PRP: una aplicación en Linux para evaluar el rendimiento del procesador Pentium

Francesc Solsona[1], Francesc Giné[1], Concepció Roig[1] y Sergi Lana

[1]Dep. Informática. Universidad de Lleida
Pl. Víctor Siurana, 1 25003 (Lleida)
francesc@eup.udl.es, sisco@eup.udl.es, roig@eup.udl.es y h8086166@est.fib.upc.es

Resumen

Con el fin de observar el comportamiento del Pentium, hemos implementado un driver para LINUX que opera en modo carácter y una aplicación gráfica, PRP (Parámetros de Rendimiento del Pentium), la cual facilita su programación. Con este entorno de programación es posible capturar y almacenar las ocurrencias de los diferentes eventos producidos en el Pentium durante un determinado intervalo de tiempo. Observando los resultados es posible evaluar tanto el funcionamiento de aplicaciones de usuario como el comportamiento del procesador en general. Su uso en prácticas de laboratorio puede ayudar al alumno a entender mejor las distintas unidades funcionales de que consta el sistema informático y muy especialmete del procesador.

1 Introducción

Los temas de estudio habituales en las asignaturas afines al área de Arquitectura de Computadores, hacen referencia básicamente a las unidades funcionales del computador, así como a las distintas alternativas de diseño a diferentes niveles (sistema informático, procesador, etc...) que llevan á obtener mejores rendimientos.

En la bibliografía que trata temas relacionados con Arquitectura de Computadores, se muestran las propuestas de distintas familias de procesadores. Normalmente son libros o manuales muy densos y de lectura muy poco agradecida. Si para estudiar un procesador tenemos primero que aprendernos todos sus componentes y a continuación intentar relacionarlos para entender su funcionamiento, puede que la tarea no sea ni tan siquiera del agrado del alumno más motivado. Además, para saber si un procesador es mejor o peor que otro, o hasta incluso si una cierta aplicación optimiza los recursos del sistema, no se dispone de medios ni recursos necesarios para comprobarlo.

Una forma de evaluar el diseño y la bondad de un procesador, puede consistir en estudiar la ocurrencia de los eventos que tienen una influencia decisiva en el rendimiento global del sistema, como por ejemplo: fallos en la memoria cache, fallos en los accesos a la TLB, lecturas en mem. principal, operaciones en punto flotante, etc...

Para poder hacer un estudio de este tipo, nos hemos basado en un procesador concreto: el Pentium; y se ha desarrollado un driver en LINUX [4], al cual hemos denominado *MSR driver*. Este gestor de dispositivo permite obtener valores referentes a la ocurrencia de los eventos que se pueden controlar en el Pentium. También se ha implementado la aplicación gráfica PRP, que facilita la programación, visualización y almacenamiento de resultados de los eventos del sistema objeto de estudio.

El objetivo que se persigue, es el de estudiar la estructura del Pentium y la influencia de ésta en el rendimiento cuando se ejecutan diversos tipos de aplicaciones, a la vez que se intenta que en base al análisis de los resultados, los estudiantes se vean con ánimo de proponer mejoras de diseño, que a su entender podrían redundar en un mejor rendimiento.

Por otro lado, a la vista de como se comporta el sistema, dicho análisis, puede llevar a la obtención de ciertas recomendaciones en la programación, para optimizar la utilización de los recursos del procesador.

2 Desarrollo del MSR driver

La familia Pentium incorpora unos registros denominados MSR (Model Specific Registers), capaces de capturar información del comportamiento del sistema. El sistema informa de los sucesos que se van produciendo mediante la generación de eventos, modificando el contenido y valor de los registros MSR. Estos a su vez, son accesibles por software mediante instrucciones en lenguaje ensamblador.

El Pentium puede detectar hasta un máximo de 41 eventos diferentes. Algunos de ellos son tan conocidos como fallos en la cache de código o datos, y algunos puede que no tanto como por ejemplo, fallos en la TLB (Translation Look Aside Buffer), aciertos en la BTB (Branch Target Buffer), o paradas del pipeline debido a un conflicto estructural o de datos. La tabla completa de los diferentes eventos que puede monitorizar el Pentium puede encontrarse en [3].

