



Departament d'Enginyeria  Informàtica i Matemàtiques  UNIVERSITAT ROVIRA I VIRGILI	Arquitectura de Computadores
	AC
	Curso 21/22
	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*.

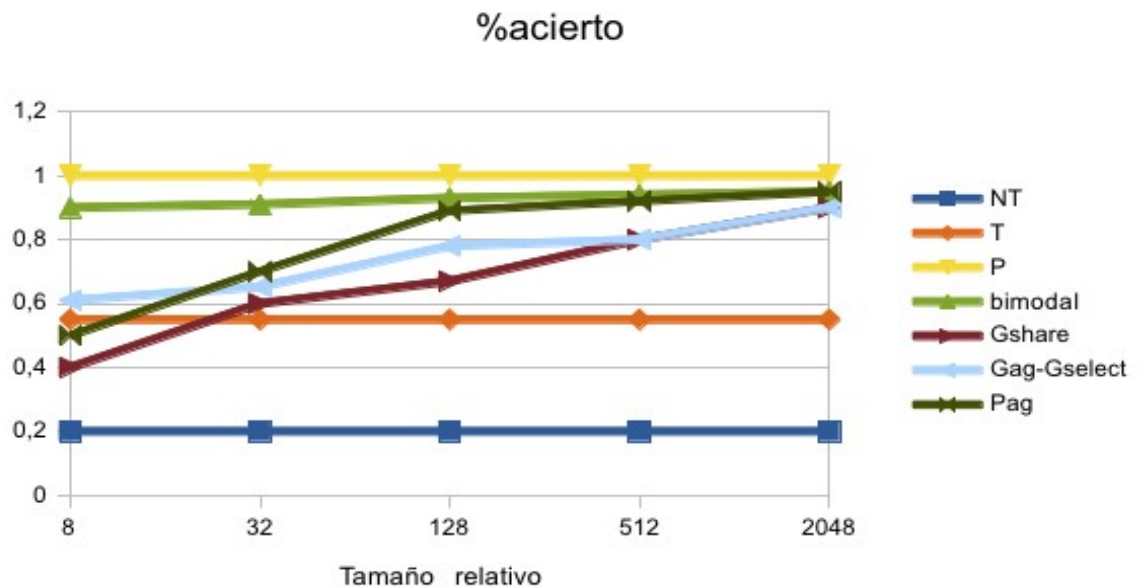
Se usarán para evaluar el comportamiento en cuanto a *IPC* y *porcentaje de aciertos* de las siguientes alternativas de predictores de saltos:

- **nottaken:** opción: `-bpred nottaken`
- **taken:** opción `-bpred taken`
- **perfect:** opción `-bpred perfect`
- **bimodal:** opción `-bpred bimod`
 - Tamaño del PHT $\langle I2\text{-size} \rangle$: 8,32,128,512,2048
 - `-bpred:bimod` `X`
- **Gshare:** opción `-bpred 2lev`
 - Tamaño del BHR $\langle I1\text{-size} \rangle$: 1 y del PHT $\langle I2\text{-size} \rangle$: 8,32,128,512,2048
 - `-bpred:2lev` `1` $\langle X \rangle$ $\langle \log_2 X \rangle$ `1`
- **Gag (Gselect):** opción `-bpred 2lev`
 - Tamaño del BHR $\langle I1\text{-size} \rangle$: 1 y del PHT $\langle I2\text{-size} \rangle$: 8,32,128,512,2048
 - `-bpred:2lev` `1` $\langle X \rangle$ $\langle \log_2 X \rangle$ `0`
- **Pag:** opción `-bpred 2lev`
 - Tamaño del BHR $\langle I1\text{-size} \rangle$ y del PHT $\langle I2\text{-size} \rangle$ respectivamente: (4-4), (8-16), (16-64), (32-256), (64-1024), (32-2048) son (Y-X)
 - `-bpred:2lev` $\langle Y \rangle$ $\langle X \rangle$ $\langle \log_2 X \rangle$ `0`

The graph shows the percentage of correct results (% acierto) on the y-axis (ranging from 0 to 1) against the relative size (Tamaño relativo) on the x-axis (values: 8, 32, 128, 512, 2048). Six data series are plotted: b1 (dark blue squares), b2 (orange diamonds), b3 (yellow crosses), b4 (green triangles), b5 (dark red left-pointing triangles), and media (light blue right-pointing triangles). b4 is the highest, starting at ~0.9 and ending at ~0.95. b5 is the lowest, starting at ~0.4 and ending at ~0.65. The media line starts at ~0.68 and ends at ~0.82. b3 starts at ~0.6 and ends at ~0.9. b1 and b2 start at ~0.7 and end at ~0.83 and ~0.75 respectively.

