Universidad Complutense de Madrid

Facultad de Informática

Framework para la extracción automática de firmas de aplicaciones HPC para estimación de energía

Autor **Jorge García Villanueva**

Directores de proyecto **José Luis Ayala Rodrigo**
Juan Carlos Salinas Hilburg

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Septiembre de 2019

Índice general

Resumen

Resumen

La reciente expansión de sectores que procesan grandes cantidades de datos (Inteligencia artificial, análisis, banca, etc.) ha elevado en los últimos años el consumo de energía, sobretodo en los centros de datos destinados a la computación, pero no solo eso, sino también gastos derivados al mantenimiento de los mismos. En un centro de datos podemos encontrarnos tres tipos principales de consumo energético: Gastos de refrigeración, de cómputo o equipos TI (Tecnología de la Información) e infraestructuras, ordenados según su contribución al propio consumo energético.

A pesar de los recientes avances en términos de optimización del consumo de energía, los centros de datos siguen generando un alto consumo de energía. Para mejorar la eficiencia energética se han desarrollado técnicas como *power budgeting*, *power capping* ó *resource management*, asociadas al consumo generado por el sector TI. Estas técnicas necesitan el dato de estimación de energía de las aplicaciones que se van a ejecutar. Las técnicas tradicionales obtienen el dato de consumo energético mediante un *profiling* dinámico de toda la ejecución de la aplicación. Esto no es asumible en escenarios de *High Performance Computing* (HPC) debido a que las ejecuciones son muy largas (horas o días).

En un trabajo previo se ha desarrollado una metodología capaz de realizar una estimación rápida del consumo de energía para aplicaciones de tipo HPC sin la necesidad de ejecutar toda la aplicación y mediante un proceso automático [1]. En esta metodología se describe la aplicación de un *profiling* dinámico sobre una versión reducida de la aplicación, la cuál se define en la metodología desarrollada como firma de la aplicación.

Por tanto, en este trabajo se ha desarrollado un *framework* capaz de implementar la metodología de estimación rápida de energía mencionada anteriormente. El *framework* funciona de forma automática sin la necesidad de interacción por parte del usuario y además, posee funcionalidades secundarias que aportan la información necesaria para poder observar los datos obtenidos durante cada etapa de la metodología.

La herramienta se ha validado con aplicaciones reales de tipo HPC, y los resultados obtenidos se han comparado con otros resultados generados previamente de forma manual, con la intención de verificar que el proceso automatizado se realiza correctamente.

Summary

The recent expansion of areas that process big amounts of data (Artificial Intelligence, data analysis, banks, etc.) has increased the energy consumption over the last years, mainly at the Data Centers focused on data computing, not only that, but it also has increased the expenses associated to the maintenance of the Data Center. There are three different types of energy consumption on a Data Center: Cooling expenses, computing or IT (Information Technology) equipment and infrastructures, ordered decreasingly based on their contribution to the energetic consumption.

Despite the recent progress in terms of optimization on energy consumption, there is still a long way to go. To improve the energy efficiency at Data Centers techniques such as power budgeting, power capping or resource management are used. Often, these techniques require the energy consumption of the applications that will run in the Data Center. Traditional techniques retrieves the data of energy consumption by applying a dynamic profiling to the whole execution of the application. This technique is not feasible nor efficient on High Performance Computing (HPC) scenarios due to the large execution times that can even spread out for several days.

On a previous work, a methodology has been developed in order to get a fast energy consumption estimation of HPC applications with no demands on complete executions of the application and via an automated process [1]. The methodology describes the procedures to apply a dynamic profiling on a reduced version of the application, which is named on the cited work as application signature.

Therefore, in this project we have developed a tool capable of performing a fast energy consumption estimation by following the steps indicated on the methodology from the previous work. The framework is executed automatically, with no user interaction and in addition to that, it has another functionalities that provides the data required to overwatch the steps performed on every module of the framework.

The tool is validated with real HPC applications, and the results are compared with another ones previously executed manually, with the purpose of veryfing that the process has been implemented succesfully.

Lista de Acrónimos

CPD Centro de procesamiento de datos

HPC High Performance Computing

TCI Tecnología de la Información y la Comunicación

GHGE GreenHouse Gas Emissions

PUE Power Usage Efectiveness

DCIM Data Center Infrastructure Management

IPC Instrucciones Por Ciclo

FLOPS Floating Points Operations Per Second

QoS Quality of Service

RC Ratio de Compresión

Palabras clave

Palabras clave en Español

- Eficiencia energética
- Computación de Alto Rendimiento
- Centro de Datos
- Uso de Energía Eficiente
- *Profiling* Dinámico
- Consumo de energía
- Firma de la Aplicación

Keywords in English

- Energy efficiency
- High Performance Computing
- Datacenters
- Power Usage Efficiency
- Dynamic Profiling
- Energy Consumption
- Application Signature

Capítulo 1

Introducción

El alto consumo generado por los centros de datos ha provocado una serie de propuestas que permiten mejorar la eficiencia energética de los gastos asociados a la Computación de Alto Rendimiento (High Performance Computing o HPC) en los propios centros de datos. La mayoría de estas propuestas requieren una estimación previa del consumo energético, que generalmente, implica una ejecución completa de la aplicación que puede prolongarse incluso durante días, generando a su vez un alto consumo energético. Para resolver el problema, vamos a seguir en este proyecto una metodología desarrollada en un estudio previo que propone una estimación rápida de consumo de energía para aplicaciones HPC [1]. En ella, se detalla cómo obtener dicha estimación a través de la extracción de la firma de la aplicación, una versión reducida de la aplicación original, en términos de tiempo de ejecución, sobre la que se va a realizar un *profiling* dinámico para obtener una serie de métricas asociadas a los contadores hardware. Dichos contadores hardware serán utilizados posteriormente para aplicar unos modelos de potencia ya desarrollados en un trabajo anterior [2].

1.1. Motivación

La motivación principal de este proyecto es la de implementar un *framework* de alto nivel que permita al usuario sin apenas interacción, realizar una estimación rápida de energía sobre aplicaciones HPC siguiendo el estudio previo mencionado anteriormente, para posteriormente, ejecutar unas técnicas que mejoren la eficiencia energética.

1.1.1. Automatización del proceso

La necesidad de realizar el proceso de una manera eficiente y rápida constituye un punto importante en el análisis de aplicaciones HPC. Llevar a cabo el proceso de estimación de energía puede demorarse demasiado, por lo que se opta por la creación de un *framework* que automatice el proceso y que además pueda soportar una amplia gama de aplicaciones no relacionadas. A pesar de automatizar el proceso en su totalidad, se desea obtener los resultados de los módulos independientes, por lo que la creación de ficheros con los datos generados en los distintos módulos será un recurso utilizado en múltiples ocasiones para poder analizar al detalle los pasos seguidos a lo largo de la ejecución.

1.2. Objetivos

El propósito de la herramienta está dirigido a usuarios con un conocimiento técnico alto de aplicaciones HPC, por lo que no se requiere una interfaz gráfica que facilite la labor del usuario, sino automatizar un proceso de estimación de energía que pueda ser utilizado por políticas proactivas de eficiencia energética en centros de datos. Por tanto, sus objetivos son:

- Automatizar el proceso de estimación de energía.
- No requerir interacción del usuario.
- Modularizar los distintos componentes para poder retomar el proceso de ejecución desde cualquier punto, sin necesidad de reiniciar completamente.
- Alcanzar una gran escalabilidad.

1.3. Plan de Trabajo

El proyecto ha sido desarrollado siguiendo una arquitectura modularizada. Esto permite eliminar dependencias significativas entre los distintos módulos para que los usuarios puedan utilizar cualquiera de estos sin tener que ejecutar la herramienta al completo.

Asimismo, el desarrollo de los módulos también ha sido realizado de forma independiente al resto, siguiendo un orden secuencial que será interpretado por el controlador de la aplicación. Facilitando así el desarrollo en cortos periodos de iteraciones, normalmente semanales con el fin de ir implementando y testeando los diferentes módulos del *framework*. Enfrentando también diversos problemas que se puedan generar al tratar el código en distintos sistemas/equipos respecto al del desarrollo original.

Una vez se encuentran implementados los distintos módulos, se realiza una regresión para observar nuevas tareas a realizar, como son las mejoras de eficiencia, la eliminación de archivos con *scripting* como finalidad o la parametrización de componentes para poder facilitar la interacción por parte del usuario.

1.4. Estructura del documento

El resto del documento describe las distintas fases realizadas durante el proyecto. El punto de partida es una breve introducción al consumo energético en los centros de datos, así como a las aplicaciones HPC y su contexto histórico, además de su aplicabilidad, estudios previos y diferentes técnicas a utilizar en cuanto a la obtención de consumo de energía se refiere. Más adelante se explica con más detalle la metodología a seguir y la implementación de los módulos, además del motivo de los lenguajes y herramientas utilizadas durante el proceso de estimación rápida. Posteriormente, se mostrarán una serie de documentos/imágenes que permitirán observar los resultados obtenidos por los distintos módulos. Por último, se incluirán las traducciones para los apartados de conclusión y trabajos futuros.

Capítulo 2

Introduction

The great consumption generated by the data centers has caused a wide variety of proposals in order to enhance the energetic efficiency in terms of expenses related to the High Performance Computing (HPC) in the datacenters. The vast majority of this proposals requires a previous estimation of the energy consumption, which normally implies a complete execution of the application that may extents even for several day, generating at the same time a greater consumption of energy. The purpose to fix this issue is to follow a methodology previously developed on another study that aims to perform a fast energy estimation for HPC applications. The methodology relates the tasks to obtain the mentioned fast energy estimation based on the extraction of the application signature, a reduced version of the original application on which a dynamic profiling shall be executed towards the obtention of metrics associated to the hardware counters that will be used on power models already developed [1].

2.1. Motivation

The motivation of the present project is to implement a high level language framework that allows the user to perform a fast energy estimation of HPC applications without any interaction, following the methodology previously mentioned so that energetic efficiency techniques can be applied later based on the results obtained.

2.1.1. Process automation

The need to carry out the whole process on a fast and efficient way becomes one of the strong points of HPC applications analysis. The lifecycle of the different techniques may take large periods of time that can be accelerated with the development of the framework, easing as well its execution with a lack of user interaction and supporting a wide range of high programming non-related languages. Even with the automation of the whole process, it is mandatory to provide the results of the different modules, so the creation of dump files would be used at multiple points in order to analyze the different steps among the execution.

2.2. Objectives

The tool's target is oriented towards users with a high technical knowledge on HPC applications, meaning that for example a graphic interface is less suitable and not really needed as users must be capable of using the command line to run the framework. With that in mind, the goal can be focus entirely focus on automate the energy estimation procedure to narrow a task that may take days or even weeks to fulfill manually. Therefore, the goals are:

- Automatize the fast energy estimation process.
- Do not require user interaction.
- Achieve a modularization that allows the user to start at any point of process.
- Be easily scalable.

2.3. Roadmap

The project has been accomplished following a modularized arquitecture. Removing great dependencies between the different modules so that users can run the application as they may need, without the restriction of starting each time they want to retrieve different aspects or results of the framework.

Likewise, the development of the modules has also been accomplished independently from each other, following the sequential order that the controller will perform at runtime. Similarly, the development of modules could also be splitted on short iterations as the modules were normally implemented/tested on weekly periods. Applying this methodology, we could face any possible arising issue as fast as possible without fearing any possible code regression or deadlock of the project.

Once the different modules are implemented, a look back is thrown to check the possible errors/enhancements to reach in order to achieve a better performance in terms of efficiency, dependencies or configuration of the tool to extend the scope of the application.

2.4. Structure of the document

The rest of the document describes the different phases of the project. The starting point is a brief introduction of energy consumption in data centers, as well as an introduction to HPC applications and its historical background, as well as its applicability, previous studies and the variety of techniques to implement in terms of energy consumption estimation. Afterwards, the methodology applied and the implementation of the frameworks is shown more in detail together with the reason of the selected languages applied over the fast estimation process. Later on, some files will be displayed to show the results obtained at different points of the application and a small comparison would be made to see the big differences between the execution of the process manually or automated. Finally, translation of conclusion and future works will be provided.

Capítulo 3

Introducción Teórica

El uso de aplicaciones de Computación de Alto Rendimiento tiene como objetivo la ejecución de un software con unas extensas capacidades de cómputo que prohíben su realización en ambientes de cómputo tradicional o incluso en servidores de grandes características[1]. Ejemplos de estos tipos de computación podrían ser procesados de imágenes, búsquedas exhaustivas o combinatorias.

Durante años, el objetivo de las aplicaciones HPC ha sido mejorar su capacidad de computación y su velocidad de procesamiento, la cual se mide en FLOPS (Floating Points Operations per Second). Una CPU de un usuario estándar trabaja con 10FLOPS, generando de 0.5 a 120GFLOPS de poder de cómputo. Estas cantidades no son suficientes para realizar tareas como las mencionadas previamente en un corto periodo de tiempo. Necesitaríamos unos 6PFLOPS, es decir, 6 millones de GFLOPS, lo que equivaldría a 6 millones de CPUs trabajando de forma paralela en la misma tarea. Esto no implica solamente un coste material, sino también energético, puesto que las CPUs deben mantenerse en buen estado para mantener su nivel de eficiencia. Para ello, es necesario disponer de un sistema de refrigeración, lo que conlleva a un coste elevado, similar al consumido por una ciudad [3].

Cada año se actualiza una lista, top500[2], con las supercomputadoras con mayor rendimiento de la actualidad, basándose únicamente en los FLOPS, sin tener en cuenta el consumo de energía. Esto da lugar a una lista alternativa, Green500[3], dónde si se considera la energía requerida por la supercomputadora, y la cual se expresa en MFLOPS por Vatio.

Si cogemos una muestra de ejemplo en las dos listas y realizamos una comparación con un periodo de tiempo mayor a 5 años [4], podemos observar como la muestra baja en el ranking de Top500 más rápido, con lo que podemos concluir que los nuevos avances están más enfocados al rendimiento y no a la eficiencia energética, ya que la muestra seleccionada cae más paulatinamente en el ranking de Green500.

Más aún, si nos fijamos en el sector de la Tecnología de la Información y la Comunicación, vemos que se genera el 2 % de las emisiones globales de CO2 [5], y se estima que los centros de datos tendrán la segunda mayor progresión en cuanto huellas de gases de efecto invernadero

[1] https://blogs.msdn.microsoft.com/warnov/2010/09/30/high-performance-computing-hpc-y-windows-azure/
[2] https://www.top500.org/
[3] https://www.top500.org/green500/

(GHGE, por sus siglas en inglés) de todo el sector TIC. Por eso, se considera muy importante mejorar la eficiencia energética de los centros de datos, sobretodo de la computación, ya que es el mayor contribuyente de gasto energético [6].

