Plataforma de Comparación para la Predicción Dinámica de Saltos en Hardware FPGA

Amilcar Arfel Molina Díaz

Resumen— **Los cambios en el flujo de ejecución de un programa (dependencias de control) suponen una degradación importante en el rendimiento debido a que el procesador al encontrar un salto, ha de bloquear la búsqueda de instrucciones hasta que no sea ejecutado en su unidad funcional.**
Una técnica utilizada para evitar estos bloqueos consiste en predecir el comportamiento de cada instrucción de salto (si será efectivo y su dirección destino) antes que sea resuelto. De esta manera el procesador no se bloquea y puede hacer la búsqueda de instrucciones en cada ciclo. La predicción de saltos se puede realizar en tiempo de compilación (estática) o en tiempo de ejecución (dinámica).
Este trabajo consiste en establecer una plataforma para la comparación de estrategias de predicción dinámica de saltos en hardware, específicamente empleando los dispositivos FPGA. Las técnicas de predicción de saltos a comparar son la Bimodal, GAg, PAg, gshare, McFarling, perceptron y perceptron rápido. Las mismas se integraron al simulador SimpleScalar, extendiendo el código del simulador simsafe, lo que permite la validación o confirmación de los resultados vía software. Se realiza una prueba con algunos benchmark del SPEC2000 compilados para una máquina PISA. Obteniéndose los resultados esperados. La idea es comparar estas estrategias a posteriori con métricas tales como consumo de energía, consumo de área y latencia que se generarán de la correspondiente síntesis y prueba directamente sobre el dispositivo Hardware reconfigurable FPGA.

Palabras clave— **Predictores Dinámicos de Salto, perceptron, FPGA, reconfiguración parcial dinámica.**

I. Introducción

LOS procesadores, los cuales son capaces de leer y ejecutar más de una instrucción por ciclo, en las instrucciones de salto suponen un inconveniente para la obtención de un rendimiento alto. Estas instrucciones provocan dependencias de control. Cuando una instrucción de salto es decodificada, el decodificador bloquea el fetch de instrucciones hasta que el salto no es resuelto, es decir, hasta que no se conoce la dirección destino, y no se sabe si el salto es tomado (si salta a la dirección destino) o no. Como el procesador ha estado varios ciclos sin leer ninguna instrucción de la cache de instrucciones (ICache), nos encontramos con que varios stages del pipeline están vacíos y que la ventana de instrucciones (donde están las instrucciones pendientes de ser ejecutadas) tiene pocas instrucciones. A este hecho se le llama introducir una burbuja en el pipeline. Esta burbuja provoca una bajada en el rendimiento del procesador ya que éste tendrá menos instrucciones donde elegir para lanzar a ejecutar.

Universitat Politecnica de Catalunya, Departament d'Arquitectura de Computadors, Carrer Jordi Girona 1-3, Modul D6 116-10, (Campus Nord) 08034 Barcelona (Spain). e-mail: amolina@ac.upc.edu.

Para evitar la penalización introducida por los cambios en el flujo de ejecución de un programa una solución adoptada por muchos procesadores consiste en tener métodos precisos que predigan la dirección de los saltos condicionales [1], así como anticipar lo antes posible el cálculo de la dirección destino.

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

Las técnicas de predicción de salto se pueden dividir, básicamente, en dos tipos: Las que realizan predicción estática y las que realizan predicción dinámica.

La diferencia radica en el momento en el que se realiza la predicción, en el caso de la predicción estática se realiza en tiempo de compilación, mientras que la dinámica se realiza en tiempo de ejecución. Ambas tienen sus ventajas y sus inconvenientes.

Por ejemplo con la predicción dinámica se consigue mayor precisión a cambio de un coste en hardware mayor, mientras que la estática tiene menos precisión, no requiere tanto hardware, pero a cambio en algunas implementaciones puede llegar a incrementar la longitud del ejecutable considerablemente (hasta un 30% [2]).

Existen también en la literatura una serie de técnicas, no basadas en la predicción, para reducir el coste de los saltos [3] [4]. Estas son por ejemplo: salto retardado, avance en el cálculo de las instrucciones de las que el salto depende, o la ejecución concurrente de los dos posibles caminos que se pueden coger al llegar a un salto condicional.

El objetivo de este trabajo es el de establecer una plataforma de comparación en Hardware empleando

los dispositivos FPGA, se pretenden comparar las siguientes técnicas de predicción dinámica de saltos: bimodal [5], GAg, PAg, gshare, McFarling [6] [7], perceptron [8] y perceptron rápido [9].

La organización del trabajo es la siguiente: En el apartado II se presentan las técnicas de predicción dinámica de saltos que componen la herramienta de comparación. En el apartado III se investiga el funcionamiento de los predictores de salto de los procesadores MIPS R8000, MIPS R10000, HP-PA8000, ALPHA 21164, ALPHA 21264, AMD-K5, AMD-K6, AMD-ATHLON, INTEL (Pentium MMX, Pentium Pro, Pentium II), INTEL Pentium III, INTEL Pentium 4, SUN UltraSparc II, SUN UltraSparc II y SUN UltraSparc III. En el apartado IV se implementan de manera exprimental las estrategias de predicción de salto, aquí se define el entorno de simulación y los benchmarks a utilizar. Luego en el apartado V se presentan los resultados de la fase experimental y por último en el apartado VI se hace una exposición acerca del trabajo futuro que conduce la investigación realizada.

II. Estrategias de Predicción Dinámica de Saltos

En estas estrategias, la predicción sobre el resultado de un salto se basa en información conocida sólo en tiempo de ejecución (al contrario de la predicción estática). Dos estructuras son necesarias para realizar una predicción dinámica:

- *Branch History Table (BHT):* Es una tabla donde se guarda información sobre las últimas ejecuciones de los saltos, esta información se refiere a si el salto ha sido efectivo o no. A partir de esta información se predice si el siguiente salto será tomado o no.

