura facilita la clasificación de programas en función de su sensibilidad a ambos parámetros. Los programas de enteros se muestran en gris con forma redonda y las de coma flotante en negro con forma de diamante. A modo de ejemplo, en CPU2006 vemos un grupo de programas muy sensibles a la prebúsqueda y poco sensibles al aumento de tamaño de LLC integrado por `462.libquantum`, `481.wrf`, `459.GemsFDTD`, `410.bwaves`, `470.lbm`, `437.leslie3d` y `450.soplex`.

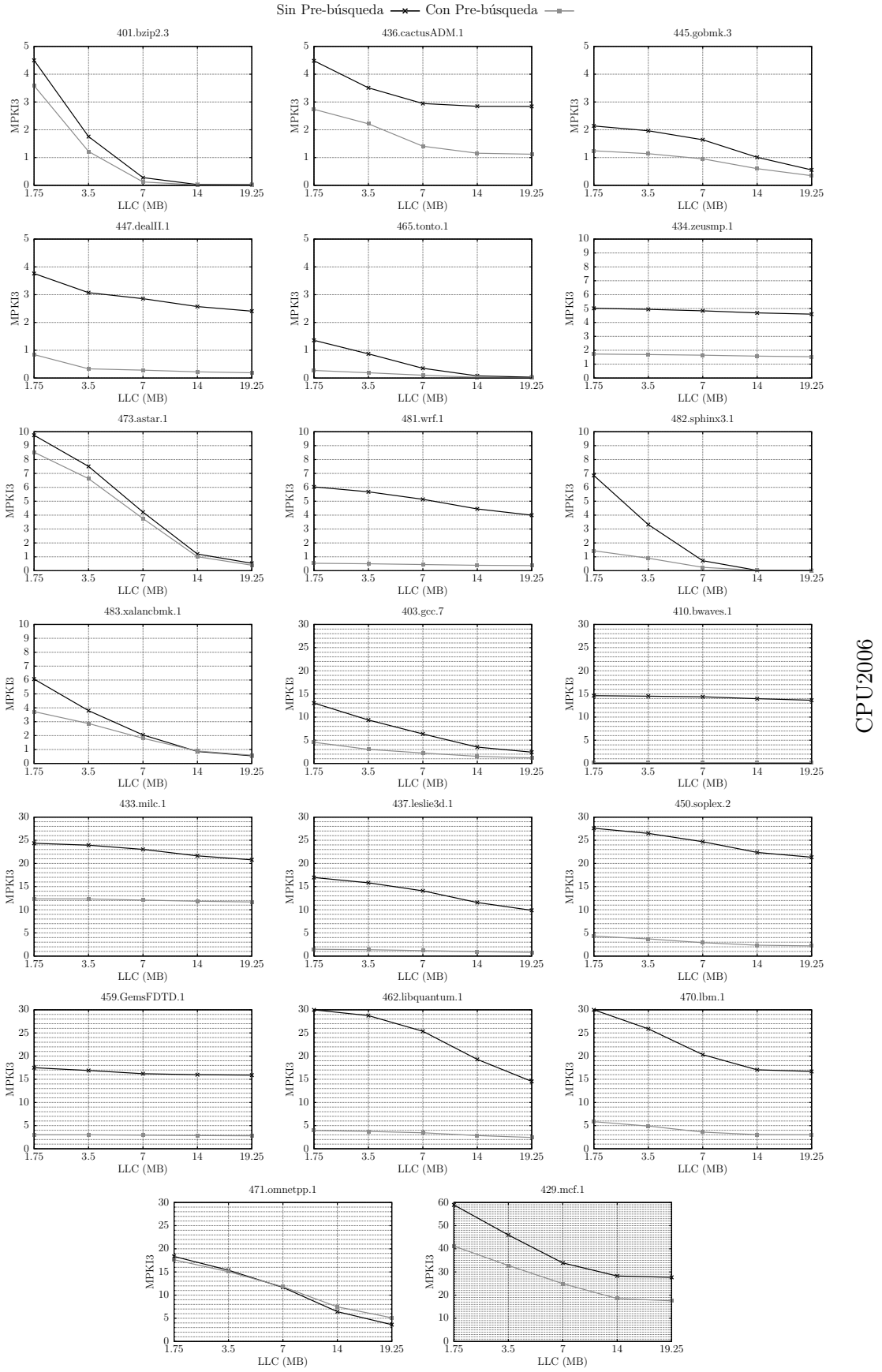


Figura 4.3: MPKI3 de las parejas programa-entrada de los CPU2006 seleccionadas para distintos tamaños de LLC.

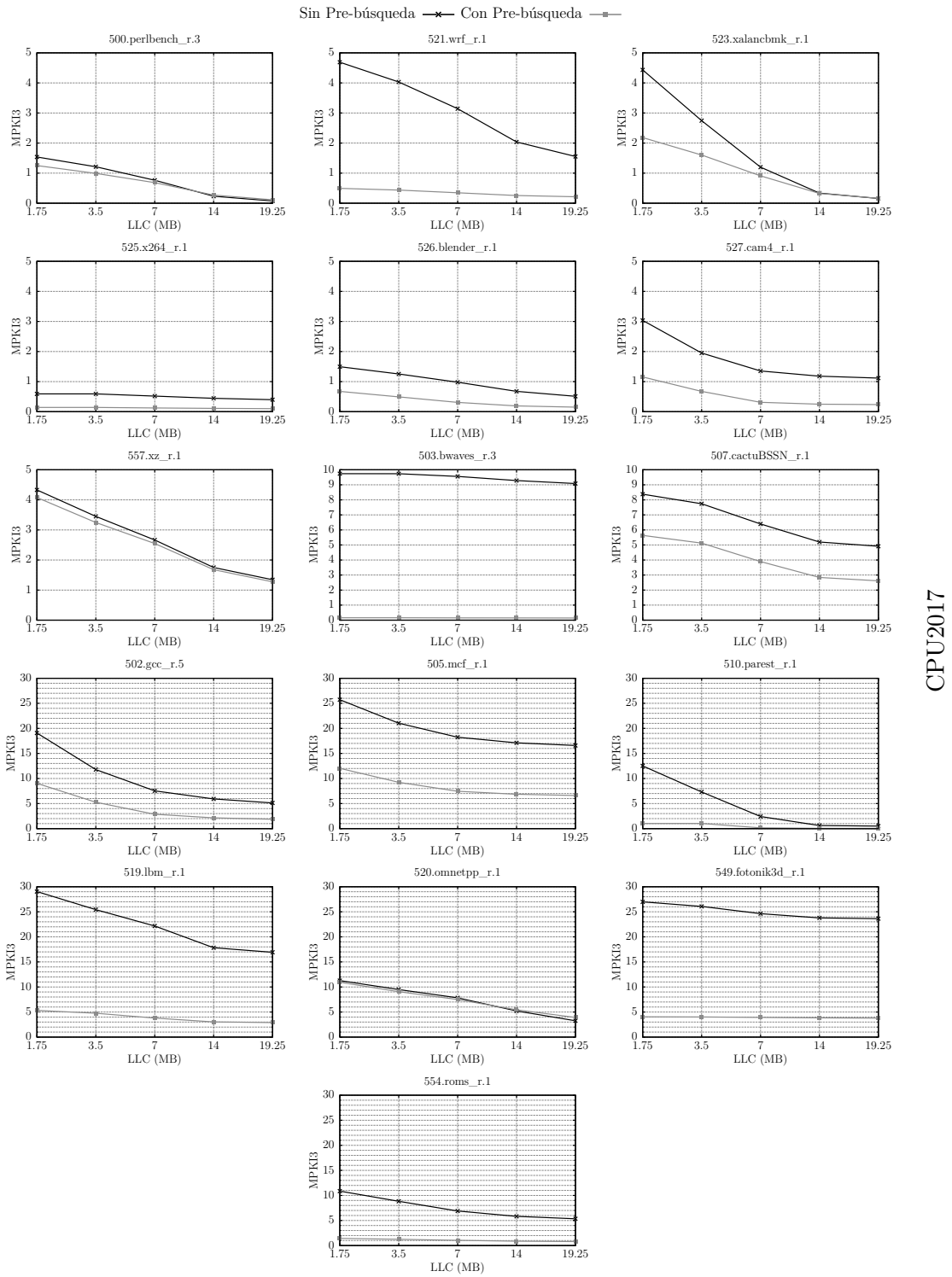
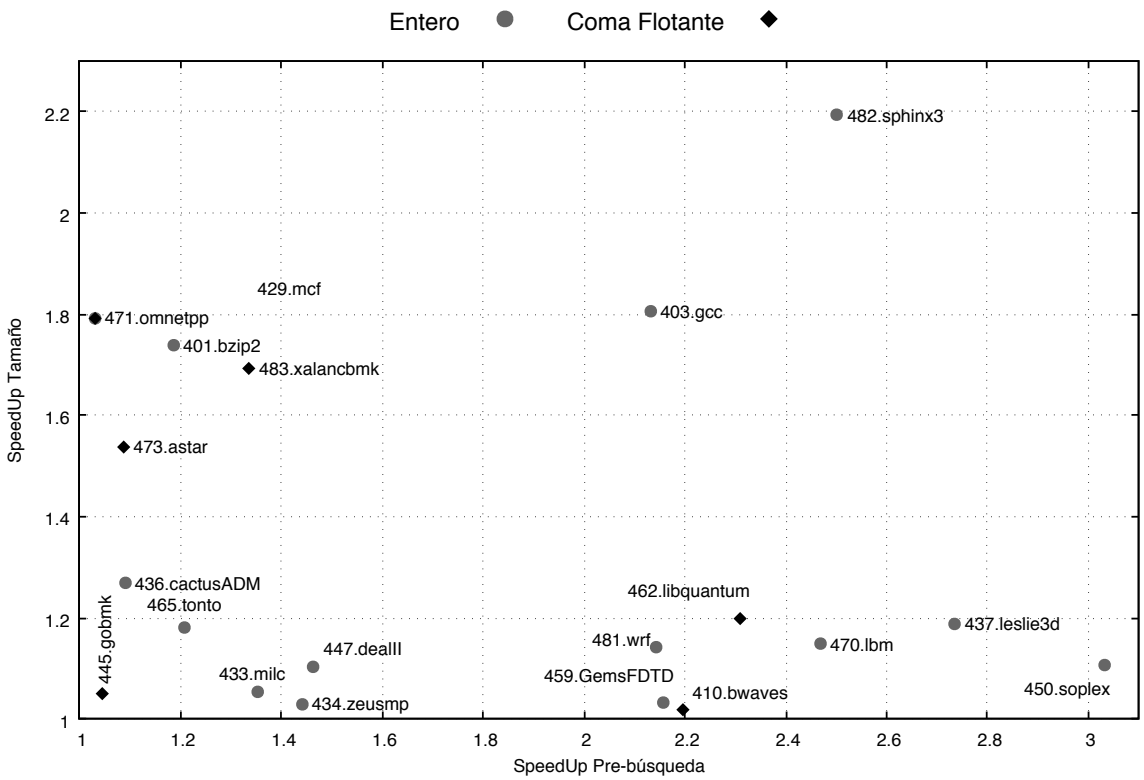
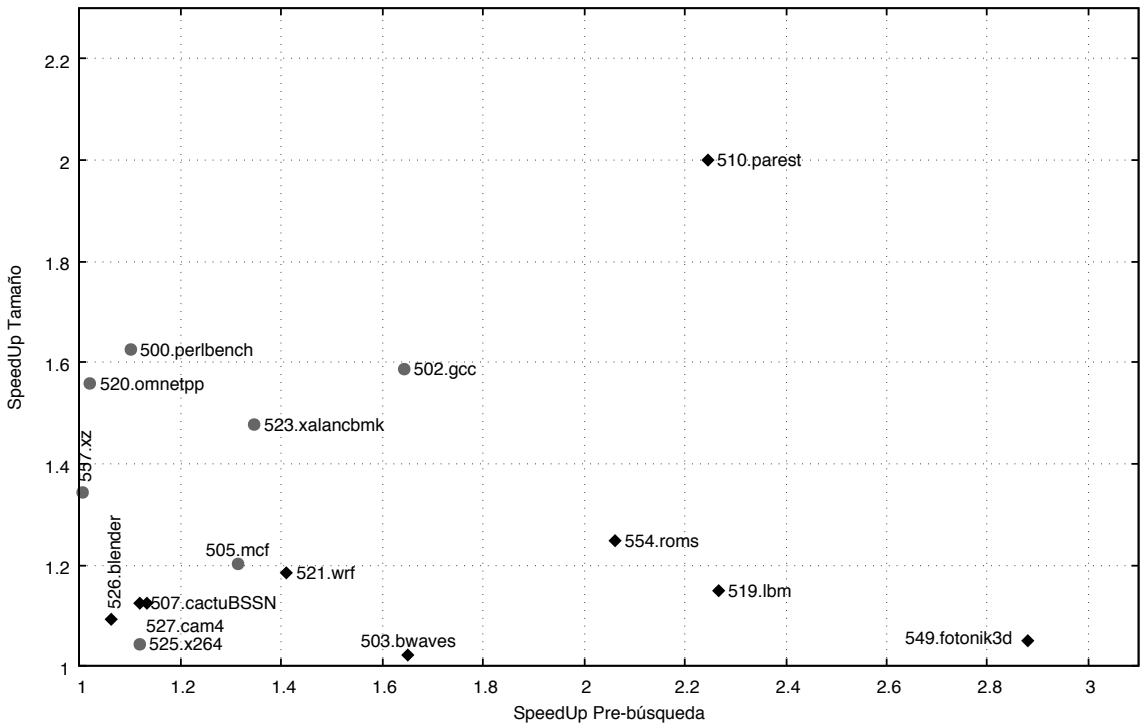


Figura 4.4: MPKI3 de las parejas programa-entrada de los CPU2017 seleccionadas para distintos tamaños de LLC.



(a) CPU2006



(b) CPU2017

Figura 4.5: Speed-up de la prebúsqueda hardware y del tamaño para los SPEC CPU2006 y CPU2017.

4.3.1. Correlación entre MPKI3 y CPI

Las figuras 4.6 y 4.7 muestran la correlación entre MPKI3 y CPI. En el eje X de cada gráfica se encuentra el MPKI en el último nivel de cache, y en el eje Y los ciclos por instrucción. Se incluyen valores para distintos tamaños de cache con y sin prebúsqueda. La pendiente de la recta de interpolación dibujada indica el incremento en CPI por cada incremento de 10 fallos por instrucción en la tasa de fallos de LLC.

Como se observa en la figura, la correlación es muy clara aunque el aumento de MPKI3 impacta de forma diferente en el CPI de cada programa. La pendiente de la recta de interpolación varía entre 0,03 para varias aplicaciones de las dos suites y 0,19 para `500.perlbench_r.3`. La pendiente de la recta proporciona otro eje de clasificación de los programas, ya que es consecuencia de propiedades como el paralelismo a nivel de instrucción o el agrupamiento y solapamiento temporal de los fallos en LLC.

4.4. Rendimiento de los prebuscadores Hardware

En esta sección vamos a analizar el impacto de los distintos prebuscadores hardware sobre el rendimiento de los programas y el ancho de banda consumido. El procesador Intel SKL-SP tiene cuatro prebuscadores hardware asociados al primer y segundo nivel de cache: *L1 Data cache unit prefetcher* (DCUI), *L1 Data cache instruction pointer stride prefetcher* (DCUP), *L2 Data cache spatial prefetcher* (L2A) y *L2 Data cache streamer* (L2P).

Todos los benchmarks seleccionados en la sección 4.2 han sido ejecutados con distintas configuraciones: todos los prebuscadores activados, todos los prebuscadores desactivados y cada prebuscador activado de manera individual. Los experimentos se han realizado con el tamaño máximo de LLC. En las figuras 4.8 y 4.9 se muestran su rendimiento medido en ciclos por instrucción (CPI, eje izquierdo) y el número de bytes leídos de memoria principal (BPKI, eje derecho).

L2P es el mejor prebuscador con mucha diferencia sobre los demás. Para los 14 programas de CPU2006 cuyas tasas de fallos se reducen al activar la prebúsqueda con el tamaño máximo de LLC, L2P en solitario obtiene más del 70 % de la reducción en CPI conseguida con todos los prebuscadores activos. En 10 de los 14 programas, L2P es responsable de más del 90 % de la mejora en CPI. En CPU2017 ocurre algo similar. Para los 10 programas que mejoran con la prebúsqueda en el tamaño mas grande de LLC, activando sólo L2P se consigue más del 82 % de la reducción en CPI conseguida cuando se activan todos los prebuscadores, y este porcentaje es mayor del 90 % en 7 de los 10 programas.

El segundo mejor prebuscador es DCUI, seguido por DCUP. L2A es el que consigue peores resultados, logrando reducir el CPI más de un 5 % sólo en 8 programas de CPU2006 y en 6 de CPU2017, con un máximo de un 20 % para `450.soplex`.

La prebúsqueda hardware es muy precisa, puesto que causa un aumento significativo en el ancho de banda utilizado en solamente 3 programas de la suite CPU2006 (`403.gcc.7`, `433.milc` y `471.omnetpp`) y en 3 programas de los CPU2017 (`520.omnetpp`, `549.fotonik3d` y `554.roms`). Además, exceptuando `471.omnetpp` y `520.omnetpp`, hay una disminución de CPI considerable, por lo que el resultado global es positivo.

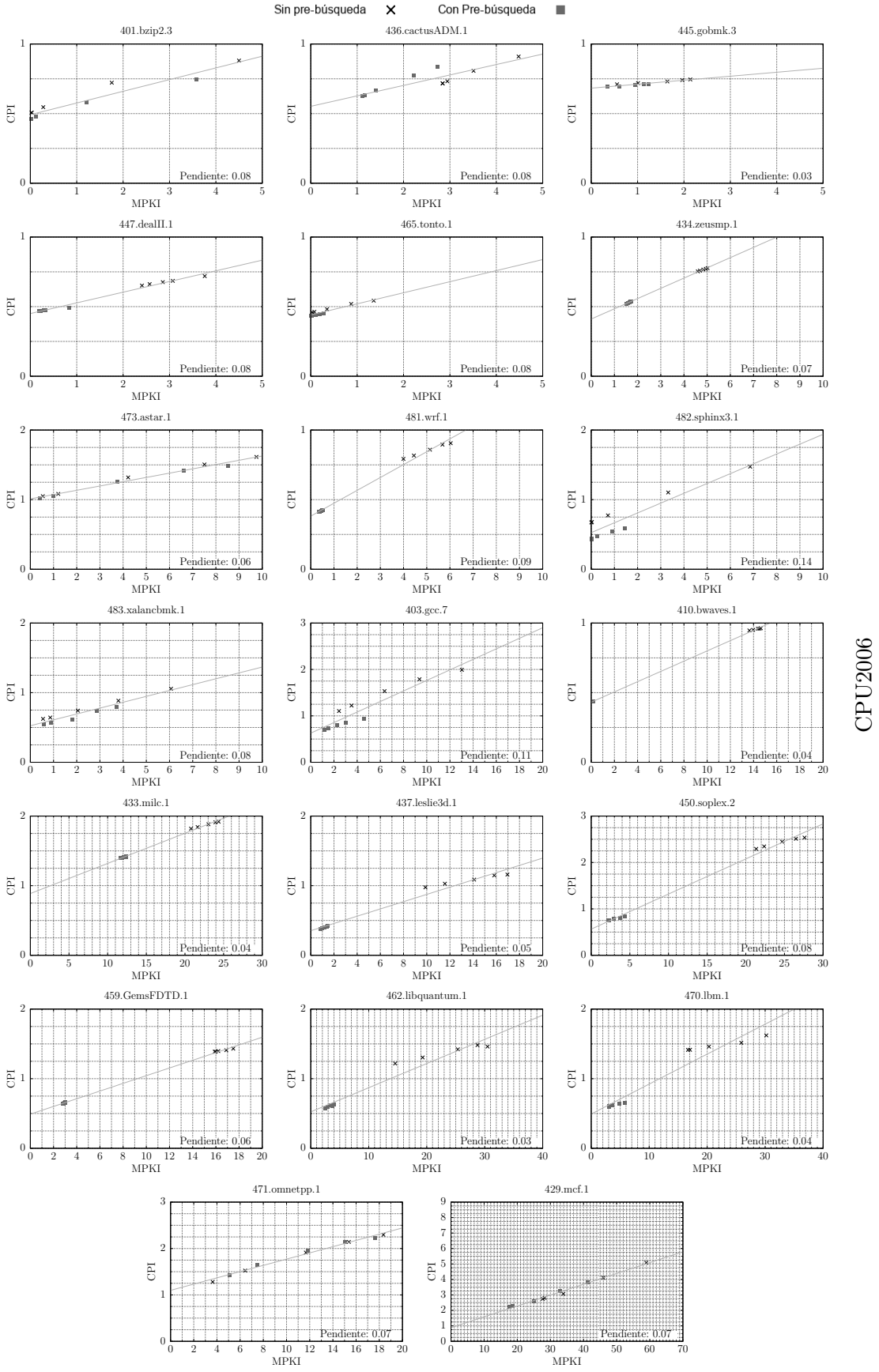


Figura 4.6: Relación entre MPKI3 y CPI de las parejas programa-entrada de CPU2006 seleccionadas.

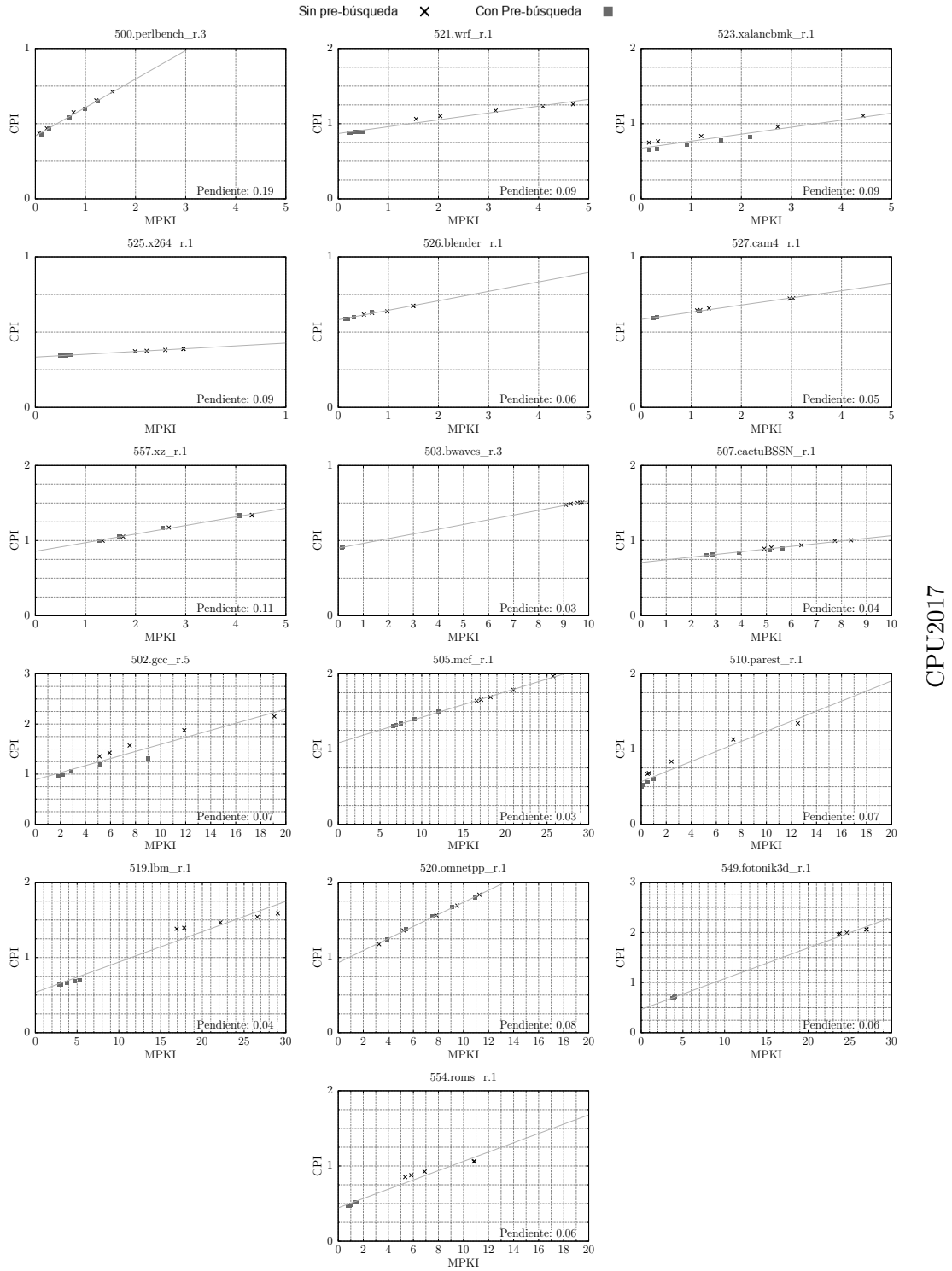


Figura 4.7: Relación entre MPKI3 y CPI de las parejas programa-entrada de CPU2017 seleccionadas.

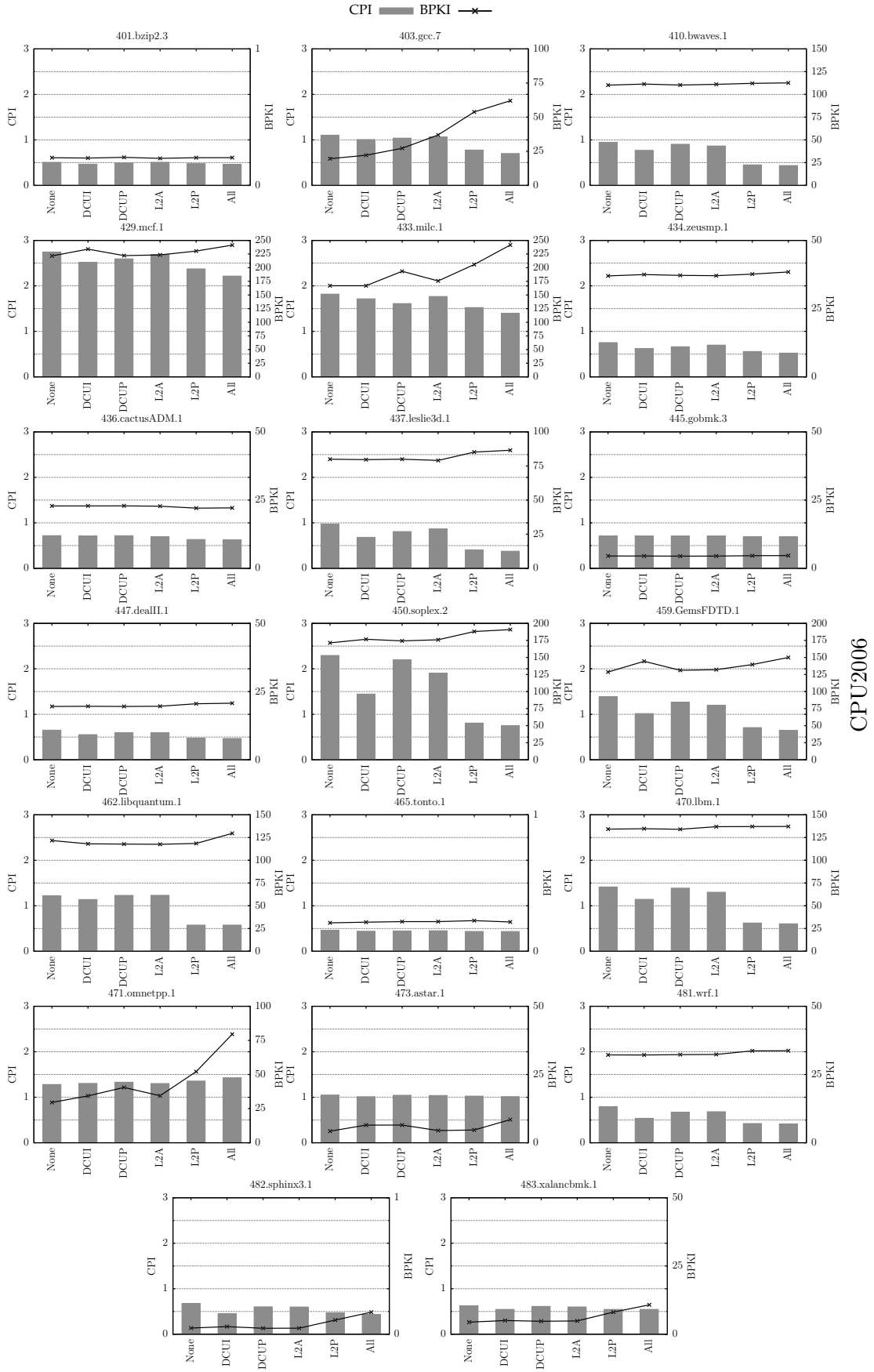


Figura 4.8: Impacto de los prebuscadores hardware en el CPI y BPKI de las parejas programa-entrada de CPU2006 seleccionadas.

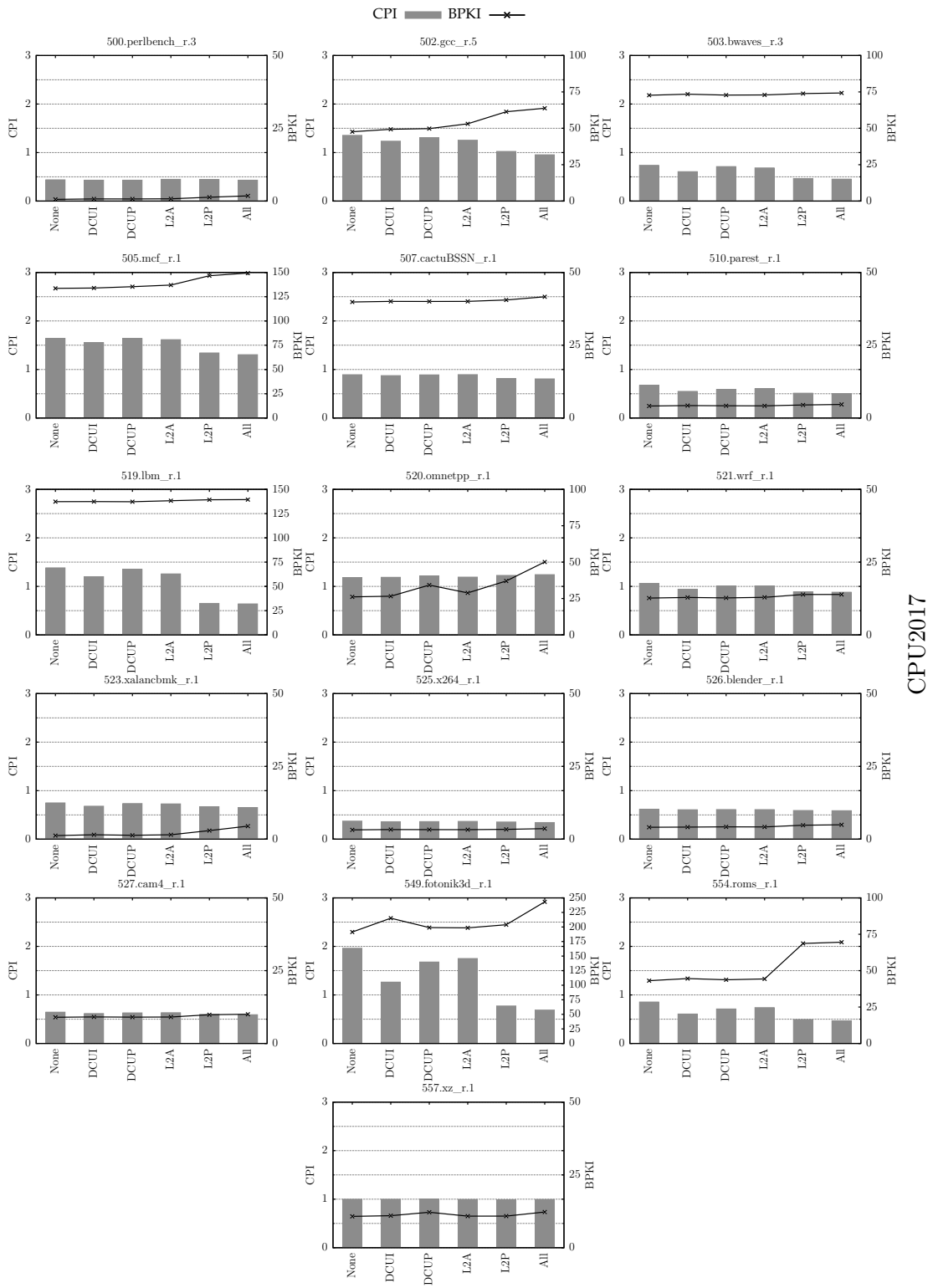


Figura 4.9: Impacto de los prebuscadores hardware en el CPI y BPKI de las parejas programa-entrada de CPU2017 seleccionadas.

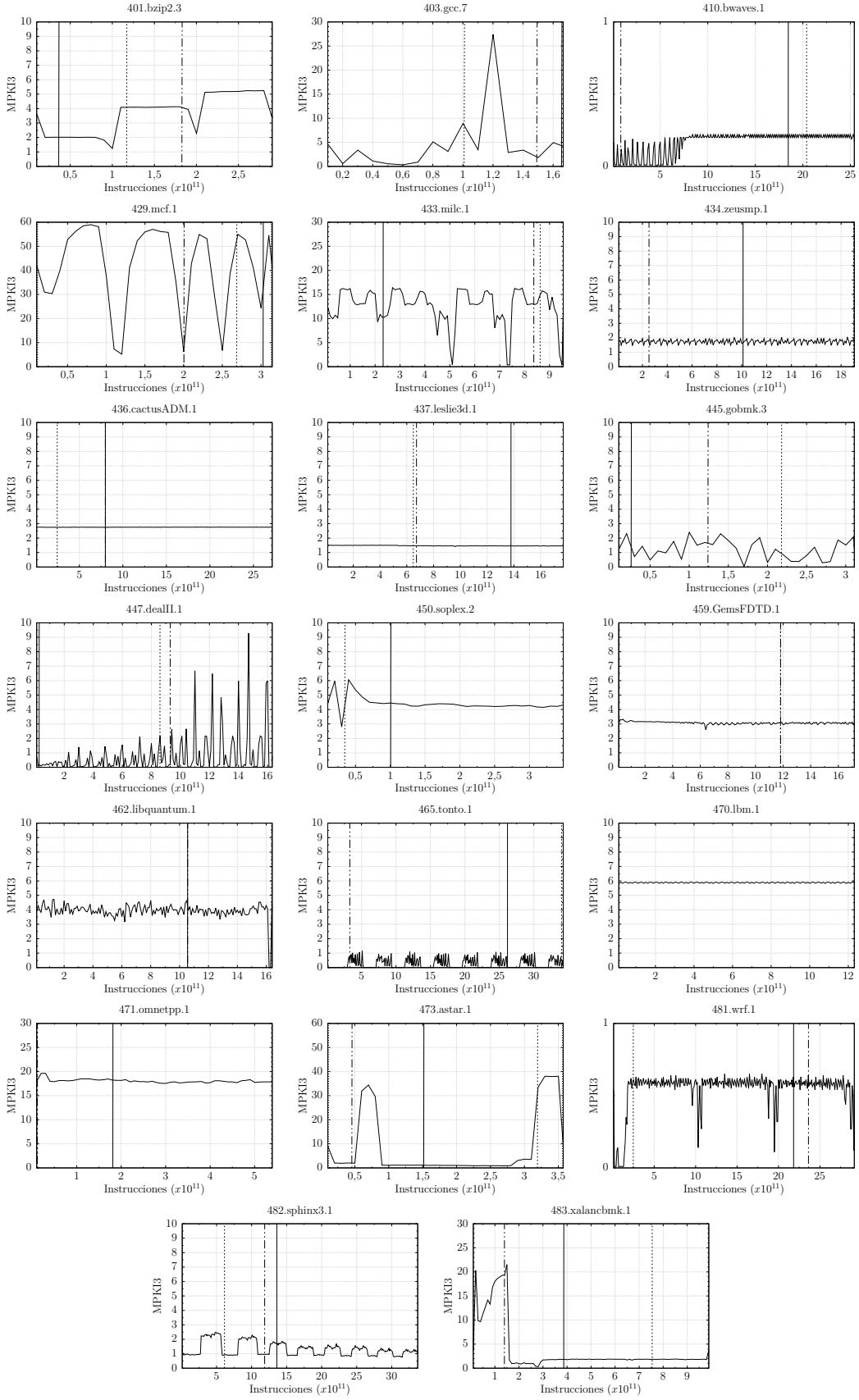
4.5. Evolución temporal de los programas

En esta sección se presentan los resultados de caracterización de la evolución temporal de los programas a lo largo de su ejecución. Las figuras 4.10 y 4.11 se muestran MPKI3 (eje Y) para cada millón de instrucciones (eje X). Esto nos permite conocer las distintas fases por las que pasa un programa y nos ayuda a seleccionar puntos de simulación.

En la misma gráfica se han incluidos los tres primeros puntos de simulación obtenidos mediante la metodología *Simpoint* en forma de líneas verticales con distintos patrones. La línea continua corresponde al punto de simulación más representativo, la punteada al segundo punto más representativo y la rallada al tercero. El grosor de las líneas no es proporcional al número de instrucciones que representan. Se ha incrementado para mejorar su visibilidad.

Los puntos seleccionados por *Simpoint* son a veces representativos de las distintas fases del programa y otras no. Como ejemplo de estos últimos, podemos observar que `401.bzip2` pasa claramente por tres fases con transitorios entre ellas. El primer punto de *Simpoint* está en la primera fase y los dos siguientes en la segunda. No hay ningún punto en la tercera.

Este análisis pone de manifiesto las limitaciones de esta metodología para obtener puntos representativos de la ejecución de un programa desde el punto de vista del uso de la jerarquía de memoria. El problema es todavía mayor si pensamos que la mayoría de los trabajos de investigación que usan esta metodología únicamente usan el primer intervalo.



CPU2006

Figura 4.10: Evolución temporal del MPKI3 y *Simpoints* de las parejas programa-entrada de los CPU2006 seleccionadas.

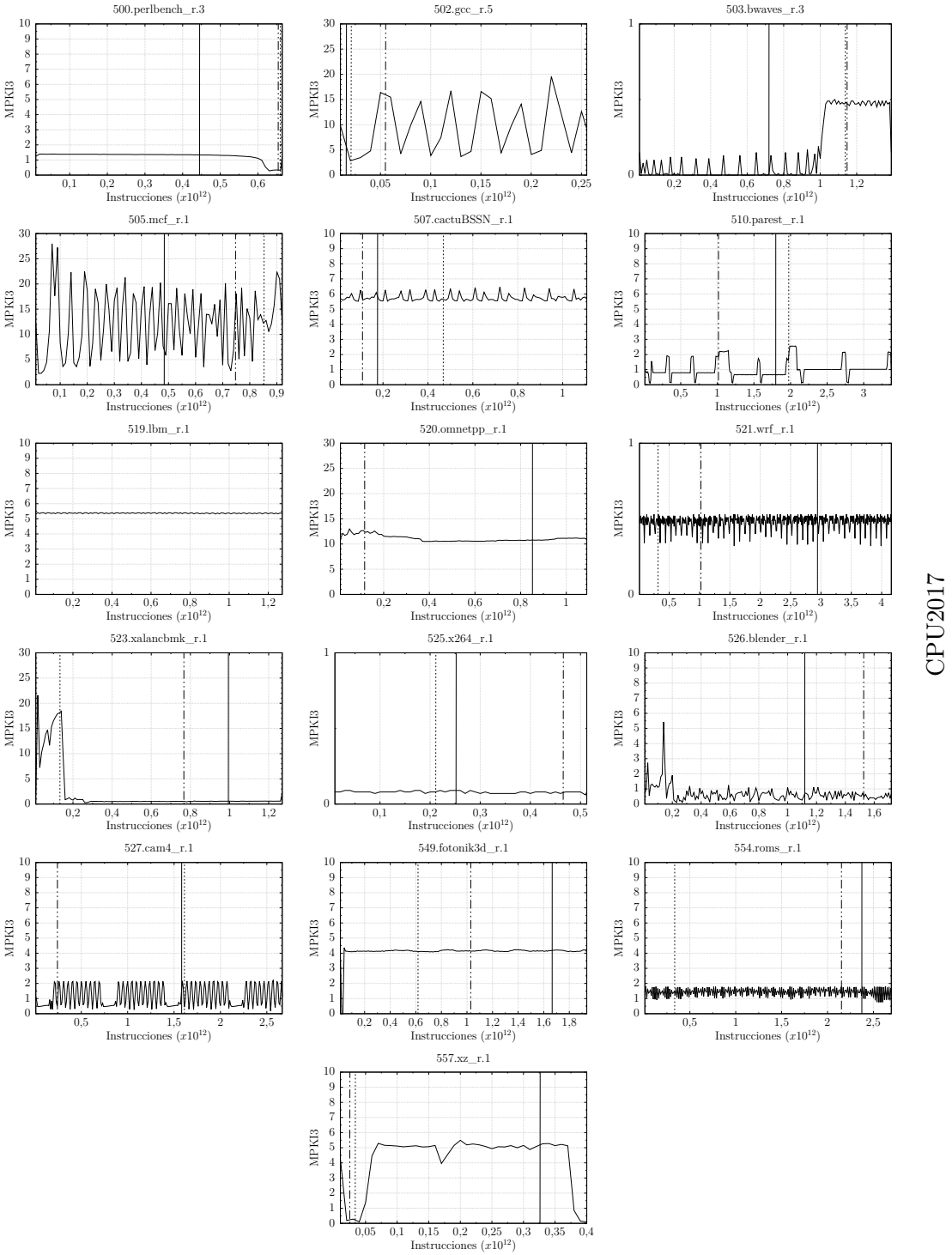


Figura 4.11: Evolución temporal del MPKI3 y *Simpoints* de las parejas programa-entrada de los CPU2017 seleccionadas.

Capítulo 5

Conclusiones y líneas abiertas

5.1. Conclusiones técnicas

El objetivo de este proyecto ha sido caracterizar el comportamiento en la LLC de las suites SPEC CPU2006 y CPU2017.

Para ello se han utilizado contadores hardware para medir los eventos que ocurren en el procesador, y herramientas de *Intel* y coloreo de pagina para limitar la cantidad de memoria disponible. Además, la caracterización se ha realizado sobre un procesador de última generación, el *Intel Xeon Gold 5120*.

Para la caracterización se han realizado los siguientes experimentos:

- Se han identificado aquellas parejas programa-entrada que son intensivas en memoria.
- Se ha medido el impacto del tamaño de la memoria cache en las prestaciones de los programas.
- Se ha analizado el impacto de los distintos prebuscadores hardware en el rendimiento de los programas, y en el ancho de banda con memoria.
- Se ha obtenido el comportamiento temporal de los programas a lo largo de su ejecución. Esto nos ha permitido detectar sus distintas fases para seleccionar sus partes representativas.

A continuación se resumen las principales conclusiones que podemos extraer de esta caracterización.

Una parte importante de las parejas programa-entrada tienen ratios de fallos muy bajas en el segundo y tercer nivel de caches, incluso con un tamaño pequeño de LLC y sin prebúsqueda hardware. La demanda de recursos de la jerarquía en CPU2017 es menor que en CPU2006.

Ofrecemos una clasificación de los programas que sí usan los niveles altos de la jerarquía según su sensibilidad al tamaño de LLC y a la prebúsqueda hardware. La prebúsqueda hardware es muy efectiva reduciendo los fallos en LLC en una buena parte de los programas, incluso con los tamaños más pequeños de LLC. El aumento del tamaño de LLC también es efectivo reduciendo la tasa de fallos en LLC para muchos programas.

El mejor prebuscador de los implementados en el procesador Intel Skylake SP es L2P. Este prebuscador es el responsable de más del 90 % de la mejora conseguida al aplicar prebúsqueda en la mayoría de los programas.

Los prebuscadores son muy precisos. En general, aumentan poco el número de bytes leídos de memoria principal.

El análisis temporal de las programas muestra como la utilización de *Simpoint* no garantiza obtener puntos de simulación representativos desde el punto de vista del uso de la jerarquía de memoria.

5.2. Problemas encontrados

Durante el desarrollo del proyecto se han encontrado distintos escollos. Algunos de ellos se resolvieron con relativa facilidad y otros han supuesto graves problemas para el desarrollo del mismo. Por ejemplo:

- La documentación oficial de Linux se encuentra desactualizada. Además, la organización del propio código de Linux dificulta encontrar determinadas funciones.
- No se ha encontrado una herramienta que permitiera leer contadores hardware de un evento cada n ocurrencias de un otro. Dicha herramienta se ha tenido que desarrollar (*Perf++*).
- La función hash de asignación de banco de Intel ha cambiado respecto a lo que muestra la literatura. Esto ha imposibilitado el uso de coloreo de página para el analizar la respuesta a reducción del tamaño de LLC.

5.3. Líneas abiertas

Durante la realización del trabajo han aparecido líneas que no han podido ser abordadas, ya que excedían el alcance y el tiempo disponible. Destaco las tres siguientes:

- Desarrollar una metodología que permita clasificar y obtener puntos de simulación de los programas según el comportamiento en la jerarquía de memoria.
- Extender la caracterización a los programas multi-hilo que componen la suite SPEC CPU2017.
- Con un horizonte más amplio, usar la información proporcionada por la caracterización para desarrollar técnicas implementadas en software para la gestión y asignación de recursos hardware compartidos por núcleos/hilos. Este será el objetivo principal de la tesis doctoral que he iniciado.

5.4. Conclusiones personales

Este proyecto me ha permitido aplicar las aptitudes y conocimientos adquiridos durante la carrera y el consiguiente máster. Además, he aprendido herramientas para trabajar con los contadores hardware, he conocido el estado del arte en los procesadores comerciales modernos y he ampliado mis conocimeintos sobre el kernel de Linux. También he adquirido conocimientos transversales relacionados con la investigación como es la escritura de artículos científicos.

Por último destacar que este trabajo me ha servido para empezar mis estudios de doctorado, escribir mis primeros artículos científicos y presentar este proyecto ante otros

estudiantes en la escuela de verano *ACACES* y otros investigadores en las jornadas *SARTECO*. Todo ello hace que me sienta satisfecho con el proyecto desarrollado.

Capítulo 6

Bibliografía

[1] Intel® 64 and IA-32 architectures optimization reference manual. page 788.

[2] Jorge Albericio, Pablo Ibáñez, Víctor Viñals, and José M. Llabería. The reuse cache: Downsizing the shared last-level cache. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-46, pages 310–321, New York, NY, USA, 2013. ACM.

[3] Gorka Irazoqui Apecechea, Thomas Eisenbarth, and Berk Sunar. Systematic reverse engineering of cache slice selection in intel processors. *IACR Cryptology ePrint Archive*, 2015:690, 2015.

[4] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The parsec benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, PACT '08, pages 72–81, New York, NY, USA, 2008. ACM.

[5] Standard Performance Evaluation Corporation. SPEC CPU 2017. `https://www.spec.org/cpu2017/`, 2017. [Online; accessed 14-Feb-2018].

[6] Arnaldo Carvalho de Melo. The new linux 'perf' tools. 2010.

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

[8] John L. Henning. Spec cpu suite growth: An historical perspective. *SIGARCH Comput. Archit. News*, 35(1):65–68, March 2007.

[9] Intel. Are noisy neighbors in your data center keeping you up at night? Technical report.

[10] Intel. Improving real-time performance by utilizing cache allocation technology. Technical report, 2015.

[11] Aamer Jaleel. Memory characterization of workloads using instrumentation-driven simulation.

[12] N. P. Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *[1990] Proceedings. The 17th Annual International Symposium on Computer Architecture*, pages 364–373, May 1990.

[13] Kenneth C. Knowlton. A fast storage allocator. *Commun. ACM*, 8(10):623–624, October 1965.

[14] Wendy Korn and Moon S. Chang. Spec cpu2006 sensitivity to memory page sizes. *SIGARCH Comput. Archit. News*, 35(1):97–101, March 2007.

[15] Ankur Limaye and Tosiron Adegbija. A workload characterization of the spec cpu2017 benchmark suite, 04 2018.

[16] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee. Last-level cache side-channel attacks are practical. In *2015 IEEE Symposium on Security and Privacy*, pages 605–622, May 2015.

[17] Clémentine Maurice, Nicolas Le Scouarnec, Christoph Neumann, Olivier Heen, and Aurélien Francillon. Reverse engineering intel last-level cache complex addressing using performance counters. In Herbert Bos, Fabian Monrose, and Gregory Blanc, editors, *Research in Attacks, Intrusions, and Defenses*, pages 48–65, Cham, 2015. Springer International Publishing.

[18] Shirley V. Moore. A comparison of counting and sampling modes of using performance monitoring hardware. In Peter M. A. Sloot, Alfons G. Hoekstra, C. J. Kenneth Tan, and Jack J. Dongarra, editors, *Computational Science — ICCS 2002*, pages 904–912, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[19] R. Panda, S. Song, J. Dean, and L. K. John. Wait of a decade: Did spec cpu 2017 broaden the performance horizon? In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 271–282, Feb 2018.

[20] Erez Perelman, Greg Hamerly, Michael Van Biesbrouck, Timothy Sherwood, and Brad Calder. Using simpoint for accurate and efficient simulation. *SIGMETRICS Perform. Eval. Rev.*, 31(1):318–319, June 2003.

[21] Lior Rokach. *A survey of Clustering Algorithms*, pages 269–298. Springer US, Boston, MA, 2010.

[22] T. Sherwood, E. Perelman, and B. Calder. Basic block distribution analysis to find periodic behavior and simulation points in applications. In *Proceedings 2001 International Conference on Parallel Architectures and Compilation Techniques*, pages 3–14, 2001.

[23] Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. *Operating System Concepts*. Wiley Publishing, 8th edition, 2008.

[24] William Stallings. *Computer Organization and Architecture: Designing for Performance (7th Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2005.

[25] Lu Peng Tribuvan Kumar Prakash. Performance characterization of spec cpu2006 benchmarks on intel core 2 duo processor.

[26] Zhipeng Wei, Zehan Cui, and Mingyu Chen. Cracking intel sandy bridge's cache hash function. 08 2015.

[27] T. F. Wenisch, R. E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J. C. Hoe. Simflex: Statistical sampling of computer system simulation. *IEEE Micro*, 26(4):18–31, July 2006.

[28] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The splash-2 programs: characterization and methodological considerations. In *Proceedings 22nd Annual International Symposium on Computer Architecture*, pages 24–36, June 1995.

[29] Yuval Yarom, Qian Ge, Fangfei Liu, Ruby B. Lee, and Gernot Heiser. Mapping the Intel last-level cache. http://eprint.iacr.org/, September 2015.

[30] Q. Zou, J. Yue, B. Segee, and Y. Zhu. Temporal characterization of spec cpu2006 workloads: Analysis and synthesis. In *2012 IEEE 31st International Performance Computing and Communications Conference (IPCCC)*, pages 11–20, Dec 2012.

[31] Sarah Bird Φ, Aashish Phansalkar Φ, Lizy K. John Φ, Alex Mericas A, and Rajeev Indukuru A. Performance characterization of spec cpu benchmarks on intel's core microarchitecture based processor.

Anexos

Apéndice A

Jerarquía de Memoria

La memoria de un procesador esta organizada de forma jerárquica, compuesta principalmente por cuatro niveles:

1. Los registros del procesador.
2. La memoria cache, normalmente compuesta por tres niveles (L1, L2 y L3 o LLC).
3. La memoria principal.
4. Memoria externa al sistema como los discos duros.

El objetivo final de dicha jerarquía es mejorar su rendimiento a medida que esta se acerca al procesador, ver fig. A.1, a cambio de un mayor coste y un menor tamaño.

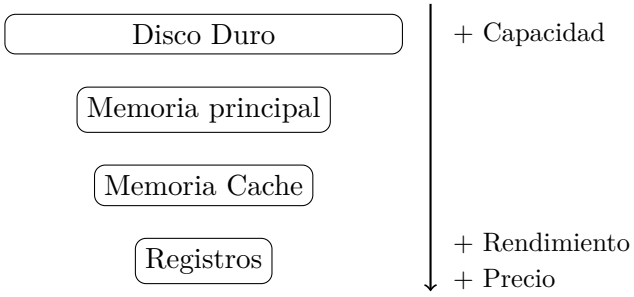


Figura A.1: Características de la jerarquía según nivel

Entrando ligeramente en la implementación una memoria cache suele estar implementada mediante una CAM *Cache Addressable Memory*, y está dividida en dos secciones, ver A.2:

– **TAGs**: identificador que permite solucionar conflictos de direccionamiento.

