UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

Estudio y Mejora de una herramienta de análisis del efecto sobre el WCET de la ejecución Multi-Núcleo y del bus compartida de acceso a la cache L2.

Autor: Javier Enrique Barrera Herrera

Grado en Ingeniería Informática

Escuela de Ingeniería Informática (EII)

Universidad de Las Palmas de Gran Canaria (ULPGC)

Tutores:

Enrique Fernández García[1]

Francisco Javier Cazorla Almeida[2]

Julio 2018

1: Universidad de Las Palmas de Gran Canaria

2: Barcelona Supercomputing Center –
Centro Nacional de Supercomputación

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

TFT04

SOLICITUD DE DEFENSA DE TRABAJO DE FIN DE TÍTULO

D. Javier Enrique Barrera Herrera, autor del Trabajo de Fin de Título ***"Estudio y Mejora de una herramienta de análisis del efecto sobre el WCET de la ejecución Multi-Núcleo y del bus compartido de acceso a la cache L2"***, correspondiente a la titulación Grado en Ingeniería Informática, en colaboración con la empresa/proyecto (indicar en su caso) Barcelona Supercomputing Center

S O L I C I T A

que se inicie el procedimiento de defensa del mismo, para lo que se adjunta la documentación requerida.

Asimismo, con respecto al registro de la propiedad intelectual/industrial del TFT, declara que:

[X] Se ha iniciado o hay intención de iniciarlo (defensa no pública).
[ ] No está previsto.

Y para que así conste firma la presente.

Las Palmas de Gran Canaria, a 10 de Julio de 2018.

El estudiante

Firmado por BARRERA HERRERA JAVIER ENRIQUE - 45353816R el día 13/07/2018 con un certificado emitido por AC FNMT Usuarios

Fdo.: Javier Enrique Barrera Herrera

A rellenar y firmar **obligatoriamente** por el/los tutor/es

En relación a la presente solicitud, se informa:

[X] Positivamente

[ ] Negativamente
(la justificación en caso de informe negativo deberá incluirse en el TFT05)

Firmado por FERNANDEZ GARCIA ENRIQUE - 43253427H el día 13/07/2018 con un certificado emitido por AC FNMT Usuarios

Fdo.: Enrique Fernández García

Fdo.: Francisco Javier Cazorla Almeida

DIRECTOR DE LA ESCUELA DE INGENIERÍA INFORMÁTICA

Agradecimientos

En primer lugar, quiero agradecer el Barcelona Supercomputing Center – Centro Nacional de Supercomputación, y en especial a Francisco J. Cazorla Almeida así como a los integrantes del grupo CAOS por esta oportunidad de realizar un Trabajo de Fin de Grado que desde el primer momento me ha cautivado e interesado. También quiero agradecer a David Trilla, autor original de SIM1, y a Mikel Fernández por enseñarme la base teórica necesaria para realizar el mencionado TFG.

También quiero agradecer a mi tutor, Enrique Fernández García, con la paciencia que ha tenido y su dedicación en la enseñanza, estando disponible para dudas hasta altas horas de la madrugada.

A mi amigo y compañero Jeremy Jens Giesen León, le agradezco 4 años llenos de colaboración y apoyo mutuo desde los primeros ‘pinitos’ que hacíamos en el Grado en Ingeniería Informática.

Quiero agradecer a mi familia y a mi pareja, Patricia, que, de forma desinteresada, me han escuchado balbucear términos técnicos y me han apoyado en todo momento, demostrando cada día la suerte que he tenido de poder contar con ellos en cualquier momento.

Por último, agradecer a todos aquellos profesores que han permitido nutrirme de conocimiento y en especial a aquellos que han dedicado tiempo libre para la enseñanza y resolución de dudas, así como a todos los compañeros que han hecho de esta vida universitaria, una experiencia completa.

Gracias a todos.

Contenido

Resumen

El trabajo se organiza en distintos apartados, de los cuales se mencionará de forma breve el contenido de cada uno de ellos.

El primero trata del problema que trata de resolver el uso de la herramienta de análisis, así como la importancia de la propia solución y de implementar mejoras en ella. Por otro lado, el segundo capítulo muestra el estado actual de la herramienta, especificando diversas características de la misma para dar a conocer así exactamente qué es capaz de hacer, qué no, y cómo se organiza.

Al llegar al tercer apartado, se diseñan experimentos con benchmarks para afianzar los conocimientos de uso y validar el cálculo del WCET según el estado actual.

El cuarto capítulo trata sobre las mejoras exploradas e implementadas en el código del simulador, estando éstas debidamente documentadas. A continuación, en el quinto punto, se presenta el diseño de experimentos haciendo uso de las mejoras y el análisis de los resultados obtenidos en dichos experimentos.

Por último, el sexto apartado reúne las conclusiones obtenidas a lo largo de la elaboración de los experimentos, del análisis de los resultados, y de la propia confección del Trabajo de Fin de Grado. También se incluyen en este apartado los trabajos futuros a realizar sobre este proyecto.

Alcance del trabajo

Este Trabajo de Fin de Grado está encuadrado en una de las líneas de investigación existente en el Grupo de investigación *"Computer Architecture - Operating System"* (**CAOS**) de la institución *"Barcelona Supercomputing Center – Centro Nacional de Supercomputación"* (**BSC**). El proyecto en sí consiste en el análisis de una de las mayores necesidades en los sistemas empotrados de tiempo real crítico, la garantía de que, incluso en el peor de los casos, dicho sistema es capaz de realizar sus tareas dentro de un período de tiempo en concreto.

El cumplimiento de la tarea en un tiempo dado es fundamental para estos sistemas, puesto que, de lo contrario, podría acarrear problemas graves, es por ello por lo que se busca la estimación del *"Worst-Case Execution Time"* (**WCET**). El proyecto, por tanto, plantea un modelo de estimación combinando dos técnicas compatibles con la técnica de análisis *"Measurements Based Probabilistic Timing Analysis"* (**MBPTA**).

El alcance de trabajo de este Trabajo de Fin de Grado es el de utilizar el modelo de estimación propuesto por el proyecto dadas unas herramientas de simulación. Del procesador de referencia (**LEON3**) que dispone de 4 núcleos, se utilizarán 2 de esos núcleos, uno de ellos actuará como la *"Task Under Analysis"* (**TUA**), y el otro núcleo como *"Task Contender"* (TC). El núcleo TUA será el que hemos observado mediante las herramientas de simulación existentes en el grupo CAOS, mientras que TC será el contendiente, la tarea que compite contra la TUA por los recursos de la placa de referencia.

Dentro de este trabajo, se han revisado las herramientas existentes, analizándolas y comprendiéndolas, para, posteriormente, implementar las nuevas funcionalidades que mejoran el uso de las mismas y el cálculo del WCET, así como posibilitar la ampliación de posibles combinaciones de experimentos.

Objetivos

Se ha propuesto, en general, estudiar y explorar los modos de mejorar una herramienta existente para la medición del efecto sobre el cálculo del **WCET** (*"Worst-Case Execution Time"*) cuando se planifica la ejecución multinúcleo, y, además, se comparte el bus de acceso a la cache L2.

Dicha herramienta se compone de dos partes, las cuales, se ha realizado un análisis y exploración para implementar mejoras. Estas partes son:

- SIM1, el cual se encuentra programado en C++ y se trata de un simulador que existe actualmente en el grupo CAOS. Este obtiene múltiples estadísticas de la ejecución de cada uno de los benchmarks ejecutados en solitario.
- SIM2, simulador programado en Python, que utiliza los resultados de SIM1. Se trata de otro simulador del grupo CAOS, y su funcionalidad es la combinación de una tarea bajo estudio (**TUA:** *Task Under Analysis*) con otras tareas (**TC:** *Task Contenders*), para valorar, así, el efecto que la ejecución concurrente de las distintas tareas produce sobre el WCET de la TUA.

Para lograr los objetivos anteriores, se ha seguido una metodología bien definida:

- Estudio de la bibliografía para permitir conocer el problema y el enfoque del mismo que se realiza, así como de las posibles mejoras a realizar en los métodos utilizados.
- Estudio de la implementación de la herramienta actual de simulación (SIM1) que calcula estadísticas de ejecución mono-hilo a partir de trazas de ejecución de benchmarks. Identificación de las partes de dicho simulador a modificar y las funcionalidades a añadir.
- Estudio de la implementación de la herramienta actual de simulación (SIM2) que calcula WCET haciendo uso de las estadísticas de ejecución en mono-hilo obtenidas previamente por SIM1. Identificación de las partes de dicho simulador a modificar y las funcionalidades a añadir.
- Programación de las modificaciones y validación de dicho código generado.
- Documentación del código.
- Diseño y ejecución de un experimento de análisis con benchmarks reales. Análisis y presentación de los resultados, así como de las conclusiones obtenidas a raíz de ellos.

Competencias específicas

La elaboración de este Trabajo de Fin de Grado cumple con las competencias comunes, así como la competencia específica IC03.

"**IC03:** Capacidad de analizar y evaluar arquitecturas de computadores, incluyendo plataformas paralelas y distribuidas, así como desarrollar y optimizar software para las mismas."

Capítulo 1

Introducción

1.1. Problemática

La necesidad de capacidad de cómputo ha venido creciendo de forma continuada en los dominios de los sistemas empotrados de tiempo real crítico, debiéndose esto a la complejidad y cantidad de los datos que se esperan que manejen. Es por esto, que el uso de hardware que sea capaz de atender estas necesidades de cómputo trae consigo ciertas características, como puede ser un procesador multinúcleo, que resultan difíciles de modelar debido a la variabilidad que presentan en tiempo de ejecución, complicando así la validación de tiempo y su verificación.

En estos sistemas empotrados de tiempo real, la necesidad de disponer de garantías de que los sistemas sean capaces de procesar toda la información dentro de un tiempo dado es esencial, y por eso mismo, surge el concepto “Worst-Case Execution Time” (**WCET**), el cual marca el peor tiempo posible para un sistema dado. Si para una situación dada, se calcula el WCET y se encuentra dentro de los valores permitidos de tiempo, entonces se puede considerar que el sistema empotrado da las garantías necesarias para realizar su labor en la reducida cantidad de tiempo permitida.

1.2. Técnicas de Análisis

Para poder realizar la validación de tiempo y verificarlo, se han diseñado distintos enfoques con los cuales intentar alcanzar estos datos tan difíciles de calcular. El enfoque "*Measurement-Based Timing Analysis*" (**MBTA**) predomina en los dominios de tiempo real, puesto que tiene como objetivo el cálculo del "*Worst-Case Execution Time*" (**WCET**). Sin embargo, el grado de fiabilidad de la estimación WCET que calcula MBTA reside en la habilidad de diseñar escenarios con una carga de trabajo altamente estresante, acercándose así a situaciones similares en las que se daría el WCET. Esto provoca que la confianza en la estimación del WCET decaiga, provocando así que se evite usar características de hardware en los sistemas empotrados de tiempo real crítico.

La variante "*Measurement-Based Probabilistic Timing Analysis*" (**MBPTA**) trata de incrementar la fiabilidad y confianza en la estimación del WCET. Para ello, controla el impacto de estos recursos de forma implícita, algunos mediante el "*upperbounding*", haciendo que trabajen con las peores latencias, y otros de forma aleatoria.

1.3. Aproximación MC2

El proyecto donde se encuadra este trabajo de fin de grado propone la técnica de análisis MC2 (Multi-Core and Cache), la cual es una aproximación para el análisis de procesadores multinúcleos equipados con cache multinivel y que son comercializados actualmente (*COTS: Commercial-Off-The-Shelf*). Para poder propiciar la rápida adopción de las técnicas de análisis MBPTA, es fundamental analizar la aplicabilidad de las mismas a los procesadores multinúcleo COTS.

MC2 expone, por tanto, de una forma combinada compatible con la técnica MBPTA, la inestabilidad a través del tiempo de las caches y la contención multinúcleo de las mediciones de tiempo de ejecución tomadas en el análisis. Como resultado, la estimación del WCET generada por el análisis MBPTA se perfila como cota superior a la hora de observar el impacto producido por los recursos multinúcleo y cache multinivel. MC2 logra esto gracias a la combinación de dos técnicas compatibles con el MBPTA, estas son:

- Aleatoriedad por software para el manejo de la inestabilidad de cache.
- Retraso del límite superior para el manejo de la contención multinúcleo.

Como esta contención multinúcleo puede generar estimaciones muy pesimistas, MC2 dispone una herramienta de estimación adaptable a las contenciones producidas por los contendientes.

1.4. Plataforma de Referencia

La plataforma de referencia utilizada en el proyecto dispone de un procesador de 4 núcleos **LEON3**, y se encuentra implementada mediante una FPGA. Cada núcleo del procesador LEON3 dispone de un *"pipeline"* de 7 etapas y comprende las caches de primer nivel para instrucciones (**IC**) y para datos (**DC**).

Estas caches de nivel 1 implementan una política de Write-Through No Write Allocate, por lo que el bus de interconexión con la cache de nivel 2 propaga todas las instrucciones de "store", los fallos en IC y los fallos en DC. Esta cache de nivel 2 dispone de 4 vías, pudiendo ser totalmente asociativa o particionada fijando una vía a cada núcleo para evitar así las interferencias entre los núcleos. La plataforma también abarca los denominados *"Performance Monitoring Counters"* (**PMCs**), de los que se capturan eventos que se verán en capítulos más adelante.

1.5. Modelo de Contención fTC

El modelo de contención *"fully Time Composable"* (**fTC**) realiza una estimación del WCET que acota superiormente al incremento en tiempo de ejecución que sufre la TUA sin importar la carga de las TC. Esto provoca que este modelo utilice N_núcleos – 1 *"Task Contenders"*, además de asumir que, por cada petición de la TUA, los contenders realizan una acción del peor tipo, provocando la mayor contención posible.

1.6. Modelo de Contención pTC

A diferencia del model fTC, el modelo *"partially Time Composable"* (**pTC**) no asume que se dispone de N_núcleos -1 contendientes, sino que toma el valor real de contendientes. Además, pTC contabiliza el número de peticiones de cada tipo, ofreciendo de esta manera una solución para reducir el WCET estimado por fTC.

Capítulo 2

Situación Inicial

Las herramientas de simulación SIM1 y SIM2 han sido analizadas, en diversos aspectos, desde la organización de directorios hasta el formato de salida. Esto permitió conocer a fondo la situación de partida de dichas herramientas, y la planificación de las modificaciones y nuevas funcionalidades a implementar.

2.1. Estado Actual SIM1.

A continuación, se detalla el análisis de la situación inicial de la herramienta de simulación SIM1 hecho con el fin de poder definir con claridad cómo era el simulador en su versión original. Se han estudiado una serie de puntos a tratar. Estos puntos, en líneas generales son:

- Uso del Simulador.
- Análisis Previo de Uso (Especificación de carga de trabajo, Estructura de directorios…).
- Análisis Previo del Código del Programa (Inventario de rutinas, formato I/O…).