- *Branch Target Address Cache (BTAC):* Es una tabla donde se almacena la dirección destino de los últimos saltos ejecutados. De esta manera cuando un salto es predicho como tomado, se mira si está en la tabla, y si es así, se obtiene la dirección destino de ella, así se puede calcular rápidamente la dirección destino, incluso en saltos indirectos.

La base de la mayoría de los predictores dinámicos es el llamado `Branch Target Buffer` [10] [11]. Esta estructura combina las dos mencionadas anteriormente (BTAC y BHT). Cada entrada contiene los bits necesarios para realizar la predicción de efectividad, así como la posible dirección destino.

A partir de esta estructura básica se han ido diseñando los distintos predictores, desde el más sencillo que para cada salto guarda unos bits de información sobre su historia más reciente, hasta los más complicados. Existen predictores híbridos que combinan distintas técnicas (estáticas y dinámicas), escogiendo en cada momento la salida del que se cree que es el mejor (el que lleva más aciertos hasta entonces).

Un aspecto a considerar es la influencia de los fallos de predicción en el rendimiento de los procesadores. Debido a ellos pueden aparecer dos tipos de penalización (misfetch penalty y/o mispredict penalty). Los procesadores en cada ciclo hacen el fetch de las instrucciones de la ICache para mantener el rendimiento. Cuando aparece una instrucción de salto y la predicción de salto tomado se hace en la etapa de decodificación, las instrucciones leídas de la ICache en ese ciclo y en el anterior no son válidas y han de ser descartadas. Esto provoca que se introduzca una burbuja en el procesador conduciendo esto a una `misfetch penalty`. El procesador lee de la ICache las instrucciones correspondientes al bloque destino del salto en el siguiente ciclo de realizar la predicción de una instrucción de salto. Cuando la instrucción de salto se ejecuta en su unidad funcional se comprueba si la predicción realizada anteriormente ha sido correcta. Si ha habido un fallo en la predicción, el procesador debe realizar un vaciado del pipeline, quitando todas las instrucciones del camino incorrecto pendientes de ser ejecutadas y restaurando el estado de los registros como estaba antes de predecir el salto, esto trae consigo una `mispredict penalty` y a su vez se deberá realizar el fetch de las instrucciones del camino correcto. Esto supone una degradación importante en el rendimiento del procesador, ya que al vaciarse el pipeline se introduce una nueva burbuja.

Al realizar la comparación entre distintas arquitecturas de predicción de saltos es necesario contar con métricas, tales como consumo de potencia, consumo de área y latencia asociadas al hardware y una métrica que indique el rendimiento que puede ser de manera sencilla la precisión de aciertos, es decir, el porcentaje de saltos que han sido bien predichos. Otras, más complicadas se basan en el misfetch penalty. El problema de rendimiento de la predicción de los saltos puede dividirse en dos subproblemas, en primer lugar se trata de encontrar si el salto es tomado o no y en segundo lugar para los saltos que se toman se debe asegurar que las instrucciones del salto objetivo esten disponibles para que se ejecuten con un retardo mínimo. Una manera de ofrecer las instrucciones del salto objetivo rápidamente es empleando un Branch Target Buffer (BTB) el cual es una pequeña memoria asociativa que guarda las direcciones de los últimos saltos ejecutados así como su destino. A su vez guarda información que permite predecir si el salto será tomado o no [10]. Este trabajo se enfoca en la predicción de la dirección de los saltos. Las posibilidades para ofrecer las instrucciones del salto no se discutirán.

TABLA I

NOMENCLATURA Y ESTIMACIÓN DEL COSTO EN HARDWARE DEL ESQUEMA $GAg(k)$ Y EL $PAg(k)$.

Nombre del Esquema	**Longitud BHR**	**Número de PHT**	**Costo Hardware (Bits)**
$GAg(k)$	k	1	$k + 2^k.2$
$PAg(k)$	k	1	$b \cdot k + b.2^k.2$

En un esquema de predicción de saltos dinámico, la predicción se hace sobre la historia de cálculos del programa, los predictores aquí presentados se basan en dos niveles de historia los cuales mejoran los resultados que ofrece el BTB. Estos predictores utilizan mayor cantidad de recursos dentro del procesador, pero ofrecen unos porcentajes de aciertos más altos.

Diversos autores [12] [13] han propuesto predictores de dos niveles de historia. En un primer nivel recogen la historia de los n últimos saltos ejecutados o de las n últimas ejecuciones de un salto concreto (dependiendo de la implementación). Con esta información, en un segundo nivel se indexa una tabla de patrones que en cada entrada guarda una máquina de estados (normalmente codificada en dos bits) con la que se decide si el salto es tomado o no.

Yeh y Patt [12] presentan una primera aproximación que consiste en dos estructuras principales: Branch History Register Table (BHRT) y Branch History Pattern Table (BHPT) (Figura 1).

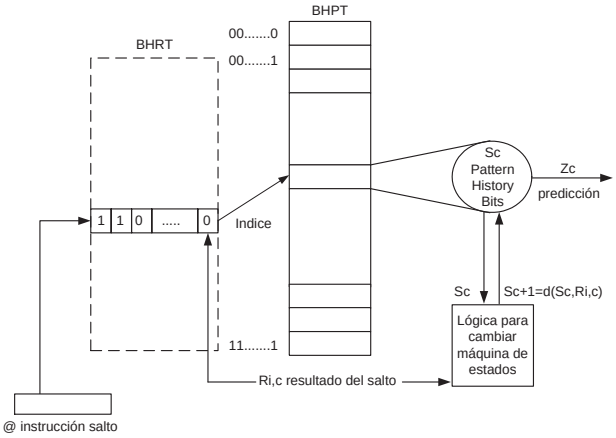

Fig. 1. Estructura del predictor de saltos de dos niveles.

La BHRT es una tabla que contiene un registro de m bits para cada salto. Este registro es de desplazamiento y almacena la historia de las últimas m ejecuciones de dicho salto.

