Sistemas Informáticos
Curso 2003 - 2004

Herramienta para la simulación y visualización de procesadores superscalares

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

Dirigido por:
Manuel Prieto Matías y Luis Piñuel Moreno
Dpto. Arquitectura de Computadores y Automática

Facultad de Informática
Universidad Complutense de Madrid

ÍNDICE

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

RESUMEN DEL PROYECTO DE SISTEMAS INFORMÁTICOS

Nuestro proyecto ha consistido en la elaboración de predictores de saltos para usar en el simulador Simplescalar. Simplescalar es una potente herramienta que permite la simulación de un procesador superescalar, desde distintos puntos de vista. Uno de estos puntos es la predicción de saltos, algo fundamental para el buen funcionamiento y rendimiento de un microprocesador.

El simulador se encuentra dividido en varios módulos, cada uno de ellos tiene el código abierto, por lo que se permite su modificación para así poder asegurar a los investigadores que puedan probar con comodidad aquello en lo que estén interesados. El lenguaje de este código es C, y está estructurado y modulado de tal forma que se permiten hacer cambios con relativa facilidad.

Para elaborar nuevos predictores hemos tenido que añadir nuevos tipos de datos, así como cambiar algunos de los ya existentes. También se tuvieron que modificar algunos de los métodos existentes, así como también tuvimos que crear alguno nuevo.

Una vez implementamos en el simulador los distintos predictores (a saber, Agree, Bi-Mode, Skew, Filter, YAGS y Perceptron) se realizaron distintas pruebas con multitud de benchmarks de diferentes características para comprobar cómo afectaban al rendimiento del computador cada una de las configuraciones.

Este estudio es importante porque una buena predicción de los saltos nos proporciona un rendimiento muy superior, ya que podemos ejecutar múltiples instrucciones a la vez sin tener que volver hacia atrás.

Los predictores que implementamos han sido muy distintos entre sí, por lo que los resultados que hemos obtenido son significativos, ya que nos permiten hacernos una idea de cómo se va a comportar cada tipo de predictor en un computador real, no simulado.

Además, para poder realizar una comparación más veraz, realizamos comparaciones con los predictores que ya estaban implementados, para así comprobar con datos ya contrastados el trabajo que habíamos realizado.

En la memoria que presentamos se pueden observar todos estos datos, así como las conclusiones que para cada tipo de predictor hemos obtenido, y también un resumen de lo que cada benchmark probado hacía y por qué era importante su estudio. Además presentamos el código en C de los módulos que hemos modificado, así como otros datos de interés que ayudan a comprender la importancia y validez del proyecto que hemos realizado.

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

INFORMATIC SYSTEMS PROJECT'S SUMMARY

Our project deals with the elaboration of branch predictors to use them with the Simpescalar simulator. Simplescalar is a powerful tool that allows a superscalar processor's simulation from several viewpoints. One of these points is branch prediction, which is basic to have a good work and performance in a microprocessor.

The simulator is divided into several modules, each one of them with their open code. That is the reason why it's permitted its modification in order to make sure to investigators a comfortable testing of the topics they are interested in. The language of this code is C, and it is structured and moduled in such form that allows to make changes with relative easiness.

To make new predictors, we had to add new data types and change some of the existing. We also had to change some of the existing methods and add some new.

Once the different predictors were implemented in the simulator (Agree, Bi-Mode, Skew, Filter, YAGS and Perceptron), we did several tests with many benchmarks with different characteristics to check the way in which each of the configurations affect to the processor's performance.

This study is important as a good branch prediction gives a much higher performance, and we can execute several instructions without having to return back.

The branch predictors that we have implemented were very different. This is the reason why the results we obtained are significant, as we can make an idea about the way a real computer, not simulated, is going to tolerate each type of predictor.

Besides, in order to be able to make a truthful comparison, we compared our predictors with the ones that they had been already implemented, in order to check the work that we have made with confirmed data.

These data can be observed in the paper we submit, together with the conclusions we have got for every branch predictor type. We have also included a summary of what each benchmark used did and why its study was important. In addition we submit the C code of the changed modules as well as other interesting data that help to know the importance and validity of the project that we have made.

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

LAS 10 PALABRAS CLAVE DEL PROYECTO

A continuación ponemos la lista de las 10 palabras claves de este proyecto, que han de servir para futuras búsquedas y referencias bibliográficas:

Acierto
Benchmark
Dirección
Perceptrón
Predicción
Predictor
Rendimiento
Salto
Simplescalar
Simulador
Superescalar

Introducción

El proyecto que a continuación presentamos, aunque en principio consistía tanto en la simulación como en la visualización, finalmente ha consistido en la implementación de nuevas funcionalidades para una herramienta de simulación de microarquitecturas: el SimpleScalar. Esta herramienta posee algunas limitaciones en cuanto a la cantidad de estructuras que es capaz de simular, ya que la cantidad de ellas que existen son demasiadas, por lo que nuestro trabajo se ha basado en la ampliación de sus posibilidades en un determinado aspecto: predicción de los comportamientos de los saltos.

SimpleScalar proporcionaba una implementación que permitía simular la ejecución de programas usando tres tipos de predictores de salto: predictor de dos niveles (Gshare), predictor bimodal y un predictor que combina ambos (meta predictor que selecciona aquel que mejor se comporte en cada salto). Debido a la gran importancia que posee para la mejora del rendimiento una buena predicción del comportamiento futuro de los saltos, evitando así la ejecución de instrucciones no válidas y por tanto un aumento del IPC, hemos estado trabajando a lo largo de este curso en la ampliación de este código del SimpleScalar de modo que permita simular otros muchos predictores de salto cuyo comportamiento es de gran interés en este campo de la arquitectura de computadores.

Además de su implementación, presentamos a continuación un estudio de cada uno de ellos y una comparativa de los resultados obtenidos sobre una serie de benchmarks utilizando estos nuevos predictores, y los que esta herramienta ya tenía implementados.

SimpleScalar

1. *¿Qué es SimpleScalar?*

SimpleScalar es una herramienta utilizada para la simulación de microarquitecturas superesclares, reproduciendo el comportamiento exacto que se obtendría con ellas: para unos parámetros de entrada que serían los que recibiría la arquitectura a simular, se obtienen cuáles serían las salidas que proporcionaría, así como también, unas medidas del rendimiento de dicho sistema.

Este simulador proporciona un conjunto de herramientas también utilizadas para el diseño de computadores y análisis de sus estructuras.

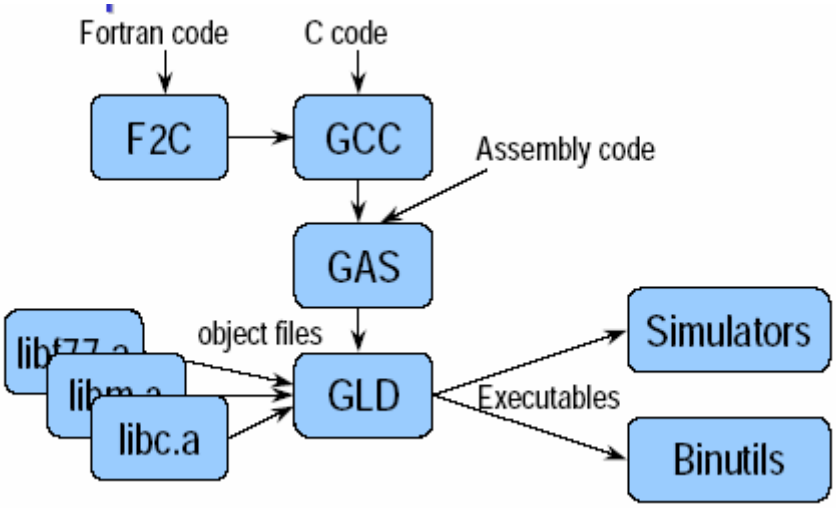
La razón por la que se usa este simulador es, al igual que ocurre en el uso de simuladores en todos los campos, que el hecho de usar un simulador en ámbitos de investigación, posee un gran número de ventajas frente a la investigación directa sobre el H/W: mayor rapidez, mayor flexibilidad, posibilidad de validación del H/W previa a su lanzamiento al

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

mercado, posibilidades de mejora en el sistema, etc. Pero además, presenta una serie de ventajas adicionales:

- Extensible: todo el código fuente se encuentra disponible, tanto de simuladores, librerías, compilador, etc.
- Portable: es soportado por la mayoría de las plataformas utilizadas actualmente.
- Gran nivel de detalle: incluye ejemplos de simuladores. Soporta ejecuciones de instrucciones erróneas y especulación sobre el control y los datos del programa.
- Rendimiento: Sim-Fast (10+MIPS). Sim-Out-Order (350+KIPS).

A modo intuitivo, podemos ver cómo esta herramienta genera la simulación de los programas pedidos:



2. *Perspectiva histórica de los procesadores Superescalares*

El procesamiento Superescalar, la capacidad para iniciar varias instrucciones durante el mismo ciclo de reloj, es la última de una larga serie de innovaciones arquitectónicas, cuyo objetivo era el producir microprocesadores cada vez más rápidos. Introducidos a principios de la década de los 90, los microprocesadores superescalares están siendo ahora diseñados y producidos por todos los fabricantes de microprocesadores de alto nivel. Aunque muchos lo han visto como una extensión de los computadores con juego de instrucciones reducido (Reduced Instruction Set Computer, RISC) (que surgieron en los 80), las implementaciones superescalares están, de hecho, liderando un incremento de la complejidad. Los métodos superescalares están siendo aplicados conjuntos de instrucciones desde el spectrum, o el DEC Alpha (los "nuevos" juegos de instrucciones RISC), hasta juegos de instrucciones que nada tienen que ver con la idea RISC, como pueden ser los Intel x86.

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

Un procesador superescalar normal busca y decodifica la instrucción entrante como si fuera un flujo de varias instrucciones a la vez. Como parte del proceso de búsqueda de la instrucción, la salida de las instrucciones de salto condicional son normalmente predichas, intentando asegurar la no interrupción del flujo de instrucciones. La instrucción entrante es analizada, en busca de dependencias de datos, y después se distribuirán a las unidades funcionales, de acuerdo con el tipo de la instrucción. Lo siguiente es la inicialización de las instrucciones para la ejecución en paralelo (basada en primer lugar en la disponibilidad de los operandos), en lugar de seguir el orden secuencial del programa. Esta es una característica importante, presente en muchas de las implementaciones de procesadores superescalares, y es denominada Programación dinámica de instrucciones. Cuando la ejecución de la instrucción se completa, los resultados son redirigidos para que puedan ser usados para actualizar el estado del proceso en el orden correcto del programa (el orden secuencial original), cuando la condición de interrupción tuvo lugar. Como las instrucciones individuales son las entidades que se están ejecutando en paralelo, los procesadores superescalares explotan lo que se llama Paralelismo a Nivel de Instrucción (ILP, de sus siglas en inglés).

El paralelismo a nivel de instrucción se ha venido usando desde hace décadas. El pipeline actúa como una cadena de montaje, en la que las instrucciones son procesadas en varias fases a lo largo del pipeline. Con el pipelining simple, una sola instrucción era iniciada, pero varias instrucciones pueden estar ejecutandose a la vez, en distintas fases.

El pipelining fue desarrollado a finales de los 50, y llegó ser algo fundamental en los grandes computadores de los 60. El CDC 6600 usaba un pipelining gradual, siendo su ILP logrado gracias al uso de varias unidades funcionales en paralelo. Aunque era capaz de mantener la ejecución de sólo una instrucción por ciclo, los conjuntos de instrucciones, las unidades funcionales en paralelo y la programación dinámica de instrucciones de los 6600 era muy parecida a los de los procesadores superescalares de hoy en día. Otro procesador a tener en cuenta en los 60 es el IBM 360/91, que tenía mucho pipelining y disponía de una programación dinámica de instrucciones conocida como Algoritmo de Tomasulo (gracias a su inventor). Tal y como pasaba con el CDC 6600, el IBM 360/91 solo permitía una instrucción por ciclo, y no era superescalar, aunque el algoritmo de Tomasulo es una influencia muy evidente en los procesadores superescalares de hoy.

El lanzamiento de una sola instrucción por ciclo continuó durante muchos años, y siempre se creyó que sería un grave cuello de botella para el avance de la velocidad en los procesadores. Mientras, se iban descubriendo otras vías para mejorar el rendimiento por medio del uso del paralelismo, como por ejemplo el procesamiento vectorial y el multiprocesamiento. Aunque algunos procesadores eran capaces de lanzar varias instrucciones durante los 60 y 70, nunca eran puestos en el mercado. Fue a mediados de los 80 cuando comienzan a aparecer los procesadores superescalares.

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

Disponían de múltiples pipelines en los que se podían lanzar a la vez varias instrucciones, y con esto se deshacía el cuello de botella que suponía el lanzamiento individual de instrucciones en cada ciclo. En los años posteriores a su introducción en el mercado, la mejora de estos procesadores ha sido tal que se ha convertido en un método estándar para la implementación de procesadores de alto rendimiento.

En la actualidad, los procesadores superescalares más utilizados son el MIPS R10000 (en el que se basa la arquitectura del simulador usado durante el proyecto, el SimpleScalar), el DEC Alpha 21164 (con un juego de instrucciones de gran simplicidad) y el AMD K5 (este último implementa el juego de instrucciones de los Intel x86).

3. *La arquitectura del SimpleScalar*

La arquitectura del SimpleScalar está derivada del procesador MIPS-IV ISA. El conjunto de herramientas define tanto la versión little-endian como la versión big-endian de la arquitectura para facilitar la portabilidad. La semántica del SimpleScalar ISA es un superconjunto del MIPS con las siguientes diferencias y añadidos importantes:

- Loads, Stores y transferencias de control no ejecutan instrucciones sucesivas (no hay slots de retraso en la arquitectura).

- Loads y Stores permiten dos tipos de direccionamiento (para todos los tipos de datos) además de los que se encuentran en la arquitectura del MIPS. Estos son: indexado (registro – registro) y auto-incremento (o auto-decremento).

- Instrucción de raíz cuadrada, que implementa la raíz cuadrada de números en coma flotante, tanto de precisión simple como doble.

- Codificado de instrucciones de 64 bits extendido.

Todas las instrucciones tienen 64 bits de longitud. El formato de instrucciones de registro es usado para instrucciones computacionales. Este formato permite la inclusión de constantes de 16 bits. Las instrucciones de salto permiten la especificación de destinos de longitud igual a 24 bits. Los campos de registro serán todos de 8 bits, para permitir la extensión de los registros de la arquitectura hasta 256 registros enteros y en coma flotante. Cada formato de instrucción tiene una localización fija, 16 bits de código de operación que facilitan el decodificado de la instrucción.

El campo annote tiene una longitud de 16 bits, que puede ser modificado después de la compilación, con anotaciones de instrucciones de los archivos del ensamblador. La interfaz de anotación es útil para sintetizar nuevas instrucciones sin tener que cambiar y recompilar el

ensamblador. Las anotaciones están atadas al código de operación, y tienen dos comportamientos: anotaciones de campo y anotaciones de bit.

Las llamadas al sistema en SimpleScalar están dirigidas por un handler ubicado en el archivo syscall.c que intercepta las llamadas al sistema hechas por el binario simulado, decodifica la llamada al sistema, copia sus argumentos, y realiza la correspondiente llamada en el sistema operativo de la máquina que ejecuta el simulador, y por último copia el resultado de la llamada en la memoria del programa simulado. Si se lleva SimpleScalar a una plataforma nueva, se necesitan traducir las llamadas al sistema de la nueva máquina en syscall.c.

4. *Módulos que componen SimpleScalar*

A nivel software, simpleScalar posee diversos módulos en los que encontramos archivos con extensión ".c", en los que encontramos la implementación de las interfaces que el módulo exporta, los archivos del mismo nombre pero con extensión ".h" en los que únicamente se encuentra la definición de dichas interfaces y los archivos cuyo nombre es de la siguiente forma: sim-*.c, módulos que implementan la simulación de una parte de la arquitectura. Todos ellos son:

- bpred.h/c – Predictores de salto.
- cache.h/c – Módulo para la caché.
- eventq.h/c – Cola de eventos.
- libcheetah/ – Librería de simulación de la caché .
- ptrace.h/c – Módulo de trazas del pipe.
- res.h/c – Módulo de recurso.
- sim.h – Definiciones de las interfaces del código principal del simulador.
- textprof.pl – Script en Perl.
- pipeview.p – Script en Perl.
- dlite.h/c – Módulo para el debug.
- eio.h/c – Módulo de trazas para operaciones exteranas de E/S.
- loader.h/c – Cargador del programa.
- memory.h/c – Módulo para el espacio de memoria.
- regs. h/c – Módulo de registros.
- machine.h/c – Rutinas de interfaces ISA.
- machine.def – Definición de la arquitectura ISA de SimpleScalar.
- symbol.h/c – Módulo para la tabla de símbolos.
- syscall. h/c – Módulo de implementación de las llamadas al sistema.
- eval.h/c – Evaluador de expresiones genéricas.
- libexo/ - Librería de estructuras de datos.
- misc.h/c – Miscelánea.
- options.h/c – Paquete de opciones.
- range.h/c - Paquete de rangos para las expresiones.

- stats.h/c - Paquete de estadísticas para evaluar los resultados que proporciona el simulador.

De todos ellos, el llamado bpred.c / bpred.h es el que contiene las implementaciones de los predictores de salto. Es en este módulo, en el que ha quedado principalmente concentrado nuestro trabajo aunque también nos ha sido necesario modificar el módulo principal que mandaba simular el funcionamiento de todos los predictores de salto: sim-bpred.c.

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

INTRODUCCIÓN A LOS BENCHMARKS

1. ¿Qué es un benchmark?

Un benchmark es un conjunto de procedimientos (programas de computación) para evaluar el rendimiento de un ordenador. Estos procedimientos se ejecutan sobre una máquina con el objetivo de estimar el rendimiento de un elemento en concreto, o la totalidad de la misma, y poder comparar los resultados con máquinas similares. Ello nos permite conocer sus prestaciones y el ratio coste/prestaciones.

Por ejemplo, un benchmark podría ser realizado en cualquiera de los componentes de un ordenador, ya sea CPU, RAM, tarjeta gráfica, etc. e incluso ser dirigido específicamente a una función dentro de un componente, por ejemplo, la unidad de coma flotante de la CPU.

Existen cuatro categorías generales de benchmarks:

- Pruebas aplicaciones-base (application-based) las ejecuta y las cronometra.

- Pruebas playback (playback test), las cuales usan llamadas al sistema durante actividades especificas de una aplicación(Ej.: Llamados a gráficos o uso del disco) y las ejecuta aisladamente.

- Prueba sintética (synthetic test) , la cual enlaza actividades de la aplicación en subsistemas específicos.

- Prueba de inspección (inspection tests), la cual no intenta imitar la actividad de la aplicación, sino que las ejecuta directamente en los subsistemas específicos.

- Algunas de estas pruebas de rendimiento o benchmarks, son programas software creados específicamente con este objetivo; se les denomina “pruebas de sintéticas”. Por el contrario, otras de ellas evalúan el rendimiento usando aplicaciones reales en la forma en que se utilizan.

Todas las técnicas de benchmarking, proporcionan sus resultados en base a unos aspectos básicos:

- Medición de las prestaciones más relevantes.
- Portabilidad, facilidad de implementación en diferentes sistemas informáticos.
- Escalabilidad, aplicable también a sistemas con numerosos elementos de proceso.
- Simplicidad.

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

2. Nuestros benchmarks

Para nuestra investigación en este proyecto, hemos utilizado dos conjuntos de benchmarks con los que hemos realizado todas nuestras pruebas. A lo largo de la presente documentación, haremos constantes referencias a ellos y a los resultados de nuestras investigaciones sobre unos u otros. Ambos pertenecen a la organización SPEC:

La Corporación para Estandarización de las Evaluaciones de Rendimiento (**Standard Performance Evaluation Corporation (SPEC)**) es una organización sin ánimo de lucro creada para establecer, mantener y reforzar una conjunto de benchmarks relevantes, estandarizados para ser aplicados a la más moderna generación de computadores de alto rendimiento. SPEC desarrolla conjuntos de programas y también revisa y publica aquellos benchmarks creados por sus miembros y otras organizaciones.

Una de las suites más reciente es el conocido como **SPEC CPU2000**, en el cual nos hemos basado; lo hemos hecho tomando dos de sus posibles representaciones: en modo traza y en modo "ventana completa de ejecución". El modo traza viene caracterizado porque el benchmark concreto no tiene ni entradas ni salidas, la traza contiene todo lo que sería el flujo de programa desplegado, es decir, todo el conjunto de instrucciones que un procesador recibiría a lo largo de una ejecución completa. Esto es un modo de "limitar" las entradas, y de controlar la ejecución del programa. La traza es la ordenación lineal de las instrucciones de acuerdo con el flujo de programa. En cuanto al modo "ventana completa", lo que se carga es una ejecución "real" del programa, en la que en tiempo de ejecución se van solicitando entradas y produciéndose salidas, por lo que es una ejecución más real y menos limitada. Sus resultados pueden ser considerados como más correctos, o al menos, como más cercanos a lo que se obtendría en una ejecución normal. El modo de ventana completa requiere que cada benchmark sea invocado con unos parámetros específicos, mientras que en el modo traza, no suele necesitar una invocación específica.

A continuación, exponemos los benchmarks pertenecientes a ambos conjuntos, sobre los cuales hemos realizado nuestras mediciones:

- Benchmarks de "Trazas 2000":

  En cuanto a las trazas, sólo hemos medido basándonos en aquellas que sobrecargan la unidad entera de la ALU. Las trazas escogidas para las simulaciones son las siguientes (para una mayor concreción de lo que hace cada benchmark, mirar lo referido en "Ejecuciones Completas"):

  - gcc.scilab. Carga el compilador C con el programa scilab.i
  - gzip.source. Se comprime un archivo tar.
  - mcf. Optimización combinatoria
  - parser. Procesamiento de palabras.

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

- perl.splitmail. Se carga el compilador perl con el script splitmail
- twolf. Simulador de enrutamiento y localización
- vortex.1. Sólo ejecuta la primera de las 3 posibles taréas de enrutamiento
- vpr.place. Recibe el emplazamiento de un circuito y devuelve el enrutamiento.

- Benchmarks de "Ejecuciones Completas", modo ventana completa de ejecucion:

  - CINT2000 (Componentes Enteros de SPEC CPU2000):

Trabajan sobrecargando la unidad entera de la ALU del procesador

Benchmark	**Lenguaje en que está escrito**	**Área de aplicación**
164.gzip	C	Compresión
175.vpr	C	Enrutamiento y deseño de circuitos con FPGAs
176.gcc	C	Compilador de lenguaje C
181.mcf	C	Optimización combinatoria
186.crafty	C	Juego de ajedrez
197.parser	C	Procesamiento de palabras
252.eon	C++	Vision computacional
253.perlbmk	C	Lenguaje de programación PERL
254.gap	C	Interprete de teoría de grupos
255.vortex	C	Base de datos orientada a objetos
256.bzip2	C	Compresión
300.twolf	C	Simulador de enrutamiento y localización

  - CFP2000 (Componentes en Coma Flotante de SPEC CPU2000)

Trabajan sobrecargando la unidad en coma flotante de la ALU del procesador

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

Benchmark	Lenguaje en que está escrito	Área de aplicación
168.wupwise	Fortran 77	Física/ Cromodinámica
171.swim	Fortran 77	Modelado bajo el agua
172.mgrid	Fortran 77	Solucionador multi-grid: campos potenciales en 3D
177.mesa	C	Librería gráfica en 3-D
178.galgel	Fortran 90	Dinámica de fluidos
179.art	C	Reconocimiento de imágenes / Redes neuronales
183.equake	C	Simulación de propagación de ondas sísmicas
187.facerec	Fortran 90	Procesamiento de imágenes: Reconocimiento facial
188.ammp	C	Química computacional
189.lucas	Fortran 90	Teoría de números / Test de primalidad
191.fma3d	Fortran 90	Simulación de accidentes con elementos finitos

3. Bibliografía

[1] *http://www.specbench.org/osg/cpu2000/*
[2] *http://es.wikipedia.org*
[3] Carlos García Sánchez, *Evaluación de Rendimiento de Configuraciones (ERC), Universidad Complutense, 2003*

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

MODELADO DE PROCESADORES SUPERESCALARES USANDO MÉTODOS ESTADÍSTICOS

1. Motivación

La simulación estadística es una técnica para la evaluación rápida del rendimiento de procesadores superescalares. En primer lugar, se recoge información estrictamente estadística desde un programa de simulación detallado. Esta información es usada para generar una traza de instrucciones que será usada en un procesador simple con sus predicciones de salto y caché estadísticas. Por la naturaleza probabilística de la simulación, rápidamente se converge a la tasa de rendimiento esperada. La simplicidad y la velocidad de la simulación permiten que esta técnica ofrezca un modelo de diseño rápido y útil, además de representar un buen complemento para los análisis más detallados.

La eficacia de esta técnica es evaluada para distintos niveles de complejidad del modelado. Tanto los errores como las propiedades de convergencia deben de ser estudiadas en detalle. En un modelo de instrucción simple comete un error medio del 8 % comparado con una simulación más detallada. En un modelo de instrucción más detallado este error se reduce al 5 %, pero se necesita un tiempo tres veces superior para llegar a la convergencia con el modelo simulado.

2. Información más detallada - Metodología

La simulación es muy usada como herramienta efectiva para evaluar el rendimiento de un computador. Para los procesadores superescalares los modelos muy detallados de simulación son desarrollados, normalmente, para aportar decisiones de diseño y permitir mejorar el rendimiento esperado antes de la fabricación del chip. También se usa esta técnica en el entorno de las investigaciones en las mejoras producidas en las diferentes innovaciones que se puedan añadir a una determinada microarquitectura. Para estos modelos detallados de procesador, los tiempos de simulación son relativamente largos, requiriendo a menudo varias horas para una ejecución simple. Además, las simulaciones están basadas en benchmarks específicos y consecuentemente sólo producen información sobre el rendimiento para programas similares a estos benchmarks. Una simulación detallada es crítica para evaluar los puntos de diseño específicos y es usada tanto a nivel académico como a nivel industrial.

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

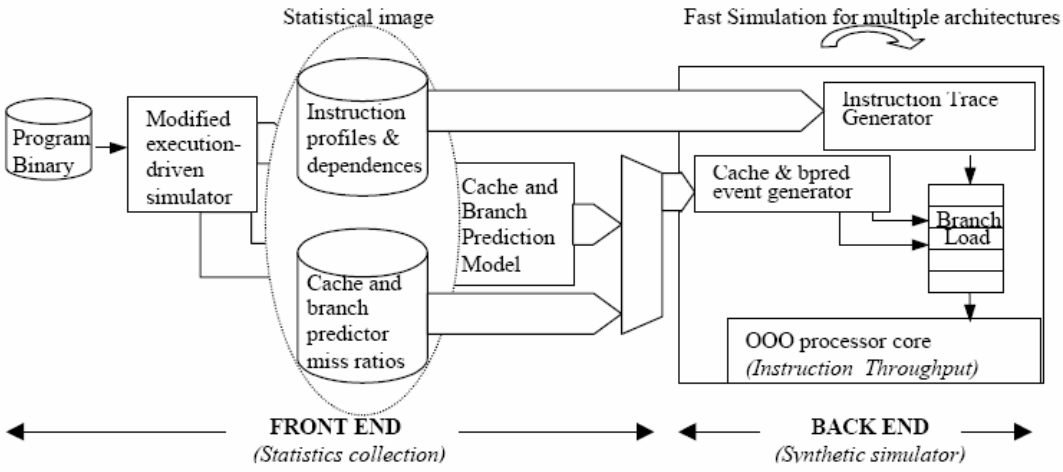


Figure 1: Statistical simulation process flow

En la figura 1 podemos observar cuál es el proceso normal en la simulación de una arquitectura. Lo que se hace en primer lugar es una simulación del benchmark a tratar mediante un simulador que nos proporcionará una traza dinámica del código ejecutado. La traza de instrucciones del programa es analizada para obtener unas tablas estadísticas de las *características intrínsecas del programa* y otras estadísticas sobre *eventos localizados*. Las características intrínsecas del programa suelen ser independientes de la arquitectura en la que se ejecute el benchmark (no hay información ni de especulación ni de caché), pero pueden depender del compilador usado. Se consideran eventos locales a los fallo en la predicción de saltos o fallos de caché, que dependen específicamente de la arquitectura usada en la simulación, así como la caché que posea el equipo y el tipo de predictor de saltos usado. Esta parte del proceso es la que aparece como *Front End* en la figura 1. Esta primera parte sólo se ha de realizar una vez por cada benchmark que se quiera estudiar.

Después, para llevar a cabo la simulación estadística, las estadísticas anteriores del programa son usadas para guiar la generación aleatoria de una traza de ejecución sintética. Esta traza de ejecución sintética, que incluye los saltos y los aciertos y fallos en la caché es llevada a un simulador de procesador superescalar. El modelo de procesador suele estar simplificado, recreando la mayoría de los recursos del *pipeline*, pero sin tener que calcular valores, almacenar resultados o reproducir fielmente la jerarquía de memoria.

La segunda parte de la figura 1, llamada *Back End* está generada aleatoriamente a partir de la traza de ejecución sintética, y se asume que converge a la traza esperada del procesador simulado cuando la desviación estándar de las instrucciones producidas es menor del 1 % para una secuencia de 2.000 ciclos de ejemplo. Normalmente esta convergencia ocurre después de unas pocas decenas de miles de ciclos de reloj en simulación.

3. Aplicaciones

El modelo de simulación estadística es simple, y después de la creación de la traza característica inicial, la simulación puede ser mejorada eficazmente (en órdenes de magnitud, literalmente) de una forma más rápida de lo que se haría con la simulación convencional con el benchmark completo. Asumiendo, por el momento, que los resultados son razonablemente precisos, hay un gran número de posibles aplicaciones a esta técnica. Entre las más importantes tenemos:

a. Evaluación del procesador

Esta es, quizá, la aplicación más obvia de esta técnica porque es la aplicación más común para la simulación detallada convencional. Aunque la simulación estadística no es probablemente lo suficientemente eficaz como para sustituir a la simulación detallada a la hora de tomar decisiones importantes en el diseño del procesador, puede resultar útil como herramienta complementaria para una evaluación rápida del espacio de diseño. Debido al modelo de procesador simple, las modificaciones en la microarquitectura son relativamente sencillas con respecto a las modificaciones que ofrecería una simulación convencional. Los nuevos benchmarks pueden ser creados fácilmente para atender a varios aspectos del procesador que tal vez no estuvieran comprendidos en el conjunto de los benchmarks existentes. Un espacio de diseño puede ser cubierto rápidamente por repetidas ejecuciones de conjuntos de benchmarks pre-parametrizados sobre el conjunto de variaciones de la microarquitectura, o un espacio de programas de aplicación se puede cubrir por completo variando los parámetros que conducen al generador de benchmarks sintéticos.

b. Evaluación del sistema

Esta aplicación puede ser incluso mejor que la evaluación en un monoprocesador. Los grandes sistemas contienen muchos procesadores, grandes sistemas de memoria, y ejecutan aplicaciones tan grandes que simplemente no pueden ser simuladas en escalas de tiempo real. Para el diseño del sistema la microarquitectura del procesador está normalmente fijada y es la estructura del sistema la que se quiere evaluar. Es muy fácil simular sistemas con los números de procesadores y cantidad de memoria requeridos. Además, ajustando los parámetros del área de trabajo basados en los pronósticos de las características de la aplicación, es posible extrapolar y predecir la mejora del sistema para aplicaciones futuras.

c. Caracterización del programa

En el proceso de generación de las versiones sintéticas de los benchmarks originales y validar la eficacia de la simulación estadística, se aislan, efectivamente, las características del programa que afectan al funcionamiento de aquellas que no lo hacen. Y estos parámetros clave del

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

programa nos pueden proporcionar una forma de caracterizar de forma concisa a los benchmarks. Estos parámetros se pueden usar para extrapolar el rendimiento de un programa de aplicación a partir de un benchmark de características similares, o para probar la cobertura de un conjunto de benchmarks.

d. Diseño físico

Un modelo de simulación estadística rápido puede ser incorporado a un conjunto de herramientas de diseño sofisticadas. Por ejemplo, dado un plano, los retardos estimados pueden ser suministrados a un modelo de rendimiento de un procesador y el rendimiento puede ser evaluado rápidamente a través de la simulación estadística. Un generador de planos automático puede incorporar un modelo de funcionamiento tal que busque sistemáticamente planos de funcionamiento óptimos.

e. Modelado de alto nivel de energía

Otra característica física de importancia creciente es el consumo de energía. La simulación estadística se puede incorporar a una herramienta de estimación energética para evaluar rápidamente el consumo medio de energía de un programa. Puede resultar particularmente útil para aquellos sistemas que tienen encajados un pequeño conjunto de programas que son los que normalmente usan el resto de las aplicaciones.

4. Bibliografía

[4] Sébastien Nussbaum, James E. Smith, *Modeling Superscalar Processors via Statistical Simulation*. Department of Electrical and Computer Engineering,University of Wisconsin.

PREDICTORES IMPLEMENTADOS EN SIMPLESCALAR-3.0

1. Motivación

A la hora de evaluar unas implementaciones, se hace necesaria una base o punto de referencia, a partir del cual medir cuán buenos son los resultados obtenidos en nuestra configuración. En nuestro caso, la herramienta con la que hemos estado trabajando (Simplescalar v3.0) nos proporciona dicha base, ya que en el módulo sim-bpred, están implementados varios predictores de saltos, cuyo funcionamiento está contrastado, y cuyos resultados podemos utilizar como origen de nuestra "vara de medir".

El propósito de esta sección es el de explicar los fundamentos en los que están basados los predictores implementados en la herramienta, y que hemos utilizado como referencia. Estos predictores son los siguientes: bimod (o predictor bimodal), 2lev (o predictor basado en 2 niveles) y por último comb (predictor basado en la combinación de varios predictores).

2. El predictor bimodal (bimod)

Descripción

Un predictor de saltos bimodal, posee una tabla de contadores saturados de dos bits, a los que se accederá utilizando los bits menos significativos de la dirección de salto como índice. Al contrario que las caches de instrucciones, las entradas del predictor bimodal no suelen tener tags o etiquetas, de modo que un contador en particular, puede corresponder a dos saltos distintos, en cuyo caso su comportamiento podría ser menos preciso.

Cada uno de los contadores se encuentra en uno de los siguientes cuatro estados:

- **Fuertemente no tomado**
- **Débilmente no tomado**
- **Débilmente tomado**
- **Fuertemente tomado**

Cuando un salto es evaluado, el contador correspondiente es actualizado. Los saltos evaluados a falso, es decir, aquellos que no son tomados, decrementan el estado (es decir, cambian el estado del contador en dirección al estado correspondiente a fuertemente no tomado), mientras que aquellos que son evaluados a cierto, y por tanto tomados, incrementan el estado (en direccióna fuertemente tomado).

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

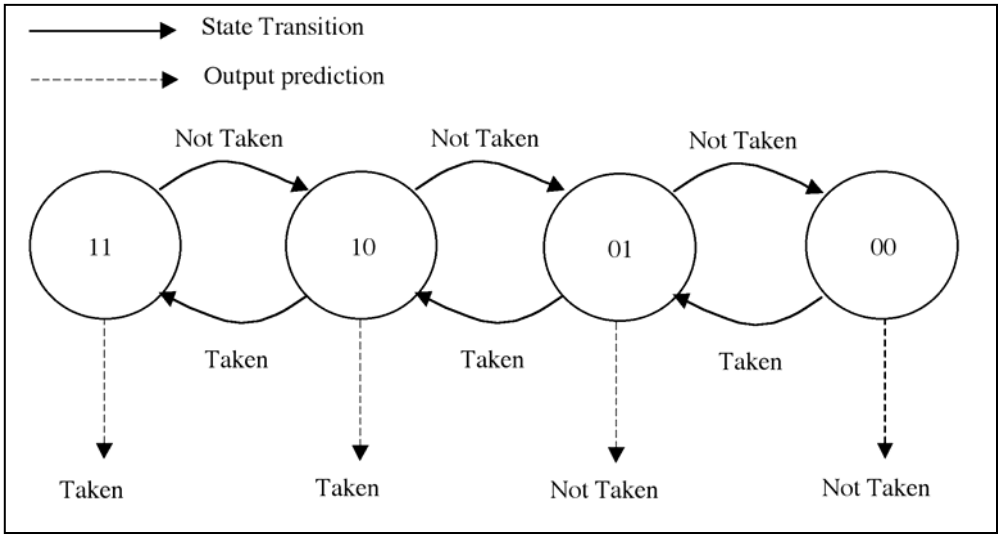


Figura1: Estados de un predictor de 2 bits

El beneficio obtenido de este esquema de contadores saturados de dos bits viene dado por que los saltos que evalúan el final de los bucles siempre se predicen como tomados. Un esquema de un contador de un solo bit falla siempre al comienzo y al final de la ejecución del bucle, esto es, falla en el primer y último salto del mismo. El esquema de dos bits unicamente falla en el último salto. De el mismo modo, en aquellos saltos que solo ocasionalmente tienen una dirección fijada, un esquema basado en un solo bit falla dos veces para cada salto "extraño" (que se sale de lo habitual), mientras que un esquema basado en dos bits, solamente fallaría en uno.

Pruebas realizadas sobre el conjunto de tests SPEC'89, se alcanza un 93.5% de aciertos para un predictor bimodal (siempre que cada salto sea mapeado en un contador único para él).

El problema del aliasing

Cuando los saltos son mapeados dentro de una tabla de predictores, varios saltos distintos pueden coincidir en el mismo predictor. Si el comportamiento de los saltos es similar, las predicciones se verán reforzadas constructivamente. Los comportamientos distintos dan como resultado lo que se ha llamado aliasing destructivo, es decir, equivocan al contador (al menos si pensamos en el otro salto que corresponde a la tabla), produciendose de esta manera un mayor número de fallos.

Implementación en Simplescalar

La herramienta construye este predictor creando una sola tabla de contadores, a los que se accede indexando a través de la dirección del salto, tal y como hemos explicado en los párrafos anteriores.

Para pedirle que implemente este predictor en concreto, en la línea de comando añadiremos lo siguiente:

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
# bimodal predictor config (<table size>)
-bpred:bimod             2048
```

Indicando de ese modo el tamaño de la tabla de contadores de dos bits que queremos que se construya (a mayo tabla, menos aliasing, pero mayor coste de la implementación)

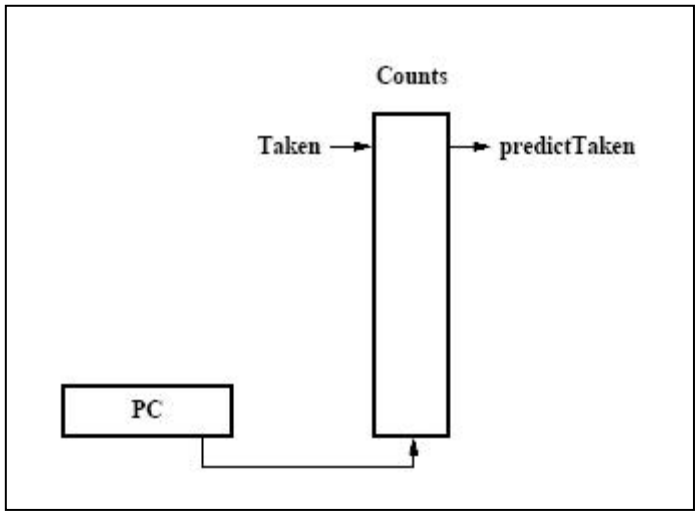


Figura2: Estructura del predictor bimodal

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

3. El predictor de dos niveles (2lev)

Descripción

El predictor de dos niveles utiliza, como su nombre indica, dos niveles de historia para llevar a cabo las predicciones. El primer nivel es la historia de los últimos k saltos encontrados en el flujo del programa. El patrón de historia de segundo nivel es el comportamiento del salto para las últimas s apariciones del mismo, dentro de los k saltos anteriores.

Tanto el primer como el segundo nivel de historia son creados en tiempo de ejecución.

Existen varios tipos de configuraciones de estos niveles de historia, unos utilizan un primer nivel de historia global para todos los saltos mientras que otros utilizan un primer nivel particular para cada conjunto de saltos. Lo mismo ocurre con el segundo nivel de historia, mientras que unos utilizan una tabla para todos los saltos, los hay que utilizan una tabla particular para cada conjunto de saltos.

En nuestro caso nos basamos en un esquema Gag (Global-global)

El esquema Gag

En un predictor de dos niveles Gag, existe un único registro de historia global y una única PHT (Pattern History Table) para todos los saltos. Todas las predicciones de saltos están basadas en el mismo registro de historia y en la misma tabla de historia global, que son actualizadas después de que cada salto sea resuelto.

Dado que las salidas de distintos saltos actualizan el mismo registro de historia y la misma PHT, la información contenida por ambas, está generada por los resultados de diferentes saltos. La predicción obtenida para un salto condicional en este tipo de esquemas, realmente depende de los resultados de otros saltos (aquellos que van antes que este en el flujo de ejecución).

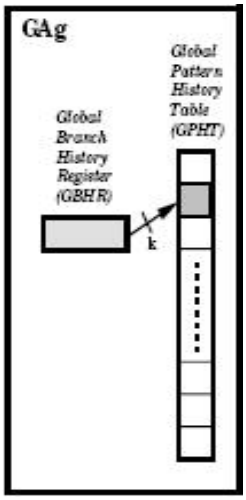


Figura3: El esquema de dos niveles de historia Gag

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

Implementación en Simplescalar

Simplescalar nos permite implementar cualquiera de las 4 posibles configuraciones de un predictor de dos niveles:

```
BPred2Level:  two level adaptive branch predictor
*
*             It can simulate many prediction mechanisms that have up to
*             two levels of tables. Parameters are:
*                  N   # entries in first level (# of shift register(s))
*                  W   width of shift register(s)
*                  M   # entries in 2nd level (# of counters, or other FSM)
*             One BTB entry per level-2 counter.
*
*             Configurations:   N, W, M
*
*                 counter based: 1, 0, M
*
*                 GAg          : 1, W, 2^W
*                 GAp          : 1, W, M (M > 2^W)
*                 PAg          : N, W, 2^W
*                 PAp          : N, W, M (M == 2^(N+W))
```

En nuestro caso nos decidimos por implementar, como medida de referencia, un G-Share, con un solo registro para el primer nivel de historia, y una única PHT para el segundo nivel de historia, a la cual se accede indexando a partir de la O-exclusiva bit a bit entre la dirección del salto y el registro de historia global.

Para pedirle que implemente este predictor en concreto, en la línea de comando añadiremos lo siguiente:

```
# 2-level predictor config (<l1size> <l2size> <hist_size> <xor>)
-bpred:2lev            1 1024 8 0
```

Indicando de ese modo que lo que queremos es un solo registro de primer nivel, con una anchura de 10 bits (el comportamiento de los últimos 10 saltos será el que determine la historia) y con una PHT de 1024 entradas.

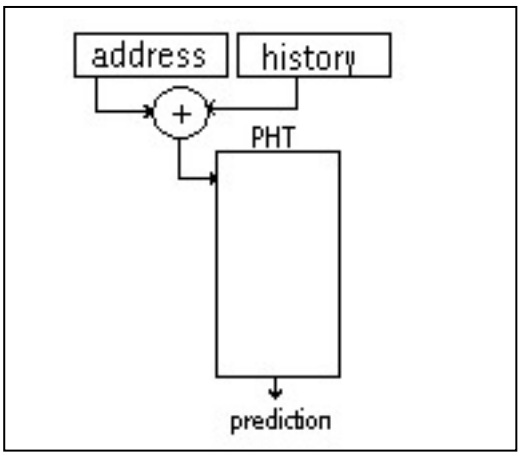


Figura4: El predictor G-Share

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

4. El predictor combinado (comb)

Descripción

Scott Mc Farling propuso la implementación de predictores de saltos combinados en 1993, aseverando que combinándolos se obtienen unos resultados tan precisos como en una predicción local (si para cada salto hubiese un contador concreto) a una velocidad de una predicción global.

La predicción combinada utiliza tres predictores en paralelo, un predictor bimodal (ver sección 2.1) un G-Share (ver sección 3.2) y un predictor basado en un bimodal, que selecciona cuál de los dos anteriores se ha de elegir como base de la predicción salto a salto. El predictor que elige cuál de los otros dos se ha de tomar (llamemoslo choice), es un contador saturado de dos bits, de modo que el bit más significativo del estado de dicho contador, será quién decida qué predicción hay que tomar de las generadas por el bimodal y el G-Share. El contador es actualizado cuando las predicciones de uno y otro, G-Share y bimod, están en desacuerdo, de modo que se trata de favorecer aquel que tenía la predicción correcta.

Funcionamiento

Los esquemas utilizados clásicamente como predictores, poseen diferentes ventajas. Una pregunta obvia sería qué ventajas pueden ser combinadas para obtener un nuevo método de predicción más ajustado.

Un método para obtener dichas ventajas es el mostrado en la figura 5.

El predictor combinado contiene una tabla adicional de contadores saturados de 2 bits. Cada uno de los contadores tiene registrado cuál de los dos predictores es más ajustado para el conjunto concreto de saltos al que dicho contador se refiere. Tomemos la notación P1c y P2c para denotar cuál de los predictores son correctos respectivamente. El contador será incrementado o decrementado siguiendo la siguiente tabla:

P1c	**P2c**	**P1c-P2c**	
0	0	0	**(sin cambios)**
0	1	-1	**(decrementar contador)**
1	0	1	**(incrementar contador)**
1	1	0	**(sin cambios)**

Tabla1: Funcionamiento del predictor combinacional

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

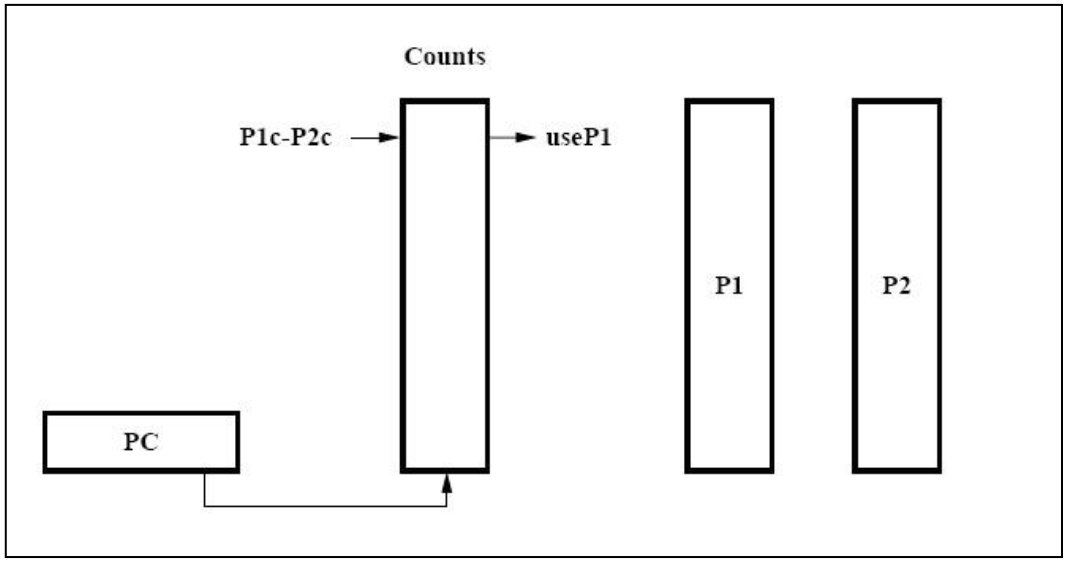


Figura5: El predictor combinado

Implementación en Simplescalar

Para implementar un predictor de este estilo, la llamada que se ha de realizar es la siguiente:

