



Departament d'Enginyeria  Informàtica i Matemàtiques  UNIVERSITAT ROVIRA I VIRGILI	Arquitectura de Computadores
	AC
	Curso 22/23
	Primera Convocatoria
	Práctica 2: Comportamiento Predictores de Salto <i>Simplescalar</i>

Simulación procesador Superescalar:

Predictores de Salto

La práctica consiste en el análisis de la estructura y el comportamiento de diferentes predictores de salto en un procesador Superescalar con ejecución *fuera de orden* (OoO). Para ello se utilizará el simulador Simplescalar y se evaluará el comportamiento del procesador ejecutando los programas de prueba SpecCPU2000.

La predicción de saltos permite en un procesador superescalar la ejecución especulativa de las instrucciones de una de las dos alternativas de un salto (instrucción de salto: branch). Así el procesador delante de un hazard (riesgo) de control puede aprovechar los ciclos que perdería hasta saber la resolución del salto, en probar una de las dos alternativas y ganar esos ciclos de espera ejecutando instrucciones con una cierta probabilidad de acertar.

Diremos que un predictor es bueno cuando acierta el camino correcto con una alta probabilidad. Con frecuencia los predictores buenos son complejos y utilizan estructuras de datos para almacenar el comportamiento de las instrucciones de salto. Un *Branch Address/Target Buffer* (BTB) almacena las direcciones efectivas de las instrucciones de salto a medida que las va ejecutando. Un *Return Address Stack* (RAS) almacena en forma de pila las direcciones de retorno de las instrucciones Call. Del mismo modo, un *Branch History Register* (BHR) almacena la resolución (Taken / Not_Taken) de los últimos saltos acumulados. Puede ser *Global* (GBHR) o asociado a cada instrucción de salto de manera individual *Per Address* BHR (PaBHR). Por último, un Pattern History Table (PHT) almacena la predicción que se hará en un posible salto. De este modo, mantiene un contador saturado de 2-bits y utiliza el bit de más peso para la predicción. Esta estructura puede ser global, individual a cada salto o incluso compartirse entre algunos saltos que tengan comportamientos parecidos.

Simplescalar, mediante los simuladores sim-bpred y sim-outorder permite configurar cada una de estas cuatro estructuras para analizar el comportamiento de diferentes configuraciones.

Para esta práctica, se hará uso de un subconjunto de 5 benchmarks disponibles a través de la imagen y también del espacio moodle de la asignatura.

Comentarios

- La práctica se realizará **en GRUPOS DE 2 PERSONAS**
- Se realizará una entrevista con todos los integrantes del grupo en la sesión de laboratorio que tienen asignada.
- El informe (obligatoriamente en PDF) junto con el código implementado y el vídeo resumen de la práctica se comprimirán en un único archivo ZIP y se guardará en el moodle antes de realizar la entrevista.

Especificación

La tarea de esta práctica se divide en dos fases: (1) Estudio de los predictores que se pueden simular con el Simplescalar y (2) Modificación del Simplescalar para simular un nuevo predictor de saltos.

1a Fase:

