UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Informática de Sistemas y Computadores

Impacto de la ubicación en núcleos SMT de aplicaciones paralelas para sistemas móviles

Trabajo Fin de Máster

Máster Universitario en Ingeniería de Computadores y Redes

AUTOR/A: Ben Ali, Yemna

Tutor/a: Petit Martí, Salvador Vicente

Cotutor/a: Sahuquillo Borrás, Julio

CURSO ACADÉMICO: 2022/2023

Departamento de Informática de Sistemas y Computadores
Universitat Politècnica de València

Impacto de la ubicación en núcleos SMT de aplicaciones paralelas para sistemas móviles

Trabajo fin de Máster

Máster Universitario en Ingeniería de Computadores y Redes

Autor: Yemna Ben Ali

Tutor: Salvador Petit Martí
Julio Sahuquillo Borrás

Curso 2022/2023

Resum

Els dispositius mòbils personals, com ara rellotges intel·ligents, telèfons i altres sistemes de baix consum, executen una àmplia gamma d'aplicacions amb elevades demandes computacionals. Moltes d'aquestes aplicacions han estat dissenyades per ser paral·leles, aprofitant la capacitat dels processadors multinucli que aquests sistemes incorporen per a executar múltiples fils simultàniament.

Recentment, s'han introduït en el mercat dispositius mòbils amb processadors que inclouen nuclis amb suport a l'execució de múltiples fils simultanis (SMT). En aquest context, la decisió de com distribuir els fils d'execució de les aplicacions paral·leles entre els nuclis del processador té un fort impacte en les prestacions i el consum energètic d'aquests sistemes.

Aquest Treball de Fi de Màster (TFM) es centra en l'estudi de polítiques de col·locació de fils d'execució en sistemes mòbils amb processadors SMT. S'analitza en profunditat com aquestes polítiques afecten tant a les prestacions com al consum d'energia. Els resultats mostren que la política més adequada depèn de i) les característiques de les aplicacions relacionades amb la microarquitectura del processador, com ara la precisió del predictor de salts i les taxes d'errors de les memòries cau i ii) la distribució de tipus d'instruccions de cada aplicació. La política òptima per a cada cas en termes de prestacions i consum energètic depèn de la complementarietat respecte a aquestes característiques esmentades.

Paraules clau: Dispositius mòbils, Prestacions, Consum energètic, Ubicació de fils, Geekbench, Perf, RAPL

Resumen

Los dispositivos móviles de uso personal, como relojes inteligentes, teléfonos, tablets y otros sistemas de bajo consumo, ejecutan una amplia gama de aplicaciones con amplias exigencias computacionales. Muchas de estas aplicaciones se han diseñado para ser paralelas, aprovechando la capacidad de los procesadores multinúcleo que estos sistemas incorporan para ejecutar múltiples hilos de manera simultánea.

Recientemente, se han introducido en el mercado dispositivos de gama *mobile* con procesadores que incluyen núcleos con soporte a la ejecución de múltiples hilos simultáneos (SMT). En este contexto, la decisión de cómo distribuir los hilos de ejecución de las aplicaciones paralelas entre los núcleos del procesador tiene un fuerte impacto en las prestaciones y el consumo energético de dichos sistemas.

Este TFM se centra en estudiar políticas de ubicación de hilos de ejecución en sistemas móviles con procesadores SMT. Se analiza en profundidad cómo las políticas afectan tanto a las prestaciones como al consumo de energía. Los resultados muestran que la política más adecuada depende de i) características de las aplicaciones relacionadas con la microarquitectura del procesador, tales como la precisión del predictor de saltos y las tasas de fallos de las caches y ii) la distribución de tipos de instrucciones de cada aplicación. La mejor política para cada caso en lo que respecta a prestaciones y consumo energético depende de la complementariedad respecto a las mencionadas características.

Palabras clave: Dispositivos móviles, Prestaciones, Consumo energético, Ubicación de hilos, Geekbench, Perf, RAPL

Abstract

Personal mobile devices, such as smartwatches, phones, tablets, and other low-power systems, run a wide range of applications with significant computational demands. Many of these applications have been designed to be parallel, taking advantage of the multi-core processors that these systems incorporate to execute multiple threads simultaneously.

Recently, mobile devices with processors that include cores supporting simultaneous multi-threading (SMT) have been introduced to the market. In this context, the decision of how to distribute the execution threads of parallel applications among the processor cores has a strong impact on the performance and energy consumption of these systems.

This Master's thesis focuses on studying thread placement policies in mobile systems with SMT processors. It delves into how these policies affect both performance and energy consumption. The results show that the most suitable policy depends on i) application characteristics related to the processor's microarchitecture, such as branch predictor accuracy and cache miss rates, and ii) the distribution of instruction types within each application. The optimal policy for each case in terms of performance and energy consumption depends on the complementarity with respect to these mentioned characteristics.

Key words: Mobile devices, Energy consumption, Thread-to-core allocation, Geekbench, Perf, RAPL

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

En la actualidad, los sistemas móviles y otros sistemas de bajo consumo trabajan con una gran cantidad de tipos de aplicaciones que requieren grandes recursos computacionales para obtener las máximas prestaciones. Al mismo tiempo, las limitaciones de consumo energético, la disipación, y la eficiencia energética son factores clave que deben considerarse para evaluar el éxito en el mercado de un determinado sistema móvil. En otras palabras, el objetivo perseguido por este tipo de sistemas no es sólo maximizar las prestaciones, sino hacerlo cumpliendo con las limitaciones energéticas impuestas por la tecnología y el usuario y con la máxima eficiencia energética.

En este contexto, un aspecto determinante tanto para las prestaciones y el consumo energético del sistema es la planificación del sistema operativo. Por ejemplo, cuando existen más procesos a ejecutar que núcleos de cómputo disponibles, el planificador puede decidir qué conjunto de procesos ejecutar al mismo tiempo en un momento dado. De esta manera, el planificador decide qué aplicaciones compiten en cada instante por los recursos del sistema, afectando a las prestaciones y el consumo energético resultantes.

Otro ejemplo es el de los sistemas con núcleos que soportan la ejecución simultánea de múltiples hilos (*Simultaneous Multithreading* o SMT) [4]. En este tipo de sistemas, un factor determinante tanto en lo que respecta a prestaciones como a consumo es la asignación de los procesos a cada núcleo o su *ubicación*. Así, si por ejemplo contamos con 4 procesos (a,b,c,d) y 2 núcleos ($C0$,$C1$) con capacidad de ejecutar dos hilos simultáneamente (SMT-2), el impacto tanto en prestaciones como en consumo puede variar ampliamente dependiendo de si los procesos a y b se ubican en mismo núcleo $C0$ (y por tanto c y d en $C1$), o bien asignar a y c a $C0$ (y dejar b y d en el núcleo restante). La razón por la que las prestaciones y el consumo de una ubicación difiere respecto a la otra reside en la *complementariedad* de los procesos en lo que respecta a la contención en los recursos compartidos dentro del núcleo, como pueden ser las caches de primer (L1) y segundo (L2) nivel. Por ejemplo, si se ubican juntos dos procesos cuyos tamaños de *working set* superan la capacidad de la cache en L2, cabe esperar que la tasa de fallos de la L2 se incremente, afectando negativamente a las prestaciones. Este comportamiento se podría mejorar asignando al mismo núcleo procesos complementarios, donde la suma de las capacidades requeridas por ambos procesos no supera el espacio disponible en L2.

El problema se complica cuando en vez de procesos independientes (es decir aplicaciones secuenciales con un sólo hilo de ejecución) se considera el caso más realista donde los procesos o hilos son parte de la misma aplicación paralela. Si por ejemplo contamos con dos aplicaciones a y b con dos hilos cada una ($a1$,$a2$ y $b1$,$b2$), una aproximación sen-

cilla podría ser colocar los hilos de una misma aplicación en el mismo núcleo. Así los hilos $a1$ y $a2$ se ejecutarían en uno de los núcleos del procesador, mientras que $b1$ y $b2$ lo harían en el núcleo restante. Esta estrategia (denominada *non-combined* en este trabajo) se fundamenta en facilitar la comunicación de los hilos de la misma aplicación mediante los mecanismos de memoria compartida que funcionan a través de las caches. Sin embargo, como se ha comentado previamente, no es la única opción: se podría ubicar hilos de distintas aplicaciones en el mismo núcleo (lo que en este TFM se denomina estrategia *combined*). Como se verá en este trabajo, dependiendo de las aplicaciones, asignar los hilos de la misma aplicación a los mismos núcleos físicos no siempre conlleva ni las mejores prestaciones ni el mejor consumo energético, ya que el impacto de la complementariedad de los hilos supera en muchos casos a las ventajas que provienen de facilitar la compartición de datos a través de las caches.

En resumen, la planificación del sistema operativo y, en particular, la política de ubicación de los hilos de las aplicaciones en los núcleos de los procesadores de los sistemas móviles tiene un impacto significativo en su rendimiento y eficiencia energética [5, 6]. Por ello, en este trabajo se presenta un primer estudio que servirá de base para el diseño de nuevas políticas de ubicación. Este estudio explora las prestaciones y el consumo energético de la ejecución de cargas compuestas por múltiples aplicaciones representativas de sistemas móviles.

Como carga de trabajo, utilizamos la suite de *benchmarks* Geekbench, ampliamente utilizada para comparar las prestaciones de dipositivos móviles y que incluye un variado conjunto de aplicaciones actualmente en funcionamiento en los dispositivos móviles modernos, como reconocimiento de voz, tratamiento de imágenes, geolocalización, etc. El estudio se centra en la asignación de los hilos de las aplicaciones estudiadas entre los núcleos del procesador y su impacto en prestaciones y consumo de energía. Los experimentos se realizan sobre una plataforma Intel NUC con procesador del segmento *mobile*. Esta plataforma no sólo permite medir las prestaciones y los eventos del hardware relacionados con las prestaciones, sino que proporciona soporte a Intel RAPL, una interfaz que permite medir el consumo energético de diversos componentes del sistema.

Los resultados del estudio demuestran que la mejor estrategia de ubicación de hilos (*combined* o *non-combined*) tanto para maximizar las prestaciones o minimizar el consumo energético depende de: i) métricas relacionadas con la microarquitectura, como la precisión del predictor de saltos o las tasas de fallos de las caches de L1 y L2; y ii) características de las aplicaciones, como el porcentaje de instrucciones de larga latencia, por ejemplo, las instrucciones de coma flotante o criptográficas. Este TFM realiza un análisis pormenorizado de estas métricas y características para encontrar la correlación con las prestaciones en términos de instrucciones por ciclo (IPC). En particular, se observa que *combined* mejora las prestaciones cuando hilos de diversas aplicaciones muestran un comportamiento complementario en lo que respecta a la contención en los recursos compartidos de un núcleo físico y las instrucciones de alta latencia. Es más, los beneficios de este comportamiento complementario superan en muchos casos a los obtenidos por *non-combined*, que asegura que hilos de la misma aplicación paralela comparten el mismo núcleo físico. Con respecto al consumo energético, en general, se observa una correlación entre el IPC y el consumo (a mayor IPC, mayor consumo), pero ese comportamiento no se da en algunos casos donde la asignación de hilos con mayores prestaciones es también la que menos consumo presenta. El estudio apunta que la causa subyacente tras este último

comportamiento es la alta tasa de fallos en L2 de una de las aplicaciones en ejecución, lo que reduce el consumo energético.

1.1 Objetivo

El objetivo principal de este estudio es realizar un análisis pormenorizado del comportamiento de cargas multiprograma compuestas por aplicaciones paralelas típicas de dispositivos móviles. Los resultados obtenidos proporcionarán una base sólida para que un futuro planificador pueda tomar decisiones que actúen sobre la distribución de los hilos de las aplicaciones paralelas entre los núcleos del procesador. Esta base permitirá diseñar estrategias de planificación de procesos que mejoren la eficiencia energética y por tanto prolonguen la vida útil de la batería y reduzcan la disipación de los sistemas móviles, beneficiando tanto a usuarios como al medio ambiente.