```
# combining predictor config (<meta_table_size>)
-bpred:comb            1024
```

Una vez hecho esto, simplescalar genera tres tablas, la primera de ellas es un predictor bimodal, con una tabla de `meta_table_size` contadores saturados de dos bits, la segunda es un predictor de dos niveles con la configuración por defecto (G-Share) y la tercera es una tabla a la que se llama meta, que es a su vez otro predictor bimodal de `meta_table_size` contadores saturados de dos bits. A la hora de actualizar la tabla, se verá cuál de los dos(o los dos) predictores han acertado en la predicción, y se actualizará tal y como hemos expuesto en la tabla1.

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

Bibliografía

[5] Todd Austin, Dan Ernst, Eric Larson, Chris Weaver University of Michigan, *SimpleScalar Tutorial (for release 4.0).*

[6] *SimpleScalar source. bpred.c, bpred.h, sim-bpred.c.*

[7] Scott McFarling , *CombiningBranch Predictors*, WRL Technical Note TN-36, June 1993.

[8] A. N. Eden and T. Mudge ,*The YAGS Branch Prediction Scheme.*

[9] T.-Y. Yeh and Y. Patt. *A Comparison of Dynamic Branch. Predictors that use Two Level of Branch History.* Proc. 20th Ann. Int. Symp. on Computer Architecture, May 1993.

[10] Tse-Yu Yeh and Yale N. Patt, *Alternative Implementations of Two_Level Adaptive Branch Prediction Pp 124 -134,* The 19th Annual International Symposium on Computer Architecture, May 1992 Gold Coast, Australia.

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

PREDICTOR AGREE

1. Motivación

El predictor agree es un predictor de salto propuesto con el objetivo de reducir el número de interferencias en la PHT (aliasing) debidas a la confluencia de distintos saltos en la misma entrada de la PHT, con repercusiones negativas en las predicciones proporcionadas.

A diferencia de otros predictores que eliminan el aliasing destructivo, este predictor, más que intentar eliminar el aliasing destructivo directamente, propone un método para convertirlo en uno de los otros tipos de aliasing que existen: aliasing neutro y aliasing constructivo, modificando para ello la estructura y uso de la PHT.

2. El predictor AGREE

El predictor presentado a continuación difiere de las técnicas de predicción usadas por los predictores de salto tradicionales de dos niveles, tanto en la estructura como en el significado que le proporciona a los contadores de la PHT.

a. Diferencias estructurales

En un predictor tradicional de dos niveles, cada entrada de la PHT es un contador saturado de dos bits cuyo valor se usa para predecir si el salto correspondiente a esa entrada será tomado (si el contador es 0 o 1) o no tomado (en caso de que sea 2 o 3). Dicho contador se actualizará al resolverse el salto, incrementándose si fue tomado y en caso contrario si no lo fue. De este modo, será más probable que las futuras apariciones de saltos que usen esta misma entrada de la PHT, sean predichas de modo correcto.

Sin embargo, como podemos observar en la figura 1, el predictor agree introduce una modificación en la estructura del tradicional predictor: añade un bit de 'bias' a cada salto en la BTB. El bit de bias también puede encontrarse, en otras implementaciones alternativas, añadido en la caché de instrucciones (nuestra implementación está basada en la figura indicada). Este bit de bias, proporcionará el sentido más probable para ese salto.

En cuanto al retardo que puede producir el accesoal bit de bias, podemos observar que el acceso se realiza en paralelo a la búsqueda del índice, por lo que el retardo adicional será, únicamente, el correspondiente a la operación entre dicho bit y la predicción que proporcione contador correspondiente de la PHT.

Este predictor sólo modifica las estructuras de segundo nivel de predicción, pudiéndose utilizar cualesquiera en las tablas de primer nivel. En nuestra implementación, hemos utilizado la misma tabla de primer nivel que usa el Gshare.

Basándose en la idea de que *"todos los saltos tienden a repetir su comportamiento inicial"* dicho bit de bias, será inicializado con el sentido tomado por la primera ejecución del salto, sentido que ha sido

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

demostrado ser tomado, de media, en un 85% de las apariciones totales del salto. Por este hecho, una vez asignado este primer valor al bit de bias, no volverá a ser modificado.

En este punto, encontramos una segunda mejora con respecto a los predictores de dos niveles: el tiempo en el que el valor del contador alcanza su valor deseado (tiempo en el que el contador "se está entrenando") es mucho menor.

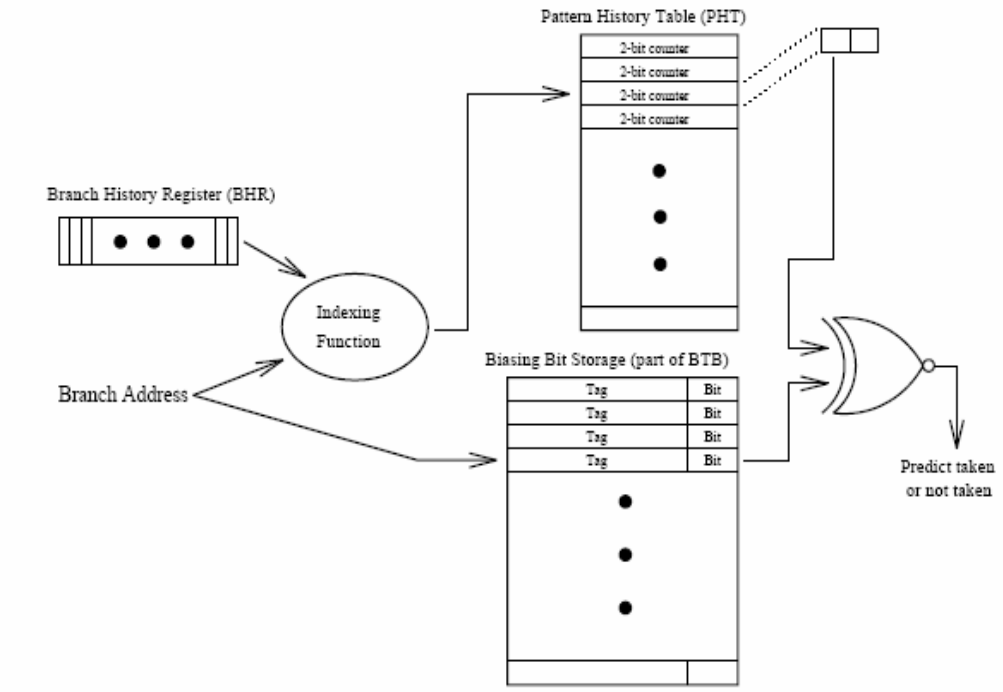


Figura 1: Predictor agree

b. Diferencias semánticas

Además de la variación estructural, introduce un importante cambio en cuanto a la interpretación semántica de los contadores de la PHT, cuyo significado ya no será si el salto es o no tomado, sino si la predicción del salto será la misma que proporcione el bit de bias o la contraria. Por tanto, la predicción de la PHT ahora será si está de acuerdo o no lo está ("*if agree or not*") con la predicción que proporciona el bias.

c. Actualización de los contadores de la PHT

La actualización de los contadores de la PHT se realizará cuando el salto haya sido resuelto (al igual que en los predictores de dos niveles tradicionales), sin embargo, el contador se incrementará si el sentido del salto coincidió con la predicción dada por el bit de bias y se decrementará en caso contrario:

Si la predicción dada por el bit de bias ha sido correcta (si el salto se ha tomado y el bias era 1 o el salto no se ha tomado y el bias era 0) se incrementa el contador correspondiente de la tabla de segundo nivel, lo que significará una mayor confianza en el bit de bias para próximas predicciones, ya que ha acertado en la predicción. En caso de fallo en la predicción dada por el bit de bias, los contadores se decrementan, disminuyendo así la confianza en este bit para la próxima vez que se trate el salto.

3. Comparativa. Análisis de resultados

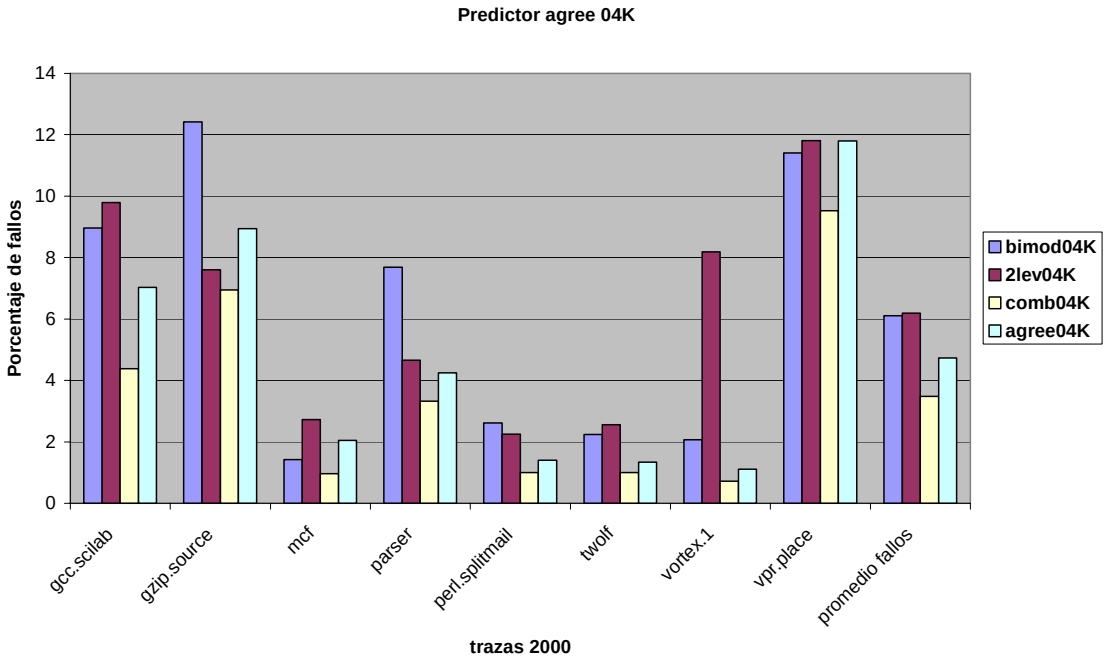
A continuación presentamos una comparación entre los resultados obtenidos entre los predictores que ya se encontraban implementados en la herramienta SimpleScalar (bimod, 2lev y comb) y el predictor agree, con configuraciones de 4 y 16 Kb, para los dos conjuntos distintos de benchmarks utilizados. Únicamente resaltar que las gráficas muestran los porcentajes de fallos en las predicciones de salto.

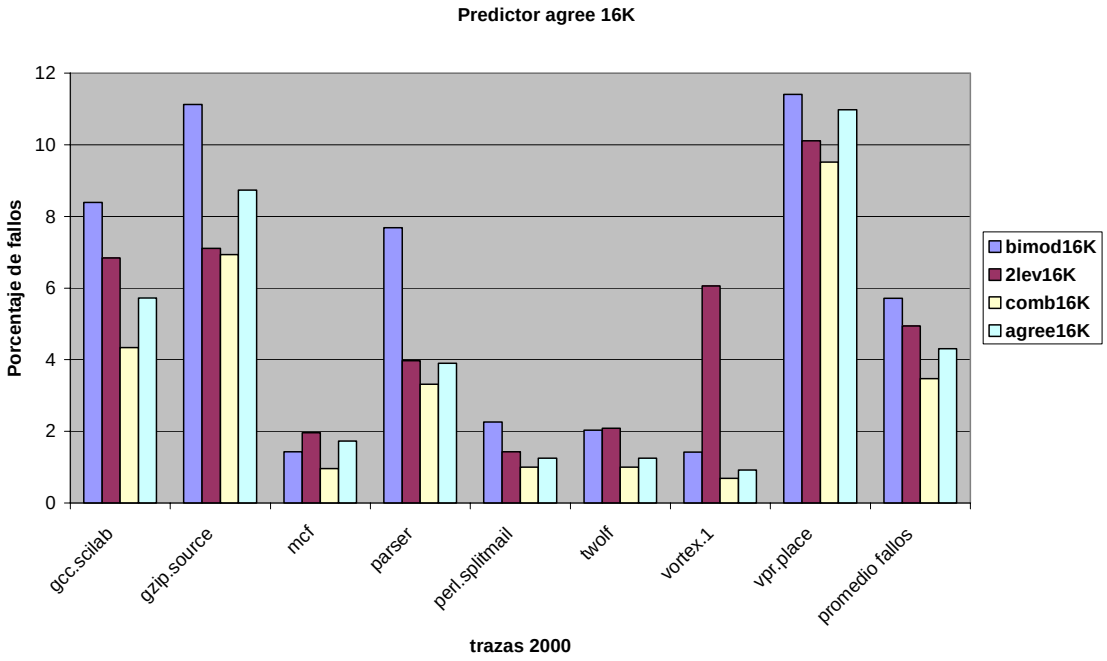
En el estudio de este predictor, lo más significativo es comparar sus resultados frente a los del predictor de dos niveles, ya que es una mejora de éste. Por tanto, observaremos que el porcentaje de fallos para la gran mayoría de los benchmarks es menor en el agree que en dicho predictor, y por tanto, también será menor que el número de fallos obtenido en predicciones realizadas con el predictor de 2-bits (bimod).

Se puede ver que aún este predictor no consigue disminuir la tasa de fallos que logra el predictor combinado, algo que era previsible debido a la implementación del predictor “comb”, que aprovecha los resultados de predicción de dos predictores distintos, aumentando por tanto las posibilidades de acierto de modo muy significativo.

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

Benchmarks "*Trazas 2000*":



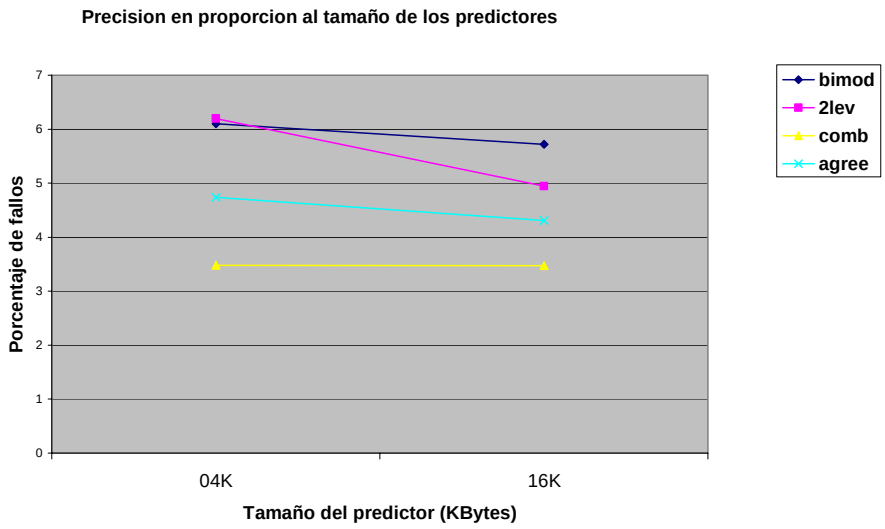


En la gráfica siguiente mostramos la precisión de los predictores (en base a sus porcentajes de fallos como anteriormente) según los tamaños con los que se han implementado: para cada uno de los tres predictores que tomamos inicialmente de referencia, y para nuestro predictor agree, comparamos sus resultados para configuraciones de 4 y 16Kb.

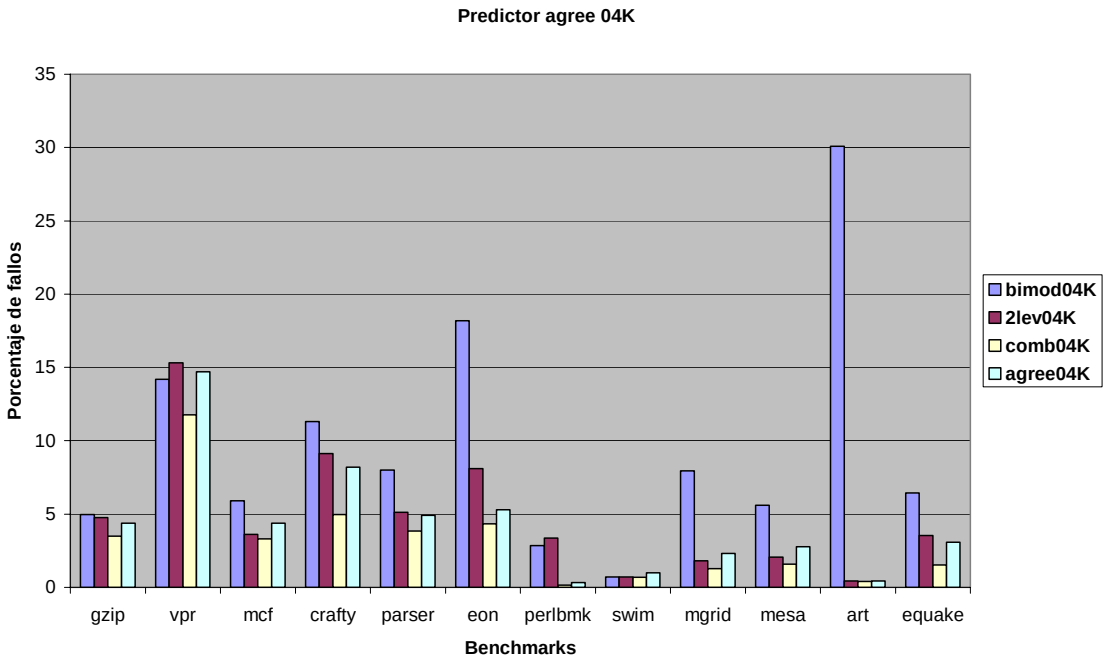
El predictor agree muestra un descenso del porcentaje de fallos al aumentar su tamaño de 4Kb a 16Kb, puesto que este aumento de tamaño

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

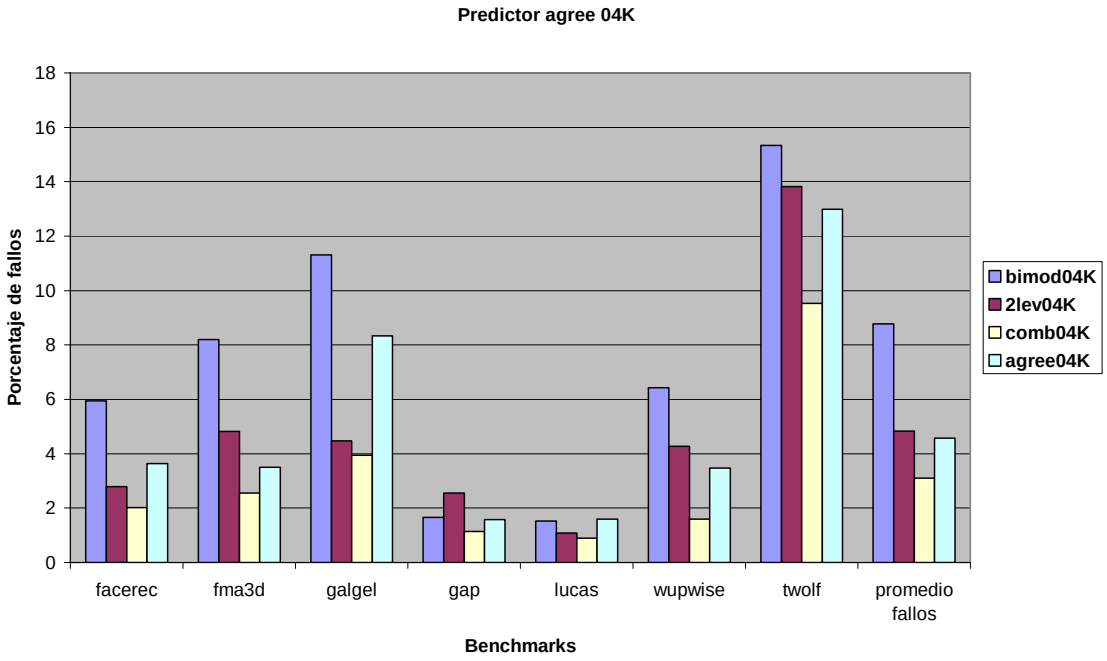
del predictor supone una mayor capacidad para albergar las predicciones de un número mayor de saltos.

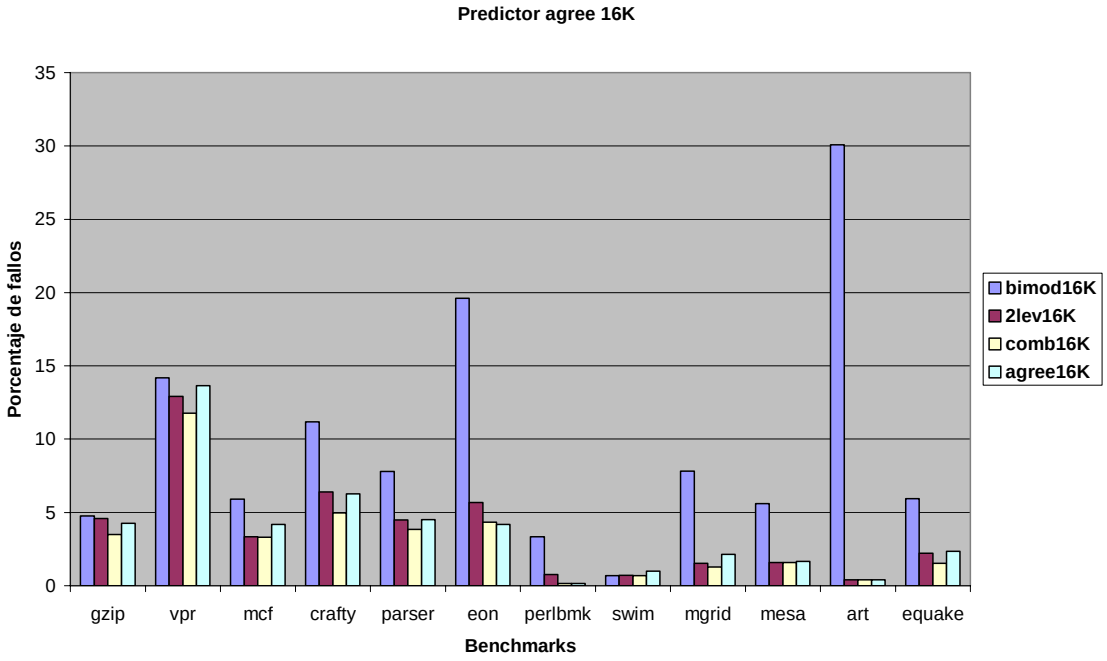


Benchmarks *"Ejecuciones completas":*

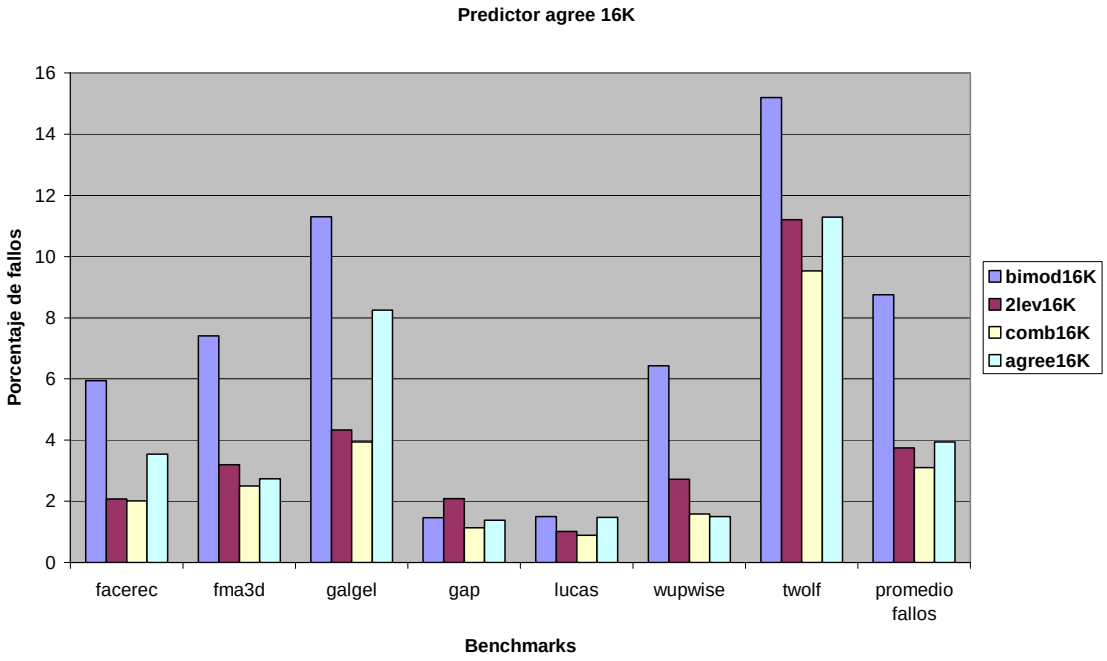


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

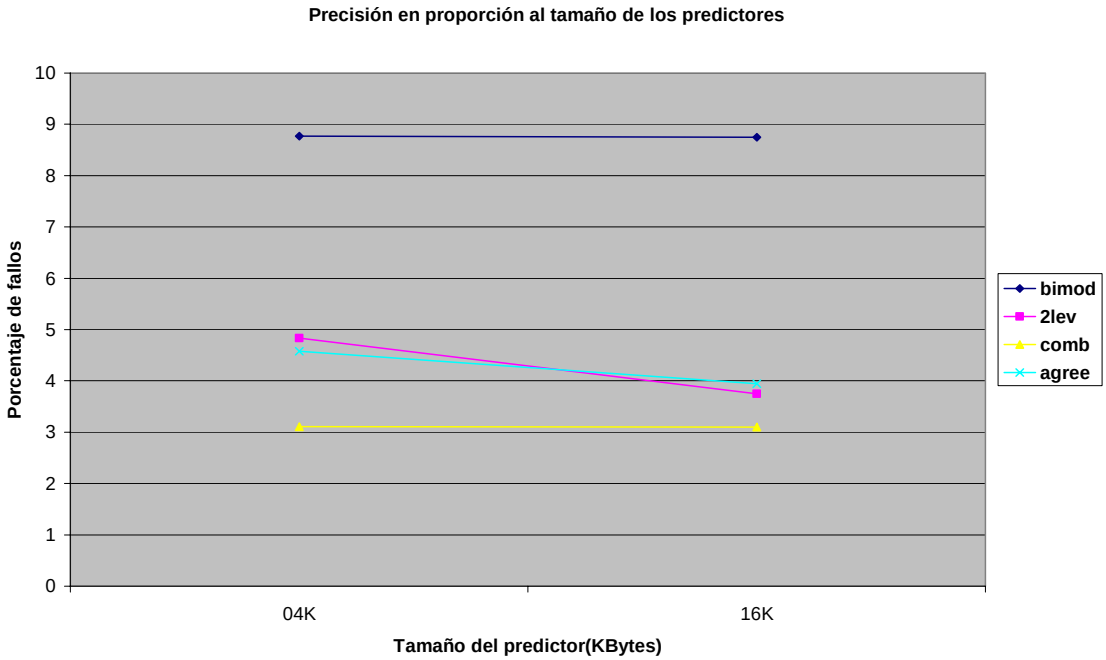




Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas



Para estos tests también mostramos la variación de la eficacia de los predictores en función de su tamaño: 4 y 16Kb. Al ser esto característico de los predictores en sí, y no de los tests con los que se estén probando, vemos que tanto el predictor agree como los otros tres, siguen manteniendo la variación que ya mostraban en la gráfica para el primer conjunto de benchmarks (“*Trazas 2000*”).



Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

4. Conclusión

Este predictor propone una alternativa para reducir el aliasing negativo (interferencias negativas entre los saltos), de modo que si dos saltos distintos utilizasen la misma entrada en la PHT, será muy probable que ambos incrementen el contador en el mismo sentido (hacia el estado '*agree*'). Este cambio cambio en la interpretación de la PHT juega un papel fundamental en la conversión de dichas interferencias negativas en otras neutras o positivas, aumentando de este modo, la tasa de aciertos del predictor.

5. Bibliografía

[11] "The Agree Predictor: A Mechanism for Reducing Negative Branch History Interference"
Eric Spranglezy Robert S. Chappellyz Mitch Alsupz Yale N. Patty

[12] "Advanced Computer Architecture Laboratory"
Department of Electrical Engineering and Computer Science
The University of Michigan

[13] Presentación en power point:
www.ece.rochester.edu/
www.ece.rochester.edu/~mihuang/TEACHING/OLD/
ECE404_SPRING03/Branch%20Prediction.ppt

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

PREDICTOR BI-MODE

1. Motivación

La capacidad para minimizar el número de paradas o de burbujas en el pipeline que pueden provocar las instrucciones de salto comienza a volverse crítica en los diseños de hardware que implementan grandes cantidades de paralelismo a nivel de instrucción. Hay varias técnicas para reducir la penalización por salto, como la predicción estática y dinámica de saltos o la ejecución especulativa. Entre estas técnicas, la predicción dinámica de saltos es la más popular porque proporciona buenos resultados que se puede implementar sin realizar cambios en el repertorio de instrucciones o en los programas realizados previamente para estas máquinas.

El poder de la predicción dinámica de saltos es tal que se puede deducir el comportamiento de los saltos, de forma muy cercana a su conducta real, en tiempo de ejecución, proporcionando un grado de adaptación que otras técnicas están desaprovechando. Esta adaptación es especialmente crítica cuando la conducta de los saltos se puede ver afectada por los datos de entrada de las distintas ejecuciones de los programas. Con la introducción de los esquemas de dos niveles, la eficacia de los predictores dinámicos de saltos ha sido elevada hasta el 90% aproximadamente. La consecuencia de esto es que muchos de los microprocesadores más actuales incorporen predictores dinámicos de saltos, como por ejemplo el Pentium Pro y el Alpha 21264.

Entre los predictores de dos niveles que usan esquemas de historia global se han observado los mejores resultados cuando se han probado con benchmarks enteros. Sin embargo, para alcanzar altos niveles de acierto, los actuales predictores de saltos dinámicos requieren considerables cantidades de hardware porque su principal punto débil es el problema del aliasing destructivo (que se resuelve mejor aumentando el tamaño del predictor). El predictor de saltos bi-mode es una nueva técnica lo suficientemente económica y sencilla para evitar los problemas descritos anteriormente. Además, se ha demostrado que en los benchmarks IBS y SPEC CINT95 el predictor bi-mode, por el mismo precio, tiene, en media, resultados mejores que el predictor gshare, que es uno de los mejores predictores dotados de historia global.

2. Funcionamiento del predictor

Como se ha visto anteriormente, la principal motivación del predictor bi-mode es la eliminación del aliasing destructivo presentes en los esquemas de indexación por historia global. Como se puede ver en la figura 1, el esquema divide la tabla de contadores de dos bits en dos mitades. Dado un patrón de historia, dos contadores, uno de cada mitad, son seleccionados. Estos contadores serán denominados predictores de dirección. Mientras tanto, otra tabla de contadores de dos bits, indexada sólo por la dirección del salto, es usada para seleccionar cuál de los dos

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

predictores de dirección será elegido como predicción final para el salto. La predicción final será hecha teniendo en cuenta el bit más significativo del contador elegido. Un aspecto muy importante es que sólo se actualizará con el resultado final del salto el contador elegido para realizar la predicción, mientras que el otro contador permanecerá inalterado.

Por otro lado, el predictor usado para elegir entre los dos contadores será siempre actualizado con el resultado final del salto, excepto cuando la elección sea contraria al resultado final del salto y la dirección de destino dada sea una dirección correcta. Esta política de actualización parcial es muy efectiva para presupuestos pequeños de hardware.

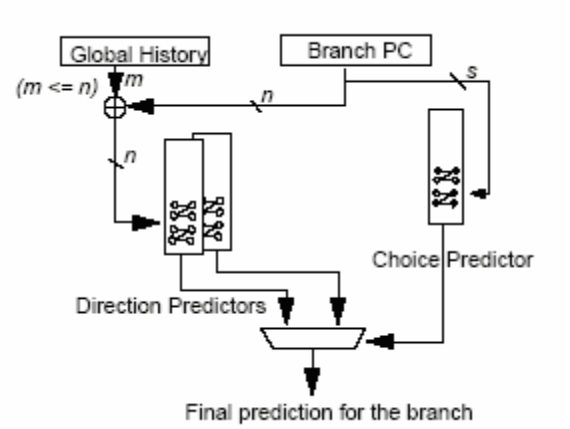


Figura 1: Esquema del modelo de predicción de saltos propuesto.

3. Implementación

En nuestra implementación para el emulador Simplescalar 3.0 hemos hecho, en primer lugar una nueva estructura de predictor (basada en la estructura del predictor de dos niveles genérico) que contiene las 3 nuevas tablas en sustitución de la tabla de segundo nivel que poseía el predictor de dos niveles.

Posteriormente hemos inicializado correctamente las estructuras de memoria correspondientes a estas nuevas tablas, en las funciones habilitadas para ello (bpred_create y bpred_dir_create). El resto de inicializaciones se hace de igual manera a como se hacía en el predictor de dos niveles.

Lo último que quedaba por modificar para implementar este predictor eran los procedimientos de búsqueda y actualización. Estos procedimientos (bpred_lookup, bpred_dir_lookup y bpred_update) fueron modificados de acuerdo a las especificaciones del predictor dichas más arriba. Las nuevas tablas están hechas del mismo modo que la tabla de segundo nivel del predictor de dos niveles básico, y por tanto lo único que contiene son contadores de dos bits, que se usarán para dar las distintas predicciones. El modo de acceso a las tablas es el mismo que se ha comentado anteriormente.

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

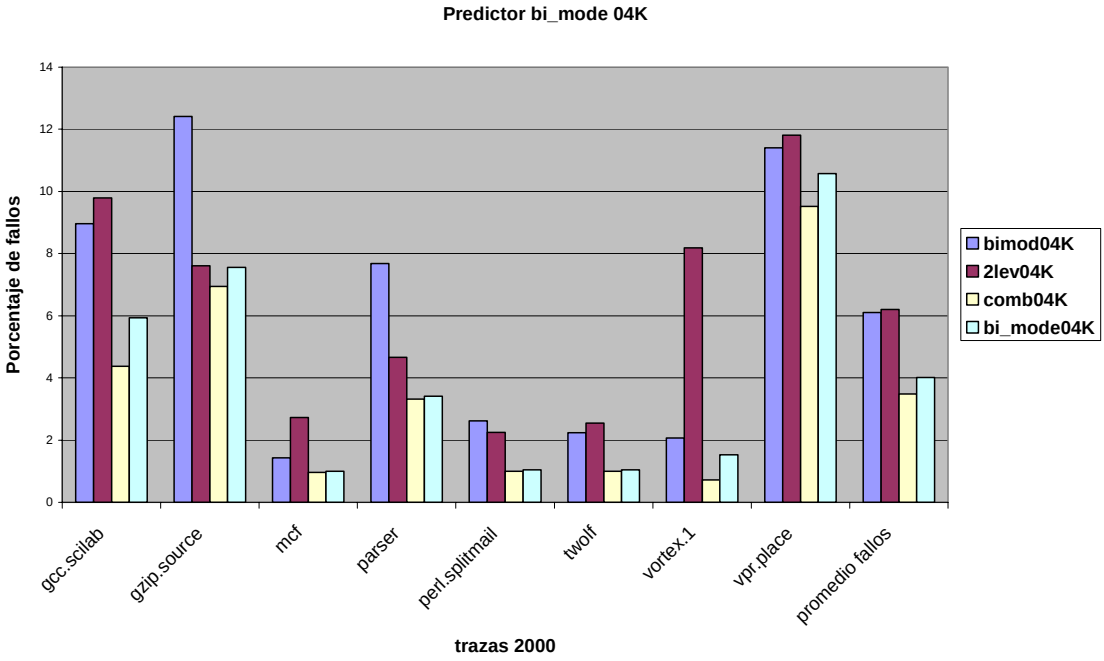
4. Resultados

En este predictor también mostramos para los dos conjuntos de benchmarks estudiados, unas gráficas que permiten ver la mejora que supone este predictor con respecto a los predictores que hemos considerado siempre como iniciales.

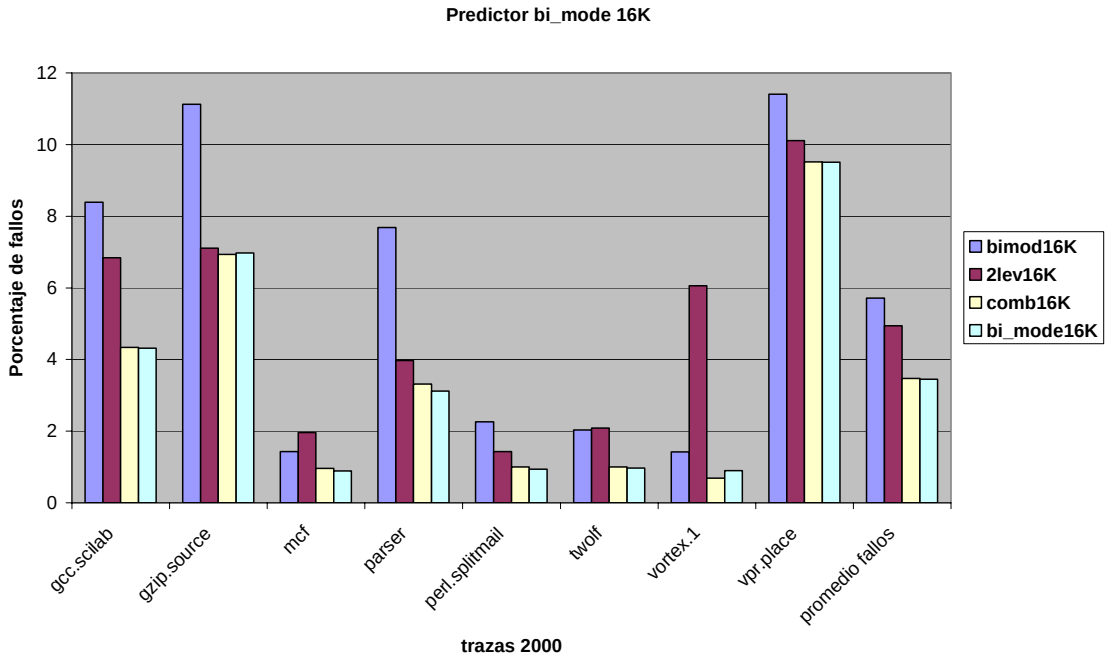
A simple vista se observa una disminución significativa de la tasa de fallos en el predictor bi-mode con respecto al predictor bimodal y al de dos niveles. Esta disminución es vertiginosa en el caso del benchmark *"vortex.1"* en el conjunto primero *"Trazas 2000"*.

Con relación al primer predictor implementado por nosotros, el predictor agree, recordemos que sus resultados no eran tan excelentes como los del predictor combinado. Ahora es de relevante importancia hacer notar que los resultados obtenidos por el predictor bi-mode se asemejan mucho más a los del predictor "comb", siendo en muchos casos iguales de buenos (caso de los benchmarks *"mcf"*, *"twolf"* en el conjunto primero, o caso de los benchmarks *"swim"*, *"mgrid"*, o *"art"* en el conjunto segundo de benchmarks).

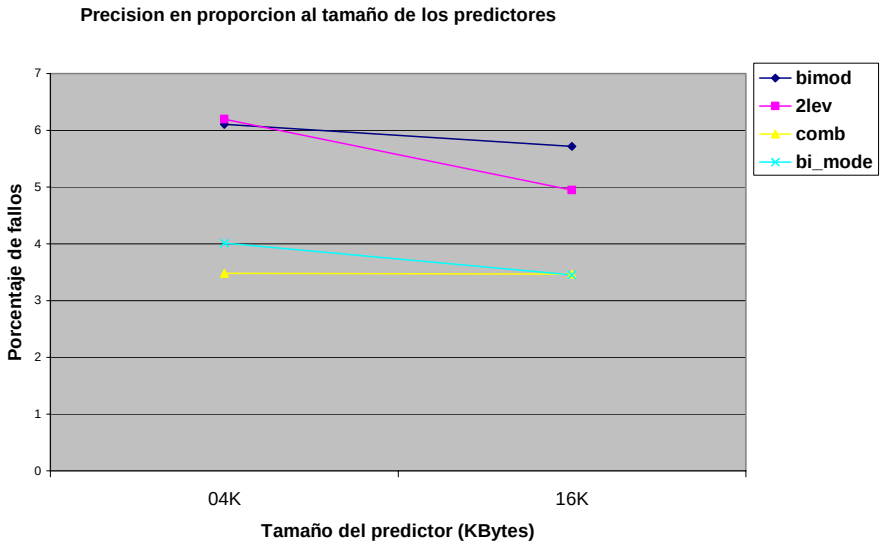
Benchmarks *"Trazas 2000":*



Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

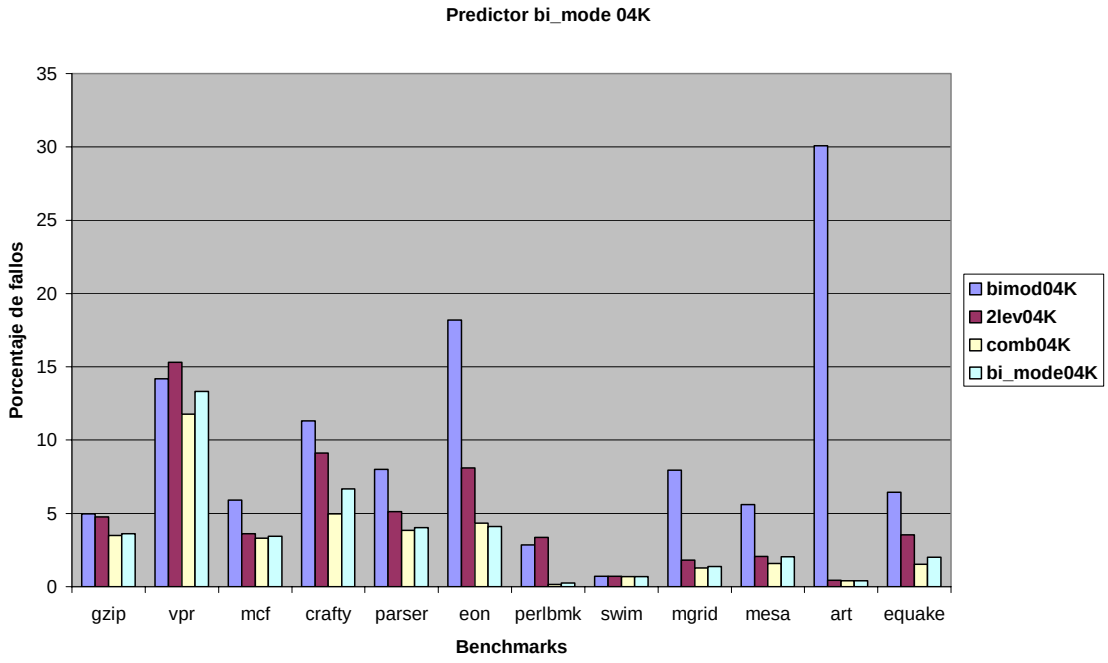


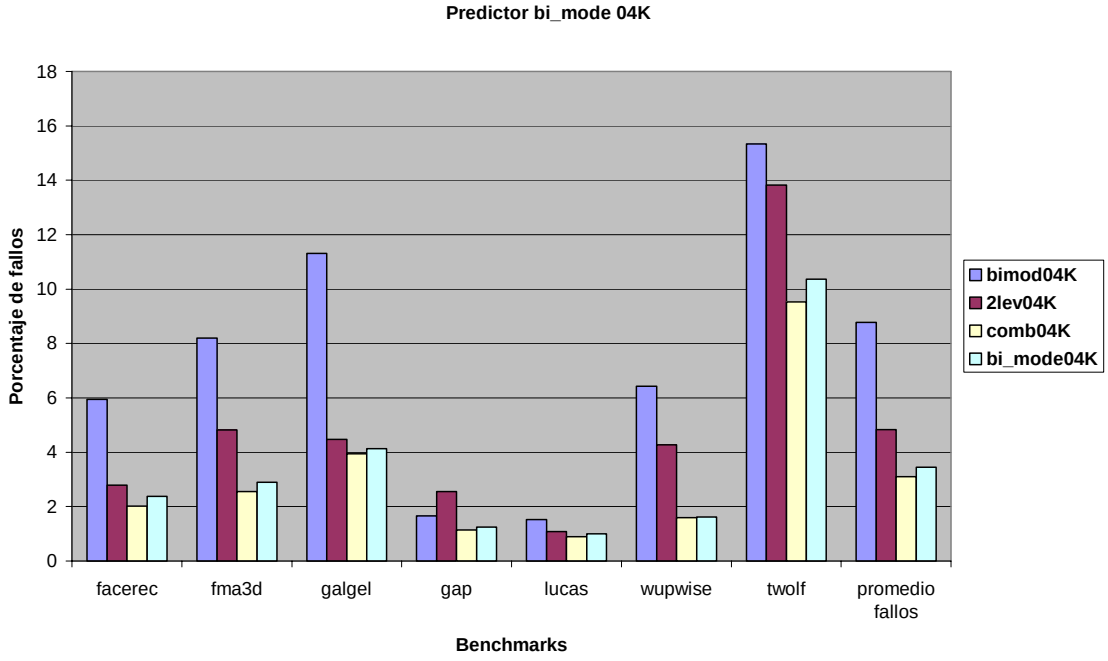
En la gráfica que compara los resultados en base al tamaño de los predictores, podemos observar que este predictor consigue una mejora notable en su comportamiento al aumentar su tamaño. Vemos cómo la línea que une sus porcentajes de fallos, se inclina hacia el tamaño mayor, implicando, por tanto, un descenso del porcentaje de fallos en el predictor bi-mode de 16Kb con respecto al mismo de 4Kb.



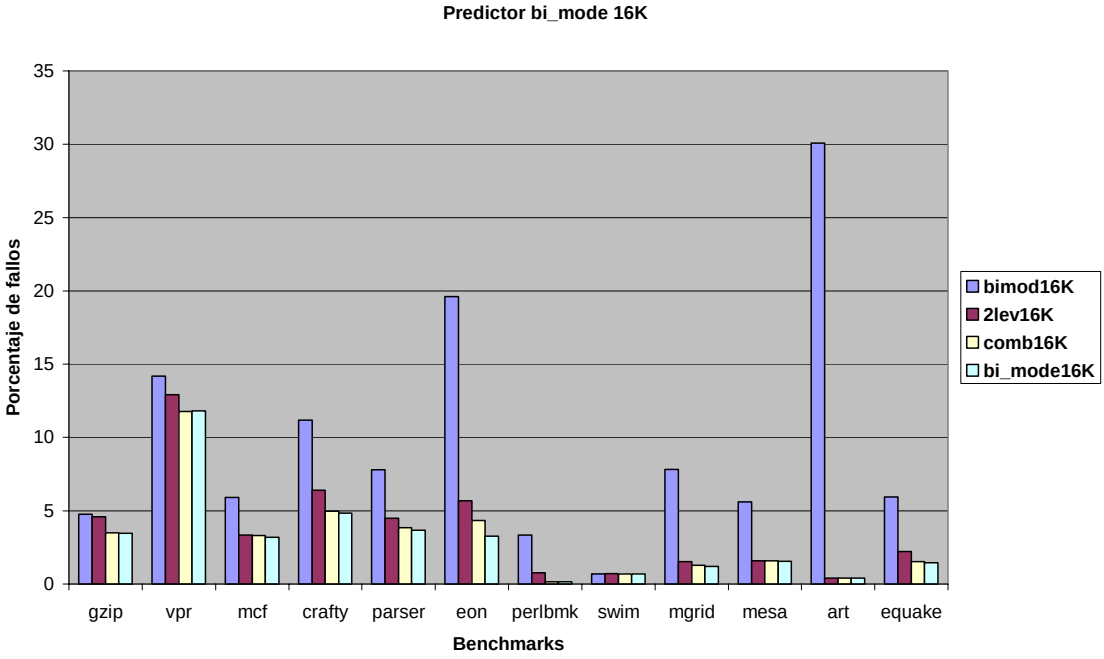
Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

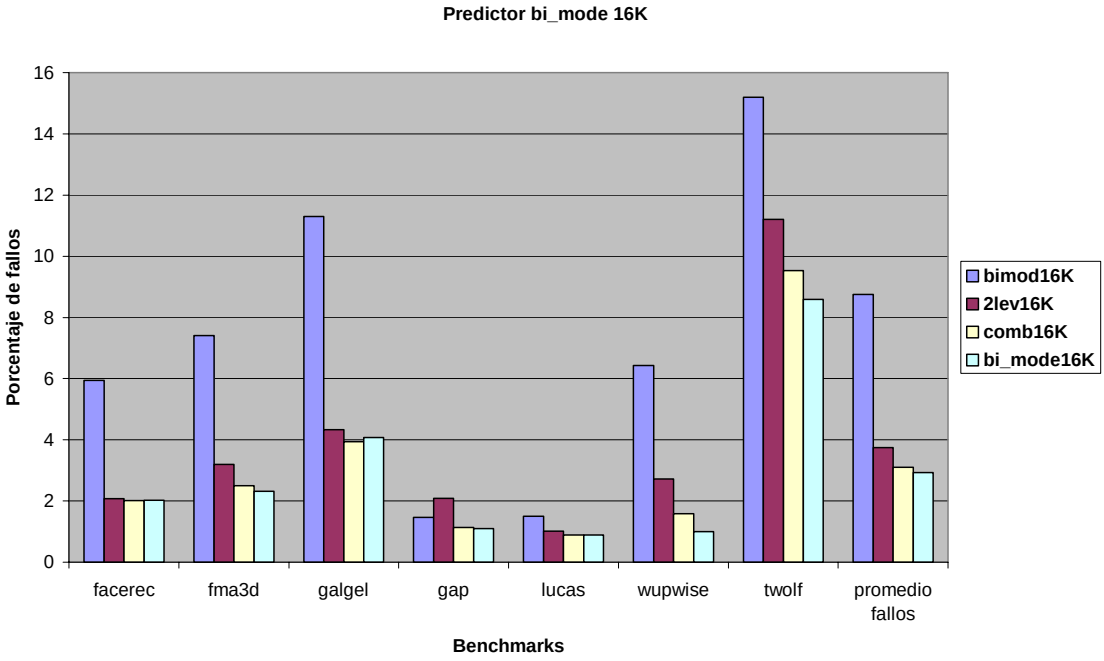
Benchmarks *"Ejecuciones completas":*





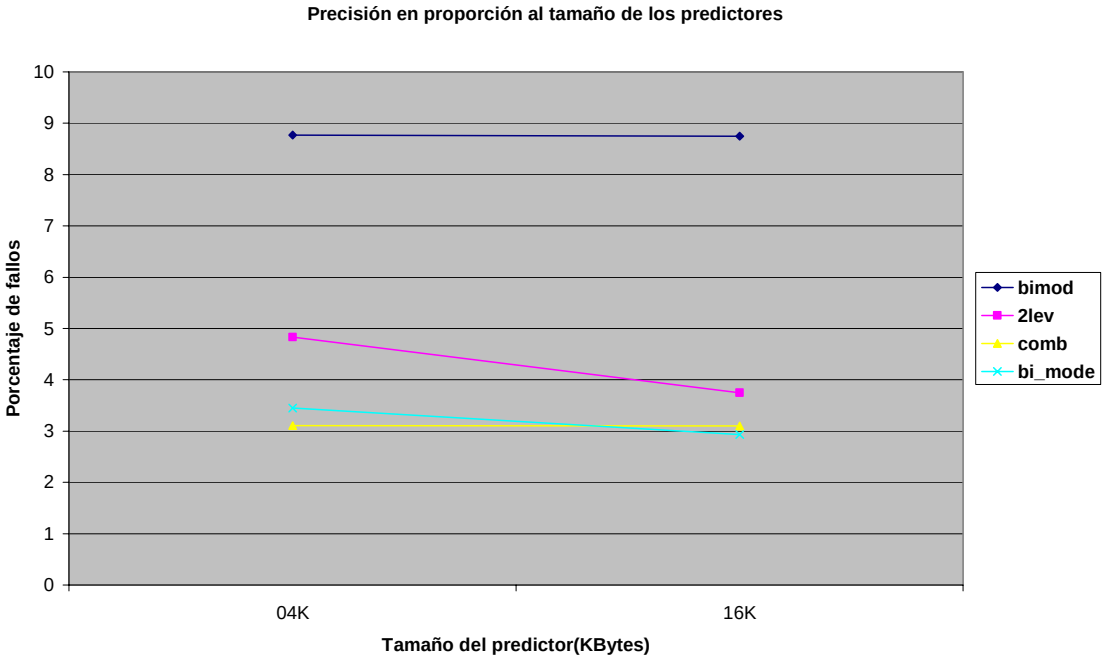
Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas





Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

En la siguiente gráfica, vemos que se sigue manteniendo el descenso de la tasa de fallos del predictor de 16KB como ocurría para el anterior conjunto de tests.



5. Conclusión

Este predictor ha sido diseñado para mejorar las predicciones por medio de la eliminación del aliasing que tiene lugar en los predictores de salto dinámicos. El éxito del predictor recae en la determinación de forma dinámica de la dirección (tomado o no tomado) por medio de un predictor elector (choice predictor en el original) simple pero exacto. Esta clasificación puede ayudar a eliminar mucho del aliasing destructivo, a la vez que se mantiene el aliasing neutro común a las dos tablas de contadores.

6. Bibliografía

[14] "**The Bi-Mode Branch Predictor".**
Chih-Chieh Lee, I-Cheng K. Chen, and Trevor N. Mudge.
EECS Department, University of Michigan.

PREDICTOR SKEW

1. Motivación

El empleo por parte de los microprocesadores modernos de pipelines cada vez más profundos y lanzamiento de múltiples instrucciones por cada ciclo de reloj del procesador, está aumentando la importancia de una predicción de saltos precisa.

El uso de unos recursos de hardware limitados implica que no sea posible mantener toda la historia de saltos para todos los saltos que se encuentran activos al mismo tiempo, de modo que el problema resultante, conocido como aliasing, es similar al que ocurre con las caches de tamaño finito, esto es, no se puede direccionar únicamente un salto; varios saltos pueden tener iguales aquellos bits que se usan para direccionarlos en la caché.

Estudios recientes muestran que el nivel de aliasing mostrado por trabajos multiproceso bajo diversos sistemas operativos, es bastante grande, y se hacen necesarios predictores mucho mayores que los que se utilizaban antaño, para alcanzar un nivel de precisión cercano al ideal (aquel en el que no exista aliasing).

2. El predictor SKEW

Idealmente, nos gustaría tener una tabla de predicción con capacidad infinita, de modo que cada par dirección historia relativo a cada salto, tuviese un predictor en exclusiva. Evidentemente, a nivel hardware no se puede implementar algo así.

La alternativa que surge está basada en el modelo de las tres C's de rendimiento de caches, propuesto por Hill. Como en los fallos de caché, el aliasing puede ser clasificado como de capacidad, de conflicto y obligatorio.

De este modo, sabemos que una caché asociativa por conjuntos puede reducir en una buena medida las pérdidas producidas por conflicto, sin embargo, la adaptación de una caché a esa forma de solucionar los problemas, requiere la adicción de tags, con el notable incremento de tamaño de la tabla de predicciones.

La idea del predictor Agree es la de tomar el modelo utilizado para las skew-caches asociativas.

El predictor de saltos skew se construye a partir de un número impar de bancos de predicción, cada uno de los cuales funciona con una tabla sin ningún tipo de tag o etiqueta.

Cuando se está realizando la predicción, se accede a cada una de las tablas en paralelo, pero con una función de indexación diferente, y el voto mayoritario será el que decida cuál es el resultado de la predicción.

El modo de indexar el acceso a los bancos se hace a través de funciones hash. La idea de usar dichas funciones es que dos posibles instrucciones de salto con una dirección cuyos primeros bits fuesen iguales (de modo que al indexar directamente o vía módulo el registro accedido fuese el mismo) se dispersen. Así, a la hora de acceder para leer

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

los datos, el valor de los registros correctos se corresponderá en mayor medida con el valor esperado para el salto.

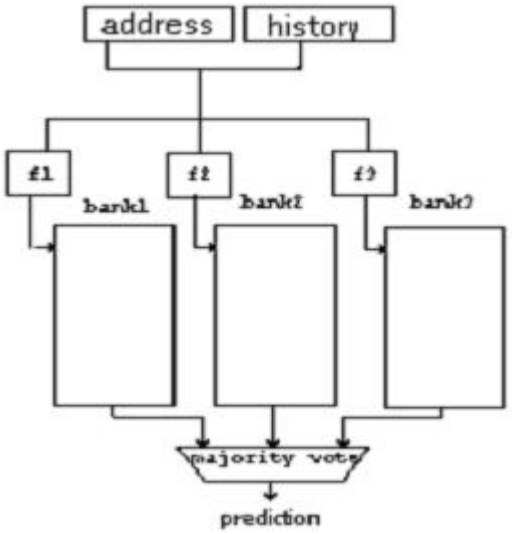


Figura 1: Esquema del predictor de saltos Skew.

3. Implementación

En nuestra implementación basada en el Simplescalar 3.0 hemos hecho, en primer lugar una nueva estructura de predictor dentro de *bpred.h* en el que se incluyen 3 bancos en lugar de la tabla PHT que se utilizaba como contador saturado para las distintas configuraciones del *sim-bpred*.

Posteriormente hemos inicializado correctamente las estructuras de memoria correspondientes a estas nuevas tablas, en las funciones habilitadas para ello (bpred_create y bpred_dir_create). El resto de inicializaciones se hace de igual manera a como se hacía en el predictor de dos niveles. Cada uno de los tres bancos se inicializa de modo que las decisiones sean débilmente tomado y débilmente no tomado.

Una cuestión importante relativa al funcionamiento del predictor Skew es la que tiene que ver con las funciones hash. En un primer momento utilizamos unas funciones hash sencillas, el módulo 2, el producto, y la indexación directa, pero más adelante utilizamos las funciones hash especificadas en el documento de Michaund[1] y que comento en el párrafo siguiente, con las que el resultado obtenido mejoró sensiblemente.

La idea desarrollada por Michaund, consiste en tener un vector de información relevante, al cual se le aplicarán ciertas transformaciones para indexar las tablas. Este vector V será la concatenación de la dirección de salto con los k bits de historia global: V = (aN ... a2 hk ... h1).

Las funciones f1,f2 y f3 usadas para indexar los tres bancos de 2^n entradas son las siguientes (suponiendo que hemos descompuesto el vector V en tres subvectores de n bits (V1,V2,V3)):

Consideremos H como:

$H : \{0.. 2^n -1\} \rightarrow \{0.. 2^n -1\}$

(yn,yn-1,..,y1) → (yn x y1,yn,yn-1,...,y3,y2)

Siendo x la xor nos quedan las siguientes funciones hash:

$f1 : V \rightarrow \{0... 2^n -1\}$

(V3, V2, V1) → H(V1) x H^-1(V2) x V2

$f2 : V \rightarrow \{0... 2^n -1\}$

(V3, V2, V1) → H(V1) x H^-1(V2) x V1

$f3 : V \rightarrow \{0... 2^n -1\}$

(V3, V2, V1) → H^-1(V1) x H(V2) x V2

Lo más interesante de estas funciones es que dos vectores distintos que accedan a la misma posición en uno de los bancos, no provocará conflicto en los otros dos bancos.

4. Comparativa. Análisis de resultados

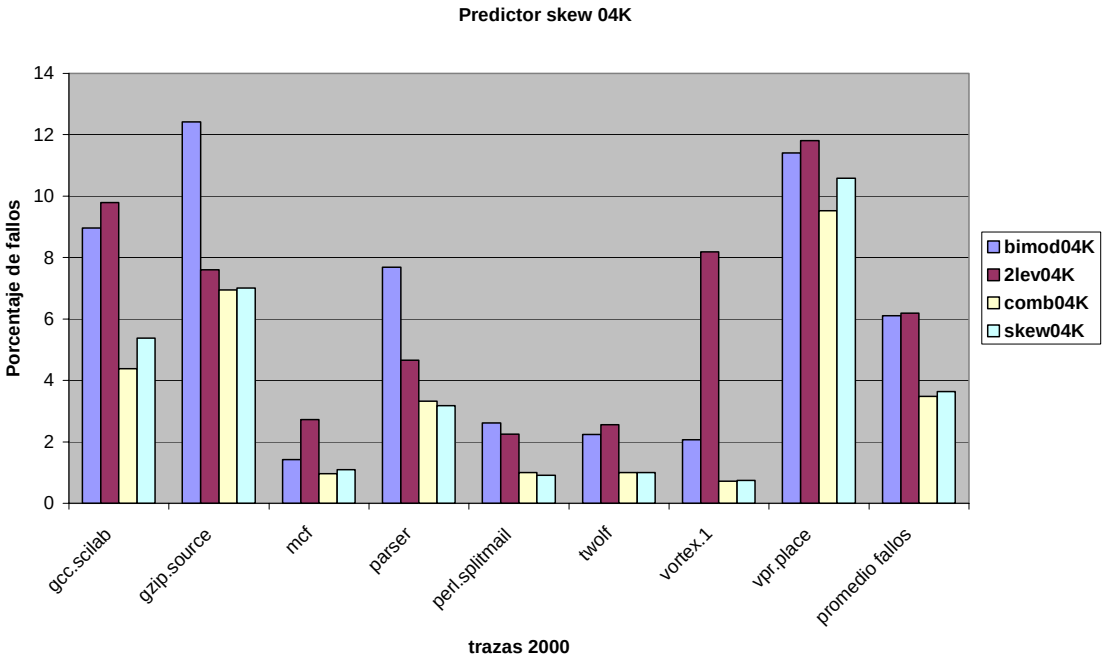
Las gráficas que mostramos a continuación dejan ver cómo el predictor filter, consigue mejorar un poco más los resultados que veníamos obteniendo hasta el momento. Esto ocurre para los dos conjuntos de benchmarks estudiados: el predictor mejora sea cual sea el test sobre el que lo ejecutemos.

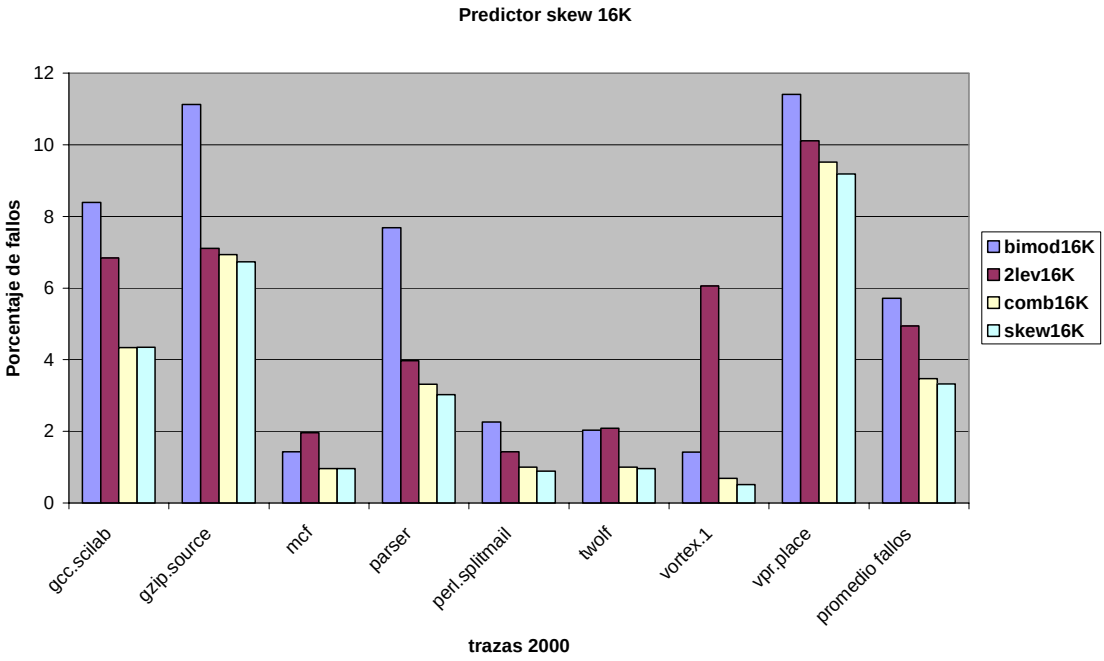
Comparamos este nuevo predictor con el de dos niveles, el bimodal y el combinado, siendo muy superior a los dos primeros y consiguiendo, en mayor número de ocasiones que había logrado el predictor anterior, mejorar al predictor combinado, obteniendo resultados parecidos en el resto de ocasiones.

En cuanto a la mejora al aumentar el tamaño del predictor, en este caso el porcentaje de fallos disminuye pero no de modo tan notable como en los dos anteriores, o en el predictor *"2-lev"* , debido a que ya los resultados no dependen tan directamente de la cantidad de saltos que puedan albergarse, y porque, también, los resultados obtenidos con un tamaño de 4Kb, ya eran bastante buenos.

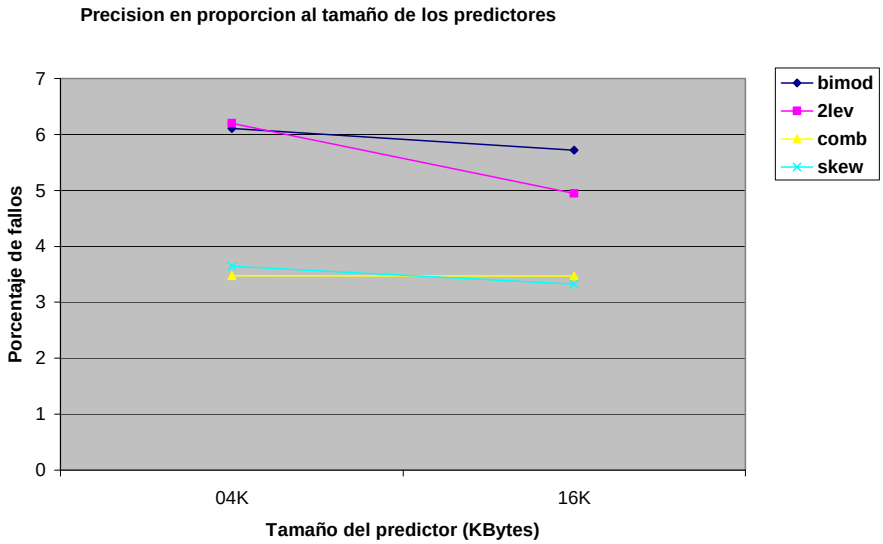
Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

Benchmarks "*Trazas 2000*":

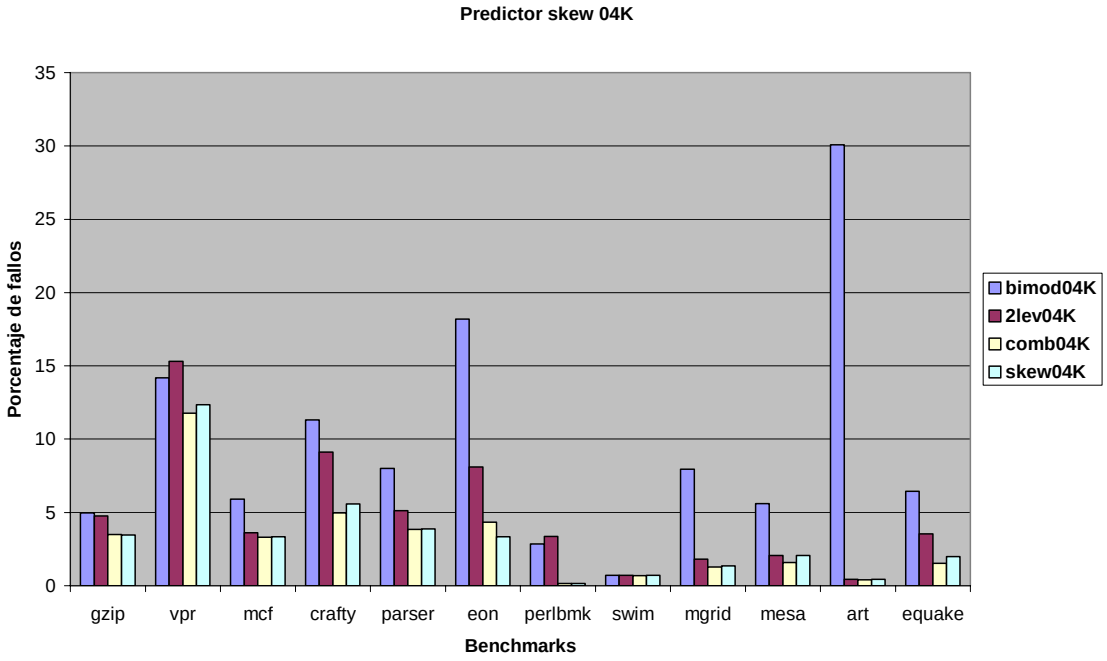




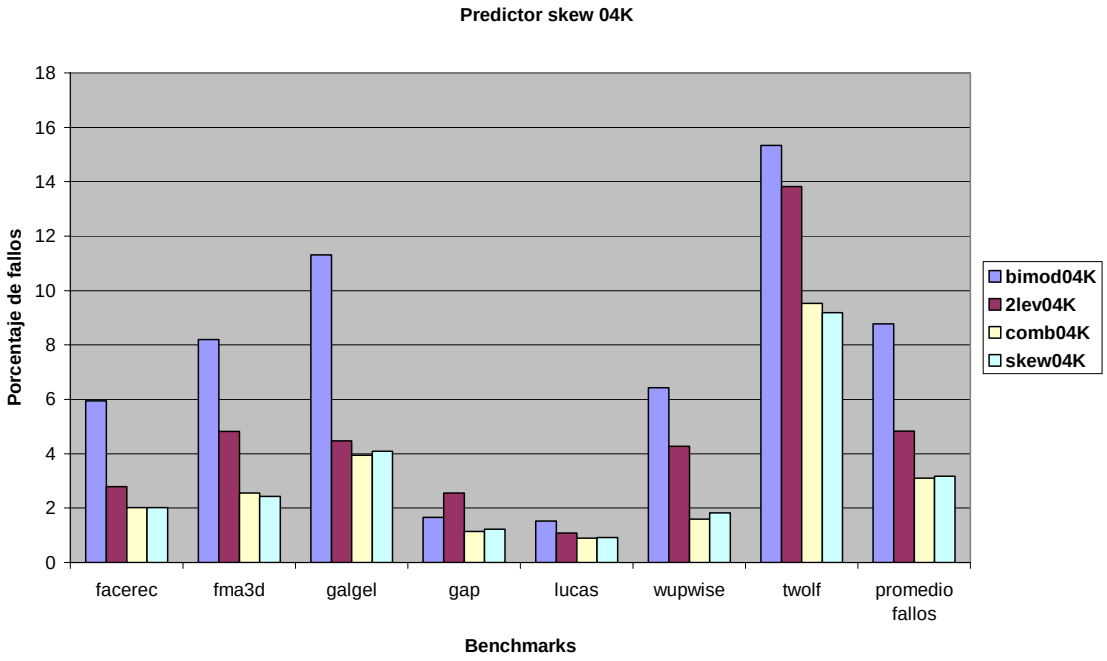
Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

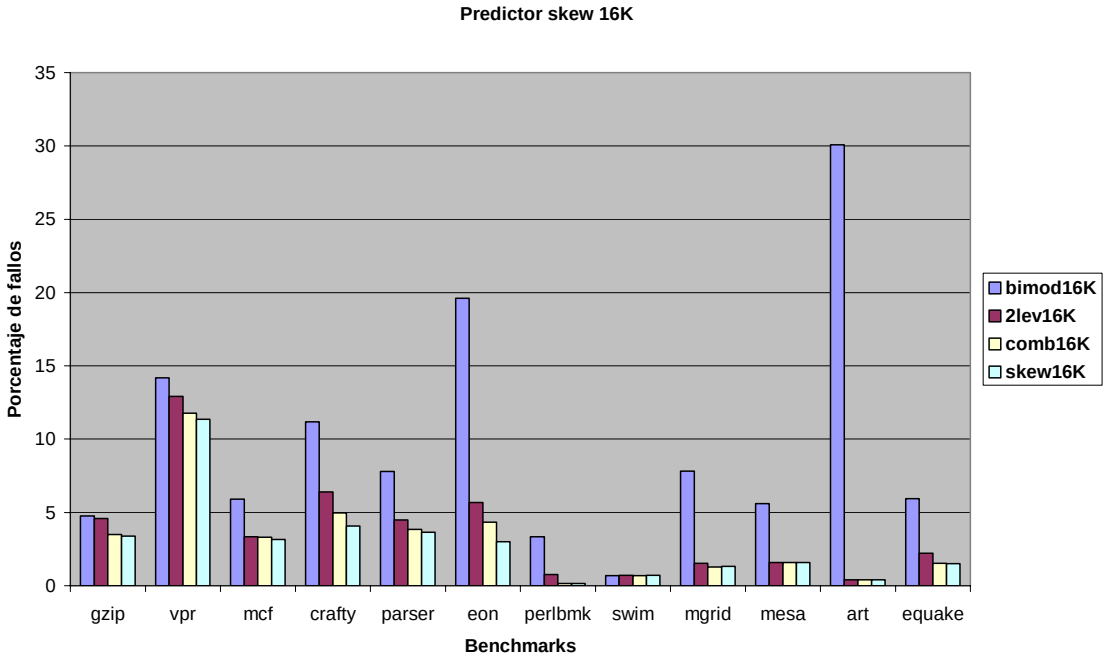


Benchmarks "*Ejecuciones completas*":

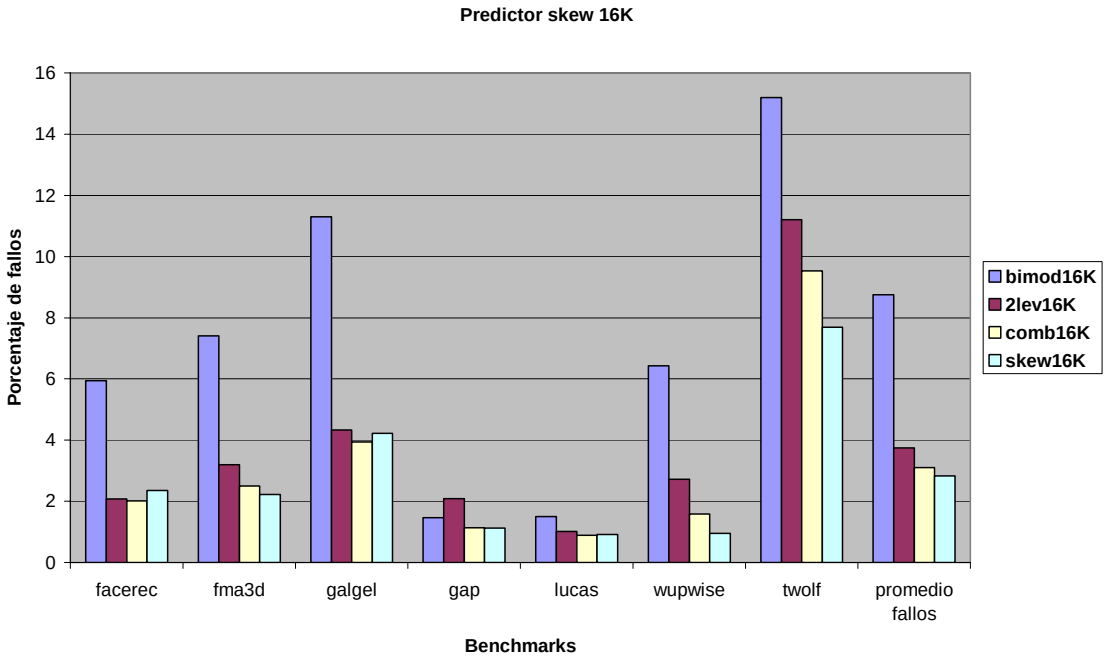


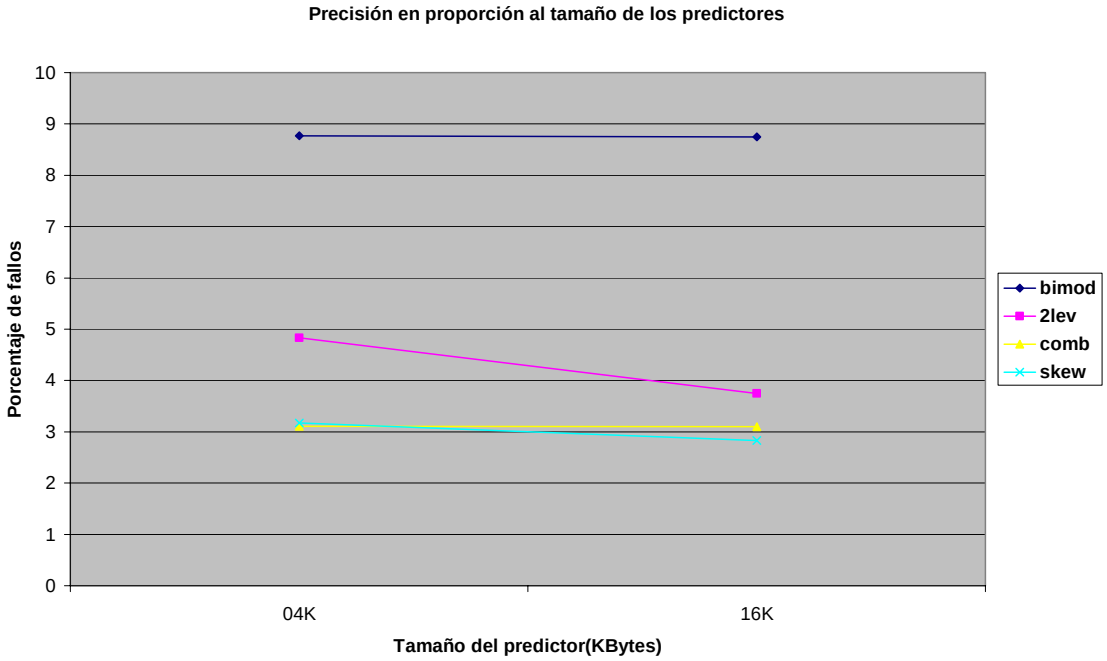
Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas





Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas





5. Conclusión

La idea del voto mayoritario es una idea aplicada anteriormente en la mejora de las caches. Es una idea contrastada, que reduce el uso de hardware eliminando la utilización de tags, pero que por otro lado lo aumenta multiplicando el número de tablas utilizadas. El resultado es bastante interesante, se consigue reducir en gran medida el aliasing, pero el gasto en hardware para la construcción de nuevas tablas hace que el coste sea elevado.

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

Uno de los mayores peros es el aumento del aliasing capacitivo, "las tablas se llenan antes", ya que en el mismo espacio que antes construiamos una sola tabla, ahora hemos de construir 3.

6. Bibliografía

[15] P. Michaud, *A. Seznec, R. Uhlig Trading Conflict and Capacity Aliasing in Conditional Branch Predictors.* 24th Annual Symposium on Computer Architecture, 1997

[16] M.D. Hill *Aspects of Cache Memory and Instruction Buffer Performance*. PhD thesis, University of California, Berkeley, 1987.

[17] A. Seznec *A case for two-way skewed associative caches.* Proceedings of the 20th Annual Symposium on Computer Architecture

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

PREDICTOR FILTER

1. Motivación

La idea de este predictor se basa en que mediante la observación de los comportamientos de los saltos, puede predecirse cuál será la dirección que tomará en ocasiones futuras sin necesidad de usar un predictor global. Esto nos recuerda el fundamento del predictor agree: los saltos que presentan un comportamiento muy fiel al bit de bias, pueden ser predichos con gran exactitud sólo con dicho bit. Sin embargo, este nuevo predictor, no calculará el bit de bias de un modo estático, sino que se hará con un contador dinámico.

Este predictor, trata de eliminar dos tipos de *aliasing:* el neutro y el destructivo.

Además de intentar reducir al máximo las interferencias en la PHT, este predictor también persigue el objetivo de eliminar la información redundante que se almacene en la PHT, para lo cual, borrará de la PHT los saltos estáticos.

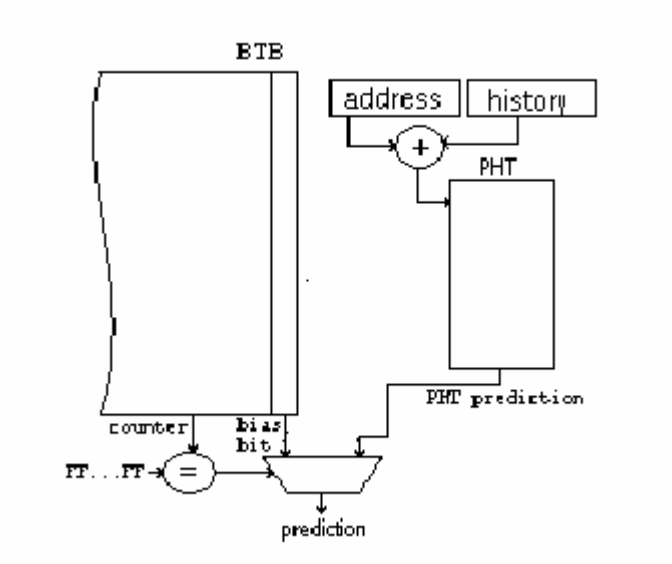
2. El predictor Filter

Diferencias estructurales

Este predictor, como puede verse en la figura de abajo, introduce para cada salto albergado en la BTB un contador saturado y un bit de bias. Los contadores serán contadores saturados de n bits, donde n es el número de bits que posee el registro de historia global, por lo que el contador variará en el rango: 0 < = x < 2^n.

La inicialización de este bit de bias y de los contadores tiene lugar cuando es introducido el salto en la BTB por primera vez. Existen dos alternativas para la inicialización de los contadores: tomar el máximo valor que puedan alcanzar (2^n - 1), forzando así el comienzo del mecanismo de filtrado de modo inmediato, o inicializarlos al mínimo valor posible (0), caso en el que el mecanismo de filtrado comenzaría a actuar tras un intervalo de tiempo durante el que las predicciones serían proporcionadas por los contadores de la PHT.

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas



Predictor Filter

Diferencias semánticas

Del mismo modo que ocurría en el predictor agree, y debido a que ambos se basan en una idea similar, este predictor también posee un matiz semántico distinto en la predicción dada por los contadores saturados de la PHT, que en los predictores de dos niveles proporcionaban directamente la predicción del salto. En este caso, sólo su valor será utilizado como predicción cuando el contador saturado de la BTB asociado al salto en cuestión, no haya alcanzado la saturación. En este caso, si el contador de la PHT es 0 o 1 se predecirá el salto como *no tomado* y si el contador es 2 o 3, se predecirá como *tomado*.

En caso contrario, si el contador de la BTB está saturado, la predicción será dada por el bit de bias, ya que el hecho de haber alcanzado la saturación significa una plena confianza en la predicción que proporcione este bit; por esto, cuando este contador no se encuentra saturado, tomaremos la predicción dada por la PHT, no considerando aún la predicción dada por el bit de bias como suficientemente segura.

Actualización del bit de bias y de los contadores de la BTB

Al resolverse la dirección de un salto, si ésta es la misma que la predicción dada por el bit de bias, el contador asociado en la BTB a dicho salto será incrementado, aumentando entonces la confianza en el bit de bias. En caso de que el contador hubiese alcanzado el grado de saturación, se mantendría en él, sin poder seguir aumentando su valor ya que habría alcanzado el máximo posible. Si la predicción proporcionada por el bit de bias no hubiese sido correcta, el contador

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

se inicializará a 0 y el bit de bias será invalidado. De este modo y para futuras predicciones, sabremos que el bit de bias no proporcionará una buena predicción del salto.

El contador asociado al salto en la PHT no será actualizado con el sentido tomado por el salto en el caso de haber usado la predicción del bit de bias (caso de saturación del contador asociado en la BTB), por tanto esta información es *filtrada* de la PHT. Sin embargo, si la predicción fue dada por la PHT, sí será actualizado tal contador, actualizando de manera habitual su valor: será incrementado si el salto se ha tomado, y decrementado en caso contrario (el incremento y decremento será una operación módulo 3, ya que los contadores de la PHT, siguen siendo contadores saturados de 2 bits, variando su valor en el rango: 0<= x <=3).

3. Comparativa. Análisis de resultados

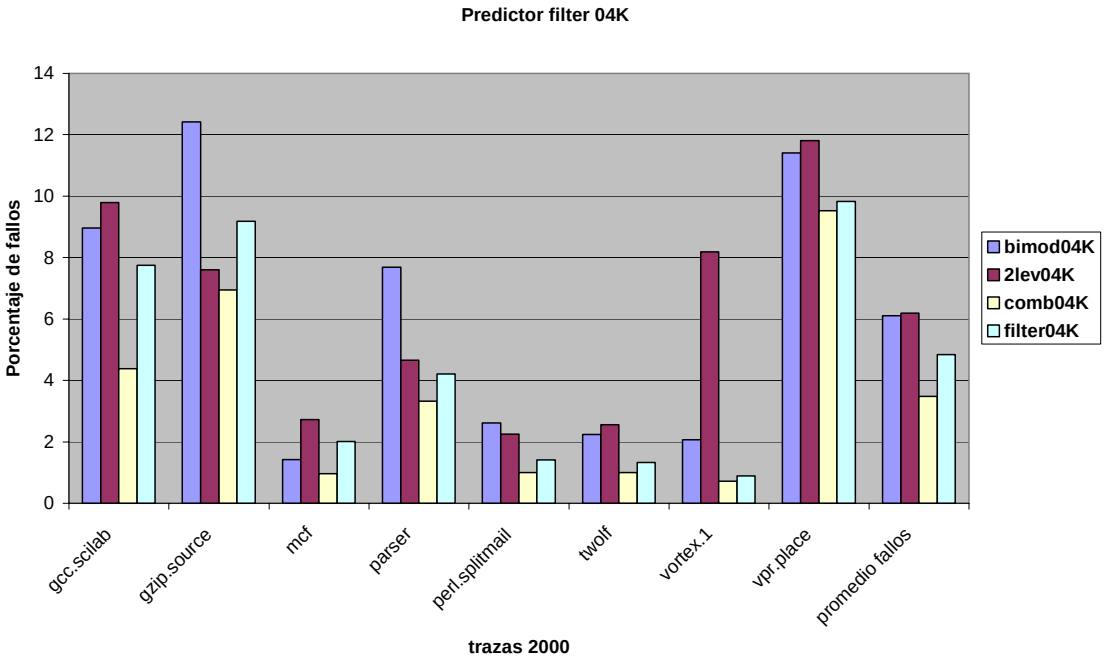
Al encontrarse este predictor basado en la idea principal del predictor agree, deberemos fijarnos como hicimos al comparar los resultados de dicho predictor, con la mejora que va a suponer el predictor filter en relación al predictor de dos niveles principalmente, aunque como es de esperar, también supondrá una mejora frente al predictor bimodal.

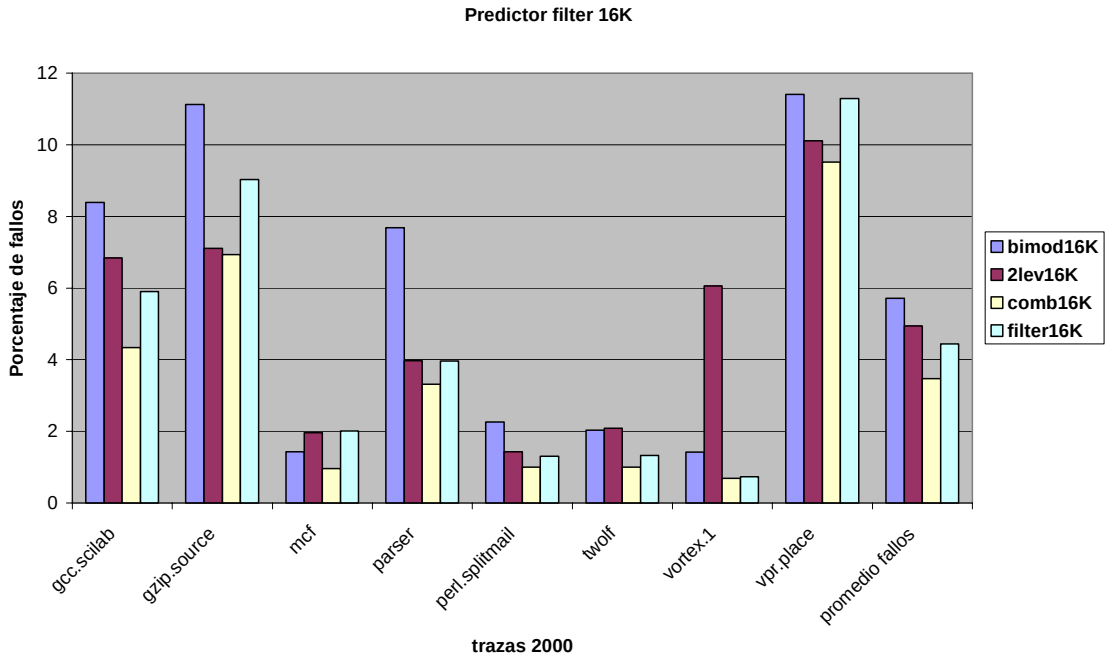
En las siguientes gráficas, es precisamente esto lo que cabe destacar: el predictor filter presenta una disminución de la tasa de fallos en ambos conjuntos de tests con los que ha sido estudiado con respecto a esos dos predictores. Sin embargo, no se comporta tan bien como queda demostrado que lo hace el predictor que combina las predicciones de dos, y de hecho, este predictor supone un leve descenso de la mejora gradual que veníamos obteniendo en los predictores estudiados.

En cuanto a la comparativa según tamaños utilizados en la implementación, el predictor filter de 16Kb sí que logra mejorarse a sí mismo cuando es implementado con un tamaño menor. Para dar una idea del descenso del porcentaje de fallos que supone, la pendiente de la recta que podría representar este descenso, es similar a la que presentaba el primer predictor y menor que la presentada por el predictor bi-mode, por ejemplo.

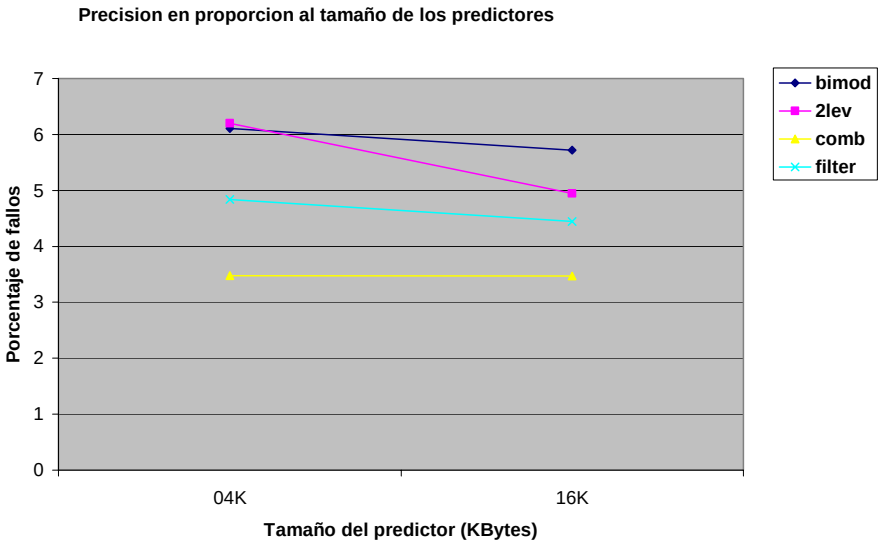
Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

Benchmarks "*Trazas 2000*":

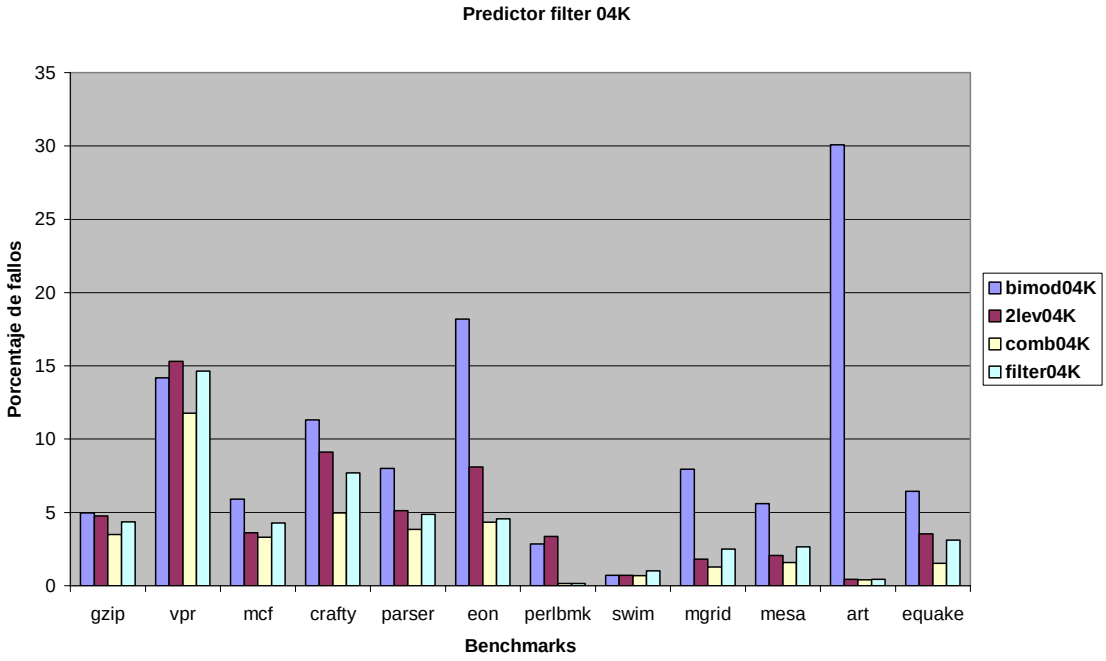




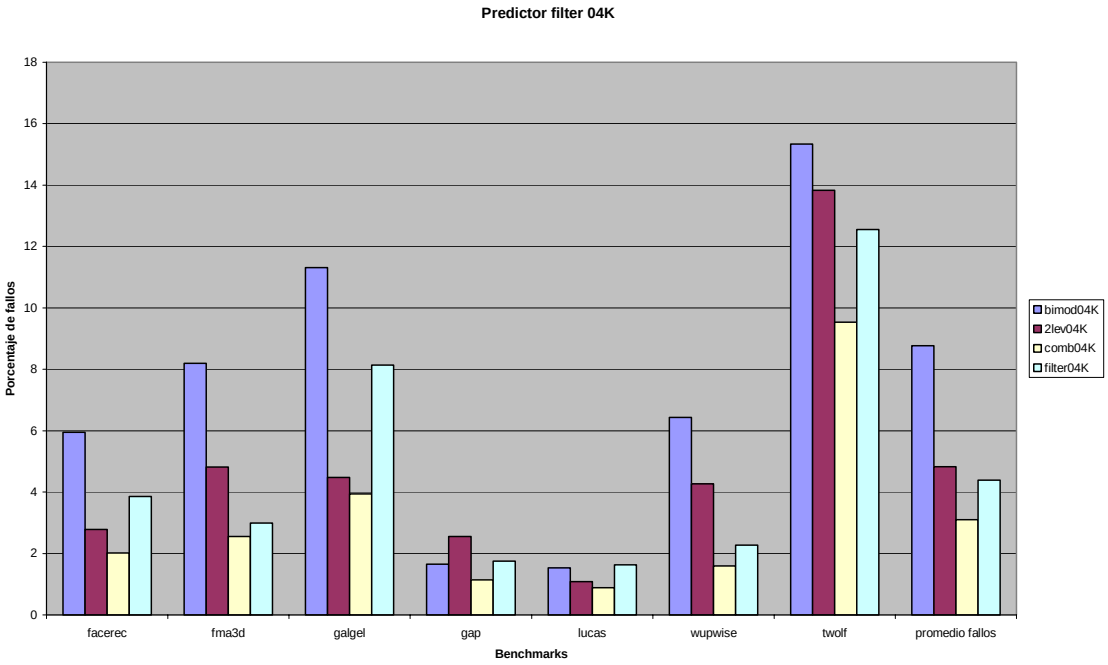
Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

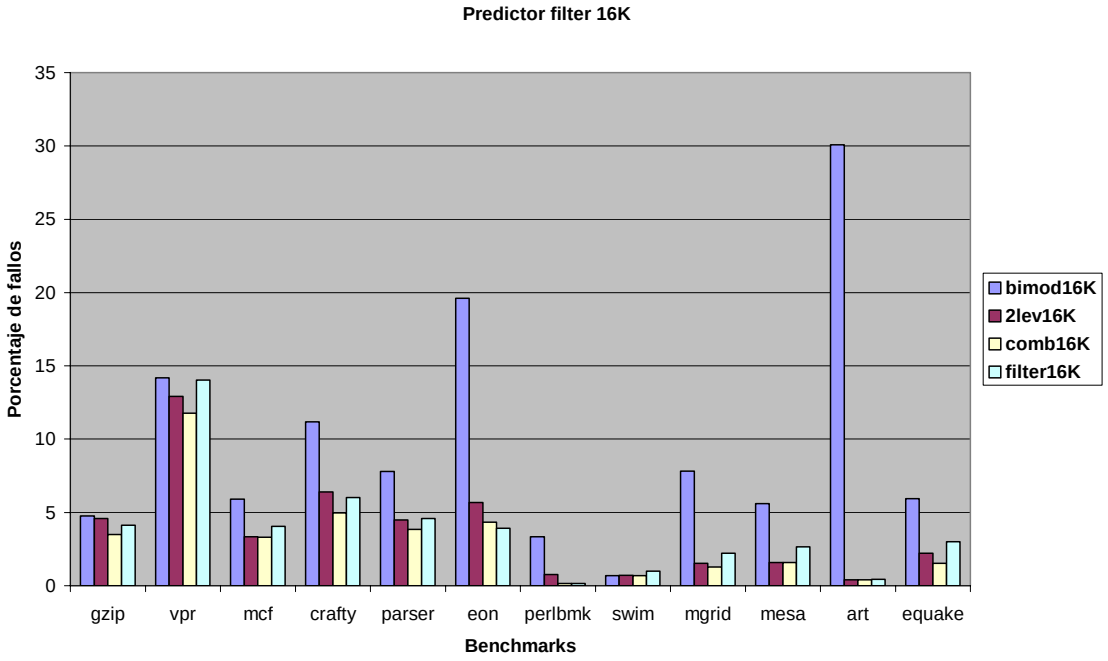


Benchmarks "*Ejecuciones completas*":

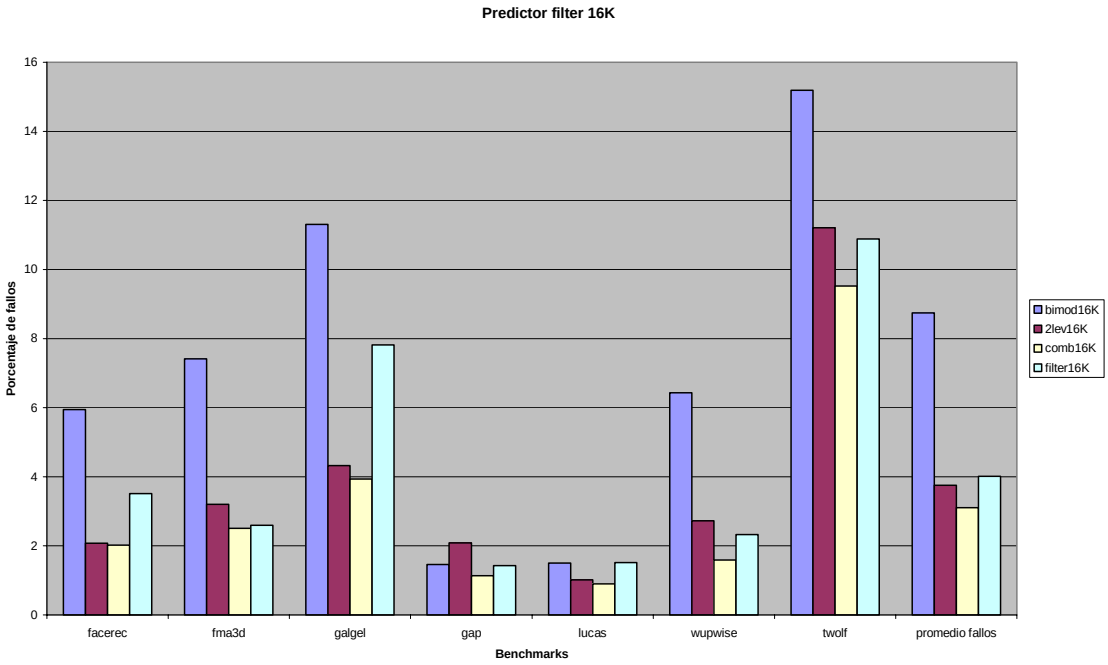


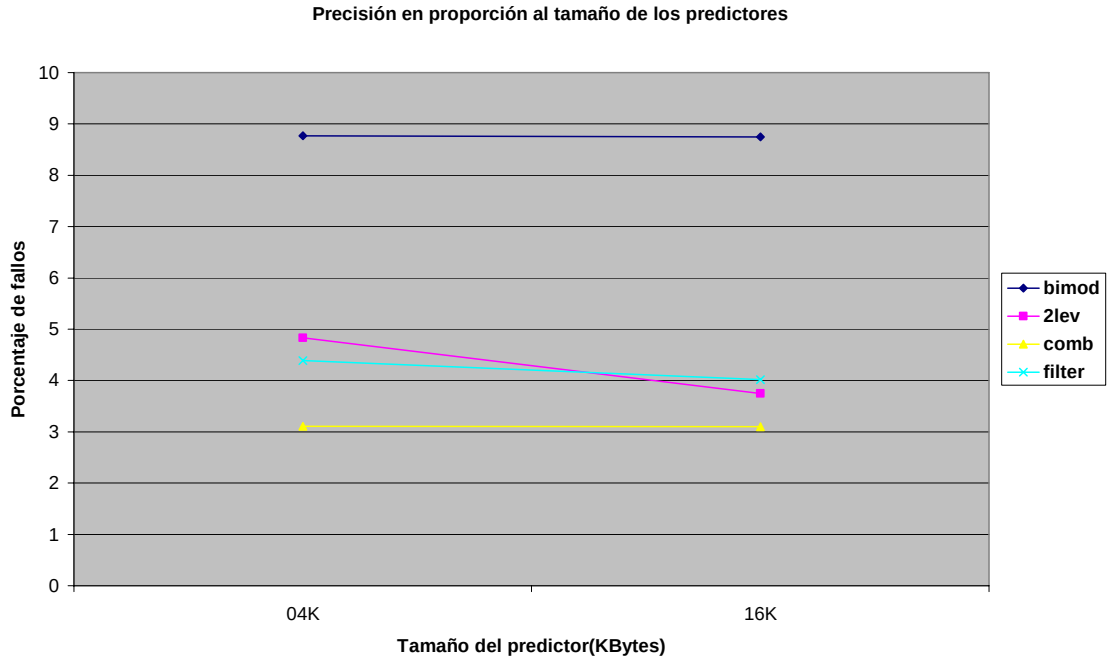
Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas





Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas





4. Conclusión

En resumen, el predictor Filter presenta otra nueva técnica de predicción de saltos basada, principalmente, en la disminución del aliasing entre los saltos de la PHT, ya que con la implementación del GShare se demostró que este fenómeno era culpable en gran medida de los fallos en las predicciones. En

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

concreto, este predictor intenta eliminar aquellas interferencias neutras y negativas.

5. Bibliografía

[18] *"Design and Performance Evaluation of Global History Dynamic Branch Predictors":* Chih-Chieh Lee (Digital Equipment Corp.), I-Cheng K. Chen (California Microprocessor Division Advanced Micro Devices), Trevor Mudge (EECS Department University of Michigan).

[19] *"The YAGS Branch Prediction Scheme":* A. N. Eden and T. Mudge, (Dept. EECS, University of Michigan, Ann Arbor).

[20] *"Branch Transition Rate: A New Metric for Improved Branch Classification Analysis":* Michael Haungs, Phil Sallee, and Matthew Farrens (Department of Computer Science, University of California, Davis)

PREDICTOR YAGS

1. Motivación

Para llevar a cabo las mejoras potenciales de los, hoy ampliamente usados, procesadores superescalares, es esencial poder contar con un buen mecanismo de predicción de saltos. La introducción de los esquemas adaptativos de dos niveles fue un paso importante en este sentido. Somos capaces de predecir instrucciones de salto con un éxito del 90% o más. De los esquemas de dos niveles, el esquema de historia global parece trabajar mejor para código entero. Esto, en parte, es debido a que en estos programas hay un mayor número de instrucciones condicionales (if-then-else), además de que éstas están, usualmente, muy correlacionadas.