Estudiando el comportamiento de las aplicaciones s/w, mediante la obtención de uno o varios parámetros de rendimiento (p.e. número de ocurrencias de un determinado evento) durante su ejecución, podremos determinar las posibles modificaciones que deberemos realizar para optimizar dichos parámetros y por consiguiente, optimizar los recusos utilizados.

En [7], sus autores comentan la existencia de unos registros que permiten obtener unos parámetros de rendiminento en el procesador Pentium. Terje Mathisen [5], esboza de forma clara y concisa los secretos del Pentium, que además, con la ayuda de [6] y de los manuales del Pentium, [3] [1] [2], ha hecho posible que los secretos del Pentium no sean tan secretos.

Para facilitar el acceso a los registros MSR del Pentium, se ha desarrollado un gestor de dispositivo de tipo carácter y se ha incorporado al s.o. Linux.

2.1 Registros Específicos del Modelo (MSR)

La familia de procesadores Pentium, incorpora 20 *Registros Específicos del Modelo* (MSR), que sirven para obtener características relacionadas con su implementación particular. Algunas de sus posibles aplicaciones son las siguientes:

- Acceso directo a las caches internas, TLB's, BTB's y control de paridad.
- Activación y desactivación selectiva del seguimiento de un programa (execution tracing), de la ejecución concurrente de instrucciones en los dos pipes (instruction pairing) y de la predicción de saltos (branch prediction).
- Control del rendimiento.
- Control de excepciones.

Valor ECX	Registro del Model MSR
10H	*Time Stamp Counter*
11H	*Control and Event Select*
12H	*Counter 0*
13H	*Counter 1*

Tabla 1: MSR's relacionados con el *control del rendimiento*

Los 4 registros MSR relacionados con el *control del rendimiento*, objeto de estudio en este trabajo, son los siguientes (Tabla 1):

- Contador de tiempo (**TSC**: Time Stamp Counter). Cuenta los ciclos de reloj producidos desde el último RESET.
- Control y selección de evento (**CESR**: Control and Event Select Register). En este registro se codifica los eventos que han de contar tanto el Contador 0 como el 1.
- Contador 0 (**CTR0**: Counter 0). Contador programable de eventos 0.
- Contador 1 (**CTR1**: Counter 1). Contador programable de eventos 1.

Para poder acceder a estos registros, la familia de procesadores Pentium han incorporado las instucciones **RDMSR** (Read Model Specific register) y **WRMSR** (Write Model Specific register), las cuales permiten leer y escribir respectivamente en los registros MSR. Cuando se ejecuta una de estas instrucciones, el contenido del registro ECX especifica cual de los registros MSR está siendo accedido. Existe también otra instrucción para leer el registro TSC, **RDTSC**.

El **CESR** se divide en 9 campos (Figura 1). Contiene un campo de 6 bits para cada contador, **ES0** y **ES1** (*event select*), un bit *PC* (Pin

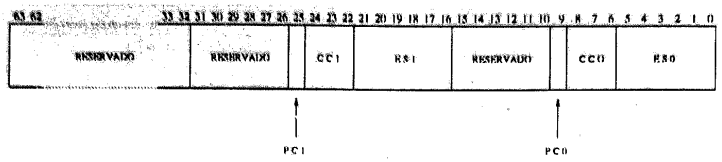

Figura 1: El registre *Control and Event Select Register* - CESR

Control), y también un campo de control de tres bits, uno para cada contador.

Los campos del registro CESR son:

- Selección de evento - ES0,ES1

  Este campo contiene el valor del evento que CTR0 (ES0) o CTR1 (ES1) tiene que contar (00 ... 3Fh). Así pues, podemos controlar dos eventos al mismo tiempo.

- Control del Contador - CC0,CC1

  Para controlar la operación de los contadores se utiliza un campo para cada uno de ellos. Estos campos (CC0 y CC1), son de tres bits y según el valor que contengan, se actuará de la forma especificada en la tabla 2.

  Como podemos ver en la Tabla 2, los dos registros CTR0 y CTR1 pueden programarse para contar:

  1. *Ocurrencias* (O): número de eventos de un determinado tipo. Cuantas veces se ha producido dicho evento.
  2. *Duración* (D): ciclos de reloj que un determinado evento ha permanecido activo. Cada vez que se produce el evento, se cuentan los ciclos de reloj que ha durado. Por ejemplo, una lectura en memoria tarda un tiempo de ciclo a memoria, que es un múltiplo del tiempo de ciclo del procesador [3].

