(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 381 961**

(21) Número de solicitud: 201001284

(51) Int. Cl.:

G06F 9/455 (2006.01)
G06F 12/10 (2006.01)
G06F 11/34 (2006.01)

(12) SOLICITUD DE PATENTE A1

(22) Fecha de presentación: **30.09.2010**

(43) Fecha de publicación de la solicitud: **04.06.2012**

(43) Fecha de publicación del folleto de la solicitud: **04.06.2012**

(71) Solicitante/s: **Universidad de Cantabria**
Pabellón de Gobierno
Avda. de los Castros, s/n
39005 Santander, Cantabria, ES

(72) Inventor/es: **Posadas Cobo, Héctor;**
Villar Bonet, Eugenio y
Díaz Suárez, Luis

(74) Agente/Representante:
No consta

(54) Título: **Método y sistema de modelado de memoria cache.**

(57) Resumen:
Método y sistema de modelado de memoria cache.
Un método de modelado de una memoria cache de datos de un procesador destino, para simular el comportamiento de dicha memoria cache de datos en la ejecución de un código software en una plataforma que comprenda dicho procesador destino, donde dicha simulación se realiza en una plataforma nativa que tiene un procesador diferente del procesador destino que comprende dicha memoria cache de datos que se va a modelar, donde dicho modelado se realiza mediante la ejecución en dicha plataforma nativa de un código software que se basa en dicho código software a ejecutar en dicha plataforma destino, extendido con información para modelar dicho comportamiento de dicha memoria cache de datos del procesador destino, donde el método comprende las etapas de: analizar (102) el código software a ejecutar en la plataforma destino (101) para identificar unos bloques básicos (104) de dicho código y una pluralidad de variables accedidas en cada bloque; añadir (106) a dicho código anotaciones relativas a la memoria cache de datos a simular, donde dichas anotaciones comprenden información para modelar el efecto de dicha memoria en el procesador destino, obteniéndose un código anotado (107); compilar (108) dicho código anotado; ejecutar (109) dicho código anotado compilado junto con un modelo hardware de dicha memoria cache de datos. La etapa de añadir (106) a dicho código anotaciones relativas a la memoria cache de datos a simular comprende añadir información que permite obtener las direcciones de las variables que dicha memoria cache de datos simulada debe acceder, para así estimar si cada acceso a dichas variables resulta en un acierto o en un fallo de memoria cache de datos.

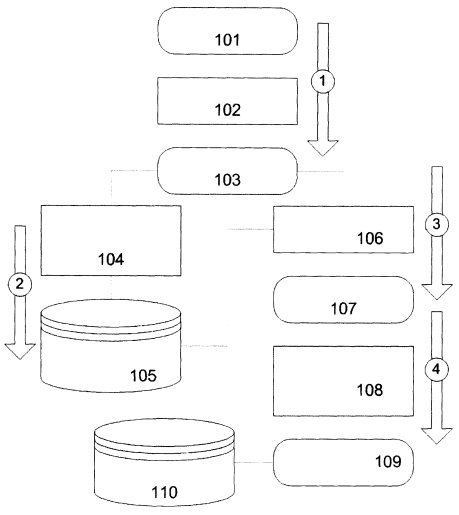

FIGURA 1

ES 2 381 961 A1

DESCRIPCIÓN

Método y sistema de modelado de memoria cache.

Campo de la invención

La presente invención pertenece a los campos de la electrónica y la informática. Más concretamente, la invención pertenece a las áreas de diseño de procesadores y sistemas que contienen elementos procesadores, incluyendo tanto desarrollo de programas (software, SW) como de la plataforma física sobre la que se ejecuta el código (hardware, HW). La invención es especialmente relevante en aquellos ámbitos donde el tiempo de ejecución y/o el consumo son importantes.

Antecedentes de la invención

Un micro-procesador en un componente electrónico diseñado para ejecutar código. En general un procesador está diseñado de tal forma que el número de instrucciones ejecutadas sea similar a su frecuencia. Por ejemplo, un procesador a 2 Gigahertzios ejecuta en torno a dos mil millones de instrucciones por segundo. Sin embargo, para poder ejecutar, el procesador necesita saber qué tiene que hacer (instrucciones) y con qué valores ha de operar (datos).

Esta información (instrucciones y datos) se guarda en una memoria, de tal forma que el procesador ha de acceder a esa memoria para obtener cada instrucción y los datos asociados, y guardar el resultado de la ejecución. Dado que el acceso a las memorias es, por lo general, mucho más lento que la velocidad de ejecución del procesador, el procesador ha de estar la mayor parte del tiempo parado esperando a las memorias, por lo que la velocidad del sistema se reduce enormemente. Además, el acceso a las memorias incrementa el consumo y entorpece el funcionamiento de otros elementos del sistema que utilicen las memorias y/o los canales de comunicación del sistema, al no permitir estos elementos la realización de accesos simultáneos.

Para evitar estas ineficiencias, la solución convencional es colocar una o varias memorias junto al procesador. Estas memorias son mucho más pequeñas y más rápidas que la memoria principal. Los datos e instrucciones se mantienen en dichas memorias durante su uso, reduciendo drásticamente el número de accesos a la memoria principal, y acelerando el sistema. Estas memorias se llaman habitualmente memorias cache o caches.

El funcionamiento es el siguiente: cada vez que el procesador requiere información (instrucción o dato), realiza una petición al exterior: envía la dirección de memoria donde está la instrucción o el dato y si hay que leer o escribir. Si hay cache, esta está conectada al procesador, recibe la petición y comprueba si tiene la información. En caso afirmativo responde al procesador. Si no, realiza una petición a la memoria principal, espera a que responda y a su vez responde al procesador.

El desarrollo de estos sistemas, tanto parte de la plataforma física, como el software, se realiza utilizando inicialmente simulaciones para poder crear finalmente el sistema real con el menor coste posible. Para realizar estas simulaciones, es necesario que el ordenador donde se realizan las simulaciones contenga una infraestructura que modele el comportamiento esperado del sistema real. Esto implica estimar cuál será el comportamiento del sistema real, e introducir esta información en la simulación.

Dado el enorme impacto de las caches en el rendimiento del sistema, es fundamental tenerlas en cuenta a la hora de tratar de mejorar la eficiencia del procesador y del sistema en general. En consecuencia, los efectos de las caches se han tenido en cuenta en una amplia variedad de intentos de estimar el rendimiento del sistema, como por ejemplo por M. Lajolo y otros, en “Fast instruction cache simulation strategies in a HW/SW co-design environment”, ASP-DAC 1999. Así, cualquier técnica de estimación (predicción) del tiempo o consumo requerido para ejecutar un programa en un sistema ha de modelar el funcionamiento de las memorias caches. Dicho modelado se basa en comprobar para cada instrucción o dato requerido por el procesador, si el dato está en la cache correspondiente (llamado acierto, en inglés, “hit”) o si hay que ir a la memoria principal a por el dato (fallo de cache, en inglés, “miss”).

Las técnicas más comunes para realizar estas operaciones se basan en ejecutar el código SW añadiendo el modelado de las caches. En nuestro ámbito, esto se traduce en añadir un sistema de modelado de caches que se integre en la simulación del código SW y pueda conectarse con un modelo de la plataforma HW.

En general se distinguen dos tipos de técnicas de modelado y estimación de caches: estáticas o dinámicas (basadas en simulación).

Las técnicas estáticas se basan en realizar análisis matemáticos que predigan el funcionamiento de las caches. Las caches almacenan valores en sus líneas internas dependiendo de las direcciones de memoria. En consecuencia, el modelado de caches estático requiere normalmente como entrada las trazas de acceso a memoria. Las trazas de acceso a memoria se obtienen usando modelos ISS (del inglés “Instruction Set Simulator”, ISS), lo cual hace que estas técnicas sean muy lentas. Por ejemplo, E.S. Sorenson y otros, en “Cache Characterization Surfaces and Predicting Workload Miss Rates”, In proc. of the Workload characterization, 2001, WWC-4. 2001 IEEE International Workshop, emplean las trazas de acceso a memoria para construir una superficie de pérdida (miss surface) analizando la localidad del acceso.

Estos análisis suelen ser bastante pesimistas y proporcionan resultados generales. En consecuencia, presentan limitaciones en la precisión del modelado de programas concretos y en el modelado del efecto en el resto de la plataforma HW. Estas técnicas están diseñadas para modelar la probabilidad de fallos media, pero no generan información que permita predecir en qué momento ocurrirá cada fallo de cache, con lo que no se pueden modelar dichos fallos en la simulación. Es decir, la aplicación de modelados estáticos o híbridos no es la solución más adecuada para modelado de caches en co-simulación nativa. La co-simulación nativa se ejecuta en un computador anfitrión o computador host (del inglés, *host computer*), por lo que no es posible obtener una traza válida de acceso a memoria para modelar la ejecución de un software en la plataforma destino (del inglés, *target platform*). Además, los modelos de cache deben enviar dinámicamente transferencias de datos al modelo de bus cuando ocurren fallos (del inglés, *miss*).

