Departament d'Enginyeria Informàtica i Matemàtiques
[DEIM] Universitat Rovira i Virgili

Arquitectura de Computadores
AC
Curso 17/18
Primera Convocatoria
Práctica 2: Comportamiento predictores de salto *Simplescalar*

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 evaluara 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). Y por último, un Pattern History Table (PHT) almacena la predicción que se hará en un posible salto. Almacena 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 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: Estudio de los predictores que se pueden simular con el Simplescalar y 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 en predictores de saltos:

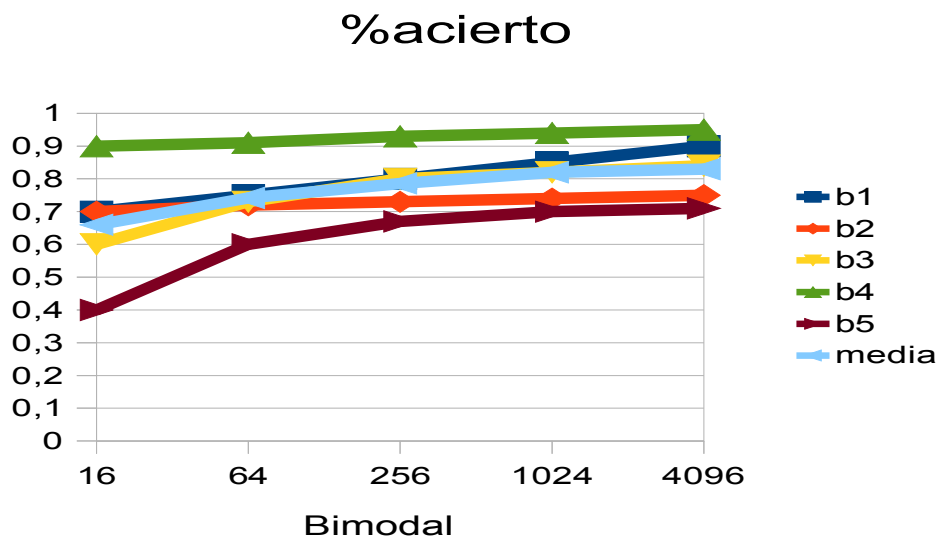
- **nottaken:** opción: `-bpred nottaken`
- **taken:** opción `-bpred taken`
- **perfect:** opción `-bpred perfect`
- **bimodal:** opción `-bpred bimod`
  - Tamaño del PHT <l2-size> : 16,64,256,1024,4096
  - `-bpred:bimod X`
- **Gshare:** opción `-bpred 2lev`
  - Tamaño del BHR <l1-size>: 1 y del PHT<l2-size>: 16,64,256,1024,4096
  - `-bpred:2lev 1 <X> <log2X> 1`
- **Gag (Gselect):** opción `-bpred 2lev`
  - Tamaño del BHR <l1-size>: 1 y del PHT<l2-size>: 16,64,256,1024,4096
  - `-bpred:2lev 1 <X> <log2X> 0`
- **Pag:** opción `-bpred 2lev`
  - Tamaño del BHR <l1-size> y del PHT<l2-size> respectivamente: (8-8), (16-32), (32-128), (64-512), (128-2048), (64-4096) son (Y-X)
  - `-bpred:2lev <Y> <X> <log2X> 0`

La idea es observar y obtener 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 de que se disponen (applu, crafty, twolf, vortex y vpr)