3.1. Gasto energético en Centros de Datos

Cuando hablamos de gasto energético en centros de datos, hay que tener claro que no solo nos referimos a la computación, o aspectos similares a TI (Servidores, redes, etc), sino también a la refrigeración y a las infraestructuras asociadas, como la iluminación. Durante años, la forma de evaluar el rendimiento de estos sistemas ha sido el PUE[4], *Power Usage Efectiveness*, para determinar la eficiencia energética de un centro de datos. La forma de calcular el PUE es comparando el total de la energía consumida con la que llega a los equipamientos de TI. Con lo cual se obtiene el gasto energético provocado por otros sistemas como el de refrigeración. Pero también existe una desventaja a la hora de aplicar esta fórmula, ya que no existe una normalización que regule la metodología del cálculo del PUE, por los que los centros de datos podrían utilizar distintos parámetros de referencia.[5]

3.2. Técnicas de reducción de Energía

Se calcula que alrededor del 50 % del consumo de energía deriva de sistemas de refrigeración, mientras que el 35-40 % es generado por sistemas informáticos y lo demás por el resto de sistemas.[6] Por tanto es muy importante aplicar algunas de las técnicas más recomendadas para reducir el gasto energético asociado a sistemas de refrigeración o de infraestructuras, aunque siempre se debe realizar un estudio previo del centro de datos, ya que por ejemplo según el estado de las instalaciones o el orden en que se ejecutan las técnicas, se puede llegar a alcanzar hasta un 52 % de ahorro. Algunas de las técnicas pueden ser las siguientes:

- Elevar la temperatura de funcionamiento centro de datos.
- Virtualización de servidores.
- Sustituir iluminación por iluminación LED.
- Optimizar el flujo del aire.
- Actualizar el software de los equipos de refrigeración.
- Introducción al sistema DCIM.

Con el Sistema de Gestión de Infraestructuras de Centros de Datos(DCIM) permite obtener un control y una vigilancia más eficaz e integral[7]. El DCIM permite obtener una vista de pájaro

[4] https://es.wikipedia.org/wiki/Power_usage_effectiveness
[5] https://www.sinceo2.com/eficiencia-energetica-centros-de-proceso-de-datos/
[6] https://www.sinceo2.com/eficiencia-energetica-centros-de-proceso-de-datos/
[7] https://www.deltapowersolutions.com/es-co/mcis/white-paper-overview-of-green-energy-strategies-and-techniques-for-modern-data-centers.php

que permite aumentar la eficiencia ayudando al administrador del centro de datos a identificar servidores comatosos para una re-asignación. En definitiva, hay una gran variedad de técnicas que permiten una mayor eficiencia energética asociada a sistemas de refrigeración e infraestructuras, así como existen técnicas para ahorrar en consumo derivado de sistemas TI, las cuáles en su gran mayoría requieren una estimación previa de la energía consumida.

Esto da lugar a un coste mayor, puesto que se requiere realizar un *profiling* sobre cada una de las aplicaciones, lo cual se puede prolongar incluso hasta días. Por tanto, en los siguientes capítulos vamos a ver los pasos seguidos en la metodología asociada a este TFG para eliminar ese requisito y mejorar la eficiencia energética en los centros de datos.

3.3. Contadores Hardware

Una de las tareas que se van a realizar en este trabajo es la obtención de las métricas de los contadores hardware. Entendemos por contador hardware aquellos registros de la CPU que sirven como memoria interna temporal con el fin de almacenar datos sobre las instrucciones que se ejecutan. La información obtenida de esta tarea es muy importante, puesto que genera datos sobre el procesador en tiempo de ejecución, ya que es el componente que aporta un mayor consumo energético. Asimismo, éstas métricas permiten obtener datos importantes como las Instrucciones por Ciclo (IPC) de la aplicación o los fallos de caché de último nivel (LLC). De igual manera, permite identificar posibles cuellos de botella o incluso realizar mejoras significativas en tiempo de ejecución sobre sistemas multicore [7]. Por tanto, los modelos de potencia que explicaremos en capítulos posteriores, estarán en función de las métricas de los contadores hardware [2].

Capítulo 4

Estado del arte

En este capítulo vamos a hacer un repaso de las distintas técnicas relacionadas con la eficiencia energética en un centro de datos respecto al consumo generado por sistemas TI. Concretamente aquellas técnicas asociadas a aplicaciones HPC que tienen como requisito una estimación previa del consumo energético de las aplicaciones. Además, vamos a estudiar y comparar otros trabajos con las propuestas realizadas en la metodología asociada a este trabajo.

Las principales técnicas de eficiencia energética que vamos a ver asociadas a una estimación previa del consumo energético de aplicaciones HPC son las siguientes:

- *Power budgeting*
- *Power capping*
- *Resource Management*

La técnica de *power budgeting* (Presupuesto de potencia) permite establecer las reglas para distribuir el balance de potencia entre todos los servidores y los sistemas de refrigeramiento del centro de datos [8] con el fin de maximizar el rendimiento total, o lo que es lo mismo, reducir el tiempo medio de ejecución [9].

La técnica de *power capping* permite asegurar que el consumo de energía máximo de cada servidor permanece por debajo del límite de capacidad, es decir, limita cuanta energía puede consumir un servidor. Normalmente un servidor mantiene un consumo constante de energía, y rara vez se llega al máximo rendimiento del propio servidor, por lo que de forma general, se establecen unos límites por debajo del consumo máximo[1]. Además, esta técnica permite que la capacidad de potencia se destine con preferencia a aquellos servidores con una carga de trabajo mayor, mediante la reducción del *power budgeting* de aquellos servidores con una carga de trabajo menor [10].

Por último, *resource management* permite realizar una gestión más eficiente de los recursos disponibles en un centro de datos, como pueden ser las comunicaciones con los servidores ante una petición de los clientes, o la propia distribución de tareas entre los distintos nodos para proporcionar la respuesta correspondiente [11].

Para todas las técnicas mencionadas, se asume que existe un *profiling* dinámico sobre la energía consumida por las aplicaciones, o bien un modelo de energía que estima el consumo de los

[1] https://www.infoworld.com/article/2631095/power-capping-yields-savings-and-floor-space.html

servidores en un centro de datos. Es ahí dónde reside el punto principal de la metodología de estimación rápida de energía para aplicaciones de tipo HPC [1], en la propuesta de una serie de técnicas que permitan estimar rápidamente el consumo de energía de las aplicaciones sin la necesidad de ejecutar la aplicación completa, y que cuyo proceso se automatiza gracias al *framework* desarrollado en este TFG.

A continuación vamos a analizar algunos trabajos similares enfocados en la estimación de energía de aplicaciones. Para ello, es importante saber cómo se consume la energía en los centros de datos y qué componentes contribuyen a un mayor consumo energético. Esto se realiza a través de unos modelos de potencia que deben ser validados para comprobar su exactitud [12].

Algunos trabajos analizan el consumo de potencia a nivel de sistema para proponer unos modelos de potencia basados en distintos componentes del servidor [13]. Otros trabajos enfocan el consumo de energía en el mayor contribuyente del consumo energético, el procesador. Para ello se analizan los contadores hardware, los cuáles ofrecen unas estadísticas sobre el rendimiento del procesador, con las cuáles se pueden realizar técnicas estáticas [14].

Otro método empleado para hacer un uso eficiente de la técnica *power capping* es la planificación de la carga de trabajo de una manera proactiva para evitar que se alcancen los límites de consumo por parte de los servidores. Para ello, se emplea una programación restrictiva en adición a técnicas de *machine learning* para implementar un planificador que cumpla con las restricciones de *power capping* a la misma vez que se reduce la disrupción de la calidad del servicio (QoS) mediante una planificación inteligente en el orden de ejecución de las tareas [15].

El siguiente modelo plantea la idea de utilizar los datos extraídos de un *framework* de monitorización para construir un modelo de consumo de energía para cada usuario y usarlo para predecir el consumo de energía de trabajos futuros. Algunas de las técnicas usadas en dicho trabajo son la optimización del planificador de tareas, facturación de energía o un modelado de potencia a escala del sistema [22].

También existen trabajos que proponen un modelo de estimación de energía basado en el número de nodos utilizados por las distintas aplicaciones HPC, tomando como entrada del modelo los datos históricos sobre el consumo de energía o el perfil de potencia de las aplicaciones, alcanzando una mayor precisión a medida que se obtienen más datos tras cada ejecución [16].

Para obtener la estimación de energía, utilizamos unos modelos de potencia de CPU y memoria en función de los contadores hardware. Este método ha sido utilizado también en otros trabajos de forma independiente para estimar los modelos de potencia de un servidor en tiempo real [17]. Estos contadores nos proporcionan métricas que permiten analizar el consumo generado por la CPU, el cuál supone una gran parte del consumo de los servidores. La forma de obtener estas métricas va a estar basada en la ejecución de la firma de la aplicación durante un periodo de tiempo determinado.

Alternativamente, también hay propuestas que consideran la estimación de consumo de energía con técnicas similares a las presentadas en este proyecto. En ellos se plantea la posibilidad de generar una firma paralela de la aplicación, es decir, una versión reducida de la aplicación generada en base a la actividad del flujo de mensajes de la propia aplicación original, de los cuales se identifican y extraen las fases más representativas para generar dicha firma [18]. En dicho traba-

jo la firma se extrae a partir de una ejecución completa de la aplicación, a diferencia del proceso presentado en este TFG donde la firma se extrae sin tener que ejecutar la aplicación en su totalidad.

Por último, también podemos encontrar modelos que optan por la ejecución parcial de la aplicación original para estimar el rendimiento. En ellos se establece el término "esqueleto de la aplicación", entendiéndose como una versión reducida que al ejecutarse refleja el rendimiento de la aplicación original. Dicho modelo se basa en la captura de computación y comunicación de la aplicación ejecutada para determinar el esqueleto a generar [19]. De igual manera, hay proyectos que proponen la observación de ejecuciones parciales para estimar el consumo de energía, puesto que la mayoría de aplicaciones son iterativas y predecibles tras un corto periodo de arranque [20].

En resumen, existe una amplia variedad de técnicas que permiten realizar una estimación de energía de aplicaciones HPC que en su gran mayoría requiere una ejecución completa de la aplicación analizar o incluso datos históricos tras varias ejecuciones. Esto supone un gasto energético adicional, además de un coste en términos de tiempo muy significativo ya que hablamos de aplicaciones HPC. Con motivo de eliminar dichos costes, se plantea una metodología capaz de extraer la firma de las aplicaciones para un posterior *profiling* dinámico, con el cuál poder obtener las métricas de los contadores hardware en tiempo de ejecución. Dichas métricas aportarían la información suficiente para utilizar un modelo de potencia desarrollado previamente y que nos aportaría una estimación sobre el consumo de energía de las aplicaciones HPC. Todo ello sin la necesidad de ejecutar en su totalidad la aplicación en cuestión.

Capítulo 5

Metodología

En el presente trabajo se ha desarrollado una herramienta de código abierto para el *framework* de estimación rápida de energía con el fin de automatizar el proceso. A continuación se explicará el *framework* de estimación rápida de energía.

Las técnicas propuestas en la metodología asociada en este TFG son la extracción de la firma de las aplicaciones HPC para un posterior *profiling* dinámico de dicha firma. Se entiende por firma de la aplicación una versión reducida de la original en términos de ejecución. Esto facilita la obtención de datos relacionados con los contadores hardware en tiempo de ejecución, es decir, datos sobre el comportamiento del procesador. Estos datos serán tratados y utilizados para aplicar unos modelos de potencia y memoria que permiten estimar y contrastar el consumo de energía generado por las aplicaciones HPC. Este proceso descrito es el que deseamos automatizar en el *framework* desarrollado.

Por otro lado, se define el Ratio de Compresión (RC) como el ratio entre el tiempo total de ejecución de la aplicación original y el tiempo requerido por el *framework* desarrollado para estimar el consumo de energía. Por tanto, el RC nos indicará la aceleración del proceso de estimación de energía.

En este proyecto se propone automatizar el proceso de estimación rápida de energía a partir de una firma de la aplicación sobre la cuál se realiza un *profiling* dinámico para obtener las métricas de los contadores hardware, los cuales serán utilizados en los modelos de potencia previamente mencionados. La ejecución de la firma se compone mediante la ejecución de los caminos independientes que pueden ser ejecutados en la aplicación original. Entendemos por camino las distintas funciones/subrutinas que pueden ser llamadas en tiempo real. Esto hace que el tiempo de ejecución acumulado por los distintos caminos sea muy inferior al de la aplicación completa. En base a dicha firma, obtenemos las métricas de los contadores hardware para cada camino independiente.

A continuación, se debe realizar una reconstrucción con los datos obtenidos de las señales de los contadores hardware, ya que la firma de la aplicación se ejecuta únicamente durante un corto periodo de tiempo para cada camino. Esto quiere decir, que la media de las métricas de los contadores hardware obtenidas durante ese corto periodo de tiempo, deben ser prolongadas hasta el periodo de tiempo estimado en el que se ejecuta un camino, por ejemplo: Ejecutamos la firma de la aplicación para un camino independiente durante 60s, y previamente se ha estimado que el camino completo dura 600s, entonces los datos obtenidos en esos 60s deben ser extendidos hasta

llegar a 600s.

Una vez realizada la reconstrucción utilizamos modelos de potencia de CPU y memoria que están en función de los contadores hardware y que utilizan como entrada al modelo el perfil reconstruido de la aplicación para generar unos perfiles de potencia, a partir de los cuáles se estima el consumo de energía.

5.1. Framework de estimación rápida de energía

En esta sección se va a entrar un poco más en detalle de cómo desarrollar la metodología planteada anteriormente. Para ello, vamos a explicar los puntos de entrada de la aplicación y los módulos que facilitarán las técnicas comentadas, para en capítulos posteriores detallar la implementación de los mismos paso a paso.

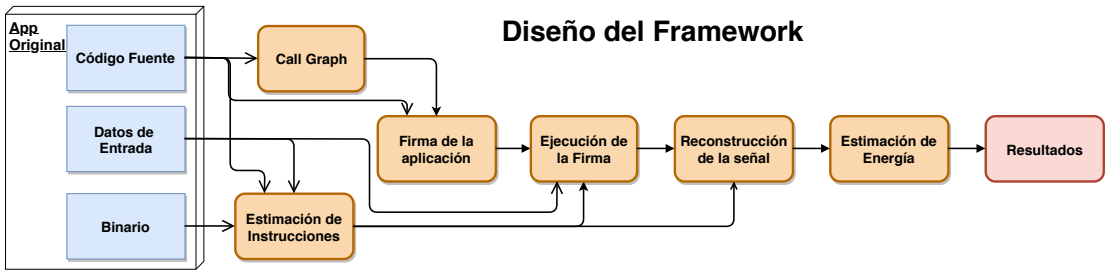

Figura 5.1: Diseño del framework desarrollado

En la figura 5.1 podemos observar el diseño del *framework* desarrollado. En ella se muestran tantos los datos utilizados de la aplicación, como los distintos módulos implementados junto a sus entradas correspondientes. De la misma manera, podemos apreciar el orden secuencial en que son llamados los módulos y las dependencias que hay entre ellos.