La BPHT es una tabla de 2^m entradas, indexada por el registro de historia. Esta tabla guarda la historia de las últimas ejecuciones de todos los saltos que tienen un mismo patrón, representado por el registro de historia.

Cuando llega un nuevo salto B_i, se obtiene su registro de historia de la BHRT, el contenido del cual es utilizadopara indexar la Pattern Table obteniendo S_c, que es la historia de la ejecución siguiente del salto cuando tenia el patrón dado por el registro de historia. Sobre estos bits es aplicada la función de predicción Z_c, la cual nos dice si el salto va a ser tomado o no.

Después que el salto es ejecutado, se actualiza el registro de historia del mismo colocando el resultado $R_{i,c}$ en el bit de menos peso del registro, desplazándose éste una posición hacia la izquierda, a su vez la entrada correspondiente en la Pattern Table es actualizada (S_{c+1}) mediante una función (d) que tiene como entradas la ejecución del salto ($R_{i,c}$) y el estado anterior (S_c). Dicha función puede ser un simple contador up-down de 2 bits que se incrementa cada vez que el salto es tomado y se decrementa cuando el salto no es tomado. Para realizar la predicción se puede coger el bit de más peso que indicará si el salto es efectivo o no.

Dos aspectos importantes a la hora de implementarse este método es el grado de asociatividad de la tabla de registros de historia así como la longitud en bits de cada uno de los registros.

Yeh y Patt [14] [15] y [16] recopilarón los distintos métodos de predicción a dos niveles y crearon una nomenclatura con la que se puede distinguir las diferentes implementaciones existentes de los predictores que usan dos niveles de predicción de salto. Esta nomenclatura se muestra en la Tabla I (página 3).

Las primeras dos letras describen la estructura de los registros, es decir un registro de historia global (GA) o un registro de historia por cada salto (PA). La última letra describe la estructura de la tabla de patrones, que puede ser: una tabla de patrones global (g), una tabla de patrones para cada salto (p) o una tabla de patrones para cada conjunto de saltos (s).

El coste de cada una de estas estructuras en bits viene descrito en la Tabla I de una manera sencilla. Para cada implementación, k representa el número de bits que tienen los registros de historia y b representa el número de entradas en la tabla de registros de historia.

A. Predictor de Saltos Bimodal

El predictor de saltos bimodal toma ventaja de la distribución bimodal que presenta el comportamiento de los saltos. Hay distintas maneras de realizar esto, en la Figura 2 (página 4) se muestra una manera simple de realizar una estructura de predicción de saltos bimodal. Se observa una tabla (BHT) de contadores indexados por los bits menos significativos de la dirección de la instrucción de salto (@ instrucción salto). Cada contador es de 2 bits de longitud. Para cada salto tomado, el correspondiente contador es incrementado, del mismo modo para cada salto no tomado, el correspondiente contador es decrementado. Este contador es saturado, es decir, cuando llega a cero no se decrementa, y no se incrementa después de tres. El bit más significativo determina la predicción. Los saltos tomados de manera repetida se predecirán a ser tomados y los saltos no tomados de manera repetida se predecirán a ser no tomados.

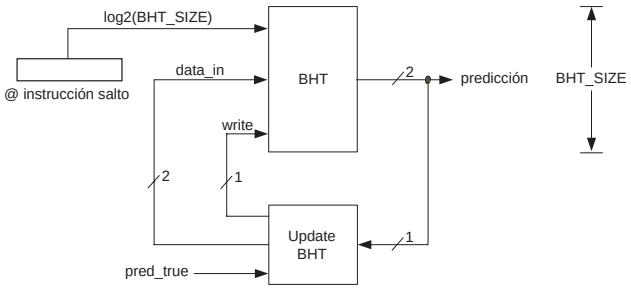

Fig. 2. Estructura del predictor de saltos bimodal.

Mediante el uso de un contador de 2 bits, el predictor puede tolerar que un salto vaya a una dirección no usual una vez y mantener la predicción de la dirección usual del salto.

Para tablas de contadores grandes, cada salto corresponderá a un único contador. Para tablas pequeñas, múltiples saltos pueden compartir el mismo contador, resultando en una degradación de la precisión de la predicción.

B. Predictor de Saltos GAg (Global)

El predictor de saltos GAg toma ventaja de otros saltos recientes para hacer una predicción. En la Figura 3 se muestra una implementación del mismo.

Un simple registro de desplazamiento GBHR, registra la dirección tomada por los n saltos condicionales más recientes. Debido a que la historia de los saltos es global para todos los saltos, esta estrategia es denominada Predictor de Saltos Global.

El predictor de saltos global es capaz de tomar ventaja de dos tipos de patrones. Primero la dirección tomada por el salto actual puede depender fuertemente de otros saltos recientes. La segunda manera en que la predicción de saltos global puede

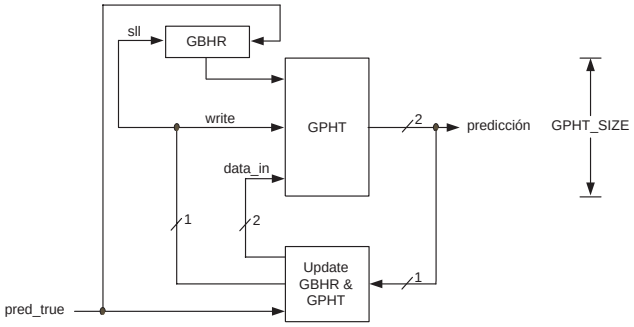

Fig. 3. Estructura del predictor de saltos GAg.

ser efectiva es mediante la duplicación del comportamiento de la predicción de los saltos locales. Esto puede ocurrir cuando la historia global incluya toda la historia local necesaria para hacer una predicción exacta.