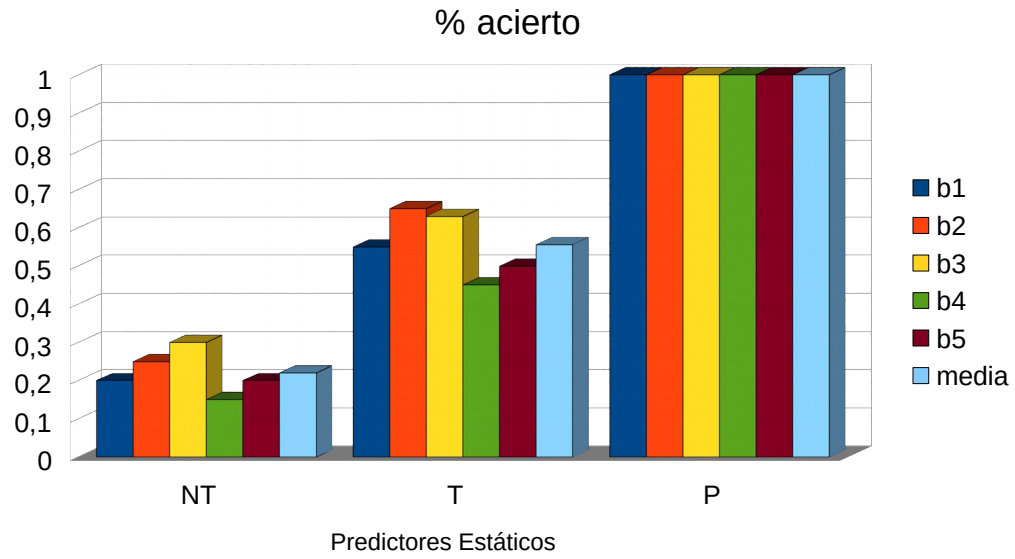
Los resultados se mostrarán en gráficas.

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, 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.

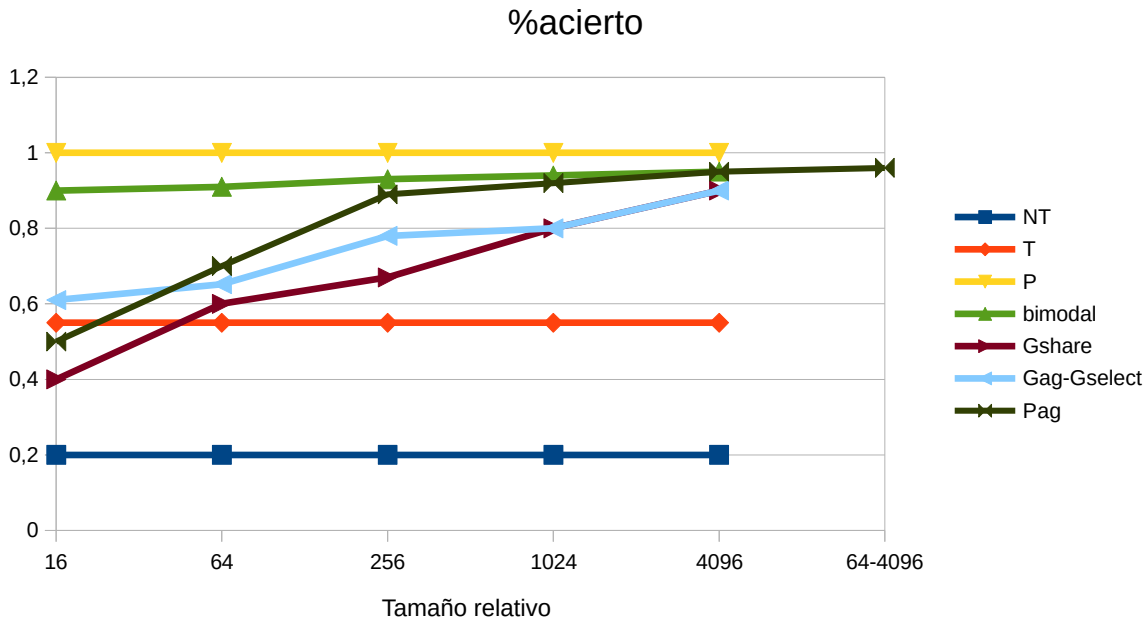


2) Para los predictores estáticos se pueden resumir todos en una única gráfica de la forma:



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.