  La tabla completa, así como la descripción de la mayoría de eventos, puede encontrarse en [3] y [6]. En la tabla 3 listamos solamente algunos de ellos.

- Pin Control - PC0,PC1

  Los pins *PM0* y *PM1* (*PM0/BP0*, *PM1/BP*), y los dos bits *PC0* y *PC1*, controlan la operación de los contadores *CTR0* y *CTR1* respectivamente. Con la ayuda de los bits *PC0* y *PC1* del *CESR*, los pins PM0 y PM1 se pueden programar para indicar si el contador asociado se ha incrementado o bién se ha producido un desbordamiento (*overflow*).

  Un mismo evento puede producirse dos veces en un mismo clock interno del procesador. Aunque el contador (CTR0/1) incrementará en dos, el pin asociado (PM0/1) en cambio, lo notificará solo una vez.

N	Evento	tipo
0	*Data Read*	O
1	*Data Write*	O
2	*Data TLB miss*	O
3	*Data Read Cache miss*	O
4	*Data Write Cache miss*	O
13	*Code TLB miss*	O
19	*BTB hits*	O
23	*Instructions Executed in the v pipe e.g. parallelism/pairing*	O
24	Clocks mentre el cicle de bus està corrent(utilització del bus)	D
31	*Pipeline stalled because of an address generation interlock*	D
34	*FLOP's*	O

Tabla 3: Eventos.

2.2 MSR driver

Un gestor de dispositivo (device driver) es un software encargado de operar con un determinado tipo de hardware.

En Linux (al igual que cualquier sistema de la familia Unix), los dispositivos físicos tienen asociados unos ficheros especiales permitiendo a los programadores acceder a los dispositivos y a los mismos ficheros especiales con un único repertorio de llamadas a sistema.

Hay dos tipos básicos de dispositivos:

- Dispositivos de bloque. La unidad de trans-

CC0-CC1	Significado
000	No cuenta nada, el contador está deshabilitado
001	Cuenta el evento seleccionado mientras CPL=0,1 o 2
010	Cuenta el evento seleccionado mientras CPL=3
011	Cuenta el evento seleccionado sea cual sea el valor de CPL
100	No cuenta nada, el contador está deshabilitado
101	Cuenta tics mientras CPL=0,1 o 2
110	Cuenta tics mientras CPL=3
111	Cuenta tics sea cual sea el valor de CPL

Tabla 2: Control del Contador en el registro CESR. (CPL Current Privilege Level)

ferencia es el bloque. Un bloque es igual a una potencia de dos octetos(normalmente 512 bytes). Por ejemplo discos, cintas, etc..

- Dispositivos de carácter. La unidad de trasferencia es un carácter o una línea. Por ejemplo, el terminal, impresoras, etc..

El hardware que gestiona el *MSR driver* tiene bastante similitud con un dispositivo de tipo carácter, ya que el número máximo de bits transferidos en una determinada transacción entre él mismo y cualquier registro MSR no supera el equivalente a 4 palabras, 64 bits (= tamaño de un registro MSR).

El Pentium puede operar en tres modos distintos, real, protegido y virtual. El modo real se conserva por compatibilidad con los procesadores anteriores. El modo protegido, apareció con el procesador 80286 para proteger las diferentes tareas en un sistema operativo multiproceso. Proporciona 4 niveles de privilegio (NP = 0, 1, 2 y 3), siendo el 0 el nivel más prioritario y el 3 el nivel de menor prioridad.

Para acceder a los registros MSR, es necesario hacerlo en modo supervisor (con un nivel de privilegio igual a 0); debido a ello, se ha realizado un gestor de dispositivo (MSR driver). Los drivers, en el s.o. Linux están integrados en el núcleo y se ejecutan con permisos de superusuario, convirtiéndose de esta forma en el medio ideal para acceder a los registros de los dispositivos periféricos y en nuestro caso, a los registros MSR.

Por otra parte, las instrucciones que sirven para acceder a los registros MSR: RDMSR, WRMSR y RDTSC, no están definidas en el lenguaje ensamblador de gnu (as), incorporado en cualquier distribución de Linux. Primero por tanto, ha sido necesario definir dichas instrucciones para poder ser utilizadas posteriormente en el MSR driver.