Las técnicas dinámicas se asocian habitualmente a simuladores de procesadores, denominados simuladores de conjunto de instrucciones (del inglés *Instruction Set Simulator*, ISS). Estos simuladores contienen descripciones del funcionamiento interno del procesador y reciben las mismas entradas y salidas que el procesador real. Un ISS recibe el código SW compilado para el procesador a modelar y los datos en su formato, que son habitualmente distintos a los utilizados por el computador donde se realiza la simulación. Cada acceso a memoria realizado por el modelo de procesador se gestiona por el correspondiente modelo de cache. En este contexto, se han usado ampliamente soluciones, como DineroIV (http://pages.cs.wisc.edu/~markhill/DineroIV/).

A partir de estos simuladores, el modelado de caches se realiza analizando la información que pide el ISS mediante un modelo de cache con la información de su funcionamiento interno. Dado que el ISS pide exactamente lo que pediría el real (las direcciones de memoria y si es lectura o escritura), se emula directamente el comportamiento real del sistema. Esta técnica de modelado (tanto del procesador como de las caches) basada en el modelado de lo que el sistema realiza en cada ciclo de reloj es precisa, pero bastante lenta. Las soluciones basadas en ISS no son adecuadas para estimaciones de rendimiento tempranas, ya que la ejecución de ISS requiere mucho tiempo, y esto es una gran limitación cuando se requiere un gran número de simulaciones de sistemas complejos, como ocurre en los primeros pasos del desarrollo de sistemas complejos.

Para obtener estimaciones más rápidas se han desarrollado técnicas que evitan la ejecución con un ISS. Se ha comprobado que la co-simulación basada en ejecución nativa del software es una técnica más efectiva. El código se ejecuta directamente en el computador de la simulación. Para simular retrasos y consumos se añade información adicional en el código, se compila y se ejecuta. Esto permite obtener simulaciones muy rápidas (mil veces o más que un ISS) con un error un poco mayor, de en torno al 5-10%, para sistemas sin caches.

Por ejemplo, C.M. Kirchsteiger y otros, en "A Software Performance Simulation Methodology for Rapid System Architecture Exploration", In proc. of ICECS, 2008, emplean ejecución nativa de software embebido con anotaciones de rendimiento basadas en el código ensamblador equivalente (del inglés, *equivalent assembly code*) compilado para el procesador destino (del inglés, *target processor*). A. Bouchhima, y otros, en "Automatic Instrumentation of Embedded Software for High Level Hardware/Software Co-Simulation". ASP-DAC 2009, usan código intermedio para evitar las limitaciones que las optimizaciones de compilador provocan cuando intenta correlar código fuente y binario.

Sin embargo, no sólo se requiere el modelado del tiempo de ejecución cuando se aumenta el nivel de abstracción de la simulación; también deben tenerse en cuenta las memorias caches. En este contexto, se han propuesto muy pocas soluciones. El aumento de la velocidad de simulación conseguido cuando se reducen los detalles de modelos de alto nivel limita la viabilidad de aplicar soluciones de bajo nivel conocidas. En este sentido, el modelado de caches de datos utilizando esta técnica de simulación está por resolver. El modelo de caches utilizado por C.M. Kirchsteiger en la cita anterior es puramente estadístico y no se proporcionan resultados exactos.

Por ejemplo, Synopsys, la mayor compañía de herramientas para diseño electrónico del mundo, dice en su página web de modelos basados en ISS que el modelado de caches con técnicas más precisas es imposible (http://www.synopsys.com/TOOLS/SLD/VIRTUALPROTOTYPING/Pages/CoMET-METeor.aspx, sección "Platform development").

En la actualidad las soluciones propuestas para solventar este problema son las siguientes:

- Utilizar un modelado estadístico (RHEiMS-ISS). Este modelo supone que las caches tienen una probabilidad previamente conocida de acierto/fallo. Sin embargo, en muchos casos, el modelado se realiza precisamente para obtener ese número, ya que no es fácil de obtener *a priori*. La elección del tipo de cache óptimo se realiza en base a ese número, que se ha de obtener de la simulación.

- Utilizar una traza obtenida de un ISS. Esto puede ser útil para ejecuciones múltiples, pero en la mayoría de los casos, la necesidad de una ejecución con ISS previa anula el aumento de velocidad de la técnica.

Para modelar el manejo de instrucciones en las caches de instrucciones, los inventores han publicado un trabajo que se basa en conocer el número de instrucciones de cada bloque básico de código ("Fast Instruction Cache Modeling for Approximate Timed HW/SW Co-Simulation", J. Castillo, H. Posadas, E. Villar y M. Martínez, GLSVLSI'10, Providence, US). Sin embargo, esto no es aplicable para el modelado de acceso a datos en las caches.

A. Gerslauer en "Modeling Cache Effects at the Transaction Level", Third IFIP TC 10 International Embedded Systems Symposium, IESS 2009, Langenargen, Germany, September 2009, ha publicado un trabajo de modelado de caches de datos para un tipo de datos, las variables globales. Estas variables, por ser estáticas, tienen la cualidad de que su dirección de memoria se conoce antes de ejecutar. Sin embargo, el resto de datos (ej. variables locales, variables en memoria dinámica, memoria compartida, etc.) no cumplen esta cualidad, con lo que su modelado no es posible.

Por último, la compañía Interdesign Technologies, en su herramienta Fastveri, modela los accesos a las variables utilizando direcciones inventadas por el mecanismo de simulación.

El problema general a resolver es cómo realizar una simulación rápida que permita conocer qué datos requiere el procesador modelado y estime si cada acceso resulta un acierto o un fallo de cache. El problema principal es que, para poder modelar el funcionamiento de la cache de datos, se necesitan las direcciones de los datos a acceder. Sin embargo, si ejecutamos la simulación en un computador convencional, con un procesador distinto al que tendrá el diseño final, y una plataforma HW diferente, las direcciones son completamente distintas a las que utilizará el sistema real.

Por tanto, se plantean dos problemas. En primer lugar, se plantea el problema de obtener direcciones válidas. La herramienta de Interdesign Technologies no resuelve este problema, ni es conocida por los inventores ninguna otra herramienta que lo resuelva. En segundo lugar, se plantea el problema de que los chequeos realizados para comprobar si cada acceso es un acierto o un fallo, y en general todo el modelado de la cache, se realicen con una técnica lo más rápida posible, que reduzca lo menos posible la ganancia en velocidad de simulación con respecto a los ISS.

Resumen de la invención

La presente invención trata de resolver los inconvenientes mencionados anteriormente mediante un método para el modelado rápido de alto nivel de memorias caches de datos para usarse en simulaciones nativas del software embebido.

Concretamente, en un primer aspecto de la presente invención, se proporciona un método de modelado de una memoria cache de datos de un procesador, para simular el comportamiento de dicha memoria cache de datos en la ejecución de un código software en una plataforma destino que comprenda dicho procesador, donde dicha simulación se realiza en una plataforma nativa que tiene un procesador diferente del procesador que comprende dicha memoria cache de datos a modelar. El modelado se realiza mediante la ejecución en dicha plataforma nativa de un código software que se basa en el código software a ejecutar en la plataforma destino, extendido con información para modelar el comportamiento de la memoria cache de datos del procesador de la plataforma destino. El método comprende las etapas de: analizar el código software a ejecutar en la plataforma destino para identificar unos bloques básicos de dicho código y una pluralidad de variables accedidas en cada bloque; añadir al código anotaciones relativas a la memoria cache de datos a simular, donde esas anotaciones comprenden información para modelar el efecto de esa memoria en el procesador destino, obteniéndose un código anotado; compilar el código anotado; ejecutar el código anotado compilado junto con un modelo hardware de la memoria cache de datos. La etapa de añadir al código anotaciones relativas a la memoria cache de datos a simular comprende añadir información que permite obtener las direcciones de las variables que dicha memoria cache de datos simulada debe acceder, para así estimar si cada acceso a dichas variables resulta en un acierto o en un fallo de memoria cache de datos.

Preferentemente, a los mencionados bloques básicos se les añade información del rendimiento, en términos de instrucciones ensamblador, de cada bloque básico.

Preferentemente, para chequear si cada variable accedida se encuentra en la memoria cache de datos o no, se accede a la información de la memoria cache de datos utilizando la dirección asociada a dicha variable mediante llamada a una función de dicho modelo hardware de la memoria cache de datos.

Preferentemente, la información añadida al código que permite obtener las direcciones de las variables que la memoria cache de datos simulada debe acceder comprende un identificador único para cada variable para comprobar cada vez si ésta se encuentra almacenada en la memoria cache de datos o no.