1.2 Estructura de la memoria

El resto del presente trabajo se organiza como sigue:

- En el capítulo 2 se discuten algunos artículos y TFM relevantes relacionados con este trabajo.
- En el capitulo 3 se presenta la plataforma experimental utilizada, la cual incluye el sistema NUC donde se llevan a cabo los experimentos, y se proporciona información detallada sobre la microarquituctura del procesador *Tiger Lake*. Además, se explican RAPL y Perf, herramientas esenciales para medir el consumo energético y las métricas de rendimiento, así como las dos políticas de ubicación estudiadas: *combined* y *non-combined*.
- En el capitulo 4 se exponen las aplicaciones analizadas, sus características principales (que serán clave en el análisis de los resultados experimentales) y los *scripts* desarrollados para ejecutarlas y monitorizarlas.
- En el capítulo 5 se analizan los resultados experimentales sobre prestaciones y consumo energético y se discute como las asignaciones de hilos estudiadas afectan a estas métricas. En primer lugar, se tratan los experimentos realizados con aplicaciones paralelas ejecutándose en solitario y, en segundo lugar, los de las cargas multiprograma compuestas por aplicaciones paralelas.
- En el capítulo 6 se exponen las principales conclusiones extraídas en este TFM.

CAPÍTULO 2
Trabajo relacionado

Existen pocas publicaciones sobre la caracterización del comportamiento de las cargas móviles teniendo en cuenta la arquitectura del procesador. Esto se debe en parte a la poca información pública disponible específica sobre la microarquitectura de los procesadores en dispositivos móviles. A este respecto, este capítulo presenta algunos trabajos relacionados, distinguiendo artículos de investigación y TFM relevantes publicados en la Universitat Politècnica de València.

2.1 Artículos de investigación

En [7] se realiza un análisis exhaustivo de varias cargas de trabajo móviles representativas de la suite Geekbench 5 [8]. Para ello, los autores utilizan la herramienta ARM DS5 [9] para obtener información de los contadores de eventos hardware en la *Performance Monitoring Unit* (PMU) y del sistema operativo. El análisis se realiza en un sistema Samsung Galaxy 10+ con SoC Exynos9820 compuesto por 8 núcleos heterogéneos. El objetivo global del estudio es ayudar a los arquitectos de computadores a comprender mejor el comportamiento de sistemas móviles ejecutando aplicaciones típicas en ese tipo de entornos. En este sentido, el estudio analiza diversas claves para entender el impacto en las prestaciones de diversos elementos del sistema, como las caches, el paralelismo a nivel de instrucción, o el predictor de saltos. Por ejemplo, el estudio demuestra que la capacidad de la cache dedicada a cada aplicación tiene un gran impacto en las prestaciones, lo cuál implica que el espacio de cache debe ser distribuido cuidadosamente para evitar injusticias cuando varias aplicaciones se ejecutan simultáneamente. Este trabajo es el más relacionado con el presente TFM, pero se centra en una arquitectura del procesador completamente diferente, que no soporta SMT, por lo que no puede responder a las cuestiones que se abordan en este trabajo sobre la ubicación de hilos en núcleos SMT. Además, no realiza ningún análisis sobre el consumo energético del sistema.

En [10], Banerjee y John se centran en el escalado dinámico de voltaje y frecuencia (DVFS) de los diferentes componentes de un sistema móvil, como son los núcleos de la CPU y la GPU. El objetivo principal del estudio es analizar exhaustivamente las prestaciones y eficiencia energética de los gobernadores que controlan el DVFS en los sistemas móviles. Los sistemas móviles modernos cuentan con varios gobernadores con diferentes objetivos en lo que respecta al compromiso entre prestaciones y consumo energético. El estudio busca identificar el gobernador óptimo para distintas plataformas móviles va-

riando el tipo de carga de trabajo, lo que permite determinar la estrategia más eficiente en diversos escenarios. El estudio muestra que una estrategia de gobierno DVFS óptima debe seleccionar inteligentemente el modo DVFS no solo para los núcleos, sino también para otros componentes del dispositivo móvil, con el fin de lograr las mejores prestaciones con el menor consumo energético. En comparación, las estrategias que gobiernan el modo DVFS de cada componente de forma independiente no siempre garantizan la mejor eficiencia energética. Por ejemplo, para cargas de trabajo con un uso intensivo de CPU, gobernar cada DVFS de forma independiente resulta subóptimo debido al aumento del tiempo de ejecución. Por otra parte, se constata que, en la mayoría de los escenarios donde un sólo componente es el responsable de la mayor parte del consumo una estrategia del tipo *race-to-idle* [11] es la más eficaz. En resumen, diseñar una estrategia que ofrezca las prestaciones deseadas junto con una alta eficiencia energética requiere una caracterización adecuada de la carga de trabajo y una distribución inteligente de la corriente eléctrica a los diversos componentes del sistema.

La propuesta publicada en [12] presenta un nuevo enfoque para el análisis de las prestaciones de las cargas móviles. Esta investigación se centra en utilizar datos históricos obtenidos en fuentes públicas para desarrollar modelos de prestaciones precisos. Mediante el análisis de diversas características de las cargas de trabajo y las métricas de prestaciones, el enfoque propuesto busca predecir cómo se comportarán nuevas cargas de trabajo en distintas arquitecturas. Para ello, los autores crean un modelo basado en una *Deep Neural Network* (DNN). Detallan el proceso de recopilación de datos para entrenar el modelo, el preprocesamiento de los datos y el entrenamiento en sí. Se destaca la importancia de seleccionar cuidadosamente las características relevantes de la carga y normalizar los datos para lograr la máxima precisión. La precisión del modelo se comprueba utilizando las suites de benchmarks Geekbench 3 [13] y SPEC CPU2006 [14]. Los resultados experimentales muestran que la precisión del enfoque propuesto es comparable a metodologías de caracterización y predicción de prestaciones convencionales, lo que demuestra el potencial del uso de conjuntos de datos públicos para predecir las prestaciones de las cargas de trabajo en los sistemas móviles.

2.2 Trabajos fin de máster

En la Universitat Politècnica de València se han publicado algunos TFM centrados en SMT y en dispositivos con restricciones de consumo energético, como sistemas empotrados con cargas de tiempo real y móviles.

En [15] se presentan dos planificadores para la reducción del consumo en la ejecución de tareas de tiempo real en procesadores multinúcleo y multihilo de grano grueso. Ambos planificadores utilizan *Dynamic Voltage Scaling* (DVS) para ahorrar energía sin comprometer la planificabilidad del sistema. El primer algoritmo está diseñado para tareas de tiempo real no estrictas mientras que el segundo se enfoca a tiempo real estricto gobernado por un planificador EDF (*Earliest Deadline First*). Ambos algoritmos se evalúan con tres niveles de frecuencia y diversas distribuciones de tareas. Los resultados experimentales demuestran que la implementación de técnicas DVS en un planificador ofrece mayores beneficios en comparación con uno que no las utiliza. Además, aumentar el número de niveles DVS resulta en mayores beneficios. La combinación de DVS con

heurísticas adecuadas de ubicación de tareas en los núcleos permite reducir, en algunos casos, hasta la mitad del consumo de energía.

En [16], la autora se enfoca a estrategias de ubicación de aplicaciones secuenciales en núcleos de procesadores con soporte SMT-2 de Intel, donde cada núcleo admite hasta dos hilos en ejecución simultánea. El trabajo presenta diferentes políticas que mejoran gradualmente el rendimiento de las aplicaciones y analiza las razones detrás de estas mejoras. Los resultados de las políticas se comparan con la política utilizada por Linux y con una política del estado del arte. El estudio se realiza en dos máquinas Intel diferentes: una orientada a alto rendimiento (Intel Xeon E5-2620 v4) y otra utilizada en investigación de dispositivos móviles (Intel Core i5-1145G7). Los resultados experimentales muestran que los algoritmos desarrollados logran mejoras significativas en ambas máquinas. En la máquina de alto rendimiento, las políticas alcanzan hasta un 21,6 % de mejora en comparación con Linux, con un promedio del 14,5 %. En la máquina orientada a dispositivos móviles, se logran mejoras de hasta un 22,8 %, con un promedio del 11,3 % en comparación con Linux. Al contrario que en el presente TFM, este trabajo no considera aplicaciones paralelas, lo cuál no es representativo de las cargas móviles actuales, que, en muchos casos, presentan un alto grado de paralelismo. Además, como en otros estudios, no se analiza el consumo energético.

CAPÍTULO 3
Plataforma experimental

En este capítulo se presenta la plataforma experimental. Primero se introduce el sistema NUC donde se desarrollan los experimentos y se detalla la microarquitectura *Tiger Lake* del procesador. En segundo lugar, se explican RAPL y Perf, componentes necesarios para la medición del consumo energético y las métricas de prestaciones. Finalmente se presentan las 2 políticas de ubicación de procesos en núcleos estudiadas: *combined* y *non-combined*.

3.1 Sistema NUC

Los experimentos se han realizado sobre un sistema NUC (*Next Unit of Computing*) NUC11TNBv5 [17]. Los sistemas NUC [18] son una línea de computadores de pequeño formato desarrollados por Intel que incorporan procesadores destinados a sistemas móviles.

En particular, nuestro sistema NUC dispone de un procesador Intel Core i5-1145G7 que incluye 4 núcleos. Cada núcleo es capaz de ejecutar 2 hilos simultáneamente (*Simultaneous multithreading* o SMT [4]). La memoria principal del sistema tiene una capacidad de 32GB distribuida en 2 módulos DDR4 DDR4-2666. El almacenamiento secundario está compuesto por una unidad de estado sólido (SSD) de 256GB conectada al sistema mediante el estándar de expansión M.2. Incluye una tarjeta de red LAN Intel I225-LM con interfaz Ethernet de 2.5 Gigabit.

3.2 Microarquitectura Tiger Lake

El procesador Intel Core i5-1145G7 se basa en la microarquitectura *Tiger Lake*. Esta microarquitectura se implementa en *System on Chip* (SoC) con una tecnología de 10nm dirigido al mercado de dispositivos móviles. La figura 3.1 muestra el diagrama de un procesador de 4 núcleos con microarquitectura *Tiger Lake*. El procesador consta de 5 componentes principales: núcleos (*core*) CPU, los cuales incluyen las correspondientes caches L1I, L1D y L2 y una porción de la cache L3 o de último nivel (*Last Level Cache* o LLC) en nuestro sistema; la red de interconexión (*Network on Chip* o NoC); el controlador de

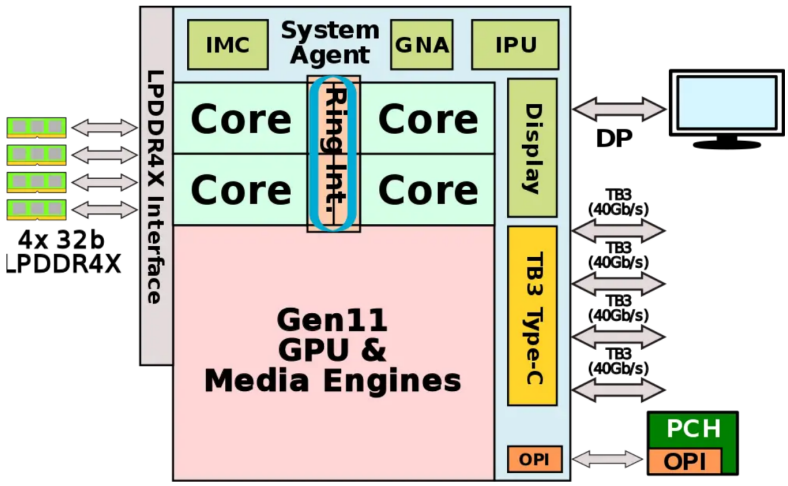

Figura 3.1: Diagrama de la microarquitectura *Tiger Lake*. Fuente [1].

memoria (denominado *LPDDR4X Interface* en la figura[1]); el *System Agent* para gestión de periféricos y almacenamiento secundario; y la GPU.

Los 4 núcleos CPU que puede incluir un procesador *Tiger Lake* se conectan, junto con el *System Agent* y la GPU, a través de una NoC con topología anillo. La memoria principal alimenta a los núcleos a través del controlador de memoria con 4 canales, brindando un ancho de banda en el rango de 50-60 GBps.

Los núcleos implementan internamente la microarquitectura *Willow Cove*. La figura 3.2 muestra un diagrama con las tres partes principales del *pipeline* de esta microarquitectura: el *front-end* (marcado en azul), el *back-end* (en rojo), y el subsistema de memoria de datos (en verde). El objetivo del *front-end* es suministrar al *back-end* un flujo de microoperaciones (µOP) para su ejecución. Para ello, el *front-end* dispone de dos rutas principales: la ruta de cache de µOP y la ruta clásica. En la ruta clásica las µOP se obtienen a partir de la decodificación de instrucciones x86 almacenadas en la cache de instrucciones (L1I$ en la figura). Esta ruta resulta más lenta y menos eficiente que la ruta de la cache de µOP (µOP$ en la figura), la cual proporciona directamente las µOP para alimentar al *back-end*. La microarquitectura *Willow Cove* permite alimentar al *back-end* con hasta 6 µOPs por ciclo.