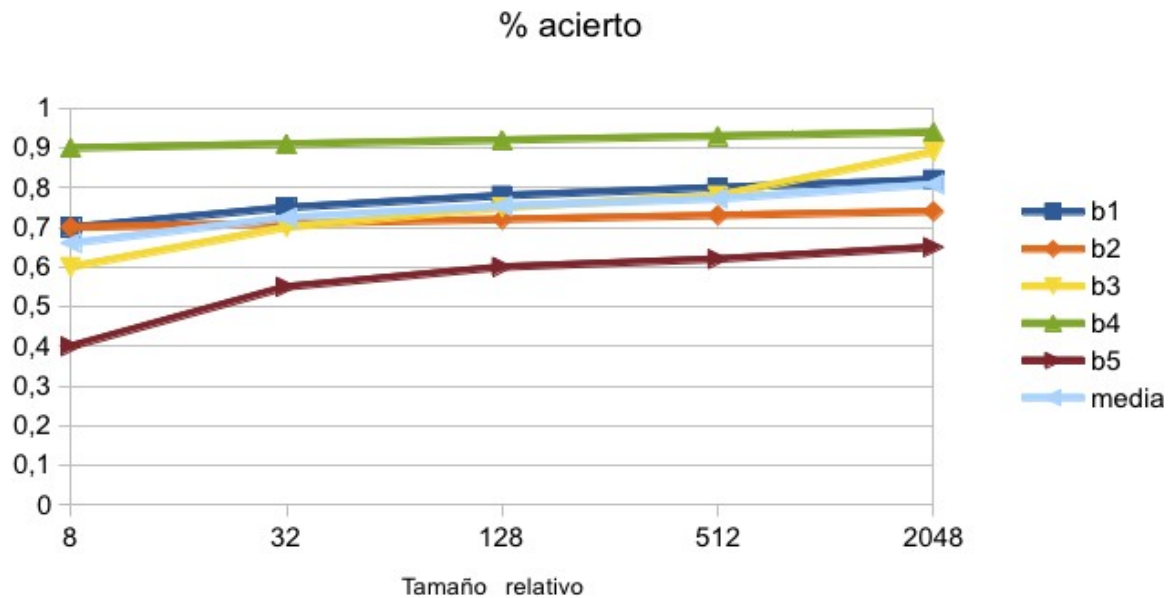
Los predictores que implementa Simplescalar son: *nottaken*, *taken*, *perfect*, *bimodal*, *2-level* y *comb*.

Se usarán para evaluar el comportamiento en cuanto a *IPC* y *porcentaje de aciertos* de las siguientes alternativas de predictores de saltos:

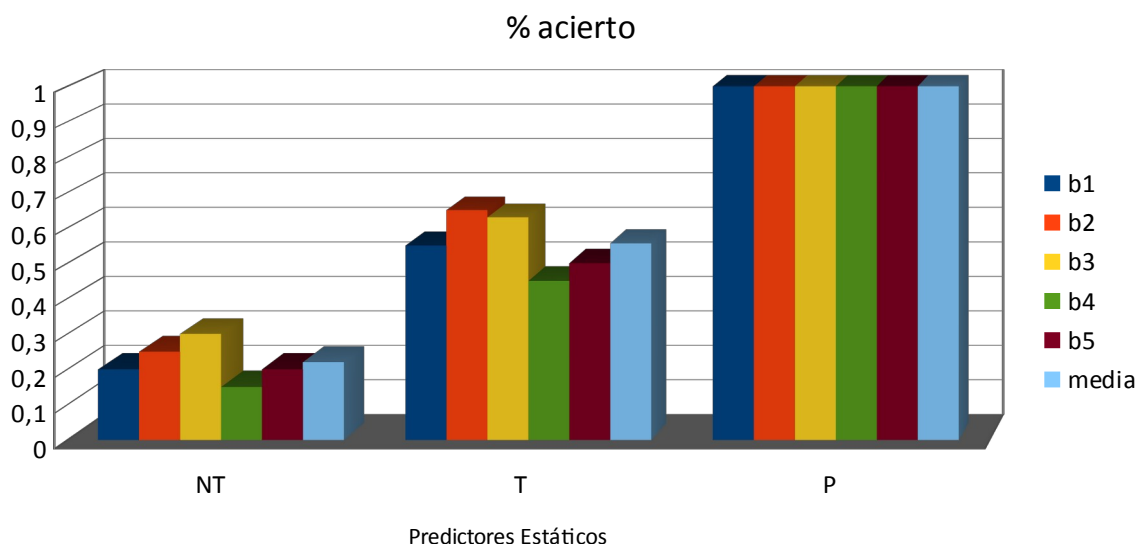
- **nottaken**: opción: `-bpred nottaken`
- **taken**: opción `-bpred taken`
- **perfect**: opción `-bpred perfect`
- **bimodal**: opción `-bpred bimod`
 - Tamaño del PHT <I2-size> : 8,32,128,512,2048
 - `-bpred:bimod X`
- **Gshare**: opción `-bpred 2lev`
 - Tamaño del BHR <I1-size>: 1 y del PHT<I2-size>: 8,32,128,512,2048
 - `-bpred:2lev 1 <X> <log2X> 1`
- **Gag (Gselect)**: opción `-bpred 2lev`
 - Tamaño del BHR <I1-size>: 1 y del PHT<I2-size>: 8,32,128,512,2048
 - `-bpred:2lev 1 <X> <log2X> 0`
- **Pag**: opción `-bpred 2lev`
 - Tamaño del BHR <I1-size> y del PHT<I2-size> respectivamente: (4-4), (8-16), (16-64), (32-256), (64-1024), (32-2048) son (Y-X)
 - `-bpred:2lev <Y> <X> <log2X> 0`