Para comenzar con la ejecución del *framework*, vamos a tomar como puntos de entrada el código fuente original de la aplicación, así como el archivo binario y los datos de entrada. A partir de ahí, los módulos serán llamados de manera secuencial en el siguiente orden:

- Módulo *Call Graph*.
- Módulo de estimación de instrucciones.
- Firma de la aplicación.
- Módulo de ejecución de la firma.
- Reconstrucción de la señal.
- Módulo de estimación de energía.

1. En primer lugar se ejecuta el *Call Graph*, el cual tomando el código fuente original como entrada, genera una serie de caminos independientes que representan las funciones/subrutinas llamadas por la aplicación. Por tanto, definimos camino independiente como la secuencia de funciones desde el nodo raíz (función main), hasta una de las funciones extremo. Obtenemos así un grafo completo con las funciones invocadas. En la figura 5.2 se observa un ejemplo de los caminos independientes, dónde partiendo de la función $main$, se añaden las distintas funciones Fn a la secuencia de llamadas según son invocadas por la aplicación.

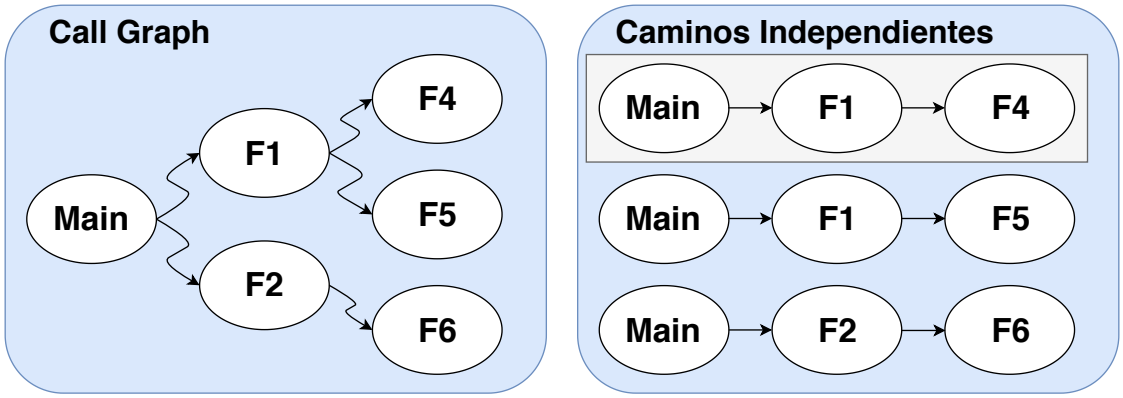

Figura 5.2: Ejemplos de caminos generados

2. A continuación, es invocado el módulo de estimación de instrucciones. La implementación de este módulo fue realizada previa al comienzo de este trabajo, por lo que se va a reutilizar en el *framework* desarrollado. Su objetivo es estimar el número de instrucciones de cada uno de los caminos generados sin tener que ejecutar la aplicación completamente. Esta tarea se realiza a través de un *profiling* estático del código fuente, tomando como puntos de entrada el resultado del *Call Graph* y toda la información de la aplicación original. Del código fuente y datos de entrada de la aplicación original se extraen los límites superiores de los bucles y del binario se extraen los instrucciones CPU de cada camino independiente.

3. Posteriormente, se procede con la generación de la firma de la aplicación. La firma de la aplicación está formada por los binarios de cada camino independiente. La creación de la firma se realiza en base a los caminos obtenidos durante el módulo *Call Graph* y el código fuente original, el cual se adapta para cada camino y se compila automáticamente para generar un archivo binario por camino.

4. Una vez generada la firma de la aplicación, vamos a realizar un *profiling* dinámico mediante la ejecución de los archivos binarios generados en el módulo anterior durante un corto periodo de tiempo para obtener las métricas de los contadores hardware. Por tanto, el resultado obtenido en este módulo serán los perfiles de los contadores hardware de cada camino independiente.

5. Para la reconstrucción de la señal vamos a tomar como puntos de entrada los perfiles de

los contadores hardware, además de la estimación de instrucciones por camino generado previamente para construir el perfil de la ejecución completa de la aplicación. Este perfil reconstruido será la salida del módulo, y es equivalente al perfil de la aplicación original en cuanto a términos de energía se refiere.

6. Por último, en el módulo de estimación de energía vamos a construir los perfiles de potencia de CPU y de memoria de la aplicación tomando como referencia el perfil de la señal reconstruida. Como resultado, obtendremos finalmente la estimación de energía de CPU y memoria consumida por la aplicación.

Capítulo 6

Implementación

En este capítulo se explicará la herramienta desarrollada para cumplir con los objetivos del proyecto. Es decir, demostrar que es posible construir un *framework* capaz de automatizar el proceso por el cual deseamos estimar rápidamente el consumo de energía de una aplicación HPC, siguiendo la metodología descrita en el capítulo anterior, esto es, mediante la extracción de la firma de las aplicaciones y un *profiling* dinámico que nos permitan obtener los datos necesarios para aplicar unos modelos de potencia con los que estimar el consumo de energía de la aplicación HPC sin la necesidad de ejecutarla al completo.

Para ello, se irán explicando de forma detallada, en cuanto a funcionalidades y lenguaje de programación se refiere, los módulos explicados con anterioridad referentes a la figura 5.1, dejando los resultados obtenidos para un capítulo posterior.

6.1. Características

El *framework* desarrollado tiene como objetivo el funcionamiento bajo un servidor Linux, con distribución CentOS[1]. En el caso de la herramienta desarrollada, las pruebas se han hecho en un servidor de la Universidad Complutense de Madrid (UCM). Esto es importante, ya que como se ha comentado en el capítulo anterior, vamos a obtener datos asociados a los contadores hardware para posteriormente generar los perfiles de potencia. Por tanto, estos contadores hardware van a ser específicos de dicho servidor.

En cuanto al desarrollo del *framework*, se ha optado por utilizar el lenguaje *Python* ya que es totalmente compatible con las distribuciones de Linux, además de las facilidades para implementar scripts, ofrece una gran cantidad de librerías que facilita la interacción con el sistema operativo y que nos permiten realizar tareas en paralelo. Por otra parte, vamos a hacer uso de la librería pandas[2] de *Python*, la cuál es muy eficiente para el análisis de datos, así como para la lectura/exportación de datos alojados en ficheros ".csv". Pandas también nos permite tratar los datos como si estuviéramos trabajando con tablas y realizando operaciones vectoriales sobre ellas.

El único requisito para poder lanzar la aplicación es tener instalado el intérprete de *Python* con

[1] https://www.centos.org/
[2] https://pandas.pydata.org/

la versión 3.6[3]. Por otro lado, necesitaremos ubicar los ficheros asociados a la aplicación HPC a analizar bajo la carpeta "source_folder". Estos ficheros deben incluir el código original, los datos de entrada de la aplicación y un archivo binario ya compilado. A la hora de lanzar la aplicación se deberá indicar la ruta donde se ubica el código fuente original.

6.2. Módulo Call Graph

De igual manera que en la sección 5.1, vamos a detallar los módulos en orden secuencial según el orden en el que son ejecutados. Como se ha mencionado previamente, es necesario tener ubicado bajo un mismo directorio el código fuente de la aplicación, puesto que se comienza por la generación de los caminos independientes.

La tarea es llevada a cabo por una herramienta externa de código abierto llamada *Doxygen*[10]. La principal función de esta herramienta es la de generar documentación sobre las funciones implementadas en un *framework*. En este trabajo nosotros vamos a adaptar su funcionalidad modificando el archivo "doxyfile". En él especificaremos las técnicas que *Doxygen* debe utilizar para realizar un análisis del flujo de las funciones desarrolladas. Es decir, los distintos caminos que la aplicación puede tomar y el flujo de llamadas entre las distintas funciones. Esta información es generada en un archivo nombrado como "cgraph.dot", el cuál tendrá un formato que también podemos definir en el archivo de configuración "Doxyfile". En la salida 6.1 podemos observar un extracto de dicho formato. En él vemos que se asigna un nodo concreto a cada función de la aplicación, así cómo la pila de llamadas que hay entre ellas. Por ejemplo: Se puede apreciar que el Nodo 1 pertenece a la función "BT" (Línea 4), mientras que el Nodo 2 corresponde a la función "adi" (Línea 6), y que ambos están conectados (Línea 5). Aunque el resultado obtenido tras ejecutar *Doxygen* nos ayuda a obtener el *Call Graph* de la aplicación, no nos proporciona una estructura de datos con la que trabajar directamente ni un flujo correcto en los caminos, ya que únicamente tenemos los nodos conectados entre sí, pero no los caminos independientes, así que debemos eliminar aquellos que se encuentran incluidos en otros caminos más extensos, proceso que se describe a continuación.

Salida 6.1: Extracto del resultado de la herramienta doxygen

```
1 diagraph G {
2 edge {[}fontname="Helvetica",fontsize="10",...{]}
3 node {[}fontname="Helvetica",fontsize="10",...{]}
4 Node1 {[}label="BT",height=0.2,width=0.4,color="black",...{]}
5 Node1 -\textgreater Node2 {[}color="midnightblue",fontsize="10",...{]}
6 Node2 {[}label="adi",height=0.2,width=0.4,color="black",...{]}
7 Node2 -\textgreater Node3 {[}color="midnightblue",fontsize="10",...{]}
8 Node3 {[}label="add",height=0.2,width=0.4,color="black"...{]}
9 }
```

Una vez obtenemos el resultado de la herramienta *doxygen*, pasamos a realizar un análisis de grafo sobre el Call Graph de la aplicación original. Los pasos son los siguientes:

[3] https://www.python.org/downloads/release/python-360/

[10] http://www.doxygen.nl/

- Se cargan en memoria los datos generados por *doxygen*. Es decir, la conexión uno-a-uno de nodos entre sí. Como hemos visto previamente, estas conexiones indican, primeramente, la función que llama a la siguiente subrutina, y cual es esa subrutina (Línea 5 de la salida 6.1).

- A partir de ahí, generamos un mapa clave-valor, dónde las claves son las funciones que llaman a las siguientes subrutinas, y los valores son los propios nombres de las subrutinas. Siguiendo el ejemplo anterior, la clave sería el Nodo 1, y los valores el Nodo 2, y algún Nodo más si apareciera en el documento conectado al Nodo 1.

- A continuación, se vuelven a analizar las conexiones entre nodos para crear los caminos independientes. Esto se hace con ayuda del mapa generado previamente, pues los valores del mapa (subrutinas invocadas) deben ser añadidos recursivamente a la conexión entre nodos hasta que no se realicen más llamadas y se vuelva a un nodo ya iterado. Por ejemplo: En la conexión Nodo 1-Nodo 2, deberíamos agregar el Nodo 3, pues es llamado por el Nodo 2 (Línea 7 de la salida 6.1), y así sucesivamente hasta que no haya más nodos conectados.

- Por último, Se comparan los caminos obtenidos y se eliminan aquellos que se encuentran incluidos dentro de otros.

En la figura 6.1 se observa un ejemplo gráfico de lo que sería el resultado de aplicar el proceso previamente conectado. Esto es, la obtención de los caminos independientes que ya tenemos cargados en memoria. Como hemos explicado en el último punto del proceso, analizamos todos los caminos en búsqueda de aquellos que son redundantes para eliminarlos. Por ejemplo: No nos interesaría realizar un análisis posterior del camino [’main’, ’c2’], si no uno más extenso, como sería [’main’, ’c2’, ’c4’, ’c7’], puesto que ya incluiría al anterior.

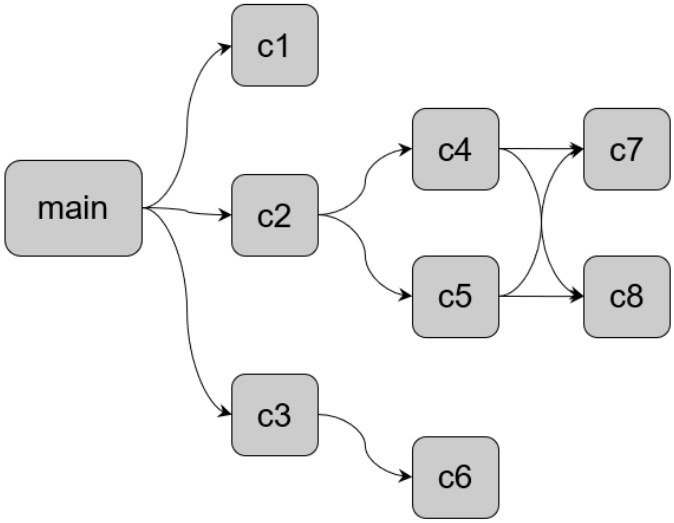

Figura 6.1: Estructura de los caminos

6.3. Módulo de Estimación de Instrucciones

En esta sección se explica como realizar la estimación de instrucciones CPU totales de la aplicación en tiempo de ejecución, requisito necesario para conseguir estimar más adelante el consumo de energía. Este módulo ya se encontraba implementado a la hora de comenzar con el diseño del *framework*. Aún así, se explican sus tareas para una mayor comprensión del mismo.

Este módulo estima el número de instrucciones CPU para cada camino independiente mediante un *profiling* estático. Para ello necesitamos el código fuente original, así como el archivo binario y el resultado del *call graph*. Los pasos a seguir son calcular los límites superiores de los bucles existentes en el código fuente y la extracción de las instrucciones CPU de cada camino partiendo del archivo binario. Esta tarea se realiza a través de la herramienta externa MAQAO[11], la cuál nos permite desensamblar el archivo binario original además de detectar las regiones con bucles en el binario desensamblado. Por tanto, obtenemos una estimación del número de instrucciones por camino independiente, con los cuáles podemos predecir el número de instrucciones CPU totales de la aplicación mediante la suma de las mismas, tal y como se puede observar en la ecuación 6.1. En dicha ecuación se muestra como el número estimado de instrucciones de la aplicación original I_{app}, es igual a la suma del número de instrucciones CPU de cada camino (I_{EP1}, I_{EP2}, ..., I_{EPn}). El resultado de este módulo será por tanto el número estimado de instrucciones CPU por cada camino independiente (I_{EPn}), y será guardado en memoria para su posterior utilización. Además, se generarán unos archivos, tanto ".csv" como ".xlsx" dónde se podrá comprobar el propio resultado.

$$\widehat{I}_{app} = I_{EP1} + I_{EP2} + ... + I_{EPn} \tag{6.1}$$

6.4. Firma de la Aplicación

En esta sección vamos a explicar los pasos seguidos para obtener la firma de la aplicación, que recordemos es una versión reducida de la aplicación original. Las entradas de este módulo van a ser el código fuente original y los caminos independientes obtenidos durante el módulo *Call Graph*. La salida será un binario por cada camino independiente, los cuales se generan después de la realización de un análisis de código.

