Simulación de procesadores Superescalares con Predicción Dinámica de Saltos

Elizalde Federico
Matías Ezequiel Prado

Arquitectura de Computadoras y Técnicas Digitales
Facultad de Cs. Exactas
Universidad Nacional Del Centro de la Provincia de Buenos Aires

Ing. Martín Menchón
Ing. Marcelo Tosini

2017

Resumen

Los ejercicios prácticos de la materia *Arquitectura de Computadoras y Técnicas Digitales*, suelen ser extensos y difíciles de seguir por los alumnos, causando incertidumbre en estos a la hora de ponerse a practicar por su propia cuenta y sin ayuda de los docentes.

El presente trabajo pretende proveer de una herramienta que sirva de apoyo a los alumnos a la hora de resolver los ejercicios prácticos, y le facilite al estudiante la comprensión de las técnicas de predicción de saltos en procesadores superescalares.

Palabras-claves: superescalar, predicción de saltos, buffer-target-branch

Introducción

Uno de los mayores inconvenientes que los estudiantes de los cursos de arquitectura de computadoras actuales debe enfrentar, es que los ejercicios prácticos de estos cursos son extensos y difíciles de seguir. A través de la práctica, estos cursos buscan que los estudiantes adquieran las bases del funcionamiento del hardware que subyace en los dispositivos modernos.

A la hora de resolver la práctica los estudiantes están continuamente propensos a errores, debido a la gran cantidad de conceptos que tienen que tener en cuenta en el momento de su resolución. Los estudiantes deben lidiar con conceptos bases (tales como la segmentación, banco de registros, tipos de riesgos, entre otros), hasta llegar a comprender conceptos avanzados de técnicas digitales cómo algoritmos de planificación, tratamiento de excepciones, técnicas de predicción de saltos, etc.

Por estos motivos, se cree conveniente y necesario proveer de nuevos métodos que ayuden a los estudiantes a resolver y verificar la correctitud de los ejercicios prácticos, de manera tal que les permita una mayor comprensión de los conceptos relevantes.

El presente trabajo pretende proveer de una herramienta que le facilite al estudiante la comprensión de las técnicas de predicción de saltos en procesadores superescalares. En particular, busca abordar las estrategias de predicción dinámica basadas en la información sobre la historia de cada salto. El trabajo hace uso, y continúa el desarrollo, de “ReOrder Buffer” una aplicación web desarrollada por Tomás Juaréz y Guillermo Pacheco, la cual permite simular la emisión y ejecución desordenada de un conjunto de instrucciones.

Marco teorico

Predicción de Saltos en Procesadores Superescalares

En los procesadores superescalares, los cuales son capaces de leer y ejecutar más de una instrucción por ciclo, las instrucciones de salto suponen un inconveniente para la obtención de un rendimiento alto. Debido a que estas instrucciones provocan dependencias de control: Cuando una instrucción de salto es decodificada, el procesador bloquea el fetch de instrucciones hasta que el salto

no es resuelto, es decir, hasta que no se conoce la dirección destino y no se sabe si el salto es tomado o no. Como el procesador ha estado varios ciclos sin leer ninguna instrucción, nos encontramos con que varios ciclos en los cuales el *dispatcher* está vacío, las estaciones de reserva vacías y muchas unidades funcionales vacías, lo cual afecta en gran medida a la performance del procesador.

Para evitar la penalización introducida por los cambios en el flujo de ejecución de un programa, una solución adoptada por muchos procesadores consiste en tener métodos precisos que predigan la dirección de los saltos condicionales, así como anticipar lo antes posible el cálculo de la dirección destino.

