# Extracción de Parámetros de Caches en Sistemas Reales

R. Paz Vicente, A. Linares, D. Cascado, M. A. Rodríguez Jódar, F. Díaz del Río
Grupo de Robótica y Arquitectura de Computadores Aplicada a la Rehabilitación.
Facultad de Informática. Avda. Reina Mercedes s/n 41012 SEVILLA.
E-mail: alinares@atc.us.es

**Resumen**

En esta ponencia-demo se presenta una práctica donde se analizan los accesos a la jerarquía de memoria de un procesador real (Intel Pentium P6), a bajo nivel. Con ello se pretende que el alumno extraiga los parámetros de la jerarquía de memoria (tamaño, tamaño de línea o bloque, y número de vías de los caches de datos), a la vez que se enfrenta a las dificultades que entraña cualquier sistema real. Para ello se usan los contadores internos de monitorización del rendimiento (PMC o "Performance Monitoring Counters") [1][4] de esta familia de procesadores. Este método se compara con otros existentes, resaltando la ventaja que nos ofrece éste para un curso avanzado de arquitectura de computadores. Se describen una serie de rutinas para acceder a los contadores y se da como ejemplo uno de los muchos programas con los que se pueden calcular tales parámetros.

## 1. Motivación y contexto

La presente demo se encuadra dentro de la asignatura troncal Arquitectura de Sistemas Paralelos 1 y 2 dividida en 2 cuatrimestres (3 créditos teóricos más 1.5 prácticos cada una) de 4º Ingeniería Informática (Universidad de Sevilla).

En estas asignaturas se pretende dar una visión conceptual, pero también cuantitativa, de las distintas posibilidades de paralelismo en la arquitectura de computadores. El primer cuatrimestre asienta las bases del paralelismo, centrándose en las arquitecturas segmentadas o encadenadas, mientras que en el segundo se exploran los paralelismos más avanzados (a nivel de instrucciones, de datos y de procesos o hilos).

Fruto del enfoque práctico y cuantitativo, que no se quiere perder en estas asignaturas, se han buscado herramientas para realizar prácticas con sistemas reales. Dada la baja dotación de prácticas y la dificultad de las herramientas usadas, se ha creído conveniente utilizar una herramienta que pueda ser usada en prácticas sucesivas.

Una herramienta que cumple estos requisitos son los contadores internos PMC [1][4] de la familia de procesadores más extendida en nuestro entorno: los Intel Pentium P6. Gracias a estos contadores se puede *monitorizar exhaustivamente* la ejecución de una tarea, puesto que en ellos se registran todos los detalles de más bajo nivel, como el tiempo transcurrido en ciclos, el número de accesos a memoria de datos e instrucciones, las instrucciones ejecutadas en una u otra tubería, los fallos de cache, el número de líneas modificadas, los aciertos o fallos de predicción de la BTB, etc. Tales medidas son totalmente realistas ya que no perturban en nada la ejecución (siempre están activas de forma transparente).

Para medir el comportamiento de rutinas sobre sistemas reales, se podrían haber usado otros métodos, como la ejecución en algún modo "trap", o disponer de una herramienta ICE. Pero el problema del modo "trap" es que produce una ejecución "controlada", que no será idéntica a la real, ya que, por ejemplo, la rutina de interrupción generada tras cada instrucción puede interferir en el tiempo de ejecución o en el conjunto de datos e instrucciones de la jerarquía. El principal inconveniente de las herramientas ICE son su precio (teniendo en cuenta un curso tan numeroso habrían de montarse bastantes puestos de trabajo). Por último existen métodos indirectos, como el propuesto en los ejercicios 5.2, 5.3 de [2]. Aquí se realizan repetidos accesos a un vector (de tamaño variable), calculando finalmente la media del tiempo de acceso para todos. El principal problema de este método son las mayores "interferencias" que sufren los accesos, pues para hacer más comprensible el código, éste se ha de escribir en un lenguaje de alto nivel. Y la experiencia nos ha demostrado que el ejercicio es más difícil de entender por el alumnado, por la

relación indirecta entre los tiempos de acceso medios obtenidos y los parámetros buscados en los caches. Por el contrario, el método que se describe a continuación, obtiene tales parámetros de una forma más directa, dejando el código necesario para acceder a los contadores y al vector encapsulado en funciones, con las que se puede trabajar de forma transparente.

