UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Modelado de una Arquitectura de Monitorización de Prestaciones para el Simulador gem5

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Pecino Fernández, Manuel Antonio

Tutor/a: Petit Martí, Salvador Vicente

Cotutor/a: Sahuquillo Borrás, Julio

CURSO ACADÉMICO: 2023/2024

Resum

La simulació de processadors s'ha convertit en una ferramenta essencial en l'enginyeria de computadors. Oferix una solució àgil i econòmica a l'hora d'analitzar de manera precisa i eficient propostes estructurals i arquitectòniques de les CPUs, sent la principal alternativa enfront de la tradicional metodologia de desenvolupar i construir costosos prototips per cada solució proposada. Un dels seus principals avantatges és la capacitat d'anàlisi intensiva del rendiment del processador, ja que proporciona un gran detall i coneixement sobre les etapes d'execució, l'estat de la màquina i el seu acompliment, permetent analitzar així, condicions extremes de funcionament i situacions dificilment emulables en un entorn físic de laboratori.

En el present treball es realitza la implementació d'una unitat de monitoratge de prestacions per al simulador de codi obert Gem5. La unitat es prova de manera intensiva mitjançant l'execució de benchmarks. S'implementa també la capacitat de monitorar aplicacions executant-se en Simultaneous Multi Threading, després es valida esta implementació mitjançant l'execució simultania de benchmarks SPEC de parelles acuradament seleccionades per a després, realitzar un anàlisi Top-Down.

Paraules clau: Monitorització i avaluació de prestacions, Gem5, microarquitectura, Anàlisi Top-Down, Simulació de Processador, PMU

Resumen

La simulación de procesadores se ha convertido en una herramienta esencial en la ingeniería de computadores. Ofrece una solución ágil y económica a la hora de analizar de forma precisa y eficiente propuestas estructurales y arquitectónicas de las CPUs, siendo la principal alternativa frente a la tradicional metodología de desarrollar y construir costosos prototipos por cada solución propuesta. Una de sus principales ventajas es la capacidad de análisis intensivo del rendimiento del procesador, otorgando un gran conocimiento sobre las etapas de ejecución, el estado de la máquina y su desempeño, permitiendo analizar así, condiciones extremas de funcionamiento y situaciones difícilmente emulables en un entorno físico de laboratorio.

En el presente trabajo se realiza la implementación de una unidad de monitorización de prestaciones para el simulador de código abierto Gem5. La unidad se prueba de forma intensiva mediante la ejecución de benchmarks SPEC. Se implementa también la capacidad de monitorizar aplicaciones ejecutándose en Simultaneous Multi Threading, luego se valida esta implementación mediante la ejecución simultanea de benchmarks SPEC de parejas cuidadosamente seleccionadas para posteriormente realizar un análisis Top-Down.

Palabras clave: Monitorización y evaluación de prestaciones, Gem5, Microarquitectura, Análisis Top-Down, Simulación de Procesador, PMU

Abstract

Simulation has become an essential tool in computer engineering. It offers an agile and cost-effective solution to accurately and efficiently analyse structural and architectural proposals for CPUs, being the main alternative to the traditional methodology of developing and building costly prototypes for each proposed solution. One of its main advantages is the capacity for intensive analysis of processor performance, providing a great deal of knowledge about the stages of execution, the state of the machine and its performance, thus making it possible to analyse extreme operating conditions and situations that are difficult to emulate in a physical laboratory environment.

This paper implements a performance monitoring unit for the Gem5 open source simulator. The unit is intensively tested by means of the execution of SPEC BENCHMARKS. The ability to monitor applications running in Simultaneous Multi Threading is also implemented, then this implementation is validated by running simultaneous SPEC BENCHMARKS of carefully selected pairs, after, a Top-Down analysis is done.

Key words: Performance monitoring and evaluation, Gem5, Microarchitecture, Top-Down Analysis, Processor Simulation, PMU

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

1.1 Descripción del Problema

El paradigma tecnológico es fuertemente cambiante. Vivimos en un mundo cada vez más digitalizado en el cual la tecnología ha conquistado cada aspecto de nuestras vidas. El mundo moderno, aunque en muchos aspectos reacio al cambio, es innegable que es un mundo dominado por el avance tecnológico. Socialmente nos hemos acostumbrado a que cada pocos meses los productos y servicios punteros ya estén obsoletos. Prueba de esto es la llamada Ley de Moore. Este ingeniero predijo ya en la década de los 60 la rápida evolución de las prestaciones de las unidades de procesamiento aumentando la cantidad de transistores por chip en un 40 %-55 % anual duplicándose cada 18-24 meses [2]. A pesar de que ya no vivimos en tiempos donde se cumpla esta ley, aunque decelerado, el aumento no se detiene. Esta situación causa que se eleve también su coste de producción, problemática queda muy bien explicada por J.Hennessy *et al.* en [1].

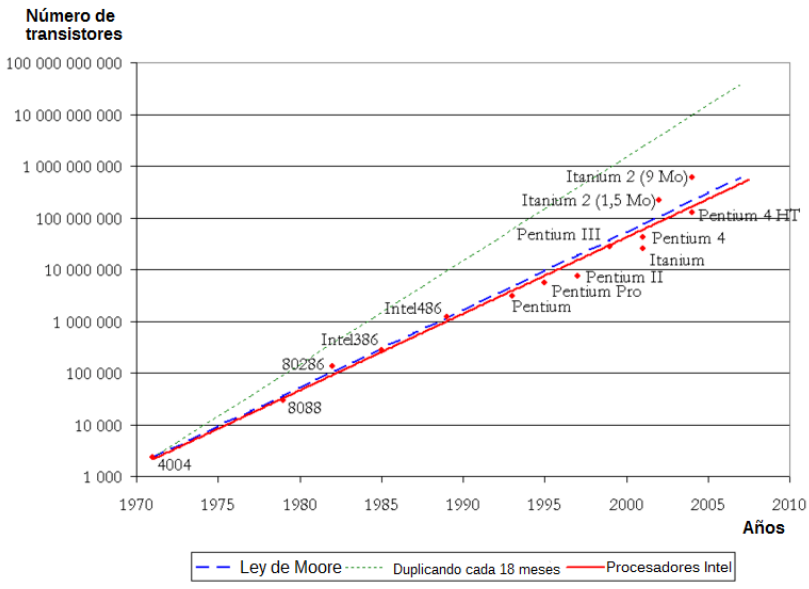

Figura 1.1: Ley de Moore. Fuente [9].

Este avance no sería posible sin una vanguardia, tanto de la industria como académica, que proponga y estudie soluciones a los problemas que van surgien-

do. Esta vanguardia se ha valido siempre de las mejores técnicas para evaluar las problemáticas del momento y lograr las mejores soluciones posibles. El problema surge con los costes de producción, diseñar, prototipar y probar un diseño es un procedimiento no solamente costoso en tiempo sino también en dinero. Esto se indica en [1]: cada año toma más importancia la necesidad no sólo de buscar soluciones para mejorar el rendimiento, sino hacerlo de las formas más económicas posibles.

Es bajo este paradigma que emergió como una posible solución la simulación de CPU. Esta técnica supone una alternativa más ágil y económica frente a la ardua labor de desarrollar, fabricar y evaluar diferentes prototipos para una sola propuesta. La simulación nos permite agilizar el desarrollo, abaratar costes y sobretodo un gran ahorro de recursos. Otra de las ventajas que presenta la simulación de CPU es que nos permite evaluar situaciones de gran estrés o situaciones muy específicas de funcionamiento las cuales son difícilmente replicables en un laboratorio.

La evaluación de prestaciones de los programas sobre entornos reales de ejecución se realiza mediante las llamadas *Performance Monitoring Unit* (PMU). Estas unidades son unidades funcionales físicas que son capaz de medir los rendimientos de los programas en procesadores reales y su desempeño en las diferentes regiones de la CPU. Estas unidades son esenciales si queremos dimensionar correctamente tanto nuestra máquina como los programas que vamos a ejecutar en ella.

Sin embargo, aun teniendo los simuladores una gran importancia en el diseño de procesadores, no modelan el funcionamiento de las PMU de los procesadores reales. Es por ello que los análisis que se pueden realizar en máquina real, como el análisis Top-Down de Intel [6], no se pueden realizar en entornos simulados. Este tipo de análisis de prestaciones resultan muy útiles para dimesionar las estructuras del processador en la fase de diseño, precisamente donde más utilizados son los simuladores.

En este trabajo se persigue abordar este problema modelando una PMU en un simulador detallado de procesadores que permita realizar análisis antes sólo disponibles en máquinas reales. Para llevar a cabo este objetivo, se ha modificado el simulador Gem5[1]. Gem5 es un simulador en constante desarrollo. Su modularidad y construcción en código abierto permite un amigable entorno donde diferentes personas alrededor del mundo pueden contribuir de forma anónima y libre. Gem5 a pesar de aportar cuantiosos datos y estadísticas sobre la ejecución de los programas, no cuenta hasta el momento con una implementación propia de una PMU. Este Trabajo de Fin de Grado se centrará en desarrollar e implementar una PMU para este simulador.

[1] `https://www.gem5.org/`

1.2 Motivación

Como miembro del Grupo de Arquitecturas Paralelas[2] (GAP) he desarrollado un amplio interés por el desarrollo de nuevas soluciones a problemas existentes. El grupo cuenta con profesionales con amplia experiencia en el sector. Recientemente en el grupo ha surgido la idea de estudiar y optimizar la *performance stack* y para ello se me ha propuesto desarrollar en un simulador una unidad de monitorización de prestaciones para estudiar como afectan pequeños cambios de la microarquitectura al rendimiento de un programa. Debido a estos motivos expuestos es por lo que he decidido desarrollar el presente Trabajo de Fin de Grado.

1.3 Objetivos

El principal objetivo de este trabajo es implementar en el simulador Gem5 una unidad de monitorización de prestaciones conocida por sus siglas en inglés PMU. Para ello se han perseguido los siguientes objetivos:

- Implementación de una unidad de monitorización de prestaciones en el simulador de propósito general Gem5 [4].
- Validar el funcionamiento de la implementación mediante la ejecución de la *suite* de *benchmarks* SPEC CPU2006[3] y SPEC CPU2017[4] y la realización de un análisis Top-Down.
- Ampliación del uso de la PMU y la *performance stack* a ejecución multihilo SMT.
- Validar la implementación del SMT mediante parejas seleccionadas metódicamente de *benchmarks* SPEC[5].

1.4 Metodología

La metodología empleada ha sido el desarrollo en cascada, el cual es una forma rígida, metódica y estructurada de desarrollar un programa. Para la culminación del trabajo se han ido completando etapas de forma secuencial hasta lograr el objetivo. Estas etapas han sido las siguientes.

- Etapa de estudio y comprensión del modelo de CPU simulado en Gem5, etapas, pipeline y funcionamiento.
- Diseño de la solución, donde se ha planteado la *performance stack* a implementar.

[2] `https://www.gap.upv.es/`
[3] `https://www.spec.org/cpu2006/`
[4] `https://www.spec.org/cpu2017/`
[5] `https://www.spec.org/`

- Desarrollo de la unidad de monitorización de prestaciones.
- Validación del modelo, donde se ha puesto a prueba y se ha evaluado su funcionamiento mediante el análisis Top-Down.

1.5 Estructura del Trabajo

El trabajo está organizado en los siguientes capítulos:

- El Capítulo 2 enumera simuladores actuales del mercado. Posteriormente comenta trabajos relacionados tanto a nivel profesional como presentados con anterioridad en la ETSINF.
- El Capítulo 3 presenta el análisis mediante el cual se va a evaluar la implementación de la unidad.
- El Capítulo 4 describe la estructura y funcionamiento de la unidad funcional propuesta para este Trabajo de Fin de Grado, la *performance stack* desarrollada y sus motivos.
- El Capítulo 5 detalla el procedimiento llevado a cabo para desarrollar la unidad, implementarla en el simulador y lanzar las cargas de ejecución.
- El Capítulo 6 presenta las tecnologías, herramientas, y cargas de aplicaciones empleadas para la evaluación y análisis de la propuesta.
- El Capítulo 7 presenta los resultados experimentales obtenidos de la ejecución de las *suites* de *benchmarks* SPEC CPU2006 y CPU2017 y la realización del análisis Top-Down.
- Finalmente, el Capítulo 8 concluye el trabajo y muestra de forma ordenada las conclusiones, la relación con los estudios cursados y ampliaciones a futuro.
- Se han añadido tres apéndices, archivos importantes, una explicación de los *benchmarks* usados y para finalizar, la relación del Trabajo de Fin de Grado con los ODS.

CAPÍTULO 2
Estado del Arte

En este capítulo se estudia el estado del arte y el contexto tecnológico en el que se ha llevado a cabo el trabajo.

2.1 Simuladores

Dentro de la Ingeniería de Computadores se puede encontrar una amplia gama de diferentes simuladores, cada uno diseñado con un propósito y que pueden clasificarse en tres grandes categorías: simuladores de propósito general, especializados e industriales.

2.1.1. Simuladores de Propósito General

Este tipo de simuladores juegan un papel crucial en la Arquitectura de Computadores, ya que ofrecen a ingenieros y académicos las herramientas necesarias para modelar y analizar una gran variedad de sistemas complejos. Están pensados para ser versátiles y soportar un amplio rango de arquitecturas para probar diferentes cargas de trabajo. Algunos ejemplos actuales de este tipo de simuladores son:

- **Gem5:** Simulador de código abierto que cuenta con una gran comunidad desarrolladora. Soporta un amplio rango de arquitecturas como son: x86, ARM, SPARC, MIPS y POWER. Es ampliamente configurable y cuenta con simulación detallada de la CPU y los subsistemas de memoria. Esto se detalla en [4].

- **SimpleScalar:** Simulador de código abierto que fue desarrollado entre otros propósitos, para el diseño de los juegos de instrucciones de las CPU. A pesar de haber sido desarrollado a finales de la década de los 90, aún hoy en día cuenta con herramientas interesantes para modelar y analizar el rendimiento de procesadores. Este simulador se detalla en [12].

2.1.2. Simuladores Especializados

Los simuladores especializados cubren la necesidad de experimentar detalladamente con ciertos subsistemas de los procesadores. Este tipo de simulación suele estar especializada en una de las áreas de la CPU, ofreciendo una simulación detallada y compleja de una funcionalidad mientras ofrece una solución más simplista pero funcional del resto de áreas. También se puede clasificar aquí los que simulan de forma detallada una única arquitectura, permitiendo explorarla en profundidad. Ejemplos actuales de este tipo de simuladores son:

- **GPGPU-Sim:** Simulador ampliamente empleado en el diseño y desarrollo de unidades de procesamiento gráfico para computación de propósito general. Se especializa en la ejecución de programas CUDA de la empresa NVIDIA[1]. Su uso es intensivo en el diseño de arquitecturas GPU, computación paralela y optimización de programas GPU. El simulador detalla en [13].
- **McPAT:** Simulador especializado en el cálculo de los costes energéticos de las microarquitecturas de los procesadores. Soporta una variedad de arquitecturas, incluyendo sistemas CPU-GPU y diferentes granularidades de análisis. Este simulador es ampliamente empleado en análisis energético, modelos de consumo y optimización energética. Este simulador fue presentado en el simposio [14].

2.1.3. Simuladores Industriales

La industria tecnológica está liderada por grandes empresas que en su mayoría toman un rol fundamental en el desarrollo y diseño de las nuevas tecnologías. Estas empresas desarrollan sus propios simuladores especializados en simular de forma detallada y precisa las máquinas que fabrican y diseñan estas empresas. Un claro ejemplo en el desarrollo de simuladores propios (*in-house*) es la empresa Intel [2]. Por otro lado la empresa Wind River[3] diseña simuladores industriales pensados para el desarrollo de productos, contando con los siguientes ejemplos:

- **Simics:** Simulador desarrollado por la empresa Wind River. este simulador está pensado para simular sistemas completos, incluyendo no solamente el procesador sino además, periféricos e interfaces de red. Usado para el desarrollo industrial de *hardware* periférico para ordenadores, sus características se detallan en [15].
- **Intel SDE[4]:** Simulador desarrollado por Intel para apoyar la investigación y desarrollo de cambios de arquitecturas y microarquitecturas de la empresa. Cuenta con grandes herramientas de *debug* para programas pensados para sus arquitecturas, además de centrarse en investigación y desarrollo de herramientas para sus sistemas.

[1] `https://www.nvidia.com/es-es/`

[2] `https://www.intel.com/content/www/us/en/products/details/processors.html`

[3] `https://www.windriver.com/company`

[4] `https://www.intel.com/content/www/us/en/developer/articles/tool/software-development-emulator.html`

2.2 Trabajos Relacionados

Numerosas publicaciones científicas y trabajos abordan el empleo de simuladores. Muchos otros abordan el uso de la metodología de análisis Top-Down. No obstante, no se ha planteado nunca un diseño o implementación de una unidad de monitorización de prestaciones para un simulador como es la que se propone en este Trabajo de Fin de Grado.

En lo referente al análisis Top-Down, como expone el documento de Saito *et al.* [5], supone una herramienta muy útil para estructurar análisis de prestaciones. Es por eso que ha sido empleado en un gran número de ocasiones. Por ejemplo, en el trabajo de Yasin [6] se emplea el análisis Top-Down para identificar de forma eficiente cuellos de botella en procesadores fuera de orden mediante el uso jerárquico de contadores hardware. El mismo procedimiento sirve de inspiración para este trabajo donde se programarán contadores hardware y una PMU para posteriormente realizar un análisis Top-Down.

Como expone el mencionado trabajo, el análisis de microarquitecturas de ordenadores se ha vuelto extremadamente complejo. Es por eso que las PMU han ido ganando importancia con el avance de los procesadores. Debido a los motivos expuestos, las PMUs han sido objetivo de numerosos estudios. En el trabajo de Zhang *et al.* [7], se propone una alternativa al uso de las PMU desarrollando una funcionalidad del sistema operativo que permite consultar directamente los contadores *hardware* para optimizar la planificación del procesador. London *et al.* [8] propone el uso de PMU y contadores *hardware* para desarrollar herramientas para el usuario final mediante el uso de librerías. De esta forma los usuarios disponen de un método de contabilizar y acceder a la información del rendimiento de sus procesadores. Esencialmente, el presente Trabajo de Fin de Grado propone crear una herramienta equivalente en un simulador de propósito general para el desarrollo de microarquitecturas de procesadores.

2.3 Trabajos Relacionados en la ETSINF

Dentro de la ETSINF, también se han propuesto ya con anterioridad Trabajos de Fin de Grado o Máster relacionados, como son trabajos con simuladores o trabajos que emplean la metodología Top-Down.

Calero [17] trabaja con Contadores *Hardware* para estudiar el consumo energético de un procesador ARM Thunder X2[5] con microarquitectura Vulcan[6], la misma microarquitectura que se ha tratado de emular en este trabajo.

Un trabajo de la ETSINF relacionado con el análisis Top-Down es el de Navarro [18], donde realiza este análisis para desarrollar algoritmos de políticas de asignación de aplicaciones a núcleos. Navarro realiza un desarrollo completo de este tipo de análisis focalizado en un procesador Intel. En este trabajo se ha optado por modificar y adaptar el análisis a las necesidades del mismo.

[5] `https://fuse.wikichip.org/news/tag/thunderx2/`

[6] `https://en.wikichip.org/wiki/cavium/microarchitectures/vulcan`

Pons [20], Carmona [21], Navarro y Calero emplean todos ellos las SPEC CPU como herramienta experimental, ratificando la decisión que se ha tomado en este trabajo de emplearlas también como herramienta experimental para validar la solución.

Por último, en la ETSINF también se cuenta con amplia experiencia trabajando con simuladores. Esto lo demuestran trabajos como los de Catalá [23] y Vivas [22], los cuales simulan el subsistema de memoria de una CPU para estudiar la jerarquía de *cache*. Avargues [19] desarrolla para el simulador Gem5 un controlador de memoria NVRAM, implementando un diseño propio de un controlador y probando su funcionamiento en el simulador. En este Trabajo de Fin de Grado también se desarrollará un nuevo componente para el simulador Gem5 y se probará su correcta implementación.

CAPÍTULO 3
Análisis Top-Down

El análisis Top-Down, propuesto en [6], consiste en una estrategia de identificación de cuellos de botella en la microarquitectura de los procesadores. La propuesta de Intel consiste en monitorizar mediante contadores y las PMU las distintas etapas del pipeline de ejecución, siendo capaz de identificar y distinguir las partes que merman las prestaciones del sistema. Este análisis busca organizar de forma jerárquica las diferentes situaciones que afectan a las prestaciones.

En este capítulo se va a presentar y detallar el análisis Top-Down, ya que ha sido la estrategia seleccionada para estudiar la correcta implementación de la PMU propuesta. Para ello, primero se introducen conceptos básicos del *pipeline* para facilitar la comprensión del análisis Top-Down. Seguidamente se explica el análisis Top-Down y las diferentes categorías y subcategorías que lo componen.

3.1 *Pipeline* del Procesador

La figura 3.1 muestra de forma simplificada el *pipeline* de un núcleo de procesador Intel con microarquitectura Skylake. Las instrucciones entran en el pipeline desde la cache L1 de instrucciones en la etapa de búsqueda o *Fetch* (*Instruction Fetch & Predecode*) y salen cuando se retiran del *Reorder buffer* en la etapa de confirmación o *Commit*. En las primeras etapas del *pipeline*, denominadas *Frontend*, las instrucciones fluyen en orden de entrada (también conocido como orden de programa). El *frontend* finaliza en la etapa de *Dispatch*, donde las instrucciones reservan los recursos necesarios para su ejecución en el *Backend* (compuesto en la figura de la *Execution Engine* y el *Memory Subsystem*). Una vez en el *backend*, las instrucciones se emiten para su ejecución en los operadores (*Execution Units*) en la etapa conocida como *Issue*. En este punto, el *pipeline* se subidivide entre los diversos operadores (de enteros, de coma flotante, de carga, almacenamiento, etc.). Las instrucciones se emiten fuera del orden del programa una vez que están listas para operar, por lo que se ejecutan y escriben sus resultados de forma desordenada. Tras su ejecución, las instrucciones esperan en el *reorder buffer* hasta su retiro en *commit*, donde se garantiza que salgan del *pipeline* en orden de programa.

Las prestaciones se pueden caracterizar en cualquier etapa, siempre que todas las instrucciones ejecutadas atraviesen ese punto. Por ejemplo, las prestaciones se podrían monitorizar en *dispatch*, *issue*, o *commit*, ya que, a excepción de las

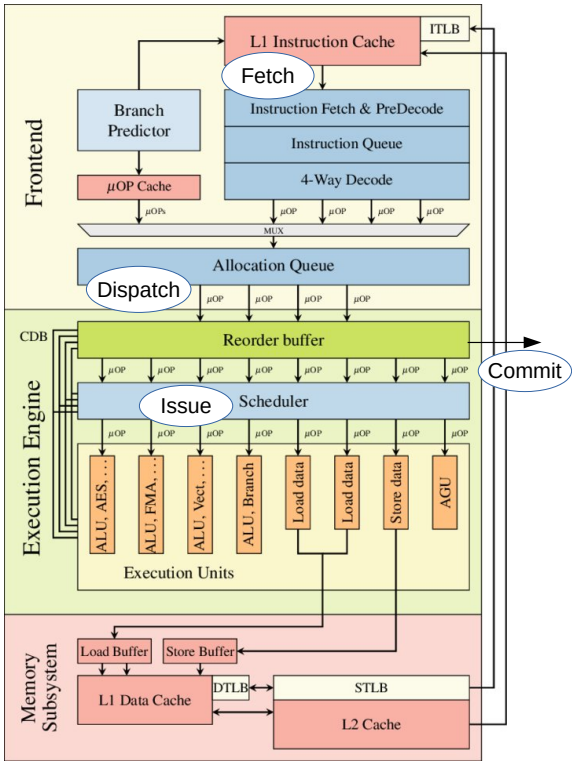

Figura 3.1: Diagrama del pipeline de un núcleo del procesador Intel Skylake. Figura original en [24].

instrucciones que se cancelan por fallos de especulación, en general, todas las instrucciones atraviesan esas etapas.

Cada fabricante de procesadores utiliza una etapa diferente para medir las prestaciones con la PMU. Por ejemplo, los procesadores de Intel monitorizan las prestaciones en *issue* [6] mientras que IBM en sus procesadores POWER lo hace en *commit*. Finalmente, la PMU de los procesadores ARM se enfoca a la medida de prestaciones en *dispatch*.

El análisis Top-Down fue propuesto inicialmente por Intel para sus procesadores y PMU, por lo que las categorías y subcategorías de la propuesta de Intel se encuentran claramente influenciadas por este hecho. Como se verá más adelante, el lugar más adecuado para implementar la PMU en Gem5 con el objetivo de medir las prestaciones es la etapa con funciones equivalentes a *dispatch*, lo que deberá tenerse en cuenta para adaptar las categorías del análisis en este trabajo.

3.2 Jerarquía Top-Down

El objetivo principal de este análisis es la ordenación en categorías de los cuellos de botella de un procesador. Este análisis emplea diferentes granularidades a la hora de clasificar las causas de estos fallos de rendimiento y árboles de decisiones para asignar estos fallos a las distintas categorías como se indica en la figura 3.2.

Para llevar a cabo este análisis se ha de medir de forma eficiente el tiempo que emplea la CPU en cada una de las distintas partes de su microarquitectura,

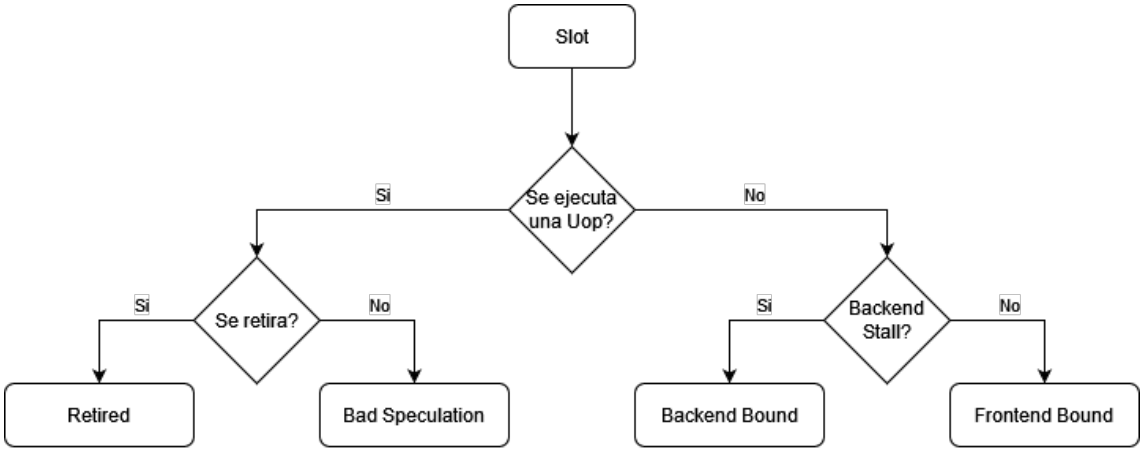

Figura 3.2: Árbol de decisión del análisis Top-Down.