La predicción de saltos pretende reducir la penalización producida por los saltos, haciendo prefetching y ejecutando instrucciones del camino destino antes que el salto sea resuelto. Esta circunstancia es conocida como ejecución especulativa, ya que se ejecutan instrucciones sin saber si son las correctas en el orden del programa. Para ello se intenta predecir si un salto será efectivo (si será tomado) o no, así como realizar el cálculo de la dirección destino antes que la instrucción de salto llegue a la unidad de saltos del procesador. De esta manera se intenta no romper el flujo de ejecución del programa, leyendo continuamente instrucciones de la *I-cache, y haciendo un uso efectivo de los recursos del procesador.*

Las técnicas de predicción de salto se pueden dividir, básicamente, en dos tipos:

- Las que realizan predicción estática.
- Las que realizan predicción dinámica.

La diferencia entre las técnicas radica en el momento en el que se realiza la predicción. La predicción estática es realizada en tiempo de compilación, mientras que la predicción dinámica es realizada en tiempo de ejecución. El presente trabajo se centra en esta última.

Técnicas de Predicción dinámica

Estas técnicas predicen el resultado de un salto en base a información recolectada y conocida sólo en tiempo de ejecución. Por lo general, dos estructuras son necesarias para realizar una predicción dinámica:

- **Branch History Table (BHT):** Consiste en una tabla, la cual guarda información sobre las últimas ejecuciones de los saltos. La información almacenada se refiere a si el salto ha sido efectivo o no. A partir de esta información, se predice si el siguiente salto será tomado o no.
- **Branch Target Address Cache (BTAC):** Consiste en una tabla que almacena la dirección destino de los últimos saltos ejecutados. De esta manera, cuando un salto es predicho como tomado, se mira si está en la tabla, y si es así, se obtiene la dirección destino de ella, así se puede calcular rápidamente la dirección destino, incluso en saltos indirectos.

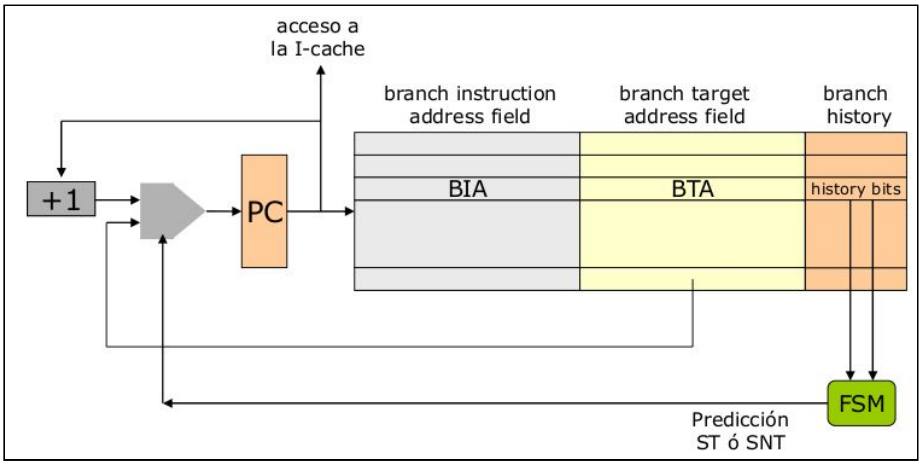
La base de la mayoría de los predictores dinámicos es el llamado ***Branch Target Buffer***. Esta estructura combina las dos mencionadas anteriormente (BTAC y BHT). Cada entrada contiene los bits necesarios para realizar la predicción de efectividad, así como la posible dirección destino.

A partir de esta estructura básica se han ido diseñando los distintos predictores, desde el más sencillo que para cada salto guarda unos bits de información sobre su historia más reciente, hasta los más complicados que usan dos niveles de predicción.

Branch Target Buffer

El *Branch Target Buffer* (BTB) es una pequeña memoria asociativa que guarda las direcciones de los últimos saltos ejecutados así como su destino. A su vez guarda información que permite predecir si el salto será tomado o no.

En la etapa de fetch se mira si la dirección de la instrucción está en el BTB. Si es así se miran los bits de predicción y se decide si el salto ha de ser tomado o no. Si el salto no es tomado o la dirección no está en el BTB en el siguiente ciclo se hace el fetch de la siguiente instrucción en orden. Si el salto es tomado, en el siguiente ciclo se hace el fetch del nuevo camino de ejecución.



