UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Caracterización de prestaciones y análisis consumo energético en un procesador ARM Thunder X2

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Calero Quintana, Ibai

Tutor/a: Petit Martí, Salvador Vicente

Cotutor/a: Sahuquillo Borrás, Julio

CURSO ACADÉMICO: 2022/2023

Resum

En l'actualitat, l'eficiència energètica és de gran importància en tot tipus de dispositius, des de servidors fins a supercomputadors, passant per ordinadors d'escriptori. En aquest context, en els últims anys, els sistemes basats en l'arquitectura *ARM*, tradicionalment relegats a sistemes mòbils i integrats, han guanyat una major quota de mercat en nínxols ocupats per processadors basats en l'arquitectura *x86*. Aquesta irrupció es deu al fet que els processadors *ARM* d'alt rendiment ofereixen una major eficiència energètica que la competència.

El present treball persegueix caracteritzar comportaments típics de les aplicacions i analitzar les correlacions entre diverses mètriques de rendiment i consum energètic per a servir d'ajuda en el disseny de noves propostes de planificació centrades en processadors *ARM*.

La principal conclusió de l'anàlisi dels resultats indica que l'eficiència energètica, en termes de prestacions per watt, i l'escalabilitat de les aplicacions estan altament lligades a les prestacions dels nuclis. Així, les aplicacions que pateixen una menor degradació de prestacions a causa de la contenció pels recursos compartits són també les que aconsegueixen una major eficiència en el consum energètic, a més d'una major escalabilitat. Els resultats també posen de manifest la importància de triar quins tipus d'aplicacions s'executen simultàniament, perquè les millors prestacions s'obtenen combinant aplicacions amb diferent grau d'escalabilitat. No obstant això, si es prioritza el consum d'energia i el compromís entre el consum d'energia i el temps de finalització de les aplicacions, combinar aplicacions amb una escalabilitat similar és la millor opció.

Paraules clau: Caracterització, prestacions, consum energètic, ARM, Marvell, ThunderX2, comptadors de prestacions

Resumen

En la actualidad, la eficiencia energética es de gran importancia en todo tipo de dispositivos, desde servidores hasta supercomputadores, pasando por ordenadores de escritorio. En este contexto, en los últimos años, los sistemas basados en la arquitectura *ARM*, tradicionalmente relegados a sistemas móviles y empotrados, han ganado una mayor cuota de mercado en nichos copados por procesadores basados en la arquitectura *x86*. Esta irrupción se debe a que los procesadores *ARM* de altas prestaciones ofrecen una mayor eficiencia energética que la competencia.

El presente trabajo persigue caracterizar comportamientos típicos de las aplicaciones y analizar las correlaciones entre diversas métricas de prestaciones y consumo energético, con el objetivo de servir de ayuda en el diseño de nuevas propuestas de planificación centradas en procesadores *ARM*.

La principal conclusión del análisis de los resultados indica que la eficiencia energética, en términos de prestaciones por vatio, y la escalabilidad de las aplicaciones están altamente ligadas a las prestaciones de los núcleos. Así, las aplicaciones que sufren una menor degradación de prestaciones debido a la contención por los recursos compartidos son también las que alcanzan una mayor eficiencia en el consumo energético, además de una mayor escalabilidad. Los resultados también ponen de manifiesto la importancia de elegir qué tipos de aplicaciones se ejecutan simultáneamente, pues las mejores prestaciones se obtienen combinando aplicaciones con distinto grado de escalabilidad. Sin embargo, si se prioriza el consumo de energía y el compromiso entre el consumo de energía y el tiempo de finalización de las aplicaciones, combinar aplicaciones con una escalabilidad similar es la mejor opción.

Palabras clave: Caracterización, prestaciones, consumo energético, ARM, Marvell, ThunderX2, contadores de prestaciones

Abstract

Nowadays, energy efficiency is of great importance in all types of devices, from servers to supercomputers, including desktop computers. In this context, in recent years, systems based on the *ARM* architecture, traditionally relegated to mobile and embedded systems, have gained a greater market share in niches dominated by processors based on the *x86* architecture. This breakthrough is due to the fact that high-performance *ARM* processors offer greater energy efficiency than the competition.

The present work aims to characterize typical behaviors of applications and analyze the correlations between various performance metrics and energy consumption in order to provide assistance in the design of new scheduling proposals focused on *ARM* processors.

The main conclusion of the analysis of the results indicates that the energy efficiency, in terms of performance per watt, and the scalability of the applications are highly related to the performance of the cores. Thus, the applications that suffer less performance degradation due to contention by shared resources are also those that achieve greater efficiency in energy consumption, as well as greater scalability. The results also show the importance of choosing which types of applications run simultaneously, since the best performance is obtained by combining applications with different degrees of scalability. However, if power consumption and the trade-off between power consumption and application completion time are prioritized, then combining applications with similar scalability is the best option.

Key words: Characterization, performance, energy consumption, ARM, Marvell, ThunderX2, performance counters

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

1.1 Descripción del problema

Las transiciones digital y energética son esenciales para lograr una economía sostenible y constituyen un pilar fundamental de la nueva estrategia industrial a nivel global. Ambas, además, están estrechamente relacionadas entre sí en varios aspectos, incluyendo una preocupante paradoja. Por un lado, se necesitan centros de proceso de datos (CPD) cada vez más potentes para apoyar estas transiciones y, por otro lado, el aumento vertiginoso de la cantidad de datos y de la demanda de servicios soportados por dichos sistemas, implican un riesgo de incremento desmesurado del consumo energético. Es por ello que necesitamos desarrollar CPDs que sean mucho más eficientes energéticamente que los existentes.

Uno de los ámbitos donde se pueden lograr mejoras en la eficiencia es a nivel de procesador, lo cual ha adquirido especial relevancia tanto en la industria como en el ámbito académico en los últimos años. Se ha pasado de considerar importante únicamente las prestaciones de un procesador a valorar también su eficiencia.

Tradicionalmente, la importancia de la eficiencia energética se ha centrado en dispositivos alimentados por batería, como los dispositivos móviles, dado que la mejora de la eficiencia energética en estos dispositivos es clave, al ser la duración de la batería una de las características principales a valorar por el consumidor. Sin embargo, en la actualidad, la eficiencia energética es de gran importancia en todo tipo de dispositivos, desde servidores hasta superordenadores, pasando por ordenadores de escritorio.

Entre las opciones para mejorar la eficiencia, por un lado encontramos un incremento del uso de procesadores basados en la arquitectura *ARM*. Estos procesadores han sido asociados tradicionalmente con un consumo y prestaciones más reducidos, limitándose principalmente a ámbitos como el de los dispositivos móviles. Por otro lado, los equipos personales y servidores han utilizado mayoritariamente procesadores basados en la arquitectura *x86*, que suelen tener un mayor consumo y prestaciones, pero una menor eficiencia.

No obstante, esta situación está cambiando a medida que los procesadores *ARM* se vuelven más competitivos en términos de prestaciones. Un ejemplo de esta mejora en las prestaciones se puede observar en el ámbito de los superordenadores. Existen varias clasificaciones para estos, como *TOP500*, que evalúa las prestaciones en base a los PFlop/s de cada superordenador; *Green500*, que se basa en la eficiencia energética; y *HPCG*, que también evalúa las prestaciones, pero a diferencia del *TOP500*, el *benchmark* que utiliza simula una carga más realista, con un mayor número de accesos a memoria, entre otros aspectos. En la actualidad, el superordenador más potente del mundo, según la lis-

ta *HPCG* y segundo según *TOP500*, es el *Fugaku*, que utiliza procesadores *A64FX* basados en *ARM*, poniendo de manifiesto el potencial de *ARM*, no solo en el bajo consumo, sino también para sistemas en los que se requieren prestaciones elevadas.

Otra opción para mejorar la eficiencia energética consiste en el uso de arquitecturas heterogéneas. Estas arquitecturas utilizan diferentes tipos de núcleos o *cores*, algunos con un consumo y prestaciones más elevados y otros con menor consumo y prestaciones. Esto contrasta con las arquitecturas homogéneas, que utilizan un solo tipo de núcleo, como la mayoría de los equipos personales y servidores en la actualidad, incluido el utilizado en este trabajo. Las arquitecturas heterogéneas están ganando popularidad debido a la potencial mejora en la eficiencia energética que ofrecen. Un ejemplo de esto es la arquitectura *big.LITTLE*, introducida en 2011 por *ARM*. Además, en 2021, *Intel* anunció la generación de procesadores *Alder Lake*, que implementa una arquitectura heterogénea con núcleos de altas prestaciones y núcleos eficientes.

A pesar del auge de las arquitecturas heterogéneas, no se observan muchas propuestas para arquitecturas homogéneas, aunque estas son mayoría en servidores y equipos personales. Por esta razón, el presente trabajo se centra en obtener métricas que puedan ser utilizadas en este tipo de procesadores para planificar las aplicaciones que se ejecutan en un sistema, ya sea maximizando las prestaciones, la eficiencia o logrando un equilibrio entre ambas. Para ello, se realiza un estudio y caracterización del comportamiento de diferentes aplicaciones en un procesador *ARM*.

El principal objetivo del estudio es caracterizar comportamientos típicos de las cargas *SPEC CPU* y encontrar correlaciones entre métricas de prestaciones y consumo energético que sean de ayuda para el diseño de nuevas propuestas de planificación centradas en procesadores *ARM*.

Los resultados muestran que la eficiencia energética, en términos de prestaciones por vatio, y la escalabilidad de las aplicaciones están altamente ligadas a las prestaciones de los núcleos. Así, las aplicaciones que sufren una menor degradación de prestaciones debido a la contención por los recursos compartidos son también las que alcanzan una mayor eficiencia en el consumo energético, además de una mayor escalabilidad. Los resultados también ponen de manifiesto la importancia de elegir qué tipos de aplicaciones se ejecutan simultáneamente, pues las mejores prestaciones se obtienen combinando aplicaciones con distinto grado de escalabilidad. Sin embargo, si se prioriza el consumo de energía y el compromiso entre el consumo de energía y el tiempo de finalización de las aplicaciones (por ejemplo, el *Energy-Delay Product* o *EDP*), combinar aplicaciones con una escalabilidad similar es la mejor opción.

1.2 Objetivos

Los principales objetivos del presente trabajo son los siguientes:

- Analizar los distintos contadores de consumo energético en un procesador *ARM* de última generación y recoger medidas con los mismos.
- Analizar eventos monitorizables por contadores de prestaciones que puedan ser útiles para estudiar el comportamiento de las aplicaciones.
- Determinar qué métricas se pueden utilizar en tiempo de ejecución con el objetivo de identificar el comportamiento de una aplicación. A partir de los resultados obtenidos se identifican métricas que pueden ser usadas en tiempo de ejecución para determinar la mejor planificación, dependiendo de si el objetivo es reducir el con-

sumo por debajo de un umbral, mejorar el *performance per watt* u obtener las mejores prestaciones sin importar el consumo.

- Familiarizarse con *Stratus*, el *manager* de aplicaciones utilizado para realizar los experimentos y otras herramientas usadas en el Grupo de Arquitecturas Paralelas (GAP).
- Modificar *Stratus* para adaptarlo a las necesidades del presente TFG.

1.3 Estructura de la memoria

El resto del trabajo se organiza como sigue. En el Capítulo 2 se discute el estado del arte. El Capítulo 3 presenta el gestor *Stratus* y se explican las modificaciones realizadas en este TFG. En el Capítulo 4 se describe el entorno experimental, formado por un ordenador con el procesador **Thunder X2**, los contadores de prestaciones, los *benchmarks* y las herramientas y metodologías utilizadas en los experimentos. A continuación, en el Capítulo 5 se realiza la caracterización de los *benchmarks* utilizados y se estudia su comportamiento al ejecutarlos en parejas y variando el número de instancias de cada uno. Además, se proponen métricas que podrían utilizarse por un planificador tanto para maximizar las prestaciones como la eficiencia. Finalmente, el Capítulo 6 constituye el último capítulo del TFG en el que se presentan las principales conclusiones y la relación de este trabajo con los estudios cursados.

CAPÍTULO 2
Estado del arte

En este capítulo se presenta, en primer lugar, el estado del arte en lo que respecta a técnicas *hardware* para mejorar la eficiencia energética en los procesadores actuales. En segundo lugar, se resumen algunos trabajos que proponen políticas de planificación orientadas a mejorar la eficiencia energética de los sistemas computacionales. Entre las diversas propuestas, se introduce con un mayor nivel de detalle la política de planificación actualmente disponible en el núcleo de *Linux*. Finalmente, se exploran varios Trabajos Fin de Grado (TFG) relacionados con el tema tratado.

2.1 Técnicas *hardware* para mejorar la eficiencia energética

2.1.1. DFS y DPM

Muchas de las propuestas centradas en la planificación de tareas para mejorar la eficiencia energética utilizan técnicas como *Dynamic voltage and frequency scaling (DVFS)* y *Dynamic Power Management (DPM)*. *DVFS* [17] es una técnica que permite ajustar, en tiempo de ejecución, el voltaje y la frecuencia de los núcleos de un procesador, permitiendo reducir el consumo a costa de las prestaciones. *DVFS* está presente en la mayoría de procesadores hoy en día y puede estar implementado a nivel de núcleo o a nivel de procesador. A nivel de núcleo ofrece una gran flexibilidad, al permitir controlar el voltaje y la frecuencia de cada núcleo de forma independiente. Sin embargo, esto último implica una mayor complejidad *hardware*. Por otro lado, *DPM* [19] es una técnica complementaria que permite apagar diversos componentes del sistema, como los propios núcleos, cuando no se están usando, con el consiguiente ahorro energético.

2.1.2. Procesadores heterogéneos

Además de las mencionadas técnicas, los fabricantes de procesadores han introducido durante los últimos años nuevas arquitecturas de procesadores multinúcleo con núcleos heterogéneos o asimétricos. Es decir, procesadores que implementan varios tipos de núcleos. Así, en 2011 *ARM* anunció la arquitectura *big.LITTLE*, mientras que en 2021 *Intel* introdujo en el mercado los procesadores con arquitectura *Alder Lake*. En ambos casos, los procesadores incluyen dos tipos de núcleos: por un lado, núcleos diseñados para obtener las mayores prestaciones posibles (núcleos *big* o *P* según terminología de *ARM* o *Intel*, respectivamente); y por otro lado, núcleos cuyo principal objetivo es la eficiencia energética (núcleos *LITTLE* o *E*), incluso a costa de las prestaciones.

En general, las arquitecturas heterogéneas resultan más eficientes debido a las características y limitaciones de las aplicaciones actuales. Por ejemplo, para muchas aplicacio-

nes, los núcleos *P* no son capaces de ofrecer un incremento de prestaciones significativo con respecto a los núcleos *E*; mientras que, para esas mismas aplicaciones, estos últimos núcleos proporcionan un ahorro energético considerable. Mediante la selección del núcleo adecuado, según las características de cada aplicación, se consigue una mejora de la eficiencia energética global del sistema.

2.2 Políticas de planificación

2.2.1. Propuestas académicas

La eficiencia energética de los sistemas computacionales genera cada vez más interés, tanto en la industria como en el ámbito académico. Es por ello que hay un gran número de estudios y trabajos sobre el desarrollo de planificadores que la tienen en cuenta. A continuación, se comentan algunos de ellos.

En [16], el objetivo principal es mejorar tanto las prestaciones como la eficiencia energética mediante la reducción de la contención en el acceso a los recursos compartidos. Para ello se modifica el planificador de *Linux* a fin de que considere el comportamiento de las aplicaciones y detecte cuál es el recurso crítico en cada momento mediante contadores *hardware*. Con esta información se obtienen los denominados *activity vectors*, que representan la utilización de los recursos de cada aplicación y son usados por el planificador para elegir las aplicaciones a ejecutar y asignarlas a los núcleos del sistema, minimizando la contención.

Otoom et al. [13] proponen un planificador que trata de prevenir el sobrecalentamiento de núcleos individuales. Mediante la prevención del sobrecalentamiento, se evita la activación de los mecanismos de reducción de frecuencia y prestaciones (*thermal throttling*) implementados en los procesadores actuales. La propuesta se basa en el hecho de que un mayor consumo energético implica una mayor disipación de calor y, por tanto, una mayor temperatura. Teniendo esto en cuenta, se distribuyen las aplicaciones balanceando el consumo energético entre los núcleos, obteniendo un gradiente térmico más uniforme y evitando *hot spots*.

En [18], se propone una técnica para gestionar el consumo energético de forma global que utiliza, entre otros, mecanismos como *DVFS*. El objetivo de la propuesta es obtener las máximas prestaciones para un presupuesto energético determinado. En este trabajo se propone usar modelos basados en aprendizaje por refuerzo para abordar la alta complejidad que implica la gestión de energía en múltiples núcleos.

Enfocado a procesadores *ARM* heterogéneos, en [14] se plantea un planificador centrado en aplicaciones de navegación web que utiliza *DVFS* para regular prestaciones y energía. El planificador propuesto realiza predicciones sobre el tiempo de carga y consumo de energía de una página web considerando diferentes tipos de núcleos y niveles de frecuencia. El objetivo es determinar en qué núcleo se debe planificar la carga de la página web de manera que se cumplan restricciones temporales, mientras que se minimiza el consumo.

A diferencia de la mayoría de las propuestas recientes del estado del arte, centradas en arquitecturas heterogéneas para sistemas de bajo consumo, este trabajo se enfoca a sistemas de altas prestaciones basados en procesadores *ARM* con núcleos homogéneos. Este tipo de procesadores representan una alternativa energéticamente eficiente en el ámbito de la computación de altas prestaciones con respecto a arquitecturas más asentadas como las basadas en procesadores *Intel* o *AMD*. En este contexto, este trabajo representa un primer paso en el estudio de las métricas más adecuadas para caracterizar el com-

portamiento de las aplicaciones en tiempo de ejecución, con el objetivo de utilizarlas en trabajos subsiguientes para planificar la ejecución de las aplicaciones, mejorando las prestaciones, la eficiencia energética y la equidad en el sistema.

2.2.2. Linux EAS

Actualmente, el planificador del núcleo del sistema operativo *Linux* incorpora una política que tiene en cuenta el consumo de energía en la selección de las aplicaciones a ejecutar y su asignación a los núcleos del sistema. Esta política es conocida como *Linux Energy Aware Scheduling* (EAS) [3]. *EAS* se introdujo en la versión 5.0 del *kernel* de *Linux* y es capaz de predecir el impacto de las decisiones del planificador en las prestaciones y en la energía consumida. Se ha diseñado específicamente para arquitecturas heterogéneas y tiene como objetivo minimizar el consumo de energía por instrucción sin afectar significativamente a las prestaciones.

EAS hace uso de un *framework* denominado *Energy Model (EM)* [4], que permite al *kernel* obtener información sobre el consumo de energía de los distintos componentes del sistema para determinar en qué núcleo o núcleos se ejecuta cada aplicación. Este *framework* es especialmente útil cuando se va a planificar una tarea y existen varios núcleos disponibles, permitiendo elegir el núcleo que proporcione un menor consumo sin impactar significativamente en las prestaciones. Por otro lado, también es capaz de detectar situaciones donde ejecutar una aplicación determinada en núcleos *E* pueda causar grandes pérdidas de prestaciones que no justificarían el potencial ahorro energético.

Para modelar las prestaciones de los diferentes tipos de núcleos, *EAS* define una nueva métrica denominada *capacidad*. La capacidad representa la cantidad de cómputo que puede realizar un núcleo a su máxima frecuencia comparada con la del núcleo más potente del sistema [3]. *EAS* utiliza esta métrica conjuntamente con el consumo para evaluar la eficiencia energética.

2.3 TFG relacionados

En cuanto a trabajos de fin de grado relacionados con el este trabajo, en [22], se propone una política de asignación de aplicaciones a núcleos para procesadores *x86* que soportan *SMT*. El objetivo de la propuesta es asignar parejas de aplicaciones a los núcleos *SMT* de manera que interfieran entre ellas lo menos posible, mejorando así sus prestaciones. Para ello, se caracterizan distintas aplicaciones tanto individualmente como distintas mezclas, identificando así las mejores parejas.

Por otra parte, en [21] el objetivo es el diseño de una política de particionado de la *LLC* (la caché de último nivel) usando la tecnología *Cache Allocation Technology*, presente en algunos procesadores *Intel*, para mejorar las prestaciones de cargas multiprograma. La propuesta caracteriza la relación de varias aplicaciones con la *LLC* e identifica aquellas cuyas prestaciones se ven más afectadas por la compartición de esta, protegiéndolas en tiempo de ejecución.

El presente trabajo se relaciona con los dos expuestos en el *framework* común para la gestión de aplicaciones y monitorización de contadores de prestaciones. Sin embargo, en comparación con los mecionados trabajos, el foco de este TFG está centrado en la mejora de la eficiencia energética además de las prestaciones.

CAPÍTULO 3
Gestión y monitorización de aplicaciones

En este capítulo se describe el *framework* utilizado para gestionar la ejecución y monitorización de las aplicaciones estudiadas. Además, se comenta cómo este *software* mide eventos de prestaciones y consumos energéticos en el sistema. Finalmente, se explican las modificaciones realizadas sobre el *framework* que han permitido completar con éxito el presente TFG.

3.1 *Stratus*

El *manager Stratus* [20], desarrollado en el Grupo de Arquitecturas Paralelas de la Universidad Politécnica de Valencia, es un *framework* para facilitar la ejecución y control de aplicaciones en diversos tipos de sistemas, entre ellos, aquellos que cuentan con procesadores **ThunderX2**. Además de gestionar las aplicaciones, *Stratus* soporta la monitorización de contadores de prestaciones *hardware* por medio de la utilidad *perf* [5].

Stratus es capaz de lanzar una serie de aplicaciones especificadas por el usuario y monitorizar el estado del sistema durante la ejecución de estas. Periódicamente monitoriza el valor de los eventos que el usuario especifique. Estos eventos (detallados en la Sección 4.3) son monitorizados por medio de contadores *hardware*, con los que *Stratus* interactúa por medio de la utilidad *perf* de *Linux*. Además de los eventos indicados por el usuario, también recopila información sobre los consumos (en vatios) y voltajes de distintos componentes del procesador (núcleos, *SoC*, L3 y estructuras de memoria SRAM internas), a los que accede con ayuda de la utilidad *tx2mon* (ver Sección 3.2).

Los principales parámetros del *Stratus* que se han utilizado durante la realización de este trabajo han sido:

Aplicaciones a ejecutar. Nombre de las aplicaciones a ejecutar.

Número máximo de instrucciones. Número de instrucciones que tiene que ejecutar cada aplicación hasta terminar.

Núcleos ocupados. Identificación de los núcleos donde se ejecuta cada aplicación.

Duración de los intervalos. Determina cada cuánto tiempo se miden los eventos que se están monitorizando.

Número de intervalos. Número máximo de intervalos a ejecutar antes de finalizar la ejecución de las aplicaciones.

```
<%include file="applications.mako"/>
<%include file="instr_60s.mako"/>

<%!
lc = {0:56, 1:57, 2:58, 3:59}
%>

tasks:
  % for app in apps:
  - app: *${app}
    max_instr: *${app}_mi
    cpus: [${lc[loop.index]}]
  % endfor

cmd:
    ti: 0.1
    mi: 4000
    event: ["INST_RETIRED,cycles,INST_SPEC"]
```

Listing 3.1: Ejemplo de fichero *main.mako*.