En el *back-end*, las µOP se almacenan en el *reorder buffer* (ROB) y se les asignan los recursos necesarios para su ejecución (como entradas en colas de instrucciones). Desde el ROB, las µOP se envían al planificador. Este cuenta con varios puertos de lanzamiento (salida), cada uno conectado a un conjunto de unidades de ejecución diferentes. Ciertas unidades pueden realizar cálculos aritméticos elementales, mientras que otras ejecutan operaciones de mayor complejidad como multiplicación, división y operaciones vectoriales, entre otras. Adicionalmente, hay unidades destinadas específicamente a la carga y el almacenamiento de datos en el subsistema de memoria. El planificador se encarga de encolar las µOP en el puerto adecuado para que puedan ser ejecutados por la unidad correspondiente. En *Willow Cove*, el ROB puede almacenar hasta 352 µOP y el planificador cuenta con 10 puertos de salida.

Las µOP de carga y almacenamiento operan sobre el subsistema de memoria de datos, donde se encuentran las cache de datos. *Willow Cove* cuenta con una caché de datos de

[1] Los procesadores *Tiger Lake* se pueden encontrar en sistemas con memoria principal DDR4 o LPDDR4.

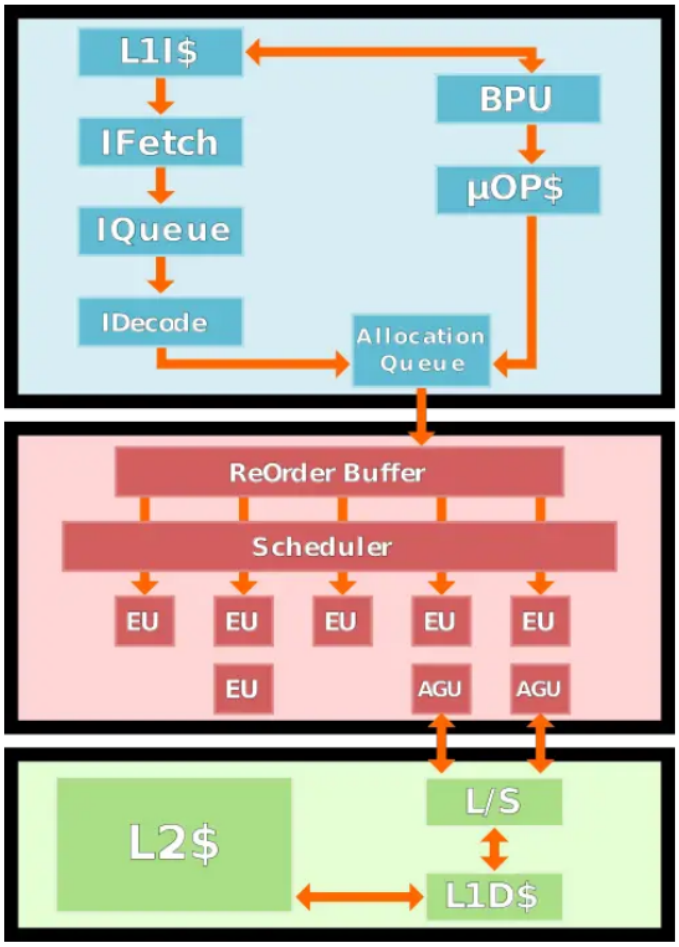

Figura 3.2: Diagrama de bloques de la microarquitectura de los núcleos Willow Cove. Fuente [2].

nivel 1 (L1D$) de 48 KB y una caché de instrucciones (L1I$) de 32 KB. La cache L2$ es unificada con una capacidad de 1,25 MB. Las caches de nivel 1 y nivel 2 son privadas para cada núcleo mientras que la cache de nivel 3 es compartida y almacena 2MB por núcleo, lo que supone 8MB en total para un procesador de 4 núcleos.

3.3 Interfaz RAPL

La mayoría de los procesadores modernos, incluidos los de Intel, disponen de contadores hardware para monitorizar eventos de la microarquitectura y métricas de prestaciones (p.ej. IPC). En las implementaciones más recientes, estos contadores hardware también monitorizan el consumo energético. En el caso de Intel, el consumo de energético de diversos componentes del procesador se expone a través de la interfaz *Running Average Power Limit* (RAPL).

RAPL se introdujo en la microarquitectura *Sandy Brdige* [19]. En líneas generales, RAPL proporciona dos funciones principales. En primera instancia, posibilita la medición del consumo de energía con granularidad fina (del orden de decenas de microjulios) y con alta frecuencia de muestreo. En segundo término, permite la reducción del consumo promedio de energía en los diversos componentes del procesador, lo que a su vez

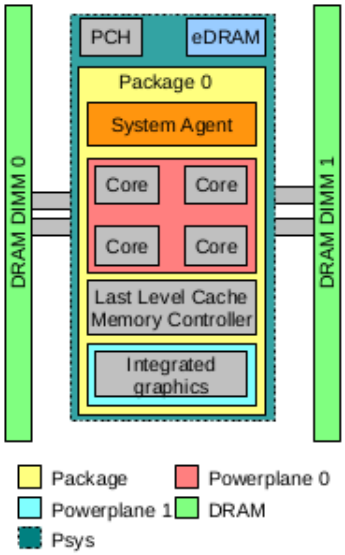

Figura 3.3: Dominios de potencia soportados por RAPL. Fuente [3].

disminuye la disipación. En este estudio, nos concentraremos específicamente en la capacidad de medición de la energía.

La interfaz RAPL ofrece acceso a diversos *dominios de potencia*, cada uno de los cuales posee sus propias capacidades para medir el consumo de energía y limitarlo. Los dominios soportados en nuestro sistema se ilustran en la figura 3.3. Estos dominios se describen a continuación.

Package: El dominio *Package* mide el consumo de energía de todo el zócalo del procesador, incluyendo el consumo de los núcleos, la GPU, las caches, el controlador de memoria, etc.

Powerplane 0: El dominio *Powerplane 0* mide el consumo de energía de todos los núcleos de procesador, lo que incluye todas las caches, exceptuando la LLC.

Powerplane 1: El dominio *PowerPlane 1* se dedica exclusivamente a la medición del consumo de la GPU.

DRAM: El dominio DRAM mide el consumo de energía de los módulos DIMM del sistema.

Psys: A partir de la microarquitectura *Skylake* se introdujo un nuevo dominio denominado *Psys*. Permite supervisar y controlar las especificaciones energéticas y térmicas de todo el SoC. Como se muestra en la figura, Psys incluye el consumo de energía del *Package*; del *Platform Controller Hub* (PCH), encargado de la conexión con los periféricos; y de la eDRAM, la cual es accedida por la GPU.

En este trabajo, utilizamos el sistema operativo Linux para llevar a cabo los experimentos con RAPL. Según [19], existen varios métodos para acceder a RAPL desde el espacio de usuario en Linux:

Lectura de los ficheros en el directorio /sys/class/powercap/intel-rapl/intel-rapl:0: Estos ficheros son exportados al sistema de archivos virtual */sys* de Linux por la interfaz

Power Capping Framework (o simplemente *powercap*) integrada en el núcleo de Linux. En general, no se necesitan permisos de superusuario para acceder a esta interfaz.

Acceso al dispositivo /dev/msr: El dispositivo */dev/msr* de Linux permite el acceso desde el espacio de usuario a los diferentes *Model Specific Registers* (MSR), los cuales, a su vez proporcionan las métricas monitorizadas por RAPL. Para habilitarlo es necesario cargar el módulo del núcleo *msr* mediante la orden `modprobe msr`. El acceso a */dev/msr* requiere permisos de superusuario.

Utilizando la herramienta Perf: Perf es la herramienta oficial de Linux dedicada al análisis de prestaciones. A partir de la versión 3.14 del núcleo de Linux, esta herramienta incorpora las métricas monitorizadas por RAPL. En general, este método requiere permisos de superusuario.

3.4 Perf

Este trabajo se centra en la medición del consumo del procesador utilizando la interfaz RAPL. Para ello utilizaremos la herramienta de análisis de prestaciones Perf [20].

Después de realizar diversas pruebas con las otras opciones mencionadas (*/dev/msr* y *powercap*), hemos elegido Perf principalmente porque, en nuestro sistema NUC, Perf soporta el acceso a un mayor número de dominios que las otras alternativas. Además, Perf presenta ventajas adicionales con respecto a las demás opciones. Por ejemplo, es más potente: puede monitorizar los contadores de prestaciones de la CPU, trazar dinámicamente la ejecución de las aplicaciones y el núcleo de Linux, y realizar un *profiling* de esta ejecución de forma periódica. Perf está integrado con el núcleo de Linux, por lo que se actualiza y mejora con frecuencia. Ofrece diversas abstracciones para entender las capacidades y el comportamiento de los diversos componentes del sistema. Entre otras, proporciona métricas por hilo de ejecución, por núcleo, y por carga.

Algunos ejemplos de uso de Perf son los siguientes:

- Analizar qué partes de una aplicación o del núcleo consume un mayor tiempo de procesador mediante el trazado de la ejecución de las funciones.
- Acceder a las *Performance Monitoring Units* (PMU) que contienen los valores de los contadores de prestaciones.
- Instrumentar la ejecución añadiendo sondas en el código de usuario (*uprobes*) o del núcleo (*kprobes*), respectivamente.

En un sistema determinado pueden coexistir varias PMU, y cada evento monitorizable corresponde a una de las PMU existentes. Las PMU correspondientes a los eventos hardware se definen en función de la arquitectura específica del procesador. Así, en los procesadores Intel con arquitectura x86 se define la `x86_pmu`. El valor de los contadores de prestaciones que monitorizan los eventos de esta PMU es obtenido por Perf de los registros MSR. La figura 3.4 representa las diversas capas de abstracción relacionadas con el acceso a un evento en Perf.

La herramienta Perf presenta una interfaz de línea de comandos. Algunos ejemplos de uso de esta interfaz de comados son:

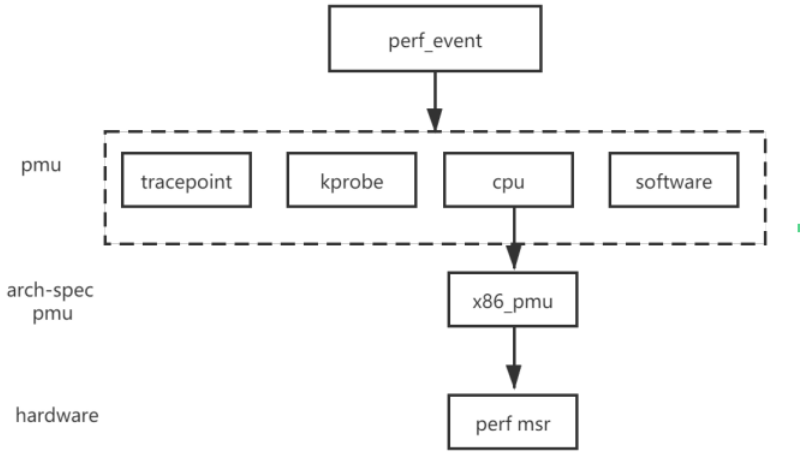

Figura 3.4: Capas de abstracción en Perf involucradas en el acceso a un evento.

perf stat: obteniene recuentos de eventos.

perf record: registra eventos en un fichero para realizar informes posteriores.

perf report: desglosa los eventos por proceso, función, etc.

perf annotate: anota código ensamblador o fuente con recuentos de eventos.

perf top: presenta recuentos de eventos durante la ejecución.

perf bench: ejecuta diferentes *microbenchmarks* orientados a evaluar las prestaciones del núcleo de Linux.

3.5 Políticas de ubicación estudiadas

Tal como se ha descrito, el procesador de nuestra plataforma implementa 4 núcleos con soporte a la ejecución de 2 hilos. Por lo tanto, puede haber hasta 8 hilos en ejecución simultánea. En otras palabras, soporta hasta 8 contextos hardware.