Con un solo bit de historia, cada vez que estemos analizando bucles anidados se predecirá incorrectamente dos veces. Cuando el salto no sea efectivo (fin del segundo bucle), va a suponer que saltó por lo que predice incorrectamente, y luego, la siguiente vez que lo evaluemos (por el primer bucle) predecirá incorrectamente que no salta.

Sin embargo, es fácil mejorar la predicción para estos casos basta con agregar otro bit de historia. De esta manera, para pasar de una predicción de salto a una de no salto, al menos se han de producir dos errores consecutivos. Estas situaciones se analizarán y verán con mayor detalle, más adelante en el informe con los casos de estudio propuestos.

Implementación

Como ya se mencionó anteriormente, el trabajo hace uso y continúa el desarrollo de una aplicación web previamente implementada. La aplicación estaba desarrollada con JavaScript, y hacía uso de varias bibliotecas y *frameworks* como JQuery, Bootstrap, require.js, sigma.js y ace.js, para diferentes propósitos. Por lo que, se decidió por continuar con el uso de estas tecnologías para la implementación de la predicción de saltos, y de esta manera mantener consistencia y coherencia con lo ya realizado.

Para poder llevar a cabo la predicción de saltos, y antes de implementar la BTB, fue necesario extender el conjunto de instrucciones que aceptaba la aplicación ya implementada, para aceptar las instrucciones de saltos condicionales.

Una vez realizado lo anterior, se construyó la BTB y se modificaron las diferentes etapas del procesador con el objetivo de hacer uso de la BTB, y para mantenerla actualizada con la historia de los saltos. Además, se debieron incorporar mecanismos para validar e invalidar aquellas instrucciones cargadas de manera especulativa.

A medida que se llevó a cabo la implementación de la predicción de saltos, se debieron corregir bugs que contenía la aplicación web previamente desarrollada, y además se modificó la interfaz gráfica de la misma para mejorar su usabilidad.

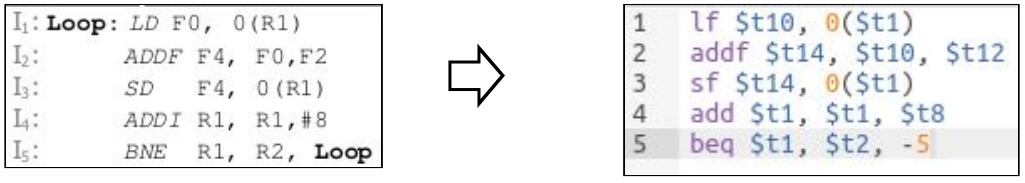
Casos de Estudio

Con el objetivo de demostrar los beneficios de la aplicación, y sus posibles usos se presentan dos casos de estudio. El primero de ellos está basado en el código de un ejercicio que se encuentra en uno de los prácticos de la materia, y con el mismo se busca demostrar cómo los alumnos pueden utilizar la herramienta para resolverlo. El segundo, es un ejercicio propuesto por nosotros para comprender la mejora otorgada por el predictor basado en 2-bits de historia en los bucles anidados.

Primer caso de Estudio: *Bucle sencillo*

El caso de estudio está basado en el código y la configuración (tipos de unidades funcionales, y latencias) del ejercicio 8 del práctico nº 3.

El código del ejercicio fue adaptado al conjunto de instrucciones aceptado por la aplicación:



Además, el ejercicio suponía de un sistema con las siguientes características:

- Una unidad de acceso a memoria para flotantes (2 ciclos)
- Una unidad aritmética de flotantes (3 ciclos)
- Una unidad aritmética de enteros (2 ciclos)
- Una unidad de tratamiento de saltos (1 ciclos)

Una vez compilado el código y configurada la aplicación con las condiciones del ejercicio, los alumnos pueden determinar de manera sencilla la cantidad de ciclos que tarda en ejecutarse el código, ya sea utilizando predicción de saltos y/o sin predecir.

