Facultad
de
Ciencias

Simulador visual de cachés multinivel
(Visual multi-level cache simulator)

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Agustín Pedraja Crespo

Director: Esteban Stafford Fernández

Septiembre - 2018

Resumen

En la asignatura de Organización de computadores del grado en ingeniería informática los alumnos aprenden el funcionamiento de la jerarquía de memoria de un computador. Dada la importancia de esta cuestión en la arquitectura de computadores y la dificultad que entraña para un estudiante el comprender todos los conceptos que implica, actualmente las clases se apoyan en simuladores.

En concreto, para comprender el funcionamiento de las memorias caché, se emplean los simuladores Visual Caché y Mars MIPS, los cuales presentan una serie de limitaciones. No permiten la simulación de cachés multinivel, no muestran todos los aspectos de la simulación ni el contenido de las memorias y no se dispone de su código para realizar modificaciones.

Es por este motivo que se ha planteado la realización de este proyecto, para desarrollar un simulador visual de memorias caché multinivel enfocado en aprendizaje de los alumnos y que solucione las limitaciones de los simuladores empleados hasta ahora.

Para su desarrollo de seguirá una planificación en varias etapas: análisis, diseño, implementación pruebas y documentación. El código del programa será desarrollado en lenguaje C y se hará uso de la tecnología GTK+ para implementar la interfaz gráfica de usuario, así como del formato de ficheros INI para cargar los parámetros de configuración.

Palabras clave: memoria caché, simulador, jerarquía de memoria, arquitectura de computadores

Abstract

In the Computer Organization course of the degree in computer engineering, students learn how a computer's memory hierarchy works. Given the importance of this issue in computer architecture and the difficulty for a student to understand all the concepts involved, the classes are now supported by simulators.

Specifically, to understand how cache memories work, the Visual Caché and Mars MIPS simulators are used, which have a number of limitations. They do not allow the simulation of multilevel caches, they do not show all the aspects of the simulation nor the contents of the memories and their code is not available to make modifications.

It is for this reason that this project has been planned to develop a visual simulator of multi-level caché memories focused on student learning and to solve the limitations of the simulators used so far.

For its development, a multi-stage planning is going to be followed: analysis, design, implementation, testing and documentation. The program code will be developed in C language and the GTK+ technology will be used to implement the graphical user interface, as well as the INI file format to load the configuration parameters.

Keywords: caché memory, simulator, memory hierarchy, Computer archetucture

Índice de contenidos

1 Introducción

1.1 Motivación y solución propuesta

El conocimiento de la jerarquía de memoria es un aspecto fundamental en la formación de un ingeniero informático. Entender los componentes que la conforman, así como su diseño son claves para extraer el máximo rendimiento de cualquier computador moderno. Sin embargo, la gran cantidad de datos que este subsistema ha de manejar hace difícil a los alumnos comprender su funcionamiento adecuadamente. Por ello no es extraño que la docencia, y la investigación, en este campo de la arquitectura de computadores se apoyen en simuladores.

En concreto, en la asignatura de Organización de Computadores de la titulación de Ingeniería Informática, se usa un simulador visual, pero está limitado por varios factores. Primero, simula un solo nivel de la jerarquía de memoria, y segundo, no se dispone de código fuente para permitir su ampliación con más niveles o nuevas tecnologías.

Es por este motivo que se plantea la necesidad de crear un simulador de cachés enfocado al aprendizaje de los conceptos de la jerarquía de memoria que pueda suplir todas las limitaciones de los simuladores existentes

La solución propuesta en este proyecto es el desarrollo de un simulador que sea sencillo, completo, enfocado al aprendizaje y cuyo código sea libre.

1.2 Objetivos

Para que el simulador desarrollado sea útil y funcional desde el punto de vista del aprendizaje del alumno, en este proyecto se pretende cumplir los siguientes objetivos:

-Simular cachés multinivel, mejorando los simuladores actuales que solo permiten un único nivel de caché.

-Permitir la simulación del funcionamiento de la memoria caché de forma completa incluyendo todos los pasos que componen su funcionamiento.

-Permitir la visualización de estadísticas sobre el estado de la simulación.

-Crear un simulador que sea sencillo de utilizar por el alumno, rápidamente configurable mediante ficheros, y que abstraiga aquellos conceptos que no sean relevantes o entorpezcan el proceso de aprendizaje.

-Desarrollar código que permita su modificación por alumnos en el marco de una práctica.

2 La jerarquía de memoria

En la arquitectura Von Neumann se distinguen como elementos que componen el computador el procesador, la memoria y los dispositivos de entrada/salida.
Tanto las instrucciones como los datos del programa se encuentran almacenados en la memoria y el procesador debe acceder a ellos realizando operaciones de lectura y escritura.

La velocidad de acceso a estos datos es algo muy relevante y el rendimiento del computador depende en gran medida de lo rápido que sea. Por ello uno de los objetivos más importantes de la arquitectura de un computador es reducir el tiempo de acceso a memoria. De lo contrario se convierte en un cuello de botella para el procesador.

No obstante, esto supone un problema. Al mismo tiempo que se quiere alcanzar una gran velocidad de acceso también se espera tener una memoria de gran capacidad. Ambos objetivos son incompatibles, ya que cuanto mayor es un hardware menor es su velocidad. Por el contrario, cuanto menor es el tamaño de una memoria su tiempo de acceso es menor.

Es por esto que en un computador moderno encontramos distintos tipos de sistemas de almacenamiento de información todos ellos con distintas velocidades y capacidades a fin de poder aprovechar las ventajas que supone cada uno de ellos:

Registros del procesador: Son la memoria contenida en el propio procesador. Su capacidad de almacenamiento es muy limitada, únicamente unas decenas de valores. Es la memoria más rápida. Trabaja a la misma velocidad del procesador.

Memorias caché: Son memorias estáticas y volátiles. Es decir, permiten el almacenamiento de la información mientras existe alimentación eléctrica y no requieren refresco de los datos almacenados. Este tipo de memorias permiten un acceso mucho más rápido que la memoria principal del computador, pero su capacidad es muy inferior. Se utiliza como almacenamiento intermedio entre la memoria principal y el procesador.

Memoria principal: Se trata de una memoria dinámica y volátil. Al ser volátil también pierde la información almacenada en ausencia de alimentación eléctrica. Al tratarse de una memoria dinámica que necesita refresco de los datos almacenados los accesos son más lentos, pero permite una capacidad mucho mayor.

Almacenamiento secundario: Se trata de una memoria no volátil. Es decir, la información permanece en ausencia de alimentación eléctrica. Tradicionalmente se basa en propiedades magnéticas y permite unas capacidades muy elevadas. Al mismo tiempo, su tiempo de acceso es muy superior ya que el acceso a un dato implica procesos mecánicos de rotación del disco y movimiento de la cabeza lectora. Como alternativa a los discos duros tradicionales, en la actualidad también existen los discos SSD, basados en tecnología flash, los cuales alcanzan velocidades de acceso superiores. No obstante, a día de hoy su capacidad de almacenamiento es inferior a la de un disco magnético y presentan un coste/ bit más elevado.

Como se ha podido ver todos estos dispositivos tienen diferentes velocidades y capacidades. Es por este motivo que el computador hace uso de todos ellos para optimizar tanto el tiempo de acceso como la capacidad de almacenamiento.

En un computador todos los dispositivos de almacenamiento se encuentran organizados en una jerarquía de memoria en función de estas características.

De esta forma, los dispositivos más rápidos y pequeños se encuentran más cerca del procesador mientras que los dispositivos más grandes y lentos se encuentran en posiciones inferiores de la jerarquía. Cuanto más próximo al procesador el dispositivo de almacenamiento es más pequeño, más rápido y con un coste por bit mayor. Por otra parte, cuanto más alejado del procesador, mayor es su capacidad, mayor su tiempo de acceso y menor su coste por bit.

Los dispositivos más rápidos deben ser los que contengan los datos que se acceden con más frecuencia a fin de poder aprovechar su gran velocidad de acceso.

Por otra parte, los dispositivos más lentos deben almacenar todos los datos, los que se usan más frecuentemente y los que no, a fin de aprovechar su gran capacidad de almacenamiento.

2.1 El principio de localidad

El objetivo final de la jerarquía de memoria es dar la sensación de tener una memoria rápida, barata y de gran capacidad. Para aprovechar todas las ventajas de la jerarquía de memoria es necesario decidir qué nivel de la jerarquía deben ocupar los datos en cada momento. Es decir, la información debe cambiar de nivel bajo demanda.

Para desempeñar esa tarea se aprovecha el hecho de que los accesos que un programa realiza a memoria no son aleatorios, sino que tienden a centrarse en zonas concretas de la memoria. Esto es lo que se conoce como localidad. Existen dos tipos:

-Localidad temporal: Si una posición de memoria ha sido accedida recientemente es muy probable que vuelva a ser accedida pronto. Este es un caso típico de los bucles.

-Localidad espacial: Si una posición de memoria ha sido accedida recientemente es muy probable que pronto se acceda a posiciones cercanas. Esto es muy habitual al trabajar con estructuras de datos como vectores o matrices.

Aprovechando este principio de localidad el controlador de memoria y el sistema operativo pueden hacer una correcta gestión de la jerarquía de memoria moviendo la información entre los distintos niveles en función de la probabilidad con que puedan ser accedidas en cada momento.

2.2 Funcionamiento de la jerarquía de memoria

Tal como se ha explicado, la jerarquía de memoria se compone de varios niveles entre los cuales la información asciende o desciende en función de su demanda. Para ello la jerarquía de memoria funciona en base a una serie de reglas y conceptos.

La información siempre se transfiere entre dos niveles contiguos. El intercambio de información entre dos niveles se realiza en unidades mínimas de intercambio de información denominadas bloques.

El nivel más bajo de la jerarquía contiene toda la información del sistema. A medida que ascendemos en la jerarquía los niveles son más rápidos, pero contienen una porción más pequeña de la información.

El nivel superior de la jerarquía es el único que es accedido por la CPU. Este es el nivel más rápido pero también el que menos información contiene. En él se deberían ubicar los datos que se acceden con mayor probabilidad.

Si en el acceso a un cierto nivel encontramos el dato que buscamos se ha producido un acierto.
En caso contrario se denomina fallo. Si se produce un fallo es necesario acceder al nivel inferior y transferir los datos desde este al nivel actual. Esto conlleva un tiempo de **penalización**.

Para un nivel dado se denomina **tasa de fallos** a la fracción de accesos que provocan un fallo en dicho nivel mientras que denominamos **tasa de aciertos** a la fracción de accesos que provocan un acierto.

De esta forma, para un determinado nivel de la jerarquía el tiempo medio de acceso se expresa mediante la fórmula:

tiempo de acceso = tiempo de acierto + tiempo de penalización * tasa de fallos

Tal como se puede entender en la formula anterior, el fundamento para que la jerarquía de memoria suponga una mejora de rendimiento se basa en obtener una tasa de aciertos elevada y una tasa de fallos baja y así se minimiza el impacto del tiempo de penalización.

2.3 La memoria caché

Dentro de la jerarquía de memoria encontramos la memoria caché, cuya simulación es el objetivo de este trabajo. La memoria caché ocupa el primer nivel de la jerarquía de memoria, entre la CPU y la memoria principal. Almacena aquellas instrucciones y datos que el procesador está utilizando actualmente e idealmente los que va a usar próximamente.

2.3.1 Funcionamiento y conceptos básicos

Esta memoria es accedida de forma directa por el procesador. Para ello se traen a esta memoria desde el nivel inferior los bloques en los que se encuentran las posiciones de memoria que están siendo accedidas por el procesador en este momento. En base al principio de localidad se espera que el resto de datos contenidos en ese bloque sean accedidos pronto. De esta manera la memoria caché persigue el objetivo final de reducir el tiempo de acceso a la memoria principal.

Tal como se ha explicado, la información se intercambia entre la memoria principal y la memoria caché en unidades llamadas bloques. Cada bloque se compone de un mismo número de palabras, siendo una palabra la unidad natural de organización de la memoria.

Teniendo en cuenta esto podemos dividir la memoria principal en un cierto número de **bloques.** Por otra parte, la memoria caché se encuentra dividida en **líneas** con la misma capacidad que un bloque. De esta forma, cuando un bloque de memoria principal es llevado a la caché, este es almacenado en una línea. La asignación de un bloque a una línea concreta se realiza mediante una función de correspondencia, cuyo funcionamiento se explicará más adelante.

2.3.2 Acceso a una palabra

Conociendo ya estos conceptos básicos sobre la memoria caché y su funcionamiento podemos explicar en detalle como accede el procesador a una palabra de la memoria:

- En primer lugar, se intenta leer la palabra de la memoria caché.
- En caso de que el dato se encuentre allí (acierto) se lleva la palabra a la CPU.
- En caso de que no se encuentre (fallo) se realizan las siguientes tareas:
    - Se busca un espacio en caché para el nuevo bloque.
    - Si no hay espacio se reemplaza un bloque que se llevará a memoria principal. Esto se

hace si el bloque de caché a reemplazar ha sido modificado. (Indicado por el bit de validez).
- Se lee el bloque de memoria principal que contiene la palabra a acceder.
- Se lleva el bloque a memoria caché.
- Se envía la palabra solicitada a la CPU.