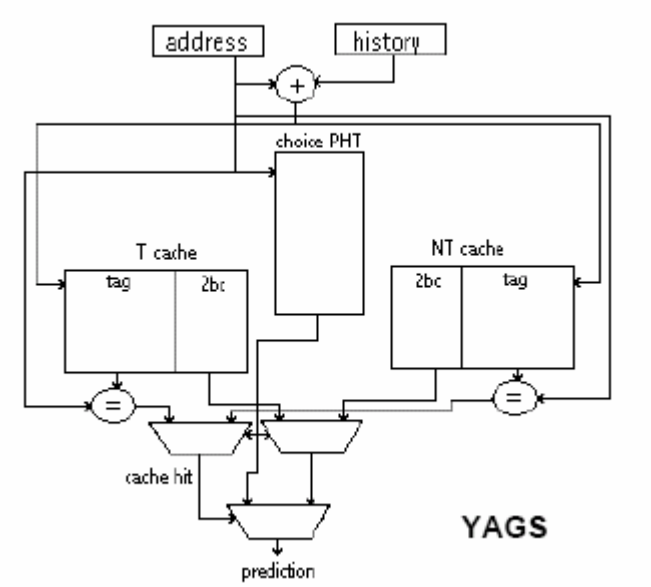
El principal problema que reduce la tasa de predicción en los esquemas globales es el aliasing entre dos índices (un índice está formado normalmente por los bits de historia y dirección) que señalan la misma entrada en la tabla histórica de patrones (Pattern History Table, PHT). Como la información almacenada en la PHT siempre es "Tomado" o "No Tomado", dos saltos que tengan la misma dirección de PHT y que tengan la misma información, no producirán fallos en la predicción. Esta situación es definida como aliasing neutro. Por otro lado, puede haber dos saltos que tengan información distinta y que posean la misma dirección de PHT, lo que produce un fallo en la predicción.

El predictor YAGS (Yet Another Global Scheme, "Justamente Otro Esquema Global") combina la fuerza de otros esquemas globales para eliminar el aliasing.

2. Funcionamiento del predictor

El estudio de otros predictores para reducir el aliasing en los esquemas globales, sugiere que dividiendo la PHT en dos flujos de saltos (correspondientes a los comportamientos de "Tomados" y "No tomados"), tal y como sucede en el predictor Agree y en el predictor Bi-Mode, es una buena idea. Sin embargo, tal y como ocurre en el predictor Skew, no queremos despreciar los efectos beneficiosos del aliasing neutro. Además, sería bueno que se pudiera reducir la cantidad innecesaria de información almacenada en la PHT en predictores como el Filter, aunque no a expensas de fallar en algunas de las predicciones de saltos.

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas



La motivación del YAGS es que lo que realmente es necesario almacenar es el comportamiento del salto en aquellas instancias en las que no esté de acuerdo con el comportamiento general. Si usamos un predictor bimodal para almacenar el comportamiento, tal y como sucede en el esquema del predictor Bi-Mode, todo lo que necesitamos almacenar en las PHTs de dirección son las instancias en las que el salto no se comporta según su comportamiento sugiere. Esto reduce la cantidad de información guardad en las PHTs de dirección, y por tanto las PHTs de dirección pueden ser más pequeñas que la PHT de elección. Para identificar estas instancias in las PHTs de dirección necesitamos añadir unos pequeños tags (6-8 bits) para cada entrada, refiriéndonos a ellos ahora como Cachés de dirección. Estos tags guardan los bits menos significativos de la dirección del salto, y virtualmente eliminan el aliasing entre dos saltos consecutivos. Cuando un salto tiene lugar en el flujo de instrucciones, se accede a la PHT de elección. Si la PHT de elección indica "tomado", se accede a la caché de "no tomados" para comprobar si el salto es un caso especial en el que la predicción no está de acuerdo con el comportamiento. Si hubiera un fallo en la caché de "no tomados", se usará como predicción lo que indique la PHT de elección. Si hubiera un éxito en la caché de "tomados", lo que indique ésta será usado como predicción. Un conjunto similar de acciones se harán para el caso en el que la PHT de elección indique "no tomado", en cuyo caso la caché que se examinará será la caché de "tomados". La PHT de elección es direccionada y actualizada tal y como se hacía en el predictor Bi-Mode. La caché de "No tomados" se actualiza si una predicción suya ha sido usada. También es actualizada si la PHT de dirección indica "Tomado" y el salto finalmente no se toma. Lo mismo ocurre con la caché de "Tomados".

Necesitamos tener en cuenta el aliasing para las instancias de los saltos que no estén de acuerdo con el comportamiento del salto. Después

de tener la solución con los tags, la solución natural para el problema del aliasing es añadir asociatividad a las cachés de dirección.

Cuando hacemos que las cachés de dirección sean asociativas por conjuntos, estamos teniendo un coste extra para mantener una buena política de reemplazamiento. Por ejemplo, en una caché asociativa por conjuntos de dos vías, un bit para cada dos entradas será suficiente para conocer cuál entrada fue reemplazada la última vez. Nosotros usamos una política de reemplazamiento LRU, con una sola excepción: una entrada en la caché de "Tomados" que indica "No tomado" será reemplazada antes para eliminar la información redundante. La información de una entrada de la caché de "Tomados" indica "No tomado" ya se encuentra en la PHT de elección, y por tanto es redundante y puede ser reemplazada.

3. Resultados

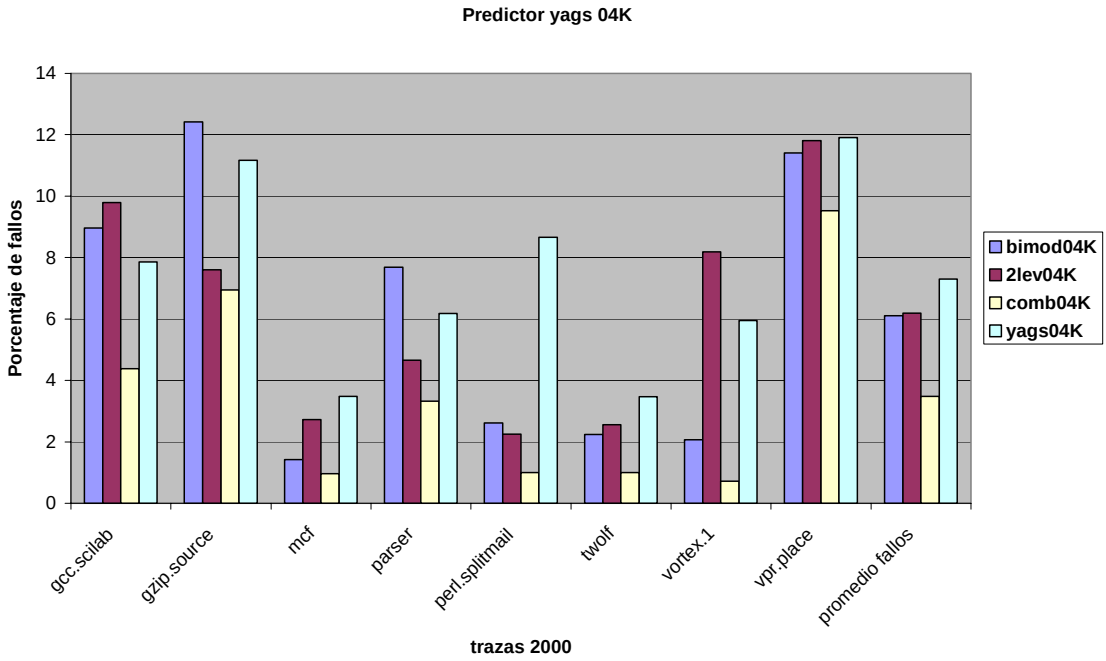
Presentamos a continuación, como venimos haciendo hasta ahora en el estudio de cada uno de los predictores, los resultados obtenidos para ejecuciones de ambos grupos de tests usando como predictor de salto nuestra implementación del predictor Yags. En este caso, los resultados no son tan buenos como cabría de esperar: la tasa de fallos es demasiado alta, siendo menor que la obtenida por el predictor de dos bits en la mayor parte de las ocasiones, y en muchas ocasiones, mejor que la obtenida por el predictor de dos niveles que hemos considerado entre los tres básicos (*"2lev"*), al que supera, por ejemplo, en los tests de *"Trazas 2000"*: "gcc.scilab", "vortex.1", "vpr.place ", y en los tests de *"Ejecuciones completas":* "vpr", "gzip", "eon".

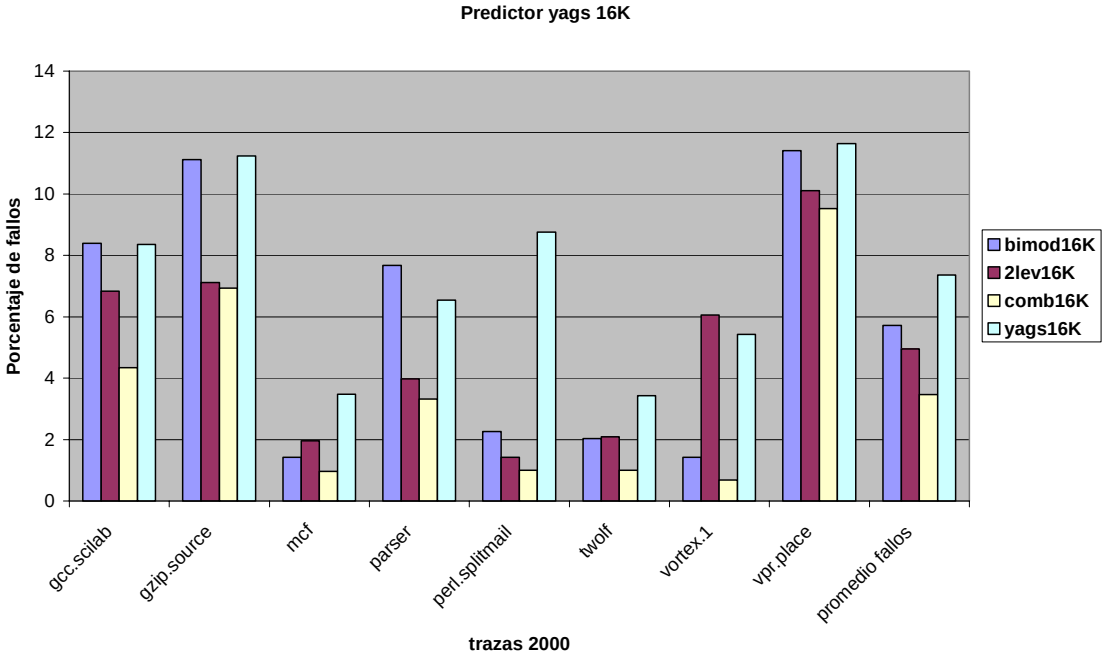
Nuestra implementación del predictor Yags, no alcanza los buenos resultados del predictor *"comb"* ni tampoco continúa con el descenso de la tasa de fallos que veníamos consiguiendo con los predictores presentados en este documento.

En cuanto a la diferencia de resultados obtenida para las implementaciones de 4 y 16Kb, podemos observar que en ambos conjuntos de tests, apenas se nota descenso de la tasa de fallos (nótese que la línea que une ambos puntos, es prácticamente paralela al eje de abcisas).

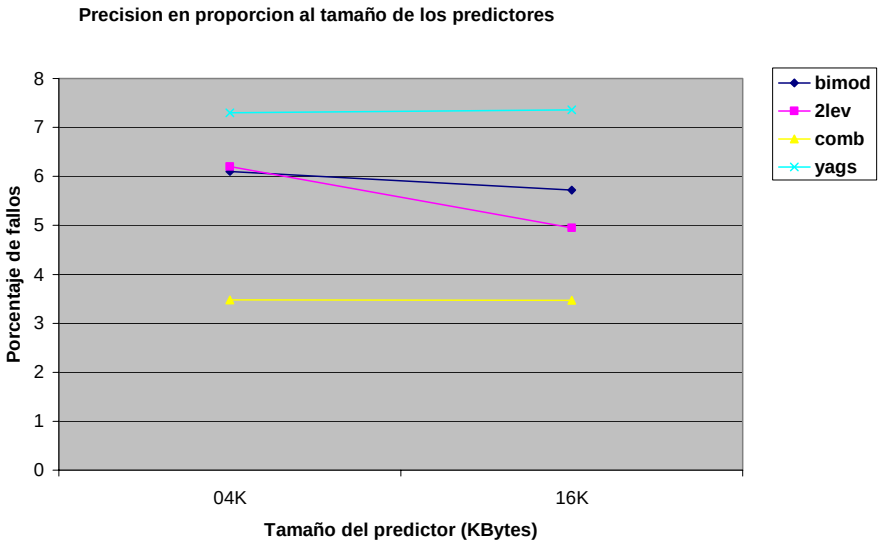
Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

Benchmarks *"Trazas 2000":*

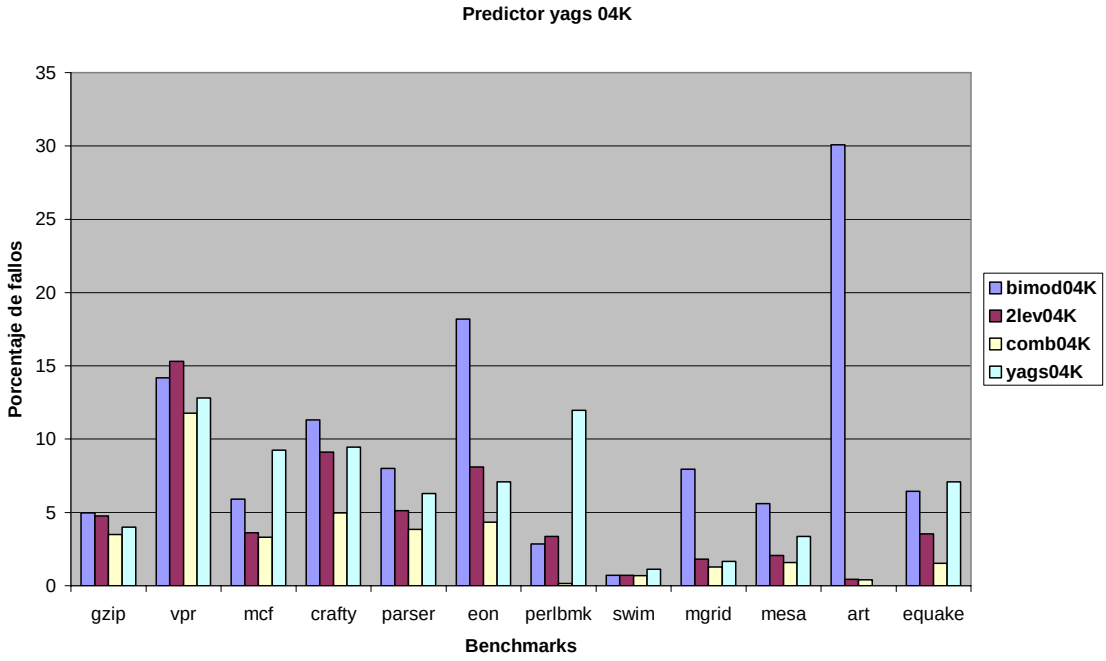




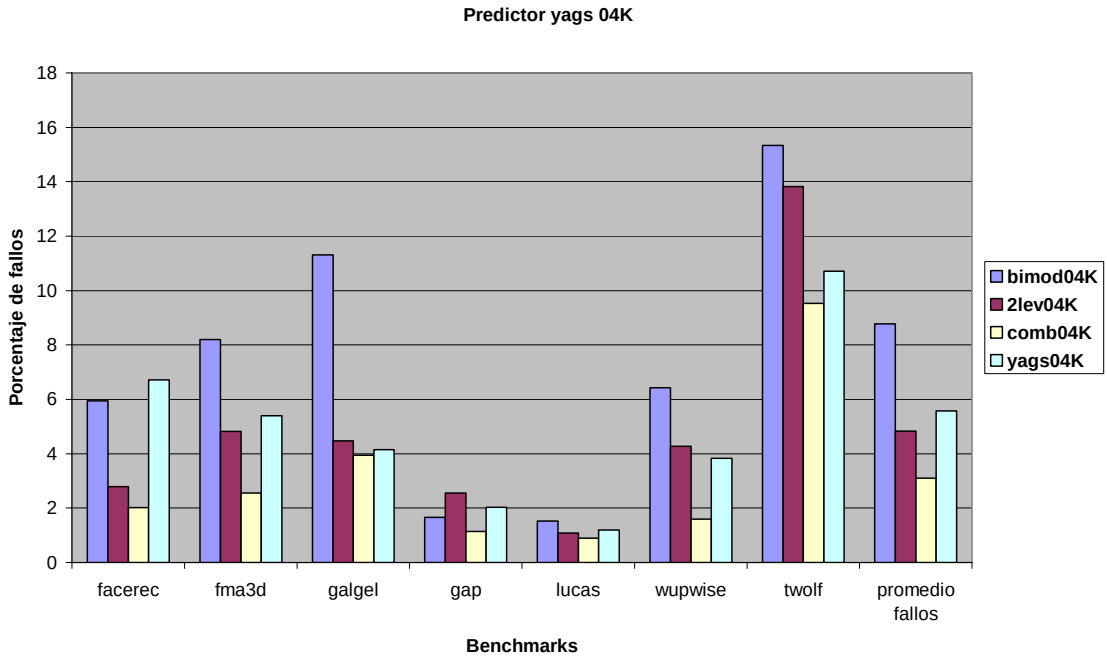
Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

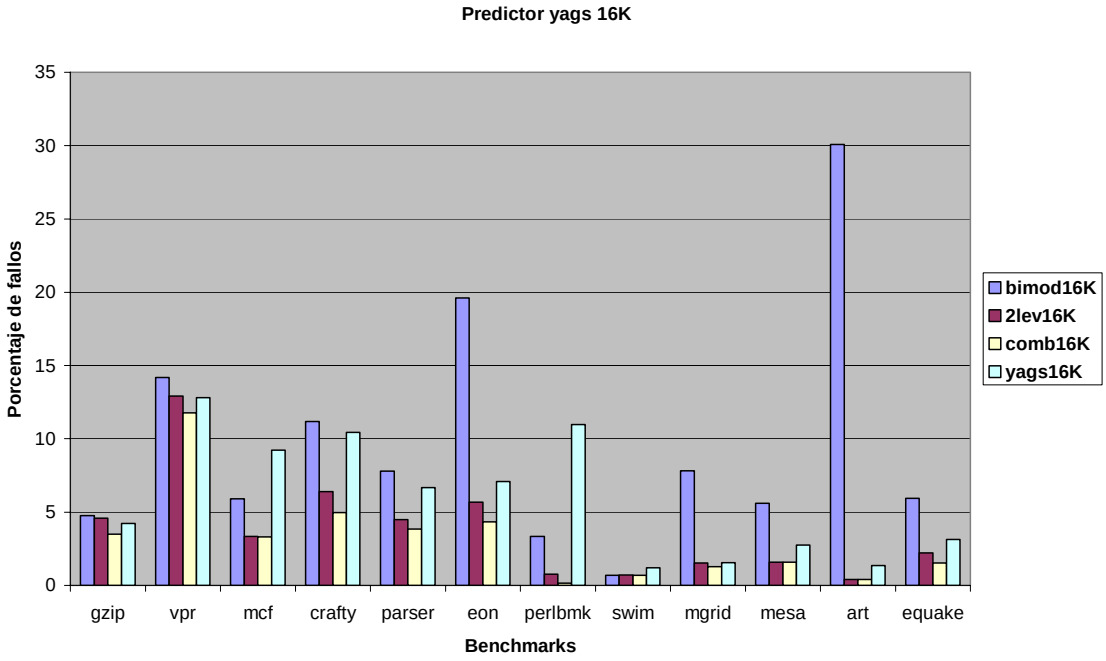


Benchmarks "*Ejecuciones completas*":

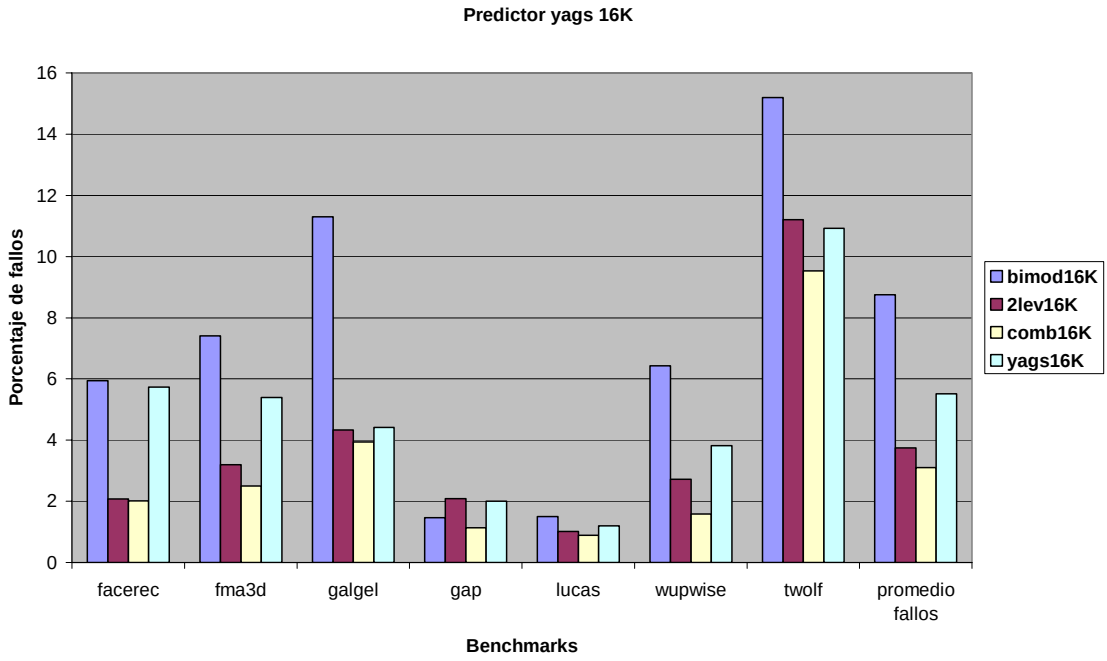


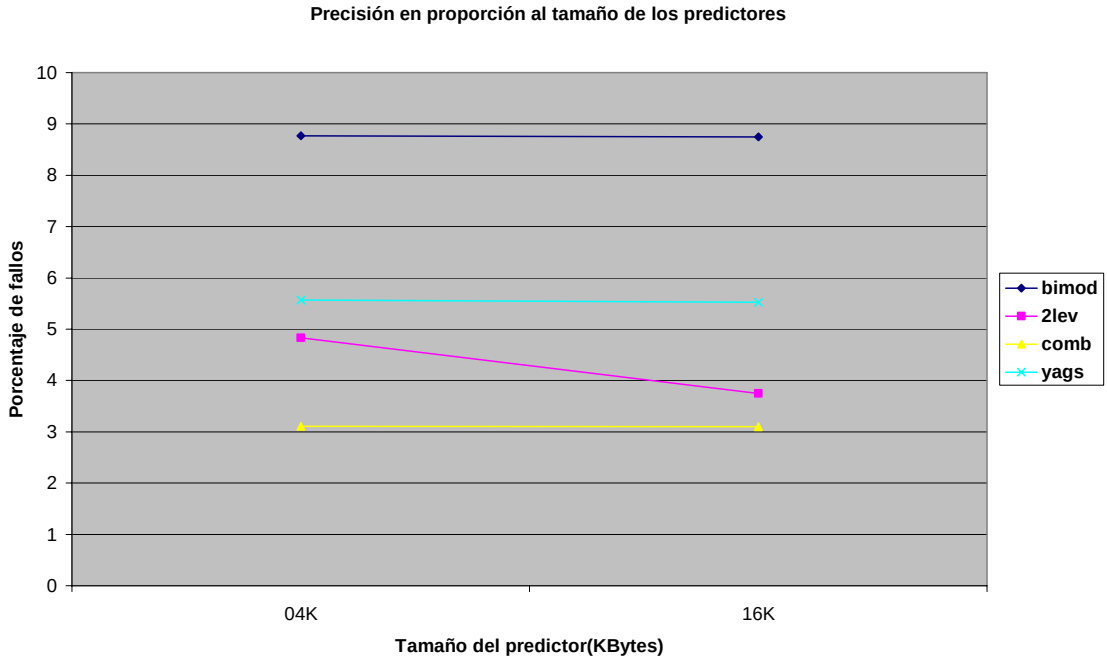
Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas





Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas





4. Conclusión:

El YAGS es un esquema de predicción global de dos niveles que intenta eliminar el aliasing en la PHT combinando las ventajas de otros esquemas de predicción. YAGS actúa mejor que los otros esquemas probados. En muchos casos es considerablemente mejor. Los predictores YAGS y Bi-Mode actúan mejor en los cambios de contexto.

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

Algo sobre lo que todavía hay que investigar es el diseño del espacio. Incrementando la longitud de los tags sólo mejora la tasa de aciertos en un punto. Después de todo, incrementar el tamaño del tag puede hacer perder porcentaje de aciertos, y una tasa de predicción mejor no justifica los recursos que se gastan por tener un tag mayor. Hemos visto que el tamaño de las cachés de dirección debe estar en consonancia con el tamaño del predictor.

Finalmente, la idea básica del YAGS es la de poder combinar varios esquemas, particularmente el mecanismo del Filter. Una mejora que se podría probar es añadir una pequeña caché para capturar instancias filtradas por la PHT que no estén de acuerdo con el bit de comportamiento.

5. Bibliografía

[21] “The YAGS Branch Prediction Écheme” (A. N. Eden and T. Mudge, {ane, tnm}@eecs.umich.edu; Dept. EECS, University of Michigan, Ann Arbor).

PREDICTOR BASADO EN PERCEPTRONES

1. Motivación

Las arquitecturas de los computadores modernos han incrementado la especulación de saltos hasta alcanzar el paralelismo a nivel de instrucción. Por ejemplo, los datos contenidos en instrucciones que es posible que sean ejecutadas por el procesador en los próximos ciclos, son lanzados especulativamente, y los resultados obtenidos de la ejecución son utilizados como valores reales para avanzar una ejecución "posible" especulativa. El diseño e implementación de unos mecanismos de predicción acertados se ha convertido en la clave que sirve para desarrollar estas técnicas basadas en la especulación dinámica, de este modo, un incremento de la precisión en los predoctores implica un incremento en la precisión de la especulación, con lo que los valores antes especulativos, ahora se convierten en el flujo real del programa; así todo aquello que fue lanzado especulativamente, ahora se convierte en un avance del pipeline.

Las técnicas de aprendizaje automático (machine learning) ofrecen la posibilidad de mejorar el rendimiento de un procesador, incrementando la precisión de las predicciones. El perceptrón en concreto, utiliza una de las muchas técnicas de aprendizaje automático para incrementar dicho rendimiento.

La técnica de aprendizaje automático utilizada para construir este predictor, se basa en "neuronas" artificiales (lo que llamaremos perceptrones) que, como en la misma naturaleza, reciben una serie de entradas y proporcionan una salida, salida que a su vez tendrá una cierta "memoria", es decir, la salida de un perceptrón tiene en cuenta tanto el valor de las entradas, como el valor que recibió en ocasiones anteriores. De esta manera, el perceptrón va "aprendiendo" hasta alcanzar una salida estática que se equivoca en un mínimo de ocasiones y que no varía con el tiempo.

Si en lugar de un solo perceptrón tuviésemos varios, tendríamos una red de perceptrones (red de neuronas), de modo que lo que habría que hacer sería "entrenar" cada neurona para una entrada característica, y así se incrementaría a su vez la precisión del ajuste de la salida. Cuanto más se especialice la entrada, más concreta sería la salida.

Estas redes neuronales, utilizadas para funciones de aprendizaje automático, suelen pasar por dos fases. La primera fase es una fase de "entrenamiento" o aprendizaje, en la cual se recibe un subconjunto representativo de los valores de entrada para que la red, y dentro de la red cada perceptrón en concreto, recibe una muestra de los datos que recibirá a lo largo de una ejecución normal, de modo que ajusta sus parámetros para dar así una salida que en la mayoría de los casos se ajuste a lo esperado. La segunda fase es la de ejecución en concreto, la red, con sus perceptrones entrenados, recibe valores "reales" y responde basándose en los resultados del entrenamiento. Los parámetros no se reajustan.

Sabiendo que los resultados obtenidos en el aprendizaje automático son bastante acertados, la idea es utilizar los perceptrones sobre casos

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

reales, sin reentrenarlos. De esta manera, las salidas generadas tienen que ver con el entrenamiento anterior, siendo aún muy ajustadas a lo esperado.

2. El predictor basado en perceptrones

La predicción dinámica de saltos ha sido un foco de estudio intensivo en los últimos años. Recientes investigaciones se centran en un diseño de los predoctores basado en un esquema de dos niveles, una PHT (Pattern History Table) compuesta de contadores saturantes de dos bits. El mayor problema de este tipo de implementaciones es el aliasing, es decir, que varios saltos distintos correspondan al mismo contador, disminuyendo la efectividad del contador.

La predicción con métodos neuronales, más en concreto basada en perceptrones, es un rico área de estudio. Los métodos neuronales son capaces de clasificar la instancia de un salto (predecir en qué conjunto de clases va a caer una instancia concreta de un salto) y así, conociendo ciertas características de la clase a la que pertenece dicha instancia, predecir el funcionamiento del salto.

Una red neuronal es una colección de neuronas, algunas de las cuales reciben una entrada y algunas de las cuales producen una salida. Cada neurona está conectada por una serie de enlaces que tienen un peso asociado, de este modo, para una serie de entradas y una función matemática sobre el valor de las entradas y de los pesos de los enlaces, se genera una salida, que será el resultado.

Los pesos de cada uno de los enlaces son ajustados a lo largo de una fase de entrenamiento, usando un algoritmo apropiado para ello.

Para aplicar la idea de una red neuronal a la predicción de saltos, se toman como entradas las salidas generadas de la ejecución de los saltos ejecutados recientemente, lo que sería el registro de historia; el resultado de aplicar estas entradas sobre la red será la predicción de cuándo el salto será tomado. Cada vez que un salto es ejecutado y la salida correcta es conocida, el registro de historia se actualiza con dicho valor, y puede ser utilizado para la siguiente predicción, de modo que se consiga una predicción más ajustada en el futuro.

El perceptrón fue explicado por primera vez en 1962 por Rosenblatt como un método para estudiar el comportamiento del cerebro. Nosotros utilizaremos el más simple de los perceptrones conocidos, un perceptrón de una sola capa que consiste en una neurona que conecta varias entradas a través de unas funciones de peso, con una sola salida. La salida aprende una función booleana de la entrada. En nuestro caso, las entradas serán los bits correspondientes al registro de historia global, y el resultado predecirá si el salto por el que preguntamos se toma o no.

Una vez que la salida del perceptrón ha sido calculada, se utiliza un algoritmo que detallaremos en adelante, para entrenar el perceptron, es

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

decir, para darles a los pesos un valor más adecuado; para que se aprenda la función booleana de la clase a la que pertenece la entrada.

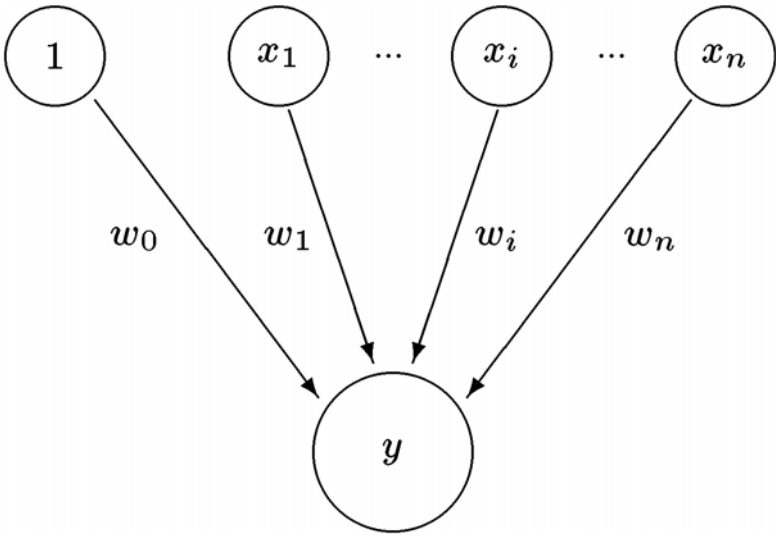


Figura 1: Esquema de un Perceptrón.

3. Implementación

En nuestra implementación basada en el Simplescalar 3.0 hemos hecho, en primer lugar una nueva estructura de predictor dentro de *bpred.h* en la que se incluyen todos aquellos campos que se nos antojan necesarios para la utilización del perceptrón. Viene a ser una lista de perceptrones, en lugar de lo que antes era una PHT, de modo que cada uno de los elementos de la antigua PHT, pasa ahora a ser un nuevo tipo de datos que hemos llamado perceptron_t. Cada uno de los perceptrones, contiene los campos siguientes: pesos y resultado, de modo que pesos es un vector que contiene los pesos propiamente, y resultado es el valor devuelto por el perceptrón.

Posteriormente hemos creado una función que inicializa correctamente las estructuras de memoria correspondientes a los perceptrones, en las funciones habilitadas para ello (bpred_create_percep y bpred_dir_create_percep). Los pesos se inicializan a valores pequeños, de 1, ya que el perceptrón estará entrenado cuando el resultado alcance un valor absoluto de 50 (más o menos, dependiendo de la configuración que queramos).

Como antes hemos dicho, la idea es que el perceptrón reciba una serie de entradas, y genere una salida. Esta salida se genera aplicando la siguiente función:

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

$$y = w_0 + \sum_{i=1}^{n} x_i w_i.$$

Donde y es la salida, w0 es el peso 0-ésimo, y xi es la entrada i-ésima, procedente del registro de historia.

Esta salida se calcula en *bpred_dir_lookup,* de este modo se halla el resultado para un patrón de historia y para un perceptrón concreto, al cual se accede a través de una función hash que utiliza la dirección de memoria del salto.

Las entradas de nuestro perceptrón son bipolares, esto es, si en el registro de historia aparece un 0, "no tomado", se interpreta como un -1, y si aparece un 1, "tomado", se interpreta como un 1. Si el resultado de y es negativo, se predice como no tomado, mientras que si el resultado de y es positivo, se predice como tomado.

Como ya habíamos comentado anteriormente, cada perceptrón ha de ser entrenado por un conjunto de datos de entrada, preferiblemente representativos, es decir, que en dicho conjunto aparezca un cierto numero de elementos de cada una de las clases que van a conformar lo que luego será la entrada. Para hacer esto, hay dos opciones, una de "profiling", donde utilizaríamos una función que, tras un conjunto de datos de entrada, guardase los pesos resultantes como entrenados en un archivo, y otra función que cargaría dichos datos en la siguiente ejecución. La segunda opción es poner un límite de entrenamiento, y que cada perceptrón se entrene dinámicamente hasta alcanzar ese límite, momento a partir del cual ya no se entrenará más, y dará siempre los resultados adecuados a los pesos. Esta es la opción elegida por Daniel. A. Jiménez para su perceptrón, y es la que nosotros utilizamos.

El algoritmo de entrenamiento estaría implementado en *bpred_update*, y sería la versión en c de lo siguiente:

