eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Grado en Ingeniería Informática

Ingeniería de Computadores

Trabajo de Fin de Grado

Análisis de los efectos de la consolidación en un entorno HPC

Autor

Borja Moralejo Tobajas

2023

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Grado en Ingeniería Informática

Ingeniería de Computadores

Trabajo de Fin de Grado

Análisis de los efectos de la consolidación en un entorno HPC

Autor

Borja Moralejo Tobajas

Director

Jose A. Pascual Saíz

Codirector

Javier Navaridas Palma

Resumen

En este proyecto se han analizado los efectos que tienen algunas de las técnicas empleadas en la planificación de los supercomputadores. El objetivo principal de estas técnicas es el aumento del rendimiento del sistema en general, afectando de forma indirecta a la reducción del consumo eléctrico de los centros HPC (High performance computing o computación de alto rendimiento)(Hankendi and Coskun, 2012).

Las técnicas que se van a analizar son la consolidación y la variabilidad. Para llegar a medir los efectos de estas mismas, se implementará un sencillo perfilador que obtenga los tiempos y los datos de los contadores hardware de las CPUs. Estos contadores no solo se utilizarán para medir el rendimiento, si no que también se emplearán en el descubrimiento de la naturaleza de la aplicación. Se van a realizar los análisis sobre los paquetes de prueba o bechmark de PARSEC. Con el fin de aprovechar los beneficios de la consolidación, es imprescindible tener un conocimiento de la naturaleza de la aplicación que se ejecutará.

Dado que este proyecto tiene un enfoque de investigación eso implica describir de manera detallada el proceso utilizado para recolectar los datos, así como las técnicas y herramientas empleadas en su análisis y procesamiento. Esta metodología se ha seguido de varios artículos (Dwyer et al., 2012) y (Kundan et al., 2022). Esta metodología proporcionará una base sólida para comprender cómo se obtuvieron los resultados y asegurará la validez y fiabilidad de los hallazgos obtenidos en el proyecto.

Los resultados indican una posible diferenciación de las naturalezas de las aplicaciones mediante las métricas de perfilado. La consolidación, a coste de ejecuciones más lentas (75 % de media), muestra un 9 % de incremento del uso del sistema en el mejor caso, y un 0.43 % en el peor de ellos. En cuanto a la variabilidad, se observa un potencial sencillo de aprovechar y con un incremento sin sobrecoste de un 0.1 %.

Índice general

Índice de figuras

Índice de tablas

1. CAPÍTULO

Introducción

A continuación, se describe la motivación del proyecto y su alcance. También se incluye un glosario de términos y la organización del resto de la presente documentación.

1.1. Motivación

El consumo de energía de los supercomputadores es una gran parte de los gastos económicos de los centros de cómputo y por eso mismo es importante que las ejecuciones de los programas que utilicen eficientemente los recursos a su disposición. Los tiempos de ejecución en este ámbito es dinero y mientras más rápido puedas ejecutar la aplicación, más gastos te ahorras. Este consumo energético se puede reducir de varias maneras que se van a explicar a lo largo del documento. Los otros costes son refrigeración y distribución eléctrica, entre otros, pero esos temas no se investigarán en este documento.

Una de las formas para reducir el consumo se logra cuando no hay tareas y el procesador se pone en power-saving, pero eso en un supercomputador no importa porque siempre hay demanda de su potencia de cálculo por parte de programas de investigación. Además, en los centros HCP se busca activamente no tener los sistemas inactivos, ese tiempo de actividad no solo cuesta dinero, cada supercomputador tiene un periodo de vida útil, algo definido por la obsolescencia. Los nuevos componentes son más eficientes en comparación con los antiguos y llega un momento que no merece la pena mantener el supercomputador con el equipo viejo y sale más rentable comprar un equipo nuevo.

La segunda forma es aumentar la eficacia de la máquina. Ya sea aumentando la eficiencia de los programas paralelizados, que son manejados por personal especializado, o analizando y tratando las interacciones que se dan entre los componentes de los nodos. En este proyecto se centrará en ese último aspecto. Con un estudio del estado del arte se puede llegar a determinar qué técnicas funcionan bien y no tan bien a la hora de aumentar el rendimiento y uso de los componentes en un sistema.

Para llegar a analizar los efectos que llegan a tener estas técnicas en los componentes, es necesario una herramienta que llegue a contabilizar los eventos que ocurren en un equipo y esa herramienta es un perfilador. No sirve cualquier perfilador para estudiar los efectos de las interacciones, los perfiladores generalmente detallan todo el proceso que ha tenido cierto intervalo de ejecución, desde las llamadas a sistemas y cuánto tiempo se han retenido en cierta función. Estos útiles son conocidos por agregar un pequeño sobrecoste a las ejecuciones y se necesita uno que minimice ese extra y que solo obtenga los contadores.

De forma indirecta se necesita estudiar la naturaleza de las aplicaciones para saber qué recursos utiliza y cuál de ellos es el más importante para su rendimiento. Si se llegase a detectar un patrón que dictamina la naturaleza de la aplicación, una buena implementación para detectar las naturalezas podría ser el aprendizaje automático, dado que pueden llegar a aproximar una función implícita y agruparlas por clases. Este sería un trabajo futuro que se podría investigar sobre ello.

Hay varias políticas de planificación de tareas que reducen el consumo de energía y en este proyecto se van a analizar los efectos que generan esas técnicas. Una de ellas es la consolidación (Zacarias et al. (2021),Bhadauria and McKee (2010)), que consiste en incrementar el uso del sistema, sin llegar a impactar demasiado el rendimiento del sistema, a base de lanzar programas en parejas o grupos que lleguen a complementar sus faltas de uso de recursos.

Otra técnica o efecto que puede aumentar el rendimiento de una aplicación es la variabilidad (Teodorescu and Torrellas, 2008), que consiste en aprovechar las pequeñas diferencias que tienen las CPU al ser fabricadas. Esta técnica es más sustancial en el ámbito de la supercomputación porque tienen grandes periodos de tiempo con ejecuciones que pueden ser periódicas y de larga duración. En esos casos, un 0.1 % de diferencia es indiferente a escala de segundos, con tiempos de ejecución de días e incluso semanas, serán pocos los segundos o minutos que se lleguen a ahorrar, pero se obtienen solo por tener en cuenta ese efecto y ejecutar una aplicación en una CPU u otra, sin sobrecoste alguno.

Este proyecto permitirá, por un lado, descubrir el grado de utilidad de las técnicas utilizadas y, por otro lado, preparar una base revisada para que en un trabajo futuro se desarrolle un scheduler que podría implementarse como un plugin de SLURM. Con los datos que se obtengan tras analizar los efectos de la consolidación y variabilidad, se podría generar un scheduler que tuviera en cuenta los tipos de aplicaciones para aumentar la eficiencia (Blagodurov et al., 2010). Realizar un scheduler es principalmente desarrollo, pero sin tener los efectos y análisis que mejoran el rendimiento, ese desarrollo puede resultar ser imposible o fútil, de ahí la importancia de la investigación.

1.2. Alcance y Objetivos

El objetivo es analizar los efectos de la consolidación en un entorno HPC, este deriva del objetivo de la reducción consumo eléctrico de un supercomputador. Lograr reducir el gasto energético es una tarea demasiado amplia para este proyecto y por esa misma razón se ha escogido esta tarea que consiste en estudiar los efectos de una técnica que podría reducirlo. Para ello se trabaja en los siguientes aspectos:

- Conocer el estado del arte en el campo de la supercomputación y la eficiencia computacional.
- Desarrollar un perfilador de aplicaciones.
- Encontrar y analizar rasgos en datos de perfilado de aplicaciones que permitan caracterizarlos por su naturaleza.
- Analizar y cuantificar el efecto de las diferentes técnicas de incremento de eficiencia, principalmente la consolidación y la variabilidad.

Resumiendo, se van a analizar y explorar las técnicas que podrían incrementar la eficiencia de un supercomputador. El resultado de este análisis se podría continuar e implementar en un scheduler en caso de ser positivo.

1.3. Glosario de términos

- **Scheduler** : El scheduler es un sistema que se encarga de asignar y administrar eficientemente los recursos computacionales, como procesadores, memoria y alma-

cenamiento, entre las múltiples tareas o procesos que se ejecutan en un computador. Hay dos niveles de scheduling, alto nivel que se encarga de repartir tareas entre los nodos de los supercomputadores, y a nivel de sistema operativo. Se entra en detalle en la Sección 2.2.3.

- **Consolidación**: Consolidar es una técnica de planificación que consiste en maximizar el uso de los recursos lanzando en el mismo nodo programas que se complementen.
- **Variabilidad**: La variabilidad que se habla en este proyecto no es la estadística, se refiere a las pequeñas diferencias que se crean a la hora de fabricar un procesador.
- **Perfilador**: Es una herramienta que se utiliza para medir y analizar las ejecuciones de programas recopilando datos durante la ejecución de este.
- **Benchmark**: Es una prueba estandarizada y repetible diseñada para evaluar el rendimiento y la capacidad de un sistema, componente de hardware o software.
- **HPC**: High-performance computing o computación de alto rendimiento.
- **LLC**: Caché de último nivel, Last Level Cache por sus siglas en inglés.
- **L1**: Caché de primer nivel.
- **IPC**: Instrucciones por ciclo, Instructions Per Cycle por sus siglas en inglés. Es una métrica común del rendimiento de los sistemas informáticos.
- **Stalled-cycles**: Son ciclos de reloj en un procesador durante los cuales no se realiza ninguna instrucción útil debido a diversas razones, como esperar a que se completen operaciones de memoria, esperar datos de entrada o condiciones de ejecución, o esperar por otros recursos del sistema.
- **Workload**: Es la carga de trabajo o conjunto de tareas que deben realizarse en un servidor, centro HPC o centro de datos.

1.4. Organización del documento

En el primer capítulo se encuentra la introducción, repasando los objetivos y alcances del proyecto. La segunda sección es el contexto y el estado del arte para dar unas bases

sobre las que apoyarse durante la lectura. El tercer capítulo es todo lo relacionado con la planificación del proyecto y el cuarto detalla la metodología que se va a emplear en el análisis. El quinto capítulo está relacionado con el desarrollo del perfilador y los scripts. En el sexto capítulo se analizarán los datos obtenidos y en el séptimo se acabará con las conclusiones y trabajos futuros.

2. CAPÍTULO

Contexto

En este capítulo se desarrollan los conceptos necesarios para que el lector tenga una comprensión de la memoria más clara y amena. Aquí se comentarán las herramientas y sistemas que se han utilizado, además de un marco teórico explicando brevemente los fundamentos de la paralelización y las técnicas que se van a estudiar.

2.1. Equipo y Arquitectura de los computadores

En la investigación es imprescindible conocer el estado previo o estado inicial sobre el que se trabaja. En este caso se ha trabajando sobre un nodo con dos procesadores Intel Xeon X5660 de 1.6-2.8 GHz. Cada procesador tiene 6 núcleos con caches L1 de datos e instrucciones de 384 KiB, caches L2 de 3 MiB y cada CPU su propia caché L3 de 24 MiB. El sistema tiene una memoria RAM total de 98434692 kB, alrededor de 99 GB.

Hoy en día la mayoría de las CPUs tienen contadores hardware, por lo que no había necesidad de ser de un fabricante en específico. Además, al ser Intel el fabricante de la CPU se podría haber utilizado Running Average Power Limit (RAPL) [1] de Intel para medir el consumo energético, pero esta arquitectura de CPU no es compatible.

[1] RAPL Web (último acceso 23 de junio de 2023)

2.1.1. Perfilador

El perfilador en términos generales, es una herramienta que permite analizar el rendimiento de una aplicación en tiempo de ejecución, lo que proporciona valiosa información para identificar cuellos de botella, ineficiencias o áreas de mejora. Al recopilar datos sobre el uso de recursos y el flujo de ejecución, el perfilador puede ayudar a los desarrolladores a comprender mejor el comportamiento de su código y tomar decisiones informadas para optimizarlo.

Existen diferentes tipos de perfiladores, sin embargo, el tipo de perfilador con el que se va a trabajar es a nivel de kernel y hardware, capturando eventos de bajo nivel como interrupciones de hardware o métricas específicas, ciclos e instrucciones de la CPU, o el número de accesos a la caché y su tasa de aciertos.

Los datos recopilados por un perfilador suelen presentarse en forma de informes o visualizaciones gráficas, que permiten a los desarrolladores analizar y comprender los patrones de rendimiento de una aplicación. Esta información es fundamental para optimizar el código, realizar ajustes en la configuración del sistema o identificar posibles problemas de rendimiento. En este contexto esas visualizaciones y métricas se van a utilizar para determinar la naturaleza de las aplicaciones y examinar el nivel de consolidación que obtienen.

2.1.2. Computación de alto rendimiento

La computación de alto rendimiento o HPC, son sistemas informáticos con grandes capacidades computacionales, siendo eficientes para realizar cálculos y procesamientos intensivos a gran escala. Estos sistemas están diseñados para manejar grandes volúmenes de datos y ejecutar programas a una velocidad y escala mucho mayores que los computadores convencionales.

El HPC se utiliza en varios campos, incluyendo la investigación científica, la simulación y modelado de fenómenos complejos, el análisis de grandes conjuntos de datos, el procesamiento de imágenes y el diseño de productos, entre otros. Estos sistemas están compuestos por múltiples nodos interconectados, que trabajan en paralelo para acelerar los cálculos y obtener resultados en tiempos reducidos.

A cierta agrupación de nodos se le suele llamar clúster y un componente de estos sistemas, un nodo, será estudiado para comprobar los efectos de las diversas técnicas que pueden llegar a incrementar el rendimiento general de los supercomputadores.

2.1.3. Paralelización

La paralelización se utiliza para mejorar la eficiencia y el rendimiento de los sistemas informáticos al permitir que múltiples procesos se ejecuten al mismo tiempo. En lugar de esperar a que un proceso se complete antes de comenzar otro, la paralelización permite que varios procesos se ejecuten en más de un núcleo a la vez, lo que puede reducir significativamente el tiempo necesario para completar una o varias tareas.

Existen diferentes técnicas y estrategias de paralelización que se pueden utilizar en función de las características específicas de la tarea o el proceso que se desea paralelizar. Algunas técnicas incluyen el uso de múltiples núcleos de procesamiento en una CPU, el uso de múltiples procesadores en un computador, el uso de clústeres de computadores y el uso de herramientas y lenguajes de programación específicos para la paralelización. En los supercomputadores se mezclan esas técnicas para obtener el mayor factor de aceleración posible con cierta eficiencia. Además de incrementar el número de procesadores, buscan la forma de maximizar la utilización de recursos de forma que no hayan interferencias.

Hay factores que son intrínsecos de las aplicaciones y es posible que estén limitados y no se logre una buena eficiencia. Esos factores son los algoritmos de la aplicación, las implementaciones de la paralelización, la granularidad de la paralelización y la dependencia de datos.

Los algoritmos de la aplicación trata de la estructura de la aplicación, que puede hacer más fácil o difícil separar las tareas que la constituyen y por ello dificultar la paralelización.

La granularidad se refiere al tamaño de las tareas que se dividen en paralelo. Si las tareas son muy pequeñas, el tiempo dedicado a la comunicación y coordinación entre ellas puede ser mayor que el tiempo requerido para completar cada tarea de manera individual. Por otro lado, si las tareas son demasiado grandes, es posible que no se aprovechen por completo los beneficios de la paralelización, ya que una tarea grande puede bloquear o retrasar otras tareas en ejecución. Por lo tanto, es importante encontrar un equilibrio adecuado en la granularidad de las tareas para garantizar una eficiencia óptima en la ejecución paralela.

La dependencia de datos es un problema de coordinación y requiere de cuidado especial. Si una tarea depende de los resultados de otra tarea, no se puede paralelizar y es necesario esperar a que la tarea anterior se complete antes de continuar con la siguiente. Estas dependencias pueden dificultar la paralelización y reducir la eficiencia.

Todo está relacionado, los algoritmos de la aplicación tienen ciertas dependencias de da-

tos que dictaminan la granularidad en la que se puede trabajar y en consecuencia en la implementación de la paralelización.

2.1.4. Factores limitantes dentro del computador

Conocer los factores limitantes de cada aplicación es complicado, pero en general estos son los factores generales.

Memory intensive: Una de las limitaciones más comunes en las aplicaciones es el uso excesivo de memoria. Si una aplicación requiere más memoria de la que está disponible, puede provocar una disminución en el rendimiento al tener que esperar por los datos con mayor frecuencia o incluso provocar errores en el sistema.

Ancho de banda de los buses: El ancho de banda de los buses, como el bus de datos o el bus de direcciones, puede limitar la cantidad de datos que una aplicación puede transmitir a través del sistema. Si una aplicación requiere un ancho de banda de bus más alto del que está disponible, puede provocar una disminución en el rendimiento.

Cache intensive: Si una aplicación requiere una gran cantidad de datos que no se encuentran en la caché, puede provocar un retraso en el rendimiento mientras se accede a los datos en la memoria principal.

CPU intensive: Las aplicaciones que requieren un uso intensivo de la CPU pueden limitar el rendimiento de otras aplicaciones en ejecución en el sistema.

IO intensive: Las aplicaciones que acceden frecuentemente a la entrada salida, como puede ser un disco duro, pueden limitar el rendimiento del sistema debido a la velocidad de lectura y escritura limitada del dispositivo. Si una aplicación requiere una gran cantidad de acceso al disco, puede provocar un retraso en el rendimiento.

Al ejecutar programas que utilizan los mismos recursos los programas competirán por estos y reducirán la eficiencia del sistema. Sin embargo, si los recursos comunes son mínimos, la eficiencia general del computador incrementará. A este efecto se le llama consolidación.

2.2. State of the Art

La eficiencia energética es notoriamente investigada en el ámbito de las investigaciones sobre los schedulers. En esta sección se detalla cada técnica que se va a estudiar.

2.2.1. Consolidación

La consolidación es un proceso de combinar programas para que los múltiples recursos informáticos de un sistema sean utilizados. Esta técnica se utiliza para mejorar la utilización de los recursos y optimizar la eficiencia del sistema. Se suele referir a esta técnica también como el co-scheduling (Bhadauria and McKee, 2010). La consolidación muestra buenos resultados en el incremento de eficiencia combinando aplicaciones de diferentes naturalezas o tipos, de media un 7 % (Zacarias et al., 2021). Se podrían agrupar en cuatro categorías o naturalezas principales. Las categorías son CPU intensive, caché intensive, IO y memory intensive.