2.3.3 Cachés multinivel

Es posible que la diferencia entre la velocidad del procesador y la memoria sea tan grande que un solo nivel de caché no obtenga el rendimiento adecuado. Por este motivo los procesadores actuales emplean habitualmente cachés multinivel. Es decir, la caché se compone ahora de su propia jerarquía en la que se explotan los principios de localidad espacial y temporal con el objetivo de lograr en conjunto una caché más rápida y de mayor capacidad.

Habitualmente las cachés multinivel constan de dos o tres niveles.

El primero de ellos, conocido como caché L1 funciona a la velocidad del procesador. El objetivo de este nivel es minimizar el tiempo de acierto y por tanto poder aumentar la velocidad de reloj del procesador. Este nivel de caché en ocasiones es dividido, es decir se compone de dos cachés independientes, una para datos y otra para instrucciones.

La caché L2 tiene una capacidad mayor, con el objetivo de absorber los fallos producidos es L1 y minimizar la tasa de fallos. De esta forma tendrán que realizarse menos accesos a memoria principal.

2.3.4 Función de correspondencia

Dado que la memoria caché tiene una capacidad menor que la memoria principal, es necesario encontrar una función de correspondencia que se encargue de seleccionar una línea concreta en la que colocar un bloque cuando este es llevado desde la memoria principal del computador hasta la memoria caché. Dicha función nos permite además conocer la línea de caché en que se debe buscar una determinada palabra.

Principalmente se emplean tres tipos de funciones de correspondencia, en función de las cuales las cachés se clasifican en **asociativas**, de **correspondencia directa** o **asociativas por conjuntos**.

Correspondencia Directa.

Cada bloque de la memoria principal solamente puede almacenarse en una única línea de caché.
En caso de que dos bloques se asocien a una misma línea se produce una colisión y debe realizarse un reemplazo del bloque.

La línea de caché en la que se debe ubicar un bloque se decide en función de su dirección, para lo cual se divide está en los siguientes campos:

Etiqueta: Identifica el bloque cuando está almacenado en la memoria caché de manera que se puede saber si es el que se está buscando.
Línea: Determina el número de línea dentro de la caché.
Palabra: Indica el número de palabra dentro de un bloque.

Esta función de correspondencia da lugar a cachés más rápidas y baratas, aunque no muy eficientes.

Totalmente asociativa.

No existe una línea concreta en la que deba almacenarse cada bloque. Un bloque puede almacenarse en cualquiera de las líneas de caché. Mientras existan líneas vacías en la caché no se producirá ninguna colisión, por lo que hasta ese momento no se realizarán reemplazos. Esto hace que sean más eficientes.

La dirección de memoria a acceder se descompone en los siguientes dos campos:

Etiqueta: identifica el bloque almacenado en la caché para saber si es el que realmente se busca.
Palabra: Identifica la palabra dentro del bloque.

Para realizar la búsqueda de un dato deben compararse todas las etiquetas simultáneamente, lo cual aumenta su coste notablemente. Si una de las etiquetas coincide con la que estamos buscando decimos que se ha producido un acierto y se lee la palabra indicada por el campo palabra. En caso de producirse un fallo se debe buscar el dato en el siguiente nivel de la jerarquía.

Asociativa por conjuntos

Este tipo de caché combina ambos tipos de correspondencia con el objetivo de obtener un rendimiento mejor que la correspondencia directa, pero con un coste inferior que las totalmente asociativas. La caché se divide en conjuntos de varias líneas, también llamadas vías. Un bloque puede ubicarse únicamente en un conjunto concreto, es decir está asociado a ese conjunto por correspondencia directa. Por otra parte, dentro de un mismo conjunto, cualquier bloque puede ubicarse en cualquier línea vacía. Es decir, dentro de un conjunto se utiliza la correspondencia asociativa.

La dirección se descompone en:

Etiqueta: Identifica si el bloque almacenado en el conjunto es el que se está buscando.
Conjunto: Determina el conjunto de la caché donde será almacenado.
Palabra: Indica la palabra dentro del bloque.

2.3.5 Política de reemplazo

La política de reemplazo se establece para decidir que bloque debe ser sustituido en la caché cuando es necesario incorporar un bloque nuevo y no existe espacio libre en la memoria caché.

Es necesario tener en cuenta que cada función de correspondencia implica un mecanismo de reemplazo distinto. En el caso de una caché que utilice una función de correspondencia directa el bloque a reemplazar está completamente determinado. No hay elección posible. Por otra parte, en las cachés asociativas o asociativas por conjuntos es necesario escoger un bloque a reemplazar dentro de los disponibles. Es aquí donde es necesario establecer una política de reemplazo.

La política de reemplazo ideal sustituiría aquel bloque que vaya a ser utilizado en un futuro más lejano. No obstante, esto es imposible de predecir, por lo que se adoptan otras estrategias que tratan de aproximarse a esta política de reemplazo ideal de alguna forma.

Least Frecuently Used (LFU): Selecciona el bloque utilizado menos frecuentemente. Se basa en la localidad temporal de los accesos a memoria. Su funcionamiento parte del hecho de que los bloques que se acceden con mayor frecuencia tienen una probabilidad mayor de ser accedidos en un futuro próximo. Su mayor problema es que penaliza a los bloques más recientes y su implementación en

hardware es costosa.

Least Recently Used (LRU): Elige el bloque que hace más tiempo que se utilizó por última vez. Es sencillo de implementar para pocos bloques y su rendimiento es bueno. También se basa en la localidad temporal.

Aleatorio: Selecciona un bloque aleatoriamente para reemplazar. Su principal ventaja es ser muy sencillo y rápido. No tiene en cuenta la localidad temporal. La opción de reemplazo escogida habitualmente no es la óptima pero tampoco penaliza a bloques concretos ya que es aleatorio.

First-In First-Out (FIFO): Se basa en reemplazar el bloque más antiguo de la caché. Es muy costoso de implementar y su funcionamiento no es muy eficiente ya que solo tiene en cuenta la antigüedad del bloque y no considera sus accesos posteriores.

2.3.6 Política de actualización

En una jerarquía de memoria es posible encontrar copias de un mismo dato en diferentes niveles. Por ello, en el momento en que se realiza una escritura en el nivel superior de la jerarquía, surge un problema de coherencia, ya que los niveles inferiores tendrían un valor diferente para el mismo dato. Para evitar este problema se emplea una política de actualización que se encarga de propagar la modificación hacia los niveles inferiores de la jerarquía.

Gracias a los mecanismos que implementa la política de actualización se consigue que los mismos datos que se encuentren replicados en varios niveles de la jerarquía de memoria contengan la misma información.

Existen dos políticas de actualización:

Escritura Inmediata (Write Through) Esta política establece que todas las escrituras deben realizarse simultáneamente en todos los niveles de la memoria caché y en la memoria principal. Su principal inconveniente es que todas las escrituras implican necesariamente acceso a memoria principal con el correspondiente incremento de tiempo. Habitualmente se emplea un buffer de escritura que almacena de forma temporal los datos hasta que la operación se complete. De esta forma el procesador no tiene que detenerse mientras exista espacio en el buffer.

Post-escritura (Write Back) Esta política de actualización establece que las escrituras se realizan únicamente en el primer nivel de la memoria caché. Los bloques modificados se actualizan en los siguientes niveles o la memoria en el momento en que son reemplazados. Para distinguir si un bloque ha sido modificado mientras ha estado en la caché se emplea un bit de actualización (Dirty bit).

2.3.7 Modelos de comportamiento

Un modelo de comportamiento de las memorias caché permite comprender cuál es el origen de los fallos de caché y como se ven afectados por la estructura de la jerarquía de memoria.

Fallos de calentamiento: Se producen cuando se accede a un bloque nunca ha estado en la caché.

Fallos de capacidad: Estos fallos se producen cuando el conjunto de los datos de un programa es mayor que la capacidad de la caché, por lo que es necesario reemplazar bloques para incorporar otros.

Fallos de conflicto o colisión: Estos fallos se producen cuando varios bloques tienen que ser

asignados necesariamente a una misma línea de caché. De esta forma, aunque haya líneas disponibles es necesario reemplazar continuamente el mismo bloque provocando sucesivos fallos.

2.3.8 Rendimiento de la memoria caché

En el caso de la memoria caché, al igual que ocurre en el resto de los niveles de la jerarquía de memoria, el tiempo de acceso obtenido depende tanto de la tasa de aciertos y fallos como del tiempo de acierto y el tiempo de penalización.

tiempo de acceso = tiempo de acierto + tiempo de penalización * tasa de fallos

En función del valor que tome cada una de estas variables, el tiempo de acceso medio se verá afectado. De esta forma, cuanto mayor sea la tasa de aciertos y menor la tasa de fallos mayor debería ser el rendimiento. Por otra parte, cuanto menor sean el tiempo de acierto y el tiempo de penalización mayor será el rendimiento.

El principal problema que encontramos es que estos conceptos son contrarios entre sí. Habitualmente, si modificamos la caché para que la tasa de aciertos sea mayor, estaremos incrementando el tiempo de acierto o el tiempo de penalización.

Podemos reducir la tasa de fallos con las siguientes estrategias:

- Reducir los fallos de calentamiento. Para ello podemos aumentar el tamaño de bloque, pero aumentamos el tiempo de fallo al tener que copiar desde la memoria un bloque mayor en caso de fallo.

- Reducir los fallos de capacidad. Para ello podemos aumentar el tamaño de la memoria caché, pero aumentamos el tiempo de fallo al tener que acceder a una memoria mayor.

- Reducir los fallos de conflicto. Esto se soluciona aumentando la asociatividad, pero aumentamos el tiempo de acierto al complicarse el proceso de búsqueda del bloque dentro del conjunto.

Por estos motivos es importante alcanzar un compromiso entre diversos factores en el diseño de una caché buscando aquella configuración que permite obtener un mayor rendimiento. Para ello es especialmente importante tener en cuenta el problema a tratar, especialmente el tipo y tamaño de las estructuras de datos utilizadas, ya que son las que condicionan la forma en que accede el programa a la memoria. Determinados diseños de caché pueden ser más adecuados para el uso de ciertas estructuras de datos y ser poco eficiente para otros.

Todos estos conceptos y sus implicaciones son difíciles de comprender, por lo que el uso de un simulador ayuda en gran medida, pudiendo emplear diferentes configuraciones, mecanismos y trazas de acceso a memoria, para visualizar de forma sencilla el impacto que tiene cada uno de ellos en el rendimiento de la jerarquía de memoria.

3 Evaluación de simuladores existentes

Como punto de partida para el desarrollo de este proyecto se ha estudiado el funcionamiento de algunos simuladores de cachés existentes con el objetivo de evaluar sus diferentes ventajas y limitaciones.

De esta forma, a la hora de desarrollar el nuevo simulador se pudieron tener en cuenta aquellos aspectos positivos de los simuladores existentes que el nuevo programa también debería implementar. De la misma forma también se pudieron considerar aquellos otros aspectos que no eran implementados de forma adecuada, o simplemente no se tenían en cuenta, de forma que el nuevo simulador fuera capaz de suplir esas limitaciones.

Principalmente, se estudió el funcionamiento de los dos simuladores empleados hasta ahora en la asignatura de Organización de computadores, el Visual Caché y el Mars Mips. También se establecieron reuniones con el profesor de la asignatura, en adelante llamado cliente, con el fin de realizar una buena captura de requisitos.

3.1 Visual Caché

Se trata de un simulador de cachés desarrollado por Alfredo González Naranjo y Carlos Manuel Vaquero Rivas como proyecto de fin de carrera del curso 98/99 en la facultad de informática de la universidad de Sevilla. A continuación se exponen las características más relevantes de este simulador.

3.1.1 Aspectos positivos

- La simulación se representa con claridad indicando fallos, aciertos y reemplazos mediante un código de colores. Se proporcionan datos estadísticos sobre la simulación: tasa de aciertos y fallos.
- Permite simular cachés separadas y unificadas.
- Admite muchos parámetros de configuración de caché: política de escritura, función de correspondencia, algoritmo de reemplazo, tamaño de la caché, tamaño de línea y asociatividad.
- Utiliza un fichero de traza en el que se indician las operaciones de memoria, evitando la complejidad de emplear lenguaje ensamblador concreto.
- Permite la simulación continua, paso a paso o hasta punto de ruptura.

3.1.2 Limitaciones

- La configuración se realiza necesariamente a través de un cuadro de diálogo de la interfaz gráfica. Permite utilizar fichero de configuración, pero es un fichero binario que debe ser generado necesariamente a través de la aplicación.
- La simulación está limitada a un solo nivel de caché.
- Los ficheros de traza tienen un formato muy estricto que no permiten comentarios ni espacios en blanco.
- Las estadísticas de ejecución únicamente se muestran al finalizar la simulación.
- No permite la modificación del fichero de traza durante la ejecución.

- No muestra visualmente todos los detalles que tienen lugar en el proceso de simulación (aunque si los tiene en cuenta). Únicamente indica acierto, fallo y expulsión.

- No se muestra el contenido de la memoria ni del único nivel de caché que se simula. Tampoco permite visualizar números de línea, conjunto, palabra, ni bits de validez o suciedad. Únicamente se muestra la etiqueta de la línea almacenada.