Análisis de Uso

Uso del simulador

El uso del simulador es sencillo, puesto que consta de un total de cuatro pasos, estos son:

1. Abrir una consola de comandos y situarnos en el directorio principal del simulador, donde podemos encontrar los subdirectorios donde se dividen los distintos ficheros del simulador según su función.
2. Ejecutar el comando “make” en la consola de comandos, lo cual recurrirá al makefile para realizar la compilación del código fuente del simulador.
3. Ejecutar el binario compilado en el paso previo de la forma: “./bin/sim”.
4. Los resultados de los experimentos se mostrarán por pantalla.

Especificación de carga de trabajo

La especificación de la carga de trabajo solo se puede modificar accediendo al fichero “main.cpp” que se encuentra bajo el directorio “src”. En este fichero se especifican las latencias de los eventos (Miss de L1, Miss de L2…) y de los tipos de instrucciones soportados en la función “read”. Para poder modificar la carga de trabajo de cada *núcleo*, se debe modificar en la función “fill”. Tras estas modificaciones era necesario recompilar la aplicación.

Ficheros de entrada

Los ficheros de entrada del simulador son ajenos al usuario en lo que respecta a la ejecución, puesto que estos ficheros ya se encuentran especificados en el código principal.

El fichero “Bench-Id” se encuentra en el directorio principal del simulador, y sirve como diccionario para el simulador a la hora de traducir el nombre del benchmark a su ID correspondiente.

Por otra parte, se puede encontrar el directorio “input”, en el cual se encuentra una copia del fichero “Bench-Id” y el subdirectorio “EPDB”, que concentra todos los benchmarks.

Estructura de directorios

Dentro del directorio principal del simulador SIM1 se pueden encontrar varios ficheros y directorios que conforman el código del simulador.

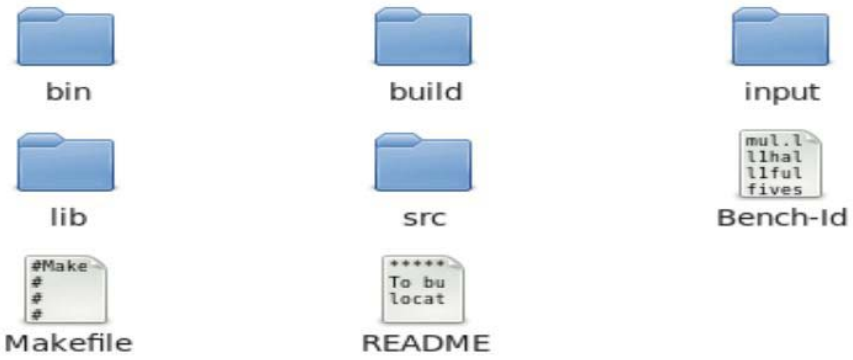


Ilustración 1 – Directorio Principal SIM1

Se puede observar que, en cuanto a ficheros, dispone de un "ReadMe" en el que se describe información básica del simulador; se encuentra el fichero Makefile, en el cual se encuentra la configuración del comando "make" además de otros como "clear", y permite la compilación del simulador guardando el binario resultante en el directorio "bin"; el fichero "Bench-Id" dispone de un listado completo de parejas con el formato "nombre_benchmark,id_benchmark" que sirve como diccionario a la hora de traducir del nombre al ID del benchmark.

En cuanto a los directorios, se dividen por funcionalidades:

- "bin": Directorio donde se aloja el binario compilado mediante el comando "make".
- "build": Directorio donde se almacenan los objetos compilados del simulador.
- "input": Directorio donde se alojan los ficheros que sirven como datos de entrada para el simulador.
- "lib": Directorio donde se almacena el simulador como librería.
- "src": Directorio donde se encuentra el código fuente del simulador.

Estadísticas y valores resultantes.

El simulador SIM1 calcula estadísticas para las cache L1 de instrucciones y de datos de cada núcleo, así como para la cache L2 compartida.

Los valores calculados en el simulador son los siguientes:

- **Number of accesses:** Número de accesos que se producen en la cache correspondiente. Dentro de los accesos a la cache, se pueden distinguir dos tipos: Lectura y Escritura.
- **Number of read accesses:** Número de accesos de tipo lectura que se producen en la cache.
- **Number of write accesses:** Número de accesos de tipo escritura que se producen en la cache.
- **Number of miss reads:** Número de accesos de tipo lectura que han producido "miss" en la cache.
- **Number of miss writes:** Número de accesos de tipo escritura que han producido "miss" en la cache.
- **Number of evicted dirty lines:** Número de líneas sucias desalojadas de la cache L2 y escritas consecuentemente en memoria.
- **IPCs in Isolation:** IPC en Isolación para cada núcleo.
- **IPCs in Competition:** IPC en contención para cada núcleo.

En la siguiente tabla se puede mostrar qué estadísticas calcula el simulador para cada uno de los componentes, L1 Instrucciones, L1 Datos, L2 e Información de Núcleo.

	L1 Instructions	L1 Data	L2	Core Information
Number Acceses	✓	✓	✓	
Number Read Acesses	✓	✓	✓	
Number Write Acesses	✓	✓	✓	
Number Miss Reads	✓	✓	✓	
Number Miss Writes	✓	✓	✓	
Number Evicted Dirty Lines			✓	
IPCs in Isolation				✓
IPCs in Competition				✓

Tabla 2 – Estadísticas SIM1

Todos los resultados se muestran por pantalla, de forma ordenada por experimento y núcleos. No hay otra forma de sacar los resultados y no se guardan en ningún sitio.

Sobre la ejecución el único control que existe es elegir la cantidad de tests a realizar, y qué benchmarks usar sobre cada núcleo. Al ejecutar cada test, se ejecuta la totalidad de cada benchmark, empezando todos al unísono. Una vez acabe la TUA o un TC, el núcleo correspondiente será marcado como "done" y no se tendrán en cuenta las estadísticas obtenidas tras acabar la primera ejecución. Una vez acaben todos los núcleos de ejecutar su benchmark correspondiente, la simulación se da por finalizada.

Análisis previo del Código del Programa.

Listado

Como resumen del análisis realizado, se describe en este apartado las rutinas y funciones existentes.

Main

Es el programa principal, se encarga de organizar los datos de entrada para el simulador y llama a las rutinas de inicialización y simulación. Dispone de un total de tres funciones propias:

- **read():** Esta función se encarga de establecer los parámetros de latencia.
- **fill():** Al iniciarse esta rutina, se rellena un vector con los nombres de los benchmarks que se van a similar, y, tras esto, traduce dichos nombres por su IDs correspondientes que serán almacenados en otro vector que servirá como entrada para el simulador. Esta función se vale del fichero Bench-Id para poder realizar la traducción de nombre a ID.
- **main():** Es la función principal, la cual se encarga de inicializar el "Trace Directory", o directorio de trazas, y llama al inicio de la simulación.

HLIM

Se trata del manejador de simulación, su cometido es servir como herramienta al programa principal *Main* para la inicialización del "Trace Directory" y para iniciar la simulación. A su vez, HLIM recurre a SEM para ejecutar la simulación propiamente dicha. Las funciones definidas en HLIM son:

- **hlim_initialize()**: Esta función se encarga de inicializar las estructuras de datos y de generar los objetos HLIM.
- **hlim_start():** Inicia HLIM.
- **hlim_stop():** Detiene HLIM.
- **hlim_thread_func():** Esta función no se encuentra implementada.
- **hlim_execution_profile_start():** Inicia la ejecución del proceso de generación de perfil para un objeto UoPManager dado.
- **hlim_execution_profile_end():** Finaliza la ejecución del proceso de generación de perfil para un objeto UoPManager dado.
- **hlim_execution_profile_instruction():** Guarda el rastreo de instrucciones del UoP y genera valores estadísticos necesarios.
- **hlim_simulation_execution():** Ordena HLIM para realizar simulaciones de ciertas UoPs y provee de estimaciones de IPC.

Utils

Dispone de diversas funciones que sirven como un conjunto de herramientas para obtener, por ejemplo, los valores que se encuentran en el diccionario global o para calcular la media de un histograma.

- **Públicas:**
    - **compute_Mean():** Obtiene la media de un histograma.
    - **getValueS():** Obtiene el valor del diccionario global.
    - **getValueI():** Obtiene el valor del diccionario local.
    - **old_cpi():** Calcula el antiguo valor de CPI

EPGM

Su función principal es el manejo de entrada y salida de datos, puesto que se encarga de leer los ficheros de traza y de rellenar los buffers con los valores de entrada debidos.

- **Públicas:**
  - **EPGM():** Constructor del objeto EPGM.
  - **~EPGM():** Destructor del objeto EPGM.
  - **start_profile():** Función a la que recurre "hlim_execution_profile_start" para iniciar la ejecución del proceso de generación de perfil.
  - **generate_profile():** Función a la que recurre "hlim_execution_profile_instruction" para guardar el rastreo.
  - **end_profile():** Función a la que recurre "hlim_execution_profile_end" para finalizar la ejecución del proceso generador de perfil.
  - **end_heavy_sim():** Función a la que recurre "initSim" para finalizar "heavy_sim".
  - **read_epdb_heavy_entry():** Función para leer los datos del fichero EPDB en caso de "heavy entry".
  - **read_epdb_light_entry():** Función para leer los datos del fichero EPDB en caso de "heavy entry".

- **Privadas:**
  - **read_epdb():** Función utilizada por el constructor EPGM() para clasificar los ficheros que se encuentran en el directorio EPDB entre los tipos "heavy" y "light".
  - **getline():** Obtiene una nueva línea y retorna "0" en el caso de que no queden más líneas por leer.

SEM

Dispone de tres funciones, de las cuales dos son el constructor y el destructor, y la otra es "simulate". SEM se encarga de inicializar los valores de la simulación, así como de comprobar qué núcleos serán utilizados en la simulación, contabilizándolos y pasando esta información a la función de inicio de simulación del objeto NGMPSim creado.

- **Públicas:**
  - **SEM():** Constructor del objeto SEM.
  - **~SEM():** Destructor del objeto SEM.
  - **simulate():** Se encarga de inicializar las variables necesarias y de iniciar la simulación recurriendo a la función "initSim()".

DetailedModel

Es el fichero principal del simulador. En él se realiza la parte de cómputo de la simulación lanzada desde SEM.

- **Publicas:**
  - **NGMPSim():** Constructor del objeto NGMPSim.
  - **NGMPSim():** Constructor Copy del objeto NGMPSim.
  - **NGMPSim():** Destructor del objeto NGMPSim.
  - **initSim():** Inicia la simulación, usa como entrada el ID de los benchmarks a ejecutar, y como resultado escribe el IPC en Isolation (aislamiento) y en Competition (competición) en vectores.
  - **resetCaches():** Recurre a la función "CacheSim::invCache()" para invalidar la cache y resetea las variables utilizadas por la simulación.
  - **printStatistics():** Recurre a la función "CacheSim::printStatistics" para obtener estadísticas de los accesos a memoria cache.
- **Privadas:**
  - **fill_inst():** Se encarga de rellenar con datos los buffers utilizados por la simulación.
  - **check_done():** Comprueba si todos y cada uno de los núcleos que son utilizados en la simulación han finalizado la ejecución del benchmark asignado.

StatsModule

Graba estadísticas complejas de los accesos a cache tales como distancia de la pila o distancia reutilizada.

- **Públicas:**
  - **StatsModule():** Constructor del objeto StatsModule.
  - **~StatsModule():** Destructor del objeto StatsModule.
  - **track():** Inicia el rastreo de datos y su almacenamiento en variables para su posterior volcado.
  - **end_track():** Finaliza el rastreo borrando los datos que se han usado en el proceso de rastreo.
  - **put_stats():** Vuelva los valores obtenidos en distintas representaciones, como puede ser un histograma de distancia de la pila (Stack Distance Histogram) o un histograma de aciertos en los accesos a cache (Access Histogram for Hits).

CacheSim

Es el simulador de cache, el cual maneja los accesos a la cache privada (nivel 1) y compartida (nivel 2). Actúa como simulador de temporización para los accesos de memoria, contabilización de latencias y también maneja los buffers de escritura e cada núcleo.

- Publicas:
    - **CacheSim():** Constructor del objeto CacheSim para un número de núcleos igual a 1.
    - **CacheSim():** Constructor del objeto CacheSim para un número de núcleos dado.
    - **CacheSim():** Constructor Copy del objeto CacheSim.
    - **~CacheSim():** Destructor del objeto CacheSim.
    - **void instruction_access():** Acceso a cache de instrucciones en Isolation.
    - **void data_access():** Acceso a cache de datos en Isolation.
    - **int instruction_access():** Acceso a cache de instrucciones en Competition.
    - **int data_access():** Acceso a cache de datos en Competition.
    - **invCache():** Invalida las caches.
    - **printStatistics():** Devuelve estadísticas obtenidas de las caches.
    - **getL2Stats():** Obtiene estadísticas de la cache L2.
    - **getdcStats():** Obtiene estadísticas de la cache L1 de Datos.
    - **geticStats():** Obtiene estadísticas de la cache L1 de Instrucciones.
    - **flushBuffers():** Función pública por la cual se puede invocar a la función flushBuffer(), la cual es privada, pasando como parámetro el núcleo deseado.
    - **getSetHistogram():** Obtiene un histograma de Set (Histogram of set).
    - **getStackDistance():** Obtiene un histograma de distancia de pila (Stack Distance).
    - **getSetDistance():** Obtiene un histograma de distancia de set (Set Distance).
    - **getSetDistanceHits():** Obtiene un histograma de distancia de set para los hits (Set Distance for hits).
    - **getFoa():** Obtiene un histograma de la frecuencia de accesos por instrucción.
    - **getLor():** Obtiene un histograma de longitud de peticiones a la cache L2.
    - **arbitrationRound():** Ajusta las prioridades para acceder al bus.
      check_bus_free(): Comprueba si el bus se encuentra libre, de ser así, realiza un flush parcial del buffer de escritura.

- **Privadas:**
    - **partial_flush():** Realiza un flush parcial del buffer de escritura de un núcleo dado.
    - **flush_buffer(int core):** Realiza un flush del buffer de escritura de un núcleo dado.
    - **flush_buffer(int core, unsigned long time_stamp):** Realiza un flush del buffer de escritura de un núcleo dado y calcula estadísticas para posterior uso.
    - **compute_memory_timing():** Calcula la cantidad de ciclos que pasan desde que se lanza el acceso a memoria, teniendo en cuenta el tiempo que tarda hasta que se pueda acceder a la cache L2, hasta que el acceso termina.
- **Funciones no pertenecientes a la clase CacheSim**
    - **RoundRobin:** Esta función sirve como método de comparación para la función "sort".