Eventos a monitorizar. Nombres de los eventos que se van a monitorizar en cada intervalo.

Los ficheros en los que se especifica la configuración son:

***applications.mako*.** Fichero en formato *YAML* [6]. Contiene una entrada por cada aplicación soportada por *Stratus*. Estas entradas proporcionan la información necesaria para iniciar la ejecución de la aplicación, como el comando a ejecutar para iniciarla o los directorios que contienen ficheros que la aplicación utiliza durante su ejecución (si procede).

***instr_60s*.** Fichero en formato *YAML* que contiene una entrada por aplicación e indica el número máximo de instrucciones que cada una puede ejecutar.

***spec-06-17.yaml*.** Fichero en formato *YAML* que contiene los nombres de las aplicaciones a ejecutar.

***Main.mako*.** Fichero principal de configuración con la mayoría de los parámetros de las ejecuciones, como los eventos que se van a medir, los núcleos en los que se va a ejecutar cada aplicación, la duración de los intervalos en los que se medirán los contadores, etc. Desde aquí se importan los ficheros mencionados anteriormente (*applications.mako* e *instr_60s*). Esto permite indicar tanto el número máximo de instrucciones a ejecutar como la información necesaria para ejecutar cada aplicación.

***launch.bash*.** *Script* que se invoca para iniciar *Stratus*. Recibe como parámetro la ruta del fichero *spec-06-17.yaml* con las aplicaciones a ejecutar. Se encarga, entre otras cosas, de crear directorios temporales y construir el nombre de los archivos de resultados. Finalmente, inicia la ejecución del *manager Stratus*.

En el Listado 3.1 se observa uno de los ficheros *main.mako* utilizados durante los experimentos. Al comienzo del mismo, el comando *include* permite hacer uso de las *keys* definidas en los ficheros *applications.mako* y *instr_60s*. Además de esto, podemos observar cómo se configuran algunos parámetros para la ejecución de las aplicaciones. Entre ellos cabe destacar:

***app*:** proporciona a cada aplicación la información necesaria para su ejecución (como el comando para lanzarla). En este caso **${app}* obtiene la *key* definida en *applications.mako* para la aplicación cuyo nombre está en la variable *app*.

***cpus*:** indica en qué núcleo se ejecutará cada aplicación. En este caso el número de cada núcleo se obtiene a partir del vector *lc* definido anteriormente.

***ti*:** duración de los intervalos. En este caso 0,1 segundos.

***mi*:** máximo número de intervalos.

***event*:** eventos a monitorizar.

Tras la ejecución de las aplicaciones, *Stratus* genera cuatro ficheros con información sobre diferentes aspectos de la ejecución. El nombre de estos resulta de la concatenación de los nombres de las aplicaciones ejecutadas junto con un sufijo único para cada archivo. Estos cuatro ficheros son:

***0.csv*:** contiene los valores de los eventos a monitorizar para cada aplicación, intervalo por intervalo, hasta que finalizan todas.

***tot.csv*:** contiene el total de ocurrencias de los eventos monitorizados para cada aplicación, desde su inicio hasta que todas las aplicaciones terminan. En caso de que una aplicación termine antes que el resto y sea relanzada, las ocurrencias de los eventos después del relanzamiento también se incluyen.

***fin.csv*:** contiene el total de ocurrencias de los eventos monitorizados para cada aplicación, desde su inicio hasta que termina cada una. En caso de que una aplicación termine antes que el resto y sea relanzada, las ocurrencias de los eventos a partir del relanzamiento no son tenidos en cuenta.

***times.csv*:** contiene estadísticas sobre el uso de la *CPU* durante la ejecución de cada aplicación.

3.2 Monitorización de eventos y consumo energético

Con respecto a la monitorización de la ejecución, en cada intervalo se distingue entre la monitorización de los eventos indicados por el usuario en la configuración y la monitorización de los consumos del procesador. A continuación, se profundiza en el soporte de cada uno de estos tipos de monitorización.

3.2.1. Eventos

Los eventos que es posible monitorizar varían en función del procesador. En el caso del procesador de nuestro sistema, el documento [10] presenta algunos de los eventos relacionados con las prestaciones. Estos eventos son medidos mediante contadores de prestaciones *hardware*, que son accesibles a nivel de usuario por la utilidad *perf*. Esta herramienta abstrae las diferencias de *hardware* de la *CPU* en la monitorización de prestaciones y soporta versiones del núcleo de *Linux* iguales o superiores a la 2.6.

Stratus no interactúa con *perf* directamente. En su lugar hace uso de una librería llamada *libminiperf*, que simplifica la interacción con los contadores. Las funciones más relevantes que exporta esta librería son las que permiten configurar los contadores de prestaciones y leer sus valores. Estas funciones se muestran en el Listado 3.2 y se describen a continuación:

```
struct perf_evlist* setup_events(const char *monitor_target,
                                 const char *events,
                                 const char *type);

int create_perf_stat_counter(struct perf_evsel *evsel,
                             struct perf_stat_config *config,
                             struct target *target);

static int read_counter(struct perf_evlist *evsel_list,
                        struct perf_evsel *counter);

int perf_evsel__read_counter(struct perf_evsel *evsel,
                             int cpu,
                             int thread);
```

Listing 3.2: Prototipos de las funciones de *libminiperf* y *perf* utilizadas para monitorizar los eventos.

setup_events. Esta función se encarga de configurar los contadores de prestaciones para que midan los eventos que resultan de interés. Entre los parámetros que recibe encontramos una lista con los eventos a monitorizar. Los otros dos parámetros permiten configurar la granularidad de la monitorización.

La función permite monitorizar los eventos a nivel de *CPU*, *PID* y *TID*. En este trabajo se ha usado la monitorización a nivel de *PID*, excepto para los eventos relacionados con la caché L3.

Indicar una *CPU* implica que los eventos solo se monitorizarán para aplicaciones que se ejecutan en la *CPU* especificada. En cambio, monitorizar a nivel de *PID* implica que solo se cuentan las ocurrencias de los eventos cuando se ejecuta el proceso con el *PID* que estamos monitorizando.

Los eventos no tienen por qué ser necesariamente compatibles con todas estas opciones de monitorización. Por ejemplo, para los eventos de la caché L3 no es posible especificar un *PID*, pues solo soporta la monitorización a nivel de *CPU* (la causa de esta limitación se comenta en la Sección 4.3).

create_perf_stat_counter. Internamente *setup_events* realiza una invocación a la función *create_perf_stat_counter* por cada evento a monitorizar. Esta función pertenece a la utilidad *perf* y realiza la configuración de los contadores de prestaciones.

read_counter. Esta función de *libminiperf* facilita la lectura de los contadores de prestaciones *hardware* y permite leer el valor de un único contador.

perf_evsel__read_counter. Internamente *read_counter* llama a *perf_evsel__read_counter*. Esta función pertenece a la utilidad *perf* y se encarga finalmente de leer el valor del contador.

3.2.2. Consumo energético

En el caso de los consumos del procesador no disponemos de ningún evento para monitorizarlos. En su lugar, se hace uso de la utilidad *tx2mon* [7]. Este es un programa compuesto de un módulo para el núcleo de *Linux* y un componente de usuario que interactúa con este módulo. *Stratus* incorpora parte del código del componente de usuario e interactúa con el módulo del núcleo directamente.

```
static inline int
sys_perf_event_open(struct perf_event_attr *attr,
                    pid_t pid, int cpu, int group_fd,
                    unsigned long flags)
{
    int fd;
    if (attr->config == 0xD || attr->config == 0x17) {
        attr->type = 12;
        pid = -1;
    }

    fd = syscall(__NR_perf_event_open, attr, pid, cpu
                 group_fd, flags);
    ...
    return fd;
}
```

Listing 3.3: Modificación realizada en la función sys_perf_event_open.

El módulo obtiene información del *SoC management controller* (*MC*), que reporta datos tales como la temperatura actual, las frecuencias y las mediciones de potencia del procesador [7] y los hace accesibles a aplicaciones como *Stratus*.

3.3 Modificaciones realizadas

Para desarrollar este TFG ha sido necesario realizar modificaciones sobre *Stratus* con el objetivo de mejorar este gestor en el contexto del trabajo y solventar problemas que han ido apareciendo. Estas han sido las principales modificaciones:

Soporte a múltiples *PMU*. El procesador utiliza varias *PMU* (*performance monitoring units*) para medir el número de ocurrencias de los eventos deseados. Existen varias *PMU* en el procesador, que monitorizan diferentes eventos. Por un lado, hay una *PMU* en cada núcleo del procesador, que mide los eventos que ocurren en estos, como el número de ciclos o de instrucciones especuladas. También hay otras *PMU* que miden eventos a nivel de procesador y que son compartidas por todos los núcleos, como los eventos de la caché L3 o del controlador de memoria.

Cada evento lleva un asociado código, utilizado para monitorizarlo con la *PMU* correspondiente. No obstante, un mismo código puede hacer referencia a distintos eventos, dependiendo de la *PMU* que se utilice. Esto es problemático ya que *Stratus* solo permite especificar el código del evento a monitorizar y no la *PMU* a utilizar. Por ejemplo, durante la realización de este trabajo, al tratar de monitorizar eventos de la caché L3 (comunes a todos los núcleos), en lugar de utilizar la *PMU* encargada de monitorizar los eventos de la L3, *Stratus* erróneamente usaba la *PMU* del núcleo en el que se ejecutaba la aplicación, por lo que se terminaba midiendo otro evento.

Este problema se ha solucionado modificando *Stratus* para que determinados códigos de eventos, utilicen la *PMU* de la caché L3. En el Listado 3.3 se observa la modificación sobre la función *sys_perf_event_open*. Esta modificación permite leer los eventos de la caché L3 en la *PMU* correcta. *attr->config* contiene el código del evento a modificar y *attr->type* es la *PMU* tal y como se indica en la documentación de esta función [9].

La solución propuesta consiste en comprobar si el código del evento que se pretende medir es el de un evento de la caché L3. De ser así, nos aseguramos de leerlo de

```
std::vector<std::string> get_events_for_task(std::vector<std::string> events,
    const task_ptr_t &task_ptr) {
    events_to_be_logged.clear();

    if (task_ptr->id == 0) {
        events_to_be_logged = events;
    }
    else {
        events_to_be_logged = removeStringAndCommaFromVector(events, "r000d", "
            cycles");
        events_to_be_logged = removeStringAndCommaFromVector(
            events_to_be_logged, "r0017", "cycles");
    }

    return events_to_be_logged;
}
```

Listing 3.4: Implementación de la función *get_events_for_task*.

```
for (const auto &task_tpr : runlist) {
    auto events_to_setup = get_events_for_task(events_task_ptr);

    if (task_ptr->get_status() == Task::Status::exited) {
        ...
        task_ptr->task_restart_or_set_done(perf, events_to_setup);
        ...
    }
}
```

Listing 3.5: Ejemplo de invocación a la función *get_events_for_task*.

la *PMU* correcta mediante *attr->type = 12*, siendo 12 el número de la *PMU* encargada de medir los eventos de la L3. Este valor se obtiene a través del comando "`cat /sys/bus/event_source/devices/uncore_l3c_1/type`". Además, también es necesario configurar el *PID* a -1. Esto se debe a que estos eventos no pueden medirse por *PID* en nuestro procesador.

Soporte a la medición de eventos y consumos comunes a todos los núcleos. Algunos datos a monitorizar, como los relacionados con el consumo de energía o los eventos de la caché L3 o el controlador *DDR*, son comunes a todos los núcleos de un mismo procesador. *Stratus* mide esta información cada intervalo, tantas veces como aplicaciones se están ejecutando. Esto puede generar problemas cuando se monitoriza un número elevado de aplicaciones. Por ejemplo, durante la realización de los experimentos, medir el evento *L3_READ_REQUEST* ejecutando 28 aplicaciones provocaba que el equipo se apagase.

Este problema se ha solucionado modificando *Stratus* para realizar las mediciones comunes a todos los núcleos una única vez en cada intervalo, sin importar el número de aplicaciones que se ejecuten. Para ello, se ha introducido una modificación en la parte del código en la que se asigna a cada aplicación los eventos a monitorizar. La modificación introduce la función *get_events_for_task* (ver Listado 3.4). Esta función recibe el nombre de una aplicación y el nombre de los eventos a monitorizar y, a continuación, comprueba si alguno de los eventos es común a todos los núcleos del procesador (únicamente comprobamos que no sean *L3_EVENT_READ_REQ* (código 0xD) y *L3_EVENT_READ_HIT* (código 0x17)). En caso de que sea común, solo se monitoriza para una aplicación.

```
for ((REP=${INI_REP};REP<${MAX_REP};REP++)); do
    for MASK in ${MASKS[@]}; do
  ...
    # while read WL; do
    # WL=$(echo $WL | tr '\-[],' " ")
  # if [ ${#MASKS[@]} -eq 1 ]; then
  #   ID=$(join_by - ${WL[@]})
  # else
  #   ID=$(join_by - ${WL[@]})_${MASK}
  # fi


    ID=$(echo "$WL" | awk '{
      for (i = 1; i <= NF; i++) {
        count[$i]++
      }
    }
    END {
      for (word in count) {
        printf "%d-%s-", count[word], word
      }
    }')

    ...

  CONFIG=configs/${ID}.yaml
  OUT=data/${ID}_${REP}.csv
  FIN_OUT=data/${ID}_${REP}_fin.csv
  TOT_OUT=data/${ID}_${REP}_tot.csv
  TIMES_OUT=data/${ID}_${REP}_times.csv
  LOG=log/${ID}_${REP}.log
```

Listing 3.6: Fragmento de *launch.bash*.

El Listado 3.5 muestra un ejemplo de invocación a esta función. En este caso, se utiliza *get_events_for_task* para filtrar los eventos a monitorizar para cada aplicación y posteriormente devolver la lista filtrada.

Es necesario destacar que esta modificación puede provocar problemas en el procesamiento de los ficheros de resultados. Como en estos ficheros los eventos a monitorizar se ordenan en columnas, aplicar esta solución provoca que una aplicación tenga más columnas que el resto (los eventos comunes), por lo que ha sido necesario modificar los *scripts* que procesan los resultados para que lo tengan en cuenta.

Reducción de la longitud de los nombres de los ficheros de resultados. Los resultados de los experimentos se guardan en ficheros cuyo nombre completo se construye concatenando los nombres de todas las aplicaciones que se han ejecutado. Esto puede provocar que, cuando ejecuta un número elevado de aplicaciones, el nombre resultante supere los 255 caracteres (máximo soportado en el equipo utilizado [8]), causando problemas al tratar de guardar los ficheros de resultados.

Este problema se ha solucionado modificando el fichero *launch.bash*. Este *script* es el encargado de realizar algunos preparativos antes de iniciar *Stratus*. El *script* recibe como argumento el nombre del fichero con las aplicaciones a ejecutar. Después de leerlo, concatena todas las aplicaciones que se ejecutarán para construir el nombre de los ficheros de resultados.

En el Listado 3.6 se muestra la modificación realizada. *ID* es la variable que contiene el futuro nombre de los ficheros de resultados. Originalmente esta variable contiene la unión de los nombres de todas las aplicaciones (*join_by - WL[@]*), siendo *WL*

una lista con los nombres de todas las aplicaciones. En cambio, la modificación introducida hace uso de la utilidad *awk* para construir el nombre de los ficheros de resultados como el número de instancias ejecutadas de cada aplicación seguido del nombre de la aplicación una única vez. Por ejemplo, la ejecución de dos instancias de *calculix* y otras dos de *mcf* se guardará ahora en ficheros llamados *2-calculix-2-mcf* en lugar de *calculix-calculix-mcf-mcf*.

CAPÍTULO 4
Entorno experimental

En este capítulo se describe el *hardware* del sistema evaluado, los *benchmarks* estudiados, los contadores de prestaciones y métricas evaluadas y las herramientas empleadas durante los experimentos. Finalmente, se explican las técnicas empleadas para ajustar el número de instrucciones a ejecutar y las mediciones del consumo energético.

4.1 Sistema basado en procesador ThunderX2

4.1.1. Configuración del equipo

El equipo utilizado durante la realización de los experimentos utiliza el sistema operativo *CentOS Linux 7* y la versión 4.18.0 del núcleo de *Linux*. Dispone de un total de 64 GB de memoria *RAM* en 4 módulos *DIMM DDR4* 2666 MHz de 16 GB cada uno. El equipo cuenta además con 2 procesadores **ThunderX2 CN9975** de 28 núcleos físicos configurados para operar en modo *SMT-2*. En total, el sistema cuenta con 56 núcleos físicos, que pueden ejecutar hasta 112 hilos en paralelo a una frecuencia base de 1 GHz y una máxima de 2,5 GHz.

4.1.2. Arquitectura del procesador

El procesador **ThunderX2 CN9975** [11] pertenece a la serie de procesadores *multicore* de 64 bits **ThunderX2** introducida por *Cavium*. Estos procesadores están basados en la microarquitectura *Vulcan* [12] e implementan el conjunto de instrucciones *ARM v8.1*. Los procesadores de esta serie soportan *SMT-4* y pueden llegar a tener hasta 32 núcleos físicos. En la Figura 4.1 se observa el *layout* de un **ThunderX2** de 32 núcleos.

El procesador soporta hasta *2 TiB* de memoria *DDR4* 2666 MT/s y, en caso de una configuración *dual-socket* (como en la plataforma experimental), hasta 4 TiB con un ancho de banda de hasta 158.95 GB/s.

En cuanto a la jerarquía de memoria, el procesador dispone de 3 niveles de caché:

Nivel 1 (L1): compuesto por cachés privadas por núcleo con una capacidad de 64 KB, dividida en 32 KB para datos y 32 KB para instrucciones.

Nivel 2 (L2): una cache privada por núcleo con una capacidad de 256 KB compartida por datos e instrucciones.

Nivel 3 (L3): una cache compartida entre todos los núcleos con una capacidad de 32 MB particionada en *slices* de 2 MB cada uno. Se trata de una caché exclusiva que alma-

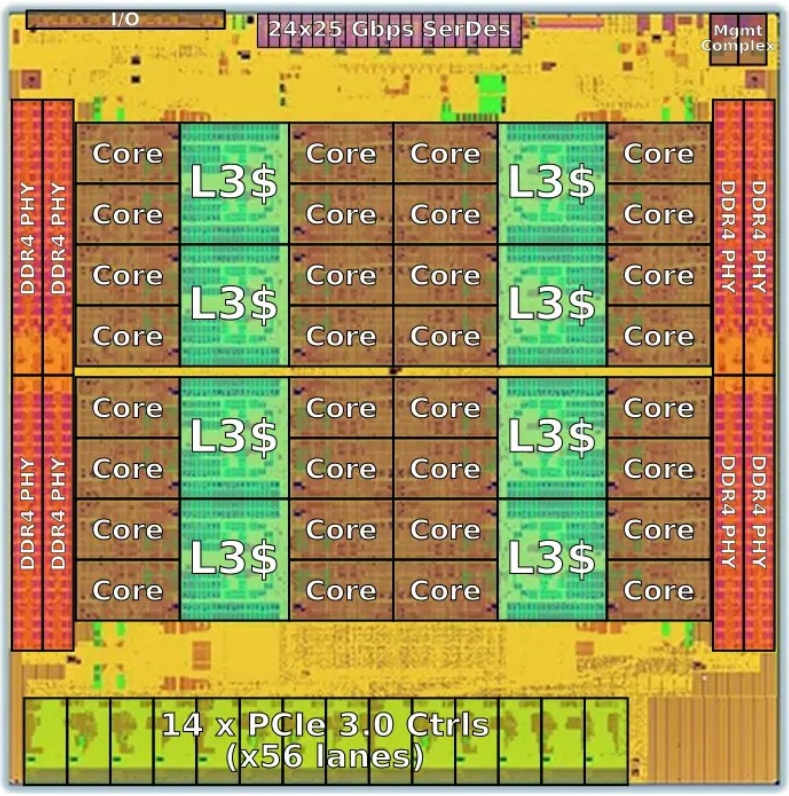


Figura 4.1: *Layout* de un procesador ThunderX2 [1].

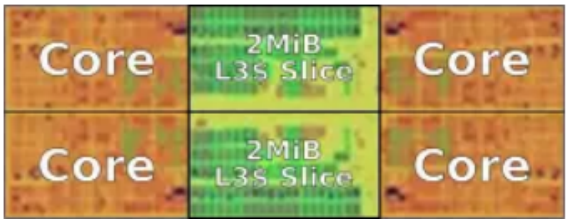


Figura 4.2: *Layout* de un *cluster* de núcleos en un procesador ThunderX2 [2].

cena los bloques que son reemplazados de la L2. En la Figura 4.2 se puede observar la distribución de la caché L3 en *slices* repartidos por cada 2 núcleos.

La microarquitectura *Vulcan* ofrece soporte al multiprocesamiento, permitiendo conectar 2 procesadores por medio de la arquitectura de interconexión *Cavium Coherent Processor Interconnect* (CCPI2). Esta proporciona un ancho de banda de 600 Gbps entre 2 procesadores y es capaz de comunicar hasta 64 núcleos y 256 hilos.

4.2 Benchmarks

En este trabajo se estudia y caracteriza, desde el punto de vista del consumo, el comportamiento de 16 *benchmarks* de la *suites SPEC CPU2006* y *SPEC CPU2017*. Estas *suites* han sido desarrolladas por la organización *SPEC (Standard Performance Evaluation Corporation)* cuyo principal objetivo es estandarizar la evaluación de las prestaciones y la eficiencia energética de los sistemas.

SPEC ha desarrollado varías *benchmark suites* en función del tipo de carga que se pretende evaluar. Actualmente existen *benchmarks* que evalúan cargas de *Cloud*, *HPC*, gráfi-

cos, etc. Pueden existir múltiples ediciones de una *benchmark suite*. Por ejemplo, la *suite* de *CPU*, que utiliza cargas de trabajo de cómputo intensivo y de altas prestaciones, dispone de las ediciones *SPEC CPU2017*, *SPEC CPU200*, y *SPEC CPU2000*.

A continuación, se describen los *benchmarks* estudiados en este trabajo, indicando, para cada uno de ellos, el lenguaje en el que ha sido programado:

***calculix* (*Fortran 90* y *C*):** *CalculiX* es un *software* utilizado para el cálculo, simulación y postprocesado de modelos de estructuras tridimensionales, como puentes o edificios.

***deepsjeng_r* (*C++*):** Versión modificada del programa *Deep Sjeng WC2008* (programa ganador del *2008 World Computer Speed-Chess*). Trata de encontrar el mejor movimiento en ajedrez utilizando técnicas como la poda alfa-beta, ordenación avanzada de movimientos y evaluación posicional, entre otras.

***exchange2_r* (*Fortran 95*):** Su función es desarrollar sudokus no triviales de 9x9. Hace un uso intensivo de la recursión, llegando hasta a 8 niveles.

***imagick_r* (*C*):** *ImageMagick* es una aplicación usada para crear, editar o convertir imágenes de mapa de bits.

***leela_r* (*C++ 2003* y *C++11*):** Se trata de un programa que juega a Go y utiliza técnicas como la estimación de posición basada en Monte Carlo y valoración de movimientos basada en clasificaciones *Elo*.

***nab_r* (*C*):** La aplicación de modelado molecular *Nucleic Acid Builder* (*NAB*) hace un uso intensivo de cálculos en coma flotante.