El objetivo de este trabajo es evaluar el impacto en consumo energético y prestaciones de diversas políticas de ubicación de hilos de aplicaciones paralelas (del ámbito de los sistemas móviles) entre los diferentes contextos hardware. En particular, para 2 aplicaciones paralelas compuestas por 4 hilos, nos planteamos estudiar dos políticas: *combined* y *non-combined*.

En la política de ubicación *combined* cada uno de los 4 hilos de una aplicación paralela se sitúan en un núcleo diferente del procesador. Como cada núcleo SMT dispone de 2 contextos hardware, esto implica que en cada núcleo queda libre uno de los contextos para ejecutar un hilo de la aplicación paralela restante.

Por otro lado, en la política de ubicación *non-combined*, los 4 hilos de una aplicación paralela se sitúan en 2 núcleos SMT. Así, todos los contextos hardware de esos 2 núcleos quedan ocupados, mientras que los núcleos libres se pueden dedicar a la otra aplicación paralela.

Núcleo físico	0		1		2		3	
Núcleo lógico	0	4	1	5	2	6	3	7
Combined	A_0	B_0	A_1	B_1	A_2	B_2	A_3	B_3
Non-combined	A_0	A_1	A_2	A_3	B_0	B_1	B_2	B_3

Tabla 3.1: Distribuciones de hilos combined y non-combined.

Para identificar el contexto hardware donde se ejecuta un hilo determinado, Linux numera los contextos tal como se muestra en la Tabla 3.1. Los núcleos SMT (núcleos físicos en la tabla) se numeran de forma ascendente, mientras que los 2 contextos hardware (o núcleos lógicos, siguiendo la terminología de Linux) soportados por cada núcleo se numeran de forma entrelazada. Así, para 4 núcleos físicos, el núcleo físico 0 soporta los núcleos lógicos 0 y 4, el núcleo físico 1 los núcleos lógicos 1 y 5, etc.

La tabla también muestra a qué núcleos lógicos se asignan los hilos de las aplicaciones paralelas (A y B) para las políticas *combined* y *non-combined*. Como se puede apreciar, en la política de ubicación *combined* los hilos de la aplicación A ($A_{0...3}$) se colocan en el primer núcleo lógico de cada núcleo físico, mientras que los de la aplicación B se asignan al segundo núcleo lógico. En comparación, *non-combined* coloca todos los hilos de A en los núcleos lógicos de los 2 primeros núcleos físicos, y los hilos de B en los núcleos lógicos de los núcleos físicos restantes.

CAPÍTULO 4
Aplicaciones estudiadas

En este capítulo se presenta las aplicaciones estudiadas y los *scripts* desarrollados para ejecutarlas y monitorizarlas. Primero se describen las aplicaciones incluidas en la suite de *benchmarks Geekbench*. En segundo lugar, se explica el *script* que utilizamos para realizar los experimentos. Finalmente, se introducen las características principales de las aplicaciones que serán de utilidad en el análisis de los resultados experimentales.

4.1 Geekbench

En el contexto de la evaluación del consumo energético del procesador en dispositivos móviles, la elección de una suite de *benchmarks* adecuada desempeña un papel crucial en la obtención de resultados realistas y relevantes. En este trabajo hemos optado por la suite de *benchmarks Geekbench* [8], ya que presenta una serie de ventajas en comparación a otras suites del estado del arte en lo que respecta a la evaluación de dispositivos móviles. En particular, *Geekbench* se compone de aplicaciones y código sintético comunes en el ámbito de los sistemas móviles, como procesamiento de imágenes, criptografía, reconocimiento de voz, etc. En efecto, *Geekbench* es una suite específicamente diseñada para la evaluación de dispositivos móviles. En comparación, las suites orientadas a sistemas de alto rendimiento se centran en plataformas de servidores y estaciones de trabajo, por lo que no capturan de manera precisa las particularidades de las cargas de los dispositivos móviles.

Geekbench dispone de *benchmarks* secuenciales y paralelos. En este trabajo nos centramos en los *benchmarks* paralelos, los cuales se organizan en la suite en tres categorías: aplicaciones de criptografía, aplicaciones de enteros, y aplicaciones de coma flotante. En las siguientes secciones se describen las aplicaciones de estas tres categorías.

4.1.1. Aplicaciones de criptografía

Esta sección está compuesta de una sola aplicación, **AES-XTS**: El estándar AES (*Advanced Encryption Standard*) define un algoritmo de cifrado de bloques simétrico. El cifrado AES se usa ampliamente para proteger los canales de comunicación (por ejemplo, HTTPS) y para proteger la información (por ejemplo, cifrado de almacenamiento, cifrado de dispositivos).

La aplicación AES-XTS cifra un *buffer* de 128 MB utilizando AES que se ejecuta en modo XTS con una clave de 256 bits. El buffer se divide en bloques de 4KB. Para cada bloque, la carga de trabajo deriva un contador XTS usando el *hash* SHA1 del número de bloque. A continuación, el bloque se procesa en fragmentos de 16 bytes utilizando AES-XTS, que implica un cifrado AES, dos operaciones XOR y una multiplicación GF(2128).

4.1.2. Aplicaciones de enteros

La categoría de aplicaciones de enteros está compuesta por los siguientes *benchmarks*:

Text Compression: La aplicación *Text Compression* utiliza LZMA (*Lempel-Ziv-Markov chain-Algorithm*) para comprimir y descomprimir un fichero HTML. LZMA es un algoritmo de compresión sin pérdidas. El algoritmo utiliza un esquema de compresión de diccionario (el tamaño del diccionario es variable y puede ser tan grande como 4 GB). LZMA presenta una alta relación de compresión (más alta que *bzip2*).

Esta aplicación comprime y descomprime un fichero HTML de 2399 KB mediante el algoritmo de compresión LZMA con un tamaño de diccionario de 2048 KB. La aplicación utiliza el SDK de LZMA para la implementación del algoritmo LZMA principal.

Image Compression: La aplicación *Image Compression* comprime y descomprime una fotografía utilizando el algoritmo de compresión de imágenes con pérdida JPEG, y comprime y descomprime un sprite CSS utilizando el algoritmo de compresión de imágenes sin pérdida PNG.

La fotografía es una imagen de 24 megapíxeles y el parámetro de calidad JPEG se establece en "90", un ajuste de uso común para los usuarios que desean una imagen de alta calidad. La compresión JPEG es implementada por la librería *libjpeg-turbo*.

El sprite CSS es una imagen de 3 megapíxeles. La compresión PNG se implementa mediante las librerías *libpng* y *zlib-ng*.

Navigation: La aplicación *Navigation* calcula las ruta de conducción entre una secuencia de destinos utilizando el algoritmo de Dijkstra. Se utilizan técnicas similares para calcular rutas en juegos y para enrutar el tráfico de redes informáticas.

El conjunto de datos contiene 216548 nodos y 450277 aristas con pesos que aproximan al tiempo de viaje a lo largo de la carretera representada por la arista. La ruta incluye 13 destinos. El conjunto de datos se basa en datos de *OpenStreetMap* para Ontario, Canadá.

SQLite: SQLite es un motor de base de datos SQL autónomo (sin servidor) y el que cuenta con un mayor número de instalaciones en el mundo.

La aplicación SQLite ejecuta consultas SQL en una base de datos en memoria principal. La base de datos se crea sintéticamente para imitar datos financieros y se genera utilizando técnicas descritas en [21]. La aplicación está diseñada para estresar el motor de la base de datos utilizando una variedad de características SQL (como claves primarias y externas) y comandos clave de consulta como: SELECT, COUNT, SUM, WHERE, GROUP BY, JOIN, INSERT, DISTINCT, y ORDER BY.

PDF Rendering: El formato PDF (*Portable Document Format*) es un formato de archivo estándar utilizado para presentar e intercambiar documentos en formato impreso entre sistemas con distinto software o hardware. Los archivos PDF se utilizan de muchas maneras, desde documentos y formularios gubernamentales hasta libros electrónicos.

Esta aplicación analiza y muestra un mapa en PDF del Parque Nacional *Crater Lake* a 200 ppp. Para ello, se utiliza la librería *PDFium* (la misma que Google Chrome utiliza para mostrar archivos PDF).

Text Rendering: La aplicación *Text Rendering* analiza un documento con formato *Markdown* y lo representa como texto enriquecido en una imagen. La aplicación utiliza las siguientes librerías:

- *GitHub Flavored Markdown*: se utiliza para analizar el formato Markdown.
- *Freetype*: se utiliza para representar tipos de letra.
- ICU (*International Components for Unicode*): se utiliza para el análisis de límites.

El archivo de entrada de tiene una longitud de 1721 palabras y produce un mapa de bits de 1275 píxeles por 9878 píxeles de tamaño.

Clang: *Clang* es un front-end de compilación para los lenguajes de programación C, C++, Objective-C, Objective-C++, OpenMP, OpenCL y CUDA. Utiliza LLVM como *backend*.

Clang compila un archivo fuente C de 1094 líneas (de las cuales 729 líneas son código) utilizando AArch64 como arquitectura de destino para la generación de código.

Camera: La aplicación *Camera* simula algunas de las acciones que puede realizar una aplicación de cámara o una aplicación de red social para compartir fotos. Se simula el empleo de un filtro sobre una imagen y la preparación de esta para subirla a la nube:

- Recorte de la imagen para lograr una relación de aspecto cuadrada.
- Carga de la definición de filtro desde un archivo JSON y ejecución de las operaciones individuales del filtro:
  - Ajuste del contraste.
  - Desenfoque.
  - Composición de un efecto viñeta.
  - Composición de un borde.
- Compresión de la imagen de salida en un archivo JPEG.
- Calculo del *hash* SHA2 del archivo JPEG.

La aplicación también simula la preparación de fotos para su visualización en un *UI Picker* mediante la generación de miniaturas para la imagen:

- Consulta a una base de datos SQLite para determinar de qué imágenes faltan miniaturas.
- Generación de miniaturas.

4.1.3. Aplicaciones de coma flotante

N-Body Physics: La aplicación *N-Body Physics* calcula una simulación de gravitación en 3D utilizando el método Barnes-Hut. Para calcular exactamente la fuerza gravitacional que actúa sobre un cuerpo X en un campo de N cuerpos se requieren N-1 cálculos de fuerzas. El método de Barnes-Hut reduce el número de cálculos al aproximar como un solo cuerpo cualquier grupo de cuerpos cercanos entre sí y que se encuentren lejos de X. Esto lo hace eficientemente dividiendo el espacio en octantes (ocho cubos de igual tamaño) y subdividiendo recursivamente cada octante en octantes, formando un árbol, hasta que cada octante en las hojas del arbol contiene exactamente un cuerpo. Esta subdivisión recursiva del espacio requiere operaciones de coma flotante y accesos a memoria no contiguos.

N-Body Physics opera sobre 16384 cuerpos dispuestos en una galaxia *plana* con un agujero negro masivo en su centro.

Gaussian Blur: *Gaussian Blur* desenfoca una imagen mediante un filtro espacial gaussiano. Los desenfoques gaussianos se utilizan ampliamente en software, tanto en sistemas operativos para proporcionar efectos de interfaz como en software de edición de imágenes para reducir los detalles y el ruido en una imagen. Los desenfoques gaussianos también se utilizan en aplicaciones de visión por computadora para mejorar las estructuras de la imagen a diferentes escalas.

La aplicación *Gaussian Blur* desenfoca una imagen de 24 megapíxeles mediante un filtro espacial gaussiano. Mientras que la implementación de desenfoque gaussiano admite una sigma arbitraria, la carga de trabajo utiliza una sigma fija de 3,0 f. Esta sigma se traduce en un diámetro de filtro de 25 píxeles por 25 píxeles.

Face Detection: *Face Detection* es una técnica de visión por computadora que identifica caras humanas en imágenes digitales. Una aplicación es la fotografía, donde se utiliza para el enfoque automático. La aplicación utiliza el algoritmo presentado en [22].

Horizon Detection: *Horizon Detection* busca la línea del horizonte en una imagen. Si se encuentra, se rota la imagen para nivelar el horizonte.

La aplicación primero aplica un detector de bordes a la imagen para reducir los detalles, luego detecta líneas en la imagen usando la transformada de Hough y selecciona la línea con la puntuación máxima como horizonte. Finalmente, se rota la imagen para nivelarla.

HDR: La aplicación HDR toma cuatro imágenes de rango dinámico estándar (HDR) y produce una imagen de alto rango dinámico (HDR). Cada imagen de entrada tiene un tamaño de 3 megapíxeles. La carga de trabajo HDR utiliza el algoritmo descrito en [23] y produce imágenes superiores en comparación con el algoritmo de mapeo de tonos en *Geekbench* 4.

