Un simulador de memorias cache multinivel.

Ricardo Almisas, Rafael Paz, Alejandro Linares, Claudio Amaya, José Luis Sevillano(*).

Facultad de Informática. Universidad de Sevilla.
Avda. Reina Mercedes, s/n. 41012, Sevilla.
e-mail: (*) sevi@atc.us.es

Resumen

En este trabajo, se describe un simulador gráfico denominado Simula Cache 1.0, que facilita la realización de prácticas sobre jerarquía de memoria en cursos de arquitectura y estructura de computadores. Está inspirado en el conocido Dinero III [1], aunque diseñado para un entorno W9x-NT, con una interfaz gráfica muy intuitiva y fácil de utilizar. Además, incluye algunas características avanzadas (hasta tres niveles de cache, prebúsqueda, subbloques, etc.) así como parámetros normalmente no incluidos en otros simuladores. Por ello, puede también ser útil para evaluación de sistemas realistas o como validación de modelos de prestaciones.

1. Introducción

Aunque existe un buen número de simuladores de caches y jerarquías de memoria [2-9], es difícil encontrar uno que cubra todos los aspectos requeridos. En general, parecen existir dos aproximaciones diferentes. Por un lado, simuladores orientados a la docencia de los conceptos fundamentales, a la ilustración del funcionamiento interno y al efecto de parámetros básicos de diseño. Estos simuladores suelen tener una interfaz de usuario gráfica, sencilla de utilizar, pero tienen como limitaciones fundamentales el hecho de que no admiten configuraciones complejas (típicamente están limitados a un solo nivel de cache, por ejemplo) y que los resultados ofrecidos suelen reducirse al *miss rate* (razón de fallos) de un conjunto de accesos. Esto es lógico, puesto que estarían especialmente indicados para asignaturas introductorias o básicas, en las que incluir más detalles o configuraciones complejas resulta casi contraproducente. En efecto, eso requeriría dedicar mucho tiempo de aula en asignaturas en las que el estudio de la jerarquía de memoria debe ser necesariamente breve. Ejemplos de este tipo de simuladores serían VC [7] (ejecutado en una página Web), *Visual Cache*, diseñado en nuestro grupo [6], CVT [4] (aunque este simulador es bastante más ambicioso), y otros [5,8].

En el otro extremo estarían simuladores que aunque también pueden ser usados como herramienta docente, tienen una serie de características que los hacen más adecuados al estudio de aspectos más avanzados o incluso para la realización de pequeños proyectos de investigación. En la mayoría de estos simuladores, la interfaz con el usuario suele ser pobre, normalmente requiriendo dar la configuración en línea de comandos o en ficheros. Este es el caso del popular Dinero III [1], y su versión más reciente Dinero IV [3], el *Argent Cache Simulator* [2], *Acme* [9], etc.

El simulador presentado aquí, denominado SimulaCache1.0, pretende situarse en un punto intermedio entre estos dos extremos. Mantiene una interfaz de usuario gráfica e intuitiva que permitiría su uso en prácticas sin demasiado esfuerzo por parte de los alumnos, pero incluye también algunas características (como caches multinivel), que permitirían su uso en digamos estudios más avanzados como el comportamiento de sistemas reales ante aplicaciones reales, validación de modelos de prestaciones, etc. Estas características serán descritas con más detalle en las siguientes secciones.

2. Formato de trazas.

La primera decisión a la hora de diseñar un simulador de jerarquías de memoria es la elección del formato de las trazas. En efecto, la inmensa mayoría de los simuladores son *trace-driven* [13],

es decir, la entrada del simulador es un fichero de trazas de memoria, cuyos registros poseen información referente a la dirección de memoria accedida por el procesador, al tipo de acceso del que se trata (lectura o en escritura) y el tipo de información al que se está accediendo (Datos o Instrucciones). Esto es lo habitual en los estudios de prestaciones de sistemas de memoria. No interesa que la entrada de datos sea código máquina, que necesariamente es dependiente del procesador, ni código de alto nivel, donde es difícil aislar los accesos al sistema de memoria (aunque por ejemplo CVT [4] permite simular pequeños bucles en Fortran).