Preferentemente, la información añadida comprende la dirección de cada variable a chequear en la plataforma en la que se realiza la simulación o plataforma nativa, para analizar la localidad espacial de dichas variables. En este caso, para obtener la dirección de cada variable a chequear, se utiliza preferentemente un operador de indirección seguido del nombre de la variable. En caso de acceso a una variable simple, se realiza un único chequeo de la memoria cache de datos. En caso de acceso a un puntero, a un array o a un miembro de una estructura o de una clase, se realiza un acceso adicional por cada nivel de indirección.

Preferentemente, si el procesador destino al que pertenece la memoria cache de datos a simular tiene un ancho de palabra diferente del ancho de palabra del procesador de la plataforma sobre la que se simula dicha memoria cache de datos o plataforma nativa, durante la etapa de compilación se modifican los tipos de los datos del procesador destino para que se ajusten a tipos de tamaño equivalentes del procesador de la plataforma nativa.

En una posible realización, para reducir el número de chequeos, se analiza la localidad de los accesos en el código.

En una realización alternativa, para reducir el número de chequeos, y dado que en la memoria cache de datos las variables se agrupan por líneas, se chequea solamente si la línea está en memoria cache de datos, identificándose la línea con la dirección de la variable.

En otra realización alternativa, para reducir el número de chequeos, si al menos dos variables se han declarado consecutivamente, es suficiente con comprobar el acceso a la primera de dichas variables.

Preferentemente, se traslada el proceso de búsqueda de la dirección en la memoria cache de datos, desde el modelo de memoria cache de datos a la anotación de código, de tal forma que no es necesario llamar a las funciones del modelo de memoria cache de datos para realizar la comprobación de aciertos y fallos.

Preferentemente, el modelo hardware de memoria cache de datos es un modelo basado en arrays de memoria, asignándose una zona de memoria en la que en cada bit se indica si una línea de memoria está en memoria cache o no, siendo así suficiente chequear el bit correspondiente a la línea para saber si está en cache o no.

En otro aspecto de la presente invención, se proporciona un sistema que comprende medios adaptados para llevar a cabo el método descrito anteriormente.

En otro aspecto de la presente invención, se proporciona un programa informático que comprende medios de código de programa informático adaptados para realizar las etapas del método descrito anteriormente, cuando el programa se ejecuta en un ordenador, un procesador de señal digital, una disposición de puertas de campo programable, un circuito integrado de aplicación específica, un microprocesador, un microcontrolador, y cualquier otra forma de hardware programable.

Las ventajas de la invención se harán evidentes en la descripción siguiente.

Breve descripción de las figuras

Con objeto de ayudar a una mejor comprensión de las características de la invención, de acuerdo con un ejemplo preferente de realización práctica del mismo, y para complementar esta descripción, se acompaña como parte integrante de la misma, un juego de dibujos, cuyo carácter es ilustrativo y no limitativo. En estos dibujos:

La figura 1 muestra un diagrama del método de modelado de acuerdo con una posible realización de la invención.

La figura 2 muestra un modelo de memoria cache de datos de acuerdo con una posible implementación de la invención.

La figura 3 muestra una comparación entre tres técnicas de simulación: simulación basada en ISS, una técnica basada en búsqueda convencional y la técnica propuesta en la invención.

Descripción detallada de la invención

En este texto, el término “comprende” y sus variantes no deben entenderse en un sentido excluyente, es decir, estos términos no pretenden excluir otras características técnicas, aditivos, componentes o pasos.

Además, los términos “aproximadamente”, “sustancialmente”, “alrededor de”, “unos”, etc. deben entenderse como indicando valores próximos a los que dichos términos acompañen, ya que por errores de cálculo o de medida, resulte imposible conseguir esos valores con total exactitud.

Las siguientes realizaciones preferidas se proporcionan a modo de ilustración, y no se pretende que sean limitativas de la presente invención. Además, la presente invención cubre todas las posibles combinaciones de realizaciones particulares y preferidas aquí indicadas. Para los expertos en la materia, otros objetos, ventajas y características de la invención se desprenderán en parte de la descripción y en parte de la práctica de la invención.

Cuando un código SW se compila y simula en una determinada plataforma, normalmente una plataforma de propósito general (por ejemplo un PC) para ejecutarse en esa misma plataforma, esto es, utilizando el lenguaje de instrucciones utilizado por el/los procesadores de esa plataforma, a dicho código y a las operaciones que se realizan con él se les denomina nativas. Cuando un código se desarrolla, compila y simula en una determinada plataforma, normalmente una plataforma de propósito general (por ejemplo un PC) para ejecutarse en otra plataforma (plataforma destino o un modelo muy realista de una plataforma destino), normalmente una plataforma de propósito específico, esto es, generando un ejecutable en el lenguaje de los procesadores de la plataforma destino, no donde se compila, y simulándolo mediante un sistema de traducción de ambos lenguajes, a dicho código se le denomina cruzado.

A su vez, co-simular se refiere a simular conjuntamente el software y el hardware de un sistema.

En este texto, a menudo se utiliza el término “cache” para identificar la memoria cache de datos.

Para realizar la ejecución de cualquier programa informático, como una simulación, se realiza un proceso de tres etapas: se genera el código, se compila y se ejecuta. La invención se localiza en las etapas de compilación y ejecución.

Como punto de partida para la solución proporcionada se toman los mecanismos de anotación de código tradicionales, por los cuales se añade una etapa al proceso de compilación. En esta etapa se añade información adicional a la proporcionada por el código SW original. En esta ocasión se añade información para el modelado de caches de datos, sin perjuicio de otros posibles añadidos orientados al modelado de tiempo, consumo, caches de instrucciones, etc. Llamamos a este proceso "anotación". El código anotado se compila nativamente.

En función de la aplicación concreta, en una realización particular, este código anotado compilado nativamente puede co-simularse junto con un modelo de plataforma hardware del sistema a modelar. El modelo de plataforma HW normalmente lo elige el diseñador. En una realización alternativa, no se realiza simulación de un modelo de plataforma hardware.

En una realización particular, para realizar la co-simulación, se añade un modelo de sistema operativo (SO) abstracto al motor de simulación de SystemC. SystemC es un lenguaje de co-simulación (extensión de C++) que permite el modelado de componentes y plataformas HW, y la integración de modelos del SW. El modelo de sistema operativo (SO) se encarga de controlar la ejecución de las tareas SW y de proporcionar funciones para comunicación de entrada/salida (I/O) sobre la infraestructura básica proporcionada por SystemC. Estas funciones conectan el SW con un modelo de bus desarrollado siguiendo el estándar TLM2 (www.systemc.org). Los fallos de cache (*cache misses*) y actualizaciones de los valores en la memoria principal se modelan enviando las transferencias correspondientes a través del modelo de bus desde la cache a la memoria principal. En consecuencia, al considerar tanto las transferencias de entrada/salida (sistema operativo) como las transferencias de cache, se modelan todas las comunicaciones del procesador con el resto del sistema, permitiendo que se pueda considerar correctamente el efecto que el procesador tiene en el resto del sistema (ej. buses, caches de orden superior, periféricos, etc.).

La figura 1 muestra un diagrama del método de simulación o modelado de acuerdo con una posible realización de la invención. El método está basado en cuatro etapas de modelado, tres en la etapa de compilación y la última en la de simulación:

En primer lugar (referencia 1 de la figura 1), se analiza el código software de aplicación (SW), detectando sus bloques básicos. Un bloque básico es una secuencia de instrucciones que se ejecuta completamente, de forma secuencial, con un único punto de entrada y un único punto de salida. Este código SW proporcionado por el usuario para cada caso particular comprende la información sobre las instrucciones y datos a ejecutar por el procesador, pero no contiene la información necesaria para modelar el rendimiento de dicho procesador, y el funcionamiento de las caches. Este código está escrito en un lenguaje de programación de los habituales empleados en diseño de software (SW), tal como C ó C++.

A continuación (referencia 2 de la figura 1), se estiman las métricas de rendimiento para cada bloque de la plataforma destino (*target platform*), con el objetivo de extraer información del rendimiento de los bloques. Ejemplos no limitativos de métricas de rendimiento son tiempo, consumo, número de accesos de entrada/salida, etc.) Cómo se estiman las métricas de rendimiento queda fuera del alcance de la presente invención. Como ejemplo no limitativo, pueden estimarse como proponen los autores en el capítulo: H. Posadas, J. Castillo, D. Quijano, V. Fernández, E. Villar, Marcos Martínez, "SystemC Platform Modeling for Behavioral Simulation and Performance Estimation of Embedded Systems", del libro L. Gomes and J. M. Fernandes (Eds.): "Behavioral Modeling for Embedded Systems and Technologies: Applications for Design and Implementation", IGI Global. 2009-07.