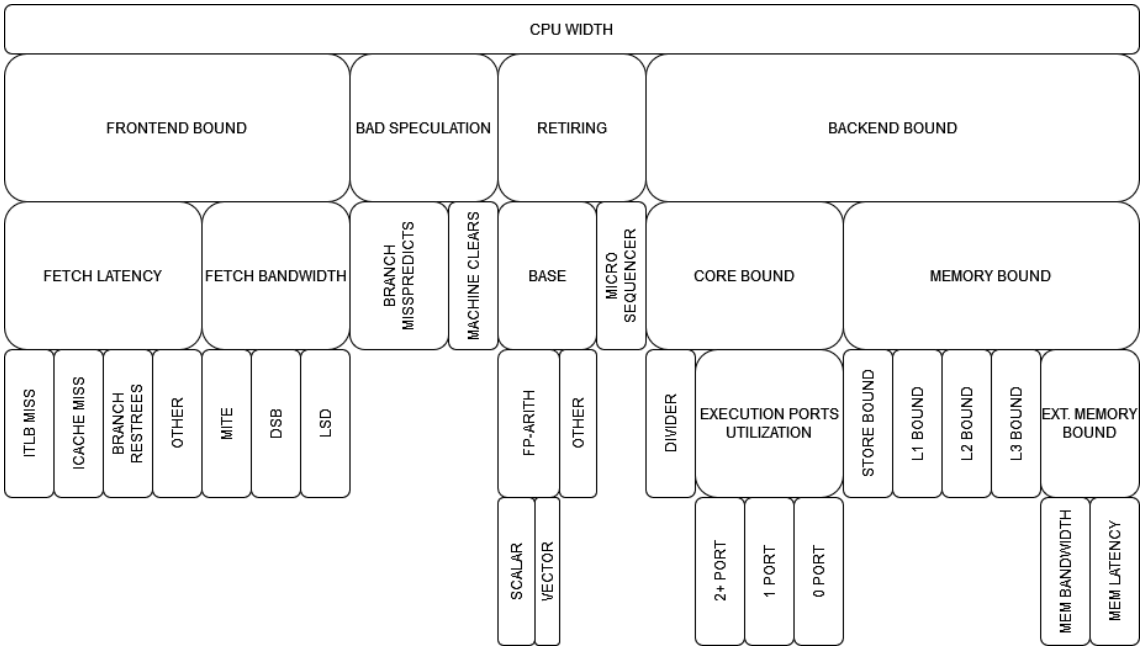

Figura 3.3: Jerarquía Top-Down, figura original en [6]

y en función del nivel de detalle, la información obtenida sobre las razones de pérdidas de prestaciones será más precisa.

En este contexto, el tiempo de ejecución de una aplicación se contabiliza en ciclos. Cada uno de esos ciclos se subdivide en *slots*. Así, un procesador superescalar de 4 vías dispone de 4 *slots* por ciclo, ya que es capaz de procesar 4 instrucciones cada ciclo. Sin embargo, un *slot* puede dedicarse o bien al procesamiento de una instrucción o no. En el segundo caso, se considera que el *slot* desperdiciado contribuye al conteo de ciclos de parada o *stalls*.

El primer nivel de categorías, es decir, el de mayor granularidad del análisis Top-Down proporciona mucha información sobre el impacto de las partes principales del procesador a las prestaciones. A continuación, se describe cada una de las categorías del primer nivel así como las subcategorías que la componen. Estas categorías y subcategorías se muestran en la figura 3.3.

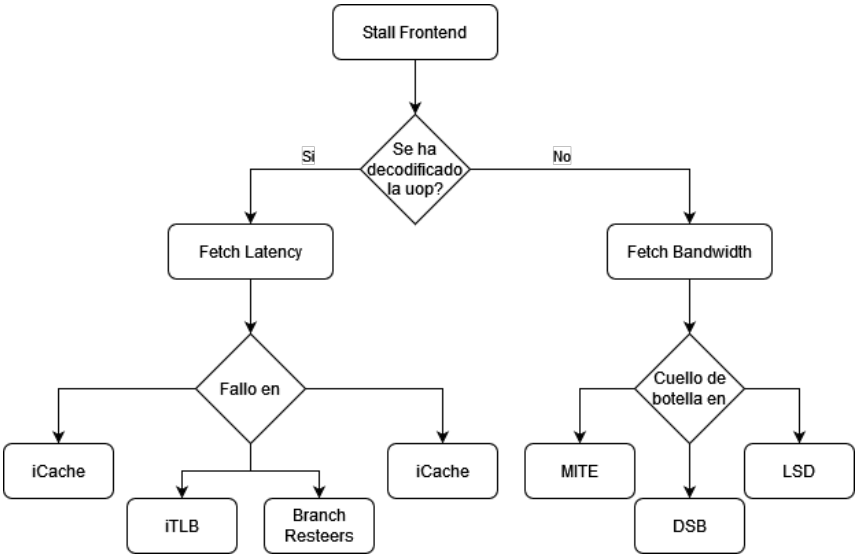

Figura 3.4: Árbol de decisiones de la categoría *Frontend Bound*.

3.2.1. *Frontend Bound*

El *frontend* de la CPU es el encargado de hacer llegar las instrucciones al *backend*. Cuando el *frontend* no es capaz de abastecer al *backend* en la medida que el *backend* puede procesar instrucciones se considera que el *frontend* limita el rendimiento. El análisis Top-Down propone esta categoría para cuantificar en qué medida el *frontend* no es capaz de abastecer al *backend*.

El *frontend* se corresponde con las primeras etapas del *pipeline* de los procesadores, en concreto desde *instruction fetch* hasta *instruction dispatch*. Estas etapas del *pipeline* se encargan de buscar la próxima instrucción a ejecutar, decodificarla y enviarla a ejecución, parte que es gestionada por el *backend*.

Tal como se muestra en la figura 3.4, Intel identifica dentro del *frontend* dos grandes subcategorías, *Fetch Latency* y *Fetch Bandwidth*. *Fetch latency* referencia la fracción de *slots* en el que la CPU se encuentra en parada debido a problemas de latencia en las etapas del *frontend*. *Fetch bandwith* hace referencia a la fracción de *slots* en la cual la CPU se encuentra en parada debido a problemas en el ancho de banda de la etapa *frontend*.

Estas subcategorías se encuentran a su vez divididas en más subcategorías que profundizan en el análisis de las ineficacias del *pipeline*. En el caso de *fetch latency*, esta detalla la categorización de los *stalls* con tres subcategorías adicionales:

- ***iTLB Miss:*** Esta subcategoría es representante de la fracción de *slots* en las cuales la CPU se encuentra en *stall* debido a un fallo en el componente *Translation Lookside Buffer* (TLB), el cual es el encargado de traducir direcciones virtuales a direcciones físicas.

- ***iCache Miss:*** Esta subcategoría es representante de la fracción de *slots* en las cuales la CPU se encuentra en *stall* debido a un fallos en la caché de instrucciones.

- ***Branch Resteers:*** Esta subcategoría es representante de la fracción de *slots* en las cuales la CPU se encuentra en *stall* por los *branch resteers*, los cuales se encargan de la recuperación del camino correcto tras un fallo de predictor de salto.

A su vez, *fetch bandwidth* referencia la fracción de *slots* que la CPU se encuentra en *stall* debido a problemas en el ancho de banda del subsistema de memoria referente a la etapa *frontend*, debido a que el *pipeline* no puede suministrar suficientes instrucciones al ritmo que pueden ser procesadas. Esta categoría se divide en tres categorías de forma similar a lo antes expuesto:

- **MITE:** Esta subcategoría es representante de la fracción de *slots* en los cuales la CPU ha sido limitada por debajo de sus máximas prestaciones por culpa de MITE, que se encarga de traducir instrucciones CISC x86 a *Uops*.
- **DSB:** Esta subcategoría es representante de la fracción de *slots* en los cuales la CPU ha sido limitada por debajo de sus máximas prestaciones por culpa de DSB (*decoded Uop cache*).
- **LSD:** Esta subcategoría es representante de la fracción de *slots* en los cuales la CPU ha sido limitada por debajo de sus máximas prestaciones por culpa de la unidad LSD (*Loop Stream Detector*).

3.2.2. *Bad Speculation*

La categoría *bad speculation* referencia a la influencia en las prestaciones de las instrucciones incorrectamente especuladas por fallos de la predicción de saltos los cuales causan que se lancen instrucciones que nunca finalizan con éxito debido a que son canceladas. Esta categoría se encuentra a su vez dividida en subcategorías (ver figura 3.5) de la siguiente forma:

- ***Branch Mispredicts:*** Esta subcategoría es representante de la fracción de *slots* en los cuales la CPU ha gastado capacidad de su *pipeline* ejecutando instruccionesbuscadas por un fallo de predicción de saltos.

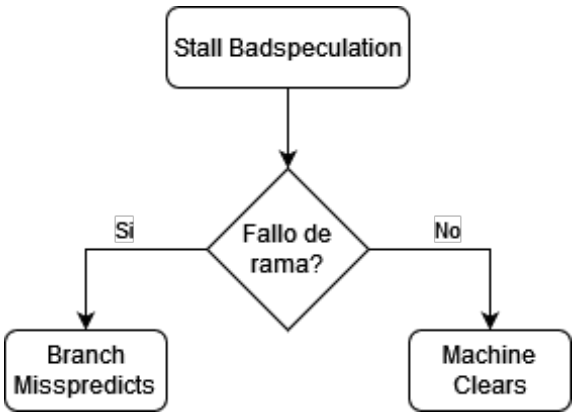

Figura 3.5: Árbol de decisiones de la categoría *Bad Speculation*.

- ***Machine Clears:*** Esta subcategoría es representante de la fracción de *slots* malgastados debido a la limpieza del *pipeline* que se realiza debido a un fallo de especulación, lel cual requiere vaciar parte de el *pipeline* para continuar su ejecución.

3.2.3. *Retiring*

Retiring es la categoría que representa la medición de la fracción de *slots* en los cuales se han ejecutado y retirado del *pipeline* correctamente las instrucciones, es decir, la fracción de *slots* en los que la CPU ha realizado trabajo útil. Una aplicación como objetivo fundamental busca maximizar el IPC, instrucciones retiradas por ciclo, optimizando el uso del pipeline y sacando el máximo rendimiento del procesador. Aumentar el IPC supone aumentar la categoría retiring, al igual que un alto porcentaje de retiring es símil de un uso eficiente de la CPU. Esta categoría, al igual que las demás se encuentra dividida en subcategorías (figura 3.6) de la siguiente forma:

- **BASE:** Esta subcategoría representa la fracción de *slots* que emplea la CPU retirando microoperaciones (*Uops*) ejecutadas. Esta subcategoría representa casi en su totalidad las instrucciones ejecutadas y es por eso que se divide en otras subcategorías para diferenciar los tipos de instrucciones que han sido retiradas. No obstante, en el paper de Intel [5] se hace especial énfasis en la importancia de las operaciones aritméticas de coma flotante proponiendo

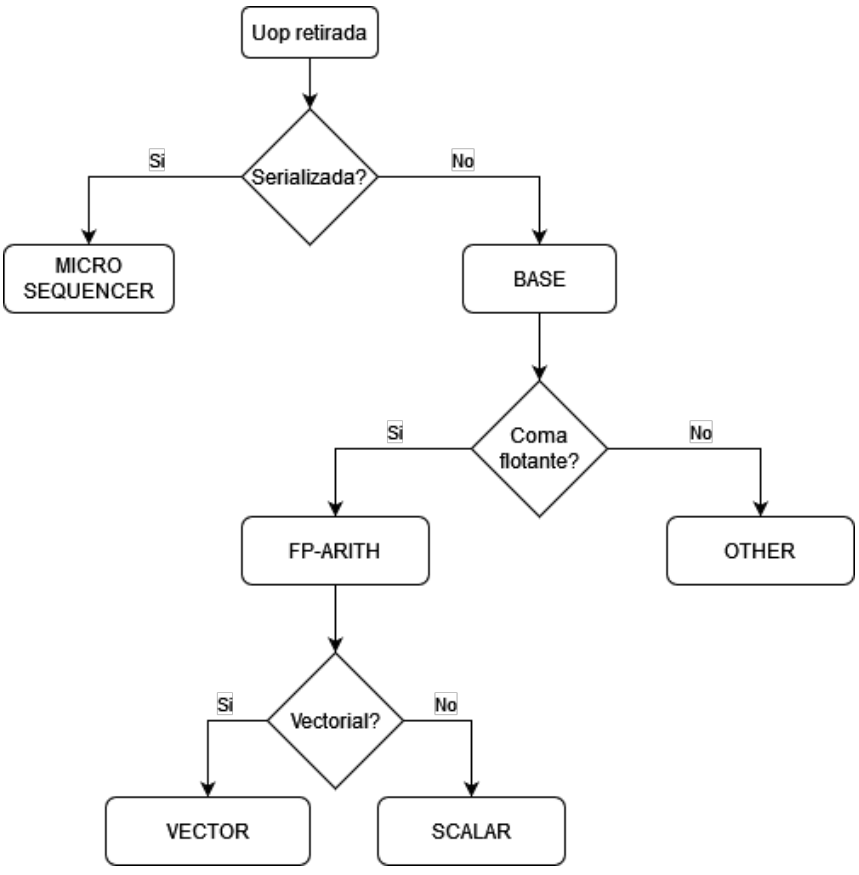

Figura 3.6: Árbol de decisión de la categoría *Retiring*.

como principal la subcategoría *FP-Arith*, y dejando el resto de subcategorías a decidir para cada aplicación. *FP-Arith* representa la fracción de *slots* que la CPU emplea ejecutando instrucciones aritméticas de coma flotante.

- ***Microcode Sequencer:*** Esta subcategoría referencia a la fracción de *slots* que la CPU emplea en gestionar la ejecución y retirada de algunas instrucciones que, debido a su complejidad, han de ser divididas en *Uops* y ser ejecutadas secuencialmente.

3.2.4. *Backend Bound*

El *backend* de la CPU es la parte encargada de ejecutar las instrucciones o *Uops* que llegan desde el *frontend*. Cuando el *backend* no es capaz de aceptar más instrucciones provenientes del *frontend* se dice que el *backend* limita el rendimiento. El análisis Top-Down propone esta categoría para cuantificar en qué medida el *backend* no es capaz de soportar el ritmo en el que el *frontend* le abastece de instrucciones.

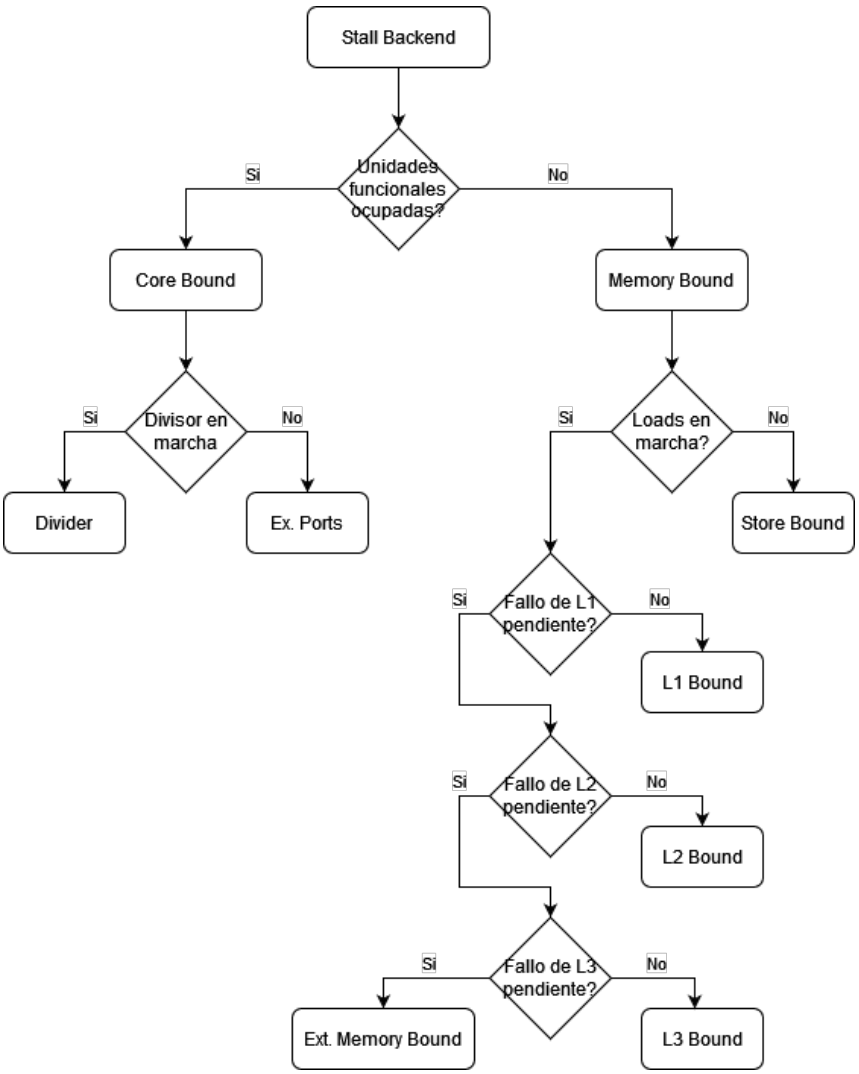

Figura 3.7: Árbol de decisiones de la categoría *Backend Bound*.

El *backend* se corresponde con las etapas intermedias y finales del *pipeline* e incluye el subsistema de memoria. En otras palabras, esta etapa engloba la ejecución de una instrucción, desde que es lanzada, hasta que es retirada.

La categoría *backend bound* representa los *slots* en los cuales la CPU se encuentra en *stall* debido a que el *pipeline* no puede entregar *Uops* al *backend* debido a la falta de recursos necesarios para poder lanzar a ejecución estas instrucciones. De la etapa *backend* se encuentra al mando el planificador de instrucciones fuera de orden y que envía las instrucciones listas para ejecutarse a los distintos operadores siguiendo el algoritmo de Tomasulo [3].

Intel propone dos subcategorías principales (ver figura 3.7): *core bound* y *memory bound*. *Core bound* hace referencia a los problemas con el *backend* referentes al núcleo del procesador, más en concreto a todo lo que no tiene que ver con el subsistema de memoria. Esta subcategoría engloba todos los *slots* en los que la CPU se encuentra en *stall* debido a problemas en el núcleo de la CPU. Por otro lado, *memory bound* hace referencia a los *stall slots* causados por el subsistema de memoria. Es decir, esta subcategoría engloba todos los *slots* en los que la CPU se encuentra en *stall* debido a problemas en el acceso de lecturas y escrituras.

Estas subcategorías se dividen a su vez en otras subcategorías para profundizar en el estudio de las ineficacias del *pipeline*. Para *core bound* se detallan las subcategorías siguientes:

- ***Divider:*** Esta subcategoría es representante de la fracción de *slots* en los que la CPU se encuentra en *stall* debido a que la unidad de división se encuentra ocupada, esta unidad por lo general es la que más tiempo de cálculo gasta, por eso la importancia de darle una categoría a ella sola.
- ***Execution Ports Utilization:*** Esta subcategoría es representante de la fracción de *slots* en los que la CPU se encuentra en *stall* debido a que los operadores aritméticos no aceptan más instrucciones.

En el caso de la subcategoría *memory bound*, se detallan las siguientes subcategorías:

- ***Store Bound:*** Fracción de *slots* que emplea la CPU en *stall* debido a las escrituras (*stores*).
- ***L1 Bound:*** Fracción de *slots* que emplea la CPU en *stall* debido a las lecturas que obtienen sus datos en la memoria *cache L1*.
- ***L2 Bound:***Fracción de *slots* que emplea la CPU en *stall* debido a las lecturas que obtienen sus datos en la memoria *cache L2*.
- ***L3 Bound:***Fracción de *slots* que emplea la CPU en *stall* debido a las lecturas que obtienen sus datos en la memoria *cache L3*.
- ***Ext. Memory Bound:*** Fracción de *slots* que emplea la CPU en *stall* debido a la latencia generada por las solicitudes del controlador de memoria.

CAPÍTULO 4
Diseño de la PMU

En este capítulo se detallan las decisiones y características del diseño propuesto de la PMU para este Trabajo de Fin de Grado. Primero se explica la decisión de implementar la PMU en el modelo de ejecución fuera de orden. Tras esto, se explica el *pipeline* del simulador en Gem5 y la integración de la PMU en este. Finalmente, se detalla la *performance stack* a implementar dentro de la PMU.

4.1 Ejecución Fuera de Orden frente a Secuencial

En este trabajo se han estudiado las diferentes alternativas a la hora de desarrollar la unidad de monitorización de prestaciones. La primera decisión a tomar se presenta a la hora de seleccionar para que modelo de CPU conviene desarrollar la unidad.

Para este trabajo se ha decidido trabajar sobre la simulación de Gem5 de procesadores con soporte a la ejecución fuera de orden en su modelo de CPU O3CPU. Se ha seleccionado la ejecución fuera de orden frente a la ejecución secuencial por los siguientes motivos:

- **Complejidad de ejecución:** Las CPU fuera de orden y su asignación de recursos es dependiente de la utilización del *pipeline* del procesador, este no es el caso de la ejecución secuencial donde la utilización de recursos es totalmente dependiente del orden de las instrucciones que componen la aplicación. Como se puede observar en la figura 4.1 la ejecución fuera de orden supone un *pipeline* mucho más complejo que la ejecución secuencial. Esto implica que la optimización del *pipeline* del procesador supone un mayor impacto en las prestaciones en la ejecución fuera de orden que en la ejecución secuencial.
- **Impacto en el rendimiento:** Las CPU fuera de orden están diseñadas para explotar al máximo el paralelismo a nivel de instrucciones. El estudio de ciclos de parada y su causas nos permiten minimizar su aparición manteniendo así al máximo posible la utilización de los operadores. En el caso de la ejecución secuencial, reducir el número de ciclos de parada no supone un aumento tan grande del rendimiento de la máquina ya que no es posible aprovechar todas las oportunidades de uso de los operadores siguiendo estrictamente el orden de programa.

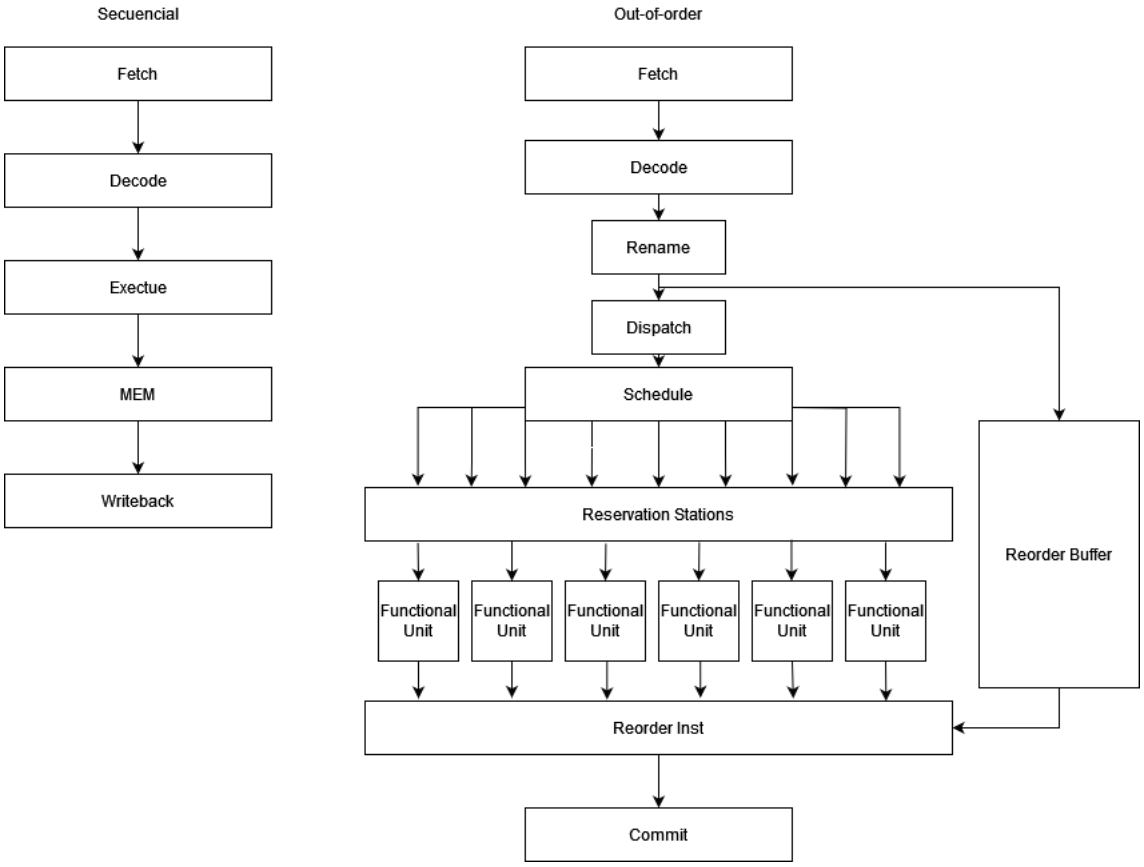

Figura 4.1: Pipelines con soporte a ejecución secuencial y a ejecución fuera de orden.

- **Utilización de recursos:** Los procesadores fuera de orden cuentan con muchas más estructuras para el control de recursos, como pueden ser: el *reorder buffer*, la IQ, la LSQ, etc. Dimensionar estas estructuras adecuadamente supone un problema esencial en el diseño de este tipo de procesadores.

- **Impacto en aplicaciones modernas:** Procesamiento paralelo, procesamiento multimedia y computación científica son varias de las áreas donde la ejecución fuera de orden domina el mercado frente a la ejecución secuencial. Estas áreas son fundamentales en el desarrollo del trabajo al estar validado sobre aplicaciones científicas. Consideramos que el impacto de esta herramienta es mucho mayor en estos casos.

Expuestos estos hechos, queda clara la motivación de desarrollar esta unidad para una microarquitectura con soporte a la ejecución fuera de orden.

4.2 *Pipeline* del Simulador

Con el objetivo de realizar un diseño e implementación adecuados de la PMU para el simulador Gem5, se ha estudiado previamente el funcionamiento de este simulador y su *pipeline* fuera de orden. La documentación del simulador indica que las etapas del *pipeline* implementadas son las siguientes:

- ***Fetch***: Busca instrucciones a ejecutar cada ciclo, selecciona el hilo del cual se buscan las instrucciones en función de una política configurable. En esta etapa también se implementa la predicción de saltos.

- ***Decode***: Decodifica las instrucciones que se van a ejecutar cada ciclo y realiza resoluciones tempranas de saltos.

- ***Rename***: Esta etapa es fundamental para la ejecución fuera de orden. Renombra los operandos de las instrucciones para evitar dependencias *write after write* (WAW) y *write after read* (WAR), encadena las instrucciones dependientes y proporciona los recursos necesarios para la ejecución fuera de orden. Además, esta etapa se encarga de gestionar los ciclos de parada. En otras palabras, esta etapa incluye las acciones necesarias para realizar el *dispatch* en un procesador típico.