– **Datos**: información asociada.

A.1. Memoria Cache

La cache es una memoria de alta velocidad que permite almacenar aquellos datos que son frecuentemente utilizados por el procesador, de forma que el procesador los tenga disponibles en unos pocos ciclos de reloj. En la siguiente sección se describirá de forma breve los elementos más importantes dentro del diseño de las memorias cache.

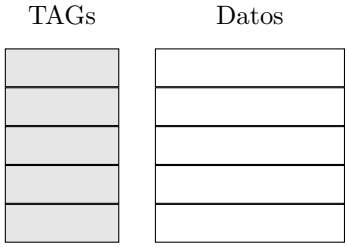


Figura A.2: Estructura de una memoria cache con vector de *TAGs* y de datos

A.1.1. Estructura

Las memorias caches de los procesadores de alto rendimiento se encuentran normalmente dividas en tres niveles (L1, L2 y LLC) según su velocidad y tamaño, ver fig. A.3. La utilización de múltiples niveles permite aumentar el tamaño disponible de la cache, sin sacrificar rendimiento.

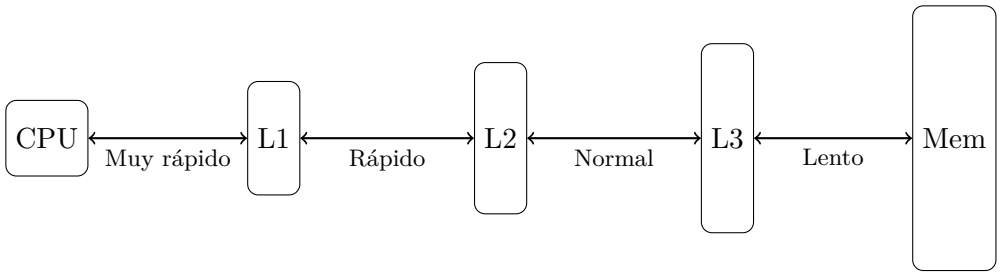
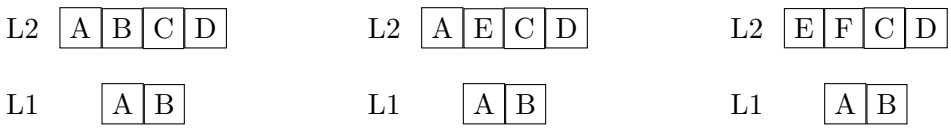


Figura A.3: Niveles de cache y velocidad asociada

Inclusión

Una estructura multinivel de cache da lugar a diversas configuraciones según el tratamiento que se hace de los datos en cada nivel según se encuentren en los niveles inferiores o no, ver fig. A.4:

- **Inclusiva**: los datos del nivel *N-1* se encuentran también en el nivel *N*.
- **No inclusiva**: los datos del nivel *N-1* pueden encontrarse o no en el nivel *N*.
- **Exclusiva**: los datos del nivel *N-1* no se encuentran en el nivel *N*.



(a) Inclusiva **(b)** No Inclusiva **(c)** Exclusiva

Figura A.4: Inclusividad gráfica de una memoria cache multinivel

Compartición

La aparición de procesadores multi-núcleo ha provocado que los distintos niveles de la jerarquía de memorias cache puedan, o no, estar compartidos entre varios núcleo, ver fig. A.5:

– **Privada**: solo puede acceder el núcleo asociado a dicha cache

– **Compartida**: más de un núcleo puede acceder a la cache.

Por ejemplo, en los procesadores *Intel Xeon* los niveles L1 y L2 de cache suelen ser privados a cada núcleo y la L3 compartida entre todos ellos.

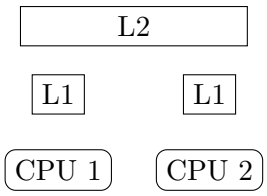


Figura A.5: Jerarquía de cache con dos cores, dos L1 privada y una L2 compartida

Bancos de memoria

En los actuales procesadores la cache de último nivel se suele encontrar compartida entre varios núcleos. Para conseguir distribuir de forma equitativa la latencia de acceso entre los distintos núcleos, y por restricciones de construcción, la LLC se encuentra dividida en bancos[1] situados cerca de cada núcleo e interconectados entre ellos, ver fig. A.6.

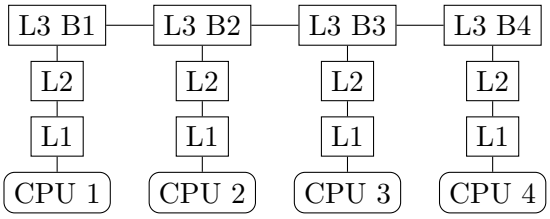


Figura A.6: Jerarquía de cache con cuatro cores con L1 y L2 privada, y una L3 compartida dividida en cuatro bancos

Correspondencia

Existen tres posibles funciones de correspondencia para definir la ubicación de un bloque en la memoria cache, ver fig A.7:

– **Directa**: el bloque de memoria solo puede ubicarse en una determinada linea de cache.

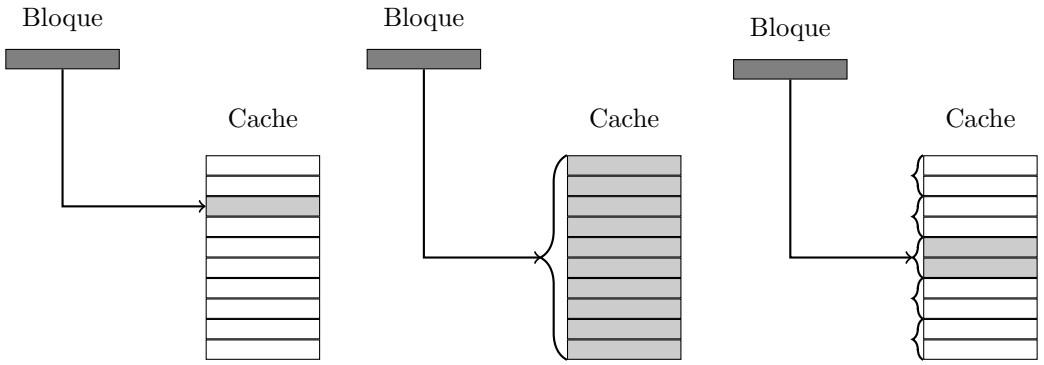
– **Completamente asociativa**: el bloque puede situarse en cualquier linea de la memoria cache.

– **Por conjuntos**: la memoria esta divida en conjuntos de x lineas de cache (o vías), y el bloque puede situarse en cualquier línea de un conjunto.

Función de Hash

La selección del conjunto al cual va un bloque de memoria se realiza a partir de su dirección:

[1] También llamados por Intel *Slice*



(a) Directa **(b)** Completamente Asoc. **(c)** Por conjuntos

Figura A.7: Correspondencia de un bloque a la memoria cache según tipo. Las lineas grises representa donde puede ir el bloque y los corchetes los conjuntos de líneas

- **Posición en el bloque (offset)**: 6 bits de menor peso.
- **Banco**: $\log_2(NdeBancos)$ bits.
- **Conjuntos (o set)**: $\log_2(TamañoCache) - \log_2(TamañoBloque) - Asociatividad$ bits de conjunto.
- **TAG**: bits restantes

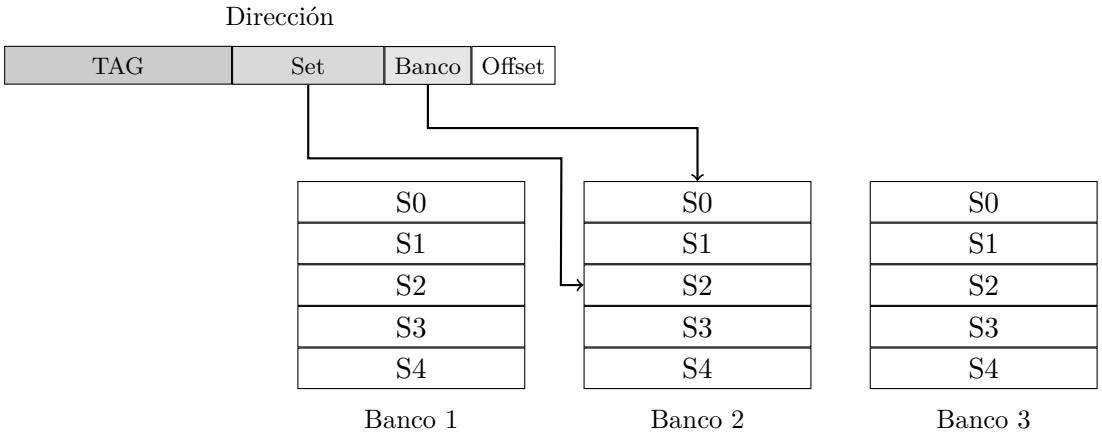


Figura A.8: Descomposición de la dirección de memoria de un bloque para la selección del conjunto y del banco a donde va dicho bloque

Por ejemplo para un procesador de 16MB de LLC, 8 bancos de memoria, asociativadad 16, tamaño de bloque 64bits y dirección de 48bits, la partición de su dirección seria:

- **Posición en el bloque**: bits 0 a 5.
- **Banco**: bits 6 a 8.
- **Conjunto**: bits 9 a 20.
- **TAG**: bits 21 a 48.

A.1.2. Algoritmo de Remplazo

El espacio en cache es limitado y por lo tanto si un bloque va a parar a un conjunto sin vías libres, es necesario expulsar a algún bloque anterior. La selección del bloque a expulsar es realizada por el algoritmo de reemplazo que sigue una determinada heurística, como por ejemplo:

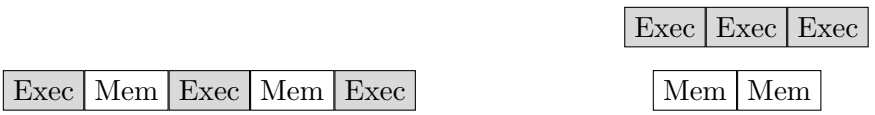
- **Aleatorio**: el bloque expulsado es seleccionado al azar.
- **LRU (Last Recently Used)**: se sustituye al bloque que hace más tiempo que no ha sido referenciado[2].
- **LFU (Leas Frequently Used)**: se expulsa el bloque con menos referencias.

A.1.3. Prebúsqueda

La latencia de la jerarquía de memoria es considerablemente mayor que la velocidad del procesador. Con el fin de disminuir la diferencia de velocidades se intentan prebuscar los datos antes de que estos sean necesitados por el programa en ejecución. En la figura A.9 se plasma de manera gráfica como la ausencia de prebúsqueda provoca una parada en la ejecución del programa, *Exec*, para ir a memoria, *Mem* (sub-figura a). Sin embargo la prebúsqueda mitiga dichas paradas al ocurrir los accesos a memoria de forma simultanea a la ejecución del procesador.

Algunas de las técnicas más utilizadas en la pre-búsqueda son:

- **Next-K**: pre-busca las K siguientes líneas a partir de un fallo. El más extendido es el *stride prefetcher*, que prebusca a partir de un patrón. Por ejemplo, si la secuencia de accesos ha sido: A, A + Q, A + 2Q, entonces pre-buscara A + 3Q, A + 4Q, ….
- **Spatial Prefetcher**: asume localidad espacial y prebusca las lineas vecinas al fallo actual.
- **Temporal Prefetcher**: asume que la direcciones recientemente buscadas volverán a ser pedidas, por tanto prebusca según el historial actual de fallos.



(a) Sin pre-búsqueda **(b)** Con pre-búsqueda

Figura A.9: Diferencia entre ejecución con pre-búsqueda y sin ella

[2]Una referencia es una lectura o escritura en dicho bloque

Apéndice B

Skylake-SP Scales Server Systems

En el siguiente anexo se presenta la nueva familia de procesadores para servidores *Skylake-SP*. Estos procesadores cuentan con importantes cambios respecto a la familia *Skylake* de escritorio, como son la inclusión de unidades de calculo vectorial de 512 bits, un nuevo subsistema de memoria y un nuevo sistema de interconexión. Además estos nuevos procesadores están pensados para la integración de aceleradores como FPGAs.

B.1. Características generales

Los procesadores de la familia *Skylake-SP* están construidos en 14nm, pueden integrar hasta 28 cores por procesador, 6 canales de memoria DDR4 que permiten un total de 1.5TB de memoria, tres conectores UPI, 48 PCIe 3.0 y cuatro lineas DMI3.0. Siendo posible conectar hasta ocho procesadores en un mismo socket sin utilizar controladores de terceros.

Los nuevos procesadores incluyen la nueva unidad de calculo vectorial *AVX-512* que implementa operaciones vectoriales de hasta 512-bits. Estos procesadores también doblan la capacidad de calculo de sus predecesores, pudiendo realizar hasta 32 operaciones de doble precisión o 64 operaciones de precisión simple por ciclo.

Con el fin de mejorar la eficiencia energética de los procesadores, estos cuentan con tres curvas *DVFS (Dynamic Voltage and Frequency Scale)*. *DVFS* es una técnica de ahorro energético que consiste en limitar la frecuencia del procesador para disminuir el consumo del mismo. Estás tres curvas se activan según el tipo de calculo que se realiza.

B.2. Subsistema de memoria

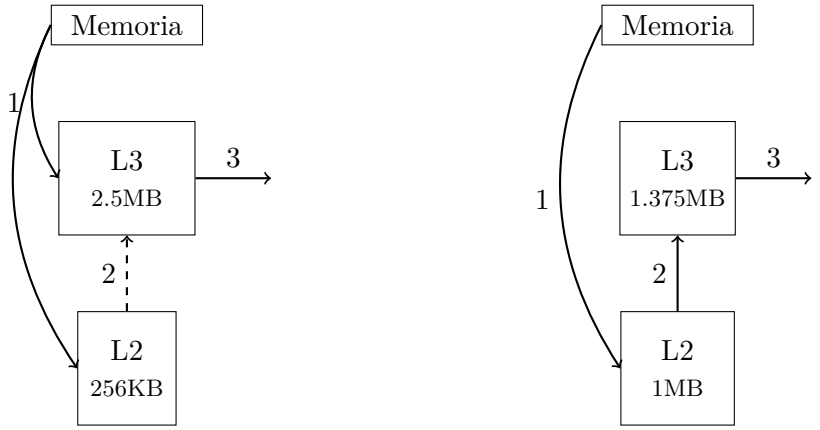
La jerarquía de memoria cache ha sido mejorada con el fin de añadir soporte para las operaciones vectoriales *AVX-512* y mejorar el rendimiento general de los procesadores. Los buses de memoria han sido ampliadas a 512-bits permitiendo a la L1 servir dos lecturas de 64-bits y una escritura de 64bits por ciclo.

La L2 ha cuadruplicado su tamaño de 256KB a 1MB con asociatividad 16, los fallos de cache son traídos de memoria principal a la L2 sin ser copiados en L3 y la prebúsqueda actúa directamente sobre L2.

La cache L3 o LLC pasa a ser mayormente no-inclusiva, actuando como almacenamiento de victimas, los desalojos de L2 son copiados a L3. Además si una petición de L2 es compartida, dicha petición será escrita en L3, en la figura. B.1 se muestra una descripción del sistema. El

tamaño de L3 por banco ha disminuido de 2.5MB, de las generación anterior, a 1.375MB con asociatividad 11.

La interconexión de los distintos bancos de L3 ha pasado de un anillo doble, en las versiones anteriores, a una malla *mesh*. Este cambio permite reducir la latencia media de acceso de 8 ciclos a 5.3 ciclos.



(a) L3 Inclusiva (generaciones anteriores) **(b)** L3 Mayormente no-inclusiva (Skylake)

Figura B.1: Comparación entre el subsistema de memoria de Skylake y las generaciones anteriores. En *Skylake-SP* LLC es mayormente no inclusiva. Las peticiones de lectura (1) van directamente a memoria y las victimas de L2 van a L3 (2). Los datos compartidos entre múltiples cores pueden mantenerse en L3 (3).

Apéndice C

Análisis

En este apéndice se muestran todos los datos y gráficas obtenidos de los experimentos para la suite SPEC CPU2006 y CPU2017.

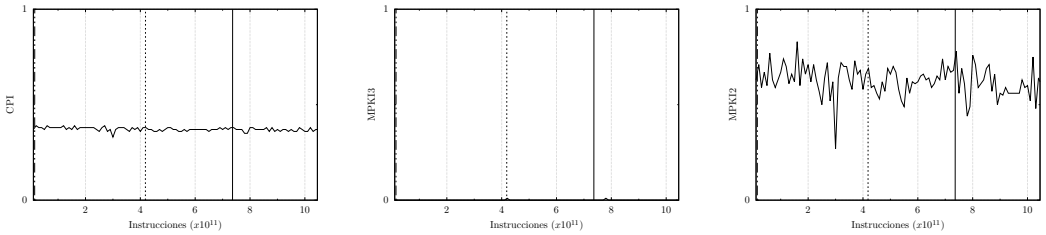
C.1. SPEC CPU2006

C.1.1. Caracterización General

Benchmark	Instr	Lect	Escr	MPKI1	MPKI2	MPKI3	CPI
400.perlbench.1	1052	26 %	12 %	7,52	0,58	0,01	0,38
400.perlbench.2	360	29 %	16 %	13,35	0,03	0,01	0,35
400.perlbench.3	656	28 %	9 %	2,70	0,22	0,02	0,31
401.bzip2.1	409	30 %	11 %	14,30	1,23	0,02	0,66
401.bzip2.2	174	24 %	11 %	12,18	1,67	0,01	0,47
401.bzip2.3	291	24 %	7 %	18,69	3,69	0,01	0,46
401.bzip2.4	532	27 %	12 %	8,30	0,60	0,02	0,51
401.bzip2.5	583	34 %	6 %	14,14	1,69	0,02	0,54
401.bzip2.6	327	29 %	11 %	14,18	1,35	0,02	0,63
403.gcc.1	69	23 %	15 %	34,49	5,02	0,38	0,70
403.gcc.2	140	24 %	13 %	21,62	2,64	0,44	0,69
403.gcc.3	123	18 %	20 %	43,88	2,30	0,31	0,61
403.gcc.4	91	20 %	17 %	33,66	2,61	0,30	0,58
403.gcc.5	100	20 %	17 %	38,17	3,05	0,48	0,60
403.gcc.6	137	19 %	17 %	38,57	3,20	0,53	0,64
403.gcc.7	168	22 %	14 %	36,08	2,94	1,07	0,70
403.gcc.8	149	19 %	19 %	39,04	3,51	0,60	0,70
403.gcc.9	54	24 %	13 %	19,05	1,72	0,24	0,67
410.bwaves.1	2554	37 %	7 %	19,61	0,16	0,14	0,44
416.gamess.1	109	32 %	11 %	8,14	0,00	0,00	0,6
416.gamess.2	85	33 %	9 %	10,32	0,00	0,00	0,33
416.gamess.3	371	31 %	8 %	5,47	0,01	0,00	0,30
429.mcf.1	315	31 %	9 %	123,23	50,33	17,48	2,20
433.milc.1	96	29 %	11 %	26,40	10,03	9,79	1,40
434.zeusmp.1	1915	21 %	8 %	22,08	1,30	1,07	0,52
435.gromacs.1	1948	36 %	13 %	11,75	0,03	0,00	0,45
436.cactusADM.1	2725	56 %	19 %	8,06	2,54	0,64	0,63
437.leslie3d.1	178	30 %	7 %	29,47	0,34	0,16	0,38
444.namd.1	2284	23 %	6 %	10,37	0,02	0,00	0,48
445.gobmk.1	23	25 %	13 %	12,65	0,27	0,02	0,76
445.gobmk.2	61	25 %	14 %	11,39	0,26	0,02	0,73
445.gobmk.3	31	24 %	10 %	11,50	0,12	0,01	0,69
445.gobmk.4	23	25 %	14 %	12,52	0,22	0,02	0,75
445.gobmk.5	33	25 %	14 %	10,77	0,17	0,02	0,71
447.dealII.1	1645	35 %	9 %	17,77	2,75	0,18	0,47
450.soplex.1	344	21 %	7 %	53,59	14,12	1,01	0,93
450.soplex.2	350	29 %	4 %	28,97	4,39	2,03	0,75
453.povray.1	940	33 %	15 %	24,95	0,00	0,00	0,41
454.calculix.1	4248	20 %	2 %	5,59	0,06	0,02	0,50
456.hmmer.1	860	42 %	15 %	6,80	0,03	0,00	0,36
456.hmmer.2	1823	41 %	15 %	4,73	0,01	0,00	0,37
458.sjeng.1	2260	22 %	8 %	2,67	0,38	0,26	0,56
459.GemsFDTD.1	1723	40 %	9 %	29,55	3,05	2,78	0,64
462.libquantum.1	1647	13 %	4 %	21,01	2,63	2,45	0,57
464.h264ref.1	495	41 %	11 %	4,08	0,12	0,00	0,33
464.h264ref.2	322	40 %	16 %	14,24	0,08	0,00	0,40
464.h264ref.3	2887	42 %	15 %	15,81	0,10	0,01	0,38
465.tonto.1	3443	24 %	10 %	10,38	0,16	0,01	0,43
470.lbm.1	1235	19 %	8 %	50,38	0,04	0,02	0,60
471.omnetpp.1	541	27 %	16 %	37,19	19,40	4,62	1,41
473.astar.1	359	32 %	9 %	31,22	6,31	0,32	1,02
473.astar.2	752	29 %	8 %	20,77	1,73	0,01	0,90
481.wrf.1	2931	24 %	6 %	11,92	0,22	0,14	0,41
482.sphinx3.1	3384	24 %	2 %	17,36	1,54	0,00	0,44
483.xalancbmk.1	994	29 %	6 %	27,81	3,82	0,28	0,54
AVG	933	28 %	11 %	21,46	2,99	0,87	0,61

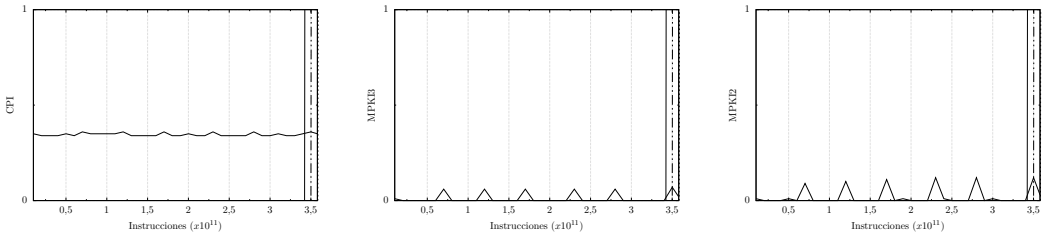
Cuadro C.1: Tabla resumen con los millones de instrucciones, porcentaje de instrucciones de lectura, porcentaje de instrucciones de escritura, MPKI en L1, MPKI en L2, MPKI en L3 y CPI de los programas de SPEC CPU2006

C.1.2. Evolución Temporal



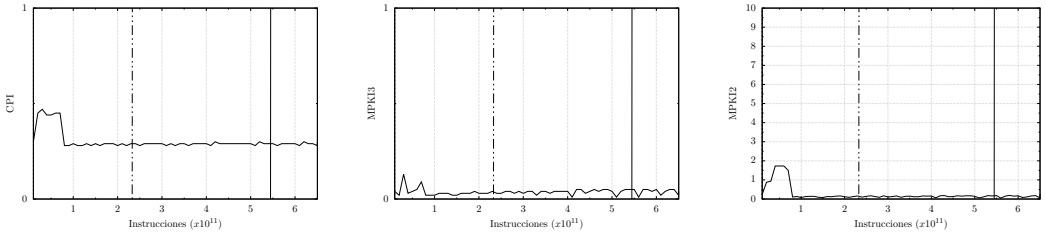
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.1: 400.perlbench.1



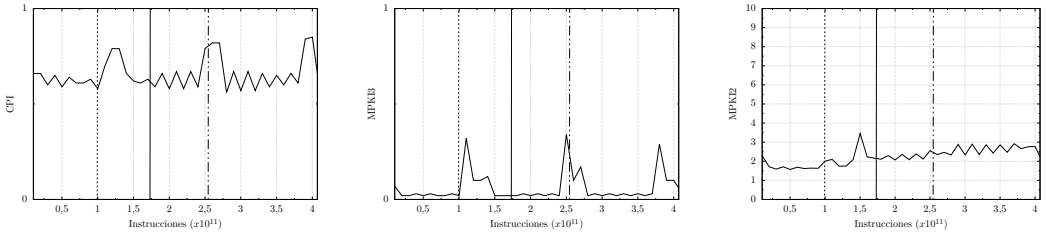
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.2: 400.perlbench.2



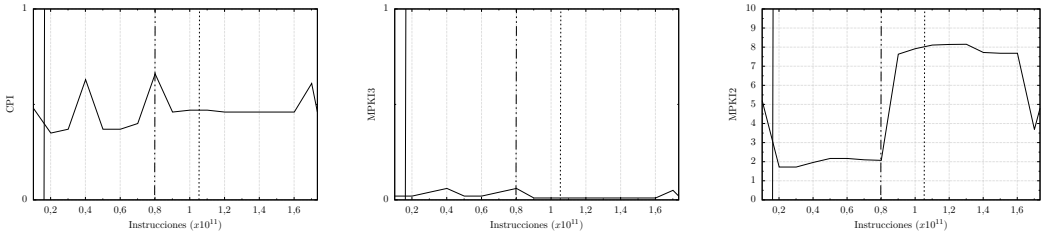
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.3: 400.perlbench.3



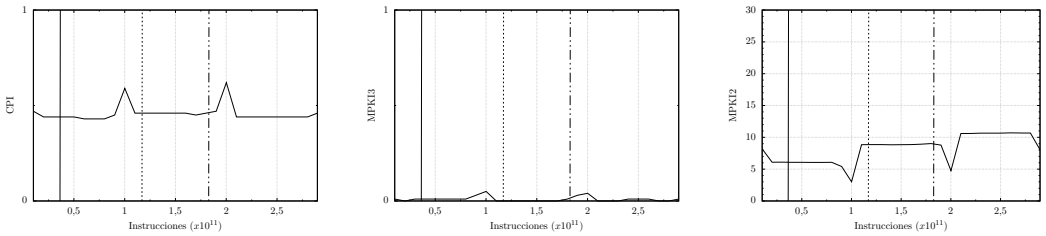
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.4: 401.bzip2.1



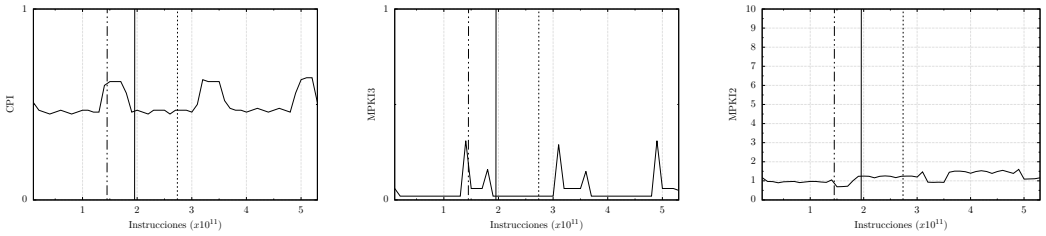
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.5: 401.bzip2.2



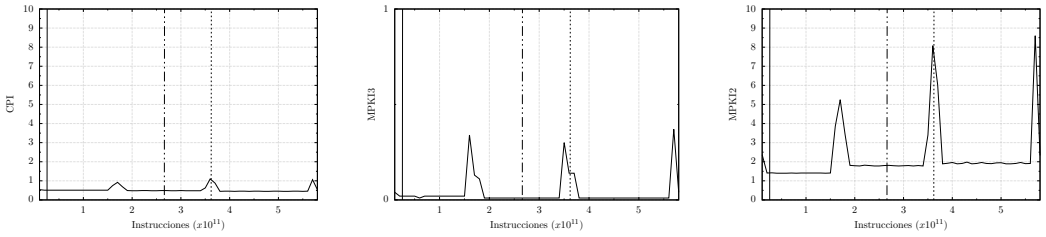
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.6: 401.bzip2.3



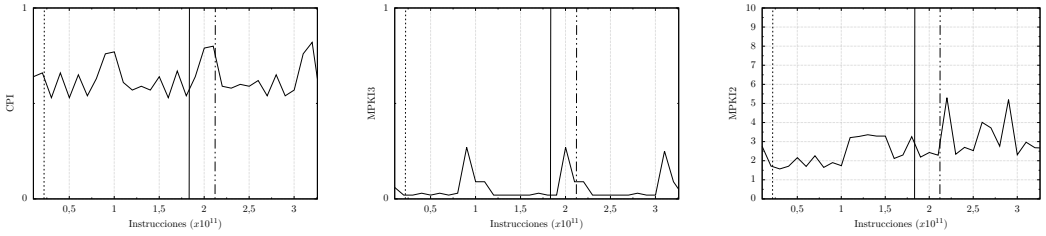
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.7: 401.bzip2.4



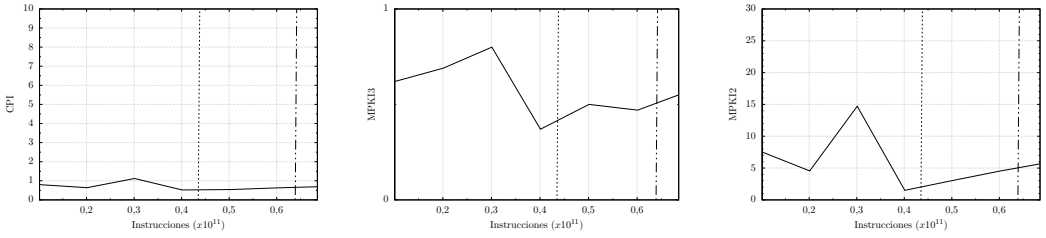
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.8: 401.bzip2.5



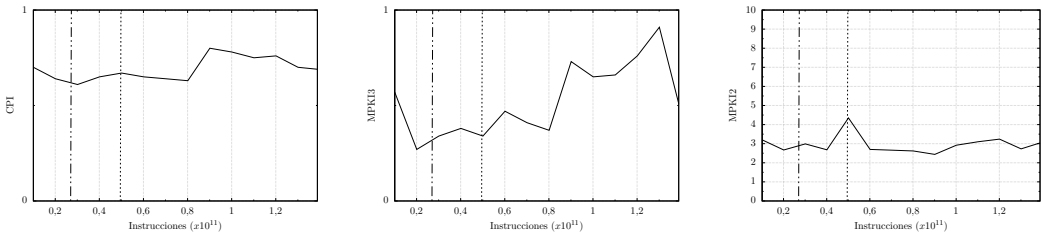
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.9: 401.bzip2.6



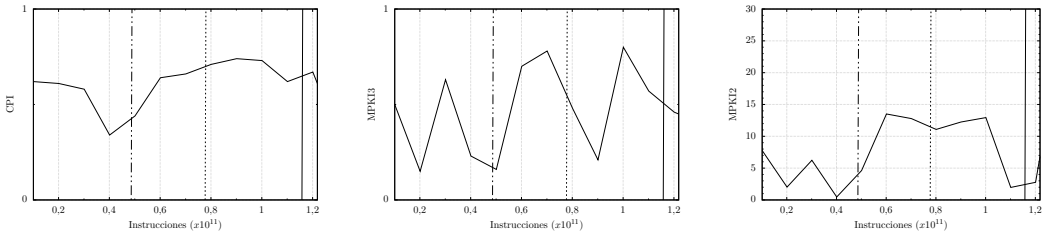
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.10: 403.gcc.1



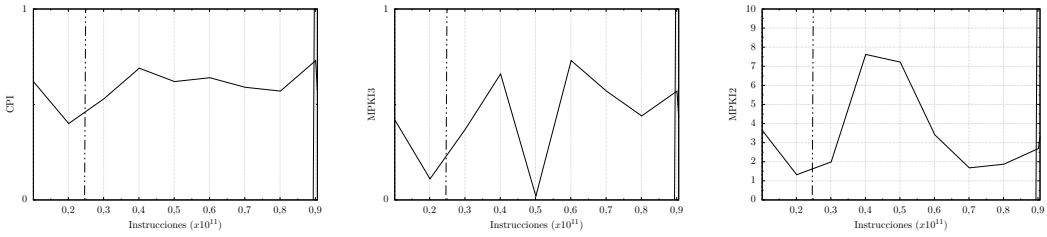
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.11: 403.gcc.2



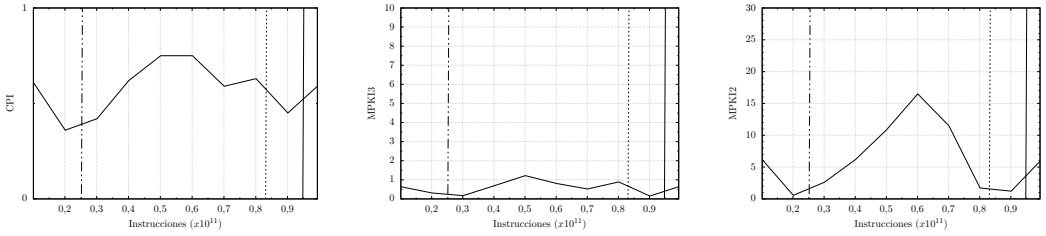
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.12: 403.gcc.3



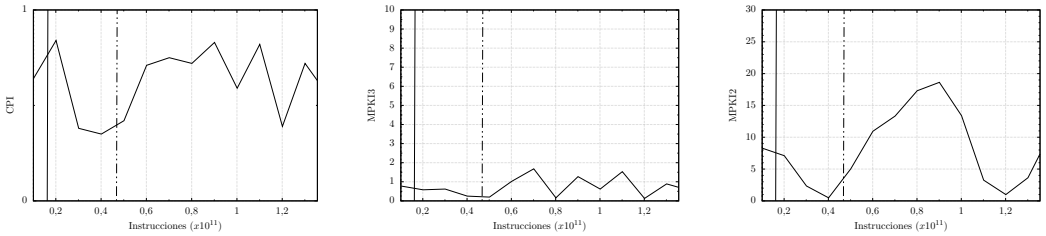
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.13: 403.gcc.4



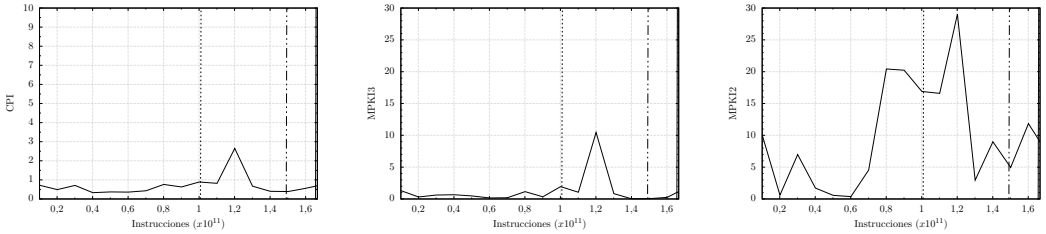
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.14: 403.gcc.5



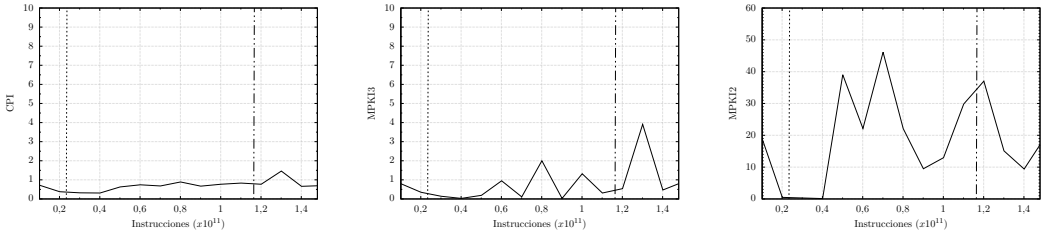
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.15: 403.gcc.6



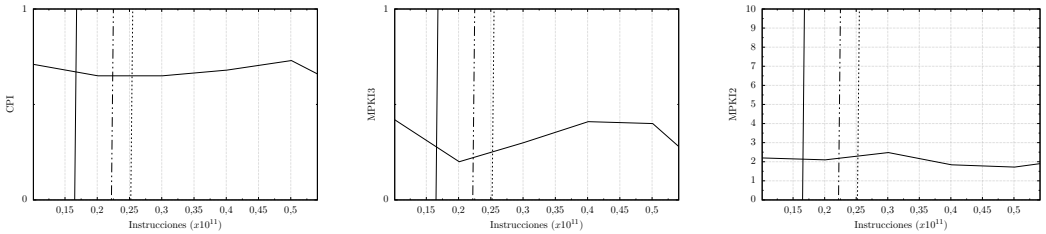
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.16: 403.gcc.7



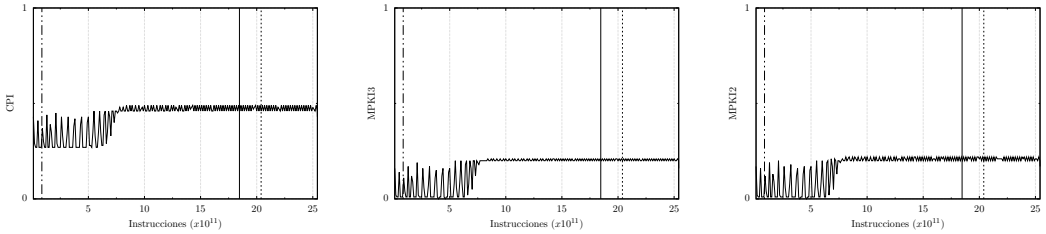
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.17: 403.gcc.8



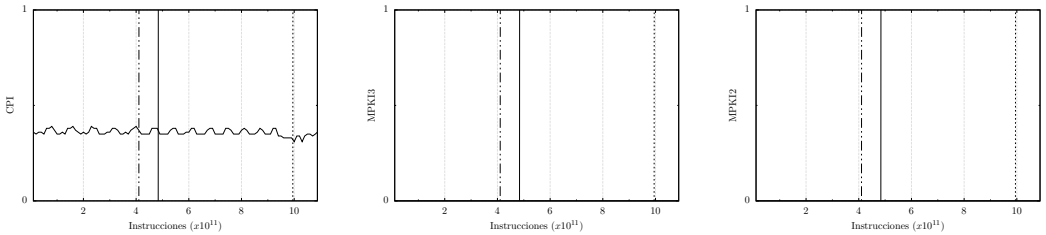
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.18: 403.gcc.9



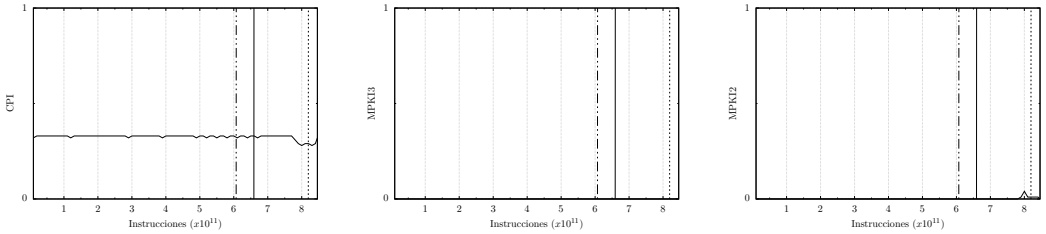
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.19: 410.bwaves.1



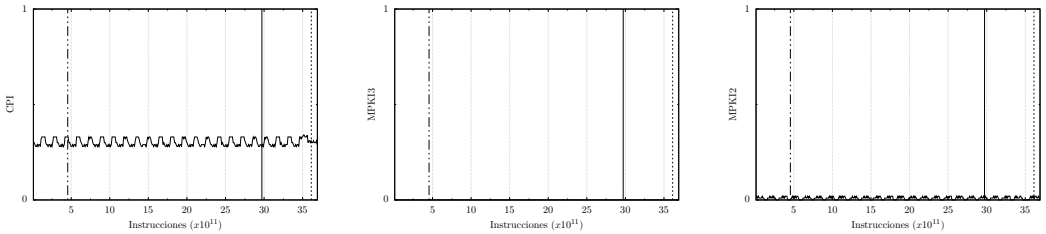
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.20: 416.gamess.1



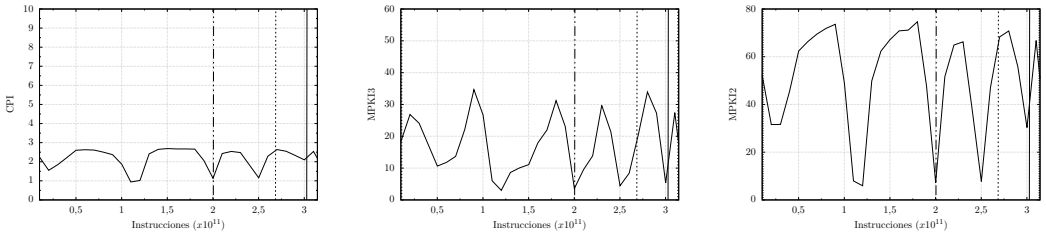
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.21: 416.gamess.2



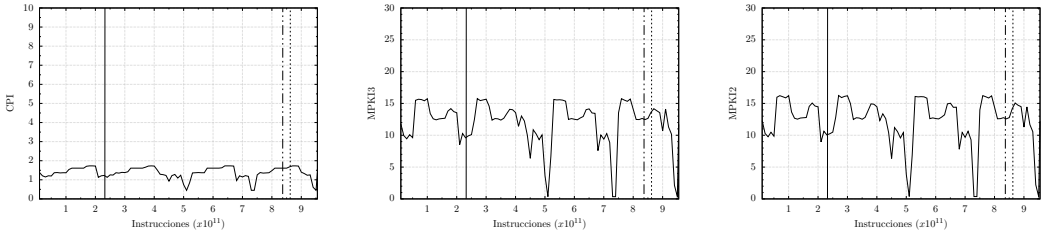
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.22: 416.gamess.3



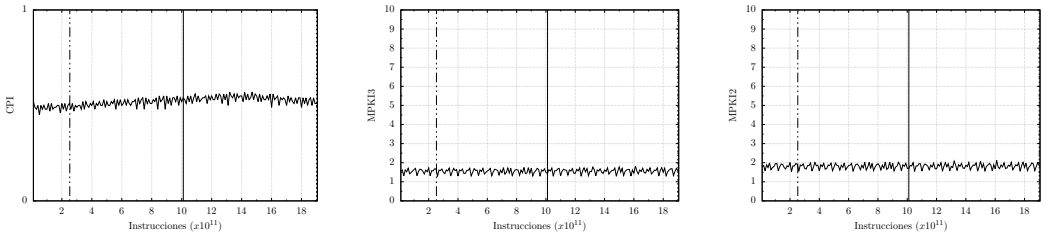
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.23: 429.mcf.1



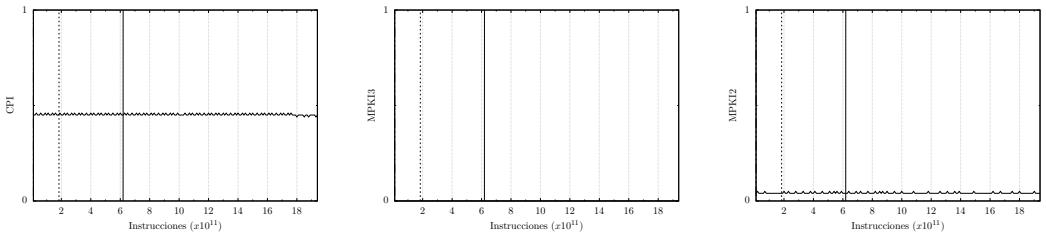
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.24: 433.milc.1



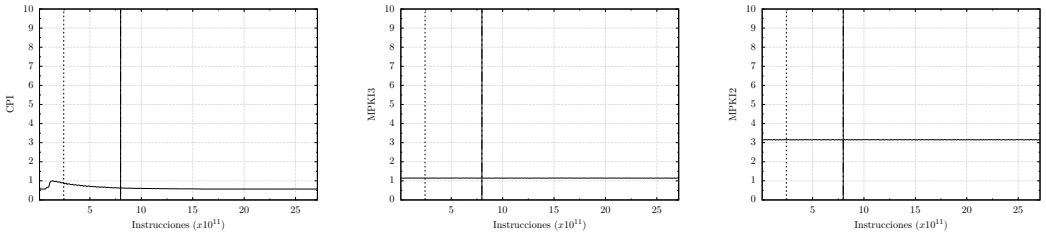
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.25: 434.zeusmp.1



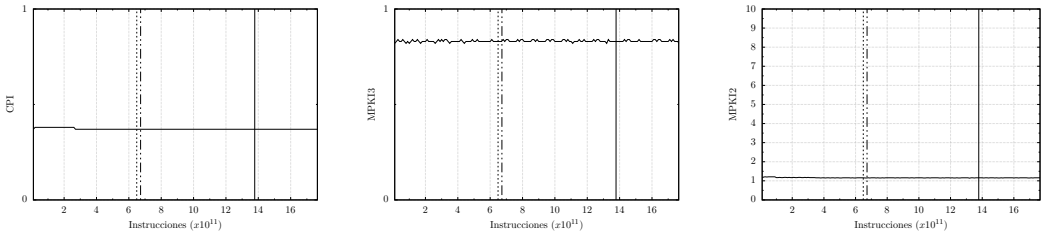
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.26: 435.gromacs.1



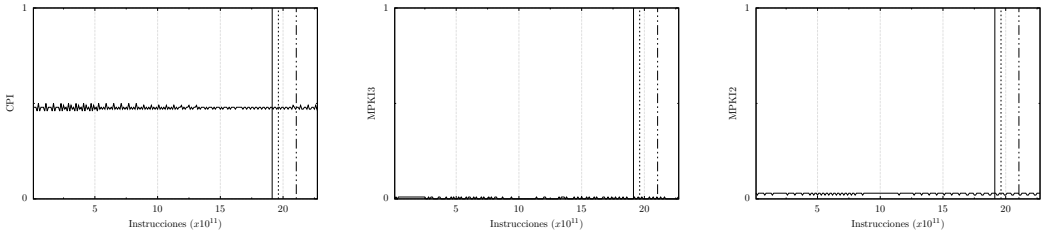
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.27: 436.cactusADM.1



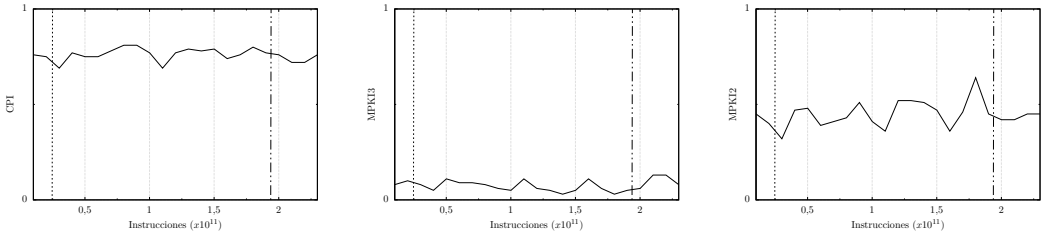
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.28: 437.leslie3d.1



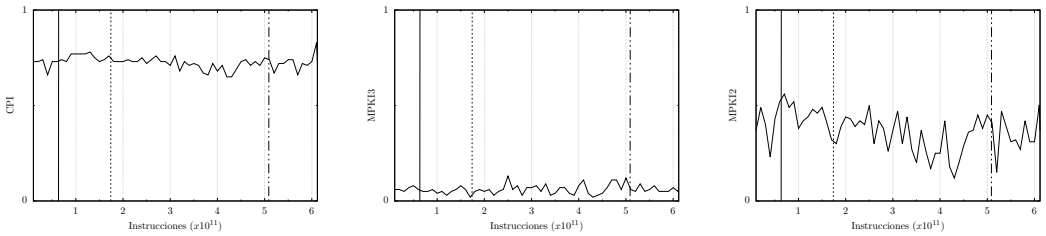
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.29: 444.namd.1



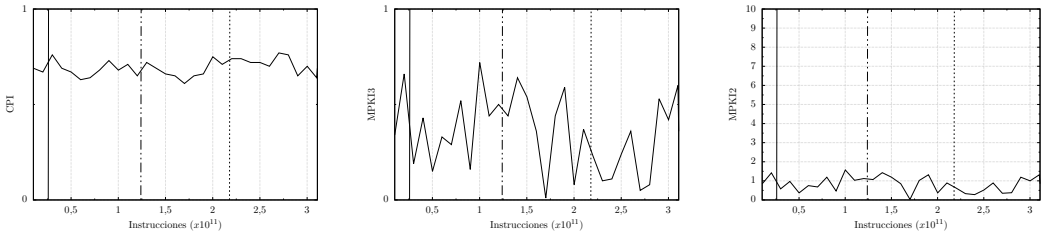
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.30: 445.gobmk.1



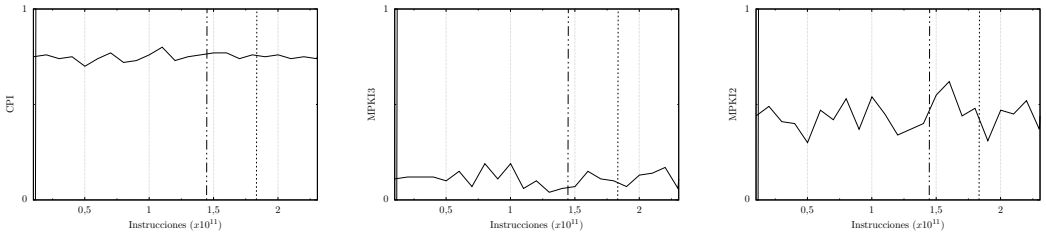
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.31: 445.gobmk.2



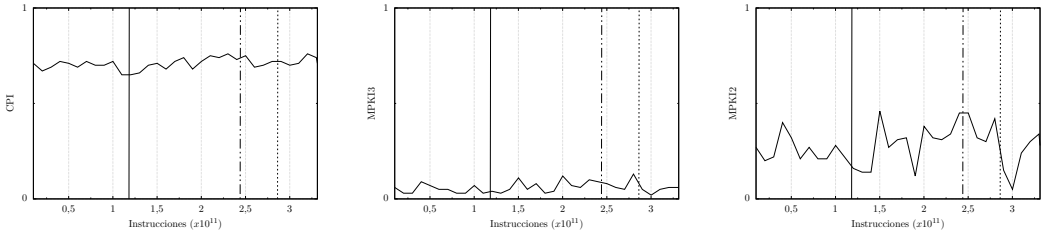
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.32: 445.gobmk.3



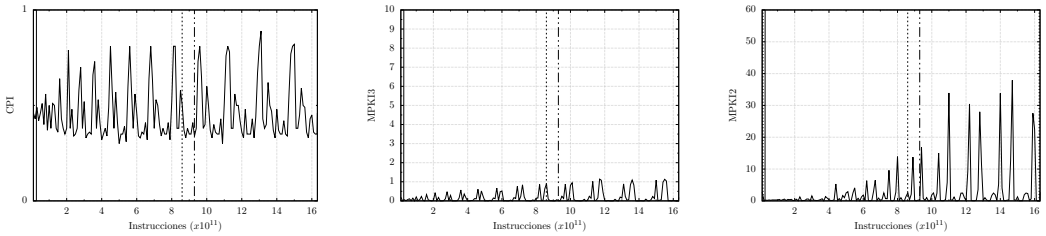
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.33: 445.gobmk.4



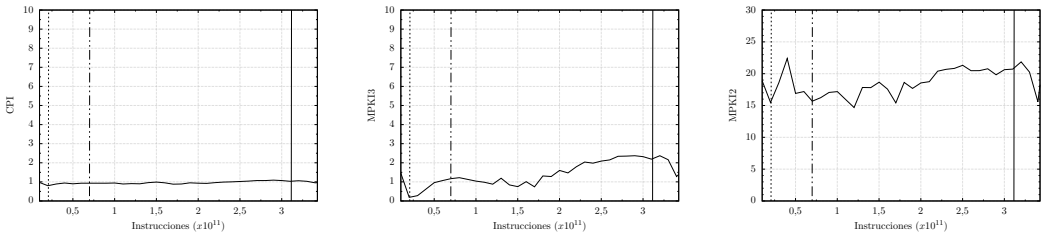
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.34: 445.gobmk.5



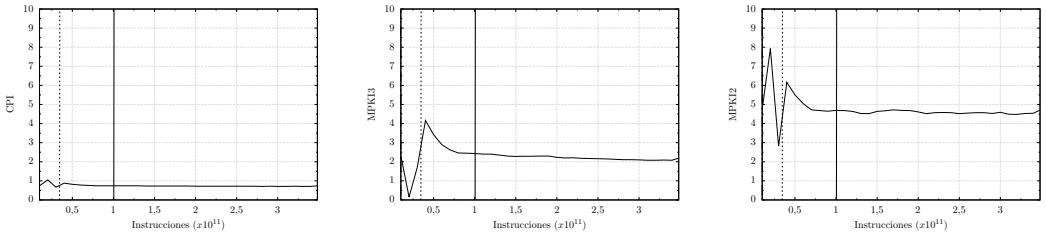
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.35: 447.dealII.1



