COMPARTICION DE ESPACIO ENTRE INSTRUCCIONES Y DATOS EN LA MEMORIA CACHE

Rodriguez C., Viñals V., Labarta J.

Facultad de Informática de Barcelona.
Jordi Girona Salgado 34, 08034 Barcelona.

Resumen. Un gran número de computadores en la actualidad usan memoria cache para adaptar la velocidad de la memoria a la del procesador, o bien reducir el tráfico en sistemas multiprocesadores. En la actualidad un tema discutido es la conveniencia de utilizar una sola cache o de descomponerla en una de instrucciones y otra de datos. En este artículo se estudia el comportamiento de instrucciones y datos en una cache unificada bajo un algorítmo de reemplazo LRU global y se compara con un sistema con equipartición estática del espacio y LRU local.

Trabajo subvencionado por la CAICYT (ref.1739/82)

INTRODUCCION.

El concepto de memoria cache fue introducido a nivel teórico por M. Wilkes [Wilk65] en 1965, siendo el IBM 360/85 el primer computador donde fue implementado en el año 1968. Desde entonces han aparecido estudios basados en modelos matemáticos y en simulaciones de trazas de programas reales, con el objetivo fundamental de presentar alternativas que aumenten el rendimiento global de la memoria cache en sistemas uni o multiprocesador. Una de las técnicas propuestas para sistemas uniprocesador consiste en almacenar en caches separadas instrucciones y/o datos.

Podemos dividir a los computadores que utilizan esta técnica en cuatro categorías: (1) Grandes computadores científicos [CRAY79, CDC81], (2) Sistemas con grandes prestaciones [DATA80, AMDA82, SI78], (3) Microprocesadores CISC [Macg84], (4) Máquinas RISC [Patt83, Radi82].

La existencia de caches separadas permite paralelismo en el acceso a datos e instrucciones, así como una mayor proximidad física entre la cache y los elementos del procesador. En el mejor de los casos esto proporciona un incremento del ancho de banda efectivo del sistema y una disminución del tiempo de acceso. Otras ventajas adicionales son: (1) Mayor sencillez de la cache de instrucciones en arquitecturas que no permiten código automodificable, (2) Adaptación de los parámetros de la cache a las características particulares de datos e instrucciones.

La desventaja fundamental de la separación consiste en un eventual desaprovechamiento del espacio que se

instrucciones en una cache unificada es variable. A este fenómeno se le llama reasignación. La reasignación es función del tiempo, de la capacidad de la cache unificada y de las características de localidad de datos e instrucciones.

Este tema se ha estudiado en la memoria principal de los computadores multiprogramados, cuando los procesos compiten de forma libre por la posesión de páginas. Uno de los modelos más completos se encuentra en [Smit80], donde la reasignación se calcula en base a la tasa de fallos de los procesos individuales y el tiempo de transferencia entre memoria secundaria y principal. En todos ellos se reconoce la superioridad, en cuanto aciertos, de la aplicación global del algoritmo de reemplazo LRU frente a su aplicación local.

En nuestro contexto de un solo procesador, aún cuando estuviera multiprogramado, no tiene sentido introducir el tiempo de reemplazo en el modelo, ya que un fallo en datos o instrucciones implica la espera del procesador.

Definiciones y objetivos.

La notación utilizada es la siguiente:

$f_u(C_u)$ es la probabilidad de fallo de la traza en la cache unificada de tamaño C_u.

$f_{ui}(C_u)$, $f_{ud}(C_u)$ es la probabilidad de fallo de instrucciones/datos en la cache unificada de tamaño C_u.

$f_{si}(c)$, $f_{sd}(c)$ es la probabilidad de fallo de instrucciones/datos en caches separadas de tamaño c.

$C_i(C_u)$, $C_d(C_u)$ es el número de bloques ocupados por instrucciones/datos en la cache unificada de tamaño C_u.

R_i es el cociente entre el número de referencias a instrucciones y a datos.

$d_i(C_u)$,$d_d(C_u)$ es la probabilidad de encontrar un bloque de instrucciones/datos a distancia C_u en la cache unificada.

t_i,t_d es el número de referencias a instrucciones/datos en un intervalo T, donde $T = t_i+t_d$.

El objetivo perseguido por el modelo consiste en obtener el comportamiento de la cache unificada a partir del comportamiento de las caches de datos e instrucciones por separado, es decir, la reconstrucción de $f_u(C_u)$ y de $C_i(C_u)$ a partir de $f_{si}(c)$, $f_{sd}(c)$ y R_i.