```
if sign(y_out) ≠ t or |y_out| ≤ θ then
    for i := 0 to n do
        w_i := w_i + t x_i
    end for
end if
```

Siendo en este caso t el signo del salto (negativo si no tomado, positivo en caso contrario), yout el signo predicho por el perceptrón en *bpred_lookup*, cada wi es el peso correspondiente del perceptrón, y cada xi es la entrada del registro de historia.

Se ve que una vez que se alcanza el valor de la salida que supere a Zeta, se deja de entrenar, de modo que el valor de salida del perceptrón sería estático para el conjunto dado de datos de entrada.

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

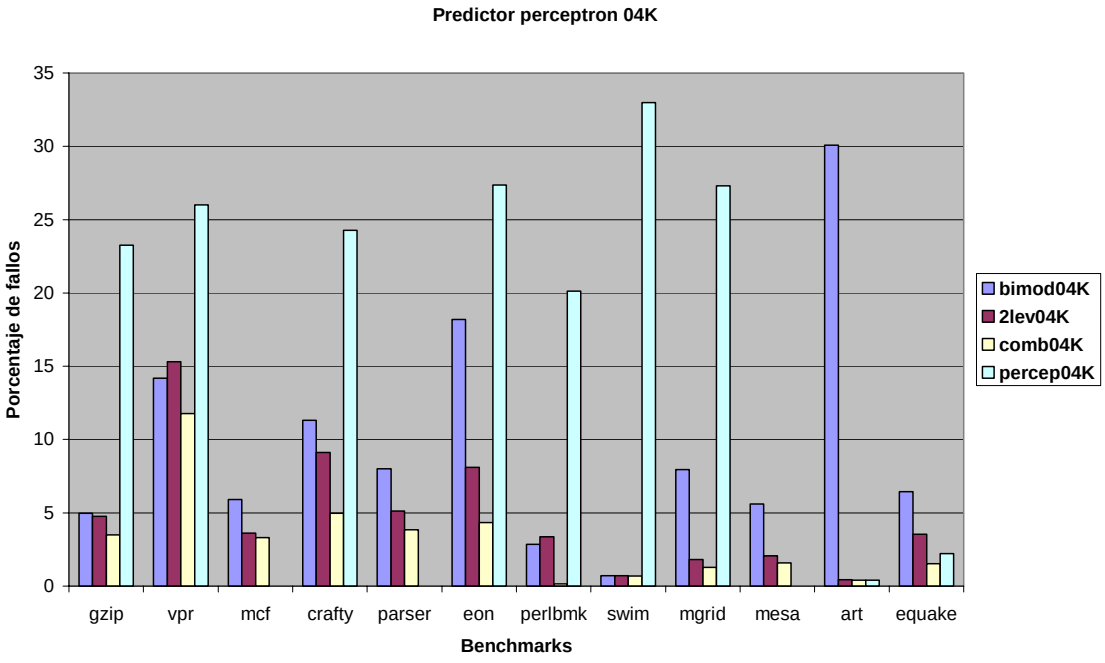
4. Comparativa. Análisis de resultados

Realmente este predictor debería haber alcanzado unos resultados tales que marcasen un óptimo global frente a cualquiera de los predictores presentados hasta aquí. A continuación mostraremos las tasas de fallos de los predictores iniciales y las obtenidas con nuestra implementación del predictor *"perceptrón"*, para poder comparar la efectividad de cada uno frente a la de los demás y, parece obvio, que nuestra implementación no parece alcanzar los resultados esperados.

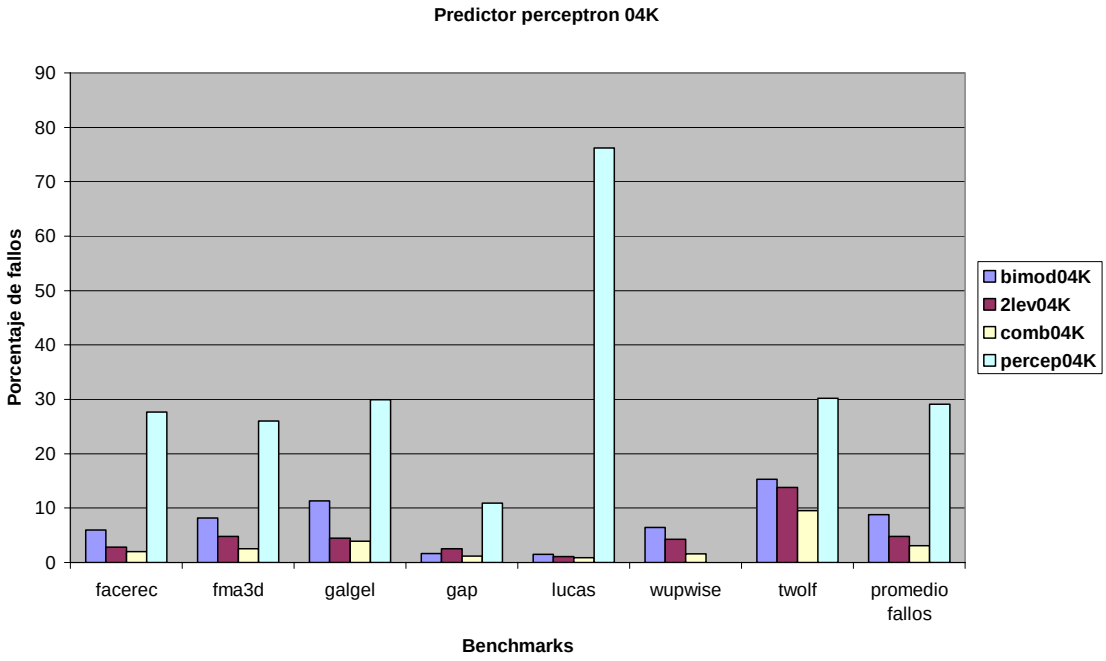
Para este predictor, mostramos los resultados obtenidos en la ejecución del predictor sobre el segundo conjunto de tests (*"Ejecuciones Completas"*) ya que son de mayor tamaño, lo que nos ha llevado a descartar la posibilidad de atribuir estos malos resultados a los tamaños pequeños del primer grupo de tests, y a orientar la búsqueda de posibles mejoras en otra dirección.

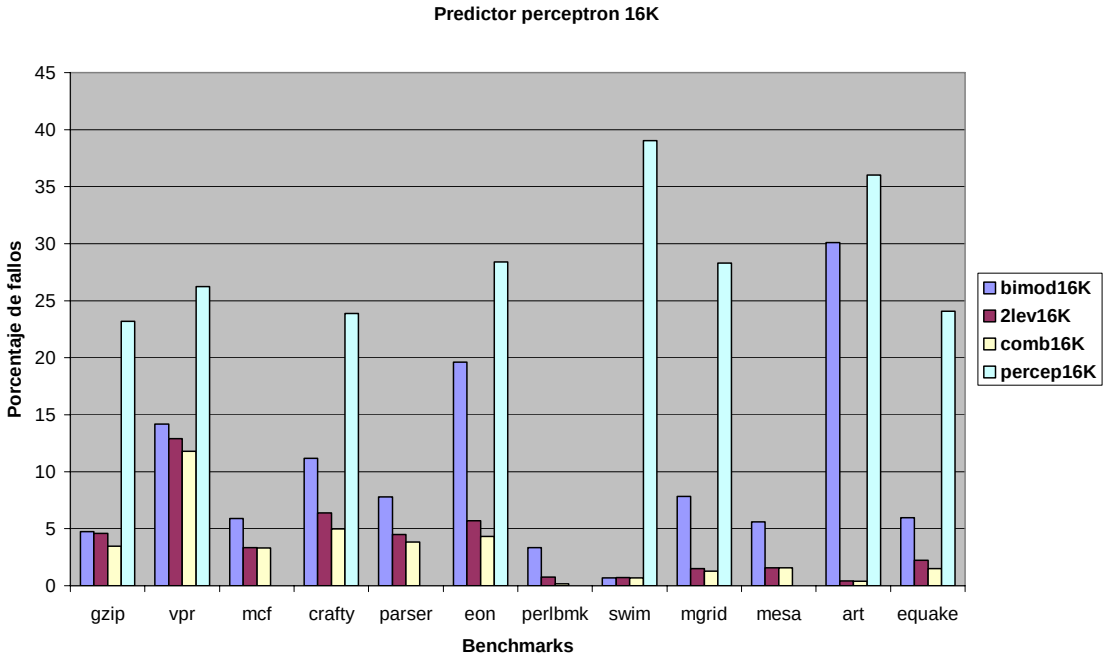
La gráfica que venimos mostrando sobre la efectividad de predictores según el tamaño con el que han sido implementados, parece adquirir en este predictor una tónica totalmente distinta; en este caso, la menor tasa de fallos es obtenida para la implementación de tamaño menor (4Kb).

Benchmarks *"Ejecuciones completas":*

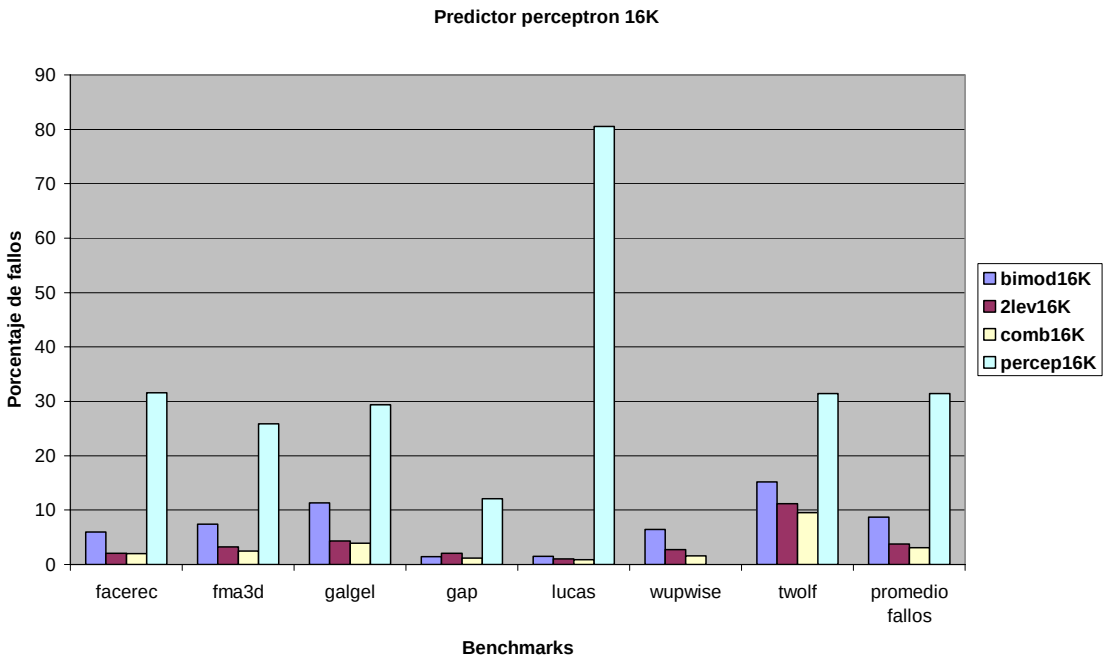


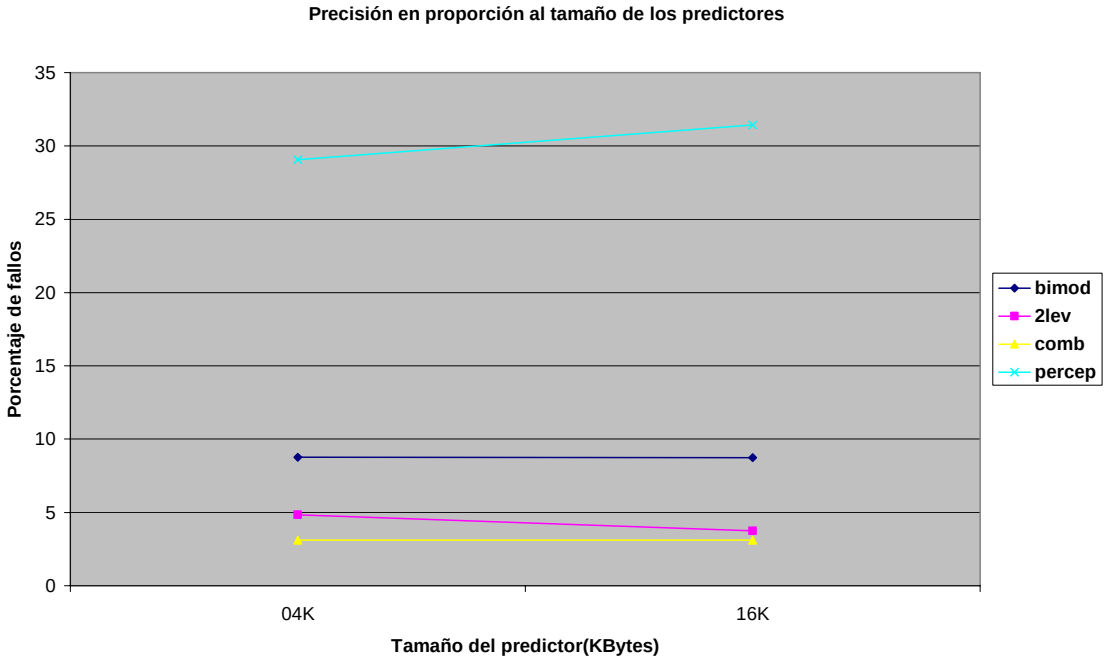
Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas





Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas





5. Conclusión

A pesar de basarnos casi letra por letra en lo explicitado por Daniel A. Jiménez en su documento "Neural Methods for Dynamic Branch Prediction", los resultados no podían ser más desastrosos. El perceptrón es el peor de los predictores estudiados, de modo que suponemos que el autor de dicho documento se ha guardado algún as en la manga, gracias al cual, sus resultados se ven gratamente mejorados. En nuestro caso,

tratamos también de implementar el perceptrón basándonos en las ideas de "profiling", esto es, entrenando los perceptrones con un conjunto pequeño de datos (entradas pequeñas, explicitadas para los SPEC 2000 (trazas completas)) guardando los conjuntos de pesos en archivos, y luego lanzando la ejecución, pero en lugar de inicializar los pesos de cada perceptrón con 1, inicializarlo con los pesos resultado del entrenamiento. En ningún caso este entrenamiento sirvió para mejorar los resultados.

Nos queda la duda de saber si los autores del documento ocultan algún tipo de información. De no ser así, aún siendo la idea de utilizar métodos neuronales a la predicción dinámica de saltos, el campo necesita aún de mucha investigación.

6. Bibliografía

[22] Daniel A. Jiménez, Calvin Lin *Neural Methods for Dynamic Branch Prediction*. ACM Transactions on Computer Systems, Vol.20, No4

[23] Pedro A. Calero, *Aprendizaje Automático*. Apuntes del seminario impartido en la facultad de Informática de la UCM, curso 2003-2004

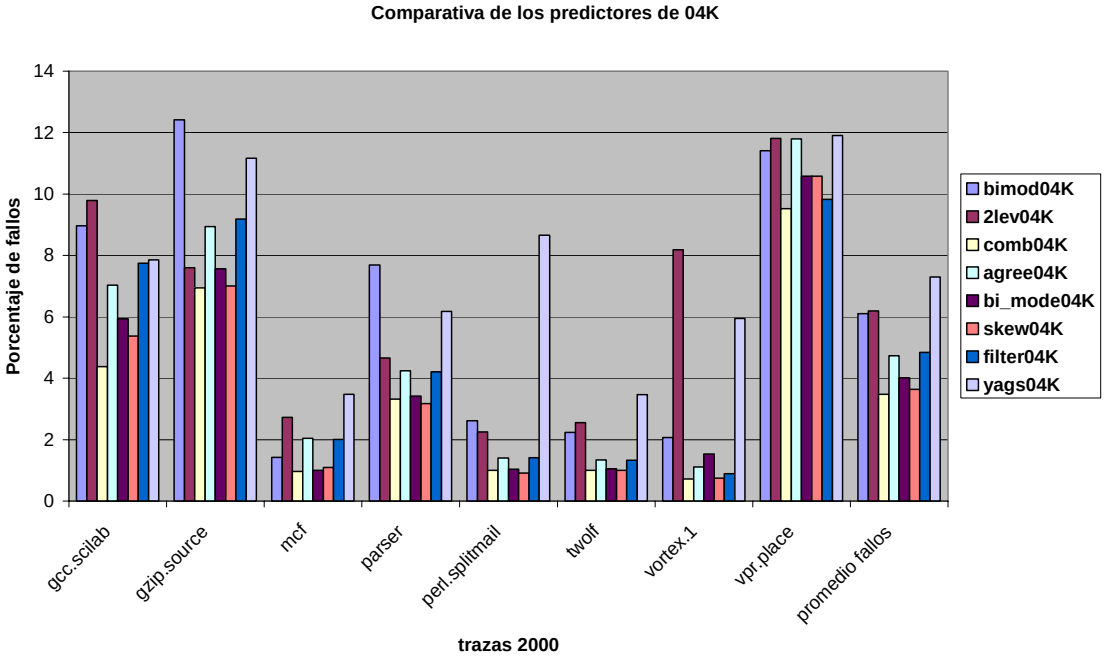
[24] Rosenblatt, F. 1962. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan, New York.

COMPARATIVA ENTRE TODOS LOS PREDICTORES

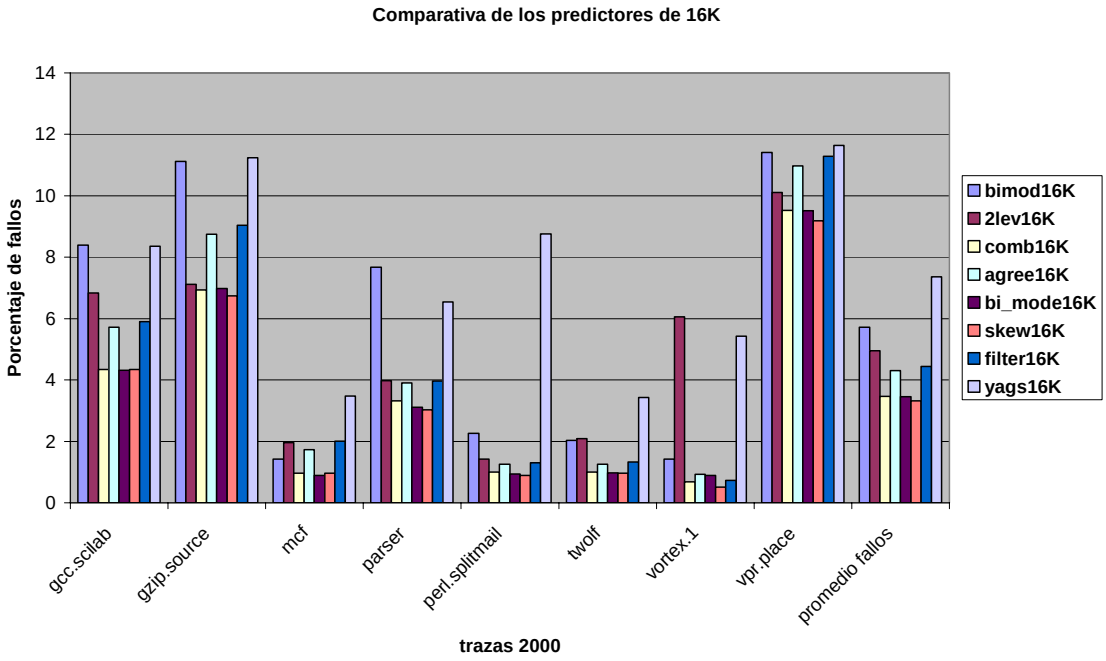
Benchmarks *"Trazas 2000":*

Para realizar una comparativa de conjunto podríamos ver cómo, aunque para cada test el mejor predictor puede variar, siempre hay un predictor que consigue mayor número de tasas mínimas de fallos. Tanto para el diagrama de predictores de 4Kb como para el de 16Kb, observamos que el mejor comportamiento se obtiene en el predictor Skew.

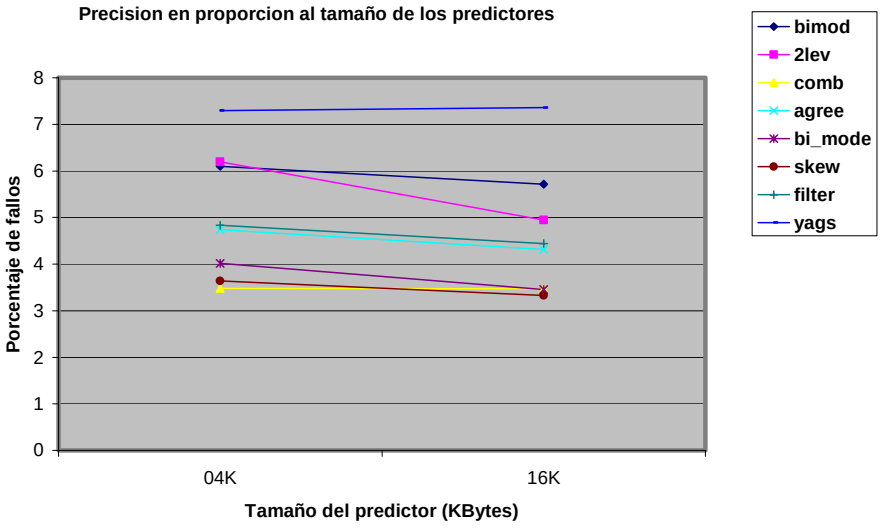
Para mostrar esto de una manera más ilustrativa, hemos incluido como en todas las gráficas anteriores, la comparativa de todos los predictores en cuanto al promedio de fallos para todos los tests. Así puede verse claramente lo que en el párrafo anterior comentábamos sobre el predictor Skew, ya que la barra que representa la tasa media de fallos de este predictor sobre todos los tests es menor que las demás.



Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas



En la siguiente gráfica que compara la efectividad según tamaños, vemos de modo conjunto las variaciones de todos los predictores que ya habíamos analizado de modo individual. Como habíamos visto, todos los predictores (a excepción del predictor Yags que mantiene la tasa de fallos prácticamente invariable) muestran una mejora al aumentar su tamaño de 4 a 16Kb.

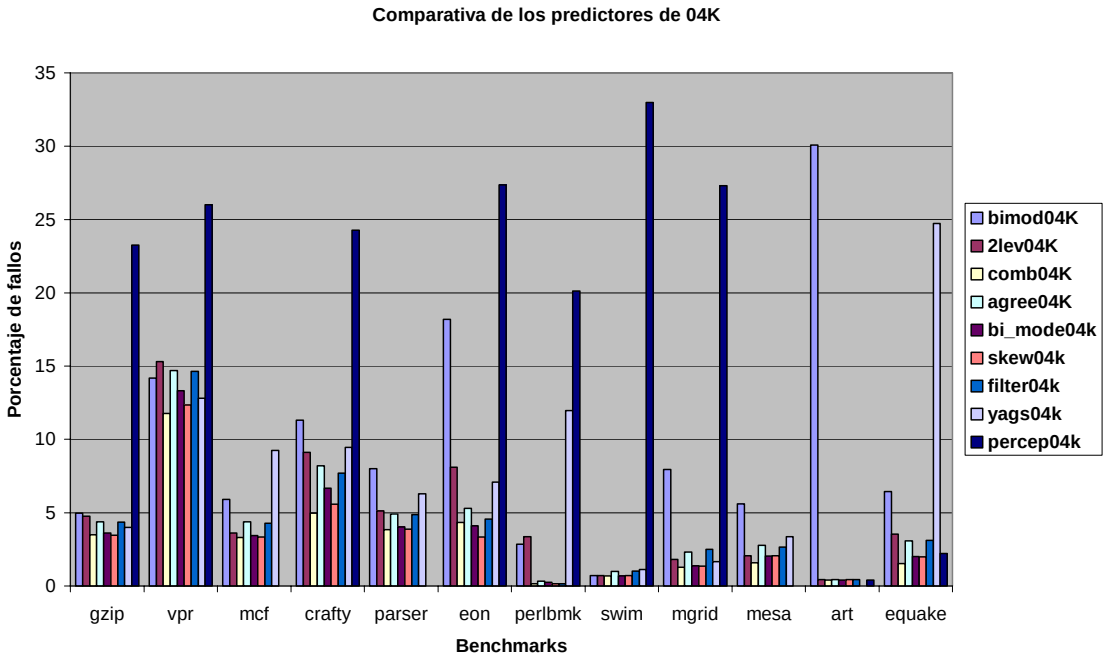


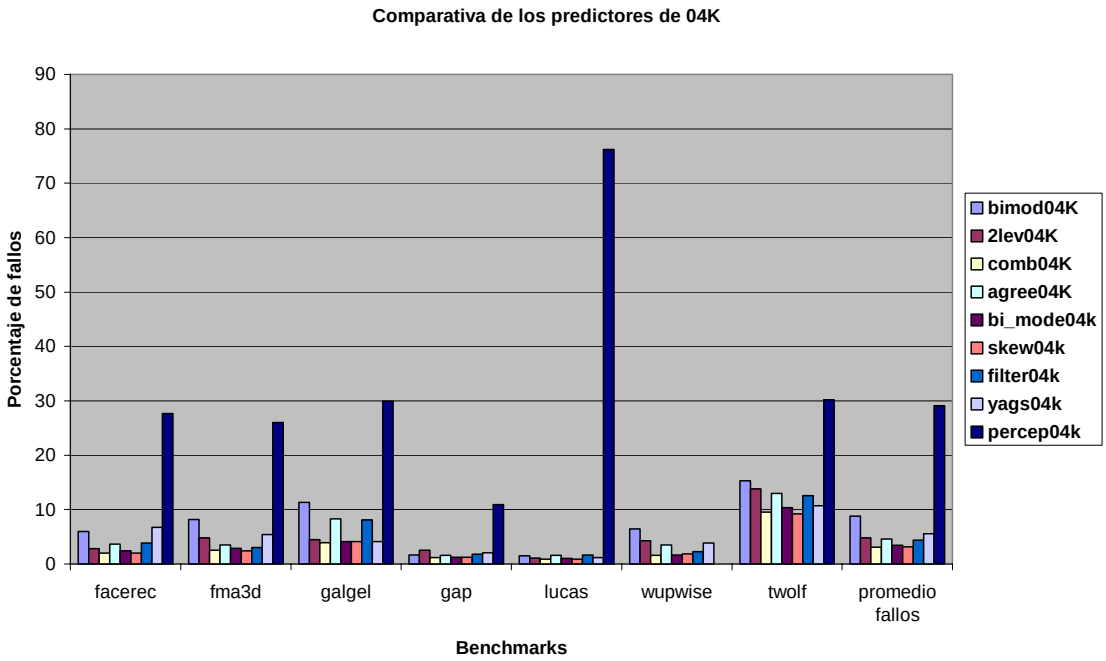
Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

Benchmarks *"Ejecuciones completas":*

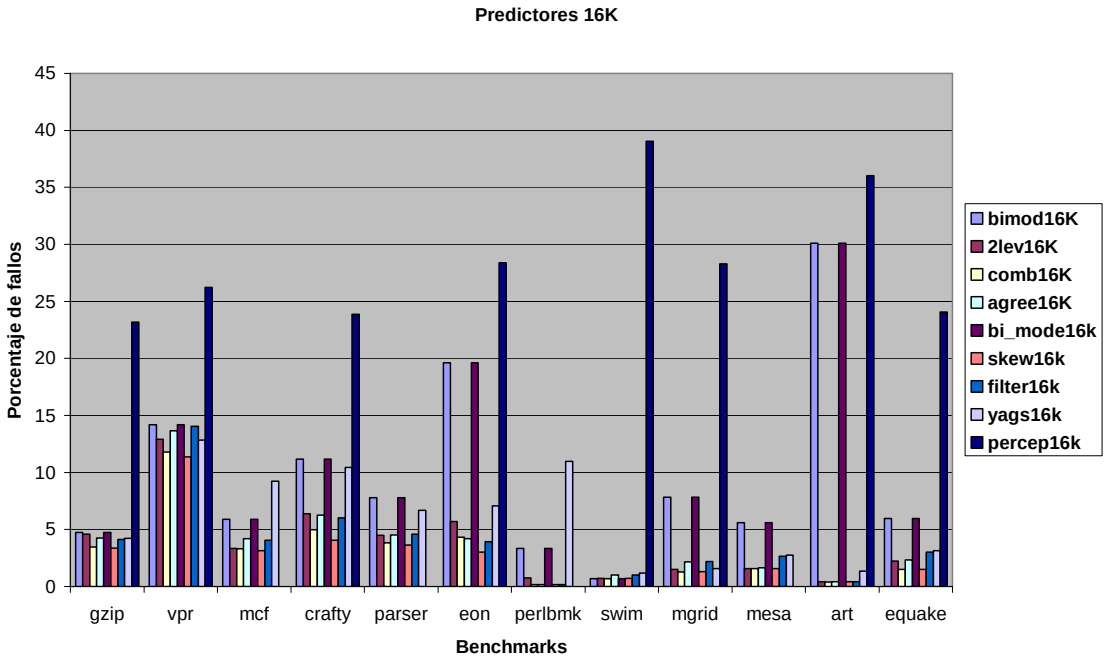
Para este segundo conjunto de tests en los que añadimos los resultados para el predictor perceptrón, podemos observar que sigue siendo el predictor Skew el que obtiene las mínimas tasas de fallos seguido, y en ocasiones muy igualado, por el predictor bi-mode.

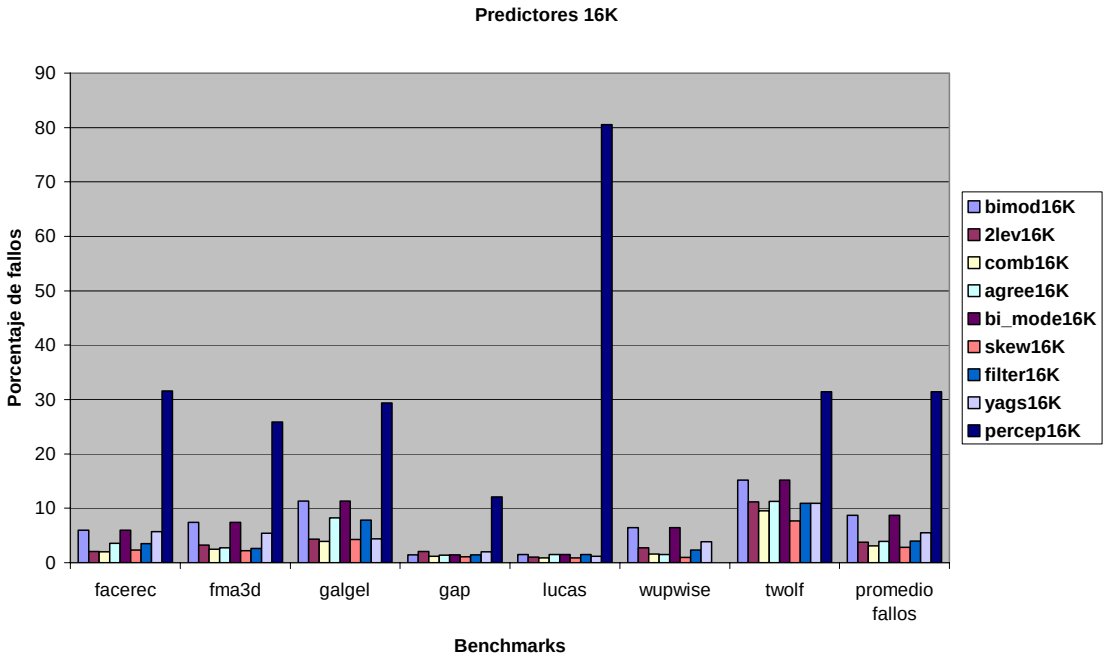
Esto es así para los dos tamaños de implementación estudiados, 4 y 16Kb, obteniendo mejores resultados no sólo en estos dos predictores sino para todos los demás, en la implementación que utiliza un tamaño mayor.





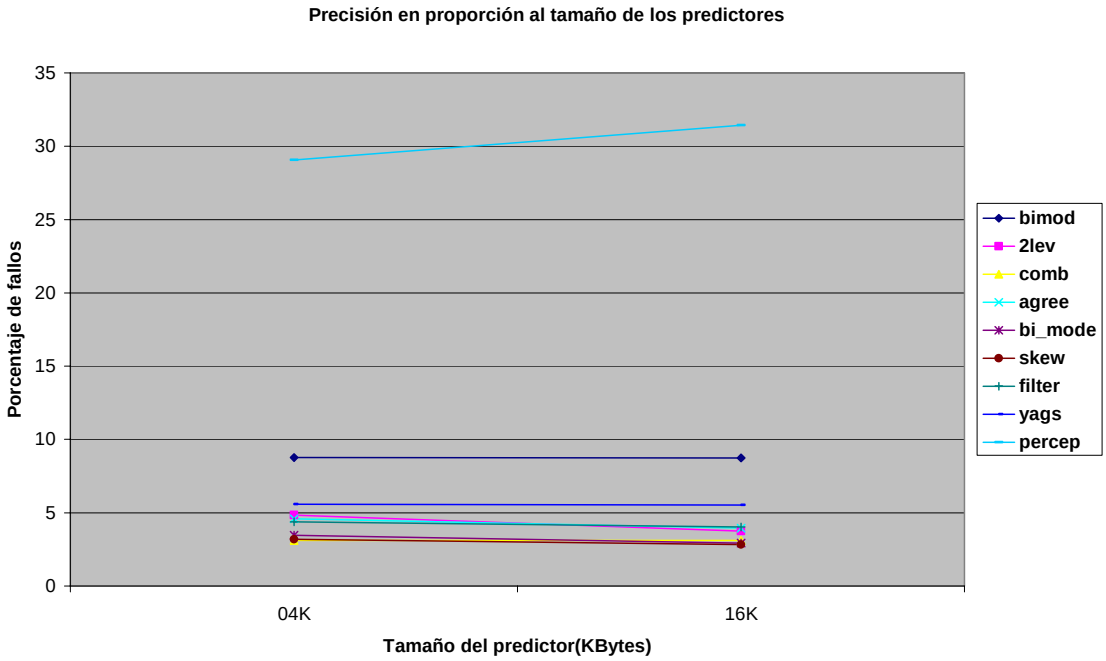
Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas





Esta gráfica muestra una vez más, como el comportamiento de los predictores mejora al aumentar su tamaño en la implementación, salvo el caso excepcional ya comentado en su momento del predictor perceptrón.

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas



Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

ANEXO I: RESULTADOS OBTENIDOS (BASE PARA LA GENERACIÓN DE LAS GRÁFICAS)

A continuación presentamos los resultados computados para los tests del conjunto "Trazas 2000" y del conjunto "Ejecuciones Completas" por separado para hacer constar en base a qué hemos generado las gráficas de resultados previamente comentadas.

- Tests de "Trazas 2000":

config	num_insn	num_branches	ac_dir	ac_dst	num_misses	%fallos	bench	promedio fallos
bimod04K	211937326	33339638	29608931	30349390	2990248	8,96904759	gcc.scilab	6,10331718
2lev04K	211937326	33339638	29336716	30075741	3263897	9,78983935	gcc.scilab	6,19696123
comb04K	211937326	33339638	31122148	31881172	1458466	4,37457059	gcc.scilab	3,47915183
agree04K	211937326	33339638	30246228	30997367	2342271	7,0254842	gcc.scilab	4,73676664
bi_mode04K	211937326	33339638	30611657	31361952	1977686	5,93193603	gcc.scilab	4,01282771
skew04K	211937326	33339638	30812278	31546499	1793139	5,37839973	gcc.scilab	3,64184644
filter04K	211937326	33339638	29824163	30756558	2583080	7,74777459	gcc.scilab	4,83549481
yags04K	211937326	33339638	29964676	30718419	2621219	7,86216995	gcc.scilab	7,29649827
bimod16K	211937326	33339638	29792534	30542569	2797069	8,38962019	gcc.scilab	5,71733128
2lev16K	211937326	33339638	30306809	31059376	2280262	6,83949238	gcc.scilab	4,94622937
comb16K	211937326	33339638	31132706	31892672	1446966	4,34007712	gcc.scilab	3,46938544
agree16K	211937326	33339638	30684654	31432720	1906918	5,71967218	gcc.scilab	4,30981801
bi_mode16K	211937326	33339638	31138429	31900741	1438897	4,3158747	gcc.scilab	3,45113687
skew16K	211937326	33339638	31151112	31890432	1449206	4,34679585	gcc.scilab	3,3262814
filter16K	211937326	33339638	30419190	31372010	1967628	5,90176774	gcc.scilab	4,44382512
yags16K	211937326	33339638	29791775	30551939	2787699	8,3615155	gcc.scilab	7,35757284
bimod04K	4294967295	600363585	525839922	525840067	74523518	12,4130643	gzip.source	
2lev04K	4294967295	600363585	554712739	554712917	45650668	7,60383693	gzip.source	
comb04K	4294967295	600363585	558679651	558679776	41683809	6,94309416	gzip.source	
agree04K	4294967295	600363585	546715785	546715932	53647653	8,93586059	gzip.source	
bi_mode04K	4294967295	600363585	554948659	554948802	45414783	7,56454657	gzip.source	
skew04K	4294967295	600363585	558323691	558323877	42039708	7,00237474	gzip.source	
filter04K	4294967295	600363585	545214526	545214697	55148888	9,1859149	gzip.source	
yags04K	4294967295	600363585	533326401	533326551	67037034	11,1660726	gzip.source	
bimod16K	4294967295	600363585	533597769	533597914	66765671	11,1208729	gzip.source	
2lev16K	4294967295	600363585	557666404	557666581	42697004	7,11185773	gzip.source	
comb16K	4294967295	600363585	558739047	558739172	41624413	6,93320082	gzip.source	
agree16K	4294967295	600363585	547883148	547883301	52480284	8,74141692	gzip.source	
bi_mode16K	4294967295	600363585	558475573	558475720	41887865	6,9770829	gzip.source	
skew16K	4294967295	600363585	559919045	559919244	40444341	6,73664126	gzip.source	
filter16K	4294967295	600363585	546134935	546135101	54228484	9,03260713	gzip.source	
yags16K	4294967295	600363585	532892811	532892959	67470626	11,2382942	gzip.source	
bimod04K	201013800	22981225	22652544	22652809	328416	1,4290622	mcf	
2lev04K	201013800	22981225	22354839	22355108	626117	2,72447182	mcf	
comb04K	201013800	22981225	22760509	22760732	220493	0,95944842	mcf	
agree04K	201013800	22981225	22510270	22510534	470691	2,04815453	mcf	
bi_mode04K	201013800	22981225	22752131	22752397	228828	0,99571716	mcf	
skew04K	201013800	22981225	22729141	22729430	251795	1,09565526	mcf	
filter04K	201013800	22981225	22520272	22520531	460694	2,0046538	mcf	
yags04K	201013800	22981225	22182453	22182715	798510	3,47461896	mcf	
bimod16K	201013800	22981225	22652543	22652807	328418	1,4290709	mcf	
2lev16K	201013800	22981225	22530877	22531145	450080	1,95846827	mcf	
comb16K	201013800	22981225	22760509	22760730	220495	0,95945712	mcf	
agree16K	201013800	22981225	22584381	22584645	396580	1,72566954	mcf	
bi_mode16K	201013800	22981225	22777465	22777731	203494	0,88547934	mcf	

skew16K	201013800	22981225	22760657	22760953	220272	0,95848676	mcf
filter16K	201013800	22981225	22518594	22518856	462369	2,01194236	mcf
yags16K	201013800	22981225	22182460	22182723	798502	3,47458414	mcf
bimod04K	718357124	113220858	104468637	104522606	8698252	7,68255263	parser
2lev04K	718357124	113220858	107893491	107947924	5272934	4,65721078	parser
comb04K	718357124	113220858	109409450	109463515	3757343	3,31859612	parser
agree04K	718357124	113220858	108359456	108413141	4807717	4,24631741	parser
bi_mode04K	718357124	113220858	109302321	109356378	3864480	3,41322268	parser
skew04K	718357124	113220858	109570091	109624198	3596660	3,17667616	parser
filter04K	718357124	113220858	108397456	108452112	4768746	4,21189707	parser
yags04K	718357124	113220858	106176032	106230129	6990729	6,17441797	parser
bimod16K	718357124	113220858	104470324	104524403	8696455	7,68096546	parser
2lev16K	718357124	113220858	108664773	108719201	4501657	3,97599619	parser
comb16K	718357124	113220858	109409684	109463736	3757122	3,31840093	parser
agree16K	718357124	113220858	108750509	108804219	4416639	3,90090579	parser
bi_mode16K	718357124	113220858	109636303	109690478	3530380	3,11813571	parser
skew16K	718357124	113220858	109743077	109797257	3423601	3,02382534	parser
filter16K	718357124	113220858	108682356	108737092	4483766	3,96019433	parser
yags16K	718357124	113220858	105762562	105816811	7404047	6,53947261	parser
bimod04K	206635427	37835490	36568293	36843973	991517	2,62060039	perl.splitmail
2lev04K	206635427	37835490	36709195	36984897	850593	2,24813528	perl.splitmail
comb04K	206635427	37835490	37181218	37456825	378665	1,0008196	perl.splitmail
agree04K	206635427	37835490	37032323	37307965	527525	1,39425973	perl.splitmail
bi_mode04K	206635427	37835490	37166330	37442023	393467	1,0399416	perl.splitmail
skew04K	206635427	37835490	37215230	37490938	344552	0,91065822	perl.splitmail
filter04K	206635427	37835490	37024341	37300014	535476	1,41527439	perl.splitmail
yags04K	206635427	37835490	34281065	34556760	3278730	8,66575271	perl.splitmail
bimod16K	206635427	37835490	36704354	36980034	855456	2,26098829	perl.splitmail
2lev16K	206635427	37835490	37019644	37295357	540133	1,42758294	perl.splitmail
comb16K	206635427	37835490	37181213	37456818	378672	1,0008381	perl.splitmail
agree16K	206635427	37835490	37087978	37363656	471834	1,24706724	perl.splitmail
bi_mode16K	206635427	37835490	37203246	37478942	356548	0,9423639	perl.splitmail
skew16K	206635427	37835490	37223772	37499484	336006	0,88807096	perl.splitmail
filter16K	206635427	37835490	37067772	37343445	492045	1,30048534	perl.splitmail
yags16K	206635427	37835490	34246785	34522482	3313008	8,75635019	perl.splitmail
bimod04K	201356087	34310757	33544360	33544600	766157	2,23299358	twolf
2lev04K	201356087	34310757	33435544	33435781	874976	2,55015067	twolf
comb04K	201356087	34310757	33968231	33968436	342321	0,99770751	twolf
agree04K	201356087	34310757	33851087	33851510	459247	1,33849276	twolf
bi_mode04K	201356087	34310757	33949747	33949990	360767	1,05146908	twolf
skew04K	201356087	34310757	33967066	33967314	343443	1,00097762	twolf
filter04K	201356087	34310757	33856204	33856629	454128	1,32357325	twolf
yags04K	201356087	34310757	33119386	33119626	1191131	3,47159639	twolf
bimod16K	201356087	34310757	33613817	33614057	696700	2,03055852	twolf
2lev16K	201356087	34310757	33594300	33594546	716211	2,08742407	twolf
comb16K	201356087	34310757	33968229	33968433	342324	0,99771626	twolf
agree16K	201356087	34310757	33882390	33882817	427940	1,24724733	twolf
bi_mode16K	201356087	34310757	33977125	33977369	333388	0,97167195	twolf
skew16K	201356087	34310757	33980818	33981070	329687	0,96088524	twolf
filter16K	201356087	34310757	33857284	33857711	453046	1,32041972	twolf
yags16K	201356087	34310757	33132569	33132809	1177948	3,43317403	twolf
bimod04K	308895839	49511110	48185663	48486468	1024642	2,06951935	vortex.1
2lev04K	308895839	49511110	45166965	45456488	4054622	8,18931751	vortex.1
comb04K	308895839	49511110	48847401	49155483	355627	0,71827717	vortex.1
agree04K	308895839	49511110	48452949	48964480	546630	1,10405523	vortex.1
bi_mode04K	308895839	49511110	48448504	48754137	756973	1,52889523	vortex.1
skew04K	308895839	49511110	48850100	49141087	370023	0,74735347	vortex.1
filter04K	308895839	49511110	48374034	49070884	440226	0,88914589	vortex.1
yags04K	308895839	49511110	46257179	46565515	2945595	5,94936167	vortex.1

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

bimod16K	308895839	49511110	48503014	48809502	701608	1,41707185	vortex.1
2lev16K	308895839	49511110	46217046	46512304	2998806	6,05683452	vortex.1
comb16K	308895839	49511110	48863293	49171441	339669	0,68604602	vortex.1
agree16K	308895839	49511110	48543298	49054340	456770	0,92256061	vortex.1
bi_mode16K	308895839	49511110	48759946	49069139	441971	0,89267035	vortex.1
skew16K	308895839	49511110	48965482	49258199	252911	0,51081666	vortex.1
filter16K	308895839	49511110	48445095	49149427	361683	0,73050877	vortex.1
yags16K	308895839	49511110	46516686	46826541	2684569	5,42215474	vortex.1
bimod04K	4294967295	587050434	520069623	520069756	66980678	11,4096974	vpr.place
2lev04K	4294967295	587050434	517703613	517703766	69346668	11,8127275	vpr.place
comb04K	4294967295	587050434	531159004	531159117	55891317	9,52070108	vpr.place
agree04K	4294967295	587050434	517769504	517769626	69280808	11,8015087	vpr.place
bi_mode04K	4294967295	587050434	524958609	524958736	62091698	10,5768933	vpr.place
skew04K	4294967295	587050434	529386210	529386370	57664064	9,82267633	vpr.place
filter04K	4294967295	587050434	517157679	517157826	69892608	11,9057246	vpr.place
yags04K	4294967295	587050434	518905508	518905644	68144790	11,6079958	vpr.place
bimod16K	4294967295	587050434	520070768	520070902	66979532	11,4095022	vpr.place
2lev16K	4294967295	587050434	527686688	527686844	59363590	10,1121789	vpr.place
comb16K	4294967295	587050434	531166954	531167065	55883369	9,51934719	vpr.place
agree16K	4294967295	587050434	522627364	522627493	64422941	10,9740045	vpr.place
bi_mode16K	4294967295	587050434	531246369	531246499	55803935	9,50581616	vpr.place
skew16K	4294967295	587050434	533131276	533131442	53918992	9,18472909	vpr.place
filter16K	4294967295	587050434	520756585	520756733	66293701	11,2926756	vpr.place
yags16K	4294967295	587050434	518746759	518746897	68303537	11,6350373	vpr.place

- Tests de “Ejecuciones Completas”:

ac_dst	num_misses	porcentaje fallos	config	bench	promedio fallos
61170373	3067287	4,77490463	2lev04K	gzip	4,83172012
61424125	2813535	4,37988401	agree04K	gzip	4,5748721
61047161	3190499	4,96671112	bimod04K	gzip	8,76824366
61908832	2328828	3,62533131	bi_mode04K	gzip	3,45304499
62000091	2237569	3,48326667	comb04K	gzip	3,10537348
61445675	2791985	4,34633671	filter04K	gzip	4,39151811
62016083	2221577	3,45837162	skew04K	gzip	3,17556135
61667155	2570505	4,00155454	yags04K	gzip	5,57128582
49300574	14937086	23,2528489	percep04K	gzip	29,0762417
61294925	2942735	4,58101214	2lev16K	gzip	3,74758731
61511041	2726619	4,2445802	agree16K	gzip	3,9423604
61177013	3060647	4,76456801	bimod16K	gzip	8,74730705
62018271	2219389	3,45496551	bi_mode16K	gzip	2,93158707
62001184	2236476	3,48156518	comb16K	gzip	3,10156399
61590294	2647366	4,12120554	filter16K	gzip	4,01471024
62067808	2169852	3,37785031	skew16K	gzip	2,83044833
61513644	2724016	4,24052806	yags16K	gzip	5,52173323
49342393	14895267	23,1877484	percep16K	gzip	31,4249622
143197875	25860503	15,2967888	2lev04K	vpr	
144190309	24868069	14,7097525	agree04K	vpr	
145058160	24000218	14,1964085	bimod04K	vpr	
146550849	22507529	13,3134656	bi_mode04K	vpr	
149143379	19914999	11,7799539	comb04K	vpr	
144283130	24775248	14,6548478	filter04K	vpr	

148167552	20890826	12,3571669	skew04K	vpr
147389980	21668398	12,8171098	yags04K	vpr
125085485	43972893	26,0104785	percep04K	vpr
147239932	21818446	12,905865	2lev16K	vpr
145976789	23081589	13,6530288	agree16K	vpr
145069939	23988439	14,1894411	bimod16K	vpr
149088994	19969384	11,8121233	bi_mode16K	vpr
149152183	19906195	11,7747462	comb16K	vpr
145309073	23749305	14,0479906	filter16K	vpr
149864869	19193509	11,3531842	skew16K	vpr
147378196	21680182	12,8240802	yags16K	vpr
124693439	44364939	26,2423782	percep16K	vpr
165619412	6231446	3,62607791	2lev04K	mcf
164316519	7534339	4,38423124	agree04K	mcf
161701650	10149208	5,90582329	bimod04K	mcf
165940883	5909975	3,43901396	bi_mode04K	mcf
166139643	5711215	3,32335553	comb04K	mcf
164475728	7375130	4,29158753	filter04K	mcf
166096909	5753949	3,34822245	skew04K	mcf
155966306	15884552	9,24321949	yags04K	mcf
0	0	#¡DIV/0!	percep04K	mcf
166136573	5714285	3,32514197	2lev16K	mcf
164668678	7182180	4,17930995	agree16K	mcf
161702958	10147900	5,90506217	bimod16K	mcf
166364897	5485961	3,19228025	bi_mode16K	mcf
166139656	5711202	3,32334797	comb16K	mcf
164886885	6963973	4,05233531	filter16K	mcf
166423365	5427493	3,15825773	skew16K	mcf
155982410	15868448	9,23384857	yags16K	mcf
0	0	#¡DIV/0!	percep16K	mcf
85567538	8592928	9,12583419	2lev04K	crafty
86447847	7712619	8,19093121	agree04K	crafty
83515668	10644798	11,3049547	bimod04K	crafty
87871042	6289424	6,67947416	bi_mode04K	crafty
89471783	4688683	4,97946028	comb04K	crafty
86923200	7237266	7,68609832	filter04K	crafty
88910052	5250414	5,57602805	skew04K	crafty
85271406	8889060	9,44033136	yags04K	crafty
71295680	22864786	24,2827876	percep04K	crafty
88133375	6027091	6,4008721	2lev16K	crafty
88270914	5889552	6,25480337	agree16K	crafty
83640047	10520419	11,1728621	bimod16K	crafty
89602214	4558252	4,84094036	bi_mode16K	crafty
89487277	4673189	4,96300539	comb16K	crafty
88491140	5669326	6,02091965	filter16K	crafty
90322047	3838419	4,07646559	skew16K	crafty
84327794	9832672	10,4424632	yags16K	crafty
71689212	22471254	23,86485	percep16K	crafty
681233558	36718170	5,11429509	2lev04K	parser
682661617	35290111	4,91538771	agree04K	parser
660466477	57485251	8,00684068	bimod04K	parser
689064643	28887085	4,0235414	bi_mode04K	parser
690390288	27561440	3,83889876	comb04K	parser
683022127	34929601	4,86517403	filter04K	parser

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

690155396	27796332	3,87161573	skew04K	parser
672754923	45196805	6,29524287	yags04K	parser
0	0	#¡DIV/0!	percep04K	parser
685816722	32135006	4,4759285	2lev16K	parser
685561749	32389979	4,51144245	agree16K	parser
661934145	56017583	7,80241635	bimod16K	parser
691533073	26418655	3,6797258	bi_mode16K	parser
690406740	27544988	3,83660724	comb16K	parser
685039550	32912178	4,584177	filter16K	parser
691787814	26163914	3,64424417	skew16K	parser
669980603	47971125	6,6816644	yags16K	parser
0	0	#¡DIV/0!	percep16K	parser
108764956	9600086	8,11057542	2lev04K	eon
112081285	6283760	5,30879704	agree04K	eon
96824290	21540752	18,1985759	bimod04K	eon
113505268	4859774	4,10575109	bi_mode04K	eon
113240004	5125041	4,32986022	comb04K	eon
112954362	5410683	4,57118316	filter04K	eon
114409535	3955510	3,34178895	skew04K	eon
109985649	8379393	7,07928022	yags04K	eon
85982129	32382913	27,358511	percep04K	eon
111637740	6727302	5,68352098	2lev16K	eon
113409833	4955212	4,18638121	agree16K	eon
95150059	23214983	19,61304	bimod16K	eon
114493649	3871393	3,27072329	bi_mode16K	eon
113240009	5125033	4,32985357	comb16K	eon
113712527	4652518	3,93065199	filter16K	eon
114795518	3569524	3,01569107	skew16K	eon
109991101	8373941	7,07467413	yags16K	eon
84770498	33594544	28,3821502	percep16K	eon
267006683	9316254	3,37150947	2lev04K	perlbmk
275375431	947506	0,34289806	agree04K	perlbmk
268473055	7849882	2,84083619	bimod04K	perlbmk
275629170	693767	0,25107109	bi_mode04K	perlbmk
275886361	436576	0,15799485	comb04K	perlbmk
275883681	439256	0,15896473	filter04K	perlbmk
275883833	439104	0,15890972	skew04K	perlbmk
243239381	33083556	11,9727868	yags04K	perlbmk
220697518	55625419	20,1305833	percep04K	perlbmk
274232512	2090425	0,75651519	2lev16K	perlbmk
275896321	426616	0,15439037	agree16K	perlbmk
267107128	9215809	3,33515889	bimod16K	perlbmk
275892043	430894	0,15593856	bi_mode16K	perlbmk
275886387	436550	0,15798544	comb16K	perlbmk
275892671	430266	0,15571129	filter16K	perlbmk
275888246	434691	0,15731267	skew16K	perlbmk
245989301	30333636	10,9776034	yags16K	perlbmk
0	0	#¡DIV/0!	percep16K	perlbmk
5511700	39429	0,71028794	2lev04K	swim
5495792	55337	0,99686028	agree04K	swim
5512116	39013	0,70279397	bimod04K	swim
5512612	38517	0,69385885	bi_mode04K	swim
5513067	38062	0,68566232	comb04K	swim
5495133	55996	1,00873174	filter04K	swim

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

5511427	39702	0,71520586	skew04K	swim
5489142	61987	1,11665573	yags04K	swim
3720035	1830925	32,9839343	percep04K	swim
5511891	39238	0,7068472	2lev16K	swim
5495580	55549	1,00067932	agree16K	swim
5512269	38860	0,70003778	bimod16K	swim
5512948	38181	0,68780603	bi_mode16K	swim
5513075	38054	0,68551821	comb16K	swim
5495591	55538	1,00048116	filter16K	swim
5511421	39708	0,71531395	skew16K	swim
5485140	65989	1,18874917	yags16K	swim
3384500	2166460	39,0285644	percep16K	swim
892856	16462	1,81036777	2lev04K	mgrid
888149	21169	2,32800846	agree04K	mgrid
837077	72241	7,94452546	bimod04K	mgrid
896829	12489	1,37344691	bi_mode04K	mgrid
897750	11568	1,27216221	comb04K	mgrid
886726	22592	2,48449937	filter04K	mgrid
897086	12232	1,34518397	skew04K	mgrid
894356	14962	1,64540898	yags04K	mgrid
660954	248364	27,3132172	percep04K	mgrid
895477	13841	1,52212977	2lev16K	mgrid
889761	19557	2,15073275	agree16K	mgrid
838139	71179	7,82773463	bimod16K	mgrid
898457	10861	1,19441164	bi_mode16K	mgrid
897753	11565	1,2718323	comb16K	mgrid
889252	20066	2,20670876	filter16K	mgrid
897313	12005	1,32022021	skew16K	mgrid
895165	14153	1,5564412	yags16K	mgrid
652078	257240	28,2893333	percep16K	mgrid
257699327	5455463	2,0731004	2lev04K	mesa
255865142	7289648	2,77009892	agree04K	mesa
248395288	14759502	5,60867693	bimod04K	mesa
257767345	5387445	2,04725325	bi_mode04K	mesa
259014259	4140531	1,57342034	comb04K	mesa
256188650	6966140	2,64716443	filter04K	mesa
257701843	5452947	2,07214431	skew04K	mesa
254302517	8852273	3,36390343	yags04K	mesa
0	0	#¡DIV/0!	percep04K	mesa
259013036	4141754	1,57388509	2lev16K	mesa
258814173	4340617	1,64945392	agree16K	mesa
248395326	14759464	5,60866249	bimod16K	mesa
259078911	4075879	1,54885229	bi_mode16K	mesa
259014256	4140534	1,57342148	comb16K	mesa
256191692	6963098	2,64600846	filter16K	mesa
259013255	4141535	1,57380187	skew16K	mesa
255941342	7213448	2,74114258	yags16K	mesa
0	0	#¡DIV/0!	percep16K	mesa
134893765	575991	0,42518051	2lev04K	art
134897958	571798	0,42208535	agree04K	art
94724532	40745224	30,0769893	bimod04K	art
134917175	552581	0,4078999	bi_mode04K	art
134927389	542367	0,40036021	comb04K	art
134896229	573527	0,42336165	filter04K	art

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

134899642	570114	0,42084227	skew04K	art
133710034	1759722	1,29897776	yags04K	art
0	0	#¡DIV/0!	percep04K	art
134901191	568565	0,41969884	2lev16K	art
134913211	556545	0,41082601	agree16K	art
94714551	40755205	30,084357	bimod16K	art
134924677	545079	0,40236213	bi_mode16K	art
134927387	542369	0,40036169	comb16K	art
134898546	571210	0,42165131	filter16K	art
134912378	557378	0,41144091	skew16K	art
133654563	1815193	1,33992491	yags16K	art
86676177	48793579	36,0180607	percep16K	art
144459124	5320233	3,5520469	2lev04K	equake
145175020	4604337	3,07407983	agree04K	equake
140126639	9652718	6,44462508	bimod04K	equake
146783287	2996070	2,00032238	bi_mode04K	equake
147500932	2278425	1,5211876	comb04K	equake
145117564	4661793	3,11244025	filter04K	equake
146821516	2957841	1,97479884	skew04K	equake
144447977	5331380	3,55948918	yags04K	equake
112741086	37038271	24,7285552	percep04K	equake
146460677	3318680	2,21571254	2lev16K	equake
146283665	3495692	2,33389438	agree16K	equake
140871464	8907893	5,9473436	bimod16K	equake
147614098	2165259	1,44563246	bi_mode16K	equake
147500947	2278410	1,52117758	comb16K	equake
145269936	4509421	3,01070928	filter16K	equake
147543624	2235733	1,49268434	skew16K	equake
145073636	4705721	3,14176873	yags16K	equake
113731782	36047575	24,0671183	percep16K	equake
29507930	844962	2,78379405	2lev04K	facerec
29248615	1104277	3,63812779	agree04K	facerec
28547818	1805074	5,94695886	bimod04K	facerec
29632045	720847	2,37488737	bi_mode04K	facerec
29740578	612314	2,01731683	comb04K	facerec
29182216	1170676	3,85688454	filter04K	facerec
29742180	610712	2,01203892	skew04K	facerec
28315348	2037544	6,71284964	yags04K	facerec
21954361	8399121	27,6710296	percep04K	facerec
29721751	631141	2,07934387	2lev16K	facerec
29276796	1076096	3,54528326	agree16K	facerec
28548245	1804647	5,94555207	bimod16K	facerec
29739467	613425	2,02097711	bi_mode16K	facerec
29740596	612296	2,01725753	comb16K	facerec
29285530	1067362	3,51650841	filter16K	facerec
29637378	715514	2,35731739	skew16K	facerec
28613227	1739665	5,73146374	yags16K	facerec
20758670	9594812	31,6102515	percep16K	facerec
87692173	4441890	4,82111594	2lev04K	fma3d
88914188	3219875	3,49477153	agree04K	fma3d
84582479	7551584	8,19629978	bimod04K	fma3d
89468112	2665951	2,89355632	bi_mode04K	fma3d
89785845	2348218	2,54869689	comb04K	fma3d
89372370	2761693	2,99747228	filter04K	fma3d

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

89889254	2244809	2,43645936	skew04K	fma3d
87166187	4967876	5,39200795	yags04K	fma3d
68132251	24001917	26,0510487	percep04K	fma3d
89185594	2948469	3,20019426	2lev16K	fma3d
89609982	2524081	2,73957418	agree16K	fma3d
85303158	6830905	7,41409287	bimod16K	fma3d
89993036	2141027	2,32381698	bi_mode16K	fma3d
89825899	2308164	2,50522329	comb16K	fma3d
89742094	2391969	2,59618313	filter16K	fma3d
90083083	2050980	2,22608223	skew16K	fma3d
87168324	4965739	5,3896885	yags16K	fma3d
68288064	23846104	25,8819334	percep16K	fma3d
21564031	1010545	4,47647389	2lev04K	galgel
20692779	1881797	8,33591293	agree04K	galgel
20021859	2552717	11,3079289	bimod04K	galgel
21640430	934146	4,1380445	bi_mode04K	galgel
21685766	888810	3,93721681	comb04K	galgel
20738696	1835880	8,13251155	filter04K	galgel
21649909	924667	4,09605478	skew04K	galgel
21639589	934987	4,14176993	yags04K	galgel
15811324	6763252	29,959597	percep04K	galgel
21597691	976885	4,3273681	2lev16K	galgel
20712920	1861656	8,24669309	agree16K	galgel
20022272	2552304	11,3060994	bimod16K	galgel
21653608	920968	4,07966909	bi_mode16K	galgel
21685870	888706	3,93675611	comb16K	galgel
20809222	1765354	7,82009815	filter16K	galgel
21620817	953759	4,22492542	skew16K	galgel
21577609	996967	4,41632658	yags16K	galgel
15944812	6629764	29,368277	percep16K	galgel
3067081	80230	2,54916022	2lev04K	gap
3097486	49825	1,58309744	agree04K	gap
3095122	52189	1,65820918	bimod04K	gap
3107978	39333	1,2497335	bi_mode04K	gap
3111476	35835	1,13859101	comb04K	gap
3092115	55196	1,75375106	filter04K	gap
3109044	38267	1,21586332	skew04K	gap
3083539	63772	2,02623764	yags04K	gap
2803101	344210	10,9366377	percep04K	gap
3081725	65586	2,08387414	2lev16K	gap
3103949	43362	1,37774754	agree16K	gap
3101308	46003	1,46166045	bimod16K	gap
3112693	34618	1,09992308	bi_mode16K	gap
3111494	35817	1,13801909	comb16K	gap
3102459	44852	1,42508954	filter16K	gap
3112099	35212	1,11879633	skew16K	gap
3084392	62919	1,99913513	yags16K	gap
2768079	379232	12,0493971	percep16K	gap
3111921	34013	1,08117335	2lev04K	lucas
3095993	49941	1,58747768	agree04K	lucas
3097914	48020	1,52641473	bimod04K	lucas
3114497	31437	0,99928988	bi_mode04K	lucas
3117849	28085	0,89273964	comb04K	lucas
3094774	51160	1,62622611	filter04K	lucas

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

3116991	28943	0,92001294	skew04K	lucas
3108176	37758	1,2002159	yags04K	lucas
748522	2397412	76,2066846	percep04K	lucas
3113949	31985	1,01670919	2lev16K	lucas
3099394	46540	1,47936988	agree16K	lucas
3098709	47225	1,50114402	bimod16K	lucas
3117740	28194	0,89620443	bi_mode16K	lucas
3117857	28077	0,89248535	comb16K	lucas
3098376	47558	1,51172911	filter16K	lucas
3117073	28861	0,9174064	skew16K	lucas
3108458	37476	1,19125195	yags16K	lucas
612346	2533588	80,5353196	percep16K	lucas
475252823	21201967	4,27067427	2lev04K	wupwise
479241973	17212817	3,46714693	agree04K	wupwise
464541456	31913334	6,42824576	bimod04K	wupwise
488411288	8043502	1,62018821	bi_mode04K	wupwise
488556343	7898447	1,59097005	comb04K	wupwise
485162205	11292585	2,27464519	filter04K	wupwise
487387243	9067547	1,82645977	skew04K	wupwise
477431339	19023451	3,83185969	yags04K	wupwise
0	0	#¡DIV/0!	percep04K	wupwise
482939141	13515649	2,72243299	2lev16K	wupwise
489010360	7444430	1,49951821	agree16K	wupwise
464542290	31912500	6,42807777	bimod16K	wupwise
491487701	4967089	1,00051185	bi_mode16K	wupwise
488556348	7898442	1,59096904	comb16K	wupwise
484903294	11551496	2,32679717	filter16K	wupwise
491742119	4712671	0,94926489	skew16K	wupwise
477481465	18973325	3,8217629	yags16K	wupwise
0	0	#¡DIV/0!	percep16K	wupwise
98013686	15729977	13,8293216	2lev04K	twolf
98964919	14778737	12,9930209	agree04K	twolf
96301287	17442376	15,3348112	bimod04K	twolf
101946483	11797180	10,3717251	bi_mode04K	twolf
102902775	10840888	9,53098196	comb04K	twolf
99472287	14271376	12,5469636	filter04K	twolf
103292329	10451334	9,18849782	skew04K	twolf
101555427	12188236	10,7155297	yags04K	twolf
79414143	34329506	30,1814706	percep04K	twolf
100996289	12747374	11,207107	2lev16K	twolf
100905258	12838405	11,2871387	agree16K	twolf
96464268	17279395	15,1915232	bimod16K	twolf
103969340	9774323	8,59329016	bi_mode16K	twolf
102904366	10839297	9,5295832	comb16K	twolf
101363190	12380473	10,8845387	filter16K	twolf
104998756	8744907	7,68825864	skew16K	twolf
101322384	12421279	10,9204141	yags16K	twolf
78000753	35742910	31,4240891	percep16K	twolf

Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

ANEXO II: CÓDIGO DE SIMPLESCALAR REALIZADO EN EL PROYECTO

A continuación vamos a mostrar el código final que hemos realizado para llevar a cabo todas las pruebas y tests anteriores. Los archivos de Simplescalar que hemos modificado son `sim-bpred.c`, bpred.h y `bpred.c`.

1. Contenido de SIM-BPRED.C

A continuación tenemos el contenido del archivo `sim-bpred.c` que es el archivo que contiene el módulo que hace funcionar el predictor de saltos dentro de la arquitectura Simplescalar:

