Análisis a Bajo Nivel de Procesadores Superescalares Reales

A. Linares, R. Paz Vicente, S. Vicente, 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 analiza la influencia en el tiempo de ejecución de algunas optimizaciones sobre el código máquina para procesadores reales (familia Intel Pentium). Se propone el estudio del efecto del emparejamiento de instrucciones y de las dependencias entre registros, la aceleración obtenida con el desenrollado de iteraciones, la comparación entre instrucciones simples (del núcleo RISC) y complejas, etc. Además todo ello se ejecuta desde código ensamblador, midiendo el tiempo y otros eventos directamente sobre los Pentium, gracias a los contadores internos de monitorización del rendimiento ("Performance Monitoring Counters", PMC) [1] de esta familia de procesadores. El alumno se enfrenta a las dificultades que supone trabajar con el sistema real y a bajo nivel. Se compara con otros métodos existentes y se describen una serie de rutinas para acceder a los contadores dando varios ejemplos de las posibilidades que brindan para el análisis de la estructura superescalar de estos procesadores.

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 de 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, las instrucciones ejecutadas en una u otra tubería, los fallos de cache, 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 aplicaciones sobre sistemas reales, se podrían haber usado otros métodos, que suelen producir una ejecución "controlada", que no será idéntica a la real, por la rutina de interrupción generada tras cada instrucción que puede interferir en el tiempo de ejecución o en los datos e instrucciones de los caches o de la BTB. El principal inconveniente de las herramientas ICE son su precio (teniendo en cuenta que en un curso tan numeroso como el nuestro habrían de montarse bastantes puestos de trabajo). Por el contrario, el método que se describe, obtiene tales parámetros de una forma más directa. Por último, aunque trabajar con simuladores es un método muy didáctico (se utiliza en las primeras sesiones de prácticas de esta asignatura), al adolecer del componente de realidad, suele motivar menos al alumnado.

Esta sesión pretende trabajar en ensamblador directamente, para aislar de la influencia del compilador, (la interacción del compilador con el procesador superescalar se incluye en otra

práctica) apreciando con toda claridad la importancia del orden de las instrucciones, de las dependencias reales, del desenrollado, etc.

El acceso a bajo nivel de un sistema real tan complejo como los PC con Pentium, implica que cualquier "perturbación" en el sistema puede falsear los resultados de la práctica. Por tal motivo se decidió ejecutar la práctica sobre un sistema monotarea, ya que el "ruido" que introduce un S.O. multitarea es bastante considerable. De hecho, aunque ya hay aplicaciones visuales [3] que muestran el valor de los contadores, estas corren sobre Windows, de forma que no es fácil, por ejemplo, determinar la diferencia de tiempo de ejecución entre dos fragmentos de código ordenados de forma distinta. En concreto hemos usado MS-DOS por su disponibilidad por parte de una amplia mayoría del alumnado.

2. Contenidos y descripción de la sesión

En el boletín de prácticas se da al alumnado un resumen de la estructura de tuberías de los Pentium y de las reglas de emparejamiento de instrucciones. Al comienzo de la sesión se recuerdan rápidamente los aspectos del juego de instrucciones que más pueden influir en los resultados de la práctica.

Recordemos brevemente que la cadena de los Pentium contiene 5 fases [5]:

- PF: Prebúsqueda o "PreFetch".
- ID1: Etapa primera de decodificación.
- ID2: Etapa segunda de decodificación.
- EX: Ejecución.
- WB: Escritura en memoria o registros.

El Pentium no dispone de planificación (*scheduling*) ni de especulación dinámicas. Con sus dos tuberías (U y V) puede alcanzar, en principio, un CPI=1/2 (para instrucciones enteras). Pero existen unas restricciones sobre la emisión en paralelo de las mismas. Se llama *emparejamiento* a la emisión de dos instrucciones en paralelo. La tubería U puede ejecutar cualquier instrucción (tubería genérica), mientras que la tubería V sólo puede ejecutar saltos o instrucciones enteras "simples" (núcleo RISC). Cuando dos instrucciones se emparejan, la emitida en la tubería V es siempre la siguiente instrucción a la emitida en la tubería U, según el orden código.