Tamaño relativo	b1	b2	b3	b4	b5	media
8	0.70	0.70	0.60	0.90	0.40	0.68
32	0.76	0.71	0.70	0.91	0.55	0.72
128	0.79	0.72	0.72	0.92	0.60	0.76
512	0.81	0.73	0.78	0.93	0.62	0.79
2048	0.83	0.75	0.90	0.95	0.65	0.82

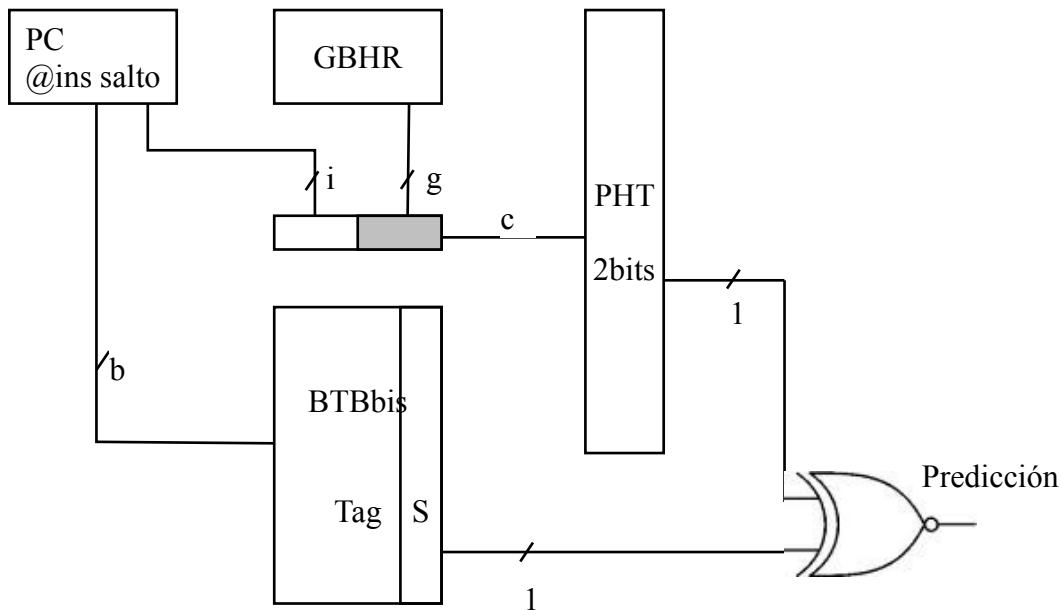
- 3) Para cada estudio se presentarán gráficas con los resultados individuales de cada benchmark y con la media de todos ellos.
- 4) También se generarán gráficas resumen del comportamiento de cada uno de los predictores. En la misma gráfica aparecerán los siete predictores teniendo en el eje de las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



El predictor mantiene la información de saltos ejecutados en cuatro estructuras.