***namd_r* (*C++*):** *NAMD* es un programa paralelo para la simulación de grandes sistemas biomoleculares.

***povray_r* (*C++*):** *POV-Ray* es un programa de trazado de rayos de código abierto.

***cam4_r* (*Fortran90* y *C*):** Deriva de *CESM*, que es una simulación numérica del sistema terrestre que consiste en componentes atmosféricos, oceánicos, de hielo, de superficie terrestre, del ciclo de carbono y otros.

***mcf* (*C*):** *MCF* es un programa utilizado para la planificación de vehículos en empresas de transporte. Hace un uso muy reducido de aritmética de coma flotante.

***wrf_r* (*Fortran90* y *C*):** Versión 3.6.1 del *Weather Research and Forecasting Model* (*WRF*), que es un sistema de pronóstico meteorológico, de última generación a escala mesoscala diseñado para satisfacer necesidades como la de investigación atmosférica.

***roms_r* (*Fortran 2003*):** *Regional Ocean Modeling System* (*ROMS*) es un modelo oceánico, ampliamente utilizado por la comunidad científica, que tiene en cuenta la superficie libre y sigue el relieve del terreno.

***omnetpp_r* (*C++*):** El simulador *OMNeT++* lleva a cabo una simulación de una red Ethernet de 10 Gb.

***xalancmk_r* (*C++*):** Versión modificada de *Xalan-C++*, que es un procesador de *XSLT*. Está diseñado para convertir documentos *XML* a otros formatos, como puede ser *HTML*.

***blender_r* (*C* y *C++*):** *Blender* es una *suite* de creación en 3D para hacer renderizados.

***cactuBSSN_r* (*C, C++* y *Fortran*):** El *Cactus Computational Framework* hace uso del *EinsteinToolkit* para resolver las ecuaciones de Einstein en el vacío.

4.3 Contadores de prestaciones

Para la evaluación y caracterización de las aplicaciones es necesario determinar los efectos que su ejecución tiene en el sistema. Una manera de hacer esto es mediante los contadores de prestaciones *hardware*.

Los contadores de prestaciones son contadores *hardware* que permiten obtener el número de ocurrencias de determinados eventos soportados por el procesador.

Los contadores de prestaciones se agrupan *Peformance Monitoring Units* (*PMU*). El procesador dispone de una *PMU* (con 6 contadores) por cada núcleo del procesador, pudiendo monitorizar hasta 6 eventos que ocurren en estos, como el número de ciclos o instrucciones ejecutadas. Dispone también de varias *PMU* adicionales que monitorizan eventos comunes al sistema como los relacionados con la caché L3 (teniendo esta 4 contadores).

Para monitorizar los eventos deseados es necesario programar los contadores de prestaciones. En este trabajo tanto la configuración de los contadores como la lectura de estos se ha realizado mediante *Stratus* (ver Sección 3.2).

Los eventos que pueden contabilizarse dependen de la arquitectura del procesador. En el caso del procesador utilizado durante la realización de este trabajo (**ThunderX2 CN9975**) disponemos de un documento, proporcionado por el fabricante [10], en el que se listan algunos de los eventos más relevantes relacionados con las prestaciones.

Los principales eventos monitorizados durante la realización de los experimentos han sido:

***INST_RETIRED*:** número de instrucciones completamente ejecutadas.

***cycles*:** número de ciclos ejecutados por el procesador.

***L2D_CACHE_RD*:** número de accesos de lectura a la caché de nivel 2.

***L2D_CACHE_WR*:** número de accesos de escritura a la caché de nivel 2.

***L2D_CACHE_REFILL_RD*:** número de *refills*, en lecturas, en la caché 2 de datos.

***L2D_CACHE_REFILL_WR*:** número de *refills*, en escrituras, en la caché 2 de datos.

***L2D_CACHE*:** Acceso a la cache de nivel 2.

***L3_EVENT_READ_REQ*:** número de peticiones de lectura recibidas por la cache de nivel 3 (incluye tanto las lecturas como las lecturas exclusivas).

***L3_EVENT_READ_HIT* :** número de peticiones de lectura recibidas por la caché L3 que resultaron en aciertos en la L3.

Además de los eventos indicados previamente, también se han medido los consumos de los distintos componentes del procesador. Estos consumos no pueden medirse mediante los contadores de prestaciones y, en su lugar, se accede a ellos mediante el módulo del núcleo de *Linux* que incluye la utilidad *tx2mon*. *Stratus* incorpora el código del componente de usuario *tx2mon* que interactúa con el módulo para leer los siguientes consumos:

***power_cores*:** consumo de todos los núcleos del procesador.

***power_sram*:** consumo de las estructuras de memoria *SRAM* internas del procesador.

***power_mem*:** consumo de la caché L3.

***power_soc*:** consumo del *SoC*.

4.4 Métricas

A partir de los eventos monitorizados y de los consumos se han calculado varias métricas para tratar de caracterizar el comportamiento de las aplicaciones del procesador. Estas métricas son:

Instrucciones por ciclo (*IPC*): cuantifica las prestaciones de una aplicación por medio del número de instrucciones ejecutadas por cada ciclo del procesador.

$$IPC = \frac{INST\_RETIRED}{cycles}$$

L2 cache misses per kilo instructions (*MPKI_L2*): cuantifica la tasa de fallos de la caché L2 por cada 1000 instrucciones ejecutadas.

$$MPKI\_L2 = \frac{L2D\_CACHE\_REFILL\_RD + L2D\_CACHE\_REFILL\_WR}{INST\_SPEC/1000}$$

L3 cache misses per kilo instructions (*MPKI_L3*): cuantifica la tasa de fallos de la caché L3 por cada 1000 instrucciones ejecutadas.

$$MPKI\_L3 = \frac{L3\_EVENT\_READ\_REQ - L3\_EVENT\_READ\_HIT}{INST\_SPEC/1000}$$

Prestaciones por vatio (*performance per watt*): se trata de la relación entre las prestaciones de una aplicación (cuantificadas mediante el *IPC*) y el consumo del procesador.

$$Performance\ per\ watt = \frac{IPC}{total\_power}$$

Degradación de las prestaciones (*slowdown*): cuantifica la pérdida de prestaciones que experimenta una aplicación respecto a si se ejecutase ella sola en el procesador.

$$Slowdown = \left(\frac{IPC\ sola}{IPC\ medido} - 1\right) \times 100$$

Equidad (*fairness*): estima lo *justo* que está siendo el sistema. Un sistema se considera justo si todas las aplicaciones se benefician o perjudican en la misma proporción. La fórmula utilizada para cuantificar usada en este trabajo fue propuesta en [15]. Como se aprecia, considera la media y la desviación típica del *individual speedup* de las aplicaciones. La ecuación proporciona un resultado menor o igual a 1, siendo 1 cuando el sistema es *justo*, esto es, si todas las aplicaciones se degradan de igual manera.

$$Fairness = 1 - \frac{\sigma_{IS}}{\mu_{IS}}$$

Tiempo de finalización. Esta métrica se define como el tiempo que tarda en finalizar la aplicación más lenta de la carga. Como el *IPC*, es una métrica orientada a prestaciones, pero da una mayor importancia a la la latencia de finalización de la carga estudiada.

$$Tiempo\ de\ finalizacion = Tiempo\ de\ terminacion - Tiempo\ de\ llegada$$

***Energy-Delay Product* (*EDP*).** Se define como la energía consumida por una aplicación multiplicado por el tiempo de ejecución de la misma. Permite identificar aquellas combinaciones con mejor compromiso entre energía consumida y tiempo de finalización.

$$EDP = Energia\ consumida \times Tiempo\ de\ finalizacion$$

4.5 Herramientas

Para realizar los experimentos, además de *Stratus*, también se han usado las siguientes herramientas:

***Cpupower*:** Utilidad de *Linux* que se ha utilizado para ajustar la frecuencia de los núcleos en los que se ejecutan las aplicaciones a la máxima soportada (2,5 GHz), mientras que la del resto se ha establecido en la mínima soportada (1 GHz). Reduciendo así el consumo estático de estos, como se indica en la Sección 4.6.2.

***Shell scripts*:** Estos ficheros contienen órdenes que se ejecutan de forma sucesiva. Se han utilizado para automatizar parte de la realización de los experimentos, principalmente para el lanzamiento de estos (ajustar la frecuencia de los núcleos con *cpupower*, invocar al *manager* y mover los archivos de resultados a los directorios deseados).

***Scripts* en *Python*:** Utilizados para realizar el procesamiento de los datos. Haciendo uso de la librería de *pandas* de *Python*, se importan los datos de los archivos de resultados en formato *CSV*, se calculan las métricas deseadas a partir de ellos y, con ayuda de la librería *matplotlib*, se representan gráficamente.

***tx2mon*:** Utilidad compuesta por un módulo del *kernel* de *Linux* y una aplicación de usuario (comentados en la Sección 3.2), que se ha utilizado para obtener el consumo estático del procesador, ayudando así a identificar su impacto en el consumo global (ver Sección 4.6.2 para más detalles).

***Perf*:** Herramienta de análisis de rendimiento para *Linux* que se ha utilizado para identificar todos los eventos soportados por el procesador.

4.6 Ajustes sobre los resultados

4.6.1. Tiempo de ejecución

Dado que los distintos *benchmarks* tienen diferentes tiempos de ejecución, para poder compararlos de manera razonable entre sí se ha limitado el número máximo de instrucciones que cada aplicación puede ejecutar. Este límite se ha fijado en las instrucciones que ejecuta una única instancia de esa aplicación durante un minuto a la máxima frecuencia soportada por el procesador (2,5 GHz).

Pese a fijar el límite de instrucciones, al ejecutar múltiples aplicaciones simultáneamente, estas interfieren entre sí, lo que provoca que no todas tarden lo mismo. Es por esto que el *manager* se ha configurado para relanzar las aplicaciones a medida que van terminando, de modo que todas las aplicaciones terminan cuando lo hace la más lenta. De esta manera, se evita que el nivel de contención en el sistema se reduzca a medida que las aplicaciones más rápidas terminan.

4.6.2. Consumo estático del procesador

Debido a que las mediciones de los consumos del procesador incluyen el consumo estático, es necesario determinar este consumo para identificar su impacto en el total de la energía global.

El consumo estático o *leakage*, también conocido como corrientes de fuga, representa la energía consumida por el simple hecho de que los transistores se encuentren activos. Este consumo se mide cuando el procesador no ejecuta ninguna aplicación.

El porcentaje sobre el consumo global que representa el consumo estático puede llegar a ser relativamente alto, especialmente cuando se ejecuta un número reducido de aplicaciones. Para abordar este problema se han utilizado las siguientes técnicas:

Reducción de la frecuencia. Se ha reducido la frecuencia de los núcleos del sistema que no se encuentran en uso a la mínima permitida (1 GHz), minimizando el consumo estático de estos.

Sustracción del consumo estático. Se ha medido el consumo energético estático, tanto de los núcleos como de la caché, cuando no se ejecuta ninguna carga. Este consumo se ha sustraído de los resultados de consumo energético obtenidos en los experimentos donde se ejecuta una sola aplicación (ejecución individual). De esta manera, se puede determinar con exactitud el consumo dinámico atribuible a la ejecución de una sola aplicación en ejecución individual.

CAPÍTULO 5
Evaluación experimental

Como se ha comentado anteriormente, el objetivo de este trabajo es identificar una serie de métricas que puedan ser usadas para determinar la mejor manera de planificar las aplicaciones, en función de si se pretende maximizar sus prestaciones, la eficiencia energética o alcanzar un equilibrio entre ambos.

Para ello, se estudia y caracteriza, desde el punto de vista del consumo, el comportamiento de 16 *benchmarks* de la *suite SPEC CPU*, mencionados en la Sección 4.6. En los siguientes experimentos, estos *benchmarks* se ejecutan de forma individual, variando el número de instancias de cada uno y, por último, cuando se ejecutan por parejas, variando también el número de instancias.

5.1 Caracterización individual de aplicaciones: prestaciones

En primer lugar, se evalúan las prestaciones de los *benchmarks* seleccionados utilizando el *IPC* como métrica. También se analiza la relación entre las prestaciones y la cantidad de accesos a memoria, así como entre las prestaciones y el consumo de los núcleos del procesador.

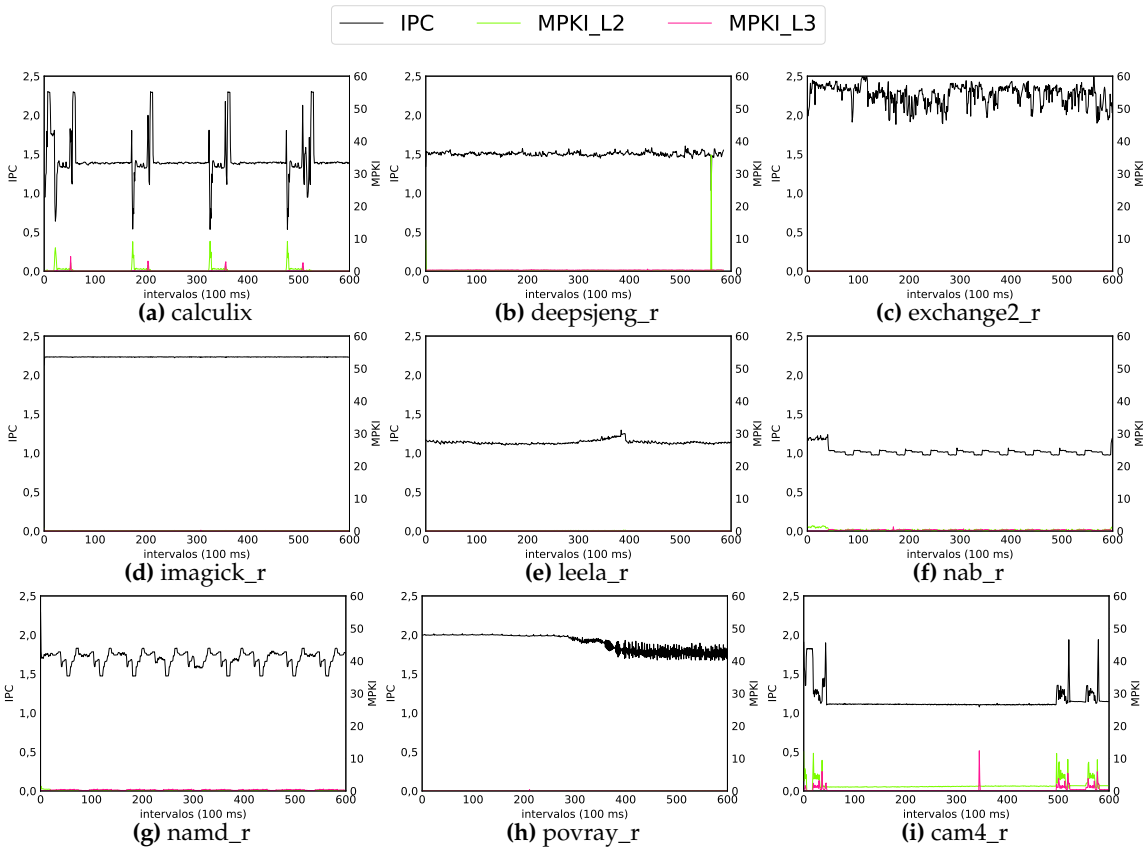
Para cuantificar los fallos de las distintas cachés, se utiliza la métrica $MPKI_{Li}$ (*cache misses per kilo instructions*) para la caché de nivel i. Esta métrica puede aplicarse a las cachés de todos los niveles y permite conocer, durante la ejecución de una aplicación, la cantidad de fallos por cada mil instrucciones. Resulta especialmente útil aplicarla a la caché L3, ya que los fallos de este nivel se traducen en accesos a memoria principal, que penalizan significativamente las prestaciones de las aplicaciones.

Las Figuras 5.1, 5.2, 5.3 y 5.4 muestran la evolución del *IPC*, $MPKI_{L2}$ y $MPKI_{L3}$ de cada uno de los *benchmarks*, que se han agrupado en función del comportamiento observado. Estos comportamientos son:

CPU bound. En la Figura 5.1 se muestran las 9 aplicaciones que presentan este comportamiento. Todas ellas presentan un *IPC* relativamente alto y muy pocos fallos en las memorias caché ($MPKI_{L2}$ y $MPKI_{L3}$ próximos a 0).

En la mayoría de estas aplicaciones el *IPC* presenta pocas variaciones. Una de las aplicaciones en las que más varía es *calculix* (Figura 5.1a) y, como puede observarse en la figura, estas variaciones ocurren cuando se producen cambios en los *MPKI* de las cachés, que llegan al 5 %.

Memory bound. En el extremo opuesto tenemos a las 4 aplicaciones limitadas por memoria (Figura 5.2). Estas aplicaciones presentan un mayor *MPKI*, especialmente en la caché L3.

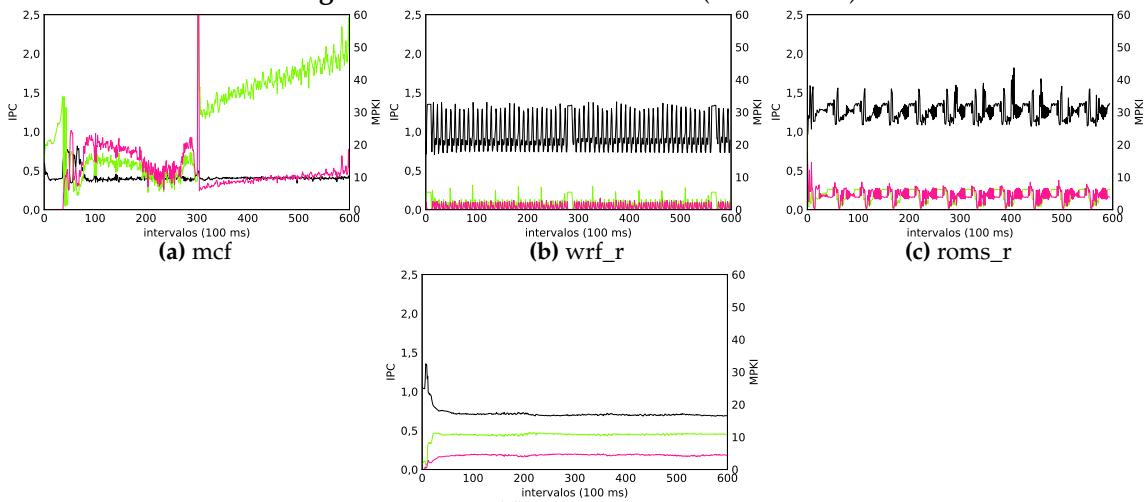


(a) calculix **(b)** deepsjeng_r **(c)** exchange2_r

(d) imagick_r **(e)** leela_r **(f)** nab_r

(g) namd_r **(h)** povray_r **(i)** cam4_r

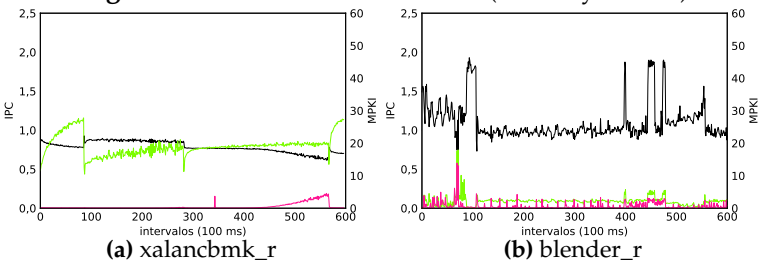
Figura 5.1: MPKI e IPC dinámicos (CPU bound)



(a) mcf **(b)** wrf_r **(c)** roms_r

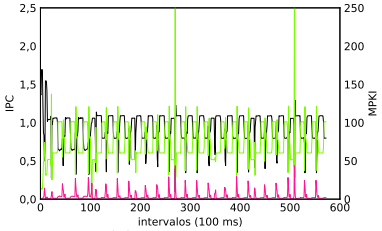
(d) omnetpp_r

Figura 5.2: MPKI e IPC dinámicos (memory bound)



(a) xalancbmk_r **(b)** blender_r

Figura 5.3: MPKI e IPC dinámicos (L3 bound)



(a) cactuBSSN_r

Figura 5.4: MPKI e IPC dinámicos (Extreme L3 bound)

Dentro de estas podemos distinguir entre *mcf* (a partir del intervalo 300) y *omnetpp_r*, que presentan una tasa de fallos constante en las cachés y, por consiguiente, un *IPC* estable. Por otro lado, *wrf_r* y *roms_r* presentan un *MPKI* mucho más variable, lo que se traduce en un *IPC* también variable.

L3 bound. Frente a las aplicaciones *memory bound*, las *L3 bound* (Figura 5.3) presentan muchos más aciertos en la caché L3. Esto es especialmente notable en el caso de *xalancbmk_r* (Figura 5.3a), que presenta la mayor tasa de accesos a la L3 de las 2 aplicaciones ($MPKI_{L2}$ más elevado) y, durante la mayor parte de su ejecución, la mayoría de estos son aciertos ($MPKI_{L3}$ muy reducido).

Extreme L3 bound. Solo *cactuBSSN_r* presenta este comportamiento (Figura 5.4a) alternando periódicamente una tasa de accesos alta a la caché L3 con una muy alta. Supera, la mayor parte del tiempo, 50 accesos por cada mil instrucciones ejecutadas. En estos casos, el *IPC* se encuentra alrededor de 1,1, pero en los valores muy altos de accesos a la caché L3 (superiores a 100), el *IPC* cae drásticamente a 0,4. Aunque la localidad de la L3 es, en general, alta ($MPKI_{L3} < 7$), esta presenta picos regulares que superan un valor de 20. Esta elevada localidad contribuye a reducir el número de accesos a la memoria principal, que tienen una gran latencia. La menor latencia de L3 en comparación con memoria, junto con la ejecución fuera de orden del procesador, disminuye el impacto de tantos accesos a la L3 en las prestaciones, siendo los fallos de la caché L3, y no los de la L2, los que reducen drásticamente el *IPC*.