L1cache

Modela y mantiene el estado interno de una cache con política writethrough con reemplazo LRU.

- **Públicas:**
    - **L1cache():** Constructor del objeto L1cache.
    - **L1cache():** Constructor Copy del objeto L1cache.
    - **~L1cache():** Destructor del objeto L1cache.
    - **read():** Lee una posición de memoria y retorna una pareja "(resultado de acceso[HIT/MISS] - tag)". Además, acumula estadísticas para su posterior uso.
    - **write():** Escribe en una posición de memoria y retorna una pareja "(resultado de acceso[HIT/MISS] - tag)". Además, acumula estadísticas para su posterior uso.
    - **invCache():** Invalida las líneas de cache, adicionalmente, resetea estadísticas y el indicador "lru".
    - **printStats():** Imprime por pantalla las estadísitcas:
        - Número de accesos.
        - Número de accesos de lectura.
        - Número de accesos de escritura.
        - Número de lecturas MISS.
        - Número de escrituras MISS.
    - **getStats():** Guarda las estadísitcas "Nº de accesos de lectura", "Nº de lecturas MISS", "Nº de accesos de escritura" y "Nº de escrituras MISS" en un vector de estadísticas.
- **privadas:**
    - **increase_lru():** Actualiza el valor del campo "lru" de las vías de un set dado.

L2cache

Modela y mantiene el estado interno de una cache con política writeback con reemplazo LRU.

- **Públicas:**
  - **L2cache():** Constructor del objeto L2cache.
  - **L2cache():** Constructor Copy del objeto L2cache.
  - **~L2cache():** Destructor del objeto L2cache.
  - **read():** Lee una posición de memoria y retorna una pareja "(resultado de acceso[HIT/MISS/DIRTY_MISS] - tag)". Además, acumula estadísticas para su posterior uso.
  - **write():** Escribe en una posición de memoria y retorna una pareja "(resultado de acceso[HIT/MISS/ DIRTY_MISS] - tag)". Además, acumula estadísticas para su posterior uso.
  - **invCache():** Invalida las líneas de cache, adicionalmente, resetea estadísticas y los indicadores "lru" y "dirty".
  - **printStats():** Imprime por pantalla las estadísitcas:
    - Número de accesos.
    - Número de accesos de lectura.
    - Número de accesos de escritura.
    - Número de lecturas MISS.
    - Número de escrituras MISS.
    - Número de líneas "dirty" desalojadas.
  - **getStats():** Guarda las estadísitcas "Nº de accesos de lectura", "Nº de lecturas MISS", "Nº de accesos de escritura" y "Nº de escrituras MISS" en un vector de estadísticas. No guarda el número de líneas "dirty" desalojadas.
- **privadas:**
  - **increase_lru():** Actualiza el valor del campo "lru" de las vías de un set dado.

Estructuras de datos.

El conocimiento de las estructuras de datos usadas internamente por el simulador resulta fundamental para poder llevar a cabo las modificaciones necesarias. Seguidamente se detallan estas estructuras de datos:

Main

- **test**: Contiene los distintos experimentos que se van a realizar (definidos mediente los nombres de los benchmarks).
- **test_case**: Contiene los distintos experimentos que se van a realizar (definidos mediante el ID traducido).
- **test_case_final**: Vector que contiene el id de los benchmarks a ejecutar para un "test_case" en concreto.
- **test**: Contiene los distintos experimentos que se van a realizar (definidos mediante los nombres de los benchmarks).
- **ipc_solo**: Vector de floats que contiene el valor del IPC para cada núcleo en "Isolation".
- **ipc_cmp**: Vector de floats que contiene el valor del IPC para cada núcleo en "Competition".
- **configHLIM**: Mapa de claves y valores de tipo string que contiene como clave el tipo de evento o instrucción, y como valor la latencia que tarda dicho evento o instrucción.

Utils

- **trace_struct:** Contiene la dirección de la instrucción, la dirección de datos si es pertinente y el nombre del tipo de instrucción.
- **epdb_entry:** Contiene la información necesaria para realizar simulaciones ligeras y medias (Light y Half).
- **pe_instruction_type_t:** Enumerador utilizado por la función "old_cpi" para comparar valores pasados por parámetro. Contiene los tipos de instrucciones en la enumeración.

EPGM

- **buffer_struct:** Estructura de datos utilizada para manejar los datos en los ficheros de entrada/salida.

SEM

- **UoPManager:** Objeto EPGM que recibe el constructor del objeto SEM. Una vez inicializados los vectores de contadores, se crea un objeto NGMPSim usando como parámetro el mismo UoPManager y el número de núcleos. Sobre este objeto lanzaremos la función “initSim” para iniciar la sección de cómputo de la simulación.

DetailedModel

- **cmp_Cache:** puntero a un objeto de tipo CacheSim, específicamente, al que se usa para la simulación en “Competition”.
- **isol_Cache:** vector de punteros a objetos de tipo CacheSim, específicamente, a los que se usan para la simulación en “Isolation”.
- **core_done:** Vector de booleanos que indican si el núcleo “i” (siendo esta la posición en el vector) ha finalizado la ejecución de todas las instrucciones de su carga de trabajo.
- **n_inst_left:** Vector de enteros que indica la cantidad de instrucciones restantes. Es usada en la función “fill_inst”.
- **cmp_inst_left:** Vector de booleanos que indican si para el núcleo “i” (siendo esta la posición del vector) quedan instrucciones en “Competition”.
- **isol_inst_left:** Vector de booleanos que indican si para el núcleo “i” (siendo esta la posición del vector) quedan instrucciones en “Isolation”.
- **cmp_stamp:** Vector de long que contiene un “timestamp” que indica el tiempo de ejecución de cada núcleo en “Competition”.
- **inst_executed:** Vector de enteros que indica el número de instrucciones ejecutadas para cada núcleo.
- **sim_time:** timestamp que indica el tiempo total de ejecución de la simulación.

StatsModule

- **StatsUnit:** Estructura utilizada por la función “track()”,permite recabar información como el número de instrucciones. También es utilizada por otras funciones como “put_stats” a la hora de plasmar las estadísitcas en histogramas.
- **sdinfo:** Estructura de datos que contiene un puntero a un vector de enteros y un puntero a un mapa de <int,int>. Se usaba para los histogramas de set y distancia de la pila (Stack distance & Set Histogram), sin embargo, esta parte del código se trasladó a CacheSim

CacheSim

- **num_cores:** número de núcleos que son utilizados en el experimento.
- **IC:** Vector de punteros a objetos L1cache que representan las caches de instrucciones de los núcleos. Se trata de una cache con política “write-through no allocate”.
- **DC:** Vector de punteros a objetos L1cache que representan las caches de datos de los núcleos. Se trata de una cache con política “write-through no allocate”.
- **L2C:** Puntero a objeto L2Cache que representa la cache de nivel 2 compartida por todos los núcleos. Se trata de una cache con política “copy-back allocate”.
- **write_buffer:** Se trata de un vector de punteros unsigned int. Cada puntero señaliza a otro vector el cual está adjudicado a un núcleo en concreto. Cada vector señalizado por un puntero representa el vector de direcciones a escribir en la cache L2.
- **previous:** pareja de datos de tipo char que informa sobre el acceso previo, más específicamente sobre si ha sdo un acierto, fallo o fallo en sucio en la cache L2 y si fue al leer o al escribir.
- **l2_stamp:** Time stamp del último acceso a la cache L2.
- mem_stamp: Time stamp del último acceso a memoria principal.
- **l1_stamp:** Vector de time stamps de la cache L1.
- **results:** vector de parejas de enteros que contiene la pareja resultante (HIT/MISS/MISS_DIRTY – READ/WRITE) del acceso a cache L2.
- **sdinfo:** Estructura de datos que contiene un puntero a un vector de enteros y un puntero a un mapa de <int,int>. Se usa para los histogramas de set y distancia de la pila (Stack distance & Set Histogram).

L1cache

- **l1cinfo:** Estructura de datos que contiene información sobre la memoria cache. Se implementa como una matriz de [NSets]x[NWays], por lo que cada línea de cache (cada posición de la matriz) dispone de la siguiente información:
    - **tag:** tag de la línea.
    - **lru:** indicador con el que saber si es la línea que se va a sustituir en el way al que pertenece
    - **valid:** indicador de validez de la línea cache.
- **cache:** Implementación de la estructura l1cinfo.
- **aread:** número de accesos de lectura.
- **awrite:** número de accesos de escritura.
- **mread:** número de accesos que han producido “miss” en lectura.
- **mwrite:** número de accesos que han producido “miss” en escritura.

L2cache

- **l2cinfo:** Estructura de datos que contiene información sobre la memoria cache. Se implementa como una matriz de [NSets]x[NWays], por lo que cada línea de cache (cada posición de la matriz) dispone de la siguiente información:
    - **tag:** tag de la línea.
    - **lru:** indicador con el que saber si es la línea que se va a sustituir en el way al que pertenece
    - **valid:** indicador de validez de la línea cache.
    - **dirty:** Indicador de suciedad de la línea de cache.
- **aread:** número de accesos de lectura.
- **awrite:** número de accesos de escritura.
- **mread:** número de accesos de lectura que han producido miss.
- **mwrite:** número de accesos de escritura que han producido miss.
- **copy_back:** número de líneas de cache sucias desalojadas y escritas en memoria.

Ficheros de entrada/salida

Como entrada tenemos las trazas que alimentar al simulador, que son archivos comprimidos en formato estándar “gzip” y que se encuentran contenidos dentro del directorio “HLIM_Heavy/input/EPDB”. Además, como fichero auxiliar de entrada disponemos del fichero “Bench-Id”, en el cual se pueden encontrar el nombre e ID de cada una de las trazas que se encuentran dentro del directorio anteriormente mencionado.

No dispone de ningún fichero de salida, sino que toda la información de salida en la versión original del programa es volcada en la salida estándar "stdout", apareciendo por tanto en la consola de comandos. Esto permite que la información pueda ser redirigida a un fichero al ejecutar de la forma: “/bin/sim > outputInfo.txt”.

Formato Salida

Al no haber fichero de salida, no dispone de un formato. Por consola, para cada “test case” se mostrará, por orden, el cálculo del IPC de los cuatro núcleos, por un lado, en “Isolation” y por otro lado en “Competition”.

2.2. Estado Actual SIM2.

En este apartado se describe la situación inicial en la que se encuentra la herramienta de simulación SIM2. Esta herramienta está escrita en Phyton y cuenta con dos versiones, pTC ("*partially Time Composable"*) y fTC ("*fully Time Composable*"), de las cuales se analizará la primera versión puesto que es la que se encuentra dentro del alcance del desarrollo de este TFG. Con el fin de poder comprender el algoritmo de cálculo del pWCET hay que referirse a la referencia bibliográfica [1], se debe analizar el fichero con el código del simulador, "SWRand-pTC-WB-configurable.py", y, además, se debe comprender qué entrada y salida produce esta herramienta de simulación (SIM2). Para poder definir con claridad cómo es el simulador en su versión original, se han decidido comprobar una serie de puntos a tratar. Estos puntos, en líneas generales son:

- Uso del Simulador.
- Análisis Previo de Uso (Especificación de carga de trabajo, Estructura de directorios…).
- Análisis Previo del Código del Programa (Inventario de rutinas, formato I/O…).

Análisis de Uso

Uso del simulador

El uso del simulador es sencillo, puesto que consta de un total de cuatro pasos:

1. Abrir una consola de comandos y situarnos en el directorio donde se halle el código del simulador.
2. Asegurarse de conocer la localización de los ficheros de entrada, por comodidad es aconsejado tener dichos ficheros en el mismo directorio que el código del simulador.
3. Ejecutar el código en Python mediante el comando "python SWRand-pTC-WB-configurable.py -t test.txt -c test.csv -x contender.csv".
4. Los resultados de los experimentos se muestran por pantalla.

Especificación de carga de trabajo

La especificación de la carga de trabajo se especifica de la siguiente forma:

- "-t" para especificar el fichero que contiene los valores de tiempo.
- "-c" que contiene la especificación de la TUA (Task Under Analysis).
- "-x" que contiene la especificación de los TC (Task Contenders).

Ficheros de entrada

Los ficheros de entrada necesitan ser especificados por el usuario del simulador mediante las opciones mencionadas con anterioridad a ejecutar el simulador.

Los nombres de dichos ficheros pueden ser cambiados sin ningún problema, pero se ha de respetar el formato de los mismos.

El fichero de tiempo, "test.txt" en este caso, dispone de dos líneas, una de ellas con el indicador de TimeStamp inicial y dicho TimeStamp, y la otra con el indicador de TimeStamp final y dicho TimeStamp. Lógicamente, los valores de la segunda fila han de ser mayores que los utilizados en la primera fila.

Los ficheros que especifican la TUA y los TC contienen un total de 6 campos, los cuales son:

- **icmiss:** Indica el número de accesos a cache de instrucciones que han producido "miss".
- **dcmiss:** Indica el número de accesos a cache de datos que han producido "miss".
- **store:** Indica el número de instrucciones de "store" ejecutadas.
- **extv01:** Indica el número de "miss" en la cache de nivel L2.
- **fpu:** Campo no utilizado.
- **time:** Contiene el tiempo que ha tardado TUA en realizar la tarea que haya tenido previamente.

Estructura de directorios

El simulador SIM2 no cuenta con ninguna estructura de directorios, se compone del fichero con el código y de los ficheros de entrada que pueden encontrarse en distintos directorios.

Estadísticas y valores resultantes.

El simulador SIM2 calcula el tiempo de ejecución del peor caso (WCET) dadas unas trazas de ejecución. Los valores utilizados para el cálculo del WCET en el simulador son los campos descritos en los ficheros de entrada que especifican la TUA y los TC.

Todos los resultados en tiempo de ejecución se muestran por pantalla, primero el TimeStamp de inicio, y luego el TimeStamp final. No hay otra forma de sacar los resultados y no se guardan en ningún sitio.

Análisis previo del Código del Programa.

Listado

El listado de las funciones que aparecen en el fichero “SWRand-pTC-WB-configurable.py” son:

Error

Función que redirige un texto de error a la salida típica de error “stderr”. Termina la ejecución.

warning

Función que redirige un texto de aviso a la salida típica de error “stderr”. No termina la ejecución.

help

Función utilizada al usar la opción “-h” o “--help", imprime por pantalla el formato admitido en la ejecución del código.

read_time_trace

Lee las tuplas de valores “identificador” y “TimeStamp” del fichero de tiempo mencionado anteriormente.

read_csv_trace

Lee del fichero csv (TUA o TC) los valores para los distintos campos organizándolos pro filas.

delta_ptc

Calcula el WCET a partir de los datos extraídos de los ficheros de entrada.

main

Controla la ejecución general del simulador, captando las opciones de ejecución y tratándolas, también llamada a los métodos pertinentes.

Estructuras de datos.