Además, con un tamaño de dispatch y de estaciones de reserva igual a 1, el alumno puede observar de manera sencilla cómo con la predicción de saltos a partir de la segunda iteración del bucle el procesador, en vez de esperar la resolución del salto (S4) y bloquear el fetching de instrucciones, hace fetching de la instrucción (S0) que se encuentra en la dirección del salto (BTA).

Por otro lado, si aumentamos el tamaño del dispatch y de las estaciones de reserva podemos observar los beneficios en términos de aceleración del uso de técnicas de predicción en los procesadores superescalares.

Por ejemplo, con un tamaño de dispatch y de estaciones de reserva igual a 2, e iterando 5 veces (o sea, que la condición del salto se evalúa 4 veces de manera correcta) la ejecución sin predicción tardó 36 ciclos, y con predicción (para 1 bit y 2 bits) tardó 28 ciclos. Por lo tanto, la aceleración obtenida por el uso de ambos predictores es del 28% para este caso en particular.

En estos casos en donde no hay anidamientos de saltos, con ambos predictores (1 bit y 2 bits) se logran iguales tiempos ya que ambos fallan la última vez (cuando el salto se evalúa como falso).

Segundo caso de Estudio: *Bucles anidados*

En este caso se quiso demostrar la ventaja de utilizar un predictor de dos bits sobre uno de un bit. Esto se logra apreciar en el caso que hay loops anidados, debido a que con el predictor de 1 bit en loops anidados cuando el salto no es efectivo se predice incorrectamente 2 veces.

Para demostrar esto, se creó el siguiente código:

```
1 loop_1 lf $t20, 0($t1)
2    addf $t21, $t20, $t12
3 loop_2 lf $t10, 0($t1)
4    addf $t14, $t10, $t12
5    sf $t14, 0($t1)
6    add $t1, $t1, $t8
7    beq $t1, $t2, -5  <-loop_2
8    beq $t2, $t2, -8  <-loop_1
```

Se configuró un sistema con las mismas características que el primer caso de estudio. También se configuró un dispatcher de tamaño 2 e igual tamaño que para las estaciones de reserva.

Por otro lado, por fines prácticos se configuró que cada salto se evalúe como verdadero 2 veces, es decir, 3 iteraciones cada uno. Ya que al estar uno contenido dentro del otro, el que está contenido (*loop_2*) iterara 3 veces por cada iteración del de afuera (*loop_1*).

La ejecución sin predicción de saltos tarda 76 ciclos. Con el predictor de un bit tardó 66 ciclos obteniendo una aceleración de 1.15. En cambio, al utilizar el predictor de 2 bits el total de ciclos bajó a 62 ciclos mejorando aún más la primera aceleración obtenida. La diferencia de ciclos entre el predictor de 1 bit y el de 2 bits se da por predecir incorrectamente 2 veces con el predictor de 1 bit en vez de una sola vez como lo hace el predictor de 2 bits. Estos 4 ciclos de diferencia involucran invalidar las instrucciones cargadas especulativamente y corregir con el camino correcto.

A medida que aumentamos la cantidad de iteraciones para el *loop_1 (*es decir, aumentamos la veces que se evalúa como verdadero el salto *beq $t2, $t2, -8*) la tasa de fallos del predictor de 1 bit va a ser cada vez mayor, ya que se incrementa el número de veces que se llama al *loop_2* y en cada salida y entrada de este bucle se predecirá de manera incorrecta.

Conclusiones

Este trabajo incluye el desarrollo de una aplicación que es capaz de beneficiar tanto a los alumnos como a los docentes. Por un lado, los alumnos cuentan con una aplicación fácil de usar para resolver los ejercicios de predicción de salto y por otro, los docentes pueden hacer uso de la misma para explicar estas técnicas. Cabe destacar que también es útil a la hora de corregir ejercicios y/o crear nuevos.

Se espera que la herramienta aporte beneficios reales a los alumnos de la materia, y que los ayude a comprender y asentar los temas que se dictan durante la cursada de una manera práctica y fácil para estos.