Las condiciones que deben de cumplir dos instrucciones para que se puedan emitir a la vez ("emparejarse" en las tuberías U y V), se llaman *reglas de emparejamiento*. En los Pentium estas reglas son muchas pero las más básicas, con las que trabajaremos en esta demo, se resumen en que las dos instrucciones deben de ser del núcleo RISC (sin desplazamientos complejos ni prefijos), no pueden tener dependencias y que los saltos sólo pueden emparejarse si van en la tubería V (se permite el emparejamiento de las instrucciones de comparación con un salto condicional).

Por el contrario a los Pentium, en la familia Pentium P6 (cuyo primer miembro fue el Pentium Pro [4], y después los Pentium II y III), se introdujo planificación y especulación dinámicas, de forma que la endoarquitectura se modificó radicalmente. Las fases de los P6 son 10 (o más si está a la espera de datos):

- IFU1, IFU2 e IFU3: Búsqueda, separación y alineamiento en frontera de 16 bytes de las instrucciones.
- DEC1 y DEC2: Decodificación y traducción de instrucciones complejas en micro-operaciones (micro-ops).
- RAT: Renombrado dinámico de registros.
- DIS y ROB: Espera por datos en estaciones de reserva y encola en buffer de reordenación.
- EX: Ejecución.
- RET1 y RET2: Retira la instrucción del buffer de reordenación.

Las instrucciones simples se traducen por una micro-op, mientras que las complejas pueden convertirse hasta en 4 micro-ops o incluso en una secuencia de microcódigo de la ROM interna.

Los P6 Pentium tiene 5 unidades funcionales (aritmética genérica, aritmética simple más saltos, una de carga y dos de almacenamiento), pero en cada ciclo sólo pueden renombrar los registros de hasta 3 instrucciones, de forma que, en principio, se puede alcanzar un ritmo máximo sostenido de CPI=1/3 (para instrucciones enteras). Por el contrario, las restricciones de ejecución son mucho más leves (vendrían de comparar el tipo de instrucciones que esperan en el buffer de reordenación con las unidades funcionales), permitiendo en algunos ciclos lanzar hasta 5 micro-ops.

Por otro lado en esta sesión se recuerda cómo trabajar con los PMC (ya se ha trabajado con ellos en prácticas). Éstos se incluyeron a partir de los procesadores Pentium, y en los P6 se modificaron. A partir de aquí, Intel garantiza su mantenimiento para futuros Pentium (si bien añade nuevos

eventos orientados, por ejemplo, a las nuevas instrucciones MMX). Para poder acceder a los contadores la tarea debe ser de privilegio 0. Todos los Pentium tienen sólo dos contadores internos de eventos, más otro de tiempo (los "ticks" o ciclos de reloj). Hay muchísimos eventos [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*, 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 han escrito librerías de rutinas para su programación y acceso (*P5stats.asm, P5stats.h* para Pentium y *P6stats.asm, P6stats.h* para P6), 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 las funciones anteriores 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 desean 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, y los programas de esta demo son un ejemplo. En el "código de prueba" se han de insertar las instrucciones ensamblador que se quieren monitorizar.

Todos los ficheros anteriores se suministran al alumno, aparte de un proyecto (*.prj de BorlandC, TurboC* o similar), para poder compilar y linkar correctamente, de manera que éste no ha de entrar en detalles sobre la implementación antes explicada, y sólo debe preocuparse por las medidas sobre el código de prueba. Con todo ello, el alumno deberá elegir los eventos relacionados con las tuberías e ir extrayendo conclusiones sobre el tiempo de ejecución de cada fragmento.