- ***Issue/Execute/Writeback***: En estas etapas se lanzan, ejecutan y almacenan los resultados de las instrucciones.

- ***Commit***: Retira las instrucciones cada ciclo en orden de programa (orden en el que se almacenan en el *reorder buffer*). Cada instrucción retirada es una instrucción que completa su ejecución correctamente. También gestiona fallos de especulación (por ejemplo, fallos en la predicción de saltos).

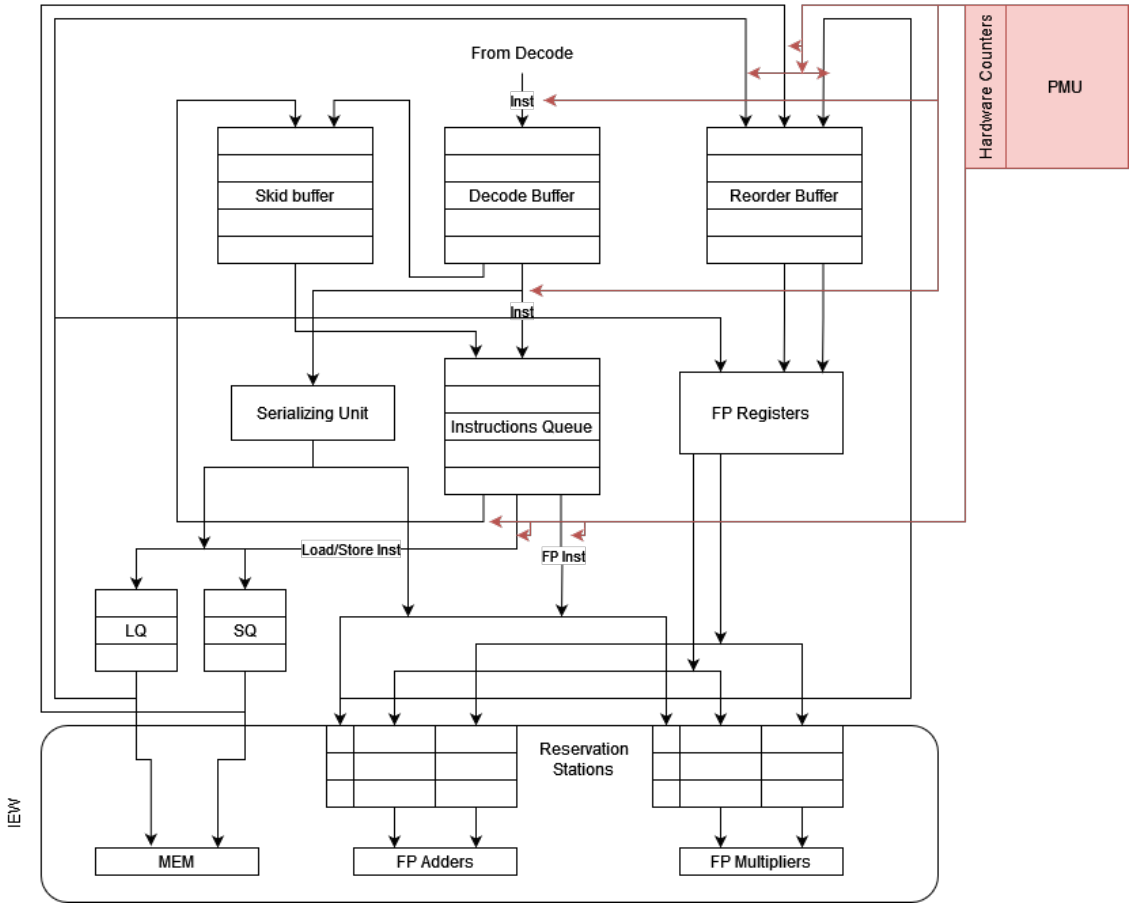

Figura 4.2: Pipeline OOO de Gem5 e integración de la PMU.

Por la etapa de *rename* pasan todas las instrucciones y además es la etapa donde se introducen los ciclos de parada. Por estas razones, se ha decidido implementar en esta etapa una PMU que permita medir a qué se dedica cada uno de los ciclos de ejecución y permita construir una *performance stack* similar a la construida en el análisis Top-Down. La figura 4.2 muestra el *pipeline* del simulador integrada con la propuesta de PMU desarrollada en este trabajo.

Nótese que al contrario que en un procesador real segmentado, donde en cada ciclo actúan las diferentes etapas de forma paralela, en la simulación de Gem5 los métodos que modelan las distintas etapas se ejecutan de forma secuencial, tal como se indica en el listado siguiente:

```
1  CPU::tick()
2  {
3  ...
4  ...
5  //Tick each of the stages
6      fetch.tick();
7      decode.tick();
8      rename.tick();
9      iew.tick();
10     commit.tick();
11 ...
12 }
```

Esta característica del simulador dificulta la construcción de la *performance stack*, ya que en cada etapa tenemos la información de las etapas anteriores pero no disponemos de la información de las etapas posteriores a la que se está simulando. Esto impide caracterizar detalladamente las pérdidas de prestaciones por instrucciones que han llegado a las etapas *issue/execution/writeback* pero han sido canceladas por fallos de especulación (como fallos del predictor de saltos).

4.3 Performance Stack

Mediante la PMU implementada, se pretende, aplicando el análisis Top-Down, caracterizar y catalogar adecuadamente los ciclos de ejecución incluyendo los ciclos de parada. Para cumplir este objetivo, la metodología Top-Down propone la creación de una *performance stack* la cual es una pila donde se clasifican en categorías los ciclos de ejecución de una aplicación.

Debido a que la PMU propuesta se centra en la etapa *rename*, y a la limitación de la simulación secuencial mencionada previamente, no es posible utilizar las mismas categorías y subcategorías de los procesadores Intel, cuya PMU mide las prestaciones para el análisis en *issue* y se implementa en un sistema real donde las etapas actúan en paralelo. Como consecuencia se han cambiado y adaptado algunas categorías de acuerdo con las características y limitaciones de nuestra plataforma. En primer lugar, se ha sustituido la categoría *retiring* por la categoría *dispatch*. A diferencia de *retiring*, *dispatch* cuantifica todas las instrucciones enviadas al *backend* para su ejecución y no sólo las retiradas. En segundo lugar, la cate-

goría *bad speculation* se ha reemplazado por la categoría *squashed*, que contabiliza las instrucciones que se cancelan en *rename*.

Teniendo esto en cuenta y las diversas razones por las que un *slot* puede ser desperdiciado en la etapa de *rename* de gem5, en el presente trabajo se han seleccionado las siguientes categorías:

- ***Frontend:*** Esta categoría reúne el número de *slots* en los cuales no se ha renombrado una instrucción debido a falta de alimentación de instrucciones por parte del *frontend*.
- ***Dispatch:*** Esta categoría reúne todos los *slots* en los cuales se han renombrado con éxito instrucciones.
- ***Squashed***: Esta categoría cuantifica cuantos *slots* han sido empleados por instrucciones que se cancelan debido a fallos de especulación. Se engloban tanto instrucciones canceladas por fallo de predicción de salto como otros motivos como interrupciones o excepciones.
- ***Serializing:*** Gem5 en su funcionamiento emplea un tipo de instrucciones que son del tipo *serializable*. La ejecución de estas instrucciones implica un vaciado del *pipeline*. Esta categoría reúne el número de *slots* empleados soportar esta serialización.
- ***Physical***: Esta categoría cuantifica la cantidad de *slots* que han sido desperdiciados al no haber podido renombrar una instrucción debido a que no había registro físicos disponibles.
- **ROB:** Esta categoría reúne el número de *slots* en los cuales no se ha podido renombrar una instrucción debido a falta de espacio en el *reorder buffer* (ROB).
- **IQ:** Esta categoría representa el número de *slots* que se han perdido debido a falta de espacio en la cola de instrucciones (IQ).
- **LQ:** Esta categoría reúne el número de *slots* en los cuales la cola de *loads* (LQ) se encontraba llena, impidiendo el procesamiento de una instrucción de carga.
- **SQ:** Esta categoría reúne el número de *slots* en los cuales la cola de *stores* (SQ) se encontraba llena, impidiendo el procesamiento de una instrucción de almacenamiento.

4.3.1. Priorización de la Asignación de Categorías

La PMU efectúa sus mediciones cada ciclo. Cada ciclo el *pipeline* cuenta con un ancho de instrucciones n que pueden pasar por *rename*. Es decir, cada ciclo cuenta con n *slots* y por tanto la suma de los *slots* asignados a cada categoría no puede superar un valor de n por ciclo. Esto puede resultar un problema, ya que muchas condiciones para asignar un *slot* como ciclo de parada no son excluyentes (pueden solaparse). Por ejemplo, una instrucción puede no ser renombrada

y cuantificarse como ciclo de parada en un ciclo en el cual no hay espacio en el ROB y tampoco hay espacio en la IQ. O bien no hay espacio en la IQ y tampoco se ha suministrado una instrucción desde el *frontend*, etc. En este tipo de situaciones, cuando hay más de una posible categoría donde asignar el *slot* es necesario definir una prioridad entre ellas.

La prioridad de cada categoría queda parcialmente definida por la estructura del código del simulador y el momento donde se establecen los contadores *Hardware*. En Gem5, dentro del método de la clase donde se simula la etapa de renombrado sólo pueden aparecer solapes entre las tres categorías ROB, IQ y *frontend*. Para resolver esta problemática y mantener la suma de todas las categorías igual al ancho del *pipeline* se ha empleado un sistema de prioridades en el momento de asignar un *slot* a una de estas categorías en función de su relevancia. La prioridad a la hora de asignar una categoría a un ciclo de parada es la siguiente:

1. **ROB:** El *reorder buffer* es un componente *hardware* que almacena las instrucciones en orden de programa para preservarlo en el momento de hacer *commit* de las mismas. Es crucial para el correcto funcionamiento de una aplicación en ejecución fuera de orden. Sin esta unidad se pierde el orden, no se pueden retirar instrucciones y por tanto no se puede progresar en la ejecución. Es por este motivo por el que priorizamos al ROB frente a cualquier otra casuística debido a su importancia en el *pipeline*.

2. **IQ:** La cola de instrucciones almacena instrucciones para ser emitidas en los operadores. Esta cola es importante para mantener ocupados al máximo estos operadores. No obstante, el papel del ROB sobresale por su indispensabilidad, por lo que la categoría IQ queda segunda en el orden de prioridad.

3. ***Frontend:*** El *frontend* lo situamos en última posición de prioridades ya que, aunque se diese el caso de que hubiese instrucciones para ser renombradas, sin espacio en el ROB, o espacio en la IQ estas no podrían evolucionar y pasar de etapa. Debido a esto se le asigna la prioridad más baja.

El orden de prioridades se aplica siguiendo el algoritmo siguiente:

```
1 if (calcFreeROBEntries(tid) <= 0 && Inst_exec == 0) {
2             ROB_inst = ROB_inst + renameWidth;
3         } else if (calcFreeIQEntries(tid) <= 0 && Inst_exec ==
              0) {
4             IQ_inst = IQ_inst + renameWidth;
5         } else if (insts_available_rename == 0) {
6             Frontend_inst = Frontend_inst + renameWidth;
7         }
```

En el caso de producirse un ciclo de parada se comprueba si ha sido por falta de espacio en el ROB. De ser así se asigna al ROB y se obvian el resto de comprobaciones. De haber espacio en el ROB se comprueba la IQ y se sigue un razonamiento análogo. La última de las comprobaciones es el *frontend*. Este orden de prioridades nos permite mantener una consistencia en el análisis donde se categorizan todos los *slots* sin contabilizar *slots* de más.

4.3.2. Niveles de Categorías

Para la elaboración de este trabajo se ha planteado la implementación de una *performance stack* con 2 niveles de granularidad. El segundo nivel, más inferior, desgrana el análisis en las categorías mencionadas anteriormente y que requieren de un conocimiento más profundo de la microarquitectura. En contraposición, el primer nivel proporciona una visión general del rendimiento de la aplicación. Es por ello que solamente cuenta con tres categorías que agrupan al resto:

- ***Frontend:*** Esta categoría cuantifica la cantidad de *slots* donde no se ha renombrado ninguna instrucción debido a que el *frontend* no ha suministrado ninguna instrucción o suministra instrucciones que se deben cancelar. Esta categoría reúne las subcategorías siguientes: *frontend* y *squashed*.
- ***Backend:*** Esta categoría cuantifica la cantidad de *slots* desperdiciados debido a que el *backend* no contaba con los recursos físicos necesarios para llevar a cabo la labor. Esta categoría reúne las subcategorías siguientes: *Physical*, ROB, IQ, LQ y SQ.
- ***Dispatched:*** Esta categoría contabiliza la cantidad de *slots* en los cuales se han renombrado instrucciones y se han lanzado a ejecución.

Nótese que la categoría *Serializing* no se ha agrupado en el primer nivel porque se responde a instrucciones que aparecen inusualmente (entre el 1 % y 2 % del total de instrucciones) en las simulaciones realizadas.

CAPÍTULO 5
Implementación

En este capítulo se detalla el proceso de desarrollo e implementación de la PMU para procesadores con ejecución fuera de orden con soporte monohilo y multihilo, los contadores de prestaciones, la inclusión de un sistema de medición de estadísticas y el soporte a la ejecución simulada de las *suites* de *benchmarks* SPEC CPU.

5.1 Creación del Entorno de Depuración

Como indica Josepson en [11], el proceso de depuración es altamente complejo, y requiere de un gran esfuerzo por parte del programador. A pesar de que este trabajo no es software pensado para un usuario final, se han llevado a cabo acciones continuas para comprobar que el código funciona correctamente según se ha programado. Por lo tanto, el primer paso a seguir para llevar a cabo la implementación de la unidad propuesta es configurar un entorno de desarrollo que permita estudiar e identificar fallos de forma eficiente.

Por optimización del funcionamiento de Gem5, el cual se encuentra implementado en C++, no se recomienda usar la salida estándar std::out, sino que se proporciona una función llamada DPRINTF(), la cual toma como argumentos un *flag* de depuración y una cadena de carácteres. Esta función responde a la línea de comandos a través de la opción –debug-flag, la cual habilita la impresión por la salida estándar de las funciones DPRINTF asociadas a un *flag* en concreto.

El trabajo se desarrolla sobre la versión optimizada de Gem5 "gem5.opt" la cual cuenta con una serie de *flags* de depuración que dan información del funcionamiento del simulador y sus etapas. Algunos ejemplos de estas *flags* son: CPU, *Fetch*, *Rename*, o *Execute*. Las *flags* disponibles en "gem5.opt" son mucho más limitadas que en la versión de depuración de Gem5 ("gem5.debug"). Para el desarrollo y depuración de la PMU definiremos nuestras propias *flags* que nos permitan analizar el flujo de la ejecución e identificar posibles fallos.

La instalación del simulador se detalla en 6.2 dentro del capítulo Entorno Experimental. Una vez instalado el simulador como se indica, el primer paso es declarar nuestra *flag* personalizada en un fichero SConscript que se encuentra en el mismo directorio que los ficheros de configuración. Estos ficheros se sitúan dentro del directorio gem5/src/cpu/o3 para el caso del modelo de CPU con soporte

a ejecución fuera de orden. En este directorio ya se encuentra un fichero SConscript con las *flags* y configuraciones para este modelo, el cual aprovecharemos para añadir nuestra nueva *flag* de la siguiente forma:

Nos situamos en nuestro directorio de trabajo y desde ahí accedemos al directorio.

```
cd ~/tfg/gem5/src/cpu/o3
```

Editamos el documento con el editor de texto a nuestra preferencia, para este trabajo el editor de texto seleccionado es Vim[1].

```
vim SConscript
```

Añadimos nuestra *flag* a las ya existentes mediante la inclusión de la siguiente línea de código:

```
DebugFlag('Stalls')
```

En el directorio /tfg/gem5/build/X86/debug se crea automáticamente un fichero llamado Stalls.hh con la clase donde hemos de definir el funcionamiento de nuestra *flag*. La clase queda de la siguiente forma:

```
#ifndef __DEBUG_Rename_HH__
#define __DEBUG_Rename_HH__

#include "base/compiler.hh" // For namespace deprecation
#include "base/debug.hh"

namespace gem5
{

namespace debug
{

namespace unions
{

inline union Stalls
{
    ~Stalls() {}
    SimpleFlag Stalls = {
        "Stalls", "", false
    };

} Stalls;
} // namespace unions

inline constexpr const auto& Stalls = ::gem5::debug::unions::
    Stalls.Stalls;
```

[1] https://www.vim.org/

```
27
28 } // namespace debug
29 } // namespace gem5
30 #endif // __DEBUG_Stalls_HH__
```

Como se muestra en el listado, hemos definido la *flag* de un modo muy básico, lo que permite asignar funciones DPRINTF() a la nueva *flag* siempre y cuando importemos el fichero Stalls.hh en nuestro código. Un ejemplo de uso es el siguiente:

```
#include Stalls.hh
...
DPRINTF(Stalls, "Mensaje de Debug\n");
```

5.2 La Clase *Rename*

El fichero rename.cc ubicado dentro del directorio /tfg/gem5/src/cpu/o3/ define todas las funciones y modela la ejecución de la etapa *rename* del *pipeline* de Gem5. Un ciclo de ejecución en Gem5 funciona como se muestra en la figura 5.1. La clase principal del simulador se encuentra en el fichero cpu.cc, donde se

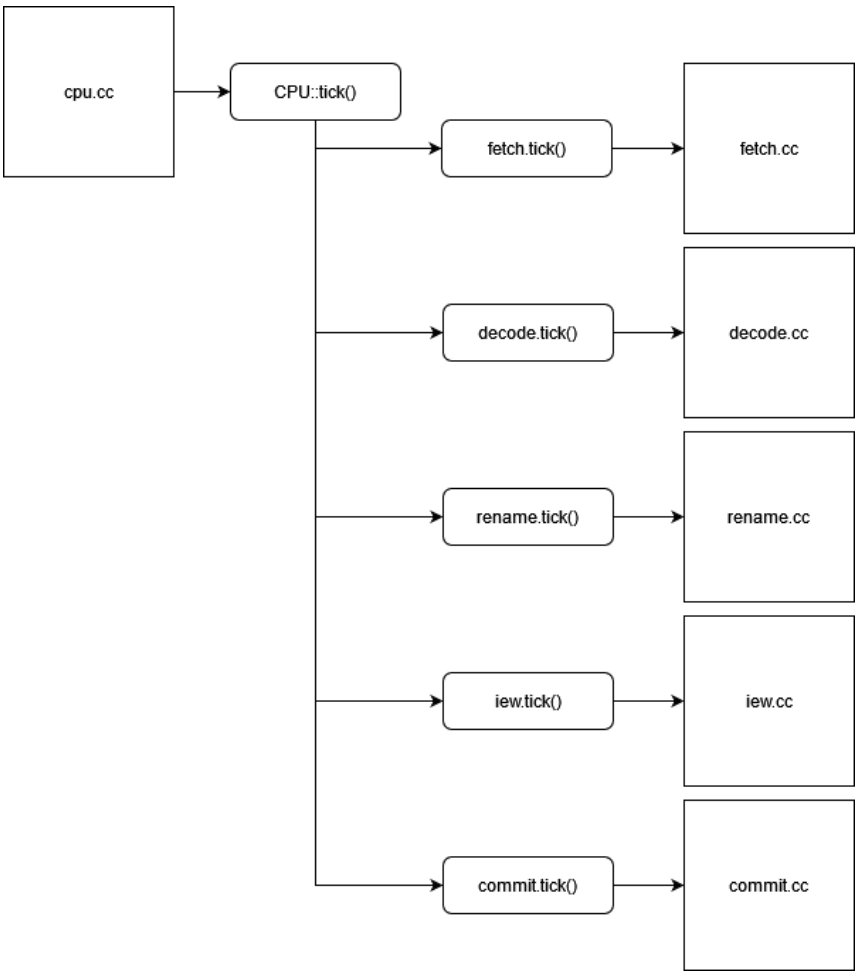

Figura 5.1: Flujo de ejecución de un ciclo en Gem5.

define un método llamado tick(). Este método llama de forma secuencial a cada uno de los métodos tick() de las distintas etapas del *pipeline*, esto simula un ciclo de procesador en cada una de estas etapas. Como ya se ha expuesto con anterioridad, la PMU ha de ser implementada en la etapa de renombrado de instrucciones (fichero rename.cc).

El funcionamiento de esta etapa es el siguiente, el método Rename::tick() inicializa variables de entorno como *buffers* y prepara el estado inicial de la etapa. Inmediatamente después se busca en la cola de instrucciones los índices de las instrucciones a ejecutar, luego por cada hilo en ejecución mediante un *while* y un sistema de prioridades se asigna los *slots* del ciclo al hilo correspondiente en la lista. En esta sección del trabajo se detalla el proceso de implementación de la PMU en monohilo, y más adelante se ampliará a multihilo.

Dentro del método Rename::tick() se llama al método Rename::rename(), el cual simula la etapa *rename* para las instrucciones correspondientes a los *slots* que le hayan sido asignadas al hilo que se ejecuta. Dentro del método Rename::rename() se realizan varias comprobaciones. Primero de todo se comprueba el estado de la etapa. Esta puede estar en 5 estados:

- ***Blocked***: En este estado no se pueden renombrar instrucciones debido a dependencias o limitaciones de recursos como por ejemplo, falta de espacio en recursos del *backend*.

- ***Squashing***: En este estado no se renombran instrucciones debido a que se están invalidando instrucciones que han sido ejecutadas de forma especulativa. Esto se debe principalmente a fallos del predictor de saltos.

- ***Serializing***: En este estado no se renombran instrucciones debido a que se están ejecutando instrucciones que no se pueden ejecutar fuera de orden y deben ser serializadas.

- ***Idle***: En este estado no se renombran instrucciones debido a falta de instrucciones proporcionadas por las etapas *fetch/decode*.

- ***Running***: En este estado se renombran y procesan instrucciones de forma normal.

En caso de que la etapa se encuentre en cualquier otro estado que no sea *Running*, se finaliza la simulación de *Rename* y se procede con la siguiente etapa. En el caso de que el estado sea *Running*, se llama al método Rename::renameInsts(), donde se gestiona el renombrado de instrucciones.

Dentro del método Rename::renameInsts(), para las instrucciones Load y Store se comprueba si existe espacio en las respectivas colas LQ y SQ. Por otro lado, si se trata de una instrucción que debe ser serializada se inicia el proceso de vaciado del *pipeline*. En este método también se notifica al resto del *pipeline* para bloquearlo en el caso de que durante el renombrado alguno de los recursos del backend se llene.

5.3 Implementación de Contadores *Hardware*

La unidad desarrollada en este trabajo simula una unidad independiente que mediante el uso de contadores es capaz de registrar información del procesamiento de las aplicaciones.

Para simular los contadores hardware y su integración en el *pipeline* se emplearán variables globales que serán actualizadas en cada ciclo del procesador. Estas variables globales se declaran en el constructor de la clase y se correspoden con las categorías de la *performance stack* planteada en la sección 4.3.

Se busca que cada variable haga referencia a una categoría de la pila o que, en su defecto, sean variables de control. Esto nos facilita el posterior análisis de las estadísticas. Estas variables simulan contadores hardware, y su funcionamiento queda descrito a continuación.

En el método Rename::rename(), cuando se hace la comprobación del estado de la etapa se añade código para contabilizar los ciclos de parada en el caso de que el estado impida el renombrado de instrucciones.

En caso de estar la etapa bloqueada, como esto se debe a la falta de espacio en unidades físicas se añade el código siguiente para comprobar el estado de las unidades.

```
1  if (renameStatus[tid] == Blocked) {
2      ++stats.blockCycle;
3      DPRINTF(Stalls, "Etapa Rename en estado Bloqued")
4      if (calcFreeROBEntries(tid) <= 0 && Inst_exec == 0) {
5          ROB_inst = ROB_inst + renameWidth;
6      } else if (calcFreeIQEntries(tid) <= 0 && Inst_exec == 0) {
7          IQ_inst = IQ_inst + renameWidth;
8      } else if (insts_available_rename == 0) {
9          Frontend_inst = Frontend_inst + renameWidth;
10     }
```

El contador implementa las prioridades definidas en la *performance stack*, donde se le da prioridad a ROB frente a IQ e IQ frente a *frontend*.

En el caso de que la etapa se encuentre en *squashing*, se asignan estos *slots* a la variable *Squashed_inst*. En caso de encontrarse la etapa en *serializing*, se asignan estos *slots* a la variable *Serializing_inst*.