Por otra parte, el acceso a bajo nivel de un sistema real tan complejo como éste, implica que cualquier "perturbación" en el sistema pueda falsear los resultados de la práctica. Por tal motivo se eligió, ya que un S.O. multitarea introduce un "ruido" considerable. De hecho, aunque ya hay aplicaciones visuales [3] que muestran el valor de los contadores, éstas corren sobre Windows, de forma que los accesos a los primeros niveles de cache se ven grandemente perturbados. En concreto hemos usado MS-DOS.

## 2. Contenidos y descripción de la sesión

El contenido de una primera práctica que usara tales contadores se ha enlazado con el primer tema de la asignatura. Tal tema estudia el análisis de prestaciones, y expone como ejemplo de este análisis el funcionamiento e impacto de la jerarquía de memoria en los computadores. La elección de este ejemplo se ha establecido por varios motivos, entre ellos la falta de homogeneidad en el conocimiento de la jerarquía de memorias que hemos detectado en los alumnos de cuarto (por la diversidad de orígenes de nuestro alumnado). Por ello en la primera práctica se analizan cuantitativamente los caches de datos (DCU) de los Pentium III. Estos tienen un primer nivel de cache L1 separado (en datos e instrucciones) y un segundo nivel unificado L2, al que sigue la memoria DRAM principal.

En la sesión de prácticas se deben lanzar unos programas de prueba que acceden de una forma estructurada a la memoria, de forma que el alumno pueda *imaginar qué está ocurriendo* en el sistema de caches. Tales programas llaman a una librería de rutinas desarrolladas por los autores, que devuelven el número de cuentas o eventos registrados por los contadores de los Pentium. En función de varias pruebas se pueden obtener diversos parámetros de la jerarquía de memoria.

Los PMC en la familia Intel x86 se incluyeron a partir de los procesadores Pentium. En la versión P6 se modificaron los mismos e Intel ha garantizado su mantenimiento para futuros Pentium, si bien añade nuevos eventos orientados, por ejemplo, a las nuevas instrucciones MMX que va incluyendo. Para poder acceder a los contadores la tarea debe ser de privilegio 0. Los P6 tienen sólo dos contadores de eventos, más otro de tiempo (los "ticks" o ciclos de reloj). Hay muchísimos eventos, como se muestra en el APPENDIX A de [1], de forma que disponemos de una información completísima de la *ejecución real* del procesador.

Para leer los dos contadores de eventos y el contador de ciclos, Intel ha introducido las instrucciones *wrmsr*, *rdpmc*, *rdtsc* [1], con las cuales se puede respectivamente: seleccionar los dos eventos para cada contador, examinar la cuenta de ellos o leer la cuenta de ciclos.

Para trabajar con más comodidad con los tres contadores se ha escrito una pequeña librería de rutinas para su programación y acceso (*P6stats.asm*, junto a *P6stats.h*), para lenguaje C. Éstas pueden inicializar la cuenta de eventos (o ciclos) o detenerla, y se describen a continuación:

- void SelecEvento (BYTE NumCont, BYTE Evento, BYTE UMask, BYTE TipoCont); /* *Selecciona en el contador 0 ó 1 con NumCon., El evento especificado en Evento. En TipoCont el valor 0 (cuenta eventos) o 1 (cuenta ciclos de reloj). UMask es un parámetro que en algunos casos especifica aún más concretamente el tipo de evento a medir, recogidos en [1]* */
- void ComenzarMonitor (void); /* *Comienza la cuenta de eventos en ambos contadores (lee y guarda el valor de los contadores)* */
- void TerminarMonitor (void); /* *Termina la cuenta de eventos en ambos contadores, devolviendo el incremento en las variables globales MPContador0 y MPContador1)* */
- void TSCComenzar (void); /* *Comienza la cuenta de ciclos del reloj (lee y guarda el valor del contador de ciclos)* */
- DWORD TSCTerminar (void); /* *Termina la cuenta de ciclos del reloj y devuelve el tiempo transcurrido con respecto a la llamada a* TSCComenzar *(lee el valor del contador de ciclos y lo resta del valor antes guardado)* */