1. 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. (Internamente será un registro de 32 bits donde cogeremos los '*g*' bits de menos peso)
2. Branch Target Buffer (BTBbis) es una tabla de dimensión 2^b entradas y cada una de ellas con 1 Tag y 1 bit de comportamiento del salto. Se accede de manera totalmente asociativa a partir de la dirección de la instrucción de salto. El objetivo de esta tabla es guardar el comportamiento de la instrucción de salto. El Tag asegura que se trata de la misma instrucción y el bit indica Taken (1) o NotTaken (0). De hecho se llena con el comportamiento de la primera vez que se ejecutó. Se podría utilizar el BTB que ya tiene implementado el Sim-outorder, pero para no mezclar más el código es preferible crear un BTB propio (por eso el nombre BTBbis)
3. Pattern History Table (PHT) es una tabla donde se almacenan dos bits para implementar un contador saturado del comportamiento de los saltos. Se modifica ligeramente el sentido de estos dos bits, de manera que cada vez que el salto coincide con el valor que está en el BTBbis se incrementa y cada vez que discrepa se decrementa. Así se obtienen cuatro estados donde dos de ellos (los que tienen el bit de mayor peso a '1') dicen que el bit del BTBbis tiene razón y los otros dos (los que tienen el bit de mayor peso a '0') dicen que el bit del BTBbis no tiene razón. Se accede de manera directa a partir del índice obtenido de la concatenación del GBHR (*g* bits) y de bits de la dirección de la instrucción de salto (*i* bits).

4. La predicción se obtiene a partir de la NXOR del bit de mayor peso del PHT y del bit del BTBbis. Así: $(PHT_1, BTBbis) \rightarrow \text{Predicción}$
- (00)->1 indica 'mal predicho NotTaken \rightarrow Taken',
 - (01)->0 indica 'mal predicho Taken \rightarrow NotTaken',
 - (10)->0 indica 'bien predicho NotTaken \rightarrow NotTaken',
 - (11)->1 indica 'bien predicho Taken \rightarrow Taken',

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.

La predicción se obtiene a partir de la función NXOR del bit de más peso de la entrada seleccionada del PHT y el bit comportamiento (S) del BTBbis. Pensad que BTBbis es complementamente asociativa y se requerirá acceder a todas las entradas para comprobar si se produce un HIT o un MISS.

La actualización se realiza siempre en el GBHR y en la PHT principal. El BTBbis sólo se actualiza si se ha producido un MISS en el mismo y se añade la nueva instrucción de salto con el primer comportamiento que haya tenido. Al ser complementamente asociativa necesita un algoritmo de reemplazo y este será el LRU.

Los parámetros a definir en este predictor són:

1. Número de bits del GBHR \rightarrow define 'g'
2. Número de entradas del BTBbis \rightarrow define 'b'
3. Número de entradas del PHT \rightarrow define 'c'
4. A partir de estos valores se deduce 'i' como 'c'-'g' (que no puede ser inferior a 1).

Los valores iniciales de estas estructuras son todo bits a '1'. De este modo la predicción de cualquier salto inicial será Taken.

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 **2lev** es suficientemente parecido y será el modelo a seguir.

Las partes de simplescalar a modificar son:

- En bpred.h:
 - Añadir el nuevo predictor en la lista de enum bpred_class.
 - En la definición de tipos de los predictores añadir en la estructura struct bpred_dir_t dentro de la subestructura two las variables y apuntadores extras. Tened en cuenta que level-1 podría ser BTBbis y level-2 podría ser PHT.

- En `sim-outorder.c`:
 - Registrar en `sim_reg_options` los parámetros de configuración del predictor: con `opt_reg_int_list(opt, "-bpred:agree",`
 - 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(,,,,,,)`. Es recomendable fijarse en el 2lev.
- 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` 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 GBHR, el BTBbis (L1 Table) y el PHT (L2 Table) según corresponda. Hay que tener en cuenta que estas últimas tablas se pueden actualizar a partir de las direcciones guardadas anteriormente (la optimización)

En general la simulación en simpleescalar 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. De este mod, 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:

- **Agree:** opción `-bpred agree`
 - Tamaño del BTBbis `<l1-size>` y del PHT`<l2-size>` respectivamente: (4-8), (8-32), (16-128), (32-512), (64-2048) son (Y-X)
 - Ancho del GBHR `<g>` que serán: 2, 3, 4, 6 y 8
 - Así la configuración de `c=i+g` quedaría: (1+2->3), (2+3->5), (3+4->7), (3+6->9), (3+8->11)
 - `-bpred:agree <Y> <X> <g> 0`

Finalmente, realizad las dos siguientes tareas:

- Comentad a que otro predictor Gag, Pag, Gap, Pap se le parece y con qué parámetros.
- Una vez identificado el predictor ya existente en SimpleScalar al que se le parece el predictor implementado, simulad ambos predictores y comparad los resultados.