3 Aplicación PRP

Con el objeto de poder interaccionar fácilmente con el gestor del dispositivo realizado (MSR driver), se ha implementado una aplicación gráfica, denominada *PRP*, que permite escribir y leer datos de los registros específicos, seleccionar el evento a monitorizar y el tipo de control del contador correspondiente, elegir y ejecutar un conjunto de tests (benchmarks o aplicaciones de usuario), así como representar gráficamente los resultados obtenidos. También permite saber en cada instante el valor del registro contador de ciclos de reloj (TSC). Esta aplicación ha sido escrita utilizando el lenguaje C y el lenguaje Tcl-Tk [8].

En esta sección, se explicará el funcionamiento de la aplicación *PRP* a nivel de usuario y posteriormente se mostrarán diferentes ejemplos de su funcionamiento.

3.1 Descripción de la aplicación PRP

La interficie gráfica de la aplicación *PRP* está formada por una barra de menús, accesible tanto por teclado como con el *mouse*, y la ventana principal que permite visualizar los valores de los contadores.

3.1.1 Ventana Principal

En la figura 2 se muestra la ventana principal de la aplicación realizada.

Como se observa, la pantalla está dividida en cuatro áreas diferentes. Aparte de la barra de

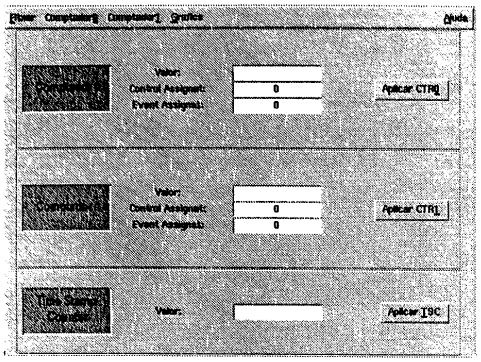

Figura 2: Ventana Principal de la aplicación.

menus, explicada en el siguiente punto, existen dos zonas que hacen referencia a los dos registros contadores y una última que da información sobre el registro contador de pulsos de tiempo (TSC). Para cada contador, aparte de su valor, se muestra el número de evento asignado y el tipo de control aplicado al evento correspondiente. El *Botón* CTR_i permite actualizar el valor del contador$_i$ en su correspondiente caja de textos denominada *Valor*.

3.1.2 Barra de Menús

La barra de menús de la aplicación está formada por los siguientes cinco menús:

1. **Fitxer**: En la figura 3 se muestran las seis diferentes opciones que presenta este menú. La función de cada una de estas entradas son las siguientes:

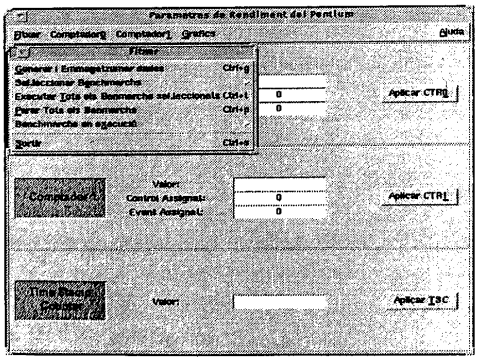

Figura 3: Cuadro de diálogos correspondiente al menú *Fitxer*

(a) *Generar i Emmagatzemar dades*: El cuadro de dialogo asociado, mostrado en la figura 4, permite al usuario escoger si desea monitorizar uno o los dos contadores. Los datos leidos de los contadores se grabarán en los ficheros nombrados por el usuario en el cuadro de texto correspondiente. La aplicación *PRP* generará, junto con el fichero de datos, dos ficheros más con extensión *ppm* y *ps*, asociados al gráfico de los datos leidos. Observar como la duración del test es otra opción a escoger por el usuario de la herramienta.