Otro de los objetivos es comprender la relación que existe entre $f_{ui}(C_u)$, $f_{ud}(C_u)$ y $f_{si}(C_u/2)$, $f_{sd}(C_u/2)$.

En la figura 1 se da una descripción gráfica del proceso de simulación, modelización y validación.

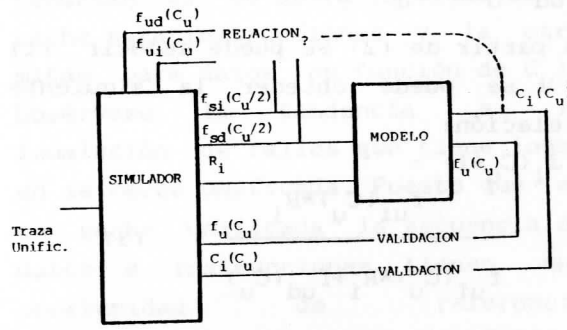

Fig.1. Proceso de Simulación.

Descripción del modelo.

Examinando los resultados previos que han conducido al modelo, se ha encontrado la siguiente relacion:

$$\frac{d_i(C_u+a)}{d_d(C_u+a)} = \frac{f_{ui}(C_u)}{f_{ud}(C_u)} * R_i \quad (1)$$

donde 'a' indica una cierta incertidumbre en la medida. Para

traduce en una relación de aciertos más baja.

Estas ventajas y desventajas han sido estudiadas parcialmente en [Smit82, Smit83].

En [Smit85] se estudian un elevado número de trazas de distintas arquitecturas que orientan sobre la elección de los distintos parámetros en el diseño de una cache unificada, así como de una cache separada para datos y/o intrucciones.

La principal aportación de este trabajo es presentar un modelo matemático sencillo, que explica el reparto dinámico de espacio en una cache unificada completamente asociativa. Con este modelo se puede predecir la posible ganancia o pérdida de aciertos de la cache unificada con respecto a la cache separada, y el espacio ocupado por datos e instrucciones en la cache unificada.

Estos resultados, de por sí clarificadores, abren la puerta a futuros estudios sobre estrategias de reparto controlado de la cache en sistemas uniprocesador, y la optimización de las memorias cache compartidas en sistemas multiprocesador.

SIMULACION.

Descripción de las trazas.

Las simulaciones se han realizado sobre tres trazas reales de un computador PDP-11/60: La traza 1 es un programa escrito en fortran que efectua una FFT de 1024 puntos; la traza 2 pertenece a la ejecución del compilador de fortran F4P; la traza 3 es un programa escrito en fortran que ejecuta una simulación discreta de memorias rápidas con acceso multiplexado.

El simulador, descrito a continuación, efectua la separación de datos e instrucciones en bloques disjuntos, considerando los datos inmediatos y las direcciones absolutas como parte del contenido de la cache de instrucciones.

Descripción del simulador.

El simulador obtiene, a partir de la traza de un programa, la relación de fallos de instrucciones y datos en caches separadas, la relación de fallos de instrucciones y datos en la cache unificada, el porcentaje de ocupación de datos e instrucciones en la cache unificada y el porcentaje de fallos en la cache unificada.

Los parámetros del simulador son:
- Tamaño de la cache unificada variable entre 0 y 4Kbytes.
- Tamaño de bloque: 8 bytes.
- Mapeo completamente asociativo.
- Algoritmo de reemplazo LRU.
- Copy Back (reescritura).
- Fetch sobre demanda.

Las medidas se han efectuado a partir de las 120000 primeras referencias hasta un total de 220000 (técnica Warm Start). El porcentaje de referencias a instrucciones en las últimas 100000 referencias es el siguiente: traza 1, 50.3; traza 2, 65.9; traza 3, 69.9.

MODELO DE REASIGNACION.

Estudios previos sobre reasignación.