- Utiliza conceptos relacionados con anchura y velocidad de bus que entorpecen el proceso de aprendizaje y serían fácilmente reemplazables por tiempos de acceso.

- En las cachés asociativas por conjuntos muestra los conjuntos en horizontal, lo que puede causar cierta confusión a los alumnos.

- No se dispone de código fuente. Únicamente se dispone del ejecutable para sistemas operativos Windows.

- No se puede automatizar para hacer baterías de simulaciones variando parámetros.

3.2 Mars MIPS

MARS es un entorno de desarrollo interactivo ligero (IDE) para programación en el lenguaje ensamblador MIPS, destinado para uso educativo con el libro Patterson y Hennessy Computer Organization and Design [1]. Entre sus funciones encontramos un simulador de memorias caché.

3.2.1 Aspectos positivos

- La simulación se representa con claridad indicando fallos y aciertos mediante un código de colores. Se proporcionan datos estadísticos sobre la simulación en tiempo de ejecución a través de un log.
- Admite muchos parámetros de configuración de caché: política de escritura, función de correspondencia, algoritmo de reemplazo, tamaño de la memoria caché, tamaño de línea y asociatividad.
- Permite la simulación hasta punto de ruptura, continua y paso a paso.

- El código fuente es libre y está programado en java. El programa se encuentra disponible tanto para sistemas Linux como Windows.

3.2.2 Limitaciones

- El simulador de cachés se integra dentro de un simulador MIPS, lo que eleva su complejidad y hace necesario la compresión de otros conceptos no relacionados con memorias caché para poder utilizar el programa.

- No existe la posibilidad de utilizar un fichero de traza. Simula las operaciones de memoria a través de un código ensamblador MIPS. Esto limita el simulador a aquellos programas escritos en dicho lenguaje.

- Únicamente simula cachés de datos. Es decir, el impacto del acceso a instrucciones es ignorado por completo.
- La configuración se realiza necesariamente a través de un cuadro de diálogo de la interfaz gráfica.
- La interfaz de la simulación es pequeña y limitada. No permite visualizar correctamente la simulación de cachés de gran tamaño.
- La simulación está limitada a un solo nivel de caché.
- No permite la modificación del código durante la ejecución.
- No muestra visualmente todos los detalles que tienen lugar en el proceso de simulación (aunque los tiene en cuenta). Únicamente indica acierto y fallo.
- Se omiten muchos valores interesantes de la simulación que no se muestran en la interfaz. Sí proporciona alguna información a través de un log.
- No se puede automatizar para hacer baterías de simulaciones variando parámetros.

4 Planificación del proyecto

Una vez hecho un estudio previo de los simuladores existentes y conocidas las posibilidades que estos ofrecen se comenzó con el proyecto de desarrollo del nuevo simulador.
En primer lugar se estableció un esquema de planificación en cascada compuesto de las varias fases.

Una fase inicial de **Análisis** y captura de requisitos, tanto funcionales como no funcionales. En esta fase se determinó que el sistema solo tiene un actor, el usuario, y se desarrollaron los casos de uso para modelar los requisitos del sistema. A esta fase se le dedicó un tiempo de dos semanas.

A continuación se planificó una fase de **diseño,** para la cual se estimó una duración de tres semanas necesarias para su realización.

Seguidamente se planificó la fase de **implementación**, en la que se desarrolló el código del sistema en base a los diseños previamente elaborados. Se planeó un tiempo de siete semanas para la implementación.

Por último, como última fase abarcada por este proyecto se planificó la realización de las fases de **pruebas** y de **documentación,** a realizarse simultáneamente a la implementación. Además, se planificó que estas fases pudieran extenderse durante un tiempo adicional de tres semanas una vez terminada la implementación.

5 Análisis de requisitos y casos de uso

5.1 Descripción del sistema

El sistema a desarrollar es, tal como ya se ha indicado anteriormente, un simulador de jerarquía de memoria caché. Se trata de una aplicación monousuario que permite la simulación del funcionamiento interno de una jerarquía de cachés multinivel. Para ello accede y simula de forma secuencial las operaciones de memoria indicadas en un fichero de traza.

Permite su uso en dos modos distintos de funcionamiento, en modo no gráfico, en el que únicamente se muestran las estadísticas finales de la simulación, y el modo gráfico en el que se puede visualizar el funcionamiento interno y el contenido de la jerarquía de memoria caché paso por paso además de disponer de las estadísticas de simulación en cada paso de la simulación.

Los parámetros concretos de la estructura interna de la jerarquía de memoria son configurables a través de un fichero que es leído por la aplicación durante su inicio.

5.2 Identificación de actores del sistema

En este caso el sistema a desarrollar es utilizado únicamente por un actor, el usuario de la aplicación. Es quién establece los parámetros de la simulación y las operaciones a simular, además de controlar el avance de la simulación y visualizar los datos y estadísticas generados.

5.3 Requisitos no funcionales

1. La aplicación debe ser compatible con sistemas Linux.
2. El simulador será configurable mediante un fichero con formato INI.
3. Por exigencias del cliente, para facilitar un posterior mantenimiento será programado íntegramente en lenguaje C.
4. La interfaz gráfica de la aplicación deberá ser adaptable al tamaño de la ventana y en ningún caso los elementos que la componen emplearán posiciones o tamaños fijos.
5. La secuencia de operaciones de memoria a simular se indicará a través de un fichero de traza, en el formato empleado por el simulador Visual Caché, de forma que se conserve su compatibilidad.
6. La interfaz gráfica, será implementada utilizando la librería GTK+ dada su amplia popularidad en lenguaje C y que las librerías Qt y WxWidgets únicamente están disponibles para lenguaje C++.
7. Todos los textos, variables, palabras clave de la configuración y comentarios estarán escritos en inglés.

5.4 Requisitos funcionales

1. La aplicación simulará el funcionamiento de una jerarquía de memoria compuesta de una memoria principal y entre 0 y 10 niveles de cachés divididas o unificadas.
2. El simulador realizará una lectura inicial de los parámetros de configuración y mostrará al usuario los errores encontrados en el fichero si los hubiera.

3. El usuario podrá configurar para cada nivel de caché el tamaño, el tamaño de línea, la asociatividad, la política de escritura (write back, write through), la política de reemplazo (lru, lfu, fifo, ramdom), el tiempo de acceso y si es dividida o separada.
4. El usuario podrá configurar para la memoria principal su tamaño, el tiempo de acceso a la primera palabra del bloque, el tiempo de acceso en ráfaga, el índice de la página a mostrar en la simulación y el tamaño de página.
5. El usuario podrá configurar para la CPU su frecuencia de reloj, el tamaño de palabra y el tamaño de dirección.
6. El simulador realizará un análisis inicial del fichero de traza e indicará al usuario los errores encontrados si los hubiese.
7. En el fichero de traza el usuario podrá configurar el tipo de acceso (lectura, escritura), la dirección a acceder, si es dato o instrucción, el tamaño accedido, y el contenido, si es escritura.
8. El programa calculará y almacenará estadísticas durante el proceso de simulación: aciertos, fallos, tasa de aciertos y tasa de fallos para cada nivel de caché y tiempo de acceso.
9. La simulación podrá realizarse en modo gráfico mostrando todos los detalles del proceso, o en modo no gráfico indicando las estadísticas al final de la simulación.
10. En la simulación en modo gráfico las estadísticas se actualizarán en pantalla a cada paso de la ejecución.
11. En el modo gráfico el usuario dispondrá de la posibilidad de simular un paso o simular todo (hasta punto de ruptura).
12. La aplicación mostrará el fichero de traza en la esquina superior izquierda de la ventana, resaltando en otro color la siguiente operación a simular.
13. El usuario podrá editar la traza de simulación desde el programa en modo gráfico cuando la simulación esté detenida.
14. Si durante la ejecución en modo gráfico se alcanza una línea de traza errónea se detendrá la simulación y se mostrará un mensaje de error emergente. La simulación no podrá continuar hasta que el usuario corrija el error.
15. La aplicación mostrará las estadísticas de la simulación en la esquina inferior izquierda de la ventana, y permitirá al usuario ocultar o mostrar la información de los niveles de la jerarquía que más le interesen.
16. La aplicación mostrará los niveles de caché de izquierda a derecha en la parte central de la ventana siendo la caché L1 la que se encuentra más a la izquierda. En caso de cachés divididas se mostrará la caché de datos en la parte superior y la de instrucciones en la parte inferior.
17. La aplicación mostrará la memoria principal en la región derecha de la ventana.
18. La aplicación mostrará los botones de control en la región superior de la ventana.
19. El simulador representara la memoria mediante una tabla en la que cada fila contiene la dirección de una palabra y su contenido.
20. El simulador representará cada caché mediante una tabla en la que cada fila representa una línea de caché. Se emplearán varias columnas para almacenar los diferentes campos de la línea de caché: línea, conjunto, etiqueta, contenido, bit de validez, bit de actualización, estadísticas de accesos previos.
21. El usuario podrá indicar mediante una máscara en el fichero de configuración las columnas de cada tabla de caché que quiere visualizar.
22. El simulador empleará un código de colores sobre diferentes filas de las tablas de caché y memoria para indicar acciones que se estén realizando durante el proceso de simulación.
23. El simulador realizará un desplazamiento automático de la barra de scroll hacia la región de cada tabla sobre las que se esté actuando en cada momento de la simulación.
24. El usuario podrá reiniciar la simulación en cualquier momento si necesidad de tener que finalizar el programa.

5.5 Casos de uso

En este apartado se describen las actividades y funciones que componen el funcionamiento de la aplicación de simulación de cachés a desarrollar. Para este propósito se ha empleado el diagrama de casos de uso mostrado en la figura 1. que más adelante se describe de forma detallada mediante el uso de plantillas.

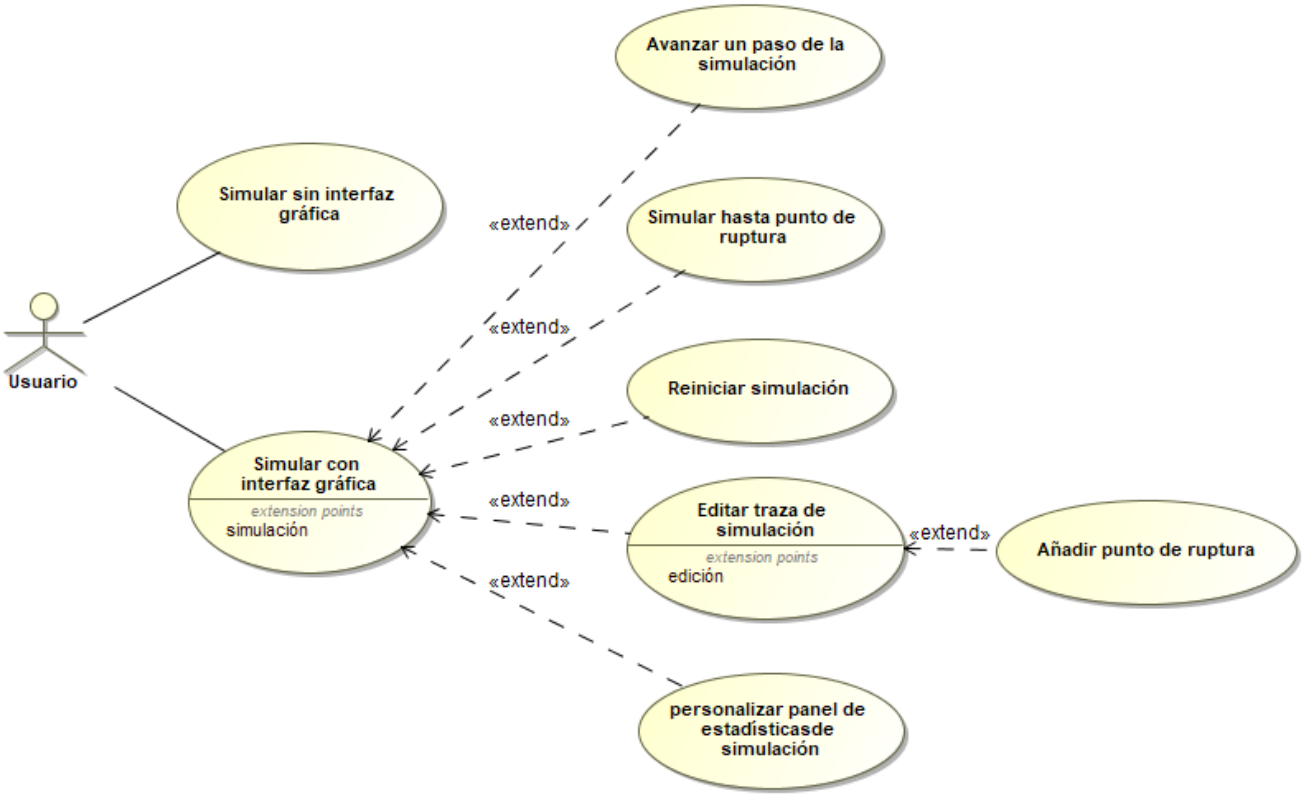

Figura 1. Diagrama de casos de uso

5.6 Plantillas

A continuación, se explican de forma detallada mediante el uso de plantillas cada uno de los casos de uso del diagrama anterior para una mejor comprensión de los mismos.