El listado de las estructuras de datos utilizadas en la ejecución del simulador SIM2 es:

csvfile

String que contiene el nombre del fichero que define a la TUA.

timefile

String que contiene el nombre del fichero de tiempo.

cont

Array al que se le añaden los contendientes.

max_ic

Array al que se le añade el valor máximo de los contadores de “icmiss”.

max_dc

Array al que se le añade el valor máximo de los contadores de “dcmiss”.

max_ms

Array al que se le añade el valor máximo de los contadores de “extev01”.

max_st

Array al que se le añade el valor máximo de los contadores de “store”.

zero

Array al que se le ha añadido un 0 como valor. Se usa para reemplazar las estructuras de datos “max_ic”, “max_dc”, “max_st” y “max_ms” cuando se quiere realizar una ejecución de “delta_ptc” con todos los valores a “0”.

pmcs

Estructura de datos que contiene los valores extraídos del fichero que define a la TUA.

time

Estructura de datos que contiene los valores extraídos del fichero de tiempo.

Ficheros de entrada/salida

Como entrada disponemos de los ficheros de entrada ya descritos anteriormente que describen las trazas de ejecución que utilizará el simulador SIM2. Las trazas de datos se sacan a partir de los datos obtenidos de SIM1.

No dispone de ningún fichero de salida, sino que toda la información de salida del programa es volcada en la salida estándar stdout, apareciendo por tanto en la consola de comandos. Para almacenarlo en ficheros habría que desviar los stdout y stderr.

Formato Salida

Una vez se ejecute SIM2, por pantalla se muestra el "timestamp" de inicio y el "timestamp" de fin. Este último se ve modificado puesto que se le suma el valor calculado como “WCET”.

Capítulo 3

Validación WCET

Para poder validar los valores obtenidos por los simuladores en su estado inicial, se ha procedido a realizar una serie de pasos con los cual se podrán verificar el correcto funcionamiento de las herramientas de simulación. Por ello, se ha realizado cuatro pasos:

1. Obtención de Trazas de SIM1 con los benchmarks en “Isolation”.
2. Emparejamiento y obtención de resultados de SIM2.
3. Obtención de resultados en modo “Competition” de las parejas de benchmarks.
4. Validación del WCET obtenido en SIM2.

3.1. Obtención de trazas de SIM1

Para realizar una comprobación inicial de resultados, se han escogido una serie de benchmarks de la lista que se encuentra en el fichero “Bench-Id”. Una vez se haya elegido los benchmarks a ejecutar, en nuestro caso se utilizan los siguientes:

- Benchmark 102 - “esa-25-25-50.log”.
- Benchmark 106 - “esa-8-12-80.log”.
- Benchmark 142533450 - “esa100%st.log”.

Se procede a introducirlos, de forma “Hard-Coded”, en el fichero “main.cpp”, dentro de la función “fill”, con el siguiente formato:

```
vector<string> aux(4);
    aux[0] ="esa-25-25-50.log"; //102
    aux[1] ="esa-8-12-80.log"; //106
    aux[2] ="";
    aux[3] ="";
    tests->push_back( aux );
    aux[0] ="esa-25-25-50.log"; //102
    aux[1] ="esa100\%st.log"; //142533451
    aux[2] ="";
    aux[3] ="";
    tests->push_back( aux );
    aux[0] ="esa-8-12-80.log"; //106
    aux[1] ="esa-25-25-50.log"; //102
    aux[2] ="";
    aux[3] ="";
    tests->push_back( aux );
    aux[0] ="esa-8-12-80.log"; //106
    aux[1] ="esa100\%st.log"; //142533451
    aux[2] ="";
    aux[3] ="";
    tests->push_back( aux );
```

Ilustración 2 – Configuración Hard-Coded SIM1

Como se puede observar en la imagen superior, hay que introducir el nombre completo del benchmark que se desea emplear para cada núcleo, correspondiendo, de manera ordenada, las posiciones del vector auxiliar "aux" con el número de núcleo, siendo, por tanto, "aux[0]" el que define el benchmark a ejecutar como TUA. Cada vez que se rellena el vector auxiliar, se almacena en "tests", finalizando así la configuración de cada experimento.

Una vez completado este paso, recompilamos la herramienta de simulación SIM1 mediante los comandos "make clean" y "make", creando el ejecutable "sim" dentro del directorio "bin" del directorio principal de SIM1.

Al ejecutar, se puede observar que ninguno de los contadores es mostrado en la salida, y por ello, se ha de modificar el código del fichero "DetailedModel.cpp" para añadir las siguientes líneas de código al finalizar la ejecución:

```
for(int i = 0; i < num_cores; i++){
    isol_Cache[i]->printStatistics();
}
```

Al añadir este bucle, obtenemos los contadores de las caches de L1, tanto IC como DC, y de la L2 para la ejecución en Isolation de cada benchmark, que son los casos que interesan observar en esta situación. Los resultados obtenidos se muestran por la consola y habrá que recopilarlos:

```
Core 0 instruction stats
-----Stats for L1cache------
Number accesses: 3834759
Number read accesses 3834759
Number write accesses 0
Number miss reads 66
Number miss writes 0
Core 0 data stats
-----Stats for L1cache------
Number accesses: 3507289
Number read accesses 3473454
Number write accesses 33835
Number miss reads 3276813
Number miss writes 17431
-----Stats for L2cache------
Number accesses: 3310714
Number read accesses 3276879
Number write accesses 33835
Number miss reads 3276866
Number miss writes 16655
Number evicted dirty lines 16652
```

Ilustración 3 – Salida básica SIM1

3.2. Obtención de resultados de SIM2

Para poder obtener resultados de SIM2, primero hay que alimentarlo con la entrada debida. Actualmente, el comando de ejecución de SIM2 es del tipo:

```
python SWRand-pTC-WB-Configurable.py -c tua.csv -t time.txt -x contender.csv
```

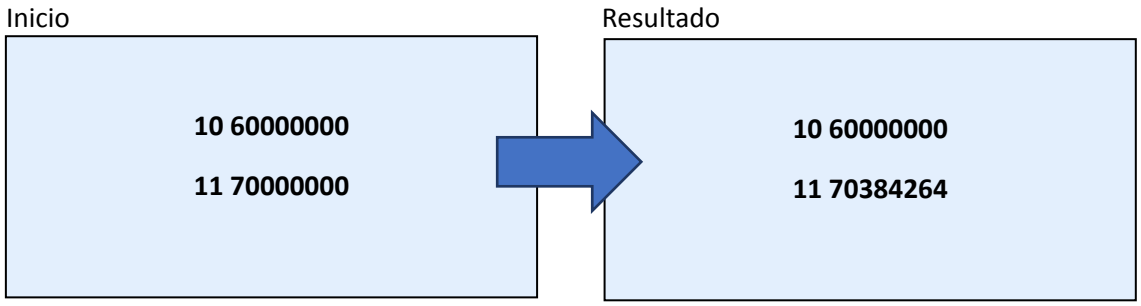
Para poder ejecutar SIM2, por tanto, se ha tenido que crear de forma manual un fichero de extensión “csv” que indique los campos separados por comas, y en la siguiente línea, los valores separados por comas:

```
1 icmiss,dcmiss,store,extev01,fpu,time
2 106,256011,258347,1651,0,2846625
```

Ilustración 4 – Traza de Salida SIM1 / Traza de Entrada SIM2

Estos valores se corresponden con los resultados en Isolation obtenidos anteriormente como resultado de la herramienta de simulación SIM1. De la misma forma obtenemos otro fichero del mismo tipo que contenga los contadores en Isolation para los TC, en este caso, será solamente un TC el que se utilice debido al alcance de trabajo que concerniente al TFG.

Por último, se ha de generar un fichero de extensión ".txt", el cual contiene los marcados sobre los que SIM2 representará el impacto de los accesos del TC sobre la ejecución de TUA. El resultado obtenido por pantalla se dispondrá de la forma:



En este ejemplo, por tanto, se ha producido una contención de 384,264 ciclos.

3.3. Obtención de resultados en "Competition"

Una vez obtenidos los resultados de SIM2, se procede a obtener los resultados de SIM1 concernientes a la ejecución en Contención (*Competition*). Para ello, habrá que modificar nuevamente el código del fichero "DetailedModel.cpp" para introducir la siguiente línea:

```
cmp_Cache ->printStatistics();
```

Esta línea escribirá en la consola los resultados concernientes a la cache de L1, tanto de IC como de DC, y los de L2 con el mismo tipo de formato visto anteriormente en la *Ilustración 3*.

3.4. Validación del WCET

Para realizar la validación, por tanto, se utilizarán algunas parejas de los benchmarks 102, 106 y 142533451 para la configuración de experimentos.

Los resultados en Isolation por test ejecutado son los siguientes:

Test 1 – 102 y 106

Benchmark:	Núcleo 0: 102			Núcleo 1: 106		
Evento\Cache	IC	DC	L2	IC	DC	L2
Nº Accesos	1076009	532071	514464	2122038	451367	413006
Nº Lecturas	1076009	273724	256117	2122038	281383	243023
Nº Escrituras	0	258347	258347	0	169984	169983
Nº Miss Lec.	106	256011	106	106	242917	884
Nº Miss Esc.	0	255686	475	0	163977	1105
Nº Evicted	-	-	0	-	-	0

Tabla 2 – 102 vs 106

Test 2 – 102 y 142533451

Benchmark:	Núcleo 0: 102			Núcleo 1: 142533451		
Evento\Cache	IC	DC	L2	IC	DC	L2
Nº Accesos	1076009	532071	514464	1067473	1029125	1027448
Nº Lecturas	1076009	273724	256117	1067473	10056	117
Nº Escrituras	0	258347	258347	0	1019069	1027331
Nº Miss Lec.	106	256011	106	106	11	117
Nº Miss Esc.	0	255686	714	0	1017044	834
Nº Evicted	-	-	0	-	-	0

Tabla 3 – 102 vs 142533451

Test 3 – 106 y 102

Benchmark:	Núcleo 0: 106			Núcleo 1: 102		
Evento\Cache	IC	DC	L2	IC	DC	L2
Nº Accesos	1081206	229863	210720	551327	271733	259112
Nº Lecturas	1081206	143260	124117	551327	141061	128440
Nº Escrituras	0	86603	86603	0	130672	130672
Nº Miss Lec.	106	124011	106	101	128339	749
Nº Miss Esc.	0	83814	167	0	129010	1413
Nº Evicted	-	-	0	-	-	0

Tabla 4 – 106 vs 102

Test 4 – 106 y 142533451

Benchmark:	Núcleo 0: 106			Núcleo 1: 142533451		
Evento\Cache	IC	DC	L2	IC	DC	L2
Nº Accesos	1081206	229863	210720	1067473	1029125	529150
Nº Lecturas	1081206	143260	124117	1067473	10056	110
Nº Escrituras	0	86603	86603	0	1019069	529040
Nº Miss Lec.	106	124011	106	106	11	110
Nº Miss Esc.	0	83814	369	0	1017044	1211
Nº Evicted	-	-	0	-	-	0

Tabla 5 – 106 vs 142533451

Una vez obtenidos los resultados, se procede al siguiente paso, el cual es obtener el valor del pWCET mediante la herramienta de simulación SIM2. Para ello, dado los datos anteriores, se construirán los ficheros “csv” pertinentes y se reutilizará de forma constante el fichero de marcadores de tiempo usado en el ejemplo previo. El resultado obtenido para cada test son los siguientes:

Test	1	2	3	4
id\Competidores	102/106	102/142533451	106/102	106/142533451
10	60000000	60000000	60000000	60000000
11	72223563	72359588	71228710	71105664

Tabla 6 – Resultados SIM2

Teniendo el incremento en tiempo de ejecución producido por el contendiente a la TUA, calculamos el pWCET sumando el incremento y el tiempo en Isolation de la TUA:

Test	1	2	3	4
Incremento	2223563	2359588	1228710	1105664
Tiempo Base TUA	3141555	3141555	2046610	2046610
pWCET	5365118	5501143	3275320	3252274

Tabla 7 – Cálculo WCET

Para validar estos resultados, se deben de obtener los tiempos en Competition de la TUA para cada experimento realizado y compararlos con los tiempos obtenidos anteriormente.

Test	1	2	3	4
pWCET	5365118	5501143	3275320	3252274
Tiempo Competition	4377399	4663509	2241560	2268676

Tabla 8 – Verificación pWCET

Capítulo 4

Implementación de Mejoras.

En este capítulo se presentan las mejoras realizadas en ambas herramientas de simulación, SIM1 y SIM2. Para poder implementar dichas mejoras, se ha, primero, analizado la herramienta en su estado inicial, y luego en una segunda etapa, diseñado las nuevas funcionalidades a añadir en cada herramienta.

4.1. Mejoras de SIM1

En el simulador SIM1 se han hecho diversas aportaciones en lo que al uso se refiere. Es por ello por lo que cada mejora se expondrá en los siguientes puntos de forma separada.

4.1.1. Fichero de Configuración

El simulador cuenta ahora con un fichero de configuración que facilita al usuario el diseño de los experimentos a ejecutar y la definición de parámetros del programa, como son, por ejemplo, las latencias. Al no tener previamente un fichero de configuración, el usuario se veía obligado a acceder y modificar el fichero "main.cpp" y posteriormente a recompilar, con lo que para realizar varios experimentos sería necesario disponer de varios ejecutables del mismo simulador. Esta implementación se ha logrado mediante el uso de la librería "<libconfig.h++>", consiguiendo así, evitar tener que usar la técnica conocida como "*hard-coded*". Para que la configuración del simulador sea sencilla de entender, se ha confeccionado un manual de usuario que recoge los distintos campos y sus posibles valores.