Cuando la etapa *Rename* se encuentra en estado *Running* se procede al renombrado de las instrucciones. Para actualizar correctamente los contadores *Hardware* simulados, contabilizando tanto instrucciones renombradas como *slots* desaprovechados, se ha de realizar un conteo detallado teniendo en cuenta las diferentes situaciones que pueden ocurrir durante el renombrado. Gem5, para renombrar instrucciones emplea un bucle *while*, el cual itera por cada *slot* disponible en el ciclo de ejecución. Es decir, tantas iteraciones como ancho o *width* de renombrado configurado en la microarquitectura. En el bucle se contabilizan instrucciones renombradas hasta que se utilice todo el ancho de renombrado o bien hasta que se quede sin espacio alguno de los recursos *hardware* necesarios. En este último caso el bucle finaliza y se bloquea el *pipeline* hasta que se libere el recurso en cues-

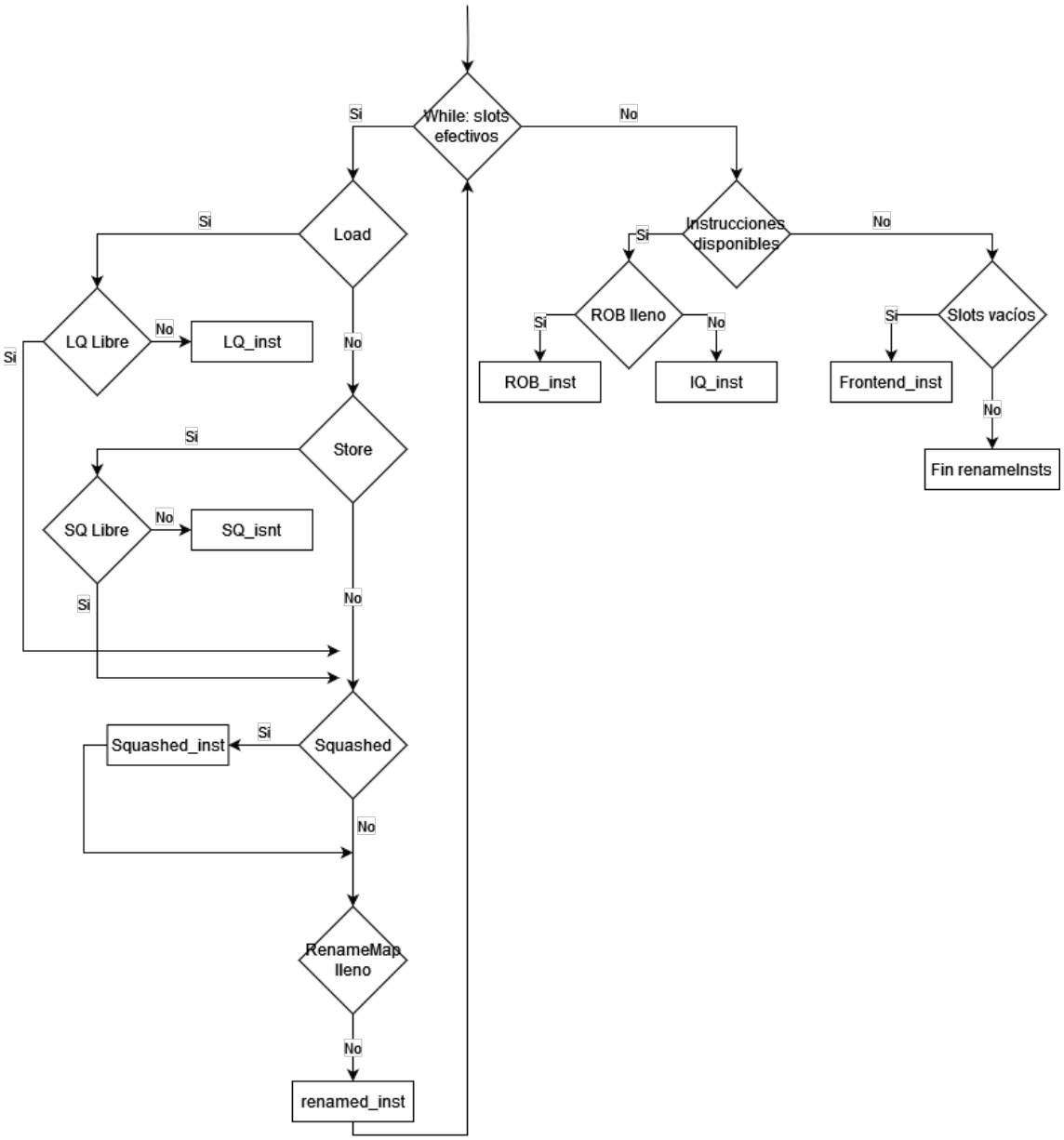

Figura 5.2: Árbol de decisiones del bucle *while*.

tión. Además, el bucle también contabiliza instrucciones canceladas (*squashed*) o serializadas.

Para poder tener en cuenta todos los casos posibles de tipos de instrucciones y *slots* desperdiciados, se ha añadido al código de la función Rename::renameInst() la variable control, la cual identifica en qué iteración del bucle *while* se ha parado el renombrado para poder contabilizar correctamente los *slots* desaprovechados.

La lógica seguida para implementar los contadores en esta función, representada por la figura 5.2, es la siguiente. Antes de entrar en el bucle *while*, que itera por cada *slot*, se contabiliza cuantos de estos *slots* son *slots* efectivos, es decir:

1. Se cuantifica el espacio disponible del ROB y la IQ, las cuales son dos unidades funcionales indispensables para el renombrado, ya que todas las instrucciones requieren de un espacio de ROB y de IQ.

2. Se mide cuantas instrucciones hay disponibles para renombrar este ciclo. Es decir, el número máximo de instrucciones que podrían llegar a ser renombradas si se utilizara todo el ancho de la etapa.

3. Se calcula el mínimo entre el espacio de ROB, espacio de IQ e instrucciones disponibles, obteniendo así el número máximo de *slots* efectivos, ya que, aunque dispongamos de un ancho ilimitado, si no disponemos de instrucciones a renombrar o de espacio suficiente en ROB e IQ, no se podrían utilizar todos los *slots*.

4. El bucle *while* itera sobre el número máximo de *slots* efectivos.

Una vez dentro del bucle *while*, a la hora de renombrar una instrucción se comprueba si esta instrucción es una *load* o una *store*. Si se trata de una de ellas se comprueba la disponibilidad de espacio en las colas LQ (*Load Queue*) o SQ (*Store Queue*). En caso de no haber espacio para la instrucción se sale del bucle *while* y se contabilizan los *slots* desperdiciados como *slots* de LQ o SQ siguiendo la siguiente fórmula.

$$Contador = Contador + (renameWidth - control); \tag{5.1}$$

Seguidamente, se comprueba si la instrucción ha sido cancelada. En ese caso, se contabiliza en la variable *Squashed_inst* y se continúa con la siguiente iteración del bucle.

En caso de que no haya ningún impedimento descrito anteriormente para renombrar la instrucción, se realiza la comprobación de espacio disponible en el banco de registros físicos para almacenar el resultado de la instrucción. De no haber espacio disponible, se emplea la fórmula 5.1 para actualizar la variable *Phys_inst*.

En caso de que la instrucción sea serializable, como el pipeline ha de ser vaciado y ninguna instrucción mas va a poder ser renombrada en el ciclo, se sale del bucle *while* y se actualiza el contador de *Serializing_inst* con la formula 5.1.

Una vez llegado a este punto del bucle *while*, se renombra la instrucción pertinente y se actualiza la variable *renamed_inst* incrementando el conteo de instrucciones renombradas. Finalmente, se continua con la siguiente iteración del bucle para el proceso del siguiente *slot* efectivo.

Una vez finalizado el bucle *while*, se comprueba si quedan instrucciones disponibles a renombrar este ciclo. De ser este el caso, se comprueba el espacio del ROB y de la IQ, y se actualizan los contadores pertinentes para contabilizar los *slots* desperdiciados. En caso de no haber instrucciones disponibles pero sí espacio en ROB e IQ, se asignan esos *slots* al contador *frontend_inst*.

5.4 Implementación de la PMU

Una vez contamos con la parte más complicada que es la implementación de los contadores *Hardware* simulados, se procede a implementar la PMU. Una PMU no es sino un componente *Hardware* que lee y almacena información de una serie de contadores *Hardware*. En nuestro caso, la PMU leerá al final de cada ciclo los

datos de los contadores, almacenándolos para poder ser accedidos por el usuario. Esta unidad juega el papel de intermediadora entre la CPU y el usuario.

Aprovechando las características de la implementación de los contadores como variables globales, en la clase *Rename* llamaremos a un método al cual hemos bautizado como Rename::PMU() al final de cada ejecución de esta etapa para poder almacenar la información de estos contadores.

Dentro de la unidad simulada, declararemos variables locales a forma de simular contadores internos de la PMU. Estos contadores internos de la PMU siguen el diseño de la pila y la *performance stack* ya planteado en este trabajo. Estos contadores internos acumulan los valores de los contadores locales en las categorías principales de la pila, como por ejemplo son *Frontend_stall*, *Backend_Stall* e *Instructions_dispatched*.

5.5 Estadísticas

Una vez contamos ya con la PMU hemos de disponer de alguna forma de acceder a la información almacenada mediante un mecanismo óptimo y cómodo para el usuario. Siguiendo ese objetivo, se ha decidido implementar un sistema mediante el cual tengamos acceso a la información de ciclos útiles y ciclos de parada ciclo a ciclo mediante la lectura y escritura de un archivo csv. Se ha elegido csv como el formato para las salidas de datos de la unidad debido a su simplicidad, portabilidad y sobrecoste mínimo.

Las estadísticas se construyen en función de las categorías que mide la PMU y las categorías descritas y propuestas en la *performance stack*. Es por eso que las columnas de nuestro archivo csv son las siguientes:

```
cycle,squashed,physical,ROB,IQ,serializing,LQ,SQ,frontend,
instructions, stalls_frontend, stalls_backend, stalls_squashed,
stalls_serializing, tid
```

La implementación de las estadísticas en Gem5 es la siguiente. Se emplea la clase std::ofstream, la cual pertenece al estándar de librerías de C++. Esta clase nos permite crear flujos de datos para poder escribir ficheros desde nuestro programa.

Creamos un fichero llamado “data.csv” o, en caso de que ya exista, sobrescribimos su contenido. Posteriormente, añadimos en la primera fila los identificadores de las columnas mencionadas.

Cada ciclo, al finalizar la etapa *rename* cuenta con la información de los *slots* ocupados por las instrucciones que se han renombrado correctamente y de los *slots* desperdiciados. Estos datos son los que se escriben en el fichero csv. De esta forma se almacena la información de la ejecución detallada ciclo a ciclo.

El fichero csv resultante es más tarde procesado por un programa Python para obtención de gráficas y resultados de análisis. Este fichero se encuentra adjunto como anexo en A.2.3.

5.6 Ejecución de los *Benchmarks*

Contando ya con la unidad implementada, los contadores y un sistema de recolección y análisis de los datos, se detalla el proceso de lanzamiento y medición de las *suites* de *benchmarks* SPEC CPU2006 y SPEC CPU2017 basadas en aplicaciones científicas reales.

Estas *suites* de me han sido proporcionadas por el Grupo de Arquitecturas Paralelas (GAP). Para poder lanzar la *suite* de 2006 lo primero que tenemos que hacer es crear un directorio de trabajo. Este directorio de trabajo, llamado working_dir, ha sido creado dentro del directorio de Gem5.

```
cd ~/tfg/gem5/
mkdir working_dir
cd working_dir
```

El proceso para realizar esto en los *clusters* computacionales del GAP Eri y CMTS es el siguiente. Primero nos conectamos vía ssh[2] a los *clusters* y posteriormente realizamos los pasos anteriores.

```
ssh -p 3322 mapecfer@cluster.gap.upv.es
cd ~/tfg/gem5
mkdir working_dir
```

Ahora se procede a cargar los *benchmarks* en el directorio de trabajo. En nuestra máquina local simplemente empleamos el comando de bash cp para copiarlos del directorio donde se encuentren al directorio de trabajo.

```
cp -r ~/Downloads/spec2006-x86-bin/* ~/tfg/gem5/working_dir/
```

En el caso de Eri y CMTS, para cargar las SPEC desde la máquina local se ha de usar un protocolo de transferencia de archivos por la red. Por ejemplo, se podría usar indistintamente sftp o scp. Para este trabajo se detalla el uso de sftp.

```
sftp -P 3322 mapecfer@cluster.gap.upv.es
cd ~/tfg/gem5/working_dir
put -r ~/Downloads/spec2006-x86-bin/*
```

Una vez situados en nuestro directorio de trabajo working_dir ya podemos proceder a lanzar la ejecución de los *benchmarks*. A continuación se proporciona un ejemplo de como lanzar un *benchmark* en la máquina local.

```
../build/X86/gem5.opt ../configs/example/se.py --cpu-type=O3CPU
--l1d_size=32kB --l1i_size=32kB --caches --l2_size=256kB --l2cache
--l1d_assoc=8 --l1i_assoc=8 --l2_assoc=8 --maxinsts=1000000
--cmd=spec2006-x86-bin/400.perlbench/perlbench_base.i386
--options="-ICPU2006/400.perlbench/data/all/input/lib
CPU2006/400.perlbench/data/all/input/diffmail.pl 4 800 10 17 19 300"
```

[2] https://www.ssh.com/academy/ssh

Como podemos observar, en la propia linea de lanzamiento se configura el subsistema de memoria, se selecciona el fichero binario del *benchmark* y se le pasan las opciones al programa.

El proceso de lanzar un *benchmark* en un *cluster* computacional difiere un poco, ya que estos *clusters* emplean SLURM para gestión de tareas. Para emplear SLURM necesitamos crear un fichero de *script* .sh desde donde configuraremos nuestro programa con vim.

```
touch spec2006.sh
vim spec2006.sh
```

Una vez dentro del fichero, configuramos todos los *benchmarks* que vamos a lanzar y configuramos SLURM para aprovechar la paralelización que nos ofrece los 64 *cores* por nodo de los *clusters* computacionales. A continuación se ejemplifica una parte del script spec2006.sh (el fichero se adjunta entero en los anexos A.2.2).

```
#!/bin/bash
#SBATCH --job-name=spec2006
#SBATCH --nnodes=1
#SBATCH --ntasks=24
#SBATCH --cpus-per-task=1

../build/X86/gem5.opt ../configs/deprecated/example/se.py --cpu-
    type=O3CPU --l1d_size=32kB --l1i_size=32kB --caches --l2_size
    =256kB --l2cache --l1d_assoc=8 --l1i_assoc=8 --l2_assoc=8 --
    maxinsts=1000000 --cmd=spec2006-x86-bin/400.perlbench/
    perlbench_base.i386 --options="-ICPU2006/400.perlbench/data/
    all/input/lib CPU2006/400.perlbench/data/all/input/diffmail.
    pl 4 800 10 17 19 300"
mv data.csv ./resultados/perlbench.csv

../build/X86/gem5.opt ../configs/deprecated/example/se.py --cpu-
    type=O3CPU --l1d_size=32kB --l1i_size=32kB --caches --l2_size
    =256kB --l2cache --l1d_assoc=8 --l1i_assoc=8 --l2_assoc=8 --
    maxinsts=1000000 --cmd=spec2006-x86-bin/401.bzip2/bzip2_base.
    i386 --options="CPU2006/401.bzip2/data/all/input/input.
    program 280"
mv data.csv ./resultados/bzip.csv
...
...
```

El proceso de lanzar la suite de *benchmarks* SPEC CPU2017 difiere un poco de su versión de 2006 ya que muchos de estos *benchmarks* requieren de entornos de lanzamiento para poder funcionar correctamente. Para poder ejecutar estas aplicaciones científicas, necesitamos crear un directorio de trabajo (en este caso lo llamaremos skel). Dentro de este directorio crearemos un directorio por *benchmark* y guardaremos en este directorio el entorno necesario para la ejecución.

```
cd ~/tfg/gem5/
mkdir skel
```

```
cd skel
mkdir specX
cd specX
cp -r ~/spec2017/benchspec/CPU/specX/data/refrate/input/* ./
```

Realizamos este proceso sustituyendo specX por el nombre correspondiente de cada *benchmark* SPEC. En el caso de los *clusters* computacionales nos conectamos por ssh para la creación de directorios y luego nos conectamos por sftp para la subida de los benchmarks.

```
ssh -p 3322 mapecfer@cluster.gap.upv.es
cd ~/tfg/gem5
mkdir skel
cd skel
mkdir specX
exit
sftp -P 3322 mapecfer@cluster.gap.upv.es
cd ~/tfg/gem5/skel/specX
put -r ~/spec2017/benchspec/CPU/specX/data/refrate/input/*
```

El lanzamiento en local es análogo a lo expuesto para la versión de 2006. En los *clusters* computacionales se emplea SLURM siguiendo una estructura similar al fichero spec2006.sh, este fichero también se encuentra en los anexos A.2.2.

5.7 Ampliación a SMT

Una vez desarrollada la unidad de medición de prestaciones, se ha propuesto la ampliación de esta unidad de monohilo a multihilo en el mismo *core* del procesador. En esta sección se detalla el proceso.

Gem5 en su modelo de ejecución fuera de orden emplea una política de planificación *round-robin* para los hilos. Esto significa que cada ciclo el *pipeline* está ocupado en su totalidad por un solo hilo, el cual varía ciclo a ciclo como se ilustra en la figura 5.3. Esta figura en el eje Y representa el ancho de la etapa *rename* con el número de *slots* disponibles, mientras que el eje X muestra como cada ciclo se procesan instrucciones de un único hilo variando este hilo de forma cíclica.

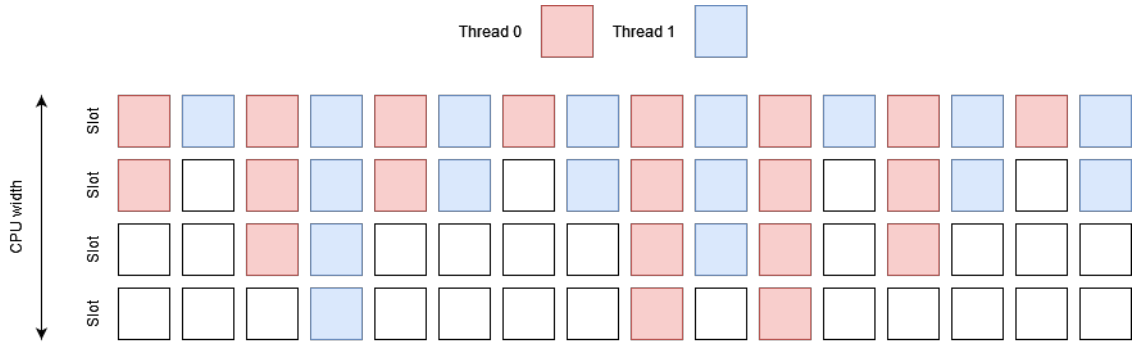

Figura 5.3: Ejemplo de política de asignación de hilos.

A partir de la implementación monohilo, basta con identificar el ciclo que corresponde a cada hilo y llevar las cuentas de cada hilo por separado. En concreto, se mantiene un fichero csv por cada hilo, de forma que sea posible obtener estadísticas de ejecución independientes por hilo. Estos ficheros se crean al principio de la ejecución y se escriben ciclo a ciclo con la información de los contadores.

La ampliación a multihilo conlleva la creación dentro de la PMU de contadores independientes por hilo para poder mantener el primer nivel de la *performance stack* por cada hilo. El código de la ampliación a SMT se adjunta en los anexos.

CAPÍTULO 6
Entorno Experimental

Con el objetivo de comprobar el funcionamiento de la PMU, se han medido las prestaciones utilizando la PMU modelada en una simulación Gem5 de un procesador realista con soporte a ejecución fuera de orden y SMT. En este capítulo se detallan las herramientas, la arquitectura del sistema y las aplicaciones utilizadas.

6.1 *Clusters* para Simulaciones

6.1.1. *Cluster* Computacional ERI

Eri es un *cluster* computacional del DISCA[1].Está formado por 14 nodos, nombrados desde Eri1 hasta Eri14. Eri1 actua como puerta de enlace siendo el nodo *frontend* desde el cual se accede al resto del sistema.

Cada nodo cuenta con biprocesadores de 6 *cores* de Intel pertenecientes al modelo Xeon E5645[2] que funcionan a una frecuencia máxima de 2.4GHz. A su vez, cada nodo cuenta con 48GB de memoria principal con tecnología DDR3 y 256GB de memoria secundaria (SSD). Los distintos nodos están interconectados por una red Ethernet de 1G. El sistema operativo instalado es Ubuntu[3] 20.4.

6.1.2. *Cluster* Computacional CMTS

CMTS es un *cluster* computacional más moderno que Eri. Cuenta con 10 nodos nombrados siguiendo el mismo criterio que el seguido con Eri, desde CMTS1 hasta CMTS10. CMTS1 actua como *frontend*, desde el cual se puede acceder al resto del *cluster*.

Cada nodo cuenta con biprocesadores AMD, en concreto procesadores del modelo Rome[4]. Cada nodo tiene una frecuencia de funcionamiento máxima de 2.8GHz, además de contar con un sistema de memoria de altas capacidades, siendo la memoria principal capaz de albergar hasta 512GB de tecnología DDR4.

[1] `https://www.upv.es/entidades/disca/`

[2] `https://ark.intel.com/content/www/xl/es/ark/products/48768/intel-xeon-processor-e5645-12m-cache-2-40-ghz-5-86-gt-s-intel-qpi.html`

[3] `https://ubuntu.com/`

[4] `https://www.amd.com/en/products/processors/server/epyc/7002-series.html`

CMTS al estar enfocado en calculo científico cuenta con gráficas de altas prestaciones como son las NVIDIA A100[5]. Para comunicar los distintos nodos entre sí emplea conexión Ethernet a 10G y tecnología Infiniband HDR. Cada nodo cuenta con una memoria secundaria de 4TB de capacidad con tecnología SSD. El sistema operativo instalado es Ubuntu 18.4.

Aunque los primeros experimentos se realizaron en el *cluster* Eri, los resultados experimentales presentados en este trabajo han sido obtenidos en CMTS, lo que nos ha permitido acortar al máximo los tiempos de simulación, agilizando el trabajo.

6.2 Gem5: Instalación y Configuración del Simulador

En esta sección se presenta la tecnología de simulación empleada para la realización de este trabajo. Además, se presenta también la microarquitectura Vulcan modelada para los experimentos y el software de control de versiones.

6.2.1. Simulador Gem5

Gem5 es un simulador de arquitecturas de CPU de código abierto ampliamente utilizado en la academia y en la industria para el desarrollo de arquitecturas de procesadores y sistemas complejos. Gem5 nace como la fusión de los proyectos M5 y GEMS. Gem5 ofrece una amplia gama de arquitecturas de CPU para simular, desde los simples procesadores *in-order*, hasta los más complejos procesadores *out of order*. También soporta simulación mononúcleo y multinúcleo, al igual que soporta simulación monohilo y multihilo (*Simultaneous Multi Threading* o SMT).

Gem5 es un simulador ampliamente configurable, contando con diferentes modos de ejecución e implementando un gran rango de parámetros de configuración. Gem5 permite estudiar de forma precisa interacciones *hardware-software*. Está conformado por las siguientes partes:

CPU

La CPU conforma el módulo principal del simulador. El simulador no está pensado para replicar el funcionamiento de una arquitectura específica, sino que implementa modelos de CPU genéricos que mediante ficheros de configuración de Python[6] se pueden adaptar para simular diferentes microarquitecturas. A continuación se presentan los modelos de CPU que incluye Gem5:

- **SimpleCPU:** Este modelo simple es meramente funcional. SimpleCPU implementa una ejecución en orden muy sencilla cuyo objetivo está pensado

[5] `https://www.nvidia.com/es-es/data-center/a100/`
[6] `https://www.python.org/`

para soportar periodos de carga e inicialización de aplicaciones o para comprobar el funcionamiento de estas. Este modelo de CPU se desgrana en tres submodelos para soportar diversos modelos de memoria. Estos submodelos son:

  - **BaseSimpleCPU**. Soporta *prefeching* y validado de instrucciones. No modela la jerarquía de memoria.
  - **AtomicSimpleCPU**. Es una versión de SimpleCPU que soporta accesos atómicos a memoria. No tiene en cuenta latencia entre niveles de *cache*.
  - **TimingSimpleCPU**. Es la versión de SimpleCPU que simula la jerarquía de memoria de forma detallada, incluyendo las latencias de acceso a los diferentes niveles de *cache*.

- **O3CPU:** La ejecución fuera de orden es crucial para modelar correctamente los procesadores actuales. O3CPU implementa un modelo de procesador detallado con ejecución fuera de orden. Cuenta con etapas segmentadas, diferentes modos de ejecución, ejecución multihilo y multinucleo, y modelo del subsistema de memoria detallado. En O3CPU es necesario declarar explícitamente un subsistema de memoria con al menos memorias *cache* de primer y segundo nivel.

Subsistema de Memoria

El subsistema de memoria es el otro de los grandes módulos configurables que tiene Gem5. Es un componente crucial que modela la jerarquía de memoria, sus estructuras, las interacciones entre ellas y con el resto del sistema simulado. Gem5 implementa dos modelos de subsistema de memoria: el sistema clásico y Ruby[7]. Se procede a explicar estos dos modelos.

- **Sistema de memoria clásico**: Este modelo aproxima un subsistema de memoria de forma conservadora, centrándose en la simplicidad y usabilidad. Modela los siguientes componentes:

  - ***Caches***: Implementa tres niveles de caché distintos: L1, L2 y L3. Estas memorias son configurables con respecto a capacidad, asociatividad, tamaño de bloque y políticas de remplazo.
  - **Memoria principal**: Por simplicidad y para optimizar la simulación, no se simula el funcionamiento completo de una memoria DRAM, sino que se utiliza un sistema de temporizadores para simular los tiempos de acceso.
  - **Controladores de memoria**: Gestionan la interacción entre la *cache* de último nivel y la memoria principal. Soportan peticiones y respuestas, el intercambio de datos con la memoria y se encargan del funcionamiento de los protocolos de coherencia.

[7] `https://www.ruby-lang.org/en/`

Este modelo es fácilmente configurable y no supone una gran sobrecarga de simulación. Sirve para agilizar el desarrollo, ya que cuenta con componentes y configuraciones predefinidas que ahorran tiempo de modelado.

- **Sistema de memoria Ruby**: Modelo más avanzado y flexible que el clásico. Diseñado para proveer simulaciones detalladas y personalizadas de la jerarquía de memoria. Su principal enfoque es ofrecer flexibilidad en el diseño de protocolos de coherencia, las interacciones entre estructuras de memoria memoria y transferencia de información entre unidades.

  Ruby soporta la simulación detallada de *Network on Chip* (NoC), controladores de memoria y múltiples tipos de componentes como las *caches* de diferentes niveles, memorias principales e incluso tecnologías de memoria no volátil.

  Ruby es altamente personalizable. Permite implementar un gran rango de funcionalidades y protocolos personalizados, además de ofrecer una simulación muy detallada del subsistema de memoria. No obstante, añade mucha sobrecarga a la ejecución del simulador, y no cuenta con estructuras predefinidas, lo cual dificulta y alarga mucho el desarrollo. Por estas razones, en este trabajo se ha optado por emplear el sistema clásico de memoria.

6.2.2. Instalación del Simulador

A continuación se detalla el proceso de instalación del simulador para la realización del trabajo.

El simulador ha sido instalado en tres máquinas distintas. En la máquina del autor de este trabajo: un ACER swift[8] 3 14 de 2020, el cual cuenta con un procesador AMD Ryzen 5500u[9] con 16GB de DRAM y Windows 11[10] como sistema operativo, en el *cluster* ERI, y en el *cluster* CMTS. El proceso de instalación ha sido similar en los tres casos.

Para instalar Gem5 en nuestra máquina lo primero es cumplir los prerrequisitos que nos indica la documentación del simulador. Estos prerrequisitos son dependencias necesarias para el funcionamiento del simulador. En concreto, para instalar la versión 23.1.0.0 requerimos de las siguientes librerías y sus respectivas versiones:

- **Git**: Para el control de versiones. Se instala con la orden:

```
sudo apt install git
```

- **Gcc** (versión 8+): Gcc es una colección de compiladores pertenecientes a GNU. Se instala con la orden:

[8] https://www.acer.com/us-en/laptops/swift/swift-3-intel

[9] https://www.amd.com/en/support/downloads/drivers.html/processors/ryzen/ryzen-5000-series/amd-ryzen-5-5500u.html

[10] https://blogs.windows.com/windowsexperience/2021/10/04/windows-11-a-new-era-for-the-pc-begins-today/

```
sudo apt install build-essential
```

- **Scons** (versión 3.0+): Scons es una herramienta para compilar y construir proyectos de software que emplea *scripts* de Python para configurar el proceso. Se instala con la orden:

```
sudo apt install scons
```

- **Python** (versión 3+: Para instalarlo se emplea la siguiente orden:

```
sudo apt install python3-dev
```

- **Protobuf** (versión 2.1+): Es una plataforma para estructurar información. Se instala con la orden:

```
sudo apt install libprotobuf-dev protobuf-compiler\\
libgoogle-perftools-dev
```

- **Boost**: Es un conjunto de librerías de propósito general para C. Se instala con la orden:

```
sudo apt install libboost-all-dev
```

Una vez cumplidos los prerrequisitos creamos un directorio para alojar el código clonado del repositorio de Gem5 y nos movemos dentro con las ordenes:

```
mkdir tfg
```

```
cd tfg
```

Una vez dentro del directorio de trabajo procedemos con la clonación del repositorio empleando Git con la orden:

```
git clone https://github.com/gem5/gem5
```

Una vez clonado el repositorio, se creará un directorio llamado gem5, al cual nos moveremos para proceder con la compilación del simulador. Para compilar emplearemos Scons, instalado anteriormente. Todo esto se realiza con las órdenes:

```
cd gem5
```

```
/usr/bin/python3 /opt/scons/bin/scons build/X86/gem5.opt -j(nprocs)
```

La compilación es lenta y se puede acelerar y realizarse de forma paralela entre distintos *cores* mediante el parámetro -j. Este parámetro nos permite lanzar diversos hilos de compilación simultáneamente. Es recomendable lanzar cuantos más hilos sea posible (teniendo en cuenta los recursos de la máquina) para acortar al máximo el tiempo de compilación.