Sin embargo, el modelado de caches no se puede obtener directamente del análisis de los bloques de código SW. Ha de ser calculado dinámicamente durante la simulación, en base a las métricas que sí se pueden obtener del código. Para las caches de instrucciones se propone una técnica en "Fast Instruction Cache Modeling for Approximate Timed HW/SW Co-Simulation", J. Castillo, H. Posadas, E. Villar, M. Martínez, GLSVLSI'10, Provicence, US. Esta técnica se basa en el hecho de que cada instrucción se usa en un único punto del código, y que la dirección de memoria donde se guarda puede ser obtenida analizando el código, sin necesidad de la etapa de simulación.

Por contra, las variables con los datos se usan en múltiples sitios, y su dirección en memoria no siempre se puede conocer analizando el código, solo se conoce durante la ejecución del código, requiriendo el paso de simulación, e invalidando el uso de las técnicas previas empleadas para otras métricas.

Seguidamente (referencia 3 de la figura 1), se reconstruye el código SW, añadiéndole las estimaciones de las métricas de rendimiento y el resto de información obtenida del paso anterior para modelado de caches. A este proceso se le llama anotación o instrumentación. Dado que las técnicas aplicadas previamente no son aplicables para el modelado de caches de datos, es necesario obtener otro tipo de información y realizar anotaciones de distinto tipo. La información a obtener y a anotar se describe posteriormente.

Finalmente, el código anotado se ejecuta nativamente. En una realización particular, esta ejecución puede realizarse junto con un modelo de la plataforma hardware (HW), como se detalla más adelante, para analizar el rendimiento del sistema completo, no únicamente de la parte SW.

A continuación se describe en detalle el proceso representado en la figura 1, y que representa un caso de implementación de los cuatro puntos anteriores.

En primer lugar, se analizar el código original o código fuente 101 con una gramática 102 del lenguaje de programación en que esté escrito el código fuente. Preferentemente, el código fuente está escrito en C ó C++. Por lo tanto, en este caso la gramática 102 es una gramática C ó C++ de código abierto. Un ejemplo de esta gramática es la descrita por E. D. Willink en "Meta-Compilation for D++", PhD thesis, University of Surrey, June 2001.

A partir de esta gramática 102 se obtiene un árbol de sintaxis abstracta (del inglés, *abstract syntax tree* AST) 103 del código, en formato XML (del inglés *Extensible Markup Language*) (lenguaje de marcas extensible). Como resultado se obtiene una identificación de todos los bloques básicos del código SW.

La descripción XML se reconstruye añadiendo primero marcas en los bloques básicos 104 y después se cros-compila, obteniéndose el coste de cada bloque básico en la plataforma destino (*target platform*) 105 en términos de instrucciones ensamblador (*assembler instructions*).

Este paso se realiza de forma preferente, pero no es esencial para la simulación de la memoria cache de datos. En realidad, la estimación de los valores de rendimiento (o coste) de cada bloque básico se requiere para realizar una simulación nativa que tenga en cuenta el tiempo de ejecución, pero el mecanismo de estimación es independiente del modelado de memoria cache de datos. Es decir, cualquier otra forma de estimación de rendimiento o coste puede usarse alternativamente.

A continuación, se vuelve a codificar el código a partir de la descripción XML. Sin embargo, en esta etapa, se añade en el código información adicional 106. En concreto, se añade información de tiempo de ejecución (*timing*), modelado de consumo, marcas de memoria cache de instrucciones y marcas de memoria cache de datos. Así, se obtiene un código anotado 107, válido para ejecución nativa. Finalmente el código se ejecuta para realizar la simulación de sistema.

Para realizar un modelado eficiente de memoria cache de datos, se deben superar dos retos: cómo obtener las direcciones de acceso a partir de ejecución nativa y cómo crear un modelo de cache rápido. Las anotaciones que se insertan en el código para permitir el modelado de la cache de datos, tienen como objetivo resolver ambos retos. A continuación se describe en detalle la etapa de anotación o adición de información de cache de datos 106 al código:

Para anotar información de caches de datos 106 se identifican las variables accedidas en el código. Esto puede realizarse de varias formas, tales como mediante un análisis gramatical o un análisis del código compilado antes de realizar dicha anotación. La obtención del AST presentada anteriormente es un ejemplo. Sin perjuicio de optimizaciones opcionales que se describen más adelante, se realiza al menos una anotación por cada acceso a variables.

Para obtener las direcciones de cada acceso de datos a memoria, se propone usar las direcciones del código fuente durante ejecución nativa. Para realizar la anotación se necesita un identificador único para cada variable que permita chequear durante su uso sucesivo si ésta se encuentra almacenada en cache o no. Además, se necesita un mecanismo que permita analizar la localidad espacial de las variables (si las direcciones son similares o no). En el sistema real se utiliza la dirección de la variable en la plataforma destino para ambos fines. Sin embargo, en la simulación propuesta no se dispone de dichas direcciones (puesto que la plataforma destino tal vez ni siquiera esté implementada).

El método propuesto utiliza, en lugar de la dirección de la variable en la plataforma destino, la dirección en la plataforma donde se realiza la simulación (las direcciones de la simulación nativa). En una realización preferente, en la que la simulación se realiza en lenguaje C (o cualquiera de sus variantes, como C++), para obtener la dirección de cada variable a chequear se utiliza el operador de indirección "&" del lenguaje C (por ejemplo, la dirección de la variable "v" es "&v"). Esto también se hace para variables complejas, como arrays y punteros. De forma adicional, para accesos a puntero, se incluye un acceso adicional al propio puntero. Es decir, en caso de accesos a variables simples, un único chequeo de la cache es necesario. En caso de acceso a variables complejas, como punteros, arrays o miembros de estructuras o clases, se realiza un acceso adicional por nivel de indirección (ej. un puntero doble requiere dos accesos adicionales).

Para chequear si cada variable accedida se encuentra en cache de datos o no, se ha de acceder a la información de la cache utilizando la dirección asociada a la variable. De forma convencional, esto se realiza llamando a una función del modelo de cache, sin perjuicio de utilizar otras soluciones más optimizadas que se presentan a continuación. Como modelo de cache se puede utilizar un modelo convencional como los utilizados habitualmente en los ISS. Para modelar los accesos a cache durante la simulación, se accede a un modelo de cache de datos usando las funciones como el que se describe más adelante. Por ejemplo, considerando una línea simple (a[i]=b), el código anotado resultante es el expresado en las siguientes funciones de acceso a cache de datos:

```
{ a[i]=b; // original code
data_cache_read(&i); data_cache_read(&b);
data_cache_read(&a); data_cache_write(&a[i]); }
```

Con estas consideraciones se resuelve el problema de identificar las variables a chequear y la obtención de un identificador válido. Como ejemplo de implementación, para cada bloque básico, durante la reconstrucción del código SW a partir del AST XML, se genera una lista de las variables leídas y escritas. Al final de cada bloque básico se anota un chequeo de existencia en cache por cada variable de la lista.

Además, con carácter opcional, el método puede aplicar diversas optimizaciones adicionales que pasamos a describir a continuación.

Opcionalmente, se puede reducir el número de chequeos analizando la localidad de los accesos en el código. Si una variable se accede en un punto del código, es de esperar que en las instrucciones siguientes, si se accede otra vez, la probabilidad de que se encuentre en cache sea cercana al 100%, por lo que se pueden evitar dichas comprobaciones.

También opcionalmente, dado que en caches las variables se agrupan por "líneas" que se manejan como una unidad, es suficiente con chequear si la línea está en cache, sin tener que chequear la dirección entera. La línea se identifica con la dirección de la variable, obviando los bits menos significativos. El número de bits depende del tamaño de línea de la cache.

También opcionalmente, si dos variables o más se han declarado consecutivamente (o casi, eso es, en el mismo bloque básico o bloques adyacentes) y ambas van a ir a la misma línea de cache, es suficiente con comprobar el acceso a la primera, evitando el chequeo de la(s) otras, de la misma forma que se puede evitar el chequeo de accesos múltiples a la misma variable (ver párrafo anterior).

Por último, para un modelado más correcto de la cache y/o de sus efectos en otros elementos del sistema, se pueden incluir algunas modificaciones en las direcciones que permitan transformar las direcciones nativas en direcciones de la plataforma destino para realizar la simulación. Estas transformaciones se han aplicado al modelado de cache de instrucciones con éxito en S. Real, H. Posadas & E. Villar: "L2 Cache Modeling for Native Co-Simulation in SystemC", SIES'2010. La presente invención extiende este procedimiento para optimizar el modelado de la cache de datos, considerando que a diferencia de las instrucciones, los datos están divididos en múltiples secciones, algunas de las cuales no se encuentran en el código compilado, sino que se generan durante ejecución.

Para transformar las direcciones nativas en direcciones destino, es necesario analizar la estructura ejecutable. En un fichero ejecutable común, las variables de datos se almacenan en varias secciones. Siguiendo el estándar ELF (Executable and Linkable Format, vers 1.2. TIS committee http://pdos.csail.mit.edu/6.828/2005/readings/elf.pdf), las variables globales se almacenan principalmente en las secciones "data", "rodata" y "bss". Las variables locales se almacenan en el "stack". Finalmente, la memoria dinámica se obtiene a partir del "heap".

