UN SIMULADOR GRÁFICO PARA LA ENSEÑANZA DE MEMORIAS CACHE

M.A. RODRÍGUEZ JODAR, D. CASCADO CABALLERO, L. MIRÓ AMARANTE, J.L. SEVILLANO RAMOS, F. DÍAZ DEL RIO
Facultad de Informática. Universidad de Sevilla. Avda. Reina Mercedes s/n. 41012. SEVILLA. España. Tfno: 954556470. Fax: 954552759

En el presente trabajo se describe un simulador gráfico denominado Visual Cache 1.0, cuyo objetivo fundamental consiste en ilustrar el funcionamiento de las memorias cache, permitiendo realizar un seguimiento del proceso que internamente tiene lugar en estos dispositivos. Esta herramienta es muy útil para valorar y comprender las distintas alternativas de diseño: caches separadas o unificadas, grado de asociatividad, política de escritura, algoritmo de reemplazo, etc.

1. Introducción

El objetivo de este trabajo es desarrollar un simulador gráfico que facilite la realización de prácticas sobre memorias cache en cursos introductorios de arquitectura y estructura de computadores. En principio, puede parecer innecesario construir un nuevo simulador de caches dado el gran número disponible [1,3,4,5]. Sin embargo, debe tenerse en cuenta que en estos cursos lo realmente importante es que el alumno sea capaz de comprender los conceptos básicos, mientras que incluir aspectos avanzados de diseño (como caches multinivel, comportamiento ante aplicaciones reales, validación de modelos de prestaciones, protocolos de coherencia de cachés en multiprocesadores, etc.) resulta casi contraproducente, pues se requeriría dedicar mucho tiempo de aula en asignaturas en las que el estudio de la jerarquía de memoria debe ser necesariamente breve. Un estudio cuidadoso de los simuladores existentes demuestra que la mayoría de ellos presentan, desde este punto de vista, muchos inconvenientes.

En primer lugar, la interfaz con el usuario suele ser pobre, normalmente requiriendo dar la configuración en línea de comandos o en ficheros. Por ejemplo, en el *Argent Cache Simulator* [3], la simulación de una caché de correspondencia directa para instrucciones, y una caché totalmente asociativa para datos requeriría el comando:

Argent /c8 /lc16 /ac1 /d8 /ld8 /a1024.

Otro simulador muy popular, Dinero IV [4], sufre de los mismos inconvenientes, que incluyen el hecho de que la ejecución paso a paso sea especialmente engorrosa. En resumen, el principal problema es que el objetivo de casi todos ellos es estudiar el efecto en las prestaciones de distintas alternativas de diseño, y no ilustrar los principios básicos de funcionamiento. Por ello, la mayoría sólo ofrece resultados globales sobre la razón de fallos

media (*miss ratio*) para comparar prestaciones.

Sin embargo, nuestro objetivo es que la configuración y ejecución del simulador pueda realizarse de forma visual e intuitiva, de forma que el alumno controle e inspeccione el funcionamiento del cache. Es decir, se pretende no sólo simular el funcionamiento del cache, sino "visualizar" gráficamente qué está ocurriendo exactamente "dentro" del sistema, un objetivo similar al del CVT [5]. Esto permite por ejemplo estudiar la optimización de pequeños trozos de código para un cache dado, un problema mucho más cercano y adaptado a nuestro entorno.

2. Formato de las trazas.

Otra dificultad en muchos simuladores de la literatura es el formato de las trazas. En efecto, la inmensa mayoría de los simuladores son *trace-driven*, 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 CVT [5] 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 [1]. Muchos de los esfuerzos en investigación van orientados a desarrollar técnicas para superar este problema, como la compresión [1] o el muestreo estadístico de los ficheros de trazas [2].