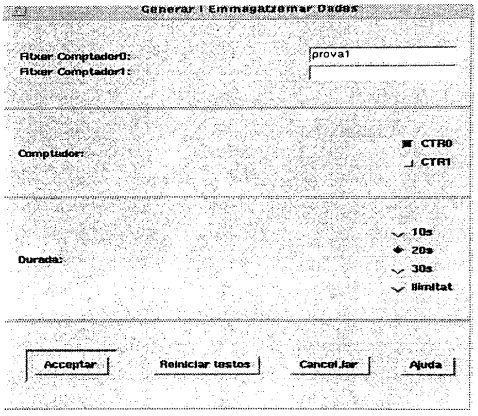

Figura 4: Cuadro de diálogos correspondiente al menu *Generar i Emmagatzemar dades*

(b) *Sel.leccionar Benchmarks*: Esta entrada permite escoger al usuario entre uno y diez tests, previamente seleccionados. Esta opción, es muy útil para el docente que dirige las sesiones de práctica, dado que permite controlar que todos los estudiantes ejecuten las mismas aplicaciones.

(c) *Executar tots els Benchmarks sel.leccionats*: Esta opción permite ejecutar los benchmarks seleccionados con la opción anterior.

(d) *Parar Tots els Benchmarks*: Esta entrada permite parar los benchmarks que se estaban ejecutando.

(e) *Benchmarks en execució*: Este comando permite visualizar cuales son los

tests que se están ejecutando.

(f) *Sortir*: Opción para salir de la herramienta *PRP*.

2. **Comptador0**:

   Este menú permite configurar el contador0, es decir, permite seleccionar el tipo de evento a leer así como el tipo de control asignado a dicho evento.

3. **Comptador1**: Lo mismo que el anterior pero para el contador 1.

4. **Gràfics**: Permite seleccionar y visualizar un gráfico en formato *.ppm*

5. **Ajuda**: Este menú muestra en diferentes cuadros de diálogo las ayudas necesarias para ejecutar correctamente la aplicación.

3.2 Ejemplos

En esta sección se mostrarán varios ejemplos ilustrativos del funcionamiento de la aplicación realizada.

Ejemplo 1

- **Tipo de evento.** Code TLB Miss.
- **Control Asociado al evento.** Número 3. (Cuenta el número de veces que se produce el evento seleccionado sea cual sea el valor del CPL).
- **Duración del test.** 20 sg.
- **Contexto 1.** El test se realizó en el entorno de ventanas del *Red Hat Linux 4.2*. Las aplicaciones ejecutadas eran dos ventanas de terminal y la propia aplicación *PRP*.
- **Contexto 2.** Igual que en el contexto 1 a excepción de que durante la ejecución del test se arrancó la aplicación *Netscape*.

En la figura 5 se observa como se incrementa el número de fallos de la TLB en el momento en que se ejecuta la aplicación del Netscape, lo que resulta totalmente lógico si se tiene en cuenta que la TLB alberga la traducción de dirección lógica a física de las páginas de memoria principal más recientemente utilizadas.

Ejemplo 2

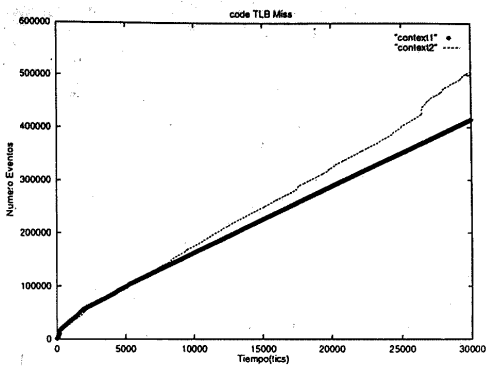

Figura 5: Ejemplo 1

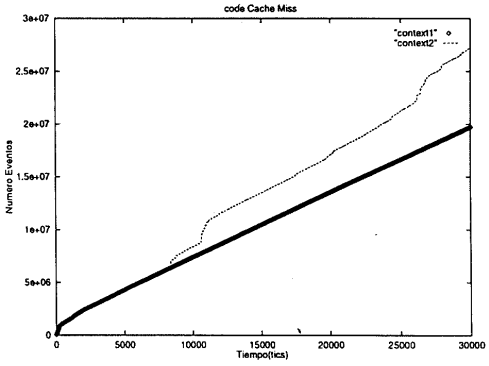

Figura 6: Ejemplo 2

- **Tipo de evento.** Code Cache Miss.
- **Control Asociado al evento.** Número 3. (Cuenta el número de veces que se produce el evento seleccionado sea cual sea el valor del CPL).
- **Duración del test.** 20 sg.
- **Contexto 1.** El test se realizó en el entorno de ventanas del *Red Hat Linux 4.2*. Las aplicaciones ejecutadas eran dos ventanas de terminal y la propia aplicación PRP.
- **Contexto 2.** Igual que en el contexto 1 a excepción de que durante la ejecución del test se arrancó la aplicación *Netscape*.