Todas las variables se colocan en su sección correspondiente, ordenadas según se declaran. Como se usan el mismo front-end del compilador y el mismo orden de enlazado, el orden de las variables es el mismo en la plataforma nativa y en la plataforma destino. Así, es posible obtener las direcciones de modelado de caches a partir de la ejecución nativa.

En una realización particular, en la que se exige conocer la dirección exacta de cada variable en la plataforma destino, se requieren algunas consideraciones adicionales. No obstante, como se indica más adelante, no es imprescindible obtener esas direcciones exactas. Las consideraciones adicionales son las siguientes:

Primero, para que las direcciones de las variables en la simulación sean válidas, el tamaño de los datos ha de ser el mismo en la plataforma de simulación (por ejemplo, computador) y en la plataforma real. Con objeto de asegurar esto, en caso de que ambas tengan procesadores con el mismo ancho de palabra (ej. 32 bits) no son necesarias modificaciones. Sin embargo, dependiendo del procesador destino y del compilador, los tamaños de los distintos tipos de datos pueden cambiar. En este caso (ej. computador de 64 bits y plataforma real de 32 bits), se modifican todos los tipos de todos los datos durante el proceso de compilación para que se ajusten a tipos de tamaño equivalentes en la plataforma real (ej. modificar un tipo "long int" de tamaño 64/32 por un "int", que tendría 32 bits en ambas plataformas). Para ello, se extiende el constructor del código XML (elemento 103 de la figura 1). Con esta extensión, es posible indicar el tamaño de cada tipo de datos de la plataforma destino y los tamaños de los datos en la ejecución nativa. Usando esta información, cuando se reconstruye el código, se modifica el tipo de cada variable, aplicando un tipo de datos que casa el tamaño esperado en la plataforma destino. Además hay que indicar al compilador utilizado para realizar la simulación que utilice la misma granularidad que el compilador para la plataforma real (ej. mediante *#pragma pack (push,2)*).

Segundo, para obtener las direcciones exactas de cada variable, es necesario aislar las variables de la aplicación SW del resto de la simulación. Para aislar las variables, el método compila la aplicación SW como una librería dinámica en la plataforma u ordenador host (anfitrión), y lo enlaza al resto de la simulación. De esta forma, las secciones "text", "data", "rodata" y "bss" se separan del motor de simulación y de los modelos de componentes HW. El orden de las variables "stack" variables' se mantiene automáticamente, ya que las llamadas a funciones del código original no se han modificado, y las variables locales no se añaden. Finalmente, las direcciones "heap" deben modelarse. En la infraestructura presentada, la infraestructura no realiza llamadas a funciones "malloc" durante la simulación. Por tanto, no se requieren mecanismos adicionales. Sin embargo, para otras infraestructuras, la solución recomendada es aplicar un gestor "heap" específico para las tareas de aplicaciones SW. El gestor incluido en la librería "newlib" (SystemC. IEEE-1666 estándar, www.systemc.org) puede adoptarse para este propósito fácilmente. En cualquier caso, como el

"heap" se usa comúnmente para crear arrays grandes, la localidad de los datos puede modelarse con un error mínimo incluso cuando la infraestructura de simulación realiza accesos "heap" adicionales.

Finalmente, debe tenerse en cuenta que las secciones de código empiezan en direcciones distintas en la plataforma host (anfitriona) y en la plataforma destino. Por eso, es necesario aplicar una corrección final para conseguir la dirección real. Por ejemplo, en sistemas Linux, la posición de cada sección de código en la simulación nativa puede encontrarse a partir de la propia información de formato de la librería dinámica y de la información de los ficheros "/proc/self/stat" y "/proc/self/maps". Si el diseñador también tiene las direcciones de las secciones correspondientes en las plataformas destino, se puede obtener la dirección de cada variable en la plataforma.

Sin embargo, como el rendimiento cache depende principalmente de la ubicación espacial, a menudo no es necesario obtener las direcciones de destino exactas. Por eso, como se dijo anteriormente, este tercer paso es opcional. Los bits más altos de la dirección pueden usarse como etiquetas en la cache, pero la identificación de la línea cache depende sólo de los bits más bajos. Más adelante se demuestra que aplicar esta simplificación del modelo produce una tasa de error baja.

Hasta ahora se ha explicado cómo resolver el problema de obtener direcciones válidas. El segundo problema que debe resolverse es cómo realizar un chequeo eficiente para comprobar si las variables están o no en la cache.

Los modelos de memoria cache convencionales basados en ISS se basan en búsquedas de las direcciones en el modelo de cache. Estas búsquedas pueden ser demasiado lentas para el tipo de simulación propuesto para la aplicación de la técnica. La solución propuesta es simplificar el proceso de chequeo quitándolo del modelo de cache (con lo que se obtiene un modelo de cache simplificado) e introduciéndolo como parte de la anotación, lo que permite evitar la realización de búsquedas durante el chequeo.

Aunque las llamadas a funciones propuestas anteriormente en este documento representan una solución válida para el modelado de caches de datos, producen una carga adicional (overhead) de recursos de simulación. En simulación nativa, cada bloque básico de código SW requiere para su modelado unas pocas instrucciones máquina. Sin embargo, se requieren muchas instrucciones de la máquina anfitriona para accede a las funciones del modelo cache de datos y realizar una búsqueda por etiquetas para averiguar si la variable está en cache o no. Por ejemplo, en un ARM920, cada dirección puede mapearse en 64 líneas. Comprobar las 64 etiquetas en el modelo requiere mucho tiempo.

Para reducir esa sobrecarga de simulación, se propone modificar las llamadas a cache, trasladando parte del esfuerzo de chequeo a la anotación. Para realizar esto, una posible implementación se basa en sustituir los modelos hardware de la memoria cache de datos basados en búsquedas por un modelo basado en arrays de memoria. En este tipo de modelo, se designa una zona de memoria en la que en cada bit se indica si una línea de memoria está en cache o no. Así, se divide todo el espacio de memoria utilizado en líneas, conforme se guarda en cache. En este caso, es suficiente chequear el bit correspondiente a la línea para saber si está en cache o no, evitando el proceso de búsqueda. Más adelante se detalla un modelo simplificado de memoria cache de datos que puede usarse.

Los procesadores embebidos convencionales son procesadores de 32 bits. Así, se puede acceder a un máximo de 2^{32} direcciones de memoria. Considerando una línea de cache convencional de 8 palabras de 4 bytes, se puede mapear en cache un máximo de 2^{27} líneas diferentes durante toda la ejecución del código. Entonces, es posible crear un array de 16 Mbytes donde cada bit indica si la línea está o no en cache. Considerando que los ordenadores (host, anfitriones) actuales tiene del orden de GigaBytes de RAM, esta cantidad de memoria supone un requisito pequeño.

Utilizando el modelo basado en arrays de memoria se puede simplificar el chequeo de la cache. En lugar de realizar una llamada a una función del modelo de cache para comprobar cada variable, se puede integrar un código de chequeo (un "if", por ejemplo) en la anotación del código SW, evitando el coste de realizar llamadas a función.

Usando este array, es posible encontrar una variable en cache leyendo el valor del bit correspondiente en el array de memoria. Por ejemplo, este valor puede obtenerse aplicando la siguiente instrucción:

```
#define CACHED(addr) mem[(addr>>8)]&1<<((addr>>5)&7)
```

Entonces la función "data_cache_read" puede sustituirse por ejemplo por una expresión "if", que compruebe el bit correspondiente, y solo en caso de que no esté se llame al modelo simplificado de cache:

```
"If(CACHED(&variable)) dcache_insert_line(&addr);"
```

Consecuentemente, se evita la búsqueda de etiquetas (*tag search*) y la llamada a función para acceder al modelo de cache de datos se realiza solo en caso de fallos (*misses*). Como en operaciones normales la mayoría de los accesos a cache resultan en un éxito (*hit*), esta solución minimiza la sobrecarga de modelado de cache, haciendo viable su aplicación para co-simulación nativa.

Para caches de escritura retardada (write-back), se necesita otro array de memoria para almacenar los bits sucios (*dirty bits*). Para pasar los bits a verdaderos (*true*) cuando se almacena un dato, se puede usar, por ejemplo, la siguiente macro:

#define DIRTY(addr) dirty[(addr>>8)] | 1<<((addr>>5)&0x7)

La figura 2 ilustra un posible modelo de cache de datos 201 para usar durante la simulación. Internamente, este modelo de cache es similar a la infraestructura real de cache. Principalmente comprende uno o varios arrays en los que se almacenan las direcciones de las líneas que se encuentran en un momento dado en cache. Cuando se requiere una nueva línea 202, tras comprobar si está o no en cache por los mecanismos explicados anteriormente de verificación del array (mapa de bits del modelo de memoria 203), se debe confirmar la existencia de una línea vacía, verificando los arrays del modelo de memoria cache 201. Si no existe, se elimina una línea de la cache siguiendo una política o táctica aleatoria o de eliminación (todos contra todos, *round robin*). Una vez que la línea se ha eliminado, se chequea el bit de sucio correspondiente y se limpia el bit en el array de memoria (mapa de bits de líneas modificadas (dirty bits) 204). En caso de estar el bit de sucio activo, se realiza un acceso a la memoria principal o a caches de orden superior 205, directamente o a través de un buffer intermedio. Por último guarda la información de la nueva línea en el modelo de cache 201 y en el mapa de bits del modelo de memoria 203.