C. Predictor de Saltos PAg (Local)

Una manera para mejorar la predicción bimodal es reconociendo que muchos saltos ejecutan patrones repetitivos. Lo más común como ejemplo de este comportamiento es los saltos de los lazos de control. La Figura 4 (página 4) muestra un predictor de saltos que usa dos tablas. En esta consideración se tomo en cuenta que son simples arreglos indexados por los bits menos significativos de las direcciones de los saltos. Yeh and Patt asumio una tabla histórica de saltos conjunto-asociativa.

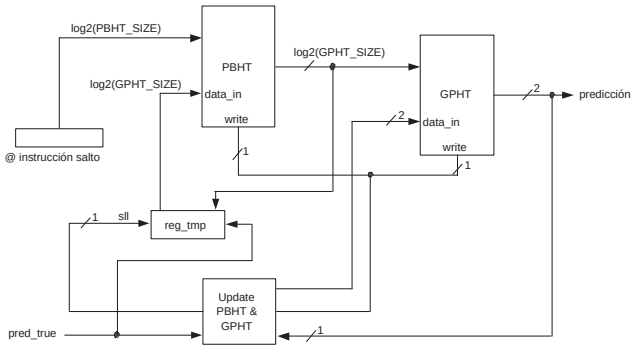

Fig. 4. Estructura del predictor de saltos PAg.

Cada entrada de la tabla histórica registra la dirección tomada por los n más recientes saltos cuyas direcciones corresponden a esta entrada, donde n es la longitud de la entrada en bits. La segunda tabla es un arreglo de contadores de 2 bits idénticos a los empleados en el predictor de saltos bimodal. Por tanto, ellos aquí son indexados por la historia de saltos almacenada en la primera tabla. Se le llama local porque la historia utilizada es local al salto actual. Según la nomenclatura de Yeh and Patt este método es un esquema per-address.

D. Predictor de Saltos gshare

Este predictor consiste en realizar el OR exclusivo de la dirección del salto con la historia global para

tener más información que con tal sólo un componente. La Figura 5 (página 5) muestra la estructura del predictor gshare.

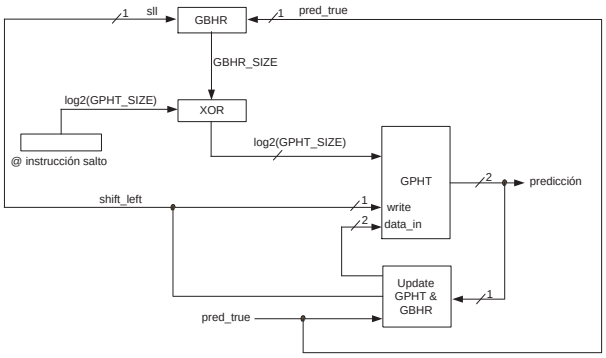

Fig. 5. Estructura del predictor de saltos gshare.

E. Predictor de Saltos McFarling

Los diferentes predictores vistos hasta ahora tienen sus ventajas y sus inconvenientes, lo cual hace pensar que si se pudiera combinar su uso podríamos sacar partido de sus características y conseguir mayor precisión en la predicción. Esta es la filosofía de los predictores híbridos, que fueron propuestos por primera vez por McFarling [7].

Su esquema consiste en un conjunto de predictores y un mecanismo de selección entre ellos. Para cada salto todos los predictores realizan su predicción y el selector elige el que hasta entonces haya tenido mejores resultados. Su propuesta es utilizar dos predictores y una tabla de contadores saturados up-down de dos bits, que sería indexada por la dirección de la instrucción de salto (Figura 6) (página 5). Cada contador sería el encargado de seleccionar el predictor adecuado para cada salto, basándose en los aciertos o fallos que han tenido los predictores hasta entonces.

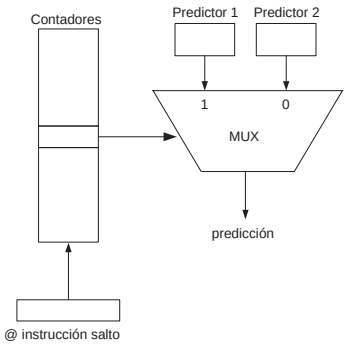

Fig. 6. Esquema de Predictor Híbrido propuesto por McFarling.

Si en la última predicción de salto P1 y P2 fallarón el contador no se cambia, si en cambio P1 falló y P2 acertó el contador es decrementado. Si P1 acertó y P2 falló el contador se incrementa, y por último si los dos aciertan el contador no se toca. Con el bit de más peso del contador se elige P1 (si vale 1) o P2 (si vale 0), ya que si el contador tiene un número alto significa que P1 ha acertado más veces que P2, y si tiene un número bajo significa que P2 ha acertado más veces que P1.

El siguiente paso sería escoger qué predictores se utilizarán. McFarling concluye su trabajo diciendo que el mejor sistema sería utilizar un predictor que use historia local (por ejemplo un BTB con un contador de 2 bits para realizar la predicción), junto con otro basado en la historia global, como el gshare. Con este sistema llega a conseguir hasta un 98.1% de aciertos en la predicción realizando las simulaciones sobre los 10 primeros millones de instrucciones del conjunto de programas SPEC89.

Chang et al [17] hacen un estudio más estricto tanto de los predictores a utilizar como del método de selección de los mismos. Los predictores que utiliza en su trabajo son:

- Un predictor estático basado en profiling.
- Un predictor de un nivel basado en una lista de contadores de dos bits.
- El esquema introducido por Yeh y Patt PAs(m,n) que tiene 1K de registros de historia de m bits y n tablas de patrones.
- El predictor creado por McFarling basado en la historia global (gshare).

Haciendo combinaciones de estos cuatro predictores llega a la conclusión que la mejor predicción se consigue utilizando los predictores gshare y PAs.

Respecto al mecanismo de selección de los predictores, proponen una técnica llamada algoritmo de selección del predictor del salto de dos niveles (parecido al creado por Yeh y Patt para la predicción de saltos).