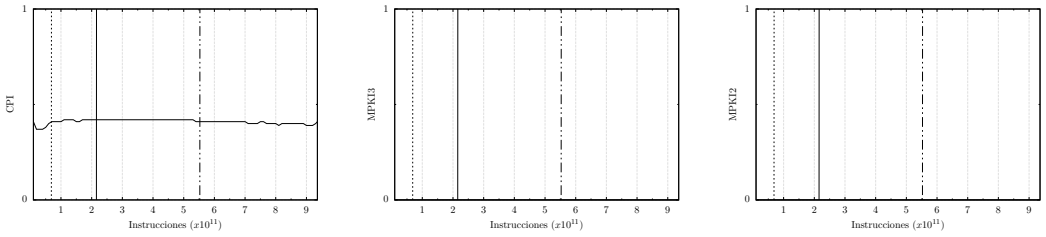
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.36: 450.soplex.1



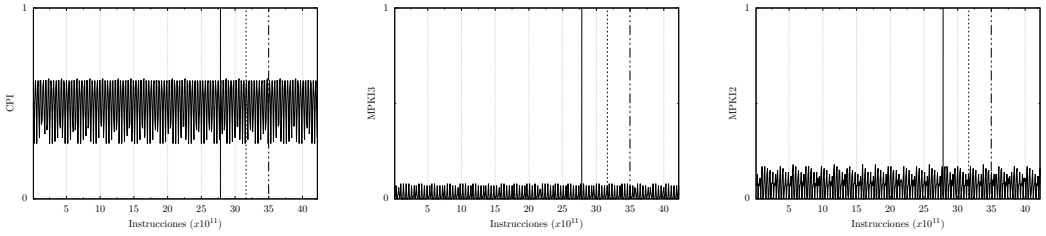
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.37: 450.soplex.2



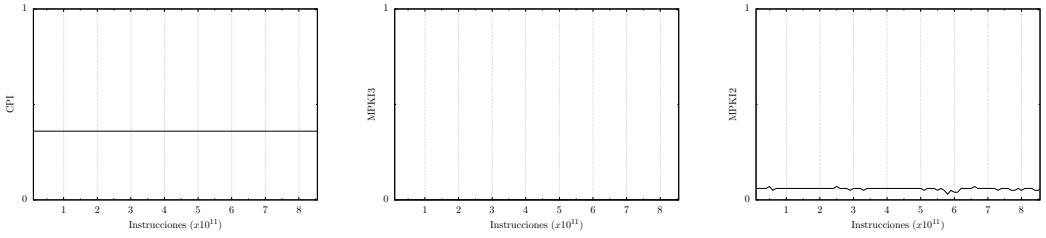
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.38: 453.povray.1



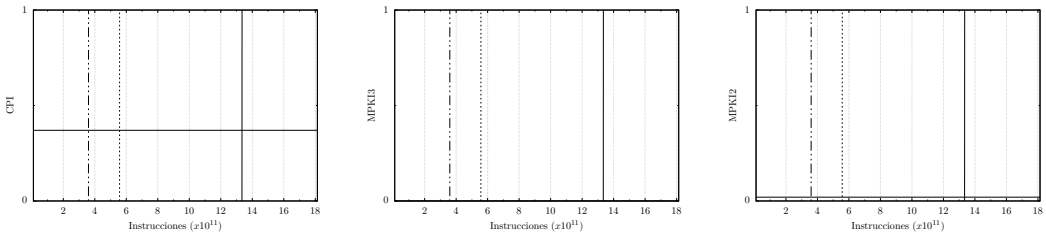
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.39: 454.calculix.1



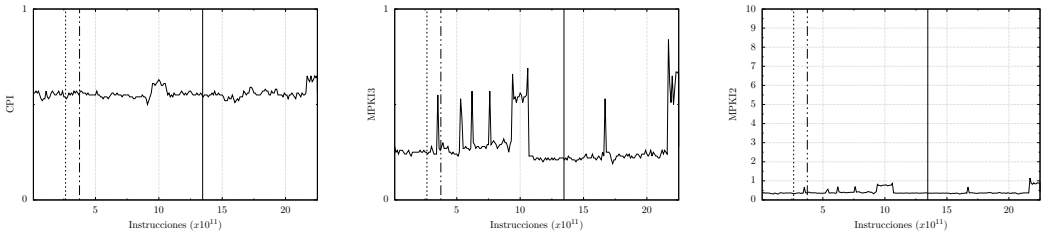
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.40: 456.hmmer.1



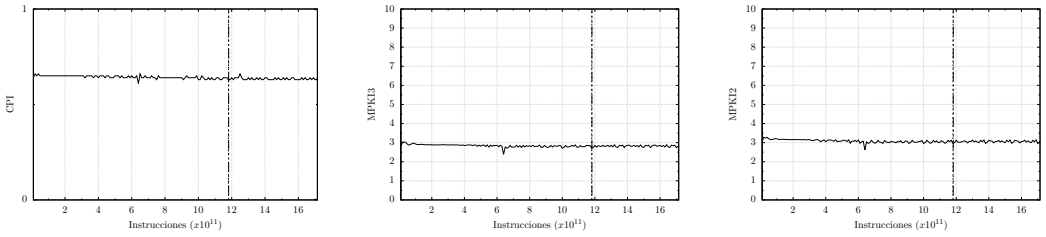
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.41: 456.hmmer.2



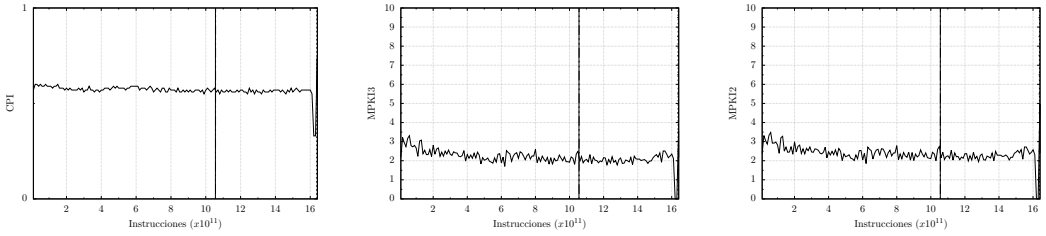
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.42: 458.sjeng.1



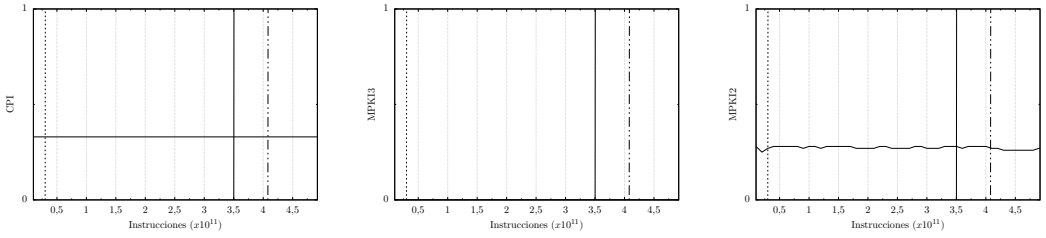
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.43: 459.GemsFDTD.1



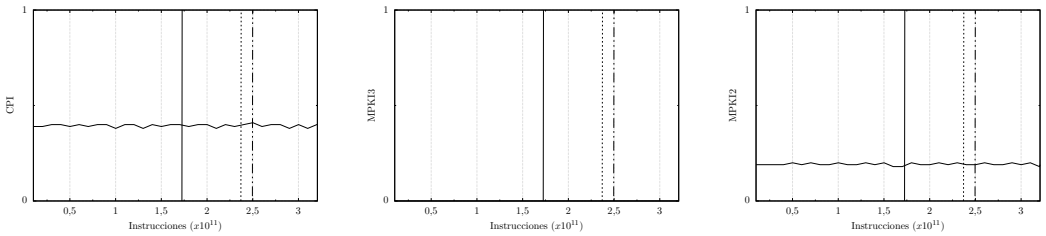
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.44: 462.libquantum.1



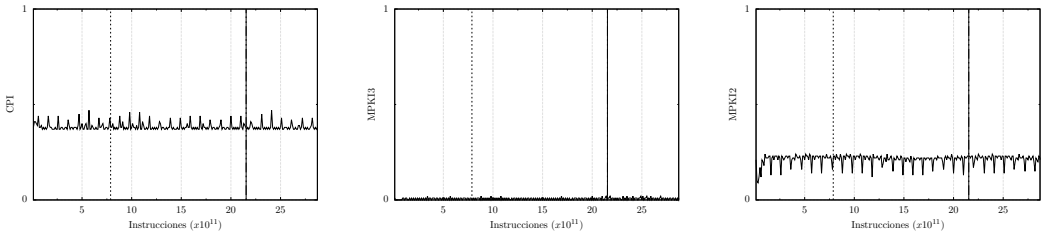
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.45: 464.h264ref.1



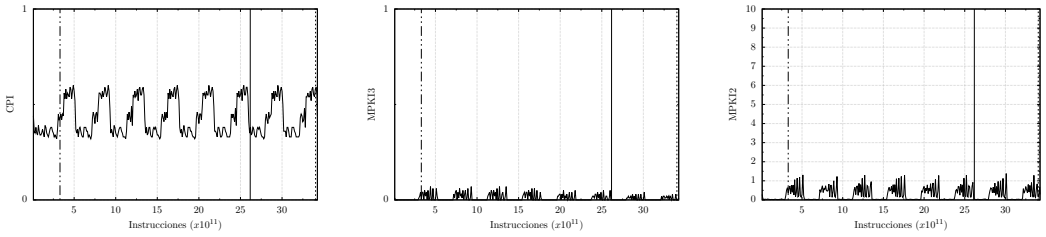
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.46: 464.h264ref.2



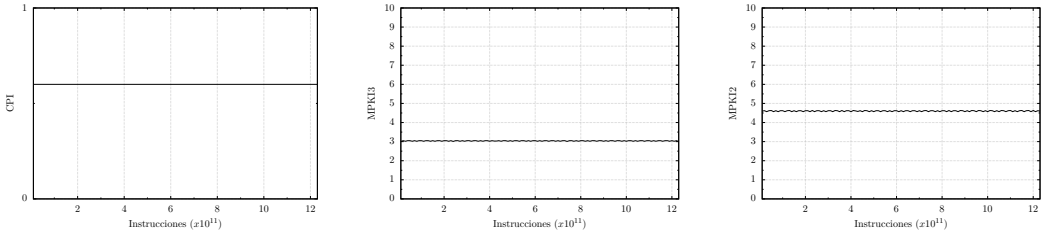
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.47: 464.h264ref.3



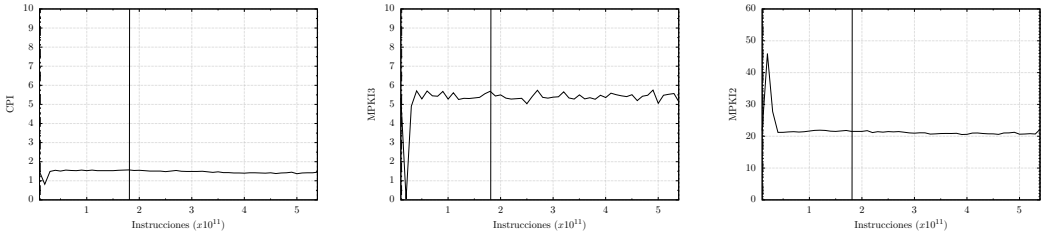
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.48: 465.tonto.1



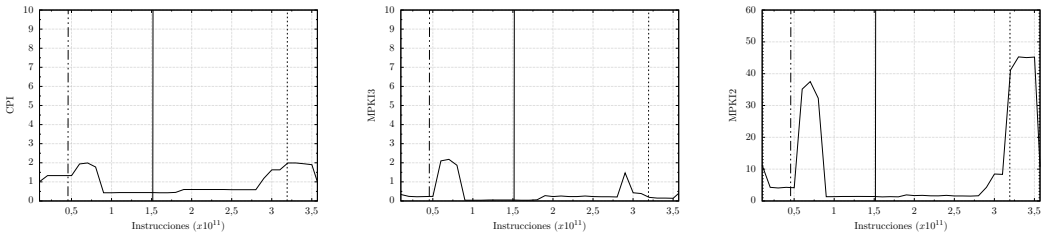
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.49: 470.lbm.1



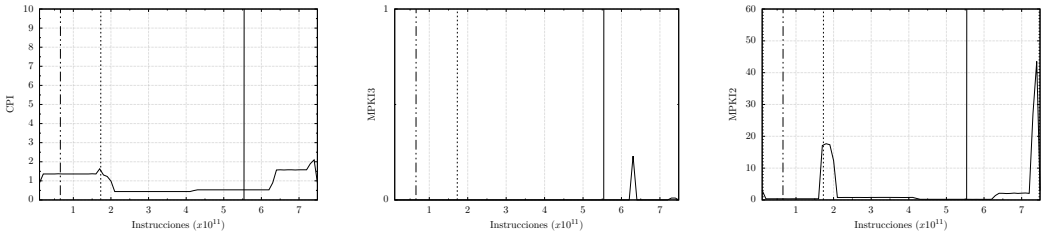
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.50: 471.omnetpp.1



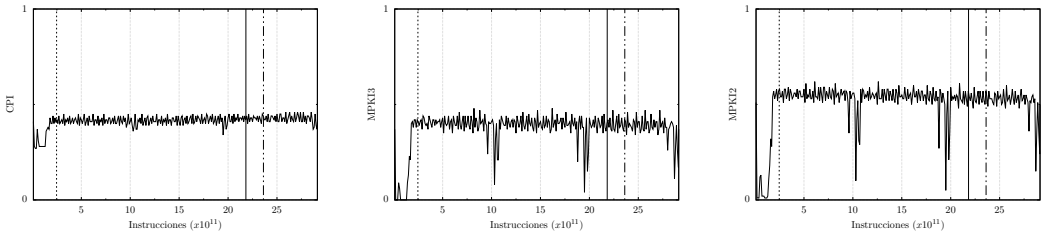
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.51: 473.astar.1



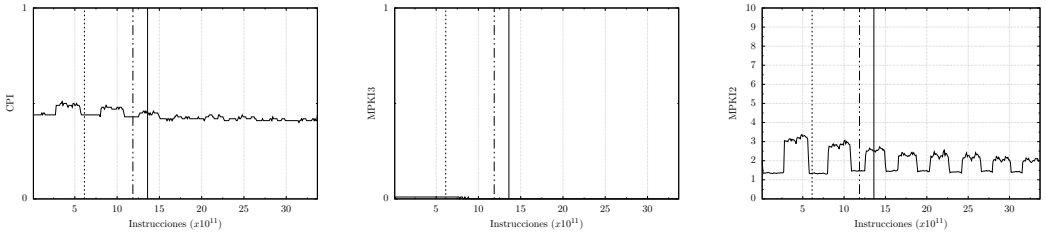
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.52: 473.astar.2



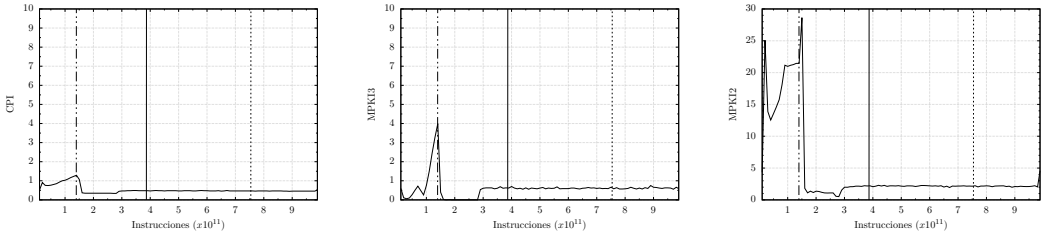
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.53: 481.wrf.1



(a) CPI **(b)** MPKI **(c)** APKI

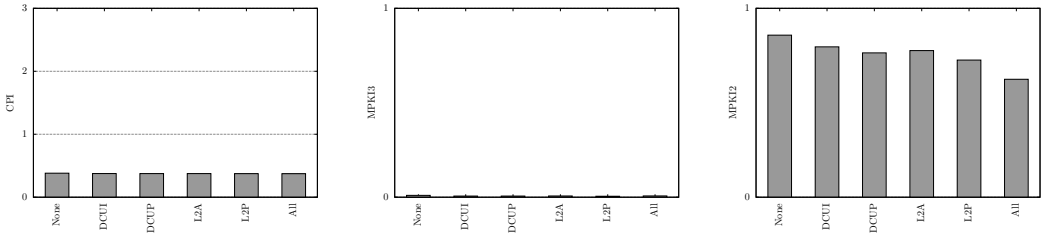
Figura C.54: 482.sphinx3.1



(a) CPI **(b)** MPKI **(c)** APKI

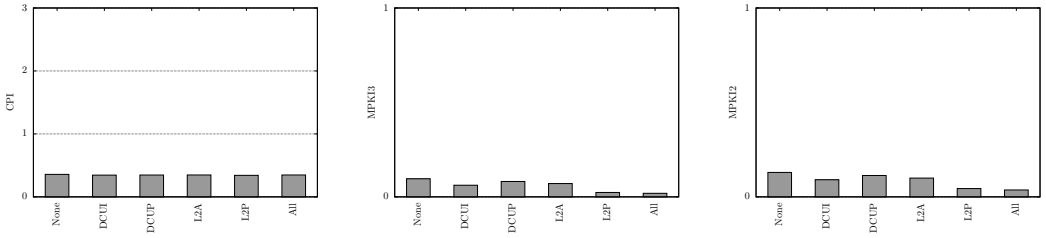
Figura C.55: 483.xalancbmk.1

C.1.3. Pre-búscadores Hardware



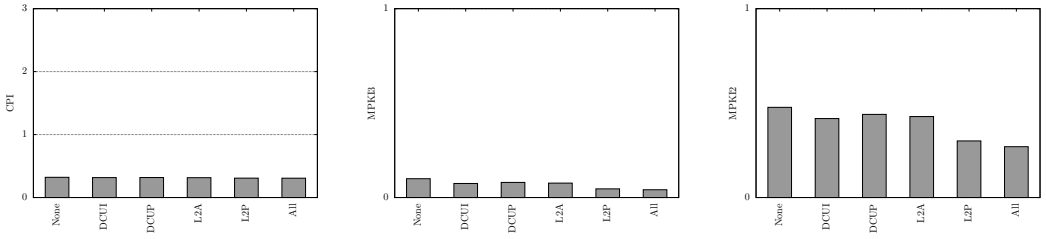
(a) CPI (b) MPKI (c) APKI

Figura C.56: 400.perlbench.1



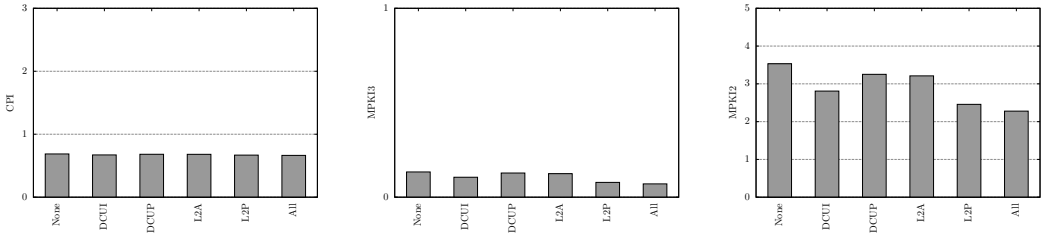
(a) CPI (b) MPKI (c) APKI

Figura C.57: 400.perlbench.2



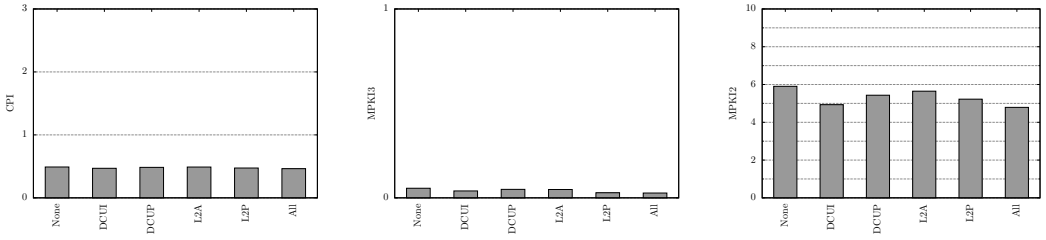
(a) CPI (b) MPKI (c) APKI

Figura C.58: 400.perlbench.3



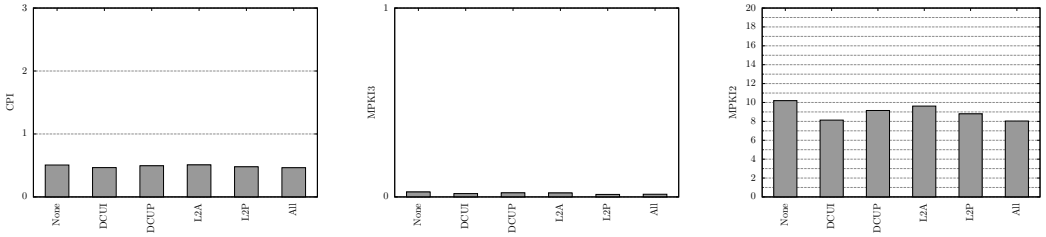
(a) CPI (b) MPKI (c) APKI

Figura C.59: 401.bzip2.1



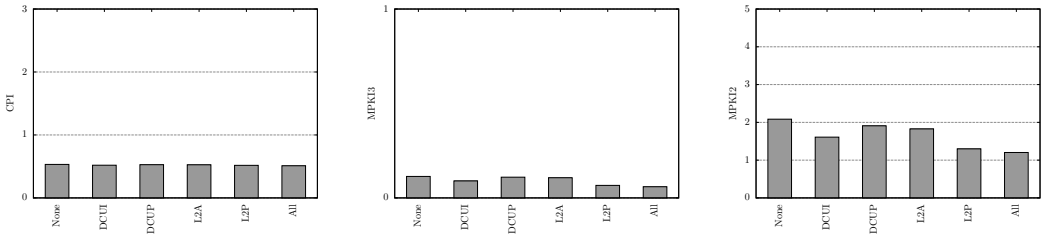
(a) CPI (b) MPKI (c) APKI

Figura C.60: 401.bzip2.2



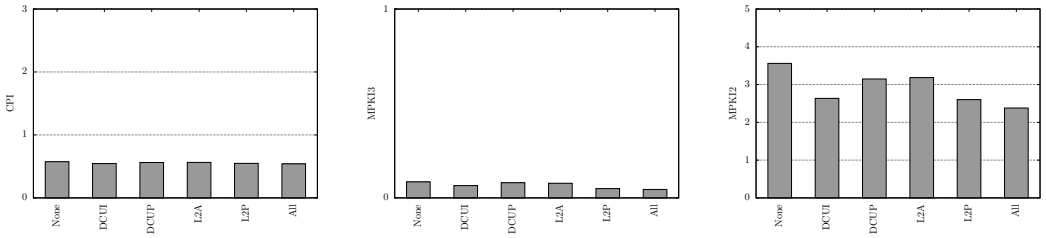
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.61: 401.bzip2.3



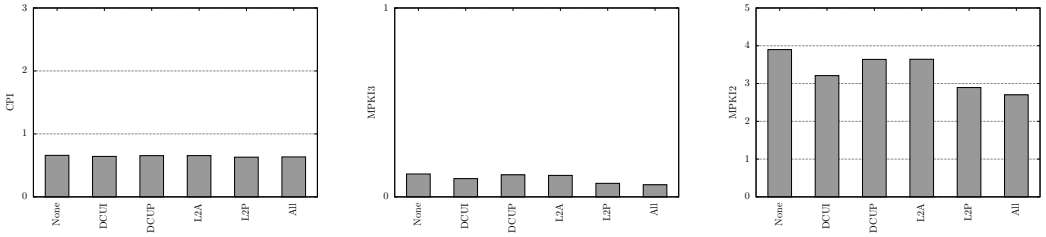
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.62: 401.bzip2.4



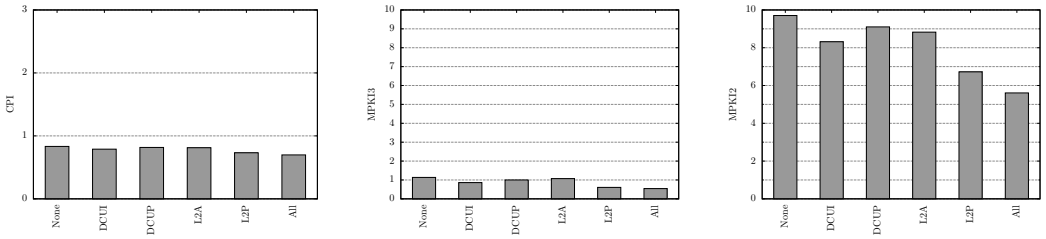
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.63: 401.bzip2.5



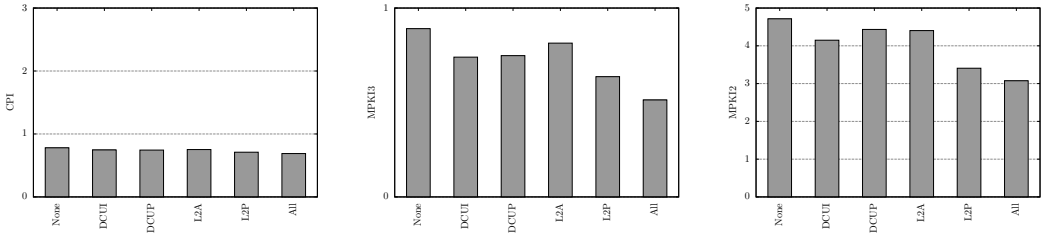
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.64: 401.bzip2.6



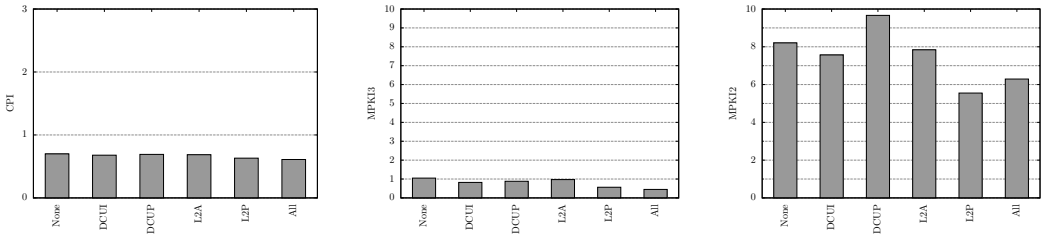
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.65: 403.gcc.1



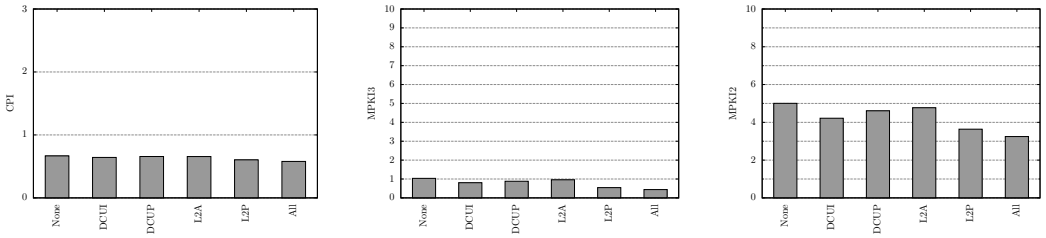
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.66: 403.gcc.2



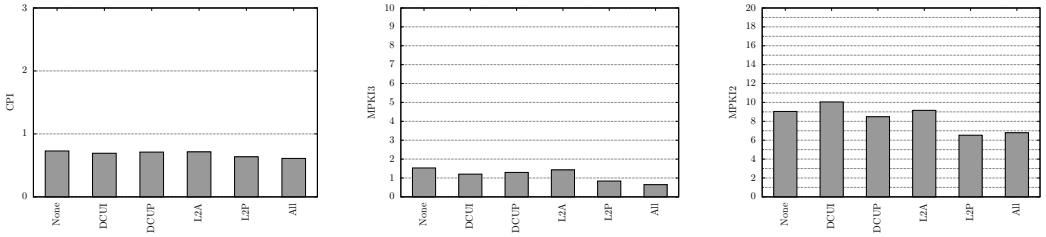
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.67: 403.gcc.3



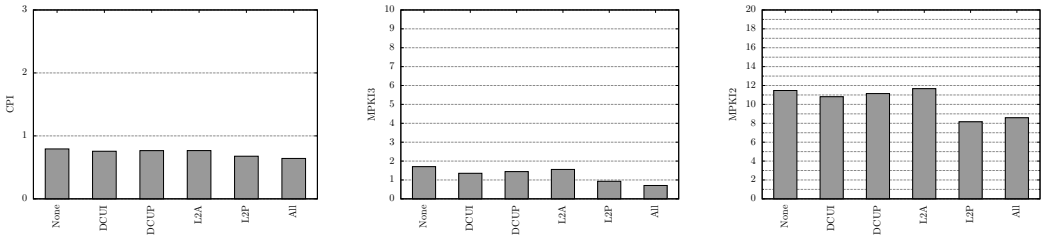
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.68: 403.gcc.4



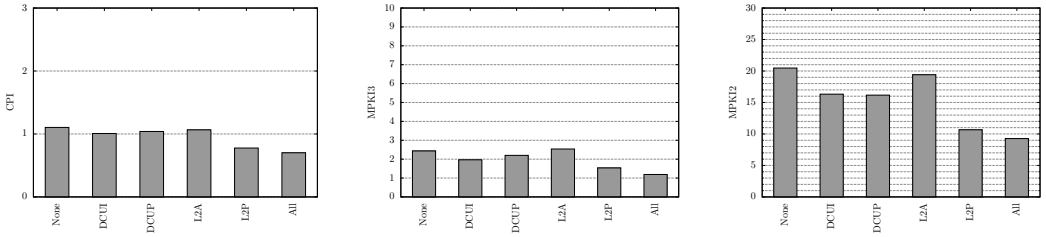
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.69: 403.gcc.5



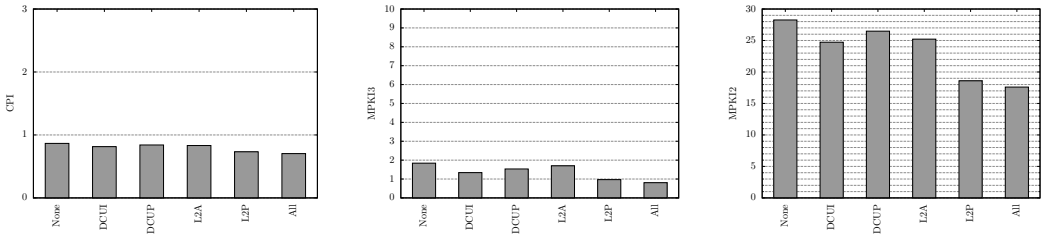
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.70: 403.gcc.6



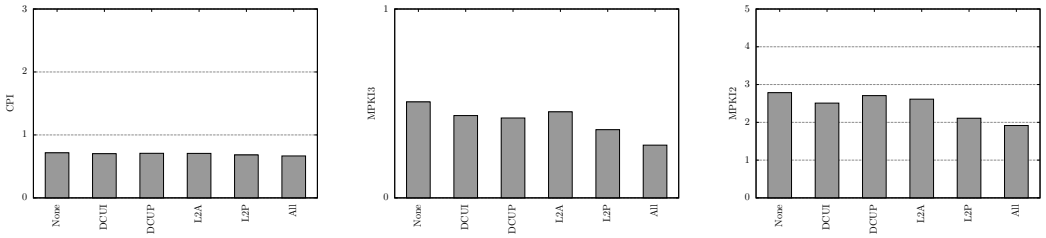
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.71: 403.gcc.7



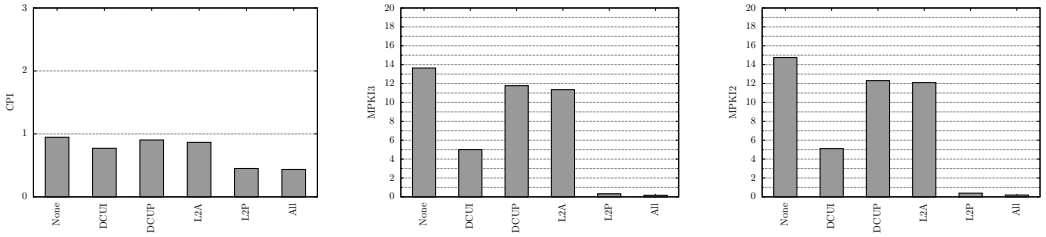
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.72: 403.gcc.8



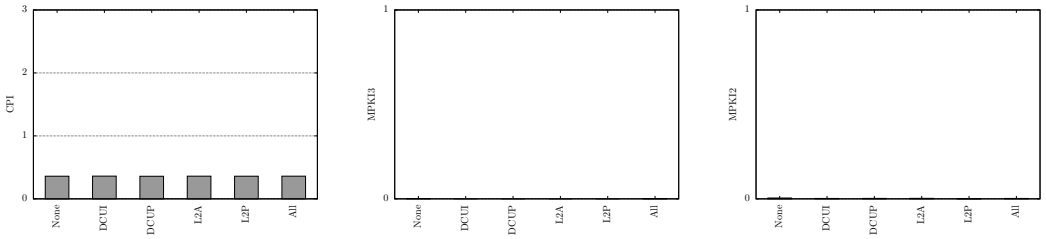
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.73: 403.gcc.9



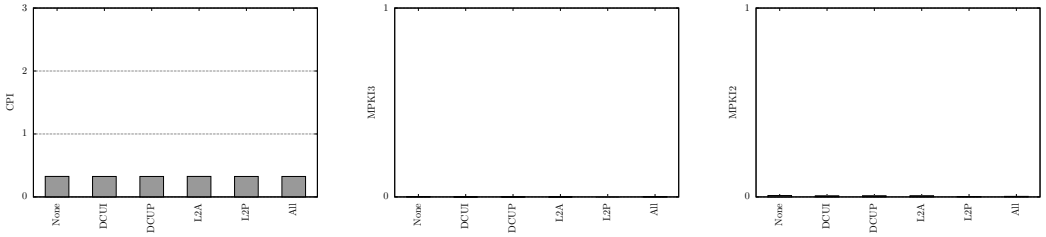
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.74: 410.bwaves.1



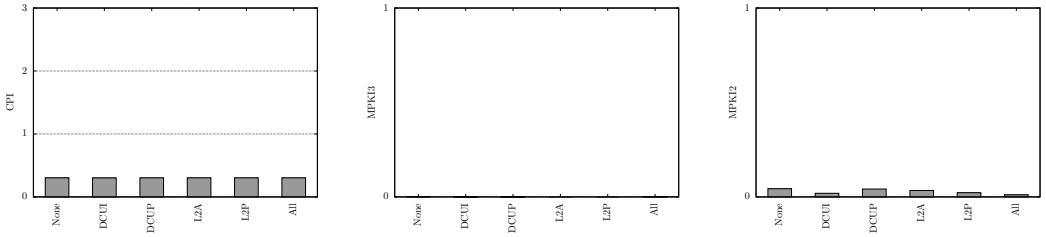
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.75: 416.gamess.1



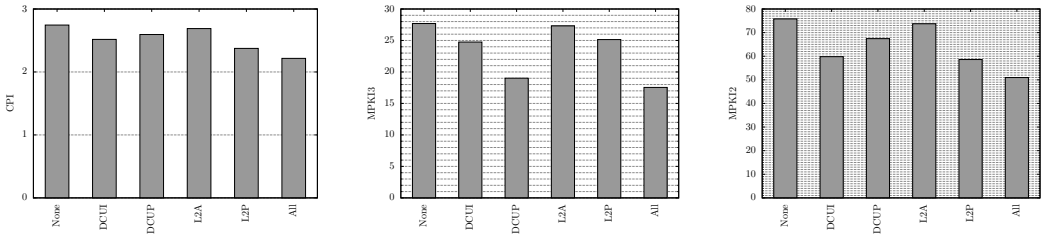
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.76: 416.gamess.2



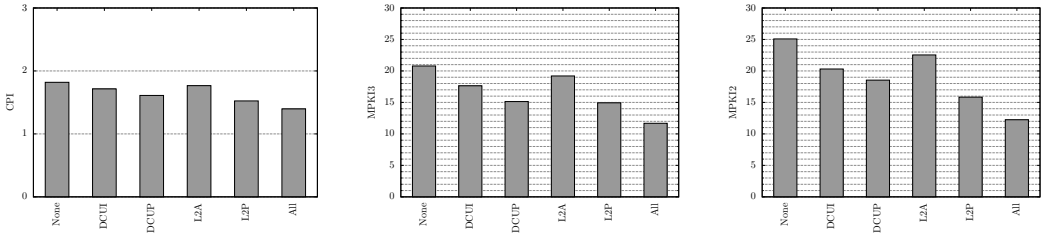
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.77: 416.gamess.3



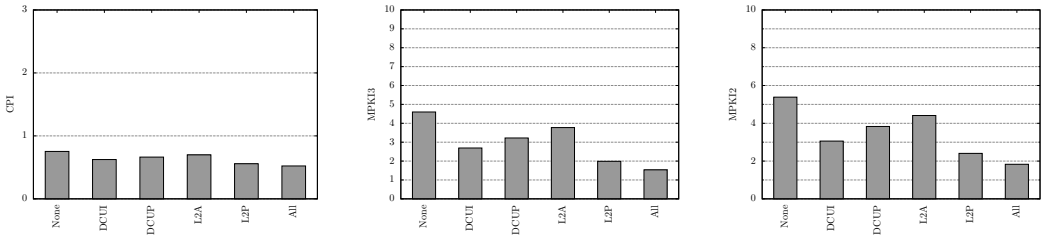
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.78: 429.mcf.1



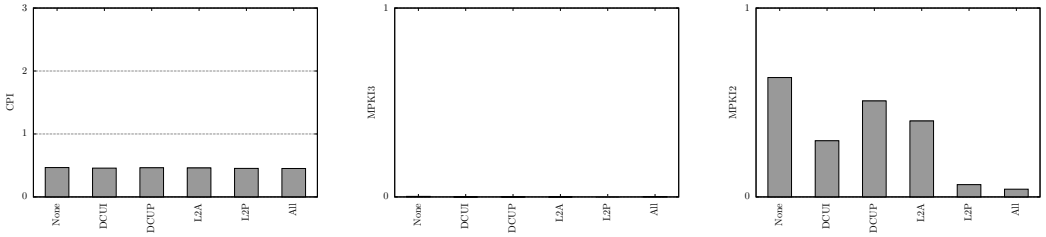
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.79: 433.milc.1



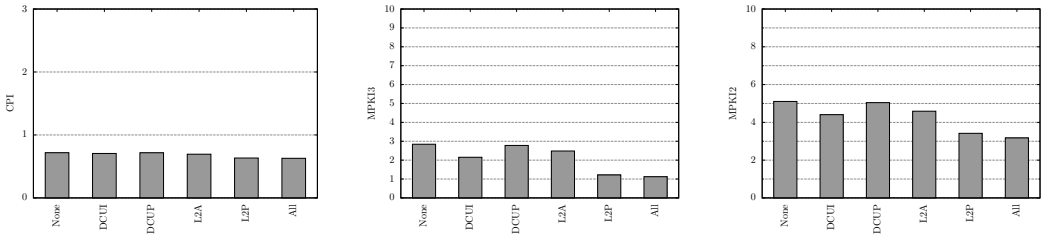
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.80: 434.zeusmp.1



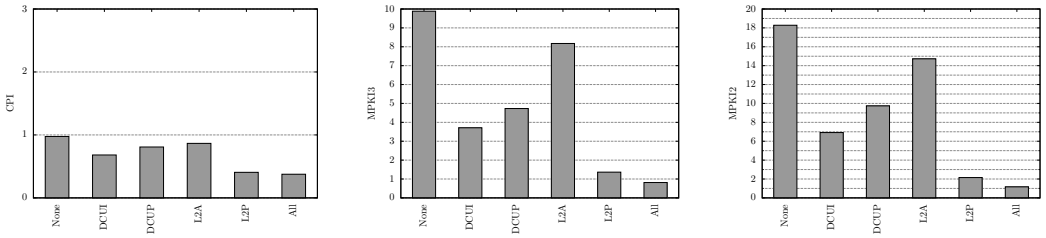
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.81: 435.gromacs.1



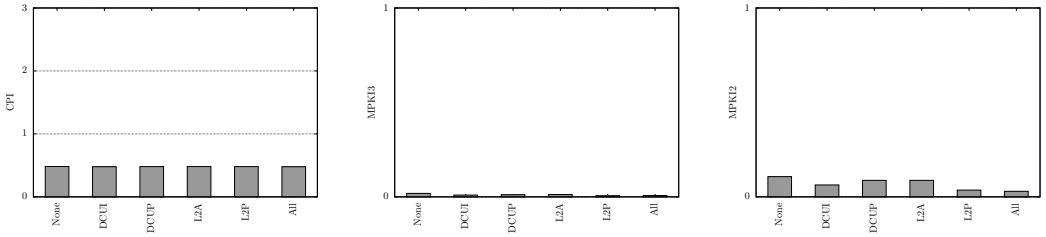
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.82: 436.cactusADM.1



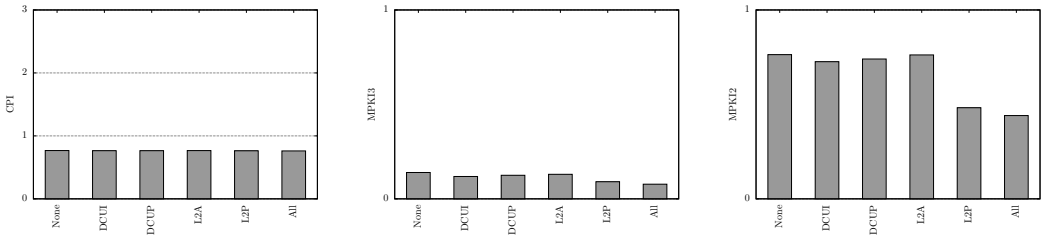
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.83: 437.leslie3d.1



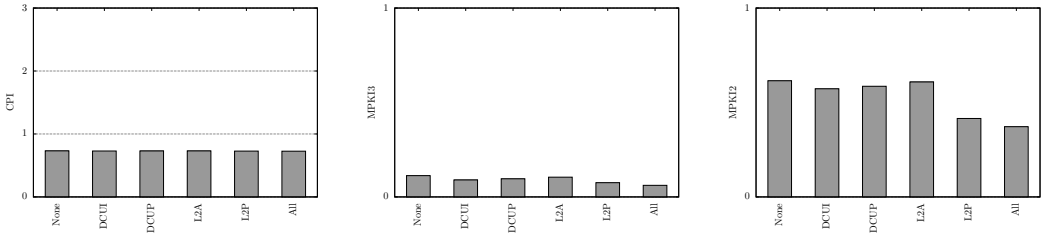
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.84: 444.namd.1



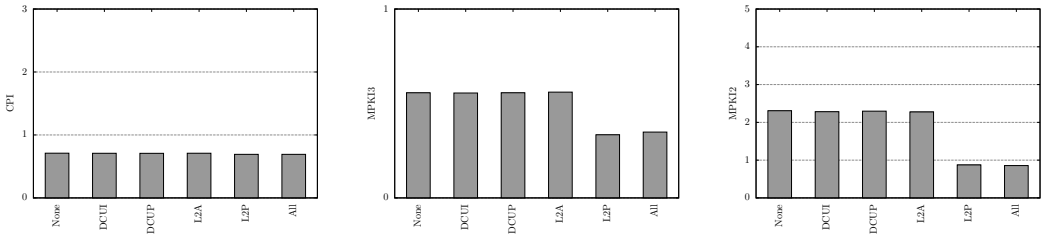
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.85: 445.gobmk.1



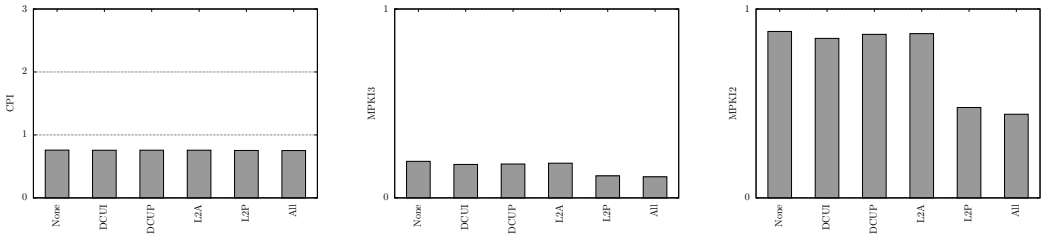
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.86: 445.gobmk.2



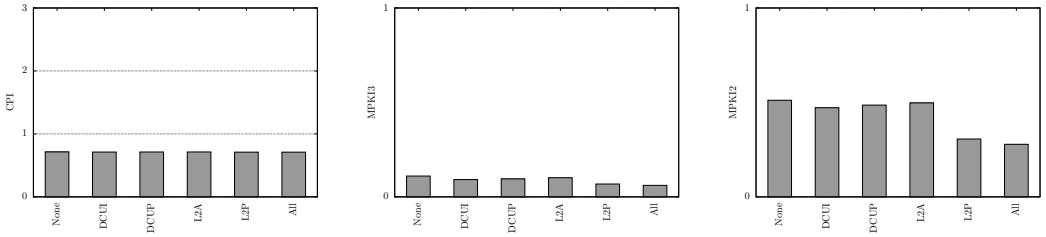
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.87: 445.gobmk.3



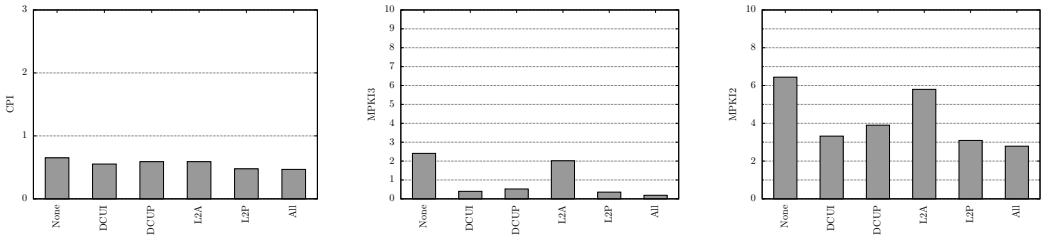
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.88: 445.gobmk.4



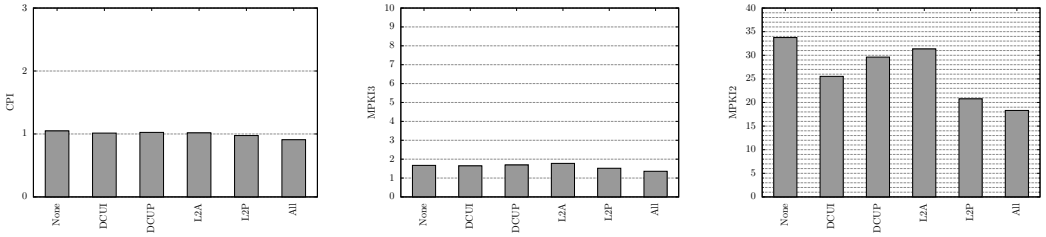
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.89: 445.gobmk.5



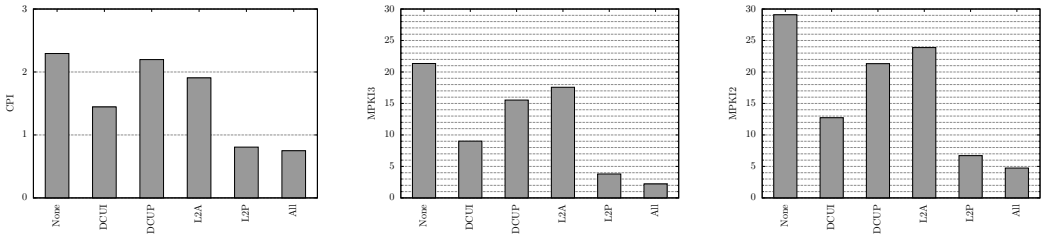
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.90: 447.dealII.1



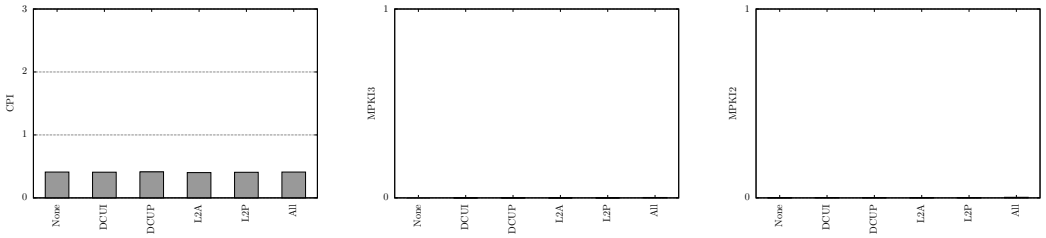
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.91: 450.soplex.1



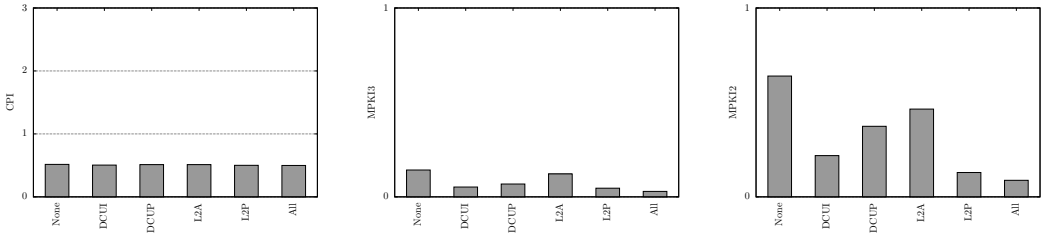
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.92: 450.soplex.2



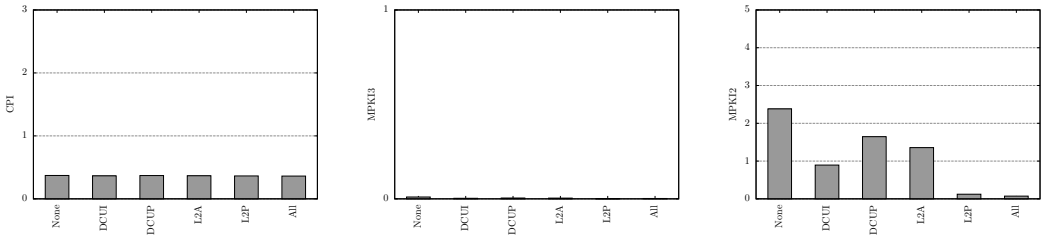
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.93: 453.povray.1



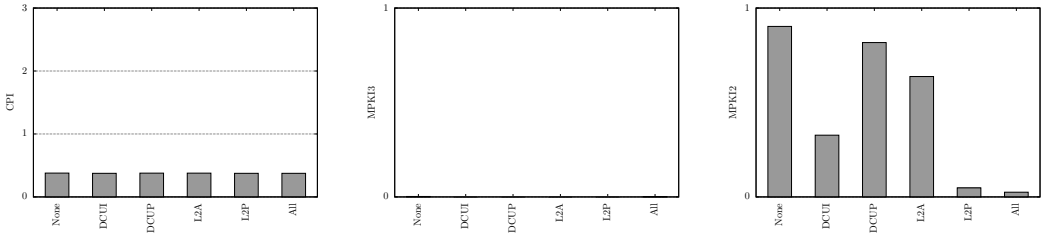
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.94: 454.calculix.1



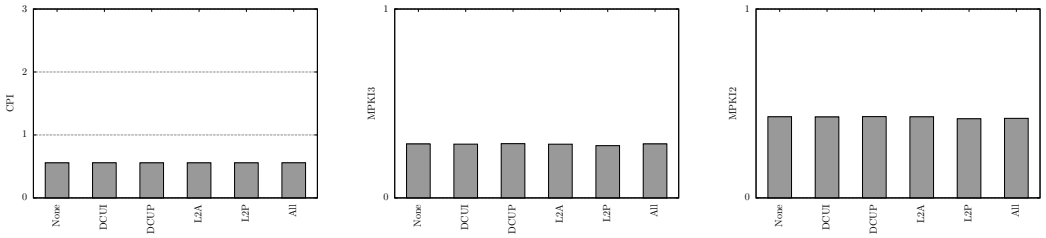
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.95: 456.hmmer.1