El modelo desarrollado permite configurar el tamaño de cache de datos, el grado de asociatividad, el tamaño de línea y el método de sustitución.

A continuación se muestran unos ejemplos y resultados del método de la invención. Para probar la validez del método de la invención, las técnicas propuestas en el mismo han sido testadas de dos formas diferentes. Primero, se han aplicado algunos tests para demostrar la exactitud obtenida con el método. Además, el aumento de velocidad obtenido se compara con técnicas basadas en búsqueda y la exactitud se compara con simulaciones ISS.

Para comparar la solución, se ha usado una plataforma destino con un procesador ARM920T. Este procesador incluye una cache de instrucciones de 16KB con una asociatividad de 64 vías. Su tamaño de línea es de 32 bytes.

Para obtener comparaciones de aumento de velocidad, se han ejecutado varios puntos de referencia comunes con las tres siguientes técnicas: la simulación basada en ISS, la técnica basada en búsqueda y la técnica optimizada de la invención. La figura 3 muestra una comparación entre las tres técnicas y la tabla 1 contiene las correspondientes tasas de error.

El aumento de velocidad medio conseguido por la técnica de modelado de la invención con los puntos de referencia elegidos es de más de dos órdenes de magnitud con respecto a las simulaciones ISS.

TABLA 1

Precisión de puntos de referencia

	Fallos de cache de datos			
Ejemplo	*Invención*	*Basado en búsqueda*	*ISS*	*Error (%)*
Bubble 1000 elements	*127*	*127*	*126*	*0.8*
Bubble 10000 elements	*5207383*	*5207383*	*5199310*	*0.16*
Hanoi	*44*	*44*	*41*	*7.32*
Factorial	*500*	*500*	*375*	*33.33*

Se han elegido cuatro códigos diferentes: Bubble de 1000 elementos, Bubble de 10000 elementos, Hanoi y Factorial. Pequeños puntos de referencia específicos son muy adecuados para analizar las limitaciones de la técnica propuesta. Como puede observarse, en puntos de referencia con un número pequeño de llamadas a funciones, tales como el burbuja (*bubble*), la el error de estimación de la tasa de fallos (*miss rate estimation error*) es menor del 1%, pero cuando el número de llamadas a funciones aumenta, la tasa de errores aumenta también. Hanoi tiene un alto porcentaje de llamadas a funciones en sus instrucciones, siendo su error de estimación de la tasa de fallos (*miss rate estimation error*). Factorial es un código compuesto principalmente por llamadas a funciones, siendo su error de estimación de la tasa de fallos del 33%.

El error se debe a la cantidad variable de información almacenada en la pila cuando se introduce una función. En ejemplos comunes, puede estimarse analizando las variables de la pila, pero en el ejemplo Factorial no hay variables locales en todo el código, por lo que la estimación falla. No obstante, los códigos comunes no se componen solamente de llamadas a funciones y usan variables locales, por lo que el error es mínimo. Para asegurar esto, el último ejemplo es un programa representativo de Vocoder de GSM con diferentes entradas. Los resultados obtenidos con la técnica de la invención se han contrastado con simulación ISS. También se proporcionan los tiempos de simulación. La tabla 2 muestra la comparación. El error resultante es menor del 4%.

TABLA 2

Precisión del Vocoder GSM

	Fallos de cache de datos			Tiempo de simulación		
Frames	Invención	ISS	Error (%)	Invención (sec)	ISS (sec)	Aumento velocidad
1	660	670	-1.515	0.038	12.104	318.526
4	2370	2452	-3.460	0.121	47.550	392.975
7	4104	4235	-3.192	0.201	83.951	417.667
10	5915	6026	-1.877	0.277	119.212	430.368

Para diseños muy específicos, la técnica puede mejorarse para solucionar el error de la "llamada a función". La técnica de la invención es independiente de la plataforma, para minimizar el esfuerzo de exploración. En consecuencia, la información específica de ensamblador no se usa en el modelado. Sin embargo, la técnica puede mejorarse fácilmente con información específica del procesador para reducir el error.

Este error es causado por el hecho de que los compiladores de diferentes arquitecturas guardan un número diferente de registros en llamadas a funciones. Sin embargo, el problema puede resolverse especificando el valor correcto. En la metodología de modelado propuesta (figura 1), la información puede obtenerse a partir del código cros- compilado usado para la estimación temporal, si se proporciona el des-ensamblador adecuado.

Como conclusiones, se ha mostrado que es posible un modelado de cache de datos eficiente en simulación nativa a la vez que garantizar precisión. Se pueden obtener direcciones de acceso a cache válidas a partir de simulación nativa. Además, se puede conseguir modelado cache rápido usando técnicas apropiadas desarrolladas específicamente para beneficiarse de la ejecución de código fuente anotado.

Se pueden aplicar direcciones nativas si las variables de aplicación se aíslan en librerías dinámicas y se modifican los tamaños de los datos para ajustarlos a los tamaños de la plataforma destino. Además, se pueden ajustar las direcciones usando la dirección inicial de cada sección de código para obtener la dirección de la variable real. Sin embargo, los resultados demuestran que esta modificación de la dirección no es necesaria para obtener buena precisión de cache de datos.

En simulación nativa, el coste computacional de modelar una instrucción SW es equivalente al coste de una instrucción SW nativa. Sin embargo, el coste de llevar a cabo las llamadas a funciones y las búsquedas de etiquetas requeridas reduce la velocidad de simulación considerablemente. La técnica de la invención traslada la mayor parte del coste computacional del chequeo de línea del modelo de cache al de anotación. Para comprobar si la línea actual está en cache, se requiere una única instrucción. De esta forma, se mantiene la velocidad de simulación.

Los resultados demuestran que aplicando estas técnicas, los efectos de la cache de datos en el rendimiento del sistema pueden estimarse con precisión en una infraestructura de co-simulación temprana y rápida. Además, la exactitud del modelo permite la exploración de la mejor configuración de cache para el sistema bajo desarrollo al principio del proceso de diseño.

En conclusión, como puede observarse, el método de la invención es rápido y no requiere trazas de acceso a memoria, por lo que evita la dependencia de simulaciones ISS lentas para obtener la traza. Además, el mecanismo de detección de fallo (miss) no se basa en búsqueda. El tiempo de detección de éxito (hit) es siempre constante. Esta característica aumenta la velocidad al menos en un orden de magnitud con respecto a técnicas nativas basadas en búsqueda.

En este documento se ha explicado cómo conocer, mediante una simulación en una plataforma, el efecto que la memoria cache de datos de un procesador distinto tendría en la ejecución de código. Por tanto, la invención es aplicable a las áreas de diseño de programas (software, SW) y a las de diseño de la plataforma física sobre la que se ejecuta dicho código (hardware, HW), especialmente en aquellos ámbitos donde el tiempo de ejecución y/o el consumo son importantes. La invención puede aplicarse, por ejemplo, al modelado y diseño de procesadores y microprocesadores.

REIVINDICACIONES

1. Un método de modelado de una memoria cache de datos de un procesador, para simular el comportamiento de dicha memoria cache de datos en la ejecución de un código software en una plataforma destino que comprenda dicho procesador, donde dicha simulación se realiza en una plataforma nativa que tiene un procesador diferente del procesador que comprende dicha memoria cache de datos a modelar, donde dicho modelado se realiza mediante la ejecución en dicha plataforma nativa de un código software que se basa en dicho código software a ejecutar en dicha plataforma destino, extendido con información para modelar dicho comportamiento de dicha memoria cache de datos del procesador de la plataforma destino, donde el método comprende las etapas de:

- analizar (102) el código software a ejecutar en la plataforma destino (101) para identificar unos bloques básicos (104) de dicho código y una pluralidad de variables accedidas en cada bloque;

- añadir (106) a dicho código anotaciones relativas a la memoria cache de datos a simular, donde dichas anotaciones comprenden información para modelar el efecto de dicha memoria en el procesador destino, obteniéndose un código anotado (107);

- compilar (108) dicho código anotado;

- ejecutar (109) dicho código anotado compilado junto con un modelo hardware de dicha memoria cache de datos;

estando el método **caracterizado** por que para chequear si cada una de dicha pluralidad de variables accedidas en cada bloque (104) se encuentra en dicha memoria cache de datos o no, se accede a la información de la memoria cache de datos utilizando la dirección asociada a dicha variable mediante la llamada a una función de dicho modelo hardware de la memoria cache de datos.