Ray Tracing: *Ray Tracing* es una técnica de renderizado. Genera una imagen trazando la trayectoria de la luz a través de un plano de imagen y simulando los efectos de sus encuentros con objetos virtuales. Este método es capaz de producir imágenes de alta calidad, pero estas imágenes tienen un alto costo computacional.

La aplicación utiliza un árbol k-d, una estructura de datos de partición del espacio, para acelerar los cálculos de intersección de rayos.

Speech Recognition: La aplicación de reconocimiento de voz realiza el reconocimiento de voz utilizando *PocketSphinx*, una librería ampliamente utilizada que utiliza *Hidden Markov Models*.

El uso del habla para interactuar con *smartphones* se está volviendo más popular con la introducción de Siri, el Asistente de Google y Cortana. Esta aplicación prueba la rapidez con la que un dispositivo puede procesar el sonido y reconocer el habla.

Machine Learning: *Machine Learning* es una aplicación de inferencia que ejecuta una red neuronal convolucional para realizar una tarea de clasificación de imágenes. Utiliza *Mobile Net* v1 con un alfa de 1.0 y un tamaño de imagen de entrada de 224 píxeles por 224 píxeles. El modelo se entrenó con el conjunto de datos *ImageNet*.

4.2 Lanzamiento de las aplicaciones y obtención de resultados

Para realizar los experimentos en este trabajo y obtener los resultados que se analizarán posteriormente se ha desarrollado un *script* de *bash*. En esta sección se explica el funcionamiento del *script* y de los comandos que ejecuta.

El *script* se llama `ejecuta_benchmark` y responde a la siguiente línea de comandos:

```
$ ejecuta_benchmark <código benchmark 1> <código benchmark 2>
                    <combined/non-combined/alone-nc/null>
```

Como se aprecia, el *script* tiene 3 argumentos obligatorios. Cuando se ejecutan dos aplicaciones con las políticas *combined* o *non-combined*, los dos primeros argumentos indi-

Código	Aplicación
101	AES-XTS
201	Text Compression
202	Image Compression
203	Navigation
205	SQLite
206	PDF Rendering
207	Text Rendering
208	Clang
209	Camera
301	N-Body Physics
303	Gaussian Blur
305	Face Detection
306	Horizon Detection
308	HDR
309	Ray Tracing
312	Speech Recognition
313	Machine Learning

Tabla 4.1: Códigos identificativos de las aplicaciones Geekbench.

```
if [ $3 = "combined" ]; then
    CPULIST_A="0-3"
    CPULIST_B="4-7"
elif [ $3 = "non-combined" ]; then
    CPULIST_A="0,4,1,5"
    CPULIST_B="2,6,3,7"
elif ...
```

Figura 4.1: Selección de los núcleos lógicos donde se asignarán los hilos de las aplicaciones paralelas.

can el código de la aplicación *Geekbench* a ejecutar. Estos códigos se muestran en la tabla 4.1.

El último argumento indica la ubicacion de los hilos de las dos aplicaciones. Tal como se muestra en la figura 4.1, dependiendo de la política de ubicación, el *script* asigna a las variables `CPULIST_A` y `CPULIST_B` una cadena identificando los núcleos lógicos donde asignar los hilos de las dos aplicaciones A y B. Recuérdese que la identificación de los núcleos se explica en la sección 3.5 de este trabajo.

Una vez seleccionados los núcleos lógicos donde se asignarán los hilos de las aplicaciones, estas últimas se lanzan, tal como se muestra en el código de la figura 4.2. La primera y segunda líneas del código corresponden al lanzamiento de las aplicaciones A y B, respectivamente. Ambas líneas tienen la misma estructura e involucran los mismos comandos y argumentos. En particular:

sudo: Este comando se utiliza para ejecutar el comando subsiguiente con privilegios de superusuario, lo que permite al siguiente comando en la línea de código realizar tareas que requieren permisos administrativos.

taskset: El comando taskset se utiliza para establecer la afinidad a los núcleos lógicos de una aplicación, lo que determina los núcleos donde esta se ejecutará. En nuestro código, la aplicación a ejecutar y sus argumentos se pasan a su vez como argumentos a taskset.

–cpu-list: Con este argumento se indica a taskset los núcleos lógicos para cada aplicación (`CPULIST_A` para la aplicación A y `CPULIST_B` para B).

/home/benchmark/Geekbench-5.4.1-Linux/geekbench5: El segundo argumento de taskset es a su vez la ruta del comando que ejecuta la aplicación *Geekbench* seleccionada. La aplicación en particular se establece con los argumentos subsiguientes.

```
sudo taskset --cpu-list $CPULIST_A /home/benchmark/Geekbench-5.4.1-Linux/
    geekbench5 --no-upload --workload $1 --multi-core --cpu-workers 4 --
    iterations 100000 >benchmark-A.log 2>&1 &
sudo taskset --cpu-list $CPULIST_B /home/benchmark/Geekbench-5.4.1-Linux/
    geekbench5 --no-upload --workload $2 --multi-core --cpu-workers 4 --
    iterations 100000 >benchmark-B.log 2>&1 &
```

Figura 4.2: Lanzamiento de las aplicaciones a estudiar.

–no-upload: Este argumento indica a *Geekbench* que no suba los resultados del experimento a la base de datos en línea de *Geekbench*. Los resultados sólo se almacenarán localmente.

–workload: Identifica la aplicación *Geekbench* que se ejecutará usando los códigos de la tabla 4.1. Estos códigos se proporcionan como argumentos al *script* para ambas aplicaciones (argumento $1 para A y $2 para B).

–multi-core: Indica que el *benchmark* debe ejecutarse en modo multinúcleo (es decir, como una aplicación paralela).

–cpu-workers 4: Especifica el número de hilos de la aplicación. En nuestros experimentos, el número de hilos es 4 para ocupar exactamente 4 núcleos lógicos por aplicación paralela.

–iterations 100000: Esta opción establece el número de repeticiones para el *benchmark*. Establecemos un valor de 100000 porque encontramos que representaba un compromiso adecuado entre el tiempo de ejecución de los experimentos y la estabilidad de los resultados.

Tanto la salida estándar como la de error de las aplicaciones se redirigen al mismo fichero (`benchmark-A.log` o `benchmark-B.log`) mediante los operadores de redirección > y 2>.

Nada más lanzar las aplicaciones, se lanza la herramienta perf para monitorizar los eventos hardware estudiados durante los experimentos. El código correspondiente se muestra en la figura 4.3. Tal como taskset, perf requiere permisos de superusuario, por lo que es necesario invocarlo mediante sudo. Los argumentos de perf son los siguientes:

stat: : El subcomando stat de perf se utiliza para recopilar y mostrar diversas estadísticas de rendimiento.

-x ';': El argumento '-x' especifica que los datos de la salida deben estar separados por el carácter punto y coma.

-I 500: El argumento '-I' establece el intervalo de recogida de los contadores de rendimiento. En este caso, los datos se recopilarán con un intervalo de 500 milisegundos.

-e: El argumento '-e' especifica los eventos hardware que perf debe monitorizar. Los eventos especificados aquí son:

instructions: Instrucciones ejecutadas.

cycles: Ciclos de reloj usados.

power/energy-psys/: Consumo de energía de Psys (ver sección 3.3).

```
1 sudo ../bin/perf stat -x ';' -I 500 -e instructions,cycles,power/energy-psys/,
    power/energy-pkg/,power/energy-cores/,power/energy-gpu/ >perf-metrics.txt
    2>&1 &
```

Figura 4.3: Lanzamiento de perf para la monitorización de la ejecución de las aplicaciones.

power/energy-pkg/: Consumo de energía del package.

power/energy-cores/: Consumo de energía de los núcleos.

power/energy-gpu/: Consumo de energía de la GPU.

Como en el código de lanzamiento de las aplicaciones, las salidas estándar y de error de perf se redirigen a un sólo fichero (`perf-metrics.txt`). Estos ficheros son la fuente de los resultados que se presentarán y discutirán posteriormente.

4.3 Características de las aplicaciones

De cara a analizar los resultados de los experimentos que se presentarán y discutirán posteriormente, esta sección muestra características reportadas en [24] sobre la ejecución de *Geekbench* en un sistema Dell Precision 3430 con un procesador Intel Core i3-8100. Estas características son las siguientes:

Tasa de fallos del predictor de saltos: La tasa de fallos del predictor de saltos mide la frecuencia de predicciones incorrectas del predictor de saltos situado en el *front-end* del procesador. Se suele reportar como un porcentaje sobre el total de saltos ejecutados. Cada vez que un salto se busca para ejecutarse, se ofrece una predicción del resultado del salto, lo que sirve para anticipar la búsqueda de instrucciones subsiguientes. Predicciones incorrectas suponen una pérdida de prestaciones ya que las instrucciones buscadas con motivo de la predicción deben descartarse.

	Tasa de fallos (%)				
Aplicación	Saltos	L1I	L1D	L2	L3
101	0,1	0,0	0,4	6,8	13,3
201	10,5	0,0	4,0	55,6	23,2
202	2,1	0,0	1,5	3,4	5,8
203	5,0	0,0	8,2	45,5	39,2
205	0,8	3,3	2,2	9,3	3,2
206	1,3	0,5	0,8	11,8	52,9
207	0,9	0,5	0,5	24,5	57,5
208	2,7	6,1	3,1	39,2	8,3
209	0,3	0,0	6,5	5,1	6,4
301	5,7	0,0	24,0	56,1	2,7
303	0,1	0,4	6,1	1,4	80,0
305	0,2	0,0	1,5	15,9	0,0
306	2,6	0,0	3,2	11,6	67,6
308	0,1	0,0	0,3	6,7	26,3
309	0,6	0,0	25,3	5,4	1,1
312	4,9	0,0	10,8	70,8	59,2
313	0,2	0,0	6,7	33,1	47,9

Tabla 4.2: Características de las aplicaciones estudiadas cuando se ejecutan en un sistema Dell Precision 3430 con un procesador Intel Core i3-8100.

Tasas de fallos de las caches: Las tasa de fallos en una cache mide la frecuencia con la que el procesador no encuentra instrucciones o datos en la cache al intentar leer o escribir en memoria. Se mide como un porcentaje de los accesos totales a esa cache. Cuando una aplicación intenta acceder a instrucciones o datos en memoria, primero verifica si esa información se encuentra almacenada en la cache más cercana (y por tanto más rápida) al procesador (por ejemplo, la cache L1D para los datos). Si los datos se encuentran en la cache, se puede operar más rápidamente con ellos que si se hubiera necesitado buscarlos en caches más lentas o incluso directamente en la memoria principal. Por ello, altas tasas de fallos afectan negativamente a las prestaciones.

La tabla 4.2 muestra características previamente reportadas de las aplicaciones estudiadas que serán de utilidad para los análisis realizados sobre los resultados experimentales obtenidos en este trabajo.

CAPÍTULO 5
Resultados Experimentales

En este capítulo se presentan y discuten los resultados experimentales. En primer lugar se analizan los experimentos realizados con aplicaciones paralelas ejecutándose en solitario en nuestro sistema. En segundo lugar, se analizan los resultados de cargas multiprograma compuestas por dos aplicaciones paralelas. Se analizan resultados sobre prestaciones y consumo energético y se discute el impacto de las políticas de ubicación estudiadas sobre estas métricas.

5.1 Ejecución en solitario

La figura 5.1 muestra el IPC medio de cada hilo de los 4 de una aplicación cuando esta se ejecuta en solitario. En este experimento, los 4 hilos de la aplicación se ejecutan en 2 núcleos. Estos resultados se presentan como base para analizar el impacto de la compartición del sistema con otras aplicaciones, los cuales se discutirán en la siguiente sección. Como se puede observar, las aplicaciones presentan una amplio rango de prestaciones, desde las que muestran un bajo IPC (alrededor o por debajo de 0,5: 101, 203 y 312) hasta las que muestran uno alto (por encima de 2: 202, 206, 305 y 308).