El objetivo es observar y generar gráficas que muestren el comportamiento del **IPC** y porcentaje de acierto (**.bpred_dir_rate**) para diferentes valores de configuración de los predictores con los cinco benchmarks que se disponen de la práctica anterior.

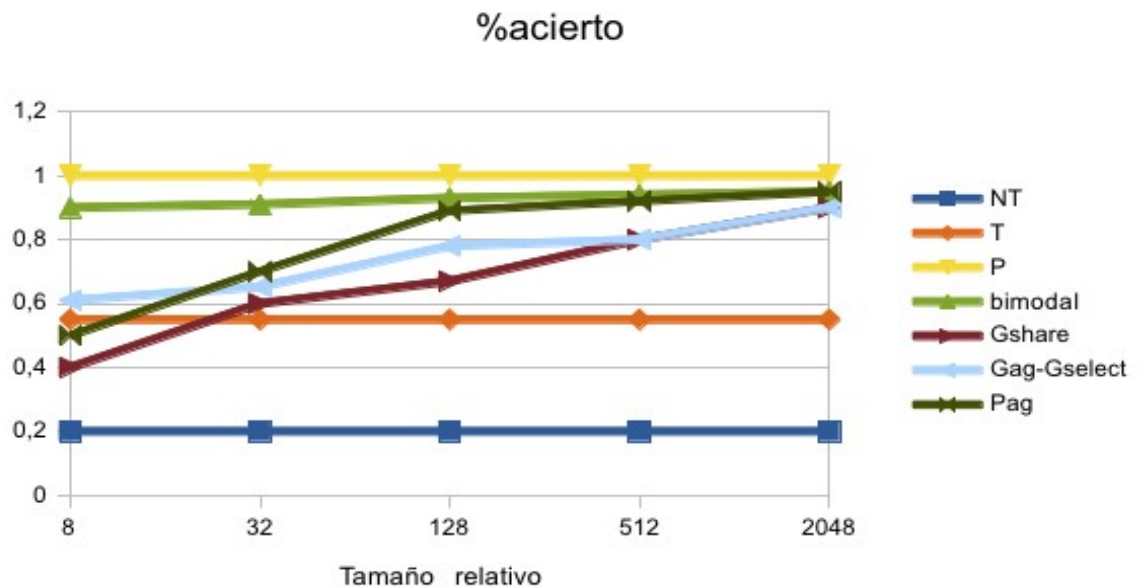
Para los distintos estudios que se deben realizar, y si no se indica lo contrario, tened en cuenta los siguientes comentarios:



- 1) Se mostrará una gráfica de cada parámetro para cada uno de los siete predictores indicados anteriormente, donde en el eje de las Y estará el parámetro y en el eje de las X las diferentes configuraciones en tamaño del predictor estudiado. En la misma gráfica aparecerán los cinco benchmarks y la media de los mismos.
- 2) Para los predictores estáticos se pueden resumir todos en una única gráfica de la forma siguiente:



- 3) Para cada estudio se presentarán gráficas con los resultados individuales de cada benchmark y con la media de todos ellos.
- 4) También se generarán gráficas resumen del comportamiento de cada uno de los predictores. En la misma gráfica aparecerán los siete predictores teniendo en el eje de las X las diferentes configuraciones.