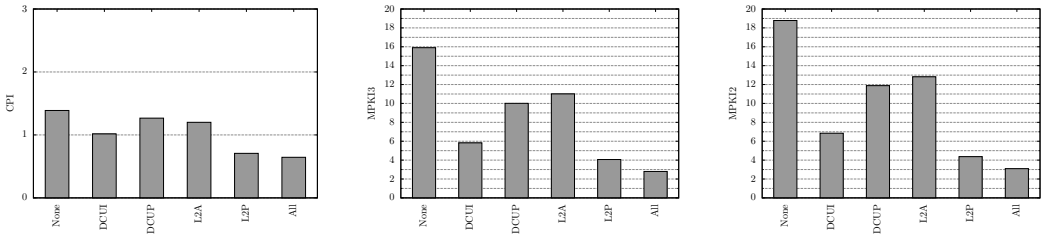
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.96: 456.hmmer.2



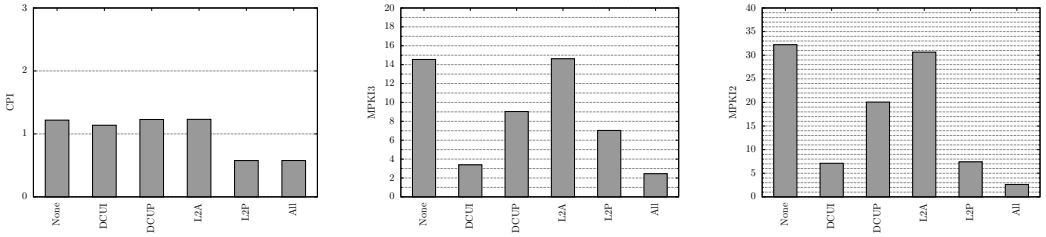
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.97: 458.sjeng.1



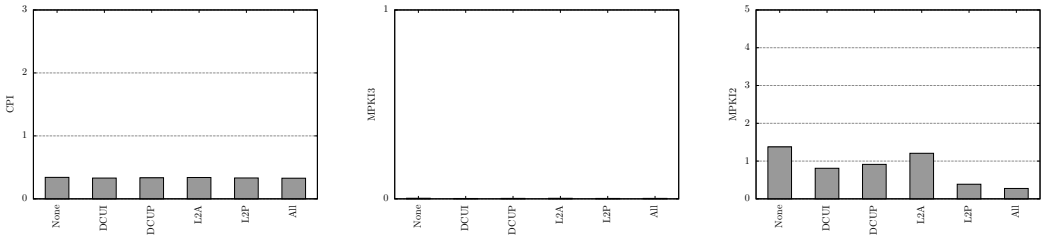
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.98: 459.GemsFDTD.1



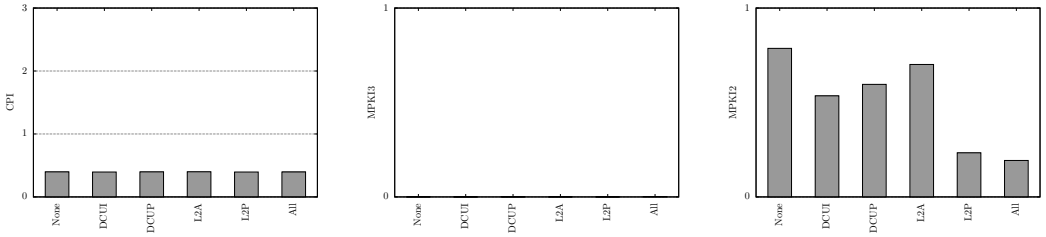
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.99: 462.libquantum.1



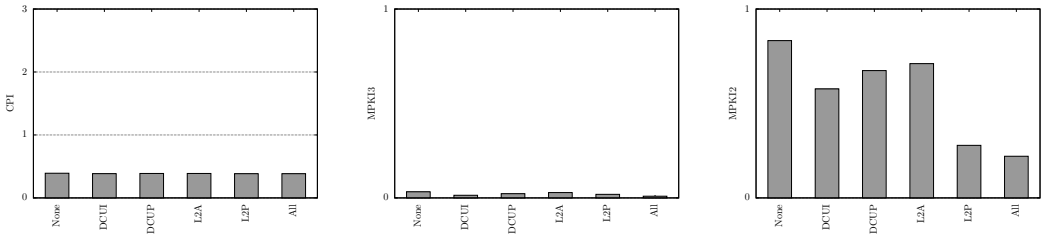
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.100: 464.h264ref.1



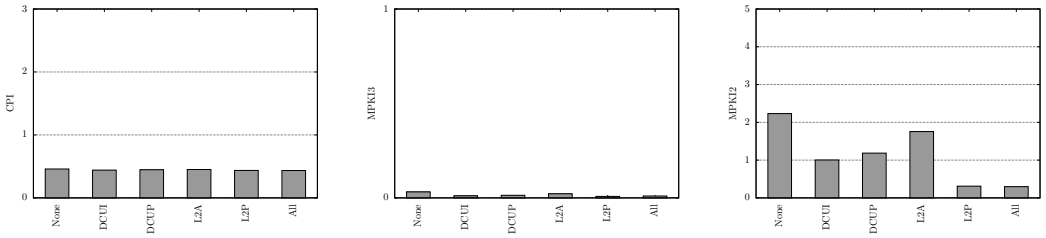
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.101: 464.h264ref.2



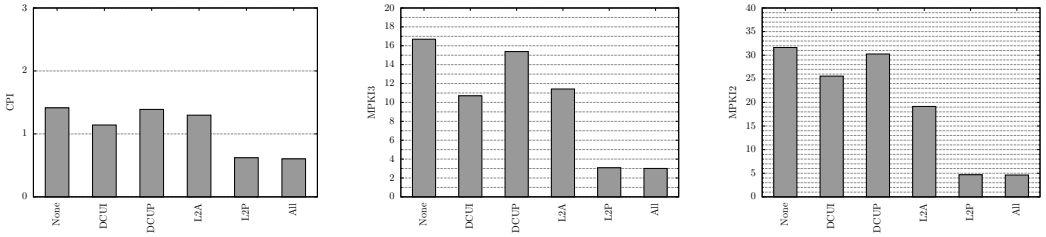
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.102: 464.h264ref.3



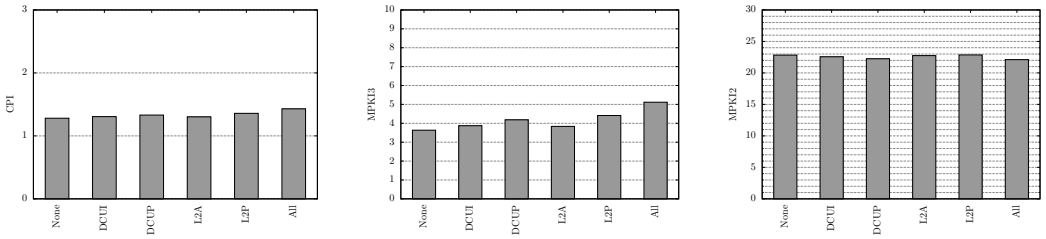
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.103: 465.tonto.1



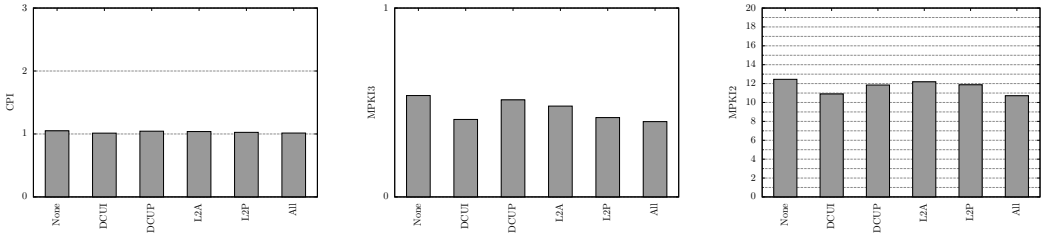
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.104: 470.lbm.1



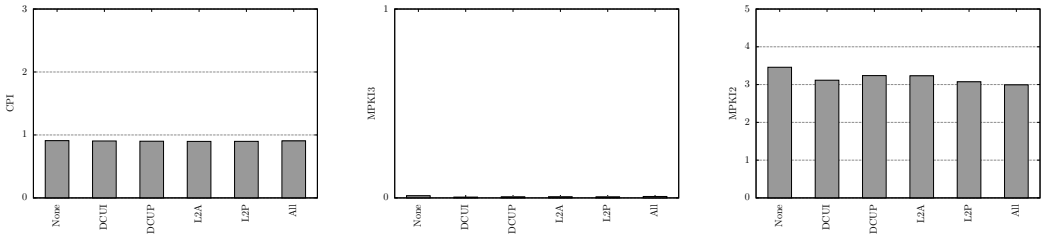
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.105: 471.omnetpp.1



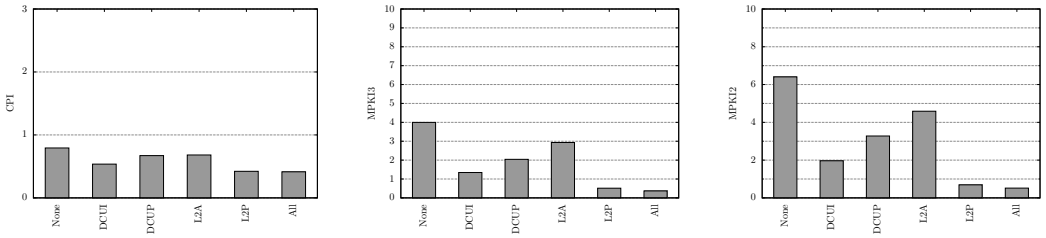
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.106: 473.astar.1



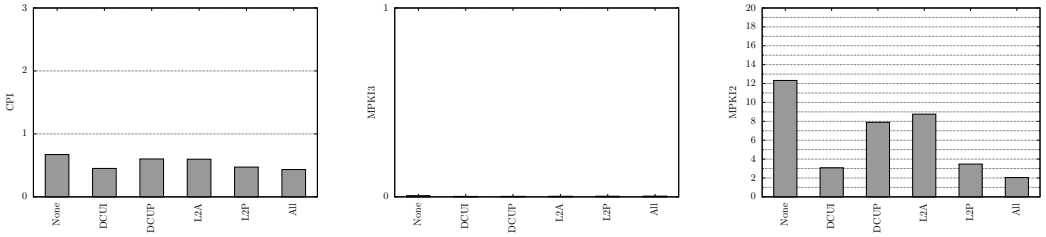
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.107: 473.astar.2



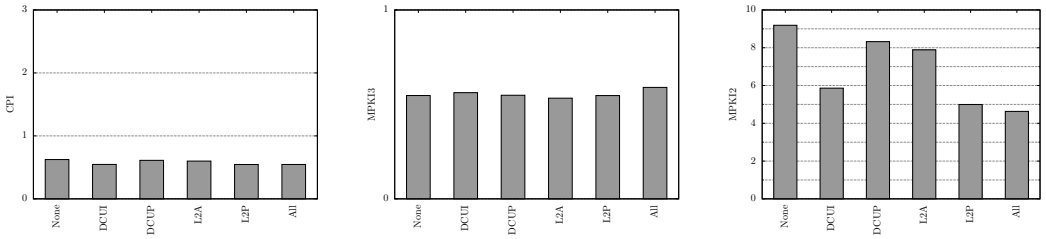
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.108: 481.wrf.1



(a) CPI **(b)** MPKI **(c)** APKI

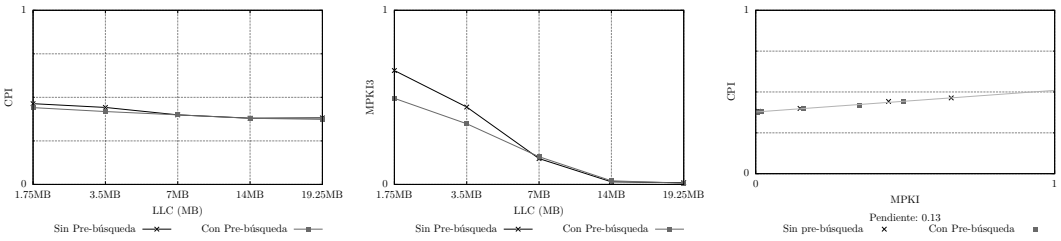
Figura C.109: 482.sphinx3.1



(a) CPI **(b)** MPKI **(c)** APKI

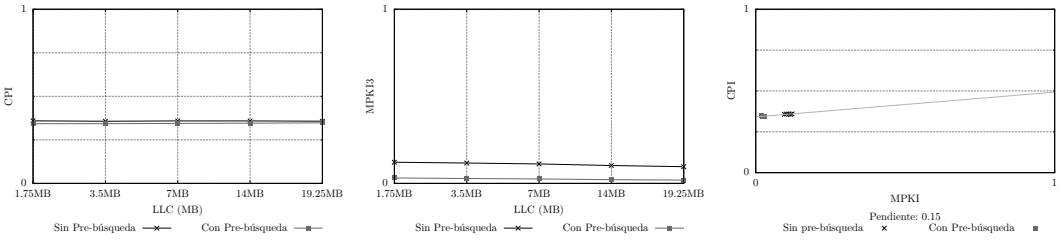
Figura C.110: 483.xalancbmk.1

C.1.4. Tamaño de LLC



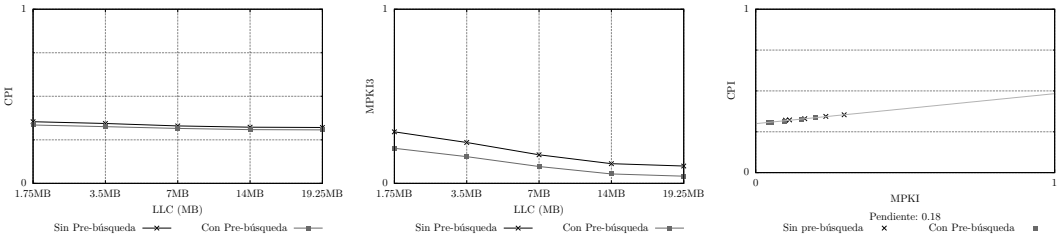
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.111: 400.perlbench.1



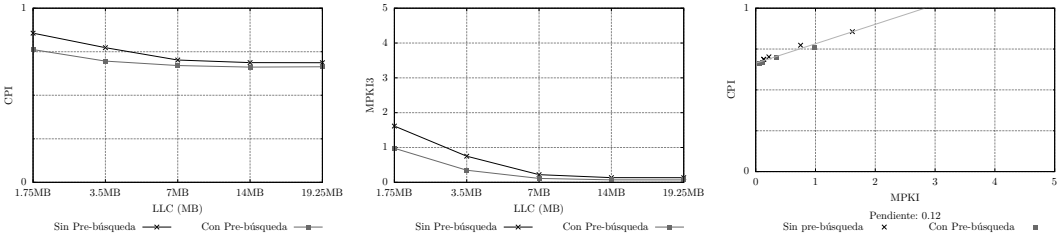
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.112: 400.perlbench.2



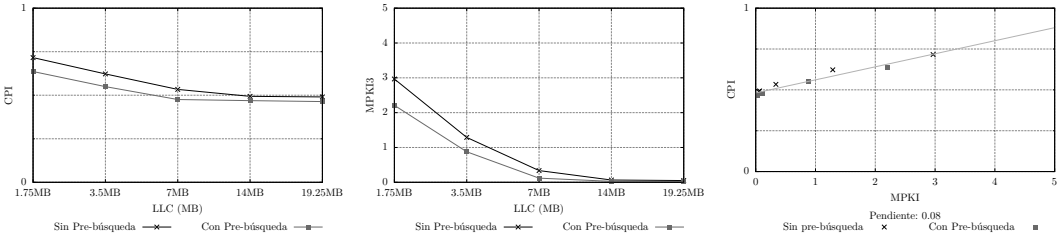
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.113: 400.perlbench.3



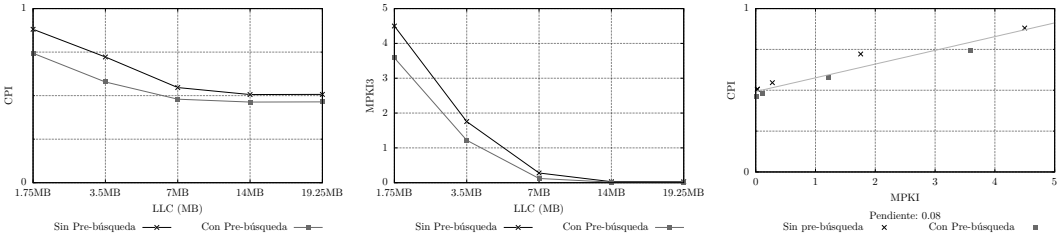
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.114: 401.bzip2.1



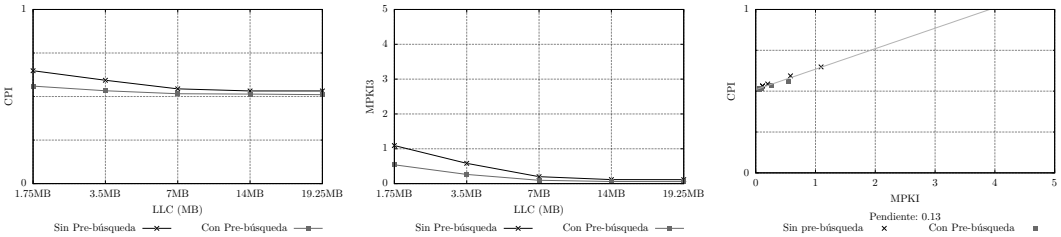
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.115: 401.bzip2.2



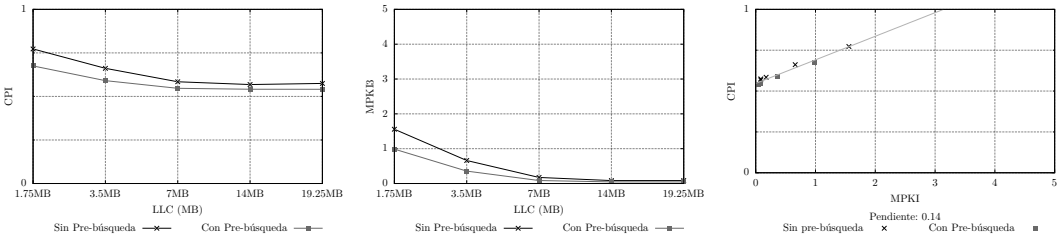
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.116: 401.bzip2.3



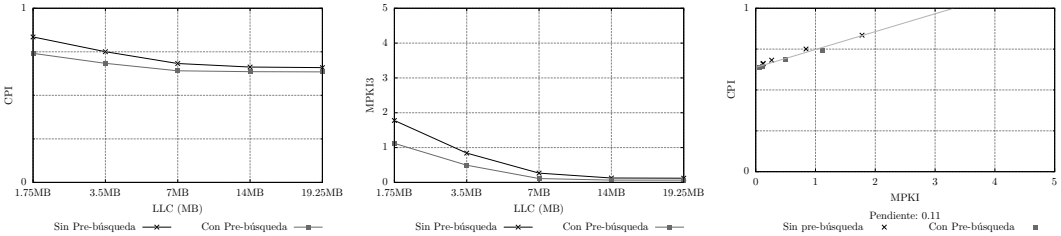
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.117: 401.bzip2.4



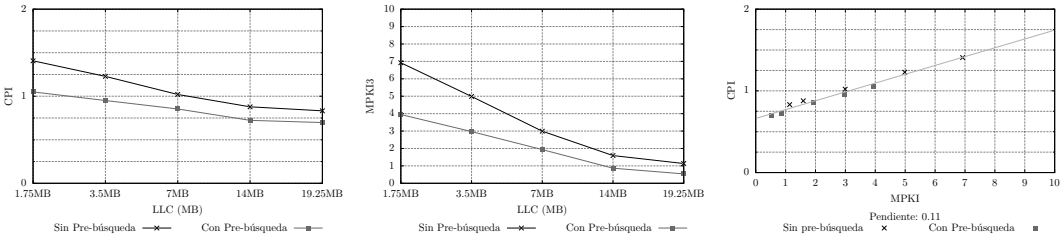
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.118: 401.bzip2.5



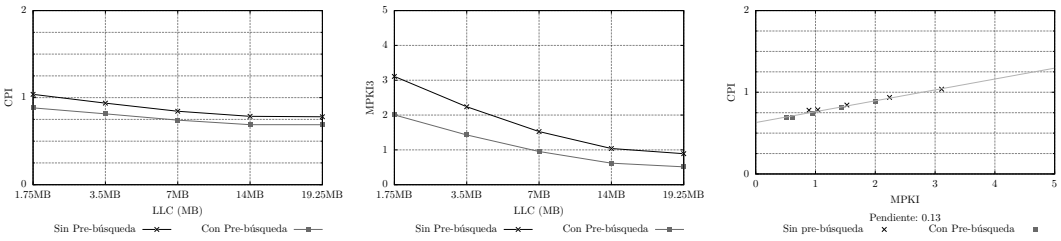
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.119: 401.bzip2.6



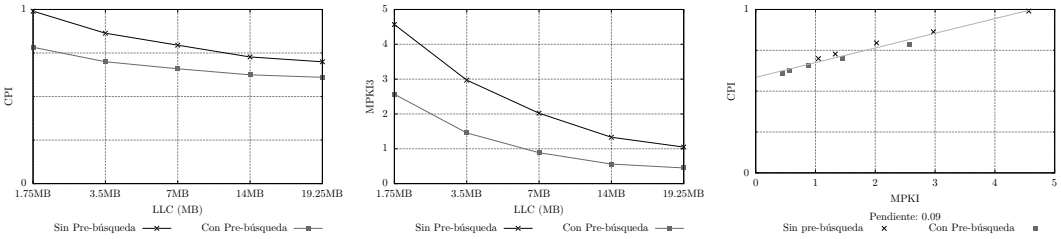
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.120: 403.gcc.1



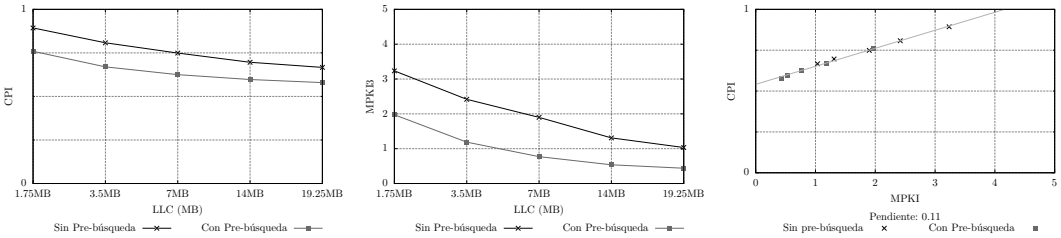
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.121: 403.gcc.2



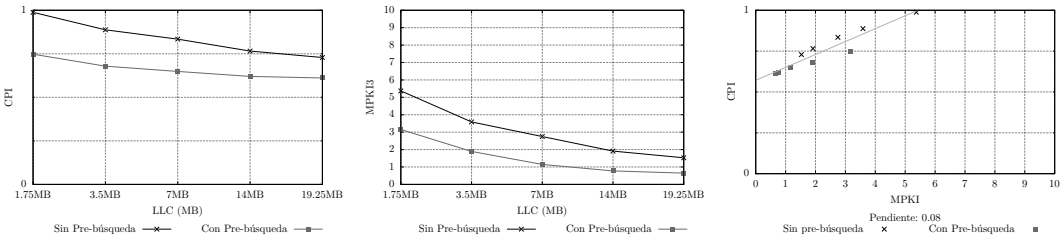
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.122: 403.gcc.3



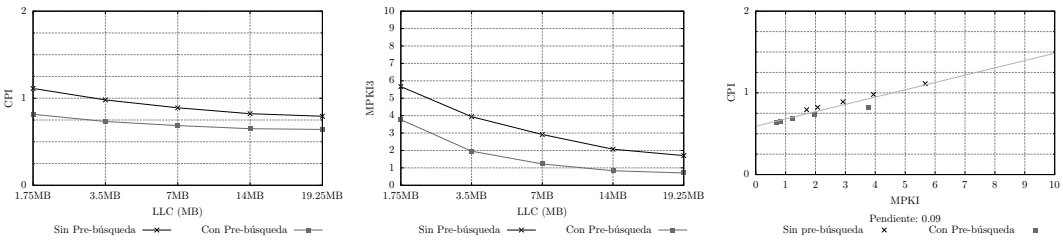
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.123: 403.gcc.4



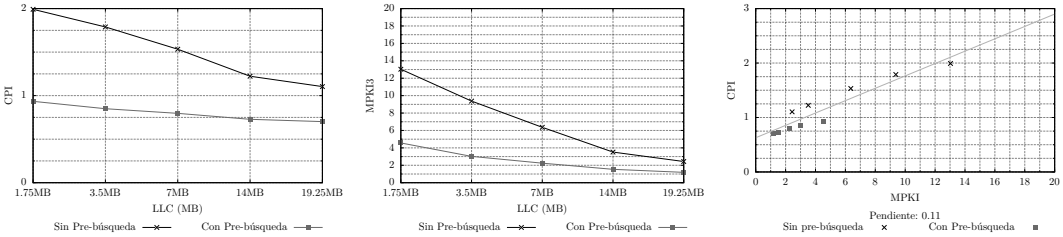
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.124: 403.gcc.5



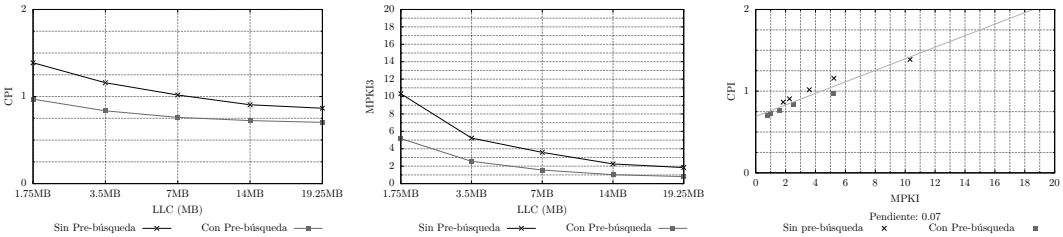
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.125: 403.gcc.6



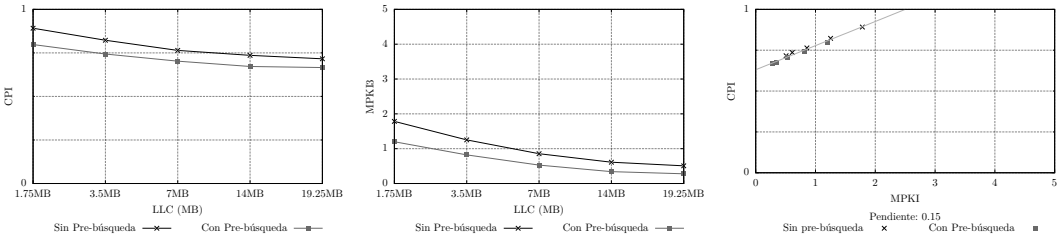
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.126: 403.gcc.7



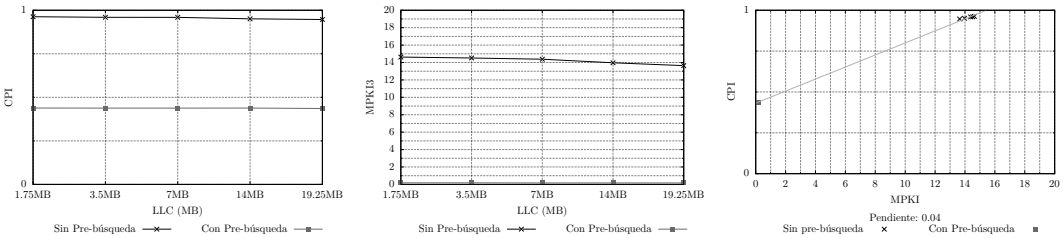
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.127: 403.gcc.8



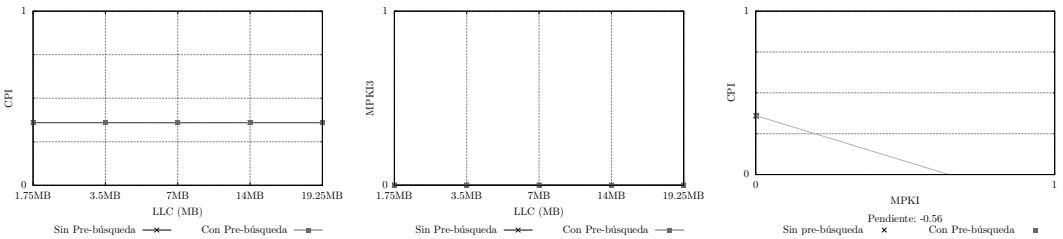
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.128: 403.gcc.9



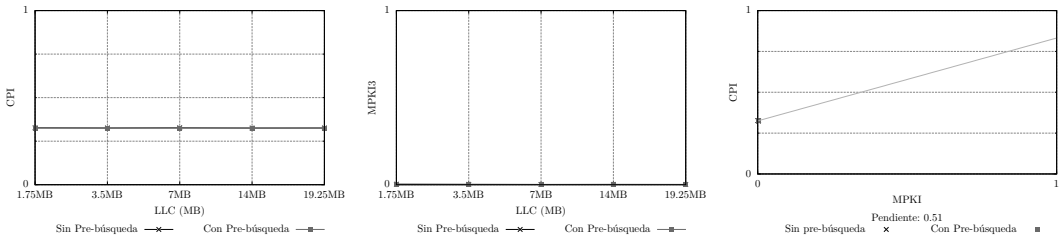
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.129: 410.bwaves.1



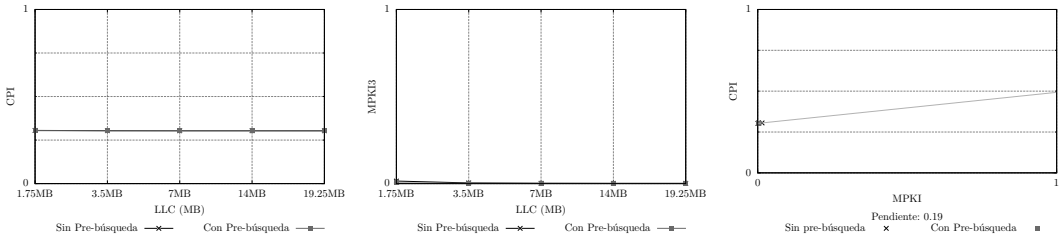
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.130: 416.gamess.1



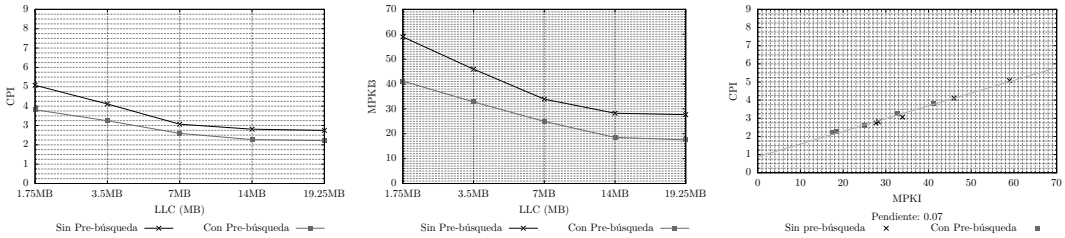
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.131: 416.gamess.2



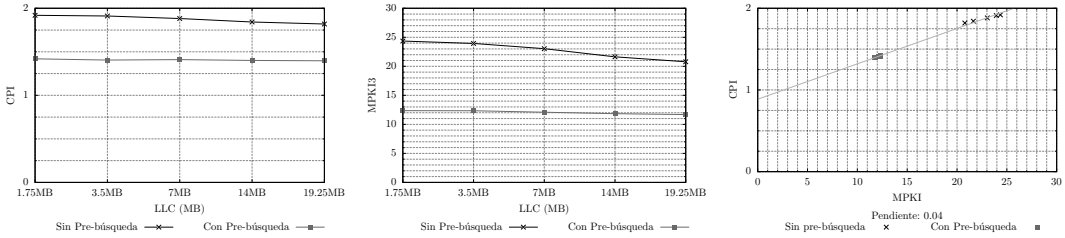
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.132: 416.gamess.3



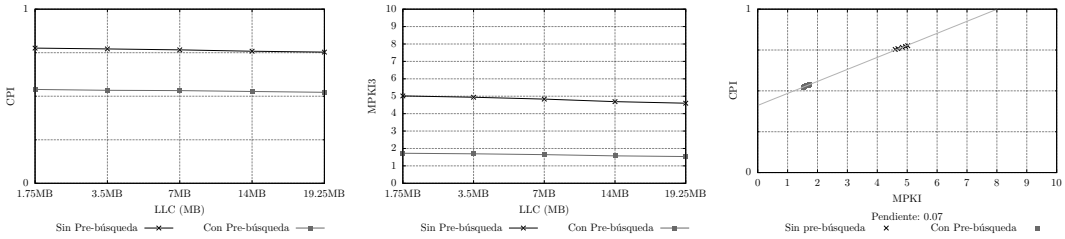
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.133: 429.mcf.1



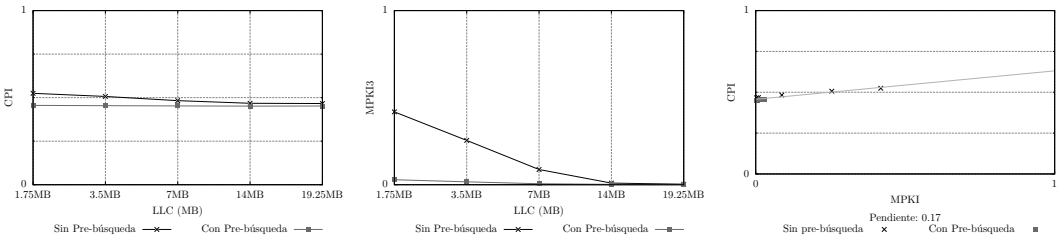
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.134: 433.milc.1



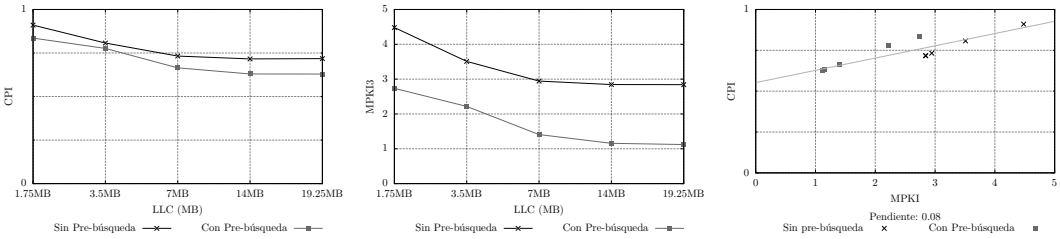
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.135: 434.zeusmp.1



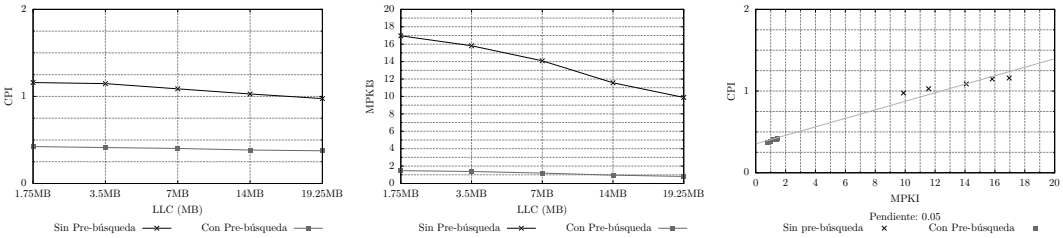
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.136: 435.gromacs.1



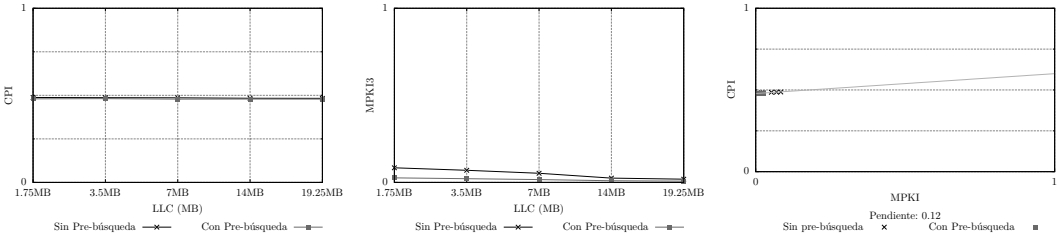
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.137: 436.cactusADM.1



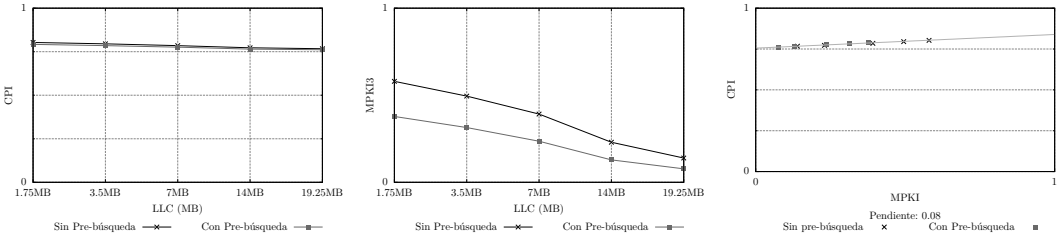
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.138: 437.leslie3d.1



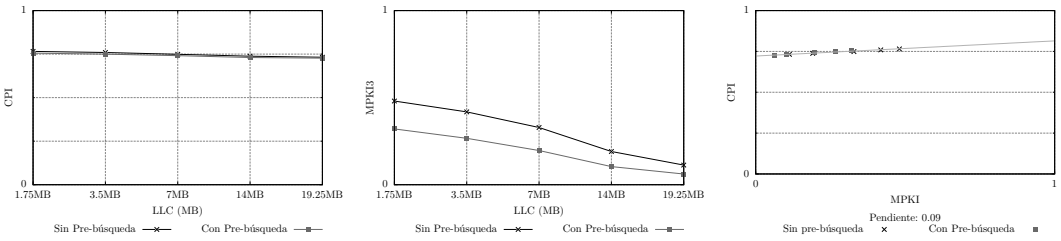
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.139: 444.namd.1



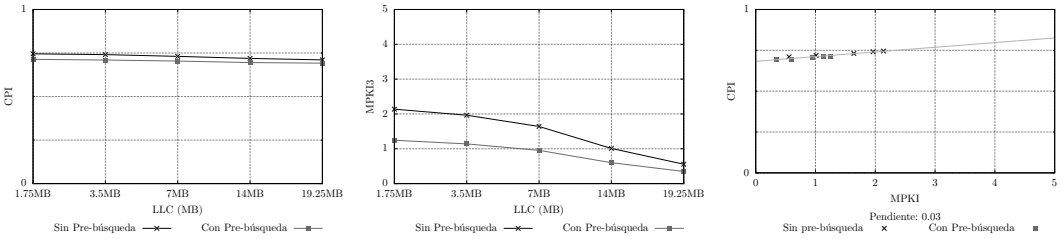
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.140: 445.gobmk.1



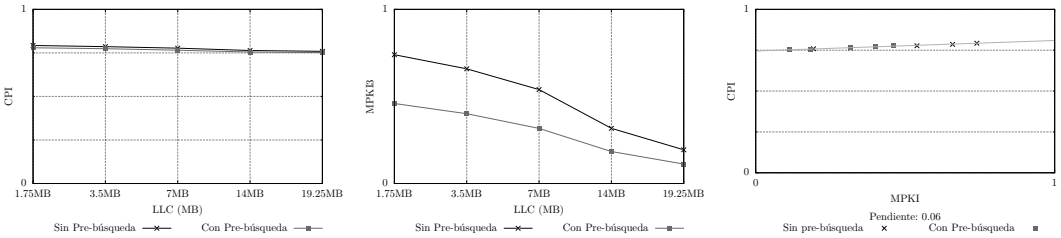
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.141: 445.gobmk.2



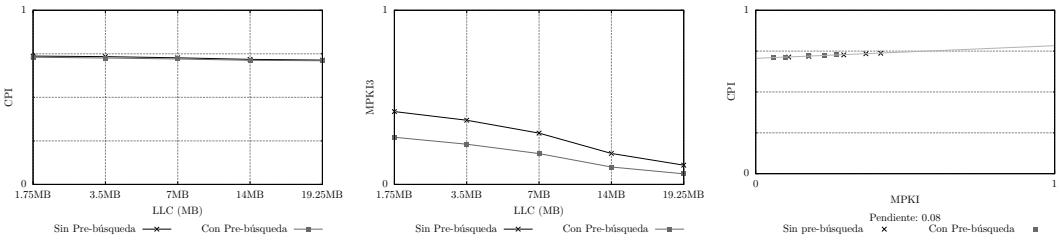
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.142: 445.gobmk.3



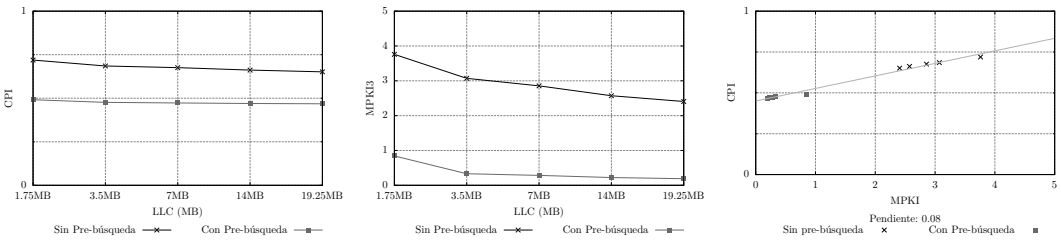
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.143: 445.gobmk.4



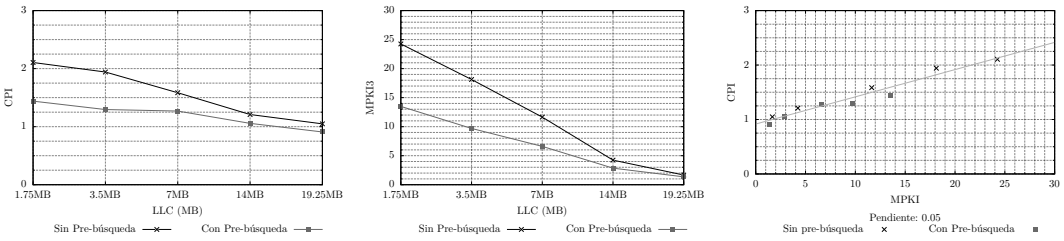
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.144: 445.gobmk.5



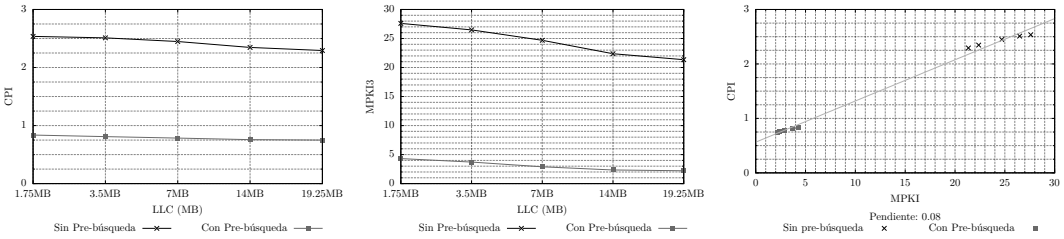
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.145: 447.dealII.1



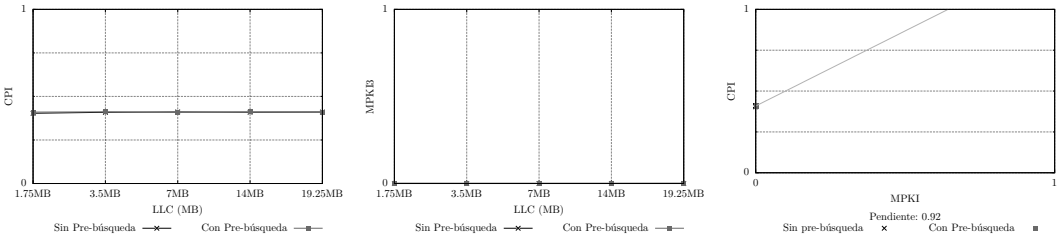
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.146: 450.soplex.1



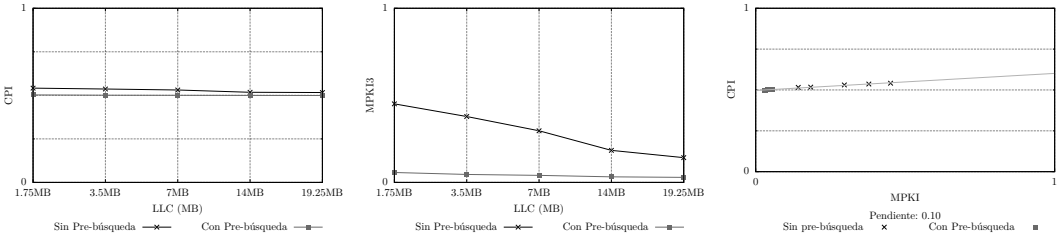
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.147: 450.soplex.2



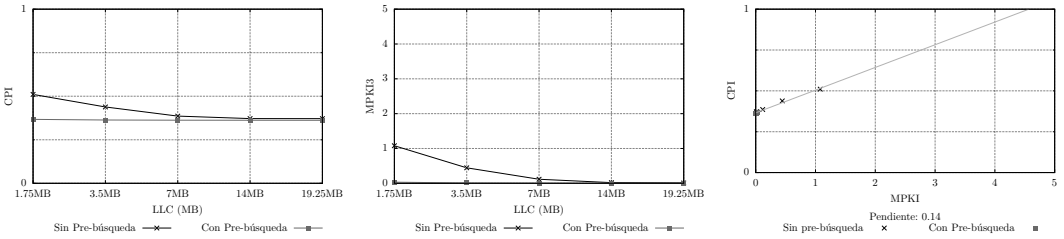
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.148: 453.povray.1



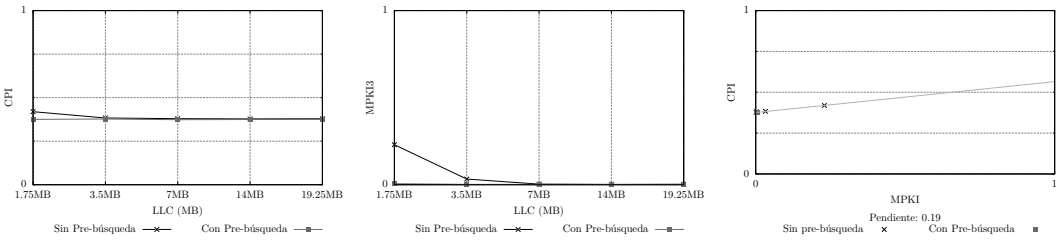
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.149: 454.calculix.1



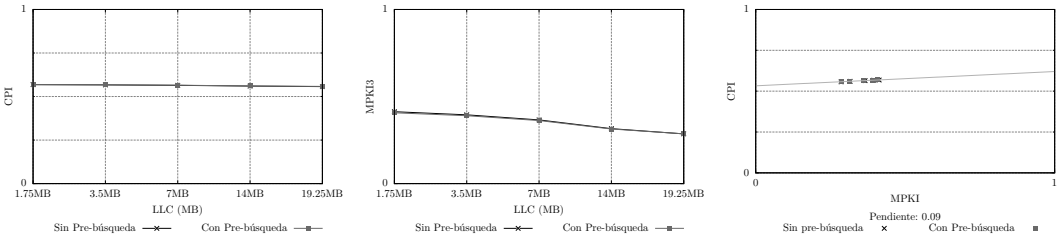
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.150: 456.hmmer.1



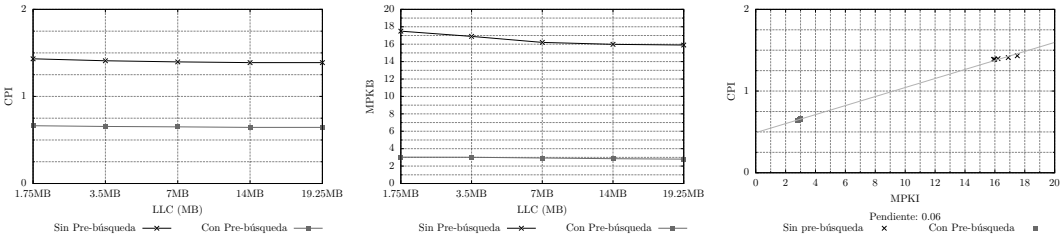
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.151: 456.hmmer.2



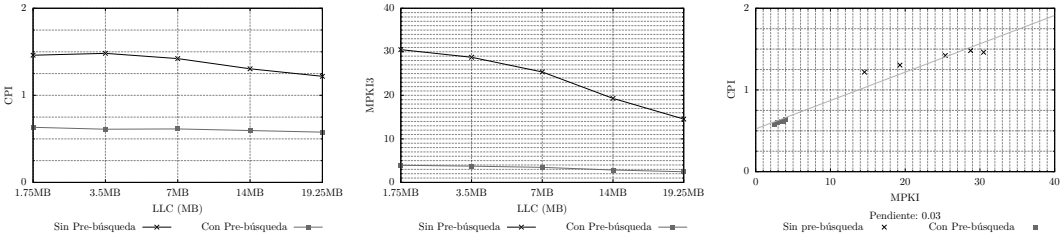
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.152: 458.sjeng.1



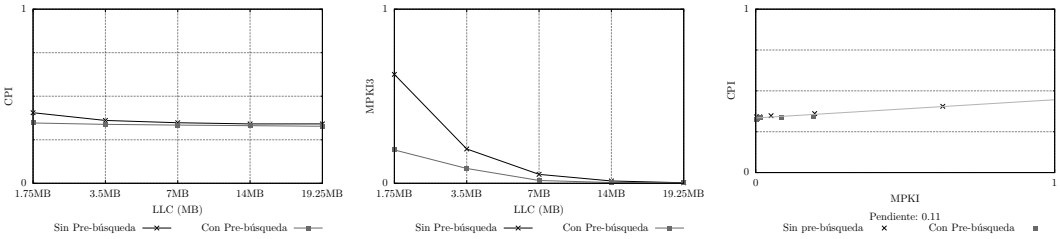
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.153: 459.GemsFDTD.1



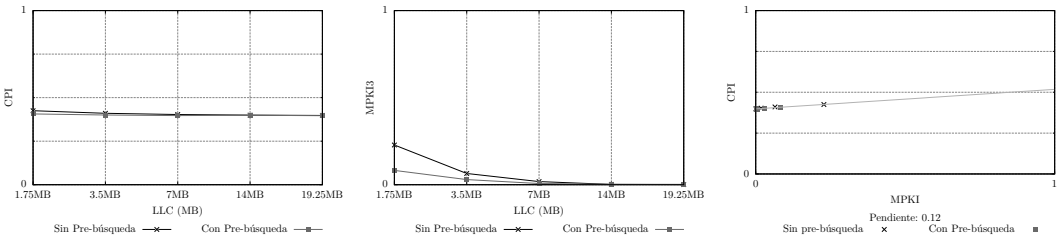
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.154: 462.libquantum.1



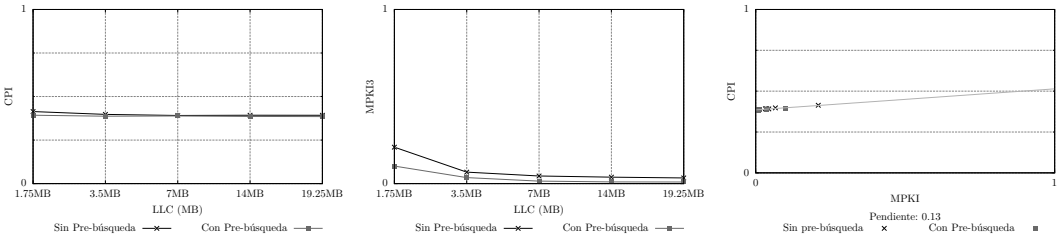
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.155: 464.h264ref.1