En general, las prestaciones de las aplicaciones obedecen a las características que se muestran en la tabla 4.2. En particular, una alta tasa de fallos del predictor de saltos o de las caches L2 y L3 tienen un alto impacto en el IPC. Así, aplicaciones con una alta tasa de fallos del predictor muestran un IPC por debajo de 1 (por ejemplo, 201, con una tasa de fallos del 10,5 % o 301, con un 5,7 %). Con respecto a la tasa de fallos de las caches, esta es especialmente relevante en L2 y L3, donde un fallo tiene un alto coste en términos de latencia. Esto se puede observar, por ejemplo, en las aplicaciones 203, 312 y 313, todas con altas tasas de fallos en L2 y L3, y que, en consecuencia muestran un IPC muy bajo (entre 0,4 y 0,6).

5.2 Parejas de aplicaciones

Una vez analizado el IPC en solitario de las aplicaciones y su relación con diferentes características microarquitecturales, en esta sección se analiza el impacto de la interferencia entre varias instancias de aplicaciones paralelas variando la asignación de hilos a núcleos (*combined*/*non-combined*, ver sección 3.5). Para diferenciar el comportamiento individual de las aplicaciones, se evaluan primero dos instancias de la misma aplica-

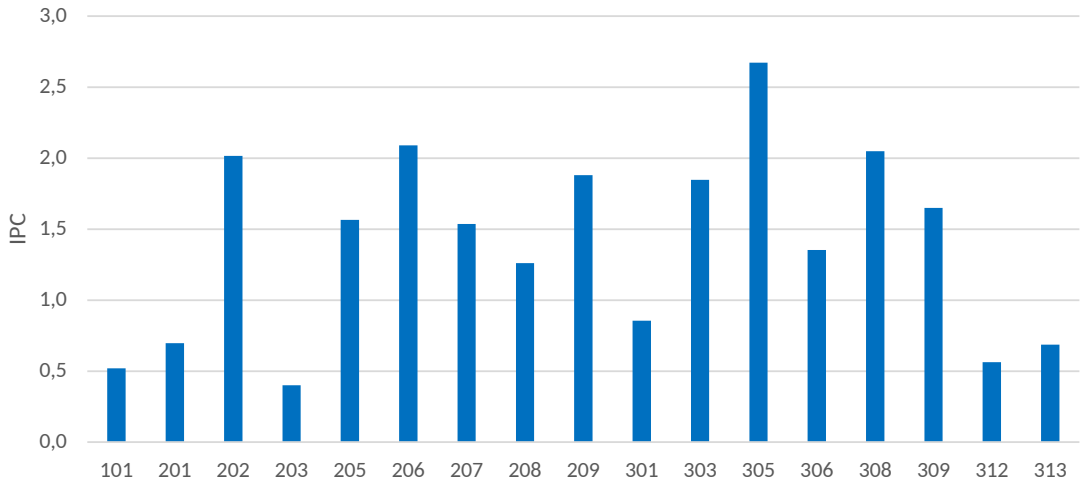

Figura 5.1: IPC de las aplicaciones estudiadas cuando se ejecutan en solitario.

ción para posteriormente analizar el comportamiento de parejas compuestas por distintas aplicaciones. En ambos casos se analizan tanto los resultados de prestaciones como los de consumo energético.

5.2.1. Parejas compuestas por la misma aplicación

Prestaciones

La figura 5.2 muestra el IPC medio por núcleo físico cuando en el sistema se ejecutan dos instancias de la misma aplicación bajo las políticas *combined* (C) y *non-combined* (NC). Si se comparan los resultados de esta figura con los de la 5.1, se pueden observar dos comportamientos. Por un lado, están aquellas aplicaciones que apenas presentan interferencias entre instancias, y que por tanto muestran un IPC parecido entre la ejecución en solitario y en pareja. Es el caso de las aplicaciones 201, 203, 205, 207, 208, 209, 303, 309 y 312. Por otro lado, se pueden diferenciar las aplicaciones que sí que sufren una pérdida de IPC por núcleo lógico cuando hay dos instancias en ejecución: 101, 202, 206, 301, 305, 306, 308 y 313. Cabe destacar el comportamiento de la aplicación 306, cuya degradación del IPC no es significativa si se aplica la ubicacion *combined* frente a *non-combined*. Esto confirma que la distribución de los hilos entre los núcleos lógicos tiene un impacto que debe

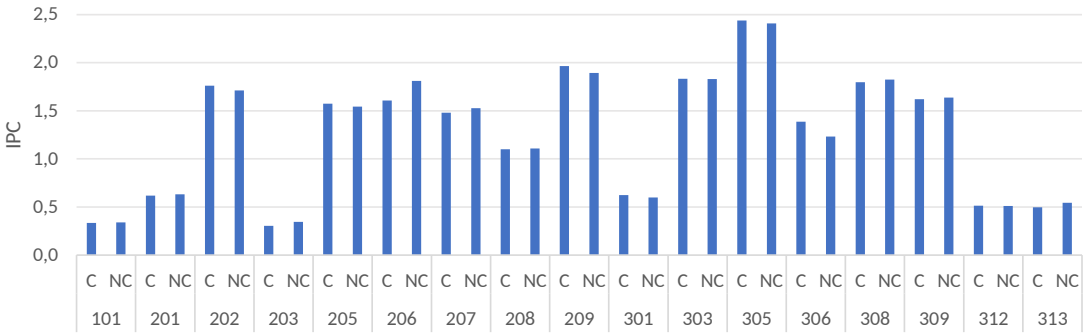

Figura 5.2: IPC de las aplicaciones estudiadas cuando se ejecutan en pareja con otra instancia de la misma aplicación.

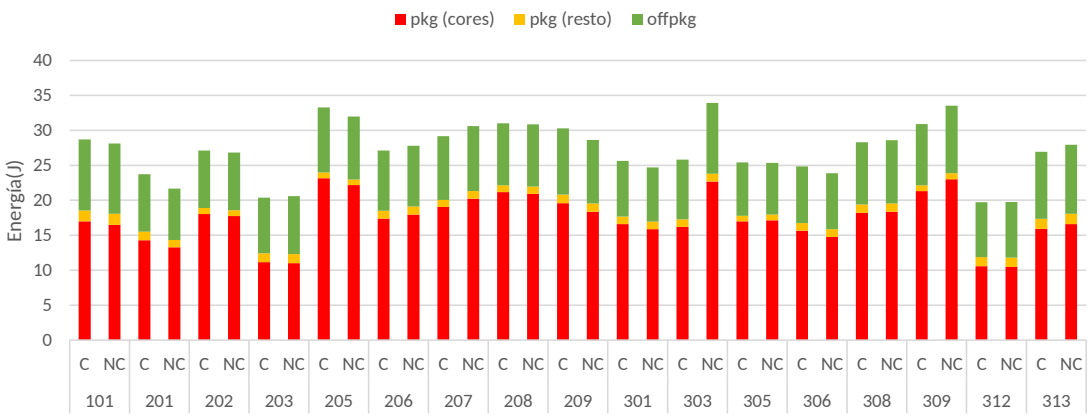

Figura 5.3: Energía consumida por las aplicaciones estudiadas cuando se ejecutan en pareja con otra instancia de la misma aplicación.

ser considerado por las políticas de planificación del sistema operativo para asegurar las máximas prestaciones. Nótese que efectos significativos diferenciados en las prestaciones de *combined* y *non-combined* también pueden observarse en otras aplicaciones como 206 o 209. Como se verá, este efecto es más acusado cuando se consideran parejas de distintas aplicaciones.

Consumo energético

La figura 5.3 muestra la energía media consumida por núcleo físico cuando en el sistema se ejecutan dos instancias de la misma aplicación variando la política de ubicación de los hilos (C/NC). El valor absoluto en Julios corresponde al consumo medio realizado en un intervalo de ejecución. Recuérdese que tal como se menciona en la sección 4.2, los intervalos son de 500ms. El consumo se divide en tres categorías, el consumo de los núcleos – *pkg (cores)* –, el del resto del package – *pkg (resto)* y el externo al package – *offpkg*. En todas las aplicaciones, la mayor parte del consumo energético se debe al *package*, y prácticamente la totalidad del consumo del *package* es causado por los núcleos. El consumo externo al *package*, aún siendo significativo, es bastante menor y además presenta menores variaciones en función de la política de ubicación de los hilos.

En general, comparando las figuras 5.2 y 5.3 no se observa una correlación entre el IPC y el consumo energético. Es decir, no siempre una aplicación con mayor IPC consume más energía que otra y viceversa. Por ejemplo, 101 presenta un bajo IPC (cercano a 0,4) y pero su consumo energético es mucho mayor que el de 201, cuyo IPC es prácticamente el doble. La razón, como se verá más adelante, es que ambas aplicaciones presentan características muy diferentes en lo que respecta a los tipos de instrucciones que ejecutan y su comportamiento con respecto a la jerarquía de cache, los cuales tienen un impacto en el consumo energético total.

Si se comparan las diferencias entre aplicaciones en la figura 5.3, se observa una menor variabilidad en el consumo que en el IPC. Aun así, se constatan diferencias significativas entre las aplicaciones que menos consumen (203, 312) y las que más (205, 309).

Observando el efecto de la política de ubicación de los hilos en cada aplicación por separado, se comprueba que este efecto es mucho más acusado en el consumo energético que en el IPC (veáse por ejemplo, la aplicación 303). Así, se puede categorizar las

aplicaciones en función del impacto que tiene la política de ubicación de los hilos en el consumo energético cuando se ejecutan dos instancias:

Aplicaciones *Combined-Friendly*: 206, 207, 303, 309 y 313. En estas aplicaciones la política *combined* presenta un menor consumo.

Aplicaciones *Non-Combined-Friendly*: 201, 205, 209, 301 y 306. Para estas aplicaciones es la política *non-combined* la que presenta un consumo significativo menor.

Aplicaciones *Location-Insensitive*: 202, 203, 208, 305, 308, y 312. En estas aplicaciones no hay variaciones significativas en el consumo energético al cambiar la política de ubicación de hilos.

La categorización de las aplicaciones presenta cierta correlación entre las variaciones de IPC al cambiar la ubicación de los hilos. Por ejemplo, 206, categorizada como *Combined-Friendly* desde el punto de vista energético, presenta un menor IPC con *combined* que con *non-combined*. Por otro lado, 209, categorizada como *Non-Combined-Friendly*, presenta el comportamiento inverso. Este resultado es esperable, ya que si se observa una aplicación dada, el consumo de los núcleos es el que más pesa en el balance energético total, y es normal que los núcleos que realizan una mayor cantidad de cómputo por unidad de tiempo consuman también una mayor cantidad de energía. En este contexto, parece que un planificador debería centrarse exclusivamente en, o bien en mejorar las prestaciones, o bien en reducir el consumo.

Sin embargo, no todas las aplicaciones responden a este patrón. Por ejemplo, 303 y 309 son aplicaciones *Combined-Friendly* desde el punto de vista energético, pero su IPC cuando se emparejan con ellas mismas apenas varía. Por lo tanto, existen oportunidades para el planificador para distribuir los hilos de manera que se observen beneficios tanto en prestaciones como en consumo. Como se verá en la siguiente sección, estas oportunidades están mucho más presentes cuando se combinan distintas aplicaciones.

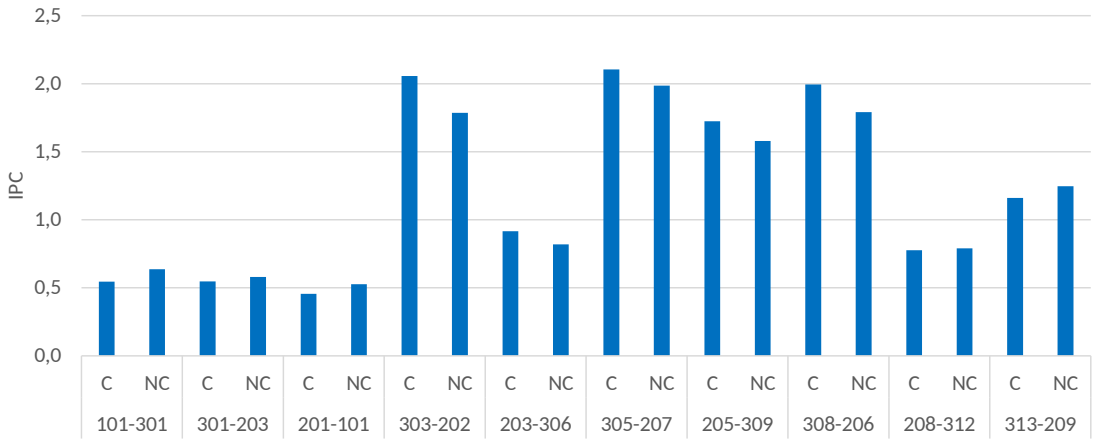

Figura 5.4: IPC presentado por parejas de distintas aplicaciones.