En la paralelización ocurre frecuentemente que una aplicación no utilice todos los recursos del sistema. La consolidación tiene como objetivo utilizar estos recursos adicionales para mejorar el rendimiento general del sistema. Por ejemplo, si un clúster de servidores de bases de datos no está funcionando al máximo de su capacidad, se pueden agregar más servidores al clúster para aumentar la capacidad de procesamiento y la eficiencia, siempre y cuando esos servidores no sobrecarguen los otros recursos en utilización.

Los programas CPU intensive transcurren la mayoría de la ejecución del programa con un uso de la CPU y accesos a la caché L1 alto pero con accesos a memoria y otros elementos limitados. Mientras más rápida sea la CPU, más rápido se ejecutarán en general.

Los LLC users utilizan la caché de último nivel y son sensibles a que sus datos sean reemplazados, siendo los principales culpables de ese efecto los memory streaming. Estos últimos leen archivos o continuamente están guardando y leyendo datos en las cachés de mayor nivel y acaban interfiriendo con los programas LLC.

Con la consolidación lo que se busca es agrupar estos programas categorizados para minimizar la interferencia entre ellos, maximizando la utilización de recursos del sistema.

En general, la consolidación en la paralelización es una estrategia efectiva para maximizar la utilización de los recursos y mejorar la eficiencia del sistema. Al hacer un uso más efectivo de los recursos, se puede lograr un mejor rendimiento en general y reducir los costos asociados, como el coste energético.

2.2.2. Variabilidad

La variabilidad se le llama a la pequeña diferencia que se genera entre dos procesadores del mismo fabricante y serie en el proceso de fabricación o deterioro. (Chasapis et al.,

2016)

La variabilidad es un fenómeno común en la fabricación de procesadores, donde pequeñas diferencias en el proceso de fabricación pueden resultar en variaciones en el rendimiento del procesador. Aunque los procesadores se fabrican en serie, cada chip es único y presenta ligeras variaciones en su comportamiento y rendimiento.

En el proceso de fabricación de los procesadores, se utilizan técnicas de litografía para crear patrones en la superficie del silicio que luego se utilizarán para construir los transistores y otros componentes del procesador. Sin embargo, la litografía es un proceso complejo y susceptible a variaciones, lo que puede resultar en pequeñas diferencias en el tamaño y la forma de los componentes del procesador.

Además de las variaciones en el proceso de fabricación, los procesadores también pueden experimentar variaciones en su rendimiento debido al desgaste y el envejecimiento, lo que puede afectar el rendimiento del procesador con el tiempo.

La variabilidad es un desafío importante para los fabricantes de procesadores, ya que deben garantizar que los procesadores funcionen dentro de un rango de especificaciones y que cumplan con los requisitos de calidad y rendimiento. Para abordar este problema, los fabricantes de procesadores utilizan técnicas como la calibración y la clasificación de procesadores para garantizar que los procesadores funcionen dentro de los límites especificados.

Esa variación puede llegar a ser muy pequeña, pero en largas ejecuciones esa diferencia se va sumando y puede llegar a tener un impacto considerable. En computadores convencionales ese efecto es despreciable, pero en centros de computación de altas prestaciones que están en ejecución el mayor tiempo posible, este efecto no debe menospreciarse dado que está formado desde cientos hasta miles de procesadores. Además, es suficientemente importante como para analizar sus efectos junto a la consolidación (Teodorescu and Torrellas, 2008). Como es de esperar, este efecto mejora en teoría el rendimiento de las aplicaciones CPU intensive.

2.2.3. Scheduling

La planificación o scheduling consiste en repartir las tareas entre los recursos del sistema y existen dos niveles en los entornos HPC. El scheduling de alto nivel gestiona el reparto de las tareas en los clústers, SLURM es un scheduler de alto nivel. El scheduler de bajo nivel se encarga de asignar y controlar los procesos de un sistema operativo.

Hay investigaciones que tratan sobre la planificación de alto nivel y los parámetros que se utilizan para planificar las tareas, una de ellas es Chasapis et al. (2019). Diferentes políticas de reparto y tiempos permitidos en ejecución. En este trabajo, la planificación de alto nivel no se ha investigado por las siguientes razones.

Primero, se tiene que cumplir unas reglas para asegurar la ejecución equitativa de las aplicaciones de los usuarios. Se tiene que tener en cuenta el número de tareas que lanza un usuario, el tiempo estimado que va a estar en ejecución, el tiempo que se acaba tomando, etc. Esto limita el reparto a cierto marco y al ser una primera investigación se ha optado por no seguir este camino.

Segundo, para analizar los efectos de una planificación de alto nivel es necesario coordinar los nodos y obtener muchos datos de ellos. La técnica que se utilice para obtener y transferir los datos está también fuera del alcance.

La planificación de bajo nivel tiene menos requisitos o exigencias, las tareas que tiene que repartir dentro de un nodo ya está determinado por el scheduler superior y este tiene que repartirlo en su sistema local que ya conoce. Otras investigaciones miran los efectos de la variación de los chips que se genera por las técnicas de fabricación. Otros estudian el efecto de tener las aplicaciones que requieren de mucho uso de memoria en los núcleos más cercanos físicamente y lógicamente a la memoria.

3. CAPÍTULO

Gestión y planificación del Trabajo de Fin de Grado

En el transcurso de cualquier proyecto, la gestión y planificación resultan indispensables a la hora de mantener una evolución y progreso del mismo, así como de cara a prevenir posibles riesgos. En esta sección se describen todos los aspectos relativos a la gestión del proyecto: metodología, organización, costes, planificación, riesgos y aseguramiento de la calidad.

3.1. Metodología del proyecto

La metodología se ha dividido en varias fases durante la fase de vida del proyecto. Antes de especificar los detalles en cada una de ellas, se va a explicar la metodología general.

En cada fase, se realizarán reuniones cada dos semanas con los directores para comprobar el progreso del proyecto y corregir o tratar con problemas que surjan.

El estudiante trabajará continuamente y regularmente cumpliendo como sea posible y necesario con las estimaciones de tiempos de cada tarea. Cada tarea tiene en su sección especificado lo que se realizará.

3.1.1. Fase de Gestión de Proyecto

Esta esencial fase consiste en preparar la planificación inicial, la metodología a seguir durante el desarrollo del proyecto, detectar posibles riesgos y todo relacionado con la

gestión de proyectos. Esta fase tiene tareas que se extienden hasta el final del proyecto.

A continuación se especificarán las tareas que se realizarán durante esta fase:

1. **Concretar objetivos y limitar alcance:** Especificar los objetivos que se quieren lograr con el proyecto y limitar teniendo en cuenta los recursos limitados el alcance para cumplirlo. Para ello se diseñarán unos paquetes de trabajo y se estimará su coste temporal.

2. **Diseño de la metodología del proyecto:** Para obtener el objetivo de forma eficiente se desarrolla esta metodología del proyecto. Se documenta y especifica lo que se realizará a lo largo de la vida del proyecto. Lo que implica es identificar herramientas adecuadas para la tarea, diseñar metodología de análisis de los datos y tratar el código a implementar y documentar.

3. **Control y seguimiento:** Controlar y registrar tiempo empleado en las tareas para la documentación o toma de decisiones por desviaciones. Cada dos semanas se realizará una reunión con los directores y se evaluará el progreso.

4. **Análisis y detección de riesgos:** Detectar posibles riesgos que conlleva este proyecto, analizarlos y plantear posibles acciones en caso de que ocurran. En caso de que ocurra algún evento no previsto, se documentará en su apartado.

3.1.2. Fase de documentación

La documentación del proyecto y la memoria del proyecto se realiza en paralelo al trabajo de la gestión del proyecto, investigación, desarrollo y análisis. Se documentará todo en el formato LaTeX.

1. **Memoria:** Redacción de esta misma memoria cumpliendo con los requisitos definidos.

2. **Referencias:** Citas a los artículos relacionados con la investigación.

3. **Presentación:** Elaborar presentación para la defensa del trabajo fin de grado mostrando el trabajo del proyecto de forma resumida.

3.1.3. Fase de investigación

En esta fase se desarrollará la sección del estado del arte y metodología necesaria para la investigación. Para realizar una investigación es necesario conocer el estado del arte del tema sobre el que se va a trabajar, clasificación y efectos de los diferentes tipos de aplicaciones, específicamente del área de la computación y cálculos científicos. La información obtenida para la investigación será resumida, citada y referenciada adecuadamente para los lectores y así evitar tener que leer los mismos artículos para entender el informe, pero como se mencionó, se incluirán referencias para profundizar en temas de interés.

Además, se estudiarán las herramientas, aunque más específicamente son interfaces, que se utilizan para obtener las métricas de una aplicación en ejecución.

1. **Contexto:** Investigar estado del arte del aumento de eficiencia en computadores de altas prestaciones y analizar las metodologías que utilizan para analizar los resultados.

2. **Herramientas:** Buscar y escoger herramienta que permita obtener métricas en tiempo real de las aplicaciones en ejecución para realizar un perfilador.

3.1.4. Fase de desarrollo

Esta fase tiene su propia sección en la memoria sobre el desarrollo del perfilador. Un perfilador es una herramienta que ayuda a analizar el rendimiento de un programa mediante la medición de varias métricas, como el uso de la memoria, la utilización de la CPU y el tiempo de ejecución de las funciones individuales. Se puede utilizar para identificar cuellos de botella, localizar código lento o ineficiente y utilizando esos datos, optimizar el programa para un mejor rendimiento. En este proyecto, se quiere utilizar para determinar la naturaleza de la aplicación a paralelizar o paralelizada. En esta misma fase también se tomarán los datos de ejecución y se procesarán para su análisis posterior.

1. **Desarrollar perfilador:** Utilizando los contadores hardware e Intel RAPL, desarrollar un perfilador que tome el número de ciclos de CPU, los accesos a memoria RAM, los fallos de cache y otro número de métricas que ayudarán a clasificar una aplicación. Se clasificarán como CPU-intensive, exigentes de ancho de banda y sensibles a cambios en los últimos niveles de la cáche. Esta clasificación se determinará en su apropiada sección con más detalle.

2. **Obtener datos de ejecución y procesarlos:** Utilizar el perfilador desarrollado para obtener una medición de las métricas de interés de una específica lista de aplicaciones. Estas aplicaciones serán seleccionadas sabiendo de antemano sus características para poder calibrar o localizar con facilidad otro tipo de aplicaciones con naturaleza parecida. Además, se tendrá que preparar los diversos entornos en los que se toman los datos, con el menor ruido posible y los datos tendrán que ser exportados a herramientas de análisis de elementos separados por coma para su análisis.

3.1.5. Fase de análisis

Los datos de la fase de desarrollo se analizarán en esta fase. El objetivo de esta es la obtención de gráficos, patrones o algunos rasgos que ayuden a caracterizar las aplicaciones. Se quieren analizar los efectos de ejecutar aplicaciones con combinaciones de distintas o mismas naturalezas. Se va a estudiar lo siguiente:

1. **Efectos del perfilador:** Medir la carga que produce el perfilador software sobre el computador en cuanto a tiempo de ejecución. Habrá que poner interés especial en aplicaciones sensibles a cambios en la caché.

2. **Efectos de la consolidación:** Calcular mejora de rendimiento al consolidar los recursos del sistema.

3. **Efectos de la variabilidad:** Valorar la diferencia entre dos CPUs del mismo fabricante que se generan por las técnicas de fabricación. Comprobar si esa pequeña diferencia crea un efecto para tenerlo en cuenta al planificar tareas.

3.2. Tecnología y herramientas

Las herramientas de este proyecto son comunes entre otros proyectos: un entorno de programación, un software de hojas de cálculo y un software para tomar notas. En cuanto a la tecnología, eso es más específico de la rama de arquitectura.

3.2.1. RAPL Intel

RAPL (Running Average Power Limit) es una tecnología de Intel que permite monitorear y controlar el consumo de energía de un procesador u otros componentes del sistema, como la memoria y los gráficos. Se puede usar para optimizar la eficiencia energética, diagnosticar problemas relacionados con la energía y proteger contra fallas relacionadas con la energía. RAPL es capaz de ajustar dinámicamente los límites de energía en función de la carga de trabajo, la temperatura y otros factores, y puede proporcionar información detallada sobre el uso de energía en diferentes niveles del sistema. El uso principal que se le dará en este proyecto es el de la obtención de las métricas desde las herramientas de perf event. [1]

3.2.2. perf tools

Su nombre completo es Performance Counters for Linux es una herramienta de análisis de rendimiento. Se utiliza para realizar un perfíl estadístico de todo el sistema utilizando contadores hardware, contadores software o puntos de rastreo. A pesar de ser reconocida como la herramienta de perfilado de contadores de rendimiento más utilizada, la documentación no detalla los eventos ni los alias utilizados. El uso que se le dará a esta herramienta en este proyecto es de ser el recolector de los datos. Posee acceso a los contadores hardware y software de forma que genere las menores sobrecargas posibles. [2]

3.2.3. GNUPlot

Esta herramienta de código abierto es un visualizador de datos mediante líneas de comando. Su uso principal será graficar los datos obtenidos de las ejecuciones y luego examinar y discutir esos gráficos. Al visualizar los datos podremos obtener con mayor facilidad las características de las aplicaciones de distintas naturalezas al observar la media de los fallos de acceso a memoria, cantidad de escrituras realizadas en memoria o números busy del procesador. Los datos recopilados en la fase de desarrollo se guardarán en un formato .csv para ser importados en software de hojas de cálculo con comodidad, separados por ':' en lugar de ','.

[1] RAPL Web (último acceso 23 de junio de 2023)
[2] perf.wiki.kernel.org (último acceso 23 de junio de 2023)

3.2.4. Obsidian

Este software es un software que funciona sobre en una carpeta local de archivos de texto sin formato. El uso principal que se le dará a esta aplicación será anotar citas de los artículos leídos, apuntar conocimientos teóricos para incorporar en la memoria y poder agrupar los artículos en base a los temas sobre los que tratan. De esta forma, el citado y referenciado de los artículos estará bien organizado y facil de gestionar. [3]

3.3. Estimación temporal

En esta sección, un diagrama de Gantt mostrará los hitos centrales de los proyectos. Luego la Tabla 3.3.3 muestra el tiempo estimado y la dedicación puesta en el proyecto.

3.3.1. Diagrama EDT

Con la intención de relacionar las fases planteadas en la metodología del proyecto con paquetes de trabajo y después visualizarlos en el diagrama Gantt, se ha realizado un diagrama EDT (Estructura de Descomposición del Trabajo). En la Figura 3.1 se puede observar la descomposición.

3.3.2. Diagrama Gantt

Para la visualización de las tareas programadas se ha utilizado en diagrama Gantt. De esta forma, podemos identificar los momentos con más carga de trabajo colocando las tareas en una línea temporal, como se puede observar en la Figura 3.2 .

3.3.3. Estimación de tiempos y dedicación

A cada apartado del proyecto se le ha estimado un tiempo para completarlo. Este apartado recoge esas estimaciones para registrar y medir la desviación en caso de que ocurra. La tabla 3.3.3 muestra dicha estimación.

[3] obsidian.md (último acceso 23 de junio de 2023)

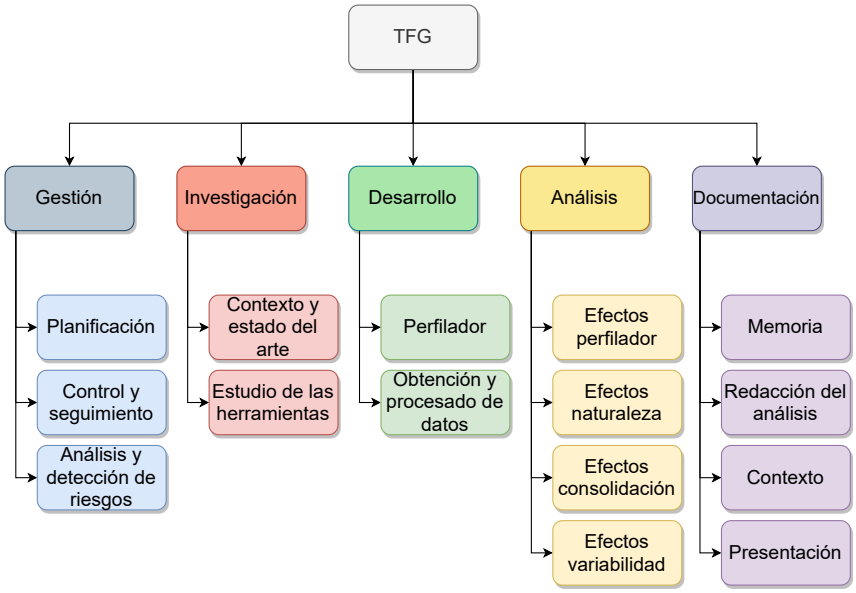


Figura 3.1: Diagrama EDT

3.4. Análisis de las desviaciones

En general, se puede decir que tres han sido los principales factores que han producido las desviaciones más significativa respecto a las horas estimadas en los diferentes paquetes. Por un lado, el colchón de horas extra puesto en paquetes como *Desarrollo* ha cumplido su función y no se han generado demasiadas horas extra. Esto se ha debido al problema con la implementación que no generaba bien los datos al ejecutarse en el nodo seleccionado.

Además por otra parte se ha producido otra importante desviación de 8 horas extra sobre el paquete de trabajo correspondiente a la *Gestión*. Esto se ha debido sobre todo al control y tomas de decisión tomadas para reaccionar al problema de la implementación.

Por último, mencionar que los días extra reservados antes de la fecha límite para acabar el proyecto han sido de gran ayuda, ya que no se pudo invertir el número de horas estimado para cumplir ese plazo, siendo el día 14 de Junio el día de finalización del proyecto. A pesar de ello mencionar que se siguieron retocando ciertos detalles hasta el mismo día de la entrega (25 de Junio).