El espacio asignado a datos e

explicar esta relación y fijar 'a' proponemos el siguiente razonamiento. En equilibrio, la suma de todos los bloques introducidos en la cache unificada deben ser igual a la suma de todos los bloques expulsados. Para un número arbitrario de referencias a datos e instrucciones T debe cumplirse: (1) el número de bloques de instrucciones introducido es $f_{ui}(C_u) * t_i$, (2) el número de bloques de datos introducidos es $f_{ud}(C_u) * t_d$, (3) el número de bloques de instrucciones expulsados es $d_i(C_u+1) * T * (f_{ui}(C_u)+f_{ud}(C_u))$, (4) el número de bloques de datos expulsados es $d_d(C_u+1) * T * (f_{ui}(C_u) + f_{ud}(C_u))$. Si suponemos que al principio y al final del intérvalo T la asignación de datos e instrucciones no ha variado o ha variado ligeramente alrededor de una media [Smit80], podemos afirmar:

$$d_i(C_u+1) * T * (f_{ui}(C_u)+f_{ud}(C_u)) = f_{ui}(C_u) * t_i$$

$$d_d(C_u+1) * T * (f_{ui}(C_u)+f_{ud}(C_u)) = f_{ud}(C_u) * t_d \qquad (2)$$

A partir de (2) se puede deducir (1) y se puede obtener la siguiente relación:

$$d_i(C_u+1) = \frac{f_{ui}(C_u)*R_i}{f_{ui}(C_u)*R_i+f_{ud}(C_u)} \qquad (3)$$

La asignación media de instrucciones para una capacidad C_u es:

$$C_i(C_u) = \sum_{K=1}^{C_u} d_i(k) \qquad (4)$$

sin embargo, la asignación instantánea oscilará en torno a este valor medio. Aún cuando la relación de fallos sólo depende de la asignación instantánea, podemos escribir la siguiente aproximación:

$$f_{ui}(C_u) = f_{si}(C_i(C_u)) \qquad (5)$$

En esta aproximación estamos introduciendo un error que será tanto mayor cuanto menos lineal sea la curva de fallos y mayor varianza tenga la asignación.

De (3), (4) y (5) se deduce una expresión para $C_i(C_u)$ que constituye nuestro modelo de reasignación:

$$C_i(C_u) = \sum_{K=1}^{C_u} \frac{f_{si}(C_i(k-1))*R_i}{R_i*f_{si}(C_i(k-1))+f_{sd}(C_d(k-1))} \qquad (6)$$

que puede resolverse iterativamente aplicando:

$$C_i(C_u) = C_i(C_u-1) + \frac{f_{si}(C_i(C_u-1))*R_i}{R_i*f_{si}(C_i(C_u-1))+f_{sd}(C_d(C_u-1))} \qquad (7)$$

siendo $C_i(0) = 0$.

La probabilidad de fallo de la traza unificada puede calcularse de la siguiente forma:

$$f_u(C_u) = f_{ui}(C_i(C_u)) * P_i + f_{ud}(C_d(C_u) * (1 - P_i) \qquad (8)$$

donde $P_i = R_i / (R_i + R_d)$. El error cometido por este cálculo es debido a las razones anteriores de no linealidad y varianza, así como la interpolación que debe efectuarse en el cálculo de f_{ui} y f_{ud} cuando C_i y C_d no son enteras.

VALIDACION DEL MODELO.

En las figuras 2, 3, 4 se muestran las asignaciones para las instrucciones $(C_i(C_u)/C_u*100)$ simuladas y calculadas por el modelo para las tres trazas. El error observado es del orden de un 5 por ciento de la capacidad unificada en el peor de los casos.

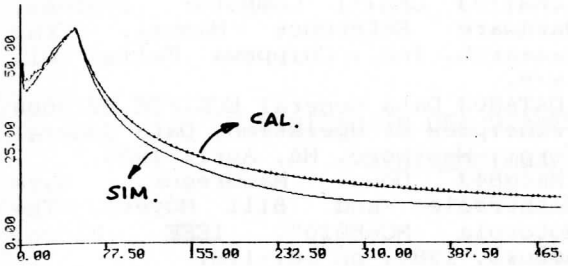

Fig.2. $C_i(C_u)$ para la traza 1.

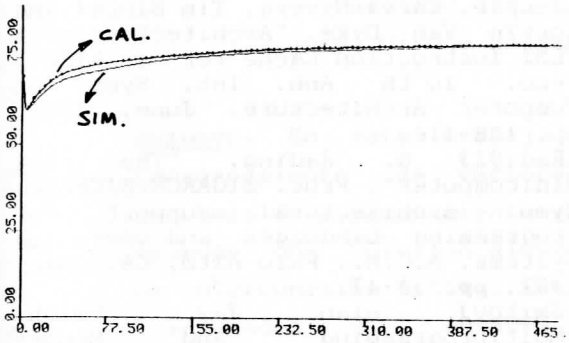

Fig.3. $C_i(C_u)$ para la traza 2.