En la implementación del predictor McFarling se selecciono el predictor que tiene el procesador 21264 que para el año 2001 era el más sofisticado [6]. El mismo presenta la siguiente estructura: El primer predictor de saltos (P1) es un predictor global (GAg) el cual tiene un GBHR y un GPHT con contadores saturados de 2 bits indexados por el GBHR (Ver Figura 3). El segundo predictor de saltos (P2) es un predictor local (PAg) el cual tiene un PBHT y un GPHT con contadores saturados de 2 bits indexados por el PBHT (Ver Figura 4). El PBHT es indexado por los bits menos significativos de la dirección de la instrucción de salto correspondiente.

La combinación de los predictores de salto anteri-

ores, emplean un arreglo de contadores saturados de 2 bits adicional el cual sirve para seleccionar el mejor predictor a utilizar. Cada contador mantiene la pista de cual predictor es más preciso para los saltos que comparte el contador. Es la implementación de un predictor bimodal (Ver Figura 2, página 4) el cual tiene un BHT de contadores de dos bits saturados, indexados por los bits menos significativos de la dirección de la instrucción de salto.

F. Predictor de Saltos Perceptron

Los predictores de saltos con aprendizaje neuronal son los predictores más precisos que existen en la literatura, pero son imprácticos debido a su alta latencia. Esta latencia se debe principalmente a los complejos cálculos que deben ejecutarse para determinar la excitación de una neurona artificial.

El predictor perceptron emplea aprendizaje perceptron para predecir las direcciones de los saltos condicionales [8] [18]. Este es un predictor de correlación que hace una predicción del salto actual sobre la base de un patrón histórico observado de los saltos previos.

Un predictor perceptron utiliza una matriz de pesos enteros W de tamaño $N * (h + 1)$, donde N y h son parámetros de diseño. Los pesos son típicamente bytes de 8 bits. Cada fila de la matriz es un vector de pesos de longitud $(h + 1)$ (Figura 7). Cada vector de pesos almacena los pesos de un perceptron que es controlado mediante aprendizaje perceptron.

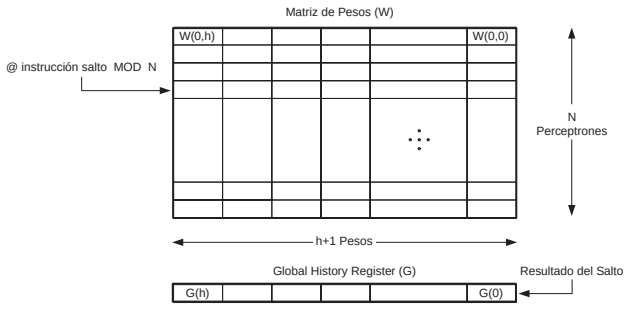

Fig. 7. Estructura de Datos del Predictor de Saltos Perceptron.

En el vector de pesos $w[0..h]$, el primer peso $w(0)$, se conoce como el peso de sesgo. Por lo tanto, la primera columna de W contiene los pesos de sesgo de cada vector de pesos. El vector booleano $G[1..h]$ representa el registro de desplazamiento de la historia global que contiene los valores (tomado, no_tomado), almacenando el resultado de los h saltos previos.

Para predecir un salto con una dirección de memoria M se hace la operación $(M \, mod \, N)$ la cual indexa una fila particular en la matriz W. Esta fila es un perceptron artificial que es responsable de predecir múltiples instrucciones de salto. La predicción se realiza basándose en los valores de los pesos en esa fila y sobre el resultado de los "h" más recientes saltos, lo cual es almacenado en el registro de historia global G.

Una vez que el resultado del salto se conoce se entrena el perceptron que fue responsable de la predicción de una manera especifica dependiendo si la predicción fue correcta y también basándose en el patrón de historia almacenado, que es almacenado en el registro de historia global G. Esto se logra actualizando los pesos en la fila correspondiente de la matriz W.

El algoritmo de entrenamiento toma un parámetro entero θ. Empíricamente se ha determinado que para obtener la mejor precisión se debe seleccionar $\theta = [1.93 * h + 14]$, donde h es la longitud de la historia. Por lo tanto para una longitud de historia dada, θ es constante.

El perceptron es una red neuronal muy simple. Cada perceptron es un conjunto de pesos los cuales se entrenan para reconocer patrones o correlaciones entre sus entradas y los eventos a ser predichos. Una predicción se realiza mediante el cálculo del producto escalar de los pesos y un vector de entrada (Figura 8). El signo del producto escalar es usado para realizar la predicción. En el contexto de los predictores de salto, cada peso representa la correlación de un bit de historia (global, de trayectoria o local) con el salto a predecir.

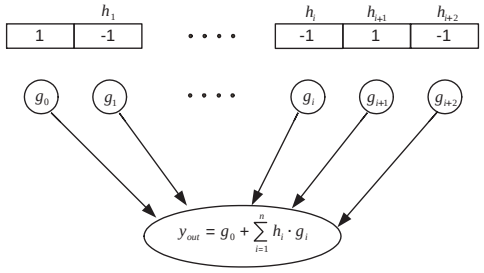

Fig. 8. Cálculo del producto escalar y_{out}.

En hardware cada peso es implementado como un entero con signo de n bits, donde típicamente en la literatura n es 8, almacenado en un arreglo SRAM. El vector de entrada consiste de $+1$ para los saltos tomados y -1 para los saltos no tomados. El producto escalar puede ser calculado como una suma sin necesidad de circuitos para multiplicar.

El perceptron asigna pesos para cada elemento de la historia del salto y hace su predicción basada sobre el producto escalar de los pesos y la historia de los saltos más un sesgo que representa la tendencia completa del salto. Hay que observar que la historia de los saltos puede ser global, local o en algunas ocasiones muy compleja.

La idea de este perceptron fue introducida originalmente por Vintan [19] y Jiménez mostró en [8] que el perceptron global puede ser más preciso que cualquier otro conocido. El perceptron original de Jiménez empleaba un sumador Wallace tree para calcular la salida del perceptron, pero todavía incurre en más de 4 ciclos de latencia.