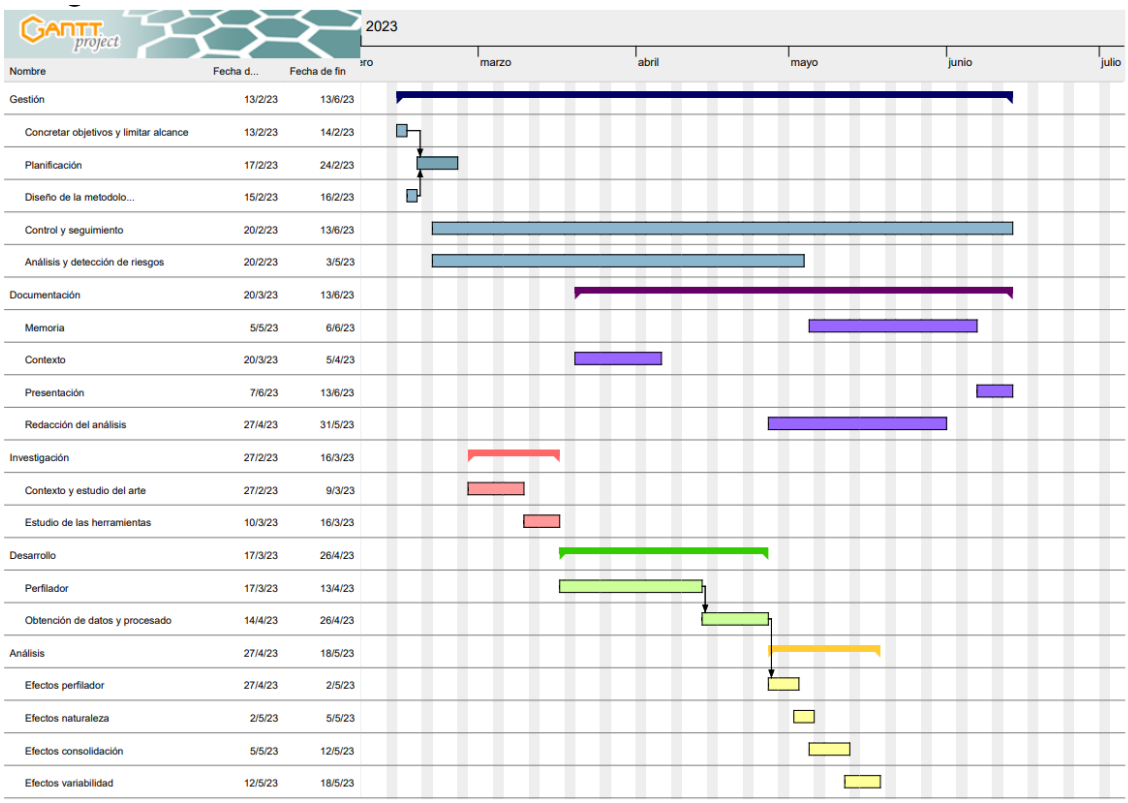


Figura 3.2: Diagrama Gantt

3.5. Organización

En este proyecto hay tres personas involucradas, el estudiante y los directores. El estudiante investigará y trabajará bajo la guía de los directores.

El estudiante tiene un equipo personal para desarrollar el perfilador y analizar los resultados, un portátil regular. Sin embargo, el computador sobre el que se ejecutarán las aplicaciones para su medición será proporcionado por los directores. Este equipo es un nodo y sus características se han detallado en la sección 2.1.

3.6. Riesgos

Existen algunos riesgos que retrasarían el proyecto o lo harían demasiado complicado para una medición o análisis factible. Esos riesgos son los siguientes, junto a sus posibles soluciones para la reducción de daños:

Plan temporal		
Asunto	**Tiempo estimado (h)**	**Tiempo dedicado (h)**
Gestión	30	38
Planificación	10	12
Seguimiento	14	14
Control	6	12
Investigación	50	48
Contexto y estudio del arte	35	34
Estudio de las herramientas	15	14
Desarrollo	50	60
Desarrollar perfilador	48	57
Obtención y procesado de datos	4	3
Análisis	45	44
Efectos del perfilador	5	3
Efectos de la naturaleza	15	21
Efectos de consolidación	20	16
Efectos de variabilidad	5	4
Documentation	115	118
Memoria	60	71
Redacción de los análisis	30	31
Contexto	7	6
Presentación	10	10
Total	300	308

Tabla 3.1: Tabla de dedicación del proyecto

Inaccesibilidad de hardware compatible, para ser específicos, acceso a un nodo o compatibilidad de RAPL

- Probabilidad: Moderada.
- Impacto: Alto.
- Posible solución: La facultad tiene un clúster en su disponibilidad para asignaturas del grado. Se podría pedir una cuenta al administrador para emplearla en el TFG.

No conseguir correlaciones/resultados adecuados en los datos obtenidos

- Probabilidad: Baja.
- Impacto: Medio.

- Posible solución: Utilizar otros contadores hardware que hay en disposición.

Obtener datos con mucho ruido

- Probabilidad: Baja.
- Impacto: Bajo/Medio.
- Posible solución: Buscar y desactivar servicios que produzcan el ruido o trabajar con variaciones más grandes al clasificar programas.

Problemas con la implementación del perfilador desarrollado

- Probabilidad: Baja.
- Impacto: Alto.
- Posible solución: Utilizar uno de los perfiladores que se han explorado y adaptar la implementación.

No encontrar características que determinan la naturaleza de las aplicaciones

- Probabilidad: Baja.
- Impacto: Bajo/Medio.
- Posible solución: Dejar el análisis a grano fino de lado y continuar analizando en base al tiempo de ejecución.

No encontrar efectos notorios en los análisis de interés

- Probabilidad: Media.
- Impacto: Ninguno/Bajo.
- Efecto: Empobrecimiento de la memoria al no obtener resultados satisfactorios.
- Posible solución: Investigar la razón de la falta de efectos.

4. CAPÍTULO

Metodología

En una investigación, es de suma importancia diseñar u obtener una metodología a la cual ceñirse durante las pruebas. De esta forma, se tienen asentados los benchmarks que se van a utilizar, las diferentes herramientas de perfilado que se tienen en disposición para realizar los experimentos y la forma en la que se van a realizar los diferentes experimentos. Este capítulo trata sobre ello.

4.1. Aplicaciones de benchmark

Las aplicaciones que se han utilizado para investigar los diversos efectos se han sacado de PARSEC[1]. Parallel Application Research for Scalable Computing (PARSEC) es un conjunto de aplicaciones de prueba que se usan para evaluar la capacidad y el rendimiento de sistemas de procesamiento paralelo y de alto rendimiento. Es una herramienta que ha sido desarrollada por un equipo de investigadores de la Universidad de Princeton, y su objetivo es proporcionar una variedad de aplicaciones de prueba realistas que permitan a los desarrolladores y a los investigadores optimizar el rendimiento y la eficiencia de sus sistemas.

PARSEC es una herramienta muy utilizada en la investigación de computación paralela y de alto rendimiento y ha sido adoptada por numerosos grupos de investigación y empresas de tecnología de todo el mundo. Además, es un software de código abierto.

[1] PARSEC (último acceso 23 de junio de 2023

Unas de las razones principales por las que se ha escogido PARSEC sobre otro conjunto como puede ser SPLASH, es que se han encontrado más menciones sobre PARSEC que SPLASH en la fase de investigación del proyecto (entre las más notorias, Chasapis et al. (2019) Hankendi and Coskun (2012)). Además, es fácil de utilizar y está enfocada a la investigación, mientras que SPLASH está más enfocada al HPC.

De las 12 aplicaciones que incluye el paquete de trabajo de PARSEC (Bienia et al., 2008) se trabajará con 5: blacksholes, bodytrack, fluidanimate, freqmine y streamclusters. La Tabla 4.1 muestra resumidamente los tipos de las aplicaciones y sus características de paralelización. Las *X* indican su naturaleza principal y la + significa que tiene un elevado uso de ese recurso.

Tabla 4.1: Tipos de aplicación

Benchmark	IO	CPU	Caché	Memoria	Granularidad	Sincronización
blackscholes		X			Media	Baja
bodytrack		X	+		Media	Media
fluidanimate		+	X	X	Fina	Alta
freqmine	X				Media	Media
streamcluster	X	+			Media	Baja

- **blacksholes**: La aplicación Blackscholes es un benchmark de referencia de Intel RMS. Calcula los precios para una cartera de opciones europeas analíticamente con la ecuación diferencial parcial de Black-Scholes.

  Su naturaleza principal es CPU intensive, pero tiene varias fases de ejecución y se puede considerar que es variada. Su paralelización es de grano grueso con sincronización muy baja entre los procesos.

- **bodytrack**: La aplicación de visión por computadora bodytrack es un benchmark de Intel RMS que rastrea una pose 3D de un cuerpo humano sin marcadores con múltiples cámaras a través de una secuencia de imágenes. Consta de tres fases:

  - Detección de esquinas utilizando un detector de gradiente.
  - Aplica filtro gaussiano, dos fases paralelas, filtro vertical y horizontal.
  - La fase más costosa computacionalmente, detecta las siluetas y calcula los pesos.

    Su naturaleza es CPU intensive. Tiene sincronización media entre procesos de granularidad media.

- **freqmine**: Es una aplicación que busca patrones frecuentes en grandes conjuntos de datos. Esta aplicación realiza operaciones de minería de datos.

  - Construye un árbol, tiene que leer la base de datos
  - Construye el árbol de prefijos, vuelve a leer la base de datos.
  - Fase de minería de datos, crea más arboles de forma recursiva

  Su naturaleza es IO al utilizar mucho la entrada y salida, tiene una paralelización de grano medio y sincronización media entre procesos.

- **fluidanimate**: La aplicación de simulación de fluidos suavizado es un benchmark de Intel RMS. Simula los fluidos para animaciones interactivas. Se utiliza las ecuaciones de Navier-Stokes.

  Por cada time-step esto es lo que ejecuta:

  - Reconstruye el índice espacial suavizando el entorno
  - Calcula las densidades de las partículas
  - Calcula las fuerzas y se detectan las colisiones
  - Gestiona las colisiones con la geometría de la escena
  - Actualizar la posición de las partículas

  Su naturaleza es memory y caché intensive junto a un gran uso de la CPU. Su uso de la caché y memoria aumenta con el tamaño del problema y escala rápidamente. Su paralelización es de grano fino y requiere de una alta necesidad de sincronización entre sus procesos.

- **streamcluster**: Es una aplicación que realiza clustering de datos en tiempo real. Es una aplicación intensiva en IO con un gran uso de CPU.

  Es un programa que hace stream de datos, por lo tanto realiza muchas lecturas, llena la cache y la memoria de datos que va a utilizar pocas veces. Requiere que tenga libre el ancho de banda para leer los datos. Su paralelización es de grano medio y sincronización baja.

4.2. Aplicaciones de perfilado

Las aplicaciones de perfilado son herramientas de diagnosis de rendimiento. Hay diversas formas de medir el rendimiento de una aplicación, desde el caso más simple, medir el tiempo de ejecución de una aplicación hasta medir el número de ciclos que requiere un programa en ejecución. Su uso general es encontrar cuellos de botella para que el desarrollador lo modifique y aumente el rendimiento de la aplicación. En este caso, no se utiliza para modificar la aplicación, si no para determinar el tipo de aplicación y medir los efectos de las diferentes técnicas de consolidación.

De la misma forma que las aplicaciones de benchmark, se han investigado diversas opciones para elegir el programa principal de perfilado. Se ha acabado escogiendo perf tools pero se van a comentar las otras tres alternativas encontradas:

- OProfile: Oprofile (OProfile, 2023) es una herramienta de perfilado de sistema libre y de código abierto para sistemas Linux. Se utiliza para medir el rendimiento del sistema y las aplicaciones en tiempo real, y proporciona información detallada sobre el uso de la CPU, la memoria y otros recursos del sistema. El problema que tiene respecto a este proyecto, es que es demasiado detallada en cuanto a proceso. Realiza un desglose de las llamadas de las funciones internas de las aplicaciones que está perfilando.

- Valgrind: Valgrind (Valgrind, 2023) es una herramienta de análisis de memoria y depuración de código abierto y multiplataforma disponible para sistemas Linux y macOS. Se utiliza para encontrar errores y problemas en el código fuente de una aplicación, especialmente aquellos relacionados con el uso incorrecto de la memoria, como fugas de memoria, uso de punteros no válidos y acceso a memoria no inicializada. También puede proporcionar información detallada sobre el rendimiento del código, como la cantidad de tiempo que se dedica a cada función y la frecuencia de ejecución de las instrucciones.

  Aunque utilice contadores de recursos, está más enfocada a la depuración y a la gestión de memoria, es por eso que se ha optado por no utilizarla.

- Extrae: Extrae (Extrae, 2023) es una herramienta de perfilado de alto rendimiento desarrollada por el grupo BSC (Barcelona Supercomputing Center) para analizar y optimizar el rendimiento de aplicaciones de computación de alto rendimiento.

Extrae se utiliza para medir el rendimiento de aplicaciones paralelas y distribuidas, como aquellas escritas en MPI, OpenMP y CUDA.

Extrae proporciona una variedad de información detallada sobre el rendimiento de la aplicación, como el tiempo de ejecución de cada función, la frecuencia de llamadas a funciones específicas, la carga de trabajo en cada núcleo y la cantidad de tiempo dedicado a la comunicación y sincronización entre procesos.

Aunque estaría bien utilizar Extrae, depende mucho de la implementación de la paralelización.

En este proyecto se utiliza perf event tools por la facilidad que tiene para elegir los parámetros a leer. Perf event tools es un conjunto aplicaciones de perfilado que se incluye en Linux. Utiliza los contadores hardware que tienen los procesadores modernos para obtener una lectura más precisa que sus parecidos. Aunque la implementación sea mas compleja, el potencial de los resultados de las ejecuciones es mayor al tener más margen con la opción de utilizar los contadores hardware. Además, no es una aplicación en segundo plano lo que ayuda a reducir el ruido de las mediciones ya que se realizan llamadas al sistema para indicar cuando empieza y para la medición.

4.3. Metodología a seguir

Tener definida una metodología es importante en cualquier proyecto, pero en una investigación es esencial. Sirve como guía en el desarrollo del proyecto y define el enfoque concreto que va a tener. De esta forma aumenta la eficiencia del proyecto. En este apartado se va a definir la metodología que se ha seguido en la obtención de datos para el análisis.

4.3.1. Metodología común

A la hora de obtener los datos de las ejecuciones es importante que las unidades que se obtengan sean las mismas o estén medidas en un intervalo conocido. Es por eso que el perfilador tendrá una frecuencia de 5 mediciones por segundo. Es decir, cada 0.2 segundos realizará una lectura de los contadores. Se ha escogido esta frecuencia porque no interesa saber que está ocurriendo en la aplicación con mucho detalle, solo interesa saber la naturaleza general de la aplicación. Si se llega a perfilar con mayor frecuencia, no solo puede

generar un impacto mayor en la aplicación perfilada, sino que esos datos resultarían redundantes, quiero decir, el número de eventos que ocurren en 1 segundo van a ser más o menos los mismos si se miden cada 0.1s o cada 0.01s, solo que en el segundo caso hay más sondeos. Además, con un perfilado grueso evitamos crear ficheros pesados con los que dificultaría la fase de análisis y gestión de los mismos archivos. Al recopilar datos, el último ciclo de cada ejecución se descartará, dado que puede terminar antes del intervalo de medición y midiendo menos tiempo que el periodo seleccionado y eso alteraría el equilibrio de las cuentas.

Los eventos que van a recopilar son generales para todas las aplicaciones, pero en cada una se profundizará más un campo que otro, según la naturaleza. Los eventos se pueden encontrar en la documentación de perf_event_open [2] o con el comando *perf stat list* muestran los que la máquina puede recopilar. Estos eventos son los siguientes:

- timestamp: Tiempo en el que se han obtenido los eventos.
- instructions: Cantidad de instrucciones ejecutadas en el intervalo.
- cycles: Cantidad de ciclos de procesador realizados en el intervalo.
- stalled-cycles-frontend: Ciclos parados o abortados por saltos condicionales o dependencias.
- stalled-cycles-backend: Ciclos parados por falta de recursos o esperando a datos. Esto tiene que ver con los accesos a memoria.
- major-faults: Fallos de página que requieren traer la página de memoria a la memoria RAM.
- page-faults: Número de fallos de página. Nos da una métrica de cuanta memoria utiliza o la localidad de los datos a los que está accediendo.
- LLC-load-misses: Accesos fallados de carga de a caché de último nivel.
- LLC-loads: Accesos de carga a cache de último nivel.
- LLC-stores: Accesos a escritura de datos a cache de último nivel.
- LLC-store-misses: Accesos fallados de escritura de datos a cache de último nivel.
- L1-dcache-load-misses: Fallos de carga a la cache de datos del primer nivel.

[2] https://www.man7.org/linux/man-pages/man2/perf_event_open.2.html

- L1-dcache-loads: Número de cargas a la cache de datos del primer nivel.
- L1-dcache-stores: Número de escrituras en la cache del primer nivel.
- L1-dcache-store-misses: Número de escrituras falladas en la cache del primer nivel.
- cs: Número de cambios de contexto.
- migrations: Número de migraciones de procesos entre CPUs.

De esos datos, luego se obtendrán datos derivados, como porcentaje de fallos a los diferentes niveles de cache, ciclos perdidos o IPC (instructions per cycle).

Las unidades de accesos a memoria, instrucciones ejecutadas y en general, las mediciones realizadas por perf tools no tienen unidades que puedan tener un significado complejo. Por lo que la única unidad que se tiene que definir es el tiempo, en segundos.

Cabe mencionar que se iba a medir otro evento, bus-cycles, pero es incompatible con el equipo.

Una ejecución de un programa nunca es la misma aunque sea el mismo equipo el que esté ejecutando por segunda vez el programa, a esa incertidumbre lo llamamos ruido de ejecución. Este ruido puede que a veces genere grandes atascos en la cache y la ejecución temporal de un programa cambia. Para minimizar el ruido ya se han tomado medidas como reducir el ruido del sistema operativo utilizando uno de menor impacto. Para suavizar aún más el ruido, se van a ejecutar 16 veces los experimentos y se tomarán las medias y desviaciones típicas de las ejecuciones. De esas 16 ejecuciones, se tomarán los datos de la ejecución más cercana a la media y en caso de detectar outliers, serán reemplazados por otras ejecuciones realizadas en las mismas condiciones. Con la desviación típica podremos entender como de precisa es la medición, o en este caso, podemos saber cuánto ruido sufre ese experimento.