5.6.1 Plantilla: simular sin interfaz gráfica

Plantilla	
ID	SIMULADOR_CACHÉ_01
Nombre	Simular sin interfaz gráfica
Descripción	El usuario ejecuta la simulación desde la consola y visualiza las estadísticas finales de la simulación.
Actores	Usuario de la aplicación.
Precondiciones	El usuario se encuentra en la consola del sistema. Existe un fichero de configuración.

	Existe un fichero de traza.
Flujo Principal	1. El usuario inicia la aplicación desde la consola indicando la opción sin interfaz gráfica "-g". 2. La aplicación analiza los ficheros de configuración y de traza en busca de posibles errores. 3. La aplicación ejecuta la simulación completa de las operaciones indicadas por el fichero de traza. 4. La aplicación muestra en la consola del sistema las estadísticas finales de la simulación y finaliza.
Flujo Alternativo	3.1 El sistema indica por consola que se ha detectado un error de configuración o un error en el fichero de traza y finaliza.

5.6.2 Plantilla: simular con interfaz gráfica

Plantilla	
ID	SIMULADOR_CACHÉ_02
Nombre	Simular con interfaz gráfica
Descripción	El usuario ejecuta la aplicación de simulación en el modo con interfaz gráfica.
Actores	Usuario de la aplicación.
Precondiciones	El usuario se encuentra en la consola del sistema. Existe un fichero de configuración. Existe un fichero de traza.
Flujo Principal	1. El usuario inicia la aplicación desde la consola indicando la opción con interfaz gráfica. 2. La aplicación analiza los ficheros de configuración y de traza en busca de posibles errores. 3. La aplicación abre una ventana en la que se muestra la simulación en su estado inicial y preparado para ejecutar la primera instrucción. 4. El sistema queda a la espera de una acción por parte del usuario.
Flujo Alternativo	3.1 El sistema indica por consola que se ha detectado un error de configuración o un error en el fichero de traza y finaliza.

5.6.3 Plantilla: avanzar un paso en la simulación

Plantilla	
ID	SIMULADOR_CACHÉ_03
Nombre	Avanzar un paso de la simulación
Descripción	El usuario ejecuta la simulación de la siguiente línea de traza.
Actores	Usuario de la aplicación.
Precondiciones	La aplicación se encuentra ejecutándose en el modo con interfaz gráfica.
Flujo Principal	1. El usuario pulsa el botón de avanzar un paso. 2. La aplicación simula la línea de traza actual.

	3. La aplicación avanza a la línea de traza siguiente. 4. El sistema queda a la espera de una acción por parte del usuario.
Flujo Alternativo	2.1 La aplicación detecta que se ha alcanzado el final del fichero de traza y no simula ninguna operación.
Flujo Alternativo 2	2.1 La aplicación detecta un error en la línea de traza actual y muestra un mensaje por pantalla.

5.6.4 Plantilla: ejecutar hasta punto de ruptura

Plantilla	
ID	SIMULADOR_CACHÉ_04
Nombre	Ejecutar hasta punto de ruptura
Descripción	El usuario ejecuta la traza hasta que se encuentra un punto de ruptura.
Actores	Usuario de la aplicación.
Precondiciones	La aplicación se encuentra ejecutándose en el modo con interfaz gráfica.
Flujo Principal	1. El usuario pulsa el botón de avanzar hasta punto de ruptura. 2. La aplicación ejecuta la simulación de la traza hasta que se encuentra una línea con punto de ruptura o hasta que alcanza el final del fichero. 3. El sistema queda a la espera de una acción por parte del usuario.
Flujo Alternativo	2.1 La aplicación detecta un error en una línea de traza y define la simulación. 2.2 La aplicación muestra un mensaje de error en la pantalla.

5.6.5 Plantilla: reiniciar simulación

Plantilla	
ID	SIMULADOR_CACHÉ_05
Nombre	Reiniciar simulación
Descripción	El usuario reinicia la simulación del fichero de traza actual.
Actores	Usuario de la aplicación.
Precondiciones	La aplicación se encuentra ejecutándose en el modo con interfaz gráfica.
Flujo Principal	1. El usuario pulsa el botón de reinicio. 2. La aplicación resetea el contenido de la jerarquía de memoria a su valor inicial. 3. La aplicación resetea el contenido del panel de estadísticas. 3. La aplicación establece la primera línea del fichero de traza como próxima línea a ejecutar. 4. La aplicación queda a la espera de otra acción por parte del usuario.
Flujo Alternativo	

5.6.6 Plantilla: editar fichero de traza

Plantilla	
ID	SIMULADOR_CACHÉ_06
Nombre	Editar fichero de traza.
Descripción	El usuario edita el fichero de traza.
Actores	Usuario de la aplicación.
Precondiciones	La aplicación se encuentra ejecutándose en el modo con interfaz gráfica.
Flujo Principal	1. El usuario pulsa con el cursor del ratón en la sección del cuadro de texto en la que quiere realizar una edición del texto. 2. El usuario inserta, elimina o escribe el texto en el fichero de traza.
Flujo Alternativo	

5.6.7 Plantilla: añadir punto de ruptura

Plantilla	
ID	SIMULADOR_CACHÉ_07
Nombre	Añadir punto de ruptura
Descripción	El usuario añade un punto de ruptura.
Actores	Usuario de la aplicación.
Precondiciones	La aplicación se encuentra ejecutándose en el modo con interfaz gráfica.
Flujo Principal	1. El usuario pulsa con el cursor del ratón en el inicio de la línea de traza en la que se va añadir un punto de ruptura. 2. El usuario inserta el símbolo "**!**" al comienzo de la línea.
Flujo Alternativo	

5.6.8 Plantilla: personalizar panel de estadísticas de simulación

Plantilla	
ID	SIMULADOR_CACHÉ_08
Nombre	Personalizar panel de estadísticas de simulación.
Descripción	El usuario personaliza los datos mostrados por el panel de estadísticas.
Actores	Usuario de la aplicación.
Precondiciones	La aplicación se encuentra ejecutándose en el modo con interfaz gráfica.
Flujo Principal	1. El usuario pulsa sobre una sección del panel de estadísticas. 2. Se despliega el contenido completo de esa sección del panel.
Flujo Alternativo	2.1 La sección se oculta debido a que estaba en estado desplegado al pulsar sobre ella.

5.7 Mockups

Como apoyo a la fase de análisis y captura de requisitos se elaboró conjuntamente con el cliente un esbozo de la interfaz de la aplicación tal como se observa en la figura 2. En él se incluye la apariencia con que se representará la jerarquía de memoria, así como el fichero de traza que se está simulando o las estadísticas de la simulación en tiempo real.

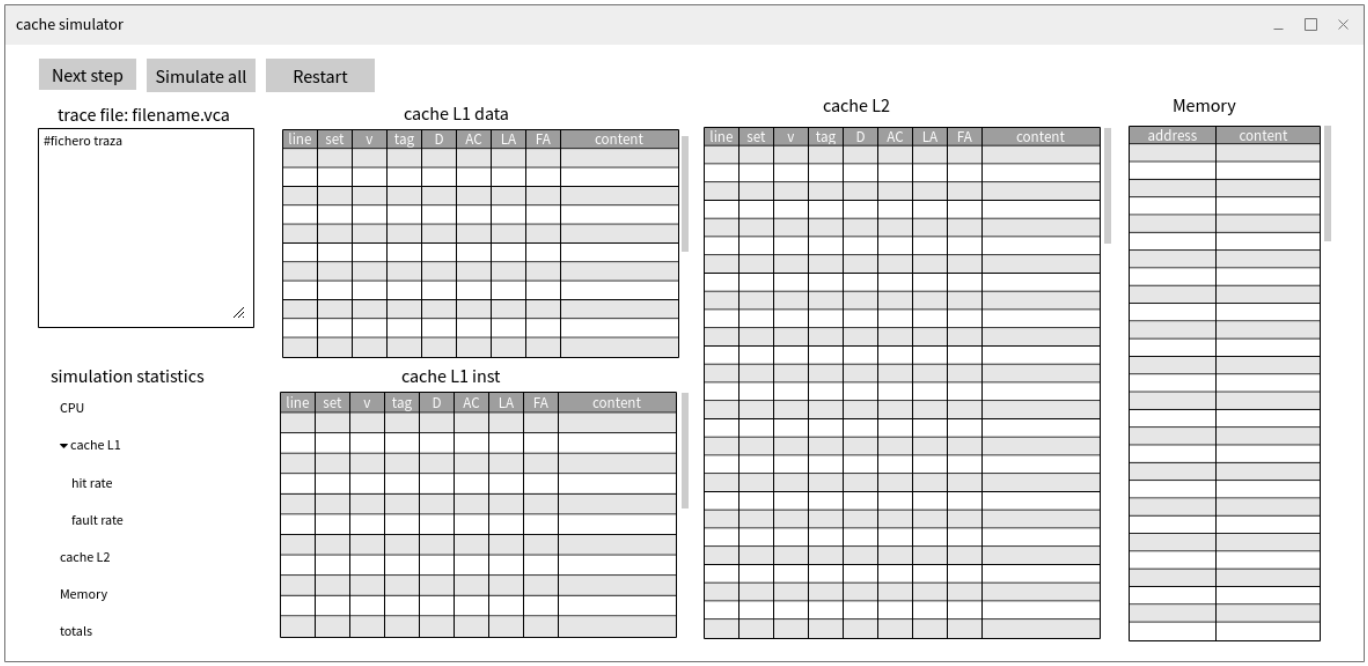

Figura 2. Mockup de la interfaz gráfica del programa.

Tal como se puede observar la región superior de la interfaz se reserva para los botones que implementan las diferentes acciones que puede ejecutar el usuario de la aplicación: avanzar un paso, simular todo (hasta punto de ruptura si existe) y reiniciar la simulación.

En la región izquierda hay dos partes diferenciadas. En la parte superior encontramos un cuadro de texto que almacenará la traza de simulación. El texto de este cuadro será editable por el usuario cuando la simulación esté detenida. En la parte inferior encontramos un panel desplegable que almacenará las estadísticas de la simulación. Su contenido se actualizará cada vez que se detenga la simulación.

A la derecha de la región anteriormente descrita encontramos una serie de tablas que representan la jerarquía de memoria. De izquierda a derecha se encuentran los distintos niveles ordenados según su proximidad al procesador, siendo la memoria principal el nivel ubicado más hacia la derecha.

Para los niveles de caché se emplea una tabla en caso de cachés unificadas o dos tablas en caso de cachés separadas (datos e instrucciones). Cada tabla tiene tantas filas como líneas tiene la memoria caché que representa. Por otra parte, existen columnas para almacenar los siguientes campos: número de línea, conjunto, bit de validez, etiqueta, bit de suciedad, accesos realizados, último acceso, primer acceso y contenido de la línea.
En el caso de la memoria principal la tabla que la representa tiene tantas filas como palabras contiene la memoria. En cuanto al número de columnas, se emplean dos, una para la dirección y otra para el contenido almacenado en dicha dirección.

5.8 Fichero de configuración

De forma similar a cómo se hizo con la interfaz gráfica del programa, como apoyo a la fase de análisis y captura de requisitos también se colaboró con el cliente en el diseño de la estructura del fichero de configuración que utilizaría la aplicación.

```
; Este es un fichero de configuración típico, con dos niveles de cache.

[ CPU] ; Obligatorio
word_width = 32 ; Número de bits en la palabra del procesador
address_width = 32 ; Número de bits en la dirección del procesador.
frequency = 1G ; Frecuencia en hercios. Ver Nota 1.
trace_file = traza.vca ; Fichero de traza

[cache1] ; Opcional
line_size = 64 ; Tamaño de línea en bytes. Ver Nota 2.
size = 4K ; Tamaño cache en bytes. Ver Nota 2.
asociativity = F ; Asociatividad. Ver nota 6
write_policy = wb ; Política de escritura. Ver nota 7
replacement = lru ; Política de reemplazo: lru, rand
separated = no ; Si la memoria cache es separada. Nota 3.
access_time = 6
column_bit_mask= 1111111111

[cache2] ; Opcional
line_size = 64 ; Tamaño de línea en bytes. Ver Nota 2
size = 16K ; Tamaño cache en bytes. Ver Nota 2.
asociativity = 4 ; Asociatividad. Ver nota 6
write_policy = wt ; Política de escritura. Ver nota 7
replacement = lru ; Política de reemplazo: lru, rand
separated = 1 ; Si la memoria cache es separada. Nota 3.
access_time = 6
column_bit_mask= 1111111111 ; Columnas a mostrar. Mascara en binario

[memory] ; Obligatorio
size = 2G ; Tamaño de la memoria en bytes. Ver Nota 2.
access_time_1 = 50m ; Timepo de acceso en segundos. Ver Nota 5.
access_time_burst = 10n ; Tiempo de acceso en ráfaga. Ver nota 5.
page_base_address = 0x8000000 ; Dirección base de la página.
page_size= 2k ; Tamaño de página. Ver Nota 2.

; Nota 1: Puede llevar un multiplicador G para 1e9, M para 1e6 o K para 1e3. Cualquier otro
carácter dará un error.
; Nota 2: Puede llevar un multiplicador G para 2^30, M para 2^20 o K para 2^10. Cualquier otro
carácter dará un error.
; Nota 3: Un valor booleano, deberá aceptar 1 o 0, true o false, yes o no, en mayúsculas y
minúsculas.
; Nota 4: Tiene que ser una 'l', mayúscula o minúscula, seguida de un entero.
; Nota 5: Puede llevar un multiplicador p para 1e-12, n para 1e-9, u para 1e-6, m para 1e-3
; Nota 6: 1 = Mapeo directo, F = totalmente asociativa, cualquier otro número potencia de dos.
; Nota 7: wt = write through, wb = write back
```

Figura 3. Ejemplo de fichero de configuración

En dicho fichero debían quedar representadas todas las propiedades de la jerarquía de memoria a configurar. Es decir, una serie de elementos (CPU, memoria, cachés) y los atributos de cada uno de ellos, por lo que se optó por emplear un fichero con formato **ini.** Este tipo de fichero contiene varias secciones en las que se almacenan claves con su respectivo valor.