Las tareas a realizar para extraer la firma de una aplicación son las siguientes:

- Replicar el código fuente original tantas veces como caminos se hayan generados durante el módulo de *Call Graph*.
- Eliminar las funciones no incluidas en cada camino independiente.
- Se genera un script en lenguaje "bash" para compilar el código modificado.
- Se ejecuta automáticamente el script implementado.

[11] http://www.maqao.org/

- Se obtiene un archivo binario por cada camino.

Para la eliminar las funciones no incluidas en cada camino, debemos hacer un parseo del código fuente. Esto quiere decir que debemos leer el código fuente original fichero a fichero en busca de aquellas funciones excluidas. Esas funciones excluidas son comentadas para que a la hora de generar los binarios no tengan influencia alguna.

Una vez generados los binarios de cada camino, no vamos a necesitar de nuevo el código fuente adaptado a cada uno de ellos, aún así, lo mantenemos intacto por si el usuario desea realizar una comprobación de las funciones eliminadas.

6.5. Módulo de ejecución de la firma

En este módulo se explicarán los pasos seguidos para realizar el *profiling* dinámico sobre la firma de la aplicación a analizar y en qué consiste esta técnica. Este módulo sólo va a tener cómo parámetros de entrada los archivos binarios generados durante la generación de la firma de la aplicación. El objetivo de este módulo es ejecutar los archivos binarios que forman la firma de la aplicación, parar su ejecución en un determinado momento y obtener la métrica de los contadores hardware en tiempo de ejecución de cada camino.

6.5.1. Tipos de ejecución

En el *framework* desarrollado existen dos formas de ejecutar este módulo, de manera secuencial y paralela. Es decir, ejecutando un binario a continuación de otro, o por el contrario, aprovechando el número de *cores* disponibles para ejecutar varios simultáneamente. Esto se debe a que en aplicaciones HPC se obtienen una gran cantidad de caminos diferentes, y por tanto, un tiempo elevado de espera si se ejecutara de manera secuencial, ya que el *profiling* requiere lanzar la aplicación durante un tiempo determinado para poder extraer la información en tiempo de ejecución. Como podremos observar más adelante, tan sólo necesitaremos especificar a la hora de lanzar la aplicación, el tipo de ejecución que queremos emplear.

Es importante tener en cuenta el número de procesadores si queremos ejecutar el módulo de forma paralela, ya que hoy en día disponemos de máquinas multicore, además de varios threads lógicos por cada *core*[4], lo cual nos permitiría lanzar un hilo en cada uno de ellos y estaríamos cometiendo un error. Dicho error se originaría al no aprovechar todos los recursos(se compartirían si lanzamos un hilo en cada *thread* lógico) mientras ejecutamos el *profiling* de los caminos, pudiendo alterar de esta forma los resultados obtenidos relacionados con las métricas de los contadores hardware. Por tanto, descartamos el número de *threads* lógicos y nos centramos en el número de *cores* físicos, obteniendo así el número máximo de hilos a lanzar simultáneamente.

Para implementar la ejecución paralela se ha usado la librería "multiprocessing"[5] de *Python*. Esta librería nos permite crear un *pool* de llamadas con las funciones a ejecutar, además de limitar el número de llamadas acorde al número de *cores* físicos de la CPU.

[4]https://www.howtogeek.com/194756/cpu-basics-multiple-cpus-cores-and-hyper-threading-explained/
[5]https://docs.python.org/3/library/multiprocessing.html

6.5.2. Implementación

Independientemente de si se realiza de manera secuencial o paralela, el *profiling* se lleva a cabo a través de una herramienta externa llamada ocount[11]. Esta herramienta es lanzada desde el *framework*, y nos permite lanzar la aplicación a analizar para extraer las métricas asociadas a los contadores hardware del servidor en el que reside el *framework*. Asimismo, podemos especificar el intervalo de tiempo en el que deseamos obtener dichas métricas, actualmente cada 100ms, las cuales serán almacenadas en un fichero temporal para su posterior procesamiento.

En contraposición, la herramienta no nos permite establecer un tiempo determinado de ejecución, por lo que debemos de mandar una señal al proceso cuando deseemos que este finalice. La implementación se puede observar en el algoritmo 1. Primero debemos definir los contadores hardware ($counters$) específicos del servidor en el que se ejecuta el *framework* (Línea 1), así como el resto de argumentos requeridos por la herramienta *ocount*, como son el nombre del binario a ejecutar (bn), el intervalo de muestreo y el nombre del fichero ($tempfile$) donde almacenar los datos (Líneas 2, 3 y 4).

A continuación ejecutamos *ocount* (Línea 6) y obtenemos el PID de la tarea en ejecución mediante una llamada al sistema (Línea 7). Después, esperamos o bien a que finalice la ejecución de la herramienta o a que se sobrepase el tiempo máximo de ejecución (Linea 10). Actualmente, dicho tiempo máximo equivale a 40s, puesto que hemos podido observar experimentalmente que es tiempo suficiente para extraer de las métricas de los contadores hardware.

Cuando se cumpla una de estas dos condiciones, procedemos a enviar un *kill* para terminar con el proceso (Línea 15). Mediante este proceso habremos obtenido las métricas de los contadores hardware ($C_{i,EPn}$) por cada camino independiente, y las habremos almacenado en un fichero de texto, listas para ser leídas en los módulos posteriores. Podemos ver un ejemplo en el cuadro 7.5, correspondiente al capítulo 7 de Resultados.

Algoritmo 1 Profiling Dinámico

```
1: counters = [...]
2: args = [...]
3: tempfile = temp
4: bn = binary_name
5: function PROFILING(path, bn)
6:     proc ← OCOUNT(args, counters, path + bn, tempfile)
7:     pid ← PID(proc)
8:     PRINT(pid)
9:     tiempo ← TIME()
10:    while pid AND tiempo <40 do
11:        Sleep 5
12:        update.pid⇒ PID(proc)
13:        PRINT(pid)
14:    end while
15:    KILL(pid)
16: end function
```

[11] http://man7.org/linux/man-pages/man1/ocount.1.html

6.6. Módulo de Reconstrucción de la Señal

En este módulo veremos como se tratan los datos obtenidos durante el *profiling* dinámico y su procesamiento para una posterior estimación del consumo de energía. Por tanto, las entradas del módulo serán las métricas de los contadores hardware generados durante la ejecución de cada camino y la estimación de instrucciones por cada camino independiente. El resultado será por tanto la señal reconstruida de los contadores hardware de toda la aplicación. Este resultado será exportado a un archivo ".csv" y ".xlsx" por si se desea realizar un análisis de los datos obtenidos, además se permanecer en memoria para que después sean reutilizados a la hora de aplicar los modelos de potencia que permitirán realizar la estimación de energía.

En la figura 6.2 se puede observar el proceso de reconstrucción de la señal completa de la aplicación que vamos a detallar a continuación.

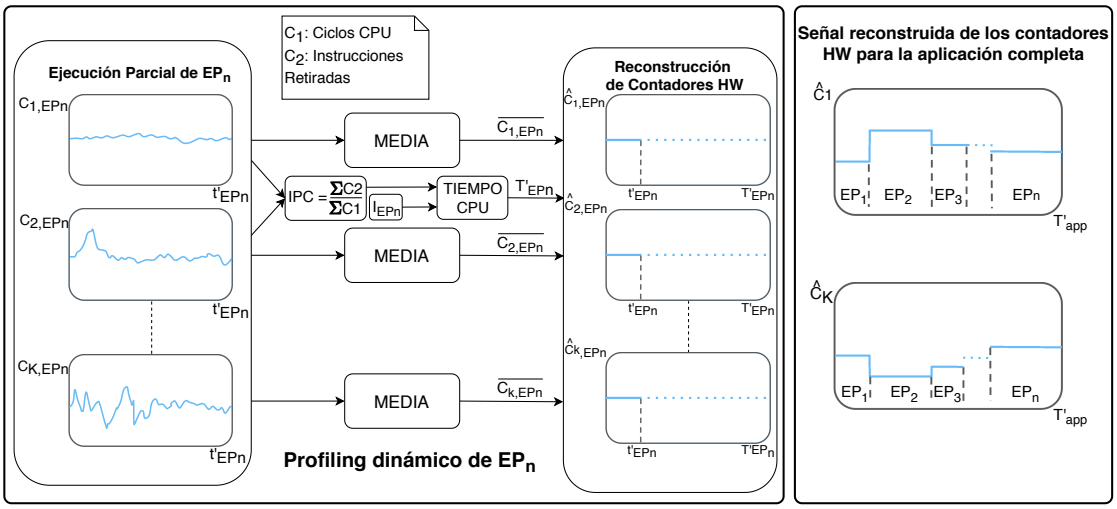

Figura 6.2: Proceso de reconstrucción de la señal

6.6.1. Lectura de Datos

En la figura 6.2 partimos en primer lugar de las métricas de los contadores hardware de cada camino independiente ($C_{i,EPn}$), que se requieren cargar en memoria y que se han obtenido al ejecutar la firma de la aplicación. A continuación se calcula la media ($\overline{C_{i,EPn}}$) de las métricas obtenidas por cada camino. Esta media de valores serán utilizadas posteriormente para realizar la reconstrucción de los contadores hardware ($\widehat{C}_{i,EPn}$) en combinación con la estimación del tiempo de ejecución (T'_{EPn}) de cada camino independiente. Por tanto, se leen los ficheros temporales generados por la herramienta *ocount* durante la ejecución de la firma. De esta forma obtenemos en memoria las métricas de los contadores durante la ejecución del programa.

6.6.2. Cálculo de Instrucciones por Ciclo

Para completar la reconstrucción de los contadores hardware, necesitamos también calcular las Instrucciones por Ciclo (IPC) de cada camino independiente. En la ecuación 6.2 se puede observar

el cálculo realizado para la obtención de las IPC por camino IPC_{EPn}. El valor de $C_{IR,EPn}[j]$ representa el contador hardware de instrucciones retiradas, mientras que $C_{Clk,EPn}[j]$ representa el número de ciclos de reloj. Tanto el número de ciclos como el número de instrucciones para cada camino están incluidos en las métricas obtenidas durante el *profiling* dinámico, por lo que podemos aplicar la ecuación directamente sobre los datos.

$$IPC_{EPn} = \frac{\sum\limits_{j \in A} C_{IR,EPn}[j]}{\sum\limits_{j \in A} C_{Clk,EPn}[j]} \tag{6.2}$$

6.6.3. Estimación del tiempo de ejecución

De igual manera, debemos de calcular el tiempo estimado de ejecución de cada camino independiente (T'_{EPn}). Para eso debemos aplicar la ecuación 6.3, donde IPC_{EPn} son las IPC del n-ésimo camino ejecutado, I_{EPn} corresponde a las instrucciones estimadas de ejecución del camino n-ésimo, y $Freq_{CPU}$ es la frecuencia a la cual trabaja la CPU mientras se ejecuta el camino independiente. El valor de $Freq_{CPU}$ es calculado como la media de los valores de los ciclos de reloj obtenido a lo largo de la ejecución del camino independiente en cuestión.

Es importante destacar que T'_{EPn} es el tiempo estimado de ejecución del camino independiente EPn, correspondiente a la ejecución original, pero obteniendo su valor mediante la ejecución de la firma, es decir, sin tener que ejecutar toda la aplicación. A su vez, si se suman todos los tiempos estimados de ejecución de todos los caminos se puede obtener el tiempo estimado total de ejecución (T'_{app}), como se observa en la ecuación 6.3.

$$\begin{aligned} T'_{EPn} &= \frac{I_{EPn}}{IPC_{EPn} \times Freq_{CPU}} \\ T'_{app} &= T'_{EP1} + T'_{EP2} + \cdots + T'_{EPn} \end{aligned} \tag{6.3}$$

Una vez se calculan estos datos para cada camino independiente, obtendremos la reconstrucción completa de los contadores hardware mostrada en la figura 6.2 para cada uno de ellos. Por lo tanto, la reconstrucción de la señal de los contadores hardware del camino independiente EPn se realiza mediante el siguiente proceso: 1) Se calcula el valor medio de los contadores hardware que se han obtenido durante la ejecución de la firma, 2) Los valores medios obtenidos se extienden en el tiempo hasta llegar al valor de T'_{EPn}. Por lo que el siguiente paso sería la reconstrucción de toda la aplicación.

6.6.4. Reconstrucción de la aplicación

Por último, la reconstrucción de la señal de los contadores hardware de toda la aplicación es generada mediante la concatenación de la reconstrucción de las métricas de los contadores hardware por cada camino independiente. Esto se realiza aplicando la ecuación 6.4, donde sm es el

tiempo de muestreo utilizado durante el *profiling* dinámico. Tras aplicar esta ecuación, obtendremos la reconstrucción completa de la aplicación mostrada en la figura 6.2.

$$\widehat{C}_i[n] = \left\{\widehat{C}_{i,EP1}[n_{EP1}], \widehat{C}_{i,EP2}[n_{EP2}], \ldots, \widehat{C}_{i,EPn}[n_{EPn}]\right\}$$
$$n \in [0, T'_{app}] \; and \begin{cases} n_{EP1} & \in [0, T'_{EP1}] \\ n_{EP2} & \in [T'_{EP1} + sm, T'_{EP2}] \\ \vdots & \\ n_{EPn} & \in [T'_{app} - T'_{EPn} + sm, T'_{EPn}] \end{cases} \tag{6.4}$$

Adicionalmente, se generan unos gráficos por cada contador a partir de los datos obtenidos tras la reconstrucción de la señal, de forma que podemos darle un formato más visual a los contenidos generados. Un ejemplo de estos gráficos lo podemos observar en la figura 7.1 del capítulo 7 de Resultados.

6.7. Módulo de Estimación de Energía

Finalmente, en este último módulo vamos a ver los pasos implementados para poder estimar el consumo de energía de la aplicación. Para ello, vamos a utilizar unos perfiles de potencia que se van a generar en función de la señal reconstruida.

En fases anteriores de este proyecto [2], se han generado modelos de potencia de memoria y de CPU en función de los contadores hardware. Por lo que aquí aplicaremos esos modelos de potencia para obtener posteriormente el consumo de energía de las aplicaciones HPC.