4.3.2. Efecto del perfilador

El objetivo de este experimento es obtener el sobrecoste del perfilador desarrollado. Conocer el efecto mejorará los resultados de la investigación al tener un valor concreto del deterioro de rendimiento y mayor o menor certeza en las mediciones.

Para ello, se ejecutarán las aplicaciones con el perfilador y sin él. Se compararán los tiempos obtenidos. En el caso de que el sobrecoste sea despreciable, no se considerará en los próximos experimentos.

4.3.3. Tipo de aplicación

Lo que se busca lograr con este experimento es llegar a determinar la naturaleza o tipo de aplicación con los datos de ejecución obtenidos con el perfilador. Nos interesa conocer la naturaleza en los casos en los que el usuario duda de los recursos que utiliza o porque el tipo de la aplicación cambia en mitad de la ejecución.

No se busca un determinante que separe las clases, más bien unos indicadores o características que compartan que puedan a llegar a ser utilizadas por un técnico o una inteligencia artificial en un trabajo futuro.

4.3.4. Efecto de consolidación

Con este experimento se busca cuantificar cuanto incremento de eficiencia se consigue en cada caso de consolidación y de esta forma conocer los mejores casos para combinar distintos tipos de programas.

Las combinaciones posibles para analizar la consolidación 15 en total, pero se han escogido estas parejas y en la Tabla 4.2 se ve la competencia por recursos:

- EXP1 - blacksholes y bodytrack: Interacción de programas principalmente CPU intensive, se espera pérdida de rendimiento.
- EXP2 - fluidanimate y bodytrack: Interacción de programas CPU intensive con alta sincronización.
- EXP3 - blacksholes y fluidanimate: Interacción de programas con principalmente alta demanda de CPU, uno requiriendo de sincronización constante. Un programa es de naturaleza variada. Puede que la pérdida y mejora de rendimiento de los diferentes recursos sea positiva en total.
- EXP4 - freqmine y blacksholes: Naturaleza variada de baja sincronización entre hilos lanzado junto a un programa que requiere de uso de lectura/escritura en disco.
- EXP5 - freqmine y fluidanimate: Las naturalezas son CPU e IO, pero ambas utilizan la cache bastante.
- EXP 6 - freqmine y streamcluster: Uno es IO intensive y el otro constantemente utiliza IO y memoria.

- EXP7 - streamcluster y bodytrack: IO intensive con sobrescritura de cache con programa que utiliza bastante memoria pero es CPU intensive.

Para cada experimento se ha hecho una hipótesis de su funcionamiento y se va a comprobar si se cumplen. Los experimentos 1,2,3,6,7 deberían dar resultados iguales o peores a ejecutarlos por separado. Por otro lado, los experimentos 4 y 5 deberían dar mejores resultados.

Tabla 4.2: Recursos compartidos por los experimentos

	Exp1	Exp2	Exp3	Exp4	Exp5	Exp6	Exp7
IO				X	X	XX	X
CPU	XX	X	X	X		+	X+
Caché	+	X+	X		X		+
Memoria		X	X		X		

4.3.5. Efecto de variabilidad

La variabilidad no está estrechamente relacionada con la consolidación pero es un efecto que puede llegar a aumentar la eficiencia en los computadores que disponen de más de un procesador. El beneficio es minúsculo a nivel de escala de tiempo a la que estamos habituados, pero en ejecuciones prolongadas puede a tener un impacto.

Por lo tanto, con el fin de analizar el efecto de la variabilidad de manera más precisa, se necesitan realizar múltiples ejecuciones del experimento para reducir el posible ruido en los resultados promedio y la desviación estándar. Según la metodología propuesta, se recomienda realizar 16 ejecuciones, sin embargo, este número es insuficiente para este experimento en particular. Por esta razón, se llevarán a cabo 128 ejecuciones del programa bodytrack, el cual se considera adecuado para observar el efecto deseado debido a su alta carga de trabajo en la CPU y su baja manipulación de datos. Se intentará ejecutar el programa utilizando los 6 hilos de cada CPU, de forma individual, pero en caso de que esto resulte incompatible con la forma en que está paralelizado, se reducirá el número de hilos a 4.

5. CAPÍTULO

Desarrollo

Este capítulo trata sobre todos los aspectos relacionados con la implementación. Como se ha mencionado en el Apartado 4.2, se va a utilizar la herramienta de perfilado perf event tools. Esto requiere una implementación sencilla por parte del usuario para poder utilizarlo de forma personalizada. Los experimentos que vamos a realizar requieren de una sensibilidad alta y es por ello que se ha decidido implementar una herramienta propia, para poder elegir que datos se acaban registrando.

5.1. Entorno de Desarrollo

En esta sección se debe indicar el marco tecnológico utilizado para la construcción del sistema: entorno de desarrollo (IDE), lenguaje de programación, herramientas de ayuda a la construcción y despliegue, control de versiones, repositorio de componentes, integración contínua, etc.

El perfilador y el resto de herramientas se han desarrollado principalmente en Visual Studio Code en el lenguaje de programación C y en el lenguaje de scripting bash. El repositorio que contiene los ficheros fuente es https://github.com/BorjaMoralejo/Efectos-CV-HPC.

5.2. Estructura de la Aplicación

Se han desarrollado diferentes herramientas para los diferentes casos de uso necesarios. Desde la recopilación de datos hasta la visualización hay un gran abanico de posibilidades. Mostrar los datos raw çrudos"puede ser útil en ciertas ocasiones, pero no siempre, y menos cuando se tratan con varias ejecuciones por experimento. En cada subsección se explicará brevemente el uso y la estructura de estas herramientas.

5.2.1. Perfilador

Esta aplicación es el núcleo del proyecto, la encargada de lanzar y recopilar los datos de ejecución de las aplicaciones.

En la sección de implementación se detalla el funcionamiento. Esta es su estructura general, compuesta por los siguientes ficheros:

- aux_func.c: Este fichero tiene las funciones auxiliares que se utilizan en profiler.c. Estas son preparar el comando para lanzar y lanzar los programas.
- profiler.c: Tiene la función principal que combina las utilidades del resto de ficheros fuente. Esta función es lo que realiza el perfilado.
- perfilador_main.c: Fichero main, se encarga de parsear los argumentos y de iniciar el perfilado.
- recorder.c: Se utiliza para guardar los datos obtenidos en el perfilado.
- counters.c: Gestiona los contadores de eventos, desde su inicialización hasta su cierre.
- comparation.c: Subtipo de perfilado que se utiliza para comparar dos tipos de aplicaciones. Se ejecutan las dos aplicaciones a la vez y se van anotando los tiempos de finalización de cada una hasta que se supere un tiempo límite.
- defines.h: Contiene variables fijas definidas, como el número eventos a medir o los contadores de energía.

5.2.2. Scripts de perf

Estos scripts se han utilizado para lanzar, obtener, procesar y guardar los datos obtenidos. Por razones que se explicarán más adelante, se ha tenido que desarrollar estos scripts para poder utilizar el comando perf de perf tools [1]. Estos scripts se pueden dividir en dos grupos, los lanzadores y los experimentos Los lanzadores son los que hacen de intermediario entre el comando perf y el resto de utilidades. En la sección de implementación se entra en detalle. Lanzadores:

- perf_launcher.sh: Este script se encarga de ejecutar el programa indicado en las CPU un número determinado de veces. Para ejecutarlo utiliza el comando taskset y perf, el primero para asignarle una afinidad y el segundo para obtener los datos de los contadores de eventos en su ejecución. Luego procesa los datos y los guarda en un formato. simul.sh: Se encarga de lanzar dos aplicaciones a la vez y se queda con los tiempos de ejecución de ámbos. perf_launcher2.sh: Utiliza simul para lanzar y obtener los datos de ejecuciones simultáneas. Al igual que perf_launcher.sh, realiza varias ejecuciones y procesa los datos de la misma forma.

Los experimentos se han preparado en un script para poder lanzarlos de forma sencilla y junto a las utilidades nohup [2] y disown[3]. Estos scripts tienen el formato de perf_exp(Número).sh. Dentro están las combinaciones o ejecuciones que se tiene que hacer y usan perf_launcher.sh o perf_launcher2.sh para ello.

5.2.3. Herramientas de procesado de datos

En este proyecto, no solo se necesita que la aplicación que recopile datos funcione y deje los datos en su formato original, se tiene que trabajar con ellos. Primero se formatean los datos, luego se escalan por si se realiza un análisis combinando diferentes escalas de contadores, sin recurrir a un segundo o varios ejes. Finalmente, los datos formateados se pueden utilizar para generar estadísticas, pero estas estadísticas son utilizadas con los tiempos de ejecución más que con los contadores.

Para ello, se utiliza este formateador, parser.c. Se le ha llamado parseador porque analiza el formato del fichero de salida del comando perf y lo formatea a otro tipo.

[1] perf.wiki.kernel.org
[2] linux.die.net/nohup
[3] Manual disown

Hay dos aplicaciones que escalan los datos, aunque más bien son mapeadores, los datos los transforma entre 0 y 1.

Uno, scaler.c, escala los datos formateados por parser.c y el otro ,scaler_medianos.c, escala todos los medianos de las diferentes aplicaciones entre ellos. Este segundo uso se utiliza para comparar los procesos entre sí de forma más sencilla. Por poner un ejemplo, es más facil distinguir que 0.7 es mucho que 738 page faults, con el resto de posibles números page faults sin escalar.

Finalmente se encuentran las utilidades que uitlizan para sacar los datos estadísticos. Estos son:

- estadisticas.c: Este programa obtiene las medias y desviaciones de los tiempos de ejecución.
- estadisticaEventos.c: Este programa obtiene las medias y desviaciones de los ficheros de datos de contadores procesados.
- mean_finder.sh: Este script devuelve el indice de la ejecución que más se acerca a la media. Se utiliza en perf_launcher.sh para obtener la ejecución mediana.

5.2.4. Visualizadores

Otra de las herramientas que se han desarrollado es un programa de C que convierte los datos seleccionados en un script para GNUPlot, este es visualizer.c.

Por otro lado, aux_visualizer.c combina las columnas de datos seleccionadas de distintos ficheros en un solo fichero de datos para poder comparar los datos en un gráfico.

5.3. Implementación

5.3.1. Perfilador

No se va a entrar en demasiado detalle dado que ha tenido problemas en la fase de ejecución en la máquina real aunque haya funcionado en el equipo de desarrollo, pero brevemente se explica como funciona. Estos problemas se explican en la sección 5.4

Para que llegue a poder ejecutarse requiere que el campo de */proc/sys/kernel/perf_event_paranoid* sea 0 o -1. Este campo permite medir los contadores por CPU en vez de por proceso. Esto es necesario puesto que se tienen que medir los eventos por CPU para ciertos experimentos.

perfilador_main.c

Este fichero como bien indica el nombre, contenía la función main de la aplicacicón. Su función principal era la de tratar los argumentos de entrada que recibía para ejecutar el programa pasado por la línea de comandos con la afinidad que se quisiera mediante el uso de taskset.

aux_func.c

Incluye funciones auxiliares que facilitan la legibilidad en otras funciones y ficheros. Dos de esas funciones auxiliares son para lanzar el programa, en estas se realiza el fork necesario para poder realizar el perfilado y una de ellas lo lanza en el núcleo indicado y la otra con la afinidad indicada. La diferencia es que uno traduce el número de núcleo a afinidad, y la otra simplemente aplica la afinidad proporcionada. Esta diferenciación existe para poder lanzar un programa multithreaded en ciertos núcleos.

Para poder lanzarlo en el núcleo indicado, se prepara el comando previamente en la otra función auxiliar llamada generate_command. Esta función procesa el comando que se le pasa a la función principal mediante los argumentos y lo procesa de forma que se ejecute con un execv y se le inserta un taskset para asignar la afinidad al proceso.

profiler.c

Aquí se encuentra el bucle principal del perfilado individual. Se ejecutan tantas veces como se desea, el caso más común serán 16 ejecuciones. Cada iteración, crea un fichero indicando la iteración que es y restablece los valores de las sumas de los contadores a 0. Luego lanza los programas e inicia los contadores y entra en el bucle principal de sondeo. Durante la ejecución del otro programa, obtiene los datos de los contadores cada periodo deseado. Para ello, primero resetea los contadores, después espera el periodo de tiempo y si el proceso sigue con vida, registra los datos de los contadores. Utiliza las funciones en record.c, contadores.c y aux_func.c para conseguir lo descrito.

Una vez finaliza el proceso, se detienen los contadores y se cierran los descriptores de ficheros y cierra el fichero en el que se guardan los datos. En caso de no ser la última iteración, se vuelve a empezar el proceso.

counters.c

Se encarga de controlar los contadores que miden los eventos. Este fichero tiene la configuración de los eventos que se van a registrar.

En cada iteración del perfilado se tienen que volver a iniciar los contadores asignándose al PID del proceso y es por eso que se realiza la inicialización y parada de procesos. Lo que ocurre es lo siguiente:

1. Se inicializan los contadores.
2. Se resetean y se espera el periodo del sondeo de perfilado.
3. Se obtienen los datos y se registran y vuelve al paso 2 hasta que se complete la ejecución del programa siendo perfilado.
4. Se cierran los contadores.

Esos pasos se realizan en cada ejecución diferente de los programas que se quieran analizar.

recorder.c

Este fichero se encarga de anotar los datos de la ejecución y del control de los descriptores de ficheros y memoria relacionada con la anotación.

Tiene que crear tantos descriptores de fichero como CPUs tiene el sistema porque perf event tools funciona a nivel de cpu, obtiene los contadores de cada CPU por separado.

comparation.c

Este fichero tiene el algoritmo que se encarga de la comparación, aunque un mejor nombre sería la ejecución simultanea. Se llama comparación porque lo que se busca con este algoritmo es encontrar la diferencia en eficiencia entre la ejecución en separado y combinado con otro programa.

Antes de lanzar un programa se empieza a medir los eventos. El sondeo de estos eventos, en vez de durar la ejecución de un solo programa, en este caso dura un tiempo mínimo para comparar el efecto de los programas en ejecución durante ese tiempo.

La idea de comparar es que se lancen durante un tiempo determinado los dos programas a comparar. Es normal que durante ese tiempo cualquiera de los dos finalice, en ese caso, se anota el tiempo de finalización del programa, se lanza otro proceso, se incrementa el número de ejecuciones totales que ha tenido ese programa y se continua con el sondeo. En el caso de que termine el periodo de sondeo, se terminan los procesos de los procesos.

5.3.2. Scripts de perf

Debido a un problema que ha ocurrido en la fase de obtención de datos con la aplicación desarrollada, la tarea de la obtención de datos de los contadores se le ha delegado a la aplicación perf [4]. Al ser un comando, se han desarrollado unos scripts que gestionan el lanzamiento de este y el procesado de los datos. Al comando perf se le pasan estos argumentos:

- -x: Indica que el separador utilizado para los datos es ":".
- -I 100 Indica que el intervalo para medir los contadores de eventos es de 100ms.
- -a Sirve para medir en todas las CPUs, -C X indica que se midan los contadores en la CPU X, siendo X 0 o 1.
- -e Los eventos que se quieren medir.

Se han desarrollado dos scripts principales para la obtención de datos. El principal, perf_expX.sh (donde X es el número del efecto a analizar), es el que ejecuta todos los experimentos que se quieran ejecutar, existen varias versiones de este, cada una para las diferentes pruebas.

El segundo, perf_launcher.sh es auxiliar del principal, este segundo se encarga de lanzar tantas pruebas como se le han indicado con la configuración deseada y generar los ficheros de datos procesados. Primero se va a explicar el auxiliar y después de conocer el principal, se comprenderá con mayor facilidad lo que hace el principal.

[4] perf.wiki.kernel.org

perf_launcher.sh

El script auxiliar se llama perf_launcher.sh, es un script de shell que como su nombre indica, lanza el perfilador y genera los ficheros con los datos tratados.

Para realizar esa segunda tarea utiliza otros scripts y funciones desarrolladas que se comentan a continuación. perf_launcher.sh recibe ciertos parámetros del script principal, esos son:

- El nombre del benchmark que se va a utilizar, para generar los ficheros.
- El nombre del experimento, para almacenarlos en una carpeta aparte.
- El número de repeticiones que se quieren hacer con el experimento para los datos estadísticos.
- El fichero temporal que contiene el comando a ejecutar.
- Los eventos que se quieren medir.

Los ficheros que se generan se guardan en el directorio procesado/*nombre_experimento*/CPU_X/. Los datos de los contadores se guardan con el nombre *benchmark_Y.dat*. Donde X es el índice de la CPU donde se ha ejecutado, *benchmark* es la aplicación ejecutada e *Y* es la repetición de la ejecución.

En cada repetición ocurre lo siguiente:

Como se ha mencionado, este script también delega su trabajo a otros scripts y programas auxiliares. Estos son parser.c y scaler.c.

Parser se encarga de dar formato a los datos que genera el comando perf para que puedan ser utilizados cómodamente para gnuplot u otras herramientas. Lo que hace exactamente es agrupar los datos en filas en base al tiempo en el que se han obtenido. El comando perf con la configuración especificada genera los ficheros de datos de esta forma:

`tiempo1:n1::tipo1-evento:`*resto*
`tiempo1:n2::tipo2-evento:`*resto*
`tiempo1:n3::tipo3-evento:`*resto*
`tiempo2:n4::tipo1-evento:`*resto*
`tiempo2:n5::tipo2-evento:`*resto*

`tiempo2:n6::tipo3-evento:`*resto*

`...`

parser.c realiza lo siguiente con esos datos, primero añade un header con los nombres de los eventos y si se pide, puede generar el IPC (instructions per cycle) del fragmento de tiempo. Para ello la segunda y tercera columna tienen que ser instrucciones y ciclos, respectivamente. Los otros eventos medidos los asigna a una columna y el *resto* lo ignora, el formato acabaría siendo :

```
tiempo:tipo1-evento:tipo2-evento:tipo3-evento(:ipc)
tiempo1:n1:n2:n3(:ipc1)
tiempo2:n4:n5:n6(:ipc2)
...
```

Por otro lado, scaler.c, escala los datos entre 0 y 1, basándose en el máximo de cada columna. Los datos de entrada son los datos procesados por parser.c y los escala por columna teniendo en cuenta el máximo de cada uno. Esto se ha hecho para que se vea mejor gráficamente pero no significan nada numéricamente entre ellos, dado que los valores estarán entre 0 y 1, escalados con diferentes números máximos.