5.2 Caracterización individual de aplicaciones: consumo vs prestaciones

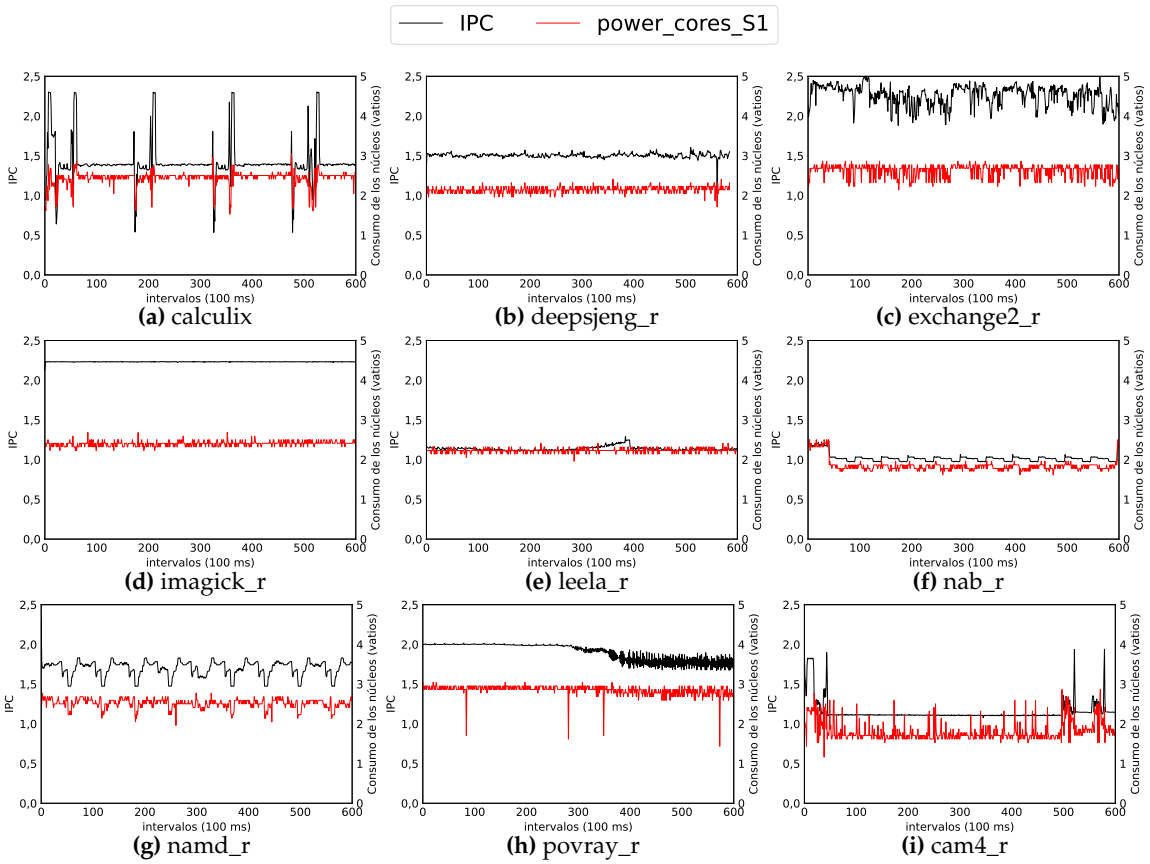
Otro aspecto importante es el efecto que las prestaciones de las aplicaciones tienen en el consumo energético del procesador. Las Figuras 5.5, 5.6, 5.7 y 5.8 muestran la evolución del *IPC* junto al consumo de los núcleos del procesador (*power_cores_S1*, medido en vatios) para los cuatro tipos de aplicaciones identificados. Asimismo, las Figuras 5.9, 5.10, 5.11 y 5.12 muestran la evolución de los accesos y aciertos a la caché L3 (número de eventos *L3_EVENT_READ_REQ* y *L3_EVENT_READ_HIT*) junto a su consumo (*power_mem_S1*, también medido en vatios) para los cuatro tipos de aplicaciones identificados. En esta sección se analiza el comportamiento de estas métricas y su interrelación, centrándose, en primer lugar, en el consumo de los núcleos y, en segundo lugar, en el consumo de la caché L3.

Nótese que se han utilizado las estrategias mencionadas en la Sección 4.6 para evitar que la interpretación y análisis de los resultados de las aplicaciones en ejecución individual se vean afectados por el consumo estático global del procesador.

5.2.1. Consumo de los núcleos

Como era de esperar, el consumo de los núcleos muestra una relación directa con el *IPC*. Siendo las aplicaciones con un mayor *IPC* (las *CPU bound*) las que presentan también los mayores consumos. Por ejemplo, *exchange2_r* e *imagick_r* (Figuras 5.5a y 5.5d), presentan los mayores *IPC* junto a algunos de los consumos más elevados, entre los 2,2 y 2,5 vatios. En el caso de *calculix*, puede verse además la estrecha relación entre el consumo y el *IPC*, pues las variaciones en este último se reflejan también en el consumo.

En cambio, otros tipos de aplicaciones como las *memory bound* y *L3 bound* tienden a presentar un *IPC* y consumo más bajos. Por ejemplo, *mcf* y *xalancbmk_r* que son *memory*

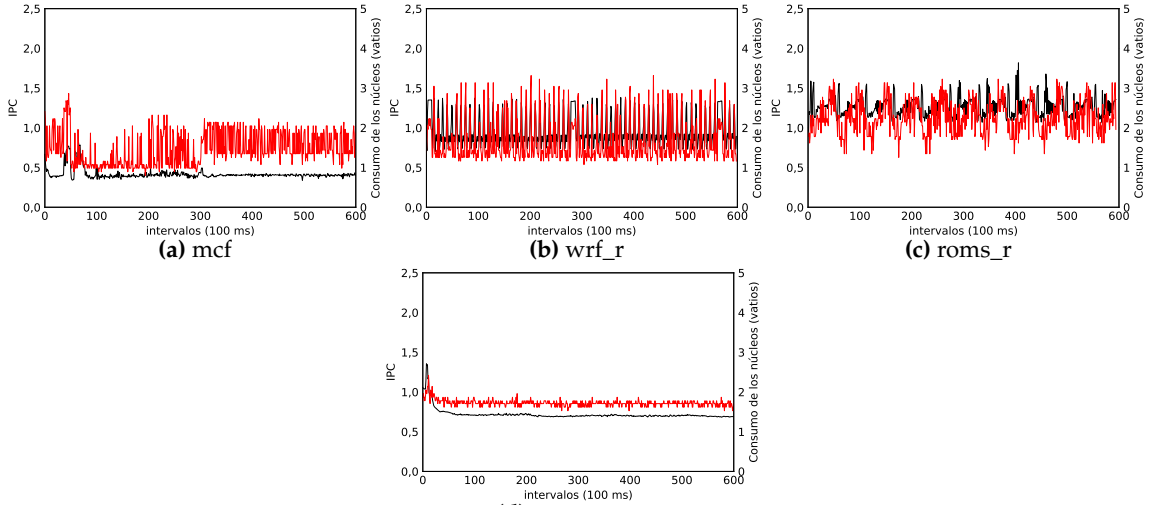


(a) calculix **(b)** deepsjeng_r **(c)** exchange2_r

(d) imagick_r **(e)** leela_r **(f)** nab_r

(g) namd_r **(h)** povray_r **(i)** cam4_r

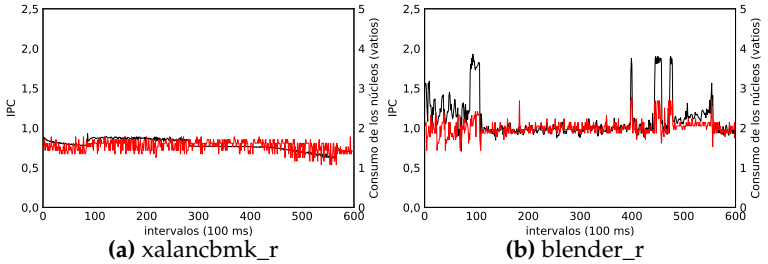
Figura 5.5: IPC y consumo dinámico de los núcleos (CPU bound).



(a) mcf **(b)** wrf_r **(c)** roms_r

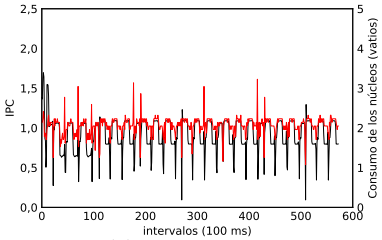
(d) omnetpp_r

Figura 5.6: IPC y consumo dinámicos de los núcleos (memory bound).



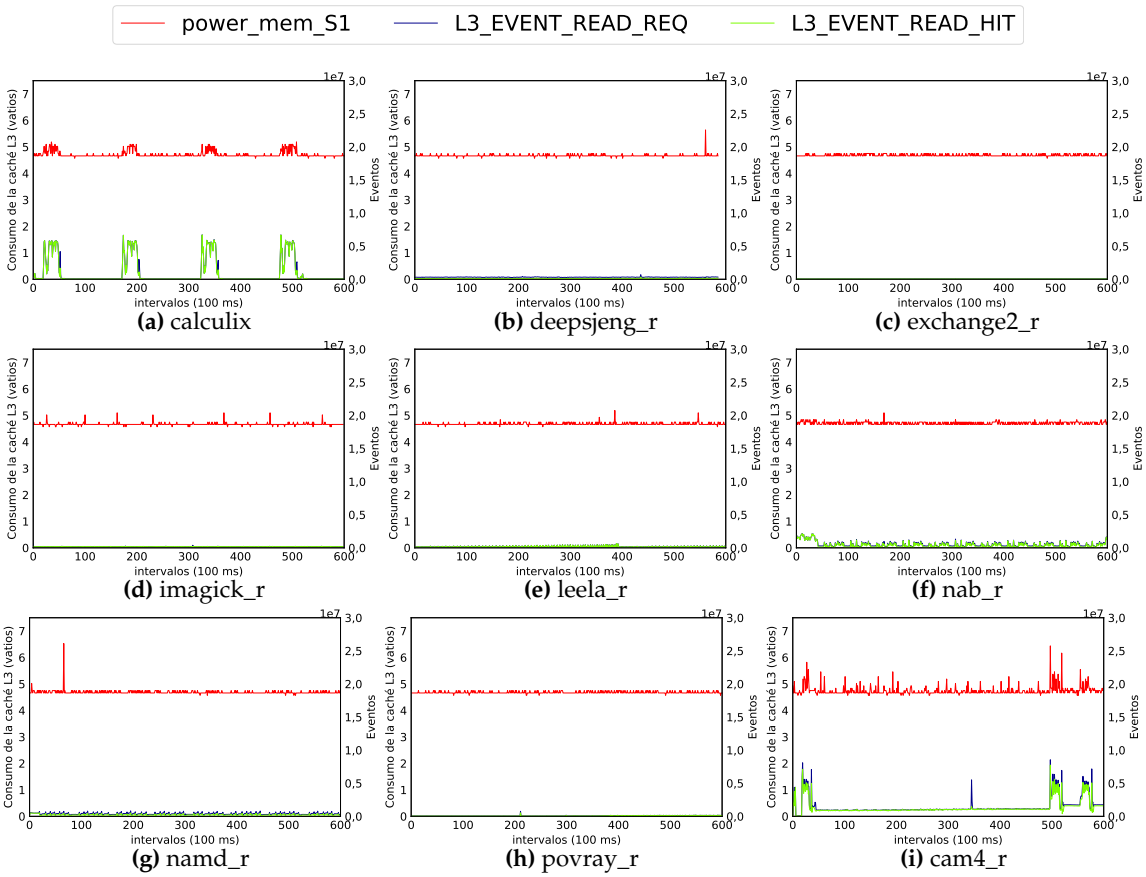
(a) xalancbmk_r **(b)** blender_r

Figura 5.7: IPC y consumo dinámicos de los núcleos (L3 bound).



(a) cactuBSSN_r

Figura 5.8: IPC y consumo dinámicos de los núcleos (Extreme L3 bound).

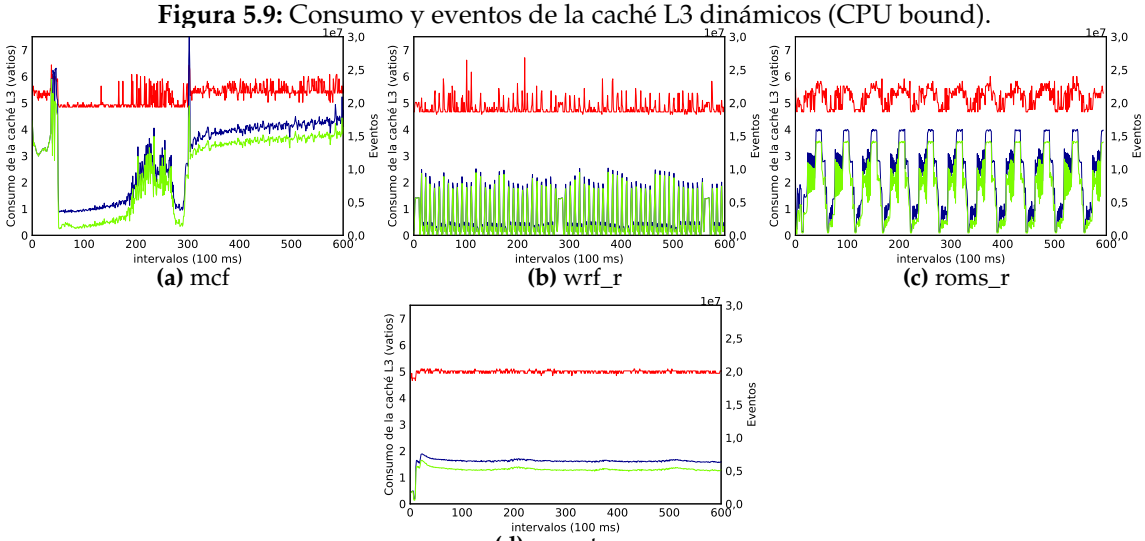


(a) calculix **(b)** deepsjeng_r **(c)** exchange2_r

(d) imagick_r **(e)** leela_r **(f)** nab_r

(g) namd_r **(h)** povray_r **(i)** cam4_r

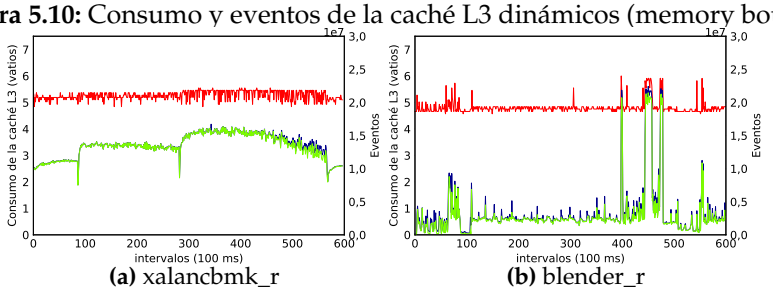
Figura 5.9: Consumo y eventos de la caché L3 dinámicos (CPU bound).



(a) mcf **(b)** wrf_r **(c)** roms_r

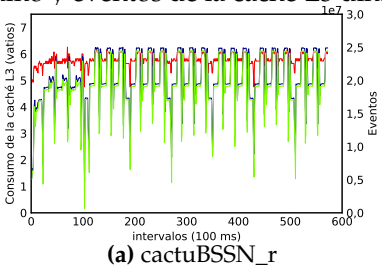
(d) omnetpp_r

Figura 5.10: Consumo y eventos de la caché L3 dinámicos (memory bound).



(a) xalancbmk_r **(b)** blender_r

Figura 5.11: Consumo y eventos de la caché L3 dinámicos (L3 bound).



(a) cactuBSSN_r

Figura 5.12: Consumo y eventos de la caché L3 dinámicos (Extreme L3 bound).

bound y *L3 bound* respectivamente, presentan unos consumos más reducidos, sobre 1,2 y 1,4 respectivamente (Figuras 5.6a y 5.7a).

A continuación, en las Figuras 5.9, 5.10, 5.11 y 5.12 se pueden ver los accesos y aciertos de las aplicaciones en la caché L3, además del consumo de esta. En ellas se aprecia que la evolución de los accesos y aciertos es inversamente proporcional a la del *IPC* y al consumo de los núcleos de ambas aplicaciones.

Podemos destacar 3 aspectos principales:

- Las aplicaciones *CPU bound* presentan los *IPC* más elevados y realizan, en general, pocos accesos a la caché (Figura 5.9), mientras que el resto de aplicaciones acceden mucho más.
- En las aplicaciones que presentan muchos accesos a la caché, como *xalancbmk_r*, hay una relación inversamente proporcional entre estos y el *IPC*. Esto se debe a que las latencias introducidas por estos accesos penalizan al *IPC*.
- Los dientes de sierra que se observan en el acceso a la caché en algunas aplicaciones, como *wrf_r* (Figura 5.10b), también tienen un impacto en el *IPC*.

5.2.2. Consumo de la caché L3

El consumo dinámico de la caché viene determinado por la cantidad de accesos y aciertos. En cada acceso se accede al *tag array* para comprobar si el bloque que contiene el dato solicitado se encuentra en la caché. En caso de fallo se accede a la memoria principal y, en caso de acierto, se accede al *data array*, lo que incrementa el consumo.

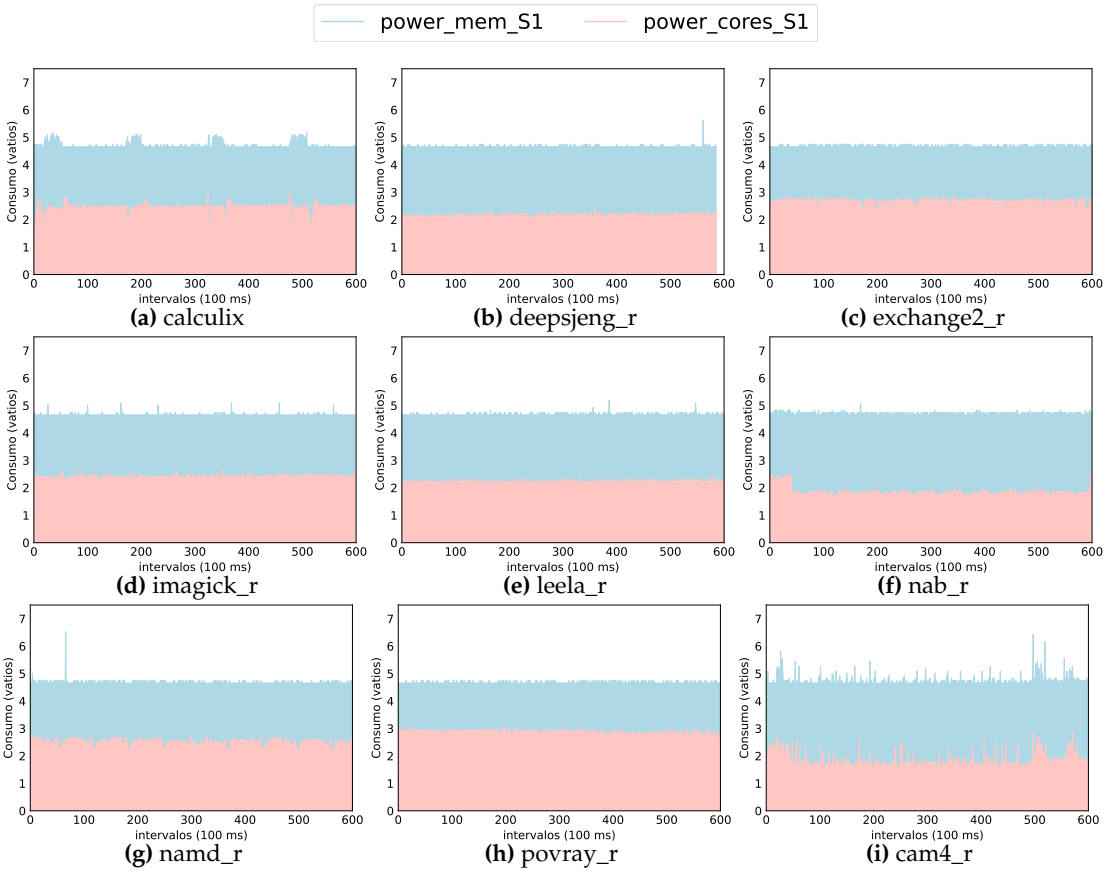
Para facilitar el análisis del consumo de la caché y su relación con el consumo de los núcleos, las Figuras 5.13, 5.14, 5.15 y 5.16 representan la evolución de ambos para todas las aplicaciones. Nótese que las figuras presentan los datos de forma *no apilada*.

Como era de esperar, las aplicaciones *CPU bound* presentan los consumos de energía de los núcleos más elevados (durante la mayor parte del tiempo todas ellas consumen 2 o más vatios, y aplicaciones como *povray_r* llegan a los 3), mientras que el consumo de la caché es más bajo en comparación con otros tipos de aplicaciones. Como se ha observado anteriormente, estas aplicaciones acceden muy poco a la memoria caché, lo que resulta en un consumo de esta más bajo en comparación con otras aplicaciones, siendo inferior a los 5 vatios en la mayoría de casos.

El comportamiento de las aplicaciones *memory bound* es diferente. Estas presentan un consumo de la caché mayor que las *CPU bound*, ya que, aunque están limitadas por memoria (la caché L3 falla constantemente) acceden mucho más a la caché, provocando que su consumo no baje de los 5 vatios. De hecho, el consumo se acerca a los 6 vatios en aplicaciones como *mcf* (Figura 5.6a).

También es destacable que, como se puede ver en la Figura 5.10, las aplicaciones *memory bound* (a excepción de *omnetpp_r*) muestran un patrón de dientes de sierra en el número de accesos y aciertos en la caché L3. Este comportamiento se refleja en el consumo de la caché que también presenta un patrón similar de dientes de sierra.

Algo similar ocurre tanto con las aplicaciones *L3 bound* como con las *extreme L3 bound*, pues tienden a un consumo menor en los núcleos y mayor en la caché. Dentro de estas existen diferencias en la cantidad de accesos a la caché que realizan. Por ejemplo, *roms_r (memory bound)* y *cactuBSSN_r (extreme L3 bound)* acceden más que el resto a la caché y esto se refleja en un consumo de esta mayor (cercano a 6 vatios).

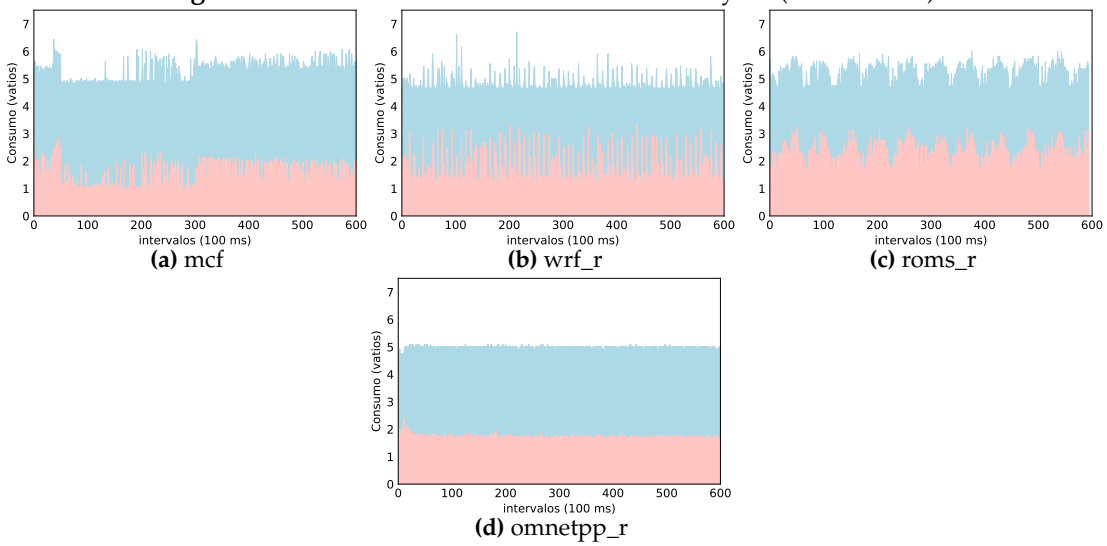


(a) calculix **(b)** deepsjeng_r **(c)** exchange2_r

(d) imagick_r **(e)** leela_r **(f)** nab_r

(g) namd_r **(h)** povray_r **(i)** cam4_r

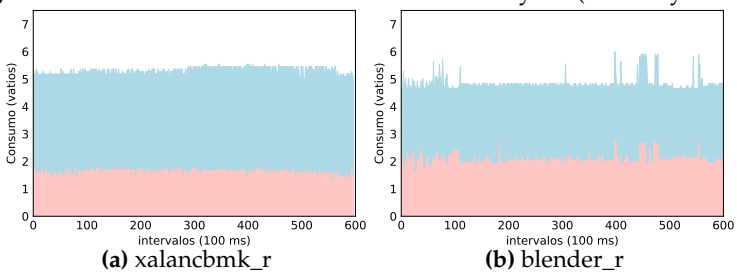
Figura 5.13: Consumo dinámico de los núcleos y L3 (CPU bound).