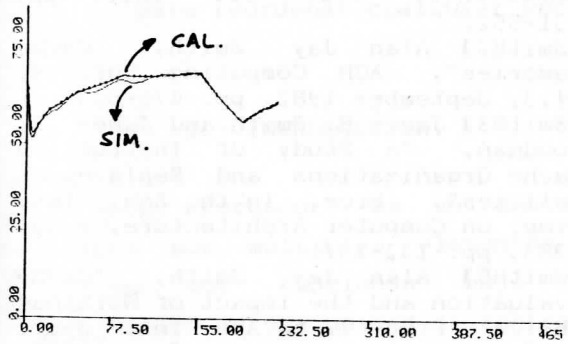

Fig.4. $C_i(C_u)$ para la traza 3.

En las figuras 5, 6, 7 se muestran las relaciones de fallos totales $f_u(C_u)$ simuladas y calculadas por el modelo para las tres trazas. Obsérvese la correlación existente entre el error y la curvatura por el fenómeno descrito con anterioridad.

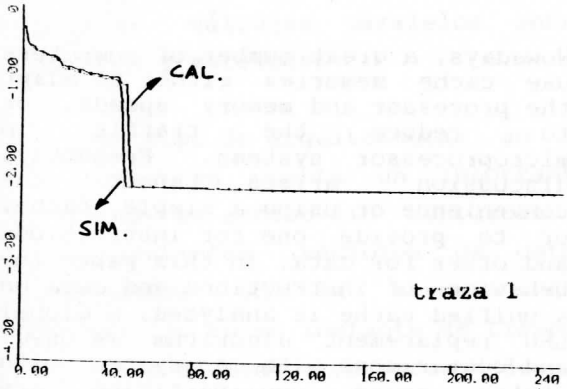

Fig.5. $f_u(C_u)$ simuladas/calculadas

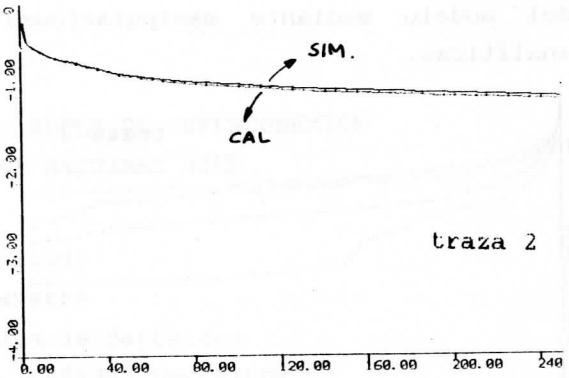

Fig.6. $f_u(C_u)$ simuladas/calculadas

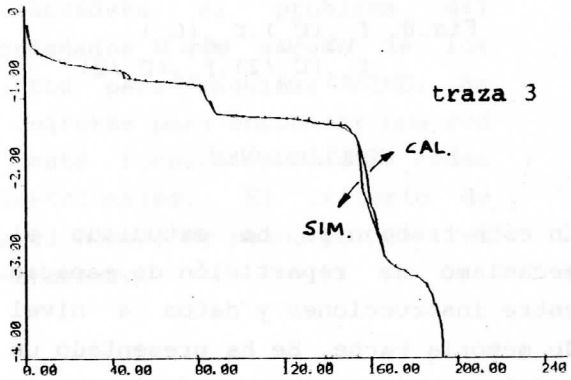

Fig.7. $f_u(C_u)$ simuladas/calculadas

En la figura 8 se comparan los fallos individuales de datos e instrucciones en la cache unificada con los que se tendrían si se usara la mitad de la cache para instrucciones y la otra mitad para datos (en función de C_u). Obsérvese la tendencia a la igualación de fallos que tiene lugar en la cache unificada. Puesto que en la cache unificada la secuencia de datos e instrucciones tienen una oportunidad de referencia independiente de su tasa de fallos, la secuencia de referencias más local, introducirá bloques en la cache unificada a velocidad menor, resultando por tanto que la menos local consigue una asignación mayor que $C_u/2$. El resultado de este mecanismo de realimentación será una reducción de la diferencia relativa de los fallos de datos e instrucciones unificados. Esta conclusión puede obtenerse a través

del modelo mediante manipulaciones analíticas.