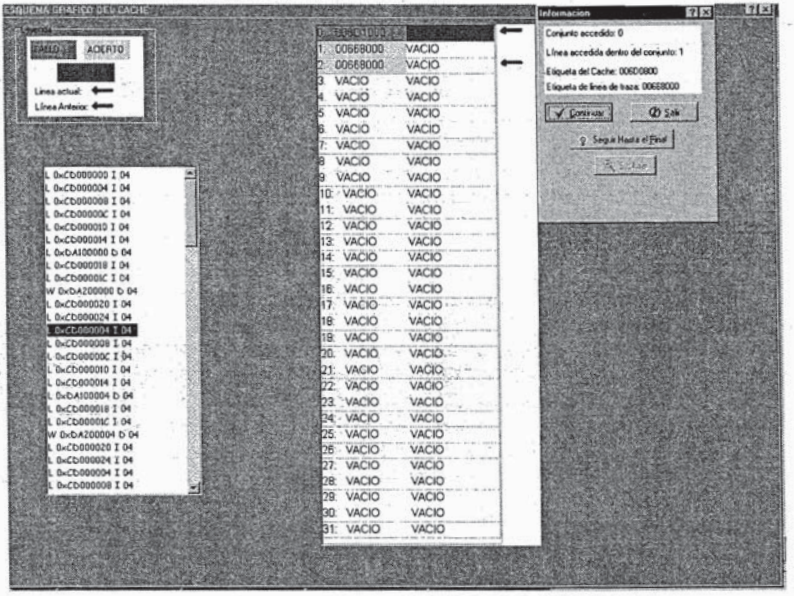
Sin embargo en nuestro caso las trazas deben ser en lo posible autoexplicativas, lo que impide usar formatos comprimidos como PDATS. Otro formato muy extendido, ETCH [6] usa códigos numéricos e incluye casos avanzados como accesos múltiples. Nosotros hemos preferido un formato ASCII que permitiera al alumno identificar fácilmente los accesos básicos. Así, las trazas usadas en Visual Caché 1.0 tienen el siguiente formato:

- 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 como ETCH, lo que permite una traducción muy sencilla.

El usuario puede desde la aplicación editar y modificar el fichero de trazas para probar distintas secuencias de acceso que ilustren los aspectos deseados. La simulación de la traza puede realizarse de una vez o, lo que normalmente resulta mucho más interesante, paso a paso para observar el funcionamiento interno de la cache. Para trazas muy largas, pueden también insertarse puntos de ruptura (*breakpoints*). El estado interno de la cache se visualiza gráficamente, mostrando todas las líneas e indicando si inicialmente están vacías, si ocurre un acierto o un fallo en el acceso actual, etc. Por supuesto, también se obtienen las estadísticas finales, que pueden almacenarse en fichero para realizar futuras comparaciones. Todas las características del sistema pueden configurarse mediante ventanas interactivas, pudiéndose también almacenar en ficheros, lo que posibilita disponer de una especie de "librería de organizaciones de cache" de los procesadores más populares. Sin embargo, y dado su carácter didáctico, sólo incluye caches de un solo nivel.

En la figura se muestra un ejemplo de simulación de un cache unificado asociativo de 2 vías. Se ilustran con distintos colores los aciertos, los fallos y las "expulsiones" de líneas cuando debe sustituirse una línea por otra. Las líneas con bit de validez no activo se muestran como "vacías", lo que resulta más intuitivo.



3. Una práctica ejemplo.

Para ilustrar el uso del simulador, describiremos brevemente un caso práctico inspirado en el problema 5.2 de [7]. En este caso nos limitaremos al estudio de los accesos a datos, suponiendo caches separadas. El enunciado sería:

Consideremos el siguiente bucle:

```
for(i=1;i<=MAX;i=i+ZANCADA)
        t=t+x[i];
```

Nótese que el acceso al array no es elemento a elemento, sino que hay un parámetro zancada que establece el salto entre elementos accedidos. Si suponemos que las variables t e i están asignadas a registros, los únicos accesos al cache de datos son las lecturas de x[i]. El tamaño MAX puede tomar el valor que más convenga. Considere un cache de datos de correspondencia directa de 1Kb con líneas de 32 bytes. Se pide generar una traza equivalente al código anterior con zancada tal que el miss rate (la razón de fallos) sea 0.25.

Si suponemos un procesador de 32 bits (direcciones y datos de 4 bytes), cada línea tendrá 8 palabras. Al leer al primer elemento, se producirá un fallo de cache, lo que traerá la línea correcta. Una zancada de 2 hará que los siguientes 3 accesos acierten en la línea que acabamos de traer. Se consigue así un acierto y tres fallos por cada cuatro accesos, lo que nos da el MR=0.25. Esto ilustra cómo un tamaño mayor de línea aprovecha la localidad de referencia espacial. La traza pedida será por tanto:

```
L 0xDA000000 D 04
L 0xDA000008 D 04
L 0xDA000010 D 04
L 0xDA000018 D 04
....
```

4. Conclusiones

Se ha descrito un simulador gráfico destinado a la realización de prácticas en cursos introductorios de arquitectura de computadores. El simulador, denominado Visual Cache, muestra gráficamente la ejecución de una traza, permitiendo comprobar la evolución interna de una memoria cache. Todos los parámetros son configurables visualmente. La aplicación está en castellano y dispone de una completa ayuda. Nuestra experiencia en el pasado curso académico ha sido un éxito, con muy buena aceptación por parte de los alumnos.

Referencias

[1] 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.

[2] 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.

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

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

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

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

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