Tras finalizar las repeticiones, se generan los ficheros *benchmark_tiempos.dat* y *stats_benchmark.dat*, en el primero se guardan los tiempos de ejecución que se han ido registrando en cada iteración y en el segundo los valores de la media y desviación típica de los tiempos de ejecución. El último fichero que se genera, aunque estaría mejor dicho que se copia, es el fichero mediano de las ejecuciones del benchmark. Este se llama *mediano_benchmark.dat*. Estos ficheros se generan mediante los programas estadisticas.c y mean_finder.sh.

La razón por la que estas funciones auxiliares están separadas de perf_launcher.sh es para facilitar la legibilidad del código y dividir el problema en problemas más pequeños.

perf_exp

El script principal tiene muchas variaciones, una variación por tipo de experimento, pero todos realizan lo mismo, solo que cambian los argumentos con los que trabajan y la versión del launcher que utilizan, ya sea perf_launcher.sh o perf_launcher2.sh.

- comentar que mas hay dentro de cada uno - nombre del exprimento - directorio de parsec - nombre y comando del benchmark, indicando su tipo (serial o multihilo), indicando

tamaño del input (nativo o simlarge) - algunos experimentos requieren de ficheros de script temporales para llegar a ejecutar los comandos. - el numero de ejecuciones

- forma en la que se han lanzado (usando nohup y disown)

5.3.3. Herramientas de procesado de datos

Las herramientas de procesado de datos parser.c y scaler.c se han explicado en la Subsección 5.3.2 al necesitar el conexto para llegar a entender la funcionalidad de perf_launcher.sh. Es por eso que no se va volver a explicar. En esta subsección se va a explicar lo que hacen estadisticas.c, estadisticasEventos.c y mean_finder.sh.

Los programas que se encargan de generar los datos estadísticos son estadisticas.c y estadisticasEventos.c. Estos dos generan las medias y desviaciones típicas de los datos que se les pasan. Al primer programa, se les pasan los ficheros de tiempo que se generan con perf_launcher.sh y genera los ficheros ya mencionados de *stat_benchmark.dat*, que contienen la media y desviación típica de los tiempos de ejecución obtenidos. El segundo programa calcula las medias y desviaciones típicas de cada evento en el fichero de entrada. La salida se muestra por la salida estándar y son dos filas, la primera muestra las medias por columnas y la segunda fila tiene las desviaciones estándar.

El script mean_finder.sh, lee el fichero de estadísticas y obtiene la media. Basandose en esa media, busca el tiempo de ejecución mediano y devuelve el número de la iteración al que pertenece. Ese número luego lo gestiona perf_launcher.sh y genera *mediano_benchmark.dat*.

5.3.4. Visualizadores

visualizer.c es programa auxiliar que se ha realizado en c y utiliza GNUPlot para mostrar gráficos de los datos obtenidos. De entrada toma el nombre del fichero de datos y se ha configurado para que tome los títulos y subtítulos de los gráficos desde los argumentos a la llamada de visualizer y luego las etiquetas de los datos los toma de los ficheros de datos procesados, la primera línea de estos es la cabecera y etiquetas de los datos. Este programa detecta automáticamente el número de las diferentes variables que se han medido en los datos.

El otro programa auxiliar, aux_visualizer.c, genera un fichero con la columna que se le

indica de los diferentes ficheros que se le pasan como argumento. Se utiliza para analizar y contrastar los datos de diferentes ejecuciones.

5.4. Problemas encontrados

Se ha encontrado un problema que ha desviado un poco el desarrollo del proyecto. El problema, en resumen, fue que no funcionaba en la máquina en la que se tenía que ejecutar. Primero por incompatibilidad por la arquitectura de la CPU y el segundo se desconoce la causa, simplemente no funcionaba en esa máquina. En cualquier caso, este problema tuvo un impacto leve por la gestión de riesgos que se planificó.

El primer problema que fue que la funcionalidad de obtener el consumo energético no es soportado en la arquitectura del nodo. Se ha intentado quitar la parte de consumo energético pero seguía dando problemas, así que se ha tenido que hacer una pequeña modificación a la implementación.

El segundo problema, que es el más grave, es que la herramienta desarrollada no funciona en el nodo. Por alguna razón, al obtener los datos de los contadores de eventos estos devuelven 0 en ciertas ocasiones. Se ha probado en equipos personales y en otro tipo de nodo y funcionaba. Para evitar perder tiempo. Se ha intentado solucionar pero no se ha conseguido, y es por eso que se ha decidido dejar de lado el programa desarrollado y utilizar el comando perf.

6. CAPÍTULO

Análisis de los resultados obtenidos

En este capitulo se explica como se ha realizado el análisis de los datos obtenidos en las ejecuciones de los experimentos, y en qué consisten esos datos recopilados.

Los datos obtenidos de las diferentes ejecuciones de los scripts tienen un formato en concreto. Este formato viene en dos formas, los eventos y los tiempos de ejecución. La sección de como se llegan a obtener y procesar estos datos están descritas en la Subsección 5.3.2.

Los eventos se guardan en diferentes ficheros en base al tipo de experimento y su iteración. Dentro de los ficheros, primero está la cabecera que indica los eventos recogidos en esa ejecución. Luego, las siguientes filas indican un momento en el tiempo diferente, donde el primer elemento separado por ':' es el timestamp en el que se obtuvo el dato. Los siguientes elementos de la fila equivalen al dato indicado en la cabecera de su columna. Existen dos tipos de ficheros de datos de eventos, los .dat y .ndat. Los .dat son los datos con los valores literales, los .ndat están escalados o mapeados (entre 0 y 1) en base a los valores máximos obtenidos en todas las ejecuciones de un programa de benchmark en la CPU ejecutada.

Para los tiempos de ejecución se ha realizado de dos formas diferentes. Por un lado, se encuentran las ejecuciones en solitario, donde solo se guardan los tiempos de ejecución, cada fila es una ejecución. Tras finalizar el número de ejecuciones que se haya indicado, se calcula la media y desviación estándar de los tiempos obtenidos. Por otro lado, existen los ficheros de ejecuciones en conjunto. Estos contienen los tiempos de finalización del primer programa y segundo programa separado por una coma. Esta segunda forma de

obtener los datos se utiliza principalmente para analizar los efectos que requieran ejecuciones simultáneas de las aplicaciones de benchmark.

De estos datos se extraen dos datos estadísticos, la media y desviación estándar. El método empleado para verificar y luego cuantificar la diferencia entre las medias y desviaciones estándares entre diferentes técnicas de aumento de rendimiento ha sido la hipótesis nula con p-value. Esta es una técnica estadística utilizada para comparar las medias de dos grupos o muestras independientes y determinar si existen diferencias significativas entre ellas. Al ser utilizada ampliamente en diversas áreas de la investigación y en el análisis de datos, se ha considerado correcta la inclusión de esta técnica.

6.1. Grado de paralelización

Los programas de benchmark del paquete de aplicaciones de PARSEC vienen con la opción de ser compilados y ejecutados de forma serial o paralelizada. Aunque las aplicaciones no se hayan desarrollado por nosotros, es interesante conocer el factor de aceleración o speedup y la eficiencia con la que se consiguen paralelizar. Se han ejecutado las aplicaciones por separado y en la Tabla 6.1 son los tiempos obtenidos en la CPU 0.

La eficiencia es decente en todas las aplicaciones menos en streamcluster. Se sospecha que se debe a su naturaleza de IO intensive y que los diferentes procesadores están compitiendo por el acceso a la memoria. También puede ser que tenga una baja fracción paralelizable, posiblemente siendo esta la fase de stream de los datos a procesar.

Tabla 6.1: Eficiencia de la paralelización

	serie	4 hilos	factor de aceleración	eficiencia
blackscholes	203.90	70.30	2.901	72.51 %
bodytrack	760.63	235.24	3.233	80.84 %
freqmine	1975.09	542.13	3.643	91.08 %
fluidanimate	476.55	126.15	3.778	94.44 %
streamcluster	479.47	364.24	1.316	**32.91** %

6.2. Analizando efecto del perfilador

En este experimento se han lanzado los diferentes programas con el perfilador midiendo eventos y los programas por separado sin el perfilador. En ambos casos se ha medido el tiempo de ejecución y se han calculado la media y desviación estándar. Los tiempos

de ejecución se han obtenido gracias al output que proporciona la propia aplicación de parsec, que utiliza el comando time.

Al ejecutar los programas con el perfilador, aunque no era el objetivo principal, se han medido los eventos descritos en el apartado de metodología comun. Esos mismos datos se han utilizado para determinar la naturaleza de las aplicaciones, aunque esto se desarrolla más en la siguiente subsección.

Las ejecuciones se han realizado empleando cuatro hilos en la paralelización. Además, los tiempos de ejecución de los programas con perfilador se van a utilizar como tiempo de control para las diferentes técnicas utilizadas.

A continuación se muestran dos tablas con las medias (Tabla 6.2) y desviaciones estándares (Tabla 6.3) obtenidas agrupados por benchmark. La N indica que es de las ejecuciones que no utilizaron contadores. A simple vista no parece que haya una diferencia en ejecutar con y sin perfilador.

Tabla 6.2: Medias de los tiempos de ejecución, en segundos.

	CPU 0	CPU 0 N	D. porcentual	CPU 1	CPU 1 N	D. porcentual
blackscholes	70.39	70.30	-0.14 %	69.46	69.47	0.01 %
bodytrack	235.27	235.24	-0.01 %	234.95	235.19	0.10 %
freqmine	126.28	126.15	-0.10 %	126.03	125.81	-0.17 %
fluidanimate	543.02	542.13	-0.17 %	540.49	540.71	0.04 %
streamcluster	365.33	364.24	-0.30 %	359.94	359.73	-0.06 %

De primera vista, en la tabla de medias (Tabla 6.2) se puede observar que hay indicios de un posible efecto negativo en la CPU 0 fijándose en la columna de diferencia porcentual. Es una diferencia muy pequeña y no podemos estar seguros que sea un impacto significativo solo con estos datos.

Tabla 6.3: Desviaciones estándar de los tiempos de ejecución, en segundos.

	CPU 0	CPU 0 N	D. porcentual	CPU 1	CPU 1 N	D. porcentual
blackscholes	0.14	0.11	-25.06 %	0.11	0.12	14.38 %
bodytrack	0.16	0.14	-13.05 %	0.18	0.58	103.80 %
freqmine	0.22	0.28	26.69 %	0.25	0.33	29.33 %
fluidanimate	1.67	1.68	0.57 %	0.87	1.02	16.33 %
streamcluster	0.47	0.41	-12.68 %	0.55	0.16	-112.14 %

Observando la tabla de desviaciones (Tabla 6.3), en las columnas de diferencia porcentual, se puede señalar que no hay claro indicativo de que haya un efecto negativo en el rendimiento, si la mayoría fuesen valoren muy negativos, saltaría a primera vista de que hay un efecto negativo.

Pero para estar seguros, se va a realizar un test de hipótesis usando la prueba T de student asumiendo que es una distribución normal.

H_0: No hay diferencia significativa entre las ejecuciones con y sin perfilador.

En este caso, la hipótesis nula asume que la presencia del perfilador no tiene un impacto significativo en los resultados de las ejecuciones. Se van a utilizar las medias de las ejecuciones y un valor crítico del 5 %.

Primero, calculamos los p valores.

Tabla 6.4: P valores de las medias de las ejecuciones con y sin perfilador.

pvalues	CPU0	CPU1
blackscholes	**0.025**	0.846
bodytrack	0.593	0.124
freqmine	0.099	0.052
fluidanimate	0.090	0.546
streamcluster	**0.000**	0.189

Para empezar, vemos dos valores que descartan la hipótesis nula, indicando que hay una diferencia significativa, pero no muestra cuanto, eso se analizará a continuación. Cabe resaltar que ocurre en la CPU 0, eso podemos suponer que es debido a que el proceso del perfilador se ejecuta principalmente en la CPU 0.

Fijándonos en el benchmark de blackscholes un programa CPU intensive, desde un principio parece no tener sentido que se vea afectado, pero este benchmark trabaja con una base de datos de finanzas, y es en la fase de carga de esta que se notan los efectos. Las bases de esa suposición vienen del otro benchmark afectado, streamcluster, que es un programa que necesita leer constantemente datos de memoria sea afectado con un casi un segundo en la media.

En conclusión, el efecto en ciertas aplicaciones es inexistente y según en que CPU se encuentra afecta al rendimiento de los programas que leen de memoria. Teniendo en cuenta que hay un benchmark que durante toda su ejecución lee de memoria y el efecto es menor a un segundo, se puede tener en consideración. Por lo tanto, se tendrán en cuenta blackscholes y streamcluster a lo largo del análisis, con un margen del 0.1s y casi un segundo de diferencia.

6.3. Analizando la naturaleza de las aplicaciones

En este experimento lo que se espera encontrar son características en los contadores medidos que puedan ayudar a determinar la naturaleza de las aplicaciones. Un scheduler que busca sacar provecho de la consolidación requiere saber el tipo de la aplicación para planificar y colocarlo junto a otra aplicación válida.

La razón que hay detrás de este experimento es encontrar la relación entre los datos de los contadores de la ejecución indicadores y el tipo de la aplicación. Teniendo eso como base se podría entrenar una inteligencia artificial para que clasifique las diferentes aplicaciones que hay en ejecución en tiempo real, y migrarlas a otra CPU. Hay diversos artículos que realizan esto hasta cierto nivel, como Mishra et al. (2017) , Hou et al. (2019) y Dwyer et al. (2012).

La otra razón por la que nos interesa saber la naturaleza de la aplicación es más sencilla. Un técnico experto de las instalaciones de un HPC podría clasificar el tipo de una aplicación nueva basándose en unos rasgos y colocarla en el nodo o planificación adecuada para aumentar el rendimiento general.

Para determinar la naturaleza, primero se utilizarán los datos obtenidos en las ejecuciones medianas de la prueba anterior. Realizar una media de todas las ejecuciones y usar esos datos para el análisis hubiera sido incorrecto, las ejecuciones aunque sean del mismo programa y pasan por las mismas fases, estas fases ocurren con algo de ruido y puede ocurrir que la carga de datos se retrase un poco.

6.3.1. Observaciones generales

Primero se va a realizar un breve análisis que compara las ejecuciones entre los diferentes tipos de aplicaciones, si no se llegan a distinguir a primera vista, el resto del trabajo será más difícil.

Se han procesado los datos obtenidos de las ejecuciones en la CPU 1, que es el procesador que le añade el menor posible ruido a los datos por el perfilador. Se han calculado las siguientes variables para digerir mejor el análisis. De los 15 eventos que se miden, se han creado estas variables y se han determinado como los mejores indicadores:

- scpi(stalled cycles per instruction): Cantidad de ciclos estancados por instrucción, mientras más alto, peor.

- pagefault: Cantidad de pagefaults que han ocurrido durante la ejecución. Están mapeadas respecto la mayor cantidad de ocurrencias.
- mLLC-ref(mapped LLC references): La cantidad total de referencias, tanto de lectura como escritura, de la caché de último nivel. Están mapeadas respecto a la mayor cantidad de referencias.
- %LLC-L: Proporción de las referencias de la caché de último nivel que son de lectura.
- %LLC-LM: Proporción de fallos de lectura de la caché de último nivel.
- %LLC-SM: Proporción de fallos de escritura en la caché de último nivel.
- %L1-L: Proporción de las referencias de la caché de primer nivel que son de lectura.
- %L1-LM: Proporción de fallos de lectura de la caché de primer nivel.
- %L1-SM: Proporción de fallos de escritura en la caché de primer nivel.

Tabla 6.5: Nuevas variables obtenidas desde las medias de los eventos

	ipc	scpi	pagefault	LLC-ref	%LLC-L	%LLC-LM	%LLC-SM	%L1-L	%L1-LM	%L1-SM
blackscholes	0.73	0.291	18.21	80816	52.50%	67.69%	12.03%	66.10%	0.26%	0.07%
bodytrack	1.70	0.035	38.17	287867	79.98%	1.34%	5.07%	63.20%	0.50%	0.05%
freqmine	1.60	0.096	52.22	351154	36.47%	27.61%	8.15%	66.52%	0.76%	0.36%
fluidanimate	1.40	0.067	5.33	151135	53.39%	61.66%	25.71%	69.92%	0.16%	0.04%
streamcluster	1.59	0.097	48.77	378263	37.16%	26.35%	7.35%	66.78%	0.82%	0.39%

En la Tabla 6.5 las variables de referencias totales a la caché de último nivel (LLC-ref) y los page faults no están escalados. Esas son las medias de toda la ejecución mediana. Lo que quiere decir es que, de media por sondeo de la aplicación (cada 0.1s), han ocurrido 80816 referencias a LLC en la aplicación blackscholes. De la misma forma para pagefaults. El resto de proporciones se sacan del momento del sondeo.

La razón por la que se ha escogido usar la media y acabar con eventos contados por sondeo, en lugar de sumar todos los contadores es sencilla. Para una aplicación que clasifica aplicaciones, es mejor clasificar por sondeo que esperar a que se cumplan ciertos criterios.

De los datos anteriores, se han realizado unos spidergraph de cada aplicación respecto al promedio de los datos, son las subfiguras que se encuentran en la Figura 6.1. El uso principal que se le da a estos gráficos es para representar gráficamente los diferentes tipos de aplicaciones y se espera que a primera vista se puedan discernir lo diferentes que son.