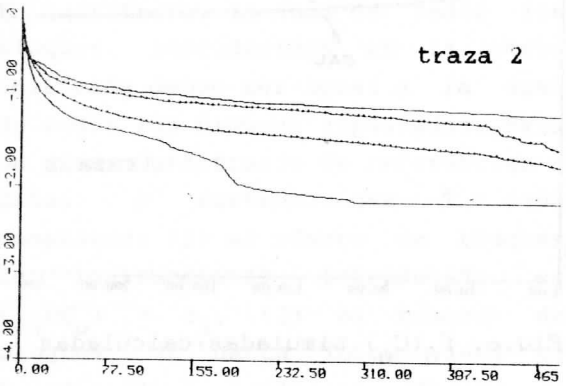

Fig.8. $f_{ui}(C_u), f_{ud}(C_u)$
$f_{si}(C_u/2), f_{sd}(C_u/2)$

CONCLUSIONES.

En este trabajo se ha estudiado el mecanismo de repartición de espacio entre instrucciones y datos a nivel de memoria cache. Se ha presentado un modelo que permite predecir este comportamiento a partir de las curvas de fallos de instrucciones y datos por separado.

Como aportación importante se ha demostrado que la repartición de espacio esta controlada por la relación de fallos entre instrucciones y datos.

Estos resultados, de por sí clarificadores, abren la puerta a futuros estudios sobre estratégias de reparto controlado del espacio de la cache en sistemas uniprocesador, y la optimización de las memorias cache compartidas en sistemas multiprocesador.

REFERENCIAS.

[AMDA82] Amdahl 580 Technical Introduction, Amdahl Corp., Sunnyvale, CA, 1982.

[CDC81] CDC CYBER 200 Model 205 Computer Systems Hardware Reference Manual, Control Data Corp., Arden Hills, MN, 1981.

[CRAY79] CRAY-1 Computer Systems, Hardware Reference Manual, Cray Research, Inc., Chippewa Falls, WI, 1979.

[DATA80] Data General ECLIPSE MV/8000 Principles of Operation, Data General Corp., Westboro, MA, April 1980.

[Macg84] Doug MacGregor, Dave Mothersole and Bill Moyer, "The Motorola MC68020", IEEE Micro, August, 1984, pp 101-118.

[Patt83] David A. Patterson, Phil Garrison, Mark Hill, Dimitris Lioupis, Chris Nyverg, Tim Sippel and Korvin Van Dyke, "Architecture of a VLSI Instruction Cache for a RISC", Proc. 10'th Ann. Int. Symp. on Computer Architecture, June, 1983, pp. 108-116.

[Radi82] G. Rading, "The 801 Minicomputer", Proc. SIGARCH/SIGPLAN, Symp. Architectural Support for Programming Languages and Operating Systems, A.C.M., Palo Alto, CA, Mar, 1982, pp. 39-47.

[Smit80] Alan Jay Smith, "Multiprogramming and Memory Contention", Software-Practice and Experience, Vol. 10, 1980, pp. 531-552.

[Smit82] Alan Jay Smith, "Cache Memories", ACM Computing Surveys, 14,3, September 1982, pp. 473-530.

[Smit83] James E. Smith and James R. Goodman, "A Study of Instruction Cache Organizations and Replacement Policies", Proc. 10'th Ann. Int. Symp. on Computer Architecture, June, 1983, pp. 132-137.

[Smit85] Alan Jay Smith, "Cache Evaluation and the Impact of Workload Choice", Proc. 12'th Ann. Int. Symp. on Computer Architecture, June, 1985, pp. 64-73.

[S178] S1 Proyect Staff, Advancet Digital Computing Technology Base Development for Navy Application, Lawrence Livermore Laboratories, Tech. Report UCID 18038, 1978.

[Wilk65] Wilkes M. "Slave memories and dynamic storage allocation". IEEE Trans. on Comp. TC-14, 2 (April 1965), pp. 270-271.

SUMMARY

Nowadays, a great number of computers use cache memories either to adapt the processor and memory speeds, or to reduce the traffic in microprocessor systems. Presently, discussion arises about the convenience or using a simple cache, or to provide one for instructions and other for data. In this paper the behavior of instructions and data on a unified cache is analyzed; a global LRU replacement algorithm is used, and comparisons with a system with static space equi-partition are provided.