Durante la compilación, pueden saltar advertencias sobre la ausencia de librerías de estilo opcionales como "gem5 style" o similares. Estas librerías no son obligatorias y se emplean para comprobar que el estilo de programación es acorde con el estándar del simulador en caso de que se desee contribuir a su desarrollo.

Cuando se compila el simulador, se puede elegir distintos binarios objetivo:

- **gem5.opt**: La versión más optimizada del simulador. Pensada para su uso general, busca un compromiso entre precisión, velocidad, e información de funcionamiento.
- **gem5.debug**: Versión pensada específicamente para depuración. Cuenta con herramientas orientadas a este objetivo y proporciona información detallada del funcionamiento de la simulación sacrificando precisión y velocidad.
- **gem5.fast**: Ofrece la mayor velocidad de ejecución de las tres versiones. Está pensada para ejecutarse durante largos periodos de tiempo y con enormes cargas de trabajo. Sacrifica información y control sobre la simulación para maximizar el rendimiento.

6.3 Instalación del Simulador para SMT

En el trabajo se propone una ampliación de la PMU para funcionamiento con multihilo SMT.

Gem5 es un simulador que soporta de manera nativa el multihilo simultáneo (SMT, por sus siglas en inglés). Sin embargo, en la versión 23.1.0.0, se ha reportado un error que impide el correcto funcionamiento de esta característica. El repositorio Git de Gem5 contiene diversas ramas. En este trabajo, hemos utilizado la rama principal, pero en la rama *develop* el error está solucionado. Para poder aprovechar esta solución en nuestra copia de Git aplicaremos el parche necesario mediante el comando *cherry-pick*.

Nos situamos en el directorio principal del simulador y desde ahí le indicamos a git que busque *commits* de otras ramas.

```
cd ~/tfg/gem5
git checkout develop
```

Acto seguido usamos el comando git log para identificar el *hash* del *commit* en *develop* que queremos añadir a nuestro repositorio.

```
git log --one line
git checkout main
git cherry-pick commit-hash
git cherry-pick --continue
```

Una vez añadidos los cambios que solucionan el *bug* reportado y nos permiten ejecutar correctamente aplicaciones en SMT, recompilamos el simulador.

```
/usr/bin/python3 /opt/scons/bin/scons build/X86/gem5.opt -j64
```

Tras recompilar el simulador, lanzamos una ejecución sencilla en SMT de dos *hello world* para comprobar que todo funciona correctamente.

```
build/X86/gem5.opt configs/deprecated/example/se.py --smt
--cpu-type=DerivO3CPU --caches --l1d_size=32kB --l1d_assoc=8
--l1i_size=32kB    --l1i_assoc=8 --l2cache --l2_size=256kB
'--cmd=tests/test-progs/hello/bin/x86/linux/hello;
tests/test-progs/hello/bin/x86/linux/hello'
```

Como resultado de la ejecución deberíamos observar por pantalla una salida similar a esta:

```
Hello World!
Hello World!
```

Esto nos indica que los cambios se han aplicado correctamente y que ya contamos con soporte a SMT en nuestro simulador.

6.3.1. Configuración del Simulador: Arquitectura Vulcan

Como ya se ha expuesto con anterioridad, Gem5 es un simulador altamente configurable mediante *scripts* Python, lo que permite simular diferentes tipos de procesador. Para la realización de los experimentos del trabajo se ha simulado una microarquitectura real. En este caso se ha seleccionado Vulcan[11] como arquitectura a modelar. La razón principal es que es el grupo GAP dispone de una máquina con esta microarquitectura y que su PMU mide los contadores *hardware* en la etapa de *dispatch*. Recuérdese que la etapa *rename* del *pipeline* del modelo O3CPU de Gem5 simula la funcionalidad de *dispatch* de un procesador típico. De hecho, la microarquitectura Vulcan dispone de contadores para contabilizar ciclos de parada similares a los de las categorías de nuestra PMU (ROB, IQ, etc.).

Vulcan es una microarquitectura 64 bits implementada en 16 nanómetros de ARM y diseñada para alto rendimiento. La figura 6.1 muestra un esquema de esta microarquitectura. Entre sus características principales se encuentran las siguientes:

- CPU:
  - Frecuencia de reloj de 2.5 GHz.
  - *Pipeline* de 15 etapas.
  - Ancho de *pipeline* de 4 instrucciones.

[11] `https://en.wikichip.org/wiki/cavium/microarchitectures/vulcan`

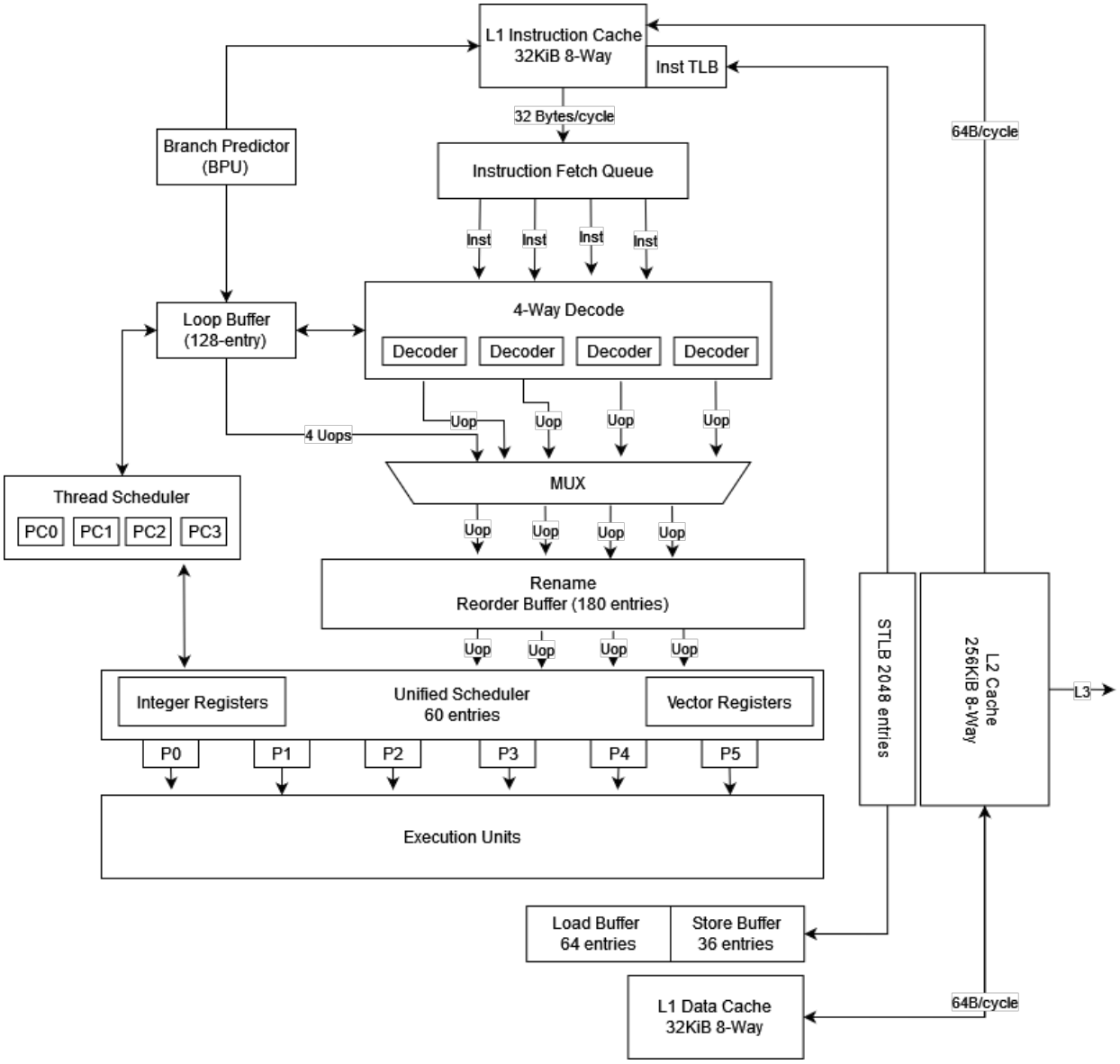

Figura 6.1: Esquema de un núcleo de microarquitectura Vulcan. Fuente [10].

- Cola de instrucciones de 60 entradas.
- ROB de 180 entradas.
- 2 unidades de coma flotante de 128 bits.

- Subsistema de Memoria:
  - Ancho de banda 128-bit por ciclo.
  - Controlador de memoria DDR4 de 8 canales.
  - *Caches*:
    - L1I 32 KiB, asociativa por conjuntos de 8 vías.
    - L1D 32 KiB, asociativa por conjuntos de 8 vías.
    - L2 256 KiB, asociativa por conjuntos de 8 vías.
    - L3 1 MiB/core.

Hay algunas limitaciones con respecto a la capacidad de modelado del simulador. Por ejemplo, el número de etapas en las que se encuentra segmentado el *pipeline* del procesador en Vulcan es 15 mientras que Gem5 implementa un menor número de etapas. Sin embargo, muchas de las características y dimensiones de las estructuras del procesador son modelables. En este sentido, en este trabajo se han realizado los siguientes cambios en la configuración de Gem5.

Nos desplazamos al directorio donde se encuentra el fichero de configuración del modelo de ejecución fuera de orden.

```
cd ~/tfg/gem5/src/cpu/o3
```

Una vez en el directorio correspondiente, abrimos con un editor de texto el archivo de configuración.

```
vim BaseO3CPU.py
```

El fichero de configuración modificado se encuentra en el anexo A.2.1. Se muestra aquí un pequeño fragemento de las modificaciones realizadas:

```
numIQEntries = Param.Unsigned(60, "Number of instruction queue
    entries")
numROBEntries = Param.Unsigned(180, "Number of reorder buffer
    entries")
...
LQEntries = Param.Unsigned(64, "Number of load queue entries")
SQEntries = Param.Unsigned(36, "Number of store queue entries")
...
commitWidth = Param.Unsigned(4, "Commit width")
squashWidth = Param.Unsigned(4, "Squash width")
...
```

Concluida la configuración contamos con un simulador instalado y preparado para simular cargas ejecutándose en una máquina real similar a la que dispone el grupo GAP.

6.4 Cargas y Aplicaciones Empleadas

6.4.1. Planificador de Trabajos SLURM

En el marco de la computación de altas prestaciones en el que nos encontramos, la problemática de la óptima gestión de recursos es esencial para la obtención de las máximas prestaciones del sistema. Es por ello que para controlar y planificar la simulación de las cargas estudiadas se ha seleccionado el sistema SLURM (*Simple Linux Utility for Resource Management*)

SLURM ha sido diseñado para ser modular y escalable. Se encarga de asignar y monitorizar los trabajos que se lanzan en un *cluster*, asignando dinámicamente los trabajos a los diferentes nodos. Para ello, emplea un sistema de colas y prioridades para asignación de recursos.

6.4.2. *Suite* de *Benchmarks* SPEC CPU2006

Para evaluar la propuesta se ha seleccionado una amplia gama de *benchmarks* basados en aplicaciones científicas reales que se encuentran en la *suite* SPEC CPU-2006. La descripción de cada una de estas aplicaciones se encuentra en el apéndice B.1.

Se han seleccionado estas aplicaciones debido a que sus características son idóneas para este proyecto, al tratarse de uno de los conjuntos de aplicaciones más populares para medir las prestaciones de la CPU. Al contar con una amplia variedad de aplicaciones, se puede sobrecargar la unidad de procesamiento en cada una de las categorías de la *performance stack*, permitiendo observar la validez de la implementación de la unidad. Otra de sus ventajas es que es un conjunto de *benchmarks* estandarizado, cuyo uso es bastante extendido en la industria y el mundo académico, existiendo una amplia variedad de trabajos anteriores que han presentado resultados empleando estos *benchmarks*.

6.4.3. Suite de *Benchmarks* SPEC CPU2017

La suite de *benchmarks* SPEC CPU2017 constituyen la evolución natural de la suite de 2006. Esta suite busca modernizar las aplicaciones ya presentadas en su versión anterior y así, actualizarse a las necesidades de prueba y rendimiento de los procesadores más modernos. Se detalla cada aplicación en el apéndice B.2.

Este conjunto cuenta con más programas de prueba diseñados para evaluar el funcionamiento de la CPU en áreas más específicas y con cargas de trabajo más ajustadas al paradigma actual en diversos campos como la inteligencia artificial, la simulación científica y el análisis de datos. Uno de los principales cambios es el salto de las instrucciones de 32 bits a instrucciones de 64 bits.

CAPÍTULO 7
Evaluación de resultados

Este capítulo expone los resultados obtenidos de la ejecución de las *suites* de *benchmarks* SPEC CPU2006 y SPEC CPU2017. Se comprueba la correcta implementación de la PMU y se detallan las pruebas realizadas para su validación. Los experimentos se han realizado tanto en configuración monohilo como multihilo (SMT).

7.1 Metodología Experimental

En esta sección se detalla la metodología empleada para la realización de los experimentos y la toma de resultados.

Cuando en Gem5 se lanza la simulación de una aplicación, se simula el comportamiento del sistema desde el inicio de la ejecución. Esto supone que tanto la CPU como el subsistema de memoria se encuentran "vacios". Esta situación es irreal ya que en un sistema convencional raramente la CPU y la memoria se encuentran desocupados, lo que provoca unas mediciones alejadas de la realidad. Para solucionar este problema, típicamente, antes de comenzar las mediciones se realiza un proceso de *warmup*.

Esta técnica, representada en la figura 7.1, supone la ejecución de un número determinado de instrucciones antes de monitorizar las prestaciones. De esta forma nos aseguramos de que el sistema, y, en particular, las *caches* almacenan datos del *working set*, lo que reduce el alto impacto en las prestaciones causado por los fallos de arranque de las *caches* que se producen al inicio de la ejecución. Para este trabajo se ha decidido que el simulador haga un *warmup* de 2 millones de instrucciones, para luego proceder a la toma de datos durante el siguiente millón de instrucciones.

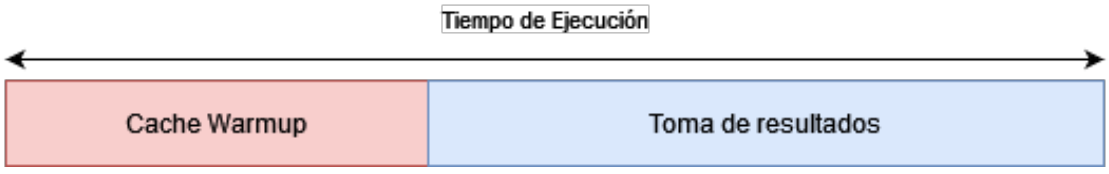

Figura 7.1: Esquema del tiempo de simulación de una aplicación.

Las aplicaciones se han caracterizado siguiendo la *performance stack* planteada en 4.3. Además, se han obtenido las siguientes métricas para cada aplicación estudiada:

- **Tiempo de ejecución:** Permite evaluar las prestaciones en términos de latencia de la ejecución.
- **IPC:** Nos permite evaluar el rendimiento de la aplicación en términos de productividad (instrucciones procesadas por ciclo).
- **IPC relativo:** En ejecución SMT, nos permite evaluar el IPC de cada aplicación con respecto a su ejecución en solitario. Cabe destacar que en SMT se espera un 30 % y un 40 % de caída con respecto al IPC en solitario. No obstante, el IPC global de la máquina (teniendo en cuenta ambos hilos ejecutándose en el núcleo) debe aumentar entre un 30 % y un 40 %. Es decir, la ejecución de 2 hilos en SMT incrementa la productividad global medida en instrucciones por ciclo.

A partir de la ejecución en solitario de los *benchmarks* y su caracterización se seleccionarán las parejas SMT para estudiar el impacto del multihilo. Para el análisis multihilo se han seguido las guias de Eyerman expuestas en [16], donde se discuten y explican diferentes técnicas para realizar estudios en sistemas SMT.

7.2 Ejecución en Solitario

En esta sección se ejecutan individualmente los *benchmakrs* SPEC CPU con el objetivo de caracterizarlos en categorías siguiendo el análisis Top-Down y así de seleccionar de forma adecuada las parejas para ejecución en SMT. También se busca con esta ejecución en solitario demostrar la efectividad del *warmup*.

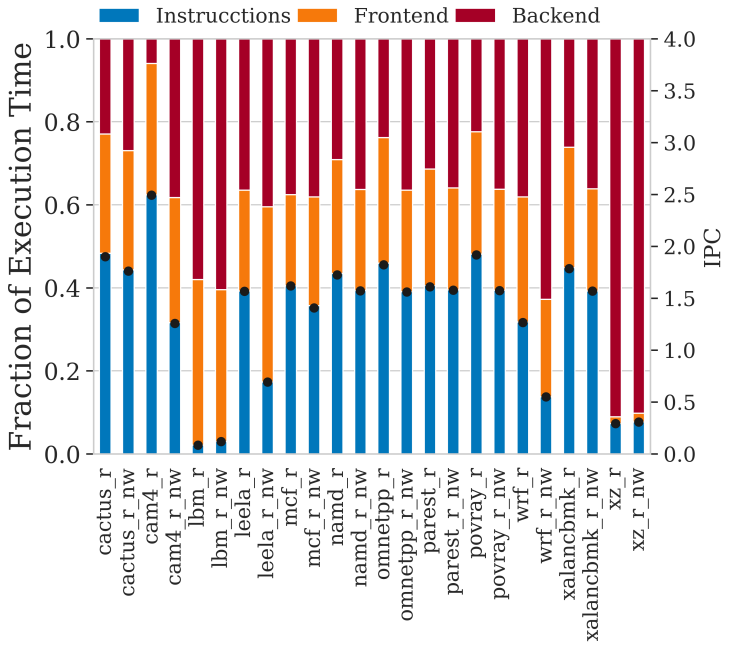

Figura 7.2: Ejecución en solitario de *benchmarks* SPEC CPU2017 con y sin *warmup*.

SPEC	cactus_r	cam4_r	lbm_r	leela_r	mcf_r	namd_r	omnetpp_r	parest_r	povray_r	wrf_r	xalancbmk_r	xz_r
IPC	1.90	2.49	0.08	1.56	1.61	1.72	1.82	1.60	1.91	1.26	1.78	0.29
Backend	0.21	0.06	0.58	0.36	0.37	0.29	0.23	0.32	0.22	0.38	0.35	0.91

Tabla 7.1: IPC y fracción de *backend* en SPEC CPU2017 con *warmup*.

SPEC	cactus_r	cam4_r	lbm_r	leela_r	mcf_r	namd_r	omnetpp_r	parest_r	povray_r	wrf_r	xalancbmk_r	xz_r
IPC	1.76	1.25	0.11	0.69	1.40	1.57	1.55	1.57	1.57	0.54	1.56	0.30
Backend	0.26	0.38	0.61	0.41	0.39	0.36	0.36	0.35	0.36	0.62	0.36	0.90

Tabla 7.2: IPC y fracción de *backend* en SPEC CPU2017 sin *warmup*.

7.2.1. *Performance Stacks* e IPC

En la figura 7.2 se muestran las pilas de la *suite* de *benchmarks* SPEC CPU2017 aplicando y sin aplicar el *warmup*. Para cada aplicación se muestra a la izquierda la pila con *warmup* y a la derecha sin *warmup* (nw, *no warmup*). La figura también muestra con una marca negra el IPC de cada aplicación. Se puede observar que este se ajusta perfectamente a la categoría *instructions*. Esto confirma la corrección en la medición de esta categoría, ya que cuenta los *slots* dedicados a instrucciones y no desperdiciados.

Como podemos apreciar, los datos tomados de las SPEC sin aplicar esta técnica difieren en gran medida, ya que gran número de los primeros ciclos de ejecución se usan para solventar fallos de *cache*, los cuales se atribuyen al *backend*. Para observar con más detenimiento el impacto de la técnica, las tablas 7.1 y 7.2 muestran los resultados de IPC y fracción de *slots* dedicados al *backend*. En promedio, el IPC pasa de 1,51 con *warmup* a 1,16 sin *warmup*, mientras que la fracción de *backend* pasa de 34 % a 45 %, respectivamente. En otras palabras, el IPC con *warmup* en un 30 % más alto que el IPC sin *warmup*, y el tiempo de *backend* sin *warmup* se eleva hasta un 45 %, un 11 % más que aplicando esta técnica.

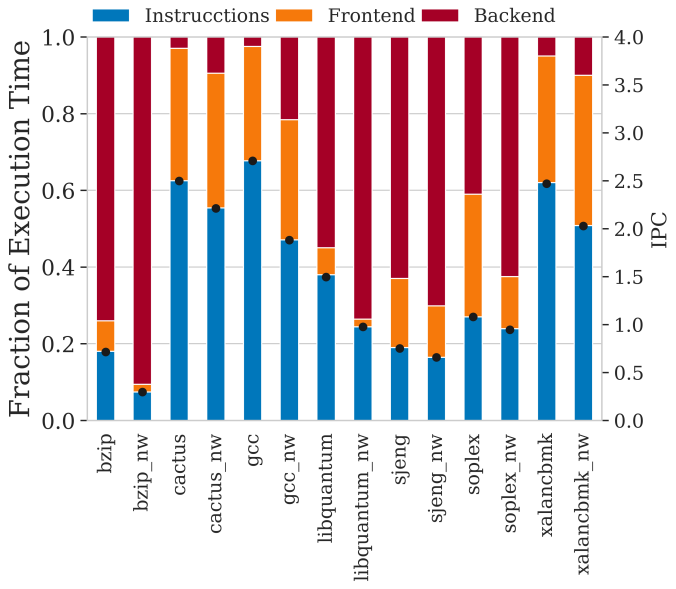

Figura 7.3: Ejecución en solitario de *benchmarks* SPEC CPU2006 con y sin *warmup*.

SPEC	bzip	cactus	gcc	libquantum	sjeng	soplex	xalancbmk
IPC	0.71	2.49	2.71	1.49	0.75	1.08	2.47
Backend	0.74	0.03	0.02	0.55	0.63	0.41	0.05

Tabla 7.3: SIPC y fracción de *backend* en SPEC CPU2006 con *warmup*.

SPEC	bzip	cactus	gcc	libquantum	sjeng	soplex	xalancbmk
IPC	0.29	2.21	1.88	0.97	0.65	0.94	2.02
Backend	0.90	0.09	0.21	0.73	0.70	0.63	0.10

Tabla 7.4: IPC y fracción de *backend* en SPEC CPU2006 sin *warmup*.

Tal como muestran los resultados presentados en la figura 7.3 y las tablas 7.3 y 7.4, el comportamiento con respecto al *warmup* de las aplicaciones SPEC CPU2006 es similar al de las SPEC CPU2017. En promedio, el IPC con *warmup* (1,67) es un 31 % más alto que sin *warmup* (1,28, y la fracción de *backend* es un 15 % más sin *warmup* (48 %) que con *warmup* (33 %).

7.2.2. Tiempo de Ejecución

La figura 7.4 presenta los tiempos de ejecución en solitario de las SPEC CPU2017 estudiadas. Podemos observar que se presentan una gran variedad de comportamientos. La figura muestra mediante las *performance stack* el porcentaje de tiempo dedicado a ejecutar instrucciones y desperdiciado por culpa del *frontend* y el *backend* del procesador. Se comprueba que estas dos últimas fracciones, especialmente la de *backend* tienen un impacto significativo en el tiempo de ejecución.

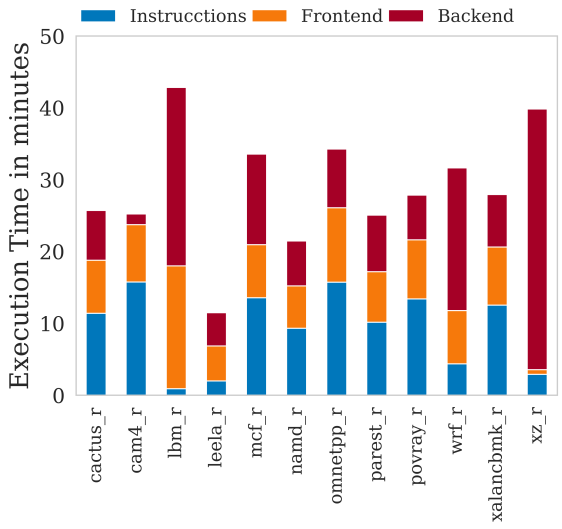

Figura 7.4: Tiempos de ejecución de las SPEC CPU2017.

SPEC	Instructions	Frontend	Backend
bzip	0.18	0.08	0.74
cactus	0.63	0.34	0.03
cactus_r	0.45	0.28	0.27
cam4_r	0.62	0.31	0.07
gcc	0.68	0.29	0.03
lbm_r	0.02	0.39	0.59
leela_r	0.17	0.42	0.41
libquantum	0.38	0.07	0.55
mcf_r	0.40	0.22	0.38
namd_r	0.43	0.28	0.29
omnetpp_r	0.45	0.31	0.24
parest_r	0.41	0.28	0.31
povray_r	0.48	0.29	0.23
sjeng	0.19	0.18	0.63
soplex	0.27	0.32	0.41
wrf_r	0.14	0.24	0.62
xalancbmk	0.62	0.33	0.05
xalancbmk_r	0.44	0.29	0.27
xz_r	0.07	0.02	0.91

Tabla 7.5: Detalle de las fracciones de categorías de las *performance stacks* para los *benchmarks* SPEC.

7.2.3. Caracterización de las aplicaciones

Aplicando el análisis Top-Down podemos clasificar los *benchmarks* en tres categorías distintas:

- **Frontend-Bound**: Agrupa aquellas aplicaciones que cuentan con un gran porcentaje de *stalls* debido al *frontend*.
- **Backend-Bound**: Agrupa aquellas aplicaciones que cuentan con un gran porcentaje de *stalls* debido al *backend*.
- **Core-Intensive**: Agrupa aquellas aplicaciones que cuentan con un gran porcentaje de ciclos procesando instrucciones.