Sin embargo, la longitud de los ficheros de trazas es mucho mayor que la del código fuente original, pues en los ficheros de trazas de memoria los bucles se "desenrollan". Además, en sistemas realistas (y por tanto complejos), para obtener estadísticas fiables deben simularse millones de referencias a memoria, necesitando cada una de ellas hasta 10 bytes, lo que nos lleva a ficheros del orden de decenas de megabytes [10]. Muchos de los esfuerzos en investigación van orientados a desarrollar técnicas para superar este problema, como la compresión [10] o el muestreo estadístico de los ficheros de trazas [11].

Sin embargo en nuestro caso las trazas deben ser en lo posible autoexplicativas. Es decir, si se pretende su uso docente, un simple vistazo de la traza debe dar una idea de los accesos que se están realizando en determinados puntos del programa, sobre todo si interesa comprobar el funcionamiento en determinados puntos de la traza. Esto impide usar formatos comprimidos como PDATS. Otro formato muy extendido, ETCH [12] usa códigos numéricos e incluye casos avanzados como accesos múltiples.

Para este caso, no hemos optado por un formato único, sino que mantenemos tres posibles formatos, permitiendo la traducción de uno a otro. Todos tienen en común que son ficheros de texto donde cada línea se corresponde con un acceso, permitiendo al alumno identificar fácilmente los accesos básicos.

2.1.- Trazas DIN.

Este es el formato que reconoce el programa Dinero en sus versiones tercera y cuarta [1,3]. Está compuesto por dos campos. El primer campo está formado por un carácter que indica el tipo de acceso a la memoria caché. El segundo campo indica la dirección del acceso y se presenta en formato hexadecimal, con la opción de tener como prefijo "0x", siendo de longitud máxima de 32 bits. En el caso de que fuera menor de 32 bits se le añadirían ceros a la izquierda.

Los diferentes caracteres que permite este formato para indicar el tipo de acceso son:

- 0 : lectura de datos en la caché.
- 1: escritura de datos en la caché.
- 2: lectura de instrucciones en la caché.
- 3: misc, lectura de datos en la caché que no genera prebúsquedas (en caso de ser posible).
- 4: invalida una posición de la caché.
- 5: copyback, realiza una escritura en el siguiente nivel de memoria si la posición referenciada estuviera "sucia".

En cuanto al tamaño de los accesos que se realizan a memoria siempre son de 4 bytes.

2.2.- Trazas X-DIN.

Este formato, usado por el programa Dinero IV [3], es una extensión del formato anterior. La principal diferencia es que en éste el tamaño del acceso está indicado expresamente como un campo más. Así, tenemos tres campos, el primero está compuesto por un carácter e indica el tipo de acceso a la memoria caché que se realiza. El segundo campo indica la dirección del acceso y estará en formato hexadecimal, con la opción de tener como prefijo "0x", y será de longitud máxima de 32 bits. En el caso de que fuera menor de 32 bits se le añadirían ceros a la izquierda. El tercer campo indica el tamaño del acceso en bytes, este campo estará en formato hexadecimal. También puede tener con carácter opcional el prefijo "0x".

Los diferentes tipos de acceso que permite indicar este formato son: R (lectura), W (escritura), I (instrucción), M (misc), V (invalidación) y C (copyback), equivalentes a los accesos DIN 0, 1, 2, 3, 4 y 5, respectivamente.