De esta forma, se decidió conjuntamente con el cliente las secciones y parámetros de configuración a incluir. Esto permitió detectar a tiempo requisitos que inicialmente no se habían tenido en cuenta y que de otra forma hubieran surgido con el proyecto más avanzado.
Este es el caso de emplear tiempos de acceso en lugar de ciclos de bus y frecuencias como se había previsto inicialmente al basarnos en la configuración de Visual Caché. Lo mismo ocurrió con la necesidad de incluir una máscara para selección los campos de caché a mostrar en la interfaz, requisito

que inicialmente no se había detectado a partir de la descripción que el cliente había aportado del sistema. Un ejemplo de configuración con dos niveles de caché se muestra en la figura 3.

6 Diseño

6.1 Modelo de domino

Una vez concluida la fase de análisis de requisitos y casos de uso del proyecto software, se comenzó la fase de diseño. Se creó el diagrama de clases de la figura 4 para representar el modelo de dominio del problema. En él podemos identificar todas las entidades involucradas en el sistema, así como sus atributos, roles y relaciones entre ellos. Esto es de una gran utilidad a la hora de comprender el problema, aunque en este caso, dado que no disponemos de orientación a objetos será necesario realizar algunas adaptaciones antes de su implementación.

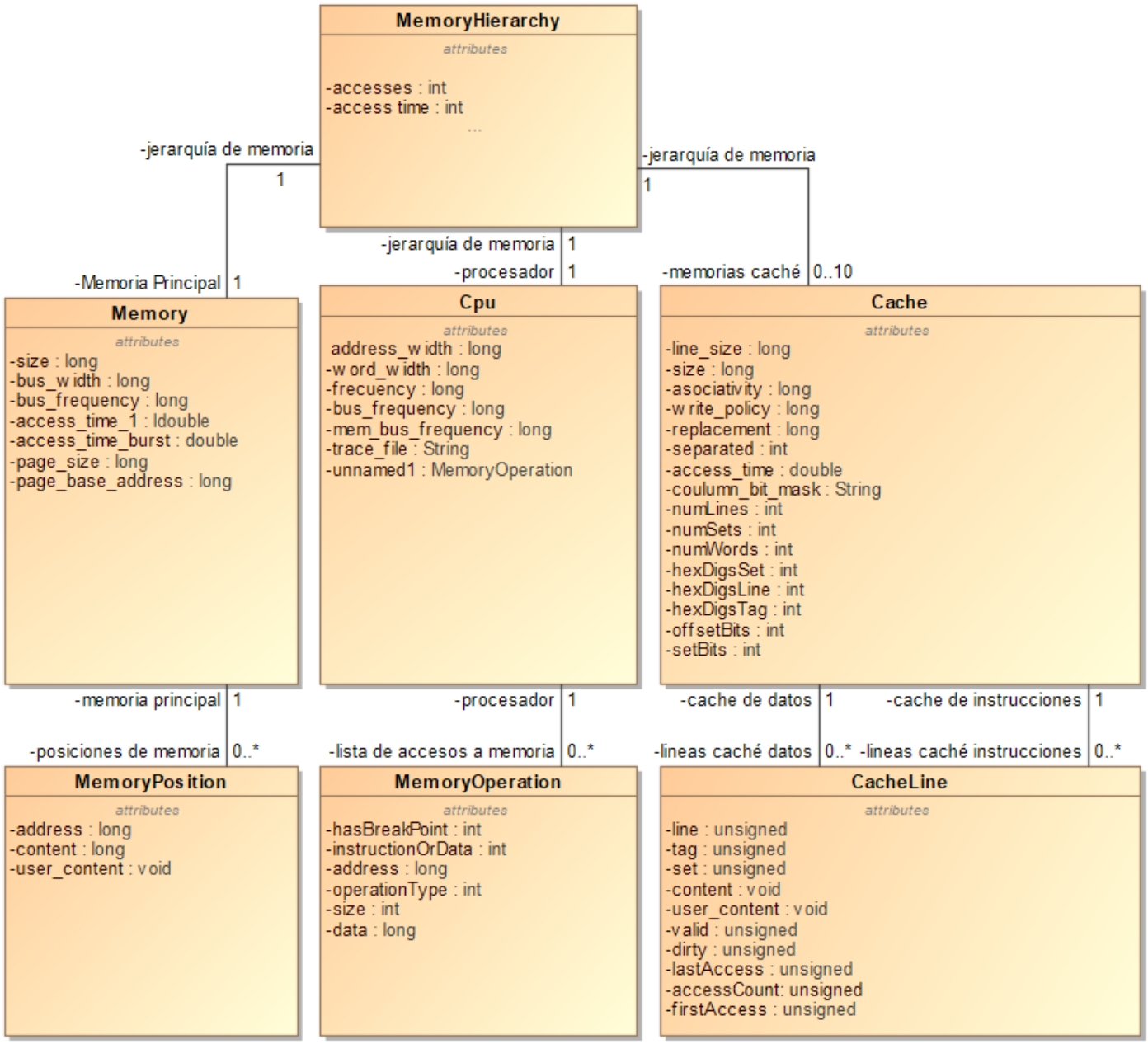

Figura 4. Diagrama de clases

6.2 Estructura del programa

Tal como se ha mencionado, partiendo del diagrama de clases anterior, a la hora realizar un diseño del futuro código hubo que realizar ciertas adaptaciones teniendo en cuenta las particularidades del lenguaje C.

En primer lugar, debido a que no disponemos de orientación a objetos, las clases anteriormente identificadas no tienen una implementación directa en este lenguaje. En su lugar, se emplearon **"structs"**, un tipo de dato de este lenguaje que se compone de varios atributos y del cual es posible crear diferentes instancias.

En segundo lugar, los programas C se estructuran habitualmente en varios módulos, conteniendo cada uno de ellos un cierto número de funciones y variables globales. Cada uno de estos módulos desempeña una función diferente dentro del programa.

En nuestro caso optamos por el siguiente diseño, mostrado en la figura 5. Como se pude observar, la aplicación consta de módulos encargados de diferentes tareas: la lectura de ficheros de configuración y traza, la gestión de las estructuras de datos, la interfaz gráfica y la propia simulación de la jerarquía de memoria.

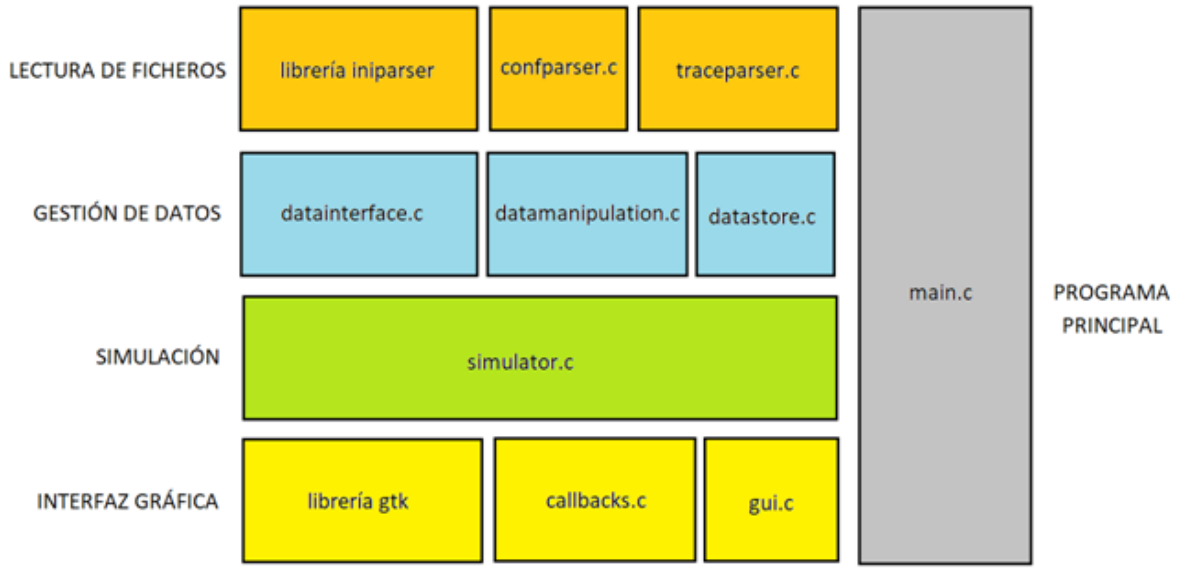

Figura 5 Módulos que componen el programa

A continuación se explicará de forma detallada cada uno de los módulos que componen el programa.

6.2.1 traceparser.c

El módulo traceparser.c se encarga de la lectura del fichero de traza que contiene las operaciones de memoria a simular. Para ello contiene varias funciones que realizan la lectura del fichero, preprocesan y analizan líneas concretas de traza o muestran las operaciones analizadas por pantalla.

Por otra parte, en este módulo encontramos el struct **memOperation**, el cual contiene los atributos propios de una operación de memoria. Existe una lista de structs de este tipo en la que se almacenan todas las operaciones de memoria leídas desde el fichero de traza, además de una variable global **numOperaciones** que indica el número tal de operaciones de memoria.

6.2.2 confparser.c

El módulo confparser.c se encarga de leer la configuración inicial de la jerarquía de memoria, tarea para la cual contiene varias funciones que a su vez hacen uso de la librería iniparser.

Además en este módulo encontramos los structs **struct CPU**, **structMemory** y **structCaché** que contienen los atributos de la CPU, la memoria y la caché respectivamente. Existe un array de structCaché con tantas posiciones como niveles de caché contenga la jerarquía de memoria. La variable global *numberCaches* indica cuantos niveles de caché hay.

6.2.3 datamanipulation.c

Este módulo contiene una serie de funciones que se encargan de realizar operaciones y conversiones de datos que son muy frecuentes en el programa. En particular se pueden resaltar las funciones que leen cadenas especificando números con multiplicador. Por ejemplo *parselongK1000* convierte la cadena "2k" en 2000, o *parseK1024*, que la convertiría en 2048. También hay funciones de comprobación de sintaxis, o condiciones especiales como ser múltiplo de 8 o potencia de 2.

6.2.4 datastore.c

Este módulo se encarga de la implementación de las estructuras de datos que utiliza internamente la aplicación para almacenar el contenido de los distintos niveles de la jerarquía de memoria, la traza de ejecución y las estadísticas de simulación. Las estructuras de datos utilizadas son específicas de la librería GTK+, cuyo uso es relativamente complejo, dada su versatilidad. El uso de estas estructuras de datos era imperativo para poder usar la representación gráfica de tablas en GTK+. Afortunadamente GTK+ separa sus estructuras de datos de la parte gráfica, lo que permitió utilizarlas incluso cuando el simulador trabajó en modo no gráfico.

6.2.5 datainterface.c

Este módulo cumple la función de ser una interfaz para el manejo de las estructuras de datos que internamente usa el programa para almacenar el contenido de la jerarquía de memoria abstrayendo las particularidades específicas que puedan tener. Esto se hizo especialmente pensando en facilitar el trabajo de futuros alumnos que deban realizar pequeñas tareas de programación complementarias al programa de simulación en el marco de una práctica. Para ello se ha creado un conjunto de funciones que realizan distintas operaciones de lectura y modificación.

Este módulo contiene las structs **cacheLine** y **memoryPosition**, las cuales almacenan los atributos propios de una línea de caché y de la posición de una palabra de memoria respectivamente. Así mismo, este módulo contiene todas las funciones necesarias para manejar estos tipos de dato. Muchas de estas funciones afectan también a la visualización. Por ejemplo, la función *readLineCache* accede a las estructuras de GTK+ para leer una línea de caché, pero también la muestra en verde y mueve la barra de scroll correspondiente para que el usuario la vea.

6.2.6 simulator.c

Este módulo es el encargado de simular las operaciones de acceso a memoria indicadas por el fichero de traza. Para ello, contiene una función que simula un único paso de la misma. Este fichero será el único que tenga que modificar el alumno para implementas aquellas funcionalidades que se soliciten en el guion de la práctica.

6.2.7 gui.c
Este módulo es el encargado de generar y gestionar la interfaz gráfica del programa. Para ello hace uso de la librería GTK+.