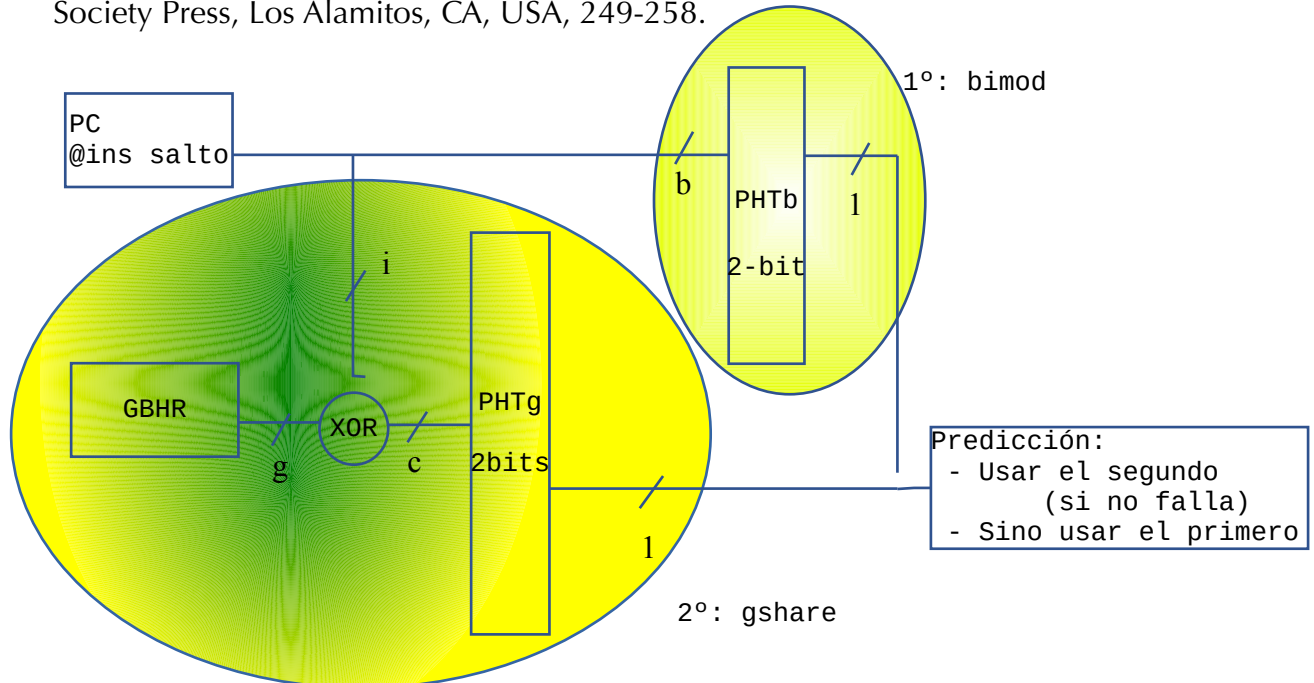


- 5) Para cada benchmark simulado, se saltarán 50 millones de instrucciones y se recolectarán las estadísticas para los siguientes 50 millones. Además, se utilizarán los datos de entrada REF.
- 6) El tamaño de BTB y RAS no se modificarán de su valor por defecto.
- 7) Para el resto de parámetros del simulador se utilizarán los valores por defecto, a excepción del bus de acceso a memoria (de 32 bytes) y de la latencia de la misma (300 ciclos iniciales y 2 por acceso consecutivo).

2 fase:

Implementación del del predictor de saltos Cascaded Branch Predictor (Leaky filter version):

Karel Driesen and Urs Hölzle. 1998. **The cascaded predictor: economical and adaptive branch target prediction.** In *Proceedings of the 31st annual ACM/IEEE international symposium on Microarchitecture (MICRO 31)*. IEEE Computer Society Press, Los Alamitos, CA, USA, 249-258.