2.3 Trazas Visual Caché 1.0.

Por último, también se permite el uso de un formato de trazas utilizado por otro simulador diseñado en nuestro grupo, denominado Visual Cache [6], y que está orientado fundamentalmente

a la ilustración del proceso interno que tiene lugar en caches sencillas. Aquí no interesa tanto el resultado estadístico final como la visualización de la evolución paso a paso de la ejecución de una traza, por lo que es más importante si cabe que la traza sea muy fácil de entender. Los campos son:

- Un carácter indicativo del tipo de acceso que se está realizando (L ⇇Lectura o W ⇇ Escritura).
- Un campo con la dirección referenciada (en hexadecimal, 8 dígitos).
- Un carácter indicativo de si el acceso es a un dato o a una instrucción (D⇇ Dato o I ⇇ Instrucción).
- Finalmente, un campo de 2 dígitos decimales con el tamaño del dato accedido en bytes.

Por ejemplo, L 0x00000FF0 I 04. Este formato de trazas puede considerarse como un subconjunto de formatos más complejos.

En realidad estos formatos son similares, lo que permite una traducción muy sencilla. Nuestra aplicación permite la traducción de trazas de un formato a otro (con las limitaciones lógicas).

3. Características simuladas.

El simulador incluye por supuesto todas las características habituales en cualquier simulador de cache: tamaño de la cache, número de palabras por bloque, modo de ubicación de los datos (Correspondencia Directa, Por vías o Totalmente asociativa), tipo de escritura (Copy Back o Write Through), modo de ubicación de la escritura (Write Allocate o No Write Allocate), primer nivel de caché unificada o separada para datos e instrucciones y algoritmo de reemplazo (LRU, FIFO o Aleatorio) cuando existe asociatividad.

Pero además de estos parámetros básicos, se incluyen otros que permiten su utilización por ejemplo en cursos más avanzados, como:

- hasta tres niveles de caché en la jerarquía de memoria, siendo cada uno de ellos configurable independientemente.
- permite dividir los bloques de caché en subbloques.
- permite realizar prebúsquedas de bloques o subbloques, dependiendo de si se ha elegido dividir los bloques en subbloques o no. Pudiéndose configurar las prebúsquedas para realizarse siempre que se simule un acceso de tipo instrucción o de lectura de datos, o para que se realizaran las prebúsquedas sólo en el caso de que ocurriese un fallo en el acceso para los tipos de accesos anteriormente comentados.
- permite configurar el tamaño, en bits, de la palabra que usará el sistema de memoria durante la simulación.
- permite configurar el ancho de los buses de memoria que comunican a cada nivel de cache con su nivel inmediatamente inferior.

Todas estas características pueden configurarse mediante ventanas interactivas (ver figura 1), pudiéndose también almacenar en ficheros (.CFG), lo que posibilita disponer de una especie de “librería de organizaciones de cache” de sistemas reales.

4. Resultados.

En lo que respecta a los resultados que ofrece esta aplicación tras la simulación, tendríamos:

- el número de accesos totales y por tipo de acceso, con sus respectivos porcentajes.
- el número de fallos totales y por tipo de acceso, con sus respectivos porcentajes.
- el número de bytes leídos en el siguiente nivel y el número de bytes escritos en el siguiente nivel.
- el número de bytes que deben ser escritos en el siguiente nivel al terminar la simulación y su porcentaje respecto al número total de bytes escritos en el siguiente nivel.

el número de bytes medio escritos en el siguiente nivel por cada acceso de tipo escritura simulado. También han sido añadidos una serie de resultados que se han considerado interesantes, y que no son habituales en otros simuladores de la literatura. Uno de los motivos ha sido que el simulador ha sido usado como validación de modelos analíticos simples sobre memorias cache desarrollados por nuestro grupo [14]. Algunos de estos parámetros son:

- el número de accesos de más simulados, distinguiendo los que hayan sido motivados por accesos desalineados o porque el ancho del bus no podía acceder en un solo acceso a todos los datos requeridos en un momento dado.
- el número medio de lecturas y escrituras realizados por cada bloque o subbloque de la caché, dependiendo de si la caché simulada tuviera los bloques divididos en subbloques o no,

en relación al número total de bloques o subbloques (también se proporciona, a modo de información, el número de bloques o subbloques total).