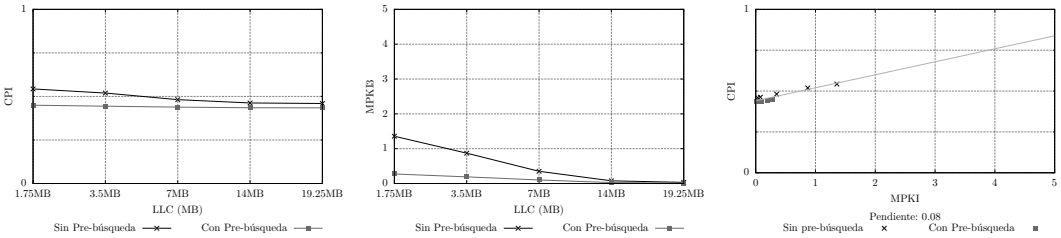
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.156: 464.h264ref.2



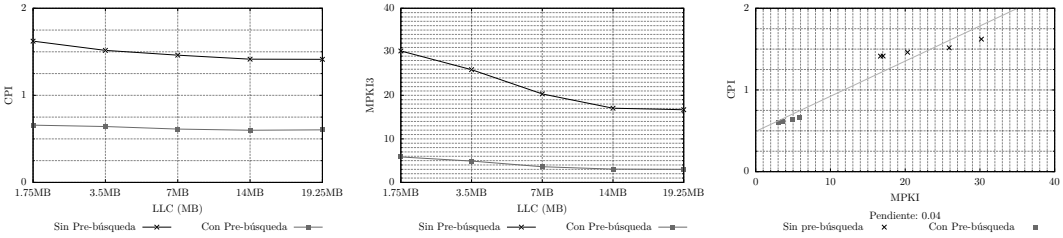
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.157: 464.h264ref.3



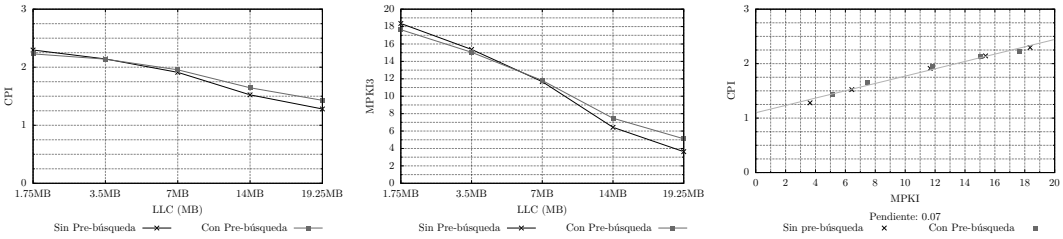
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.158: 465.tonto.1



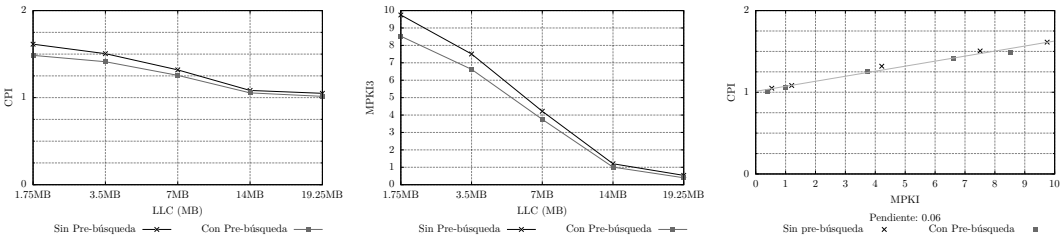
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.159: 470.lbm.1



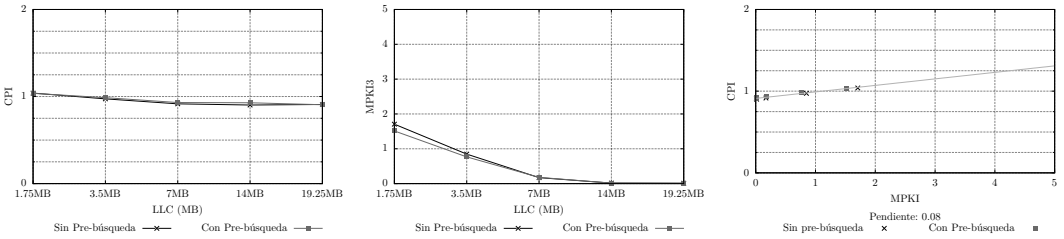
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.160: 471.omnetpp.1



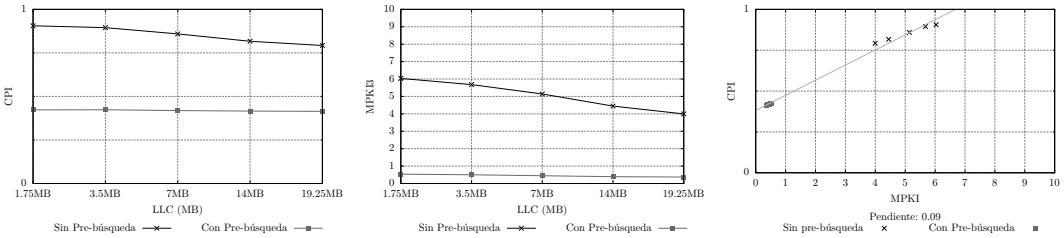
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.161: 473.astar.1



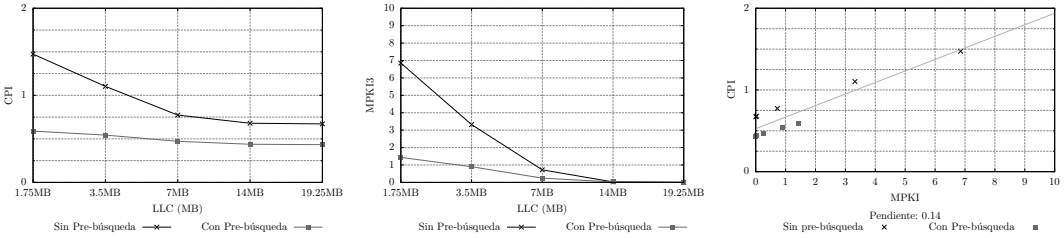
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.162: 473.astar.2



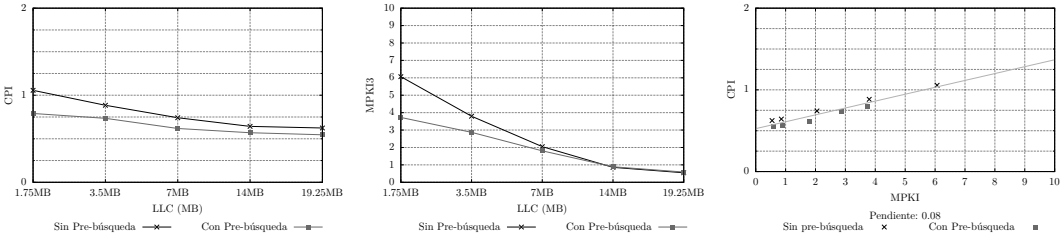
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.163: 481.wrf.1



(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.164: 482.sphinx3.1



(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.165: 483.xalancbmk.1

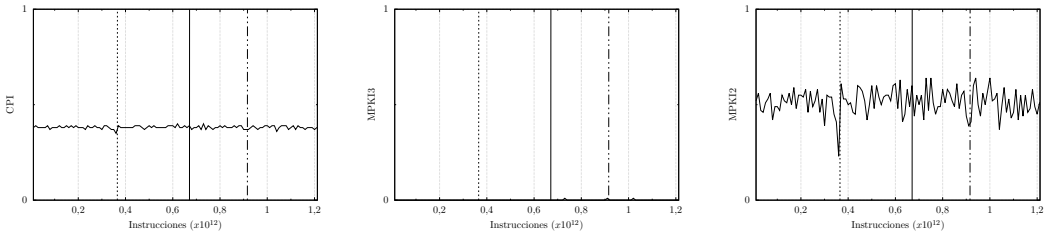
C.2. SPEC CPU2017 Rate

C.2.1. Caracterización General

Benchmark	Instr	Lect	Escr	MPKI1	MPKI2	MPKI3	CPI
500.perlbench_r.1	1218	28 %	19 %	7.22	0.49	0.01	0.38
500.perlbench_r.2	699	31 %	17 %	17.07	0.04	0.02	0.39
500.perlbench_r.3	667	31 %	18 %	6.16	1.53	0.11	0.43
502.gcc_r.1	195	28 %	13 %	25.10	2.28	0.31	0.70
502.gcc_r.2	231	29 %	14 %	24.32	3.02	0.45	0.72
502.gcc_r.3	233	30 %	11 %	27.32	2.49	0.29	0.70
502.gcc_r.4	186	29 %	14 %	26.15	4.49	1.99	0.84
502.gcc_r.5	258	26 %	21 %	42.34	3.82	1.49	0.96
503.bwaves_r.1	1300	34 %	6 %	12.28	0.10	0.08	0.43
503.bwaves_r.2	1570	36 %	6 %	16.03	0.13	0.11	0.48
503.bwaves_r.3	1403	35 %	6 %	14.25	0.12	0.10	0.45
503.bwaves_r.4	1828	34 %	6 %	13.07	0.11	0.09	0.44
505.mcf_r.1	924	33 %	13 %	59.27	16.98	6.04	1.30
507.cactuBSSN_r.1	1113	50 %	11 %	118.76	5.47	1.82	0.80
508.namd_r.1	1777	30 %	7 %	17.31	0.04	0.01	0.39
510.parest_r.1	3390	38 %	4 %	33.36	1.25	0.04	0.50
511.povray_r.1	3307	37 %	16 %	26.31	0.00	0.00	0.43
519.lbm_r.1	1277	21 %	11 %	49.11	0.08	0.04	0.64
520.omnetpp_r.1	1093	30 %	17 %	33.98	11.90	3.49	1.24
521.wrf_r.1	4204	28 %	7 %	11.10	0.31	0.09	0.88
523.xalancbmk_r.1	1274	30 %	7 %	45.03	2.35	0.12	0.66
525.x264_r.1	516	25 %	8 %	4.03	0.11	0.06	0.34
525.x264_r.2	1963	24 %	6 %	4.46	0.04	0.02	0.32
525.x264_r.3	1985	24 %	6 %	3.97	0.04	0.02	0.32
526.blender_r.1	1725	31 %	5 %	7.77	0.91	0.13	0.59
527.cam4_r.1	2685	24 %	11 %	18.79	0.94	0.08	0.59
531.deepsjeng_r.1	1869	25 %	12 %	3.61	0.42	0.25	0.59
538.imagick_r.1	4589	23 %	8 %	6.10	0.02	0.00	0.31
541.leela_r.1	2111	26 %	9 %	4.51	0.10	0.00	0.82
544.nab_r.1	2085	33 %	10 %	9.35	0.02	0.01	0.69
548.exchange2_r.1	2908	51 %	30 %	0.09	0.00	0.00	0.54
549.fotonik3d_r.1	1947	44 %	14 %	34.13	4.30	3.78	0.69
554.roms_r.1	2709	34 %	9 %	27.32	0.92	0.47	0.47
557.xz_r.1	403	23 %	8 %	13.73	4.86	1.19	0.99
557.xz_r.2	1044	23 %	3 %	9.58	0.71	0.10	0.43
557.xz_r.3	567	23 %	6 %	8.05	1.74	0.36	0.65
AVG	1590	31 %	11 %	21.70	2.00	0.64	0.61

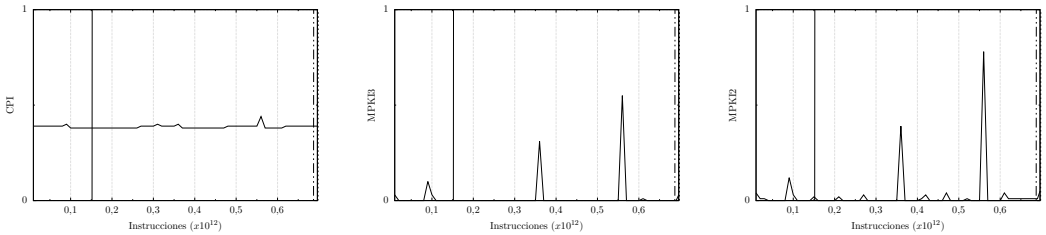
Cuadro C.2: Tabla resumen con los millones de instrucciones, porcentaje de instrucciones de lectura, porcentaje de instrucciones de escritura, MPKI en L1, MPKI en L2, MPKI en L3 y CPI de los programas de SPEC CPU2017Rate

C.2.2. Evolución Temporal



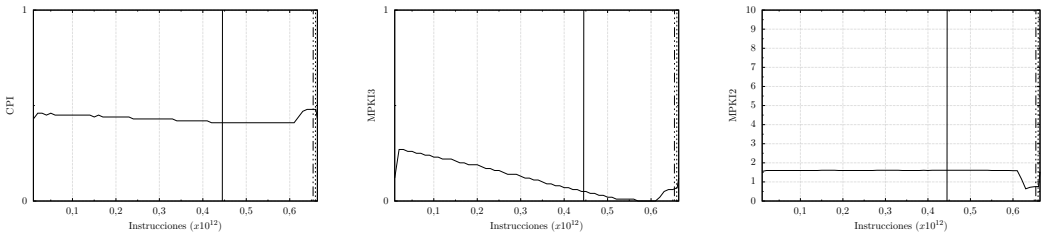
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.166: 500.perlbench_r.1



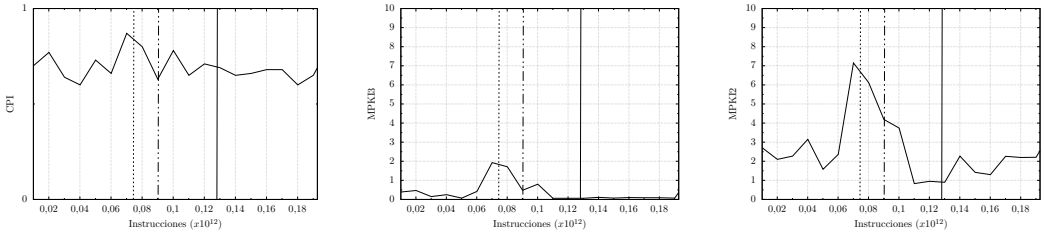
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.167: 500.perlbench_r.2



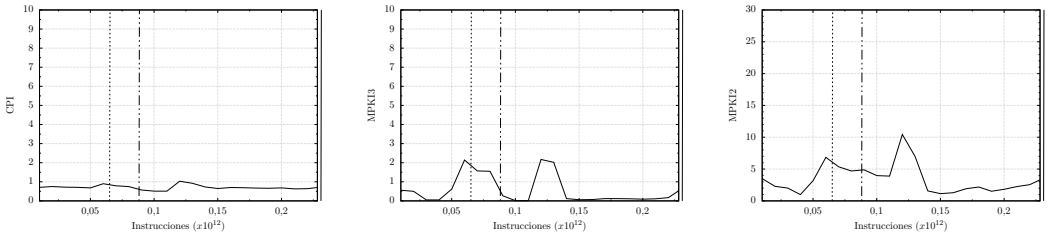
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.168: 500.perlbench_r.3



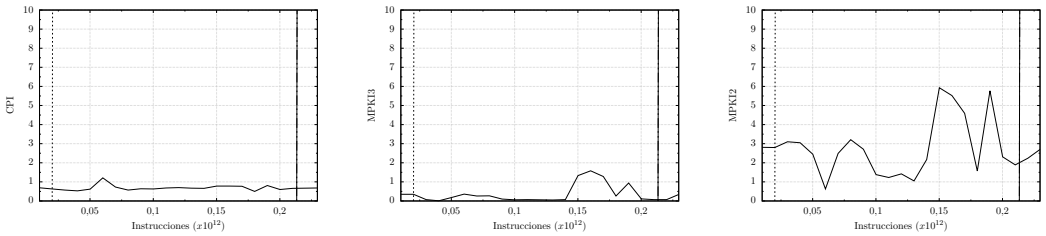
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.169: 502.gcc_r.1



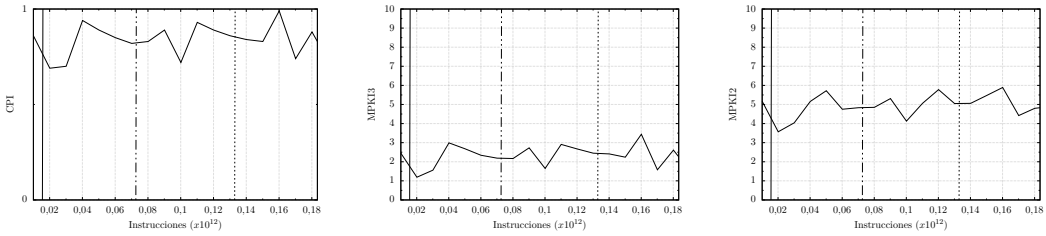
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.170: 502.gcc_r.2



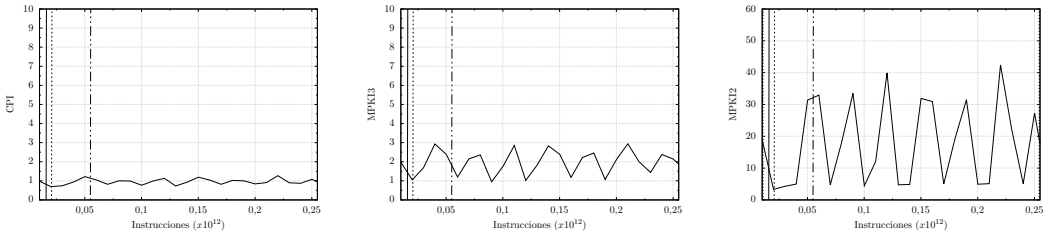
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.171: 502.gcc_r.3



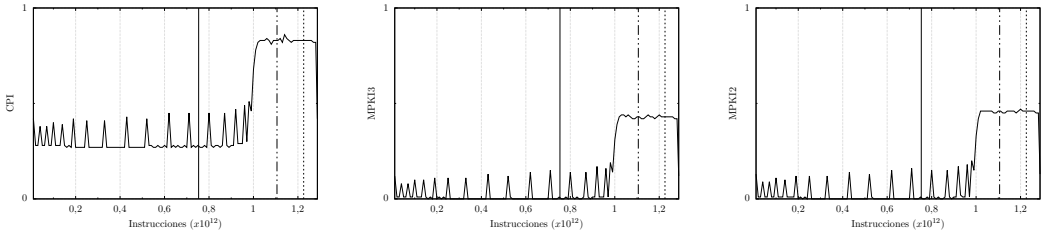
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.172: 502.gcc_r.4



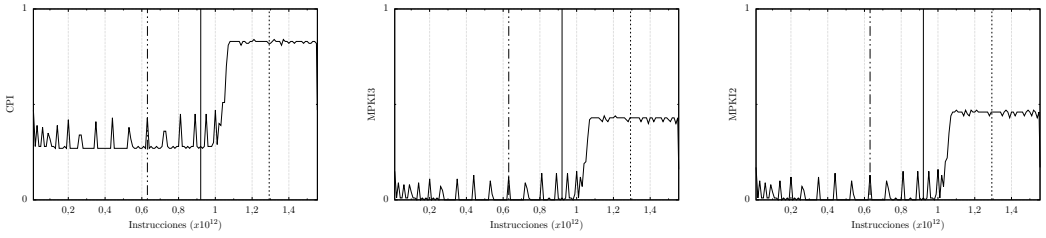
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.173: 502.gcc_r.5



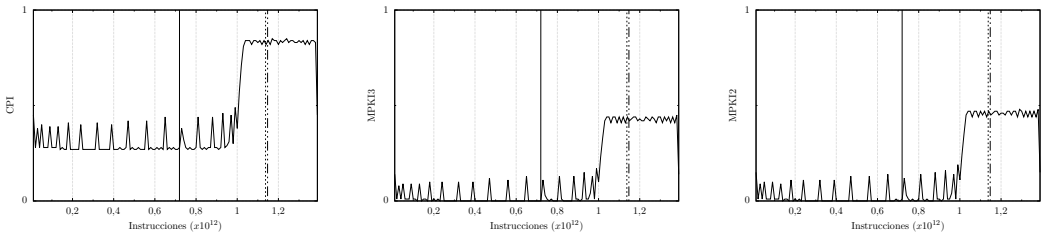
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.174: 503.bwaves_r.1



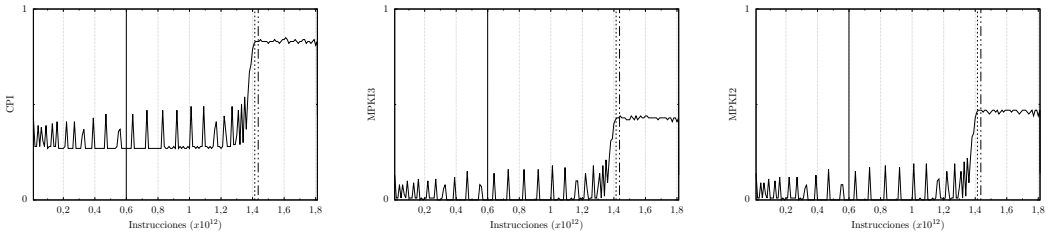
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.175: 503.bwaves_r.2



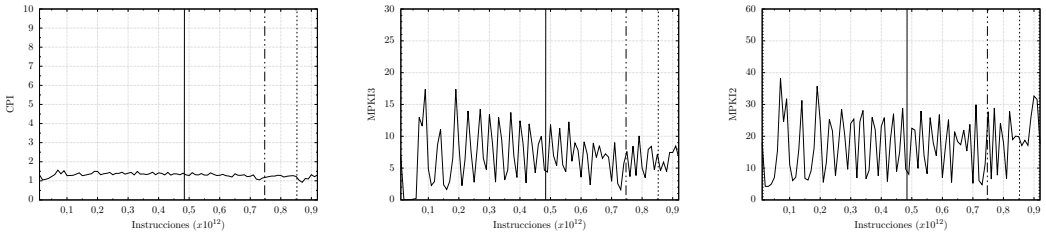
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.176: 503.bwaves_r.3



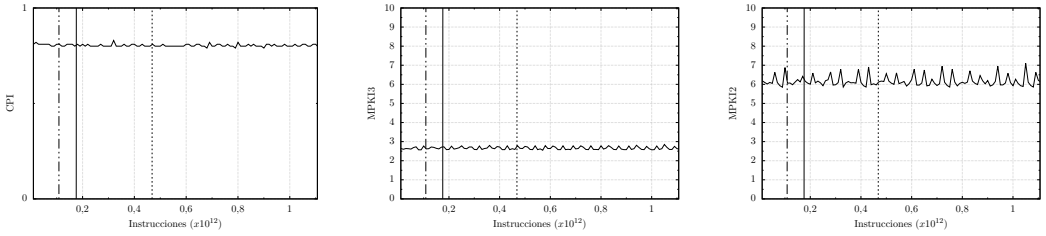
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.177: 503.bwaves_r.4



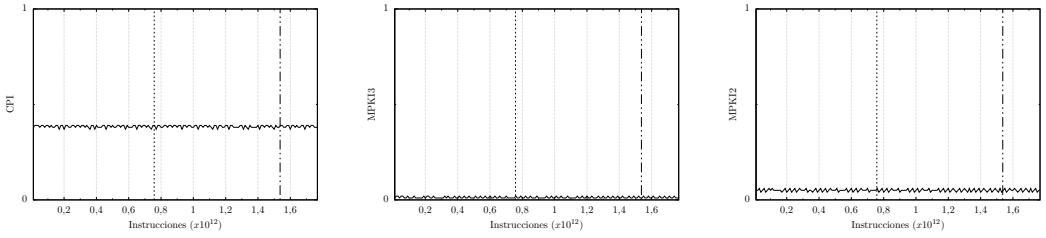
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.178: 505.mcf_r.1



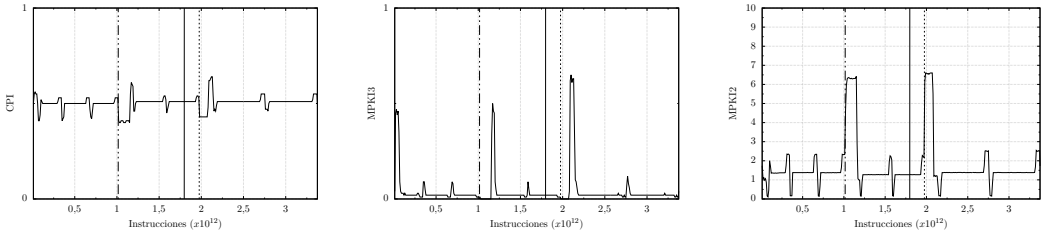
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.179: 507.cactuBSSN_r.1



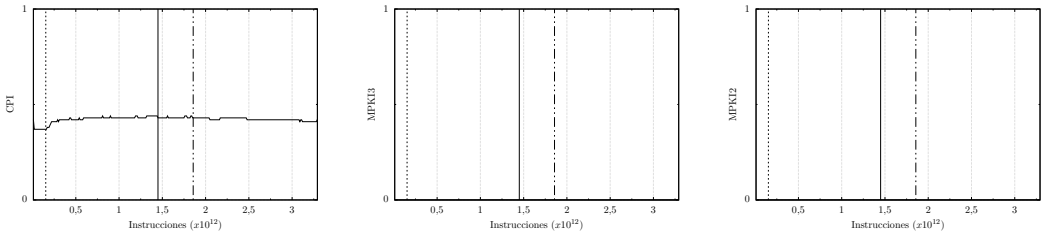
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.180: 508.namd_r.1



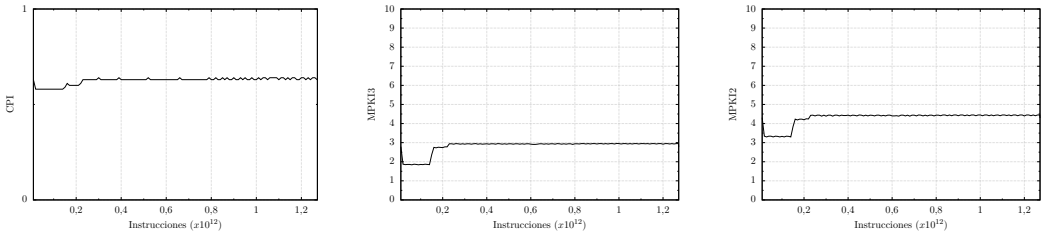
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.181: 510.parest_r.1



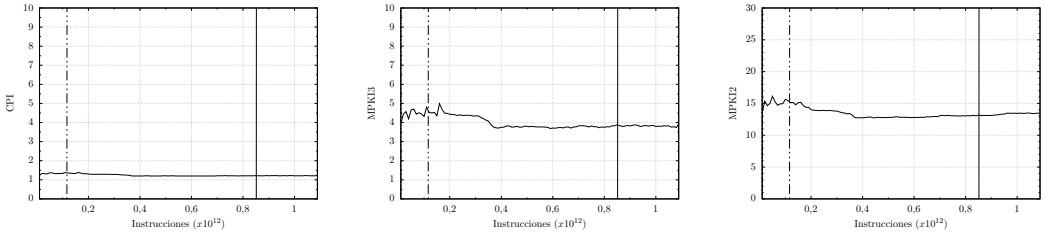
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.182: 511.povray_r.1



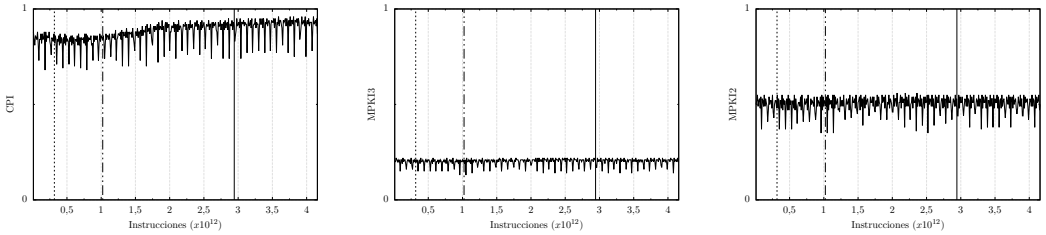
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.183: 519.lbm_r.1



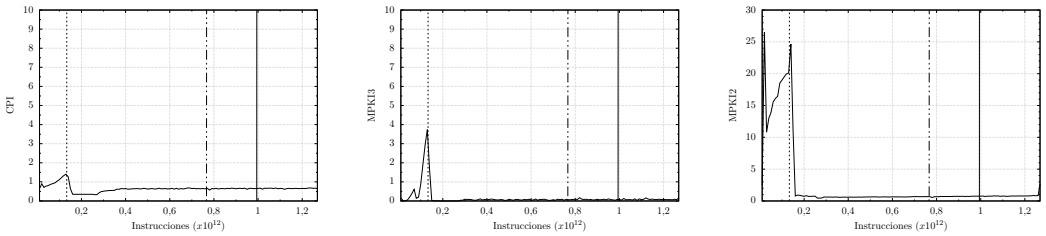
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.184: 520.omnetpp_r.1



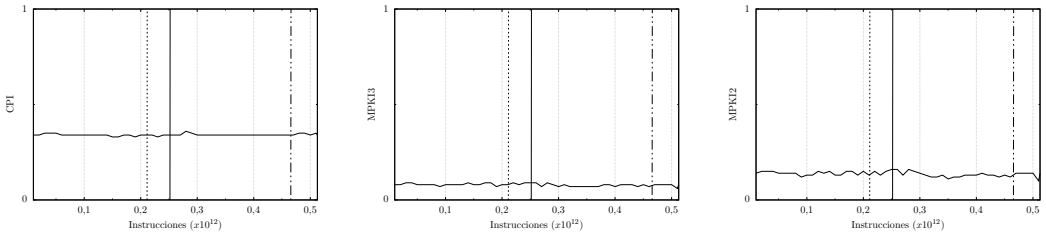
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.185: 521.wrf_r.1



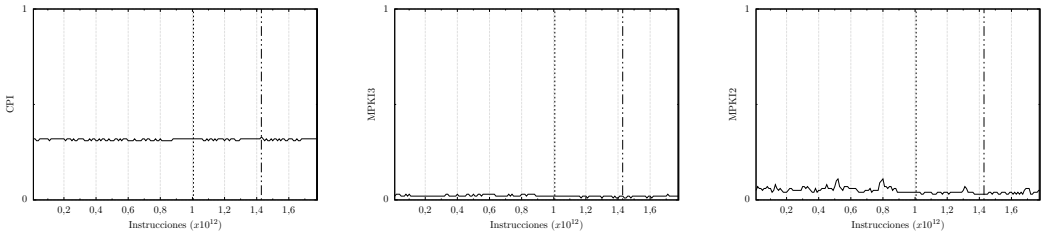
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.186: 523.xalancbmk_r.1



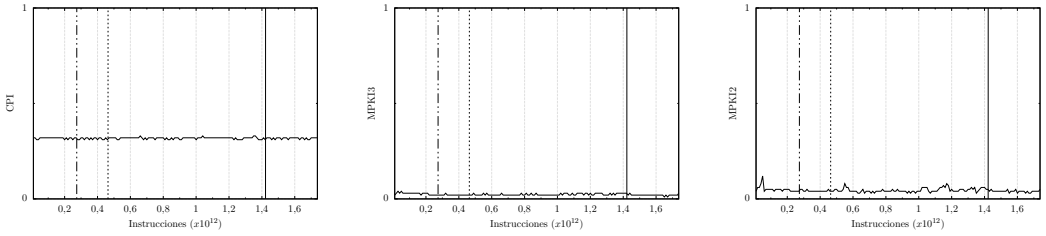
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.187: 525.x264_r.1



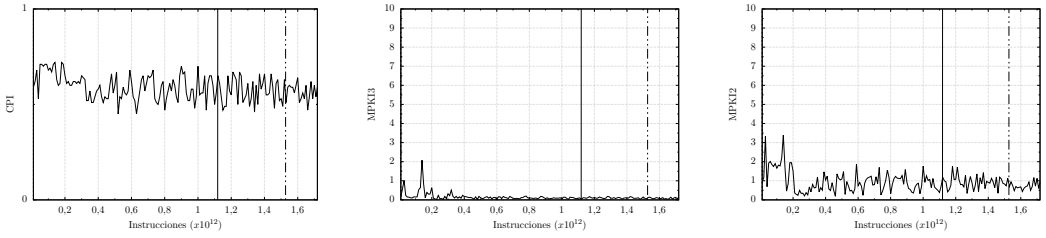
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.188: 525.x264_r.2



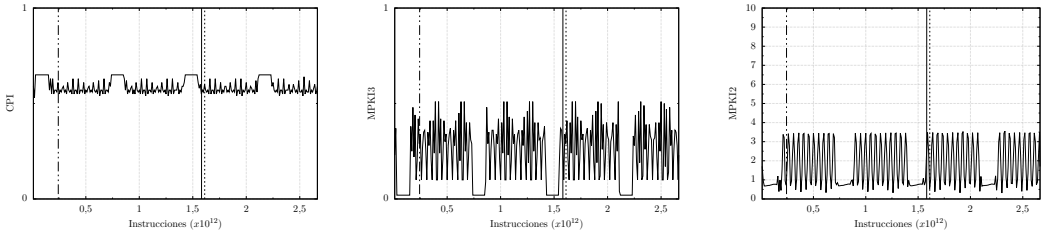
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.189: 525.x264_r.3



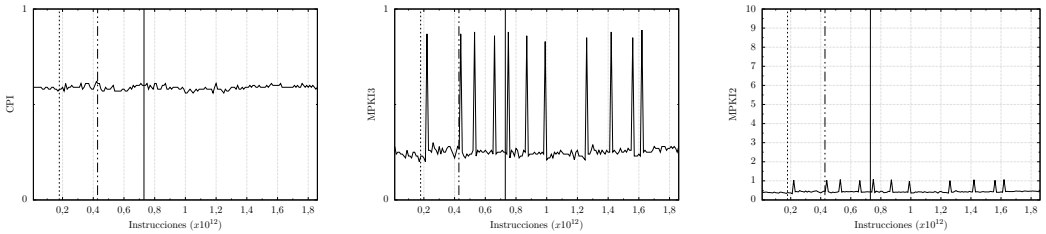
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.190: 526.blender_r.1



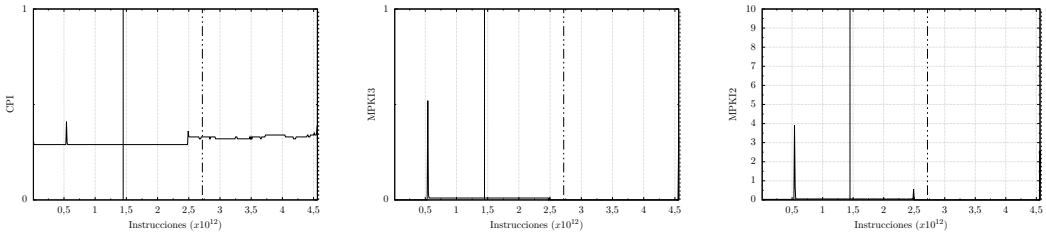
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.191: 527.cam4_r.1



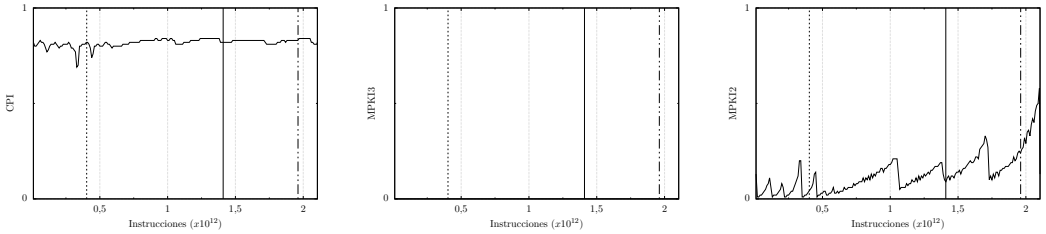
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.192: 531.deepsjeng_r.1



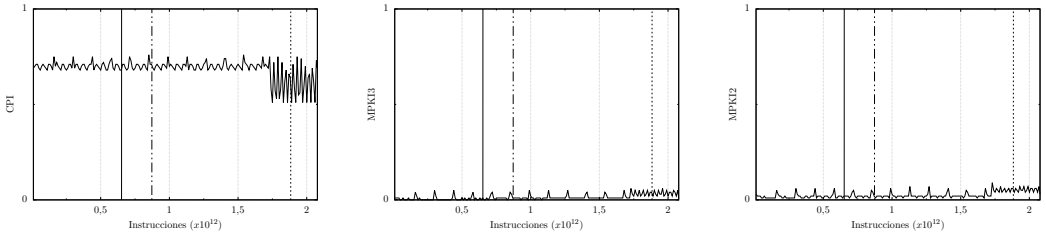
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.193: 538.imagick_r.1



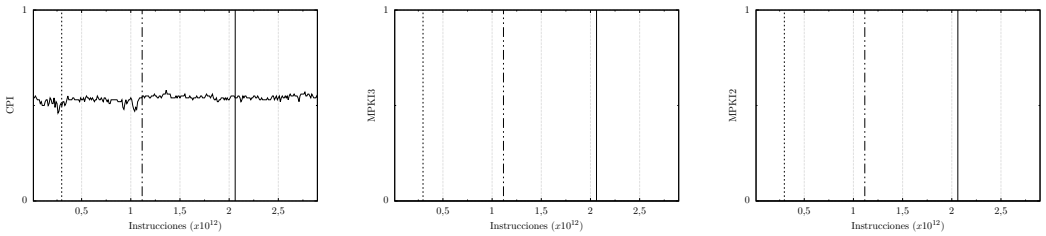
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.194: 541.leela_r.1



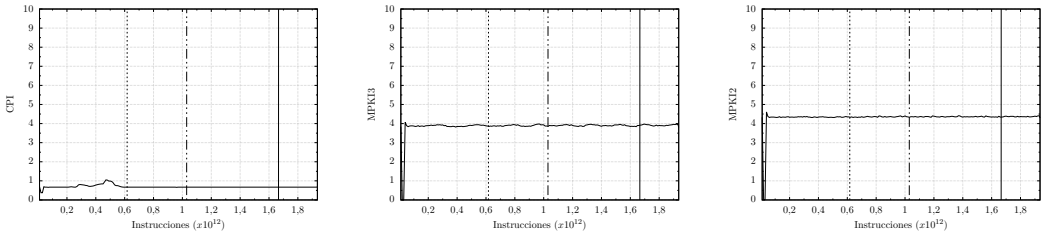
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.195: 544.nab_r.1



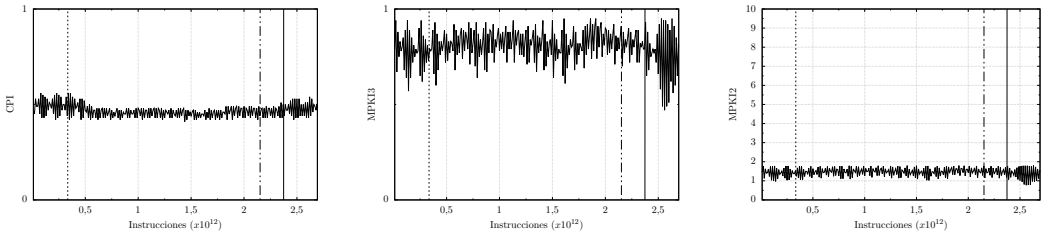
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.196: 548.exchange2_r.1



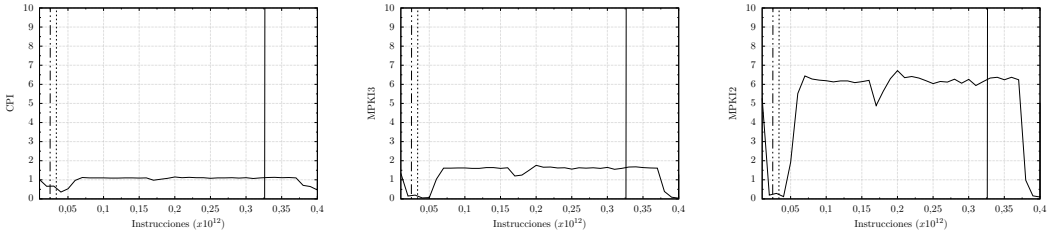
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.197: 549.fotonik3d_r.1



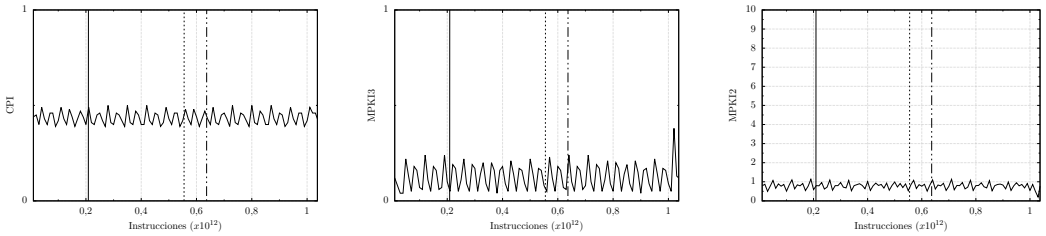
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.198: 554.roms_r.1



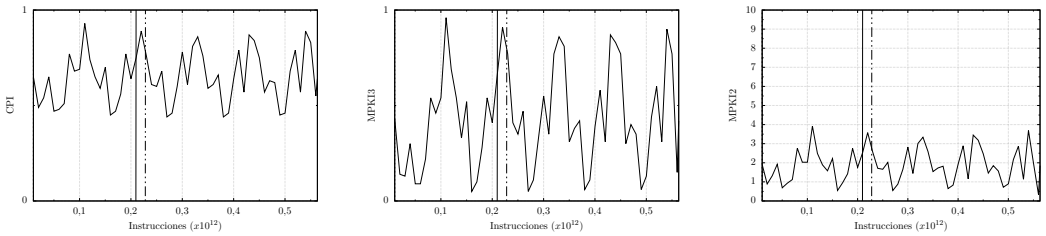
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.199: 557.xz_r.1



(a) CPI **(b)** MPKI **(c)** APKI

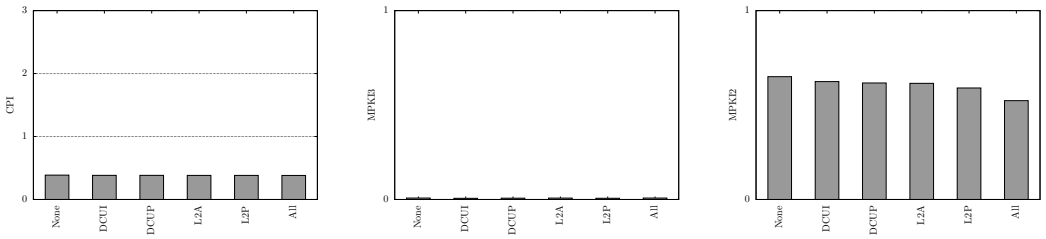
Figura C.200: 557.xz_r.2



(a) CPI **(b)** MPKI **(c)** APKI

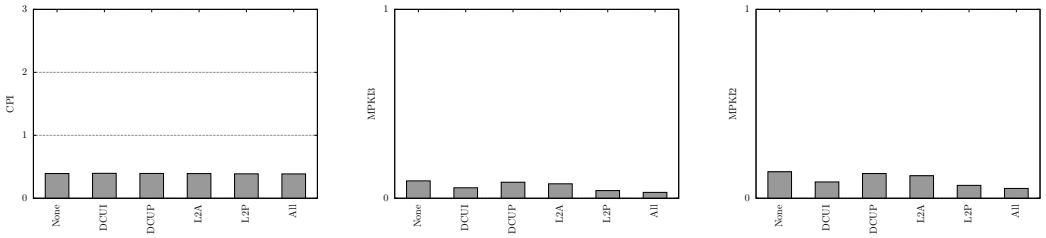
Figura C.201: 557.xz_r.3

C.2.3. Pre-búscadores Hardware



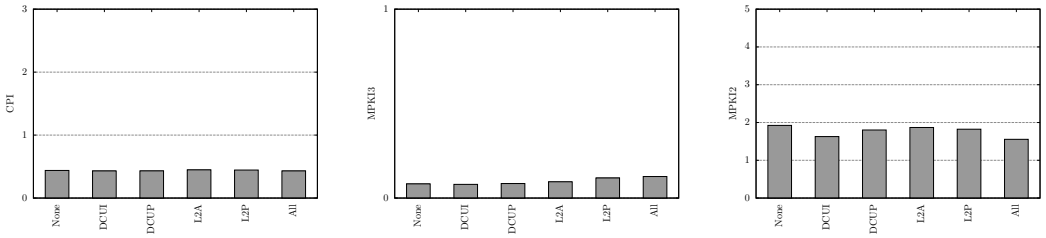
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.202: 500.perlbench_r.1



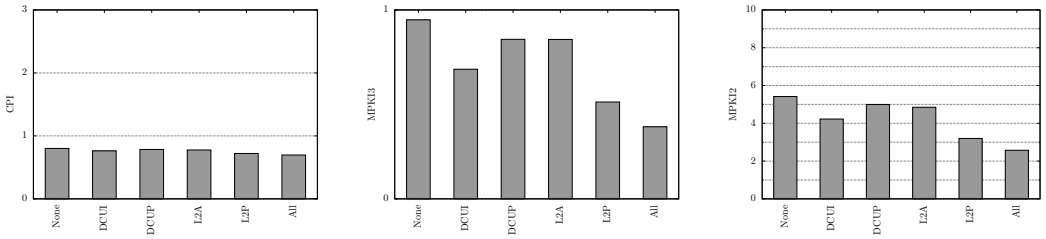
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.203: 500.perlbench_r.2



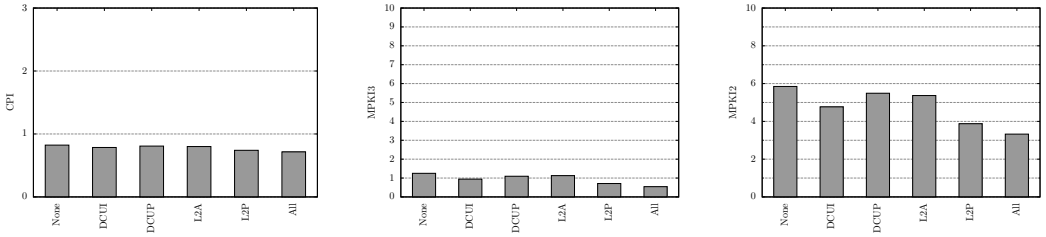
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.204: 500.perlbench_r.3



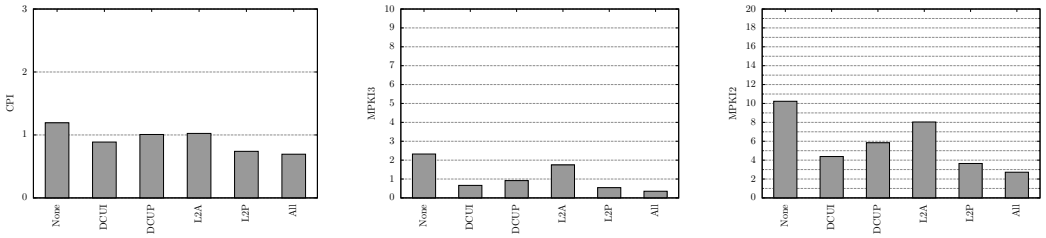
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.205: 502.gcc_r.1



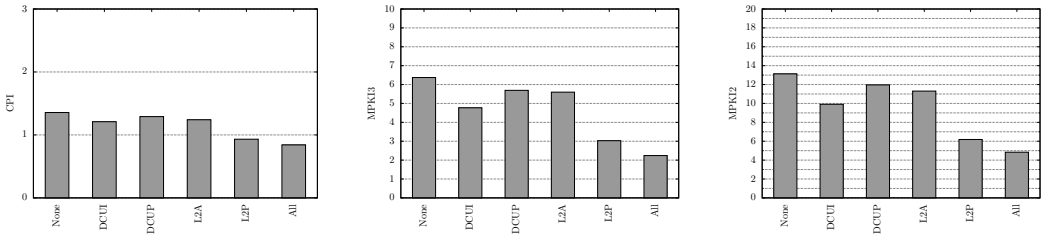
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.206: 502.gcc_r.2



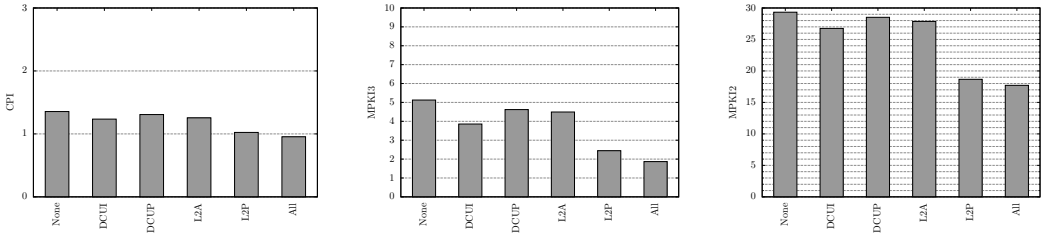
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.207: 502.gcc_r.3



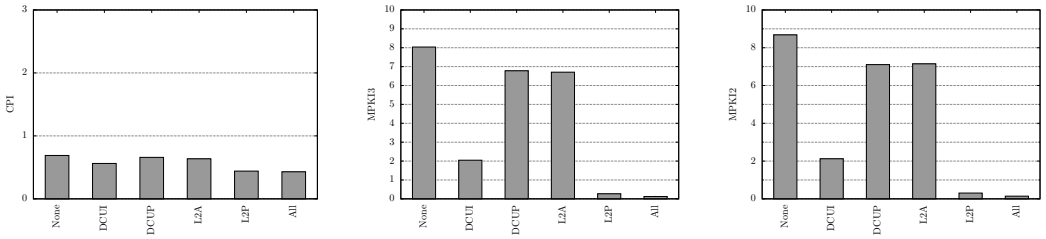
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.208: 502.gcc_r.4



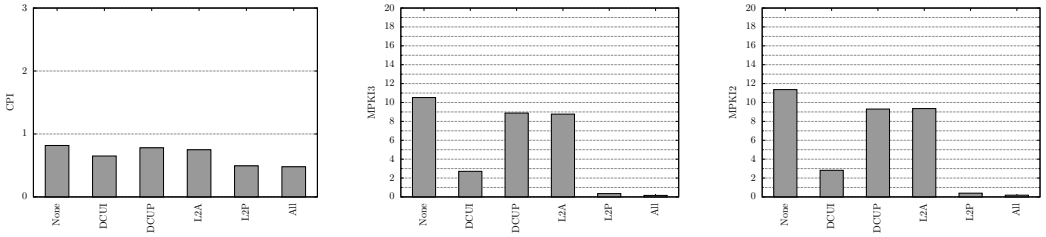
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.209: 502.gcc_r.5



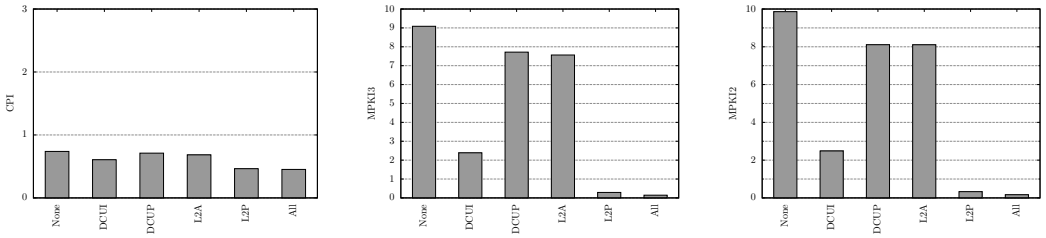
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.210: 503.bwaves_r.1



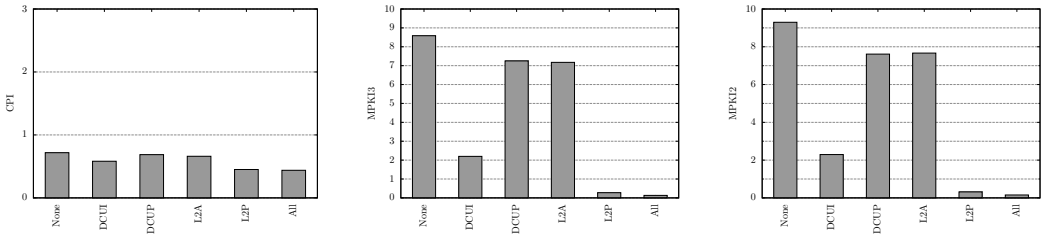
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.211: 503.bwaves_r.2