```
/* sim-bpred.c - sample branch predictor simulator implementation */

/* SimpleScalar(TM) Tool Suite
 * Copyright (C) 1994-2003 by Todd M. Austin, Ph.D. and SimpleScalar,
LLC.
 * All Rights Reserved.
 *
 * THIS IS A LEGAL DOCUMENT, BY USING SIMPLESCALAR,
 * YOU ARE AGREEING TO THESE TERMS AND CONDITIONS.
 *
 * No portion of this work may be used by any commercial entity, or
for any
 * commercial purpose, without the prior, written permission of
SimpleScalar,
 * LLC (info@simplescalar.com). Nonprofit and noncommercial use is
permitted
 * as described below.
 *
 * 1. SimpleScalar is provided AS IS, with no warranty of any kind,
express
 * or implied. The user of the program accepts full responsibility for
the
 * application of the program and the use of any results.
 *
 * 2. Nonprofit and noncommercial use is encouraged. SimpleScalar may
be
 * downloaded, compiled, executed, copied, and modified solely for
nonprofit,
 * educational, noncommercial research, and noncommercial scholarship
 * purposes provided that this notice in its entirety accompanies all
copies.
 * Copies of the modified software can be delivered to persons who use
it
 * solely for nonprofit, educational, noncommercial research, and
 * noncommercial scholarship purposes provided that this notice in its
 * entirety accompanies all copies.
 *
 * 3. ALL COMMERCIAL USE, AND ALL USE BY FOR PROFIT ENTITIES, IS
EXPRESSLY
 * PROHIBITED WITHOUT A LICENSE FROM SIMPLESCALAR, LLC
(info@simplescalar.com).
 *
 * 4. No nonprofit user may place any restrictions on the use of this
software,
 * including as modified by the user, by any other authorized user.
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
 *
 * 5. Noncommercial and nonprofit users may distribute copies of
SimpleScalar
 * in compiled or executable form as set forth in Section 2, provided
that
 * either: (A) it is accompanied by the corresponding machine-readable
source
 * code, or (B) it is accompanied by a written offer, with no time
limit, to
 * give anyone a machine-readable copy of the corresponding source
code in
 * return for reimbursement of the cost of distribution. This written
offer
 * must permit verbatim duplication by anyone, or (C) it is
distributed by
 * someone who received only the executable form, and is accompanied
by a
 * copy of the written offer of source code.
 *
 * 6. SimpleScalar was developed by Todd M. Austin, Ph.D. The tool
suite is
 * currently maintained by SimpleScalar LLC (info@simplescalar.com).
US Mail:
 * 2395 Timbercrest Court, Ann Arbor, MI 48105.
 *
 * Copyright (C) 1994-2003 by Todd M. Austin, Ph.D. and SimpleScalar,
LLC.
 */


#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "host.h"
#include "misc.h"
#include "machine.h"
#include "regs.h"
#include "memory.h"
#include "loader.h"
#include "syscall.h"
#include "dlite.h"
#include "options.h"
#include "stats.h"
#include "bpred.h"
#include "sim.h"

/*
 * This file implements a branch predictor analyzer.
 */

/* simulated registers */
static struct regs_t regs;

/* simulated memory */
static struct mem_t *mem = NULL;

/* maximum number of inst's to execute */
static unsigned int max_insts;
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
/* branch predictor type
{nottaken|taken|perfect|bimod|2lev|agree|bi_mode|
                          skew|filter|yags|perceptron} */
static char *pred_type;

/* bimodal predictor config (<table_size>) */
static int bimod_nelt = 1;
static int bimod_config[1] =
  { /* bimod tbl size */2048 };

/* 2-level predictor config (<l1size> <l2size> <hist_size> <xor>) */
static int twolev_nelt = 4;
static int twolev_config[4] =
  { /* l1size */1, /* l2size */1024, /* hist */8, /* xor */FALSE};

/* combining predictor config (<meta_table_size>) */
static int comb_nelt = 1;
static int comb_config[1] =
  { /* meta_table_size */1024 };

/* agree predictor config (<table_size>) */
static int agree_nelt = 3;
static int agree_config[3] =
  { /* l1size */1, /* l2size */4096, /* hist */12 };

/* bi-mode predictor config (< PHT tables size>, <history width>) */
static int bi_mode_nelt = 2;
static int bi_mode_config[2] =
  {/* PHT tables size */4096,/* history width */12 };

/* skew predictor config (<PHT tables size>, <history width>) */
static int skew_nelt = 2;
static int skew_config[2] =
  { /* l2size */4096, /* hist */12 };

/*filter predictor config (<PHT tables size>, <history width>)*/
static int filter_nelt = 2;
static int filter_config[2] =
  { /* l2size */4096, /* hist */12 };

/*YAGS predictor config (<PHT tables size>, <history width>, <TAG
size>,
                         <cache size>)*/
static int yags_nelt = 5;
static int yags_config[5] =
  { /* l2size */4096, /* hist */12, /* tagsize */6, /* cachesets
*/512,
    /* cacheassoc */4 };

/* perceptron predictor config (<l1size> <l2size> <hist_size> <xor>
                                <threshold>) */
static int percep_nelt = 5;
static int percep_config[5] =
  { /* l1size */1, /* l2size (number of perceptrons) */4096, /* hist
*/12,
    /* xor */FALSE, /* threshold */ 37 };

/* return address stack (RAS) size */
static int ras_size = 8;
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
/* BTB predictor config (<num_sets> <associativity>) */
static int btb_nelt = 2;
static int btb_config[2] =
  { /* nsets */512, /* assoc */4 };

/* branch predictor */
static struct bpred_t *pred;

/* track number of insn and refs */
static counter_t sim_num_refs = 0;

/* total number of branches executed */
static counter_t sim_num_branches = 0;

/* Definicion de nuestra nueva estadistica */
static int fallos;
static int sim_dif = 0;


/* register simulator-specific options */
void
sim_reg_options(struct opt_odb_t *odb)
{
  opt_reg_header(odb,
"sim-bpred: This simulator implements a branch predictor analyzer.\n"
             );

  /* branch predictor options */
  opt_reg_note(odb,
"  Branch predictor configuration examples for 2-level predictor:\n"
"    Configurations:   N, M, W, X\n"
"      N   # entries in first level (# of shift register(s))\n"
"      W   width of shift register(s)\n"
"      M   # entries in 2nd level (# of counters, or other FSM)\n"
"      X   (yes-1/no-0) xor history and address for 2nd level index\n"
"    Sample predictors:\n"
"      GAg     : 1, W, 2^W, 0\n"
"      GAp     : 1, W, M (M > 2^W), 0\n"
"      PAg     : N, W, 2^W, 0\n"
"      PAp     : N, W, M (M == 2^(N+W)), 0\n"
"      gshare  : 1, W, 2^W, 1\n"
"  Predictor `comb' combines a bimodal and a 2-level predictor.\n"
               );

  /* instruction limit */
  opt_reg_uint(odb, "-max:inst", "maximum number of inst's to
execute",
             &max_insts, /* default */0,
             /* print */TRUE, /* format */NULL);

  opt_reg_string(odb, "-bpred",
             "branch predictor type
{nottaken|taken|bimod|2lev|comb|agree"+
                 "|bi_mode|skew|filter|yags|perceptron}",
                 &pred_type, /* default */"bimod",
                 /* print */TRUE, /* format */NULL);

  opt_reg_int_list(odb, "-bpred:bimod",
              "bimodal predictor config (<table size>)",
              bimod_config, bimod_nelt, &bimod_nelt,
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
                /* default */bimod_config,
                /* print */TRUE, /* format */NULL, /* !accrue */FALSE);

  opt_reg_int_list(odb, "-bpred:2lev",
                   "2-level predictor config "
                "(<l1size> <l2size> <hist_size> <xor>)",
                    twolev_config, twolev_nelt, &twolev_nelt,
                /* default */twolev_config,
                    /* print */TRUE, /* format */NULL, /* !accrue
*/FALSE);

  opt_reg_int_list(odb, "-bpred:comb",
                "combining predictor config (<meta_table_size>)",
                comb_config, comb_nelt, &comb_nelt,
                /* default */comb_config,
                /* print */TRUE, /* format */NULL, /* !accrue */FALSE);

  opt_reg_int_list(odb, "-bpred:agree", "agree predictor config" +
                                        "(<l1size> <l2size>
<hist_size>)",
                    agree_config, agree_nelt, &agree_nelt,
                /* default */agree_config,
                    /* print */TRUE, /* format */NULL, /* !accrue
*/FALSE);

  opt_reg_int_list(odb, "-bpred:bi_mode",
                   "bi_mode predictor config "
                "(<PHT tables size> <hist_size>)",
                    bi_mode_config, bi_mode_nelt, &bi_mode_nelt,
                /* default */bi_mode_config,
                    /* print */TRUE, /* format */NULL, /* !accrue
*/FALSE);

  opt_reg_int_list(odb, "-bpred:skew",
                   "Skewed branch predictor config "
                "(<l2size> <hist_size>)",
                    skew_config, skew_nelt, &skew_nelt,
                /* default */skew_config,
                    /* print */TRUE, /* format */NULL, /* !accrue
*/FALSE);

  opt_reg_int_list(odb, "-bpred:filter",
                   "Filter predictor config "
                "(<l2size> <hist_size>)",
                    filter_config, filter_nelt, &filter_nelt,
                /* default */filter_config,
                    /* print */TRUE, /* format */NULL, /* !accrue
*/FALSE);

  opt_reg_int_list(odb, "-bpred:yags",
                   "YAGS predictor config "
                "(<l2size> <hist_size> <TAG_size> <cache_sets>"+
                    " <cache_assoc>)",
                    yags_config, yags_nelt, &yags_nelt,
                /* default */yags_config,
                    /* print */TRUE, /* format */NULL, /* !accrue
*/FALSE);

  opt_reg_int_list(odb, "-bpred:perceptron",
                   "perceptron predictor config "
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
                "(<l1size> <l2size> <hist_size> <xor> <threshold>)",
                    percep_config, percep_nelt, &percep_nelt,
                /* default */percep_config,
                    /* print */TRUE, /* format */NULL, /* !accrue
*/FALSE);

  opt_reg_int(odb, "-bpred:ras",
              "return address stack size (0 for no return stack)",
              &ras_size, /* default */ras_size,
              /* print */TRUE, /* format */NULL);

  opt_reg_int_list(odb, "-bpred:btb",
                "BTB config (<num_sets> <associativity>)",
                btb_config, btb_nelt, &btb_nelt,
                /* default */btb_config,
                /* print */TRUE, /* format */NULL, /* !accrue */FALSE);
}

/* check simulator-specific option values */
void
sim_check_options(struct opt_odb_t *odb, int argc, char **argv)
{
  if (!mystricmp(pred_type, "taken"))
    {
      /* static predictor, not taken */
      pred = bpred_create(BPredTaken, 0, 0, 0, 0, 0, 0, 0, 0, 0);
    }
  else if (!mystricmp(pred_type, "nottaken"))
    {
      /* static predictor, taken */
      pred = bpred_create(BPredNotTaken, 0, 0, 0, 0, 0, 0, 0, 0, 0);
    }
  else if (!mystricmp(pred_type, "bimod"))
    {
      if (bimod_nelt != 1)
      fatal("bad bimod predictor config (<table_size>)");
      if (btb_nelt != 2)
      fatal("bad btb config (<num_sets> <associativity>)");

      /* bimodal predictor, bpred_create() checks BTB_SIZE */
      pred = bpred_create(BPred2bit,
                    /* bimod table size */bimod_config[0],
                    /* 2lev l1 size */0,
                    /* 2lev l2 size */0,
                    /* meta table size */0,
                    /* history reg size */0,
                    /* history xor address */0,
                    /* btb sets */btb_config[0],
                    /* btb assoc */btb_config[1],
                    /* ret-addr stack size */ras_size);
    }
  else if (!mystricmp(pred_type, "2lev"))
    {
      /* 2-level adaptive predictor, bpred_create() checks args */
      if (twolev_nelt != 4)
      fatal("bad 2-level pred config (<l1size> <l2size> <hist_size>
<xor>)");
      if (btb_nelt != 2)
      fatal("bad btb config (<num_sets> <associativity>)");
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
      pred = bpred_create(BPred2Level,
                  /* bimod table size */0,
                  /* 2lev l1 size */twolev_config[0],
                  /* 2lev l2 size */twolev_config[1],
                  /* meta table size */0,
                  /* history reg size */twolev_config[2],
                  /* history xor address */twolev_config[3],
                  /* btb sets */btb_config[0],
                  /* btb assoc */btb_config[1],
                  /* ret-addr stack size */ras_size);
    }
  else if (!mystricmp(pred_type, "comb"))
    {
      /* combining predictor, bpred_create() checks args */
      if (twolev_nelt != 4)
      fatal("bad 2-level pred config (<l1size> <l2size> <hist_size>
<xor>)");
      if (bimod_nelt != 1)
      fatal("bad bimod predictor config (<table_size>)");
      if (comb_nelt != 1)
      fatal("bad combining predictor config (<meta_table_size>)");
      if (btb_nelt != 2)
      fatal("bad btb config (<num_sets> <associativity>)");

      pred = bpred_create(BPredComb,
                  /* bimod table size */bimod_config[0],
                  /* l1 size */twolev_config[0],
                  /* l2 size */twolev_config[1],
                  /* meta table size */comb_config[0],
                  /* history reg size */twolev_config[2],
                  /* history xor address */twolev_config[3],
                  /* btb sets */btb_config[0],
                  /* btb assoc */btb_config[1],
                  /* ret-addr stack size */ras_size);
    }
  else if (!mystricmp(pred_type, "agree"))
    {
      if (agree_nelt != 3)
      fatal("bad agree pred config (<l1size> <l2size> <hist_size>)");
      if (btb_nelt != 2)
      fatal("bad btb config (<num_sets> <associativity>)");

      /* bimodal predictor, bpred_create() checks BTB_SIZE */
      pred = bpred_create(BPredAgree,
                  /* bimod table size */0,
                  /* 2lev l1 size */agree_config[0],
                  /* 2lev l2 size */agree_config[1],
                  /* meta table size */0,
                  /* history reg size */agree_config[2],
                  /* history xor address */1,
                  /* btb sets */btb_config[0],
                  /* btb assoc */btb_config[1],
                  /* ret-addr stack size */ras_size);
    }
  else if (!mystricmp(pred_type, "bi_mode"))
    {
      /* bi_mode adaptive predictor, bpred_create() checks args */
      if (bi_mode_nelt != 2)
      fatal("bad bi_mode pred config (<PHT tables size>
<hist_size>)");
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
      if (btb_nelt != 2)
      fatal("bad btb config (<num_sets> <associativity>)");

      pred = bpred_create(BPredBiMode,
                  /* BiMode table size */0,
                  /* BiMode l1 size */1,
                  /* BiMode l2 size */bi_mode_config[0],
                  /* meta table size */0,
                  /* history reg size */bi_mode_config[1],
                  /* history xor address */1,
                  /* btb sets */btb_config[0],
                  /* btb assoc */btb_config[1],
                  /* ret-addr stack size */ras_size);
    }
  else if (!mystricmp(pred_type, "skew"))
    {
      /* skewed branch predictor, bpred_create() checks args */
      if (skew_nelt != 2)
      fatal("bad skew pred config (<l2size> <hist_size>)");
      if (btb_nelt != 2)
      fatal("bad btb config (<num_sets> <associativity>)");

      pred = bpred_create(BPredSkew,
                  /* bimod table size */0,
                  /* 2lev l1 size */1,
                  /* 2lev l2 size */skew_config[0],
                  /* meta table size */0,
                  /* history reg size */skew_config[1],
                  /* history xor address */1,
                  /* btb sets */btb_config[0],
                  /* btb assoc */btb_config[1],
                  /* ret-addr stack size */ras_size);
    }
  else if (!mystricmp(pred_type, "filter"))
    {
      /* filter branch predictor, bpred_create() checks args */
      if (filter_nelt != 2)
      fatal("bad filter pred config (<l2size> <hist_size>)");
      if (btb_nelt != 2)
      fatal("bad btb config (<num_sets> <associativity>)");

      pred = bpred_create(BPredFilter,
                  /* bimod table size */0,
                  /* 2lev l1 size */1,
                  /* 2lev l2 size */filter_config[0],
                  /* meta table size */0,
                  /* history reg size */filter_config[1],
                  /* history xor address */1,
                  /* btb sets */btb_config[0],
                  /* btb assoc */btb_config[1],
                  /* ret-addr stack size */ras_size);
    }
  else if (!mystricmp(pred_type, "yags"))
    {
      /* filter branch predictor, bpred_create() checks args */
      if (yags_nelt != 5)
      fatal("bad yags pred config (<l2size> <hist_size> <TAG_size>"+
              " <cache_sets> <cache_assoc>)");
      if (btb_nelt != 2)
      fatal("bad btb config (<num_sets> <associativity>)");
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
      pred = bpred_create_YAGS(BPredYags,
                    /* bimod table size */0,
                    /* 2lev l1 size */1,
                    /* 2lev l2 size */yags_config[0],
                    /* meta table size */0,
                    /* history reg size */yags_config[1],
                    /* history xor address */1,
                    /* btb sets */btb_config[0],
                    /* btb assoc */btb_config[1],
                    /* ret-addr stack size */ras_size,
                    /* tag size*/ yags_config[2],
                    /* cache sets*/ yags_config[3],
                    /* cache assoc*/ yags_config[4]);
    }
  else if (!mystricmp(pred_type, "perceptron"))
    {
      /* 2-level adaptive predictor, bpred_create() checks args */
      if (percep_nelt != 5)
      fatal("bad perceptron pred config (<l1size> <l2size>
<hist_size>"+
              " <xor> <threshold>)");
      if (btb_nelt != 2)
      fatal("bad btb config (<num_sets> <associativity>)");

      pred = bpred_create(BPredPercep,
                    /* bimod table size */0,
                    /* 2lev l1 size */percep_config[0],
                    /* 2lev l2 size */percep_config[1],
                    /* threshold value */percep_config[4],
                    /* history reg size */percep_config[2],
                    /* history xor address */percep_config[3],
                    /* btb sets */btb_config[0],
                    /* btb assoc */btb_config[1],
                    /* ret-addr stack size */ras_size);
    }
  else
    fatal("cannot parse predictor type `%s'", pred_type);
}

/* register simulator-specific statistics */
void
sim_reg_stats(struct stat_sdb_t *sdb)
{
  stat_reg_counter(sdb, "sim_num_insn",
                "total number of instructions executed",
                &sim_num_insn, sim_num_insn, NULL);
  stat_reg_counter(sdb, "sim_num_refs",
                "total number of loads and stores executed",
                &sim_num_refs, 0, NULL);
  stat_reg_int(sdb, "sim_elapsed_time",
              "total simulation time in seconds",
              &sim_elapsed_time, 0, NULL);
  stat_reg_formula(sdb, "sim_inst_rate",
                "simulation speed (in insts/sec)",
                "sim_num_insn / sim_elapsed_time", NULL);

  stat_reg_counter(sdb, "sim_num_branches",
                   "total number of branches executed",
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
                    &sim_num_branches, /* initial value */0, /* format
*/NULL);
  stat_reg_formula(sdb, "sim_IPB",
                    "instruction per branch",
                    "sim_num_insn / sim_num_branches", /* format
*/NULL);

  stat_reg_int(sdb, "sim_dif", "Diferencia entre total de saltos y
total"+
                               " de fallos", &sim_dif, 0, NULL);

  /* register predictor stats */
  if (pred)
    bpred_reg_stats(pred, sdb);
}

/* initialize the simulator */
void
sim_init(void)
{
  sim_num_refs = 0;

  /* allocate and initialize register file */
  regs_init(&regs);

  /* allocate and initialize memory space */
  mem = mem_create("mem");
  mem_init(mem);
}

/* local machine state accessor */
static char *                           /* err str, NULL for no err
*/
bpred_mstate_obj(FILE *stream,                  /* output stream */
             char *cmd,             /* optional command string */
             struct regs_t *regs,       /* register to access */
             struct mem_t *mem)         /* memory to access */
{
  /* just dump intermediate stats */
  sim_print_stats(stream);

  /* no error */
  return NULL;
}

/* load program into simulated state */
void
sim_load_prog(char *fname,          /* program to load */
            int argc, char **argv,  /* program arguments */
            char **envp)            /* program environment */
{
  /* load program text and data, set up environment, memory, and regs
*/
  ld_load_prog(fname, argc, argv, envp, &regs, mem, TRUE);

  /* initialize the DLite debugger */
  dlite_init(md_reg_obj, dlite_mem_obj, bpred_mstate_obj);
}

/* print simulator-specific configuration information */
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
void
sim_aux_config(FILE *stream)       /* output stream */
{
  /* nothing currently */
}

/* dump simulator-specific auxiliary simulator statistics */
void
sim_aux_stats(FILE *stream)        /* output stream */
{
  /* nada */
}

/* un-initialize simulator-specific state */
void
sim_uninit(void)
{
  /* nada */
}


/*
 * configure the execution engine
 */

/*
 * precise architected register accessors
 */

/* next program counter */
#define SET_NPC(EXPR)        (regs.regs_NPC = (EXPR))

/* target program counter */
#undef  SET_TPC
#define SET_TPC(EXPR)        (target_PC = (EXPR))

/* current program counter */
#define CPC            (regs.regs_PC)

/* general purpose registers */
#define GPR(N)               (regs.regs_R[N])
#define SET_GPR(N,EXPR)      (regs.regs_R[N] = (EXPR))

#if defined(TARGET_PISA)

/* floating point registers, L->word, F->single-prec, D->double-prec
*/
#define FPR_L(N)        (regs.regs_F.l[(N)])
#define SET_FPR_L(N,EXPR)    (regs.regs_F.l[(N)] = (EXPR))
#define FPR_F(N)        (regs.regs_F.f[(N)])
#define SET_FPR_F(N,EXPR)    (regs.regs_F.f[(N)] = (EXPR))
#define FPR_D(N)        (regs.regs_F.d[(N) >> 1])
#define SET_FPR_D(N,EXPR)    (regs.regs_F.d[(N) >> 1] = (EXPR))

/* miscellaneous register accessors */
#define SET_HI(EXPR)         (regs.regs_C.hi = (EXPR))
#define HI             (regs.regs_C.hi)
#define SET_LO(EXPR)         (regs.regs_C.lo = (EXPR))
#define LO             (regs.regs_C.lo)
#define FCC            (regs.regs_C.fcc)
```

```
#define SET_FCC(EXPR)        (regs.regs_C.fcc = (EXPR))

#elif defined(TARGET_ALPHA)

/* floating point registers, L->word, F->single-prec, D->double-prec
*/
#define FPR_Q(N)       (regs.regs_F.q[N])
#define SET_FPR_Q(N,EXPR)    (regs.regs_F.q[N] = (EXPR))
#define FPR(N)               (regs.regs_F.d[N])
#define SET_FPR(N,EXPR)      (regs.regs_F.d[N] = (EXPR))

/* miscellaneous register accessors */
#define FPCR                 (regs.regs_C.fpcr)
#define SET_FPCR(EXPR)       (regs.regs_C.fpcr = (EXPR))
#define UNIQ                 (regs.regs_C.uniq)
#define SET_UNIQ(EXPR)       (regs.regs_C.uniq = (EXPR))

#else
#error No ISA target defined...
#endif

/* precise architected memory state help functions */
#define READ_BYTE(SRC, FAULT)                                   \
  ((FAULT) = md_fault_none, addr = (SRC), MEM_READ_BYTE(mem, addr))
#define READ_HALF(SRC, FAULT)                                   \
  ((FAULT) = md_fault_none, addr = (SRC), MEM_READ_HALF(mem, addr))
#define READ_WORD(SRC, FAULT)                                   \
  ((FAULT) = md_fault_none, addr = (SRC), MEM_READ_WORD(mem, addr))
#ifdef HOST_HAS_QWORD
#define READ_QWORD(SRC, FAULT)                                    \
  ((FAULT) = md_fault_none, addr = (SRC), MEM_READ_QWORD(mem, addr))
#endif /* HOST_HAS_QWORD */

#define WRITE_BYTE(SRC, DST, FAULT)                             \
  ((FAULT) = md_fault_none, addr = (DST), MEM_WRITE_BYTE(mem, addr,
(SRC)))
#define WRITE_HALF(SRC, DST, FAULT)                             \
  ((FAULT) = md_fault_none, addr = (DST), MEM_WRITE_HALF(mem, addr,
(SRC)))
#define WRITE_WORD(SRC, DST, FAULT)                             \
  ((FAULT) = md_fault_none, addr = (DST), MEM_WRITE_WORD(mem, addr,
(SRC)))
#ifdef HOST_HAS_QWORD
#define WRITE_QWORD(SRC, DST, FAULT)                              \
  ((FAULT) = md_fault_none, addr = (DST), MEM_WRITE_QWORD(mem, addr,
(SRC)))
#endif /* HOST_HAS_QWORD */

/* system call handler macro */
#define SYSCALL(INST)  sys_syscall(&regs, mem_access, mem, INST, TRUE)

/* start simulation, program loaded, processor precise state
initialized */
void
sim_main(void)
{
  md_inst_t inst;
  register md_addr_t addr, target_PC = 0;
  enum md_opcode op;
  register int is_write;
```

```
  int stack_idx;
  enum md_fault_type fault;

  fprintf(stderr, "sim: ** starting functional simulation w/
predictors **\n");

  /* set up initial default next PC */
  regs.regs_NPC = regs.regs_PC + sizeof(md_inst_t);

  /* check for DLite debugger entry condition */
  if (dlite_check_break(regs.regs_PC, /* no access */0, /* addr */0,
0, 0))
    dlite_main(regs.regs_PC - sizeof(md_inst_t), regs.regs_PC,
               sim_num_insn, &regs, mem);

  while (TRUE)
    {
      /* maintain $r0 semantics */
      regs.regs_R[MD_REG_ZERO] = 0;
#ifdef TARGET_ALPHA
      regs.regs_F.d[MD_REG_ZERO] = 0.0;
#endif /* TARGET_ALPHA */

      /* get the next instruction to execute */
      MD_FETCH_INST(inst, mem, regs.regs_PC);

      /* keep an instruction count */
      sim_num_insn++;

      /* Actualizacion de nuestra nueva estadistica */
      fallos = pred->misses;
      sim_dif = sim_num_branches - fallos;

      /* set default reference address and access mode */
      addr = 0; is_write = FALSE;

      /* set default fault - none */
      fault = md_fault_none;

      /* decode the instruction */
      MD_SET_OPCODE(op, inst);

      /* execute the instruction */
      switch (op)
      {
#define DEFINST(OP,MSK,NAME,OPFORM,RES,FLAGS,O1,O2,I1,I2,I3)
      \
      case OP:                                          \
          SYMCAT(OP,_IMPL);                                 \
          break;
#define DEFLINK(OP,MSK,NAME,MASK,SHIFT)                           \
        case OP:                                          \
          panic("attempted to execute a linking opcode");
#define CONNECT(OP)
#define DECLARE_FAULT(FAULT)                              \
        { fault = (FAULT); break; }
#include "machine.def"
      default:
        panic("attempted to execute a bogus opcode");
      }
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
      if (fault != md_fault_none)
      fatal("fault (%d) detected @ 0x%08p", fault, regs.regs_PC);

      if (MD_OP_FLAGS(op) & F_MEM)
      {
        sim_num_refs++;
        if (MD_OP_FLAGS(op) & F_STORE)
          is_write = TRUE;
      }

      if (MD_OP_FLAGS(op) & F_CTRL)
      {
        md_addr_t pred_PC;
        struct bpred_update_t update_rec;

        sim_num_branches++;

        if (pred)
          {
            /* get the next predicted fetch address */
            pred_PC = bpred_lookup(pred,
                           /* branch addr */regs.regs_PC,
                           /* target */target_PC,
                           /* inst opcode */op,
                           /* call? */MD_IS_CALL(op),
                           /* return? */MD_IS_RETURN(op),
                           /* stash an update ptr */&update_rec,
                           /* stash return stack ptr */&stack_idx);

            /* valid address returned from branch predictor? */
            if (!pred_PC)
            {
              /* no predicted taken target, attempt not taken target */
              pred_PC = regs.regs_PC + sizeof(md_inst_t);
            }

            /*si devolvemos un 1, tambien hemos tomado el salto, xo no
estaba en la btb*/
            bpred_update(pred,
                     /* branch addr */regs.regs_PC,
                     /* resolved branch target */regs.regs_NPC,
                     /* taken? */regs.regs_NPC != (regs.regs_PC +
                                          sizeof(md_inst_t)),
                     /* pred taken? */pred_PC != (regs.regs_PC +
                                         sizeof(md_inst_t)),
                     /* correct pred? */pred_PC == regs.regs_NPC,
                     /* opcode */op,
                     /* predictor update pointer */&update_rec);
          }
      }

      /* check for DLite debugger entry condition */
      if (dlite_check_break(regs.regs_NPC,
                      is_write ? ACCESS_WRITE : ACCESS_READ,
                      addr, sim_num_insn, sim_num_insn))
      dlite_main(regs.regs_PC, regs.regs_NPC, sim_num_insn, &regs,
mem);

      /* go to the next instruction */
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
      regs.regs_PC = regs.regs_NPC;
      regs.regs_NPC += sizeof(md_inst_t);

      /* finish early? */
      if (max_insts && sim_num_insn >= max_insts)
      return;
    }
}
```

2. Contenido de BPRED.H

El siguiente archivo que tenemos es bpred.h que tiene los tipos de datos y cabeceras de los métodos usados en bpred.c:

```
/* bpred.h - branch predictor interfaces */

/* SimpleScalar(TM) Tool Suite
 * Copyright (C) 1994-2003 by Todd M. Austin, Ph.D. and SimpleScalar,
LLC.
 * All Rights Reserved.
 *
 * THIS IS A LEGAL DOCUMENT, BY USING SIMPLESCALAR,
 * YOU ARE AGREEING TO THESE TERMS AND CONDITIONS.
 *
 * No portion of this work may be used by any commercial entity, or
for any
 * commercial purpose, without the prior, written permission of
SimpleScalar,
 * LLC (info@simplescalar.com). Nonprofit and noncommercial use is
permitted
 * as described below.
 *
 * 1. SimpleScalar is provided AS IS, with no warranty of any kind,
express
 * or implied. The user of the program accepts full responsibility for
the
 * application of the program and the use of any results.
 *
 * 2. Nonprofit and noncommercial use is encouraged. SimpleScalar may
be
 * downloaded, compiled, executed, copied, and modified solely for
nonprofit,
 * educational, noncommercial research, and noncommercial scholarship
 * purposes provided that this notice in its entirety accompanies all
copies.
 * Copies of the modified software can be delivered to persons who use
it
 * solely for nonprofit, educational, noncommercial research, and
 * noncommercial scholarship purposes provided that this notice in its
 * entirety accompanies all copies.
 *
 * 3. ALL COMMERCIAL USE, AND ALL USE BY FOR PROFIT ENTITIES, IS
EXPRESSLY
 * PROHIBITED WITHOUT A LICENSE FROM SIMPLESCALAR, LLC
(info@simplescalar.com).
 *
 * 4. No nonprofit user may place any restrictions on the use of this
software,
 * including as modified by the user, by any other authorized user.
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
 *
 * 5. Noncommercial and nonprofit users may distribute copies of
SimpleScalar
 * in compiled or executable form as set forth in Section 2, provided
that
 * either: (A) it is accompanied by the corresponding machine-readable
source
 * code, or (B) it is accompanied by a written offer, with no time
limit, to
 * give anyone a machine-readable copy of the corresponding source
code in
 * return for reimbursement of the cost of distribution. This written
offer
 * must permit verbatim duplication by anyone, or (C) it is
distributed by
 * someone who received only the executable form, and is accompanied
by a
 * copy of the written offer of source code.
 *
 * 6. SimpleScalar was developed by Todd M. Austin, Ph.D. The tool
suite is
 * currently maintained by SimpleScalar LLC (info@simplescalar.com).
US Mail:
 * 2395 Timbercrest Court, Ann Arbor, MI 48105.
 *
 * Copyright (C) 1994-2003 by Todd M. Austin, Ph.D. and SimpleScalar,
LLC.
 */


#ifndef BPRED_H
#define BPRED_H

#define dassert(a) assert(a)

#include <stdio.h>

#include "host.h"
#include "misc.h"
#include "machine.h"
#include "stats.h"

/*
 * This module implements a number of branch predictor mechanisms.
The
 * following predictors are supported:
 *
 *    BPred2Level:  two level adaptive branch predictor
 *
 *          It can simulate many prediction mechanisms that have up to
 *          two levels of tables. Parameters are:
 *               N   # entries in first level (# of shift register(s))
 *               W   width of shift register(s)
 *               M   # entries in 2nd level (# of counters, or other
FSM)
 *          One BTB entry per level-2 counter.
 *
 *          Configurations:   N, W, M
 *
 *              counter based: 1, 0, M
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
 *
 *              GAg          : 1, W, 2^W
 *              GAp          : 1, W, M (M > 2^W)
 *              PAg          : N, W, 2^W
 *              PAp          : N, W, M (M == 2^(N+W))
 *
 *    BPred2bit:  a simple direct mapped bimodal predictor
 *
 *          This predictor has a table of two bit saturating counters.
 *          Where counter states 0 & 1 are predict not taken and
 *          counter states 2 & 3 are predict taken, the per-branch
counters
 *          are incremented on taken branches and decremented on
 *          no taken branches.  One BTB entry per counter.
 *
 *    BPredTaken:  static predict branch taken
 *
 *    BPredNotTaken:  static predict branch not taken
 *
 */

/* Perceptron type */

struct perceptron_t {
  int *pesos;
  int resultado;
};

/* branch predictor types */
enum bpred_class {
  BPredComb,                  /* combined predictor (McFarling) */
  BPred2Level,               /* 2-level correlating pred w/2-bit
counters */
  BPred2bit,                 /* 2-bit saturating cntr pred (dir
mapped) */
  BPredTaken,                /* static predict taken */
  BPredNotTaken,        /* static predict not taken */
  BPredAgree,                /* agree branch predictor */
  BPredBiMode,               /* bi-mode branch predictor */
  BPredSkew,                 /* skewed branch predictor */
  BPredFilter,               /* filter branch predictor */
  BPredYags,                   /* YAGS branch predictor */
  BPredPercep,               /* perceptron branch predictor */
  BPred_NUM
};

/* an entry in a BTB */
struct bpred_btb_ent_t {
  md_addr_t addr;       /* address of branch being tracked */
  enum md_opcode op;          /* opcode of branch corresp. to addr */
  md_addr_t target;           /* last destination of branch when taken
*/
  int bias;             /* bias bit for agree or filter predictor */
  int countFilter;            /* counter bit for filter predictor */
  struct bpred_btb_ent_t *prev,
         *next;                 /* lru chaining pointers */
};

/* an entry in YAGS cache */
struct bpred_yagscache_ent_t {
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
  int tag;                /* tag of the YAGS cache */
  int count;                    /* associated counter for tag */
  struct bpred_yagscache_ent_t
                *prev, *next;   /* lru chaining pointers */
};

/* direction predictor def */
struct bpred_dir_t {
  enum bpred_class class;     /* type of predictor */
  union {
    struct {
      unsigned int size;      /* number of entries in direct-mapped
table */
      unsigned char *table;   /* prediction state table */
    } bimod;
    struct {
      int l1size;       /* level-1 size, number of history regs */
      int l2size;       /* level-2 size, number of pred states */
      int shift_width;        /* amount of history in level-1 shift
regs */
      int xor;                /* history xor address flag */
      int *shiftregs;         /* level-1 history table */
      unsigned char *l2table; /* level-2 prediction state table */
    } two;
    struct {
      int l1size;             /* level-1 size, number of history regs
*/
      int l2size;             /* level-2 size, number of pred states */
      int shift_width;              /* amount of history in level-1
shift regs */
      int xor;                      /* history xor address flag */
      int *shiftregs;               /* level-1 history table */
      unsigned char *NTtable;       /* level-2 prediction state table
for
                                    not taken branchs */
      unsigned char *Ttable;        /* level-2 prediction state table
for
                                    taken branchs */
      unsigned char
                *choicetable;   /* choice table between PHT NT & PHT T
*/
    } bi_mode;
    struct {
      int l1size;       /* level-1 size, number of history regs */
      int l2size;       /* level-2 size, number of pred states */
      int shift_width;        /* amount of history in level-1 shift
regs */
      int xor;                /* history xor address flag */
      int *shiftregs;         /* level-1 history table */
      unsigned char *bank1;   /* bank 1 prediction state table */
      unsigned char *bank2;   /* bank 2 prediction state table */
      unsigned char *bank3;   /* bank 3 prediction state table */
    } skew;
    struct {
      int l1size;       /* level-1 size, number of history regs */
      int l2size;       /* level-2 size, number of pred states */
      int shift_width;        /* amount of history in level-1 shift
regs */
      int xor;                      /* history xor address flag */
      int *shiftregs;         /* level-1 history table */
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
      int tagsize;           /* tags size */
      int cachesize;         /* cache size, number of cache states */
      struct {
        int sets;            /* num cache sets */
      int assoc;             /* cache associativity */
        struct bpred_yagscache_ent_t *data;
      } NTcache;       /* cache for not taken branchs */
      struct {
        int sets;      /* num cache sets */
        int assoc;           /* cache associativity */
        struct bpred_yagscache_ent_t *data;
      } Tcache;              /* cache for taken branchs */
      unsigned char *choicePHT;    /* choice table between PHT NT &
PHT T */
      int cacheindex;              /* index of cache access */
      int last_tag;                /* saves de last value of the tag
used to
                                   reference a cache position */
      int new_tag;                 /* stores the tag for a new branch
*/
    } yags;
    struct {
      int l1size;      /* level-1 size, number of history regs */
      int l2size;      /* level-2 size, number of pred states */
      int l2index;           /* level-2 perceptron used for a
prediction */
      int l1index;           /* level-1 perceptron used for a
prediction */
      int threshold;         /* threshold used for train perceptrons
*/
      int shift_width;       /* amount of history in level-1 shift
regs */
      int xor;                     /* history xor address flag */
      int *shiftregs;        /* level-1 history table */
      struct perceptron_t
             *l2table;         /* level-2 prediction state table */
      int n_percep;          /* number of perceptrons in l2 table */
    } percep;
  } config;
};

/* branch predictor def */
struct bpred_t {
  enum bpred_class class;    /* type of predictor */
  struct {
    struct bpred_dir_t *bimod;  /* first direction predictor */
    struct bpred_dir_t *twolev; /* second direction predictor */
    struct bpred_dir_t *meta;   /* meta predictor */
    struct bpred_dir_t *choice; /* Bi-Mode direction predictor */
    struct bpred_dir_t *skew;   /* skewed branch predictor */
    struct bpred_dir_t *yags;   /* YAGS branch predictor */
    struct bpred_dir_t *percep; /* Perceptron branch predictor */
  } dirpred;

  struct {
    int sets;                /* num BTB sets */
    int assoc;               /* BTB associativity */
    struct bpred_btb_ent_t *btb_data; /* BTB addr-prediction table */
  } btb;
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
  struct {
    int size;                /* return-address stack size */
    int tos;                 /* top-of-stack */
    struct bpred_btb_ent_t *stack; /* return-address stack */
  } retstack;

  /* stats */
  counter_t addr_hits;       /* num correct addr-predictions */
  counter_t dir_hits;        /* num correct dir-predictions (incl
addr) */
  counter_t used_ras;        /* num RAS predictions used */
  counter_t used_bimod;      /* num bimodal predictions used
(BPredComb) */
  counter_t used_2lev;       /* num 2-level predictions used
(BPredComb) */
  counter_t used_percep;     /* num perceptron predictions used to
train */
  counter_t dirhits_after_training;
  counter_t desthits_after_training;
  counter_t branches_after_training;
  counter_t jr_hits;         /* num correct addr-predictions for JR's
*/
  counter_t jr_seen;         /* num JR's seen */
  counter_t jr_non_ras_hits; /* num correct addr-preds for non-RAS
JR's */
  counter_t jr_non_ras_seen; /* num non-RAS JR's seen */
  counter_t misses;          /* num incorrect predictions */

  counter_t lookups;         /* num lookups */
  counter_t retstack_pops;   /* number of times a value was popped */
  counter_t retstack_pushes; /* number of times a value was pushed */
  counter_t ras_hits;        /* num correct return-address predictions
*/
};

/* branch predictor update information */
struct bpred_update_t {
  char *pdir1;                  /* direction-1 predictor counter */
  char *pdir2;          /* direction-2 predictor counter */
  char *pmeta;              /* meta predictor counter */
  char *pchoice;          /* Bi-Mode & YAGS predictor counter
                             (for choice table) */
  char *pbank1;               /* prediction of bank1 (skewed
predictor) */
  char *pbank2;               /* prediction of bank2 (skewed
predictor) */
  char *pbank3;               /* prediction of bank3 (skewed
predictor) */
  struct {               /* predicted directions */
    unsigned int ras    : 1; /* RAS used */
    unsigned int bimod  : 1;    /* bimodal predictor */
    unsigned int twolev : 1;    /* 2-level predictor */
    unsigned int meta   : 1;    /* meta predictor (0..bimod / 1..2lev)
*/
    unsigned int choice : 1; /* Bi-Mode & YAGS predictor (0 - NT, 1 -
T) */
    unsigned int skew   : 1;    /* skewed branch predictor */
    unsigned int percep : 1; /* perceptron branch predictor */
  } dir;
};
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
/* create a branch predictor */
struct bpred_t *             /* branch predictory instance */
bpred_create(enum bpred_class class,     /* type of predictor to
create */
           unsigned int bimod_size, /* bimod table size */
           unsigned int l1size,     /* level-1 table size */
           unsigned int l2size,     /* level-2 table size */
           unsigned int meta_size,  /* meta predictor table size */
           unsigned int shift_width,     /* history register width */
           unsigned int xor,        /* history xor address flag */
           unsigned int btb_sets,   /* number of sets in BTB */
           unsigned int btb_assoc,  /* BTB associativity */
           unsigned int retstack_size);/* num entries in ret-addr
stack */