- el número medio de lecturas y escrituras realizados por cada bloque o subbloque de la caché en relación al número de bloques o subbloques a los que se ha accedido durante la simulación (también se proporciona este número de bloques o subbloques total).

A los ficheros de traza que son cargados en una ventana se les podrá asignar puntos de corte (*breakpoints*), para los cuales se muestran los resultados de la simulación hasta ese momento, además, claro está, de mostrar los resultados obtenidos al final de la simulación.

Finalmente, está la parte de comparativas de los resultados obtenidos. Si los resultados mostrados al final de cualquier simulación son guardados en un archivo (.RST), después estos archivos podrán ser utilizados para realizar comparativas de resultados entre ellos. Para ello, se permite elegir qué ficheros de resultados se desean mostrar en la comparativa y cuáles serán comparados. Además puede elegirse qué parámetros se desean comparar (ver figura 2). El número total de parámetros que ofrece el simulador es grande, y a menudo sólo interesan unos pocos en concreto, por lo que no tiene mucho sentido presentar todos por defecto. Tras esto la aplicación muestra una tabla, en la que por columna se pueden ver los resultados de cada fichero y por filas se podrán comparar los resultados de unos ficheros con otros (figura 3).

5. Conclusión

En este trabajo se ha descrito un simulador que permite evaluar el comportamiento de un sistema de memorias cache ante la ejecución de determinadas trazas de memoria. Se admiten tres formatos de traza, pudiéndose traducir ficheros de un formato a otro. Las características simuladas incluyen aspectos avanzados como pueden ser varios niveles de cache (hasta tres), prebúsquedas, sub-bloques, etc. Las configuraciones pueden editarse mediante ventanas interactivas, y salvarse como fichero. Como resultado de la simulación, se ofrece un buen número de parámetros que pueden ser salvados, editados e incluso comparados desde la propia aplicación. El programa está escrito usando Visual Basic, y se ejecuta bajo Windows 9x o NT.

Referencias

[1] J.L. Hennessy, D.A. Patterson “Computer Architecture: A Quantitative Approach” (2ª Ed.). Morgan Kaufmann, 1996.

[2] http://papi.ee.duke.edu/argent/argent.htm.

[3]http://www.neci.nj.nec.com/homepages/edler/d4/

[4] E. Deijl, G. Kanbier, O. Temam, E.D. Granston, “A Cache Visualization Tool”. *Computer*, vol. 30, no. 7, July 1997.

[5] R.N. Ibbett, Computer architecture visualisation techniques, Microprocessors and Microsystems 23-5 (1999) 291-300

[6] M.A. Rodríguez *et al.* “Un simulador gráfico para la enseñanza de memorias cache”. IV Congreso sobre Tecnologías aplicadas a la Enseñanza de la Electrónica. Barcelona, Septiembre de 2000.

[7]http://www.cc.gatech.edu/fac/Bill.Leahy/vcsa/vcs.html

[8]http://www.ece.gatech.edu/research/labs/reveng/cachesim/

[9] http://atanasoff.nmsu.edu/~acme/acs.html.

[10] E.E. Johnson, J. Ha, “PDATS: Lossless Address Trace Compression for Reducing File Size and Access Time”. Proc. *IEEE Int. Phoenix Conf. Comp. and Commun.*, 1994.

[11] T.M. Conte, M.A. Hirsch, W.W. Hwu, “Combining Trace Sampling with Single Pass Methods for Efficient Cache Simulation”. *IEEE Trans. Comp.*, vol. 47, no. 6, June 1998.

[12]http://memsys.cs.washington.edu/memsys/html/etch.html.

[13] R.A. Uhlig, T.N. Mudge, Trace-Driven Memory Simulation: A Survey, ACM Computing Surveys, 29-2 (1997) 129-170.

[14] F. Díaz *et al.* “A Dynamic Equilibrium based Model for Caches”. En proceso de revisión para *Microprocessors and Microsystems.*