G. Predictor de Saltos Perceptron Rápido

El predictor perceptron rápido es un predictor neuronal que selecciona el vector de pesos de acuerdo a la trayectoria que conduce a los saltos, en vez de seguir solamente la dirección de los saltos. Esta técnica tiene dos ventajas. Primero, la latencia es mitigada debido a que el cálculo de y_{out} puede anticiparse a la predicción, con cada paso adelantado tan pronto como un nuevo elemento del camino se ejecute. Segundo, se mejora la precisión debido a que el predictor incorpora la información de la trayectoria en la predicción.

El perceptron rápido tiene la misma estructura que el perceptron (Figura 7, página 6). Este mantiene una matriz de vectores de pesos W. Cada vez que un salto es fetch y requiere una predicción, uno de los vectores de pesos de W es leído. Sin embargo, sólo el peso del sesgo, es empleado para ayudar en la predicción del salto actual. Este valor es añadido a un total que se ha generado de los últimos h saltos, con cada sumando agregado durante el procesamiento de un salto previo.

Este perceptron mejorado, recientemente introducido y basado en la trayectoria [9], oculta la mayoría del retardo mediante el fetch de los pesos y el cálculo de la suma en la corrida a lo largo de la trayectoria seguida por cada salto. El retardo crítico de este predictor es por tanto la suma del retardo de un pequeño arreglo SRAM, un multiplexor y un pequeño sumador. Se estima que una predicción puede estar disponible en el segundo ciclo después de que se disponga de la dirección. Por comodidad y simplicidad en este trabajo se denomina a esta organización el `perceptron rápido`.

Seznec propuso varías mejoras importantes al perceptron global original en [20] [21]. En [21] el introdujo el perceptron MAC-RHSP (multiply-add contribution redundant history skewed perceptron). El redujo el número de sumadores en un factor de cuatro (16 cuando usa historia redundante) sobre el predictor perceptron global normal, almacenando todas las 16 combinaciones posibles de cuatro pesos en entradas de la tabla separadas y seleccionadas con un multiplexor 16 a 1 después que ellas se busquen de las tablas de pesos. En nuestra terminología, el MAC-RHSP es similar al predictor perceptron global que usa una concatenación de direcciones e información histórica (GAs o gselect) para el fetch de sus pesos. Sin embargo el MAC-RHSP busca todos los pesos los cuales comparten los mismos bits de dirección de las tablas y luego emplea un multiplexor 16 a 1 para seleccionar entre ellos.

III. Funcionamiento de los predictores de salto en algunos procesadores comerciales

En esta sección, se presenta el funcionamiento de los predictores de salto implementados en algunos procesadores comerciales MIPS, HP, ALPHA, AMD, INTEL y SUN.

A. MIPS R8000

El MIPS R8000 implementa un sistema de predicción de salto similar al NLS de Calder y Grunwald [22]. Dispone de una cache de predicción de saltos (BCache) [23], que es accedida en paralelo junto con la ICache en la etapa de fetch.

La BCache tiene 1K de entradas y es de mapeo directo. La salida de esta cache contiene un bit de predicción, la línea de la cache de la instrucción destino y la posición que ocupa la instrucción dentro de la línea.

La arquitectura MIPS introduce un delay slot después de cada instrucción de salto, por lo tanto solo puede haber dos saltos en una línea de cache. Teniendo en cuenta que el compilador puede hacer optimizaciones para separar los saltos entre sí, las posibilidades de tener dos saltos en una línea de cache es muy baja.

Cuando el salto es resuelto se compara la dirección destino calculada con la de la instrucción que sigue al delay slot (la instrucción destino predicha). Si ha habido un fallo de predicción el procesador tarda tres ciclos en solucionarlo: vacíado de las etapas de fetch, decodificación y cálculo de direcciones, a la vez que se hace el fetch del camino correcto.

Una característica importante de este esquema es que su coste hardware no es muy alto ya que en la BCache se guardan sólo 10 bits que indican la línea destino, no toda la dirección virtual. Además para incrementar la eficiencia se realiza la predicción en la etapa fetch.

B. MIPS R10000

El procesador MIPS R10000 implementa una tabla de 512 contadores de 2 bits. Esta tabla está mapeada directamente por los bits 11:3 de la dirección de la instrucción de salto y representan la historia reciente del salto. Utilizando contadores saturados up-down se consigue una precisión del 87% de aciertos [24].

Cuando el procesador, en la etapa de decodificación, predice un salto como tomado, descarta todas las instrucciones que haya leído en la etapa de fetch que vengan después del salto. El R10000

carga la dirección instrucción destino en el `Program Counter` y hace el fetch de las nuevas instrucciones después de un ciclo de retardo. Esto introduce una burbuja en el pipeline del procesador, durante la cual no decodifica ninguna instrucción.

Un aspecto importante de este procesador es la manera en la guarda el estado cuando hace la predicción de un salto. El R10000 guarda en una estructura interna la dirección alternativa del salto, copias de los bancos de registros lógicos y físicos, así como una serie de bits de control. El objetivo de esta estructura es que en el caso de un fallo de predicción se pueda deshacer todo lo ejecutado en un ciclo.

C. HP-PA8000

El procesador PA-8000 permite el uso de predicción estática o dinámica [25]. Con un bit extra de control en el TLB se selecciona el tipo de predicción que se utilizará.

En el modo de predicción estática, la unidad de fetch sigue la siguiente política: Para las instrucciones de comparación y salto, en el código de operación hay un campo que indica a la unidad de fetch si ha de saltar o no.

En cuanto a la predicción dinámica, el procesador PA-8000 utiliza dos estructuras:

- *Branch History Table (BHT):* Tabla de 256 entradas, que tiene en cada una de ellas un registro de desplazamiento de 3 bits que indica la historia de las tres últimas ejecuciones de ese salto. Si la mayoría de las veces ha sido tomado, se predirá el salto como tomado. La tabla es actualizada cuando los saltos son retirados del pipeline para evitar la polución de saltos ejecutados especulativamente.