El módulo gui.c contiene una serie de funciones encargadas de la creación de todos los elementos de la interfaz, así como de otras para la manipulación de algunos de ellos. Permiten realizar acciones como desplazar la región visible de los distintos paneles, mostrar mensajes de error, o avanzar y obtener líneas del fichero de traza en tiempo de ejecución, ya que este es editable a través de una caja de texto de la interfaz.

Se puede observar que este módulo contiene además un gran número de variables. Estas se corresponden con los diferentes elementos de la interfaz del programa, todos ellos definidos en la libreríaGTK+.

6.2.8 callbacks.c
Este módulo se compone de un conjunto de funciones que se ejecutan a modo de respuesta cada vez que el usuario pulsa en uno de los botones de la interfaz. Estas son las acciones de simular todo, avanzar un paso, reiniciar y cerrar el programa.

6.2.9 mainprogram.c
Por último, el módulo main.c contiene la función main del programa. Se encarga de recoger los parámetros de la consola y llamar a las funciones correspondientes del resto de módulos para iniciar el programa. Es decir, hace las llamadas correspondientes para leer los ficheros de configuración y traza, generar las estructuras de datos correspondientes en base a la configuración leída y crear la interfaz gráfica en caso de que se esté ejecutando en ese modo.

7 Implementación

7.1 Tecnologías utilizadas

7.1.1 Lenguaje C

Tal como se ha mencionado anteriormente el programa desarrollado en este proyecto se ha codificado en lenguaje C.

Se trata de un lenguaje de programación de medio nivel originalmente desarrollado por Dennis Ritchie entre 1969 y 1972 en los Laboratorios Bell. Destaca por la eficiencia del código que produce. Es frecuente su uso para crear software de sistemas, aunque también se utiliza para desarrollar aplicaciones, como en este caso.

El uso de este lenguaje de programación es un requisito no funcional de este proyecto ya que así fue exigido por el cliente. Dado que uno de los objetivos planteados era que el código resultante pudiera ser modificado por un alumno en el marco de una práctica, es necesario que el código fuente se encuentre en un lenguaje ampliamente conocido y que ya resulte familiar a los alumnos.

7.1.2 Librería iniparser

Iniparser es una librería independiente y gratuita para lectura de fichero ini. Está escrita en ANSI C y fue desarrollada por N.Devillard. Se distribuye bajo una licencia MIT.

Con el objetivo de que se pudiera cargar de forma sencilla una configuración inicial para el sistema

de memoria a simular se decidió emplear ficheros de configuración .ini. Por este motivo, fue necesario escoger una librería de lectura en lenguaje C para este formato de ficheros, valorándose dos posibles opciones: Rwini y el ya mencionado iniparser.

Dado que la estructura de estos ficheros no es excesivamente compleja ambas librerías cumplían con los requisitos necesarios, permitiendo la lectura completa y correcta de este tipo de ficheros.

Finalmente se optó por el uso de iniparser, ya que su mecanismo de búsqueda es más directo, permitiendo obtener el valor buscado refiriéndose directamente a una sección y clave concreta, mientras que en Rwini esto tendría que hacerse en varios pasos, buscando primero la sección y a continuación la clave. Además, la librería iniparser viene acompañada de comentarios y documentación más extensos y detallados, así como de un ejemplo de uso.

7.1.3 Librería GTK+

The GIMP Toolkit, más conocido como GTK+ es un conjunto de bibliotecas multiplataforma del proyecto GNU utilizadas para el desarrollo de interfaces gráficas. Inicialmente la librería GTK+ fue creada para desarrollar GIMP, un software de edición de imagenes y fotografías . Desde entonces su uso se ha popularizado y en la actualidad es apliamente utilizado en interfaces de usuario de programas para los sietmas GNU/LINUX.

Se optó por utilizar esta tecnología por ser la única librería gráfica popular que está implementada en C. Se valoró la posibilidad de utilizar WxWidgets o Qt, pero estas están pensadas para C++.

7.2 Selección de widgets de la librería GTK+ para la interfaz de usuario

Dentro de las posibilidades que ofrece la librería GTK+ encontramos que existen diversas alternativas a la hora representar un mismo tipo de elementos, por lo que en determinadas ocasiones fue necesario evaluar las ventajas e inconvenientes de algunos widgets para escoger el que mejor se adaptase a la tarea a desempeñar.

En concreto, dado que el objetivo de este proyecto es implementar un simulador de memorias caché, fue necesario encontrar una forma adecuada de representar gráficamente los distintos niveles de la jerarquía de memoria, así como su estado o su contenido. Para ello se decidió utilizar una representación en forma de tabla en la que cada una de sus filas represente una posición de la memoria representada. Además, aspectos como disponer de varias columnas o poder colorear las casillas de la tabla fueron aspectos relevantes a valorar.

Teniendo en cuenta esto, se estudió que elementos de la librería GTK+ podían emplearse para este propósito, encontrándose las siguientes:

GtkListbox, un contenedor de widgets de la librería GTK+ que permite agrupar elementos componiendo una tabla vertical. Este elemento cumplía con los requisitos principales ya que permitía la colocación de cadenas de texto en diferentes filas. No obstante, se descartó por ser limitado en varios aspectos. Solo permite el uso de una columna, no admite encabezado de columna, no permite establecer un color de fondo a las casillas y no existe una línea de separación entre cada campo de la tabla.

GtkClist, un widget de la librería GTK+ que permite agrupar elementos en forma de tabla utilizando varias filas y columnas. Cumplía con los requisitos planteados y además ofrecía mayores

posibilidades que el GtkListbox. Este widget, a diferencia del valorado anteriormente, permite la agrupación de los datos en filas y columnas, lo que puede ser útil a la hora de separar una dirección de memoria en diferentes campos. Además, a diferencia del listbox, permite añadir encabezados a las columnas.
Como principal desventaja encontramos que, al igual que en el caso de listbox, no existen líneas de separación que delimiten visualmente los campos de la tabla.

GtkTreeView fue la tercera opción valorada. De nuevo, se trata de un widget que permite la representación de datos en forma de tabla. Permite la separación en filas y columnas, encabezados de columna. Además, permite establecer un color de fondo para dada casilla, lo que puede ser útil para resaltar esa posición de memoria indicando visualmente alguna acción que se esté realizando en ella o dar alguna información sobre el estado en que se encuentra. Este widget, a diferencia del Clist, representa visualmente la separación de filas y columnas utilizando líneas. Es por todas estas condiciones que se optó por este widget para la implementación de tablas en la interfaz gráfica del programa. Cabe destacar que para su uso es necesario emplear una estructura de datos que contenga la información de la tabla. Para ello, se valoraron las dos posibilidades que ofrece la librería GTK+.

7.3 Selección de estructuras de datos de la librería GTK+

En concreto, en la librería GTK+ existen dos estructuras de datos que pueden asociarse a un GtkTreeView para almacenar los diferentes datos de la aplicación que se mostrarán en las tablas:

GtkTreeStore. Esta estructura de datos permite el almacenamiento de la información de la tabla en una estructura en forma de árbol. Esto, en combinación con el widget GtkTreeView permite desplegar filas adicionales de información a partir de una fila de la tabla. Es decir, desplegar tablas anidadas dentro de la tabla inicial. Esta es una funcionalidad interesante para la implementación del panel de estadísticas, el cual debía ser desplegable, por lo que se utilizó para esta cuestión.

GtkListStore . Esta estructura de datos se asemeja más a una lista. Fue utilizada para almacenar el contenido de las memorias caché y la memoria principal. En combinación con el widget GtkTreeView nos permite mostrar el contenido de todas las memorias en forma de tabla.

7.4 Dificultades encontradas y soluciones aplicadas

A lo largo de la fase de implementación surgieron problemas que inicialmente no se habías previsto. Los más relevantes junto con la solución aplicada se exponen a continuación.

7.4.1 Problemas de rendimiento para memorias de gran tamaño

Una vez implementada la interfaz gráfica de usuario y conseguido que esta se genere de acuerdo a los parámetros leídos desde el fichero de configuración, se observó una gran pérdida de rendimiento para memorias de gran tamaño.

Inicialmente se había previsto que la tabla mostrada en la interfaz representara el contenido completo de la memoria. No obstante, una vez implementado de esta forma y realizadas las primeras pruebas hubo que descartar esta posibilidad al observarse que en el caso de memorias de tamaño superior a unos pocos megabytes la interfaz no llegaba a generarse o tardaba mucho tiempo. Teniendo en cuenta esto sería completamente imposible simular sistemas con memorias con una capacidad de varios

Gigabytes, similares a las de los computadores actuales.

Dado que este es un simulador orientado al aprendizaje de los alumnos, las simulaciones a realizar nunca requerirán una gran cantidad de posiciones de memoria para funcionar.
Es por este motivo que, tras consultar con el cliente sobre el problema detectado, se acordó representar en la interfaz gráfica de usuario únicamente una página de la memoria, independientemente del tamaño total de esta. Teniendo en cuenta que una página de memoria puede tener una capacidad de varios Kilobytes, se consideró que esto sería más que suficiente para cualquier simulación que pudiera realizar un alumno en el contexto de una práctica.

7.4.2 Excesiva complejidad de las estructuras de datos empleadas por la librería GTK+

Si observamos el modelo de dominio que se estudió durante la fase de diseño del proyecto nos damos cuenta que cada caché se compone de una serie de líneas de caché y que la memoria principal se compone de una serie de posiciones de memoria.

Teniendo en cuenta esto, la implementación más directa de las memorias caché y la memoria principal en lenguaje C hubiera sido mediante arrays de structs que representen las líneas de caché y las posiciones de memoria respectivamente.

No obstante, tal como se ha explicado en un apartado anterior de la implementación, para utilizar el widget GtkTreeView de la librería GTK+ hay utilizar necesariamente las estructuras datos GtkTreeStore o GtkListStore de esta misma librería. De lo contrario no puede funcionar. El problema que esto conlleva es que el uso de estas estructuras de datos es excesivamente complejo en comparación con la forma más intuitiva y directa de almacenar los datos que ya se ha expuesto.

GtkListStore puede entenderse como una matriz con filas y columnas, mientras que GtkTreeStore se asemeja a un árbol de matrices. En ambas estructuras para realizar accesos de lectura, escritura o modificación es necesario el uso de iteradores para alcanzar la posición de la estructura de datos sobre la que se quiere actuar. Para ello es necesario saber utilizar un gran número de funciones de la librería GTK+.

Si bien esto puede ser asumible desde el punto de vista de la implementación del simulador, el mayor problema surge al pensar en la posterior manipulación del código por parte de los alumnos. Tal como se expuso en la fase de captura de requisitos, se pretende que los alumnos puedan realizar pequeñas modificaciones o programar módulos adicionales en el marco de una práctica.
Si para realizar estas tareas el alumno debe estudiar el funcionamiento de estructuras de datos complejas esto entorpece enormemente el desarrollo de la práctica.

Como solución a este problema se decidió incluir un módulo adicional en el programa que hiciera de interfaz de programación para los alumnos. Dicha API consta de una serie de funciones que realizan las operaciones de escritura y lectura de las estructuras de datos internas abstrayendo la complejidad interna de su funcionamiento.

De esta forma, desde el punto de vista del usuario de la API, los datos parecen almacenados en la forma intuitiva de arrays de structs que se planificaba utilizar en un principio.

8 Pruebas

Dentro de cualquier proyecto software es fundamental asegurar en la medida de lo posible un correcto funcionamiento del software desarrollado. Es por este motivo que paralelamente al desarrollo de la fase de implementación se llevó a cabo una fase de pruebas en la cual se aplicaron diferentes técnicas.

En primer lugar, se llevaron a cabo pruebas unitarias para comprobar los diferentes módulos del programa a media que estos eran desarrollados. Para cada uno de ellos se elaboraron test que comprobaran el correcto funcionamiento de cada una de sus funciones.

Más adelante, se realizaron pruebas de integración para verificar el correcto funcionamiento en interacción entre los módulos del programa una vez que estos habían sido comprobados por separado e integrados unos con otros hasta agrupar todas las funcionalidades del sistema.

Para todas las pruebas realizadas se tuvo en cuenta el nivel de cobertura del código comprobado mediante el uso de la herramienta de análisis de cobertura de código fuente **Gcov**.

En el caso de los módulos confparser.c y traceparser.c, dado que ambos leen parámetros introducidos directamente por el usuario se prestó especial atención a elaborar test especialmente detallados, utilizando técnicas de pruebas de caja negra. En ellos se identificaron todos los posibles valores que podrían tomar los parámetros de configuración y se elaboró una batería de casos de prueba que contemplara todos ellos. De esta forma se detectaron todos los casos de error que la aplicación debería tener en cuenta en caso de que el usuario introduzca un valor no válido.

Además, también se utilizó **Radamsa,** un programa que permite introducir pequeñas modificaciones aleatorias en un fichero de configuración válido para poder detectar posibles problemas en el programa que no se habían previsto inicialmente. Para ello es necesario someter al módulo de lectura a una gran cantidad de ficheros de configuración diferentes.

9 Documentación

Tal como se estableció en la planificación inicial del proyecto paralelamente a la fase de implementación se llevó a cabo una fase de documentación del código desarrollado.