A continuación, se puede ver un ejemplo de la praxis "hard-coded" que se ha evitado con la implementación de este fichero de configuración:

```
vector<string> aux(4);
    aux[0] ="aifirf01lite.log";
    aux[1] ="l1miss.log";
    aux[2] ="l1miss.log";
    aux[3] ="l1miss.log";
    tests->push_back( aux );
    aux[0] ="aifirf01lite.log";
    aux[1] ="l1miss.log";
    aux[2] ="";
    aux[3] ="";
    tests->push_back( aux );
    aux[0] ="l2miss.log";
    aux[1] ="";
    aux[2] ="";
    aux[3] ="";
    tests->push_back( aux );
```

Ilustración 5 – Ejemplo de la praxis "Hard-Coding"

Dentro del fichero de configuración, cabe destacar la implementación del grupo "experiments", el cual permite al usuario de seleccionar los benchmarks deseados, en qué núcleo se ejecutará, así como la posibilidad de realizar múltiples experimentos independientes de forma sencilla y eficaz. El número de experimentos a realizar se especifica con "n_test_case= 4" y en la estructura test_case se describen cada una de las ejecuciones a llevar a cabo (en este caso 6 parejas de bench0 y bench1).

```
experiments =
{
    n_test_case = 4;
    test_case = ( { bench0 = "esa-25-25-50.log";
                    bench1 = "esa-8-12-80.log";},

                  { bench0 = "esa-25-25-50.log";
                    bench1 = "esa100%st.log";},

                  { bench0 = "esa-8-12-80.log";
                    bench1 = "esa-25-25-50.log";},

                  { bench0 = "esa-8-12-80.log";
                    bench1 = "esa100%st.log";}
                );
};
```

Ilustración 6 – Ejemplo Configuración de tests mediante fichero cfg

Otro aspecto importante del fichero de configuración es el de la posibilidad de especificar qué resultados queremos obtener y en qué directorio y fichero. Esto permite la obtención automática de datos, tanto para realizar trabajos de análisis, como para alimentar la ejecución de SIM2, para obtener el cálculo del WCET.

```
storageConfig =
{
    storeOldCounters = 0;
    storeNewCounters = 0;
    storeDataFlag = 1;
    traceEventFlag = 0;

    oldCountersDirectory = "output/isolation/";
    newCountersDirectory = "output/extended/pair/";
    tableDataFilename = "output/table_data_pairs_f.csv";
    traceEventFilename = "output/trace_event/prueba.txt";
};
```

Ilustración 7 – Ejemplo Configuración de almacenamiento mediante fichero cfg

El detalle de uso se encuentra en el manual de usuario

4.1.2. Cache L2 Particionable

Para poder ofrecer más posibilidades en la experimentación, se ha implementado una opción que permite particionar la cache L2 por vías, teniendo cada núcleo una vía privada. De este modo, los accesos que producen los núcleos no desalojarán de la cache los bloques pertenecientes a otro núcleo.

```
if(fixed){
    //Cambiamos LRU por vía fija
    hit |= ((cache[set][core].tag==tag)&&cache[set][core].valid)*(core+1);
    way = core;
}else{
    for( int i = 0 ; i < WAYSL2 ; ++i){
        hit |= ((cache[set][i].tag==tag)&&cache[set][i].valid)*(i+1);
        min = cache[set][i].lru == 3 ? i : min;
    }
    way = hit ? hit - 1 : min;
}
```

Ilustración 8 – Modificación producida para poder particionar la cache L2

Esta configuración de cache L2 particionada permitirá, por tanto, reducir el conflicto existente entre los núcleos al acceder a la cache L2, sin embargo, esto viene con el costo adicional de que cada núcleo dispondrá de una cuarta parte de la memoria cache, puesto que ahora son particiones privadas.

Este particionado, tendrá como contraparte la reducción del espacio de almacenamiento permitido, y con ello el aumento de misses producidos en la ejecución en Isolation.

Lo interesante será ver si esta configuración beneficia de tal forma que, en una situación de Competition, obtenga unos resultados mejores que los que se podría obtener con una cache L2 no particionada.

4.1.3. Actualización de los PMCs

Como se ha descrito previamente en la sección “Situación Inicial del simulador SIM1”, este simulador cuenta con un total de 4 contadores de monitorización para cache L1 de Instrucciones, otros 4 para la cache L1 de Datos, y otros 5 para la cache L2 común. Estos contadores son:

- **Access Read:** Cantidad de accesos de lectura producidos en la cache pertinente.
- **Miss Read:** Cantidad de accesos de lectura que han generado un fallo en la cache correspondiente.
- **Access Write:** Cantidad de accesos de escritura producidos en la cache pertinente.
- **Miss Write:** Cantidad de accesos de escritura que han generado un fallo en la cache correspondiente.
- **Copy_Back:** (*Exclusivo L2*) Cantidad de líneas de cache desalojadas y escritas en memoria debido a un dirty miss.

El contador “copy_back” es exclusivo a la cache L2 debido a que ésta es una cache de tipo Write-Back con política Allocate. Al disponer de esta configuración, en la cache L2 se poducen los fallos Dirty Miss, sin embargo, este contador no es capaz de discernir entre los Dirty Miss generados por accesos de lectura a los generados por accesos de escritura, y por ello, aunque existe, no se utiliza como salida de cara a reutilizarlo en SIM2. Estos contadores son los que se encuentran disponibles para el procesador de referencia LEON3:

Table 2: PMCs available in the reference processor.

Name	Description
pmc^{icm}	Bus reads caused by ic misses
pmc^{dcm}	Bus reads caused by dc misses
pmc^{st}	Writes to L2
pmc^{m}	Misses in the L2

Ilustración 9 - PMCs LEON3 de [1]

Esta configuración de contadores, sin embargo, conlleva que la estimación del WCET calculado posteriormente sea más pesimista, y es por ello por lo que se ha ampliado el repertorio de contadores de L2 a los disponibles según el tipo de petición (Lectura o Escritura) disponibles en la placa de referencia:

Table 1: Request types and their latency in our reference board.

Type	mcd	Description
sh	$l^{sh} = 1$	L2 st hit
lh	$l^{lh} = 8$	L2 ld hit in L2
lmc	$l^{lmc} = 28$	L2 ld clean miss
smc	$l^{smc} = 28$	L2 st clean miss
lmd	$l^{lmd} = 31$	L2 ld dirty miss
smd	$l^{smd} = 31$	L2 st dirty miss

Ilustración 10 - PMCs Board de [1]

Los contadores obtenidos de esta manera se denominan de la siguiente forma:

- Contadores L1
  - ReadHit
  - ReadMiss
  - WriteHit
  - WriteMiss
- Contadores L2
  - ReadHit
  - WriteHit
  - ReadMiss
  - WriteMiss
  - ReadDirtyMiss
  - WriteDirtyMiss

La configuración actual permite obtener los resultados de ambas versiones al mismo tiempo o por separado, sin que estas posibilidades afecten de alguna manera a los datos obtenidos.

4.1.4. Almacenamiento en fichero

Para poder visualizar de forma correcta la salida de SIM1 para las ejecuciones en Isolation, se ha introducido un sistema de almacenado de contadores en ficheros “csv” para su posterior uso en SIM2. El sistema es capaz de extraer los valores de contadores antiguos y nuevos, escribiéndolos en directorios definidos por el usuario. El formato del nombre del fichero de salida es del tipo: “benchID.csv”.

Versión PMCs Antiguos:

	Standard	Standard	Standard	Standard	Standard	Standard
1	icmiss	dcmiss	store	extev01	fpu	time
2	287	155	40648	477	0	139551

Ilustración 11 – Traza de configuración de PMCs base

Versión PMCs Nuevos:

Cache IC L1:

	Standard	Standard	Standard	Standard
1	ic_ReadHit	ic_ReadMiss	ic_WriteHit	ic_WriteMiss
2	914329	287	0	0

Ilustración 12 – Traza de configuración de PMCs actualizados (ICL1)

Cache DC L1:

Standard	Standard	Standard	Standard
dc_ReadHit	dc_ReadMiss	dc_WriteHit	dc_WriteMiss
65159	155	40648	2136

Ilustración 13 – Traza de configuración de PMCs actualizados (DCL1)

Cache L2:

Standard	Standard	Standard	Standard	Standard	Standard	Standard
L2_ReadHit	L2_ReadMiss	L2_ReadDirtyMiss	L2_WriteHit	L2_WriteMiss	L2_WriteDirtyMiss	time
442	359	40648	118	0	0	139551

Ilustración 14 – Traza de configuración de PMCs actualizados (L2)

4.2. Mejoras SIM2

Las mejoras de SIM2 se centran, principalmente, en la mejora de la estimación del pWCET a raíz de los PMCs obtenidos mediante SIM1, para ello, se han implementado las siguientes mejoras:

- Actualización del código para ser capaz de procesar los nuevos contadores.
- Salida con el cálculo del pWCET realizado en vez de utilizar "Stamps", marcadores, para representar el incremento en tiempo de ejecución generado por el conflicto de la *"Task Under Analysis"* contra los *"Task Contenders"*.
- Volcado de resultados en ficheros para su almacenamiento.

4.2.1. Integración de los PMCs actualizados

Para poder integrar los PMCs en la herramienta de simulación SIM2, se han modificado las rutinas de lectura de ficheros de entrada, así como la rutina de cómputo. Además, se han incluido las latencias correspondientes a cada contador de monitorización según la placa de referencia.

El cómputo del incremento para cada *"Task Contender"* se lleva a cabo en la función "**delta_ptc**". Esta función se encarga de estimar el incremento en ciclos que sufrirá la TUA para un contender dado. Esto se realiza observando la cantidad de eventos capturados en cada contador, así como el valor de latencia para cada uno. De esta forma, se van contabilizando los eventos de forma ordenada según sus latencias, primero los de mayor coste en ciclos, y posteriormente, los de menor coste.

	τ_b requests conflicting with τ_a requests	τ_a unpaired requests
before pairing		n_a^i
Pairing md	$\hat{c}^{md} = \min(n_a, \hat{n}_b^{md})$	$n_a'^i = \max(0, n_a^i, -\hat{c}^{md})$
Pairing mc	$\check{c}^{mc} = \min(n_a'^i, \check{n}_b^{mc})$	$n_a''^{i,} = \max(0, n_a'^i - \check{c}^{mc})$
Pairing lh	$\hat{c}^{lh} = \min(n_a''^i, \hat{n}_b^{lh})$	$n_a'''^i = \max(0, n_a''^{i,} - \hat{c}^{lh})$
Pairing sh	$\check{c}^{sh} = \min(n_a'''^i, \check{n}_b^{sh})$	$n_a''''^i = \max(0, n_a'''^i - \check{c}^{sh})$
after pairing		$n_a''''^i$

Ilustración 15 - Procedimiento cálculo de Estimación by [1]

4.2.2. Clarificación de la salida de SIM2

Como ya se ha mencionado, la salida consistía en un par de "Stamps", de marcadores, con unos ID, estos marcadores permitían al usuario conocer el incremento en tiempo de ejecución que los TC generan sobre la TUA. Sin embargo, el usuario era responsable directo de tener en cuenta la diferencia entre marcadores y de obtener el incremento dado observando el cambio producido en el marcador final.

Para hacer legibles estos datos de una manera más sencilla, se han dispuesto los resultados de la siguiente forma:

- **baseT** = XXXXXX → Indica el tiempo en **Isolation de la TUA**.
- **Delta** = YYYYYY → Indica el **incremento** en tiempo de ejecución generados por los *"Task Contenders"*.
- **pWCET** = ZZZZZZ → Muestra la estimación final del **WCET**.

Adicionalmente, se ha eliminado el fichero especificado mediante la opción "-t" puesto que ya no es necesario en el cómputo.

4.2.3. Almacenamiento en fichero

Si el usuario de la herramienta de simulación SIM2 desea salvaguardar la estimación obtenida tras la ejecución de SIM2, puede utilizar una nueva opción implementada, “-d” ó “--dump”. Esta opción es prescindible, pero de ser especificada, almacenará en un fichero con el nombre dado por el usuario la salida del programa.

Un comando de ejecución ejemplo sería:

```
python SWRand-pTC-WB-configurable.py -c ../SIM1/output/Isolation/106.csv
-d sim2_106_102_output.txt -x ../SIM1/output/Isolation/102.csv
```

4.2.4. Script ejecuciones múltiples SIM2

Se ha creado un Script en Python el cual lee líneas de configuración de un fichero de entrada y redirige los datos leídos a ejecuciones sucesivas de SIM2 con el fin de agilizar y automatizar la puesta en ejecución, obtención de resultados y organización de los mismos. La línea del fichero de entrada se compone de los siguientes campos:

- Tua: Ruta relativa o absoluta del fichero que contiene los PMC de la TUA.
- Dump: Fichero en el que volcar la información de salida de cada ejecución.
- Tc0: Task contender 0.
- Tc1: Task contender 1.
- Tc2: Task contender 2.
- Versión: Flag que indica la versión de SIM2 a ejecutar. Un 0 significa versión antigua, simulando los 4 PMCs, y un 1 significa versión nueva, utilizando los 6 PMCs para la estimación.

El resultado de las sucesivas ejecuciones de SIM2 se formaterá en una matriz adecuada al tamaño especificado por el usuario. El uso de este Script se puede revisar en el “Manual de Usuario”.

Capítulo 5

Experimentación

Debido a la extensión que abarca este Trabajo de Fin de Grado, no se puede experimentar con todos los benchmarks disponibles, es por ello que se han clasificado según su duración temporal, y se han escogido un total de 9 benchmarks pertenecientes al paquete de benchmarks "EEMBC", los cuales se emparejarán con el fin de estimar el pWCET dadas las nuevas modificaciones. Los benchmarks escogidos para realizar pruebas son los siguientes:

- Benchmark 152343765 – "bitmnp01lite"
- Benchmark 152343767 – "canrdr01lite"
- Benchmark 152343769 – "pntrch01lite"
- Benchmark 152343771 – "idctrn01lite"
- Benchmark 152343775 – "a2time01lite"
- Benchmark 152343777 – "basefp01lite"
- Benchmark 152343778 – "ttsprk01lite"
- Benchmark 152343780 – "cacheb01lite"
- Benchmark 152343782 – "puwmod01lite"

El procedimiento seguido para obtener la estimación del pWCET de todas las combinaciones posibles es:

1. Configuración del experimento en SIM1.
2. Obtención de los contadores resultantes de SIM1.
3. Configuración del experimento en SIM2.
4. Obtención de la estimación resultanto de SIM2.

5.1. Configuración y obtención de resultados – SIM1

Dados estos benchmarks, se modifica el fichero de configuración de SIM1 para poner cada pareja a testear. En este caso, se ha establecido que la cache L2 compartida se encuentre particionada por vías, además de escribir los contadores de cada núcleo de cada test ejecutado. Esta configuración final, se puede encontrar en el documento anexado a la memoria "*configuration_example.cfg*".

5.2. Configuración y obtención de resultados – SIM2

Se utiliza el script desarrollado para lanzar múltiples ejecuciones de SIM2 con los contadores obtenidos en SIM1, tanto para la versión de 4 contadores como la versión de 6 contadores. Los ficheros resultantes de salida muestran cada uno, una matriz cuadrada de dimensiones 9x9, en la que tendremos las TUA como filas y las TC como columnas.