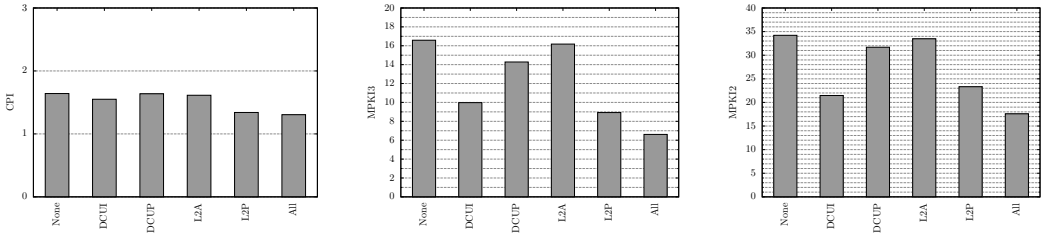
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.212: 503.bwaves_r.3



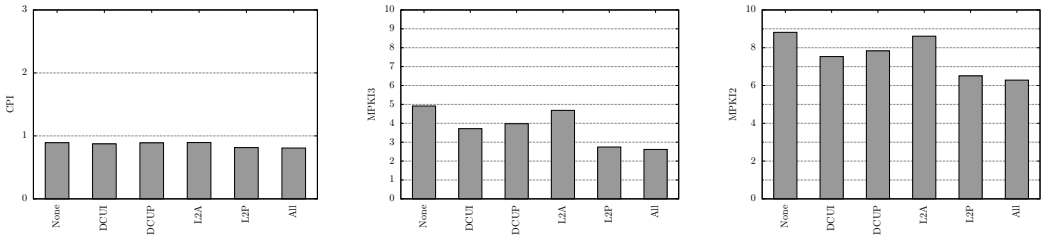
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.213: 503.bwaves_r.4



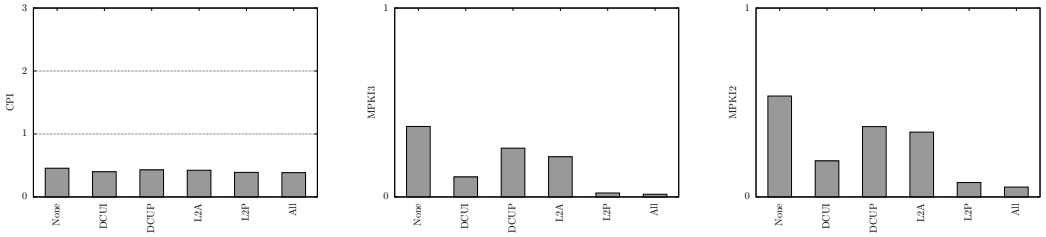
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.214: 505.mcf_r.1



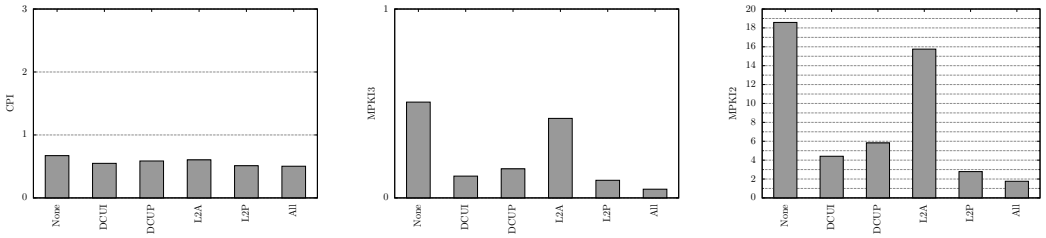
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.215: 507.cactuBSSN_r.1



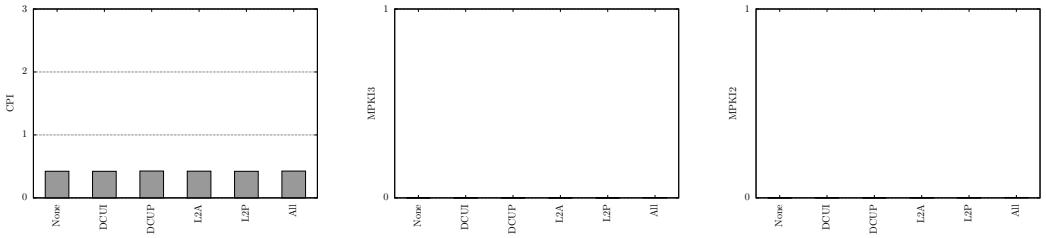
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.216: 508.namd_r.1



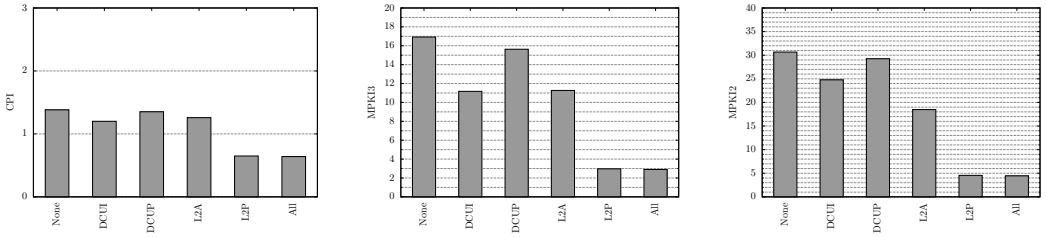
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.217: 510.parest_r.1



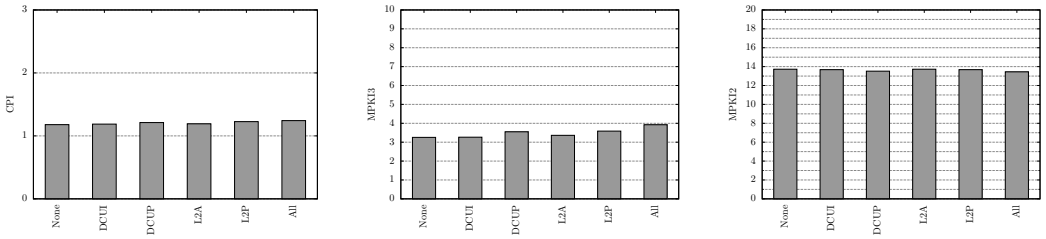
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.218: 511.povray_r.1



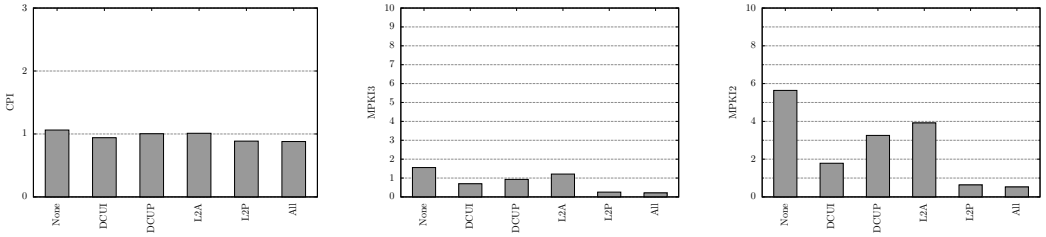
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.219: 519.lbm_r.1



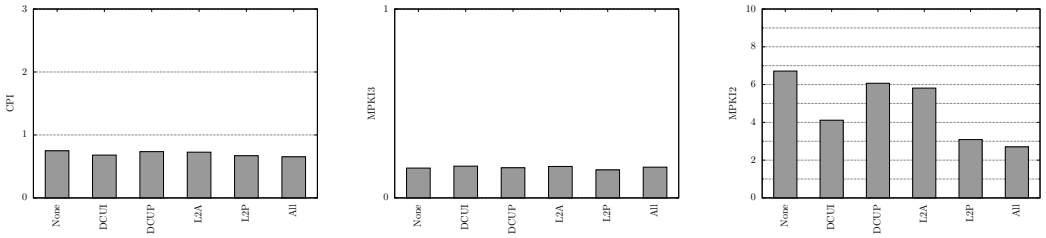
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.220: 520.omnetpp_r.1



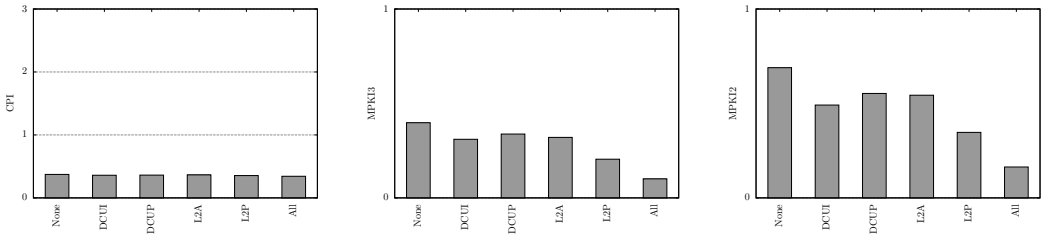
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.221: 521.wrf_r.1



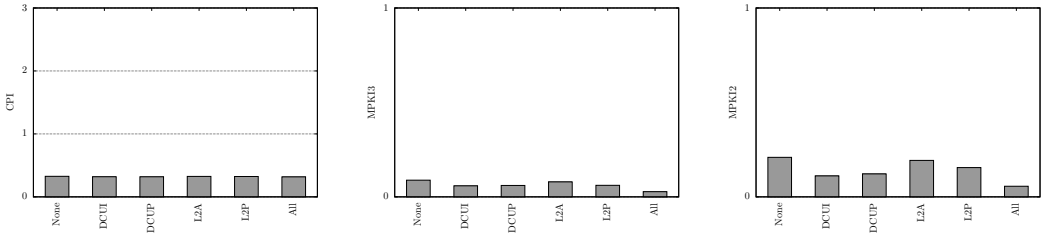
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.222: 523.xalancbmk_r.1



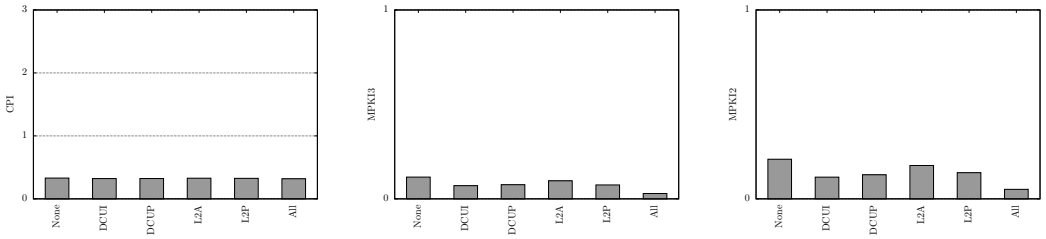
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.223: 525.x264_r.1



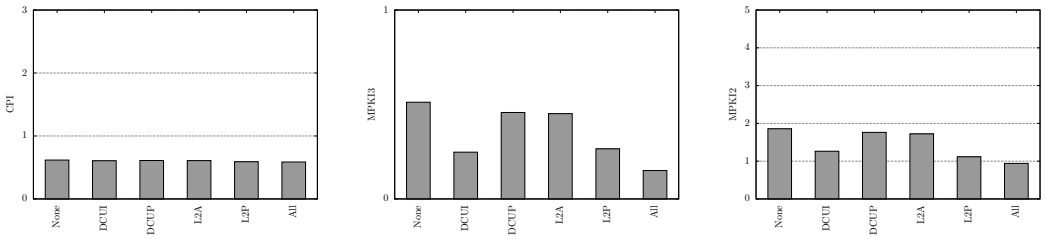
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.224: 525.x264_r.2



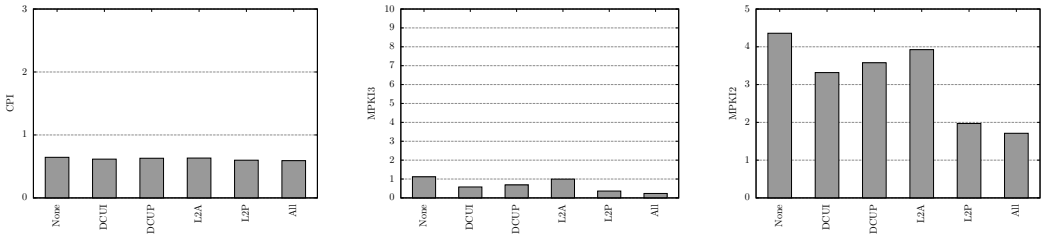
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.225: 525.x264_r.3



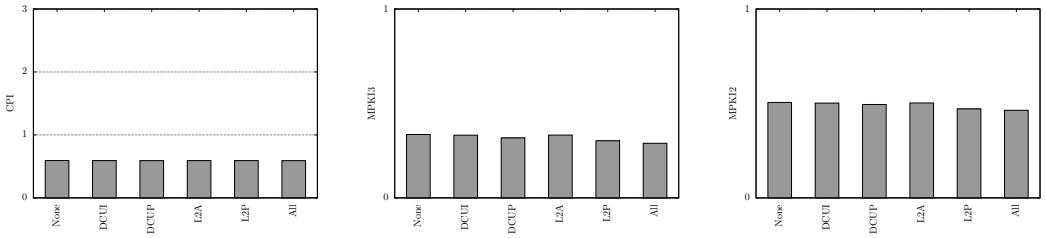
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.226: 526.blender_r.1



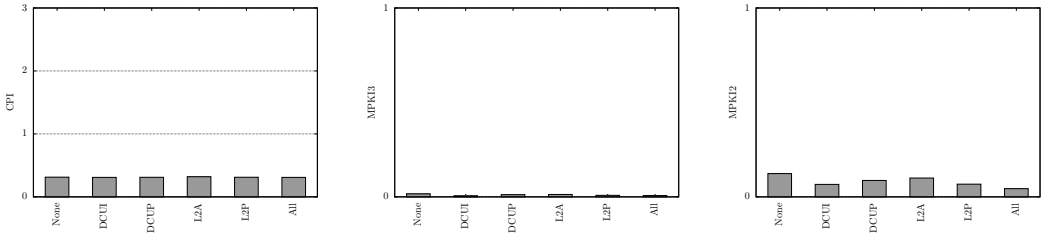
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.227: 527.cam4_r.1



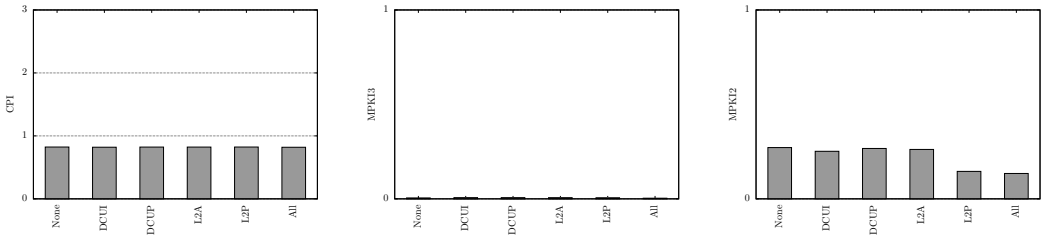
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.228: 531.deepsjeng_r.1



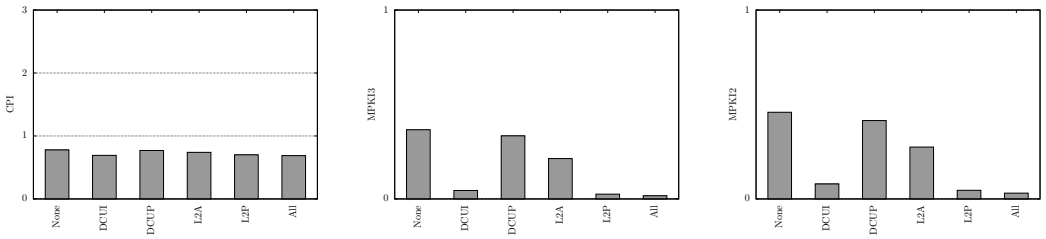
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.229: 538.imagick_r.1



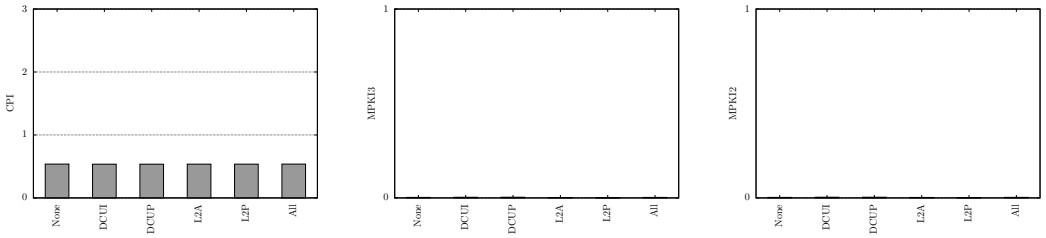
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.230: 541.leela_r.1



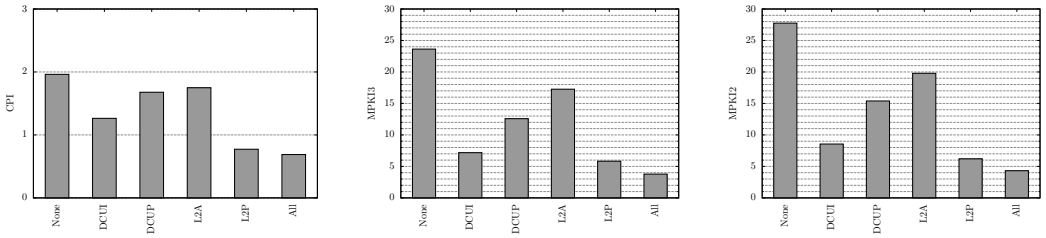
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.231: 544.nab_r.1



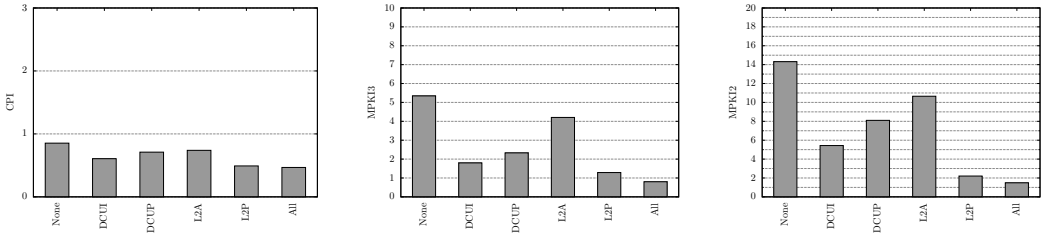
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.232: 548.exchange2_r.1



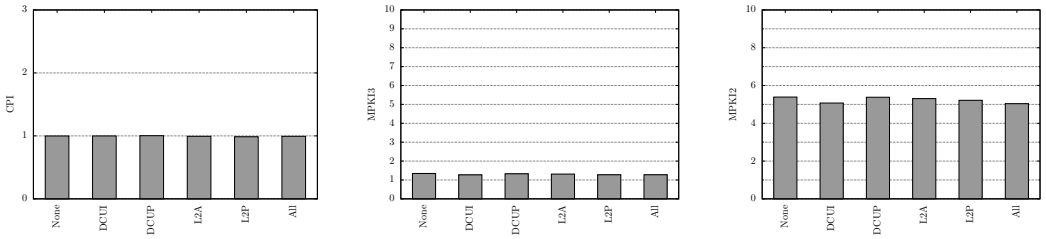
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.233: 549.fotonik3d_r.1



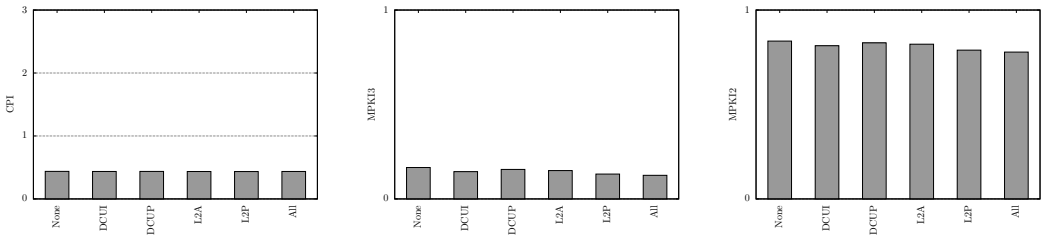
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.234: 554.roms_r.1



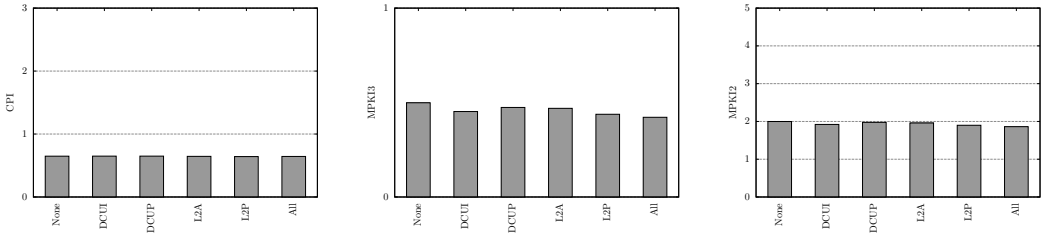
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.235: 557.xz_r.1



(a) CPI **(b)** MPKI **(c)** APKI

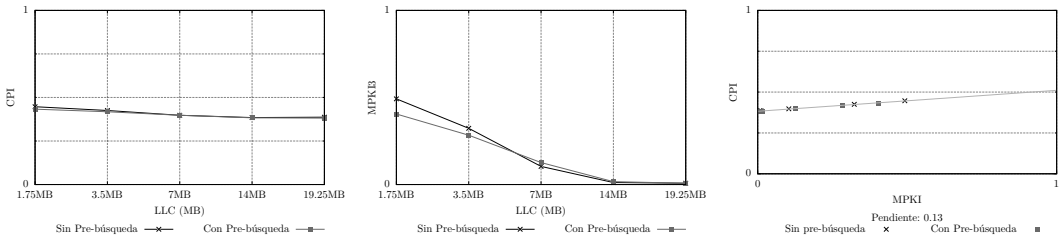
Figura C.236: 557.xz_r.2



(a) CPI **(b)** MPKI **(c)** APKI

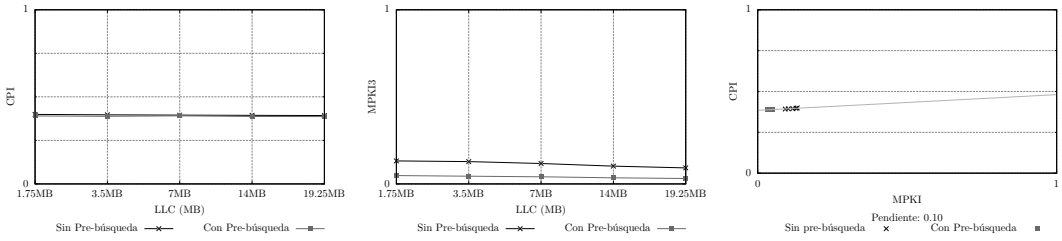
Figura C.237: 557.xz_r.3

C.2.4. Tamaño de LLC



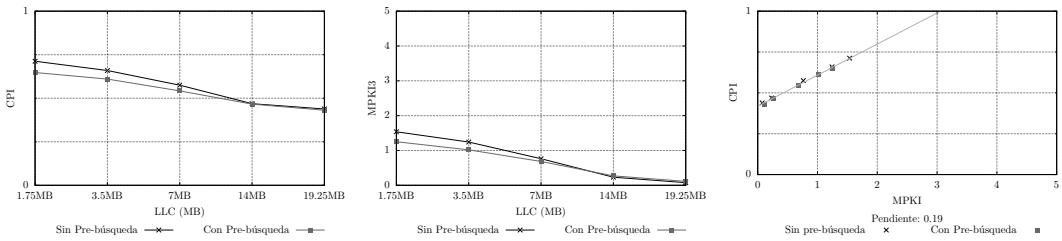
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.238: 500.perlbench_r.1



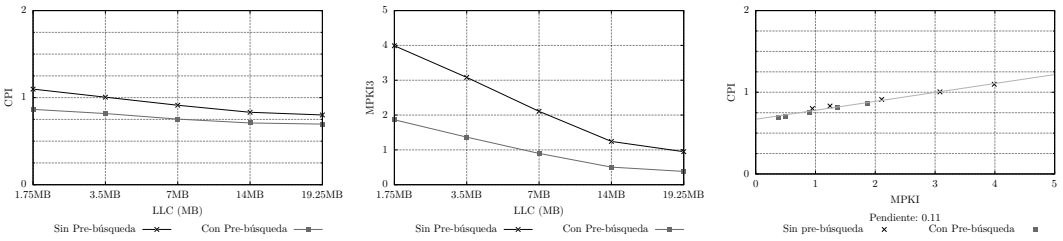
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.239: 500.perlbench_r.2



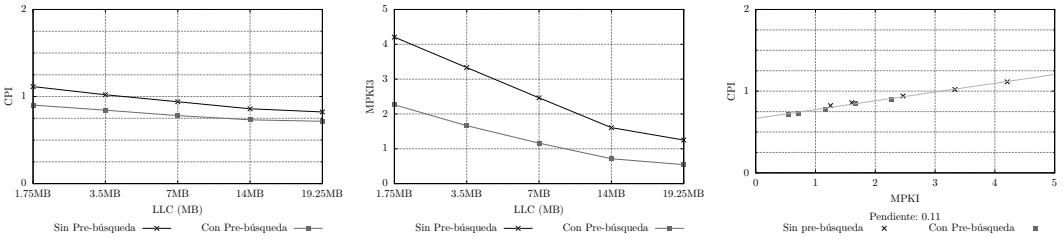
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.240: 500.perlbench_r.3



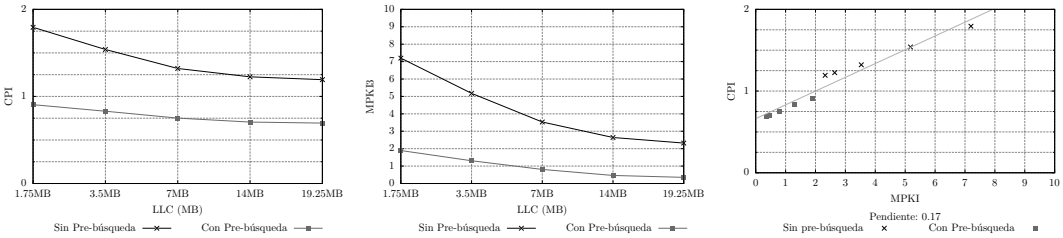
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.241: 502.gcc_r.1



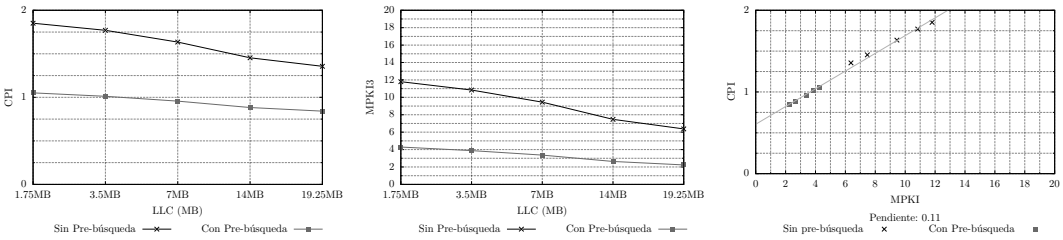
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.242: 502.gcc_r.2



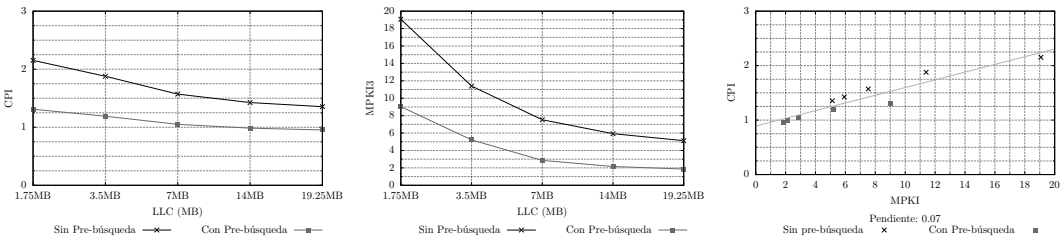
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.243: 502.gcc_r.3



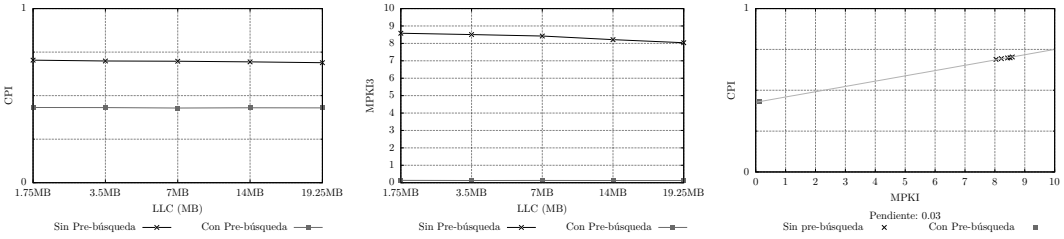
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.244: 502.gcc_r.4



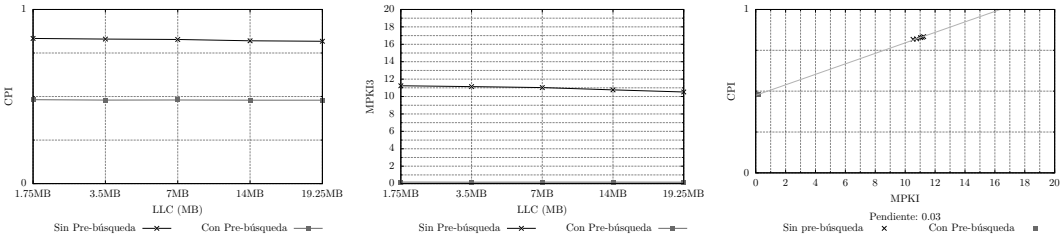
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.245: 502.gcc_r.5



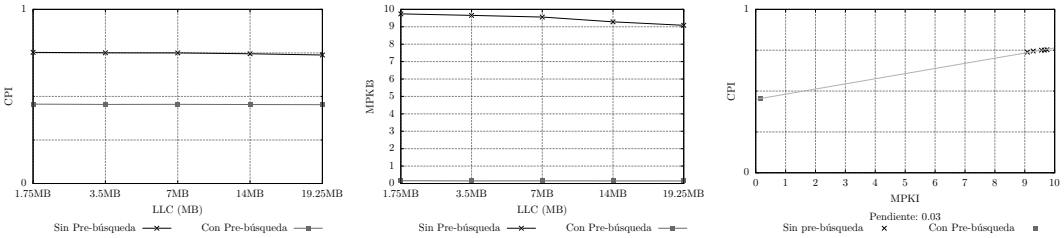
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.246: 503.bwaves_r.1



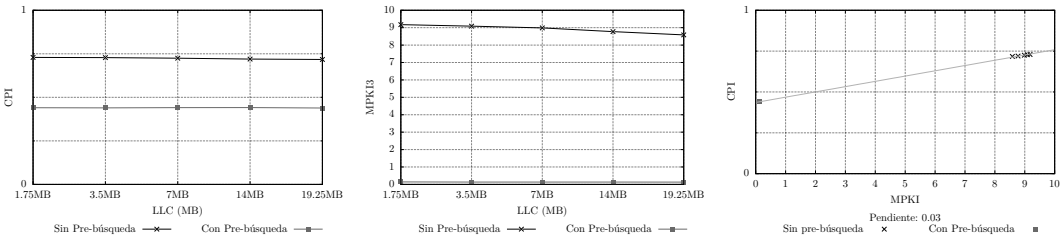
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.247: 503.bwaves_r.2



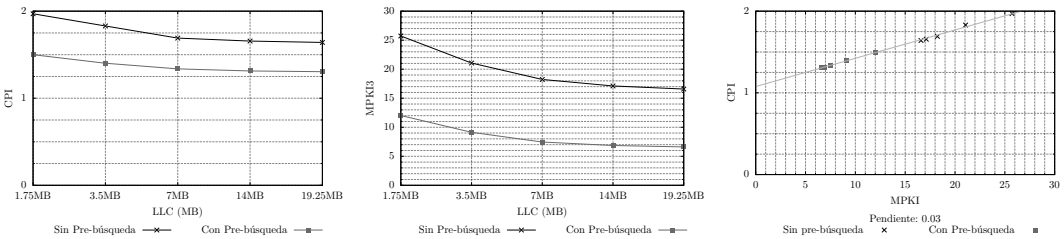
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.248: 503.bwaves_r.3



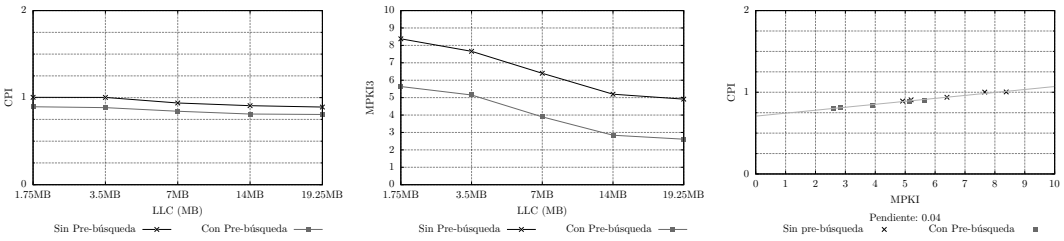
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.249: 503.bwaves_r.4



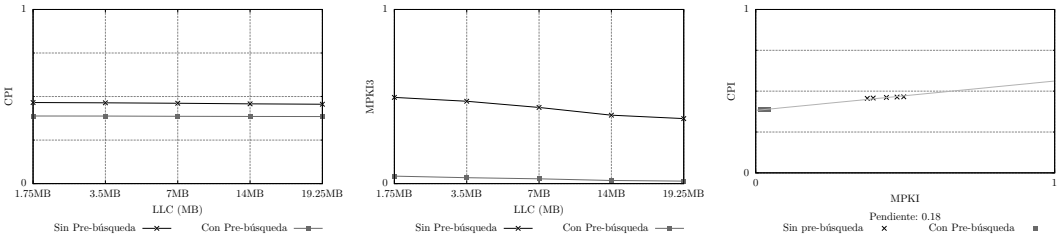
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.250: 505.mcf_r.1



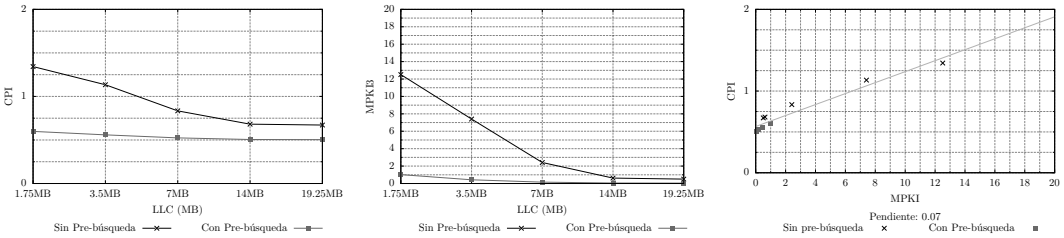
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.251: 507.cactuBSSN_r.1



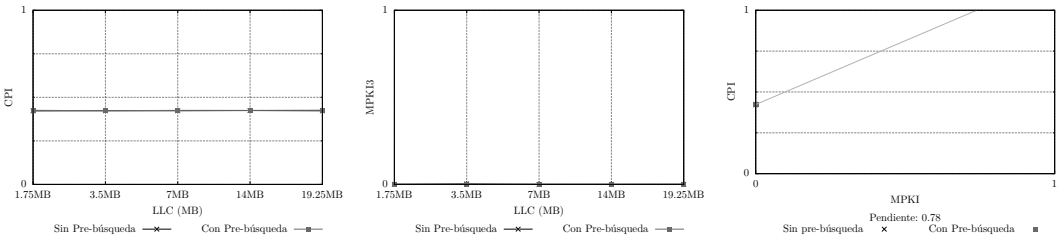
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.252: 508.namd_r.1



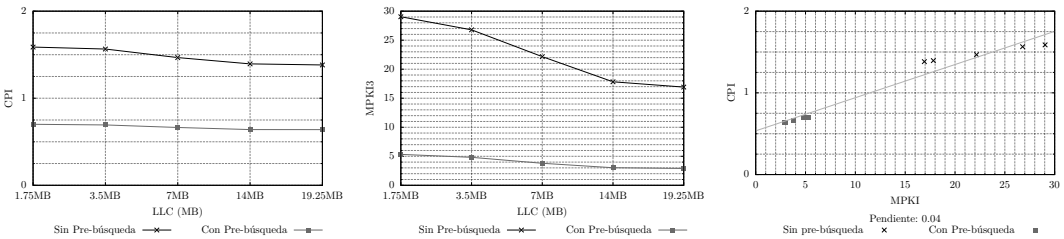
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.253: 510.parest_r.1



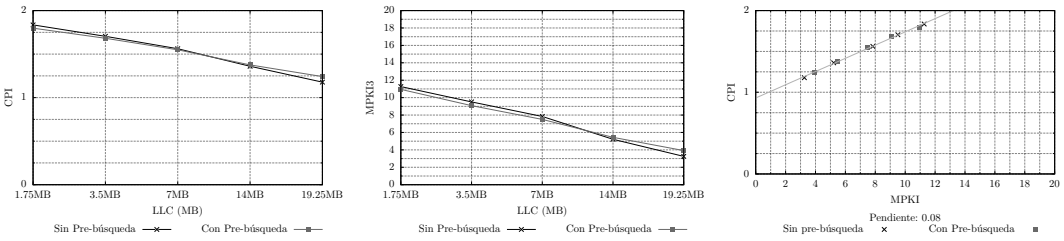
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.254: 511.povray_r.1



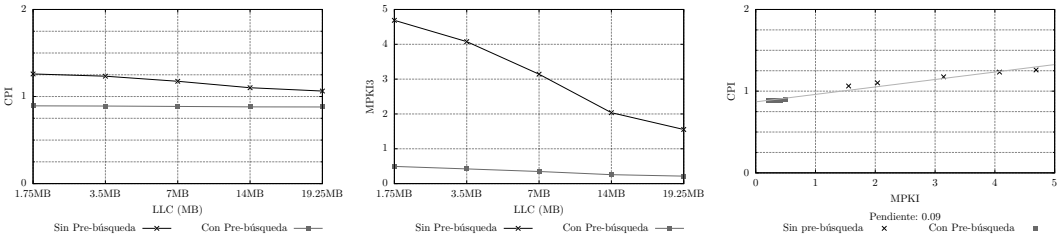
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.255: 519.lbm_r.1



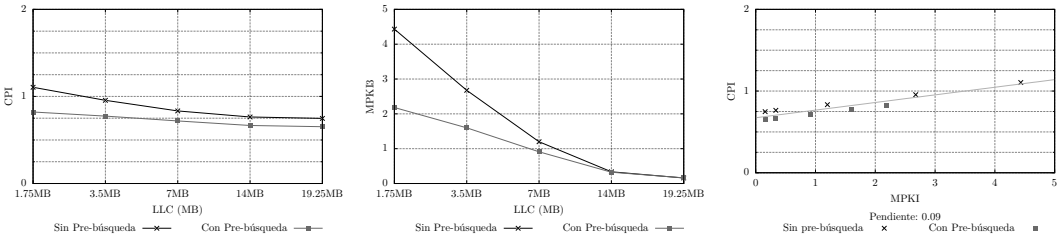
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.256: 520.omnetpp_r.1



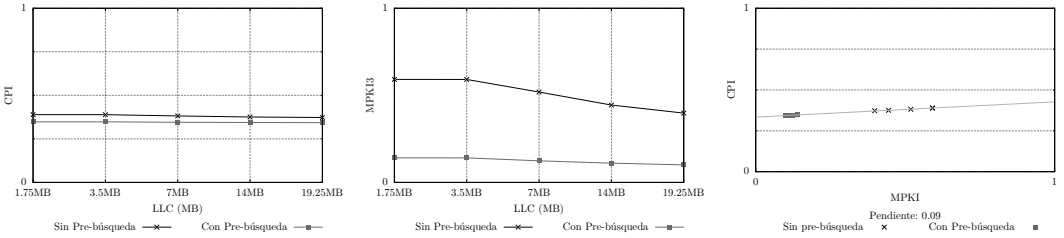
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.257: 521.wrf_r.1



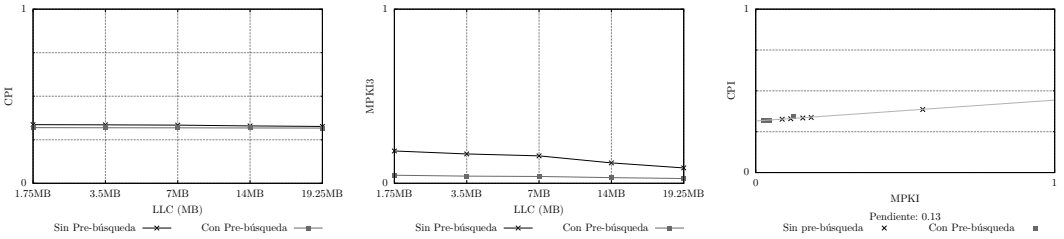
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.258: 523.xalancbmk_r.1



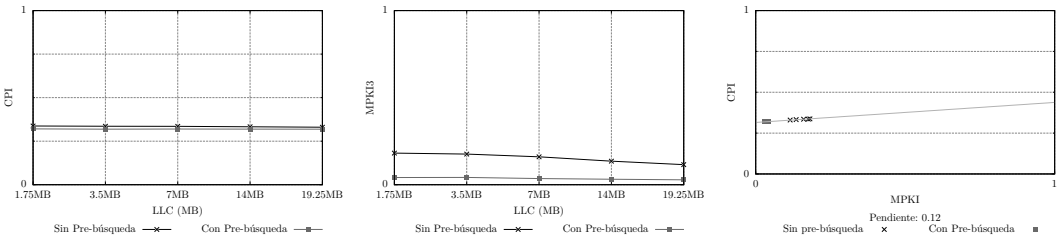
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.259: 525.x264_r.1



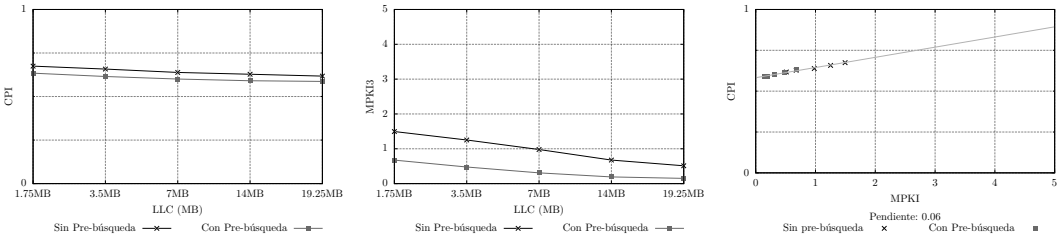
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.260: 525.x264_r.2



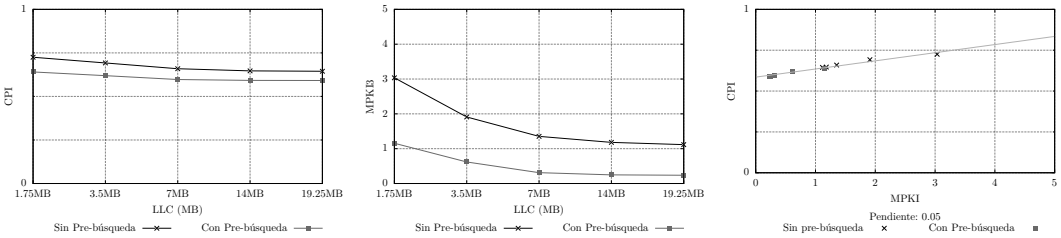
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.261: 525.x264_r.3



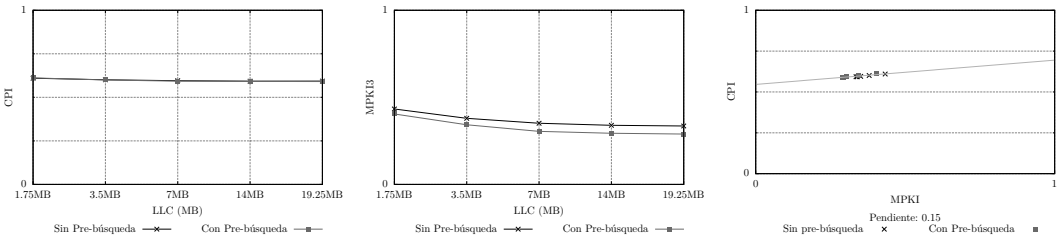
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.262: 526.blender_r.1



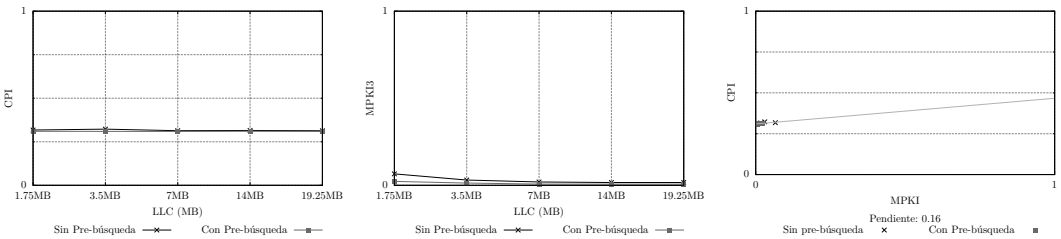
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.263: 527.cam4_r.1



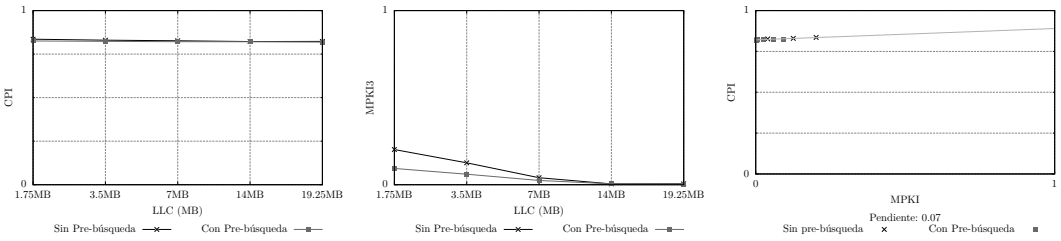
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.264: 531.deepsjeng_r.1



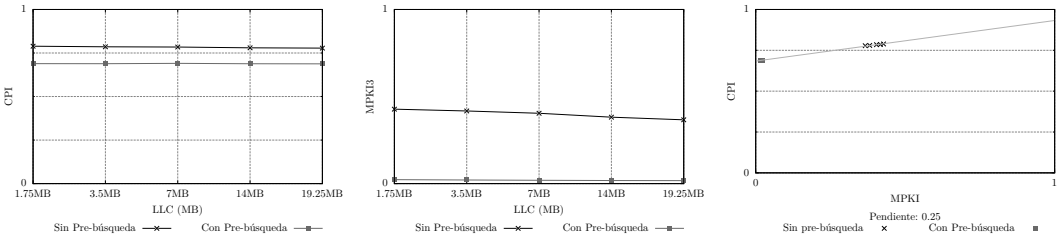
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.265: 538.imagick_r.1



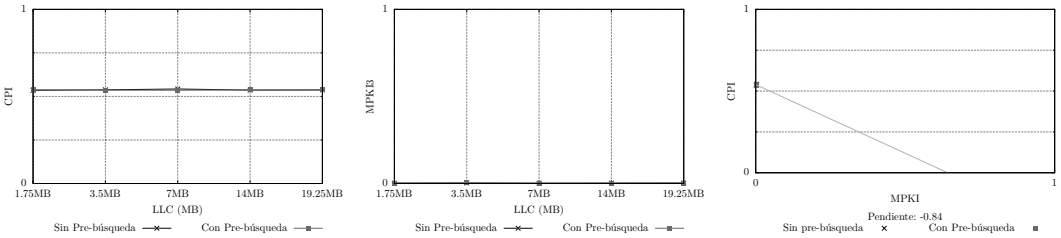
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.266: 541.leela_r.1



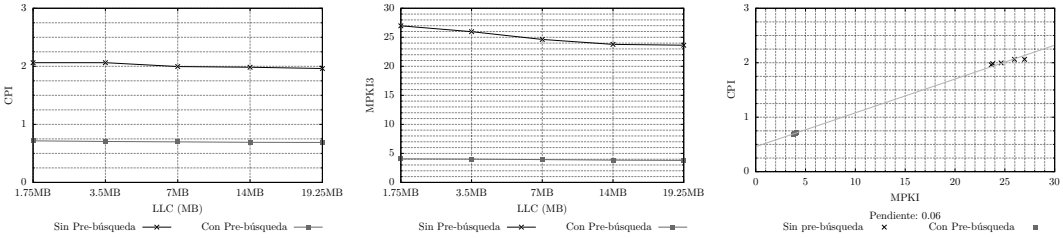
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.267: 544.nab_r.1



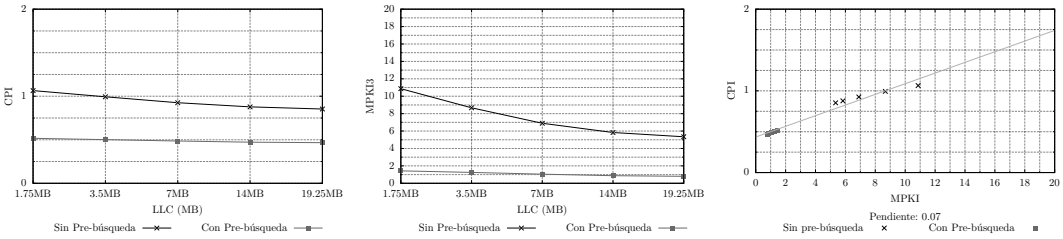
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.268: 548.exchange2_r.1



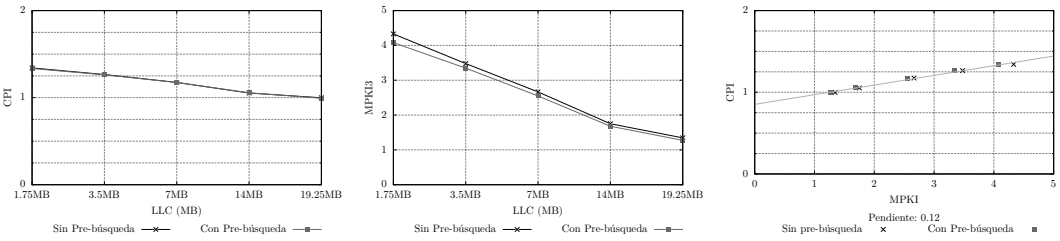
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.269: 549.fotonik3d_r.1



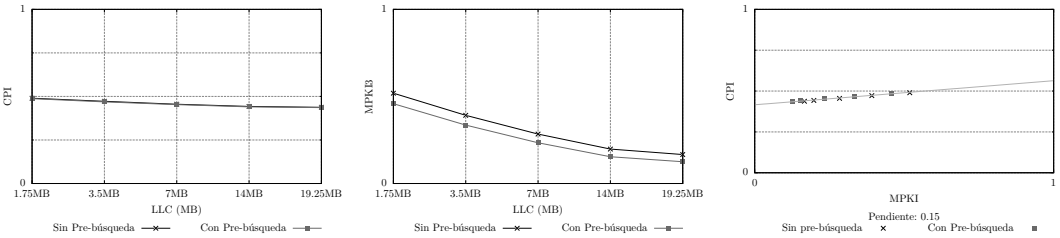
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.270: 554.roms_r.1



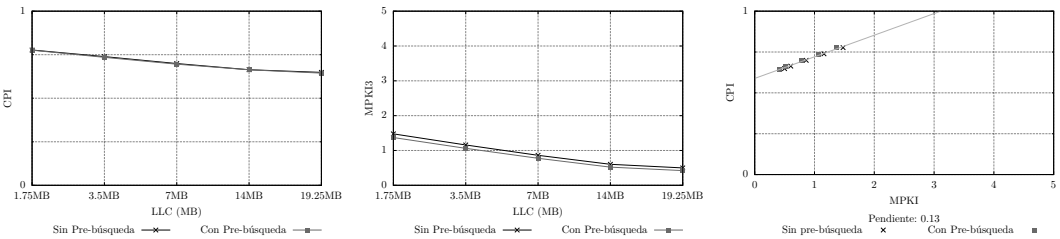
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.271: 557.xz_r.1