Como parte imprescindible de este proceso, al igual que se hace en cualquier proyecto software, se detalló el funcionamiento de cada una de las partes que componen el código de la aplicación desarrollada. Para cada función se describió su propósito, los parámetros de entrada y posibles valores de cada uno de ellos, así como los posibles valores retornados. Toda esta información fue además incluida en el código en forma de comentarios. De esta manera, futuros programadores que deban hacer modificaciones sobre dicho código podrán consultar esta documentación para comprenderlo con mayor facilidad. Especialmente en el caso de los alumnos, que aprenderán a utilizar la API de gestión de datos y completarán tareas de programación sobre el simulador disponiendo del tiempo limitado propio de una sesión de prácticas.

Además, como ayuda al futuro usuario de la aplicación se crearon manuales que describen la estructura y las reglas que deben respetar el fichero de configuración (*Anexo I*) y el fichero de traza (*Anexo II*) para ser válidos desde el punto de vista del simulador.

10 Descripción de la interfaz de programación

Como se ha explicado en el apartado de implementación, el simulador utiliza unas estructuras de datos bastante complejas de gestionar y utilizar debido a que la librería GTK+ así lo requiere.
Es por ello que fue necesario incluir en el diseño de la aplicación el módulo datainterface.c, a modo de interfaz de programación para acceder a los datos abstrayendo la complejidad de forma que los que los futuros alumnos puedan acceder a los datos sin tener conocimiento de cómo están almacenados internamente.

En este apartado se explicarán las diferentes funciones que componen esta interfaz de programación, así como un ejemplo de uso de la misma.

10.1 Funciones que componen la API

La interfaz de programación engloba todas las operaciones básicas de acceso a los datos, permitiendo realizar lecturas y modificaciones de las líneas de caché y las posiciones de memoria.

10.1.1 Funciones que actúan sobre el contenido de las memorias caché

Desde el punto de vista del alumno, una línea de caché se representa mediante un struct que almacena cada uno de sus atributos, de la forma que se ve a continuación. Entre ellos el campo user_content que permite al usuario almacenar su propio contenido.

```
struct cacheLine{
 unsigned line;
 unsigned tag;
 unsigned set;
 unsigned * content;
 void * user_content;
 unsigned valid;
 unsigned dirty;
 unsigned lastAccess;
 unsigned accessCount;
 unsigned firstAccess;
};
```

Todas las funciones de la API que realizan una lectura o escritura del contenido de una línea de caché emplean una estructura de este tipo para almacenar los atributos de esa línea. Para referirse a la línea concreta a sobre la que se quiere actuar es necesario indicar por parámetro en nivel de la jerarquía y el número de línea (a no ser que se trate de una función que actúe sobre todas las líneas de la caché).

Para muchas de las funciones de la API existen habitualmente tres variantes, según se trate de una caché unificada, una caché de datos o una caché de instrucciones. A continuación, se muestran las funciones disponibles para realizar la lectura de una línea de caché. Se pasa por parámetro el struct donde se va a almacenar la línea leída.

```
void readLineCache (int level, struct cacheLine* line, int i);
void readLineCacheData (int level, struct cacheLine* line, int i);
void readLineCacheInstructions (int level, struct cacheLine* line, int i);
```

Por el contrario, si se quiere escribir el contenido del struct en la línea de caché se debe utilizar una de las siguientes funciones:

```
void writeLineCache (int level, struct cacheLine* line, int i);
void writeLineCacheData (int level, struct cacheLine* line, int i);
void  writeLineCacheInstructions  (int  level,  struct  cacheLine*  line,  int  i);
```

Además, de las operaciones de lectura y escritura también encontramos funciones que permiten mostrar el contenido de la línea por consola.

```
void showCacheLine (int level, int i);
void showCacheLineData (int level, int i);
void showCacheLineInstructions (int level, int i);
```

Funciones que permiten resetear líneas de caché a su valor inicial de la simulación o incluso la memoria caché completa.

```
void writeBlankCacheLine (int level, unsigned line);
void writeBlankInstructionCacheLine (int level, unsigned line);
void writeBlankDataCacheLine (int level, unsigned line);
void resetCaché (int level);
```

Funciones que permiten modificar el color de la línea de caché. Estas solo tienen sentido cuando se está ejecutando la simulación en modo gráfico. De lo contrario no tienen ningún efecto.

```
void setColorCacheLine (int level, int i, int color);
void setColorDataCacheLine (int level, int i, int color);
void setColorInstructionsCacheLine (int level, int i, int color);
```

Por último, encontramos dos funciones que realizan tareas específicas. La primera se encarga de buscar el número de línea en el que está almacenada la etiqueta que se indica por parámetro.

```
long findTagInCache (int level, unsigned set, unsigned requestTag);
```

La segunda función permite leer únicamente las flags de lalínea de caché en lugar de obtener su contenido completo cuando no es necesario.

```
void readFlagsCacheData (int level, struct cacheLine* line, int i);
```

10.1.2 Funciones que actúan sobre el contenido de la memoria principal

Desde el punto de vista del alumno, una palabra de la memoria se representa mediante un struct que almacena su dirección y contenido, así como el campo user_content que permite al usuario almacenar su propio contenido.

```
struct memoryPosition{
 long address;
 long content;
 void * user_content;
};
```

En el caso de las funciones que actúan sobre una palabra de la memoria es necesario indicar en sus parámetros la dirección de la palabra sobre la que se va a actuar. A continuación, se muestran todas las funciones disponibles para manipular el contenido de la memoria principal:

Para realizar la lectura/escritura de una palabra disponemos de las siguientes funciones.

```
int readMemoryAddress (struct memoryPosition *pos, long address);
```

```
int writeMemoryAddress (struct memoryPosition *pos, long address);
```

Para mostrar por pantalla en contenido de una posición de memoria disponemos de la siguiente función.

```
int showMemoryAddress (long address);
```

Para colorear la posición de una palabra de la memoria disponemos de la función siguiente. Si se llama a esta función en el modo no gráfico no tiene ningún efecto.

```
int setMemoryAddressColor (long address, int color);
```

Por último, las siguientes funciones permiten resetear todo el contenido de la memoria o posiciones concretas de esta.

```
void resetMemory ();
void clearMemoryAddress (long address);
```

10.1.3 Funciones que actúan sobre las estadísticas de simulación

El simulador de cachés almacena su propio registro de estadísticas y en caso de encontrarse en modo gráfico las muestra en un panel desplegable en la pantalla.
Para poder acceder a los datos de las estadísticas de forma sencilla se han incluido varias funciones en la API.

Desde el punto de vista del usuario las estadísticas se agrupan en apartados dentro de los cuales hay atributos y cada uno de ellos tiene un valor. Cuando se desea leer o escribir alguno de los valores almacenados es necesario indicar el nombre del apartado y el atributo, así como el valor a almacenar, todos ellos en forma de texto.

Las siguientes funciones permiten almacenar y leer las estadísticas.

```
void setStatistics (char* component, char* property, char* value);
char* getStatistics (char* component, char* property);
```

La siguiente función muestra todas las estadísticas por pantalla.

```
void printStatistics (FILE* fp);
```

10.1.4 Funciones que actúan sobre toda la jerarquía de memoria

Por último, existe una función de la API que actúa simultanemamente sobre toda la jerarquía de memoria. Dicha función permite eliminar todos los colores que se hayan establecido en las cachés o la memoria principal. Solo tine efecto cuando se ejecuta en modo gráfico.

```
void removeAllColors ();
```

10.2 Ejemplo de uso de la interfaz de programación

Como apoyo a la explicación de la interfaz de programación que se ha construido se ha preparado un pequeño ejemplo de uso similar a una tarea que debería desempeñar un alumno en durante una práctica. Se trata de implementar el algoritmo de reemplazo LFU, en este caso mediante una función que indica cual es la línea a reemplazar (la que ha sido accedida un menor número de veces). Como se puede observar, en la función siguiente se realizan diversas llamadas a funciones de la API.

```
long getLessFrecuentlyUsedLine (int cachéLevel){
	struct cacheLine currentLine;
	minAccesses= INT_MAX;
	lessAccessedLine=0;
	//recorro todas las líneas de la caché
	for (int i=0; i<numLines; i++){
		//leo la línea actual
		readLineCaché (int cachéLevel, &currentLine, i);
		//compruebo el número de accesos a la línea actual
		if (currentLine.accessCount<minAccesses && currentLine.valid==1){
			minAccesses= currentLine.accessCount;
			lessAccessedLine= currentLine.line;
		}
	}
	//retorno la línea menos accedida
	return lessAccessedLine;
}
```

11 El producto final

Una vez explicados todos los aspectos relevantes en lo relativo al desarrollo de cada una de las fases del proyecto software, a continuación se mostrará el producto final obtenido. Para ello se ilustrará su funcionamiento a través de un pequeño ejemplo básico de simulación de cachés.

En primer lugar, tal como ya se explicó en la fase de requisitos al comienzo de esta memoria, el simulador requiere de una configuración inicial que es proporcionada al programa a través de un fichero de configuración de formato INI. En ese fichero se establecen todas las propiedades de la jerarquía de memoria que vamos a simular.

En este ejemplo, hemos optado por la configuración descrita en el fichero de la Figura 6. Se trata de un sistema con una jerarquía de caché compuesta de dos niveles, siendo la caché L1 separada en datos e instrucciones y la caché L2 unificada. En el fichero se incluyen una serie de parámetros de configuración para ambos niveles de caché, además de los parámetros correspondiente para la CPU y la memoria, elementos necesarios en cualquier sistema.

Por otra parte, como también se había establecido en la fase de requisitos, el programa debe realizar una lectura de la traza de operaciones de memoria a simular desde un fichero. En este caso, hemos decidido utilizar la traza de operaciones contenida en el fichero de la Figura 7. En ese fichero se indica al simulador que debe realizar la escritura de varios valores en una serie de posiciones de memoria contiguas. A continuación, la traza de operaciones de memoria indica que deben leerse de nuevo las mismas posiciones en las que anteriormente se había escrito. Esto permitirá observar en la simulación cómo son llevados los bloques de memoria a las líneas de caché produciéndose un error al acceder a la primera palabra del boque y un acierto en las siguientes.

```
[cpu]
word_width = 32
address_width = 32
frequency = 1G
trace_file = stride.vca

[cache1]
line_size = 16
size = 2K
asociativity = 4
write_policy = wt
replacement = lru
separated = 1
access_time = 6
column_bit_mask= 1111111111

[cache2]
line_size = 16
size = 4K
asociativity = 4
write_policy = wt
replacement = lru
separated = 0
access_time = 6
column_bit_mask= 1111111111

[memory]
size = 2G
access_time_1 = 50m
access_time_burst = 10n
page_base_address = 0x80000000
page_size= 8k
```

Figura 6. Fichero de configuración

```
S 0x80000000 D 4 19
S 0x80000004 D 4 29
S 0x80000008 D 4 39
S 0x8000000c D 4 49
S 0x80000010 D 4 59
S 0x80000014 D 4 69
S 0x80000018 D 4 79
S 0x8000001c D 4 89
S 0x80000020 D 4 99
S 0x80000024 D 4 89
S 0x80000028 D 4 99
S 0x8000002c D 4 09
S 0x80000030 D 4 19
S 0x80000034 D 4 29
S 0x80000038 D 4 39
S 0x8000003c D 4 49
L 0x80000000 D 4
L 0x80000004 D 4
L 0x80000008 D 4
L 0x8000000c D 4
L 0x80000010 D 4
L 0x80000014 D 4
L 0x80000018 D 4
L 0x8000001c D 4
L 0x80000020 D 4
L 0x80000024 D 4
L 0x80000028 D 4
L 0x8000002c D 4
L 0x80000030 D 4
L 0x80000034 D 4
L 0x80000038 D 4
L 0x8000003c D 4
```

Figura 7. Fichero de traza

Una vez iniciamos el programa, este leerá el fichero de configuración y generará la interfaz gráfica de usuario de acuerdo a los parámetros de configuración establecidos. En este caso, en la Figura 8 se puede ver como se ha generado una tabla para cada memoria caché que se estableció en la configuración. De la misma forma, su número de filas y columnas, así como los datos que se muestran en ellas también se corresponden con la configuración establecida para la caché que cada tabla está representando.

Por otra parte, el fichero de traza también ha sido leído por el simulador y se muestra en el cuadro de texto en la esquina superior izquierda de la interfaz, resaltando en azul la siguiente línea de traza a ejecutar.

Es este caso, la figura 8 muestra un momento intermedio de la simulación en el que la última línea de traza ejecutada ha sido L 0x80000004 D 4, es decir una lectura de una palabra de 32 bits en la posición de memoria 0x80000004. En dicha operación de lectura se ha producido un fallo en las cachés de nivel 1 y 2, por lo que se ha accedido a la memoria principal y se ha leído el bloque completo de palabras a llevar al nivel superior de la jerarquía (líneas en verde). A continuación han sido escritos en una línea en la caché de nivel 2 y luego en la caché de nivel 1(líneas en rojo).

Por último, cabe destacar como en la esquina inferior izquierda de la interfaz se muestran las estadísticas de la simulación. En el caso mostrado en la Figura 8 podemos ver el total de accesos realizados y el tiempo total de los accesos acumulado en ese momento de la simulación.