Versión 4 PMCs:

		Task Contender								
		152343765	152343767	152343769	152343771	152343775	152343777	152343778	152343780	152343782
Task Under Analysis	152343765	9845706	9722051	9545111	9666314	9616342	9687358	9695915	11014111	10273459
	152343767	1992679	5534742	2251939	2657714	2395771	2066219	2770106	4450941	3299488
	152343769	3905442	4455379	4374091	4473416	4385349	4036559	4444102	5018939	4658646
	152343771	10305399	12337716	11678228	12252138	12200426	10464000	12315969	13704116	12898475
	152343775	1959802	2355702	2225692	2417165	2300874	2039369	2376005	2667233	2455392
	152343777	3277709	3312531	3238603	3339441	3247698	3275652	3306754	3766924	3481510
	152343778	4739495	6322703	5369523	5807376	5667933	4854015	6295134	6942153	6565437
	152343780	2345535	4759610	2648165	3067739	2813771	2404891	3231595	5078899	3872987
	152343782	2399355	3828504	2704893	3125583	2874849	2447397	3296413	4171353	3934583

Tabla 9 –pWCET con 4 PMCs

Versión 6 PMCs:

		Task Contender								
		152343765	152343767	152343769	152343771	152343775	152343777	152343778	152343780	152343782
Task Under Analysis	152343765	9604875	9513185	9504104	9526026	9514389	9552253	9529193	9801884	9520525
	152343767	1940922	5459801	2213508	2540195	2352383	2000933	2688988	4142422	3124478
	152343769	3840986	4341124	4332464	4350608	4330675	3946602	4345890	4465108	4339984
	152343771	10176725	12104737	11614969	12106543	12095016	10325218	12127369	12373465	12104239
	152343775	1908306	2279816	2184311	2296960	2266202	1972661	2291891	2359931	2281812
	152343777	3215681	3206460	3196601	3215723	3195071	3211458	3210516	3317340	3204151
	152343778	4665053	6192415	5312529	5676044	5601972	4756466	6192320	6332197	6194699
	152343780	2289206	4668967	2606151	2948149	2765582	2337067	3145479	4745335	3672748
	152343782	2343288	3738633	2664356	3006034	2828860	2381973	3211281	3823143	3732277

Tabla 10 –pWCET con 6 PMCs

De estas matrices podemos obtener los tiempos normalizados según el tiempo en Isolation de la TUA. Al ser normalizados, las matrices se quedan de la forma:

Versión 4 PMCs:

		Task Contender								
		152343765	152343767	152343769	152343771	152343775	152343777	152343778	152343780	152343782
Task Under Analysis	152343765	1,96192	1,93728	1,90202	1,92618	1,91622	1,93037	1,93207	2,19475	2,04716
	152343767	2,01991	5,72018	2,28272	2,69404	2,42851	2,09446	2,80797	4,51177	3,34458
	152343769	1,98868	2,26872	2,22732	2,27790	2,23306	2,05545	2,26297	2,55569	2,37222
	152343771	1,96404	2,35136	2,22568	2,33505	2,32520	1,99426	2,34722	2,61178	2,45824
	152343775	1,99611	2,39935	2,26693	2,46195	2,34350	2,07715	2,42003	2,71665	2,50088
	152343777	1,97928	2,00031	1,95567	2,01656	1,96116	1,97804	1,99682	2,27470	2,10235
	152343778	1,90813	2,54554	2,16179	2,33807	2,28193	1,95424	2,53444	2,79493	2,64326
	152343780	2,00608	4,07077	2,26491	2,62376	2,40655	2,05684	2,76390	4,34385	3,31246
	152343782	2,01227	3,21085	2,26851	2,62133	2,41105	2,05256	2,76460	3,49839	3,29982

Versión 6 PMCs:

		Task Contender								
		152343765	152343767	152343769	152343771	152343775	152343777	152343778	152343780	152343782
Task Under Analysis	152343765	1,91393	1,89566	1,89385	1,89822	1,89590	1,90345	1,89885	1,95319	1,89713
	152343767	1,96745	5,53442	2,24376	2,57491	2,38453	2,02828	2,72574	4,19904	3,16718
	152343769	1,95586	2,21054	2,20613	2,21537	2,20522	2,00964	2,21296	2,27367	2,20996
	152343771	1,93952	2,30696	2,21362	2,30731	2,30511	1,96782	2,31128	2,35818	2,30687
	152343775	1,94366	2,32205	2,22478	2,33952	2,30819	2,00921	2,33435	2,40365	2,32409
	152343777	1,94183	1,93626	1,93031	1,94185	1,92938	1,93928	1,93871	2,00321	1,93486
	152343778	1,87816	2,49308	2,13884	2,28519	2,25537	1,91497	2,49305	2,54936	2,49400
	152343780	1,95790	3,99325	2,22897	2,52148	2,36533	1,99883	2,69025	4,05856	3,14121
	152343782	1,96525	3,13548	2,23452	2,52107	2,37248	1,99769	2,69321	3,20636	3,13015

Tabla 12 –pWCET con 6 PMCs normalizado

Cuando se comparan ambas matrices, se puede apreciar que la versión de 6 PMCs obtiene un pWCET menor que la versión de 4 PMCs. La mejora que se consiga con esta versión dependerá también de la combinación ejecutada:

TC\TUA		TUA	TUA	TUA	TUA	TUA	TUA	TUA	TUA	TUA
BenchName	BenchId	152343765	152343767	152343769	152343771	152343775	152343777	152343778	152343780	152343782
TC = bitmnp01lite	152343765	240831	208866	41007	140288	101953	135105	166722	1212227	752934
TC = canrdr01lite	152343767	51757	74941	38431	117519	43388	65286	81118	308519	175010
TC = pntrch01lite	152343769	64456	114255	41627	122808	54674	89957	98212	553831	318662
TC = idctrn01lite	152343771	128674	232979	63259	145595	105410	138782	188600	1330651	794236
TC = a2time01lite	152343775	51496	75886	41381	120205	34672	66708	84114	307302	173580
TC = basefp01lite	152343777	62028	106071	42002	123718	52627	64194	96238	449584	277359
TC = ttsprk01lite	152343778	74442	130288	56994	131332	65961	97549	102814	609956	370738
TC = cacheb01lite	152343780	56329	90643	42014	119590	48189	67824	86116	333564	200239
TC = puwmod01lite	152343782	56067	89871	40537	119549	45989	65424	85132	348210	202306

Tabla 13 –Diferencia entre el pWCET calculado entre 4 PMCs y 6 PMCs

En general, se puede apreciar que no resulta ningún valor negativo, lo que nos permite saber que, en cualquiera de los casos, el uso de 6 PMCs en vez de 4, mejora la estimación del pWCET. Sin embargo, debido a la diferencia en ciclos de estas combinaciones, es mejor ver cómo reduce la estimación del PWCET la versión nueva de contadores normalizando la diferencia, permitiendo así tener una vista más justa y personalizada sobre cada benchmark y su pareja TC. Además, se ha reflejado este valor como porcentaje para una visualización más sencilla:

TC\TUA		TUA	TUA	TUA	TUA	TUA	TUA	TUA	TUA	TUA
BenchName	**BenchId**	**152343765**	**152343767**	**152343769**	**152343771**	**152343775**	**152343777**	**152343778**	**152343780**	**152343782**
TC = bitmnp01lite	152343765	4,798%	21,112%	2,086%	2,671%	10,374%	8,123%	6,678%	101,634%	62,916%
TC = canrdr01lite	152343767	1,031%	7,575%	1,955%	2,238%	4,415%	3,925%	3,249%	25,867%	14,624%
TC = pntrch01lite	152343769	1,284%	11,549%	2,118%	2,339%	5,563%	5,408%	3,934%	46,434%	26,628%
TC = idctrn01lite	152343771	2,564%	23,549%	3,218%	2,772%	10,725%	8,344%	7,555%	111,563%	66,368%
TC = a2time01lite	152343775	1,026%	7,670%	2,105%	2,289%	3,528%	4,011%	3,369%	25,765%	14,505%
TC = basefp01lite	152343777	1,236%	10,721%	2,137%	2,356%	5,355%	3,860%	3,855%	37,694%	23,177%
TC = ttsprk01lite	152343778	1,483%	13,169%	2,899%	2,501%	6,711%	5,865%	4,118%	51,139%	30,979%
TC = cacheb01lite	152343780	1,122%	9,162%	2,137%	2,277%	4,903%	4,078%	3,450%	27,966%	16,732%
TC = puwmod01lite	152343782	1,117%	9,084%	2,062%	2,277%	4,679%	3,933%	3,410%	29,194%	16,905%

Tabla 14 –Diferencia entre el pWCET calculado entre 4 PMCs y 6 PMCs normalizado

A partir de estos valores, se puede apreciar cómo puede mejorar desde valores reducidos, como “1,026%”, hasta valores elevados de mejora como puede ser “101,634%”. Este cambio refleja que, dependiendo de la composición de los benchmarks, este porcentaje variará.

Es por esto por lo que, a raíz de estos resultados, se ha considerado prudente tratar los datos para hallar distintos parámetros tales como la media, la mediana o la desviación.

	Como TUA								
	152343765	**152343767**	**152343769**	**152343771**	**152343775**	**152343777**	**152343778**	**152343780**	**152343782**
Máximo	4,798%	23,549%	3,218%	2,772%	10,725%	8,344%	7,555%	111,563%	66,368%
Mínimo	1,026%	7,575%	1,955%	2,238%	3,528%	3,860%	3,249%	25,765%	14,505%
Media	1,740%	12,621%	2,302%	2,413%	6,250%	5,283%	4,402%	50,806%	30,315%
Mediana	1,236%	10,721%	2,118%	2,339%	5,355%	4,078%	3,855%	37,694%	23,177%
Desviación	1,242%	5,820%	0,440%	0,192%	2,588%	1,822%	1,581%	32,985%	20,273%
Coef.Variac.	71,373%	46,116%	19,108%	7,966%	41,404%	34,484%	35,923%	64,922%	66,876%

Tabla 15 –Estadísticas obtenidas de los resultados

Los valores máximo y mínimo nos permiten conocer el rango de mejora que se consigue en las distintas combinaciones para cada TUA, este rango puede ser bastante reducido, como es el caso del “Benchmark 152343771”, el cual obtiene una mejoría entre 2,238 y 2,772, o muy amplios, como es el caso del “Benchmark 152343780”, con valores en el rango 25,765 – 111,563.

Por otro lado, la media da una idea de cuál es la ganacia en cada combinación si consideramos cada una de las combinaciones como un conjunto y queremos saber en total, cuanto ha mejorado, esto ayuda a descartar picos de mejora, ya sean bajos o altos. También se dispone de la mediana, la cual determina el punto medio si se ordenasen las combinaciones, de aquí podemos extrapolar que al menos la mitad de los conjuntos de combinaciones para cada TUA obtiene ganancias inferiores a la media, lo cual indica que existen valores superiores con ganancias significativamente superiores que provocan este aumento en la media.

Como es de esperar, en conjuntos de combinaciones para una misma TUA que cuenten con un rango reducido de mejora, dispondrá de una desviación menor, lo cual indica que, para cada combinación, se obtiene un valor cercano al calculado en la media. De la misma forma, en aquellos conjuntos que presentan un rango de mejoría más amplio, se puede apreciar un mayor valor de desviación estándar. Dado que la mejora para cada combinación de esos casos tiene un valor alejado de la media, se pueden usar para observar el efecto de la implementación de 6 PMCs en vez de 4

PMCs, puesto que cada caso tendrá unos contadores distintos, y que provocará esa diferencia amplia en mejoría.

5.3. Obtención de resultados en “Competition”

Los tiempos en “Competition” obtenidos para las distintas combinaciones son:

			Task Under Analysis								
	BenchId	BenchName	152343765	152343767	152343769	152343771	152343775	152343777	152343778	152343780	152343782
Task Contender	152343765	bitmnp01lite	10036137	1956732	3911613	10458078	1905511	3275886	4549951	2302562	2363871
	152343767	canrdr01lite	9929360	1957420	3912503	10457904	1905204	3275001	4552693	2301285	2366863
	152343769	pntrch01lite	9923779	1957250	3913693	10457124	1905026	3276642	4551336	2299312	2364409
	152343771	idctrn01lite	9935326	1955757	3909987	10461155	1903080	3274586	4551083	2298190	2362680
	152343775	a2time01lite	9945814	1957848	3913087	10457504	1908514	3276393	4549279	2297228	2364717
	152343777	basefp01lite	9938967	1964669	3913600	10457867	1906066	3281010	4552907	2305098	2367037
	152343778	ttsprk01lite	9930426	1959492	3912511	10457607	1905417	3276472	4554729	2305893	2365314
	152343780	cacheb01lite	9975129	1961977	3913797	10466235	1907964	3282715	4555303	2305622	2367491
	152343782	puwmod01lite	9938051	1956626	3911955	10458521	1903384	3276764	4552170	2297908	2364767

Tabla 16 –Tiempos en Competition de las combinaciones

Estos tiempos obtenidos en la ejecución multihilo de cada combinación en Competición se compararán con las estimaciones del pWCET obtenidas en el paso anterior.

5.4. Validación del WCET

La estimación del pWCET ha de ser, lógicamente, mayor que el tiempo obtenido por la ejecución multinúcleo en Competition. Es por ello por lo que, para comparar ambos valores, se ha hecho la resta “pWCET – Competition_time”, con el fin de obtener un valor positivo en el caso de estar correctamente calculado.