(a) mcf **(b)** wrf_r **(c)** roms_r

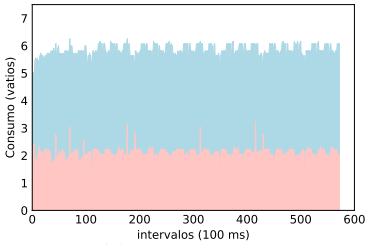
(d) omnetpp_r

Figura 5.14: Consumo dinámico de los núcleos y L3 (memory bound).



(a) xalancbmk_r **(b)** blender_r

Figura 5.15: Consumo dinámico de los núcleos y L3 (L3 bound).



(a) cactuBSSN_r

Figura 5.16: Consumo dinámico de los núcleos y L3 (Extreme L3 bound).

5.3 Prestaciones y consumo variando el número de instancias por aplicación

5.3.1. Instancias del mismo tipo

En esta sección se analizan las interferencias entre las aplicaciones en tiempo de ejecución, variando el número de instancias por aplicación hasta llegar a 28, que es el número de núcleos del procesador estudiado. Los experimentos miden los resultados para 1, 7, 14, 21 y 28 instancias. De esta manera, aumentamos el número de núcleos de 7 en 7 a partir de 7, hasta ocupar todos los núcleos, manteniendo también la ejecución individual (1 instancia).

En primer lugar, se caracterizan las interferencias asumiendo que las aplicaciones son del mismo tipo. En este sentido, existen dos opciones: o bien utilizar aplicaciones distintas del mismo tipo, o bien replicar el número de instancias de una misma aplicación. La segunda opción es la más simple y reproducible y, por tanto, más común: utilizar un *µbenchmark* y replicarlo tantas veces como sea necesario. En consecuencia, es la elegida en el presente trabajo.

Las Figuras 5.17, 5.18, 5.19 y 5.20 presentan la variación del *IPC* y del consumo del procesador a medida que aumenta el número de instancias. Dentro del *IPC* distinguimos entre el de una única instancia y el global, que es la suma del *IPC* de cada una de las instancias que se ejecutan. El consumo está normalizado con respecto al consumo estático base, que es el medido cuando no se ejecuta ninguna aplicación y se distribuye en varias categorías: núcleos (*power_cores*), *SoC* (*power_soc*), caché L3 (*power_mem*) y resto de estructuras de almacenamiento en el procesador (*power_sram*).

En estas figuras observamos tres patrones de comportamiento según la escalabilidad en prestaciones de las aplicaciones:

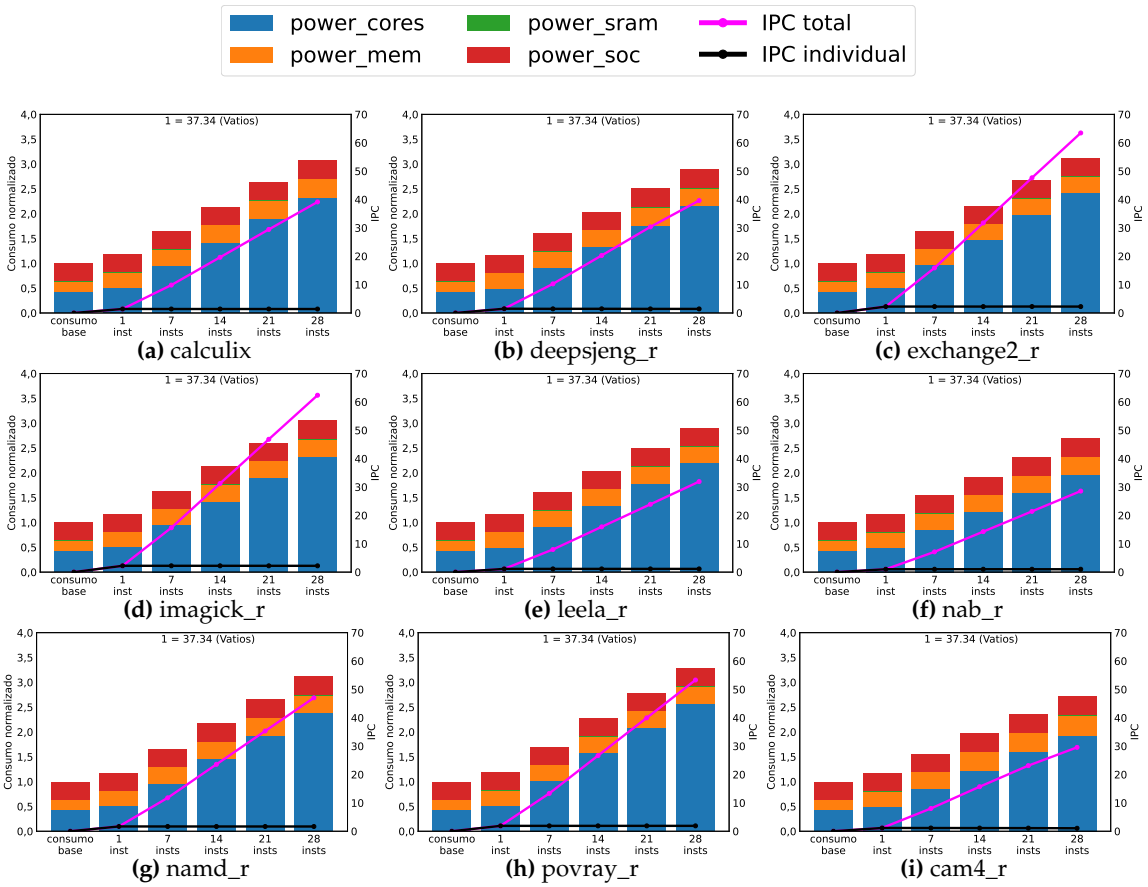
Highly-scalable performance. Este comportamiento es el presentado por las aplicaciones *CPU bound*. En este tipo de aplicaciones, al ser limitadas por la *CPU*, se genera muy poca contención entre instancias por los recursos de fuera del núcleo (por ej., la caché L3), por lo que el *IPC* por instancia se mantiene, el *IPC* global presenta un crecimiento prácticamente lineal y el consumo crece, también, de manera lineal.

Como cabe esperar, el consumo de los núcleos es también muy elevado, mientras que el consumo de la caché y del *SoC* tienen valores similares, independientemente del número de instancias.

Medium-scalable performance. Este comportamiento es presentado por *wrf_r*, *roms_r*, *blender_r* y *cactuBSSN_r*. Todas ellas presentan un crecimiento menor que las *CPU bound*, que se va reduciendo a medida que aumenta el número de instancias.

En el caso de *cactuBSSN_r*, esto se debe a que, como se ha comentado antes, hace un uso intensivo de la caché L3, con una alta localidad. Al aumentar el número de instancias, aumenta la contención en la caché, lo que resulta en más fallos y accesos a memoria. La latencia de muchos de estos accesos no puede ser ocultada por la ejecución fuera de orden, pero el hecho de que presenten una alta localidad evita que el *IPC* caiga de manera drástica.

Las otras aplicaciones (*wrf_r*, *roms_r* y *blender_r*) acceden mucho menos a la caché que *cactuBSSN_r*, pero tienen también una localidad menor, por lo que su crecimiento también se va degradando a medida que aumenta la contención en la caché con el número de instancias.

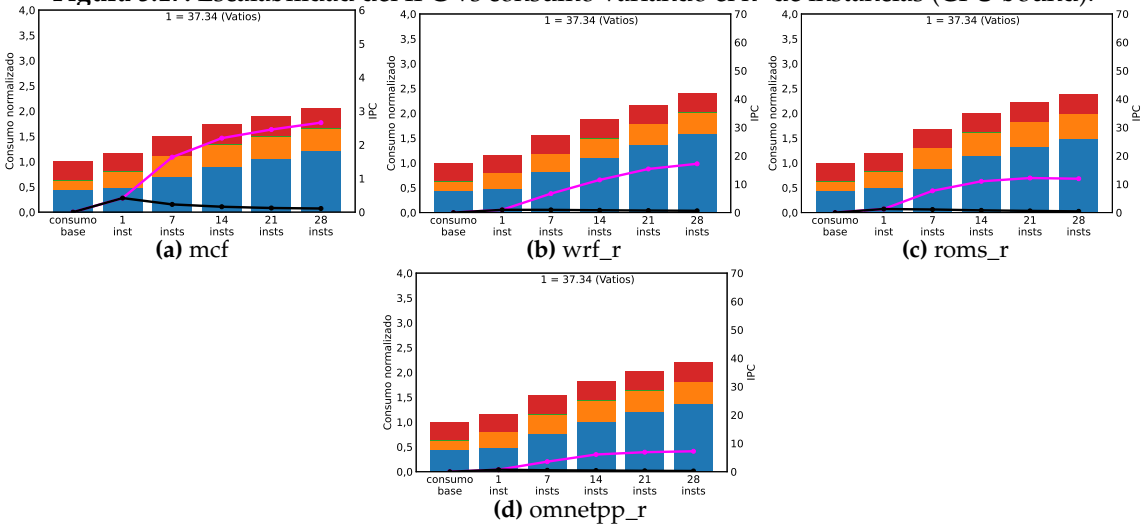


(a) calculix **(b)** deepsjeng_r **(c)** exchange2_r

(d) imagick_r **(e)** leela_r **(f)** nab_r

(g) namd_r **(h)** povray_r **(i)** cam4_r

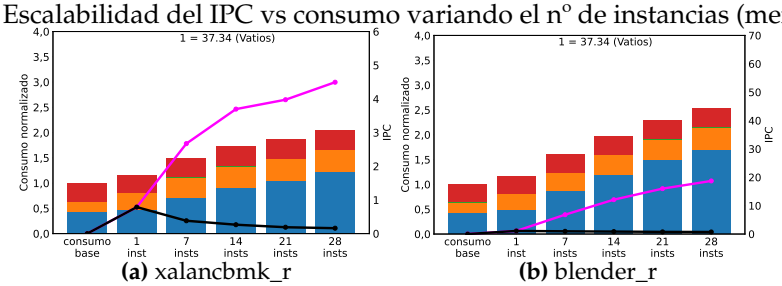
Figura 5.17: Escalabilidad del IPC vs consumo variando el nº de instancias (CPU bound).



(a) mcf **(b)** wrf_r **(c)** roms_r

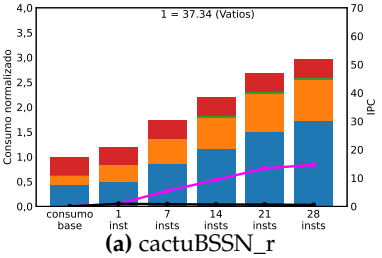
(d) omnetpp_r

Figura 5.18: Escalabilidad del IPC vs consumo variando el nº de instancias (memory bound).



(a) xalancbmk_r **(b)** blender_r

Figura 5.19: Escalabilidad del IPC vs consumo variando el nº de instancias (L3 bound).



(a) cactuBSSN_r

Figura 5.20: Escalabilidad del IPC vs consumo variando el nº de instancias (Extreme L3 bound).

Estas diferencias se reflejan también en el consumo. Toda estas aplicaciones tienen un consumo en los núcleos más bajo que el de las *CPU bound*. Sin embargo, respecto al consumo de la caché, *cactuBSSN_r* es la única aplicación en la que crece considerablemente, pues es la aplicación que realiza más accesos y tiene más aciertos en la L3.

Low-scalable performance. 3 son las aplicaciones estudiadas (*mcf*, *omnetpp_r* y *xalancbmk_r*) que no escalan cuando se incrementa su número. Para ellas, el *IPC* para 28 instancias es inferior a 8, lo que resulta en una disminución del consumo tanto en los núcleos como en la caché L3. Un bajo consumo en la caché indica que o bien se accede poco, o bien que la mayoría de los accesos fallan.

Este comportamiento se presenta para algunas aplicaciones *memory bound (mcf y omnetpp_r)* y *L3 bound (xalancbmk_r)*, que tienen unas elevadas tasas de aciertos y fallos. Aunque la caché ayuda a reducir el impacto en las prestaciones de tantos accesos, al ejecutar más instancias, estas provocan contención en la caché L3, incrementando el número de fallos y, por tanto, los accesos a memoria principal. De hecho, se puede observar que, al aumentar a siete instancias, el *IPC* individual experimenta una reducción drástica. Esta situación se agrava aún más a medida que se agregan más instancias.

Los resultados ponen de manifiesto la facilidad con la que escalan las aplicaciones limitadas por *CPU* y lo importante que resulta limitar el número de aplicaciones que realizan una gran cantidad de accesos a memoria, pues se perjudican al no ser el sistema capaz de soportar de manera eficiente un elevado *MLP* (*memory level parallelism*). También se observa el elevado peso que tiene el consumo estático sobre el dinámico, especialmente cuando se ejecuta un número reducido de instancias, donde la mayor parte del consumo se debe al estático base y no al procesamiento de la aplicación.

5.3.2. Prestaciones por vatio

Para evaluar de manera conjunta las prestaciones y el consumo energético, es necesario utilizar métricas que consideren ambos valores en la misma fórmula. Una métrica típica son las prestaciones por vatio (*performance per watt*), definida como el *IPC* dividido entre el consumo en vatios del procesador. En cierta medida, esta métrica proporciona una estimación de la eficiencia del sistema. Con el fin de mejorarla deben aumentarse las prestaciones para un consumo dado o reducir el consumo para unas determinadas prestaciones.

La Figura 5.21 presenta las prestaciones (en términos de *IPC*) por vatio para los 3 tipos de aplicaciones (*hihgly-scalable performance*, *medium-scalable performance* y *low-scalable performance*). Estos resultados se han obtenido haciendo la media aritmética de las prestaciones por vatio de las aplicaciones de cada tipo. Los aspectos más destacables son:

- Las aplicaciones *hihgly-scalable performance* siguen siendo las más escalables en prestaciones por vatio. No obstante, se aprecia que las prestaciones por vatio escalan de manera menos pronunciada que si solo se consideran las prestaciones. Esto se debe a que el consumo del procesador se incrementa aún más que las prestaciones.

- Las aplicaciones *medium-scalable performance* siguen presentando una escalabilidad intermedia cuando se considera también el consumo de energía y no solo las prestaciones. Sin embargo, el crecimiento de las prestaciones por vatio se estancan a partir de 21 instancias por aplicación.

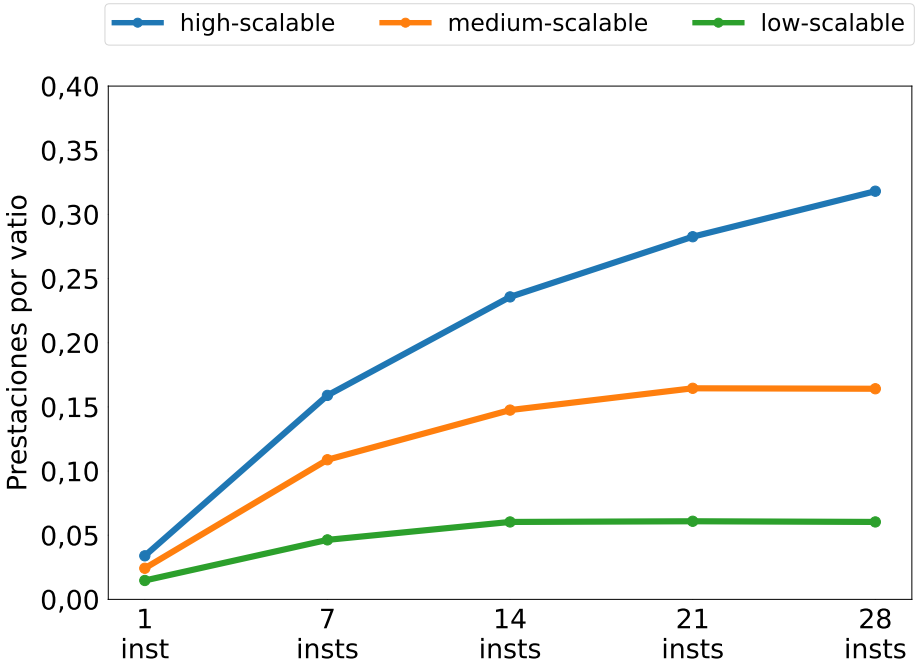


Figura 5.21: Prestaciones por vatio variando el número de instancias de las aplicaciones.

- Las aplicaciones *low-scalable performance* presentan los peores resultados. Las prestaciones por vatio apenas mejoran de 7 a 14 instancias por aplicación, para prácticamente estabilizarse a partir de dicho valor.

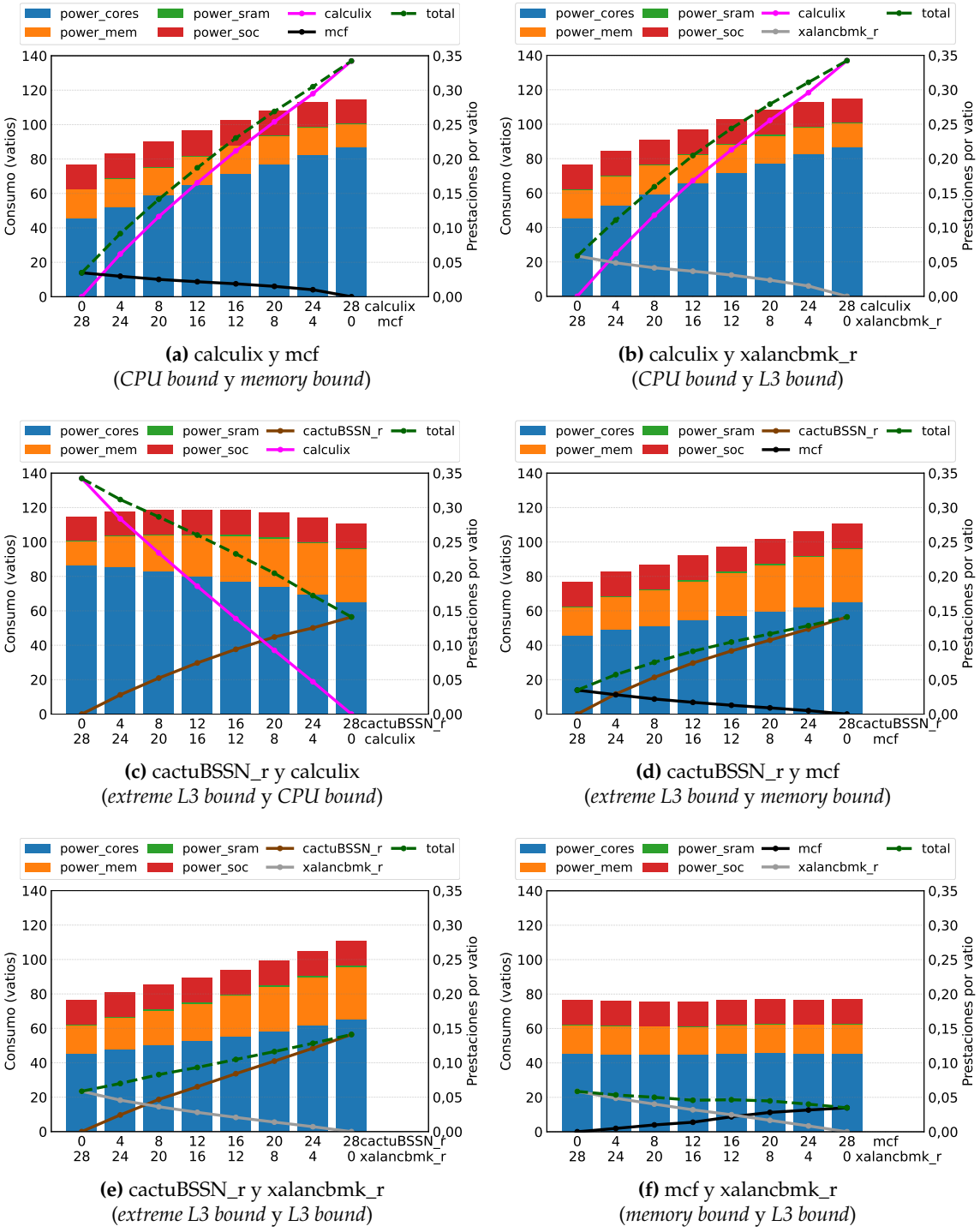
5.3.3. Instancias con distinto comportamiento

Hasta ahora se ha estudiado el efecto sobre las prestaciones y consumo al ejecutar múltiples instancias de aplicaciones exhibiendo el mismo comportamiento desde el punto de vista de las prestaciones. En esta sección se analiza el efecto de mezclar instancias de aplicaciones exhibiendo comportamientos distintos. El objetivo de este estudio es identificar el valor óptimo de instancias por aplicación de cada comportamiento, de cara a maximizar la métrica objetivo (por ej., las prestaciones por vatio).

Por cada uno de los comportamientos identificados anteriormente se ha seleccionado una aplicación que lo representa. En particular, para las aplicaciones *CPU bound* se ha seleccionado *calculix*, *mcf* para las *memory bound*, *xalancbmk_r* para las *L3 bound* y, por último, *cactuBSSN_r* para las *extreme L3 bound*.

La Figura 5.22 presenta los valores de prestaciones por vatio y consumo para todas las parejas posibles de distintos tipos. Para cada pareja se varía el número de instancias de cada aplicación, asegurando que el número de instancias ejecutadas entre dos aplicaciones sea igual al número de núcleos (28). En concreto, para un número de instancias de un determinado tipo o comportamiento 1 ($n1$) y de otro comportamiento 2 ($n2$), donde $n1 + n2 = 28$, se estudia el conjunto de combinaciones ($n1$-$n2$) siguiente: (28-0), (24-4), (20-8), (16-12), (12-16), (8-20), (4-24) y (0-28). En la figura, el consumo se muestra de manera apilada mientras que las prestaciones por vatio se muestran por puntos unidos con líneas para estudiar la tendencia. Se presentan las prestaciones por vatio de cada aplicación y el total.

Con respecto al consumo, se observa que principalmente es debido a los núcleos (*power_cores*). El mayor consumo de los núcleos (por encima de los 80 vatios) se alcanza con



(a) calculix y mcf
(*CPU bound* y *memory bound*)

(b) calculix y xalancbmk_r
(*CPU bound* y *L3 bound*)

(c) cactuBSSN_r y calculix
(*extreme L3 bound* y *CPU bound*)

(d) cactuBSSN_r y mcf
(*extreme L3 bound* y *memory bound*)

(e) cactuBSSN_r y xalancbmk_r
(*extreme L3 bound* y *L3 bound*)

(f) mcf y xalancbmk_r
(*memory bound* y *L3 bound*)

Figura 5.22: Prestaciones por vatio y consumo del sistema variando el número de instancias de las parejas.

28 instancias de *calculix* (*CPU bound*) y 0 del resto (Figuras 5.22a, 5.22b y 5.22c). En menor medida, este efecto también se observa con *cactuBSSN_r* (*extreme L3 bound*) cuando se combina con *mcf* (Figura 5.22d) y *xalancbmk_r* (Figura 5.22e).