Con estas funciones se puede realizar una medida de tiempo (en ciclos de reloj) más dos medidas de eventos para cualquier código de prueba. En nuestro caso, medidas de los accesos a vectores con tamaño y patrones de recorrido

diferentes para calcular los parámetros de los caches. Evidentemente si se desea medir más de dos tipos de eventos, deberá ejecutarse varias veces el mismo programa, escogiendo distintos eventos cada vez. La estructura de un programa que mida eventos se muestra en la Fig. 1, seguido en el programa principal de la práctica (*asp1p1.c*).

```
Pedir los eventos que se desean medir y otros parámetros para el código de prueba.
Llamada a SelecEvento ()
disable () /* inhabilita interrupciones*/
Repetir la siguiente medida varias veces, pues la primera puede dar resultados diferentes:
/*Calculo de la sobrecarga de las func “Overhead”*/
TSCComenzar(); ComenzarMonitor();
/*Código de prueba que se considere sobrecarga*/
TerminarMonitor();
TiempoOverHead=TSCTerminar();
OverHead0=MPContador0;
OverHead1=MPContador1;
TSCComenzar(); ComenzarMonitor();
/*AQUI DEBE IR EL CODIGO DE PRUEBA*/
TerminarMonitor(); Tiempo=TSCTerminar();
/*Resta de sobrecarga u overhead*/
MPContador0-=OverHead0;
MPContador1-=OverHead1;
Tiempo-=TiempoOverHead;
/*Impresión de resultados */
printf ("Duración: %ld\nNúmero de eventos del contador 0: %lu\nNúmero de eventos del contador 1: %lu\n", Tiempo, MPContador0, MPContador1);
```

Fig. 1: Estructura de un prog. para medir eventos

Por otra parte para realizar esta práctica hay que superar un problema de MS-DOS: la limitación de memoria (trabaja en el “modo real” de la familia 80x86). Para ello, se han escrito una serie de rutinas diseñadas para conmutar en cierta medida a “modo protegido” de la familia 80x86, y así poder acceder a vectores de gran tamaño (mayores de 1 segmento, 64Kbytes) desde MS-DOS. Se recogen en los ficheros *code32.asm, xms.asm, xmem.c* y *misc.asm.* Desarrollados en ensamblador permiten reservar y utilizar, sin limitación alguna, toda la memoria disponible a partir del primer megabyte, creando vectores de cualquier tamaño, que permitirán evaluar las grandes caches de segundo nivel. El cometido de cada uno de los ficheros se explica a continuación:

- En *code32.asm*, la función *initcpu32()* cambia el modo de la CPU a protegido para modificar la tabla de descriptores, así los registros (ES, DS, etc.) de segmento tendrán un límite de 4 Gb (en lugar de los 64Kb). Tras esto cambia otra vez al modo real. Para que esta operación pueda efectuarse debe estar cargado el controlador de memoria extendida HIMEM.SYS y no debe estar cargado el gestor EMM386.EXE de memoria extendida (implica el modo protegido).
- *xms.asm* define las rutinas básicas para crear un gestor de memoria extendida bajo modo real.
- *xmem.c* contiene las versiones para memoria extendida de las habituales funciones de reserva de memoria dinámica: *X_malloc(), X_free(),* etc.

Por último para esta práctica se da otro conjunto de funciones para acceder a vectores de memoria extendida cuyo espacio se reservó con *X_malloc().* El programa *misc.asm* contiene las siguientes funciones básicas para leer o escribir en el vector reservado en la memoria extendida:

- LeerVector (DWORD *dirección*, DWORD *zancada*, DWORD *tamaño*);
- EscribirVector (DWORD *dirección*, DWORD *zancada*, DWORD *tamaño*);

El valor de la dirección donde empieza el vector es fijo y fue cargado por el gestor de memoria extendida (en *asp1p1.c* está en *m0*). En *tamaño* colocaremos la dimensión del vector con el que queremos trabajar, medida en palabras (2 bytes, por ser modo 16 bits). El valor de *zancada* nos indicará de cuantas en cuantas palabras vamos a recorrer el vector, dejando las intermedias sin acceder (ver Figura 2). Este parámetro se puede usar para hallar el tamaño de línea de una cache.

Todos los ficheros anteriores se suministran al alumno, más un proyecto (*.prj de BorlandC, TurboC* o similar), para poder compilar y linkar correctamente, así éste no ha de entrar en detalles sobre la implementación descrita, y sólo debe preocuparse por las llamadas a *LeerVector()* o *EscribirVector()* y de elegir los eventos relacionados con los caches de datos y variar los valores de *zancada* y *tamaño* para obtener los parámetros y extraer conclusiones.