Para la versión de 6 PMCs, obtenemos los siguientes valores:

BenchId	BenchName	152343765	152343767	152343769	152343771	152343775	152343777	152343778	152343780	152343782
152343765	bitmnp01lite	-431262	-15810	-70627	-281353	2795	-60205	115102	-13356	-20583
152343767	canrdr01lite	-416175	3502381	428621	1646833	374612	-68541	1639722	2367682	1371770
152343769	pntrch01lite	-419675	256258	418771	1157845	279285	-80041	761193	306839	299947
152343771	idctrn01lite	-409300	584438	440621	1645388	393880	-58863	1124961	649959	643354
152343775	a2time01lite	-431425	394535	417588	1637512	357688	-81322	1052693	468354	464143
152343777	basefp01lite	-386714	36264	33002	-132649	66595	-69552	203559	31969	14936
152343778	ttsprk01lite	-401233	729496	433379	1669762	386474	-65956	1637591	839586	845967
152343780	cacheb01lite	-173245	2180445	551311	1907230	451967	34625	1776894	2439713	1455652
152343782	puwmod01lite	-417526	1167852	428029	1645718	378428	-72613	1642529	1374840	1367510

Tabla 17 – Diferencia entre pWCET (6 PMCs) y tiempo en Competition

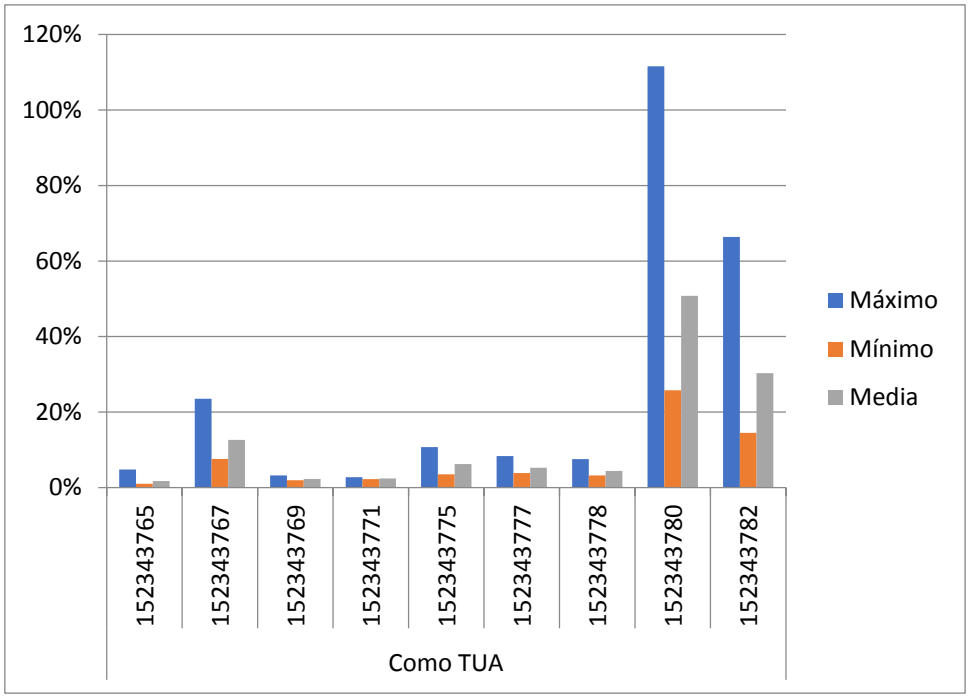
Para la versión de 4 PMCs, obtenemos los siguientes valores:

BenchId	BenchName	152343765	152343767	152343769	152343771	152343775	152343777	152343778	152343780	152343782
152343765	bitmnp01lite	-190431	35947	-6171	-152679	54291	1823	189544	42973	35484
152343767	canrdr01lite	-207309	3577322	542876	1879812	450498	37530	1770010	2458325	1461641
152343769	pntrch01lite	-378668	294689	460398	1221104	320666	-38039	818187	348853	340484
152343771	idctrn01lite	-269012	701957	563429	1790983	514085	64855	1256293	769549	762903
152343775	a2time01lite	-329472	437923	472262	1742922	392360	-28695	1118654	516543	510132
152343777	basefp01lite	-251609	101550	122959	6133	133303	-5358	301108	99793	80360
152343778	ttsprk01lite	-234511	810614	531591	1858362	470588	30282	1740405	925702	931099
152343780	cacheb01lite	1038982	2488964	1105142	3237881	759269	484209	2386850	2773277	1803862
152343782	puwmod01lite	335408	1342862	746691	2439954	552008	204746	2013267	1575079	1569816

Tabla 18 – Diferencia entre pWCET (4 PMCs) y tiempo en Competition

5.5. Representación Gráfica

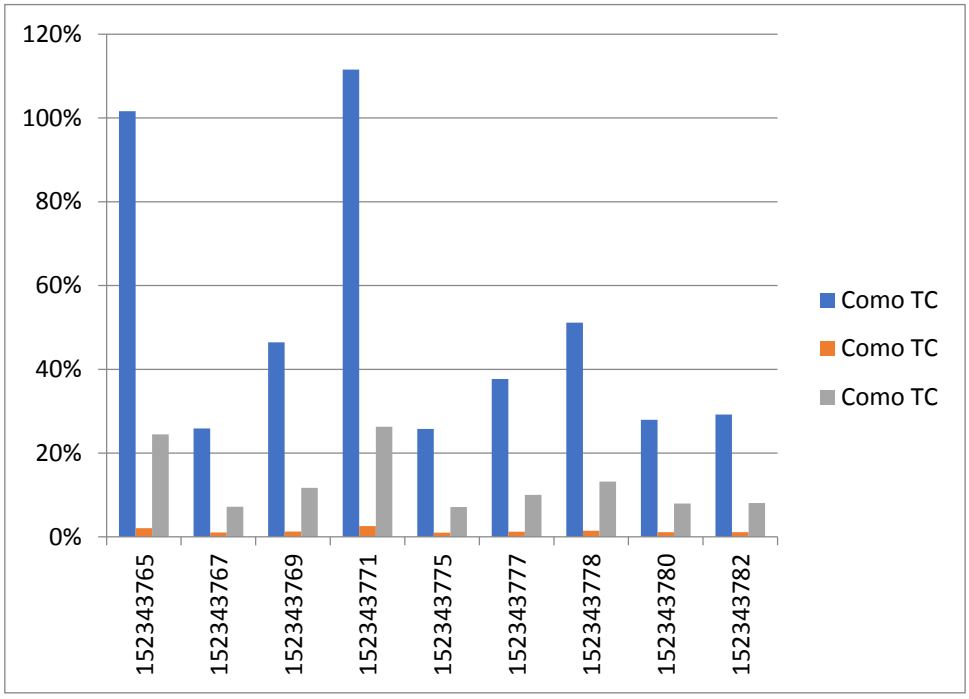
En este apartado se pueden visualizar de manera gráfica los distintos resultados. Aquí se puede visualizar la mejora obtenida al usar 6 PMCs en vez de 4 PMCs para cada Benchmark como TUA:



Gráfica 1 – Mejora en pWCET para cada benchmark como TUA

Gracias a este gráfico se puede comprobar cómo existen picos de mucha ganancia, como son los máximos de los benchmarks “152343780” y “152343782”, pero que, aun así, obtienen una ganancia media bastante alta. En contraposición tenemos benchmarks como “152343765” que obtienen una ganancia muy reducida, esta contrariedad denota que, aun siendo la versión 1 (6 PMCs) mejor que la versión 0 (4 PMCs) en cualquier caso, habrá combinaciones de benchmarks que se vean más favorecidas que otras debida a los contadores obtenidos en SIM1.

Tras analizar el comportamiento como TUA de cada benchmark, se mostrará el gráfico viendo los benchmarks usados como TC:

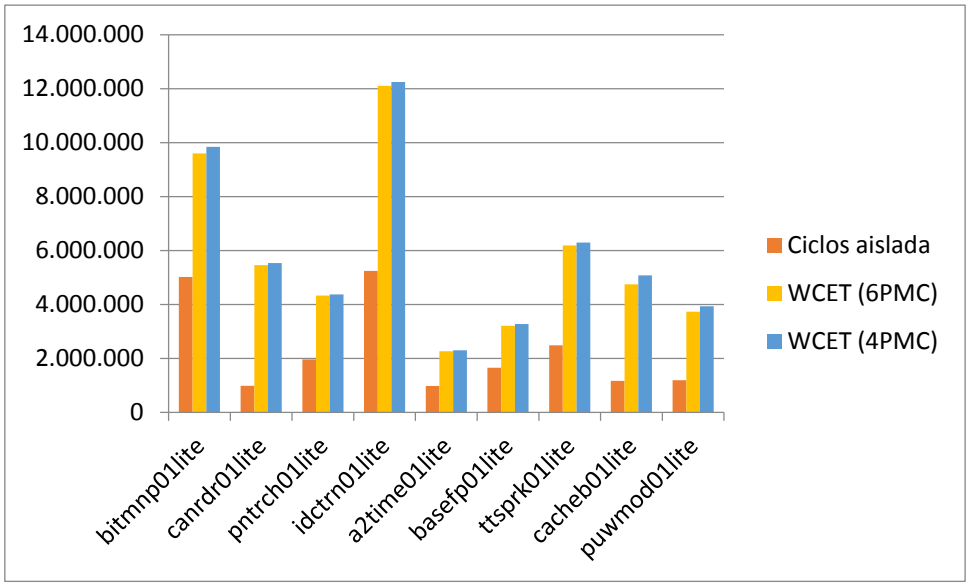


Gráfica 2 – Mejora en pWCET para cada benchmark como TC

Como se ha visto anteriormente, hay TUA con una ganancia muy reducida, y esto se ve reflejado en los valores mínimos como TC obtenidos, puesto que todos ellos producen ganancias bajas con ciertas TUA. Sin embargo, se puede observar que los mayores picos de ganancia se producen con los benchmarks “152343765” y “152343771” como TC.

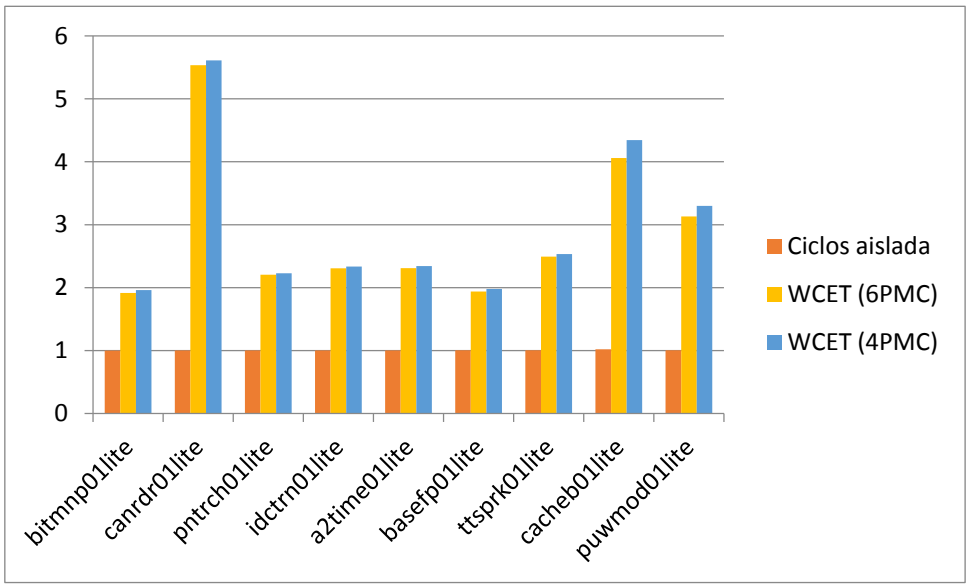
Dadas estas dos gráficas, se puede concluir que las combinaciones, del tipo TUA-TC, “152343780-152343765”, “152343780-152343771”, “152343782-152343765” y “152343782-152343765” son las que mayor ganancia consiguen.

Es interesante ver el efecto de un benchmark como TC sobre sí mismo como TUA, es por ello por lo que en la siguiente gráfica se resumen dichas combinaciones:



Gráfica 3 – Comparativa Tiempo_Isolation/Tiempo_Competition/pWCET(6 PMCs)/pWCET(4 PMCs) donde el benchmark de la TUA y la TC sea el mismo.

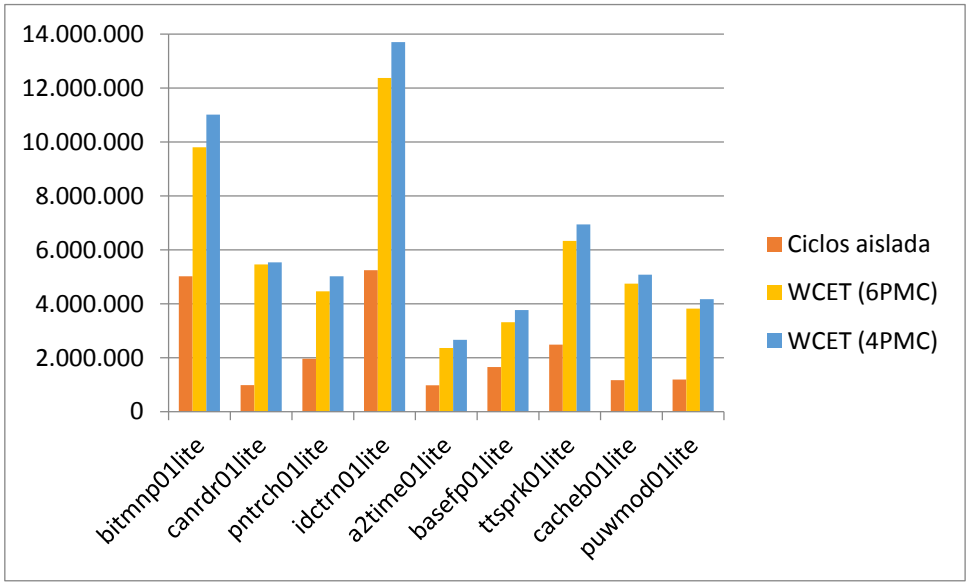
En esta gráfica se puede apreciar la diferencia existente entre el pWCET estimado y los tiempos obtenidos en las combinaciones en las que ocurre que tanto la TUA como la TC ejecutan el mismo Benchmark. De forma generalizada se puede considerar que la estimación del pWCET es bastante ajustada, además de que es observable el hecho de que la versión 1 de SIM2, obtiene una estimación más reducida que la versión 0.



Gráfica 4 – Comparativa Tiempo_Isolation/Tiempo_Competition/pWCET(6 PMCs)/pWCET(4 PMCs) normalizado donde el benchmark de la TUA y la TC sea el mismo.

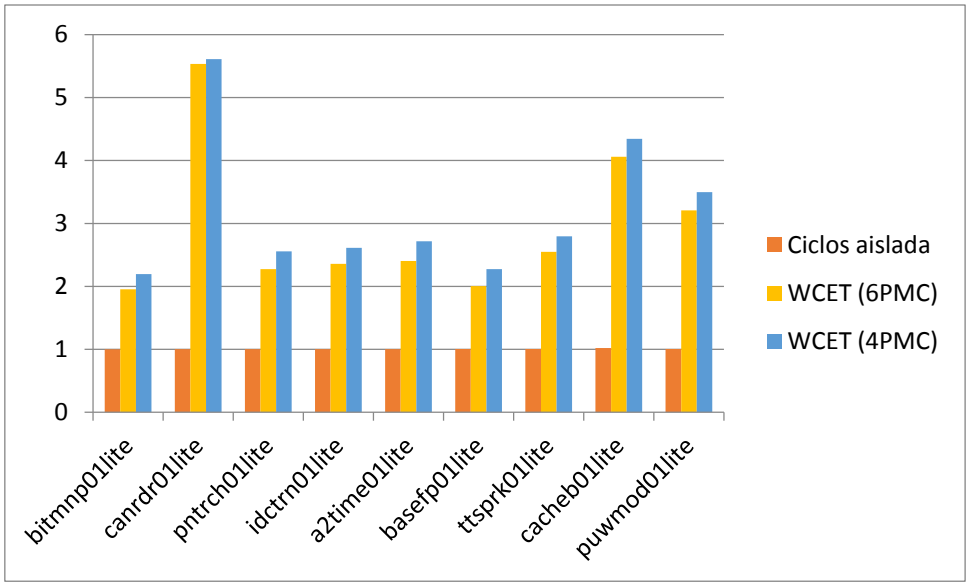
Al normalizar los resultados, se puede comprobar de manera justa qué benchmarks como TC se afectan más a sí mismo como TUA. Al hacer la normalización, destacan los calores obtenidos por las estimaciones calculadas por los benchmarks 152343767 – "canrdr01lite", 152343780 – "cacheb01lite" y 152343782 – "puwmod01lite". Para el resto de combinaciones se puede comprobar que el valor normalizado es cercano a 2.