/* create a YAGS branch predictor */
struct bpred_t *                     /* branch predictory instance */
bpred_create_YAGS(enum bpred_class class,       /* type of predictor
                                                   to create */
                unsigned int bimod_size,  /* bimod table size */
                unsigned int l1size,           /* level-1 table size
*/
                unsigned int l2size,           /* level-2 table size
*/
                unsigned int meta_size,   /* meta predictor table size
*/
                unsigned int shift_width, /* history register width */
                unsigned int xor,         /* history xor address flag
*/
                unsigned int btb_sets,    /* number of sets in BTB */
                unsigned int btb_assoc,   /* BTB associativity */
                unsigned int retstack_size,    /* num entries in ret-
addr
                                                  stack */
                  unsigned int tag_size,  /* tag size */
                unsigned int cache_sets,  /* cache sets */
                unsigned int cache_assoc);     /* cache
associativity*/

/* create a branch direction predictor */
struct bpred_dir_t *         /* branch direction predictor instance */
bpred_dir_create (
  enum bpred_class class,     /* type of predictor to create */
  unsigned int l1size,        /* level-1 table size */
  unsigned int l2size,        /* level-2 table size (if relevant) */
  unsigned int shift_width,   /* history register width */
  unsigned int xor);          /* history xor address flag */

/* create a branch direction predictor */
struct bpred_dir_t *         /* branch direction predictor instance */
bpred_dir_create_YAGS (
  enum bpred_class class,     /* type of predictor to create */
  unsigned int l1size,        /* level-1 table size */
  unsigned int l2size,        /* level-2 table size (if relevant) */
  unsigned int shift_width,   /* history register width */
  unsigned int xor,           /* history xor address flag */
  unsigned int tag_size,      /* tag size */
  unsigned int cache_sets,    /* cache sets */
  unsigned int cache_assoc);  /* cache associativity*/
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
/* create a branch direction predictor */
struct bpred_dir_t *         /* branch direction predictor instance */
bpred_dir_create_percep (
  enum bpred_class class,     /* type of predictor to create */
  unsigned int l1size,        /* level-1 table size */
  unsigned int l2size,        /* level-2 table size (if relevant) */
  unsigned int shift_width,   /* history register width */
  unsigned int xor,           /* history xor address flag */
  unsigned int threshold);    /* threshold value */

/* print branch predictor configuration */
void
bpred_config(struct bpred_t *pred,  /* branch predictor instance */
           FILE *stream);           /* output stream */

/* print predictor stats */
void
bpred_stats(struct bpred_t *pred,   /* branch predictor instance */
          FILE *stream);            /* output stream */

/* register branch predictor stats */
void
bpred_reg_stats(struct bpred_t *pred,    /* branch predictor instance
*/
            struct stat_sdb_t *sdb);/* stats database */

/* reset stats after priming, if appropriate */
void bpred_after_priming(struct bpred_t *bpred);

/* probe a predictor for a next fetch address, the predictor is probed
   with branch address BADDR, the branch target is BTARGET (used for
   static predictors), and OP is the instruction opcode (used to
simulate
   predecode bits; a pointer to the predictor state entry (or null for
jumps)
   is returned in *DIR_UPDATE_PTR (used for updating predictor state),
   and the non-speculative top-of-stack is returned in
stack_recover_idx
   (used for recovering ret-addr stack after mis-predict).  */
md_addr_t                    /* predicted branch target addr */
bpred_lookup(struct bpred_t *pred,  /* branch predictor instance */
           md_addr_t baddr,         /* branch address */
           md_addr_t btarget,       /* branch target if taken */
           enum md_opcode op,       /* opcode of instruction */
           int is_call,       /* non-zero if inst is fn call */
           int is_return,           /* non-zero if inst is fn return */
           struct bpred_update_t *dir_update_ptr, /* pred state
pointer */
           int *stack_recover_idx); /* Non-speculative top-of-stack;
                                used on mispredict recovery */

/* Speculative execution can corrupt the ret-addr stack.  So for each
 * lookup we return the top-of-stack (TOS) at that point; a
mispredicted
 * branch, as part of its recovery, restores the TOS using this value
--
 * hopefully this uncorrupts the stack. */
void
bpred_recover(struct bpred_t *pred, /* branch predictor instance */
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
            md_addr_t baddr,        /* branch address */
            int stack_recover_idx); /* Non-speculative top-of-stack;
                                       used on mispredict recovery */

/* update the branch predictor, only useful for stateful predictors;
updates
   entry for instruction type OP at address BADDR.  BTB only gets
updated
   for branches which are taken.  Inst was determined to jump to
   address BTARGET and was taken if TAKEN is non-zero.  Predictor
   statistics are updated with result of prediction, indicated by
CORRECT and
   PRED_TAKEN, predictor state to be updated is indicated by
*DIR_UPDATE_PTR
   (may be NULL for jumps, which shouldn't modify state bits).  Note
if
   bpred_update is done speculatively, branch-prediction may get
polluted. */
void
bpred_update(struct bpred_t *pred,  /* branch predictor instance */
           md_addr_t baddr,         /* branch address */
           md_addr_t btarget,       /* resolved branch target */
           int taken,               /* non-zero if branch was taken */
           int pred_taken,          /* non-zero if branch was pred
taken */
           int correct,       /* was earlier prediction correct? */
           enum md_opcode op,       /* opcode of instruction */
           struct bpred_update_t *dir_update_ptr); /* pred state
pointer */


#ifdef foo0
/* OBSOLETE */
/* dump branch predictor state (for debug) */
void
bpred_dump(struct bpred_t *pred,    /* branch predictor instance */
         FILE *stream);       /* output stream */
#endif

#endif /* BPRED_H */
```

3. Contenido de BPRED.C

Por ultimo, tenemos el archive bpred.c que es el que contiene todos los métodos y funciones que implementan cada uno de los distintos predictores de saltos estudiados a lo largo de todo este proyecto. El código es el siguiente:

```
/* bpred.c - branch predictor routines */

/* SimpleScalar(TM) Tool Suite
 * Copyright (C) 1994-2003 by Todd M. Austin, Ph.D. and SimpleScalar,
LLC.
 * All Rights Reserved.
 *
 * THIS IS A LEGAL DOCUMENT, BY USING SIMPLESCALAR,
 * YOU ARE AGREEING TO THESE TERMS AND CONDITIONS.
 *
 * No portion of this work may be used by any commercial entity, or
for any
```

```
 * commercial purpose, without the prior, written permission of
SimpleScalar,
 * LLC (info@simplescalar.com). Nonprofit and noncommercial use is
permitted
 * as described below.
 *
 * 1. SimpleScalar is provided AS IS, with no warranty of any kind,
express
 * or implied. The user of the program accepts full responsibility for
the
 * application of the program and the use of any results.
 *
 * 2. Nonprofit and noncommercial use is encouraged. SimpleScalar may
be
 * downloaded, compiled, executed, copied, and modified solely for
nonprofit,
 * educational, noncommercial research, and noncommercial scholarship
 * purposes provided that this notice in its entirety accompanies all
copies.
 * Copies of the modified software can be delivered to persons who use
it
 * solely for nonprofit, educational, noncommercial research, and
 * noncommercial scholarship purposes provided that this notice in its
 * entirety accompanies all copies.
 *
 * 3. ALL COMMERCIAL USE, AND ALL USE BY FOR PROFIT ENTITIES, IS
EXPRESSLY
 * PROHIBITED WITHOUT A LICENSE FROM SIMPLESCALAR, LLC
(info@simplescalar.com).
 *
 * 4. No nonprofit user may place any restrictions on the use of this
software,
 * including as modified by the user, by any other authorized user.
 *
 * 5. Noncommercial and nonprofit users may distribute copies of
SimpleScalar
 * in compiled or executable form as set forth in Section 2, provided
that
 * either: (A) it is accompanied by the corresponding machine-readable
source
 * code, or (B) it is accompanied by a written offer, with no time
limit, to
 * give anyone a machine-readable copy of the corresponding source
code in
 * return for reimbursement of the cost of distribution. This written
offer
 * must permit verbatim duplication by anyone, or (C) it is
distributed by
 * someone who received only the executable form, and is accompanied
by a
 * copy of the written offer of source code.
 *
 * 6. SimpleScalar was developed by Todd M. Austin, Ph.D. The tool
suite is
 * currently maintained by SimpleScalar LLC (info@simplescalar.com).
US Mail:
 * 2395 Timbercrest Court, Ann Arbor, MI 48105.
 *
 * Copyright (C) 1994-2003 by Todd M. Austin, Ph.D. and SimpleScalar,
LLC.
```

```
 */


#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <assert.h>

#include "host.h"
#include "misc.h"
#include "machine.h"
#include "bpred.h"

/* turn this on to enable the SimpleScalar 2.0 RAS bug */
/* #define RAS_BUG_COMPATIBLE */

unsigned char *skew_pred[3] = {NULL, NULL, NULL};
unsigned char *yagsPHT = NULL;

#define MAX_PESO ((1<<(7))-1)
#define MIN_PESO (-(MAX_PESO+1))

/* create a branch predictor */
struct bpred_t *             /* branch predictory instance */
bpred_create(enum bpred_class class,     /* type of predictor to
create */
           unsigned int bimod_size, /* bimod table size */
           unsigned int l1size,     /* 2lev l1 table size */
           unsigned int l2size,     /* 2lev l2 table size */
           unsigned int meta_size,  /* meta table size */
           unsigned int shift_width,     /* history register width */
           unsigned int xor,        /* history xor address flag */
           unsigned int btb_sets,   /* number of sets in BTB */
           unsigned int btb_assoc,  /* BTB associativity */
           unsigned int retstack_size) /* num entries in ret-addr
stack */ {

  struct bpred_t *pred;

  if (!(pred = calloc(1, sizeof(struct bpred_t))))
    fatal("out of virtual memory");

  pred->class = class;

  switch (class) {

  case BPredComb:
    /* bimodal component */
    pred->dirpred.bimod =
      bpred_dir_create(BPred2bit, bimod_size, 0, 0, 0);

    /* 2-level component */
    pred->dirpred.twolev =
      bpred_dir_create(BPred2Level, l1size, l2size, shift_width, xor);

    /* metapredictor component */
    pred->dirpred.meta =
      bpred_dir_create(BPred2bit, meta_size, 0, 0, 0);

    break;
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
  case BPred2Level:
    pred->dirpred.twolev = bpred_dir_create(class, l1size, l2size,
shift_width,
                                            xor);
    break;

  case BPred2bit:
    pred->dirpred.bimod = bpred_dir_create(class, bimod_size, 0, 0,
0);
    break;

  case BPredAgree:
    pred->dirpred.twolev = bpred_dir_create(class, l1size, l2size,
                                            shift_width, 1);
    break;

  case BPredBiMode:
    pred->dirpred.choice = bpred_dir_create(class, 1, l2size,
shift_width, 1);
    break;

  case BPredSkew:
    pred->dirpred.skew = bpred_dir_create(class, 1, l2size,
shift_width, 1);
    break;

  case BPredFilter:
    pred->dirpred.twolev = bpred_dir_create(class, 1, l2size,
shift_width, 1);
    break;

  case BPredPercep:
    pred->dirpred.percep = bpred_dir_create_percep(class, l1size,
l2size,
                                                   shift_width, xor,
meta_size);
    break;

  case BPredTaken:

  case BPredNotTaken:
    /* no other state */
    break;

  default:
    panic("bogus predictor class");
  }

  /* allocate ret-addr stack */
  switch (class) {
  case BPredComb:
  case BPred2Level:
  case BPred2bit:
  case BPredAgree:
  case BPredBiMode:
  case BPredSkew:
  case BPredFilter:
  case BPredPercep: {
    int i;
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
    /* allocate BTB */
    if (!btb_sets || (btb_sets & (btb_sets-1)) != 0)
      fatal("number of BTB sets must be non-zero and a power of two");
    if (!btb_assoc || (btb_assoc & (btb_assoc-1)) != 0)
      fatal("BTB associativity must be non-zero and a power of two");

    if (!(pred->btb.btb_data = calloc(btb_sets * btb_assoc,
                                      sizeof(struct bpred_btb_ent_t))))
      fatal("cannot allocate BTB");

    pred->btb.sets = btb_sets;
    pred->btb.assoc = btb_assoc;
    srand(1000);

    if (pred->btb.assoc > 1)
      for (i=0; i < (pred->btb.assoc*pred->btb.sets); i++) {
        if (i % pred->btb.assoc != pred->btb.assoc - 1)
        pred->btb.btb_data[i].next = &pred->btb.btb_data[i+1];
      else
        pred->btb.btb_data[i].next = NULL;

      if (i % pred->btb.assoc != pred->btb.assoc - 1)
        pred->btb.btb_data[i+1].prev = &pred->btb.btb_data[i];

      int aleatorio = rand() % 2;
      pred->btb.btb_data[i].bias = aleatorio;
      }

    /* allocate retstack */
    if ((retstack_size & (retstack_size-1)) != 0)
      fatal("Return-address-stack size must be zero or a power of
two");

    pred->retstack.size = retstack_size;
    if (retstack_size)
      if (!(pred->retstack.stack = calloc(retstack_size,
                                sizeof(struct bpred_btb_ent_t))))
        fatal("cannot allocate return-address-stack");
    pred->retstack.tos = retstack_size - 1;

    break;
  }

  case BPredTaken:
  case BPredNotTaken:
    /* no other state */
    break;

  default:
    panic("bogus predictor class");
  }

  return pred;
}

/* create a YAGS branch predictor */
struct bpred_t *                       /* branch predictory instance */
bpred_create_YAGS(enum bpred_class class,/* type of predictor
                                                    to create */
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
              unsigned int bimod_size,    /* bimod table size */
                unsigned int l1size,           /* 2lev l1 table size
*/
                unsigned int l2size,           /* 2lev l2 table size
*/
                unsigned int meta_size,   /* meta table size */
                unsigned int shift_width, /* history register width */
                unsigned int xor,         /* history xor address flag
*/
                unsigned int btb_sets,    /* number of sets in BTB */
                unsigned int btb_assoc,   /* BTB associativity */
                unsigned int retstack_size,    /* num entries in
                                                  ret-addr stack */
                unsigned int tag_size,    /* tag size */
              unsigned int cache_sets,    /* cache sets */
              unsigned int cache_assoc)   /* cache associativity*/ {

  struct bpred_t *pred;

  if (!(pred = calloc(1, sizeof(struct bpred_t))))
    fatal("out of virtual memory");

  pred->class = class;

  switch (class) {
  case BPredYags:
    pred->dirpred.yags =
      bpred_dir_create_YAGS(class, l1size, l2size, shift_width, xor,
tag_size,
                            cache_sets, cache_assoc);
    break;

  default:
    panic("bogus predictor class");
  }

  /* allocate ret-addr stack */
  switch (class) {
  case BPredYags: {
    int i;
    /* allocate BTB */
    if (!btb_sets || (btb_sets & (btb_sets-1)) != 0)
      fatal("number of BTB sets must be non-zero and a power of two");
    if (!btb_assoc || (btb_assoc & (btb_assoc-1)) != 0)
      fatal("BTB associativity must be non-zero and a power of two");

    if (!(pred->btb.btb_data = calloc(btb_sets * btb_assoc,
                              sizeof(struct bpred_btb_ent_t))))
      fatal("cannot allocate BTB");

    pred->btb.sets = btb_sets;
    pred->btb.assoc = btb_assoc;
    srand(1000);

    if (pred->btb.assoc > 1)
      for (i=0; i < (pred->btb.assoc*pred->btb.sets); i++) {
        if (i % pred->btb.assoc != pred->btb.assoc - 1)
          pred->btb.btb_data[i].next = &pred->btb.btb_data[i+1];
        else
          pred->btb.btb_data[i].next = NULL;
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
        if (i % pred->btb.assoc != pred->btb.assoc - 1)
          pred->btb.btb_data[i+1].prev = &pred->btb.btb_data[i];
      int aleatorio = rand() % 2;
      pred->btb.btb_data[i].bias = aleatorio;
      }

    /* allocate retstack */
    if ((retstack_size & (retstack_size-1)) != 0)
      fatal("Return-address-stack size must be zero or a power of
two");

    pred->retstack.size = retstack_size;
    if (retstack_size)
      if (!(pred->retstack.stack = calloc(retstack_size,
                                          sizeof(struct
bpred_btb_ent_t))))
        fatal("cannot allocate return-address-stack");
    pred->retstack.tos = retstack_size - 1;
    break;
  }

  default:
    panic("bogus predictor class");
  }

  return pred;
}

/* create a branch direction predictor */
struct bpred_dir_t *         /* branch direction predictor instance */
bpred_dir_create (
  enum bpred_class class,    /* type of predictor to create */
  unsigned int l1size,       /* level-1 table size */
  unsigned int l2size,       /* level-2 table size (if relevant) */
  unsigned int shift_width,  /* history register width */
  unsigned int xor)          /* history xor address flag */ {
  struct bpred_dir_t *pred_dir;
  unsigned int cnt;
  int flipflop;

  if (!(pred_dir = calloc(1, sizeof(struct bpred_dir_t))))
    fatal("out of virtual memory");

  pred_dir->class = class;

  cnt = -1;
  switch (class) {
  case BPred2Level:
  case BPredAgree:
  case BPredFilter: {
    if (!l1size || (l1size & (l1size-1)) != 0)
      fatal("level-1 size, `%d', must be non-zero and a power of two",
            l1size);
    pred_dir->config.two.l1size = l1size;

    if (!l2size || (l2size & (l2size-1)) != 0)
      fatal("level-2 size, `%d', must be non-zero and a power of two",
          l2size);
    pred_dir->config.two.l2size = l2size;
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
    if (!shift_width || shift_width > 30)
      fatal("shift register width, `%d', must be non-zero and
positive",
            shift_width);
    pred_dir->config.two.shift_width = shift_width;

    pred_dir->config.two.xor = xor;
    pred_dir->config.two.shiftregs = calloc(l1size, sizeof(int));
    if (!pred_dir->config.two.shiftregs)
      fatal("cannot allocate shift register table");

    pred_dir->config.two.l2table = calloc(l2size, sizeof(unsigned
char));
    if (!pred_dir->config.two.l2table)
      fatal("cannot allocate second level table");

    /* initialize counters to weakly this-or-that */
    flipflop = 1;
    for (cnt = 0; cnt < l2size; cnt++) {
      pred_dir->config.two.l2table[cnt] = flipflop;
      flipflop = 3 - flipflop;
    }
    break;
  }

  case BPred2bit: {
    if (!l1size || (l1size & (l1size-1)) != 0)
      fatal("2bit table size, `%d', must be non-zero and a power of
two",
            l1size);
    pred_dir->config.bimod.size = l1size;
    if (!(pred_dir->config.bimod.table =
        calloc(l1size, sizeof(unsigned char))))
      fatal("cannot allocate 2bit storage");
    /* initialize counters to weakly this-or-that */
    flipflop = 1;
    for (cnt = 0; cnt < l1size; cnt++) {
      pred_dir->config.bimod.table[cnt] = flipflop;
      flipflop = 3 - flipflop;
    }
    break;
  }

  case BPredBiMode: {
    if (!l1size || (l1size & (l1size-1)) != 0)
      fatal("level-1 size, `%d', must be non-zero and a power of two",
l1size);
      pred_dir->config.bi_mode.l1size = l1size;

    if (!l2size || (l2size & (l2size-1)) != 0)
      fatal("level-2 size, `%d', must be non-zero and a power of two",
          l2size);
    pred_dir->config.bi_mode.l2size = l2size;

    if (!shift_width || shift_width > 30)
      fatal("shift register width, `%d', must be non-zero and
positive",
            shift_width);
    pred_dir->config.bi_mode.shift_width = shift_width;
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
    pred_dir->config.bi_mode.xor = xor;
    pred_dir->config.bi_mode.shiftregs = calloc(l1size, sizeof(int));
    if (!pred_dir->config.bi_mode.shiftregs)
      fatal("cannot allocate shift register table");

    pred_dir->config.bi_mode.Ttable = calloc(l2size, sizeof(unsigned
char));
    if (!pred_dir->config.bi_mode.Ttable)
      fatal("cannot allocate second level Ttable");

    pred_dir->config.bi_mode.NTtable = calloc(l2size, sizeof(unsigned
char));
    if (!pred_dir->config.bi_mode.NTtable)
      fatal("cannot allocate second level NTtable");

    pred_dir->config.bi_mode.choicetable = calloc(l2size,
                                                  sizeof(unsigned
char));
    if (!pred_dir->config.bi_mode.choicetable)
      fatal("cannot allocate second level choicetable");

    /* initialize counters to weakly this-or-that */
    flipflop = 1;
    for (cnt = 0; cnt < l2size; cnt++) {
      pred_dir->config.bi_mode.Ttable[cnt] = 2;
      pred_dir->config.bi_mode.NTtable[cnt] = 1;
      pred_dir->config.bi_mode.choicetable[cnt] = flipflop;
      flipflop = 3 - flipflop;
    }
    break;
  }

  case BPredSkew: {
    if (!l1size || (l1size & (l1size-1)) != 0)
      fatal("level-1 size, `%d', must be non-zero and a power of two",
          l1size);
    pred_dir->config.skew.l1size = l1size;

    if (!l2size || (l2size & (l2size-1)) != 0)
      fatal("level-2 size, `%d', must be non-zero and a power of two",
          l2size);
    pred_dir->config.skew.l2size = l2size;

    if (!shift_width || shift_width > 30)
      fatal("shift register width, `%d', must be non-zero and
positive",
          shift_width);
    pred_dir->config.skew.shift_width = shift_width;

    pred_dir->config.skew.xor = xor;
    pred_dir->config.skew.shiftregs = calloc(l1size, sizeof(int));
    if (!pred_dir->config.skew.shiftregs)
      fatal("cannot allocate shift register table");

    pred_dir->config.skew.bank1 = calloc(l2size, sizeof(unsigned
char));
    if (!pred_dir->config.skew.bank1)
      fatal("cannot allocate second level bank1 table");
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
    pred_dir->config.skew.bank2 = calloc(l2size, sizeof(unsigned
char));
    if (!pred_dir->config.skew.bank2)
      fatal("cannot allocate second level bank2 table");

    pred_dir->config.skew.bank3 = calloc(l2size, sizeof(unsigned
char));
    if (!pred_dir->config.skew.bank3)
      fatal("cannot allocate second level bank3 table");

    /* initialize counters to weakly this-or-that */
    flipflop = 1;
    for (cnt = 0; cnt < l2size; cnt++) {
      pred_dir->config.skew.bank1[cnt] = flipflop;
      pred_dir->config.skew.bank2[cnt] = flipflop;
      pred_dir->config.skew.bank3[cnt] = flipflop;
      flipflop = 3 - flipflop;
    }
    break;
  }

  case BPredTaken:
  case BPredNotTaken:
    /* no other state */
    break;

  default:
    panic("bogus branch direction predictor class");
  }

  return pred_dir;
}

/* create a YAGS branch direction predictor */
struct bpred_dir_t *          /* branch direction predictor instance */
bpred_dir_create_YAGS (enum bpred_class class,  /* type of predictor
                                                   to create */
                       unsigned int l1size,     /* level-1 table size
*/
                       unsigned int l2size,    /* level-2 table size
                                                   (if relevant) */
                       unsigned int shift_width,/* history register
width */
                       unsigned int xor,       /* history xor address
flag */
                       unsigned int tag_size,  /* tag size */
                       unsigned int cache_sets,/* cache sets */
                       unsigned int cache_assoc)/* cache
associativity*/ {

  struct bpred_dir_t *pred_dir;

  unsigned int cnt;
  int flipflop;

  if (!(pred_dir = calloc(1, sizeof(struct bpred_dir_t))))
    fatal("out of virtual memory");

  pred_dir->class = class;
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
  /*Hacemos las cache asociativas por conjuntos, y las encadenamos*/
  int i;

  if (!cache_sets || (cache_sets & (cache_sets-1)) != 0)
    fatal("cache sets, `%d', must be non-zero and a power of two",
          cache_sets);
  if (!cache_assoc || (cache_assoc & (cache_assoc-1)) != 0)
    fatal("cache assoc, `%d', must be non-zero and a power of two",
          cache_assoc);

  if (!(pred_dir->config.yags.NTcache.data =
calloc(cache_sets*cache_assoc,
                                        sizeof(struct
bpred_yagscache_ent_t))))
    fatal("cannot allocate NT cache");
  if (!(pred_dir->config.yags.Tcache.data =
calloc(cache_sets*cache_assoc,
                                        sizeof(struct
bpred_yagscache_ent_t))))
    fatal("cannot allocate T cache");

  pred_dir->config.yags.NTcache.sets = cache_sets;
  pred_dir->config.yags.NTcache.assoc = cache_assoc;
  pred_dir->config.yags.Tcache.sets = cache_sets;
  pred_dir->config.yags.Tcache.assoc = cache_assoc;

  pred_dir->config.yags.last_tag = -1;

  srand(1000);

  if (cache_assoc > 1)
    for (i = 0; i < (cache_assoc*cache_sets); i++) {
      if (((i + 1) % pred_dir->config.yags.NTcache.assoc) != 0){
        pred_dir->config.yags.NTcache.data[i].next =
                                    &pred_dir-
>config.yags.NTcache.data[i + 1];
      pred_dir->config.yags.Tcache.data[i].next =
                                     &pred_dir-
>config.yags.Tcache.data[i + 1];

      }
      else {
      pred_dir->config.yags.NTcache.data[i].next = NULL;
      pred_dir->config.yags.Tcache.data[i].next = NULL;
      }

      if ((i % pred_dir->config.yags.NTcache.assoc) != 0) {
        pred_dir->config.yags.NTcache.data[i].prev = &pred_dir-
>config.yags.NTcache.data[i - 1];
        pred_dir->config.yags.Tcache.data[i].prev = &pred_dir-
>config.yags.Tcache.data[i - 1];
      }
      else {
        pred_dir->config.yags.NTcache.data[i].prev = NULL;
        pred_dir->config.yags.Tcache.data[i].prev = NULL;
      }

    }

  cnt = -1;
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
  switch (class) {
  case BPredYags: {
    pred_dir->config.yags.cachesize = cache_sets * cache_assoc;

    if (!tag_size)
      fatal("tag size, '%d', must be non-zero", tag_size);
    pred_dir->config.yags.tagsize = tag_size;

    if (!l1size || (l1size & (l1size-1)) != 0)
      fatal("level-1 size, `%d', must be non-zero and a power of two",
            l1size);
    pred_dir->config.yags.l1size = l1size;     /*tiene que ser 1 xa
agree*/

    if (!l2size || (l2size & (l2size-1)) != 0)
      fatal("level-2 size, `%d', must be non-zero and a power of two",
           l2size);
    pred_dir->config.yags.l2size = l2size;     /*tiene que ser 4096 xa
agree*/

    if (!shift_width || shift_width > 30)
      fatal("shift register width, `%d', must be non-zero and
positive",
            shift_width);
    pred_dir->config.yags.shift_width = shift_width; /*ancho del
registro de historia,12 xa agree*/

    pred_dir->config.yags.xor = xor;
    pred_dir->config.yags.shiftregs = calloc(l1size, sizeof(int));
    if (!pred_dir->config.yags.shiftregs)
      fatal("cannot allocate shift register table");

    pred_dir->config.yags.choicePHT = calloc(l2size, sizeof(unsigned
char));
    if (!pred_dir->config.yags.choicePHT)
      fatal("cannot allocate second level table");

    /* initialize counters to weakly this-or-that */
    flipflop = 1;
    for (cnt = 0; cnt < l2size; cnt++) {
      pred_dir->config.yags.choicePHT[cnt] = flipflop;
      flipflop = 3 - flipflop;
    }

    break;
  }
  default:
    panic("bogus branch direction predictor class");
  }

  return pred_dir;
}

/* create a branch direction predictor */
struct bpred_dir_t *         /* branch direction predictor instance */
bpred_dir_create_percep (enum bpred_class class,
                                              /* type of predictor
                                                 to create */
                         unsigned int l1size,  /* level-1 table size
*/
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
                          unsigned int l2size,  /* level-2 table size
                                                   (if relevant) */
                          unsigned int shift_width,/* history register
width */
                          unsigned int xor,     /* history xor address
flag */
                          unsigned int threshold)/* threshold value */
{

  struct bpred_dir_t *pred_dir;
  unsigned int cnt, cnt2;

  if (!(pred_dir = calloc(1, sizeof(struct bpred_dir_t))))
    fatal("out of virtual memory");

  pred_dir->class = class;

  cnt = -1;

  switch (class) {
    case BPredPercep: {
      if (!l1size || (l1size & (l1size-1)) != 0)
      fatal("level-1 size, `%d', must be non-zero and a power of two",
            l1size);
      pred_dir->config.percep.l1size = l1size;

      if (!l2size || (l2size & (l2size-1)) != 0)
      fatal("level-2 size, `%d', must be non-zero and a power of two",
            l2size);
      pred_dir->config.percep.l2size = l2size;

      if (!shift_width || shift_width > 30)
      fatal("shift register width, `%d', must be non-zero and
positive",
            shift_width);
      pred_dir->config.percep.shift_width = shift_width;

      pred_dir->config.percep.xor = xor;
      pred_dir->config.percep.shiftregs = calloc(l1size, sizeof(int));
      if (!pred_dir->config.percep.shiftregs)
      fatal("cannot allocate shift register table");

      *pred_dir->config.percep.shiftregs=0;

      int n_percep = (int) (l2size / (shift_width + 1));

      pred_dir->config.percep.n_percep = n_percep;

      pred_dir->config.percep.l2table = calloc(n_percep, sizeof(struct
perceptron_t));
      if (!pred_dir->config.percep.l2table)
      fatal("cannot allocate second level table");

      /* initialize counters to weakly this-or-that */
      for (cnt = 0; cnt < n_percep; cnt++) {
      pred_dir->config.percep.l2table[cnt].resultado = 0;
      pred_dir->config.percep.l2table[cnt].pesos = calloc(shift_width
+ 1, sizeof(int));
      pred_dir->config.percep.l2table[cnt].pesos[0] = 0;
        for (cnt2 = 0; cnt2 < shift_width; cnt2++)
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
        pred_dir->config.percep.l2table[cnt].pesos[cnt2+1] = 0;
      }

      pred_dir->config.percep.threshold = threshold;
      break;
    }
  default:
    panic("bogus branch direction predictor class");
  }

  return pred_dir;
};

/* print branch direction predictor configuration */
void
bpred_dir_config(struct bpred_dir_t *pred_dir, /* branch direction
predictor
                                                  instance */
                 char name[],            /* predictor name */
                 FILE *stream)                /* output stream */ {

  switch (pred_dir->class) {
  case BPred2Level:
    fprintf(stream, "pred_dir: %s: 2-lvl: %d l1-sz, %d bits/ent,"+
                    " %s xor, %d l2-sz, direct-mapped\n", name,
            pred_dir->config.two.l1size, pred_dir-
>config.two.shift_width,
            pred_dir->config.two.xor ? "" : "no", pred_dir-
>config.two.l2size);
    break;

  case BPred2bit:
    fprintf(stream, "pred_dir: %s: 2-bit: %d entries, direct-
mapped\n", name,
            pred_dir->config.bimod.size);
    break;

  case BPredTaken:
    fprintf(stream, "pred_dir: %s: predict taken\n", name);
    break;

  case BPredNotTaken:
    fprintf(stream, "pred_dir: %s: predict not taken\n", name);
    break;

  case BPredAgree:
    fprintf(stream,
            "pred_dir: %s: Agree: %d l1-sz, %d bits/ent, %s xor,"+
            " %d l2-sz, direct-mapped\n", name, pred_dir-
>config.two.l1size,
            pred_dir->config.two.shift_width,
            pred_dir->config.two.xor ? "" : "no", pred_dir-
>config.two.l2size);
    break;

  case BPredBiMode:
    fprintf(stream,
            "pred_dir: %s: bi_mode: %d l1-sz, %d bits/ent, %s xor,"+
            " %d l2-sz, direct-mapped\n", name, pred_dir-
>config.bi_mode.l1size,
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
            pred_dir->config.bi_mode.shift_width,
            pred_dir->config.bi_mode.xor ? "" : "no",
            pred_dir->config.bi_mode.l2size);
    break;

  case BPredSkew:
    fprintf(stream,
            "pred_dir: %s: skew: %d l1-sz, %d bits/ent, %s xor,"+
            " %d l2-sz, direct-mapped\n", name, pred_dir-
>config.skew.l1size,
            pred_dir->config.skew.shift_width,
            pred_dir->config.skew.xor ? "" : "no",
            pred_dir->config.skew.l2size);
    break;

  case BPredFilter:
    fprintf(stream,
            "pred_dir: %s: filter: %d l1-sz, %d bits/ent, %s xor,"+
            " %d l2-sz, direct-mapped\n", name, pred_dir-
>config.two.l1size,
            pred_dir->config.two.shift_width,
            pred_dir->config.two.xor ? "" : "no", pred_dir-
>config.two.l2size);
    break;

  case BPredYags:
    fprintf(stream,
            "pred_dir: %s: yags: %d l1-sz, %d bits/ent, %s xor,"+
            " %d l2-sz, direct-mapped\n", name, pred_dir-
>config.yags.l1size,
            pred_dir->config.yags.shift_width,
            pred_dir->config.yags.xor ? "" : "no",
            pred_dir->config.yags.l2size);
    break;

  case BPredPercep:
    fprintf(stream,
            "pred_dir: %s: percep: %d l1-sz, %d bits/ent, %s xor,"+
            " %d l2-sz, direct-mapped\n", name, pred_dir-
>config.percep.l1size,
            pred_dir->config.percep.shift_width,
            pred_dir->config.percep.xor ? "" : "no",
            pred_dir->config.percep.l2size);
    break;

  default:
    panic("bogus branch direction predictor class");
  }
}

/* print branch predictor configuration */
void
bpred_config(struct bpred_t *pred,  /* branch predictor instance */
           FILE *stream)            /* output stream */ {

  switch (pred->class) {
  case BPredComb:
    bpred_dir_config (pred->dirpred.bimod, "bimod", stream);
    bpred_dir_config (pred->dirpred.twolev, "2lev", stream);
    bpred_dir_config (pred->dirpred.meta, "meta", stream);
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
    fprintf(stream, "btb: %d sets x %d associativity",
          pred->btb.sets, pred->btb.assoc);
    fprintf(stream, "ret_stack: %d entries", pred->retstack.size);
    break;

  case BPred2Level:
    bpred_dir_config (pred->dirpred.twolev, "2lev", stream);
    fprintf(stream, "btb: %d sets x %d associativity",
          pred->btb.sets, pred->btb.assoc);
    fprintf(stream, "ret_stack: %d entries", pred->retstack.size);
    break;

  case BPred2bit:
    bpred_dir_config (pred->dirpred.bimod, "bimod", stream);
    fprintf(stream, "btb: %d sets x %d associativity",
          pred->btb.sets, pred->btb.assoc);
    fprintf(stream, "ret_stack: %d entries", pred->retstack.size);
    break;

  case BPredTaken:
    bpred_dir_config (pred->dirpred.bimod, "taken", stream);
    break;

  case BPredNotTaken:
    bpred_dir_config (pred->dirpred.bimod, "nottaken", stream);
    break;

  case BPredAgree:
    bpred_dir_config (pred->dirpred.twolev, "agree", stream);
    fprintf(stream, "btb: %d sets x %d associativity",
          pred->btb.sets, pred->btb.assoc);
    fprintf(stream, "ret_stack: %d entries", pred->retstack.size);
    break;
  case BPredBiMode:
    bpred_dir_config (pred->dirpred.choice, "BiMode", stream);
    fprintf(stream, "btb: %d sets x %d associativity",
          pred->btb.sets, pred->btb.assoc);
    fprintf(stream, "ret_stack: %d entries", pred->retstack.size);
    break;

  case BPredSkew:
    bpred_dir_config (pred->dirpred.skew, "skew", stream);
    fprintf(stream, "btb: %d sets x %d associativity",
          pred->btb.sets, pred->btb.assoc);
    fprintf(stream, "ret_stack: %d entries", pred->retstack.size);
    break;

   case BPredFilter:
    bpred_dir_config (pred->dirpred.twolev, "filter", stream);
    fprintf(stream, "btb: %d sets x %d associativity",
          pred->btb.sets, pred->btb.assoc);
    fprintf(stream, "ret_stack: %d entries", pred->retstack.size);
    break;

   case BPredYags:
    bpred_dir_config (pred->dirpred.yags, "yags", stream);
    fprintf(stream, "btb: %d sets x %d associativity",
          pred->btb.sets, pred->btb.assoc);
    fprintf(stream, "ret_stack: %d entries", pred->retstack.size);
    break;
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
  case BPredPercep:
    bpred_dir_config (pred->dirpred.percep, "perceptron", stream);
    fprintf(stream, "btb: %d sets x %d associativity",
          pred->btb.sets, pred->btb.assoc);
    fprintf(stream, "ret_stack: %d entries", pred->retstack.size);
    break;

  default:
    panic("bogus branch predictor class");
  }
}

/* print predictor stats */
void
bpred_stats(struct bpred_t *pred,   /* branch predictor instance */
          FILE *stream)       /* output stream */ {

  fprintf(stream, "pred: addr-prediction rate = %f\n",
        (double)pred->addr_hits/(double)(pred->addr_hits+pred-
>misses));
  fprintf(stream, "pred: dir-prediction rate = %f\n",
        (double)pred->dir_hits/(double)(pred->dir_hits+pred->misses));
}

/* register branch predictor stats */
void
bpred_reg_stats(struct bpred_t *pred,    /* branch predictor instance
*/
            struct stat_sdb_t *sdb) /* stats database */ {

  char buf[512], buf1[512], *name;

  /* get a name for this predictor */
  switch (pred->class) {
  case BPredComb:
    name = "bpred_comb";
  break;

  case BPred2Level:
    name = "bpred_2lev";
  break;

  case BPred2bit:
    name = "bpred_bimod";
  break;

  case BPredTaken:
    name = "bpred_taken";
  break;

  case BPredNotTaken:
    name = "bpred_nottaken";
  break;

  case BPredAgree:
    name = "bpred_agree";
  break;

  case BPredBiMode:
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
    name = "bpred_bi_mode";
  break;

  case BPredSkew:
    name = "bpred_skew";
  break;

  case BPredFilter:
    name = "bpred_filter";
  break;

  case BPredYags:
    name = "bpred_yags";
  break;

  case BPredPercep:
    name = "bpred_percep";
  break;

  default:
    panic("bogus branch predictor class");
  }

  sprintf(buf, "%s.suma_total", name);
  sprintf(buf1, "%s.misses + %s.dir_hits", name, name);
  stat_reg_formula(sdb, buf,
                   "Numero total de saltos = saltos acertados +
fallados",
                   buf1, "%9.0f");


  sprintf(buf, "%s.lookups", name);
  stat_reg_counter(sdb, buf, "total number of bpred lookups",
                &pred->lookups, 0, NULL);
  sprintf(buf, "%s.updates", name);
  sprintf(buf1, "%s.dir_hits + %s.misses", name, name);
  stat_reg_formula(sdb, buf, "total number of updates", buf1,
"%12.0f");
  sprintf(buf, "%s.addr_hits", name);
  stat_reg_counter(sdb, buf, "total number of address-predicted hits",
                &pred->addr_hits, 0, NULL);
  sprintf(buf, "%s.dir_hits", name);
  stat_reg_counter(sdb, buf,
                "total number of direction-predicted hits "
                "(includes addr-hits)",
                &pred->dir_hits, 0, NULL);

  if (pred->class == BPredComb) {
    sprintf(buf, "%s.used_bimod", name);
    stat_reg_counter(sdb, buf,
                  "total number of bimodal predictions used",
                  &pred->used_bimod, 0, NULL);
    sprintf(buf, "%s.used_2lev", name);
    stat_reg_counter(sdb, buf,
                  "total number of 2-level predictions used",
                  &pred->used_2lev, 0, NULL);
  }

  if (pred->class == BPredPercep) {
    sprintf(buf, "%s.used_percep", name);
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
    stat_reg_counter(sdb, buf,
                  "total number of perceptron predictions used for
train",
                  &pred->used_percep, 0, NULL);
    sprintf(buf, "%s.dirhits_after_t", name);
    stat_reg_counter(sdb, buf,
                  "direction hits after training",
                  &pred->dirhits_after_training, 0, NULL);
    sprintf(buf, "%s.desthits_after_t", name);
    stat_reg_counter(sdb, buf,
                  "address hits after training",
                  &pred->desthits_after_training, 0, NULL);
    sprintf(buf, "%s.branches_after_t", name);
    stat_reg_counter(sdb, buf,
                  "number of branches after training",
                  &pred->branches_after_training, 0, NULL);
  }

  sprintf(buf, "%s.misses", name);
  stat_reg_counter(sdb, buf, "total number of misses", &pred->misses,
0, NULL);
  sprintf(buf, "%s.jr_hits", name);
  stat_reg_counter(sdb, buf,
                "total number of address-predicted hits for JR's",
                &pred->jr_hits, 0, NULL);
  sprintf(buf, "%s.jr_seen", name);
  stat_reg_counter(sdb, buf,
                "total number of JR's seen",
                &pred->jr_seen, 0, NULL);
  sprintf(buf, "%s.jr_non_ras_hits.PP", name);
  stat_reg_counter(sdb, buf,
                "total number of address-predicted hits for non-RAS
JR's",
                &pred->jr_non_ras_hits, 0, NULL);
  sprintf(buf, "%s.jr_non_ras_seen.PP", name);
  stat_reg_counter(sdb, buf,
                "total number of non-RAS JR's seen",
                &pred->jr_non_ras_seen, 0, NULL);
  sprintf(buf, "%s.bpred_addr_rate", name);
  sprintf(buf1, "%s.addr_hits / %s.updates", name, name);
  stat_reg_formula(sdb, buf,
                "branch address-prediction rate (i.e., addr-
hits/updates)",
                buf1, "%9.4f");
  sprintf(buf, "%s.bpred_dir_rate", name);
  sprintf(buf1, "%s.dir_hits / %s.updates", name, name);
  stat_reg_formula(sdb, buf,
               "branch direction-prediction rate (i.e., all-
hits/updates)",
               buf1, "%9.4f");
  sprintf(buf, "%s.bpred_jr_rate", name);
  sprintf(buf1, "%s.jr_hits / %s.jr_seen", name, name);
  stat_reg_formula(sdb, buf,
               "JR address-prediction rate (i.e., JR addr-hits/JRs
seen)",
               buf1, "%9.4f");
  sprintf(buf, "%s.bpred_jr_non_ras_rate.PP", name);
  sprintf(buf1, "%s.jr_non_ras_hits.PP / %s.jr_non_ras_seen.PP", name,
name);
  stat_reg_formula(sdb, buf,
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
                "non-RAS JR addr-pred rate (ie, non-RAS JR hits/JRs
seen)",
                buf1, "%9.4f");
  sprintf(buf, "%s.retstack_pushes", name);
  stat_reg_counter(sdb, buf,
                "total number of address pushed onto ret-addr stack",
                &pred->retstack_pushes, 0, NULL);
  sprintf(buf, "%s.retstack_pops", name);
  stat_reg_counter(sdb, buf,
                "total number of address popped off of ret-addr stack",
                &pred->retstack_pops, 0, NULL);
  sprintf(buf, "%s.used_ras.PP", name);
  stat_reg_counter(sdb, buf,
                "total number of RAS predictions used",
                &pred->used_ras, 0, NULL);
  sprintf(buf, "%s.ras_hits.PP", name);
  stat_reg_counter(sdb, buf,
                "total number of RAS hits",
                &pred->ras_hits, 0, NULL);
  sprintf(buf, "%s.ras_rate.PP", name);
  sprintf(buf1, "%s.ras_hits.PP / %s.used_ras.PP", name, name);
  stat_reg_formula(sdb, buf,
                "RAS prediction rate (i.e., RAS hits/used RAS)",
                buf1, "%9.4f");
}

void
bpred_after_priming(struct bpred_t *bpred) {

  if (bpred == NULL)
    return;

  bpred->lookups = 0;
  bpred->addr_hits = 0;
  bpred->dir_hits = 0;
  bpred->used_ras = 0;
  bpred->used_bimod = 0;
  bpred->used_percep = 0;
  bpred->dirhits_after_training = 0;
  bpred->desthits_after_training = 0;
  bpred->branches_after_training = 0;
  bpred->used_2lev = 0;
  bpred->jr_hits = 0;
  bpred->jr_seen = 0;
  bpred->misses = 0;
  bpred->retstack_pops = 0;
  bpred->retstack_pushes = 0;
  bpred->ras_hits = 0;
}

#define BIMOD_HASH(PRED, ADDR)                                          \
  ((((ADDR) >> 19) ^ ((ADDR) >> MD_BR_SHIFT)) & ((PRED)-
>config.bimod.size-1))
    /* was: ((baddr >> 16) ^ baddr) & (pred->dirpred.bimod.size-1) */


/*unsigned long long*/
unsigned long long H_1 (unsigned long long Vx, int n) {
  unsigned long long result;
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
  result =(unsigned long long) Vx & ((int)pow(2,n)-1);
  result = (unsigned long long)result*2 + 1;

  return result;
}

unsigned long long H (unsigned long long Vx, int n) {
  int bit_mas, bit_menos, bit_nuevo;
  unsigned long long result;

  bit_mas =(int) Vx >> (n-1);
  bit_menos = (int)(Vx & 1);
  bit_nuevo = bit_mas ^ bit_menos;
  result = (unsigned long long)bit_nuevo*((int)pow(2,n-1)) + (Vx >>
1);
  return result;
}

unsigned long long funcion (md_addr_t baddr, int* history, int n, int
opcion) {
  unsigned long long V, V1, V2;
  unsigned long long result;

  V = (unsigned long long)((baddr >> MD_BR_SHIFT)*(int)pow(2,n)) +
*history;
  V1 = (unsigned long long)*history;
  V2 = (unsigned long long)(baddr >> MD_BR_SHIFT)&((int)(pow(2,n)-1));

  switch (opcion) {
    case 0: {
      result =(unsigned long long)  (H(V1, n)) ^ (H_1(V2, n)) ^ V2;
    }
      break;
    case 1: {
      result =(unsigned long long) H(V1, n) ^ H_1(V2, n) ^ V1;
    }
      break;
    case 2: {
      result =(unsigned long long) H_1(V1, n) ^ H(V2, n) ^ V2;
    }
      break;
    default: result = (unsigned long long)-1;
  }
  return result;
}

/* predicts a branch direction */
char *                                  /* pointer to counter */
bpred_dir_lookup(struct bpred_dir_t *pred_dir, /* branch dir predictor
inst */
             md_addr_t baddr)       /* branch address */ {

  unsigned char *p = NULL;

  /* Except for jumps, get a pointer to direction-prediction bits */
  switch (pred_dir->class) {
    case BPred2Level:
    case BPredAgree:
    case BPredFilter: {
      int l1index, l2index;
```

```
      /* traverse 2-level tables */
      l1index = (baddr >> MD_BR_SHIFT) & (pred_dir->config.two.l1size
- 1);
      l2index = pred_dir->config.two.shiftregs[l1index];
      if (pred_dir->config.two.xor) {
#if 1
        /* this L2 index computation is more "compatible" to
McFarling's
         verison of it, i.e., if the PC xor address component is only
         part of the index, take the lower order address bits for the
         other part of the index, rather than the higher order ones
        */
      l2index = (((l2index ^ (baddr >> MD_BR_SHIFT))
                & ((1 << pred_dir->config.two.shift_width) - 1))
                | ((baddr >> MD_BR_SHIFT)
                << pred_dir->config.two.shift_width));
#else
      l2index = l2index ^ (baddr >> MD_BR_SHIFT);
#endif
      }
      else {
      l2index = l2index | ((baddr >> MD_BR_SHIFT) <<
                           pred_dir->config.two.shift_width);
      }
      l2index = l2index & (pred_dir->config.two.l2size - 1);

      /* get a pointer to prediction state information */
      p = &pred_dir->config.two.l2table[l2index];
    }
    break;

    case BPred2bit:
      p = &pred_dir->config.bimod.table[BIMOD_HASH(pred_dir, baddr)];
    break;

    case BPredBiMode: {
      int l1index, l2index, choiceindex;