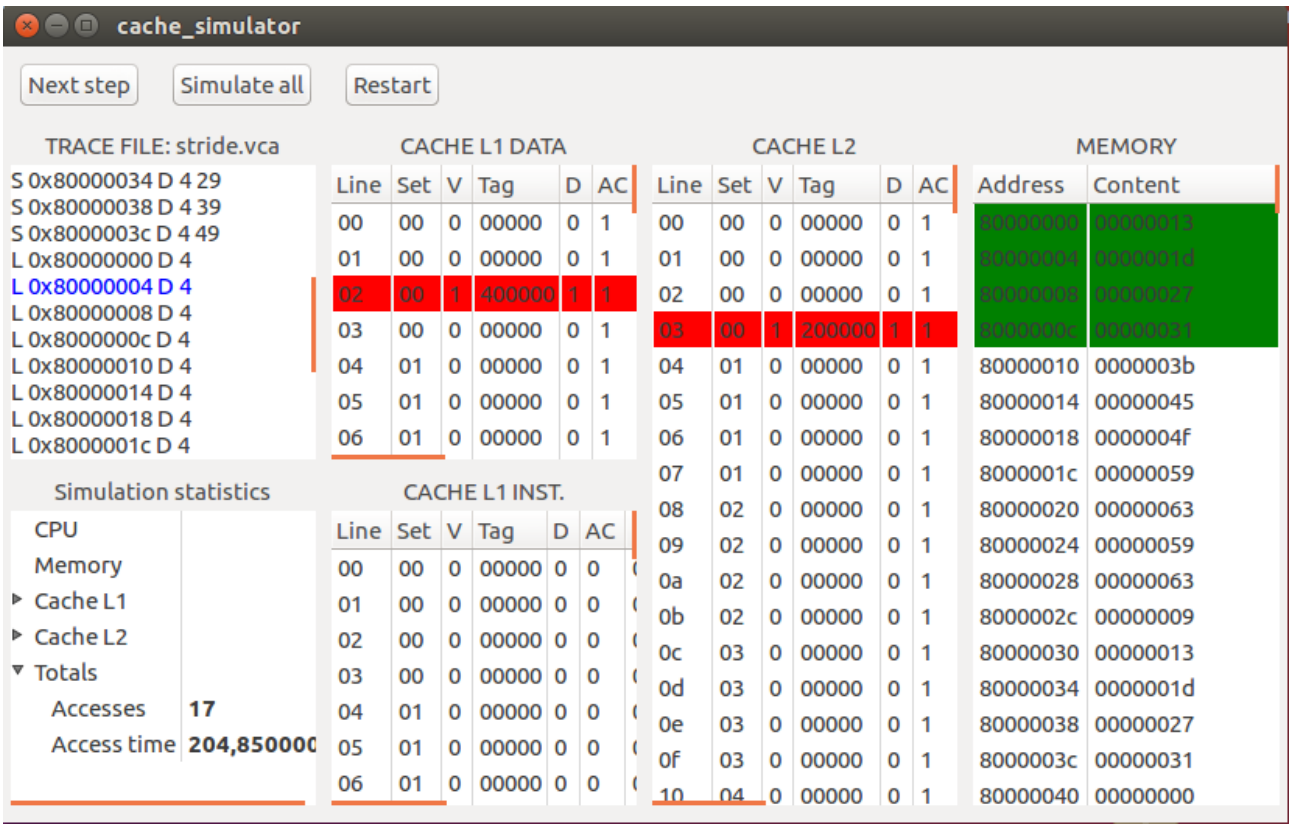

Figura 8: interfaz gráfica del programa

12 Conclusiones

Después de haber concluido la realización de este proyecto, considero que se han alcanzado todos los objetivos previstos. La aplicación desarrollada cumple con los requisitos establecidos en un comienzo. Se ha logrado crear un simulador que muestra visualmente el funcionamiento de una caché multinivel y que podrá ser utilizado por los alumnos en el proceso de aprendizaje.

En el aspecto personal, valoro positivamente la experiencia y los conocimientos adquiridos tanto en lo que se refiere a al diseño e implementación del software desarrollado como en lo referente a todos aquellos conocimientos relativos a la jerarquía de memoria que he tenido que recordar y estudiar de nuevo para entender los aspectos que el simulador tendría que representar.

En lo referente al diseño de la aplicación este proyecto me ha permitido aplicar y reforzar los conocimientos adquiridos al cursar las asignaturas de ingeniería de software. Esto es algo a valorar especialmente desde el punto de vista del alumno que haya cursado la mención de computadores y por tanto tiene una formación más limitada en este aspecto.

Respecto a la implementación de la aplicación, me ha permitido profundizar en aspectos del lenguaje C que hasta ahora había evitado por no ser necesarios muchas veces en los pequeños programas que se hacen en las prácticas de muchas asignaturas. Este es el caso de los punteros, los structs, o la reserva de memoria dinámicamente, cuestiones que si bien conocía no utilizaba con mucha frecuencia. Al tener que construir una aplicación de mayor complejidad he aprendido a valorar la importancia de estos aspectos y a utilizarlos con mayor precisión que hasta ahora.

Por otra parte, he adquirido ciertos conocimientos sobre interfaces gráficas, una cuestión que no se trata en profundidad durante la carrera y de la que ahora puedo saber un poco más. Aunque en un

principio, la librería GTK+ fue algo complejo de comprender y requirió de horas de estudio y la realización de muchos pequeños programas de prueba, finalmente he logrado manejar todos los aspectos necesarios e interiorizar ciertos conceptos que sin duda podrá extrapolarse al uso de otras librerías y lenguajes.

Por último, valoro especialmente cómo este proyecto me ha ayudado a comprender con precisión muchos aspectos de arquitectura de computadores relativos a la jerarquía de memoria que en su momento no había llegado a entender por completo o que ya había olvidado parcialmente. Tal como se explicó al comienzo de este documento, la jerarquía de memoria es algo complejo para el estudiante y lleno de conceptos que pasarían desapercibidos si no fuera por los simuladores que les permiten observar e interiorizar todas esas cuestiones. Por este motivo, el diseño y construcción del propio simulador requiere un conocimiento aún mayor de todos los aspectos de la jerarquía de memoria y su funcionamiento.

13 Trabajos futuros

Aunque el simulador ya contiene todas las funciones básicas y es apto y completo desde el punto de vista del aprendizaje de los alumnos, aún existen formas de mejorar el simulador de cara a futuras versiones, ya sea añadiendo nuevas funciones o ampliando las posibilidades de simulación a otros niveles de la jerarquía de memoria.

En lo referente a nuevas funciones del programa cabe destacar la posibilidad de incluir una opción para que el usuario pueda deshacer un paso de la simulación. Esto es algo a valorar, ya que permitiría retroceder un paso en caso de que se quiera revisar pasos anteriores de la simulación.

De la misma forma también sería posible incluir una opción para guardar el contenido de la traza en un fichero. Actualmente, si bien puede editarse la traza desde el propio simulador no pueden guardarse los cambios. Esto es debido a que la edición se permite fundamentalmente para la inserción de puntos de ruptura. Si se quiere editar el fichero de forma permanente es necesario hacerlo con un editor externo.

Por último, existe la posibilidad de expandir el simulador hacia otros niveles de la jerarquía de memoria, en concreto en lo relativo a la memoria virtual. De hecho, el simulador actual maneja ya el concepto de “página”, en previsión de una posible ampliación en este sentido.

14 Bibliografía

[1] D. Patterson, J. Hennessy, (2014) “Computer Organization and Design”, 5ªEdición, Ed. Morgan Kaufmann.

[2] Sommerville, (2012) "Ingeniería del Software” 9ª Edición, Addison-Wesley

[3] Alfredo González Naranjo, Carlos Manuel Vaquero Rivas (curso 1998/1999) Visual Caché. Facultad de informática de la Universida de Sevilla.

[4] GTK+. (10 de 05 de 2018). Obtenido de gtk.org:
https://www.gtk.org/

[5] Iniparser. (10 de 05 de 2018). Obtenido de ndevilla.free.fr:
http://ndevilla.free.fr/iniparser/

[6] Gcov. (10 de 05 de 2018). Obtenido de gcc.gnu.org:
https://gcc.gnu.org/onlinedocs/gcc/Gcov.html

[7] Radamsa. (10 de 05 de 2018). Obtenido de github.com:
https://gitlab.com/akihe/radamsa

[8] Magic Draw (10 de 05 de 2018)). Obtenido de nomagic.com
https://www.nomagic.com/products/magicdraw

[9] Mars Mips. (10 de 05 de 2018). Obtenido de courses.missouristate.edu:
http://courses.missouristate.edu/KenVollmar/mars/

Anexo I: Manual del fichero de configuración

SINTAXIS DEL FORMATO INI

El fichero de configuración permite establecer los parámetros de configuración de la simulación de cachés. Emplea el formato .ini en el cual los campos se expresan de la forma: 'clave=valor'

Las claves se agrupan en secciones. Cada sección agrupa las claves ubicadas entre su etiqueta de sección y la etiqueta de la sección siguiente, o el final del fichero.

Las etiquetas de sección se expresan de la forma '[etiqueta]'.

Los comentarios se indican mediante el carácter';' o '#'

SECCIONES OBLIGATORIAS DEL FICHERO DE CONFIGURACIÓN

El fichero de configuración debe contener las siguientes secciones obligatorias:

SECCION	CAMPOS	DESCRIPCIÓN
[cpu] Esta sección agrupa los parámetros de configuración de la CPU	word_width	Número de bits en la palabra del procesador.
	address_width	Número de bits en la dirección del procesador.
	Frequency	Frecuencia en hercios. Ver Nota 1.
	trace_file	Fichero de traza.
[memory] Esta sección agrupa los parámetros de configuración de la memoria.	size	Tamaño de la memoria en bytes. Ver Nota 2.
	access_time_1	Tiempo de acceso en segundos. Ver Nota 5.
	access_time_burst	Tiempo de acceso en ráfaga. Ver nota 5.
	page_base_address	Dirección base de la página.
	page_size	Tamaño de página. Ver Nota 2.

SECCIONES OPCIONALES DEL FICHERO DE CONFIGURACIÓN

Además, el fichero puede contener n secciones opcionales [cache] que representan las caches de la jerarquía de memoria. El nombre de cada sección se compone de la palabra "cache " y un número que indica su posición en la jerarquía de memoria. De esta forma la cache de nivel L1 será [cache1], la cache L2 será [cache2], etc.

Cada sección cache se compone de los siguientes campos:

SECCION	CAMPOS	DESCRIPCIÓN
[cacheN] Esta sección agrupa los parámetros de configuración de la cache N	line_size	Tamaño de línea en bytes. Ver Nota 2.
	size	Tamaño cache en bytes. Ver Nota 2.
	asociativity	Asociatividad. Ver nota 6
	write_policy	Política de escritura. Ver nota 7
	replacement	Política de reemplazo: lru, rand
	separated	Si la memoria cache es separada. Nota 3.
	access_time	tiempo de acceso. Ver nota 5.
	column_bit_mask	Máscara binaria de columnas a mostrar

NOTAS

Nota 1: Puede llevar un multiplicador G para 1e9, M para 1e6 o K para 1e3. Cualquier otro carácter dará un error.
Nota 2: Puede llevar un multiplicador G para 2^30, M para 2^20 o K para 2^10. Cualquier otro carácter dará un error.
Nota 3: Un valor booleano, deberá aceptar 1 o 0, true o false, yes o no, en mayúsculas y minúsculas.
Nota 4: Tiene que ser una 'l', mayúscula o minúscula, seguida de un entero.
Nota 5: Puede llevar un multiplicador p para 1e-12, n para 1e-9, u para 1e-6, m para 1e-3
Nota 6: 1 = Mapeo directo, F = totalmente asociativa, cualquier otro número potencia de dos.
Nota 7: wt = write through, wb = write back

Anexo II: Manual del fichero de traza

ESTRUCTURA DEL FICHERO DE TRAZA

El fichero de traza representa las operaciones de memoria que se van a simular.
Cada operación es representada en una línea. No puede haber más de una operación por línea. Se ejecutaran en el orden en que se encuentren el fichero.

Las líneas en banco o que solo contienen comentarios se ignoran. Los comentarios se representan con el carácter '#'.

SINTAXIS DE UNA LÍNEA DE TRAZA

Cada operación se representa mediante 5 campos separados por espacios o tabuladores y escritos en el orden siguiente.

[**breakpoint**] [**tipo de acceso**] [**dirección**] [**tipo de operación**] [**tamaño**] [**dato**]

breakpoint: Este campo es opcional. Únicamente puede tomar el valor '!'. Sirve para indicar que existe un punto de ruptura en esta operación.

tipo de acceso: Este campo es obligatorio. Puede tomar el valor 'I' para indicar el acceso a una instrucción o el valor D para indicar el acceso a un dato.

dirección: Este campo es obligatorio. Debe ser un valor hexadecimal comprendido en la página indicada en el fichero de configuración.

tipo de operación: Este campo es obligatorio. Puede tomar el valor 'S' para indicar que se trata de un store o el valor 'L' para indicar que se trata de un load.

tamaño: Este campo es opcional. Indica el tamaño que tiene el dato a acceder.

dato: Este campo es opcional. Solo puede utilizarse conjuntamente con el campo tamaño. Indica el dato a almacenar.