5.2.2. Parejas compuestas por distintas aplicaciones

Prestaciones

La figura 5.4 presenta el IPC medio de los núcleos lógicos para diferentes parejas de distintas aplicaciones con las dos asignaciones de hilos estudiadas. La selección de las aplicaciones que componen las parejas se ha hecho de forma aleatoria, aunque se han tenido en cuenta los estudios previos para incluir aplicaciones con diferentes comportamientos, de forma que se experimente una variedad suficiente de situaciones que permitan extraer conclusiones relevantes.

Si se comparan estos resultados con los presentados por parejas de la misma aplicación (figura 5.2), se observa una mayor influencia de la ubicación de los hilos en las prestaciones: prácticamente en todas las parejas existen diferencias significativas entre *combined* y *non-combined*. Tal y como ocurría en experimentos previos, ninguna política de ubicación ofrece una clara ventaja sobre la otra ni en prestaciones ni en consumo energético.

Para entender las razones por las que una políticas es mejor que la otra dependiendo de la pareja, hemos realizado un análisis exhaustivo de las características de las aplicaciones que pueden influir en cuál es la mejor política desde el punto de vista de las prestaciones. Este análisis involucra a:

1. La tabla 4.2, presentada previamente (ver sección 4.3), donde se pueden observar las tasas de fallos del predictor y de las caches.
2. La tabla 5.1, la cual muestra la distribución de tipos de instrucciones media por intervalo de ejecución para las aplicaciones estudiadas.

	Distribución de instrucciones (%)			
Aplicación	Crypto	Integer	Float	LD/ST
101	20,26	24,66	33,98	21,10
201	0,00	82,62	0,02	17,35
202	0,00	74,29	9,21	16,50
203	0,00	59,18	1,10	39,72
205	0,00	67,36	0,65	31,99
206	0,00	70,51	3,82	25,67
207	0,00	65,00	1,82	33,19
208	0,00	72,84	0,44	26,72
209	0,02	49,96	26,64	23,39
301	0,00	28,56	34,38	37,06
303	0,00	14,55	52,59	32,86
305	0,00	49,96	28,16	21,88
306	0,00	59,60	20,55	19,84
308	0,00	66,64	16,23	17,12
309	0,00	35,42	39,35	25,23
312	0,00	66,62	13,30	20,08
313	0,00	28,60	56,15	15,25

Tabla 5.1: Distribución de tipos de instrucciones en las aplicaciones *Geekbench*.

Con respecto a la última tabla, hemos comprobado que la distribución de instrucciones es coherente con los distintos tipos de aplicaciones presentados en la sección 4.1. Así, las aplicaciones de coma flotante (301...313) presentan un porcentaje de instrucciones de coma flotante siempre superior al 13 %, y en general, mayor que el 20 %. Por otro lado, la única aplicación que incluye instrucciones especializadas para criptografía es la 101, que, además, muestra un alto porcentaje de instrucciones de coma flotante. Cabe destacar también la aplicación 209, que aún estando categorizada como aplicación de enteros (201...209), presenta un porcentaje de instrucciones de coma flotante similar al de las aplicaciones de este tipo. Finalmente, nótese que el porcentaje de instrucciones de carga y almacenamiento (*LD/ST* en la tabla) ronda entre un 15 % y un 40 % de las instrucciones, independientemente del tipo de aplicación.

Para razonar qué política es mejor en cada caso, es necesario considerar lo siguiente:

- En general, la tasa de fallos en L3 (LLC) no debería tener influencia sobre la selección de la distribución, ya que se comparte por todos los núcleos lógicos en cualquier distribución. Por tanto, la localización de un hilo determinado no afecta a las necesidades que este hilo pueda tener con respecto a la LLC.
- Por el contrario, la precisión del predictor de saltos (es decir, su tasa de fallos) y las tasas de fallos de las caches de primer (L1) y segundo nivel (L2) si afectan a esta decisión, ya que todos estos componentes arquitectónicos se comporten entre 2 hilos ejecutándose en cada núcleo físico. Por ejemplo, un hilo de ejecución con una alta tasa de fallos del predictor puede causar pérdidas de prestaciones en otro hilo ejecutándose en el mismo núcleo físico.
- Intuitivamente, la política *non-combined* debería proporcionar mejores prestaciones, ya que asegura que hilos de la misma aplicación compartan las caches, favoreciendo la comunicación de los hilos a través de los mecanismos de memoria compartida. Sin embargo, *combined* podría mejorar las prestaciones de *non-combined* si los hilos de diversas aplicaciones se complementan a nivel microarquitectural. Para que esto ocurra es necesario que los hilos que comparten núcleo físico no causen interferencia en los recursos compartidos (predictor y caches de primer y segundo nivel) ni en las unidades funcionales especializadas, como son las dedicadas a las instrucciones criptográficas y de coma flotante. En otras palabras, si la política *combined* causa menos interferencias que *non-combined*, la ventaja de esta última con respecto a la memoria compartida puede no ser suficiente para alcanzar las mayores prestaciones.

La tabla 5.2 muestra las parejas donde *combined* presenta mejor IPC. Para cada aplicación de las parejas, la tabla indica las tasas de fallos y el porcentaje de instrucciones criptográficas y de salto.

Como se puede observar, todas las parejas de la tabla son complementarias a nivel de juego de instrucciones, ya que sólo una de las aplicaciones realiza un uso intensivo de instrucciones de coma flotante. En ese caso, y siempre que no exista interferencia a nivel de otros componentes compartidos en los núcleos físicos (como el predictor de saltos o las caches), *combined* presenta mejores resultados *non-combined*, ya que en esta última política, hilos de la misma aplicación que emiten muchas instrucciones de coma flotante comparten núcleo físico y por tanto sufren contención en el uso de los operadores

	Tasa de fallos (%)				Distribución de instrucciones (%)	
Pareja	Saltos	L1I	L1D	L2	Crypto	Float
303	0,1	0,0	6,1	1,4	0,00	52,59
202	2,1	0,0	1,5	3,4	0,00	9,21
203	5,0	0,0	8,2	45,5	0,00	1,10
306	2,6	0,0	3,2	11,6	0,00	20,55
305	0,2	0,0	1,5	15,9	0,00	28,16
207	0,9	0,5	0,5	24,5	0,00	1,82
205	0,8	3,3	2,2	9,3	0,00	0,65
309	0,6	0,0	25,3	5,4	0,00	39,35
308	0,1	0,0	0,3	6,7	0,00	16,23
206	1,3	0,5	0,8	11,8	0,00	3,82

Tabla 5.2: Parejas de aplicaciones donde la política *combined* presenta mayores prestaciones.

de coma flotante. Nótese que los operadores de coma flotante tienen altas latencias y se convierten en un cuello de botella cuando varios hilos compiten por usarlos. En la distribución *combined*, si dos hilos de distintas aplicaciones con requerimientos dispares con respecto a operaciones de coma flotante comparten el mismo núcleo físico, este problema no se da.

Es importante constatar que los beneficios proporcionados por la complementariedad a nivel de tipos de instrucciones pueden incluso superar el impacto negativo de la contención en otros recursos, como las caches. Este efecto se puede observar en las parejas 203-306 y 205-309, donde una de las aplicaciones presenta altas tasas de fallos en L2 (45,5 %) y L1D (25,3 %), respectivamente, y sin embargo esta interferencia no daña tanto las prestaciones como para que *combined* quede por debajo de *non-combined*. En esas parejas, la alta complementariedad a nivel de tipos de instrucciones (nótese que 203 y 205 apenas emiten instrucciones de coma flotante comparado con sus parejas) supera los inconvenientes causados por la compartición de las cache entre distintas aplicaciones.

La figura 5.5 ilustra este último hecho mencionado. Esta figura combina los datos presentados en las figuras 5.1 y 5.4. Para cada pareja, tal como en la figura 5.4, las dos

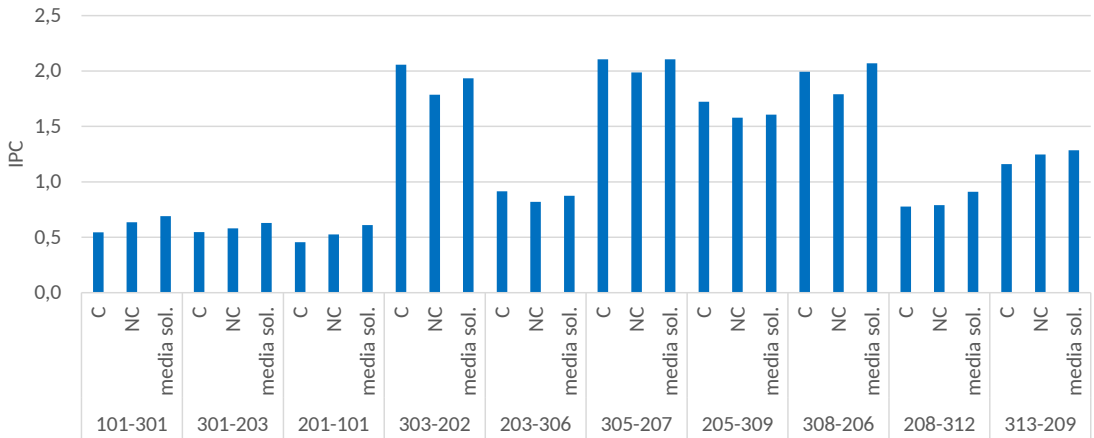

Figura 5.5: IPC presentado por parejas de distintas aplicaciones con interferencia y sin interferencia por los recursos compartidos.

	Tasa de fallos (%)				Distribución de instrucciones (%)	
Pareja	Saltos	L1I	L1D	L2	Crypto	Float
101	0,1	0,0	0,4	6,8	20,26	33,98
301	5,7	0,0	24,0	56,1	0,00	34,38
301	5,7	0,0	24,0	56,1	0,00	34,38
203	5,0	0,0	8,2	45,5	0,00	1,10
201	10,5	0,0	4,0	55,6	0,00	0,02
101	0,1	0,0	0,4	6,8	20,26	33,98
313	0,2	0,0	6,7	33,1	0,00	56,15
209	0,3	0,0	6,5	5,1	0,02	26,64

Tabla 5.3: Parejas de aplicaciones donde la política *non-combined* presenta mayores prestaciones.

primeras columnas presentan el IPC cuando dos aplicaciones se ejecutan juntas siguiendo las políticas *combined* y *non-combined*, mientras que la tercera columna dibuja la media del IPC de las aplicaciones de la pareja cuando cada una de ellas se ejecuta en solitario (es decir, la media de las columnas correspondientes en la figura 5.1). Recuérdese que los experimentos en solitario se han realizado ocupando dos núcleos físicos con los 4 hilos de una aplicación. Es decir, lo que muestra la tercera columna es el IPC de la pareja con una política *non-combined* sin tener en cuenta las interferencias entre ambas aplicaciones. Por lo tanto, lo que representa esta columna es un resultado *non-combined ideal*. Aun así, en algunas parejas (303-202, 203-306 y 205-309) las prestaciones de *combined* superan este resultado ideal. Esto es debido a que *combined*, al asignar al mismo núcleo físico aplicaciones complementarias, evita las interferencias que sufren los hilos de una misma aplicación cuando comparten núcleo, tanto en los operadores como en las caches de L1 y L2.

De manera análoga a la tabla 5.2, la tabla 5.3 muestra las parejas donde *non-combined* proporciona mayor IPC. En comparación a las parejas previas, las aplicaciones de parejas de la tabla 5.3 presentan un mayor nivel de interferencia en caches y predictor de saltos. En todas las parejas, al menos una de las aplicaciones (o incluso dos, en el caso de la pareja 301-203) presenta altas tasas de fallos en uno o más de los siguientes componentes: caches L1D, L2, y predictor de saltos. En general, es la cache L2 la que aparece con altas tasas de fallos en todas las parejas, aunque también L1D en el caso de la aplicación 301, el predictor de saltos en la aplicación 201, y, en menor medida en 301 y 203.