- *Branch Target Address Cache (BTAC):* Es una tabla completamente asociativa que guarda la dirección destino de los 32 últimos saltos ejecutados. Se introduce un salto cada vez que es tomado y se aplica una politica de round-robin para reemplazar saltos.

Cuando en la etapa de fetch una instrucción esta en la BHT y es predicha como tomada, se busca si su dirección destino está en la BTAC, y si es así en el siguiente ciclo se podrá hacer el fetch del camino predicho.

D. ALPHA 21164

El procesador 21164 implementa una estructura de predicción de saltos dinámica [26]. Para ello mantiene una estructura de 2048 contadores up/down saturados, indexados con los bits de menos peso del contador de programa. Los contadores se incrementan cuando el salto es tomado y se decrementa cuando el salto no es tomado. Un salto es predicho como tomado si el bit de más peso esta en 1.

Para el cálculo de las direcciones destino dispone de 4 sumadores (al realizar el fecth de 4 instrucciones cualquiera de ellas puede ser un salto) con desplazamiento que producen el destino teórico del salto en paralelo con el cálculo de la predicción.

Durante el segundo ciclo del pipeline, el procesador examina el bloque de instrucciones que le llega de la ICache o del refill buffer. Chequea las instrucciones de salto, su predicción y calcula la dirección destino para que se pueda leer en el siguiente ciclo, creando por lo tanto un ciclo de retardo (burbuja) en el pipeline.

E. ALPHA 21264

El procesador 21264 emplea tres algoritmos de predicción. Uno es llamado el predictor local; este se enfoca sobre los saltos locales tales como los lazos. Otro es denominado el predictor global, el cual mantiene un registro global de todos los saltos recientes. El tercer predictor decide entre el mejor de los dos predictores o el local o el global. Este predictor de decisión es un predictor bimodal que presenta una BHT de contadores saturados de 3 bits y para el desarrollo de la plataforma de comparación se tomara el mismo como predictor McFarling con tan sólo el cambio de que la BHT en vez de tener contadores saturados de 3 bits tendra contadores saturados de 2 bits.

El procesador 21264 de acuerdo a trazas digitales, falla en sólo 7 saltos por 1000 instrucciones, la mitad de lo que ofrece el procesador 21164, y aporta una tasa de precisión total de alrededor de 95% [27].

F. AMD-K5

El procesador AMD-K5 implementa un sistema de predicción insertado en la ICache [28], similar al explicado por Johnson en su trabajo [29].

En cada línea de la ICache hay información sobre si el salto es tomado o no (si hay más de un salto en la línea éstos comparten información), la siguiente línea a leer y la instrucción dentro de la línea a leer.

La predicción puede fallar por dos motivos: La dirección predicha sea diferente de la después calculada, o que el bloque destino no esté en la cache.

Con esta implementación el hardware gastado es el 25% mayor del gastado por un BTB de 256 entradas con una asociatividad de grado 4.

G. AMD-K6

El procesador AMD-K6 implementa un predictor de saltos de dos niveles adaptativo. Múltiples predicciones pueden emplearse. La Tabla II (página 9) muestra una comparación entre el predictor de saltos del Pentium MMX y el AMD-K6 [30].

TABLA II

Comparación entre los predictores de saltos del Pentium MMX y el AMD-K6.

	Pentium II	**AMD-K6**
Predicción de Saltos de 2 Niveles Avanzada	Si	Si
Predicción de Saltos Múltiples	Si	Puede ser Si
Tabla de Historia de Saltos	?	8192
Entradas al Branch target buffer	0 (on the fly calculation)	512
Precisión de la Predicción del Salto	90%	95%

H. AMD-Athlon

El procesador AMD-Athlon es un diseño más nuevo que el Pentium III, y utiliza un esquema de predicción de saltos de dos niveles global. Este contiene una historia global de 8 bits (un registro de desplazamiento contiene los resultados tomados/no tomados de los últimos ocho saltos ejecutados) que es usada junto con cuatro bits de la dirección de instrucción para seleccionar un contador saturado de predicción de dos bits de una tabla que contiene 2048 entradas [31] [32]. La penalización mínima de los saltos es 10 ciclos [33]. Hay algunas limitaciones sobre la localidad y densidad de los saltos: cada bloque de 16 bytes en la cache de instrucciones L1 sólo puede mantener dos saltos condicionales que son predichos correctamente. Si más saltos están presentes, habra un fallo de predicción [32].

I. La Familia Pentium (Pentium MMX, Pentium Pro, Pentium II)

En esta familia, esta implementada la predicción de saltos adaptativa de dos niveles.

El primer nivel es un registro de desplazamiento que almacena la historia de los últimos 14 saltos ocurridos. El segundo nivel es un contador de 2 bits con saturación el cual almacena el patron de los últimos 4 saltos [34].

J. INTEL Pentium III

El Pentium III emplea un Predictor dinámico de saltos, puesto que necesita mantener lleno un pipeline muy profundo. El predictor es uno que presenta el estilo *Pap* [15] (Ver Tabla I, página 3).

La tabla del predictor de saltos contiene 512 entradas, organizadas como una cache conjunto asociativa de 16 vías, con la dirección del salto utilizada para indexar. Para cada entrada en la tabla, hay un registro de desplazamiento de cuatro bits que sigue la pista de las últimas cuatro instancias de predicción de este salto (esto es una historia local). Estos valores de 4 bits son empleados para seleccionar un contador saturado de 2 bits [31] [35]. La penalización por una falla de predicción de un salto esta entre 10 y 26 ciclos sobre un Pentium III [36].

K. INTEL Pentium 4

El Pentium 4 posee la más compleja arquitectura de los procesadores descritos en esta investigación. El mecanismo de predicción de saltos empleado no esta documentado, pero algunas características acerca de la estructura general ha sido liberada por Intel [37].