Obsevar en la figura 6, como ya sucedía en el primer ejemplo, la influencia de ejecutar Netscape durante el test. Otro aspecto a resaltar es la gran similitud entre las gráficas

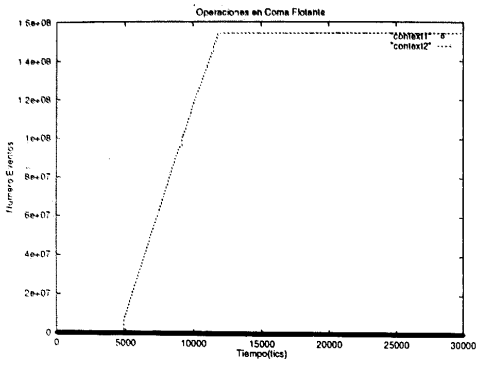

Figura 7: Ejemplo 3.

de ambos ejemplos, cosa normal si se tiene en cuenta la estrecha relación entre la TLB y la cache correspondiente.

Ejemplo 3

- **Tipo de evento.** Operaciones en Punto Flotante.
- **Control Asociado al evento.** Número 3. (Cuenta el número de veces que se produce el evento seleccionado sea cual sea el valor del CPL).
- **Duración del test.** 20 sg.
- **Contexto 1.** El test se realizó en el entorno de ventanas del *Red Hat Linux 4.2*. Las aplicaciones ejecutadas eran dos ventanas de terminal y la propia aplicación *PRP*.
- **Contexto 2.** Durante la realización del test se ejecutó el conocido benchmark *linpack*, el cual permite medir la velocidad de ejecución en instrucciones de punto flotante del procesador.

En la gráfica de la figura 7, correspondiente al contexto 2, se ve como se incrementa espectacularmente el flujo de instrucciones ejecutadas por la unidad de punto flotante en el intervalo de tiempo durante el que se ejecuta el benchmark.

4 Conclusiones

En este documento hemos presentado una herramienta que permite programar los registros específicos del procesador Pentium (MSR) y obtener la información que estos facilitan. El análisis de dicha información permitirá:

- Controlar la ocurrencia de algunos eventos, así como su frecuencia, para deducir su incidencia en el rendimiento del procesador.
- Hacer un estudio de la estructura del procesador y establecer su relación con la información que facilitan los registros MSR.
- Propiciar un debate crítico respecto a las soluciones de diseño que aporta la arquitectura del Pentium.
- Proponer alternativas de mejora en el desarrollo de los programas, para conseguir aplicaciones que usen los recursos h/w del procesador Pentium con mayor eficacia.

Con todo ello se proporciona un conjunto de facilidades que pueden ayudar, a las personas involucradas en el estudio de temas relacionados con la Arquitectura de Computadores, a analizar aspectos del diseño de un procesador no facilmente accesibles; y por otro lado a constatar que la eficiencia de un sistema informático se ve influenciada por un conjunto de parámetros muy diverso.

Referencias

[1] Intel Corporation. *Intel Architecture Software Developer's Manual, Volume 1: Basic Architecture*, 1997.

[2] Intel Corporation. *Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference Manual*, 1997.

[3] Intel Corporation. *Pentium Processos Family Developer's Manual*, 1997.

[4] Sergi Lana. *Parametres de Rendiment en el Pentium en un Sistema Operativo Linux.* 1997.

[5] Terje Mathisen. Pentium secrets. *Byte*, 1994.

[6] Hans-Peter Messmer. *The Indispensable Pentium Book.* Addison-Wesley, 1995.

[7] J.B. Chen Y. Endo K. Chan D. Mazières A. Dias M. Seltzer and M.D. Smith. The measured performance of personal computer operating systems. *Operating Systems Review*, 29(5):229–313, 1995.

[8] Brent B. Welch. *Practical Programming in Tcl and Tk*. Prentice-Hall, 1995.