(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.272: 557.xz_r.2



(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.273: 557.xz_r.3

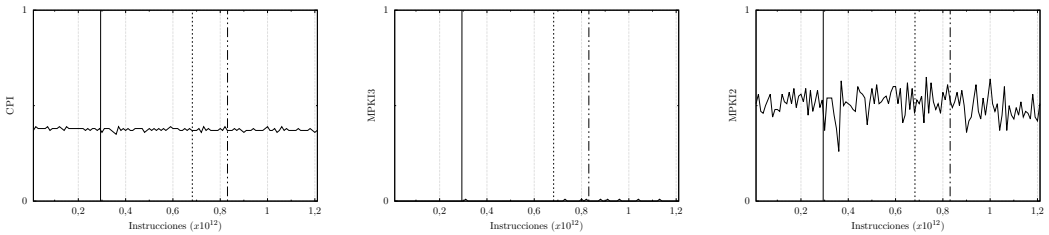
C.3. SPEC CPU2017 Speed Single-Thread

C.3.1. Evolución Temporal

Benchmark	Instr	Lect	Escr	MPKI1	MPKI2	MPKI3	CPI
600.perlbench_s.1	1218	28 %	19 %	7,26	0,50	0,01	0,38
600.perlbench_s.2	699	31 %	17 %	17,78	0,04	0,02	0,39
600.perlbench_s.3	667	31 %	18 %	6,69	1,53	0,11	0,44
602.gcc_s.1	1212	33 %	6 %	42,81	1,59	0,36	0,73
602.gcc_s.2	517	28 %	14 %	28,25	2,04	0,20	0,69
602.gcc_s.3	502	28 %	13 %	24,64	2,32	0,26	0,68
605.mcf_s.1	1653	34 %	10 %	74,58	30,47	7,37	1,41
620.omnetpp_s.1	1056	31 %	18 %	32,35	11,28	3,50	1,15
623.xalancbmk_s.1	1274	30 %	7 %	45,11	2,34	0,11	0,66
625.x264_s.1	516	25 %	8 %	3,98	0,11	0,05	0,34
625.x264_s.2	1963	24 %	6 %	4,34	0,04	0,02	0,32
625.x264_s.3	1985	24 %	6 %	3,87	0,04	0,02	0,32
631.deepsjeng_s.1	2179	25 %	12 %	3,50	0,27	0,25	0,58
641.leela_s.1	2111	26 %	9 %	4,47	0,10	0,00	0,82
648.exchange2_s.1	2908	51 %	30 %	0,10	0,00	0,00	0,54
AVG	1363	30 %	13 %	19,98	3,51	0,82	0,63

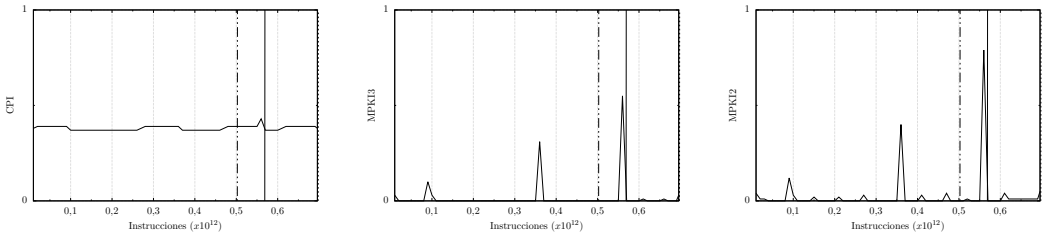
Cuadro C.3: Tabla resumen con los millones de instrucciones, porcentaje de instrucciones de lectura, porcentaje de instrucciones de escritura, MPKI en L1, MPKI en L2, MPKI en L3 y CPI de los programas de SPEC CPU2017Speed Single-thread

C.3.2. Caracterización General



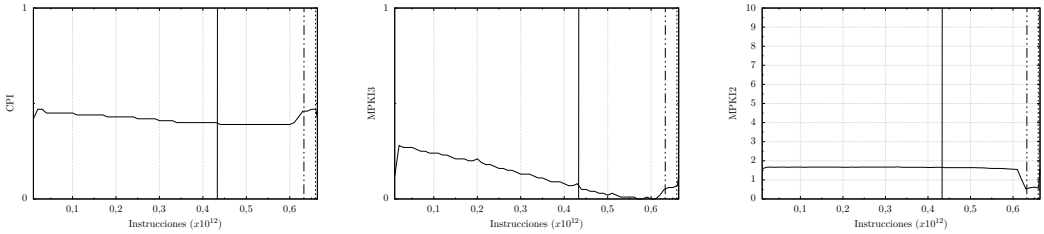
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.274: 600.perlbench_s.1



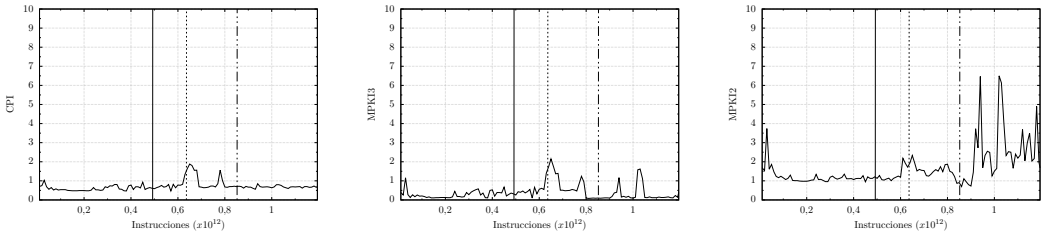
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.275: 600.perlbench_s.2



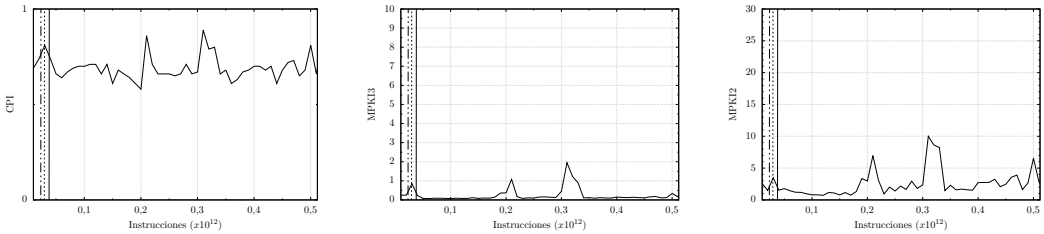
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.276: 600.perlbench_s.3



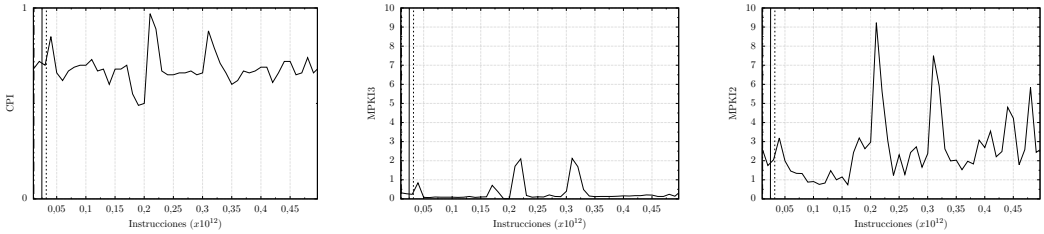
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.277: 602.gcc_s.1



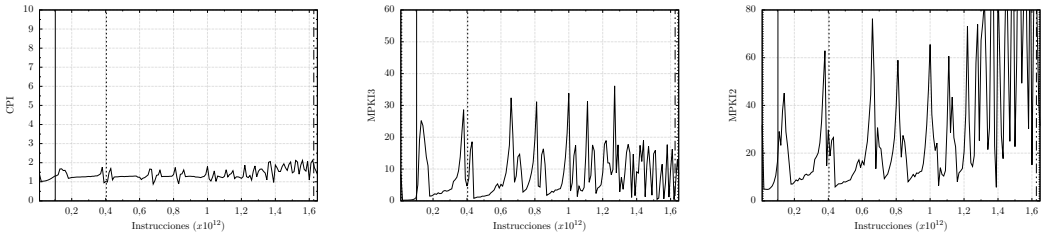
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.278: 602.gcc_s.2



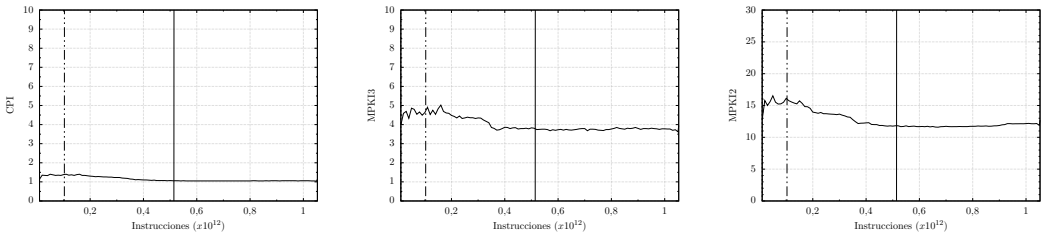
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.279: 602.gcc_s.3



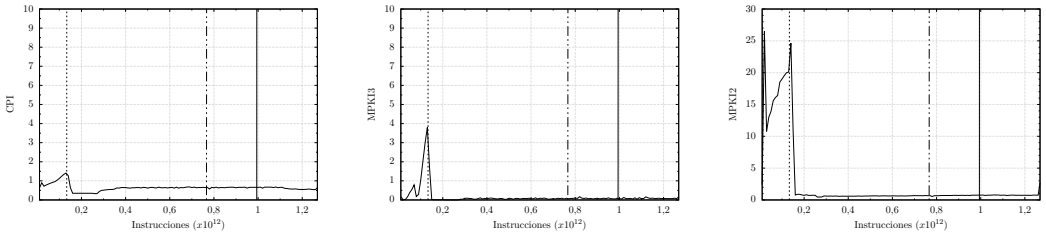
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.280: 605.mcf_s.1



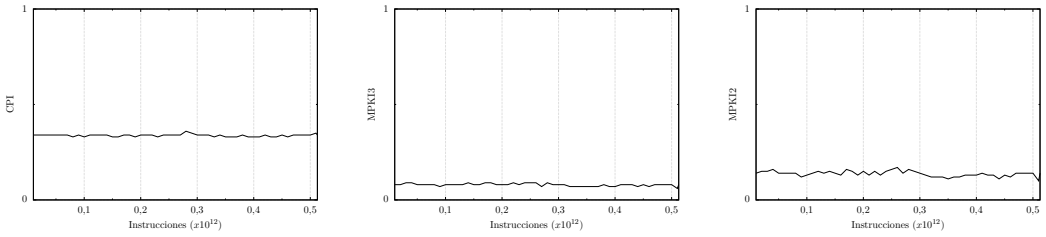
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.281: 620.omnetpp_s.1



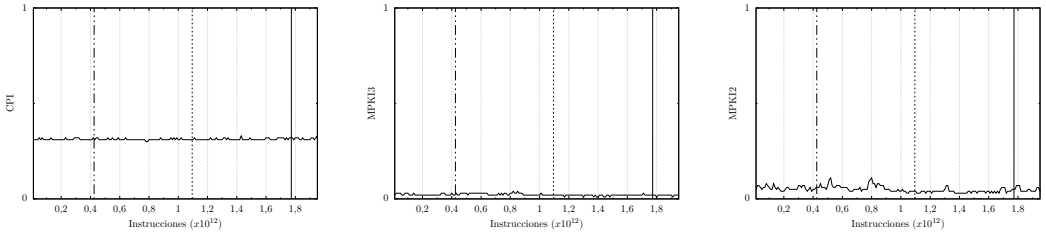
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.282: 623.xalancbmk_s.1



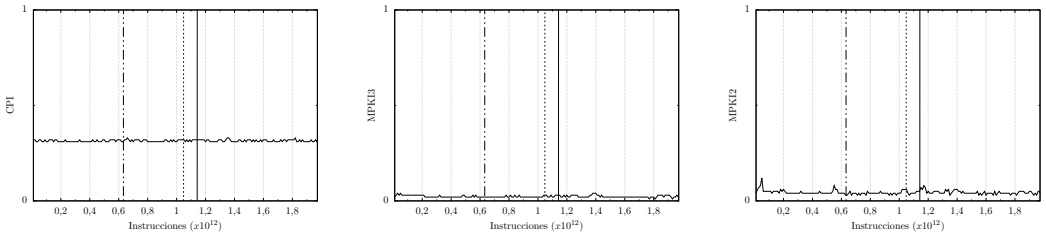
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.283: 625.x264_s.1



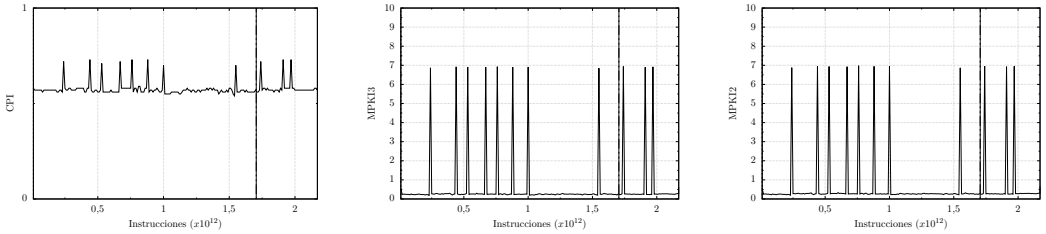
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.284: 625.x264_s.2



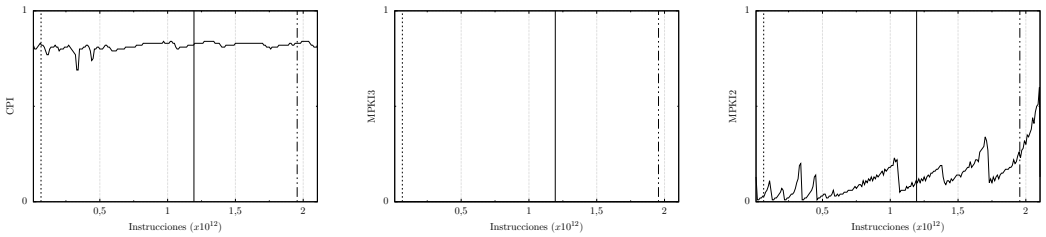
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.285: 625.x264_s.3



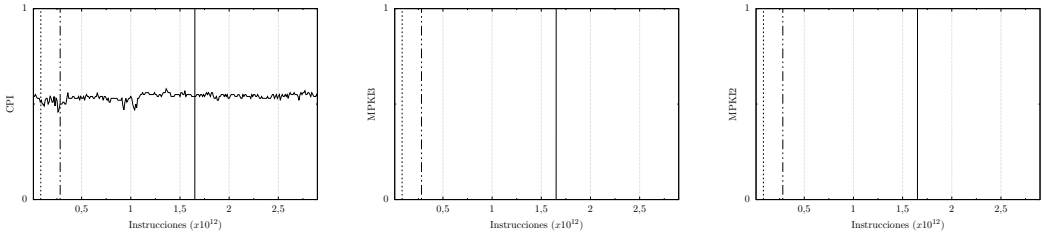
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.286: 631.deepsjeng_s.1



(a) CPI **(b)** MPKI **(c)** APKI

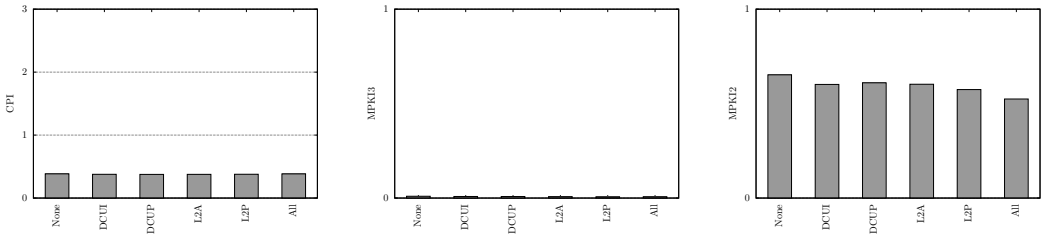
Figura C.287: 641.leela_s.1



(a) CPI **(b)** MPKI **(c)** APKI

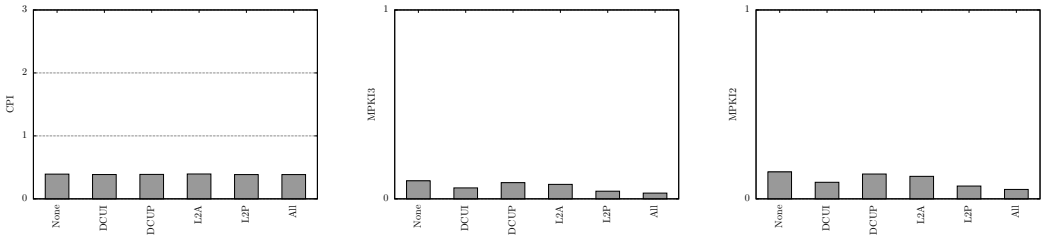
Figura C.288: 648.exchange2_s.1

C.3.3. Pre-búscadores Hardware



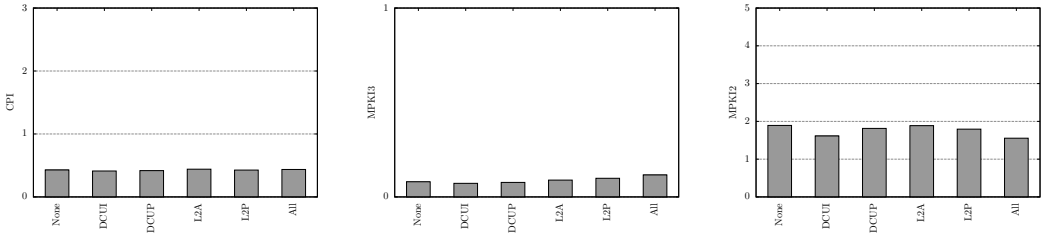
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.289: 600.perlbench_s.1



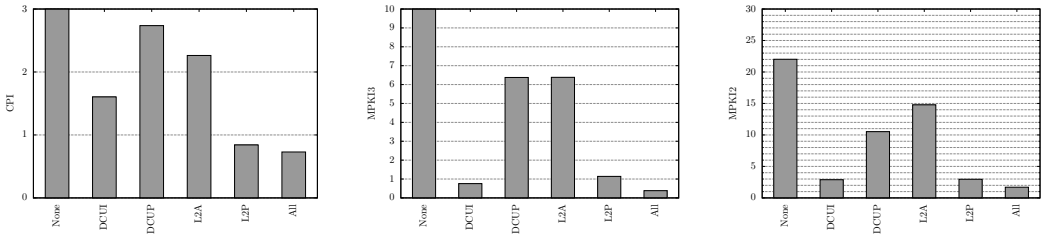
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.290: 600.perlbench_s.2



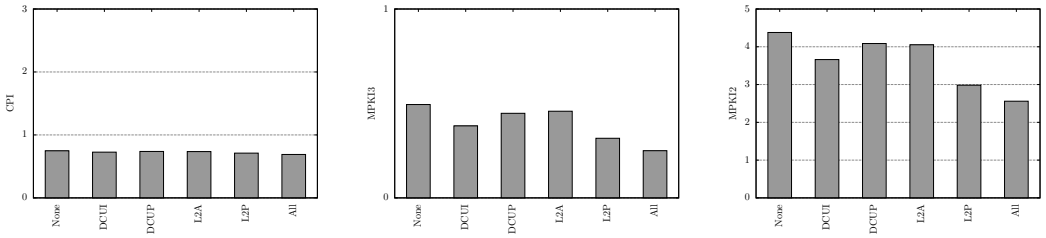
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.291: 600.perlbench_s.3



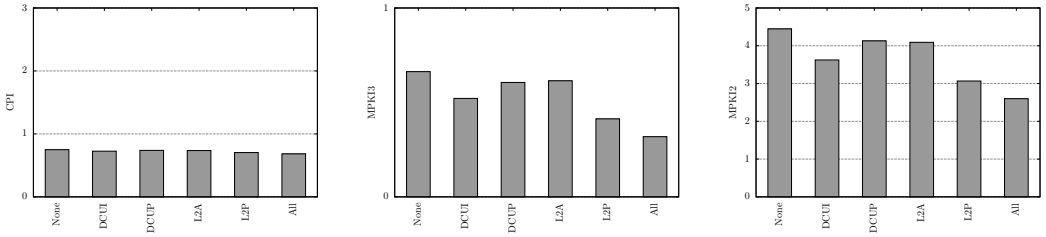
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.292: 602.gcc_s.1



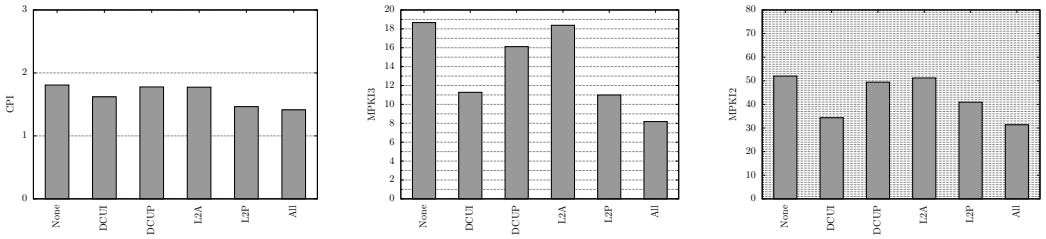
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.293: 602.gcc_s.2



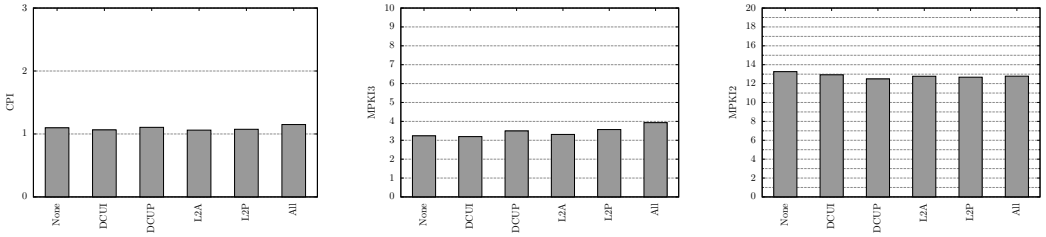
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.294: 602.gcc_s.3



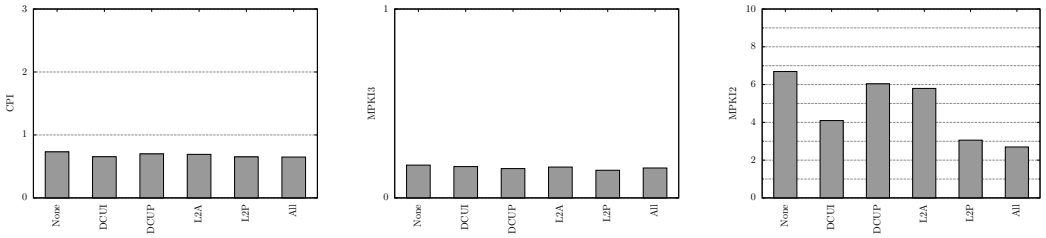
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.295: 605.mcf_s.1



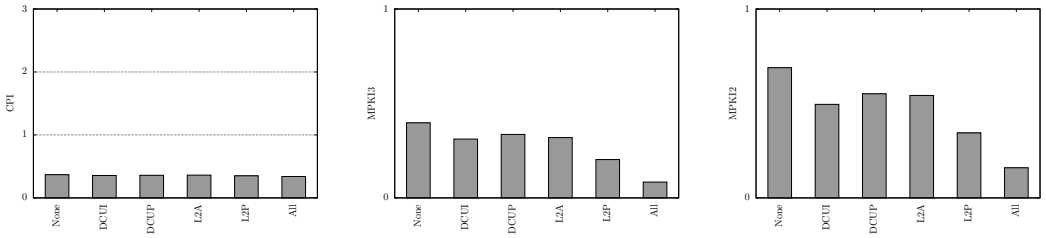
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.296: 620.omnetpp_s.1



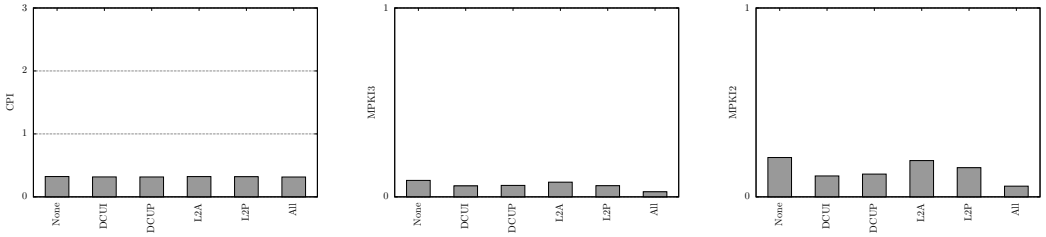
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.297: 623.xalancbmk_s.1



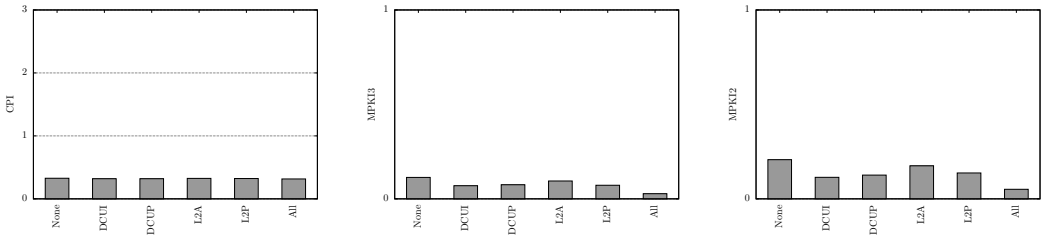
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.298: 625.x264_s.1



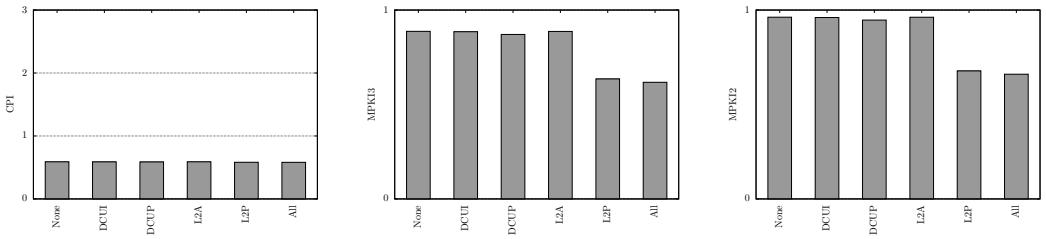
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.299: 625.x264_s.2



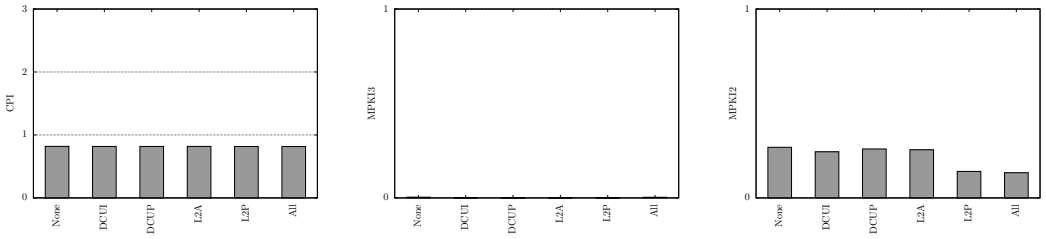
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.300: 625.x264_s.3



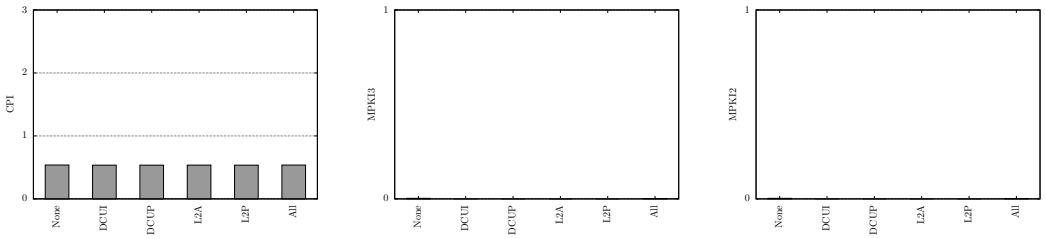
(a) CPI **(b)** MPKI **(c)** APKI

Figura C.301: 631.deepsjeng_s.1



(a) CPI **(b)** MPKI **(c)** APKI

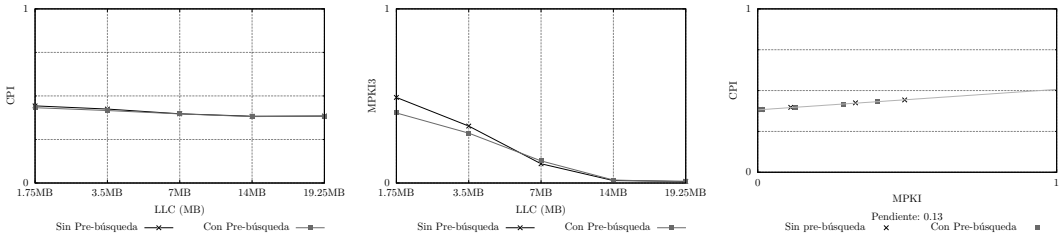
Figura C.302: 641.leela_s.1



(a) CPI **(b)** MPKI **(c)** APKI

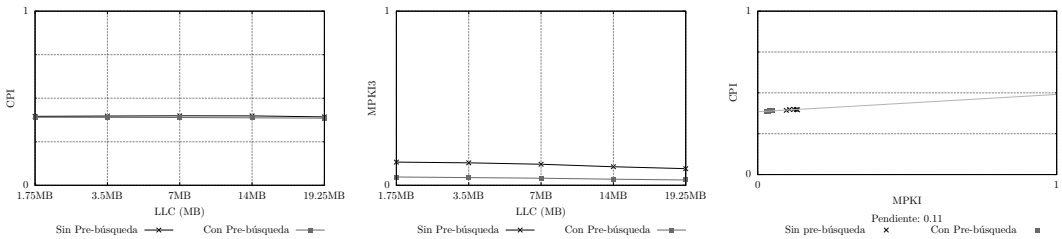
Figura C.303: 648.exchange2_s.1

C.3.4. Tamaño de LLC



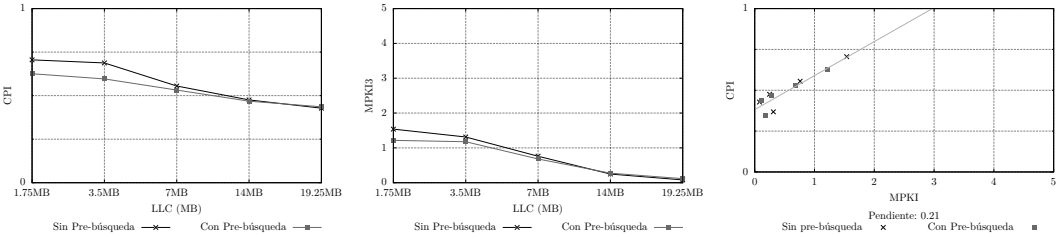
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.304: 600.perlbench_s.1



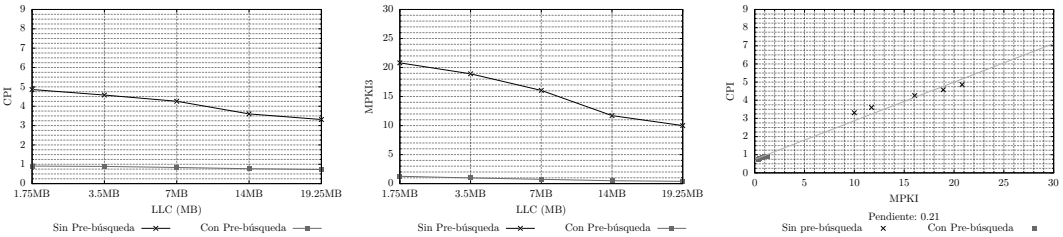
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.305: 600.perlbench_s.2



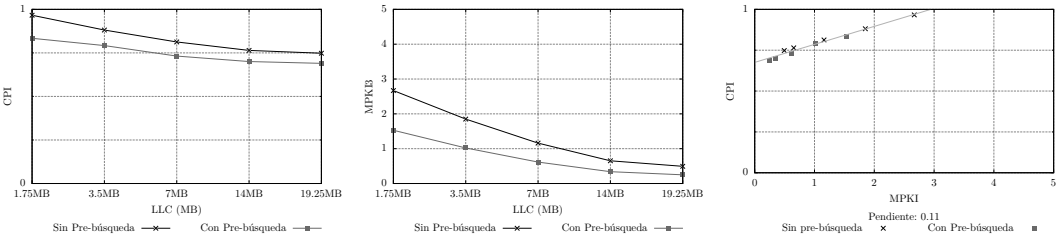
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.306: 600.perlbench_s.3



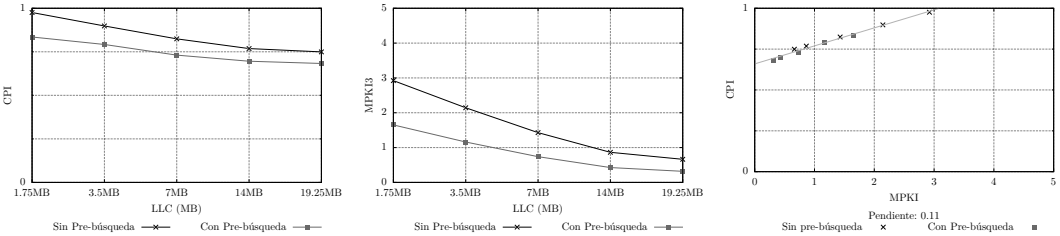
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.307: 602.gcc_s.1



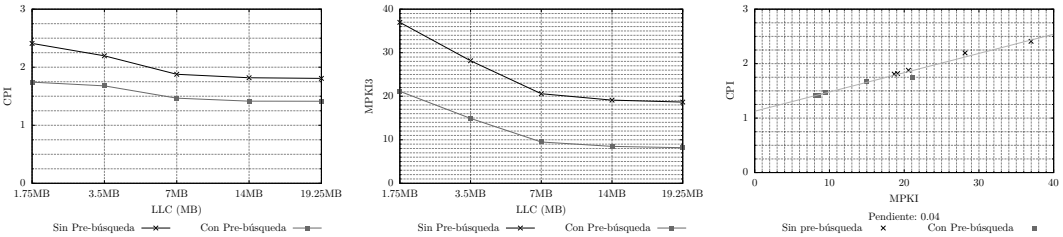
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.308: 602.gcc_s.2



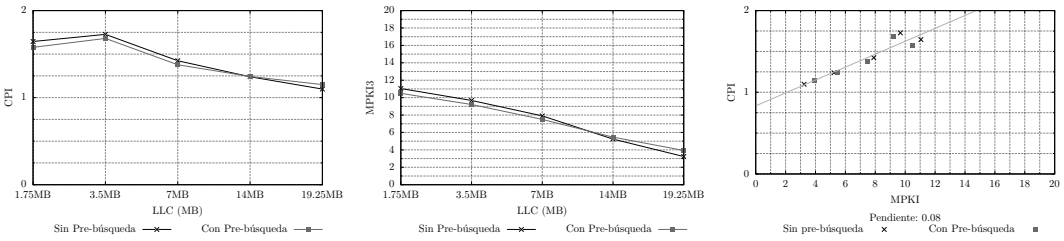
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.309: 602.gcc_s.3



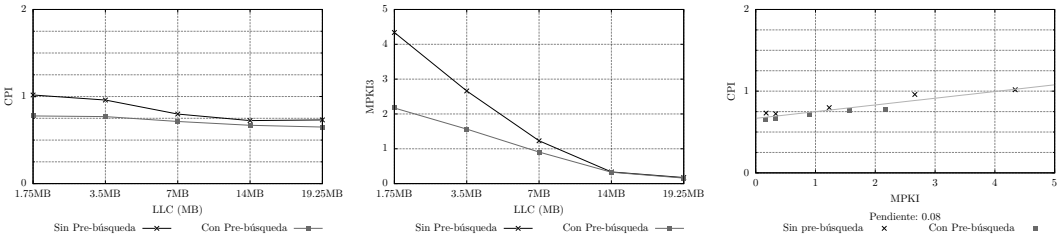
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.310: 605.mcf_s.1



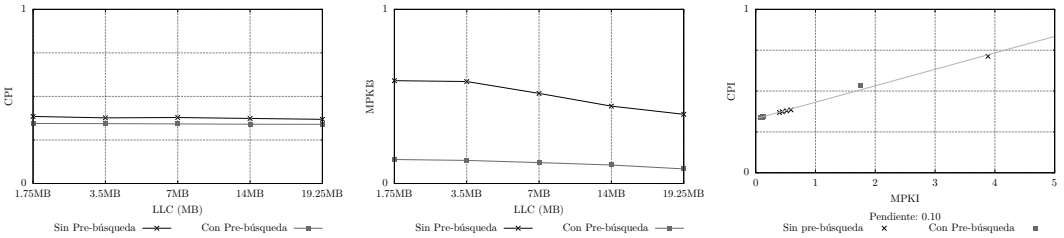
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.311: 620.omnetpp_s.1



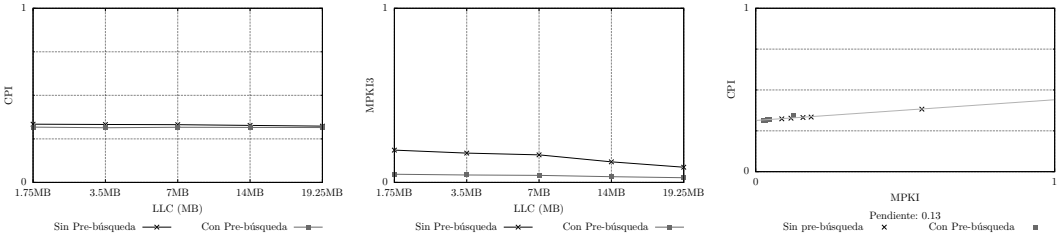
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.312: 623.xalancbmk_s.1



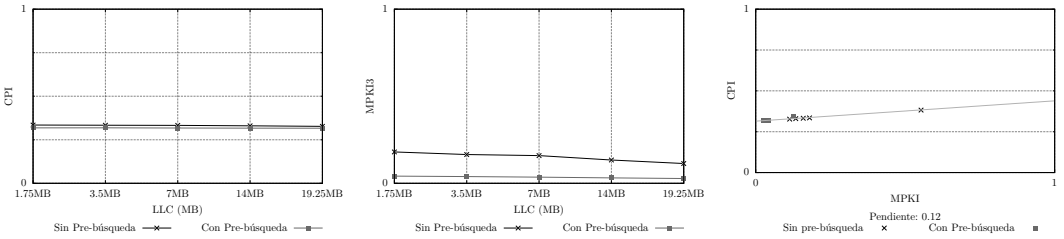
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.313: 625.x264_s.1



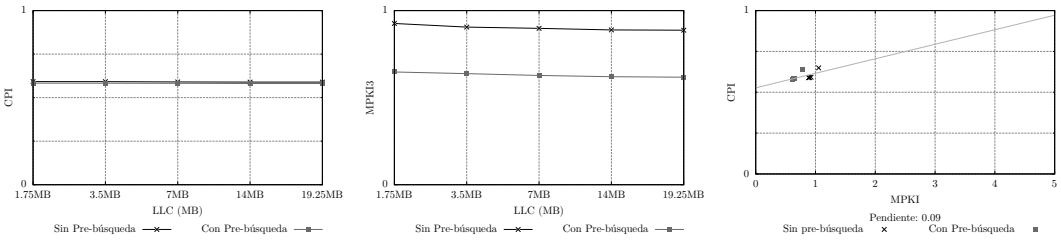
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.314: 625.x264_s.2



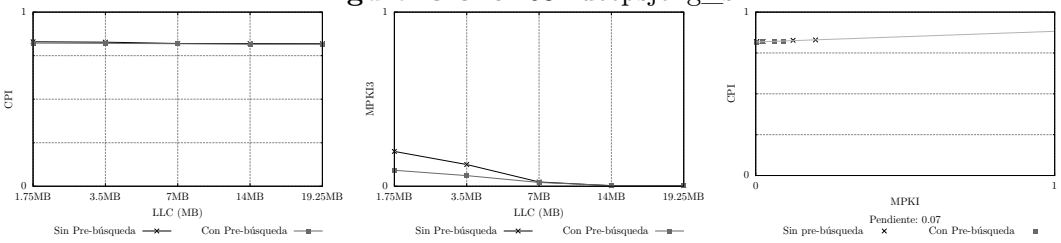
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.315: 625.x264_s.3



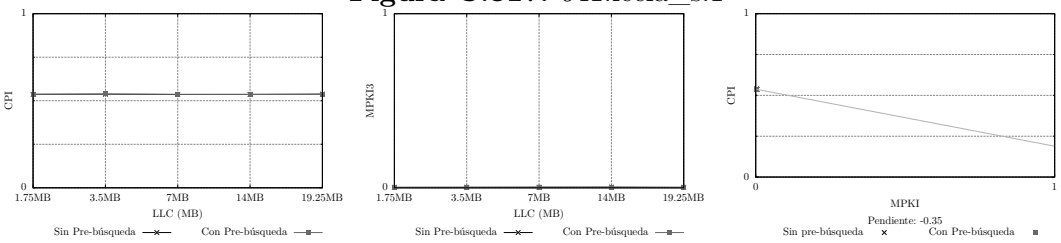
(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.316: 631.deepsjeng_s.1



(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.317: 641.leela_s.1



(a) CPI **(b)** MPKI **(c)** Scatter

Figura C.318: 648.exchange2_s.1

Apéndice D

Articulos

En el siguiente anexo se muestran un poster y un articulo científico que ha dado como resultado la realización de este trabajo de fin de máster. El primero fue presentado en la escuela de verano ACACES 2018 y el segundo en las Jornadas Sarteco 2018.

Memory Hierarchy Performance Characterization of SPEC CPU2017

Agustín Navarro-Torres[*,1],
Pablo Ibáñez-Marín[*,1],
Jesús Alastruey-Benedé[*,1],
Víctor Viñals-Yúfera[*,1]

[*] *Universidad de Zaragoza, I3A, Spain*

ABSTRACT

SPEC CPU is one of the most common suite of benchmarks used in computer architecture research. On June 2017, a new version was released to replace the old version, CPU2006, which remained state-of-the-art for 11 years. We present a detailed memory hierarchy performance analysis of all single-thread benchmarks that made up the new suite.

KEYWORDS: SPEC CPU2017; characterization; memory hierarchy; performance analysis; benchmarks; hardware counters

1 Introduction

SPEC CPU is one of the most widely used benchmark suites for high performance computing research on academia and industry. The new version, CPU2017 [1], released on June 2017 is called to replace the 2006 version. Thus, new characterization is necessary in order to help researchers to select the benchmarks with particular characteristics or pick simulation points.

In this work we analyze the memory hierarchy performance of SPEC CPU2017. Namely, we study all the single-thread benchmarks, identify the memory-intensive ones, and analyze the sensitivity of them to the last-level cache size and the different hardware prefetchers.

2 Experimental methodology

We execute all the SPEC CPU2017 single-thread benchmarks in an Intel Xeon Skylake-SP Gold 5120 (Xeon-SP in short). Several hardware performance counters were collected with the perf profiler[4] to measure the following metrics: cycles per instruction (CPI), misses per kilo instruction (MPKI) in all cache levels and bytes read from main memory per kilo instruction (BPKI).

[1]E-mail: {agusnt, imarin, jalastru, victor}@unizar.es

Our Xeon-SP has 14 cores, two levels of private cache (32 KB for instructions and 32 KB for data in the first level, and 1 MB in the second level) and a shared last-level cache (LLC, 19.25 MB, 11-way set-associative). The processor disposes of four hardware prefetchers [3]: *L1 Data cache unit prefetcher* (DCUI), *L1 Data cache instruction pointer stride prefetcher* (DCUP), *L2 Data cache spatial prefetcher* (L2A), and *L2 Data cache streamer* (L2P).

We use the *Intel Cache Allocation Technology* (CAT) feature to modify the LLC capacity that a program has available during its execution. Hardware prefetchers are selectively enabled or disabled by writing their corresponding *Model Specific Registers* (MSRs). This way we can evaluate the application performance with multiples LLC configurations on a real system.

3 Evaluation

3.1 Identification of Memory Intensive Benchmarks

We executed all CPU2017 single-thread applications with all the inputs provided by SPEC in a limited resources context: all hardware prefetchers disabled and the smallest LLC size that can be assigned to an application (1.75 MB). For each application/input pair we measured misses per kilo instructions for the three cache levels (MPKI1, MPKI2 and MPKI3). These metrics are shown in Figure 1. We plot in red the bars associated to applications that have very low MPKI2 and MPKI3 ratio, even in a limited resources context. These application/input pairs have little interest for memory hierarchy studies. Among the remaining application/input pairs, we have selected an input for each application, plotted in green, that will be used in the following experiments.

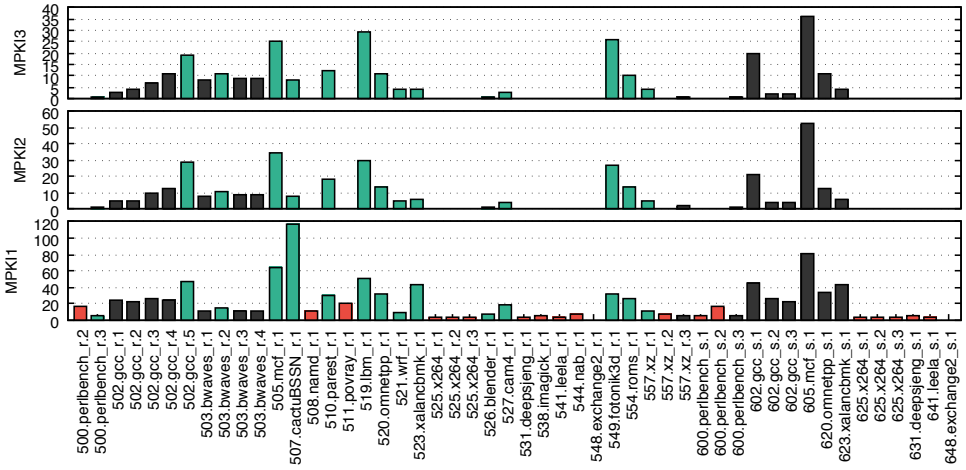


Figure 1: MPKI1, MPKI2 and MPKI3 for all SPEC CPU2017 single-thread benchmarks.

3.2 Sensitivity to the LLC Size and to Hardware Prefetching

In this experiment we study the sensitivity of the memory-intensive benchmarks to the LLC size and to the hardware prefetcher. The 15 workloads selected in the previous subsection were executed with different LLC sizes. All these runs were performed in two different ways: one with all the prefetchers enabled and the other with all disabled.

Five LLC sizes, 19.25 MB, 14 MB, 7 MB, 3.5 MB and 1.75 MB, were configured by limiting the number of ways available for the program to 11, 8, 4, 2 and 1, respectively. This resource allocation was performed by CAT. Figure 2 shows the MPKI3 for the selected LLC sizes and benchmarks.

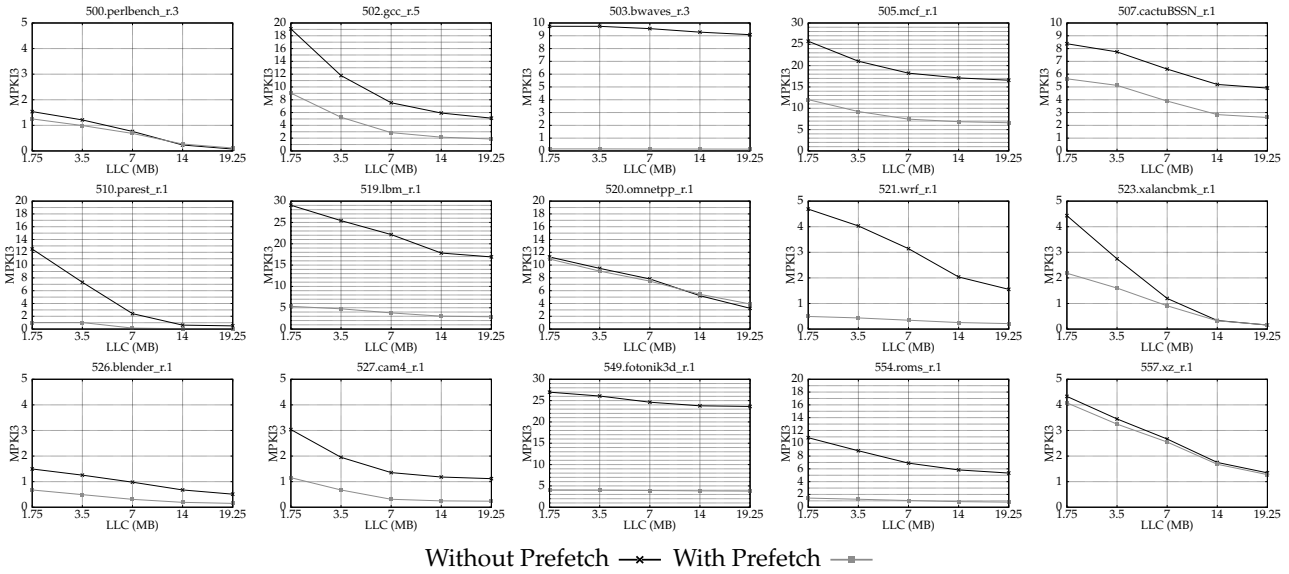


Figure 2: LLC cache misses (MPKI3) of the selected application for different LLC sizes.

Without prefetching, an increase in the LLC size translates to a significant MPKI3 reduction in all applications except `503.bwaves`. This improvement decreases considerably when prefetching is enabled for several applications: `510.parest`, `519.lbm`, `521.wrf`, `549.fotonik3d` and `554.roms`.

Prefetching is very effective in terms of MPKI3 reduction for 12 benchmarks: it reduces MPKI3 in all cache sizes, specially in the smaller ones. For two benhmarks (`500.perlbench` and `557.xz`), it gets minor improvements only for small LLC sizes. And finally, for one application (`520.omnetpp`), prefetching does not reduce LLC cache misses in any case, and even slightly increases the MPKI3 with the largest LLC.

3.3 Performance of the Hardware Prefetchers

In this section, we analyze the impact of the different hardware prefetchers on the applications performance and on the applications memory bandwidth consumption. All the selected workloads were executed with different configurations: with all prefetchers enabled, with all of them disabled, and with each one of the prefetchers enabled individually (one at a time). The experiment was accomplished with the maximum LLC size. Figure 3 shows the performance in terms of cycles per instruction (CPI, left axis), and memory bandwidth in terms of bytes read from main memory (BPKI, right axis) for the selected benchmarks.

In general, L2P is the best prefetcher. For the 12 prefetch-friendly applications, L2P alone achieves more than 82% of the CPI reduction, and more than 90% for 7 of them. The second best prefetcher is DCUI followed by DCUP. L2A gets the least improvement. It only reduces CPI more than 5% for 6 applications, with a maximum of 15% for `554.roms`.

Hardware prefetching is bandwidth-efficient since it causes significant extra bandwidth consumption in only three benchmarks (`520.omnetpp`, `549.fotonik3d` and `554.roms`).

Considering that LLC MPKI is improved for two of these three benchmarks (except `520.omnetpp`), the overall result is positive.

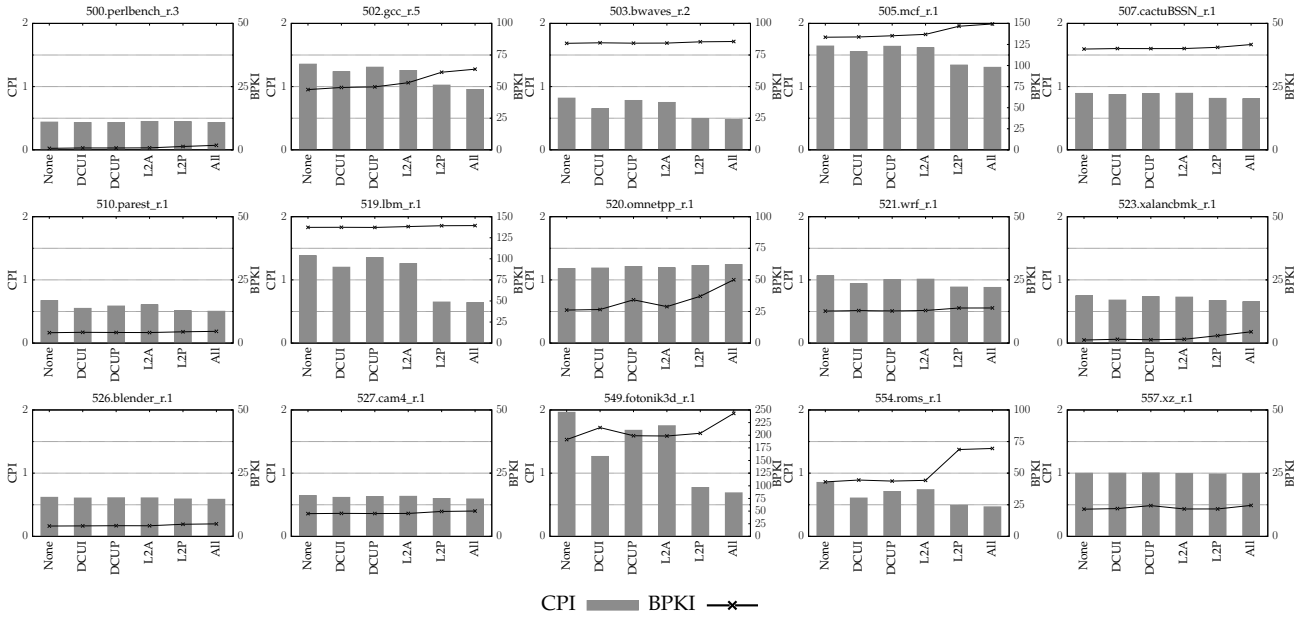


Figure 3: Impact of the hardware prefetchers on the application performance (CPI) and on the memory bandwidth (BPKI).