Otra opción de combinación interesante a tener en cuenta es el del emparejamiento realizado con peor resultado. La siguiente gráfica representa dichas combinaciones:



Gráfica 5 – Comparativa Tiempo_Isolation/Tiempo_Competition/pWCET(6 PMCs)/pWCET(4 PMCs) donde el benchmark de la TUA y la TC sea la combinación con peores resultados.

Se puede apreciar el aumento general en los valores estimados dado que estamos usando la peor combinación TUA-TC posible para cada TUA. Al igual que en la Gráfica 3, se puede observar que la versión 1 de SIM2, consigue una estimación menor que la conseguida por la versión 0 de SIM2. Al normalizar estos valores, obtenemos la siguiente matriz:



Gráfica 6 – Comparativa Tiempo_Isolation/Tiempo_Competition/pWCET(6 PMCs)/pWCET(4 PMCs) normalizado, donde el benchmark de la TUA y la TC sea la combinación con peores resultados.

Se puede ver que es muy similar a la Gráfica 4, en la que los benchmarks denominados “canrdr01lite”, “cacheb01lite” y “puwmod01lite” obtienen un valor normalizado destacable, mientras que el resto de combinaciones obtiene valores entre 2 y 3.

Capítulo 6

Conclusiones y trabajos futuros

Se han implementado un amplio conjunto de mejoras en el programa SIM1, destinadas, por un lado, a proporcionar una mayor manejabilidad de uso y capacidad de configuración, y por otro a aumentar la funcionalidad y los formatos de salida, tanto para alimentar el SIM2, como en formato de tablas "csv" para visualizar los resultados de ejecución directamente en una hoja de cálculo, y facilitar así un rápido y versátil análisis. Además, se ha trabajado el simulador SIM2 para que proporcione el cálculo de WCET con los 4 contadores existentes en el LEON3 así como con los seis contadores ideales, a fin de poder realizar un análisis comparativo.

Durante el desarrollo se ha utilizado un amplio conjunto de benchmarcs preexistentes (53 entre sintéticos y benchmars industriales) y otro bloque generado exprofeso para validar el funcionamiento. Como ejemplo y demostración de funcionamiento, en la fase esperimental se ha utilizado una carga de trabajo compuesta por un subconjunto de 9 automotive EEMBC benchs, principalmente por razones de la limitación de espacio del presente documento. Los resultados proporcionados por SIM1 y SIM2 se han mostrado en distintas tablas y gráficos. Se han mostrado el valor de WCET que se deriva de los 4 PMCs presentes en el LEON3. Adicionalmente se han calculado el WCET que se podría obtener si se dispusiesen de los 6 PMCs descritos en la memoria. Aunque la mejora es variable dependiendo de la combinación ejecutada, el uso de 6 PMCs en vez de 4 PMCs, mejora la estimación del pWCET, con una mejora que llega a ser superior al 15% y en media del orden del 4%. Este valor de WCET menos pesimista aboga por introducir en los futuros procesadores PMCs adicionales.

Como trabajo futuro se considera una experimentación ampliada, haciendo uso de las mejoras introducidas, tales como la comparación entre el uso de cache fijada por vías o completamente asociativa o el uso de un desfase variable en la composición en la ejecución de la TUA con un Task Contender, para poder valorar la variabilidad que se produce por estos factores.

Se ha dotado a SIM1 con la funcionalidad de proporcionar una traza temporal de eventos en el bus, para poder llevar a cabo trabajos futuros, que aún están en fase planificación.

Referencias

1.- Enrique Díaz, Mikel Fernández, Leonidas Kosmidis, Enrico Mezzetti, Carles Hernandez, Jaume Abella and Francisco J. Cazorla: “MC2: Multicore and Cache Analysis via Deterministic and Probabilistic Jitter Bounding.

Anexos

Fichero de configuración ejemplo: "*configuration_example.cfg*"

```
experiments =
{
        n_test_case = 81;
        test_case = (
                                // tanda 1
                                { bench0 = "bitmnp01lite.log"; //1
                                        bench1 = "bitmnp01lite.log";},          //142533465
                                { bench0 = "bitmnp01lite.log"; //1
                                        bench1 = "canrdr01lite.log";},          //142533467
                                { bench0 = "bitmnp01lite.log"; //1
                                        bench1 = "pntrch01lite.log";},          //142533469
                                { bench0 = "bitmnp01lite.log"; //1
                                        bench1 = "idctrn01lite.log";},          //142533471
                                { bench0 = "bitmnp01lite.log"; //1
                                        bench1 = "a2time01lite.log";},          //142533475
                                { bench0 = "bitmnp01lite.log"; //1
                                        bench1 = "basefp01lite.log";},          //142533477
                                { bench0 = "bitmnp01lite.log"; //1
                                        bench1 = "ttsprk01lite.log";},          //142533478
                                { bench0 = "bitmnp01lite.log"; //1
                                        bench1 = "cacheb01lite.log";},          //142533480
                                { bench0 = "bitmnp01lite.log"; //1
                                        bench1 = "puwmod01lite.log";},      //142533482
```

```
// tanda 2
{ bench0 = "canrdr01lite.log"; //1
        bench1 = "bitmnp01lite.log";},          //142533465
{ bench0 = "canrdr01lite.log"; //1
        bench1 = "canrdr01lite.log";},          //142533467
{ bench0 = "canrdr01lite.log"; //1
        bench1 = "pntrch01lite.log";},          //142533469
{ bench0 = "canrdr01lite.log"; //1
        bench1 = "idctrn01lite.log";},          //142533471
{ bench0 = "canrdr01lite.log"; //1
        bench1 = "a2time01lite.log";},          //142533475
{ bench0 = "canrdr01lite.log"; //1
        bench1 = "basefp01lite.log";},          //142533477
{ bench0 = "canrdr01lite.log"; //1
        bench1 = "ttsprk01lite.log";},          //142533478
{ bench0 = "canrdr01lite.log"; //1
        bench1 = "cacheb01lite.log";},          //142533480
{ bench0 = "canrdr01lite.log"; //1
        bench1 = "puwmod01lite.log";},      //142533482

// tanda 3
{ bench0 = "pntrch01lite.log"; //1
        bench1 = "bitmnp01lite.log";},          //142533465
{ bench0 = "pntrch01lite.log"; //1
        bench1 = "canrdr01lite.log";},          //142533467
{ bench0 = "pntrch01lite.log"; //1
        bench1 = "pntrch01lite.log";},          //142533469
{ bench0 = "pntrch01lite.log"; //1
        bench1 = "idctrn01lite.log";},          //142533471
```

```
{ bench0 = "pntrch01lite.log"; //1
        bench1 = "a2time01lite.log";},          //142533475
{ bench0 = "pntrch01lite.log"; //1
        bench1 = "basefp01lite.log";},          //142533477
{ bench0 = "pntrch01lite.log"; //1
        bench1 = "ttsprk01lite.log";},          //142533478
{ bench0 = "pntrch01lite.log"; //1
        bench1 = "cacheb01lite.log";},          //142533480
{ bench0 = "pntrch01lite.log"; //1
        bench1 = "puwmod01lite.log";},      //142533482

// tanda 4
{ bench0 = "idctrn01lite.log"; //1
        bench1 = "bitmnp01lite.log";},          //142533465
{ bench0 = "idctrn01lite.log"; //1
        bench1 = "canrdr01lite.log";},          //142533467
{ bench0 = "idctrn01lite.log"; //1
        bench1 = "pntrch01lite.log";},          //142533469
{ bench0 = "idctrn01lite.log"; //1
        bench1 = "idctrn01lite.log";},          //142533471
{ bench0 = "idctrn01lite.log"; //1
        bench1 = "a2time01lite.log";},          //142533475
{ bench0 = "idctrn01lite.log"; //1
        bench1 = "basefp01lite.log";},          //142533477
{ bench0 = "idctrn01lite.log"; //1
        bench1 = "ttsprk01lite.log";},          //142533478
{ bench0 = "idctrn01lite.log"; //1
        bench1 = "cacheb01lite.log";},          //142533480
{ bench0 = "idctrn01lite.log"; //1
        bench1 = "puwmod01lite.log";},      //142533482
```

```
// tanda 5
{ bench0 = "a2time01lite.log"; //1
        bench1 = "bitmnp01lite.log";},          //142533465
{ bench0 = "a2time01lite.log"; //1
        bench1 = "canrdr01lite.log";},          //142533467
{ bench0 = "a2time01lite.log"; //1
        bench1 = "pntrch01lite.log";},          //142533469
{ bench0 = "a2time01lite.log"; //1
        bench1 = "idctrn01lite.log";},          //142533471
{ bench0 = "a2time01lite.log"; //1
        bench1 = "a2time01lite.log";},          //142533475
{ bench0 = "a2time01lite.log"; //1
        bench1 = "basefp01lite.log";},          //142533477
{ bench0 = "a2time01lite.log"; //1
        bench1 = "ttsprk01lite.log";},          //142533478
{ bench0 = "a2time01lite.log"; //1
        bench1 = "cacheb01lite.log";},          //142533480
{ bench0 = "a2time01lite.log"; //1
        bench1 = "puwmod01lite.log";},      //142533482

// tanda 6
{ bench0 = "basefp01lite.log"; //1
        bench1 = "bitmnp01lite.log";},          //142533465
{ bench0 = "basefp01lite.log"; //1
        bench1 = "canrdr01lite.log";},          //142533467
{ bench0 = "basefp01lite.log"; //1
        bench1 = "pntrch01lite.log";},          //142533469
{ bench0 = "basefp01lite.log"; //1
        bench1 = "idctrn01lite.log";},          //142533471
```

```
{ bench0 = "basefp01lite.log"; //1
        bench1 = "a2time01lite.log";},          //142533475
{ bench0 = "basefp01lite.log"; //1
        bench1 = "basefp01lite.log";},          //142533477
{ bench0 = "basefp01lite.log"; //1
        bench1 = "ttsprk01lite.log";},          //142533478
{ bench0 = "basefp01lite.log"; //1
        bench1 = "cacheb01lite.log";},          //142533480
{ bench0 = "basefp01lite.log"; //1
        bench1 = "puwmod01lite.log";},      //142533482

// tanda 7
{ bench0 = "ttsprk01lite.log"; //1
        bench1 = "bitmnp01lite.log";},          //142533465
{ bench0 = "ttsprk01lite.log"; //1
        bench1 = "canrdr01lite.log";},          //142533467
{ bench0 = "ttsprk01lite.log"; //1
        bench1 = "pntrch01lite.log";},          //142533469
{ bench0 = "ttsprk01lite.log"; //1
        bench1 = "idctrn01lite.log";},          //142533471
{ bench0 = "ttsprk01lite.log"; //1
        bench1 = "a2time01lite.log";},          //142533475
{ bench0 = "ttsprk01lite.log"; //1
        bench1 = "basefp01lite.log";},          //142533477
{ bench0 = "ttsprk01lite.log"; //1
        bench1 = "ttsprk01lite.log";},          //142533478
{ bench0 = "ttsprk01lite.log"; //1
        bench1 = "cacheb01lite.log";},          //142533480
{ bench0 = "ttsprk01lite.log"; //1
        bench1 = "puwmod01lite.log";},      //142533482
```

```
// tanda 8
{ bench0 = "cacheb01lite.log"; //1
        bench1 = "bitmnp01lite.log";},          //142533465
{ bench0 = "cacheb01lite.log"; //1
        bench1 = "canrdr01lite.log";},          //142533467
{ bench0 = "cacheb01lite.log"; //1
        bench1 = "pntrch01lite.log";},          //142533469
{ bench0 = "cacheb01lite.log"; //1
        bench1 = "idctrn01lite.log";},          //142533471
{ bench0 = "cacheb01lite.log"; //1
        bench1 = "a2time01lite.log";},          //142533475
{ bench0 = "cacheb01lite.log"; //1
        bench1 = "basefp01lite.log";},          //142533477
{ bench0 = "cacheb01lite.log"; //1
        bench1 = "ttsprk01lite.log";},          //142533478
{ bench0 = "cacheb01lite.log"; //1
        bench1 = "cacheb01lite.log";},          //142533480
{ bench0 = "cacheb01lite.log"; //1
        bench1 = "puwmod01lite.log";},      //142533482

// tanda 9
{ bench0 = "puwmod01lite.log"; //1
        bench1 = "bitmnp01lite.log";},          //142533465
{ bench0 = "puwmod01lite.log"; //1
        bench1 = "canrdr01lite.log";},          //142533467
{ bench0 = "puwmod01lite.log"; //1
        bench1 = "pntrch01lite.log";},          //142533469
{ bench0 = "puwmod01lite.log"; //1
        bench1 = "idctrn01lite.log";},          //142533471
```

```
                                { bench0 = "puwmod01lite.log"; //1
                                        bench1 = "a2time01lite.log";},          //142533475
                                { bench0 = "puwmod01lite.log"; //1
                                        bench1 = "basefp01lite.log";},          //142533477
                                { bench0 = "puwmod01lite.log"; //1
                                        bench1 = "ttsprk01lite.log";},          //142533478
                                { bench0 = "puwmod01lite.log"; //1
                                        bench1 = "cacheb01lite.log";},          //142533480
                                { bench0 = "puwmod01lite.log"; //1
                                        bench1 = "puwmod01lite.log";}       //142533482
                                );
};


latencies =
{
        MEM_READ_LAT = "13";
        MEM_WRITE_LAT = "12";
        MEM_CLOSE_LAT = "10";
        MEM_REQ_LAT = "23";
        D_LD_ML1 = "8";
        D_LD_ML2 = "9";
        I_LD_ML1 = "8";
        I_LD_ML2 = "9";
        INSTRUCTION_TYPE_LOAD_LAT = "1";
        INSTRUCTION_TYPE_STORE_LAT = "1";
        INSTRUCTION_TYPE_INT_ARITHMETIC_LAT = "1";
        INSTRUCTION_TYPE_CONTROL_TRANSFER_LAT = "1";
        INSTRUCTION_TYPE_FLP_ARITHMETIC_LAT = "4";
        INSTRUCTION_TYPE_DIV_LAT = "35";
```

```
        INSTRUCTION_TYPE_LONG_FLP_LAT = "25";
        INSTRUCTION_TYPE_LONG_DELAY_TUA = "1000";
        NGMP_CACHE_WAYS = "4";
};
fix_l2_way = 1;
tua_delay = 0;
storageConfig =
{
        storeOldCounters = 0;
        storeNewCounters = 0;
        storeDataFlag = 1;
        traceEventFlag = 0;
        oldCountersDirectory = "output/isolation/";
        newCountersDirectory = "output/extended/pair/";
        tableDataFilename = "output/table_data_pairs_f.csv";
        traceEventFilename = "output/trace_event/prueba.txt";
};
```