Debido a la mayor relevancia del consumo de los núcleos, es de esperar que los mayores consumos totales se observen en configuraciones con un mayor consumo de procesador, que es originado por *calculix* cuando ocupa todos los núcleos. Sin embargo, cabe destacar que esta regla no se cumple cuando *calculix* se combina con *cactuBSSN_r* (Figura 5.22c). Este emparejameniento alcanza los mayores consumos totales de todos los estudiados (cerca de 120 vatios para parejas compuestas por alrededor de 12 instancias de *cactuBSSN_r* y 16 de *calculix*). Este comportamiento se debe al consumo energético de la cache L3 (*power_mem*) por parte de *cactuBSSN_r*, que alcanza los 30 vatios cuando esta aplicación ocupa todo el procesador. Nótese que, como se ha discutido anteriormente, este consumo es debido a que *cactuBSSN_r* es la aplicación que más accesos y aciertos realiza en la caché.

Las prestaciones por vatio también presentan una fuerte correlación con el consumo de los núcleos. Así, las mayores prestaciones por vatio (sobre 0,35) se alcanzan cuando la totalidad del procesador ejecuta instancias de *calculix* y, en el caso de que *calculix* no se ejecute, cuando *cactuBSSN_r* ocupa todo el procesador (prestaciones por vatio ligeramente inferiores a 0,15). Por último, las prestaciones por vatio más bajas se observan cuando se combinan aplicaciones *extreme L3 bound* y *L3 bound* (Figura 5.22f), siendo ligeramente superiores cuando *xalancbmk_r* dispone del procesador en exclusiva.

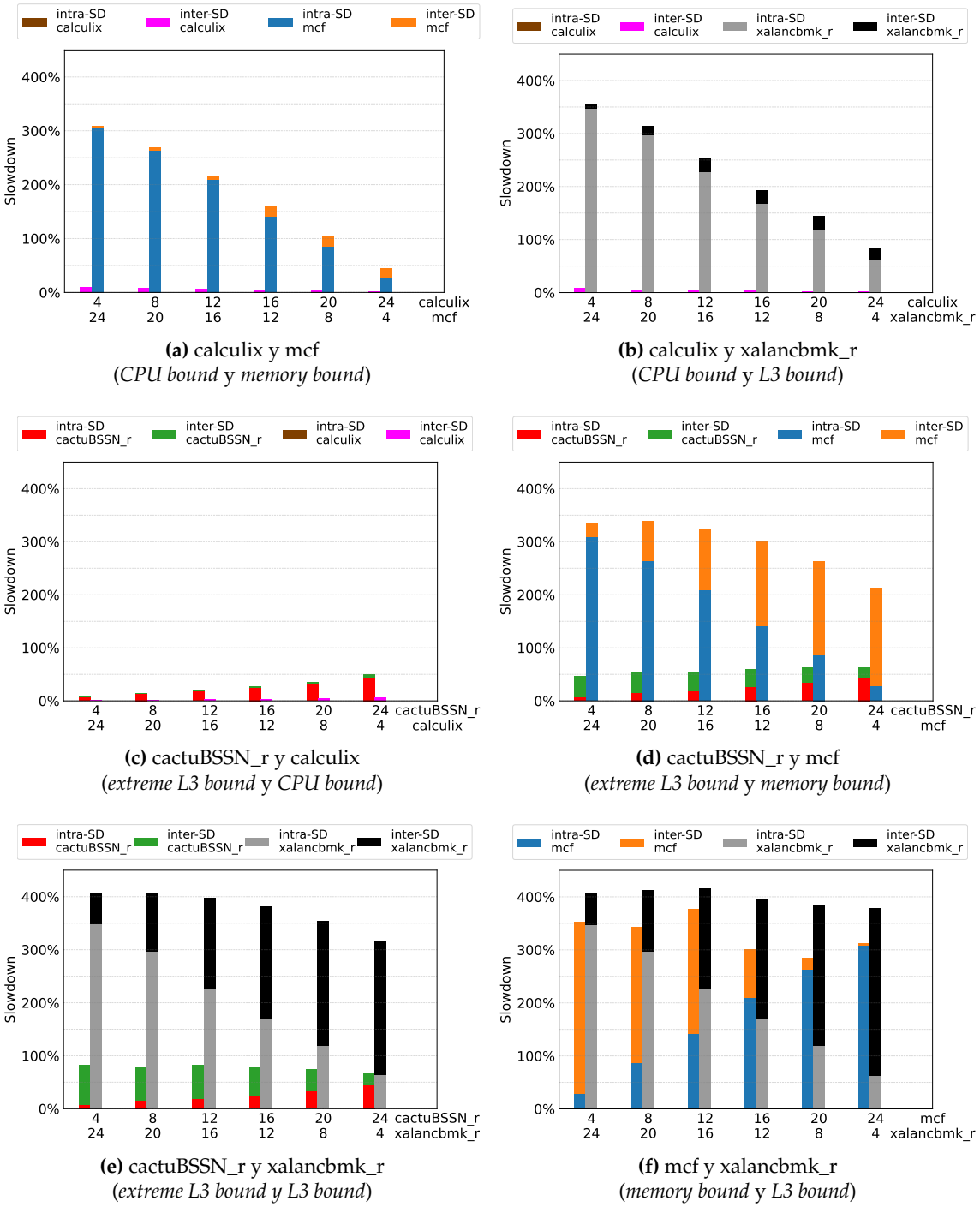
En resumen, se observa que, para cada posible emparejamiento, ejecutar exclusivamente una de las aplicaciones siempre alcanza mayores prestaciones por vatio. Esto provoca que la mejor combinación sea siempre aquella en la que esa aplicación monopolice el procesador, perjudicando el avance de la otra aplicación. En consecuencia, se puede concluir que las prestaciones por vatio no son una métrica adecuada, por sí sola, para elegir la mejor planificación si se desea que ambas aplicaciones realicen avances significativos en su ejecución.

Una posible solución consiste en utilizar una nueva métrica que actúe como contrapeso. Con frecuencia se suele utilizar el *slowdown* o degradación de las prestaciones, definida como la pérdida de prestaciones que experimenta una aplicación respecto a si se ejecutase ella sola en el procesador. Esta métrica se suele denominar en la literatura *individual speedup (IS)*. Se suele utilizar de manera directa, por ejemplo, definiendo un *slowdown* máximo, o indirecta. El *IS* se define para una instancia de una aplicación.

La Figura 5.23 muestra la degradación de prestaciones para las combinaciones previamente mencionadas. Para calcular la degradación de prestaciones de las aplicaciones, cada instancia se considera como una aplicación independiente y, por tanto, debe calcularse el *slowdown* de cada instancia respecto a su ejecución individual. En la figura, la altura de la barra representa la media del *slowdown* (en porcentaje) de las instancias de una aplicación. La barra aparece dividida en dos partes, que muestran la degradación de prestaciones provocada por el resto de instancias de aplicaciones del mismo tipo (*intra-slowdown*) y la causada por las instancias del otro tipo (*inter-slowdown*).

Dependiendo del tipo de aplicación se pueden observar los siguientes comportamientos:

CPU bound. Las Figuras 5.23a, 5.23b y 5.23c muestran la degradación causada al combinar *calculix* (*CPU bound*) con las otras aplicaciones. En cualquier emparejamiento la degradación de *calculix* es muy baja para cualquier distribución de instancias y está causada en su mayor parte por la pareja (*inter-slowdown*).



(a) calculix y mcf
(*CPU bound* y *memory bound*)

(b) calculix y xalancbmk_r
(*CPU bound* y *L3 bound*)

(c) cactuBSSN_r y calculix
(*extreme L3 bound* y *CPU bound*)

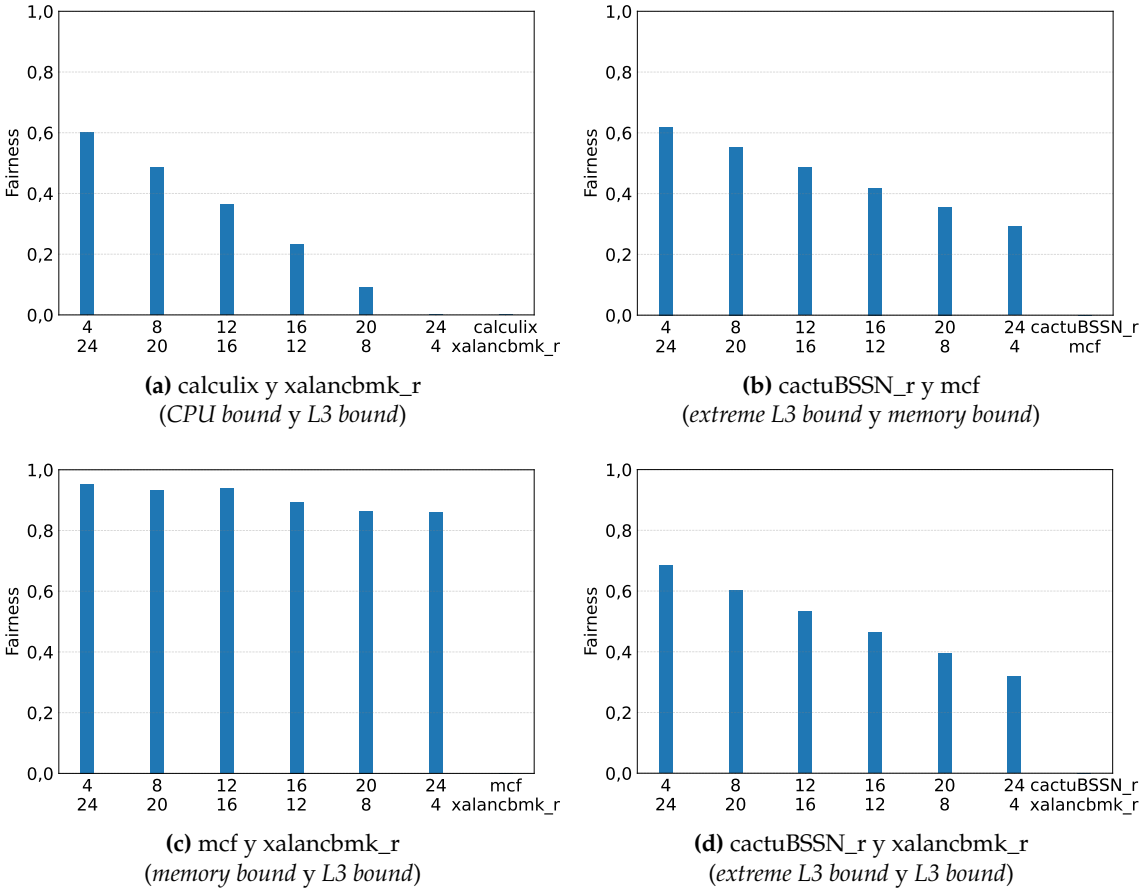
(d) cactuBSSN_r y mcf
(*extreme L3 bound* y *memory bound*)

(e) cactuBSSN_r y xalancbmk_r
(*extreme L3 bound y L3 bound*)

(f) mcf y xalancbmk_r
(*memory bound* y *L3 bound*)

Figura 5.23: Degradación de las prestaciones variando el número de instancias de cada aplicación.

Memory intensive. Representado por las Figuras 5.23a, 5.23d y 5.23f, donde *mcf* (*memory bound*) sufre una degradación mucho mayor que *calculix*. Como las aplicaciones *memory bound* acceden mucho a memoria, el hecho de ejecutar múltiples instancias de *mcf* genera *intra-slowdown interference*, que supera el *300 %* para 28 instancias. La pareja junto a la que se ejecuta es especialmente relevante cuando hay pocas instancias de *mcf*, pues las *inter-slowdown interferences* pueden ser mayores. Aplicaciones *CPU bound* como *calculix* (Figura 5.23a) acceden poco a memoria y generan pocas interferencias, provocando en *mcf* una degradación inferior al 50 % para 4 instancias de *mcf* y 24 de *calculix*. En cambio, cuando la otra aplicación accede más



(a) calculix y xalancbmk_r (*CPU bound* y *L3 bound*)

(b) cactuBSSN_r y mcf (*extreme L3 bound* y *memory bound*)

(c) mcf y xalancbmk_r (*memory bound* y *L3 bound*)

(d) cactuBSSN_r y xalancbmk_r (*extreme L3 bound* y *L3 bound*)

Figura 5.24: Equidad variando el número de instancias de las parejas.

a memoria (Figuras 5.23d y 5.23f) *mcf* sufre una degradación mucho mayor, que no baja del 200 %.

L3 bound. En las Figuras 5.23b, 5.23f y 5.23e se muestra la degradación que sufre *xalancbmk_r* en 3 combinaciones. La degradación de prestaciones se comporta de manera similar a *mcf*, tanto en las interferencias que se generan por múltiples instancias de *xalancbmk_r* como en la dependencia del tipo de pareja, por la contención que puede provocar. No obstante, *xalancbmk_r* presenta una degradación generalmente 50 puntos por encima de *mcf*.

Extreme L3 bound. Comportamiento representado en las Figuras 5.23c, 5.23d y 5.23e, donde *cactuBSSN_r* presenta una degradación muy inferior a la de *mcf* y *xalancbmk_r* aunque mayor que la de *calculix*.

Como era de preveer, al ejecutarse junto a *calculix*, *cactuBSSN_r* sufre la degradación más leve, que se acerca al 50 % en el peor caso (24 instancias de *cactuBSSN_r* y 4 de *calculix*) y es prácticamente nula en el mejor (4 instancias de *cactuBSSN_r* y 24 de *calculix*). En cambio, al ejecutarse junto a *mcf* y *xalancbmk_r* presenta una degradación mucho mayor (cercana al 50 % y al 80 %, respectivamente) y similar para cualquier número de instancias, lo que pone de manifiesto que las interferencias causadas por estas aplicaciones son similares a las causadas por las instancias de la propia *cactuBSSN_r*.

Esta baja degradación, para la elevada cantidad de accesos a la caché que realiza, se explica por su elevada tasa de aciertos (discutida anteriormente), lo que contribuye a mitigar la influencia de las interferencias.

Combinaciones	Procesador 1	Procesador 2
Combinación 1	mcf-xalanbcmk_r	calculix-cactuBSSN_r
Combinación 2	mcf-calculix	xalanbcmk_r-cactuBSSN_r
Combinación 3	mcf-cactuBSSN_r	xalanbcmk_r-calculix

Tabla 5.1: Combinaciones estudiadas.

Carga	mcf	xalanbcmk_r	calculix	cactuBSSN_r
W1	14	14	14	14
W2	12	16	16	12
W3	20	8	8	20
W4	16	12	12	16
W5	8	20	20	8

Tabla 5.2: Configuración de las cargas.

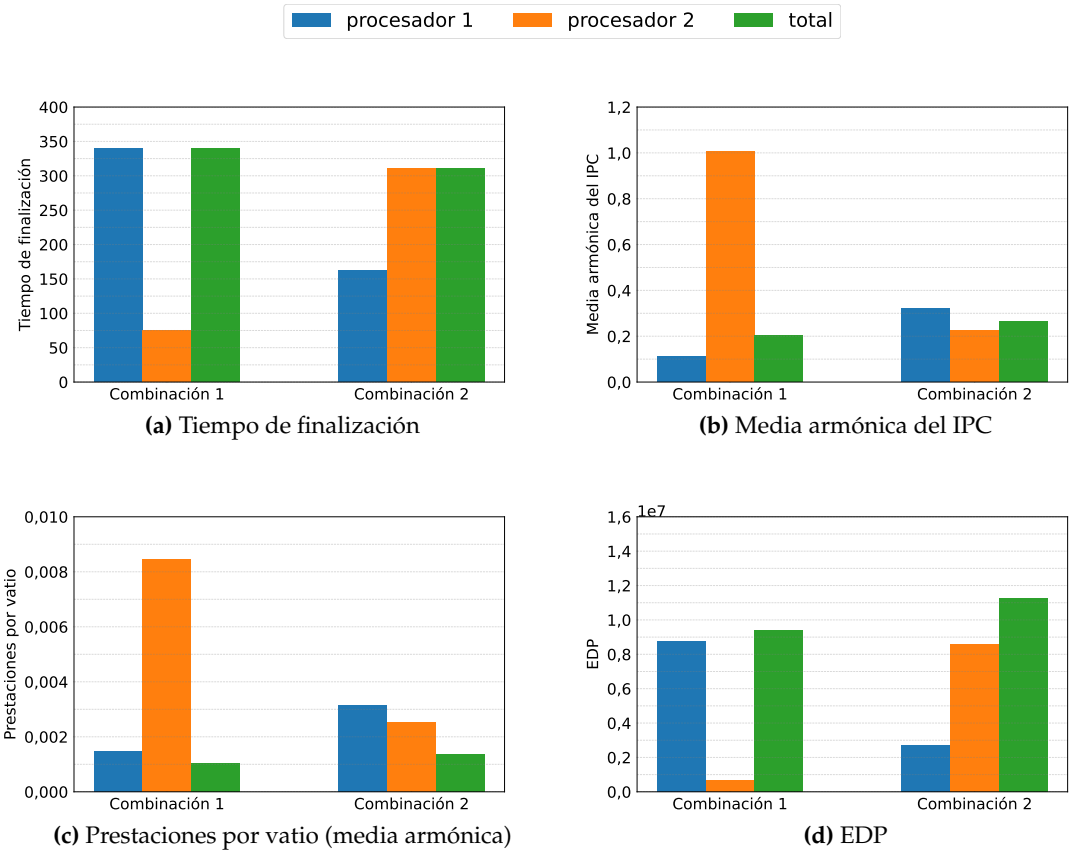
Además de la degradación de prestaciones para guíar la planificación, otra métrica a considerar es la equidad (del inglés *fairness*). En la Figura 5.24 se muestra el *fairness* para las combinaciones estudiadas anteriormente, excepto para los emparejamientos de *calculix* con *mcf* y *calculix* con *cactuBSSN_r*, donde la degradación es tan desigual que el *fairness* resulta negativo. Para 3 de las 4 parejas (Figuras 5.24a, 5.24b y 5.24d), el *fairness* varía considerablemente con el número de instancias, alcanzando, como mucho, un valor entre el 0,6 y 0,7. Esto se debe a que estas parejas se componen de una aplicación que se degrada poco (*calculix* o *cactuBSSN_r*) y de una que lo hace mucho (*mcf* o *xalancbmk_r*). En cambio, el hecho de emparejar *mcf* con *xalancbmk_r* resulta mucho más justo, pues presentan una degradación similar (Figura 5.24c). Por ello, el *fairness* en este caso es muy elevado (entre 0,9 y 1) y similar independientemente del número de instancias.

Si se comparan los resultados de *fairness* con los de las métricas analizadas previamente (prestaciones por vatio y degradación de prestaciones), se observa que el primero tiene un comportamiento opuesto, por lo que resulta una métrica apropiada para ser utilizada para alcanzar compromisos. Por ejemplo, se podría establecer un *fairness* mínimo que el sistema deba satisfacer en cada momento, lo que permitiría evitar ciertas planificaciones en las que una de las aplicaciones monopolice el procesador.

5.4 Análisis de métricas objetivo para el diseño de planificadores

Los planificadores se diseñan para optimizar una determinada métrica o función, denominada métrica objetivo. Existen diferentes políticas que el planificador puede seguir, como reducir el consumo de energía (política de *power budget*), maximizar las prestaciones, o una política mixta que busque eficiencia energética afectando lo menos posible al rendimiento de las aplicaciones. Un planificador ideal sería aquel que, conociendo de manera precisa la evolución de la métrica objetivo en tiempo de ejecución, seleccionase la mejor planificación (*schedule*). A este planificador se le conoce como oráculo ya que necesita información de lo que sucederá en los intervalos futuros.

Un buen planificador debe tener en cuenta las características de las aplicaciones para estimar el impacto en prestaciones y consumo de una determinada planificación. Esta sección estudia el comportamiento de cargas compuestas por aplicaciones de distintos tipos. Para ello, las aplicaciones se agrupan en parejas y cada pareja se asigna a un proce-



(a) Tiempo de finalización

(b) Media armónica del IPC

(c) Prestaciones por vatio (media armónica)

(d) EDP

Figura 5.25: Tiempo de finalización, media armónica, prestaciones por vatio y EDP para W3.

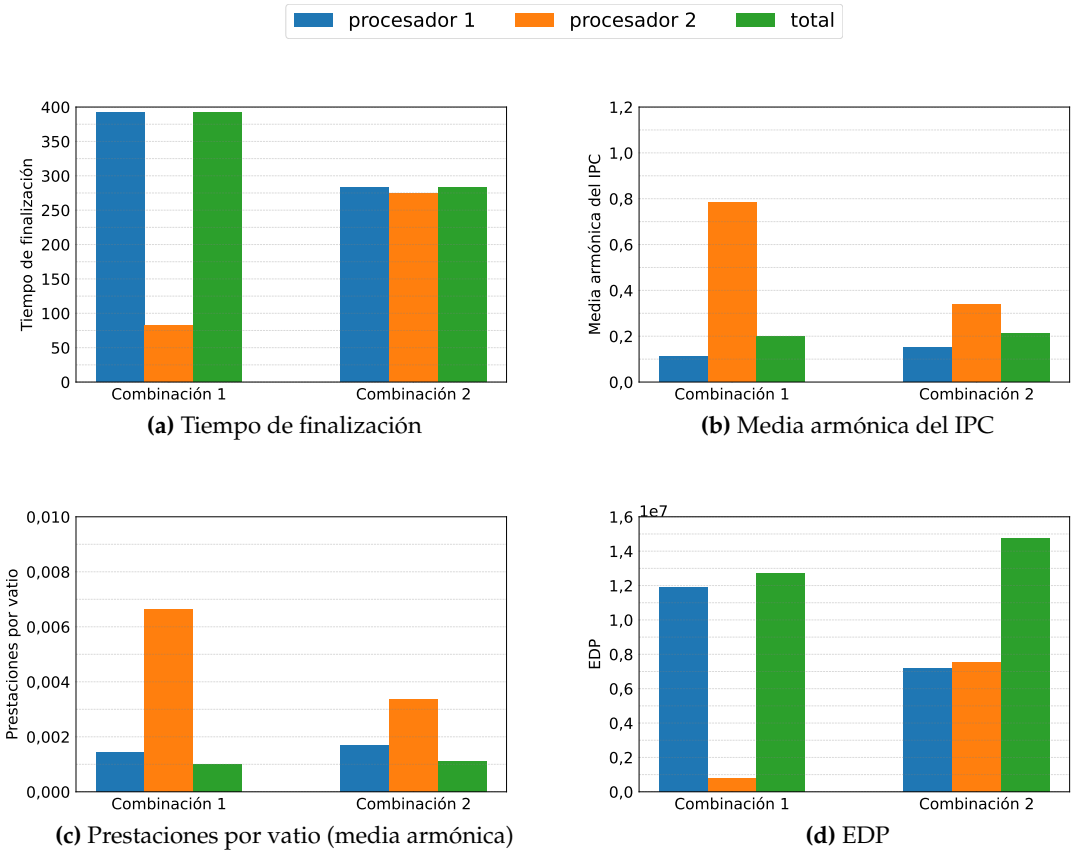
sador de los dos disponibles en el sistema, siendo el total de instancias de cada pareja 28 (número de núcleos por procesador).