4 Conclusions

In this work we analyzed the memory hierarchy performance of the SPEC CPU2017 single-thread applications. We identified 19 out of 50 application/input pairs, from 9 applications, with little use of the memory hierarchy. They have very low miss ratios in the second and third level caches, even with small LLC sizes and without hardware prefetching.

We analyzed the sensitivity to the LLC size and to the different hardware prefetchers of the 15 memory-intensive applications. Prefetching is very effective in 12 of them, and the L2P prefetcher is responsible of a large fraction of the improvement. 9 applications show important MPKI reduction when increasing the LLC size even with prefetching.

References

[1] SPEC, SPEC CPU® 2017, `https://www.spec.org/cpu2017/`

[2] Nguyen, Khang T, *Introduction to Cache Allocation Technology in the Intel® Xeon® Processor E5 v4 Family*, Intel software developer zone 2016, `https://software.intel.com/en-us/articles/introduction-to-cache-allocation-technology`

[3] Intel, *Intel® 64 and IA-32 Architectures Optimization Reference Manual*. Intel 2016, update April, 2018.

[4] Arnaldo Carvalho de Melo. The new linux 'perf' tools. 2010.

Memory Hierarchy Performance Characterization of SPEC CPU2017

Agustín Navarro-Torres, Jesús Alastruey-Benedé, Pablo Ibáñez-Marín, Víctor Viñals-Yúfera
{agusnt, jalastru, imarin, victor}@unizar.es

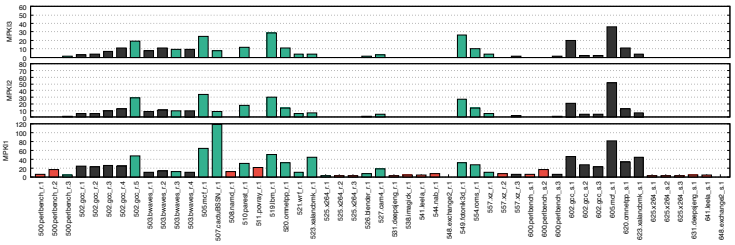
1. Introduction

SPEC CPU is one of the most widely used benchmark suites for high performance computing research on academia and industry. The last version, CPU2017 [1], was released on June 2017.

In this work we analyze the memory hierarchy performance of SPEC CPU2017. Namely, we study all the single-thread benchmarks, identify the memory intensive ones, and analyze their sensitivity to the last-level cache (LLC) size and to the different hardware prefetchers.

The characterization was performed on an Intel Xeon Skylake-SP Gold 5120. Several hardware performance counters were collected with the Perf profiler. We used the Intel Cache Allocation Technology (CAT) [2] to modify the available LLC capacity.

2. Identification of Memory Intensive Benchmarks

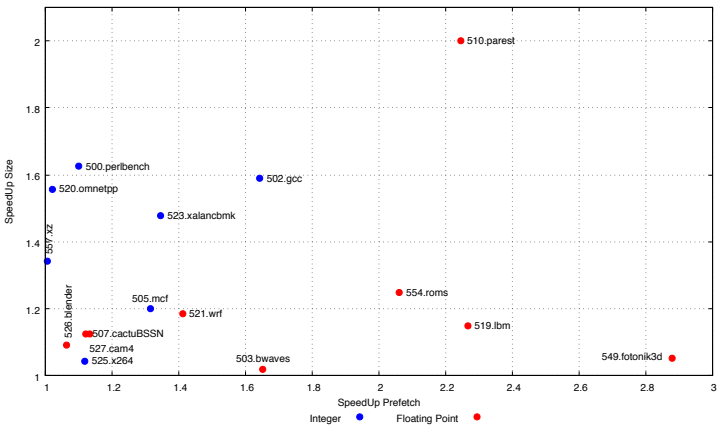


Misses per kilo-instruction (MPKI) for the three cache levels. All CPU2017 single-thread benchmark-input pairs were executed without hardware prefetching and 1.75MB of LLC. Benchmarks that have very low MPKI2 and MPKI3 ratios are plotted in red.

Only 16 out of 23 benchmarks are memory intensive

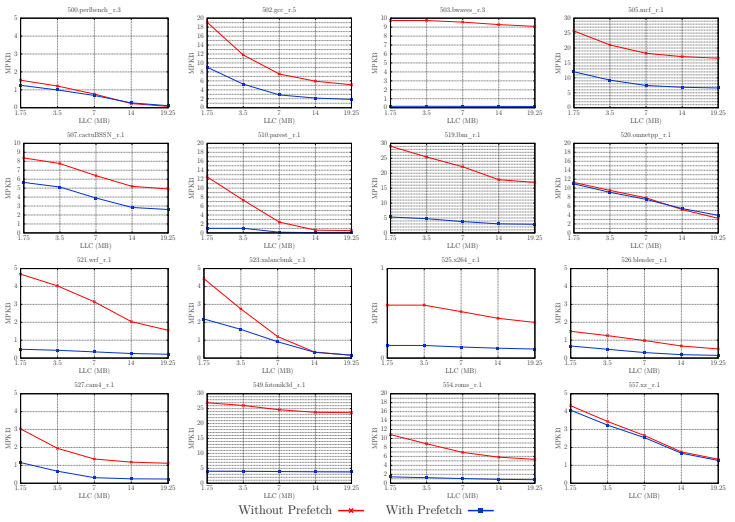
For these benchmarks only one input will be considered in the following experiments (green bars).

3. Performance Impact of Prefetch/LLC Size



Performance impact of hardware prefetching (x axis) and LLC size (y axis). Speed-ups with respect to the smallest LLC without hardware prefetching.

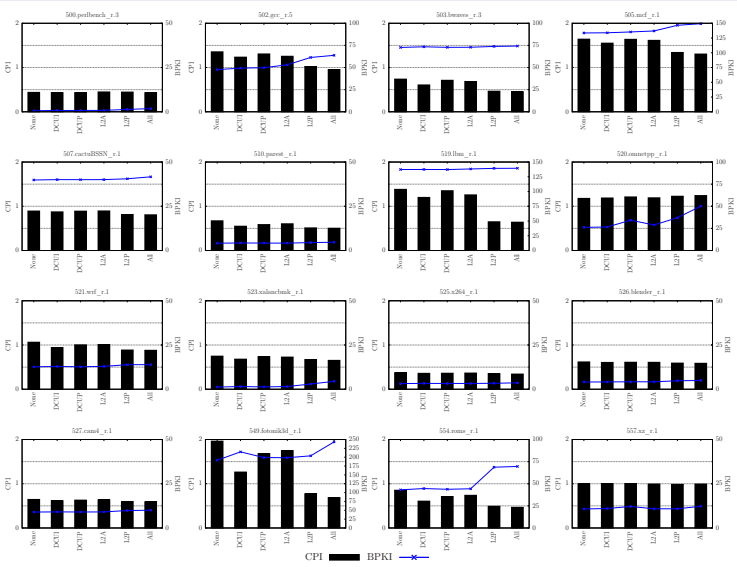
4. Sensitivity to Prefetch and to LLC Size



Sensitivity of the memory intensive benchmarks to the hardware prefetching and to the LLC size.

Hardware prefetching is very effective in reducing MPKI3 even with the smallest LLC size
With hardware prefetching, increasing LLC size translates to significant MPKI3 reductions for 9 out of 16 benchmarks

5. Hardware Prefetchers Performance



Cycles per instruction (CPI) and bytes read from main memory per kilo-instruction (BPKI) for different prefetching configurations.

Hardware prefetching is very efficient because it achieves noticeable CPI reductions with low bandwidth overhead

6. Conclusions

- Several benchmark-input pairs show very low MPKI2/3 ratios even with 1.75MB of LLC and no hardware prefetching.
- Hardware Prefetching is very effective in 13 out of 16 benchmarks.
- Increasing LLC size with hardware prefetching reduces MPKI3 in 9 out of 16 benchmarks.
- Hardware prefetching is bandwidth-efficient.
- L2P prefetcher achieves most of the CPI reduction.

7. References

1. SPEC CPU2017, Standard Performance Evaluation Corporation, *https://www.spec.org/cpu2017/*
2. Introduction to Cache Allocation Technology in the Intel® Xeon® Processor E5 v4 Family, Intel, *https://goo.gl/oZaVQ7*

This work was supported in part by grants TIN2016-76635-C2-1-R (AEI/ERDF, EU) and gaZ: T58_17R research group (Aragón Gov. and European ESF).

Caracterización del rendimiento de la jerarquía de memoria para SPEC CPU2006 y CPU2017

Agustín Navarro Torres, [1] Jesús Alastruey Benedé, [1]
Pablo Ibáñez Marín, [1] Víctor Viñals Yúfera [1]

Resumen—**SPEC CPU es una de las suites de benchmarks más utilizadas en la investigación de arquitectura de computadores. La nueva versión CPU2017, lanzada en junio de 2017, está llamada a reemplazar a la antigua versión, CPU2006, que ha sido referente durante más de 11 años. En este trabajo presentamos un análisis detallado del rendimiento de la jerarquía de memoria de un procesador Intel Xeon SP ejecutando los programas de SPEC CPU2006 y los mono-hilo de CPU2017.**

Palabras clave— **SPEC CPU2017; CPU2006; Caracterización; Jerarquía de memoria; Prebúsqueda hardware, Contadores Hardware; Perf; Intel Xeon; Skylake-SP;**

I. Introducción

La investigación experimental en arquitectura de computadores se basa en alimentar a una máquina real o a un modelo de simulación con una carga de trabajo (*workload*). La posibilidad más extendida es usar una carga de trabajo consistente en un conjunto (*suite*) de programas de prueba (*benchmarks*) seleccionados para ser representativos del software contemporáneo o futuro a la fecha de selección. Puesto que muchas ideas o mecanismos de futuro se van a evaluar con los programas de prueba disponibles, diversas agrupaciones de universidades y empresas (p.e. SPEC [1]), grupos de investigación (p.e. CloudSuite [2]), comunidades (p.e. TACLeBench [3]) e incluso empresas individuales (p.e. EEMBC [4]) proponen dichos programas.

La caracterización de estos programas es una de las primeras tareas a realizar en la comunidad de arquitectura de computadores. Entre los objetivos de los trabajos de caracterización podemos destacar la selección de muestras para simulación y la clasificación de programas según determinadas características.

La implementación de una nueva idea sobre un sistema real es inviable en la mayoría de los casos debido a su elevado coste o a la imposibilidad de modificación. Como alternativa, las nuevas ideas se implementan sobre simuladores que modelan en detalle sistemas tan complejos como un chip multiprocesador formado por varios núcleos, una jerarquía de memoria con varios niveles y una red de interconexión. La ejecución completa de programas reales sobre estos simuladores requeriría meses o incluso años. Por tanto, se utilizan técnicas de muestreo para seleccionar pequeños fragmentos de los programas que sean representativos de la ejecución completa.

En muchas publicaciones se utilizan cargas de trabajo con características conocidas a priori para analizar el comportamiento de nuevas propuestas frente a dichas cargas. Por ejemplo, en trabajos sobre algoritmos de remplazo para caches compartidas entre varios núcleos es frecuente utilizar mezclas de programas con distinto grado de presión sobre la jerarquía: unas formadas con programas que demandan mucho espacio en cache, otras con programas que demandan poco, otras mixtas, etc. La caracterización es necesaria para clasificar los programas según su comportamiento.

Por otra parte, la ejecución de estos programas en sistemas reales y el análisis de su comportamiento permiten detectar oportunidades de mejora, que pueden ser la base de nuevas propuestas hardware o software.

En este trabajo caracterizamos la interacción de los conjuntos SPEC CPU2006 y CPU2017 con la jerarquía de memoria del Intel Xeon SP. La caracterización de SPEC CPU2017 tiene especial interés porque es un conjunto muy reciente y sólo conocemos un trabajo al respecto [5]. En cuanto al procesador usado, también aporta relevancia a este estudio puesto que la familia Intel Xeon SP se lanzó en julio de 2017 e incorpora un cambio significativo en la jerarquía de memoria dentro del chip, ya que la cache compartida de tercer y último nivel (LLC) no guarda inclusión con los niveles inferiores, a diferencia de todos los procesadores Intel previos. Tampoco se han publicado estudios de comportamiento de esta nueva jerarquía.

Los objetivos concretos del trabajo son los siguientes:

- Caracterización de la sensibilidad de cada uno de los programas respecto al tamaño de LLC disponible. Para ello se obtendrán las tasas de fallos de cada programa con distintos tamaños de LLC.
- Caracterización del comportamiento de los prebuscadores hardware disponibles en el procesador en términos de influencia sobre la tasa de fallos en LLC, cantidad de información leída de memoria principal y tiempo de ejecución.
- Caracterización de la evolución temporal de los programas con el objetivo de identificar fragmentos relevantes para simulación. Se analizará la evolución, a lo largo de la ejecución completa de cada programa, de métricas relacionadas con la jerarquía de memoria: fallos en cache de segundo/tercer nivel por cada mil instrucciones (MPKI2/MPKI3) y número de ciclos por instrucción (CPI).

[1]Univ. de Zaragoza, email: {agusnt, jalastru, imarin, victor}@unizar.es

Para la obtención de las métricas se usarán los contadores hardware proporcionados por la arquitectura Intel [6], a través de herramientas como *Perf* [7]. El comportamiento de los distintos prebuscadores se estudia activándolos de forma selectiva mediante el uso del registro de control correspondiente. La obtención de datos para distintos tamaños de LLC se ha realizado en trabajos previos mediante simulación, lo que imponía el uso de muestreo e impedía la ejecución completa. En este trabajo usamos el conjunto de herramientas *Intel Resource Director* [8], que permite entre otras cosas limitar el número de vías de LLC dedicadas a un programa. De esta forma somos capaces de combinar ejecución completa y variación de tamaño.

Este artículo está organizado del siguiente modo: en la sección II se introduce el estado del arte en metodología de caracterización y selección de puntos de simulación; en la sección III se explica el entorno de ejecución, las cargas de trabajo y las métricas usadas en el estudio; en la sección IV se muestran los resultados de los distintos experimentos llevados a cabo y finalmente en la sección V se presentan conclusiones y líneas abiertas.

II. Estado del arte

La caracterización de nuevos conjuntos de programas es una actividad de investigación recurrente en la comunidad de arquitectura de computadores. En esta sección se presenta el estado del arte en dos ámbitos: las metodologías más utilizadas para la caracterización y la selección de puntos de simulación.

A. Metodología de caracterización

Los trabajos de caracterización de programas de prueba se han realizado utilizando complejos simuladores o contadores hardware.

La simulación permite modelar distintas configuraciones de memoria cache, como su tamaño o su algoritmo de reemplazo. Sin embargo, el tiempo necesario para simular un programa completo es de semanas o meses, por lo que estos trabajos sólo caracterizan una pequeña parte de la ejecución de cada programa.

Los contadores hardware permiten obtener resultados y caracterizar programas durante su ejecución real. Sin embargo, la plataforma sobre la que se realiza dicha caracterización permanece fija, impidiendo experimentar con distintas configuraciones.

Los trabajos dedicados a la caracterización de SPEC CPU2006 son numerosos. Jaleel et al. caracterizan el comportamiento de los programas de SPEC CPU2006 con distintos tamaños de cache sobre un simulador [9], Korn et al. estudian su rendimiento según el tamaño de página [10], y Bird et al. analizan su rendimiento ejecutando en un procesador Intel Core2Duo [11].

Respecto a SPEC CPU2017, sólo hemos encontrado el trabajo de caracterización de Limaye et al. [12]. Éste analiza con contadores hardware el comportamiento de los programas sobre un procesador Intel con micro-arquitectura Haswell, y ofrece datos generales sobre número de instrucciones ejecutadas, reparto por tipo de instrucciones, *footprint*, y tasas de fallos en cada nivel de la jerarquía. Termina presentando una metodología de clasificación de programas que analizamos más adelante. Respecto a la caracterización de la jerarquía de memoria, tiene varias limitaciones como que usa un sistema antiguo, presenta datos de tasa de fallos local en lugar de MPKI, y no analiza la sensibilidad al tamaño de cache ni a la prebúsqueda.

En este trabajo se han utilizado contadores hardware para la obtención de las métricas deseadas y se han usado diversas técnicas para variar algunos parámetros del sistema sobre el que se ejecuta. Los *Model Specific Registers* (MSR) de Intel nos permiten activar y desactivar de forma independiente los distintos prebuscadores hardware disponibles. La tecnología *Intel Resource Director* nos permite variar la capacidad de LLC dedicada a un programa, modificando el número de vías de LLC asociadas a dicho programa. Con todo ello conseguimos extraer datos del programa completo para distintas configuraciones de la jerarquía de memoria.

B. Selección de puntos de simulación

Como hemos descrito, SPEC CPU2006 y CPU2017 están formados por muchos programas, algunos de ellos con varias entradas distintas, lo que resulta en decenas de ejecuciones posibles. Como ejemplo, SPEC CPU2006 nos proporciona 55 parejas programa-entrada. El tiempo de ejecución de un programa completo sobre un simulador es del orden de semanas o meses, lo que hace inviable la simulación del conjunto completo. Para reducir dicho tiempo se usa muestreo a dos niveles. En primer lugar, se selecciona un subgrupo de los programas y entradas. En segundo lugar, se seleccionan fragmentos de la ejecución del programa que sean representativos del comportamiento de la ejecución completa. En la literatura se han propuesto diversas técnicas de muestreo entre las que destacamos *Hierarchical Clustering* [13] para seleccionar programas y las metodologías *SimFlex* [14] y *Simpoint* [15] para la selección de fragmentos.

B.1 *Hierarchical Clustering*

La metodología se aplica en tres pasos: i) ejecución de todos los programas con todas las entradas para obtener 20 métricas mediante contadores hardware, ii) análisis de componentes principales para reducir el número de métricas a 4 variables, y iii) *Hierarchical Clustering* para agrupar programas similares.

En el primer paso, los autores seleccionan métricas independientes de la microarquitectura, relacionadas con el tipo de instrucciones ejecutadas y su proporción. Por tanto, esta metodología no considera el comportamiento de la jerarquía como parámetro para guiar el muestreo.

En este trabajo analizamos la representatividad del subconjunto seleccionado con esta metodología

respecto al comportamiento de la jerarquía de memoria.

B.2 *SimFlex*

La metodología *SimFlex* usa teoría de muestreo estadístico para la selección de puntos de simulación. Identifica numerosos puntos de simulación de pequeño tamaño, que están distribuidos a lo largo de todo el programa, garantizando que sean representativos del mismo.

Sin embargo, *SimFlex* tiene un inconveniente importante para ser utilizada en la investigación en jerarquía de memoria cache, puesto que los puntos de simulación obtenidos no tienen la suficiente extensión para proporcionar datos correctos sin realizar calentamiento previo de las caches (*warm-up*). Este calentamiento supone una sobrecarga no asumible cuando las caches o el número de puntos de simulación son grandes.

B.3 *Simpoint*

Simpoint es una de las metodologías más utilizadas para la selección de puntos de simulación. *Simpoint* divide la ejecución de un programa en intervalos de igual número de instrucciones. Para cada intervalo calcula una firma que contiene el número de ejecuciones de cada uno de sus bloques básicos. (*Basic Block Vector*, BBV). Posteriormente *Simpoint* realiza una clusterización, con el algoritmo *K-means*, que agrupa los distintos intervalos en fases. Los intervalos de una fase ejecutan código similar y por tanto se espera que tengan un comportamiento similar en el sistema (fallos en la jerarquía de memoria, CPI...). El centroide de cada fase es el intervalo más representativo y es el punto de simulación de su fase.

Pese a que *Simpoint* ofrece varios puntos de simulación para cada programa, la mayoría de los trabajos que usan esta herramienta en el ámbito de la jerarquía de memoria sólo utilizan un fragmento, aquél que la metodología determina como más representativo.

Nuestro objetivo en este proyecto es determinar la representatividad de los fragmentos seleccionados por *Simpoint* respecto a la interacción de la aplicación con la jerarquía de memoria. Para ello se va a analizar la evolución temporal de métricas como CPI, MPKI2 y MPKI3 a lo largo de toda la ejecución de los programas. Los tres primeros fragmentos seleccionados por *Simpoint* se representarán sobre las gráficas de evolución temporal.

III. Metodología

En está sección presentamos la metodología utilizada para la caracterización, incluyendo el procesador utilizado, las cargas de trabajo analizadas y las métricas empleadas.

A. Entorno de ejecución

Se han ejecutado los programas de SPEC CPU2006 y los mono-hilo de SPEC CPU2017 en un procesador *Intel Xeon Skylake-SP Gold 5120* (para abreviar SKL-SP). La caracterización se ha realizado con contadores hardware mediante la herramienta `perf`. El entorno utilizado para la ejecución puede verse en la Tabla I. El procesador cuenta con 14 núcleos, dos niveles de cache privada (un primer nivel de 32 KiB para instrucciones y otros 32 KiB para datos, y un segundo nivel unificado de 1 MiB) y un tercer nivel de cache compartido (LLC de 19.25 MiB con 11 vías de asociatividad). Además, el Xeon-SP tiene cuatro prebuscadores hardware asociados al primer y segundo nivel de cache. Las ejecuciones se han realizado sobre Centos 7 con kernel 3.10. En cada experimento sólo se ejecuta un hilo en un núcleo y el resto están en *idle*.

Procesador	Intel Xeon Skylake-SP Gold 5120
Memoria Principal	96 GB DDR4
L1I Cache	8-asociativa 32 KiB (por núcleo)
L1D Cache	8-asociativa 32 KiB (por núcleo)
L2	16-asociativa 1 MiB (por núcleo)
L3	11-asociativa 19.25 MiB
S.O	Centos 7, Kernel: 3.10

TABLA I: Configuración del sistema usado

Utilizamos *Intel Cache Allocation Technology* (CAT) [16], herramienta del *Intel Resource Director*, para modificar la cantidad de LLC que tiene disponible un programa durante su ejecución. Además, activamos o desactivamos de forma selectiva los distintos prebuscadores hardware mediante el correspondiente *Model Specific Register* (MSRs). La utilización de ambas herramientas nos permite caracterizar el comportamiento de los programas completos sobre un sistema real con múltiples configuraciones de prebúsqueda y tamaño de LLC.

B. Cargas de trabajo

Se han compilado las suites CPU2006 y CPU2017 siguiendo los pasos de la documentación oficial proporcionada por SPEC. CPU2006 se ha compilado con `gcc 4.9.2` y las opciones `-O3 -fno-strict-aliasing`. CPU2017 se ha compilado con `gcc 6.3.1` y las opciones de compilación por defecto. Los programas han sido ejecutados con todas sus entradas disponibles para la versión de referencia.

C. Métricas

Para caracterizar los distintos programas que componen la suite vamos a utilizar las siguientes métricas: ciclos por instrucción (CPI), fallos cada mil instrucciones en los distintos niveles de memoria cache (MPKI) y bytes leídos de memoria principal cada mil instrucciones (BPKI).

IV. Evaluación

A. Identificación de los programas intensivos en memoria

Todas las parejas programa-entrada de SPEC CPU2006 (29 programas, 55 parejas) y las mono-

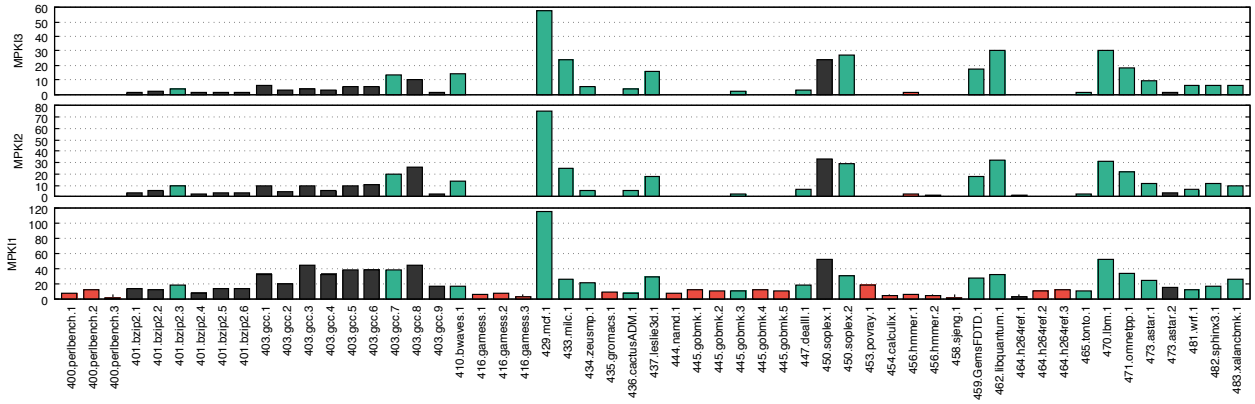


Fig. 1: MPKI1, MPKI2 y MPKI3 para todas las parejas programa-entrada de SPEC CPU2006.

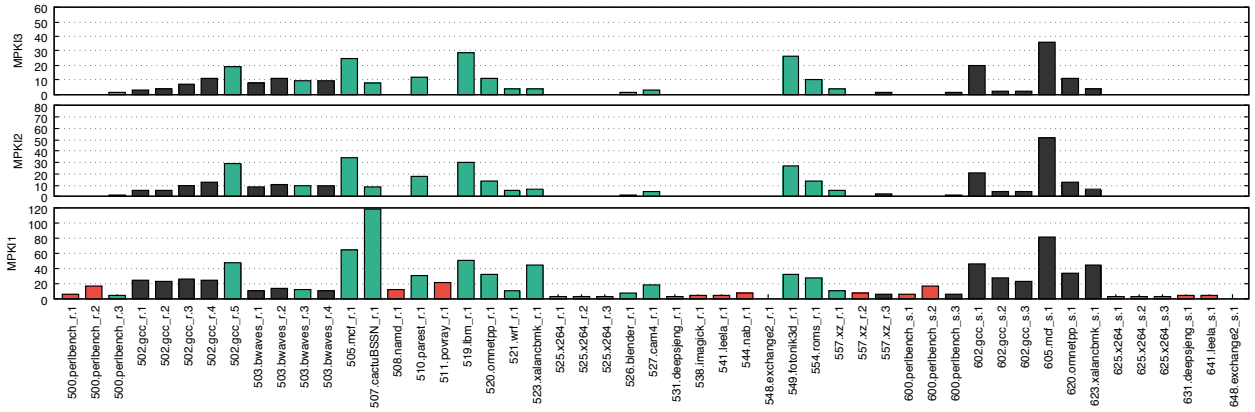


Fig. 2: MPKI1, MPKI2 y MPKI3 para todas las parejas programa-entrada mono-hilo de SPEC CPU2017.

hilo que componen SPEC CPU2017 (23 programas, 51 parejas) han sido ejecutadas en un contexto de recursos limitados: prebúsqueda hardware desactivada y el tamaño mínimo de LLC que se puede asignar a un programa (1.75 MiB). Para cada carga de trabajo hemos medido sus fallos cada mil instrucciones en los tres niveles de la jerarquía de memoria cache (MPKI1, MPKI2 y MPKI3). Estas métricas se muestran en las Figuras 1 y 2. En ellas se encuentran indicadas en rojo las parejas programa-entrada que tienen ratios MPKI2 y MPKI3 muy bajas. En total son 20 de las 55 en SPEC 2006, y 20 de las 50 consideradas en SPEC 2017. Estas cargas de trabajo carecen de interés para el estudio de la jerarquía de memoria. De las restantes, hemos seleccionado una entrada de cada programa para mostrar el resultado de su caracterización en lo que queda de esta sección (20 en CPU2006 y 16 en CPU2017, señaladas en verde en la figura).

B. *Sensibilidad al tamaño de LLC y a la prebúsqueda hardware*

En este experimento estudiamos la sensibilidad de los programas intensivos en memoria al tamaño de la LLC y a la prebúsqueda hardware. Las cargas de trabajo seleccionadas en la sección anterior han sido ejecutadas con cinco tamaños de LLC. Cada una de estas ejecuciones ha sido realizada con todos los prebuscadores activados y todos desactivados.

Los tamaños de LLC utilizados han sido: 1.75 MiB, 3.5 MiB, 7 MiB, 14 MiB y 19.25 MiB, cuyas asociatividades son 1, 2, 4, 8 y 11, respectivamente.

La Figura 3 muestra el MPKI3 de los *benchmarks* seleccionados para los distintos tamaños de LLC con y sin prebúsqueda. La figura se encuentra dividida en dos secciones, superior CPU2006 e inferior CPU2017, dentro de cada sección se ordenan de menor a mayor MPKI3 (izquierda — derecha).

Con la prebúsqueda hardware desactivada, el incremento de tamaño en LLC se traduce en una reducción de MPKI3 en todos los programas en ambas suites, a excepción de `410.bwaves`, `434.zeusmp`, `459.GemsFDTD` en CPU2006 y `503.bwaves` en CPU2017.

Cuando se activa la prebúsqueda, la mejora en MPKI conseguida al aumentar el tamaño de la LLC se reduce de forma considerable para 6 programas de CPU2006 (`433.milc`, `437.leslie3d`, `447.dealII`, `450.soplex.2`, `462.libquantum` y `481.wrf`), y para 5 de CPU2017 (`510.parest`, `519.lbm`, `521.wrf`, `549.fotonik3d` y `554.roms`).

La prebúsqueda hardware es muy efectiva para reducir MPKI3 para todos los tamaños de LLC en 14 programas de CPU2006 y en 10 de CPU2017. También obtiene buenos resultados para tamaños pequeños de LLC en otros 5 programas de CPU2006 (`401.bzip2`, `465.tonto`, `473.astar`, `482.sphinx3` y `483.xalancbmk`) y en 2 de CPU2017 (`510.parest` y `523.xalancbmk`). Sólo para `omnetpp`, presente en las dos suites, la prebúsqueda no reduce MPKI3 en

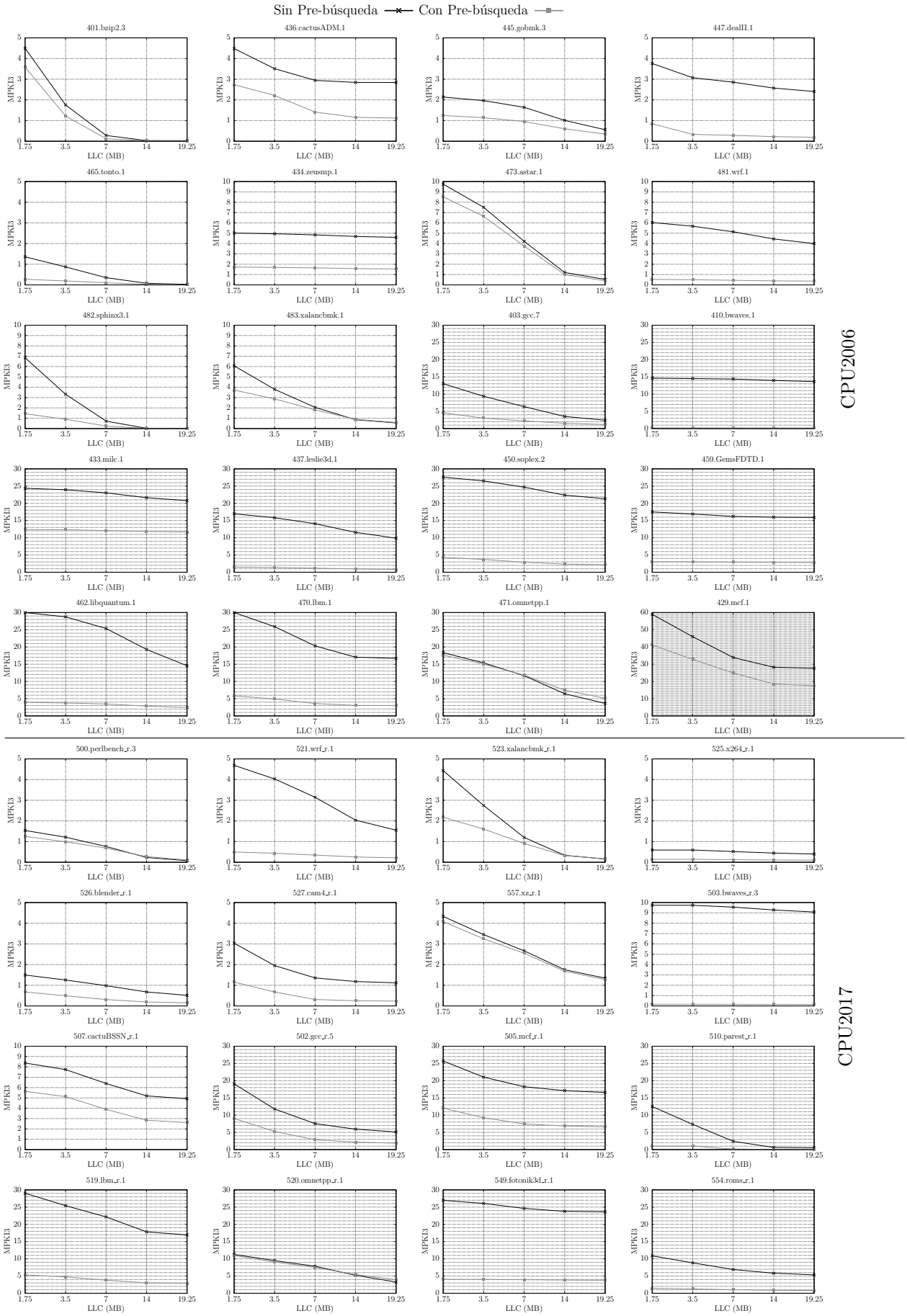
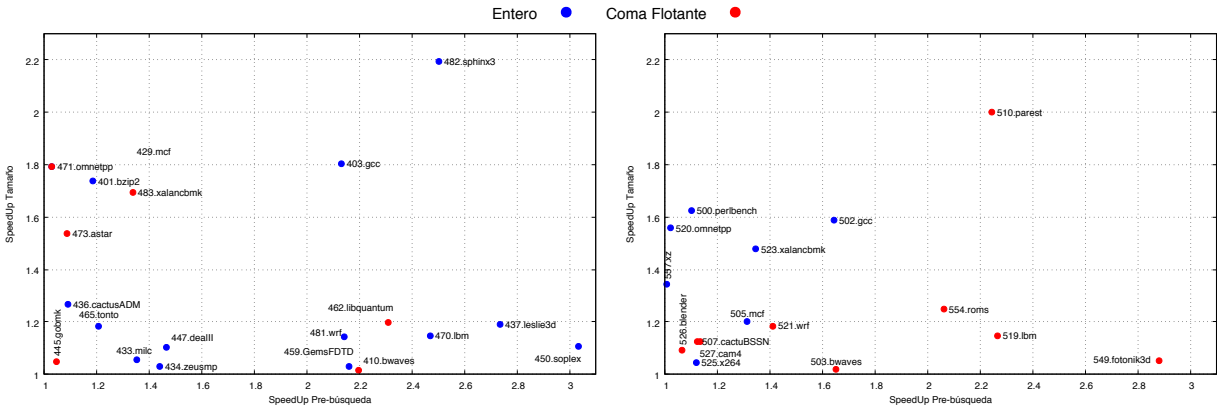


Fig. 3: Fallos en LLC (MPKI3) de las parejas programa-entrada seleccionadas para distintos tamaños de LLC, variando la asociatividad en 1, 2, 4, 8 y 11 en cada gráfica.



(a) CPU2006 (b) CPU2017

Fig. 4: SpeedUp de la prebúsqueda hardware y del tamaño para los SPEC CPU2006 y CPU2017.

ningún caso, e incluso la incrementa ligeramente con el mayor tamaño de LLC.

A modo de resumen, la Figura 4 muestra los *speedups* conseguidos para cada programa respecto al sistema con tamaño mínimo de LLC y sin prebúsqueda, al aplicar prebúsqueda (eje X), y al aumentar el tamaño de LLC hasta los 19.25 MiB (eje Y). La figura facilita la clasificación de programas en función de su sensibilidad a ambos parámetros. Las aplicaciones de enteros se muestran en azul y las de coma flotante en rojo. A modo de ejemplo, en CPU2006 vemos un grupo de programas muy sensibles a la prebúsqueda y poco sensibles al aumento de tamaño de LLC integrado por `462.libquantum`, `481.wrf`, `459.GemsFDTD`, `410.bwaves`, `470.lbm`, `437.leslie3d` y `450.soplex`.

C. Rendimiento de los prebuscadores Hardware

En esta sección vamos a analizar el impacto de los distintos prebuscadores hardware sobre el rendimiento de los programas y el ancho de banda consumido. El procesador Intel SKL-SP tiene cuatro prebuscadores hardware asociados al primer y segundo nivel de cache: *L1 Data cache unit prefetcher* (DCUI), *L1 Data cache instruction pointer stride prefetcher* (DCUP), *L2 Data cache spatial prefetcher* (L2A) y *L2 Data cache streamer* (L2P) [6].

Todos los benchmarks seleccionados en la sección IV-A han sido ejecutados con distintas configuraciones: todos los prebuscadores activados, todos los prebuscadores desactivados y cada prebuscador activado de manera individual. Los experimentos se han realizado con el tamaño máximo de LLC. En la Figura 5 se muestra su rendimiento medido en ciclos por instrucción (CPI, eje izquierdo) y el número de bytes leídos de memoria principal (BPKI, eje derecho).

L2P es el mejor prebuscador con mucha diferencia sobre los demás. Para los 14 programas de CPU2006 cuyas tasas de fallos se reducen al activar la prebúsqueda con el tamaño máximo de LLC, L2P en solitario obtiene más del 70 % de la reducción en CPI conseguida con todos los prebuscadores activos. En 10 de los 14 programas, L2P es responsable de más del 90 % de la mejora en CPI. En CPU2017 ocurre algo similar. Para los 10 programas que mejoran con la prebúsqueda en el tamaño mas grande de LLC, activando sólo L2P se consigue más del 82 % de la reducción en CPI conseguida cuando se activan todos los prebuscadores, y este porcentaje es mayor del 90 % en 7 de los 10 programas.

El segundo mejor prebuscador es DCUI, seguido por DCUP. L2A es el que consigue peores resultados, logrando reducir el CPI más de un 5 % sólo en 8 programas de CPU2006 y en 6 de CPU2017, con un máximo de un 20 % para `450.soplex`.

La prebúsqueda hardware es muy precisa, puesto que causa un aumento significativo en el ancho de banda utilizado en solamente 3 programas de la suite CPU2006 (`403.gcc.7`, `433.milc` y `471.omnetpp`) y en 3 programas de los CPU2017 (`520.omnetpp`, `549.fotonik3d` y `554.roms`). Además, exceptuando `471.omnetpp` y `520.omnetpp`, hay una disminución de CPI considerable, por lo que el resultado global es positivo.

D. Evolución temporal de los programas

En esta sección se presentan los resultados de caracterización de la evolución temporal de los programas a lo largo de su ejecución. Las figuras muestran MPKI3 (eje Y) para cada millón de instrucciones (eje X). Esto nos permite conocer las distintas fases por las que pasa un programa y nos ayuda a seleccionar puntos de simulación.

En la misma gráfica se han incluidos los tres primeros puntos de simulación obtenidos mediante la metodología *Simpoint* en forma de líneas verticales con distintos patrones. La línea continua corresponde al punto de simulación más representativo, la punteada al segundo punto más representativo y la rallada al tercero. El grosor de las líneas no es proporcional al número de instrucciones que representan. Se ha incrementado para mejorar su visibilidad.

Los puntos seleccionados por *Simpoint* son a veces representativos de las distintas fases del programa y otras no. Como ejemplo de estos últimos, podemos observar que `401.bzip2` pasa claramente por tres fases con transitorios entre ellas. El primer punto de *Simpoint* está en la primera fase y los dos siguientes

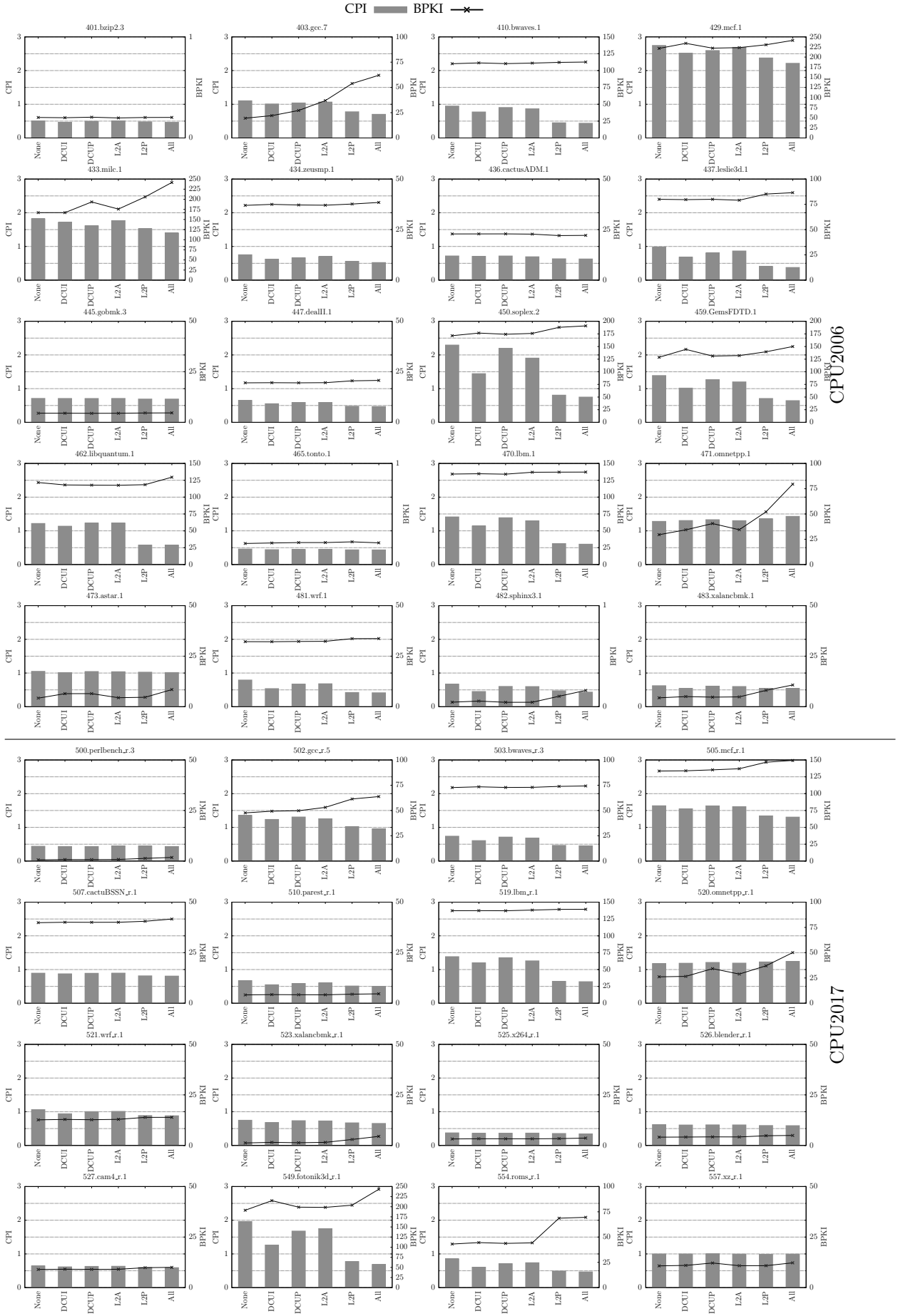


Fig. 5: Impacto de los prebuscadores hardware en el rendimiento de los programas (CPI) y en el ancho de banda utilizado (BPKI).

en la segunda. No hay ningún punto en la tercera.

Este análisis pone de manifiesto las limitaciones de esta metodología para obtener puntos representativos de la ejecución de un programa desde el punto de vista del uso de la jerarquía de memoria. El problema es todavía mayor si pensamos que la mayoría de los trabajos de investigación que usan esta metodología únicamente usan el primer intervalo.

V. Conclusiones

En este trabajo hemos analizado el rendimiento de los programas que conforman SPEC CPU2006 y CPU2017 mono-hilo en la jerarquía de memoria de un procesador Intel Xeon Skylake Scalable. A continuación resumimos las principales conclusiones que podemos extraer de esta caracterización.

Una parte importante de las parejas programa-entrada tienen ratios de fallos muy bajas en el segundo y tercer nivel de caches, incluso con un tamaño pequeño de LLC y sin prebúsqueda hardware. La demanda de recursos de la jerarquía en CPU2017 es menor que en CPU2006.

Ofrecemos una clasificación de los programas que sí usan los niveles altos de la jerarquía según su sensibilidad al tamaño de LLC y a la prebúsqueda hardware. La prebúsqueda hardware es muy efectiva reduciendo los fallos en LLC en una buena parte de los programas, incluso con los tamaños más pequeños de LLC. El aumento del tamaño de LLC también es efectivo reduciendo la tasa de fallos en LLC para muchos programas.

El mejor prebuscador de los implementados en el procesador SKL-SP es L2P. Este prebuscador es el responsable de más del 90 % de la mejora conseguida al aplicar prebúsqueda en la mayoría de los programas.

Los prebuscadores son muy precisos. En general, aumentan poco el número de bytes leídos de memoria principal.

El análisis temporal de las programas muestra como la utilización de *Simpoint* no garantiza obtener puntos de simulación representativos desde el punto de vista del uso de la jerarquía de memoria.

Agradecimientos

Este trabajo ha sido financiado en parte por los proyectos TIN2016-76635-C2-1-R (AEI/FEDER, UE) y gaZ: T58_17R research group (Gobierno de Aragón y Fondo Social Europeo).

Referencias

[1] "The SPEC organization, `https://www.spec.org/spec/`," .

[2] Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi, "A study of emerging scale-out workloads on modern hardware," p. 11.

[3] Heiko Falk, Sebastian Altmeyer, Peter Hellinckx, Björn Lisper, Wolfgang Puffitsch, Christine Rochange, Martin Schoeberl, Rasmus Bo Sørensen, Peter Wägemann, and Simon Wegener, "TACLeBench: A benchmark collection to support worst-case execution time research," .

[4] "EEMBC - embedded microprocessor benchmarks, `https://www.eembc.org/about/index.php`," .

[5] R. Panda, S. Song, J. Dean, and L. K. John, "Wait of a decade: Did spec cpu 2017 broaden the performance horizon?," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2018, pp. 271–282.

[6] "Intel® 64 and IA-32 architectures software developer's manual, combined volumes: 1, 2a, 2b, 2c, 2d, 3a, 3b, 3c, 3d and 4," p. 4844.

[7] Arnaldo Carvalho de Melo, "The new linux 'perf' tools," 2010.

[8] "Intel® resource director technology (intel® RDT), `https://www.intel.com/content/www/us/en/architecture-and-technology/resource-director-technology.html`," .

[9] Aamer Jaleel, "Memory characterization of workloads using instrumentation-driven simulation," .

[10] Wendy Korn and Moon S. Chang, "Spec cpu2006 sensitivity to memory page sizes," *SIGARCH Comput. Archit. News*, vol. 35, no. 1, pp. 97–101, Mar. 2007.

[11] Lu Peng Tribuvan Kumar Prakash, "Performance characterization of spec cpu2006 benchmarks on intel core 2 duo processor," .

[12] Ankur Limaye and Tosiron Adegbija, "A workload characterization of the spec cpu2017 benchmark suite," 04 2018.

[13] Fionn Murtagh and Pedro Contreras, "Methods of hierarchical clustering," *CoRR*, vol. abs/1105.0121, 2011.

[14] T. F. Wenisch, R. E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J. C. Hoe, "Simflex: Statistical sampling of computer system simulation," *IEEE Micro*, vol. 26, no. 4, pp. 18–31, July 2006.

[15] Erez Perelman, Greg Hamerly, Michael Van Biesbrouck, Timothy Sherwood, and Brad Calder, "Using simpoint for accurate and efficient simulation," *SIGMETRICS Perform. Eval. Rev.*, vol. 31, no. 1, pp. 318–319, June 2003.

[16] "Resource allocation in intel® resource director technology, `https://01.org/intel-rdt-linux/blogs/fyu1/2017/resource-allocation-intel%C2%AE-resource-director-technology`," .

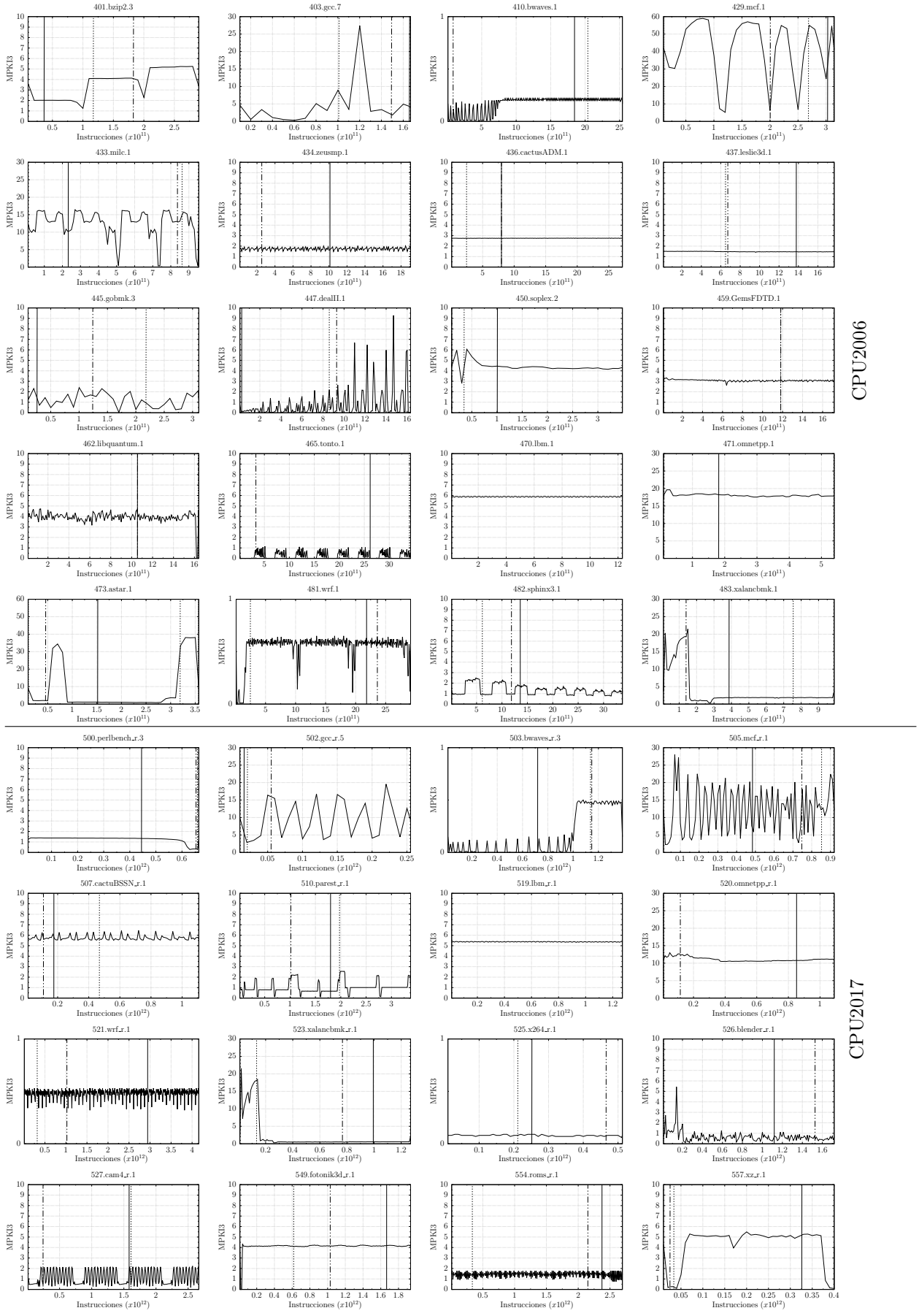


Fig. 6: Evolución temporal de los fallos en la LLC (MPKI3) y selección de *simpoints*.