6.7.1. Perfil de Potencia

Con el fin de generar los perfiles de potencia de CPU y memoria asociadas a la aplicación analizada, vamos a emplear las formulas mostradas en la ecuación 6.5. Las entradas a ambos modelos de potencia (CPU y Memoria) son las señales reconstruidas de los contadores hardware ($\widehat{C}_n$) obtenidos en el módulo de Reconstrucción de la Señal. Los contadores utilizados se especifican en el cuadro 6.1. Además, podemos ver unos valores x_n e y_n que corresponden a los coeficientes de los modelos de potencia mostrados en el cuadro 6.2.

$$\widehat{P}_{CPU} = x_0 \cdot \frac{x_1 \cdot \widehat{C}_8 + 1}{x_2 \cdot \widehat{C}_4 + 1} + x_3 \cdot \frac{(x_4 \cdot \widehat{C}_6 + 1)(x_5 \cdot \widehat{C}_3 + 1)(x_6 \cdot \widehat{C}_2 + 1)^2}{x_7 \cdot \widehat{C}_7 + 1} - x_8$$
$$\widehat{P}_{Mem} = y_0 \cdot \frac{y_1 \cdot \widehat{C}_8 + 1}{y_2 \cdot \widehat{C}_5 + 1} + y_3 \cdot (y_4 \cdot \widehat{C}_8 + 1)(y_5 \cdot \widehat{C}_2 + 1)(y_6 \cdot \widehat{C}_3 + 1) - y_7 \tag{6.5}$$

	Descripción		Descripción
C1	Clock Cycles	C5	Branch Instructions retired
C2	Instructions Retired	C6	Resource stalls
C3	LLC misses	C7	μops dispatched
C4	L2D Cache Misses	C8	L1D Cache Misses

Cuadro 6.1: Contadores hardware extraídos durante el *profiling* dinámico

Valores de los Coeficientes									
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
x	53.10	1:00E-10	1:05E-10	9.83	4.03E-12	3.72E-10	2.49E-12	1.12E-12	58.19
y	24.80	1.00E-10	1:77E-11	5.76	1.00E-10	2.49E-12	3.72E-10	24.80	-

Cuadro 6.2: Valores de los coeficientes de los modelos de potencia

Posteriormente, tras la obtención de los perfiles de potencia se generarán unos gráficos para poder analizar los resultados visualmente. Estos gráficos se verán con más detalle en el próximo capítulo.

6.7.2. Diezmado

En este apartado, vamos a explicar el tipo de tratamiento que se da a los datos tras obtener la reconstrucción de la señal, hasta la obtención de los perfiles de potencia. Los modelos de potencia utilizados en este trabajo necesitan que el intervalo entre muestras sea igual a 10s, por lo que es necesario realizar una adaptación de la señal reconstruida de los contadores hardware. Se trata de la realización de un diezmado, que aplicado a los resultados obtenidos en la reconstrucción, sería la obtención de muestras con un intervalo de 10s, necesario para poder aplicar posteriormente los modelos de potencia. Recordemos que durante el *profiling* dinámico las muestras que habíamos obtenido tenían un intervalo de 100ms. Un intervalo de tiempo muy pequeño que prolongaría el análisis de los resultados obtenidos. Esto se lleva a cabo teniendo en cuenta el tiempo estimado de ejecución de cada camino independiente, y en caso de que alguno de los caminos tenga una estimación de tiempo de ejecución inferior a 10s, se combina con otro(s) hasta alcanzar dicha cifra.

6.7.3. Estimación de Energía

Llegados a este punto, después de la obtención de los perfiles de potencia, solo quedaría pendiente la estimación de energía. La cual se puede obtener sumando los valores obtenidos en los perfiles de potencia (CPU y memoria) multiplicados por el intervalo de muestreo (sm), tal y como se muestra en la ecuación 6.6, donde x es el valor de CPU o memoria. El resultado es calculado durante un periodo de tiempo equivalente al tiempo estimado de ejecución de la aplicación T'_{app}, y para el conjunto B formado por los valores obtenidos de los perfiles de potencia $\widehat{P}_x$.

$$\widehat{E}_x = \Big[\sum_{n \in B} \widehat{P}_x[n] \Big] \times sm \tag{6.6}$$

6.8. Funcionalidades extras

Una vez concluida la explicación de las funcionalidades implementadas en el *framework* y de cómo hemos conseguido obtener una estimación del consumo de energía de una aplicación HPC, vamos a repasar algunas funcionalidades desarrolladas que facilitan al usuario tanto la interacción con el *framework* como la obtención de los resultados.

6.8.1. Interfaz

En primer lugar, cabe destacar que el *framework* no dispone de una interfaz gráfica, puesto que uno de los objetivos del presente trabajo es que no se requiera interacción por parte del usuario, además de presuponer un cierto nivel técnico como informático y un amplio conocimiento en el dominio de aplicaciones HPC. Por consiguiente, la aplicación será lanzada desde una terminal con un simple comando, como por ejemplo el mostrado en el comando 6.2.

Comando 6.2: Ejemplo de comando para lanzar la aplicación

```
python3.6 main.py -l fortran -L ../sourcecode/ -v -s
```

Como se puede observar en el comando, el *framework* ofrece una serie de argumentos, gracias a la librería click[6] de *python*, con los que el usuario puede elegir como lanzar la aplicación. Estos argumentos se explican a continuación:

- -l: Especifica el lenguaje de la aplicación a analizar
- -L: Especifica la ubicación del código fuente de la aplicación
- -v: Flag para activar el modo verbose del *framework*
- -s: Ejecuta el *profiling* de manera secuencial

6.8.2. Modularización

Otro de los objetivos de TFG era crear un *framework* modular con el fin de eliminar dependencias funcionales entre los distintos módulos. De esta forma, si algún módulo modifica su funcionalidad interna esto no afecta al resto del *framework* (mientras ese módulo siga respetando las entradas y salidas previamente descritas). Además, una arquitectura modular permite añadir más módulos al *framework* sin tener que reescribir el código de toda la herramienta.

En cuánto a código se refiere, hemos conseguido una estructura modular dividiendo en directorios los distintos módulos, incluyendo los archivos requeridos por cada uno de ellos, como pueden ser copias del código fuente original, archivos de configuración, etc.

6.8.3. Módulos independientes

Con el fin de lanzar la aplicación desde cualquier punto de partido, se han replicado los distintos módulos para poder ser ejecutados independientemente, sin necesidad de ejecutar el *framework*

[6] https://pypi.org/project/click/

al completo. Para ello, se ha implementado una función main() en cada uno de ellos para que puedan ser invocados por el interprete de *Python*. El motivo por el cual no se incluye la función main() en los módulos originales se debe al complejo sistema de importación de librerías de *Python*, el cual prohíbe el uso de funciones main() dentro de otro, ya que es considerado como un *antipattern*[7] Aunque hemos visto que algunos módulos tienen dependencias asociadas, en la gran mayoría de los casos no sería necesario ejecutar dichos módulos ya que se guardan los resultados de estos de forma que puedan ser accesible.

6.8.4. Exportación de resultados

Anteriormente se ha explicado cómo los distintos cálculos y estructuras de datos eran cargados en memoria en tiempo de ejecución. Además, el *framework* nos facilita la visualización de éstos mediante la exportación a ficheros excel/csv, por lo que en caso de duda con la estimación final del consumo de energía, el usuario podría detectar errores mediante el análisis de los ficheros generados durante todo el proceso.

[7] http://python-notes.curiousefficiency.org/en/latest/python_concepts/import_traps.html

Capítulo 7

Resultados

En este capítulo se van a mostrar ejemplos de los resultados obtenidos durante la ejecución del *framework*. Es decir, datos generados por los distintos módulos y en los cuales se basa el desarrollo para poder realizar los cálculos para estimar el consumo de energía de la aplicación HPC. Con el fin de identificar los ficheros generados por cada módulo, estos se agrupan bajo un mismo sistema de ficheros, simplificando así el acceso e identificación del usuario. Al igual que en capítulos anteriores, se irán mostrando los resultados según el orden secuencial en el que se obtienen, Para ello, se van a utilizar aplicaciones reales de tipo HPC.

7.1. Setup experimental

Las dos aplicaciones usadas para comprobar la validez de la herramienta desarrollada son procedentes del *benchmark NAS parallel suite* [21], en concreto se ha trabajado con las aplicaciones "BT" y "SP" (nos referiremos a ellas como "NAS-BT" y "NAS-SP"). Además, se ha utilizado dos clases (Clase B y D) para ambas aplicaciones. Las clases definen el tamaño del problema (datos de entrada de la aplicación) a resolver por la aplicación. La diferencia entre las Clases B y D reside en que la Clase D define un tamaño de problema mayor, por lo que la duración de la ejecución es mucho mayor que en el caso de la Clase B. Las aplicaciones que se han mencionado anteriormente se utilizan de forma amplia en escenarios de tipo HPC debido a que son computacionalmente intensivas, utilizan un paradigma *multi-threaded* o paralelo de utilización de recursos y además, tienen tiempos de ejecución muy elevados. Por lo tanto, se considera que estas aplicaciones son ideales para validar el *framework* presentado en este trabajo.

En las siguientes secciones se van a mostrar los archivos generados durante el proceso completo de estimación rápida de energía. Dicho proceso se presentará con la Clase B de la aplicación "NAS-BT", la cuál ha sido usada durante todo el ciclo de vida de desarrollo del *framework*. Se incluirán archivos ".xslx" y gráficas. Posteriormente, se mostrarán los resultados finales obtenidos al testear la herramienta con ambas aplicaciones y sus respectivas clases, es decir, se mostrará la estimación del consumo de energía obtenida por el *framework* y su comparación con el resultado esperado.

La herramienta siempre es ejecutada bajo un mismo servidor *Intel Enterprise* (S2600GZ) con la distribución CentOS 6.5 de Linux. Además, cuenta con un procesador Intel 6-core Sandy-Bridge

EP con 12 *threads hardware*, 8 módulos de memoria de 4GB, 4 discos duros, 5 ventiladores y 2 fuentes de alimentación.

Por otro lado, también se usan las herramientas externas "doxygen", "ocount" y "MAQAO" para generar el *Call Graph*, recuperar las métricas de los contadores hardware y desensamblar los binarios de los caminos independientes de la aplicación, así como detectar los bucles contenidos en los binarios desensamblados.

7.2. Archivos globales

En primer lugar se van a mostrar los ficheros que facilitan la comprensión de las tareas realizadas por el *framework*. En el cuadro 7.1 se pueden observar los distintos caminos independientes que puede tomar la aplicación desde el nodo raíz (función *main*) hasta un extremo en tiempo ejecución, e.g: [bt, adi, add, $timer\_start$, ...] (Fila 0), es decir, los resultados del call graph module y que van a ser reutilizados por los subsecuentes módulos. El número de fila asignado a cada camino será utilizado por el *framework* como *ID* en tiempo de ejecución para una mayor rapidez en cuánto a lectura y tratamiento de datos se refiere.

0	bt	adi	add	timer_start	elapsed_time	wtime	timer_init
1	bt	adi	add	timer_stop	elapsed_time	wtime	timer_init
2	bt	adi	compute_rhs	timer_start	elapsed_time	wtime	timer_init
3	bt	adi	compute_rhs	timer_stop	elapsed_time	wtime	timer_init
4	bt	adi	x_solve	timer_start	elapsed_time	wtime	timer_init
5	bt	adi	x_solve	timer_stop	elapsed_time	wtime	timer_init
6	bt	adi	y_solve	timer_start	elapsed_time	wtime	timer_init

Cuadro 7.1: Ejemplo de Paths.xlsx

Adicionalmente, la herramienta genera un archivo ".xlsx" donde se registra el tiempo de ejecución de cada módulo, tal y como se observa en el cuadro 7.2. Los tiempos mostrados corresponden a una ejecución secuencial de la clase B de la aplicación "NAS-BT".

Tiempo (s)	**Módulo**
5.70	Call Graph
114.93	Firma de la Aplicación
175.87	Ejecución de la firma
11.91	Reconstrucción de la señal
3.03	Estimación de energía
37.98	Exportación de resultados

Cuadro 7.2: Ejemplo de modulestime.xlsx

7.3. Resultados Módulo *Call Graph*

Aunque ya hemos visto el resultado de este módulo visualmente en el cuadro 7.1, en el capítulo 6 describíamos algunas funcionalidades más realizadas durante la ejecución del módulo, como el

análisis del código fuente. Por tanto, también se generará un directorio por cada camino obtenido, directorios donde se replicará el código fuente original y será adaptado acorde a las funciones incluidas en cada camino.

7.4. Resultados Módulo de Estimación de Instrucciones

A continuación podremos observar cómo se calcula el número de instrucciones estimadas para cada subrutina de la aplicación a analizar. En el cuadro 7.3 podemos ver un extracto de la estimación obtenida para algunas subrutinas de la aplicación y que posteriormente serán utilizadas para el cálculo de los perfiles de potencia. En dicho cuadro se muestra el nombre de la función, con su correspondiente número de instrucciones estimadas, así cómo el cálculo realizado para obtener la estimación.

Function	matvec_sub
Total Instructions	87
Total instructions expression	1+86*1
Function	matmul_sub
Total Instructions	485
Total instructions expression	1+484*1
Function	binvcrhs
Total Instructions	830
Total instructions expression	1+829*1
Function	binvrhs
Total Instructions	293
Total instructions expression	1+292*1

Cuadro 7.3: Ejemplo de instrucciones estimadas de varias subrutinas

Además, se obtienen los pasos intermedios realizados hasta alcanzar dichas estimaciones mediante la generación de archivos de texto. En estos archivos se pueden observar como se establecen los límites superiores de los bucles del código fuente, los cuáles serán también almacenados en una carpeta aparte, denominada “upperbounds” para una mayor accesibilidad en caso de que el usuario desee contrastar las acciones realizadas.

7.5. Firma de la Aplicación

Como se ha descrito en el capítulo 6.4, se necesita generar un archivo binario por cada camino independiente generado en el módulo *Call Graph* para después realizar el *profiling* dinámico. Por tanto, retomaremos los directorios creados para cada camino, y se generará en ellos un archivo nombrado como “make_bin.sh” con los comandos necesarios para compilar el código fuente adaptado, creando así la firma de la aplicación. Los archivos binarios se ubicarán bajo la carpeta “bin”, dentro del directorio de cada camino.

7.6. Métricas de Contadores

En esta sección veremos en qué formato se obtienen las métricas de los contadores para cada camino tras la ejecución de la firma detallada en el capítulo 6.5, así como el tratamiento que se les da posteriormente.

En primer lugar se va a obtener un archivo por cada camino con las métricas de los contadores hardware durante la ejecución de la firma con un intervalo de muestreo de 100ms, tal y cómo se ve en el cuadro 7.4, dónde se muestran por columnas los contadores analizados ($Evento$) y el valor asociado a cada contador.