El predictor en cascada funciona como un predictor híbrido donde se implementan dos predictores de diferente complejidad. EL primero es mas simple que el segundo y la idea es utilizar el primer predictor mientras vaya acertando en las predicciones. O sea que si el primer predictor acierta el segundo no se actualiza ni se utiliza, permitiendo que los datos del segundo se reserven para aquellos saltos donde el primer predictor por ser mas simple no acierta.

La implementación del predictor:

- Para obtener una predicción se busca en los dos predictores. Si el segundo lo tiene en sus estructuras, se usa la predicción del segundo, por ser mas complejo y en principio mas preciso. Si el segundo falla, se usa la predicción del primero, porque el segundo no ha guardado ninguna información de esa instrucción
- Para actualizar la información del predictor: se actualiza siempre la información del primer predictor sencillo y solo se actualiza la predicción del segundo predictor si el primer predictor ha fallado en la predicción.

Como esta descripción es genérica se implementará la siguiente concreción del predictor:

- El primer predictor será un bimodal del Semplescalar.
- El segundo predictor será un gshare que en simplescalar es un 2-level con operación XOR para acceder al PHT.

- Los cambios que se deben hacer en el gshare es que cada entrada del PHT deberá tener una marca para indicar que esa entrada no se ha usado nunca y así utilizar la predicción del bimodal. A sea almacenara 0,1,2, y 3 para su uso normal y -1 o 4 para indicar que no se ha usado nunca esa entrada.

Las estructuras son las habituales del predictor bimodal y del gshare:

1. Pattern History Table (PHTb) del predictor bimodal: es una tabla donde se almacenan dos bits para implementar un contador saturado del comportamiento de los saltos. De manera que cada vez que el salto es efectivo Taken, se incrementa y si el salto no es efectivo NotTaken se decrementa. Así se obtienen cuatro estados donde dos de ellos (los que tienen el bit de mayor peso a '1') predicen que el salto va a ser efectivo y los otros dos (los que tienen el bit de menor peso a '0') predicen que la instrucción de salto no va a saltar (contador de 2-bits saturados).
2. Global Branch History Register (GBHR) guarda la historia de los últimos 'g' saltos que se han ejecutado en el procesador. Guarda un 1 si ha sido Taken y un 0 si ha sido NotTaken.
3. Pattern History Table (PHTg) del gshare: es una tabla donde se almacenan dos bits para implementar un contador saturado del comportamiento de los saltos. Se accede de manera directa a partir del índice obtenido de la operación XOR del GBHR y de bits de la dirección de la instrucción de salto.
4. La predicción se obtiene a partir del bit de mayor peso del PHT consultado

Cada vez que se ejecuta un salto estas estructuras se acceden dos veces. La primera para obtener una predicción y actuar en consecuencia en el pipeline y la segunda para actualizar el comportamiento del salto ejecutado y afinar subsiguientes predicciones.

Los parámetros a definir en este predictor són:

Del bimodal:

1. Número de entradas del PHTb → define 'b'

Del gshare:

2. Número de bits del GBHR → define 'g' e 'i'
3. Número de entradas del PHTg → define 'c'

Los valores iniciales de estas estructuras son bits a '1' (o sea 11 → '3') a excepción de PHTg que será '4'. De este modo la predicción de cualquier salto inicial será Taken y podremos saber que entradas del PHTg no se estan usando.

Para modificar el simplescalar y añadir un nuevo predictor se puede uno fijar en la implementación de uno ya existente y duplicar añadir lo necesario. En este caso el predictor **comb** es suficientemente parecido y será el modelo a seguir. Implementación del predictor de saltos AGREE.