Como se ha argumentado previamente, en situaciones de alta interferencia, *non-combined* suele presentar mejores prestaciones, ya que al menos facilita la compartición de memoria de las aplicaciones paralelas a través de L1 y L2. Esta situación sólo se puede revertir si existe una alta complementariedad entre aplicaciones a nivel de tipos de instrucciones. En el caso de las parejas de la tabla 5.3, la complementariedad se da en las parejas 301-203 y 201-101. Sin embargo, en 301-203 ambas aplicaciones causan una fuerte interferencia en L2 (con tasas de fallos superiores al 45 %), mientras que en la última pareja, 201 no sólo causa contención en L2 (tasa de fallos del 55,6 %) sino que además introduce una alta tasa de fallos en el predictor de saltos (superior al 10 %), lo cuál afecta negativamente a las prestaciones de hilos de otras aplicaciones bajo la política *combined*.

En resumen, los resultados de prestaciones demuestran que la mejor distribución de hilos está íntimamente relacionada, en primer lugar, con la complementariedad a nivel

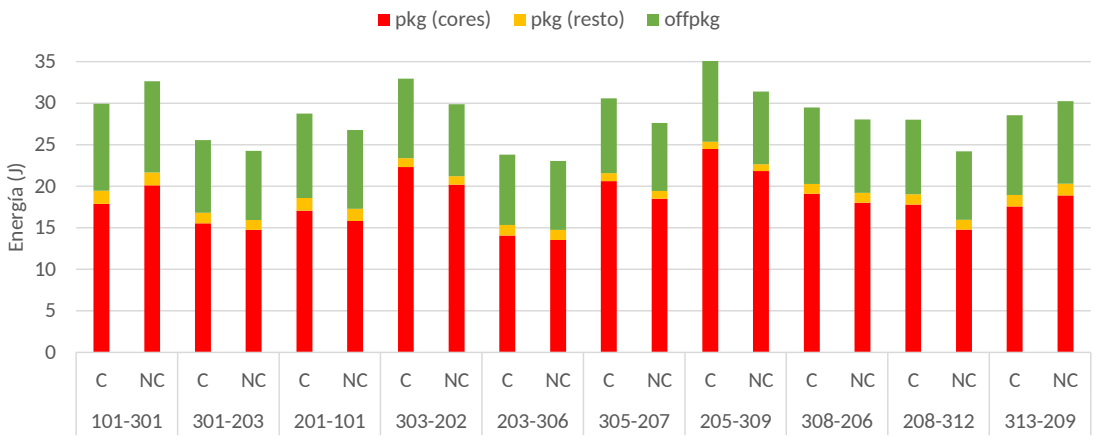

Figura 5.6: Energía presentada por parejas de distintas aplicaciones

de tipos de instrucciones, y, en segundo lugar con la interferencia en los recursos compartidos. Una política de ubicación inteligente debe tener en cuenta estas características para maximizar las prestaciones y no apostar exclusivamente por situar juntos hilos de la misma aplicación.

Consumo energético

La figura 5.6 muestra la energía media consumida cuando en el sistema se ejecutan las parejas estudiadas.

Tal como en la figura 5.3, se puede caracterizar las parejas desde el punto de vista energético como *Combined-Friendly* (101-301 y 313-209), y *Non-Combined-Friendly* (todas las demás parejas). Al contrario que en la figura 5.3, no aparecen parejas *Location-Insensitive*.

En general, comparando las figuras 5.6 y 5.4 se observa que para una pareja determinada, la política con mayor IPC presenta también el mayor consumo y viceversa. Este comportamiento es similar al discutido en el análisis de la figura 5.3, pero hay algunas excepciones. Por ejemplo, en la pareja 201-101, *non-combined* ofrece tanto mayor IPC como menor consumo energético. En menor medida esta situación también se da en la pareja 301-203. Por último, en la pareja 208-312 el IPC de las distribuciones es similar, pero el consumo energético es claramente menor en *non-combined*.

Hemos revisado las características de las aplicaciones que componen esas parejas excepcionales y el factor común es que en todas ellas una de las aplicaciones presenta una de las mayores tasa de fallos en L2 (312: 70,8 %, 301: 56,1 %, 201: 55,6). Probablemente, esto implica un ralentizamiento de la aplicación correspondiente, que es compensado en el IPC medio por la mejora de las prestaciones de la pareja. Pero, por otro lado, el mencionado ralentizamiento causa a su vez una mejora en el consumo energético medio.

Este tipo de comportamientos pueden ser aprovechados por un planificador para optimizar tanto consumo energético como prestaciones al mismo tiempo. En este TFM, se apunta que la detección de estas oportunidades está correlacionada con altas tasas de fallos en L2, pero es necesario añadir técnicas que consideren las perdidas de prestaciones que una política excesivamente centrada en el consumo energético pueda causar. La propuesta de este tipo de técnicas se deja para trabajos posteriores.

CAPÍTULO 6
Conclusiones y trabajo futuro

En este TFM se ha realizado un estudio sobre el impacto de las políticas de ubicación de los hilos de las aplicaciones paralelas en sistemas móviles basados en procesadores SMT. Para llevar a cabo este estudio, se han empleado los benchmarks incluidos en la suite Geekbench y se han realizado experimentos utilizando cargas de trabajo multiprograma en un sistema Intel NUC diseñado para el segmento de mercado móvil.

Los resultados del estudio ponen de manifiesto que la decisión de cómo asignar hilos a los núcleos del procesador tiene un efecto considerable en las prestaciones y el consumo energético del sistema. Se han evaluado dos políticas de ubicación: la política *combined* que ubica en cada núcleo físico del procesador hilos de distintas aplicaciones paralelas y la política *non-combined* que coloca hilos de la misma aplicación en el mismo núcleo físico.

Los hallazgos del estudio revelan que la elección óptima entre ambas políticas depende de varios factores, incluyendo métricas relacionadas con la microarquitectura del procesador, como la precisión del predictor de saltos o las tasas de fallos en las caches de primer y segundo nivel, así como las características específicas de las aplicaciones, como el porcentaje de instrucciones de larga latencia, por ejemplo las instrucciones de coma flotante o criptográficas. Este TFM realiza un análisis detallado de estas métricas y características para encontrar su correlación con las prestaciones medidas en términos de instrucciones por ciclo y del consumo energético del sistema.

En particular, se observa que *combined* mejora las prestaciones cuando los hilos de diferentes aplicaciones presentan un comportamiento complementario en lo que respecta a la utilización de los recursos compartidos de un núcleo físico y la ejecución de instrucciones de alta latencia. En muchos casos, los beneficios de este comportamiento complementario superan a los obtenidos mediante la estrategia *non-combined*, que asegura que los hilos de una misma aplicación paralela compartan el mismo núcleo físico.

En relación al consumo energético, en general, se observa una correlación entre el IPC y el consumo (a mayor IPC, mayor consumo). No obstante, se identifican excepciones en situaciones donde la política con un mayor rendimiento también conlleva un menor consumo energético. La causa subyacente de este último comportamiento parece radicar en las elevadas tasas de fallos en la caché de nivel 2 de una de las aplicaciones en ejecución, lo que contribuye a una reducción en el consumo de energía.

En resumen, el análisis de los resultados demuestra que la planificación de la ubicación de hilos en un sistema multinúcleo SMT con el objetivo de incrementar las prestaciones y reducir el consumo energético requiere de información de la carga disponible

en tiempo de ejecución que permite identificar la complementariedad de los hilos de las aplicaciones. Este trabajo ha identificado algunas métricas y características de las aplicaciones disponibles en tiempo de ejecución y ha analizado las bondades de políticas de ubicación en función de esas métricas y carácteristicas. El análisis puede resultar útil para diseñadores de sistemas, desarrolladores de aplicaciones y administradores que busquen optimizar las prestaciones y la eficiencia energética en entornos multinúcleo SMT para sistemas móviles.

Como trabajo futuro, pretendemos explorar el diseño de nuevas políticas de ubicación que midan dinámicamente las métricas del sistema y las características de las aplicaciones para variar las ubicaciones a medida que cambia el comportamiento de la carga. Estas políticas deben poder actuar satisfactoriamente en situaciones más realistas donde se ejecutan varias aplicaciones paralelas junto con aplicaciones secuenciales. Uno de los objetivos principales de esas políticas será aprovechar las oportunidades donde mejorar tanto las prestaciones como el consumo, optimizando la eficiencia energética.

References

[1] WikiChip, "Tiger Lake - Microarchitectures - Intel," https://en.wikichip.org/wiki/intel/microarchitectures/tiger_lake, online: accedido 01/09/2023.

[2] ——, "Willow Cove - Microarchitectures - Intel," https://en.wikichip.org/wiki/intel/microarchitectures/willow_cove, online: accedido 01/09/2023.

[3] K. N. Khan, M. Hirki, T. Niemi, J. K. Nurminen, and Z. Ou, "RAPL in Action: Experiences in Using RAPL for Power Measurements," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 3, no. 2, mar 2018.

[4] D. M. Tullsen, S. J. Eggers, and H. M. Levy, "Simultaneous multithreading: Maximizing on-chip parallelism," in *Proceedings of the 22nd annual international symposium on Computer architecture*, 1995, pp. 392–403.

[5] J. Feliu, J. Sahuquillo, S. Petit, and J. Duato, "L1-bandwidth aware thread allocation in multicore SMT processors," in *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques*, 2013, pp. 123–132.

[6] M. Navarro, L. Pons, and J. Sahuquillo, "Hy-Sched: A Simple Hyperthreading-Aware Thread to Core Allocation Strategy," *IEEE Computer Architecture Letters*, vol. 20, no. 1, pp. 26–29, 2021.

[7] W. Lee, J. Lee, B. K. Park, and R. Y. C. Kim, "Microarchitectural Characterization on a Mobile Workload," *Applied Sciences*, vol. 11, no. 3, 2021.

[8] J. Poole, "Geekbench 5," https://www.geekbench.com/blog/2019/09/geekbench-5/, sep 2019, online: accedido 01/09/2023.

[9] "ARM DS-5 Streamline User Guide," https://static.docs.arm.com/dui0482/w/DUI0482W_streamline_user_guide.pdf, online: accedido 02/03/2020.

[10] S. Banerjee and L. K. John, "Characterization of Smartphone Governor Strategies," in *Euro-Par 2018: Parallel Processing: 24th International Conference on Parallel and Distributed Computing*, 2018, pp. 120–134.

[11] S. Albers and A. Antoniadis, "Race to Idle: New Algorithms for Speed Scaling with a Sleep State," *ACM Trans. Algorithms*, vol. 10, no. 2, feb 2014.

[12] Y. Wang, V. Lee, G.-Y. Wei, and D. Brooks, "Predicting new workload or cpu performance by analyzing public datasets," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 15, no. 4, pp. 1–21, 2019.

[13] Primate Labs, "Geekbench 3," http://primatelabs.com, online: accedido 02/03/2020.

[14] "Standard Performance Evaluation Corporation (SPEC)," http://www.spec.org/, online: accedido 01/09/2023.

[15] D. Bautista Rayo, "Diseño e implementación de planificadores con ahorro energético para tareas de tiempo real estrictas y no estrictas en procesadores multicore multihilo," 2011.

[16] M. Navarro Edo, "Estrategias de ubicación de aplicaciones en núcleos smt en procesadores intel," 2021.

[17] "Placa Intel® NUC 11 Pro NUC11TNBv5," https://www.intel.la/content/www/xl/es/products/sku/205592/intel-nuc-11-pro-board-nuc11tnbv5/specifications.html, online: accedido 01/09/20203.

[18] "Next Unit of Computing," https://www.intel.com/content/www/us/en/products/details/nuc.html, online: accedido 01/09/20203.

[19] V. Weaver, "Reading RAPL energy measurements from Linux," https://web.eece.maine.edu/~vweaver/projects/rapl/, online: accedido 01/09/2023.

[20] V. M. Weaver, "Linux perf_event features and overhead," in *The 2nd international workshop on performance analysis of workload optimized systems, FastPath*, vol. 13, 2013, p. 5.

[21] J. Gray, P. Sundaresan, S. Englert, K. Baclawski, and P. J. Weinberger, "Quickly Generating Billion-Record Synthetic Databases," *SIGMOD Rec.*, vol. 23, no. 2, p. 243–252, may 1994.

[22] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition (CVPR 2001)*, vol. 1, 2001, pp. I–I.

[23] E. Reinhard and K. Devlin, "Dynamic range reduction inspired by photoreceptor physiology," *IEEE Transactions on Visualization and Computer Graphics*, vol. 11, no. 1, pp. 13–24, 2005.

[24] "Geekbench 5 CPU Workloads," https://www.geekbench.com/doc/geekbench5-cpu-workloads.pdf, nov 2019, online: accedido 01/09/2023.