La Tabla 5.1 muestra las diferentes combinaciones de parejas estudiadas. La primera combinación empareja 2 aplicaciones *low-scalable performance* (*mcf* y *xalanbcmk_r*) por un lado y *medium-scalable performance* (*cactuBSSN_r*) con *highly-scalable performance* (*calculix*) por el otro, mientras que la segunda y tercera combinación separan las aplicaciones *low-scalable performance* y se diferencian en el tipo de pareja asignada a cada una de ellas.

La Tabla 5.2 presenta el número de instancias que se van a ejecutar de cada aplicación dependiendo de la carga. En esta tabla se definen cinco *workloads* o cargas (desde *W1* hasta *W5*) compuestas por un total de 56 instancias elegidas entre cuatro aplicaciones seleccionadas de cada tipo. Nótese que tanto *W2* como *W4* utilizan el mismo número total de instancias para cada pareja (12 y 16). Sin embargo, la diferencia se encuentra en cómo se distribuyen estas instancias entre las diferentes aplicaciones. En el caso de *W2*, se asignan 12 instancias a las aplicaciones *mcf* y *cactuBSSN_r* y 16 a las otras. En cambio, en *W4*, 12 instancias se asignan a las aplicaciones *xalancbmk_r* y *calculix* y 16 al resto. Lo mismo ocurre con *W3* y *W5*, pero con 20 y 8 instancias.

Cada carga se ejecuta una vez por cada combinación, excepto para la tercera combinación, que sólo es aplicable a la carga *W1*, ya que no se dispone de suficientes núcleos por procesador para aplicar esta combinación a otras cargas. Por ejemplo, la carga *W3* con la tercera combinación ejecutaría en el primer procesador 40 instancias (20 de *mcf* y 20 de *cactuBSSN_r*), que son más de los 28 núcleos físicos de los que dispone un procesador en el sistema.

Respecto a las métricas estudiadas hasta este momento, para calcular la media de prestaciones (*IPC*) y prestaciones por vatio se ha usado la media aritmética. Sin embargo,



(a) Tiempo de finalización

(b) Media armónica del IPC

(c) Prestaciones por vatio (media armónica)

(d) EDP

Figura 5.26: Tiempo de finalización, media armónica, prestaciones por vatio y EDP para W3.

este tipo de media puede ser alterada fácilmente por valores extremos, proporcionando un valor distorsionado. Esto es especialmente relevante en estudios como el actual, en el que se ejecutan 56 aplicaciones, aumentando las posibilidades de valores extremos. Para lidiar con este problema, se utiliza como alternativa la media armónica.

Las Figuras 5.25 y 5.26 muestran: a) el tiempo de finalización, b) la media armónica del *IPC*, c) la media armónica de las prestaciones por vatio, y d) el *EDP* para las cargas *W2* y *W3* ejecutándose bajo las combinaciones 1 y 2. De todas las cargas y combinaciones, se muestran los resultados de las cargas *W2* y *W3* ejecutándose bajo las combinaciones 1 y 2 al ser estos resultados representativos de la totalidad de los experimentos.

Para ambas cargas, la primera combinación presenta sistemáticamente peores resultados en la mayoría de métricas. Esto se debe a que esta combinación empareja en el primer procesador dos aplicaciones *low-scalable performance* (*mcf* y *xalanbcmk_r*), que, como se mostró previamente, experimentan una mayor reducción de las prestaciones a medida que se aumenta la carga del procesador. Esta reducción de las prestaciones impacta directamente en el tiempo de finalización (Figuras 5.25a y 5.26a), *IPC* (Figuras 5.25b y 5.26b), y prestaciones por vatio (5.25c y 5.26c) presentados por el primer procesador. Por lo tanto, si la planificación se guía exclusivamente por estas métricas, se deduce que es mejor combinar en el mismo procesador aplicaciones de diferente tipo y separar las aplicaciones poco escalables, siguiendo políticas que hicieran efectivas las combinaciones 2 y 3 estudiadas.

Sin embargo, en el *EDP*, que tiene en cuenta tanto la energía consumida como la del tiempo de ejecución total de la carga, se observa un comportamiento diferente: la primera combinación presenta mejor *EDP* (es decir, un valor más bajo) total para ambas cargas. El motivo por el que el *EDP* total es inferior en la combinación 1 se desvela al analizar los resultados de cada procesador por separado, y, en particular, los resultados obtenidos por

el segundo procesador. Para la combinación 1, en el segundo procesador se emparejan las aplicaciones con mayor escalabilidad de la carga (*cactuBSSN_r* y *calculix*), lo que permite que finalicen mucho más rápido (ver Figuras 5.25a y 5.26a, combinación 1), lo cual tiene un gran impacto en la reducción del *EDP* del segundo procesador y del total (a costa del *EDP* exhibido por el primer procesador). Si bien es cierto que, en comparación a la primera combinación, la segunda reduce en gran medida el *EDP* del procesador 1, esta reducción no es suficiente para alcanzar un *EDP* total por debajo del de la primera combinación.

En conclusión, de los resultados se desprende que si el foco de la política de planificación a diseñar se encuentra principalmente en las prestaciones (por ejemplo, en el tiempo de finalización o en el *IPC*), es mejor utilizar políticas que distribuyan las aplicaciones entre procesadores intentando balancear el nivel de escalabilidad. Es decir, combinando en cada procesador aplicaciones con mayor y menor escalabilidad. En cambio, si lo que se desea es dar una mayor importancia a la energía consumida y al compromiso entre esta energía y el tiempo de finalización de la carga, resulta más adecuado combinar las aplicaciones más escalables en el mismo procesador, ya que finalizarán más rápido y proporcionarán un EDP total inferior.

CAPÍTULO 6
Conclusiones

En este capítulo se presentan las conclusiones del trabajo, así como la relación de este con los estudios cursados y trabajos futuros.

6.1 Visión general

Los procesadores de altas prestaciones comerciales de las últimas generaciones continúan evolucionando desde el punto de vista de contadores de consumo energético. Mientras hace apenas unos años el contador de consumo energético medía el consumo global del *processor package*, los procesadores recientes permiten medir distintos componentes del *package*. El objetivo de este trabajo ha sido realizar una primera toma de contacto con estos contadores, y utilizarlos para caracterizar tanto el consumo, como el consumo teniendo en cuenta las prestaciones.

En este trabajo se han caracterizado cuatro comportamientos típicos representativos de las 16 cargas *SPEC CPU* estudiadas. De los resultados, se pueden extraer las siguientes observaciones y *findings* principales.

- Dado que los núcleos son los que más energía consumen, la mejor combinación en términos de prestaciones por vatio viene dada por aquella que contenga las aplicaciones que alcancen las mayores prestaciones.
- La combinación con mayores prestaciones por vatio coincide con aquella en que ambos tipos de aplicaciones sufren menor *slowdown*.
- Un pequeño número de aplicaciones de memoria intensiva (o L3 intensiva), por ej. 4 en nuestros experimentos, supone un *slowdown* entre el 210 % y el 330 %, lo que inevitablemente lleva a unas pobres prestaciones por vatio. Más aún, el 80 % del *slowdown* que alcanzan las cargas compuestas por aplicaciones de distinto tipo es causado con sólo unas pocas instancias de aplicaciones de memoria intensiva.
- Combinar aplicaciones con baja y alta escalabilidad resulta en mejores prestaciones medias que ejecutar aplicaciones poco escalables por un lado y aplicaciones con una elevada escalabilidad por otro, debido al impacto tan negativo que ejecutar aplicaciones poco escalables juntas tiene en las prestaciones.

Las observaciones mencionadas conducen a las siguientes conclusiones. Un planificador orientado a proporcionar las máximas prestaciones debe cuidadosamente limitar el número de aplicaciones de memoria intensiva en la medida de lo posible, con el fin de evitar fuertes *slowdowns*. Además, se debe tratar de combinar aplicaciones con diferentes

niveles de escalabilidad, evitando así la gran reducción en las prestaciones que resulta de ejecutar una mayoría de aplicaciones con una baja escalabilidad juntas. Pero si, por el contrario, se da más importancia a la energía consumida y al compromiso entre esta y el tiempo de finalización (*EDP*), se debe tratar de combinar aplicaciones con un nivel de escalabilidad similar. Esto se debe a que las aplicaciones con mejor escalabilidad presentan un menor consumo de energía y tiempo de finalización, lo cual compensa los peores resultados de las aplicaciones con una menor escalabilidad.

6.2 Relación con los estudios cursados

A la hora de desarrollar este trabajo, el estudio de asignaturas como Estructura de Computadores (ETC), Arquitectura e Ingeniería de Computadores (AIC) y Arquitecturas Avanzadas (AAV) ha sido fundamental, proporcionando las bases para el conocimiento de arquitecturas *hardware*. Esto, a su vez, ha ayudado a sentar las bases para comprender las causas e implicaciones de los resultados obtenidos en los experimentos.

Asignaturas como Fundamentos de Sistemas Operativos (FSO) y Diseño de Sistemas Operativos (DSO) presentaron el lenguaje de programación *C*, que ha resultado ser muy útil en este trabajo al ayudar a comprender mejor el funcionamiento de las herramientas como el *manager Stratus*. Además, en estas asignaturas también se introdujo el sistema operativo *Linux* y se estudió en detalle el funcionamiento de un planificador de este, que no tenía en cuenta el consumo energético. Esto ha sido de gran ayuda para comprender mejor el objetivo y la importancia de este trabajo, al mostrar cómo, hasta hace poco, se le prestaba poca atención a la eficiencia energética.

La introducción al lenguaje de programación *Python*, impartida en la asignatura Sistemas Inteligentes (SIN), también ha resultado muy relevante, pues este lenguaje ha sido fundamental para el procesamiento y visualización de los resultados de los experimentos en este trabajo.

Dejando de lado asignaturas concretas, los trabajos y las memorias realizadas en las sesiones de prácticas han resultado útiles mejorando mis habilidades de expresión escrita.

En lo referente a las competencias transversales, las más destacadas son:

- **CT2.Aplicación y pensamiento práctico, CT9.Pensamiento crítico**: A lo largo del trabajo se han necesitado estas competencias para abordar y resolver los problemas surgidos tanto con el *framework Stratus* como en el desarrollando de los *scripts* utilizados para procesar el procesamiento de los ficheros de resultados. Además, han sido fundamentales para interpretar los resultados de los experimentos y tomar decisiones sobre los pasos a seguir.

- **CT3.Análisis y resolución de problemas, CT10.Conocimiento de problemas contemporáneos**: Este trabajo se ha centrado en un problema muy actual como es el de la eficiencia energética. Se han estudiado varias de las propuestas existentes en el ámbito de la planificación de aplicaciones y, tras observar algunas de sus carencias, se ha llevado a cabo este trabajo con el objetivo de identificar métricas que los planificadores podrían utilizar para maximizar la eficiencia energética, las prestaciones u obtener un equilibrio entre ambas.

- **CT13.Instrumental específico**: Durante la realización de los experimentos se han utilizado herramientas como el *framework Stratus*, así como *scripts* en *Python* y en *bash*.

6.3 Publicaciones derivadas y trabajos futuros

De este TFG ha derivado una publicación que se presentará en las **XXXIII Jornadas de Paralelismo** en Septiembre del 2023.

Este trabajo se enmarca dentro de un proyecto del plan nacional de Proyectos Estratégicos Orientados a la Transición Ecológica y a la Transición Digital. En este TFG hemos realizado un estudio de caracterización para ayudar al planificador a implementar la función objetivo; por ej. mejorar el *EDP*.

El siguiente paso consistirá en implementar esta función en el planificador. Por ejemplo, se ha comentado cómo sería posible utilizando información *off-line*, elegir qué aplicaciones se deberían ejecutar en cada procesador (*socket*). La idea es implementar en un planificador en modo usuario estas funciones. El funcionamiento sería el siguiente: al final de cada intervalo (por ej. 200 ms) se recogerían medidas y a partir de estas medidas, el planificador tomaría las decisiones y realizaría las planificaciones.

Hay que tener en cuenta que los procesadores continúan evolucionando y existen algunos modelos que permiten medidas a grano más fino, por ej., a nivel de núcleo. En este sentido, se pueden realizar trabajos con mejores beneficios. También existe la posibilidad de ampliar el trabajo considerando procesadores heterogéneos como el *Intel Alder Lake*.

Por otra parte, los beneficios alcanzados dependen de las características de la carga. En este primer trabajo nos hemos centrado en cargas típicas de *CPU* como las *SPEC*. En trabajos futuros nos centraremos en cargas paralelas con muchos hilos, cuyo número se puede alterar dinámicamente en tiempo de ejecución.

Bibliografía

[1] *cavium vulcan die (annotated)*. Disponible en: https://en.wikichip.org/wiki/File:cavium_vulcan_die_(annotated).png.

[2] *cavium vulcan die core cluster (annotated)*. Disponible en: https://en.wikichip.org/wiki/File:cavium_vulcan_die_core_cluster_(annotated).png.

[3] Descripción sobre Linux EAS Consultado en https://docs.kernel.org/scheduler/sched-energy.html#introduction.

[4] *Linux Kernel Documentation - Power Management: Energy Model*. Consultado el 23 de mayo de 2023. Disponible en: https://docs.kernel.org/power/energy-model.html.

[5] Descripción sobre la utilidad perf de Linux Consultado en https://perf.wiki.kernel.org/index.php/Main_Page.

[6] Documentación de sobre YAML Consultado en https://yaml.org/.

[7] Repositorio de la utilidad tx2mon Consultado en https://github.com/Marvell-SPBU/tx2mon/tree/master/tx2mon.

[8] Máxima longitud del nombre de un fichero permitida para varios sistemas de archivos Consultado en https://doc.owncloud.com/server/next/admin_manual/troubleshooting/path_filename_length.html.

[9] Entrada en el manual de Linux de la función perf_event_open Consultado en https://man7.org/linux/man-pages/man2/perf_event_open.2.html.

[10] *Marvell* 2019 *MARVELL® ThunderX2® PMU Events (Abridged)*.

[11] Descripción básica del procesador **ThunderX2**. Consultado en https://en.wikichip.org/wiki/cavium/thunderx2.

[12] Descripción de la microarquitectura Vulcan. Consultado en https://en.wikichip.org/wiki/cavium/microarchitectures/vulcan.

[13] Andreas Merkel y Frank Bellosa. "Balancing Power Consumption in Multiprocessor Systems". *SIGOPS Oper. Syst. Rev.*, vol. 40, no. 4, pp. 403-414, 2006. DOI: 10.1145/1218063.1217974.

[14] Yuhao Zhu y Vijay Janapa Reddi. "High-performance and energy-efficient mobile web browsing on big/little systems". En: *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, pp. 13-24, 2013. DOI: 10.1109/HPCA.2013.6522303.

[15] Kenzo Van Craeynest, Shoaib Akram, Wim Heirman, Aamer Jaleel y Lieven Eeckhout. "Fairness-aware scheduling on single-ISA heterogeneous multi-cores". *Proceedings of PACT*, pp. 177-187, 2013.

[16] Andreas Merkel, Jan Stoess, and Frank Bellosa. Resource-Conscious Scheduling for Energy Efficiency on Multicore Processors.Ïn *Proceedings of the 5th European Conference on Computer Systems*, EuroSys '10, pages 153–166, Paris, France, 2010. Association for Computing Machinery. doi: https://doi.org/10.1145/1755913.1755930.

[17] Khaled M. Attia, Mostafa A. El-Hosseini, and Hesham A. Ali. "Dynamic power management techniques in multi-core architectures: A survey study." *Ain Shams Engineering Journal*, vol. 8, no. 3, pp. 445-456, 2017. doi: https://doi.org/10.1016/j.asej.2015.08.010.

[18] Mwaffaq Otoom, Pedro Trancoso, Hisham Almasaeid, and Mohammad Alzubaidi. "Scalable and Dynamic Global Power Management for Multicore Chips." In *Proceedings of the 6th Workshop on Parallel Programming and Run-Time Management Techniques for Many-Core Architectures*, pp. 25-30. Association for Computing Machinery, 2015. doi: https://doi.org/10.1145/2701310.2701312.

[19] S. Zhuravlev, J. C. Saez, S. Blagodurov, A. Fedorova and M. Prieto, "Survey of Energy-Cognizant Scheduling Techniques," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 7, pp. 1447-1464, 2013, doi: 10.1109/TPDS.2012.20.

[20] Pons, Lucía and Petit, Salvador and Pons, Julio and Gómez, María E. and Huang, Chaoyi and Sahuquillo, Julio "Stratus: A Hardware/Software Infrastructure for Controlled Cloud Research", *31th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*.

[21] Lucía Pons Escat. Mejora de prestaciones de procesadores comerciales utilizando la tecnología Intel CAT, 2018.

[22] Marta Navarro Edo. Políticas de asignación de aplicaciones a núcleos en un Intel Xeon utilizando la metodología Top-Down, 2020.

APÉNDICE A

Ficheros de configuración del *manager Stratus* utilizados en los experimentos

A.1 Ejecución de las aplicaciones individualmente

```
- [calculix]
- [deepsjeng_r]
- [exchange2_r]
- [imagick_r]
- [leela_r]
- [nab_r]
- [namd_r]
- [povray_r]
- [cam4_r]
- [mcf]
- [wrf_r]
- [roms_r]
- [omnetpp_r]
- [xalancbmk_r]
- [blender_r]
- [cactuBSSN_r]
```

Listing A.1: spec-06-17.yaml

```
<%include file="applications.mako"/>
<%include file="instr_60s_500ms.mako"/>

<%!
lc = {0:56, 1:57, 2:58, 3:59, 4:60, 5:61, 6:62, 7:63, 8:64, 9:65, 10:66, 11:67,
     12:68, 13:69, 14:70, 15:71, 16:72, 17:73, 18:74, 19:75, 20:76, 21:77,
    22:78, 23:79, 24:80, 25:81, 26:82, 27:83, 28:84}
%>

tasks:
  % for app in apps:
  - app: *${app}
    max_instr: *${app}_mi
    cpus: [${lc[loop.index]}]
  % endfor

cmd:
    ti: 0.1
    mi: 50000
    event: ["INST_RETIRED,cycles,INST_SPEC,r0052,r0053,r000d,r0017"]
```

Listing A.2: template.mako

```
cpupower -c all frequency-set -r -g userspace
cpupower -c all frequency-set --min 1000MHz
cpupower -c all frequency-set -f 1000MHz
cpupower -c 56 frequency-set -f 2500MHz
sudo bash /home/icalqui/manager/scripts/launch.bash spec-06-17.yaml --max-rep 1
```

Listing A.3: launch_apps.bash

A.2 Ejecución de varias instancias del mismo tipo

```
- [calculix,calculix,calculix,calculix,calculix,calculix,calculix]
- [deepsjeng_r,deepsjeng_r,deepsjeng_r,deepsjeng_r,deepsjeng_r,deepsjeng_r,
    deepsjeng_r]
- [exchange2_r,exchange2_r,exchange2_r,exchange2_r,exchange2_r,exchange2_r,
    exchange2_r]
- [imagick_r,imagick_r,imagick_r,imagick_r,imagick_r,imagick_r,imagick_r]
- [leela_r,leela_r,leela_r,leela_r,leela_r,leela_r,leela_r]
- [nab_r,nab_r,nab_r,nab_r,nab_r,nab_r,nab_r]
- [namd_r,namd_r,namd_r,namd_r,namd_r,namd_r,namd_r]
- [povray_r,povray_r,povray_r,povray_r,povray_r,povray_r,povray_r]
- [cam4_r,cam4_r,cam4_r,cam4_r,cam4_r,cam4_r,cam4_r]
- [mcf,mcf,mcf,mcf,mcf,mcf,mcf]
- [wrf_r,wrf_r,wrf_r,wrf_r,wrf_r,wrf_r,wrf_r]
- [roms_r,roms_r,roms_r,roms_r,roms_r,roms_r,roms_r]
- [omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r]
- [xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
    xalancbmk_r]
- [blender_r,blender_r,blender_r,blender_r,blender_r,blender_r,blender_r]
- [- [cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
    cactuBSSN_r]]
```

Listing A.4: 7-spec-06-17.yaml

```
- [calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
    calculix,calculix,calculix,calculix,calculix,calculix]
- [deepsjeng_r,deepsjeng_r,deepsjeng_r,deepsjeng_r,deepsjeng_r,deepsjeng_r,
    deepsjeng_r,deepsjeng_r,deepsjeng_r,deepsjeng_r,deepsjeng_r,deepsjeng_r,
    deepsjeng_r,deepsjeng_r]
- [exchange2_r,exchange2_r,exchange2_r,exchange2_r,exchange2_r,exchange2_r,
    exchange2_r,exchange2_r,exchange2_r,exchange2_r,exchange2_r,exchange2_r,
    exchange2_r,exchange2_r]
- [imagick_r,imagick_r,imagick_r,imagick_r,imagick_r,imagick_r,imagick_r,
    imagick_r,imagick_r,imagick_r,imagick_r,imagick_r,imagick_r,imagick_r]
- [leela_r,leela_r,leela_r,leela_r,leela_r,leela_r,leela_r,leela_r,leela_r,
    leela_r,leela_r,leela_r,leela_r,leela_r]
- [nab_r,nab_r,nab_r,nab_r,nab_r,nab_r,nab_r,nab_r,nab_r,nab_r,nab_r,nab_r,
    nab_r,nab_r]
- [namd_r,namd_r,namd_r,namd_r,namd_r,namd_r,namd_r,namd_r,namd_r,namd_r,namd_r
    ,namd_r,namd_r,namd_r]
- [povray_r,povray_r,povray_r,povray_r,povray_r,povray_r,povray_r,povray_r,
    povray_r,povray_r,povray_r,povray_r,povray_r,povray_r]
- [cam4_r,cam4_r,cam4_r,cam4_r,cam4_r,cam4_r,cam4_r,cam4_r,cam4_r,cam4_r,cam4_r
    ,cam4_r,cam4_r,cam4_r]
- [mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf]
- [wrf_r,wrf_r,wrf_r,wrf_r,wrf_r,wrf_r,wrf_r,wrf_r,wrf_r,wrf_r,wrf_r,wrf_r,
    wrf_r,wrf_r]
- [roms_r,roms_r,roms_r,roms_r,roms_r,roms_r,roms_r,roms_r,roms_r,roms_r,roms_r
    ,roms_r,roms_r,roms_r]
```

```
13 - [omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r,
      omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r]
14 - [xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
      xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
      xalancbmk_r,xalancbmk_r]
15 - [blender_r,blender_r,blender_r,blender_r,blender_r,blender_r,blender_r,
      blender_r,blender_r,blender_r,blender_r,blender_r,blender_r,blender_r]
16 - [cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
      cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
      cactuBSSN_r,cactuBSSN_r]
```