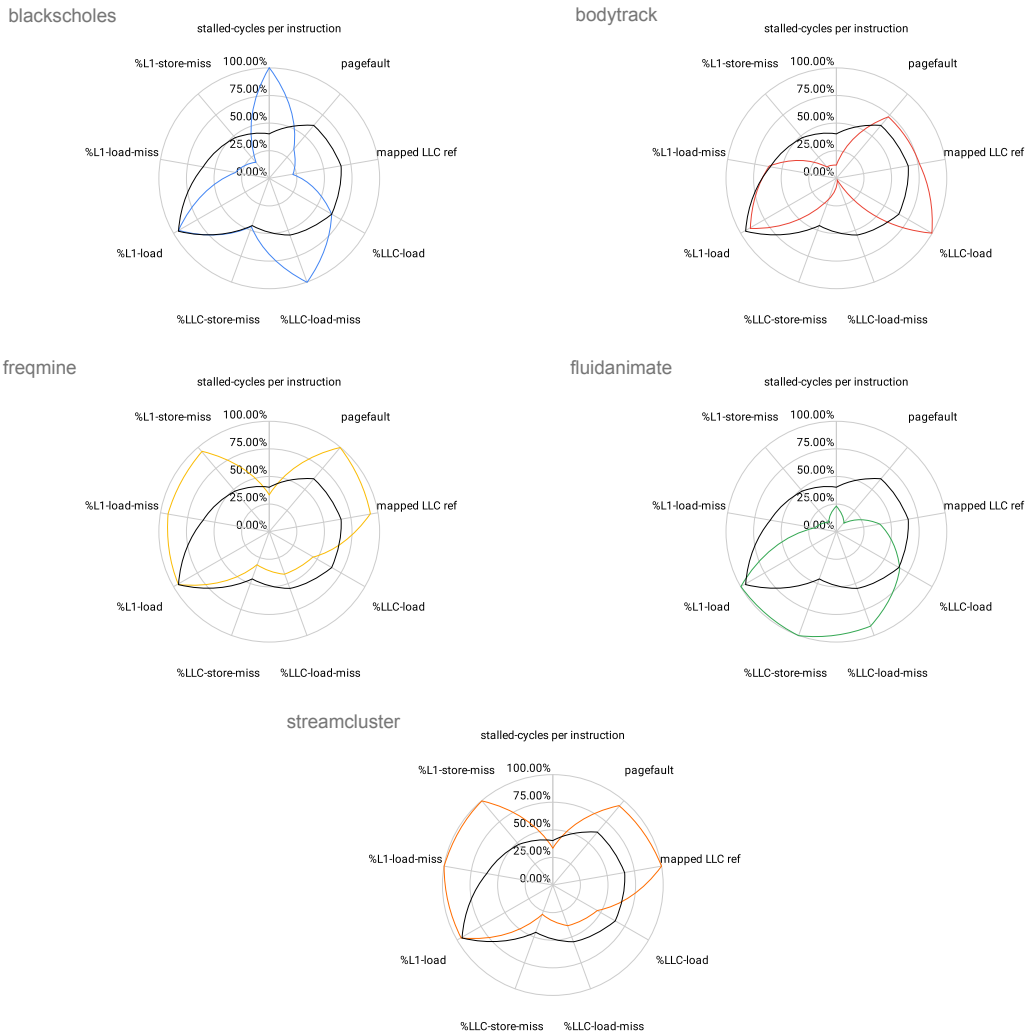


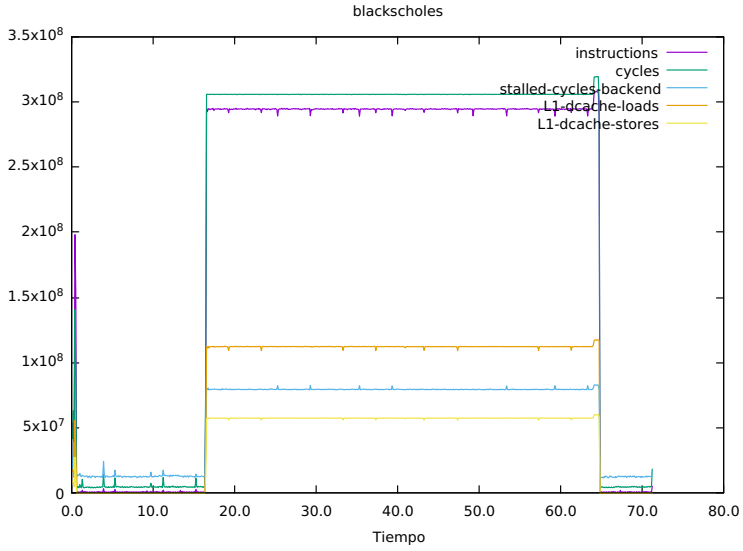
Figura 6.1: Spidergraphs de las distintas aplicaciones

Junto a las características de las aplicaciones, se ha incluido la media de las métricas para añadir contexto.

6.3.2. Observaciones de las aplicaciones

En cada subsección se hará un breve enfoque a los datos de las ejecuciones obtenidas. En cada una de ellas puede que cambie los eventos que se vayan a mostrar y lo más probable es que estén escalados entre 0 y 1. Como aclaración, se ha realizado ese mapeo porque la diferencia de escala numérica entre los diferentes eventos es demasiado grande, y se necesitarían varios ejes para poder representarlo. Se ha considerado apto puesto que analizar los gráficos minuciosamente es poco eficiente.

Figura 6.2: Gráfico de los eventos de la ejecución de blackscholes



blackscholes

Al analizar el spidergraph de la Figura 6.1, hay que destacar que la aplicación presenta un alto número de stalled cycles, el cual puede ser el origen del rendimiento pobre. En la Figura 6.2 se pueden diferenciar tres fases en la aplicación, las fases 1 y 3 son las causantes de esté efecto.

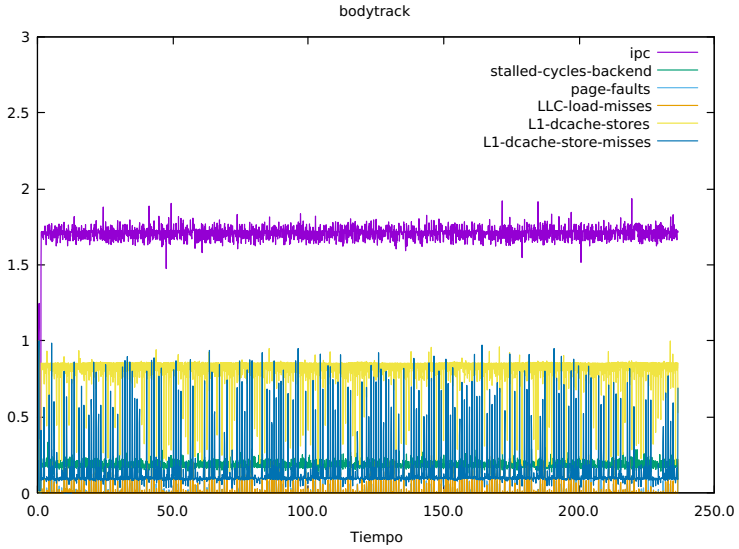
Además, se identifica que hay pocas referencias a la caché de último nivel, y cuando ocurren, se producen fallos en la caché. A pesar de que la aplicación se clasifica como CPU intensive, se evidencia un bajo rendimiento debido a un posible cuello de botella en algún punto del procesador. Sería necesario investigar más a fondo para identificar el origen de este cuello de botella en un entorno de producción.

bodytrack

Esta aplicación tiene una ejecución homogénea a lo largo de la mayoría de su ejecución, como se puede llegar a apreciar en la Figura 6.3. Puede que el principio y el final sean algo diferentes para preparar la paralelización, pero no se llega a notar.

Tiene más pages faults que el promedio, pero no parece afectar al rendimiento y a pesar de esto, no tiene tantos ciclos de espera. Tiene muy buena localidad de datos como se puede ver por los pocos fallos en la caché. Es muy buen ejemplo de aplicación de naturaleza

Figura 6.3: Gráfico de los eventos de la ejecución de bodytrack



CPU intensive. Cabe destacar que tiene más cambios de contexto y migraciones que el resto, un 200 % más y eso seguramente sea a su implementación de la paralelización.

Si se llega a hacer algo de zoom a la ejecución, como se puede ver en la Figura 6.4, no se llegan a detectar patrones.

freqmine

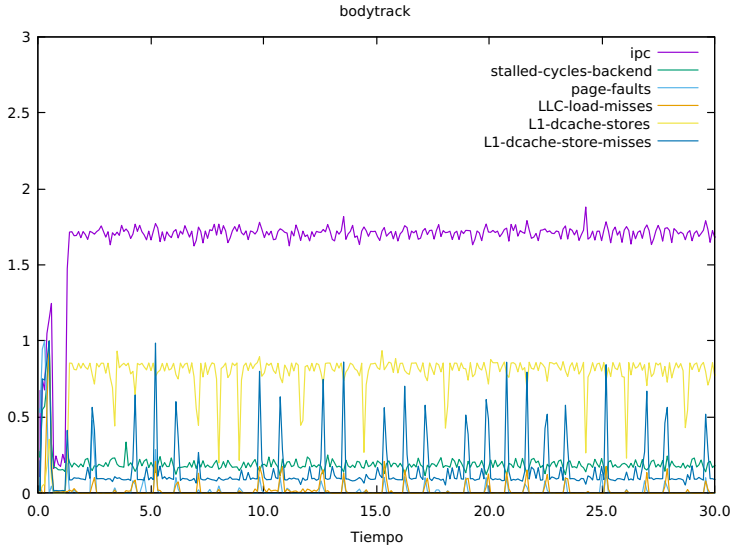
Estudiando la Figura 6.1 se llega a observar que la aplicación muestra un alto promedio de pagefaults, lo que puede indicar que se están produciendo con frecuencia intercambios de datos entre la memoria principal y la memoria secundaria. Además, se observa un uso intensivo de la caché. La tasa de aciertos de la caché de último nivel está cerca del promedio pero la del primer nivel tiene fallos tanto en la lectura como escritura.

La aplicación muestra características de ser caché e IO intensive, ya que se producen pagefaults con frecuencia y depende de la caché para acceder a los datos. Además, se observa un desempeño deficiente en la caché L1, lo que podría afectar su rendimiento general.

Al observar el gráfico de la ejecución de la Figura 6.5, se observa una clara repetición en la ejecución. Este patrón se puede ver con más detalle en la Figura 6.6 y hay ciertas cosas que comentar.

Lo que muestra es la interacción entre los niveles de las caches del procesador. El gráfico

Figura 6.4: Gráfico acercado de bodytrack



tiene una forma aserrada orientada hacia la izquierda porque al principio intenta acceder a un dato que no tiene en la caché del primer nivel, y esta tiene que buscar los datos en los niveles superiores. Eso crea un primer pico de accesos de lectura a la cache de último nivel (LLC-loads). Mientras se va llenando la caché de primer nivel, más aciertos tiene y por lo tanto, pasa menos tiempo esperando a que se transfieran los datos de la caché de nivel superior a la de inferior. En ese tiempo que no está esperando, la CPU consigue realizar más accesos a la caché. Mientras más acierta en los niveles inferiores, requiere menos accesos a los niveles superiores de la caché, y esto continua hasta que, por espacio en la memoria caché o por un acceso en memoria no contiguo, requiere datos que no tiene en la caché y vuelve a repetir lo mismo.

fluidanimate

Fluidanimate es un claro ejemplo de un programa memory intensive, gran uso de CPU y con muchos accesos a memoria. Analizando el gráfico de la Figura 6.1, se puede observar que tiene buena tasa de acierto, tanto para load o store, en la caché de primer nivel. En cuanto a la caché de último nivel, tiene la mayor tasa de fallos y esto es un indicativo de que usa mucha memoria.

En la Figura 6.7 se puede ver que la ejecución es homogénea, sin fases facilmente diferenciables.

Figura 6.5: Gráfico de los eventos de la ejecución de freqmine

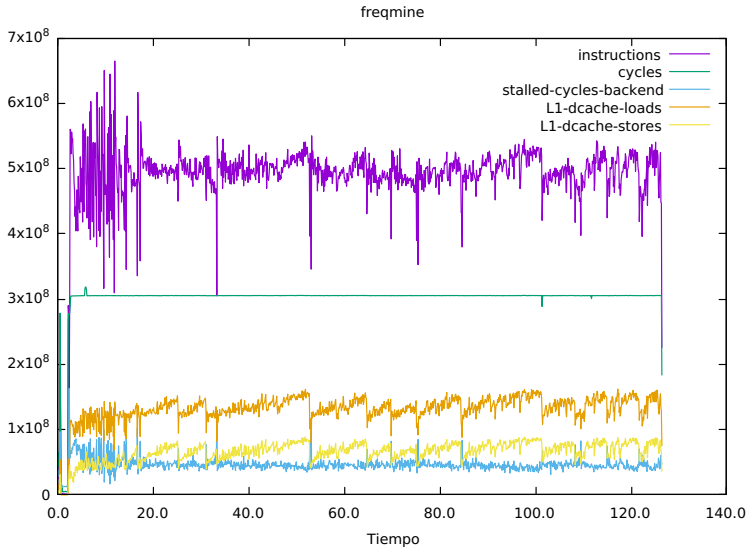


Figura 6.6: Gráfico enfocado en los patrones de freqmine

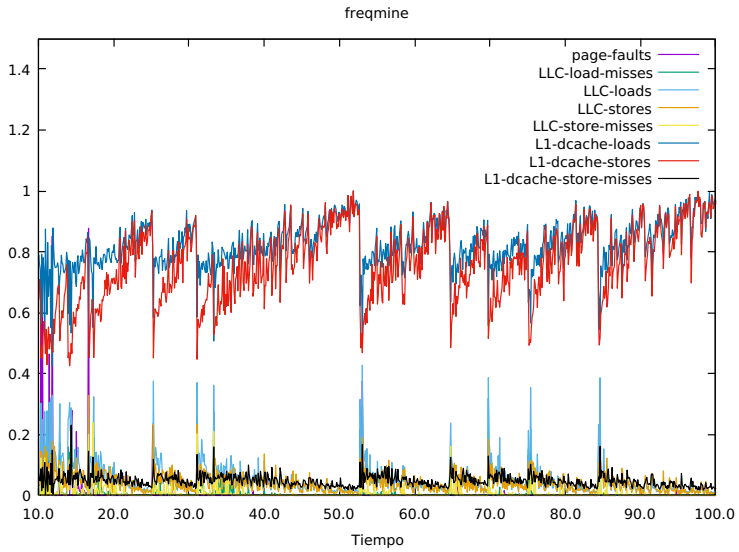
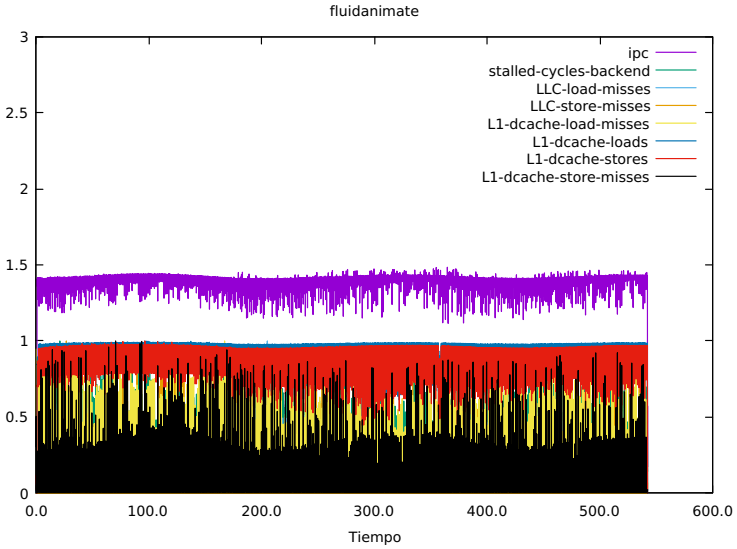


Figura 6.7: Gráfico enfocado en los patrones de fluidanimate



streamcluster

Estos datos de ejecución del benchmark streamcluster se han procesado y representado en el spidergraph de la Figura 6.1. Esa forma de gráfico ya se ha visto en el experimento, el benchmark freqmine ha generado uno muy parecido.

Aunque ya se sabía que era IO intensive, no se esperaba obtener un resultado tan parecido. Por un lado, freqmine trabaja con un frequent pattern tree y por otro lado, streamcluster ejecuta un algoritmo de clusterización que se basa en las distancias euclídeas.

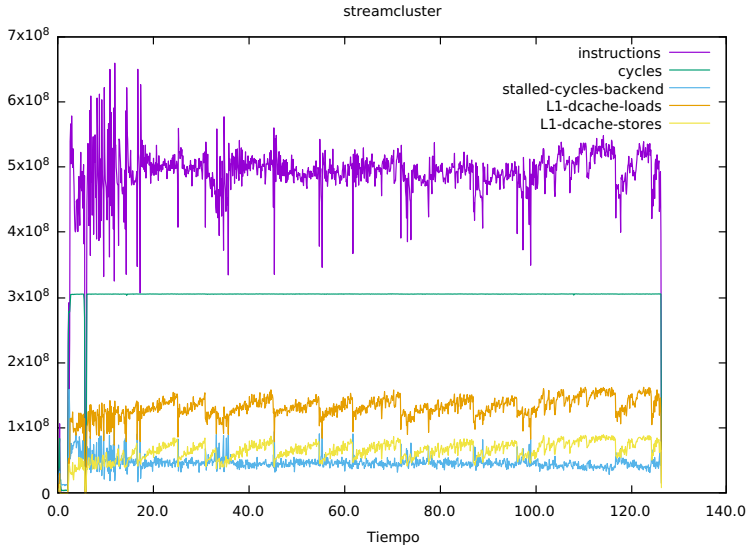
El gráfico de las ejecución de la Figura 6.8 tiene los mismos efectos descritos en la sección de freqmine.

6.3.3. Resumen

Los resultados de los experimentos han sido muy positivos, se puede observar que tienen diferentes perfiles de ejecución y claros indicadores que muestran su naturaleza. Con los suficientes datos recopilados, se podría estimar la naturaleza comparándolo con aplicaciones ya clasificadas.

Además, no solo se puede extraer la naturaleza de la aplicación en general, en la Tabla 6.2 se podía diferenciar 3 fases en la ejecución de blackscholes. Con un scheduler inteli-

Figura 6.8: Gráfico enfocado en los patrones de streamcluster



gente se podría llegar a lanzar y combinar los programas teniendo en cuenta las fases de ejecución.