      /* traverse 2-level tables */
      l1index = (baddr >> MD_BR_SHIFT) & (pred_dir-
>config.bi_mode.l1size - 1);
      l2index = pred_dir->config.bi_mode.shiftregs[l1index];
      choiceindex = (baddr >> MD_BR_SHIFT) &
                    ((int)(pow(2,pred_dir->config.bi_mode.shift_width)
- 1));
      if (pred_dir->config.bi_mode.xor) {
#if 1
      /* this L2 index computation is more "compatible" to McFarling's
         verison of it, i.e., if the PC xor address component is only
         part of the index, take the lower order address bits for the
         other part of the index, rather than the higher order ones
        */
      l2index = (((l2index ^ (baddr >> MD_BR_SHIFT)) &
                  ((1 << pred_dir->config.bi_mode.shift_width) - 1))
                | ((baddr >> MD_BR_SHIFT)
                << pred_dir->config.bi_mode.shift_width));
#else
      l2index = l2index ^ (baddr >> MD_BR_SHIFT);
#endif
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
      }
      l2index = l2index & (pred_dir->config.bi_mode.l2size - 1);
      /* get a pointer to prediction state information */
      if(pred_dir->config.bi_mode.choicetable[choiceindex] >= 2) {
      p = &pred_dir->config.bi_mode.Ttable[l2index];
      }
      else {
      p = &pred_dir->config.bi_mode.NTtable[l2index];
      }
    }
    break;

    case BPredSkew: {
      int l1index, l2index;
      int bank1index, bank2index, bank3index;

      unsigned long long bank1indexA, bank2indexA, bank3indexA;
      /* traverse 2-level tables */
      l1index = (baddr >> MD_BR_SHIFT) & (pred_dir->config.skew.l1size
- 1);
      l2index = pred_dir->config.skew.shiftregs[l1index];
      if (pred_dir->config.skew.xor) {
#if 1
      /* this L2 index computation is more "compatible" to McFarling's
         verison of it, i.e., if the PC xor address component is only
         part of the index, take the lower order address bits for the
         other part of the index, rather than the higher order ones
       */
      l2index = (((l2index ^ (baddr >> MD_BR_SHIFT))
              & ((1 << pred_dir->config.skew.shift_width) - 1))
              | ((baddr >> MD_BR_SHIFT)
              << pred_dir->config.skew.shift_width));
#else
      l2index = l2index ^ (baddr >> MD_BR_SHIFT);
#endif
      }
      else {
      l2index = l2index |
                  ((baddr >> MD_BR_SHIFT) << pred_dir-
>config.skew.shift_width);
      }

      char *pred1, *pred2, *pred3, *pno, *psi;
      l2index = l2index & (pred_dir->config.skew.l2size - 1);
      bank1indexA = funcion(baddr, pred_dir->config.skew.shiftregs,
                            pred_dir->config.skew.shift_width, 0);
      bank2indexA = funcion(baddr, pred_dir->config.skew.shiftregs,
                            pred_dir->config.skew.shift_width, 1);
      bank3indexA = funcion(baddr, pred_dir->config.skew.shiftregs,
                            pred_dir->config.skew.shift_width, 2);

      bank1index=(int) bank1indexA &
                (unsigned long long)(pow(2,pred_dir-
>config.skew.shift_width)-1);
      bank2index=(int) bank2indexA &
                (unsigned long long)(pow(2,pred_dir-
>config.skew.shift_width)-1);
      bank3index=(int) bank3indexA &
                (unsigned long long)(pow(2,pred_dir-
>config.skew.shift_width)-1);
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
      pred1 = &pred_dir->config.skew.bank1[bank1index];
      pred2 = &pred_dir->config.skew.bank2[bank2index];
      pred3 = &pred_dir->config.skew.bank3[bank3index];

      skew_pred[0] = pred1;
      skew_pred[1] = pred2;
      skew_pred[2] = pred3;

      /* get a pointer to prediction state information */
      int si = 0;
      int no = 0;

      if (*pred1 <= 1) {
      pno = pred1;
      no++;
      }
      else {
      psi = pred1;
      si++;
      }
      if (*pred2 <= 1) {
      pno = pred2;
      no++;
      }
      else {
      psi = pred2;
      si++;
      }
      if (*pred3 <= 1) {
      pno = pred3;
      no++;
      }
      else {
      psi = pred3;
      si++;
      }
      if (si > no)
        p = psi;
      else
      p = pno;
    }
    break;

    case BPredYags: {
      int l1index, l2index, choiceindex, cacheindex;

      /* traverse 2-level tables */
      l1index = (baddr >> MD_BR_SHIFT) & (pred_dir->config.yags.l1size
- 1);
      l2index = pred_dir->config.yags.shiftregs[l1index];
      choiceindex = (baddr >> MD_BR_SHIFT) &
                    ((int)(pow(2,pred_dir->config.yags.shift_width) -
1));

      if (pred_dir->config.yags.xor) {
#if 1
      /* this L2 index computation is more "compatible" to McFarling's
         verison of it, i.e., if the PC xor address component is only
         part of the index, take the lower order address bits for the
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
         other part of the index, rather than the higher order ones
        */
      l2index = (((l2index ^ (baddr >> MD_BR_SHIFT))
             & ((1 << pred_dir->config.yags.shift_width) - 1))
             | ((baddr >> MD_BR_SHIFT)
             << pred_dir->config.yags.shift_width));
#else
      l2index = l2index ^ (baddr >> MD_BR_SHIFT);
#endif
      }
      l2index = l2index & (pred_dir->config.yags.l2size - 1);

      cacheindex = l2index & (pred_dir->config.yags.cachesize-1);

      int tag = (baddr >> MD_BR_SHIFT) &
                ((int)(pow(2,pred_dir->config.yags.tagsize) - 1));
      pred_dir->config.yags.new_tag = tag;
      pred_dir->config.yags.last_tag = -1;

      pred_dir->config.yags.cacheindex = cacheindex;

      int algo = (int)((log(pred_dir-
>config.yags.Tcache.assoc))/log(2));
      int index = cacheindex >> algo;


      /* get a pointer to prediction state information */

      yagsPHT = (char *) &pred_dir-
>config.yags.choicePHT[choiceindex];
      if(pred_dir->config.yags.choicePHT[choiceindex] >= 2){
            if (pred_dir->config.yags.NTcache.assoc > 1){
          index *= pred_dir->config.yags.NTcache.assoc;
          int cambio=0;
        int i;
        for (i = index; i < (index+pred_dir-
>config.yags.NTcache.assoc) ; i++)
          if (pred_dir->config.yags.NTcache.data[i].tag == tag) {
            p = (char *) &pred_dir->config.yags.NTcache.data[i].count;
            pred_dir->config.yags.last_tag = tag;
            cambio=1;
          }
          if(!cambio) {
            p = &pred_dir->config.yags.choicePHT[choiceindex];
          }
      }
      }
      else {
      if (pred_dir->config.yags.Tcache.assoc > 1) {
        index *= pred_dir->config.yags.Tcache.assoc;
        int cambio=0;
        int i;
        for (i = index; i < (index+pred_dir->config.yags.Tcache.assoc)
; i++)
          if (pred_dir->config.yags.Tcache.data[i].tag == tag) {
            p = (char *) &pred_dir->config.yags.Tcache.data[i].count;
            pred_dir->config.yags.last_tag = tag;
            cambio = 1;
          }
          if (!cambio) {
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
            p = &pred_dir->config.yags.choicePHT[choiceindex];
          }
      }
      }
    }
    break;

    case BPredPercep: {
      int l1index, l2index;

      /* traverse 2-level tables */
      l1index = (baddr >> MD_BR_SHIFT) & (pred_dir-
>config.percep.l1size - 1);
      l2index = pred_dir->config.percep.shiftregs[l1index];
      if (pred_dir->config.percep.xor) {
#if 1
      /* this L2 index computation is more "compatible" to McFarling's
         verison of it, i.e., if the PC xor address component is only
         part of the index, take the lower order address bits for the
         other part of the index, rather than the higher order ones
        */
      l2index = (((l2index ^ (baddr >> MD_BR_SHIFT))
             & ((1 << pred_dir->config.percep.shift_width) - 1))
             | ((baddr >> MD_BR_SHIFT)
             << pred_dir->config.percep.shift_width));
#else
      l2index = l2index ^ (baddr >> MD_BR_SHIFT);
#endif
      }
      else {
      l2index = l2index | ((baddr >> MD_BR_SHIFT) <<
                           pred_dir->config.percep.shift_width);
      }
      l2index = l2index & (pred_dir->config.percep.l2size - 1);

      l2index = (baddr >> MD_BR_SHIFT) % (pred_dir-
>config.percep.n_percep);
      pred_dir->config.percep.l2index = l2index;
      pred_dir->config.percep.l1index = l1index;
      int mult;
      int i;
      int mask;
      int *w = &pred_dir->config.percep.l2table[l2index].pesos[0];
      int history = pred_dir->config.percep.shiftregs[l1index];
      int valor = *w++;
      for (mask = 1, i = 0; i < pred_dir->config.percep.shift_width;
i++, mask<<=1,w++)
        if (history & mask)
          valor += *w;
        else
          valor += -*w;

      pred_dir->config.percep.l2table[l2index].resultado = valor;

      /* get a pointer to prediction state information */
      char *numero;
      numero=calloc(1,sizeof(char));
      if(valor<0)
        *numero=0;
      else
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
        *numero=1;
      p=numero;
    }
    break;

    case BPredTaken:
    case BPredNotTaken:
    break;

    default:
      panic("bogus branch direction predictor class");
  }

  return (char *)p;
}

/* probe a predictor for a next fetch address, the predictor is probed
   with branch address BADDR, the branch target is BTARGET (used for
   static predictors), and OP is the instruction opcode (used to
simulate
   predecode bits; a pointer to the predictor state entry (or null for
jumps)
   is returned in *DIR_UPDATE_PTR (used for updating predictor state),
   and the non-speculative top-of-stack is returned in
stack_recover_idx
   (used for recovering ret-addr stack after mis-predict).  */
md_addr_t                     /* predicted branch target addr */
bpred_lookup(struct bpred_t *pred,  /* branch predictor instance */
           md_addr_t baddr,         /* branch address */
           md_addr_t btarget,       /* branch target if taken */
           enum md_opcode op,       /* opcode of instruction */
           int is_call,       /* non-zero if inst is fn call */
           int is_return,           /* non-zero if inst is fn return */
           struct bpred_update_t *dir_update_ptr, /* pred state
pointer */
           int *stack_recover_idx)  /* Non-speculative top-of-stack;
                                     * used on mispredict recovery */ {

  struct bpred_btb_ent_t *pbtb = NULL;
  int index, i;

  if (!dir_update_ptr)
    panic("no bpred update record");

  /* if this is not a branch, return not-taken */
  if (!(MD_OP_FLAGS(op) & F_CTRL))
    return 0;

  pred->lookups++;

  dir_update_ptr->dir.ras = FALSE;
  dir_update_ptr->pdir1 = NULL;
  dir_update_ptr->pdir2 = NULL;
  dir_update_ptr->pmeta = NULL;
  /* Except for jumps, get a pointer to direction-prediction bits */
  switch (pred->class) {
    case BPredComb:
      if ((MD_OP_FLAGS(op) & (F_CTRL|F_UNCOND)) != (F_CTRL|F_UNCOND))
{
      char *bimod, *twolev, *meta;
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
      bimod = bpred_dir_lookup (pred->dirpred.bimod, baddr);
      twolev = bpred_dir_lookup (pred->dirpred.twolev, baddr);
      meta = bpred_dir_lookup (pred->dirpred.meta, baddr);
      dir_update_ptr->pmeta = meta;
      dir_update_ptr->dir.meta  = (*meta >= 2);
      dir_update_ptr->dir.bimod = (*bimod >= 2);
      dir_update_ptr->dir.twolev  = (*twolev >= 2);
      if (*meta >= 2) {
        dir_update_ptr->pdir1 = twolev;
        dir_update_ptr->pdir2 = bimod;
      }
      else {
        dir_update_ptr->pdir1 = bimod;
        dir_update_ptr->pdir2 = twolev;
      }
      }
    break;

    case BPred2Level:
    case BPredAgree:
    case BPredFilter:
      if ((MD_OP_FLAGS(op) & (F_CTRL|F_UNCOND)) != (F_CTRL|F_UNCOND))
{
      dir_update_ptr->pdir1 = bpred_dir_lookup (pred->dirpred.twolev,
baddr);
      }
    break;

    case BPred2bit:
      if ((MD_OP_FLAGS(op) & (F_CTRL|F_UNCOND)) != (F_CTRL|F_UNCOND))
{
      dir_update_ptr->pdir1 = bpred_dir_lookup (pred->dirpred.bimod,
baddr);
      }
    break;

    case BPredTaken:
      return btarget;

    case BPredBiMode:
      if ((MD_OP_FLAGS(op) & (F_CTRL|F_UNCOND)) != (F_CTRL|F_UNCOND))
{
      char *choice;
      struct bpred_dir_t *tipo = pred->dirpred.choice;
      int index = (baddr >> MD_BR_SHIFT) & ((int) pow(2,
                                          tipo-
>config.bi_mode.shift_width) - 1);
      choice = &tipo->config.bi_mode.choicetable[index];
      dir_update_ptr->pchoice = (char *)choice;
      dir_update_ptr->dir.choice  = (*choice >= 2);
      dir_update_ptr->pdir1 = bpred_dir_lookup (pred->dirpred.choice,
baddr);
      }
      break;

    case BPredSkew:
      if ((MD_OP_FLAGS(op) & (F_CTRL|F_UNCOND)) != (F_CTRL|F_UNCOND))
{
      dir_update_ptr->pdir1 = bpred_dir_lookup (pred->dirpred.skew,
baddr);
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
      dir_update_ptr->pbank1 = skew_pred[0];
      dir_update_ptr->pbank2 = skew_pred[1];
      dir_update_ptr->pbank3 = skew_pred[2];
      }
    break;

    case BPredYags:
      if ((MD_OP_FLAGS(op) & (F_CTRL|F_UNCOND)) != (F_CTRL|F_UNCOND))
{
      char *choice;
      struct bpred_dir_t *tipo = pred->dirpred.yags;
      int index = (baddr >> MD_BR_SHIFT) & ((int) pow(2,
                                            tipo-
>config.yags.shift_width) - 1);
      choice = &tipo->config.yags.choicePHT[index];
      dir_update_ptr->pchoice = (char *)choice;
      dir_update_ptr->dir.choice  = (*choice >= 2);
      dir_update_ptr->pdir1 = bpred_dir_lookup (pred->dirpred.yags,
baddr);
      }
    break;

    case BPredPercep:
      if ((MD_OP_FLAGS(op) & (F_CTRL|F_UNCOND)) != (F_CTRL|F_UNCOND))
{
      dir_update_ptr->pdir1 = bpred_dir_lookup (pred->dirpred.percep,
baddr);
      }
    break;

    case BPredNotTaken:
      if ((MD_OP_FLAGS(op) & (F_CTRL|F_UNCOND)) != (F_CTRL|F_UNCOND))
{
      return baddr + sizeof(md_inst_t);
      }
      else {
      return btarget;
      }

    default:
      panic("bogus predictor class");
  }

  /*
   * We have a stateful predictor, and have gotten a pointer into the
   * direction predictor (except for jumps, for which the ptr is null)
   */

  /* record pre-pop TOS; if this branch is executed speculatively
   * and is squashed, we'll restore the TOS and hope the data
   * wasn't corrupted in the meantime. */

  if (pred->retstack.size)
    *stack_recover_idx = pred->retstack.tos;
  else
    *stack_recover_idx = 0;

  /* if this is a return, pop return-address stack */
  if (is_return && pred->retstack.size) {
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
    md_addr_t target = pred->retstack.stack[pred-
>retstack.tos].target;
    pred->retstack.tos = (pred->retstack.tos + pred->retstack.size -
1)
                         % pred->retstack.size;
    pred->retstack_pops++;
    dir_update_ptr->dir.ras = TRUE; /* using RAS here */
    return target;
  }

#ifndef RAS_BUG_COMPATIBLE
  /* if function call, push return-address onto return-address stack
*/
  if (is_call && pred->retstack.size) {
    pred->retstack.tos = (pred->retstack.tos + 1)% pred-
>retstack.size;
    pred->retstack.stack[pred->retstack.tos].target = baddr +
sizeof(md_inst_t);
    pred->retstack_pushes++;
  }
#endif /* !RAS_BUG_COMPATIBLE */

  /* not a return. Get a pointer into the BTB */
  index = (baddr >> MD_BR_SHIFT) & (pred->btb.sets - 1);

  if (pred->btb.assoc > 1) {
    index *= pred->btb.assoc;

      /* Now we know the set; look for a PC match */
    for (i = index; i < (index+pred->btb.assoc) ; i++)
      if (pred->btb.btb_data[i].addr == baddr) {
      /* match */
      pbtb = &pred->btb.btb_data[i];
      break;
      }
  }
  else {
    pbtb = &pred->btb.btb_data[index];
    if (pbtb->addr != baddr)
      pbtb = NULL;
  }

  /*
   * We now also have a pointer into the BTB for a hit, or NULL
otherwise
   */

  /* if this is a jump, ignore predicted direction; we know it's
taken. */
  if ((MD_OP_FLAGS(op) & (F_CTRL|F_UNCOND)) == (F_CTRL|F_UNCOND)) {
    return (pbtb ? pbtb->target : 1);
  }

  /* otherwise we have a conditional branch */
  if (pbtb == NULL) {
    /* BTB miss -- just return a predicted direction */
    int agree;
    if (pred->class != BPredPercep) {
      agree = ((*(dir_update_ptr->pdir1) >= 2)
            ? /* taken */ 1
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
            : /* not taken */ 0);

      if(pred->class == BPredAgree)
      return ((agree == 0) ? 1 : 0);
      else
      return (agree);
    }
    else
      return ((*(dir_update_ptr->pdir1) == 1)
           ? /* taken */ 1
           : /* not taken */ 0);
  }
  else {
    int bias;
    if (pred->class == BPredAgree) {
      int agree = ((*(dir_update_ptr->pdir1) >= 2)
                ?  1
                :  0);

      bias=pbtb->bias;
      return ((agree == bias) ? pbtb->target :0);
    }

    else if(pred->class == BPredFilter) {
      int filter=pbtb->countFilter;
      bias=pbtb->bias;
      if(filter==pred->dirpred.twolev->config.two.l2size-1) {
      return (bias ? pbtb->target :0);
      printf("El filter se ha saturado\n");
      }

      else
      return ((*(dir_update_ptr->pdir1) >= 2)
            ? /* taken */ pbtb->target
            : /* not taken */ 0);
    }

    else if (pred->class == BPredPercep) {
      int l2index = pred->dirpred.percep->config.percep.l2index;
      if (pred->dirpred.percep-
>config.percep.l2table[l2index].resultado >= 0)
      return pbtb->target;
      else
        return 0;
    }

    else {
      /* BTB hit, so return target if it's a predicted-taken branch */
      return ((*(dir_update_ptr->pdir1) >= 2)
            ? /* taken */ pbtb->target
            : /* not taken */ 0);
    }
  }
}

/* Speculative execution can corrupt the ret-addr stack.  So for each
 * lookup we return the top-of-stack (TOS) at that point; a
mispredicted
 * branch, as part of its recovery, restores the TOS using this value
--
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
 * hopefully this uncorrupts the stack. */
void
bpred_recover(struct bpred_t *pred, /* branch predictor instance */
              md_addr_t baddr,      /* branch address */
              int stack_recover_idx)  /* Non-speculative top-of-stack;
                                       * used on mispredict recovery */ {

  if (pred == NULL)
    return;

  pred->retstack.tos = stack_recover_idx;
}

/* update the branch predictor, only useful for stateful predictors;
updates
   entry for instruction type OP at address BADDR.  BTB only gets
updated
   for branches which are taken.  Inst was determined to jump to
   address BTARGET and was taken if TAKEN is non-zero.  Predictor
   statistics are updated with result of prediction, indicated by
CORRECT and
   PRED_TAKEN, predictor state to be updated is indicated by
*DIR_UPDATE_PTR
   (may be NULL for jumps, which shouldn't modify state bits).  Note
if
   bpred_update is done speculatively, branch-prediction may get
polluted. */
void
bpred_update(struct bpred_t *pred,  /* branch predictor instance */
             md_addr_t baddr,       /* branch address */
             md_addr_t btarget,     /* resolved branch target */
             int taken,             /* non-zero if branch was taken */
             int pred_taken,        /* non-zero if branch was pred
taken */
             int correct,       /* was earlier addr prediction ok? */
             enum md_opcode op,     /* opcode of instruction */
             struct bpred_update_t *dir_update_ptr)/* pred state pointer
*/ {

  struct bpred_btb_ent_t *pbtb = NULL;
  struct bpred_btb_ent_t *lruhead = NULL, *lruitem = NULL;
  int index, i;

  int predBias=0;

  /* don't change bpred state for non-branch instructions or if this
   * is a stateless predictor*/
  if (!(MD_OP_FLAGS(op) & F_CTRL))
    return;

  /* Have a branch here */
  if (correct)
    pred->addr_hits++;

  if (!!pred_taken == !!taken)
    pred->dir_hits++;
  else
    pred->misses++;

  if (dir_update_ptr->dir.ras) {
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
    pred->used_ras++;
    if (correct)
      pred->ras_hits++;
  }
  else if ((MD_OP_FLAGS(op) & (F_CTRL|F_COND)) == (F_CTRL|F_COND)) {
    if (dir_update_ptr->dir.meta)
      pred->used_2lev++;
    else
      pred->used_bimod++;
  }

  /* keep stats about JR's; also, but don't change any bpred state for
JR's
   * which are returns unless there's no retstack */
  if (MD_IS_INDIR(op)) {
    pred->jr_seen++;
    if (correct)
      pred->jr_hits++;

    if (!dir_update_ptr->dir.ras) {
      pred->jr_non_ras_seen++;
      if (correct)
      pred->jr_non_ras_hits++;
    }
    else {
      /* return that used the ret-addr stack; no further work to do */
      return;
    }
  }

  /* Can exit now if this is a stateless predictor */
  if (pred->class == BPredNotTaken || pred->class == BPredTaken)
    return;

  /*
   * Now we know the branch didn't use the ret-addr stack, and that
this
   * is a stateful predictor
  */

#ifdef RAS_BUG_COMPATIBLE
  /* if function call, push return-address onto return-address stack
*/
  if (MD_IS_CALL(op) && pred->retstack.size) {
    pred->retstack.tos = (pred->retstack.tos + 1)% pred-
>retstack.size;
    pred->retstack.stack[pred->retstack.tos].target = baddr +
sizeof(md_inst_t);
    pred->retstack_pushes++;
  }
#endif /* RAS_BUG_COMPATIBLE */

  /* update L1 table if appropriate */
  /* L1 table is updated unconditionally for combining predictor too
*/
  if ((MD_OP_FLAGS(op) & (F_CTRL|F_UNCOND)) != (F_CTRL|F_UNCOND) &&
      (pred->class == BPred2Level || pred->class == BPredComb ||
       pred->class == BPredAgree || pred->class == BPredFilter)) {
    int l1index, shift_reg;
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
    /* also update appropriate L1 history register */
    l1index = (baddr >> MD_BR_SHIFT) &
              (pred->dirpred.twolev->config.two.l1size - 1);
    shift_reg = (pred->dirpred.twolev->config.two.shiftregs[l1index]
<< 1) |
                (!!taken);
    pred->dirpred.twolev->config.two.shiftregs[l1index] = shift_reg &
                      ((1 << pred->dirpred.twolev-
>config.two.shift_width) - 1);
  }
  if ((MD_OP_FLAGS(op) & (F_CTRL|F_UNCOND)) != (F_CTRL|F_UNCOND) &&
      (pred->class == BPredSkew)) {

    int l1index, shift_reg;

    /* also update appropriate L1 history register */
    l1index = (baddr >> MD_BR_SHIFT) &
              (pred->dirpred.skew->config.skew.l1size - 1);
    shift_reg = (pred->dirpred.skew->config.skew.shiftregs[l1index] <<
1) |
                (!!taken);
    pred->dirpred.skew->config.skew.shiftregs[l1index] = shift_reg &
                        ((1 << pred->dirpred.skew-
>config.skew.shift_width) - 1);
  }

  if ((MD_OP_FLAGS(op) & (F_CTRL|F_UNCOND)) != (F_CTRL|F_UNCOND) &&
      (pred->class == BPredBiMode)) {

    int l1index, shift_reg;

    /* also update appropriate L1 history register */
    l1index = (baddr >> MD_BR_SHIFT) &
              (pred->dirpred.choice->config.bi_mode.l1size - 1);
    shift_reg = (pred->dirpred.choice-
>config.bi_mode.shiftregs[l1index] << 1) |
                (!!taken);
    pred->dirpred.choice->config.bi_mode.shiftregs[l1index] =
shift_reg &
                  ((1 << pred->dirpred.choice-
>config.bi_mode.shift_width) - 1);
  }

  if ((MD_OP_FLAGS(op) & (F_CTRL|F_UNCOND)) != (F_CTRL|F_UNCOND) &&
      (pred->class == BPredYags)) {

    int l1index, shift_reg;

    /* also update appropriate L1 history register */
    l1index = (baddr >> MD_BR_SHIFT) &
              (pred->dirpred.yags->config.yags.l1size - 1);
    shift_reg = (pred->dirpred.yags->config.yags.shiftregs[l1index] <<
1) |
                (!!taken);
    pred->dirpred.yags->config.yags.shiftregs[l1index] = shift_reg &
                         (1 << pred->dirpred.yags-
>config.yags.shift_width) - 1);
  }

  if ((MD_OP_FLAGS(op) & (F_CTRL|F_UNCOND)) != (F_CTRL|F_UNCOND) &&
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
        (pred->class==BPredPercep)) {
    int l1index, shift_reg;

    /* also update appropriate L1 history register */
    l1index = (baddr >> MD_BR_SHIFT) &
              (pred->dirpred.percep->config.percep.l1size - 1);
    shift_reg = (pred->dirpred.percep-
>config.percep.shiftregs[l1index] << 1) |
                 (!!taken);
    pred->dirpred.percep->config.percep.shiftregs[l1index] =
    shift_reg & ((1 << pred->dirpred.percep-
>config.percep.shift_width) - 1);
  }

  /* find BTB entry if it's a taken branch (don't allocate for non-
taken) */
  if((pred->class != BPredAgree)&&(pred->class != BPredFilter)) {
    if (taken) {
      index = (baddr >> MD_BR_SHIFT) & (pred->btb.sets - 1);

      if (pred->btb.assoc > 1) {
        index *= pred->btb.assoc;

      /* Now we know the set; look for a PC match; also identify
       * MRU and LRU items */
      for (i = index; i < (index+pred->btb.assoc) ; i++) {
        if (pred->btb.btb_data[i].addr == baddr) {
          /* match */
          assert(!pbtb);
          pbtb = &pred->btb.btb_data[i];
        }

        dassert(pred->btb.btb_data[i].prev != pred-
>btb.btb_data[i].next);
        if (pred->btb.btb_data[i].prev == NULL) {
          /* this is the head of the lru list, ie current MRU item */
          dassert(lruhead == NULL);
          lruhead = &pred->btb.btb_data[i];
        }
        if (pred->btb.btb_data[i].next == NULL) {
          /* this is the tail of the lru list, ie the LRU item */
          dassert(lruitem == NULL);
          lruitem = &pred->btb.btb_data[i];
        }
      }
      dassert(lruhead && lruitem);

      if (!pbtb) {
        /* missed in BTB; choose the LRU item in this set as the
victim */
        pbtb = lruitem;
        predBias=0;
      }
      else { /* else hit, and pbtb points to matching BTB entry */
        predBias=1;
      }

        /* Update LRU state: selected item, whether selected because
it
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
       * matched or because it was LRU and selected as a victim,
becomes
       * MRU */

      if (pbtb != lruhead) {
        /* this splices out the matched entry... */
        if (pbtb->prev)
          pbtb->prev->next = pbtb->next;
        if (pbtb->next)
          pbtb->next->prev = pbtb->prev;
        /* ...and this puts the matched entry at the head of the list
*/
        pbtb->next = lruhead;
        pbtb->prev = NULL;
        lruhead->prev = pbtb;
        dassert(pbtb->prev || pbtb->next);
        dassert(pbtb->prev != pbtb->next);
      }
        /* else pbtb is already MRU item; do nothing */
      }
      else
      pbtb = &pred->btb.btb_data[index];
    }
  }
  else {
    index = (baddr >> MD_BR_SHIFT) & (pred->btb.sets - 1);

    if (pred->btb.assoc > 1) {
      index *= pred->btb.assoc;

      /* Now we know the set; look for a PC match; also identify
       * MRU and LRU items */
      for (i = index; i < (index+pred->btb.assoc) ; i++) {
        if (pred->btb.btb_data[i].addr == baddr) {
        /* match */
        assert(!pbtb);
        pbtb = &pred->btb.btb_data[i];
        }

        dassert(pred->btb.btb_data[i].prev != pred-
>btb.btb_data[i].next);
        if (pred->btb.btb_data[i].prev == NULL) {
        /* this is the head of the lru list, ie current MRU item */
        dassert(lruhead == NULL);
        lruhead = &pred->btb.btb_data[i];
        }
        if (pred->btb.btb_data[i].next == NULL) {
        /* this is the tail of the lru list, ie the LRU item */
        dassert(lruitem == NULL);
        lruitem = &pred->btb.btb_data[i];
        }
      }
      dassert(lruhead && lruitem);

      if (!pbtb)
        /* missed in BTB; choose the LRU item in this set as the
victim */
        pbtb = lruitem;
      /* else hit, and pbtb points to matching BTB entry */
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
      /* Update LRU state: selected item, whether selected because it
       * matched or because it was LRU and selected as a victim,
becomes
       * MRU */
      if (pbtb != lruhead) {
        /* this splices out the matched entry... */
        if (pbtb->prev)
          pbtb->prev->next = pbtb->next;
        if (pbtb->next)
          pbtb->next->prev = pbtb->prev;
        /* ...and this puts the matched entry at the head of the list
*/
        pbtb->next = lruhead;
        pbtb->prev = NULL;
        lruhead->prev = pbtb;
        dassert(pbtb->prev || pbtb->next);
        dassert(pbtb->prev != pbtb->next);
      }
      /* else pbtb is already MRU item; do nothing */
    }
    else
      pbtb = &pred->btb.btb_data[index];
  }

  /*
   * Now 'p' is a possibly null pointer into the direction prediction
table,
   * and 'pbtb' is a possibly null pointer into the BTB (either to a
   * matched-on entry or a victim which was LRU in its set)
   */
  /* update state (but not for jumps) */
  if (dir_update_ptr->pdir1) {
    int filter;
    if (pred->class==BPredFilter) {
      if (pbtb->addr == baddr) {
      predBias = pbtb->bias;
      if (pbtb->countFilter==pred->dirpred.twolev->config.two.l2size-
1)
        filter=1;
      else
        filter=0;
      }
      else {
      predBias=taken;
      filter=0;
      }

      if (taken) {
      if (filter == 1) {
        if (predBias == 0) {
          pbtb->countFilter=0;
          pbtb->bias = 1;
        }
      }
      else {
        if (predBias==1) {
          if (pbtb->countFilter<pred->dirpred.twolev-
>config.two.l2size-1)
            pbtb->countFilter++;
        }
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
        else {
          pbtb->countFilter=0;
          pbtb->bias=1;
        }
        if (*dir_update_ptr->pdir1 < 3)
          ++*dir_update_ptr->pdir1;
      }
      }
      else {
        if(filter == 1){
        if (predBias == 1) {
          pbtb->countFilter = 0;
          pbtb->bias = 0;
        }
        }
      else {
        if (predBias == 0) {
          if (pbtb->countFilter < pred->dirpred.twolev-
>config.two.l2size-1)
            pbtb->countFilter++;
        }
        else {
          pbtb->countFilter=0;
          pbtb->bias=0;
        }
        if (*dir_update_ptr->pdir1 > 0)
          --*dir_update_ptr->pdir1;
      }
      }
    }
    else if (pred->class == BPredSkew) {
      if (!correct) {
        if (pred_taken) {
        if (*dir_update_ptr->pbank1 > 0)
          --*dir_update_ptr->pbank1;
        if (*dir_update_ptr->pbank2 > 0)
          --*dir_update_ptr->pbank2;
        if (*dir_update_ptr->pbank3 > 0)
          --*dir_update_ptr->pbank3;
      }
        else { /* not pred_taken */
        if (*dir_update_ptr->pbank1 < 3)
          ++*dir_update_ptr->pbank1;
        if (*dir_update_ptr->pbank2 < 3)
          ++*dir_update_ptr->pbank2;
        if (*dir_update_ptr->pbank3 < 3)
          ++*dir_update_ptr->pbank3;
      }
      }
      else { /* if correct */
        if (pred_taken) {
        if ((*dir_update_ptr->pbank1 > 1)&&(*dir_update_ptr->pbank1 <
3))
          ++*dir_update_ptr->pbank1;
        if ((*dir_update_ptr->pbank2 > 1)&&(*dir_update_ptr->pbank2 <
3))
          ++*dir_update_ptr->pbank2;
        if ((*dir_update_ptr->pbank3 > 1)&&(*dir_update_ptr->pbank3 <
3))
          ++*dir_update_ptr->pbank3;
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
      }
      else {/*not pred_taken*/
        if ((*dir_update_ptr->pbank1 < 2)&&(*dir_update_ptr->pbank1 >
0))
          --*dir_update_ptr->pbank1;
        if ((*dir_update_ptr->pbank2 < 2)&&(*dir_update_ptr->pbank2 >
0))
          --*dir_update_ptr->pbank2;
        if ((*dir_update_ptr->pbank3 < 2)&&(*dir_update_ptr->pbank3 >
0))
          --*dir_update_ptr->pbank3;
      }
      }
    }
    else if (pred->class == BPredPercep) {
      int t, signoY, y, threshold;
      int l2index = pred->dirpred.percep->config.percep.l2index;
      int l1index = pred->dirpred.percep->config.percep.l1index;
      struct perceptron_t *perceptron =
                           &pred->dirpred.percep-
>config.percep.l2table[l2index];
      int i, mult;

      if (taken)
      t = 1;
      else
      t = -1;
      y = perceptron->resultado;
      if (y < 0)
        signoY = -1;
      else
      signoY = 1;
      threshold = pred->dirpred.percep->config.percep.threshold;

      if ((t != signoY) || ((abs(y)) <= threshold)) {
      int mask;
      int *w = &perceptron->pesos[0];
      if (taken)
        (*w)++;
      else
        (*w)--;
      if (*w > MAX_PESO) *w = MAX_PESO;
        if (*w < MIN_PESO) *w = MIN_PESO;
          w++;
      int history = pred->dirpred.percep-
>config.percep.shiftregs[l1index];
      for (mask = 1, i = 0; i <
             pred->dirpred.percep->config.percep.shift_width; i++,
             mask<<=1,w++) {
        if (!!(history & mask) == taken) {
          (*w)++;
          if (*w > MAX_PESO) *w = MAX_PESO;
        }
        else {
          (*w)--;
          if (*w < MIN_PESO) *w = MIN_PESO;
        }
      }
      }
      else {
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
      pred->branches_after_training++;
      if (taken==pred_taken)
        pred->dirhits_after_training++;
      if(correct)
        pred->desthits_after_training++;
      }

    }
    else {
      if (pred->class == BPredAgree) {
      if (pbtb->addr == baddr) {
        predBias = pbtb->bias;
      }
      else {
        predBias=taken;
      }
      }
      if (taken) {
      if (pred->class == BPredAgree) {
        if (predBias == 1) {
          if (*dir_update_ptr->pdir1 < 3)
            ++*dir_update_ptr->pdir1;
        }
        else {
          if (*dir_update_ptr->pdir1 > 0)
            --*dir_update_ptr->pdir1;
        }
      }
      else if (pred->class == BPredYags) {
        if (*yagsPHT < 3)
          ++*yagsPHT;
      }
      else {
        if (*dir_update_ptr->pdir1 < 3)
          ++*dir_update_ptr->pdir1;
      }
      }
      else {
      if (pred->class == BPredAgree) {
        if (predBias == 0) {
          if (*dir_update_ptr->pdir1 < 3)
            ++*dir_update_ptr->pdir1;
        }
        else {
          if (*dir_update_ptr->pdir1 > 0)
            --*dir_update_ptr->pdir1;
        }
        }
        else if (pred->class == BPredYags){
        if (*yagsPHT > 0)
          --*yagsPHT;
      }
      else {
        if (*dir_update_ptr->pdir1 > 0)
          --*dir_update_ptr->pdir1;
      }
      }
    }
  }
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
  /* combining predictor also updates second predictor and meta
predictor */
  /* second direction predictor */
  if (dir_update_ptr->pdir2) {
    if (taken) {
      if (*dir_update_ptr->pdir2 < 3)
      ++*dir_update_ptr->pdir2;
    }
    else { /* not taken */
      if (*dir_update_ptr->pdir2 > 0)
      --*dir_update_ptr->pdir2;
    }
  }

  /* meta predictor */
  if (dir_update_ptr->pmeta) {
    if (dir_update_ptr->dir.bimod != dir_update_ptr->dir.twolev) {
      /* we only update meta predictor if directions were different */
      if (dir_update_ptr->dir.twolev == (unsigned int)taken) {
      /* 2-level predictor was correct */
      if (*dir_update_ptr->pmeta < 3)
        ++*dir_update_ptr->pmeta;
      }
      else {
      /* bimodal predictor was correct */
      if (*dir_update_ptr->pmeta > 0)
        --*dir_update_ptr->pmeta;
      }
    }
  }

  /* update BTB (but only for taken branches) */
  if((pred->class != BPredAgree)&&(pred->class != BPredFilter)) {
    if (dir_update_ptr->pchoice) {
      if ((pred->class == BPredBiMode) || (pred->class == BPredYags))
{
      if (taken) {
          if (*dir_update_ptr->pchoice < 3)
                ++*dir_update_ptr->pchoice;
      }
      else {
        if (*dir_update_ptr->pchoice > 0)
                --*dir_update_ptr->pchoice;
      }
      }
    }
  }
  if (pred->class == BPredYags) {

    int choicePred = dir_update_ptr->dir.choice;
    int cacheindex = pred->dirpred.yags->config.yags.cacheindex;
    int acierto;
    int tag = pred->dirpred.yags->config.yags.last_tag;

    if (!choicePred) {
      if (tag == -1) {
      struct bpred_yagscache_ent_t *lrucachei=NULL ,*lrucachef=NULL;
      struct bpred_yagscache_ent_t *pcache=NULL;

      if (!correct) {
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
        int algo = (int)((log(pred->dirpred.yags-
>config.yags.Tcache.assoc)) /
                                log(2));
        int index = pred->dirpred.yags->config.yags.cacheindex >>
algo;
        if (pred->dirpred.yags->config.yags.Tcache.assoc > 1)    {
          index *= pred->dirpred.yags->config.yags.Tcache.assoc;
          int i;
          for (i = index; i <
                 (index+pred->dirpred.yags->config.yags.Tcache.assoc);
i++) {
            if (pred->dirpred.yags->config.yags.Tcache.data[i].prev ==
NULL)
            lrucachei = &pred->dirpred.yags-
>config.yags.Tcache.data[i];
            if (pred->dirpred.yags->config.yags.Tcache.data[i].next ==
NULL)
              lrucachef = &pred->dirpred.yags-
>config.yags.Tcache.data[i];
          }
          struct bpred_yagscache_ent_t *aux = NULL;
          aux = lrucachei;
          while ((aux->next != NULL) && (!pcache)) {
             if (aux->count <= 1)
             pcache = aux;
             else
             aux = aux->next;
          }
          if (!pcache) {
            lrucachef->prev->next = NULL;
            pcache = lrucachef;
            pcache->next = lrucachei;
            pcache->prev = NULL;
            lrucachei->prev = pcache;
            pcache->count = 2;
            pcache->tag = pred->dirpred.yags->config.yags.new_tag;
          }
          else {
            pcache->count = 2;
            pcache->tag = pred->dirpred.yags->config.yags.new_tag;
          }
        }
      }
      }
      else {
        struct bpred_yagscache_ent_t *lrucachei=NULL, *lrucachef=NULL;
      struct bpred_yagscache_ent_t *pcache=NULL;

      int algo = (int)((log(pred->dirpred.yags-
>config.yags.Tcache.assoc)) /
                           log(2));
      int index = pred->dirpred.yags->config.yags.cacheindex >> algo;

        if (pred->dirpred.yags->config.yags.Tcache.assoc > 1) {
        index *= pred->dirpred.yags->config.yags.Tcache.assoc;
        int i;
        for (i = index; i <
                    (index+pred->dirpred.yags-
>config.yags.Tcache.assoc); i++) {
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
          if (pred->dirpred.yags->config.yags.Tcache.data[i].tag ==
tag) {
            pred->dirpred.yags->config.yags.cacheindex = i;
            pcache = &pred->dirpred.yags->config.yags.Tcache.data[i];
            acierto = 1;
          }
          if (pred->dirpred.yags->config.yags.Tcache.data[i].prev ==
NULL) {
            lrucachei = &pred->dirpred.yags-
>config.yags.Tcache.data[i];
          }
          if (pred->dirpred.yags->config.yags.Tcache.data[i].next ==
NULL)
            lrucachef = &pred->dirpred.yags-
>config.yags.Tcache.data[i];
        } /*end for*/

        if (!pcache) {
          pcache = lrucachef;
          acierto = 0;
        }
        if (pcache != lrucachei) {
          if (pcache->prev)
            pcache->prev->next = pcache->next;
          if (pcache->next)
            pcache->next->prev = pcache->prev;
          pcache->next = lrucachei;
          pcache->prev = NULL;
          lrucachei->prev = pcache;
        }
        if (acierto) {
            if ((taken) && (pcache->count < 3))
            ++pcache->count;
          else if ((!taken) && (pcache->count > 0))
            --pcache->count;
        }
        else {
          if (!correct) {
            pcache->count = 2;
            pcache->tag = tag;
          }
        }
      }
        else
        pcache = &pred->dirpred.yags->config.yags.Tcache.data[i];
      }
    }
    else
      if (tag == -1) {
      struct bpred_yagscache_ent_t *lrucachei=NULL, *lrucachef = NULL;
      struct bpred_yagscache_ent_t *pcache = NULL;

      if (!correct) {
        int algo = (int)((log(pred->dirpred.yags-
>config.yags.NTcache.assoc))/
                         log(2));
        int index = pred->dirpred.yags->config.yags.cacheindex >>
algo;
        if (pred->dirpred.yags->config.yags.NTcache.assoc > 1) {
          index *= pred->dirpred.yags->config.yags.NTcache.assoc;
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
          int i;
          for (i = index; i <
                   (index+pred->dirpred.yags-
>config.yags.NTcache.assoc); i++) {
            if (pred->dirpred.yags->config.yags.NTcache.data[i].prev
== NULL)
            lrucachei = &pred->dirpred.yags-
>config.yags.NTcache.data[i];
            if (pred->dirpred.yags->config.yags.NTcache.data[i].next
== NULL)
            lrucachef = &pred->dirpred.yags-
>config.yags.NTcache.data[i];
          }
          struct bpred_yagscache_ent_t *aux = NULL;
          aux = lrucachei;
          while ((aux->next != NULL) && (!pcache)) {
            if (aux->count >= 2)
            pcache = aux;
            else
            aux = aux->next;
          }
          if (!pcache) {
            lrucachef->prev->next = NULL;
            pcache = lrucachef;
            pcache->next = lrucachei;
            pcache->prev = NULL;
            lrucachei->prev = pcache;
            pcache->count = 1;
            pcache->tag = pred->dirpred.yags->config.yags.new_tag;
          }
          else {
            pcache->count = 1;
            pcache->tag = pred->dirpred.yags->config.yags.new_tag;
          }
        }
      }
      }
      else {
        struct bpred_yagscache_ent_t *lrucachei=NULL, *lrucachef =
NULL;
      struct bpred_yagscache_ent_t *pcache = NULL;
      int algo = (int)((log(pred->dirpred.yags-
>config.yags.Tcache.assoc)) /
                           log(2));
      int index = pred->dirpred.yags->config.yags.cacheindex >> algo;

        if (pred->dirpred.yags->config.yags.NTcache.assoc > 1)   {
        index *= pred->dirpred.yags->config.yags.NTcache.assoc;
        int i;
        for (i = index; i <
                   (index+pred->dirpred.yags-
>config.yags.NTcache.assoc); i++) {
          if (pred->dirpred.yags->config.yags.NTcache.data[i].tag ==
tag) {
            pred->dirpred.yags->config.yags.cacheindex = i;
            pcache = &pred->dirpred.yags->config.yags.NTcache.data[i];
            acierto = 1;
          }
          if (pred->dirpred.yags->config.yags.NTcache.data[i].prev ==
NULL) {
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
            lrucachei = &pred->dirpred.yags-
>config.yags.NTcache.data[i];
          }
          if (pred->dirpred.yags->config.yags.NTcache.data[i].next ==
NULL)
            lrucachef = &pred->dirpred.yags-
>config.yags.NTcache.data[i];
        }
        if (!pcache) {
          pcache = lrucachef;
          acierto = 0;
        }
        if (pcache != lrucachei) {
          if (pcache->prev)
            pcache->prev->next = pcache->next;
          if (pcache->next)
            pcache->next->prev = pcache->prev;

          pcache->next = lrucachei;
          pcache->prev = NULL;
          lrucachei->prev = pcache;
        }
        if (acierto) {
            if ((taken) && (pcache->count < 3))
            ++pcache->count;
          else if ((!taken) && (pcache->count > 0))
            --pcache->count;
        }
        else {
          if (!correct) {
            pcache->count = 1;
            pcache->tag = tag;
          }
        }
      }
        else
        pcache = &pred->dirpred.yags->config.yags.NTcache.data[i];
      }
    }
  }
  if (pbtb) {
    if (pbtb->addr == baddr) {
      if (!correct)
      pbtb->target = btarget;
    }
    else {
      /* enter a new branch in the table */
      pbtb->addr = baddr;
      pbtb->op = op;
      pbtb->target = btarget;
      pbtb->bias=1;
    }
  }
  else {
    if (pbtb) {
      /* update current information */
      if (taken)
      dassert(taken);

      if (pbtb->addr == baddr) {
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

```
      if (!correct)
        pbtb->target = btarget;
      }
      else {
        /* enter a new branch in the table */
      pbtb->addr = baddr;
      pbtb->op = op;
      pbtb->target = btarget;
      pbtb->bias=taken;
      if (pred->class == BPredFilter)
        pbtb->countFilter = 0;
      }
    }
  }
}
```


Javier García-Heras Jiménez
David Martín Baz
Felicidad Ramos Manjavacas

BIBLIOGRAFÍA GENERAL

[1] *http://www.specbench.org/osg/cpu2000/*

[2] *http://es.wikipedia.org*

[3] Carlos García Sánchez, *Evaluación de Rendimiento de Configuraciones (ERC), Universidad Complutense, 2003*

[4] Sébastien Nussbaum, James E. Smith, *Modeling Superscalar Processors via Statistical Simulation*. Department of Electrical and Computer Engineering,University of Wisconsin.

[5] Todd Austin, Dan Ernst, Eric Larson, Chris Weaver. University of Michigan, *SimpleScalar Tutorial (for release 4.0).*

[6] *SimpleScalar source. bpred.c, bpred.h, sim-bpred.c.*

[7] Scott McFarling , *CombiningBranch Predictors*, WRL Technical Note TN-36, June 1993.

[8] A. N. Eden and T. Mudge ,*The YAGS Branch Prediction Scheme.*

[9] T.-Y. Yeh and Y. Patt. *A Comparison of Dynamic Branch. Predictors that use Two Level of Branch History*. Proc. 20th. Ann. Int. Symp. on Computer Architecture, May 1993.

[10] Tse-Yu Yeh and Yale N. Patt, *Alternative Implementations of Two_Level. Adaptive Branch Prediction Pp 124 -134,* The 19th Annual International. Symposium on Computer Architecture, May 1992 Gold Coast, Australia.

[11] "The Agree Predictor: A Mechanism for Reducing Negative Branch History Interference". Eric Spranglezy Robert S. Chappellyz Mitch Alsupz Yale N. Patty

[12] "Advanced Computer Architecture Laboratory". Department of Electrical Engineering and Computer Science. The University of Michigan.

[13] Presentación en power point: www.ece.rochester.edu/ www.ece.rochester.edu/~mihuang/TEACHING/OLD/ECE404_SPRING03/Branch%20Prediction.ppt

[14] "The Bi-Mode Branch Predictor". Chih-Chieh Lee, I-Cheng K. Chen, and Trevor N. Mudge. EECS Department, University of Michigan.

[15] P. Michaud, *A. Seznec, R. Uhlig Trading Conflict and Capacity Aliasing in Conditional Branch Predictors.* 24th Annual Symposium on Computer Architecture, 1997

[16] M.D. Hill *Aspects of Cache Memory and Instruction Buffer Performance*. PhD thesis, University of California, Berkeley, 1987.

[17] A. Seznec *A case for two-way skewed associative caches.* Proceedings of the 20th Annual Symposium on Computer Architecture.

[18] *"Design and Performance Evaluation of Global History Dynamic Branch. Predictors":* Chih-Chieh Lee (Digital Equipment Corp.), I-Cheng K. Chen (California Microprocessor Division Advanced Micro Devices), Trevor Mudge (EECS Department University of Michigan).

[19] *"The YAGS Branch Prediction Scheme":* A. N. Eden and T. Mudge, (Dept. EECS, University of Michigan, Ann Arbor).

[20] *"Branch Transition Rate: A New Metric for Improved Branch Classification Analysis":* Michael Haungs, Phil Sallee, and Matthew Farrens (Department of Computer Science, University of California, Davis).

[21] "The YAGS Branch Prediction Écheme" (A. N. Eden and T. Mudge, {ane, tnm}@eecs.umich.edu; Dept. EECS, University of Michigan, Ann Arbor).

[22] Daniel A. Jiménez, Calvin Lin *Neural Methods for Dynamic Branch. Prediction.* ACM Transactions on Computer Systems, Vol.20, No4.

[23] Pedro A. Calero, *Aprendizaje Automático*. Apuntes del seminario impartido en la facultad de Informática de la UCM, curso 2003-2004.

[24] Rosenblatt, F. 1962. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.* Spartan, New York.