2. El método de la reivindicación 1, donde a dichos bloques básicos (104) se les añade información del rendimiento, en términos de instrucciones ensamblador, de cada bloque básico.

3. El método de cualquiera de las reivindicaciones anteriores, donde dicha información añadida al código que permite obtener las direcciones de las variables que dicha memoria cache de datos simulada debe acceder comprende un identificador único para cada variable para comprobar cada vez si ésta se encuentra almacenada en la memoria cache de datos o no.

4. El método de cualquiera de las reivindicaciones anteriores, donde dicha información añadida comprende la dirección de cada variable a chequear en la plataforma en la que se realiza la simulación o plataforma nativa, para analizar la localidad espacial de dichas variables.

5. El método de la reivindicación 4, donde, para obtener la dirección de cada variable a chequear, se utiliza un operador de indirección seguido del nombre de la variable.

6. El método de la reivindicación 5, donde, en caso de acceso a una variable simple, se realiza un único chequeo de la memoria cache de datos.

7. El método de la reivindicación 5, donde, en caso de acceso a un puntero, a un array o a un miembro de una estructura o de una clase, se realiza un acceso adicional por cada nivel de indirección.

8. El método de cualquiera de las reivindicaciones anteriores, donde, si el procesador destino al que pertenece la memoria cache de datos a simular tiene un ancho de palabra diferente del ancho de palabra del procesador de la plataforma sobre la que se simula dicha memoria cache de datos o plataforma nativa, durante la etapa de compilación (108) se modifican los tipos de los datos del procesador destino para que se ajusten a tipos de tamaño equivalentes del procesador de la plataforma nativa.

9. El método de cualquiera de las reivindicaciones anteriores, donde, para reducir el número de chequeos, se analiza la localidad de los accesos en el código.

10. El método de cualquiera de las reivindicaciones 1-8, donde, para reducir el número de chequeos, y dado que en la memoria cache de datos las variables se agrupan por líneas, se chequea solamente si la línea está en memoria cache de datos, identificándose la línea con la dirección de la variable.

11. El método de cualquiera de las reivindicaciones 1-8, donde, para reducir el número de chequeos, si al menos dos variables se han declarado consecutivamente, es suficiente con comprobar el acceso a la primera de dichas variables.

12. El método de cualquiera de las reivindicaciones anteriores, donde se traslada el proceso de búsqueda de la dirección en la memoria cache de datos, desde el modelo de memoria cache de datos a la anotación de código (106), y donde dicho modelo hardware de memoria cache de datos (110) es un modelo basado en arrays de memoria, asignán-

dose una zona de memoria en la que en cada bit se indica si una línea de memoria está en memoria cache o no, siendo así suficiente chequear el bit correspondiente a la línea para saber si está en cache o no.

13. Un sistema que comprende medios adaptados para llevar a cabo el método de cualquiera de las reivindicaciones anteriores.

14. Un programa informático que comprende medios de código de programa informático adaptados para realizar las etapas del método según cualquiera de las reivindicaciones de la 1 a la 12, cuando dicho programa se ejecuta en un ordenador, un procesador de señal digital, una disposición de puertas de campo programable, un circuito integrado de aplicación específica, un microprocesador, un microcontrolador, y cualquier otra forma de hardware programable.