El criterio de caracterización seguido ha sido el siguiente. Si más del 0.5 del tiempo se procesan instrucciones (lo que implica un IPC >2), se clasifica como *core-intensive*. En caso contrario, se clasifica de acuerdo a si ha tenido más porcentaje de *frontend* o de *backend*. Las tablas 7.5 y 7.6 presentan, respectivamente, los porcentajes de las categorías y la clasificación tomada por cada aplicación. Esta clasificación es será utilizada posteriormente para formar las parejas de aplicaciones a ejecutarse en modo SMT. Esta caracterización corrobora los resultados observados en otros trabajos previos, ya que van en consonancia con los obtenidos por Navarro [18] y Pons [20] en máquina real.

7.3 Ejecución en modo SMT

En esta sección se presentan los resultados obtenidos de la ejecución en SMT de la *suite* de *benchmarks* SPEC CPU. La tabla 7.7 muestra las parejas seleccionadas

Core-Intensive	Frontend-Bound	Backend-Bound
cactus	cactus_r	bzip
cam4_r	leela_r	lbm_r
gcc	omnetpp_r	libquantum
xalancbmk	povray_r	mcf_r
	xalancbmk_r	namd_r
		parest
		sjeng
		soplex
		wrf_r
		xz_r

Tabla 7.6: Clasificación de las aplicaciones en categorías.

Mezcla	Aplicaciones	Categoría
m1	cactus-cam4_r	Core-Intensive/Core-Intensive
m2	cactus_r-lbm_r	Frontend-Bound/Backend-Bound
m3	cactus_r-leela_r	Frontend-Bound/Frontend-Bound
m4	cam4_r-lbm_r	Core-Intensive/Backend-Bound
m5	cam4_r-leela_r	Core-Intensive/Frontend-Bound
m6	lbm_r-nambd_r	Backend-Bound/Backend-Bound

Tabla 7.7: Tabla con las mezclas seleccionadas para ejecución SMT.

para los experimentos. Siguiendo la caracterización en solitario se han seleccionado parejas para todas las combinaciones de categorías posibles.

La figura 7.5 presenta las *performance stacks* y el IPC para las aplicaciones cuando se ejecutan en las mezclas. Cada columna representa la ejecución de una aplicación y viene etiquetada por el nombre de la aplicación y su mezcla. La tabla 7.8 muestra el detalle de las categorias Si se comparan estos resultados con los mostrados en solitario (tabla 7.5), se observa, en general, una reducción de la categoria *instructions* y un incremento de las categorias *frontend* y, especialmente, *backend*. Estos cambios en la *stack* obedecen a las interferencias en el acceso a los recursos del núcleo causadas por la compartición entre los hilos de estos recursos en modo SMT.

Que los hilos compartan el núcleo afecta negativamente al IPC de las aplicaciones si se compara con su ejecución individual. Esto se observa en la tabla 7.9, que muestra, para cada aplicación de las mezclas, su IPC en solitario y el IPC obtenido en cada una de las mezclas. Como se observa, el IPC baja considerablemente en todas las aplicaciones y el impacto varia entre mezclas porque depende de la interferencia de la pareja en los recursos compartidos.

Aunque el IPC en solitario de cada aplicación sufre al ejecutarse en modo SMT con otra pareja, el resultado es un aumento de la productividad global. Para observar este efecto, la tabla 7.10 muestra el IPC global del núcleo obtenido en la ejecución cada una de las mezclas. Como puede observarse, los valores son relativamente altos. Sólo en aquellas mezclas donde se ejecuta `lbm_r`, la cual muestra un IPC muy bajo en solitario, se observan valores de IPC por debajo de 1,5. Estos resultados van en concordancia a lo esperado en un procesador físico y los resultados ya obtenidos en máquina real en el Grupo de Arquitecturas Paralelas.

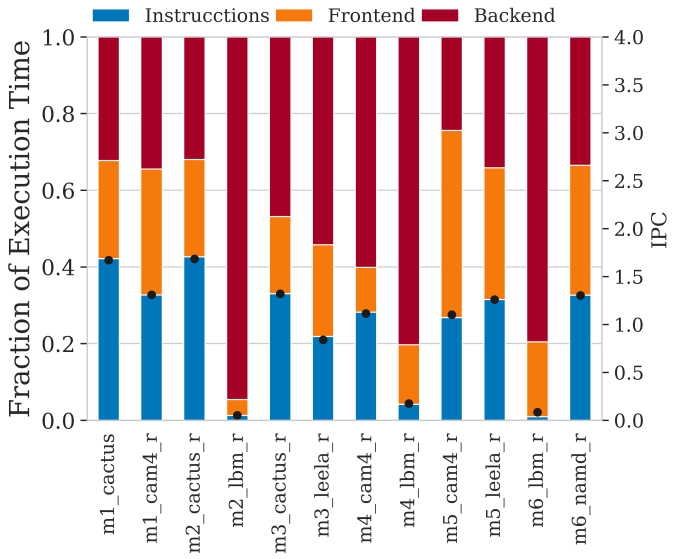

Figura 7.5: Ejecución en modo SMT por mezclas.

	Instructions	Frontend	Backend
m1_cactus	0.42	0.25	0.33
m1_cam4_r	0.30	0.27	0.43
m2_cactus_r	0.42	0.25	0.34
m2_lbm_r	0.01	0.04	0.95
m3_cactus_r	0.33	0.24	0.53
m3_leela_r	0.21	0.23	0.56
m4_cam4_r	0.28	0.11	0.61
m4_lbm_r	0.04	0.15	0.81
m5_cam4_r	0.26	0.48	0.37
m5_leela_r	0.30	0.34	0.36
m6_lbm_r	0.01	0.18	0.81
m6_namd_r	0.32	0.33	0.35

Tabla 7.8: Detalle de las categorías en modo SMT.

SPEC	IPC	IPC_m1	IPC_m2	IPC_m3	IPC_m4	IPC_m5	IPC_m6
cactus	2.49	1.72	X	X	X	X	X
cam4_r	2.49	1.43	X	X	1.18	1.19	X
cactus_r	1.90	X	1.61	1.47	X	X	X
lbm_r	0.08	X	0.04	X	0.06	X	0.03
leela_r	1.56	X	X	0.92	X	1.18	X
namd_r	1.72	X	X	X	X	X	1.42

Tabla 7.9: IPC en solitario e IPC en pareja de las aplicaciones.

Mezcla	m1	m2	m3	m4	m5	m6
IPC	3.15	1.65	2.39	1.24	2.37	1.45

Tabla 7.10: IPC global de las mezclas.

CAPÍTULO 8
Conclusiones

En este capítulo se presentan las conclusiones del trabajo realizado. También se expone su relación con los estudios cursados y trabajos futuros.

8.1 Conclusión General

El coste de producción y fabricación de procesadores, tanto coste monetario como coste en materias primas ha ido constantemente en aumento desde que comenzó esta industria. Los costes se estima que sigan creciendo a medida que los procesadores aumenten en complejidad y tamaño. La simulación busca sufragar gran parte de estos costes ofreciendo una alternativa económica a la fabricación continua de costosos prototipos vaticinando un futuro más sostenible para la informática.

En este Trabajo de Fin de Grado se ha implementado una PMU con sus correspondientes Contadores *Hardware* en uno de los principales simuladores de CPU del estado del arte. Se ha diseñado teniendo en cuenta la idiosincrasia del simulador y se ha tenido en cuenta todas las necesidades de diseño necesarias para desarrollar una buena unidad funcional que agilice y ayude al desarrollo de nuevas microarquitecturas de procesadores. Posteriormente se ha llevado a cabo un análisis Top-Down adecuándolo en todas sus categorías a las características del procesador simulado por Gem5. Para concluir, se han lanzado a ejecución una *suite* de *benchmarks* SPEC CPU para validar la implementación y se ha demostrado que los resultados son consistentes con lo esperado y comprobado en máquina real.

Este trabajo ha cumplido sus objetivos establecidos al haber desarrollado con éxito la infraestructura necesaria de monitorización de prestaciones a través de contadores simulados en una PMU y haber validado y verificado su correcta implementación. Este desarrollo se espera que pueda aportar a la comunidad una herramienta útil para analizar microarquitecturas para cualquier usuario de este simulador.

8.2 Relación con los Estudios Cursados

Para el diseño y desarrollo de la PMU propuesta en este trabajo ha sido fundamental el conocimiento de arquitecturas hardware proporcionado en asignaturas como Estructura de Computadores (ETC), Arquitectura e Ingeniería de Computadores (AIC) y Arquitecturas Avanzadas (AAV). Estas asignaturas han sentado las bases de conocimiento sobre las cuales se han asentado los análisis, causas y conclusiones de los experimentos y resultados.

Computación Paralela (CPA) introduce el sistema SLURM de planificación de procesos en un cluster, herramienta esencial para este trabajo, sistema el cual se ha empleado para lanzar trabajos en CMTS y ERI.

Fundamentos de Sistemas Operativos (FSO) introducen las bases y funcionamiento del lenguaje de programación C, Lenguajes de Programación Paralela (LPP) desarrollan tu habilidad de programar en este lenguaje. Esta habilidad ha sido esencial, ya que, aunque no se ha empleado C en este trabajo sino C++, las bases y fundamentos son iguales al C++ tratarse en sus inicios de una extensión de C.

El lenguaje Python, introducido en la asignatura Sistemas Inteligentes (SIN) y profundizado en la asignatura Seguridad en Sistemas Informáticos (SSI) ha resultado imprescindible al ser un lenguaje empleado tanto para configurar el simulador como para procesar y visualizar los resultados de los experimentos de este trabajo.

Durante el grado se estudian diversos lenguajes de programación como son Java[1], Mathematica[2] y Haskell[3] entre otros. Estos lenguajes aunque no han sido empleados en el desarrollo del trabajo ayudan a asentar las bases de buenas prácticas durante el desarrollo y simplifican la fase de adaptación a nuevos lenguajes, hecho que ha sido muy importante en este Trabajo de Fin de Grado.

En el ámbito de las competencias transversales, las que han supuesto de mayor importancia para el desarrollo de este trabajo son:

- **CT2.Innovación y Creatividad:** Para el desarrollo del trabajo ha sido crucial la capacidad de analizar fríamente problemas complejos y ser capaz de proponer soluciones creativas e innovadoras. Esta competencia me ha ayudado a plantear el diseño de la unidad y planificar su integración en un proyecto tan complejo como es el simulador Gem5.

- **CT4.Comunicación efectiva:** Esta competencia me ha ayudado a mejorar mis habilidades de redacción y ser capaz de adecuar el tono, el estilo y el vocabulario de forma correcta para la realización de esta memoria.

- **CT5.Responsabilidad y Toma de decisiones:** A lo largo de este trabajo ha sido imprescindible la capacidad autónoma de aprendizaje, tanto de nuevos lenguajes como C++14, como el estudio en profundidad del simulador.

[1] `https://www.java.com/es/`
[2] `https://www.wolfram.com/mathematica/`
[3] `https://www.haskell.org/`

La toma de decisiones me ha ayudado a abordar situaciones complejas como adaptar el análisis Top-Down a este trabajo o implementar la pila de ejecución.

8.3 Trabajos Futuros

Las PMU han evolucionado de manera significativa en los últimos años. Su mejora y dotarlas cada vez con más contadores nos permite tener un conocimiento más detallado de lo que está ocurriendo en el pipeline e identificar las causas de posibles pérdidas de prestaciones. Por este motivo, su importancia crece con el aumento continuo de la complejidad de los sistemas.

Sin embargo, realizar implementaciones en simuladores complejos es una tarea cuya curva de aprendizaje es realmente larga debido a la complejidad tanto del sistema modelado como la del propio simulador.

Los buenos resultados de la propuesta de este trabajo, la superación con éxito de la etapa de familiarización con el simulador y la importancia de mejorar las PMU, nos llevan al planteamiento de nuevos retos en este campo.

- **Aumentar la granularidad de la unidad:** Actualmente la PMU es capaz de medir de forma efectiva las categorías óptimas del análisis Top-Down para una buena aplicación del mismo. No obstante, hay categorías que se pueden refinar y dividir aún más en otras subcategorías, ya que hay aplicaciones específicas donde un análisis de grano más fino puede ser muy útil.
- **Optimizar microarquitecturas:** Actualmente con la unidad implementada esto nos permite estudiar y evaluar la efectividad de las microarquitectura para la ejecución de problemas específicos. Desarrollar y probar configuraciones específicas para optimizar aplicaciones de cálculo científico es una continuación muy natural de este trabajo.
- **Ampliar el soporte de planificación de hilos:** La unidad está pensada para funcionar con una política de planificación de hilos por round-robin, una ampliación a este trabajo sería añadirle soporte para más políticas de planificación.

Bibliografía

[1] J.Hennessy and D.Patterson, *Computer Architecture: A Quantitative Approach.* The Morgan Kaufmann Series in Computer Architecture and Design, Elsever Science, 2017.

[2] G. E. Moore, "Cramming more components onto integrated circuits," *Proceedings of the IEEE,* vol. 86, pp. 82-85, Jan 1998.

[3] R. M. Tomasulo, "An Efficient Algorithm for Exploiting Multiple Arithmetic Units," in *IBM Journal of Research and Development,* vol. 11, no. 1, pp. 25-33, Jan. 1967.

[4] N. Binkert *et al*. "The gem5 simulator." *ACM SIGARCH computer architecture news* 39.2 (2011): 1-7.

[5] D. Saito and T. Yamaura. Applying the top-down approach to beginners in programming language education, *in 2014 International Conference on Interactive Collaborative Learning (ICL)*, pp. 311-318, 2014.

[6] A. Yasin. "A top-down method for performance analysis and counter architecture". pp. 35-44, 03 2014.

[7] X. Zhang, S. Dwarkadas, G. Folkmanis and K. Shen "Processor Hardware Counter Statistics As A First-Class System Resource". *HotOS*, 2007.

[8] K. London, J. Dongarra, S. Moore, P. Mucci, K. Seymour, T. Spencer "End-user Tools for Application Performance Analysis Using Hardware Counters" *PDCS*, pp. 460-465. 2001.

[9] Seofito, "La Ley de Moore comparada con los datos históricos," *Wikipedia,* 11 Noviembre 2018. [Online] `https://es.wikipedia.org/wiki/Ley_de_Moore#/media/Archivo:Ley_de_Moore.png` Último acceso 5 de junio de 2024.

[10] Cavium, "Vulcan Block Diagram," *Wikichip,* 4 Junio 2018. [Online] `https://en.wikichip.org/wiki/cavium/microarchitectures/vulcan` Último acceso 21 de junio de 2024.

[11] D. D. Josephson, "The manic depression of microprocessor debug," *Proceedings. International Test Conference,* Baltimore, MD, USA, 2002, pp. 657-663.

[12] T. Austin, E. Larson and D. Ernst "SimpleScalar: an infrastructure for computer system modeling," *in Computer*, vol. 35, no. 2, pp. 59-67, Feb. 2002.

[13] Ghorpade, J., Parande, J., Kulkarni, M., Bawaskar, A "GPGPU processing in CUDA architecture", *arXiv preprint arXiv:1202.4347*, 2012.

[14] Li, S., Ahn, J. H., Strong, R. D., Brockman, J. B., Tullsen, D. M., Jouppi, N. P. "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proceedings of the 42nd annual ieee/acm international symposium on microarchitecture* (pp. 469-480) December 2009.

[15] Magnusson, P. S., Christensson, M., Eskilson, J., Forsgren, D., Hallberg, G., Hogberg, J., Werner, B. "Simics: A full system simulation platform" *Computer 35(2),* pp. 50-58. 2002.

[16] S. Eyerman and L. Eeckhout, "System-level performance metrics for multi-program workloads," *IEEE Micro,* vol. 28, no. 3, pp. 42–53, 2008.

[17] I. Calero. "Caracterización de prestaciones y análisis consumo energético en un procesador ARM Thuner X2," 2023.

[18] M. Navarro. "Política de asignación de aplicaciones a núcleos en un Intel Xeon utilizando la metodología Top-Down," 2020.

[19] M. A. Avargues. "Análisis de un controlador de memoria principal no volátil," 2021.

[20] L. Pons. "Mejora de prestaciones de procesadores comerciales utilizando la tecnología Intel CAT," 2018.

[21] J. Carmona. "Evaluación experimental de los mecanismos de prebúsquedaa en el ibm power8," 2017.

[22] J. Vivas. "Evaluación de la jerarquía de caché en procesadores multinúcleo," 2014.

[23] C. Catalá. "Simulación de nuevas arquitecturas de memorias caché de procesadores para sistemas empotrados," 2011.

[24] Moritz Lipp *et al*. "Meltdown", *arXiv preprint arXiv:1801.01207*, 2018.

APÉNDICE A
Archivos importantes

En este anexo se muestran archivos y ejemplos de archivos los cuales han sido imprescindibles para la realización del trabajo.

A.1 Código de la etapa *Rename*

En ese capítulo se muestran fragmentos del código desarrollado de la PMU dentro de la etapa rename.

```
    #include <fstream>
#include "cpu/o3/rename.hh"
#include <list>
#include "cpu/o3/cpu.hh"
#include "cpu/o3/dyn_inst.hh"
#include "cpu/o3/limits.hh"
#include "cpu/reg_class.hh"
#include "debug/Stalls.hh"
#include "debug/Activity.hh"
#include "debug/O3PipeView.hh"
#include "debug/Rename.hh"
#include "params/BaseO3CPU.hh"
namespace gem5
{
namespace o3
{
Rename::Rename(CPU _cpu, const BaseO3CPUParams &params)
    : cpu(_cpu),
      iewToRenameDelay(params.iewToRenameDelay),
      decodeToRenameDelay(params.decodeToRenameDelay),
      commitToRenameDelay(params.commitToRenameDelay),
      renameWidth(params.renameWidth),
      numThreads(params.numThreads),
      stats(_cpu)
{
    if (renameWidth > MaxWidth)
        fatal("renameWidth (%d) is larger than compiled limit (%
            d),\n"
              "\tincrease MaxWidth in src/cpu/o3/limits.hh\n",
```

```
29                  renameWidth, static_cast<int>(MaxWidth));
30      // @todo: Make into a parameter.
31      skidBufferMax = (decodeToRenameDelay + 1) * params.
            decodeWidth;
32      for (uint32_t tid = 0; tid < MaxThreads; tid++) {
33          renameStatus[tid] = Idle;
34          renameMap[tid] = nullptr;
35          instsInProgress[tid] = 0;
36          loadsInProgress[tid] = 0;
37          storesInProgress[tid] = 0;
38          freeEntries[tid] = {0, 0, 0, 0};
39          emptyROB[tid] = true;
40          stalls[tid] = {false, false};
41          serializeInst[tid] = nullptr;
42          serializeOnNextInst[tid] = false;
43      }
44      //Cambio
45      std::ofstream outputFile0("data_tid0.csv");
46      if (!outputFile0.is_open()) {
47          DPRINTF(Stalls, "No se ha podido crear el fichero");
48      }
49      outputFile0 << "cycle,state,squashed,physical,ROB,IQ,
            serializing,LQ,SQ,frontend,instructions,SUMA,
            stalls_frontend, stalls_backend, stalls_squashed,
            stalls_serializing, tid" << std::endl;
50      //SMT
51      std::ofstream outputFile1("data_tid1.csv");
52      if (!outputFile1.is_open()) {
53          //DPRINTF(Stalls, "No se ha podido crear el fichero");
54      }
55      outputFile1 << "cycle,state,squashed,physical,ROB,IQ,
            serializing,LQ,SQ,frontend,instructions,SUMA,
            stalls_frontend, stalls_backend, stalls_squashed,
            stalls_serializing, tid" << std::endl;
56  }
57  std::string
58  Rename::name() const
59  {
60      return cpu->name() + ".rename";
61  }
62  Rename::RenameStats::RenameStats(statistics::Group parent)
63      : statistics::Group(parent, "rename"),
64        ADD_STAT(squashCycles, statistics::units::Cycle::get(),
65                 "Number of cycles rename is squashing"),
66        ADD_STAT(idleCycles, statistics::units::Cycle::get(),
67                 "Number of cycles rename is idle"),
68        ADD_STAT(blockCycles, statistics::units::Cycle::get(),
69                 "Number of cycles rename is blocking"),
70        ADD_STAT(serializeStallCycles, statistics::units::Cycle::
              get(),
71                 "count of cycles rename stalled for serializing
                     inst"),
```

```
72          ADD_STAT(runCycles, statistics::units::Cycle::get(),
73                   "Number of cycles rename is running"),
74          ADD_STAT(unblockCycles, statistics::units::Cycle::get(),
75                   "Number of cycles rename is unblocking"),
76          ADD_STAT(renamedInsts, statistics::units::Count::get(),
77                   "Number of instructions processed by rename"),
78          ADD_STAT(squashedInsts, statistics::units::Count::get(),
79                  "Number of squashed instructions processed by
                      rename"),
80          ADD_STAT(ROBFullEvents, statistics::units::Count::get(),
81                   "Number of times rename has blocked due to ROB
                       full"),
82          ADD_STAT(IQFullEvents, statistics::units::Count::get(),
83                   "Number of times rename has blocked due to IQ
                       full"),
84          ADD_STAT(LQFullEvents, statistics::units::Count::get(),
85                   "Number of times rename has blocked due to LQ
                       full" ),
86          ADD_STAT(SQFullEvents, statistics::units::Count::get(),
87                   "Number of times rename has blocked due to SQ
                       full"),
88          ADD_STAT(fullRegistersEvents, statistics::units::Count::
                get(),
89                   "Number of times there has been no free registers
                       "),
90          ADD_STAT(renamedOperands, statistics::units::Count::get(),
91                   "Number of destination operands rename has
                       renamed"),
92          ADD_STAT(lookups, statistics::units::Count::get(),
93                   "Number of register rename lookups that rename
                       has made"),
94          ADD_STAT(intLookups, statistics::units::Count::get(),
95                   "Number of integer rename lookups"),
96          ADD_STAT(fpLookups, statistics::units::Count::get(),
97                   "Number of floating rename lookups"),
98          ADD_STAT(vecLookups, statistics::units::Count::get(),
99                   "Number of vector rename lookups"),
100         ADD_STAT(vecPredLookups, statistics::units::Count::get(),
101                  "Number of vector predicate rename lookups"),
102         ADD_STAT(matLookups, statistics::units::Count::get(),
103                  "Number of matrix rename lookups"),
104         ADD_STAT(committedMaps, statistics::units::Count::get(),
105                  "Number of HB maps that are committed"),
106         ADD_STAT(undoneMaps, statistics::units::Count::get(),
107                  "Number of HB maps that are undone due to
                      squashing"),
108         ADD_STAT(serializing, statistics::units::Count::get(),
109                  "count of serializing insts renamed"),
110         ADD_STAT(tempSerializing, statistics::units::Count::get(),
111                  "count of temporary serializing insts renamed"),
112         ADD_STAT(skidInsts, statistics::units::Count::get(),
113                  "count of insts added to the skid buffer")
```

```
114         //Cambio
115         ADD_STAT(stalls_frontend, statistics::units::Count::get(),
116                  "count of insts added to the skid buffer"),
117         ADD_STAT(stalls_backend, statistics::units::Count::get(),
118                  "count of insts added to the skid buffer"),
119         ADD_STAT(stalls_totales, statistics::units::Count::get(),
120                  "count of insts added to the skid buffer"),
121         ADD_STAT(rename_count, statistics::units::Count::get(),
122                  "count of insts added to the skid buffer"),
123         ADD_STAT(renameInst_count, statistics::units::Count::get()
                ,
124                  "count of insts added to the skid buffer"),
125         ADD_STAT(renameTick_count, statistics::units::Count::get()
                ,
126                  "count of insts added to the skid buffer")
127 {
128 ...
129 ...
```