Las partes de simplescalar a modificar son:

- En bpred.h:
 - Añadir el nuevo predictor en la lista de enum bpred_class. Por ejemplo: **cascade**
 - En la definición de tipos de los predictores modificar la estructura struct bpred_dir_t para eliminar aquellas partes del meta-predictor que no se usará. Tened en cuenta que el **bimod** y el **gshare** utilizan las mismas estructuras en el **comb** que en el **cascade**
- En sim-outorder.c:
 - Registrar en sim_reg_options los parámetros de configuración del predictor: con `opt_reg_int_list(opt, "-bpred:cascade",)`. Será necesario definir predictor_config y predictor_nelt.
 - Añadir en sim_check_options la lectura de los parámetros del predictor. Llamar a la función de creación del predictor `bpred_create(,,,,,,)`. Fijarse en el **comb**.
- En bpred.c:
 - Añadir en bpred_create y bpred_dir_create un nuevo caso (case) de inicialización de predictores donde a partir de los parámetros definidos del predictor, se reserva la memoria necesaria y se inicializan las variables y tablas a los valores necesarios.
 - Añadir en bpred_config y bpred_dir_config un nuevo caso (case) para mostrar por la salida del simulador la configuración del predictor.
 - Añadir en bpred_reg_stats un nuevo caso (case) para poder ver el resultado de la simulación.
 - Añadir en bpred_lookup y bpred_dir_lookup un nuevo caso (case) para obtener la dirección de la siguiente instrucción, atendiendo a la predicción que realiza a partir de la instrucción de salto que se ha hecho el fetch. (siempre que se encuentre en el BTB). Aclaración: bpred_dir_lookup devuelve la dirección de la entrada de la tabla PHT (L2 Table) y bpred_lookup la guarda en el campo pdir1 i pdir2 de la propia instrucción para que en la siguiente llamada a bpred_update ya tenga calculada la posición en PHT (es una optimización)
 - Añadir en bpred_update el caso para actualizar el los valores del **bimodal** y de **gshare** según indique el algoritmo del **cascade**. Hay que tener en cuenta esta ultima tabla se puede actualizar a partir de la dirección guardada anteriormente (la optimización)

En general la simulación en simplescalar empieza en main.c. Para la funciones que hemos mencionado se llama a `sim_reg_options()`, después a `sim_check_options()` y después a

sim_main(). Las tres están implementadas dentro del código principal del simulador sim-outorder y a partir de este instante empieza la simulación.

La función sim_main implementa el pipeline del superescalar. Llama a diferentes funciones para simular el comportamiento del procesador y de entre ellas nos interesa la función ruu_fetch().

ruu_fetch() realiza la lectura de instrucciones de memoria, simula la cache, el TLB y el predictor de saltos. Al leer la instrucción la decodifica parcialmente y si se da cuenta de que es una instrucción de salto llama a bpred_lookup() para que le devuelva la dirección de la siguiente instrucción que según su predicción se debe seguir buscando en memoria.

La función bpred_update() actualiza la información del predictor a partir de la ejecución real de la instrucción de salto. Se llama generalmente desde el ruu_commit() pero también se puede llamar antes desde el ruu_writeback() o incluso desde ruu_dispatch().

Una vez implementado el predictor se añadirá el comportamiento del mismo a las gráficas anteriores.

Los parámetros a tener en cuenta del nuevo predictor:

- Tamaño del PHTb del bimodal y del PHTg <l2-size> del gshared respectivamente:
(4-8), (8-32), (16-128), (32-512), (64-2048) son (Y-X)
- Ancho del GBHR <g> que serán: 3, 5, 7, 9 y 11
- Parámetros:
 - -bpred cascade
 - -bpred:bimod <Y>
 - -bpred:2lev 1 <X> <g> 1

Finalmente, teniendo en cuenta que se parece al predictor **comb**, realizad las dos siguientes tareas:

- Parámetros:
 - -bpred comb
 - -bpred:comb <Z>
 - -bpred:bimod<Y>
 - -bpred:2lev 1 <X> <g> 1
- Comentad a que valores de <Z> serian los adecuados para compararlo con el predictor **cascade** que se ha simulado
- Una vez identificados los parámetros, simulad ambos predictores y comparad los resultados mostrando el mismo tipo de gráficas.