Tiempo actual (Segundos desde la época): 1564056550.2	
Eventos Contados (Escala) results/cg/source_code_paths/0/bin:	
Evento	**Valor**
BR_INST_RETIRED	34,745,429
BR_MISS_PRED_RETIRED	89,766
CPU_CLK_UNHALTED	116,449,811
INST_RETIRED	199,711,637
LLC_MISSES:0x41	875,147
LLC_REFS:0x4f	3,345,219
arith:fpu_div_active	2,715
l1d:0x1	7,419,772
l2_rqsts:0x1	3,739,138
l2_rqsts:0x3	6,033,304
l2_rqsts:0x8	660,546
l2_rqsts:0x20	21,820
mem_trans_retired:0x2	27,643,736
mem_uops_retired:all_stores	29,512,385
misalign_mem_ref:0x1	39
misalign_mem_ref:0x2	408
resource_stalls:any	18,478,802
uops_dispatched:core	304,918,540

Cuadro 7.4: Ejemplo de métricas de los contadores

A continuación se comienza con el tratamiento de las métricas obtenidas, por lo que debemos cargar en memoria los datos generados, quedando una estructura como la mostrada en el cuadro 7.5. En él se muestra un extracto de las métricas obtenidas para el camino con “ID” 0, con un intervalo de muestreo de 100ms. Estos datos también son exportados a un fichero “.xlsx” y “.csv”.

Posteriormente, pasamos a calcular las medias de las métricas de los contadores para cada camino, que necesitaremos para terminar de reconstruir la señal. En el cuadro 7.6 se observa el archivo generado tras la realización de los cálculos, con un formato similar al cuadro 7.5, pero con una única entrada por camino. Estos datos serán los que debemos de prolongar hasta el tiempo estimado de ejecución de cada camino, tal y como se explicó en el capítulo 6.6.

CAMINO	TIEMPO(s)	BR_INST_RETIRED	BR_MISS_PRED_RETIRED	CPU_CLK_UNHALTED	INST_RETIRED	LLC_MISSES:0x41
0	100	43495944	104934	117293820	258881249	864327
0	200	52939682	76769	124343149	306036003	418332
0	300	43867567	64080	105254249	265379827	388476
0	400	52449711	76774	126225115	314866055	447216
0	500	48808300	68538	108500805	269868298	376066
0	600	46091390	71057	124137524	310185802	452527
0	700	51628683	76103	125391577	303947756	416545
0	800	43138793	63734	106910039	276309066	408965
0	900	51514329	75829	126589440	316497457	450159
0	1000	46415698	65448	105733295	264893494	377544

Cuadro 7.5: Métricas originales de los contadores

CAMINO	BR_INST_RETIRED	BR_MISS_PRED_RETIRED	CPU_CLK_UNHALTED	INST_RETIRED	LLC_MISSES:0x41
0	47813523.75	72031.93	118338939.27	294408649.24	436425.31
1	47493535.06	71735.56	117723824.86	293220240.96	444926.63
2	11514819.95	27154.06	118075866.27	281895615.41	107768.01
3	11501709.80	27078.52	118091567.12	281205886.18	105813.40
4	9289535.10	8719.07	118112314.09	245403858.86	77893.16
5	9268455.07	8580.01	118096212.90	244869860.60	74063.10
6	9255591.94	9295.38	118129576.26	246773077.56	131055.67
7	9253786.68	8401.92	118190482.91	246782029.15	134816.22
8	9442724.99	9009.50	118040785.80	250033220.32	144821.28

Cuadro 7.6: Media de los contadores por camino

Otros datos necesarios para realizar la reconstrucción de la señal son el número de instrucciones estimadas y el número de IPC de cada camino independiente, por lo que también generaremos dos archivos que permitirán analizar si su utilización para el cálculo de estimación del tiempo de ejecución total de la aplicación es correcto. En el cuadro 7.7 se observa un ejemplo del número de instrucciones estimadas para cada camino, así como el número de IPC asociado a los caminos independientes.

Camino	Instrucciones	IPC
0	3711073400	2.48
1	3711073400	2.49
2	91004741900	2.38
3	91004741900	2.38
4	2.05E+11	2.07
5	38384061400	2.07
6	55372061400	2.08
7	1.36E+11	2.08
8	38410061400	2.11

Cuadro 7.7: Ejemplo de Instrucciones estimadas e instrucciones por camino independiente

7.7. Perfiles de potencia

El primer requisito para obtener los perfiles de potencia es realizar la técnica de diezmado comentada en el capítulo 6.7.2. Por lo que procedemos a agrupar muestras cada 10s, creando un fichero similar al mostrado en el cuadro 7.8, donde la primera columna identifica a escala 1:10 el tiempo consumido para la obtención de cada muestra(10s), y el resto de columnas presentan las métricas de los contadores reconstruidas.

	BR_INST_RETIRED	BR_MISS_PRED_RETIRED	CPU_CLK_UNHALTED	INST_RETIRED	LLC_MISSES:0x41
0	1911462384.74	3635383.22	11788505136.86	27368935828.86	17331693.92
1	1069786732.16	2662697.00	11819751858.35	27896724924.18	10636041.64
2	1069786732.16	2662697.00	11819751858.35	27896724924.18	10636041.64
3	1090831618.36	2678741.00	11817554240.32	27852801962.50	10834872.74
4	1114563085.78	2696833.16	11815076075.31	27803271814.21	11059086.53
5	1114563085.78	2696833.16	11815076075.31	27803271814.21	11059086.53
6	1012300926.08	2331581.99	11816098763.64	27780633522.04	9015219.30
7	603252287.28	870577.30	11820189516.95	27690080353.36	839750.37
8	582921707.25	624699.79	11822023448.59	26992513080.22	6090205.26

Cuadro 7.8: Datos de la señal reconstruida tras aplicar la técnica de diezmado

Adicionalmente, cuando se termina de realizar la técnica de diezmado, se generan unas gráficas que permiten analizar de forma más simplificada las métricas obtenidas de los contadores. En la figura 7.1 apreciamos el número de instrucciones retiradas equivalente al de la aplicación original.

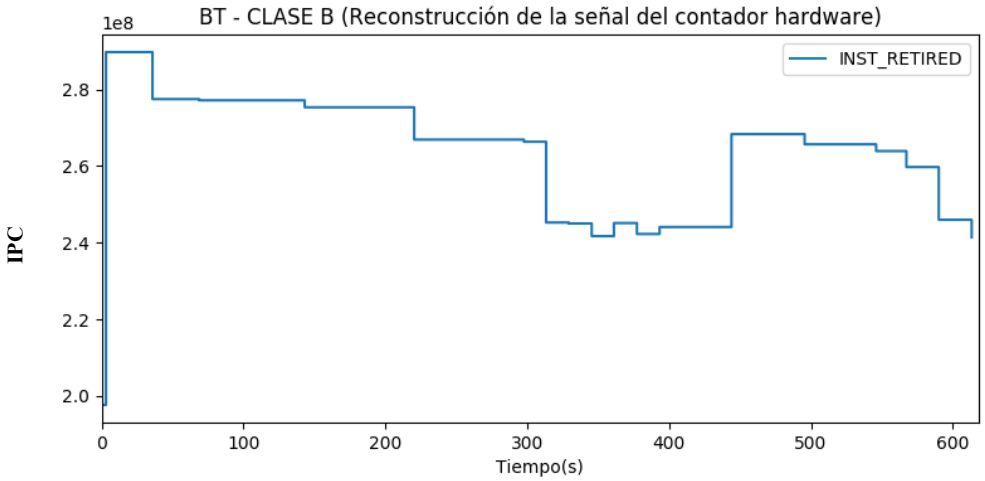

Figura 7.1: Gráfica de las instrucciones retiradas durante el profiling dinámico de la clase B de la aplicación "NAS-BT"

Finalmente procederíamos a la generación de los perfiles de potencia y a la estimación del consumo de energía. En el cuadro 7.9 se muestra un ejemplo de los perfiles de CPU y Memoria obtenidos para cada camino.

CAMINO	POTENCIA DE CPU	POTENCIA DE MEMORIA	TIEMPO
0	8.09	7.22	10.0
1	8.14	7.37	20.0
2	8.14	7.37	30.0
3	8.14	7.36	40.0
4	8.13	7.36	50.0

Cuadro 7.9: Ejemplo de perfiles de potencia

Al igual que se realiza con las métricas de los contadores, adicionalmente la herramienta genera unas gráficas de los perfiles de potencia para aportar un formato más visual y fácil de analizar

para el usuario. En la figura 7.2 se observa el perfil de potencia de CPU obtenido para la clase B de la aplicación “NAS-BT”. Por otro lado, en la figura 7.3 se muestra su perfil de potencia de memoria.

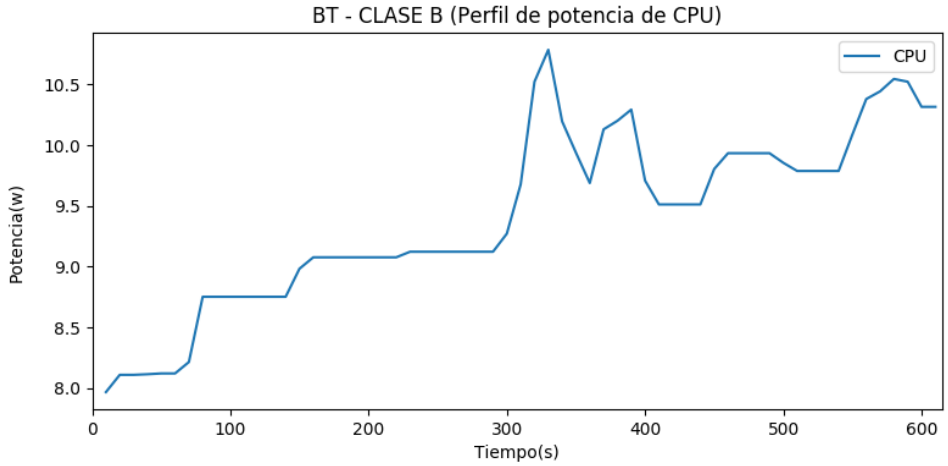

Figura 7.2: Gráfico del perfil de potencia de CPU

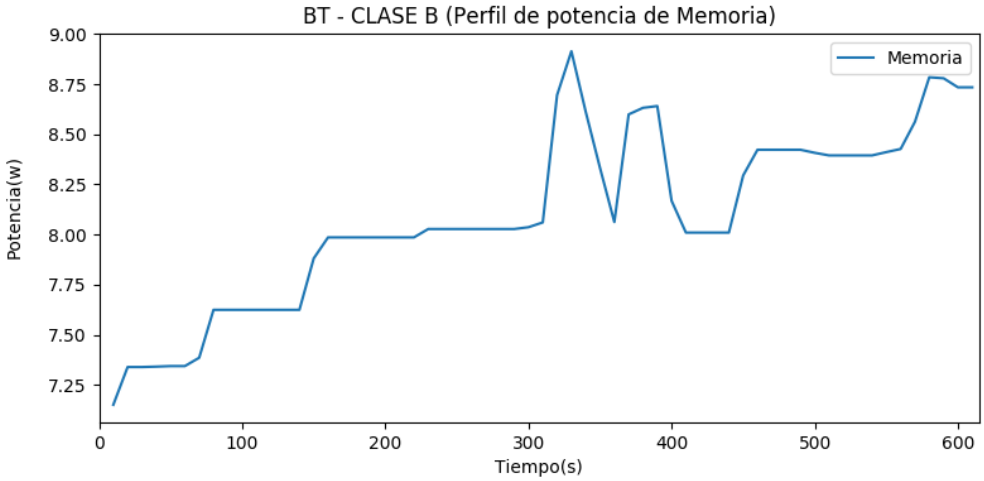

Figura 7.3: Gráfico del perfil de potencia de Memoria

Finalmente, se completa la ejecución del *framework* mediante el cálculo de estimación de energía, cuyo resultado se muestra por consola y se almacena como fichero excel/csv. En la figura 7.10 se muestra la estimación de consumo de energía para la clase B de la aplicación “NAS-BT”

Energía CPU (J)	**Energía Memoria (J)**
6076.04	5137.31

Cuadro 7.10: Ejemplo de consumo de energía estimada

7.8. Validación de resultados

En esta sección vamos a realizar una comparativa de los resultados obtenidos por el *framework* desarrollado con unas estimaciones de consumo de energía realizadas previamente de manera manual, con el fin de verificar que el proceso desarrollado sigue correctamente los pasos descritos en la metodología propuesta. La forma de validar los resultados obtenidos será mediante el cálculo del Ratio de Compresión (RC), y el error en la estimación de energía automática, mostrado en la ecuación 7.1, dónde x equivale a CPU o $Memoria$. Por tanto, E_{CPU} o $E_{Memoria}$ corresponden al valor de estimación de energía de CPU o de Memoria realizado mediante un proceso manual previo a la realización de este trabajo, mientras que $\widehat{E}_{CPU}$ y $\widehat{E}_{Memoria}$ corresponderían a las estimaciones de energías realizadas por el *framework* desarrollado.

$$Error_x = \frac{\left|E_x - \widehat{E}_x\right|}{E_x} \times 100 \tag{7.1}$$

7.8.1. Aplicación NAS-BT, Clase B

A lo largo de este capítulo se han mostrado cuadros y figuras correspondientes a la clase B de la aplicación "NAS-BT", así como la propia estimación de energía asociada a ella. Por tanto, nos queda pendiente validar que los resultados obtenidos son correctos.

Primero, vamos a mostrar el perfil de potencia obtenido manualmente, mostrado en la figura 7.4. Ahí podemos observar el perfil obtenido con un intervalo de muestreo de 10s, a diferencia del generado por el *framework* en la figura 7.2, que es de 1s. Aún así, vemos que los valores obtenidos se muestran en el mismo rango, pero no son idénticos, ya que el orden de ejecución de los caminos independientes no son iguales. Por el contrario, vemos que el tiempo total de ejecución de la aplicación si es similar, la cuál es algo superior a los 600 segundos en ambas figuras.

Ahora se van a mostrar las estimación de energía realizada tanto manualmente como automáticamente para comparar resultados. En el cuadro 7.11 vemos la estimación manual de la energía de CPU y memoria, así como el resultado obtenido por la herramienta, comprobando que son muy similares. Aplicando la ecuación de error de estimación de energía nos da un resultado de 4.65 % para la energía de CPU, y un error del 2.35 % para la energía de memoria, obteniendo así resultados muy precisos en ambos casos.

BT-B	**Energía CPU (J)**	**Energía Memoria (J)**
Estimación Manual	5806,40	5019,40
Estimación Automática	6076.04	5137,31

Cuadro 7.11: Energía estimada de la aplicación NAS-BT, clase B

7.8.2. Aplicación NAS-BT, Clase D

A continuación vamos a comparar los resultados obtenidas de la clase D de la misma aplicación, "NAS-BT". De igual manera comenzaremos visualizando los perfiles de potencia, y después el resultado de la estimación de energía.