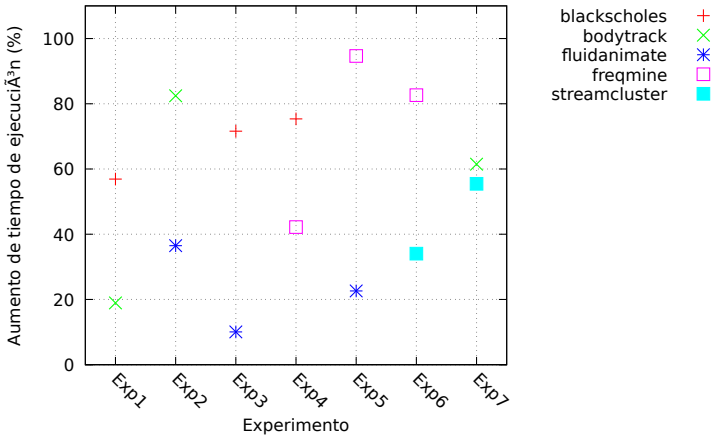
6.4. Analizando efecto consolidación

En esta sección se van a analizar los efectos de la consolidación. Se han ejecutado en una sola CPU, en la CPU 1, porque no era necesario realizar este experimento en las dos CPUs. Además, como se ha observado en la Sección 6.2, no hay efectos notorios en la segunda CPU y de esta forma facilita el análisis al no tener que tratar con correcciones por el ruido. Se han utilizado los primeros núcleos de la CPU y en cada experimento se han obtenido el los tiempos de finalización de cada programa junto a sus datos de la ejecución. Se han estudiado dos enfoques distintos en los datos, en el primero se han comparando los tiempos de ejecución de las aplicaciones con y sin consolidación, obteniendo el rendimiento general. En el segundo, se han examinado los datos de los contadores que se han generado cuando se sobreponen las ejecuciones de los dos programas.

6.4.1. Observaciones generales

Primero se ha analizado el efecto general que ha producido la consolidación respecto a los tiempos de ejecución. Al realizar este experimento, se han tenido que emparejar las

Figura 6.9: Aumento de tiempo de ejecución de las aplicaciones respecto a tiempo de control



aplicaciones de diferentes formas para medir diferentes efectos en la consolidación, como se indican los motivos en la Subsección 4.3.4. En la Figura 6.9 se muestran los efectos de los tiempos de ejecución afectadas por la consolidación agrupada por experimento. Como se puede ver, los tiempos respecto a las ejecuciones individuales han aumentado. En la Tabla 6.6 se muestran las medias y desviaciones de los tiempos de ejecución por benchmark. El benchmark fluidanimate ha sido el menos afectado por ser ejecutado junto a otro programa, con un mínimo de 10 % de incremento y freqmine ha sido el más afectado con un máximo de casi un 95 %. Se puede notar que, en general, los tiempos de ejecución individuales son peores al ser ejecutados de forma consolidada, pero eso no es un indicador de mala o buena consolidación. Ese indicador es el tiempo de ejecución de los programas de forma consolidada.

Tabla 6.6: Media y desviación estándar del rendimiento agrupado por benchmark

	media	desv. estandar
blackscholes	67.94 %	9.76 %
bodytrack	54.28 %	32.36 %
fluidanimate	23.05 %	13.23 %
freqmine	73.15 %	27.48 %
streamcluster	44.74 %	15.14 %

Como se puede ver en la Tabla 6.7, hay un aumento de rendimiento de casi un 9 % en el sistema por usar consolidación. A continuación se entra en detalle de qué son esas las columnas de la tabla:

- T Suma es la suma de los tiempos de ejecución individuales de los benchmark del experimento.

- T Consolidación es el tiempo que ha necesitado el experimento en ejecutarse, y por lo tanto, en terminar de ejecutarse los dos programas.
- "porcentaje de cambio.[es] el aumento de velocidad que han tenido el experimento en cuestión respecto al tiempo T Suma.

Tabla 6.7: Resultados de la reducción de tiempo de ejecución de los experimentos

	T Suma	T Consolidado	Porcentaje de cambio
Experimento 1	305.53	279.773	8.43 %
Experimento 2	777.37	740.012	4.81 %
Experimento 3	612.42	596.582	2.59 %
Experimento 4	196.45	179.363	8.70 %
Experimento 5	668.28	664.746	0.53 %
Experimento 6	490.40	488.212	0.45 %
Experimento 7	599.48	566.214	5.55 %

Cabe destacar que todos los experimentos han dado aumentos de velocidad positivos, pero claramente unos son mayores que otros. Los Experimentos 1 y 4 se han ejecutado más de un 8 % más rápido y esto puede ser debido al rendimiento pobre que tenía blackscholes lanzado individualmente. Los tiempos de espera por la carga de los ficheros de datos de blackscholes ha podido ser utilizados en la ejecución de los otros programas.

De la misma manera, los Experimentos 2,3 y 7 también han obtenido un buen incremento, lo que los diferencia es su largo tiempo de ejecución y que en los experimentos había cierto grado de competencia por la caché y CPU (Tabla 6.8), aunque como se puede ver, no ha sido suficiente como para impactar negativamente al tiempo consolidado.

Por otro lado, los Experimentos 5 y 6 no han tenido tan buenos resultados, pero siguen siendo positivos. El mayor indicador de la falta de incremento de rendimiento es que los programas IO intensive no combinan bien entre ellos y además, si uno de los programas con los que se combina es un memory intensive (Tabla 6.8), tampoco va a obtener buenos resultados.

En resumen, se ha conseguido demostrar que la consolidación funciona incluso con el scheduler de Linux. Es posible que su máximo potencial se haya visto reducido por algún que otro factor, como bien puede ser el sobrecoste de las migraciones y cambios de contexto. Queda como investigación futura encontrar el balance entre los cambios de contexto y los beneficios de la consolidación.

Tabla 6.8: Recursos compartidos por los experimentos

	Exp1	Exp2	Exp3	Exp4	Exp5	Exp6	Exp7
IO				X	X	XX	X
CPU	XX	X	X	X		+	X+
Caché	+	X+	X		X		+
Memoria		X	X		X		

6.4.2. Fracciones de la ejecución consolidadas

Esta subsección tiene un enfoque más centrado en los contadores de los eventos y los benchmarks. En cada experimento, se lanzan dos programas a la vez y se anota el tiempo cuando alguno termina de ejecutar, pero el experimento no acaba hasta que ambos terminan (y se apunta el tiempo de finalización correspondiente al último programa). Es por eso que en la ejecución hay dos secciones notorias, una en la que están los dos programas en ejecución y otra en la que hay un solo programa en ejecución. Queremos enfocar el análisis en la primera parte, cuando están consolidando. Para conseguir esa parte simplemente se tiene que obtener los primeros X segundos, donde X es el menor tiempo de ejecución entre los dos programas. De esta forma, y con la ayuda del comando `head` de Linux, se pueden obtener esos primeros segundos. Con una selección de las columnas de datos `instruciones, ciclos, cilos estancados` y `accesos de escritura y lectura de la caché de primer nivel` nos quedamos con los perfiles de ejecución de la Figura 6.10.

Tabla 6.9: Diferencias de las migraciones y cambios de contexto

	cs	cs incremento %	migrations	migrations incr %
blackscholes	7.61	244.20 %	0.43	2188.90 %
bodytrack	-2.98	-18.26 %	-1.31	-64.37 %
freqmine	7.31	269.50 %	0.11	1564.32 %
fluidanimate	5.60	162.03 %	0.08	153.08 %
streamcluster	19.40	727.66 %	1.00	63345.74 %

Como se puede observar, hay una gran cantidad de irregularidades en los gráficos de ejecución de la Figura 6.10 y estas irregularidades se sospechan a que se deben a los cambios de contexto, mostrados en la Tabla 6.9. Lo que se muestra en la tabla son los cambios de las medias de las aplicaciones en los trozos de consolidación y hay que comentar que es natural que ocurran los cambios de contexto cuando se ejecutan más de una aplicación en un solo procesador. Ese incremento es notorio en todas las aplicaciones. El cambio de contexto como poco se multiplican por 1.5 y como mucho 7 veces, mientras que las migraciones se llegan a disparar en comparación. Lo que ocurre con bodytrack, que se muestra en negativo, es que decrementan las migraciones y cambios de contexto. Eso

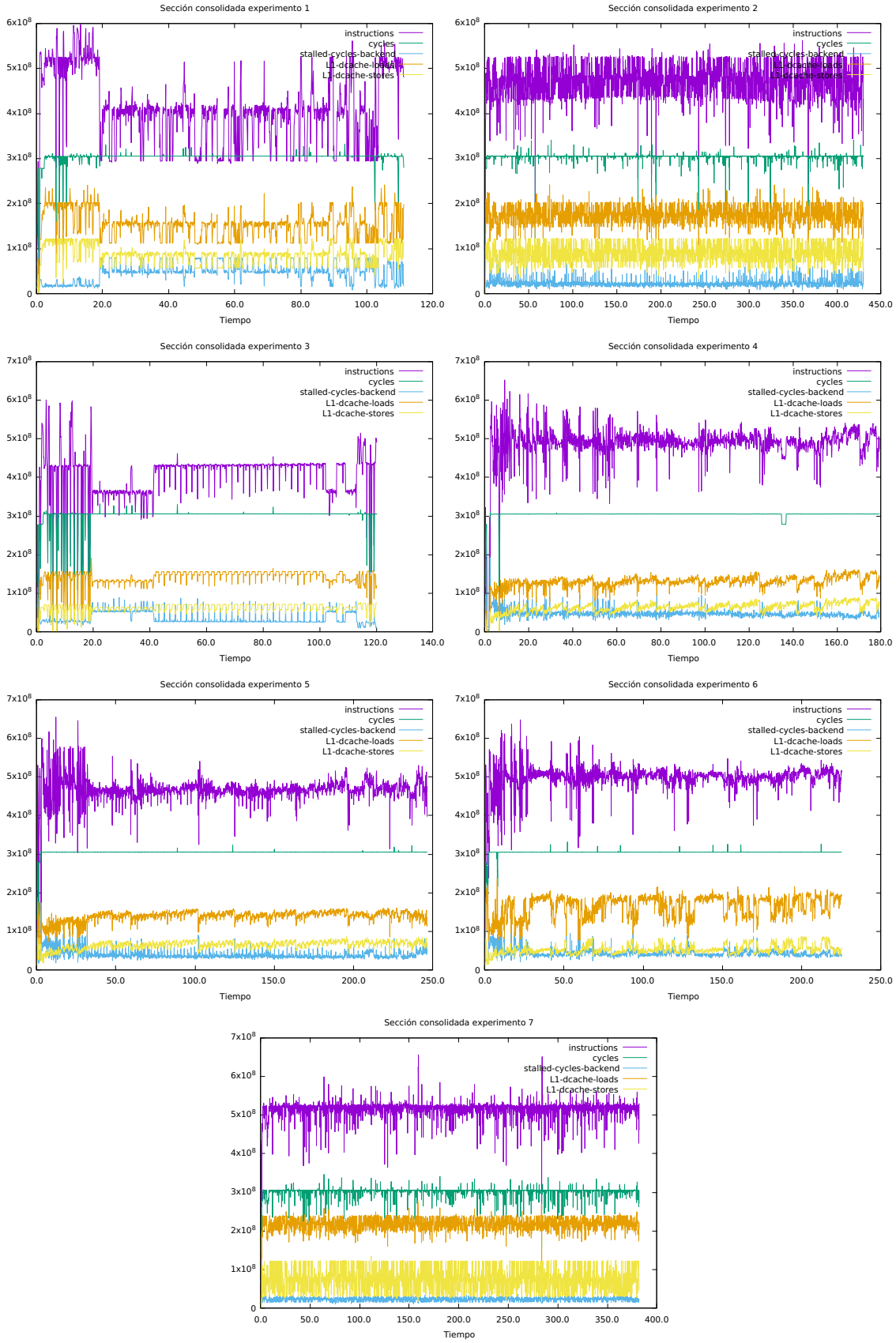


Figura 6.10: Gráficos de los datos de ejecución del segmento de consolidación

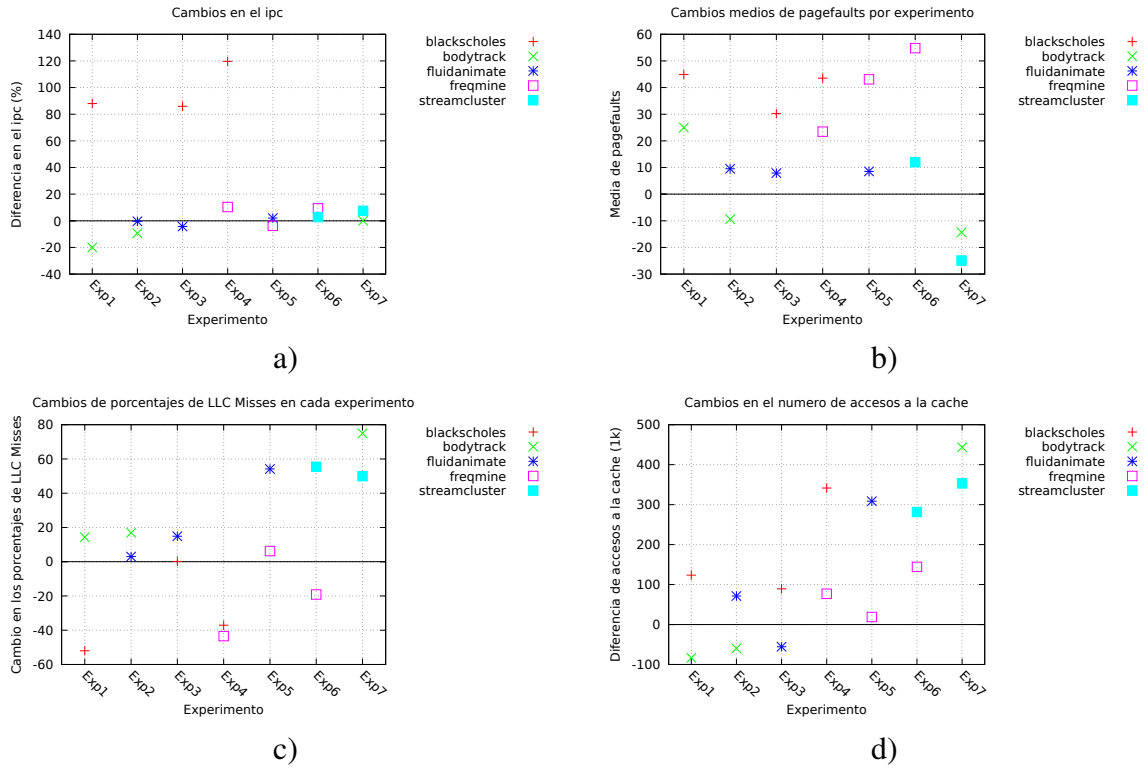


Figura 6.11: Variaciones de las diferentes métricas de los experimentos

puede ser debido a que bodytrack emplea más de un hilo en su paralelización por núcleo. Estos cambios de contexto generan efectos negativos en aplicaciones que necesiten una buena localidad espacial de los datos.

En cuanto a los datos de los contadores, por cada experimento, se han calculado las diferencias entre los contadores para cada aplicación. En lugar de explicar con detalle cada métrica por experimento, se van a comentar los casos que resalten. Cada scatterplot de la Figura 6.11 muestra las diferencias agrupadas por experimento y aplicación.

Un caso de benchmark en específico que vale mencionar es blackscholes. Este ha tenido un incremento general del uso del sistema, y eso se puede ver con la casi duplicación de sus instrucciones por segundo en el scatterplot a) de la Figura 6.11. Claro está que en esas secciones no se encuentra blackscholes solo, si no que también se encuentran los contadores de sus programas emparejados en los experimentos. Aun así, gracias a ello está incrementando el uso del sistema pero eso no quiere decir que el efecto neto haya sido positivo para el sistema.

Empezando con los contadores, primero se comenta lo observado en el scatterplot a) de la Figura 6.11, este muestra los cambios de IPC que han tenido las aplicaciones. Un

mayor índice de IPC indica una mayor utilización del sistema en general, habría que examinar con más detalle utilizando RAPL cuánto se modificó el consumo energético tras consolidar.

Con una rápida observación se pueden ver mas referencias a LLC en el scatterplot d), pero posiblemente se deba por el cambio de contexto. Algo notorio es que el programa que ha tenido el mayor número de referencias en aislamiento, streamcluster, ha conseguido incluso más referencias en este experimento. De estos datos podemos razonar que el aumento de accesos a LLC es un posible indicador de la bajada de rendimiento, dado que esas referencias provienen de fallos a caché de niveles inferiores y estos fallos de caché de primer nivel. Siguiendo la cadena de eventos, la caché que es uno de los componentes más importantes para una ejecución eficiente, su incremento de tasas de fallo seguramente hayan creado un impacto negativo en el rendimiento.

De la misma forma que han aumentado las referencias a LLC, viendo el scatterplot c), la tasa de fallos también ha crecido en general. Algunos programas han experimentado diferentes efectos en comparación, para bodytrack y streamcluster, en ciertos casos ese efecto ha sido muy grande y han alcanzado casi hasta un 80 % de tasa de fallos. Otra de las interacciones de los benchmark ha sido el experimento 6, fluidanimate ha tenido más fallos que en las otras pruebas y probablemente sea porque los benchmarks estaban compitiendo por memoria y caché. Una última observación a notar, blackscholes y freqmine han reducido su tasa de fallos, obteniendo una decente reducción en el experimento 4.

Finalmente, tenemos el scatterplot b) de pagefaults. Como era de esperar, un aumento de la tasa de fallos en LLC ha generado que se produzcan mas fallos de pagina, y esto se puede confirmar en scatterplot. Pero algo que no sigue la misma tendencia es el experimento 7. Lo extraño con esos datos, es que a pesar de que aumenta la cantidad de accesos y la tasa de fallos a LLC, los fallos de página promedio han disminuido para bodytrack y streamcluster. Se ha ponderado al respecto, pero no se llegan a sacar ideas que puedan causar ese efecto.

6.4.3. Resumen de la consolidación

En conclusión, se ha comprobado que la consolidación puede llegar a aumentar la utilización del sistema al menos un 8.7 % en algunos casos, y en los peores casos un 0.43 %. Hay que tener en cuenta que desde el momento que se termina uno de los programas del experimento, la consolidación deja de existir. Como se puede ver en la Tabla 6.10, hay

margen para mejorar, el mayor índice de consolidación es de 68 % y es por eso que se tendría que plantear otro tipo de experimento donde constantemente haya consolidación o igualar el índice entre los experimentos. En las secciones de ejecución consolidadas, los programas totalmente consolidados han tenido de media un 75 % de tiempo de ejecución más largo.