Listing A.5: 14-spec-06-17.yaml

```
1  - [calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
      calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
      calculix,calculix,calculix,calculix,calculix]
2  - [deepsjeng_r,deepsjeng_r,deepsjeng_r,deepsjeng_r,deepsjeng_r,deepsjeng_r,
      deepsjeng_r,deepsjeng_r,deepsjeng_r,deepsjeng_r,deepsjeng_r,deepsjeng_r,
      deepsjeng_r,deepsjeng_r,deepsjeng_r,deepsjeng_r,deepsjeng_r,deepsjeng_r,
      deepsjeng_r,deepsjeng_r,deepsjeng_r]
3  - [exchange2_r,exchange2_r,exchange2_r,exchange2_r,exchange2_r,exchange2_r,
      exchange2_r,exchange2_r,exchange2_r,exchange2_r,exchange2_r,exchange2_r,
      exchange2_r,exchange2_r,exchange2_r,exchange2_r,exchange2_r,exchange2_r,
      exchange2_r,exchange2_r,exchange2_r]
4  - [imagick_r,imagick_r,imagick_r,imagick_r,imagick_r,imagick_r,imagick_r,
      imagick_r,imagick_r,imagick_r,imagick_r,imagick_r,imagick_r,imagick_r,
      imagick_r,imagick_r,imagick_r,imagick_r,imagick_r,imagick_r,imagick_r]
5  - [leela_r,leela_r,leela_r,leela_r,leela_r,leela_r,leela_r,leela_r,leela_r,
      leela_r,leela_r,leela_r,leela_r,leela_r,leela_r,leela_r,leela_r,leela_r,
      leela_r,leela_r,leela_r]
6  - [nab_r,nab_r,nab_r,nab_r,nab_r,nab_r,nab_r,nab_r,nab_r,nab_r,nab_r,nab_r,
      nab_r,nab_r,nab_r,nab_r,nab_r,nab_r,nab_r,nab_r,nab_r]
7  - [namd_r,namd_r,namd_r,namd_r,namd_r,namd_r,namd_r,namd_r,namd_r,namd_r,namd_r
      ,namd_r,namd_r,namd_r,namd_r,namd_r,namd_r,namd_r,namd_r,namd_r,namd_r]
8  - [povray_r,povray_r,povray_r,povray_r,povray_r,povray_r,povray_r,povray_r,
      povray_r,povray_r,povray_r,povray_r,povray_r,povray_r,povray_r,povray_r,
      povray_r,povray_r,povray_r,povray_r,povray_r]
9  - [cam4_r,cam4_r,cam4_r,cam4_r,cam4_r,cam4_r,cam4_r,cam4_r,cam4_r,cam4_r,cam4_r
      ,cam4_r,cam4_r,cam4_r,cam4_r,cam4_r,cam4_r,cam4_r,cam4_r,cam4_r,cam4_r]
10 - [mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,
      mcf,mcf]
11 - [wrf_r,wrf_r,wrf_r,wrf_r,wrf_r,wrf_r,wrf_r,wrf_r,wrf_r,wrf_r,wrf_r,
      wrf_r,wrf_r,wrf_r,wrf_r,wrf_r,wrf_r,wrf_r,wrf_r,wrf_r,wrf_r]
12 - [roms_r,roms_r,roms_r,roms_r,roms_r,roms_r,roms_r,roms_r,roms_r,roms_r,roms_r
      ,roms_r,roms_r,roms_r,roms_r,roms_r,roms_r,roms_r,roms_r,roms_r,roms_r]
13 - [omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r,
      omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r,
      omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r]
14 - [xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
      xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
      xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
      xalancbmk_r,xalancbmk_r,xalancbmk_r]
15 - [blender_r,blender_r,blender_r,blender_r,blender_r,blender_r,blender_r,
      blender_r,blender_r,blender_r,blender_r,blender_r,blender_r,blender_r,
      blender_r,blender_r,blender_r,blender_r,blender_r,blender_r,blender_r]
16 - [cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
      cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
      cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
      cactuBSSN_r,cactuBSSN_r,cactuBSSN_r]
```

Listing A.6: 21-spec-06-17.yaml

```
1  - [calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
      calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
```

```
    calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
    calculix,calculix,calculix,calculix]
- [deepsjeng_r,deepsjeng_r,deepsjeng_r,deepsjeng_r,deepsjeng_r,deepsjeng_r,
    deepsjeng_r,deepsjeng_r,deepsjeng_r,deepsjeng_r,deepsjeng_r,deepsjeng_r,
    deepsjeng_r,deepsjeng_r,deepsjeng_r,deepsjeng_r,deepsjeng_r,deepsjeng_r,
    deepsjeng_r,deepsjeng_r,deepsjeng_r,deepsjeng_r,deepsjeng_r,deepsjeng_r,
    deepsjeng_r,deepsjeng_r,deepsjeng_r,deepsjeng_r,deepsjeng_r,deepsjeng_r,
    deepsjeng_r,deepsjeng_r,deepsjeng_r,deepsjeng_r]
- [exchange2_r,exchange2_r,exchange2_r,exchange2_r,exchange2_r,exchange2_r,
    exchange2_r,exchange2_r,exchange2_r,exchange2_r,exchange2_r,exchange2_r,
    exchange2_r,exchange2_r,exchange2_r,exchange2_r,exchange2_r,exchange2_r,
    exchange2_r,exchange2_r,exchange2_r,exchange2_r,exchange2_r,exchange2_r,
    exchange2_r,exchange2_r,exchange2_r,exchange2_r,exchange2_r,exchange2_r,
    exchange2_r,exchange2_r,exchange2_r,exchange2_r]
- [imagick_r,imagick_r,imagick_r,imagick_r,imagick_r,imagick_r,imagick_r,
    imagick_r,imagick_r,imagick_r,imagick_r,imagick_r,imagick_r,imagick_r,
    imagick_r,imagick_r,imagick_r,imagick_r,imagick_r,imagick_r,imagick_r,
    imagick_r,imagick_r,imagick_r,imagick_r,imagick_r,imagick_r,imagick_r]
- [leela_r,leela_r,leela_r,leela_r,leela_r,leela_r,leela_r,leela_r,leela_r,
    leela_r,leela_r,leela_r,leela_r,leela_r,leela_r,leela_r,leela_r,leela_r,
    leela_r,leela_r,leela_r,leela_r,leela_r,leela_r,leela_r,leela_r,leela_r,
    leela_r]
- [nab_r,nab_r,nab_r,nab_r,nab_r,nab_r,nab_r,nab_r,nab_r,nab_r,nab_r,
    nab_r,nab_r,nab_r,nab_r,nab_r,nab_r,nab_r,nab_r,nab_r,nab_r,nab_r,
    nab_r,nab_r,nab_r,nab_r]
- [namd_r,namd_r,namd_r,namd_r,namd_r,namd_r,namd_r,namd_r,namd_r,namd_r
    ,namd_r,namd_r,namd_r,namd_r,namd_r,namd_r,namd_r,namd_r,namd_r,namd_r,
    namd_r,namd_r,namd_r,namd_r,namd_r,namd_r,namd_r]
- [povray_r,povray_r,povray_r,povray_r,povray_r,povray_r,povray_r,povray_r,
    povray_r,povray_r,povray_r,povray_r,povray_r,povray_r,povray_r,povray_r,
    povray_r,povray_r,povray_r,povray_r,povray_r,povray_r,povray_r,povray_r,
    povray_r,povray_r,povray_r,povray_r]
- [cam4_r,cam4_r,cam4_r,cam4_r,cam4_r,cam4_r,cam4_r,cam4_r,cam4_r,cam4_r,cam4_r
    ,cam4_r,cam4_r,cam4_r,cam4_r,cam4_r,cam4_r,cam4_r,cam4_r,cam4_r,
    cam4_r,cam4_r,cam4_r,cam4_r,cam4_r,cam4_r,cam4_r]
- [mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,
    mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf]
- [wrf_r,wrf_r,wrf_r,wrf_r,wrf_r,wrf_r,wrf_r,wrf_r,wrf_r,wrf_r,wrf_r,wrf_r,
    wrf_r,wrf_r,wrf_r,wrf_r,wrf_r,wrf_r,wrf_r,wrf_r,wrf_r,wrf_r,wrf_r,wrf_r,
    wrf_r,wrf_r,wrf_r,wrf_r]
- [roms_r,roms_r,roms_r,roms_r,roms_r,roms_r,roms_r,roms_r,roms_r,roms_r
    ,roms_r,roms_r,roms_r,roms_r,roms_r,roms_r,roms_r,roms_r,roms_r,roms_r,
    roms_r,roms_r,roms_r,roms_r,roms_r,roms_r,roms_r]
- [omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r,
    omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r,
    omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r,
    omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r,omnetpp_r]
- [xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
    xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
    xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
    xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
    xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r]
- [blender_r,blender_r,blender_r,blender_r,blender_r,blender_r,blender_r,
    blender_r,blender_r,blender_r,blender_r,blender_r,blender_r,blender_r,
    blender_r,blender_r,blender_r,blender_r,blender_r,blender_r,blender_r,
    blender_r,blender_r,blender_r,blender_r,blender_r,blender_r]
- [cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
    cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
    cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
    cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
    cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r]
```

Listing A.7: 28-spec-06-17.yaml

```
<%include file="applications.mako"/>
<%include file="instr_60s_500ms.mako"/>
```

```
<%!
lc = {0:56, 1:57, 2:58, 3:59, 4:60, 5:61, 6:62, 7:63, 8:64, 9:65, 10:66, 11:67,
     12:68, 13:69, 14:70, 15:71, 16:72, 17:73, 18:74, 19:75, 20:76, 21:77,
    22:78, 23:79, 24:80, 25:81, 26:82, 27:83, 28:84}
%>


tasks:
  % for app in apps:
  - app: *${app}
    max_instr: *${app}_mi
    cpus: [${lc[loop.index]}]
  % endfor

cmd:
    ti: 0.1
    mi: 50000
    event: ["INST_RETIRED,cycles,INST_SPEC"]
```

Listing A.8: template.mako

```
cpupower -c all frequency-set -r -g userspace
cpupower -c all frequency-set --min 1000MHz
cpupower -c all frequency-set -f 1000MHz
cpupower -c 56-62 frequency-set -f 2500MHz
sudo bash /home/icalqui/original-manager/scripts/launch.bash 7-spec-06-17.yaml
    --max-rep 1

cpupower -c all frequency-set -r -g userspace
cpupower -c all frequency-set --min 1000MHz
cpupower -c all frequency-set -f 1000MHz
cpupower -c 56-69 frequency-set -f 2500MHz
sudo bash /home/icalqui/original-manager/scripts/launch.bash 14-spec-06-17.yaml
    --max-rep 1

cpupower -c all frequency-set -r -g userspace
cpupower -c all frequency-set --min 1000MHz
cpupower -c all frequency-set -f 1000MHz
cpupower -c 56-76 frequency-set -f 2500MHz
sudo bash /home/icalqui/original-manager/scripts/launch.bash 21-spec-06-17.yaml
    --max-rep 1

cpupower -c all frequency-set -r -g userspace
cpupower -c all frequency-set --min 1000MHz
cpupower -c all frequency-set -f 1000MHz
cpupower -c 56-83 frequency-set -f 2500MHz
sudo bash /home/icalqui/original-manager/scripts/launch.bash 28-spec-06-17.yaml
    --max-rep 1
```

Listing A.9: launch_apps.bash

A.3 Ejecución de varias instancias de distinto tipo

```
    - [mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,
       mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf
calculix,calculix,calculix,calculix,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf
    ,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf
calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,mcf,mcf
    ,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf
```

```
4 calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
      calculix,calculix,calculix,calculix,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf
      ,mcf,mcf,mcf,mcf,mcf,mcf
5 calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
      calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,mcf
      ,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf
6 calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
      calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
      calculix,calculix,calculix,calculix,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf
7 calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
      calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
      calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,mcf
      ,mcf,mcf,mcf
8 calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
      calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
      calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
      calculix,calculix,calculix,calculix
9 ]
10 - [xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
      xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
      xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
      xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
      xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r
11 calculix,calculix,calculix,calculix,xalancbmk_r,xalancbmk_r,xalancbmk_r,
      xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
      xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
      xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
      xalancbmk_r,xalancbmk_r,xalancbmk_r
12 calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
      xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
      xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
      xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
      xalancbmk_r,xalancbmk_r
13 calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
      calculix,calculix,calculix,calculix,xalancbmk_r,xalancbmk_r,xalancbmk_r,
      xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
      xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
      xalancbmk_r
14 calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
      calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
      xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
      xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r
15 calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
      calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
      calculix,calculix,calculix,calculix,calculix,xalancbmk_r,xalancbmk_r,xalancbmk_r,
      xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r
16 calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
      calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
      calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
      xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r
17 calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
      calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
      calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
      calculix,calculix,calculix,calculix
18 ]
19 - [cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
      cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
      cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
      cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
      cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r
20 calculix,calculix,calculix,calculix,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
      cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
      cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
```

```
    cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
    cactuBSSN_r,cactuBSSN_r,cactuBSSN_r
21  calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
    cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
    cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
    cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
    cactuBSSN_r,cactuBSSN_r
22  calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
    calculix,calculix,calculix,calculix,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
    cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
    cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
    cactuBSSN_r
23  calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
    calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
    cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
    cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r
24  calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
    calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
    calculix,calculix,calculix,calculix,calculix,calculix,cactuBSSN_r,cactuBSSN_r,
    cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r
25  calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
    calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
    calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
    cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r
26  calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
    calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
    calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
    calculix,calculix,calculix,calculix
27  ]
28  - [cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
    cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
    cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
    cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
    cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r
29  mcf,mcf,mcf,mcf,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
    cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
    cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
    cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
    cactuBSSN_r
30  mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r
    ,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
    cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
    cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r
31  mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,cactuBSSN_r,cactuBSSN_r,
    cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
    cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
    cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r
32  mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,cactuBSSN_r,
    cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
    cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r
33  mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf
    ,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
    cactuBSSN_r,cactuBSSN_r,cactuBSSN_r
34  mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf
    ,mcf,mcf,mcf,mcf,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r
35  mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf
    ,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf
36  ]
37  - [xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
    xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
    xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
    xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
    xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r
```

```
38 cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,xalancbmk_r,xalancbmk_r,
       xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
       xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
       xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
       xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r
39 cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
       cactuBSSN_r,cactuBSSN_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
       xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
       xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
       xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r
40 cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
       cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
       xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
       xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
       xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r
41 cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
       cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
       cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,xalancbmk_r,xalancbmk_r,
       xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
       xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r
42 cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
       cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
       cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
       cactuBSSN_r,cactuBSSN_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
       xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r
43 cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
       cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
       cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
       cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
       xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r
44 cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
       cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
       cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
       cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
       cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r
45 ]
46 - [xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
       xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
       xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
       xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
       xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r
47 mcf,mcf,mcf,mcf,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
       xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
       xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
       xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
       xalancbmk_r
48 mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r
       ,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
       xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
       xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r
49 mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,xalancbmk_r,xalancbmk_r,
       xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
       xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
       xalancbmk_r,xalancbmk_r,xalancbmk_r
50 mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,xalancbmk_r,
       xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
       xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r
51 mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf
       ,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
       xalancbmk_r,xalancbmk_r,xalancbmk_r
52 mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf
       ,mcf,mcf,mcf,mcf,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r
53 mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf
       ,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf
```

```
]
```

Listing A.10: spec-06-17.yaml

```
<%include file="applications.mako"/>
<%include file="instr_60s_500ms.mako"/>


<%!
lc = {0:56, 1:57, 2:58, 3:59, 4:60, 5:61, 6:62, 7:63, 8:64, 9:65, 10:66, 11:67,
     12:68, 13:69, 14:70, 15:71, 16:72, 17:73, 18:74, 19:75, 20:76, 21:77,
    22:78, 23:79, 24:80, 25:81, 26:82, 27:83, 28:84}
%>


tasks:
  % for app in apps:
  - app: *${app}
    max_instr: *${app}_mi
    cpus: [${lc[loop.index]}]
  % endfor

cmd:
    ti: 0.1
    mi: 50000
    event: ["INST_RETIRED,cycles,INST_SPEC"]
```

Listing A.11: template.mako

```
cpupower -c all frequency-set -r -g userspace
cpupower -c all frequency-set --min 1000MHz
cpupower -c all frequency-set -f 1000MHz
cpupower -c 56-83 frequency-set -f 2500MHz
sudo bash /home/icalqui/original-manager/scripts/launch.bash spec-06-17.yaml --
    max-rep 1
```

Listing A.12: launch_apps.sh

A.4 Ejecución de combinaciones de parejas

```
- [mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,xalancbmk_r,xalancbmk_r,xalancbmk_r,
    xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
    xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
    xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r
mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,xalancbmk_r,xalancbmk_r,
    xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
    xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
    xalancbmk_r,xalancbmk_r
mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,xalancbmk_r,
    xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
    xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r
mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf
    ,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
    xalancbmk_r,xalancbmk_r,xalancbmk_r
]
- [calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
    cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
    cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
    cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
    cactuBSSN_r,cactuBSSN_r
```

```
7 calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
      calculix,calculix,calculix,calculix,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
      cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
      cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
      cactuBSSN_r
8 calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
      calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
      cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
      cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r
9 calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
      calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
      calculix,calculix,calculix,calculix,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
      cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r
10 ]
11 - [mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,calculix,calculix,calculix,calculix,calculix
      ,calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
      calculix,calculix,calculix,calculix,calculix,calculix,calculix
12 mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,calculix,calculix,calculix,
      calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
      calculix,calculix,calculix,calculix,calculix
13 mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,calculix,
      calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
      calculix,calculix,calculix,calculix
14 mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf
      ,calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix
15 ]
16 - [xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
      xalancbmk_r,xalancbmk_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
      cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
      cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
      cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r
17 xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
      xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
      cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
      cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
      cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r
18 xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
      xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
      xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,cactuBSSN_r,cactuBSSN_r,
      cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
      cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r
19 xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
      xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
      xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
      xalancbmk_r,xalancbmk_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
      cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r
20 ]
21 - [mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,xalancbmk_r,
      xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
      xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
      xalancbmk_r
22 ]
23 - [calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
      calculix,calculix,calculix,calculix,calculix,calculix,cactuBSSN_r,
      cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
      cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
      cactuBSSN_r
24 ]
25 - [mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,mcf,calculix,calculix,
      calculix,calculix,calculix,calculix,calculix,calculix,calculix,calculix,
      calculix,calculix,calculix,calculix
26 ]
27 - [xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
      xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,xalancbmk_r,
```

```
    xalancbmk_r,xalancbmk_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
    cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,
    cactuBSSN_r,cactuBSSN_r,cactuBSSN_r,cactuBSSN_r
]
```

Listing A.13: spec-06-17.yaml

```
<%include file="applications.mako"/>
<%include file="instr_60s_500ms.mako"/>


<%!
lc = {0:56, 1:57, 2:58, 3:59, 4:60, 5:61, 6:62, 7:63, 8:64, 9:65, 10:66, 11:67,
     12:68, 13:69, 14:70, 15:71, 16:72, 17:73, 18:74, 19:75, 20:76, 21:77,
    22:78, 23:79, 24:80, 25:81, 26:82, 27:83, 28:84}
%>


tasks:
  % for app in apps:
  - app: *${app}
    cpus: [${lc[loop.index]}]
  % endfor

cmd:
    ti: 0.1
    mi: 4000
    event: ["INST_RETIRED,cycles,INST_SPEC"]
```

Listing A.14: template.mako

```
cpupower -c all frequency-set -r -g userspace
cpupower -c all frequency-set --min 1000MHz
cpupower -c all frequency-set -f 1000MHz
cpupower -c 56-83 frequency-set -f 2500MHz
sudo bash /home/icalqui/original-manager/scripts/launch.bash spec-06-17.yaml --
    max-rep 1
```

Listing A.15: launch_apps.sh

APÉNDICE B
Objetivos de Desarrollo Sostenible

En este apéndice se comenta la relación de este trabajo con los Objetivos de Desarrollo Sostenible, mostrados en la Tabla B.1.

Objetivos de Desarrollo Sostenible	Alto	Medio	Bajo	No procede
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**				X
ODS 4. **Educación de calidad.**				X
ODS 5. **Igualdad de género.**				X
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**	X			
ODS 8. **Trabajo decente y crecimiento económico.**				X
ODS 9. **Industria, innovación e infraestructuras.**				X
ODS 10. **Reducción de las desigualdades.**				X
ODS 11. **Ciudades y comunidades sostenibles.**		X		
ODS 12. **Producción y consumo responsables.**	X			
ODS 13. **Acción por el clima.**	X			
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**				X
ODS 16. **Paz, justicia e instituciones sólidas.**				X
ODS 17. **Alianzas para lograr objetivos.**				X

Tabla B.1: ODS

Desde hace años, la tecnología se ha convertido en una parte integral de nuestras vidas y nuestro mundo depende completamente de ella. No solo resulta difícil imaginar un mundo sin tecnología, sino que también se ha vuelto un requisito indispensable dominarla, al menos a nivel de usuario, en nuestra sociedad. Desde los teléfonos móviles hasta los ordenadores personales, pasando por el Internet de las cosas, vivimos en un mundo completamente interconectado.

Pese a los innumerables beneficios de esta, también ha provocado problemas, como su impacto ambiental. La cantidad de dispositivos electrónicos es tan grande que, aunque su consumo individual no sea elevado, a nivel mundial sí lo es. Es por esta razón que la eficiencia de los dispositivos resulta vital. Ya que una pequeña mejora en la eficiencia

puede tener un impacto considerable a nivel global.

En este trabajo se ha estudiado y caracterizado, desde el punto de vista del consumo, el comportamiento de *benchmarks* de la *suite SPEC*, con el objetivo de identificar métricas que puedan ser usadas para determinar la mejor manera de planificar aplicaciones, ya sea para maximizar sus prestaciones, mejorar eficiencia energética o alcanzar un equilibrio entre ambos. Debido a la naturaleza del trabajo, este se encuentra alineado con los objetivos de varios ODS.

En primer lugar, podemos destacar el **ODS 7 (Energía asequible y no contaminante)**, que está estrechamente ligado al trabajo realizado. Entre las metas de este objetivo se encuentra la duplicación de la tasa mundial de mejora de la eficiencia energética para el año 2030. Esto guarda una relación significativa con el trabajo, ya que la planificación eficiente de aplicaciones puede incrementar la eficiencia energética de los equipos. Este impacto es especialmente relevante debido a la gran cantidad de dispositivos que podrían beneficiarse de una planificación de aplicaciones, contribuyendo así a un menor consumo de energía.

Otro ODS que también está relacionado es el **ODS 11 (Ciudades y comunidades sostenibles)**. Aunque la conexión de nuestro proyecto con este objetivo no es tan evidente como en el caso anterior, también están relacionados. El trabajo realizado guarda relación con una de las metas propuestas en el ODS, que es la reducción del impacto ambiental negativo per cápita en las ciudades. La mejora de la eficiencia energética de los procesadores puede ser especialmente beneficiosa en este aspecto. Dada la elevada densidad de dispositivos tecnológicos en las ciudades, mejoras en la eficiencia, aunque solo fuese en una parte de estos, tendrían un gran impacto.

Otro objetivo con el que también guarda relación es el **ODS 12 (Producción y consumo responsables)**. Este ODS pretende hacer más y mejor con menos. Reducir el gran consumo de recursos que se realiza. La mejora en la eficiencia energética puede ayudar a reducir el consumo de energía y a facilitar el acceso de esta, lo que supondría un aprovechamiento mayor de la energía.

Por último, el **ODS 13 (Acción por el clima)** está muy relacionado con este tema, ya que la mejora en la eficiencia energética permite reducir el consumo de energía, y por consiguiente, reducir las más que posibles emisiones asociadas a la generación de esa energía con su consiguiente impacto en el cambio climático. En definitiva, se ha podido comprobar cómo la temática de este trabajo se alinea con varios de los ODS. En particular, con aquellos relacionados con la eficiencia energética y el consumo responsable de la energía.