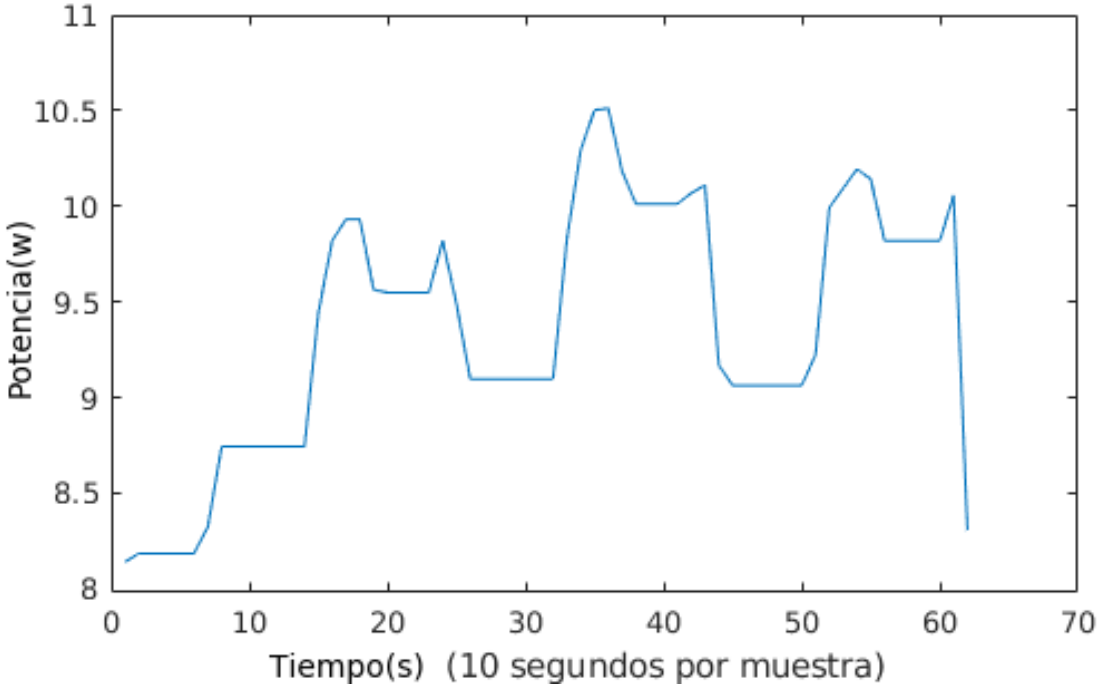

Figura 7.4: Perfil de potencia de CPU de la clase B de la aplicación NAS-BT

En la figura 7.5 se muestra el perfil de potencia realizado manualmente, y en la figura 7.6 se muestra el generado en la herramienta. Comparando ambos observamos que el rango de valores es similar, así como el tiempo total de ejecución de la aplicación, por lo que se ha obtenido el valor esperado. Al igual que en la clase B, se puede ver que las gráficas no son iguales debido al orden de ejecución de los distintos caminos independientes.

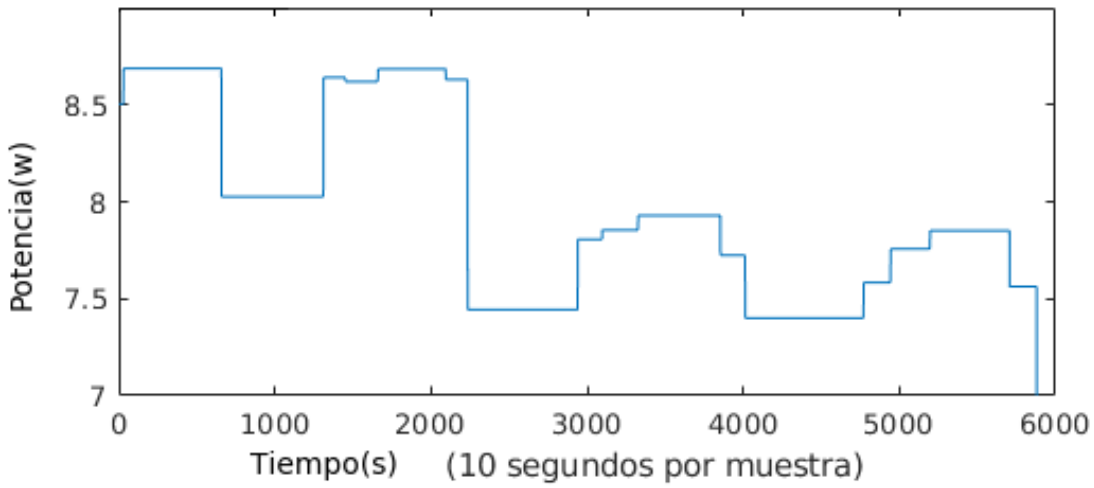

Figura 7.5: Estimación manual del perfil de potencia de CPU de la clase D, aplicación NAS-BT

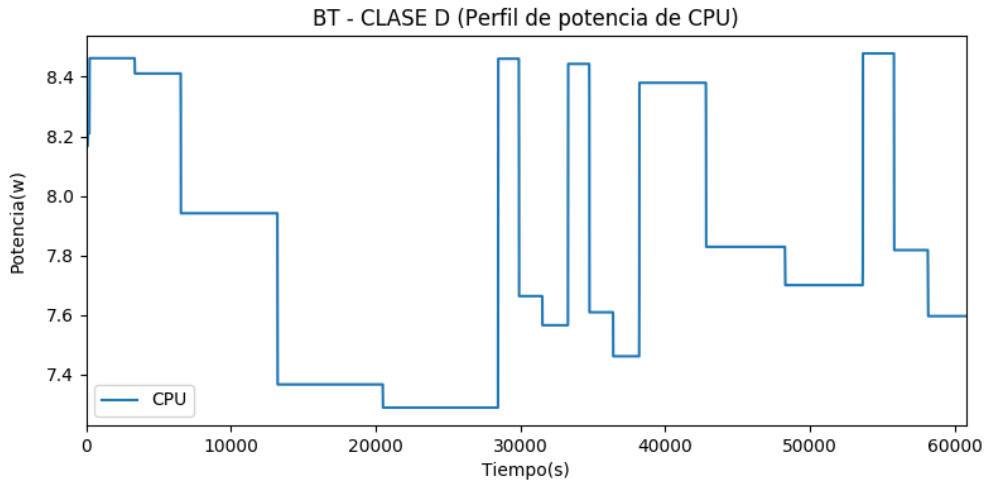

Figura 7.6: Estimación automática del perfil de potencia de CPU de la clase D, aplicación NAS-BT

Por último, vamos a ver si el consumo de energía estimado se corresponde con el proceso previo realizado manualmente. En el cuadro 7.13 se detalla el consumo de energía estimado previamente de manera manual, además del resultado obtenido tras la ejecución del *framework*, y comprobamos que son prácticamente idénticos, por lo que el proceso automatizado funciona correctamente. Para verificarlo, aplicamos la ecuación del error de estimación de energía, obteniendo unos resultados del 1.97 % y del 1.18 % respecto a las estimaciones de energía de CPU y Memoria respectivamente.

BT-D	**Energía CPU (J)**	**Energía Memoria (J)**
Estimación Manual	468281,80	468355,80
Estimación Automática	477521,20	473897,90

Cuadro 7.12: Energía estimada de la aplicación NAS-BT, clase D

7.8.3. Aplicación NAS-SP, Clase B

En este apartado vamos a analizar los resultados obtenidos para la aplicación “NAS-SP”. Al igual que la anterior, está desarrollada en el lenguaje *fortran* y cuenta con dos clases a estudiar, la clase B y la clase D.

Empezando por la clase B, en las figuras 7.7 y 7.8 podemos observar los perfiles de potencias obtenidos, tanto manual como automáticamente. Al igual que en los resultados anteriores, vemos que las señales no son similares, pero tanto el rango de valores como el tiempo total de ejecución son prácticamente idénticos.

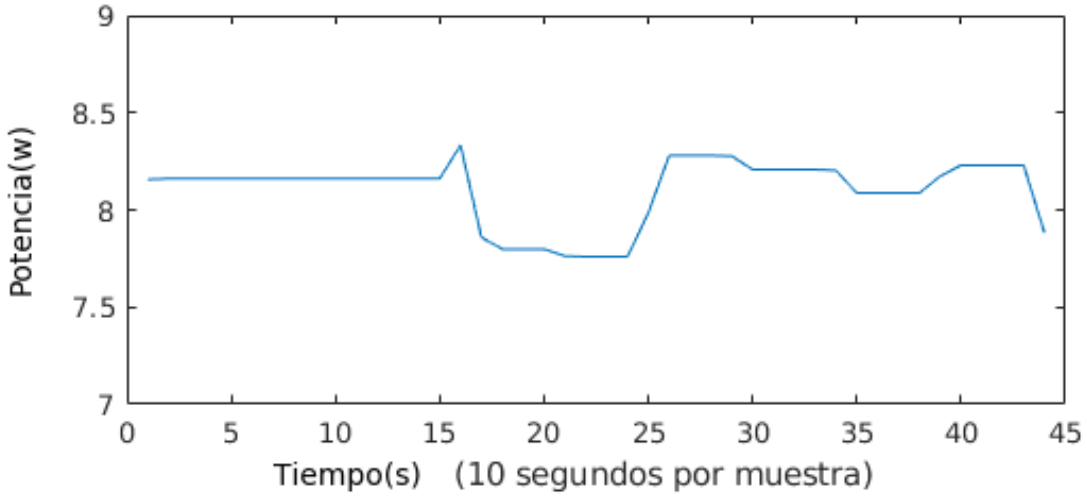

Figura 7.7: Estimación manual del perfil de potencia de CPU de la clase B, aplicación NAS-SP

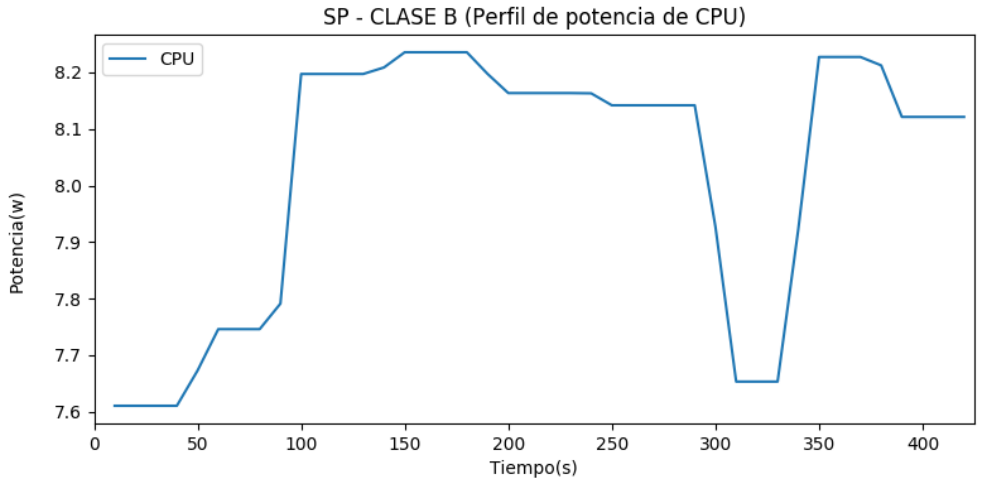

Figura 7.8: Estimación automática del perfil de potencia de CPU de la clase D, aplicación NAS-SP

En cuánto a la estimación de energía, en el cuadro 7.13 podemos apreciar la diferencia entre los resultados esperados y los obtenidos. Aplicando el error de estimación de energía, obtenemos un porcentaje de error del 5.06 % para la energía de CPU, y de un 4.79 % para la energía de memoria. Por tanto, hallamos una leve diferencia en las estimaciones de energía, pero dentro de los valores esperados, así que podemos decir que el *framework* también es válido para esta aplicación.

SP-B	**Energía CPU (J)**	**Energía Memoria (J)**
Estimación Manual	3564,97	3235,47
Estimación Automática	3384,82	3080,80

Cuadro 7.13: Energía estimada manualmente de la aplicación NAS-SP, clase B

7.8.4. Aplicación NAS-SP, Clase D

En esta sección vamos a estudiar la última aplicación testeada para la validación de la herramienta desarrollada. Se trata de la misma aplicación que en el ejemplo anterior, pero compilaremos y ejecutaremos su clase D. En las figuras 7.9 y 7.10 se pueden observar los perfiles de potencia esperados y el obtenido durante la ejecución del *framework*. Una vez más, se aprecia que ambos son similares y que el proceso automatizado se ejecuta de acuerdo a las expectativas

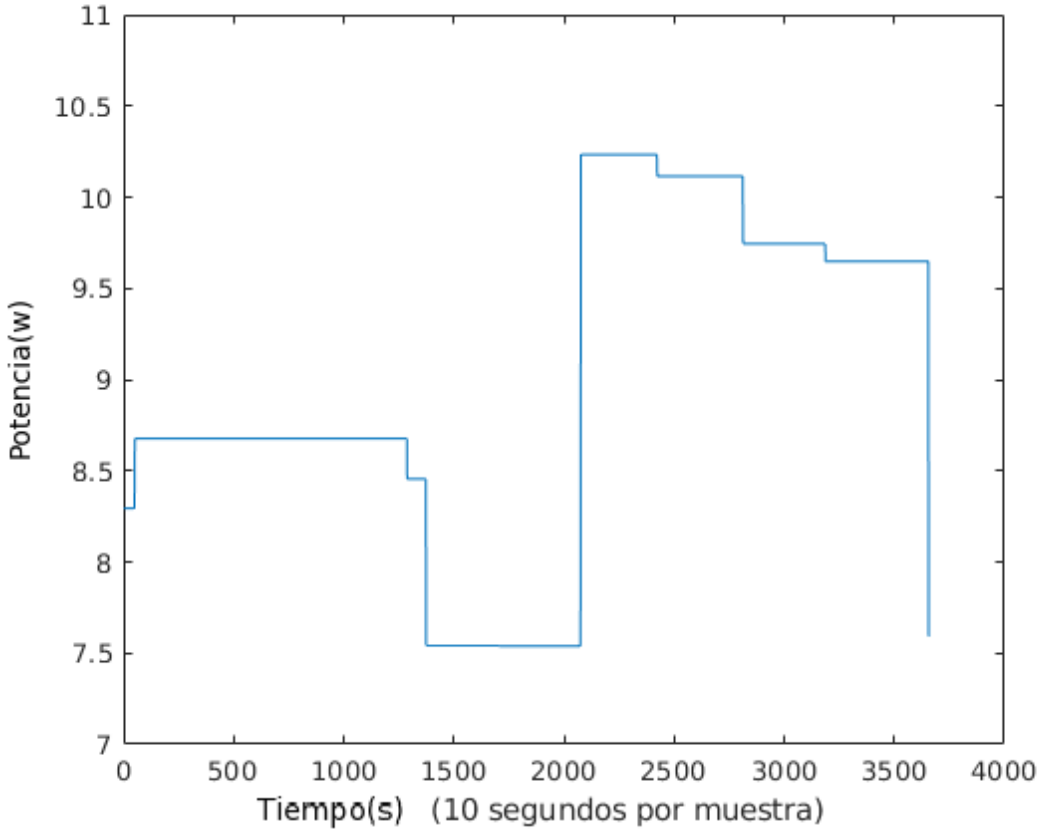

Figura 7.9: Estimación manual del perfil de potencia de CPU de la clase D, aplicación NAS-SP

Para finalizar, en el cuadro 7.14 se muestran los consumos de energía estimados previamente de forma manual, y la estimación realizada por la herramienta, comparándolos obtenemos un error en la estimación de energía de 8.14 % y un 13.60 % respecto a las estimaciones manuales de energía de CPU y memoria respectivamente.