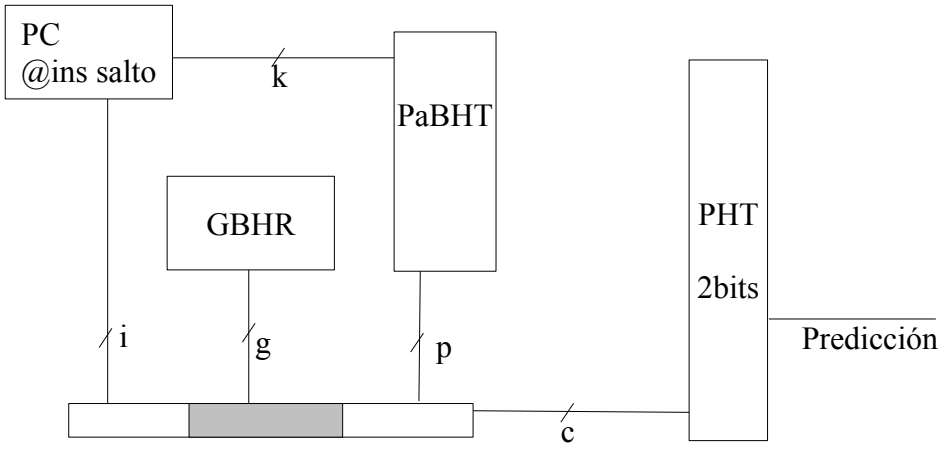
5) Para cada benchmark simulado, se saltarán 50 millones de instrucciones y se recolectarán las estadísticas para los siguientes 10 millones. Además, se utilizarán los datos de entrada REF.

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 ALLOYED.



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. *Per address Branch History Table* (PaBHT) es una tabla de dimensión 2^k entradas y cada una de ellas con 'p' bits. El objetivo de esta tabla es guardar la historia de los últimos 'p' saltos de cada instrucción por separado. Se utiliza la dirección en memoria de las instrucciones de salto como identificador de ellas mismas. Como la tabla tiene una número limitado de entradas, se utilizan únicamente los 'k' bits de menor peso para seleccionar una entrada para una instrucción de salto. Como pega, sucede que aquellas instrucciones de salto que tengan los mismos 'k' últimos bits compartirán la misma entrada y almacenarán una historia mezclada (Aliasing).
3. Pattern History Table (PHT) es una tabla donde se almacenan dos bits para implementar un contador saturado (2BitCounter) del comportamiento de los saltos. El bit de mayor peso indica la predicción. Cada vez que se tiene que actualizar, el valor se incrementa en caso de Taken y se decrementa en caso de NotTaken. Siempre se actualiza con el resultado real de la ejecución del salto. Se accede de manera directa a partir del índice obtenido de la concatenación del PaBHT (p), del GBHR (g) y de bits de la dirección de la instrucción de salto (i).

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 del bit de más peso (del 2BitCounter) de la entrada seleccionada del PHT.

La actualización se realiza siempre en el GBHR, EL PaBHT y en la PHT correspondiente.

Los parámetros a definir en este predictor són:

1. Número de bits del GBHR → define 'g'
2. Número de entradas de PaBHT y bits de cada entrada → define 'k' y 'p'
3. Número de entradas del PHT→ define'c'
4. A partir de estos valores se deduce 'i' como 'c'-'g'-'p'. Que no puede ser inferior a 1.

Los valores iniciales de estas estructuras son 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 tanto el predictor ***2lev*** como el ***comb*** son suficientementes parecidos y serán 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 PaBHT 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:alloy",.....`

    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 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 PaBHT (L1 Table) y el PHT (L2 Table). 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 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 descodifica 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:

- **Alloy**: opción -bpred alloy
  - Tamaño del PaBHT <l1-size> y del PHT<l2-size> respectivamente: (8-8), (16-32), (32-128), (64-512), (128-2048), (64-4096) son (Y-X)
  - Ancho del GBHR <g> y PaBHT <p> que concatenados con 'i' forman 'c' serán: (1-1), (2-2), (3-2), (3-3), (4-4) y (4-4)
  - Así la configuración de c=i+g+p quedaría: (1+1+1->3), (1+2+2->5), (2+3+2->7), (3+3+3->9), (3+4+4->11), (4+4+4->12).
  - -bpred:alloy <Y> <X> <p> <g> 0

Responded en el informe: A que otro predictor Gag, Pag, Gap, Pap se parece? Con que parámetros? Simulad y comparadlo con el *alloy* aquí implementado.