Como ejemplo de una práctica se propone que el alumno calcule algunos parámetros de los Pentium III (adjuntando el razonamiento aplicado) como: Tamaño de línea (o bloque) de L1; tamaño del cache de datos L1, grado de asociatividad del cache de datos L1 (número de vías); tamaño de línea (o bloque) de L2; tamaño del cache L2; grado de asociatividad del cache mixto L2.

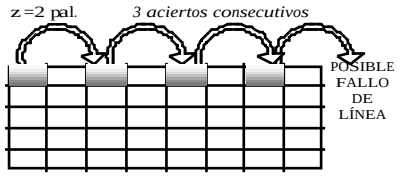


Figura 2: Acceso a un cache con zancadas

## 3. Obtención de parámetros

Para calcular el tamaño de los caches se ejecuta el programa original *asp1p1*, el cual contiene una llamada a *EscribirVector*(), con un tamaño y zancada que se piden por teclado. El tamaño del vector escrito se va incrementando; cuando se producen fallos de cache masivamente, el vector no "cabe" en el cache y se ha superado el tamaño de éste (eventos 25H, 26H, 27H para L2, y 45H, 46H y 47H para L1). Los accesos de escritura son más fiables que de lectura, ya que el número de otras variables, aparte de nuestro vector que se escribe, suele ser muy inferior al que se leen, provocando menos "interferencias" en los accesos.

Para calcular el tamaño de las líneas se puede jugar con la zancada, pero también dividiendo el tamaño total del vector (con zancada 1 y de tamaño menor que el del cache) por el número de líneas escritas (los eventos anteriores nos dan tal información). La otra forma es incrementar el tamaño de la zancada con un vector de tamaño fijo, hasta que el número de líneas accedidas se reduzca (lo normal es trabajar con potencias de dos, y entonces esta reducción será a la mitad).

Para el cálculo del número de vías se da una pista en el propio código de *asp1p1.c*. Se han introducido muchas llamadas a *EscribirVector(), e*n un comentario, con tamaño pequeño, pero cuya posición inicial está distanciada en el tamaño del cache. Por tanto todos los accesos se mapean en la misma línea de la cache, por tanto, si esta tiene N vías, no se producirán fallos apreciables en N llamadas. Sin embargo, para N+1 llamadas, todos los accesos de la última llamada han de ser fallos.

## 4. Nuestra experiencia con esta práctica

Tras realizar otras prácticas usando diversos métodos, la experiencia ha sido muy positiva. Una de las mayores motivaciones que encuentran los alumnos procede de usar un sistema real, del que se extraen parámetros tan "ocultos" como los del cache L1, y todo ello con una precisión muy alta.

Como ejemplo de la motivación, citamos en primer lugar una cuestión que les sobreviene cuando descubren que se producen algunos fallos (entre 4 y 6 generalmente) de cache si el tamaño del vector que se está escribiendo coincide exactamente con el tamaño del cache. Tras creer inicialmente que, debido a estos fallos, el tamaño del cache no es potencia de dos, caen en la cuenta de que las propias variables del código están ocupando algunas líneas, las cuales producen este efecto. Una satisfacción similar sucede cuando descubren finalmente como las sucesivas llamadas a *EscribirVector(),* van "ocupando" las vías de las caches hasta que éstas se agotan y se producen tantos fallos como líneas ocupa el vector. Por último, una ventaja adicional es la disponibilidad de las herramientas de esta demo (sólo las rutinas y el apéndice [1]), lo que les permite hacer la prueba en su casa con su propio ordenador.

## 5. Conclusiones

Se ha presentado una práctica para el estudio de caches, pero analizando un sistema real en lugar de usar simuladores. La precisión de los resultados es altísima, gracias a la librería realizada para acceder a los contadores PMC de los Pentium. Además estas librerías son de fácil disponibilidad para el alumnado. En las sesiones de prácticas realizadas en la Universidad de Sevilla se ha descubierto una alta motivación por usar un sistema real y tan extendido como los PC.

## Referencias

[1] Intel Architecture Software Developer's Manual. Volume 3: System Programming Guide. Appendix A Performance-Monitoring Events. Intel Corporation. 1997

[2] J.L. Hennessy, D.A. Patterson "Computer Architecture. A Quantitative Approach". Morgan-Kaufmann (Second Edition), 1996.

[3] Herramienta contador Intel. Página Web http: //developer.intel.com/software/idap/resources /technical_collateral/pentiumii/p6perfnt.htm

[4] Pentium Pro Processor Architecture. Tom Shanley. Addison-Wesley Developers Press.