Figura A.1: Fragmento de inicialización de rename.cc

```
1  Rename::tick()
2  {
3      //Cambio
4      DPRINTF(Stalls, "CONTROL: veces que se ha llamado a Rename::
           tick = %i \n",stats.renameTick_count.value());
5      wroteToTimeBuffer = false;
6      blockThisCycle = false;
7      bool status_change = false;
8      toIEWIndex = 0;
9      sortInsts();
10     std::list<ThreadID>::iterator threads = activeThreads->begin
           ();
11     std::list<ThreadID>::iterator end = activeThreads->end();
12     // Check stall and squash signals.
13     while (threads != end) {
14         ThreadID tid = threads++;
15         DPRINTF(Rename, "Processing [tid:%i]\n", tid);
16         status_change = checkSignalsAndUpdate(tid) ||
               status_change;
17         rename(status_change, tid);
18     }
19     if (status_change) {
20         updateStatus();
21     }
22     if (wroteToTimeBuffer) {
23         DPRINTF(Activity, "Activity this cycle.\n");
24         cpu->activityThisCycle();
25     }
26     threads = activeThreads->begin();
27     while (threads != end) {
28         ThreadID tid = threads++;
```

```
29          // If we committed this cycle then doneSeqNum will be >
                0
30          if (fromCommit->commitInfo[tid].doneSeqNum != 0 &&
31              !fromCommit->commitInfo[tid].squash &&
32              renameStatus[tid] != Squashing) {
33              removeFromHistory(fromCommit->commitInfo[tid].
                    doneSeqNum,
34
35                                      tid);
36          }
37      }
38      // @todo: make into updateProgress function
39      for (ThreadID tid = 0; tid < numThreads; tid++) {
40          instsInProgress[tid] -= fromIEW->iewInfo[tid].dispatched
                ;
41          loadsInProgress[tid] -= fromIEW->iewInfo[tid].
                dispatchedToLQ;
42          storesInProgress[tid] -= fromIEW->iewInfo[tid].
                dispatchedToSQ;
43          assert(loadsInProgress[tid] >= 0);
44          assert(storesInProgress[tid] >= 0);
45          assert(instsInProgress[tid] >=0);
46      }
47  }
48  //Cambio
49  //Variables globales para contar los ciclos de parada
50  int stalls_totales = 0;
51  int stalls_frontend = 0;
52  int stalls_backend = 0;
53  int stalls_squashed = 0;
54  int stalls_serializing = 0;
55  int control;
56  int total_inst = 0;
57  int status = 0;
58  int Squashed_inst = 0;
59  int Phys_inst = 0;
60  int ROB_inst = 0;
61  int IQ_inst = 0;
62  int Serializing_inst = 0;
63  int LQ_inst = 0;
64  int SQ_inst = 0;
65  int Frontend_inst = 0;
66  int Inst_exec = 0;
67  int Suma = 0;
68  void
69  Rename::rename(bool &status_change, ThreadID tid)
70  {
71      //Cambio
72      int insts_available_rename = renameStatus[tid] == Unblocking
            ?
73          skidBuffer[tid].size() : insts[tid].size();
74      std::ofstream outputFile1("data_tid1.csv", std::ios::app);
```

```
75      std::ofstream outputFile0("data_tid0.csv", std::ios::app);
76      // If status is Running or idle,
77      //     call renameInsts()
78      // If status is Unblocking,
79      //     buffer any instructions coming from decode
80      //     continue trying to empty skid buffer
81      //     check if stall conditions have passed
82      //Cambio
83      ++stats.rename_count;
84      DPRINTF(Stalls, "-----------------------------------\n");
85      DPRINTF(Stalls, "Rename %i \n", renameStatus[tid]);
86      if (renameStatus[tid] == Blocked) {
87          ++stats.blockCycles;
88        //Cambio
89          status = 4;
90          stalls_backend = stalls_backend + renameWidth;
91
92    if (calcFreeROBEntries(tid) <= 0 && Inst_exec == 0) {
93              ROB_inst = ROB_inst + renameWidth;
94          } else if (calcFreeIQEntries(tid) <= 0 && Inst_exec ==
              0) {
95              IQ_inst = IQ_inst + renameWidth;
96    } else if (insts_available_rename == 0) {
97        Frontend_inst = Frontend_inst + renameWidth;
98    }
99      } else if (renameStatus[tid] == Squashing) {
100         ++stats.squashCycles;
101       //Cambio
102         status = 3;
103       stats.stalls_backend += renameWidth;
104       stalls_backend = stalls_backend + renameWidth;
105         Squashed_inst = Squashed_inst + renameWidth;
106         DPRINTF(Stalls, "Rename Squashing \n");
107     } else if (renameStatus[tid] == SerializeStall) {
108         ++stats.serializeStallCycles;
109         //Cambio
110         status = 6;
111       stats.stalls_backend += renameWidth;
112       stalls_backend = stalls_backend + renameWidth;
113       serialize_stalls = renameWidth;
114         Serializing_inst = Serializing_inst + renameWidth;
115         DPRINTF(Stalls, "Rename Serializig \n");
116       // If we are currently in SerializeStall and
            resumeSerialize
117         // was set, then that means that we are resuming
            serializing
118         // this cycle.  Tell the previous stages to block.
119
120   if (resumeSerialize) {
121             resumeSerialize = false;
122             block(tid);
123             toDecode->renameUnblock[tid] = false;
```

```
124         }
125     } else if (renameStatus[tid] == Unblocking) {
126         if (resumeUnblocking) {
127             block(tid);
128             resumeUnblocking = false;
129             toDecode->renameUnblock[tid] = false;
130       }
131     }
132     if (renameStatus[tid] == Running ||
133         renameStatus[tid] == Idle) {
134         DPRINTF(Rename,
135                 "[tid:%i] "
136                 "Not blocked, so attempting to run stage.\n",
137                 tid);
138         renameInsts(tid);
139     } else if (renameStatus[tid] == Unblocking) {
140         renameInsts(tid);
141         if (validInsts()) {
142             // Add the current inputs to the skid buffer so they
                    can be
143             // reprocessed when this stage unblocks.
144             skidInsert(tid);
145         }
146         // If we switched over to blocking, then there's a
               potential for
147         // an overall status change.
148         status_change = unblock(tid) || status_change ||
               blockThisCycle;
149     }
150     //Cambio
151     DPRINTF(Stalls, "CONTROL: ciclos  Squashing = %i\n",stats.
           squashCycles.value());
152     DPRINTF(Stalls, "CONTROL: ciclos  Idle = %i\n",stats.
           idleCycles.value());
153     DPRINTF(Stalls, "CONTROL: ciclos  Bloqueado = %i\n",stats.
           blockCycles.value());
154     DPRINTF(Stalls, "CONTROL: ciclos  Serializing = %i\n",stats.
           serializeStallCycles.value());
155     DPRINTF(Stalls, "CONTROL: ciclos  Running = %i\n",stats.
           runCycles.value());
156     DPRINTF(Stalls, "CONTROL: ciclos  Unblock = %i\n",stats.
           unblockCycles.value());
157     //Cambio
158     //Print de debug para saber los stalls totales, de frontend
           y backend
159     stalls_totales = stalls_frontend + stalls_backend;
160     DPRINTF(Stalls, "Stalls back tot\t\tStalls front tot\tStalls
            totales\n");
161     DPRINTF(Stalls, "\t%i\t\t\t%i\t\t\t%i\n",
162   stalls_backend, stalls_frontend, stalls_totales);
163     ROB_inst  = calcFreeROBEntries(tid);
164     IQ_inst  = calcFreeIQEntries(tid);
```

```
165   //Cambio
166     stalls_squashed = Squashed_inst + stalls_squashed;
167     stalls_serializing = stalls_serializing + Serializing_inst;
168     stalls_backend = stalls_backend + Phys_inst + ROB_inst +
          IQ_inst +  LQ_inst + SQ_inst;
169     stalls_frontend = stalls_frontend + Frontend_inst;
170     stalls_totales = stalls_backend + stalls_frontend +
          stalls_squashed + stalls_serializing;
171     DPRINTF(Stalls, "Stalls totales = %i\n",stalls_totales);
172     Suma = Squashed_inst + Phys_inst + ROB_inst + IQ_inst +
          Serializing_inst + LQ_inst + SQ_inst + Frontend_inst +
          Inst_exec;
173     total_inst = total_inst + Inst_exec;
174     if(total_inst>2000000){
175     //if(true){
176     if(tid){
177         outputFile1 << cpu->baseStats.numCycles.value()<< ","<<
              status <<","<< Squashed_inst << ","<<Phys_inst <<","
              <<ROB_inst<<"," <<IQ_inst<<"," <<Serializing_inst<<",
              " <<LQ_inst<<","<<SQ_inst<<"," <<Frontend_inst<<","<<
               Inst_exec << "," <<Suma << stalls_frontend << "," <<
               stalls_backend << "," << stalls_squashed <<"," <<
              stalls_serializing <<"," << tid << std::endl;
178     outputFile1.close();
179     } else {
180         outputFile0 << cpu->baseStats.numCycles.value()<< ","<<
              status <<","<< Squashed_inst << ","<<Phys_inst <<","
              <<ROB_inst<<"," <<IQ_inst<<"," <<Serializing_inst<<",
              " <<LQ_inst<<","<<SQ_inst<<"," <<Frontend_inst<<","<<
               Inst_exec << "," <<Suma << stalls_frontend << "," <<
               stalls_backend << "," << stalls_squashed <<"," <<
              stalls_serializing <<"," << tid << std::endl;
181     outputFile0.close();
182 }
183 }
184     Squashed_inst = 0;
185     Phys_inst = 0;
186     ROB_inst = 0;
187     IQ_inst = 0;
188     Serializing_inst = 0;
189     LQ_inst = 0;
190     SQ_inst = 0;
191     Frontend_inst = 0;
192     Inst_exec = 0;
193     Suma = 0;
194 }
195
196 void
197
198 Rename::renameInsts(ThreadID tid)
199 {
200     //Cambio
```

```
201
202     ++stats.renameInst_count;
203     Squashed_inst = 0;
204     Phys_inst = 0;
205     ROB_inst = 0;
206     IQ_inst = 0;
207     Serializing_inst = 0;
208     LQ_inst = 0;
209     SQ_inst = 0;
210     Frontend_inst = 0;
211     Inst_exec = 0;
212     Suma = 0;
213     // Instructions can be either in the skid buffer or the
            queue of
214     // instructions coming from decode, depending on the status.
215     int insts_available = renameStatus[tid] == Unblocking ?
216         skidBuffer[tid].size() : insts[tid].size();
217
218     // Check the decode queue to see if instructions are
            available.
219     // If there are no available instructions to rename, then do
            nothing.
220     if (insts_available == 0) {
221         DPRINTF(Rename, "[tid:%i] Nothing to do, breakin out
                early.\n",
222                 tid);
223         // Should I change status to idle?
224         // Cambio
225         status = 0;
226         DPRINTF(Stalls, "Rename Idle\n");
227         ++stats.idleCycles;
228         //Cambio:
229         //NO HAY INSTRUCCIONES DISPONIBLES
230         //STALL N FRONTEND
231         //stats.stalls_frontend += renameWidth;
232         stalls_frontend = stalls_frontend + renameWidth;
233         idle_stalls = renameWidth;
234         Frontend_inst = Frontend_inst + renameWidth;
235         return;
236     } else if (renameStatus[tid] == Unblocking) {
237         //Cambio
238         DPRINTF(Stalls, "Rename Unblock\n");
239         ++stats.unblockCycles;
240         status = 5;
241     } else if (renameStatus[tid] == Running) {
242         //Cambio
243         //DPRINTF(Stalls, "Rename Running \n");
244         ++stats.runCycles;
245         status = 1;
246     }
247     // Will have to do a different calculation for the numberof
            free
```

```
248     // entries.
249     int free_rob_entries = calcFreeROBEntries(tid);
250     int free_iq_entries  = calcFreeIQEntries(tid);
251     int min_free_entries = free_rob_entries;
252     FullSource source = ROB;
253     if (free_iq_entries < min_free_entries) {
254         min_free_entries = free_iq_entries;
255         source = IQ;
256     }
257     // Check if there's any space left.
258     if (min_free_entries <= 0) {
259         DPRINTF(Rename,
260           "[tid:%i] Blocking due to no free ROB/IQ/ entries.\n"
261           "ROB has %i free entries.\n"
262           "IQ has %i free entries.\n",
263                 tid, free_rob_entries, free_iq_entries);
264         blockThisCycle = true;
265         block(tid);
266         incrFullStat(source);
267         //Cambio
268         //NO HAY ENTRADAS DISPONIBLES EN EL ROB/IQ
269         //STALL N BACKEND
270         stalls_backend = stalls_backend + renameWidth;
271         if (calcFreeROBEntries(tid) == 0){
272             ROB_inst = ROB_inst + renameWidth;
273         } else if (calcFreeIQEntries(tid) == 0) {
274             IQ_inst = IQ_inst + renameWidth;
275         }
276         stats.stalls_backend += renameWidth;
277         return;
278     } else if (min_free_entries < insts_available) {
279         DPRINTF(Rename,
280           "[tid:%i] "
281           "Will have to block this cycle. "
282           "%i insts available, "
283           "but only %i insts can be renamed due to ROB/IQ/LSQ
                limits.\n",
284           tid, insts_available, min_free_entries);
285         insts_available = min_free_entries;
286         blockThisCycle = true;
287         incrFullStat(source);
288     }
289     InstQueue &insts_to_rename = renameStatus[tid] == Unblocking
          ?
290     skidBuffer[tid] : insts[tid];
291     DPRINTF(Rename,
292         "[tid:%i] "
293         "%i available instructions to send iew.\n",
294         tid, insts_available);
295     DPRINTF(Rename,
296         "[tid:%i] "
297         "%i insts pipelining from Rename | "
```

```
298         "%i insts dispatched to IQ last cycle.\n",
299         tid, instsInProgress[tid], fromIEW->iewInfo[tid].
               dispatched);
300
301 // Handle serializing the next instruction if necessary.
302     if (serializeOnNextInst[tid]) {
303         if (emptyROB[tid] && instsInProgress[tid] == 0) {
304             // ROB already empty; no need to serialize.
305             serializeOnNextInst[tid] = false;
306         } else if (!insts_to_rename.empty()) {
307             insts_to_rename.front()->setSerializeBefore();
308         }
309     }
310 int renamed_insts = 0;
311
312 //Cambio
313
314 control = -1;
315 while (insts_available > 0 &&  toIEWIndex < renameWidth) {
316 DPRINTF(Rename, "[tid:%i] Sending instructions to IEW.\n", tid);
317
318 //Cambio
319 //print para saber el ancho de renombre
320 DPRINTF(Rename, "Rename Width %i\n",renameWidth);
321 assert(!insts_to_rename.empty());
322 DynInstPtr inst = insts_to_rename.front();
323
324 //For all kind of instructions, check ROB and IQ first For load
325 //instruction, check LQ size and take into account the inflight
       loads
326 //For store instruction, check SQ size and take into account the
327 //inflight stores
328 //
329 //Cambio
330 contro + 1;
331
332 if (inst->isLoad()) {
333     if (calcFreeLQEntries(tid) <= 0) {
334   DPRINTF(Rename, "[tid:%i] Cannot rename due to no free LQ\n",
335     tid);
336   source = LQ;
337   incrFullStat(source);
338   //Cambio
339   //NO HAY ESPACIO DISPONIBLE EN LA COLA DE LOADS
340   //STALL N BACKEND
341   stats.stalls_backend += (renameWidth - control);
342   stalls_backend = stalls_backend + (renameWidth - control);
343   LQ_inst = LQ_inst + (renameWidth - control);
344   break;
345     }
346 }
347 if (inst->isStore() || inst->isAtomic()) {
```

```
348     if (calcFreeSQEntries(tid) <= 0) {
349   DPRINTF(Rename, "[tid:%i] Cannot rename due to no free SQ\n",
350     tid);
351   source = SQ;
352   incrFullStat(source);
353   //Cambio
354   //NO HAY ESPACIO DISPONIBLE EN LA COLA DE STORES
355   //STALL N BACKEND
356   stats.stalls_backend += (renameWidth - control);
357   stalls_backend = stalls_backend + (renameWidth - control);
358   SQ_inst = SQ_inst + (renameWidth - control);
359   break;
360     }
361 }
362 insts_to_rename.pop_front();
363
364 if (renameStatus[tid] == Unblocking) {
365     DPRINTF(Rename,
366       "[tid:%i] "
367       "Removing [sn:%llu] PC:%s from rename skidBuffer\n",
368       tid, inst->seqNum, inst->pcState());
369 }
370 if (inst->isSquashed()) {
371     DPRINTF(Rename,
372       "[tid:%i] "
373       "instruction %i with PC %s is squashed, skipping.\n",
374       tid, inst->seqNum, inst->pcState());
375     ++stats.squashedInsts;
376     //Cambio
377     Squashed_inst = Squashed_inst + 1;
378     // Decrement how many instructions are available.
379     --insts_available;
380     continue;
381 }
382
383 DPRINTF(Rename,
384   "[tid:%i] "
385   "Processing instruction [sn:%llu] with PC %s.\n",
386   tid, inst->seqNum, inst->pcState());
387 // Check here to make sure there are enough destination
    registers
388 // to rename to.  Otherwise block.
389
390 if (!renameMap[tid]->canRename(inst)) {
391     DPRINTF(Rename,
392       "Blocking due to "
393       " lack of free physical registers to rename to.\n");
394     blockThisCycle = true;
395     insts_to_rename.push_front(inst);
396     ++stats.fullRegistersEvents;
397     //Cambio
398     //STALL N BACKEND
```

```
399     stats.stalls_backend += (renameWidth - control);
400     stalls_backend = stalls_backend + (renameWidth - control);
401     Phys_inst = Phys_inst + (renameWidth - control);
402     break;
403 }
404 // Handle serializeAfter/serializeBefore instructions.
405 // serializeAfter marks the next instruction as serializeBefore.
406 // serializeBefore makes the instruction wait in rename until
    the ROB
407 // is empty.
408 // In this model, IPR accesses are serialize before
409  // instructions, and store conditionals are serialize after
410  // instructions.  This is mainly due to lack of support for
411 // out-of-order operations of either of those classes of
412 //instructions.
413 if (inst->isSerializeBefore() && !inst->isSerializeHandled()) {
414
415     DPRINTF(Rename, "Serialize before instruction encountered.\n
        ");
416
417       if (!inst->isTempSerializeBefore()) {
418       stats.serializing++;
419       inst->setSerializeHandled();
420
421       } else {
422       stats.tempSerializing++;
423       }
424       // Change status over to SerializeStall so that other
          stages know
425
426       // what this is blocked on.
427       renameStatus[tid] = SerializeStall;
428       serializeInst[tid] = inst;
429       blockThisCycle = true;
430
431       //Cambio
432       //STALL
433       stats.stalls_backend += (renameWidth - control);
434       stalls_backend = stalls_backend + (renameWidth - control
          );
435       Serializing_inst = Serializing_inst + (renameWidth -
          control);
436       break;
437
438       } else if ((inst->isStoreConditional() || inst->
          isSerializeAfter()) &&
439          !inst->isSerializeHandled()) {
440        DPRINTF(Rename, "Serialize after instruction
            encountered.\n");
441        stats.serializing++;
442        inst->setSerializeHandled();
443        serializeAfter(insts_to_rename, tid);
```

```
444         }
445
446         renameSrcRegs(inst, inst->threadNumber);
447         renameDestRegs(inst, inst->threadNumber);
448
449         if (inst->isAtomic() || inst->isStore()) {
450             storesInProgress[tid]++;
451         } else if (inst->isLoad()) {
452             loadsInProgress[tid]++;
453         }
454         ++renamed_insts;
455         //Cambio
456         ++Inst_exec;
457         // Notify potential listeners that source and
               destination registers for
458         // this instruction have been renamed.
459         ppRename->notify(inst);
460         // Put instruction in rename queue.
461         toIEW->insts[toIEWIndex] = inst;
462         ++(toIEW->size);
463         // Increment which instruction we're on.
464         ++toIEWIndex;
465         // Decrement how many instructions are available.
466         --insts_available;
467     } //Fin while
468     //Cambio
469     if (!insts_available) {
470         stats.stalls_frontend += (renameWidth - control);
471         Frontend_inst = Frontend_inst + (renameWidth - control)
               -1;
472         stalls_frontend = stalls_frontend + (renameWidth -
               control) - 1;
473     }
474
475     instsInProgress[tid] += renamed_insts;
476     stats.renamedInsts += renamed_insts;
477
478     // If we wrote to the time buffer, record this.
479     if (toIEWIndex) {
480         wroteToTimeBuffer = true;
481     }
482     // Check if there's any instructions left that haven't yet
           been renamed.
483
484     // If so then block.
485     if (insts_available) {
486         blockThisCycle = true;
487     }
488
489     if (blockThisCycle) {
490         block(tid);
491         toDecode->renameUnblock[tid] = false;
```

```
492     }
493     //Cambio
494     DPRINTF(Stalls, "Squashed\tPhysical\tROB\tIQ\tSerializing\
            tLQ\tSQ\tFrontend\tInstructions\n");
495
496     DPRINTF(Stalls, "%i\t\t%i\t\t%i\t%i\t%i\t\t%i\t%i\t%i\t\t%i\
            n",    Squashed_inst,Phys_inst,ROB_inst,IQ_inst,
            Serializing_inst,LQ_inst,SQ_inst,Frontend_inst, Inst_exec
             );
497 }
```

Figura A.2: Métodos desarrollados de rename.cc

A.2 *Scripts*

A.2.1. Configuración de Gem5

Se muestra en la figura A.3 el fichero de configuración BaseO3CPU.py desarrollado para configurar el simulador según las necesidades del trabajo.

```
1  from m5.defines import buildEnv
2  from m5.params import all
3  from m5.proxy import all
4  from m5.objects.BaseCPU import BaseCPU
5  from m5.objects.FUPool import all
6  # from m5.objects.O3Checker import O3Checker
7  from m5.objects.BranchPredictor import all
8
9  class SMTFetchPolicy(ScopedEnum):
10     vals = ["RoundRobin", "Branch", "IQCount", "LSQCount"]
11
12 class SMTQueuePolicy(ScopedEnum):
13     vals = ["Dynamic", "Partitioned", "Threshold"]
14
15 class CommitPolicy(ScopedEnum):
16     vals = ["RoundRobin", "OldestReady"]
17
18 class BaseO3CPU(BaseCPU):
19
20     type = "BaseO3CPU"
21     cxx_class = "gem5::o3::CPU"
22     cxx_header = "cpu/o3/dyn_inst.hh"
23
24     @classmethod
25
26     def memory_mode(cls):
27         return "timing"
28
29     @classmethod
30
```

```
31      def require_caches(cls):
32          return True
33
34      @classmethod
35
36      def support_take_over(cls):
37          return True
38
39      activity = Param.Unsigned(0, "Initial count")
40
41      cacheStorePorts = Param.Unsigned(
42          200, "Cache Ports. Constrains stores only."
43      )
44
45      cacheLoadPorts = Param.Unsigned(200, "Cache Ports.
            Constrains loads only.")
46      decodeToFetchDelay = Param.Cycles(1, "Decode to fetch delay"
            )
47      renameToFetchDelay = Param.Cycles(1, "Rename to fetch delay"
            )
48      iewToFetchDelay = Param.Cycles(1, "Issue/Execute/Writeback
            to fetch delay")
49      commitToFetchDelay = Param.Cycles(1, "Commit to fetch delay"
            )
50      fetchWidth = Param.Unsigned(4, "Fetch width")
51      fetchBufferSize = Param.Unsigned(64, "Fetch buffer size in
            bytes")
52
53      fetchQueueSize = Param.Unsigned(
54          32, "Fetch queue size in micro-ops per-thread"
55      )
56
57      renameToDecodeDelay = Param.Cycles(1, "Rename to decode
            delay")
58      iewToDecodeDelay = Param.Cycles(
59          1, "Issue/Execute/Writeback to decode delay"
60      )
61
62      commitToDecodeDelay = Param.Cycles(1, "Commit to decode
            delay")
63      fetchToDecodeDelay = Param.Cycles(1, "Fetch to decode delay"
            )
64      decodeWidth = Param.Unsigned(4, "Decode width")
65
66      iewToRenameDelay = Param.Cycles(
67          1, "Issue/Execute/Writeback to rename delay"
68      )
69
70      commitToRenameDelay = Param.Cycles(1, "Commit to rename
            delay")
71      decodeToRenameDelay = Param.Cycles(1, "Decode to rename
            delay")
```

```
72      renameWidth = Param.Unsigned(4, "Rename width")
73
74      commitToIEWDelay = Param.Cycles(
75          1, "Commit to Issue/Execute/Writeback delay"
76      )
77
78      renameToIEWDelay = Param.Cycles(
79          2, "Rename to Issue/Execute/Writeback delay"
80      )
81      issueToExecuteDelay = Param.Cycles(
82          1, "Issue to execute delay (internal to the IEW stage)"
83      )
84
85      dispatchWidth = Param.Unsigned(6, "Dispatch width")
86      issueWidth = Param.Unsigned(4, "Issue width")
87      wbWidth = Param.Unsigned(4, "Writeback width")
88
89      fuPool = Param.FUPool(DefaultFUPool(), "Functional Unit pool
            ")
90
91      iewToCommitDelay = Param.Cycles(
92          1, "Issue/Execute/Writeback to commit delay"
93      )
94
95      renameToROBDelay = Param.Cycles(1, "Rename to reorder buffer
            delay")
96      commitWidth = Param.Unsigned(4, "Commit width")
97      squashWidth = Param.Unsigned(4, "Squash width")
98      trapLatency = Param.Cycles(13, "Trap latency")
99      fetchTrapLatency = Param.Cycles(1, "Fetch trap latency")
100
101     backComSize = Param.Unsigned(
102         5, "Time buffer size for backwards communication"
103     )
104
105     forwardComSize = Param.Unsigned(
106         5, "Time buffer size for forward communication"
107     )
108
109     LQEntries = Param.Unsigned(64, "Number of load queue entries
            ")
110
111     SQEntries = Param.Unsigned(36, "Number of store queue
            entries")
112
113     LSQDepCheckShift = Param.Unsigned(
114         4, "Number of places to shift addr before check"
115     )
116
117     LSQCheckLoads = Param.Bool(
118         True,
119         "Should dependency violations be checked for "
```

```
120             "loads & stores or just stores",
121         )
122
123         store_set_clear_period = Param.Unsigned(
124             250000,
125             "Number of load/store insts before the dep predictor "
126             "should be invalidated",
127         )
128
129         LFSTSize = Param.Unsigned(1024, "Last fetched store table
                size")
130
131         SSITSize = Param.Unsigned(1024, "Store set ID table size")
132
133         numRobs = Param.Unsigned(1, "Number of Reorder Buffers")
134
135         numPhysIntRegs = Param.Unsigned(
136             256, "Number of physical integer registers"
137         )
138
139         numPhysFloatRegs = Param.Unsigned(
140             256, "Number of physical floating point registers"
141         )
142
143         numPhysVecRegs = Param.Unsigned(256, "Number of physical
                vector registers")
144
145         numPhysVecPredRegs = Param.Unsigned(
146             32, "Number of physical predicate registers"
147         )
148
149         numPhysMatRegs = Param.Unsigned(2, "Number of physical
                matrix registers")
150
151         # most ISAs don't use condition-code regs, so default is 0
152
153         numPhysCCRegs = Param.Unsigned(0, "Number of physical cc
                registers")
154
155         numIQEntries = Param.Unsigned(60, "Number of instruction
                queue entries")
156
157         numROBEntries = Param.Unsigned(180, "Number of reorder
                buffer entries")
158
159         smtNumFetchingThreads = Param.Unsigned(1, "SMT Number of
                Fetching Threads")
160
161         smtFetchPolicy = Param.SMTFetchPolicy("RoundRobin", "SMT
                Fetch policy")
162
163         smtLSQPolicy = Param.SMTQueuePolicy(
```

```
164             "Partitioned", "SMT LSQ Sharing Policy"
165         )
166
167         smtLSQThreshold = Param.Int(100, "SMT LSQ Threshold Sharing
                Parameter")
168
169         smtIQPolicy = Param.SMTQueuePolicy("Partitioned", "SMT IQ
                Sharing Policy")
170
171         smtIQThreshold = Param.Int(100, "SMT IQ Threshold Sharing
                Parameter")
172
173         smtROBPolicy = Param.SMTQueuePolicy(
174             "Partitioned", "SMT ROB Sharing Policy"
175         )
176         smtROBThreshold = Param.Int(100, "SMT ROB Threshold Sharing
                Parameter")
177
178         smtCommitPolicy = Param.CommitPolicy("RoundRobin", "SMT
                Commit Policy")
179
180         branchPred = Param.BranchPredictor(
181             TournamentBP(numThreads=Parent.numThreads), "Branch
                    Predictor"
182         )
183
184         needsTSO = Param.Bool(False, "Enable TSO Memory model")
```