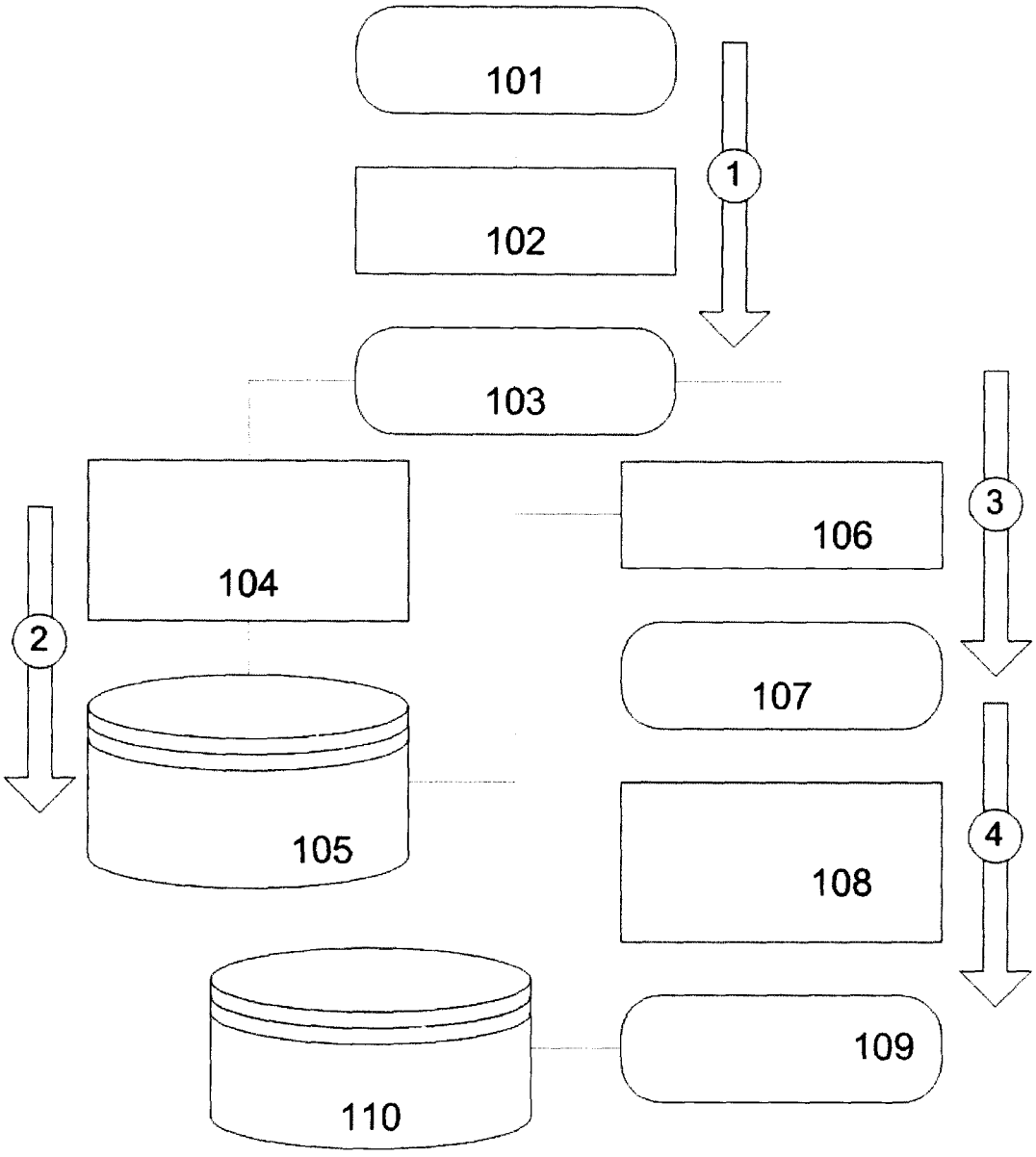

FIGURA 1

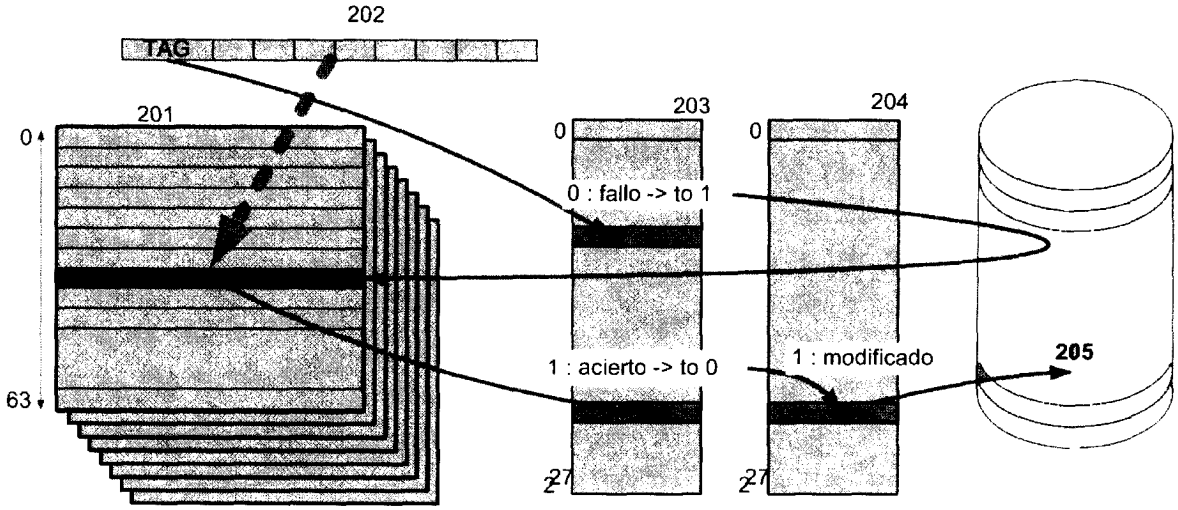

FIGURA 2

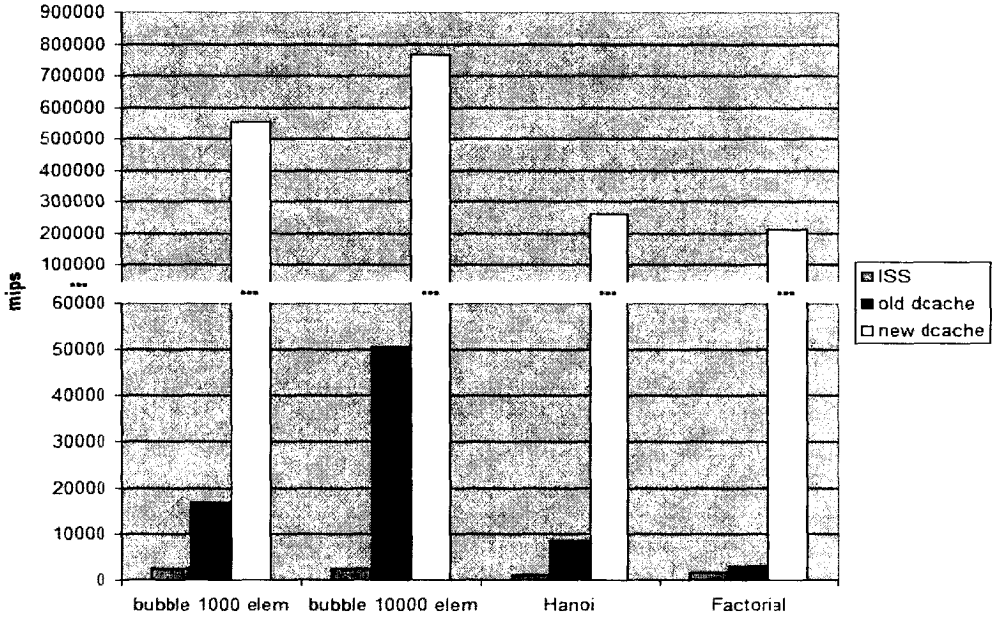

FIGURA 3

OFICINA ESPAÑOLA
DE PATENTES Y MARCAS

ESPAÑA

(21) N.º solicitud: 201001284

(22) Fecha de presentación de la solicitud: 30.09.2010

(32) Fecha de prioridad:

INFORME SOBRE EL ESTADO DE LA TECNICA

(51) **Int. Cl. :** Ver Hoja Adicional

DOCUMENTOS RELEVANTES

Categoría	(56) **Documentos citados**	**Reivindicaciones afectadas**
A	REAL S; POSADAS H; VILLAR E. "L2 cache modeling based on address modification for native co-simulation in systems" International Symposium on Industrial Embedded Systems (SIES),2010 . Pág. 225 - 228; ISBN 978-1-4244-5839-4; ISBN 1-4244-5839-0.	1
X		13
A	PIEPER J J; MELLAN A; PAUL J M;THOMAS D E; KARIM F " HIGH level cache simulation for heterogeneous multiprocessors "Design Automation, Proceedings. 41st, Conference,2004. Pág. 287 - 292, ISBN 978-1-5118-3828-3; ISBN 1-5118-3828-0	1

Categoría de los documentos citados

X: de particular relevancia
Y: de particular relevancia combinado con otro/s de la misma categoría
A: refleja el estado de la técnica
O: referido a divulgación no escrita
P: publicado entre la fecha de prioridad y la de presentación de la solicitud
E: documento anterior, pero publicado después de la fecha de presentación de la solicitud

El presente informe ha sido realizado

☒ para todas las reivindicaciones ☐ para las reivindicaciones nº:

Fecha de realización del informe	**Examinador**	**Página**
21.05.2012	M. Muñoz Sanchez	1/4

CLASIFICACIÓN OBJETO DE LA SOLICITUD

G06F9/455 (2006.01)
G06F12/10 (2006.01)
G06F11/34 (2006.01)

Documentación mínima buscada (sistema de clasificación seguido de los símbolos de clasificación)

G06F

Bases de datos electrónicas consultadas durante la búsqueda (nombre de la base de datos y, si es posible, términos de búsqueda utilizados)

INVENES, EPODOC, WPI, XPMISC, XPI3E, XPIETF, XPIEE, XPESP, NPL, COMPDX

Fecha de Realización de la Opinión Escrita: 21.05.2012

Declaración

Novedad (Art. 6.1 LP 11/1986)	Reivindicaciones	1-12, 14	**SI**
	Reivindicaciones	13	**NO**
Actividad inventiva (Art. 8.1 LP11/1986)	Reivindicaciones	1-12, 14	**SI**
	Reivindicaciones		**NO**

Se considera que la solicitud cumple con el requisito de aplicación industrial. Este requisito fue evaluado durante la fase de examen formal y técnico de la solicitud (Artículo 31.2 Ley 11/1986).

Base de la Opinión.-

La presente opinión se ha realizado sobre la base de la solicitud de patente tal y como se publica.

1. Documentos considerados.-

A continuación se relacionan los documentos pertenecientes al estado de la técnica tomados en consideración para la realización de esta opinión.

Documento	Número Publicación o Identificación	Fecha Publicación
D01	REAL S; POSADAS H; VILLAR E. "L2 cache modeling based on address modification for native co-simulation in systems" International Symposium on Industrial Embedded Systems (SIES),2010 . Pág. 225 - 228; ISBN 978-1-4244-5839-4; ISBN 1-4244-5839-0	07.07.2010
D02	PIEPER J J; MELLAN A; PAUL J M;THOMAS D E; KARIM F " High level cache simulation for heterogeneous multiprocessors "Design Automation, Proceedings. 41st, Conference,2004. Pág. 287 - 292, ISBN 978-1-5118-3828-3; ISBN 1-5118-3828-0	07.07.2004

2. Declaración motivada según los artículos 29.6 y 29.7 del Reglamento de ejecución de la Ley 11/1986, de 20 de marzo, de Patentes sobre la novedad y la actividad inventiva; citas y explicaciones en apoyo de esta declaración

Se considera D01 el documento del estado de la técnica más próximo al objeto de la invención.

Reivindicaciones independientes

Reivindicación 1: Siguiendo la redacción de la reivindicación 1 el documento D01 divulga un método de modelado de una memoria caché para simular el comportamiento de dicha memoria caché en la ejecución de un código software en una plataforma destino que
comprenda dicho procesador, donde dicha simulación se realiza en una plataforma nativa que tiene un procesador diferente del procesador que comprende dicha memoria caché a modelar y donde dicho modelado se realiza mediante la ejecución en dicha plataforma nativa de un código software que se basa en dicho código software a ejecutar en dicha plataforma destino, extendido con información para modelar dicho comportamiento de dicha memoria caché del procesador de la plataforma destino, donde el método comprende las etapas de:

- analizar el código software a ejecutar en la plataforma destino para identificar unos bloques básicos de dicho
- añadir a dicho código anotaciones relativas a la memoria caché a simular, donde dichas anotaciones comprenden información para modelar el efecto de dicha memoria en el procesador destino, obteniéndose un código anotado.

El documento D01 no expone ninguna idea concreta de cómo calcular direcciones de variables.
El documento D02 por su parte expone un método de simulación de memoria caché de alto nivel en el que se hacen anotaciones en el código fuente relativas al comportamiento de la memoria pero no hace referencia al cálculo de direcciones de variables.
Por tanto, el problema técnico expuesto en el documento de la solicitud consistente en el cálculo de las direcciones de las variables para hacer más eficiente la simulación de una memoria caché con anotación de código fuente no se había planteado en el estado de la técnica a fecha de la solicitud de patente y así la reivindicación 1 presentaría actividad inventiva según el artículo 8.1 de la Ley de Patentes.

Reivindicación 13: El sistema reivindicado podría considerarse uno de propósito general dentro del campo de la simulación ya que ninguna de las etapas del método necesita de medios específicos para su realización, como, por ejemplo los que implícitamente se usarían en el procedimiento del documento D01. Por tanto el documento D01 afectaría a la novedad de la reivindicación 13 según el artículo 6.1 de la Ley de Patentes.

Reivindicación 14: El programa informático codifica en instrucciones las etapas que se realizarían en el método. Por tanto la reivindicación 14 presentaría actividad inventiva según el artículo 8.1 de la Ley de Patentes.

Reivindicaciones dependientes

Reivindicaciones 2-12: Estas reivindicaciones dependen de la reivindicación 1 y por tanto también presentarían actividad inventiva según el artículo 8.1 de la Ley de Patentes.