Tabla 6.10: Porcentaje del tiempo de ejecución total consolidado

	Exp1	Exp2	Exp3	Exp4	Exp5	Exp6	Exp7
Porcentaje consolidada	39.42 %	58.00 %	20.22 %	68.73 %	36.94 %	47.19 %	67.08 %

Como se ha podido ver, la consolidación genera ejecuciones más eficientes pero más largas. El peor caso ha sido freqmine, un tiempo de ejecución 94.63 % más largo, es decir, ha requerido casi el doble del tiempo individual. El mejor ha sido un 10.04 % más largo con fluidanimate. Es por ello que tienen que tenerse en cuenta los programas que requieran de ejecuciones rápidas porque, por ejemplo, en las predicciones meteorológicas no quieres tardar más de un día en saber que tiempo va a hacer dentro de un día.

No se esperaban obtener resultados tan remarcables con este tipo de experimento y queda mucho por analizar. Además, no se ha extraído el máximo potencial a la consolidación y este punto también queda por explorar en trabajos futuros. Eso abre puertas a otras investigaciones y crear otras ideas a explorar es uno de los puntos más importantes de la investigación.

6.5. Analizando efecto de la variabilidad

En esta sección se va a estudiar el efecto de la variabilidad en el nodo. Como recordatorio, mencionar que la variabilidad en el entorno de HPC se refiere a las pequeñas diferencias de capacidad de cómputo que se generan en las fases de fabricación de los procesadores.

Para llegar a obtener resultados significativos se tienen que ejecutar aplicaciones que dependan mucho de la CPU. Del set de aplicaciones que se dispone, bodytrack es la aplicación adecuada, tiene alta demanda de CPU y pocos ciclos parados.

El experimento se ha realizado ejecutando 128 veces la versión de bodytrack de 4 hilos en cada CPU. Se han ejecutado con el lanzador utilizado para lanzar los procesos sin perfilador, y de esa forma se han obtenido los tiempo de ejecución.

Si existe alguna diferencia significativa entre ambos tiempos medios, se continuará analizando los datos utilizando p-value y un valor crítico del 5 %. Para ello se va a plantear la siguiente hipótesis nula:

H_0: No hay diferencia significativa entre los dos tiempos medios de ejecución.

Tabla 6.11: Resultados de las ejecuciones de bodytrack

	CPU0	CPU1
Media	234.483	235.321
Desviación	0.073	0.496

Calculando el p-value, obtenemos que es 0 y por lo tanto se rechaza la hipótesis nula. Por lo tanto, existe una diferencia significativa entre las ejecuciones. Junto a los datos estadísticos de la Tabla 6.11, se puede ver que la diferencia es de algo cercano a 0.8s, pero fijándose en las desviaciones estándar se puede notar que hay ruido considerable en el sistema. No se ha encontrado la causa por la que existe esa diferencia de desviación entre las CPU, pero es lo suficientemente grande como para impedir sacar conclusiones sobre la variabilidad.

Aunque no se sepa si ha sido causado por la variabilidad o ruido en el sistema, es un hecho que ejecutarlo en la CPU0 va a tener de promedio menos tiempo de ejecución que en la CPU1. Suponiendo que este tiempo es por la CPU y no por los otros elementos del computador, la CPU0 va un 0.36 % más rápido. Cada segundo se estaría ejecutando lo equivalente a uno 3.6ms extras en la primera CPU. Si el centro HPC tuviera una aplicación recurrente en su workload, como una aplicación de meteorología para la predicción del tiempo, se podría llegar a ahorrar unos segundos extras. Por ejemplo, si se estima que la predicción de la semana va a tardar unas dos horas en ser ejecutada, se podrían llegar a ahorrar 25 segundos de tiempo de ejecución ejecutándola en la CPU0. Este tiempo se seguiría acumulando cada vez que se ejecutase la aplicación recurrente, en un año (52 semanas, 52 ejecuciones de la aplicación meteorológica), esta ejecución hipotética llegaría a ahorrar aproximadamente 22 minutos al ser ejecutada en la CPU 0.

En conclusión, no se han obtenido resultados claros por ruido en el entorno de ejecución. Se necesitaría investigar en como preparar un entorno de ejecución con el mínimo ruido del sistema posible. Sin embargo, la variabilidad no a nivel de fabricación, si no a nivel de ejecución, muestra unos pequeños ahorros temporales que a la larga pueden resultar significantes.

6.6. Conclusiones de los resultados

Para dar fin al capítulo, se ha realizado este resumen de los resultados de los diferentes análisis realizados. Desde el planteamiento inicial de lo que se quería investigar en el capítulo de metodología, hasta su finalización pasando por el lanzamiento de los experimentos, ha transcurrido un tiempo en el que se ha necesitado una atención al detalle minuciosa para anotar y examinar los rasgos y cambios en los datos obtenidos.

Primero se obtuvieron los datos de control con los que poder comparar los números de los otros experimentos. Al ser aplicaciones paralelizadas, se calculó el factor de aceleración (fa) de estas mismas para conocer la eficiencia de paralelización con la que se estaba trabajando que resultó siendo un fa de 3 o una eficiencia del 75 % en general, siendo el benchmark streamcluster la menos paralelizable.

Después se estudió el efecto que produce el perfilador en el sistema. Para ello se ejecutaron los benchmark con y sin el perfilador y los tiempos se compararon planteando una hipótesis y utilizando p-values. Se concluyó que streamcluster y blackscholes eran susceptibles a un leve decremento de rendimiento, que en el mayor de los casos, ese efecto era de hasta un segundo. Por suerte, acabó resultando que ese efecto podría ignorarse dado que los siguientes experimentos mostraron desviaciones mucho más grandes haciendo que esa precisión no importase tanto.

Una vez obtenidos los datos de control y estudiada la posible interferencia del perfilador, se analizaron los contadores de los eventos que guardaba el perfilador. De esos datos, se consiguió generar un perfil para cada aplicación del conjunto de pruebas y se concluyó que era posible diferenciar la naturaleza de las aplicaciones en base a esos contadores. Cabe destacar que dos de esos programas generaron un perfil muy parecido al tener naturalezas parecidas, estos fueron streamcluster y freqmine.

Tras el estudio de las naturalezas, se comenzó el análisis de los efectos de la consolidación. Primero se obtuvo el rendimiento general de ejecutar los experimentos previamente planteados usando esta técnica y los efectos que tienen positivos. Incluso utilizando un scheduler de Linux, los resultados fueron remarcables, hasta un 9 % de aumento de ejecución combinada en el mejor caso. Además, el peor caso no fue negativo, fue un resultado mínimamente positivo, un 0.43 %, siendo este causado por interferencias entre programas IO intensive.

Finalmente se examinó la variabilidad y sus consecuencias. La hipótesis sugería la carencia de diferencia significativa entre los tiempos medios fue rechazada. A pesar de que

se detectase una diferencia en los tiempos medios, existía una desviación suficientemente grande como para suponer que la diferencia no era solo causado por la variabilidad. No obstante, se estimó que se podrían ahorrar 22 minutos de ejecución si se llegase a ejecutar una aplicación periódica cada semana durante un año una aplicación meteorológica de dos horas de duración.

En resumen, se han llegado a estudiar las diferentes técnicas que se pueden llegar a usar para reducir el consumo energético y han mostrado ser efectivas. La consolidación ha llegado a mostrar un grado de aumento de recursos en el sistema, pero no se tenía el entorno necesario para realizarlo de forma más eficiente, y por esa misma razón no se ha analizado todo su potencial. La variabilidad también ha demostrado ser de utilidad, pero solo en las aplicaciones CPU intensive. Personalmente, creo que el mejor resultado que se ha obtenido ha sido poder llegar a diferenciar las naturalezas de las aplicaciones basándose en sus contadores, algo de gran utilidad en el ámbito de la investigación.

7. CAPÍTULO

Conclusiones

En este último capítulo se detallan las lecciones aprendidas tras el desarrollo del presente proyecto y se identifican las posibles oportunidades de mejora sobre el software desarrollado.

7.1. Objetivos alcanzados

Para dar finalización al proyecto, se van a desenvolver los objetivos logrados durante su proceso. El objetivo original planteado era bastante ambicioso para el calibre de un Trabajo de Fin de Grado y por esa razón, se redujo el alcance de este al estudio de técnicas que podrían aumentar la eficiencia. Estas son la consolidación y la utilización de la variabilidad.

Con ese objetivo en mente, primero se estudió el estado del arte respecto el entorno HPC y las técnicas ya empleadas en estos trabajos. De esta forma, se podía centrar en métodos que ya habían generado resultados, pero realizando sobre ellos otro tipo de análisis. Para ello, era necesario una herramienta que pudiera recopilar los datos de la ejecución y de esa forma comenzó el desarrollo del perfilador. Por causas que se desconocen, no llegó a funcionar en el equipo de pruebas, pero si en otros equipos y en el equipo de desarrollo. Aun así, la interfaz o herramienta sobre la que se basaba, perf tools, tenía otra implementación que se pudo utilizar gracias a los conocimientos obtenidos en la herramienta desarrollada. El comando perf junto a scripts auxiliares se utilizó para lanzar y recopilar los datos de las ejecuciones de las diferentes técnicas.

Algo destacable de los resultados obtenidos es el hecho que se pueda llegar a diferenciar con cierta facilidad la naturaleza de las aplicaciones. Los benchmark freqmine y streamcluster han generado un perfil casi exacto, y el resto de aplicaciones que requerían de otro tipo de recursos han generado perfiles diferentes. Esto abre paso a que el aprendizaje automático pueda ser utilizado con cierta certeza para reconocer la fase o tipo de la aplicación, aunque se requerirían más aplicaciones clasificadas previamente. Además, el impacto del perfilador en tiempo de ejecución en ciertos tipos de aplicación se ha observado que no afecta significativamente a la interpretación de los datos.

Después de comprobar el efecto del perfilador, se obtuvieron los datos de los diferentes experimentos de la consolidación, que indican hasta un 8.7 % de incremento del sistema, que no es para menospreciar. El impacto que tiene sobre el sistema es el tiempo de ejecución de las aplicaciones, lo que ocurre es que se ejecutan más lentas en comparación, pero el tiempo combinado es inferior en consolidación que por separado. Se conjetura que tener un scheduler propio pueda llegar a sacar el potencial de la consolidación, encontrando el balance entre el sobrecoste de las migraciones y cambios de contexto y los beneficios de la consolidación.

En cuanto al impacto de la variabilidad, se sospecha que no se han obtenido datos indicativos de la variabilidad por algún ruido del sistema y se necesitaba un entorno más controlado que no se ha conseguido alcanzar. Sin embargo, se ha podido concluir que existe una pequeña pero significativa diferencia que se puede aprovechar aunque el origen no sea la variabilidad de la fabricación de la CPU. Se ha planteado un escenario hipotético donde una aplicación meteorológica de 2 horas de duración se ejecutaba a la semana durante un año. De ese escenario se consiguen ahorrar 22 minutos ejecutando la aplicación priorizando ejecutar la aplicación en la CPU 0 frente a la CPU 1.

En general, se puede decir que se han cumplido los objetivos planteados. A lo largo de este proyecto se han ahondado en diferentes disciplinas de la ingeniería y métodos de investigación. Al conocer el estado del arte, se han estudiado las técnicas y metodologías que se emplean para realizarlos y documentarlos. En el desarrollo del perfilador se ha entrado en el campo del desarrollo software pero sin dejar de lado los conocimientos requeridos del hardware para la gestión del perfilador. La fase de análisis del proyecto ha demostrado los conocimientos adquiridos a lo largo del grado para llegar a interpretar y procesar los datos obtenidos. Eso todo se ha empleado para asentar este proyecto.

7.2. Lecciones aprendidas

A continuación, se detallan las buenas prácticas adquiridas, tanto tecnológicas como procedimentales, así como cualquier otro aspecto de interés.

Antes de realizar cualquier análisis, es crucial realizar una investigación exhaustiva para asegurarse de contar con el conocimiento necesario. Además, es recomendable optar por herramientas genéricas en lugar de soluciones personalizadas, ya que ofrecen mayor soporte y estabilidad.

En cuanto a los datos, es importante mantener el enfoque y no perderse en la abundancia de información. Generar datos está bien, pero se debe seleccionar y utilizar únicamente los más relevantes y significativos para el análisis.

El proyecto ha sido de una escala considerable y trabajarlo no ha sido una tarea fácil. Han sido diferentes disciplinas las requeridas para terminar el proyecto y es importante que en cada una de ellas se tengan unas bases firmes.

7.3. Trabajo futuro

Este proyecto al ser una mezcla entre implementación e investigación, siempre se pueden encontrar diferentes bifurcaciones o campos de interés en los que profundizar. En las investigaciones no siempre es lo más importante sacar resultados positivos, si no explorar las múltiples opciones que se disponen en ese campo, ya sean más o menos útiles.

Para empezar, tener un scheduler que permita estudiar los efectos es de gran importancia. Ese sería un buen primer punto por donde comenzar el trabajo futuro. Otra sección que se ha quedado medio analizada son los efectos de la consolidación, se podría estudiar como afecta las diferentes políticas de la caché y scheduler.

Otro punto importante es la profundización en los análisis que se han realizado. Para ser más específicos, se ha comprobado que los efectos de la consolidación existen, pero, queda potencial por analizar y RAPL no ha podido ser utilizado, dejando de lado el análisis energético que se tenía en mente. ¿Realmente aumentar el uso del sistema ahorra energía? ¿Qué combinaciones de naturalezas de programas son combatibles? Esas y otras preguntas más que han ido surgiendo se pueden utilizar para investigar aún más la consolidación.

Al tener tiempo limitado, la implementación de un scheduler ha quedado fuera del alcance

durante el proyecto, pero este tema podría trabajarse. Los rasgos encontrados en este proyecto podrían ser utilizados en un scheduler que tuviera los siguientes objetivos:

- Utilizar un entorno simulado para obtener una estimación previa.
- Desarrollar una implementación para que pueda usarse en la práctica y no solo en un entorno de prueba. Esto se logrará mediante un plugin de SLURM que se crearía en el proyecto. Este scheduler de HPC ya ha sido utilizado en implementaciones para reducir el uso energético (Ellsworth et al., 2016).

Aunque quizás, antes de realizar ninguna implementación, se tendría que seguir analizando las otras técnicas que se han encontrado, como Gang Scheduling (Wiseman and Feitelson, 2003) o Job Stripping (Breslow et al., 2016).

Resumiendo, primero se podría analizar y explorar más técnicas que puedan incrementar la eficiencia de un supercomputador. El resultado de este análisis se podría continuar e implementar un scheduler en un trabajo futuro.

Bibliografía

Bhadauria, M. and McKee, S. A. (2010). An approach to resource-aware co-scheduling for cmps. In *Proceedings of the 24th ACM International Conference on Supercomputing*, ICS '10, page 189–199, New York, NY, USA. Association for Computing Machinery.

Bienia, C., Kumar, S., Singh, J. P., and Li, K. (2008). The parsec benchmark suite: Characterization and architectural implications. Technical Report TR-811-08, Princeton University.

Blagodurov, S., Zhuravlev, S., and Fedorova, A. (2010). Contention-aware scheduling on multicore systems. 28(4).

Breslow, A. D., Porter, L., Tiwari, A., Laurenzano, M., Carrington, L., Tullsen, D. M., and Snavely, A. E. (2016). The case for colocation of high performance computing workloads. *Concurrency and Computation: Practice and Experience*, 28(2):232–251.

Chasapis, D., Moretó, M., Schulz, M., Rountree, B., Valero, M., and Casas, M. (2019). Power efficient job scheduling by predicting the impact of processor manufacturing variability. pages 296–307.

Chasapis, D., Schulz, M., Casas, M., Ayguadé, E., Valero, M., Moretó, M., and Labarta, J. (2016). Runtime-guided mitigation of manufacturing variability in power-constrained multi-socket numa nodes. pages 1–12.

Dwyer, T., Fedorova, A., Blagodurov, S., Roth, M., Gaud, F., and Pei, J. (2012). A practical method for estimating performance degradation on multicore processors, and its application to hpc workloads. In *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–11.

Ellsworth, D., Patki, T., Schulz, M., Rountree, B., and Malony, A. (2016). A unified platform for exploring power management strategies. In *2016 4th International Workshop on Energy Efficient Supercomputing (E2SC)*, pages 24–30.

Extrae (2023). Extrae bcs tools. Disponible en: https://tools.bsc.es/extrae. Último acceso 23 de junio de 2023.

Hankendi, C. and Coskun, A. K. (2012). Reducing the energy cost of computing through efficient co-scheduling of parallel workloads. In *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 994–999.

Hou, Z., Zhao, S., Yin, C., Wang, Y., Gu, J., and Zhou, X. (2019). Machine learning based performance analysis and prediction of jobs on a hpc cluster. pages 247–252.

Kundan, S., Marinakis, T., Anagnostopoulos, I., and Kagaris, D. (2022). A pressure-aware policy for contention minimization on multicore systems. *ACM Trans. Archit. Code Optim.*, 19(3).

Mishra, N., Lafferty, J. D., and Hoffmann, H. (2017). Esp: A machine learning approach to predicting application interference. In *2017 IEEE International Conference on Autonomic Computing (ICAC)*, pages 125–134.

OProfile (2023). Disponible en: https://oprofile.sourceforge.io/about/. Último acceso 23 de junio de 2023.

Teodorescu, R. and Torrellas, J. (2008). Variation-aware application scheduling and power management for chip multiprocessors. In *2008 International Symposium on Computer Architecture*, pages 363–374.

Valgrind (2023). Disponible en: https://valgrind.org/info/about.html. Último acceso 23 de junio de 2023.

Wiseman, Y. and Feitelson, D. (2003). Paired gang scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 14(6):581–592.

Zacarias, F. V., Petrucci, V., Nishtala, R., Carpenter, P., and Mossé, D. (2021). Intelligent colocation of hpc workloads. *Journal of Parallel and Distributed Computing*, 151:125–137.