Con la librería de medida de eventos y ciclos y con un programa con la estructura de la Fig. 1, se pueden plantear en primer lugar sesiones de prácticas, donde se compruebe el impacto de las optimizaciones de bajo nivel en el tiempo de ejecución. Por ejemplo, para los Pentium la reordenación manual de fragmentos de código que contienen dependencias o que violan las reglas de emparejamiento, conduce a un aumento del IPC (Instrucciones Por Ciclo [2]). Por el contrario, estos mismos fragmentos de código pueden ser lanzados sobre un Pentium P6 sin que la reordenación afecte (en general) al IPC, debido a que la planificación dinámica de éste absorbe las dependencias. Otro tema de estudio son las instrucciones que no pertenecen al núcleo RISC. En el Pentium éstas suponen un detrimento apreciable de prestaciones, mientras que para los P6 muchas de ellas se suelen convertir en micro-ops, entrando a ejecutarse junto a las otras instrucciones simples. Usando eventos que devuelven el número de instrucciones y de micro-ops el alumno puede reconocer cómo ha sido la conversión anterior. También es interesante que comprenda que algunas instrucciones del conjunto más CISC, producen un deterioro inmenso en las prestaciones. Por ejemplo, la instrucción LOOP, que ha de ejecutarse como secuencia de microcódigo. Otros factores que pueden estudiarse son la alta penalización por fallo de predicción de la BTB, diferencia entre el número de instrucciones emitidas y el de retiradas, etc.

Por último, se puede mostrar la importancia de la técnica del desenrollado, calculando la aceleración que introduce. Además, los alumnos

pueden comprobar que el tiempo de ejecución de un bucle desenrollado depende más del orden de las instrucciones (reglas de emparejamiento) en un Pentium que en los P6. Como ejemplo de una práctica se propone que el alumno calcule los parámetros IPC, aceleración, porcentaje de instrucciones de control o sobrecarga, etc. etc. adjuntando los factores que han influido en los mismos para los diversos fragmentos de código dados. Los eventos C4h, C5h, C0h, C2h, D0h, A2h y otros nos permiten calcular lo anterior.

```
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 progr. para medir eventos

3. Nuestra experiencia con esta práctica

Una de las mayores motivaciones que encuentran los alumnos procede de usar un sistema real, del que se extraen conclusiones sobre el paralelismo ILP y la aceleración que produce en la máquina. Les hace ver que no es “oculto”, sino que pueden interferir directamente con toda esta temática (una cuestión que no es tan “realista” en los simuladores o en el código de alto nivel). Además las medidas son de una precisión muy alta, lo que es una ventaja comparable a la que ofrecen los simuladores.

Entre las cuestiones que hemos descubierto que motivan particularmente al alumnado citamos: el comprobar numéricamente la influencia de la alta penalidad introducida por un fallo de BTB o de una línea de cache (la primera ejecución es mucho más lenta que las siguientes), la alta aceleración de técnicas de planificación estática avanzada, como el desenrollado (a pesar de la dificultad de conseguirla con el reducido conjunto de registros de los Pentium), comparada con la conseguida en los P6. La complementación de la exposición teórica de estas cuestiones con los resultados reales tan precisos que se obtienen, produce una motivación adicional en el alumnado.

Por último, una ventaja adicional es la disponibilidad de las herramientas de esta demo (sólo rutinas y apéndice A de [1]), que les permite hacer pruebas con su propio ordenador.

4. Conclusiones

Se ha presentado una práctica para el estudio del ILP en procesadores superescalares con o sin planificación dinámica, pero basadas en el análisis de un sistema real, en lugar de 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. Se ha descubierto una alta motivación del alumnado para comprender un tema tan complejo como la superescalaridad y la planificación dinámica, al poder comprobar en la realidad, el efecto de la misma en al aceleración de pequeños bucles o fragmentos de código.

Referencias

[1] Intel Architecture Software Developer’s Manual. Volume 3: System Programming Guide. Appendix A. 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.

[5] Pentium Processor Optimization Tools. Michael L. Schmit. AP Professional. 1995