SP-D	**Energía CPU (J)**	**Energía Memoria (J)**
Estimación Manual	328384,30	291939,80
Estimación Automática	355107,52	331635,50

Cuadro 7.14: Comparativa de la estimación de energía de la clase D de la aplicación SP

En definitiva, se puede concluir que el desarrollo de la herramienta cumple con la expectativa

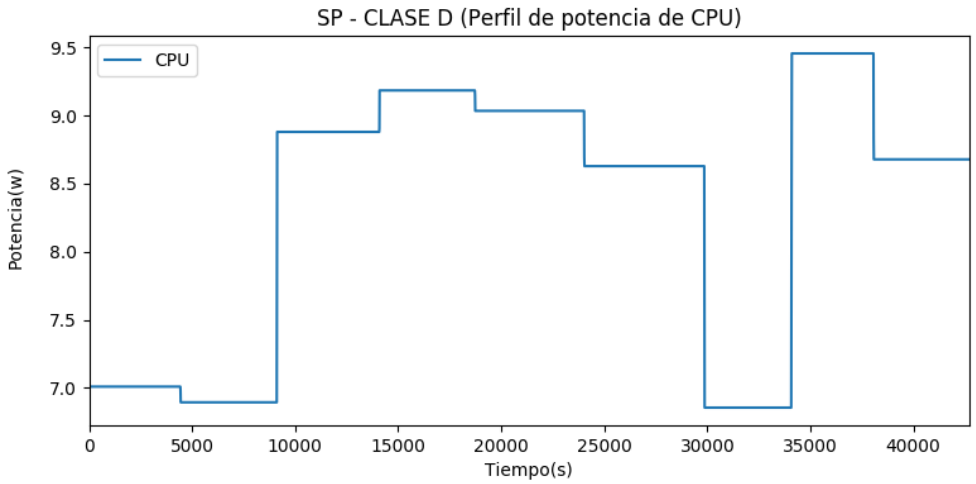

Figura 7.10: Estimación automática del perfil de potencia de CPU de la clase D, aplicación NAS-SP

de los resultados calculados de forma manual.

7.9. Ratios de Compresión

En esta última sección de resultados, se va a verificar que efectivamente, la herramienta desarrollada realiza el proceso de estimación de energía más rápidamente que si tuviéramos que ejecutar cada una de las aplicaciones por completo. Con este fin, debemos de calcular el ratio de compresión (RC) explicado en el capítulo 5, dónde se explicaba como el ratio del tiempo total de ejecución de las aplicaciones entre el tiempo empleado por el *framework* para obtener la estimación de energía.

Aplicación	Tiempo de ejecución original de la aplicación(s)	Tiempo de ejecución del Framework(s)	Ratio de Compresión
BT-B	603.8	311.44	1.9
BT-D	54928.0	531.17	103.4
SP-B	456.9	198.72	2.3
SP-D	38902.9	391.67	99.3

Cuadro 7.15: Ratio de compresión de las aplicaciones testeadas en el *framework*

En el cuadro 7.15 podemos observar el tiempo total de ejecución de cada aplicación original, así como el empleado por el *framework* para obtener el consumo de energía. También se muestra el ratio de compresión, dónde podemos ver que efectivamente, la herramienta desarrollada es mucho más rápido que la ejecución completa, sobretodo para las clases "D" de ambas aplicaciones, las cuáles tienen un mayor tiempo de ejecución (de más de 10 horas) debido a que las matrices a ejecutar son mucho mayores que las de las clases B. Por ejemplo, en el caso de "BT-D" el *framework* estima el consumo energético de forma automática en un tiempo de ejecución 103 veces más rápido que la ejecución original de "BT-D". Por tanto, se puede concluir que el *framework* es más efectivo cuánto más extensa sea la aplicación a analizar.

En el caso de las Clases B (ejecuciones por debajo de 10 minutos) se puede observar una penalización en términos de CR. Esto se puede deber a diversas causas, entre ellas: 1) La decisión actual de parar la ejecución de la firma a los 40s, lo cual implica en algunos casos ejecuciones de los caminos independientes por encima del 50 % de la ejecución original, resultando una firma no tan reducida respecto a la aplicación original. 2) Es posible que algunos módulos del *framework* no estén optimizados en cuanto a rendimiento se refiere. Es decir, algunos módulos podrían optimizarse de manera que se ejecuten más rápidamente, o incluso que se ejecuten algunos módulos en paralelo.

Capítulo 8

Conclusión y Trabajos Futuros

A continuación, vamos a presentar las conclusiones del trabajo desarrollado en este TFG, así como los posibles trabajos a realizar con vistas a corto plazo.

8.1. Conclusión

En el presente trabajo hemos detallado la implementación de un *framework* que tenía como finalidad la estimación del consumo de energía de aplicaciones HPC, mediante las instrucciones y el diseño desarrollado en una metodología previa, con la intención de demostrar que es posible automatizar el proceso de obtención de la estimación de energía sin la necesidad de ejecutar la aplicación completamente. Es decir, a través de la extracción de la firma de la aplicación.

Observando los resultados y los ratios de compresión (RC) obtenidos, podemos afirmar que la herramienta aquí implementada es capar de realizar una estimación de energía rápida, con una mayor eficiencia sobre aquellas que requieren un mayor tiempo de ejecución, ya que se han llegado a obtener unos ratios de compresión alrededor de 100 para aquellas aplicaciones que corresponden a un periodo de ejecución superiores a 10 horas (Clases D). Esto indica que el *framework* es capaz de estimar la energía 100 veces más rápido que la ejecución original de la aplicación.

En cuanto a la precisión del *framework* respecto a la estimación de energía, hemos obtenido resultados con un porcentaje de error no superior al 13.6 % en relación a aplicaciones reales de tipo HPC. Siendo la media de error de un 5.21 %, lo que hace que podamos afirmar que la herramienta desarrollada es bastante precisa cuando se compara con los resultados de estimación energética obtenidos mediante un proceso manual.

Para finalizar, podemos afirmar que se han cumplido los objetivos propuestos a principios de proyecto. Puesto que el *framework* realiza todas las tareas de manera automática, no se requiere interacción por parte del usuario, se han implementado módulos con dependencias mínimas, útil tanto para ejecutarlos de forma independiente, como para continuar con futuros desarrollos de la herramienta.

8.2. Trabajos Futuros

A lo largo del trabajo explicado, se han podido observar algunos detalles que pueden acotar el verdadero alcance del *framework* desarrollado. Los cuáles son los siguientes:

- Actualmente la herramienta solo es capaz de analizar aplicaciones HPC desarrolladas en el lenguaje Fortran, el cuál es bastante utilizado en el sector HPC, pero como idea se propone la expansión de códigos soportados, como pueden ser C o C++. El código se ha desarrollado para que se puedan añadir lenguajes de la forma más simplificada posible.
- Otra propuesta de mejora es el desarrollo de un algoritmo robusto que para la ejecución de la firma en un instante óptimo, de forma que las ejecuciones no tengan un límite de tiempo de 40s, tal y cómo ocurre actualmente, sino que se base en dos criterios concretos: 1) La ejecución de un porcentaje de instrucciones estimadas, y 2) la estabilidad de la señal de los contadores hardware.
- Por último, se podría revisar el rendimiento de la aplicación, pues hay módulos cuyo tiempo de ejecución consideramos son algo elevados. Por ejemplo, se podrían paralelizar tareas como el *parseo* de código fuente para los caminos independientes, o incluso módulos sin dependencias entre sí.

Conclusions and Future Works

On this chapter, we are going to present the conclusions related to the framework developed on this work, just like possible tasks to perform on a short-term sight.

Conclusions

In this work we have described the framework's implementation in order to achive a fast energy consumption estimation for HPC applications, throughout the instructions and design detailed on a previous study, with intends to proof that it is possible to automatize the process by which the estimation of the energy consumption is gathered without executing the application completely.

Based on the results provided and the Compression Ratios obtained, we might say that the framework is capable of performing a fast energy consumption estimation, with a robust efficiency over those applications with a larger execution time due to the fact that we have obtained CR over a 100 for those with an execution time greater than 10 hours (Class D). Meaning that the framework is able to retrieve the energy estimation a 100 times faster.

In terms of accuracy of the framework related to the energy consumption estimation, the results shows an error percentage not greater that a 13.60 % concerning real HPC applications. Being the average percentage equal to 5.21 %, so that we can say that the framework is quite precise.

Finally, we can confirm that the goals set at the beginning of the project has been satisfied, since the framework is able to perform the tasks automatically, with no user interaction and the different modules has been implemented with the least dependencies, easing both, the execution of single modules and the development of future tasks.

Future works

Over the project description, we have seen some details that can constrain the scope of the framework developed, which are the next ones:

- Currently the tool is only adapted to analyze HPC applications programmed on Fortran language, which is quite common in the sector, but as future works we think that it would be a good idea to increment the number of supported languages, such as C or C++. With this in mind, the framework has been implemented so that others languaged can be added easily.
- Another enhancement proposal is the development of a robust algorithm able to stop the execution of the application signature on the right instant, so that the profiling doesn't have

a maximum execution time of 40s, as it is right now. The idea is to implement an algorithm based on two main criterias: 1) The execution of a percentage of the estimated instructions, and 2) the stability of the hardware counter's signal.

- Lastly, the performance of the framework might have potential for improvements, as there are some modules whose executions time are a bit high. For instance, some tasks like the code parsing of the different paths, might be a good candidate for paralelizing purposes, as well as the parallel execution of modules with no dependencies associated between them.

Bibliografía

[1] J. C. Salinas-Hilburg, M. Zapater, J. M. Moya, and J. L. Ayala, "Fast energy estimation through partial execution of hpc applications," in *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pp. 1–8, July 2018.

[2] J. C. Salinas-Hilburg *et al.*, "Unsupervised power modeling of co-allocated workloads for energy efficiency in data centers," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2016.

[3] J. Balladini, M. E. Morán, C. Rozas, A. D. Giusti, R. Suppi, D. R. del Rosario, and E. Luque, "Consumo energético de sistemas de computación de altas prestaciones," 2015.

[4] S. Filiposka, A. Mishev, and C. Juiz, "Current prospects towards energy-efficient top hpc systems," *Comput. Sci. Inf. Syst.*, vol. 13, pp. 151–171, 2016.

[5] B. Whitehead *et al.*, "Assessing the environmental impact of data centres part 1: Background, energy use and metrics," *Building and Environment*, vol. 82, pp. 151–159, 2014.

[6] L. A. Barroso, J. Clidaras, and U. Hölzle, "The datacenter as a computer: An introduction to the design of warehouse-scale machines," *Synthesis lectures on computer architecture*, vol. 8, no. 3, pp. 1–154, 2013.

[7] J. C. Saez, A. Pousa, R. Rodríguez-Rodríguez, F. Castro, and M. Prieto-Matias, "Pmctrack: Delivering performance monitoring counter support to the os scheduler," *The Computer Journal*, vol. 60, pp. 60–85, Jan 2017.

[8] J. Shuja, S. Madani, K. Bilal, K. Hayat, S. Khan, and D. S. Sarwar, "Energy-efficient data centers," *Computing*, vol. 94, pp. 973–994, 09 2012.

[9] X. Zhan and S. Reda, "Techniques for energy-efficient power budgeting in data centers," pp. 1–7, 05 2013.

[10] C. Lefurgy, X. Wang, and M. Allen-Ware, "Power capping: A prelude to power shifting," *Cluster Computing*, vol. 11, pp. 183–195, 06 2008.

[11] A. Shabbir, K. A. Bakar, R. Z. R. M. Radzi, and M. Siraj, "Resource management in cloud data centers," *International Journal of Advanced Computer Science and Applications*, vol. 9, no. 10, 2018.

[12] T. Makris, "Measuring and analyzing energy consumption of the data center," g2 pro gradu, diplomityö, 2017-08-28.

[13] W. L. Bircher and L. K. John, "Complete system power estimation using processor performance events," *IEEE Transactions on Computers*, vol. 61, pp. 563–577, April 2012.

[14] R. Joseph and M. Martonosi, "Run-time power estimation in high performance microprocessors," in *ISLPED'01: Proceedings of the 2001 International Symposium on Low Power Electronics and Design (IEEE Cat. No.01TH8581)*, pp. 135–140, Aug 2001.

[15] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, and L. Benini, "Scheduling-based power capping in high performance computing systems," *Sustainable Computing: Informatics and Systems*, vol. 19, pp. 1–13, 2018.

[16] H. Shoukourian, T. Wilde, A. Auweter, and A. Bode, "Predicting the energy and power consumption of strong and weak scaling hpc applications," *Supercomputing Frontiers and Innovations*, vol. 1, no. 2, 2014.

[17] A. Lewis, S. Ghosh, and N.-F. Tzeng, "Run-time energy consumption estimation based on workload in server systems," in *Proceedings of the 2008 Conference on Power Aware Computing and Systems*, HotPower'08, (Berkeley, CA, USA), pp. 4–4, USENIX Association, 2008.

[18] A. Wong, D. Rexachs, and E. Luque, "Parallel application signature for performance analysis and prediction," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 7, pp. 2009–2019, 2015.

[19] S. Sodhi and J. Subhlok, "Skeleton based performance prediction on shared networks," in *IEEE Int. Sym. on Cluster Computing and the Grid. CCGrid 2004.*, 2004.

[20] L. T. Yang *et al.*, "Cross-platform performance prediction of parallel applications using partial execution," in *Supercomputing, 2005.*, 2005.

[21] D. Bailey *et al.*, "The Nas Parallel Benchmarks," *The International Journal of Supercomputing Applications*, vol. 5, no. 3, pp. 63–73, 1991.

[22] A. Sîrbu and O. Babaoglu, *Power Consumption Modeling and Prediction in a Hybrid CPU-GPU-MIC Supercomputer*, pp. 117–130. Cham: Springer International Publishing, 2016.

Agradecimientos

A todos aquellos que me han apoyado en la realización de este trabajo, en especial a José Luis Ayala y Juan Carlos Salinas, cuyas instrucciones han sido imprescindibles.