Figura A.3: Spec2017.sh

A.2.2. Ficheros ejemplo de SLURM

En la figura A.4 se muestra un fichero de ejemplo de SLURM para lanzar a ejecución las SPEC 2017, y en la figura A.5 un fichero de ejemplo para lanzar las SPEC 2006.

```
#!/bin/bash
#SBATCH --job-name=specs2017
#SBATCH --nnodes=1
#SBACTH --ntasks=24
#SBATCH --cpus-per-task=1

../build/X86/gem5.opt ../configs/deprecated/example/se.py --cpu-
    type=O3CPU --l1d_size=32kB --l1i_size=32kB --caches --l2_size
    =256kB --l2cache --l1d_assoc=8 --l1i_assoc=8 --l2_assoc=8 --
    maxinsts=400000000 --cmd=/mnt/beegfs/gap/mapecfer/spec2017/
    benchspec/CPU/549.fotonik3d_r/exe/fotonik3d_r_base.
    InitialTest-m64

mv data.csv ../datos/fotonik.csv

../build/X86/gem5.opt ../configs/deprecated/example/se.py --cpu-
    type=O3CPU --l1d_size=32kB --l1i_size=32kB --caches --l2_size
    =256kB --l2cache --l1d_assoc=8 --l1i_assoc=8 --l2_assoc=8 --
    maxinsts=400000000 --cmd=/mnt/beegfs/gap/mapecfer/spec2017/
    benchspec/CPU/548.exchange2_r/exe/exchange2_r_base.
    InitialTest-m64 --options="6"

mv data.csv ../datos/exchange2.csv

../build/X86/gem5.opt ../configs/deprecated/example/se.py --cpu-
    type=O3CPU --l1d_size=32kB --l1i_size=32kB --caches --l2_size
    =256kB --l2cache --l1d_assoc=8 --l1i_assoc=8 --l2_assoc=8 --
    maxinsts=1000000 --cmd=/mnt/beegfs/gap/mapecfer/spec2017/
    benchspec/CPU/527.cam4_r/exe/cam4_r_base.InitialTest-m64

mv data.csv ../datos/cam4.csv

../build/X86/gem5.opt ../configs/deprecated/example/se.py --cpu-
    type=O3CPU --l1d_size=32kB --l1i_size=32kB --caches --l2_size
    =256kB --l2cache --l1d_assoc=8 --l1i_assoc=8 --l2_assoc=8 --
    maxinsts=1000000 --cmd=/mnt/beegfs/gap/mapecfer/spec2017/
    benchspec/CPU/521.wrf_r/exe/wrf_r_base.InitialTest-m64

mv data.csv ../datos/wrf.csv

../build/X86/gem5.opt ../configs/deprecated/example/se.py --cpu-
    type=O3CPU --l1d_size=32kB --l1i_size=32kB --caches --l2_size
    =256kB --l2cache --l1d_assoc=8 --l1i_assoc=8 --l2_assoc=8 --
    maxinsts=1000000 --cmd=/mnt/beegfs/gap/mapecfer/spec2017/
    benchspec/CPU/502.gcc_r/exe/cpugcc_r_base.InitialTest-m64 --
    options="gcc-pp.c -O3 -finline-limit=0 -fif-conversion -fif-
```

```
    conversion2 -o gcc-pp.opts-O3_-finline-limit_0_-fif-
    conversion_-fif-conversion2.s"

mv data.csv ../datos/gcc.csv

../build/X86/gem5.opt ../configs/deprecated/example/se.py --cpu-
    type=O3CPU --l1d_size=32kB --l1i_size=32kB --caches --l2_size
    =256kB --l2cache --l1d_assoc=8 --l1i_assoc=8 --l2_assoc=8 --
    maxinsts=1000000 --cmd=/mnt/beegfs/gap/mapecfer/spec2017/
    benchspec/CPU/503.bwaves_r/exe/bwaves_r_base.InitialTest-m64
    --options="<bwaves_1.in"

mv data.csv ../datos/bwaves.csv

../build/X86/gem5.opt ../configs/deprecated/example/se.py --cpu-
    type=O3CPU --l1d_size=32kB --l1i_size=32kB --caches --l2_size
    =256kB --l2cache --l1d_assoc=8 --l1i_assoc=8 --l2_assoc=8 --
    maxinsts=1000000 --cmd=/mnt/beegfs/gap/mapecfer/spec2017/
    benchspec/CPU/525.x264_r/exe/x264_r_base.InitialTest-m64 --
    options="--pass 1 --stats x264_stats.log --bitrate 1000 --
    frames 1000 -o BuckBunny_New.264 BuckBunny.yuv 1280x720"

mv data.csv ../datos/x264.csv
```

Figura A.4: Spec2017.sh

```
 #!/bin/bash
#SBATCH --job-name=spec2006
#SBATCH --nnodes=1
#SBACTH --ntasks=24
#SBATCH --cpus-per-task=1

../build/X86/gem5.opt ../configs/deprecated/example/se.py --cpu-
    type=O3CPU --l1d_size=32kB --l1i_size=32kB --caches --l2_size
    =256kB --l2cache --l1d_assoc=8 --l1i_assoc=8 --l2_assoc=8 --
    maxinsts=4000000 --cmd=spec2006-x86-bin/400.perlbench/
    perlbench_base.i386 --options="-ICPU2006/400.perlbench/data/
    all/input/lib CPU2006/400.perlbench/data/all/input/diffmail.
    pl 4 800 10 17 19 300"

mv data.csv ./resultados/perlbench.csv

../build/X86/gem5.opt ../configs/deprecated/example/se.py --cpu-
    type=O3CPU --l1d_size=32kB --l1i_size=32kB --caches --l2_size
    =256kB --l2cache --l1d_assoc=8 --l1i_assoc=8 --l2_assoc=8 --
    maxinsts=4000000 --cmd=spec2006-x86-bin/401.bzip2/bzip2_base.
    i386 --options="CPU2006/401.bzip2/data/all/input/input.
    program 280"

mv data.csv ./resultados/bzip.csv

../build/X86/gem5.opt ../configs/deprecated/example/se.py --cpu-
    type=O3CPU --l1d_size=32kB --l1i_size=32kB --caches --l2_size
```

```
    =256kB --l2cache --l1d_assoc=8 --l1i_assoc=8 --l2_assoc=8 --
    maxinsts=4000000 --cmd=spec2006-x86-bin/403.gcc/gcc_base.i386
     --options="CPU2006/403.gcc/data/ref/input/200.i -o 200.s"
16
17 mv data.csv ./resultados/gcc.csv
18
19 ../build/X86/gem5.opt ../configs/deprecated/example/se.py --cpu-
    type=O3CPU --l1d_size=32kB --l1i_size=32kB --caches --l2_size
    =256kB --l2cache --l1d_assoc=8 --l1i_assoc=8 --l2_assoc=8 --
    maxinsts=4000000 --cmd=spec2006-x86-bin/429.mcf/mcf_base.i386
     --options="CPU2006/429.mcf/data/ref/input/inp.in"
20
21 mv data.csv ./resultados/mcf.csv
22
23 ../build/X86/gem5.opt ../configs/deprecated/example/se.py --cpu-
    type=O3CPU --l1d_size=32kB --l1i_size=32kB --caches --l2_size
    =256kB --l2cache --l1d_assoc=8 --l1i_assoc=8 --l2_assoc=8 --
    maxinsts=4000000 --cmd=spec2006-x86-bin/445.gobmk/gobmk_base.
    i386 --options="--quiet --mode gtp <CPU2006/445.gobmk/data/
    ref/input/13x13.tst"
24
25 mv data.csv ./resultados/gobmk.csv
26
27 ../build/X86/gem5.opt ../configs/deprecated/example/se.py --cpu-
    type=O3CPU --l1d_size=32kB --l1i_size=32kB --caches --l2_size
    =256kB --l2cache --l1d_assoc=8 --l1i_assoc=8 --l2_assoc=8 --
    maxinsts=4000000 --cmd=spec2006-x86-bin/456.hmmer/hmmer_base.
    i386 --options="--fixed 0 --mean 500 --num 500000 --sd 350 --
    seed 0 CPU2006/456.hmmer/data/ref/input/retro.hmm"
28
29 mv data.csv ./resultados/hmmer.csv
30
31 ../build/X86/gem5.opt ../configs/deprecated/example/se.py --cpu-
    type=O3CPU --l1d_size=32kB --l1i_size=32kB --caches --l2_size
    =256kB --l2cache --l1d_assoc=8 --l1i_assoc=8 --l2_assoc=8 --
    maxinsts=4000000 --cmd=spec2006-x86-bin/458.sjeng/sjeng_base.
    i386 --options="CPU2006/458.sjeng/data/ref/input/ref.txt"
32
33 mv data.csv ./resultados/sjeng.csv
34
35 ../build/X86/gem5.opt ../configs/deprecated/example/se.py --cpu-
    type=O3CPU --l1d_size=32kB --l1i_size=32kB --caches --l2_size
    =256kB --l2cache --l1d_assoc=8 --l1i_assoc=8 --l2_assoc=8 --
    maxinsts=4000000 --cmd=spec2006-x86-bin/462.libquantum/
    libquantum_base.i386 --options="1397 8"
36
37 mv data.csv ./resultados/libquantum.csv
38
39 ../build/X86/gem5.opt ../configs/deprecated/example/se.py --cpu-
    type=O3CPU --l1d_size=32kB --l1i_size=32kB --caches --l2_size
    =256kB --l2cache --l1d_assoc=8 --l1i_assoc=8 --l2_assoc=8 --
    maxinsts=4000000 --cmd=spec2006-x86-bin/464.h264ref/
```

```
    h264ref_base.i386 --options="-d CPU2006/464.h264ref/data/ref/
    input/foreman_ref_encoder_baseline.cfg"
40
41 mv data.csv ./resultados/h264ref.csv
42
43 ../build/X86/gem5.opt ../configs/deprecated/example/se.py --cpu-
    type=O3CPU --l1d_size=32kB --l1i_size=32kB --caches --l2_size
    =256kB --l2cache --l1d_assoc=8 --l1i_assoc=8 --l2_assoc=8 --
    maxinsts=4000000 --cmd=spec2006-x86-bin/471.omnetpp/
    omnetpp_base.i386 --options="CPU2006/471.omnetpp/data/ref/
    input/omnetpp.ini"
44
45 mv data.csv ./resultados/omnetpp.csv
46
47 ../build/X86/gem5.opt ../configs/deprecated/example/se.py --cpu-
    type=O3CPU --l1d_size=32kB --l1i_size=32kB --caches --l2_size
    =256kB --l2cache --l1d_assoc=8 --l1i_assoc=8 --l2_assoc=8 --
    maxinsts=4000000 --cmd=spec2006-x86-bin/473.astar/astar_base.
    i386 --options="CPU2006/473.astar/data/ref/input/BigLakes2048
    .cfg"
48
49 mv data.csv ./resultados/astar.csv
50
51 ../build/X86/gem5.opt ../configs/deprecated/example/se.py --cpu-
    type=O3CPU --l1d_size=32kB --l1i_size=32kB --caches --l2_size
    =256kB --l2cache --l1d_assoc=8 --l1i_assoc=8 --l2_assoc=8 --
    maxinsts=4000000 --cmd=spec2006-x86-bin/483.xalancbmk/
    xalancbmk_base.i386 --options="-v CPU2006/483.xalancbmk/data/
    ref/input/t5.xml CPU2006/483.xalancbmk/data/ref/input/xalanc.
    xsl"
52
53 mv data.csv ./resultados/xalancbmk.csv
54
55 ../build/X86/gem5.opt ../configs/deprecated/example/se.py --cpu-
    type=O3CPU --l1d_size=32kB --l1i_size=32kB --caches --l2_size
    =256kB --l2cache --l1d_assoc=8 --l1i_assoc=8 --l2_assoc=8 --
    maxinsts=4000000 --cmd=spec2006-x86-bin/410.bwaves/
    bwaves_base.i386
56
57 mv data.csv ./resultados/bwaves.csv
58
59 ../build/X86/gem5.opt ../configs/deprecated/example/se.py --cpu-
    type=O3CPU --l1d_size=32kB --l1i_size=32kB --caches --l2_size
    =256kB --l2cache --l1d_assoc=8 --l1i_assoc=8 --l2_assoc=8 --
    maxinsts=4000000 --cmd=spec2006-x86-bin/416.gamess/
    gamess_base.i386 --options="<CPU2006/416.gamess/data/ref/
    input/h2ocu2+.gradient.config"
60
61 mv data.csv ./resultados/gamess.csv
62
63 ../build/X86/gem5.opt ../configs/deprecated/example/se.py --cpu-
    type=O3CPU --l1d_size=32kB --l1i_size=32kB --caches --l2_size
```

```
    =256kB --l2cache --l1d_assoc=8 --l1i_assoc=8 --l2_assoc=8 --
    maxinsts=4000000 --cmd=spec2006-x86-bin/433.milc/milc_base.
    i386 --options="<CPU2006/433.milc/data/ref/input/su3imp.in"
64
65 mv data.csv ./resultados/milc.csv
66
67 ../build/X86/gem5.opt ../configs/deprecated/example/se.py --cpu-
    type=O3CPU --l1d_size=32kB --l1i_size=32kB --caches --l2_size
    =256kB --l2cache --l1d_assoc=8 --l1i_assoc=8 --l2_assoc=8 --
    maxinsts=4000000 --cmd=spec2006-x86-bin/434.zeusmp/
    zeusmp_base.i386
68
69 mv data.csv ./resultados/zeusmp.csv
70
71 ../build/X86/gem5.opt ../configs/deprecated/example/se.py --cpu-
    type=O3CPU --l1d_size=32kB --l1i_size=32kB --caches --l2_size
    =256kB --l2cache --l1d_assoc=8 --l1i_assoc=8 --l2_assoc=8 --
    maxinsts=4000000 --cmd=spec2006-x86-bin/435.gromacs/
    gromacs_base.i386 --options="-silent -deffnm CPU2006/435.
    gromacs/data/ref/input/gromacs -nice 0"
72
73 mv data.csv ./resultados/gromacs.csv
74
75 ../build/X86/gem5.opt ../configs/deprecated/example/se.py --cpu-
    type=O3CPU --l1d_size=32kB --l1i_size=32kB --caches --l2_size
    =256kB --l2cache --l1d_assoc=8 --l1i_assoc=8 --l2_assoc=8 --
    maxinsts=4000000 --cmd=spec2006-x86-bin/436.cactusADM/
    cactusADM_base.i386 --options="CPU2006/436.cactusADM/data/ref
    /input/benchADM.par"
76
77 mv data.csv ./resultados/cactus.csv
78
79 ../build/X86/gem5.opt ../configs/deprecated/example/se.py --cpu-
    type=O3CPU --l1d_size=32kB --l1i_size=32kB --caches --l2_size
    =256kB --l2cache --l1d_assoc=8 --l1i_assoc=8 --l2_assoc=8 --
    maxinsts=4000000 --cmd=spec2006-x86-bin/437.leslie3d/
    leslie3d_base.i386 --options="<CPU2006/437.leslie3d/data/ref/
    input/leslie3d.in"
80
81 mv data.csv ./resultados/leslie.csv
82
83 ../build/X86/gem5.opt ../configs/deprecated/example/se.py --cpu-
    type=O3CPU --l1d_size=32kB --l1i_size=32kB --caches --l2_size
    =256kB --l2cache --l1d_assoc=8 --l1i_assoc=8 --l2_assoc=8 --
    maxinsts=4000000 --cmd=spec2006-x86-bin/444.namd/namd_base.
    i386 --options="--input CPU2006/444.namd/data/all/input/namd.
    input --iterations 38 --output namd.out"
84
85 mv data.csv ./resultados/namd.csv
86
87 ../build/X86/gem5.opt ../configs/deprecated/example/se.py --cpu-
    type=O3CPU --l1d_size=32kB --l1i_size=32kB --caches --l2_size
```

```
    =256kB --l2cache --l1d_assoc=8 --l1i_assoc=8 --l2_assoc=8 --
    maxinsts=4000000 --cmd=spec2006-x86-bin/447.dealII/
    dealII_base.i386 --options="23"
88
89 mv data.csv ./resultados/deal.csv
90
91 ../build/X86/gem5.opt ../configs/deprecated/example/se.py --cpu-
    type=O3CPU --l1d_size=32kB --l1i_size=32kB --caches --l2_size
    =256kB --l2cache --l1d_assoc=8 --l1i_assoc=8 --l2_assoc=8 --
    maxinsts=4000000 --cmd=spec2006-x86-bin/450.soplex/
    soplex_base.i386 --options="-s1 -e -m45000 CPU2006/450.soplex
    /data/ref/input/pds-50.mps"
92
93 mv data.csv ./resultados/soplex.csv
94
95 ../build/X86/gem5.opt ../configs/deprecated/example/se.py --cpu-
    type=O3CPU --l1d_size=32kB --l1i_size=32kB --caches --l2_size
    =256kB --l2cache --l1d_assoc=8 --l1i_assoc=8 --l2_assoc=8 --
    maxinsts=4000000 --cmd=spec2006-x86-bin/453.povray/
    povray_base.i386 --options="CPU2006/453.povray/data/ref/input
    /SPEC-benchmark-ref.ini"
96
97 mv data.csv ./resultados/povray.csv
98
99 ../build/X86/gem5.opt ../configs/deprecated/example/se.py --cpu-
    type=O3CPU --l1d_size=32kB --l1i_size=32kB --caches --l2_size
    =256kB --l2cache --l1d_assoc=8 --l1i_assoc=8 --l2_assoc=8 --
    maxinsts=4000000 --cmd=spec2006-x86-bin/459.GemsFDTD/
    GemsFDTD_base.i386
100
101 mv data.csv ./resultados/gems.csv
102
103 ../build/X86/gem5.opt ../configs/deprecated/example/se.py --cpu-
    type=O3CPU --l1d_size=32kB --l1i_size=32kB --caches --l2_size
    =256kB --l2cache --l1d_assoc=8 --l1i_assoc=8 --l2_assoc=8 --
    maxinsts=4000000 --cmd=spec2006-x86-bin/470.lbm/lbm_base.i386
     --options="300 reference.dat 0 0 CPU2006/470.lbm/data/ref/
    input/100_100_130_ldc.of"
104
105 mv data.csv ./resultados/lbm.csv
```

Figura A.5: Spec2006.sh

A.2.3. Realización de gráficas

A continuación en la figura A.6 se muestra el código de Graph.py, *script* pyhton de ejemplo que se ha empleado para la realización de gráficas.

```
#!/usr/bin/python3.8
import seaborn as sns
import glob
import csv
import shutil
import pandas as pd
import os
# plot figures
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import rcParams
# Puesta a punto de seaborn
sns.set_theme(style="whitegrid")
sns.set_context("paper", font_scale=1.5)
matplotlib.rcParams['font.family'] = "serif"
csv_list = glob.glob('./data.csv')
pandas_dataframe = pd.DataFrame( columns = ['name', 'Squashed',
    'Serialized', 'ROB','IQ','LQ','SQ','Instrucctions','Frontend'
    , 'Backend', 'IPC'])
for csv_data in csv_list:
    datos_spec    = pd.read_csv(csv_data)
    #Informacion Basica
    name=csv_data.split("/")[-1].split(".")[0]
    cycles = datos_spec['cycle'].count()
    instructions = datos_spec['instructions'].sum()
    ipc = instructions / cycles
    slots = cycles x 4
    serialized = datos_spec['serializing'].sum()
    slots = slots - serialized
    #Stack
    instructions = instructions / slots
    squashed   = datos_spec['squashed'].sum() / slots
    rob        = datos_spec['ROB'].sum() / slots
    iq         = datos_spec['IQ'].sum() / slots
    lq         = datos_spec['LQ'].sum() / slots
    sq         = datos_spec['SQ'].sum() / slots
    frontend   = (datos_spec['frontend'].sum() / slots) +
       squashed
    backend = rob + iq + lq + sq
    new_row = {'name': name, 'Squashed': squashed, 'Serialized':
        serialized, 'ROB': rob, 'IQ': iq, 'LQ': lq, 'SQ': sq, '
       Instrucctions': instructions, 'Frontend': frontend, '
       Backend': backend, 'IPC': ipc}
    pandas_dataframe = pd.concat([pandas_dataframe, pd.DataFrame
       ([new_row])], ignore_index=True)
#print(pandas_dataframe)
```

```
41 pandas_dataframe.to_csv("./pandas/pandas_dataframe.csv")
42 pandas_dataframe = pandas_dataframe.sort_values(by='name')
43 lista_specs          = pandas_dataframe['name']
44 lista_dispatched    = pandas_dataframe['Instrucctions']
45 lista_squashed= pandas_dataframe['Squashed']
46 lista_frontend      = pandas_dataframe['Frontend']
47 lista_backend       = pandas_dataframe['Backend']
48 lista_ipc           = pandas_dataframe['IPC']
49 # Colores opcionales
50 # lista_colores = ["#239b56", "#58d68d", "#ec7063", "#943126"]
51 # lista_colores = [ "#0077BB", "#F6790B", "#5AAE61", "#A50026" ]
52 lista_colores = [ "#0077BB", "#F6790B", "#A50026" ]
53 # --
54 caracterizacion_estatica = pd.concat([lista_specs,
       lista_dispatched, lista_frontend, lista_backend], axis =1)
55 #print(caracterizacion_estatica)
56 # caracterizacion_estatica = pd.concat([lista_specs,
       lista_dispatched, lista_squashed, lista_frontend,
       lista_backend], axis =1)
57 ax = caracterizacion_estatica.plot(kind='bar', stacked=True,
       color = lista_colores)
58 ax.legend(fontsize=13, loc='upper center', bbox_to_anchor=(0.46,
        1.12), ncol=3,  columnspacing=0.3, frameon=False,fancybox=
       False, shadow=False)
59 # ax.legend(fontsize=10, loc='upper center', bbox_to_anchor
       =(0.5, 1.14), ncol=4,  columnspacing=0.3, frameon=False,
       fancybox=False, shadow=False)
60 ax.set_ylabel("Fraction of Execution Time", fontsize=19)
61 ax.set_ylim([0.0, 1.0])
62 plt.yticks(fontsize = 15)
63 ax.grid()
64 # ax.tight_layout()
65 ax2 = ax.twinx()
66 ax2.plot(lista_ipc.values, linestyle = 'None', marker = '.',
       markersize=20 ,linewidth=2.5,  color="k")
67 ax2.set_ylabel("IPC")
68 ax2.set_ylim([0.0, 4.0])
69 ax.grid()
70 plt.xticks(np.arange(len(lista_specs)), lista_specs)
71 plt.grid()
72 # plt.tight_layout()
73 # plt.show()
74 # ---
75 plt.savefig("./graph.png", dpi=500, bbox_inches= "tight")
```

Figura A.6: Graph.py

APÉNDICE B
Suite de *Benchmarks*

Se presentan en este apéndice los benchmaks empleados y una explicación de cada uno de ellos:

B.1 SPEC 2006

- **bzip2:** Basado en la aplicación bzip2 1.0.3 de Julian Seward, realiza una serie de operación de compresión y decompresión de archivos en memoria.
- **Cactus:** Este *benchmark* resuelve ecuaciones evolutivas para describir como el espacio-tiempo se curva a causa de la materia creadas por Albert Einstein.
- **Gcc:** Se compilan programas de prueba con todas las banderas de optimización encendidas.
- **Libquantum**: Se simula el funcionamiento de un ordenador cuántico.
- **Sjeng**: Simula partidas de ajedrez y muchas de sus variantes buscando el moviemiento más óptimo mediante el algoritmo alpha-beta.
- **Soplex**: Resuelve problemas de programación lineal con el algoritmo Simplex.
- **Xalancbmk**: Transforma archivos XML a árboles de nodos.

B.2 SPEC 2017

- **Cactus_r:** Resuelve ecuaciones evolutivas en espacios planos, ecuaciones de comportamiento de estrellas de neutrinos, enanas blancas, agujeros negros y modela la expansión de ondas gravitacionales.
- **Cam4_r:** Realiza simulaciones atmosféricas para predicción del tiempo.
- **Lbm_r:** Simulación de fluidos en superficies libres, se enfoca en simulaciones de cambio de estado de la materia, donde líquidos se transforman en gas.
- **Leela_r:** Este *benchmark* se encarga de hacer simulaciones de árboles de decisiones montecarlo masivas para inteligencia artificial.
- **Mcf_r:** Cuenta con una flota de vehículos donde, en una ciudad, se busca resolver el problema de encontrar el camino óptimo para cada vehículo.
- **Namd_r:** Dinámica de moléculas, donde se busca resolver el problema de interacción de estas partículas en distintos entornos.
- **Omnetpp_r:** Relaciones de eventos en redes de computadores complejas.
- **Parest_r:** Dentro del área de la Biomedicina, se busca optimizar las tomografías mediante elementos finitos.
- **Povray_r:** Trazado de rayos con simulaciones en entornos fotorealistas.
- **Wrf_r:** Predicción del tiempo atmosférico empleando diferentes modelos del estado del arte de la climatología.
- **Xalancbmk_r:** Conversión de archivos XML a árboles de nodos.
- **Xz_r:** Compresión y decompresión de archivos.

APÉNDICE C
Objetivos de Desarrollo Sostenible

Este apéndice expone la relación del trabajo realizado con los Objetivos de Desarrollo Sostenible (ODS) mostrado en la tabla C.1.

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

Tabla C.1: ODS

La digitalización del mundo es imparable, hace unos años era impensable que en tan poco tiempo se pudiese pasar de una sociedad donde la tecnología tenía un papel secundario a sociedades donde la tecnología vertebra el día a día tanto de nuestras vidas personales como laborales.

El progreso y desarrollo se sostiene sobre los pilares de la constancia y perseverancia, avanzar significa fallar, aprender y volverlo a intentar. Es por eso que un desarrollo tan rápido como el que hemos tenido que poco tuvo en cuenta el impacto de las acciones que se hacían para con el planeta, ha traído consecuencias tan negativas que si no se les pone remedio, se tambalea el futuro de nuestra generación y de las generaciones siguientes. Estos remedios no significa dejar de progresar, sino aprender a hacerlo de forma sostenible y concienciada con el mundo y la sociedad donde el desarrollo sea de todos y para todos pensando en

beneficiar tanto a las personas como a la casa en la que habitamos que es este mundo.

En este trabajo se ha desarrollado una PMU para un simulador de CPU de código abierto el cual es empleado a nivel global para el desarrollo de microarquitecturas de procesadores. Por estar características este trabajo se desarrolla en el marco de acción de los ODS de la siguiente forma:

- ODS 8. **Trabajo decente y crecimiento económico:** Entre sus metas se encuentra la de un desarrollo económico sostenible, teniendo en cuenta que hoy en día el mundo digital es protagonista, el desarrollo de este trabajo agiliza y abarata enormemente los costes monetarios del desarrollo de nuevas CPUs que favorece al desarrollo económico de nuestro país.

- ODS 9. **Industria, innovación e infraestructuras:** El trabajo realizado se centra principalmente en el desarrollo de una herramienta muy útil para la investigación en innovación de microarquitecturas, lo cual contribuye enormemente a la instalación sostenible de infraestructura digital en zonas hasta la fecha desconectadas.

- ODS 10. **Reducción de las desigualdades:** El I+D+I supone una gran inversión monetaria en máquinas y tecnologías punteras que son muy inaccesibles en gran cantidad de países de todo el mundo. La contribución de este trabajo es a un simulador de código abierto el cual es de libre acceso para cualquier persona y que para ejecutarse solamente requiere de un ordenador con acceso a internet, suponiendo uno de los principales avances de democratización de la investigación y reducción de desigualdades entre países.

- ODS 11. **Producción y consumo responsables:** Se persigue la meta de reducir la producción de prototipos costosos a nivel de recursos extraídos del planeta como a nivel de polución de transporte de los mismos para su fabricación y posterior abandono. Se busca contribuir a la simulación como alternativa sostenible y responsable con el planeta.

- ODS 12. **Acción por el clima:** Este trabajo espera contribuir al uso de simuladores y reutilizar al máximo posible la tecnología con la que ya se dispone para sacarle el máximo provecho con el objetivo de reducir la gran cantidad de residuos electrónicos que se generan a día de hoy. Se persigue contribuir al clima mediante tanto la reutilización de equipos para investigación como reducir el coste climático de la fabricación de costosos prototipos que serán desechados al tratarse únicamente de un prototipo.