Construido para la máxima frecuencia de reloj, el Pentium 4 emplea una *cache de traza* en lugar de una tradicional cache de instrucción. La cache de traza integra la predicción de salto en la cache de instrucción mediante el almacenamiento de trazas de instrucciones que han sido previamente ejecutadas en secuencia, incluyendo los saltos. La misma linea de cache puede incluir ambos un salto y su objetivo, con una penalización cero para ejecutar el salto a lo largo de la dirección predicha[1]. Hay un "pequeño" predictor de salto en el procesador que es utilizado para el fetching de operaciones desde la cache de traza.

La cache de traza es alimentada por un anticipador que busca instrucciones de el nivel 2 de cache. Aquí, un segundo predictor de salto con una tabla de predicción de salto de tamaño 4096 entradas es empleado para predecir donde buscar las siguientes instrucciones de la cache L2. Este predictor de salto usa alguna forma de técnica de dos niveles que depende de la historia de los saltos, y se suponen que tienen más ventajas que las técnicas empleadas sobre los procesadores Pentium III y Athlon.

El *"fallo de pipeline"* se documenta en un comienzo como dos veces tan largo como para el Pentium III, lo cual pudiera indicar que la minima penalización por un fallo de predicción de salto es al menos 20 ciclos. Las penalizaciones para los saltos mal predichos es de al menos 20 ciclos.

L. Sun UltraSparc I

El procesador UltraSparc I tiene una estructura de predicción de saltos incorporada a la cache

[1]Con una cache regular, incluso un salto con predicción exitosa sufrirá una pena pequeña de tener que volver a dirigir el fetching de instrucciones a otra linea en la cache. Las técnicas usadas para minimizar esta penalización en otros procesadores incluyen almacenamiento de copias de las *instrucciones* encontradas en el salto objetivo en las tablas de predicción de salto.

de instrucciones [38], similar a la del procesador AMD-K5.

Dentro de la ICache incorpora un campo que indica la línea de cache destino, y dos bits con los que implementa una máquina con 4 estados: no tomado, probablemente no-tomado, problablemente tomado y fuertemente tomado. Cada vez que un salto es tomado se incrementa el valor de los bits y cuando no es tomado se decrementa.

La inicialización de la máquina de estados la realiza de dos maneras: Para algunas instrucciones de salto deja que el compilador, a través de un bit en la instrucción, inicialice el contador de dos bits a probablemente-tomado o probablemente no-tomado. Para el resto de instrucciones lo inicializa como probablemente no-tomado.

M. Sun UltraSparc II

El UltraSparc II utiliza un predictor de saltos de un nivel, con dos bits de información por salto almacenado en la cache de instrucciones. La penalización por un fallo de predicción es de cuatro ciclos, y la tasa de aciertos de predicción de saltos es alrededor de 87% para los programas enteros y 93% para los programas de punto flotante [39].

Este predictor de saltos es similar al predictor de saltos del Pentium que fue analizado en [40], y es claramente analizable. En el caso específico del UltraSparc II, sin embargo, la localidad de la información de la predicción del salto en la cache de instrucciones hace el análisis más complejo. Este tipo de predictor de saltos se encuentra en procesadores embebidos como el ARM11 [41].

N. Sun UltraSparc III

El UltraSparc III tiene un predictor de salto de dos niveles global, el cual combina 14 bits de la dirección del salto con 12 bits de la historia de salto global empleando una función hash no documentada para obtener un indice para una tabla de historia de saltos. La tabla de historia de saltos contiene 16386 (16K) entradas, cada una de las cuales es un contador saturado de dos bits.

Sun reporta una precisión de predicción de alrededor de los 95% sobre los benchmark SPEC95. La penalización por fallo de predicción es de 7 ciclos para los saltos tomados y menor para los no tomados, la cual es significativamente menor que en los procesadores Athlon, Pentium III, y Pentium 4 [42].

IV. Estructura Experimental

A. Entorno de Simulación

Se emplea la infraestructura de simulación SimpleScalar [43] [44] para implementar y comprobar los diferentes predictores de salto en estudio. Al simulador se le hizo una extensión al programa sim-safe y se agrego un archivo de cabecera con las declaraciones de las estructuras de datos que soportan los predictores Bimodal, GAg, PAg, gshare, McFarling, perceptrón y perceptrón rápido.

Las configuraciones de cada uno de los predictores en cuanto al presupuesto Hardware se muestran en la Tabla III (página 11). Es de hacer notar que se ajustaron los valores de los distintos componentes de los predictores de salto con el fin de obtener un presupuesto Hardware igual en cada uno de ellos y que se aproximará al propuesto citado por Hennessy & Patterson [6] para el predictor de saltos de McFarling.

B. Benchmarks

Este estudio se realiza teniendo dos tipos de benchmarks del SPEC2000.

- Siete Benchmarks SPECint00 (enteros): *bzip2*, *gcc*, *gzip*, *mcf*, *parser*, *vortex* y *vpr*.
- Tres benchmarks SPECfp00 (punto-flotante): *ammp*, *art* y *mesa*.

En la Tabla IV (página 12) se detallan los benchmarks a utilizar. El número de instrucciones compiladas para una máquina PISA que se ejecutaron fue de 1500 millones de instrucciones, de las cuales el número de instrucciones de salto para cada benchmark se presenta en la Tabla V (página 12).

V. Resultados de Simulación

A. Precisión de la Predicción de los Saltos

En la Tabla VI (página 11) se muestra el porcentaje de precisión de la predicción de los saltos para los siete benchmarks SPECint00 y en la Figura 9 (página 12) se muestra un diagrama de barras que representa dichos porcentajes de precisión de los predictores.

En estos resultados se observa lo siguiente: Los mejores predictores de salto en promedio son el perceptron y el perceptron rápido con porcentajes de predicción de 96.48% y 96.47% respectivamente.

La menor precisión se observa en el predictor Bimodal con un valor de 92.48% en la aplicación gcc y en promedio un 94.21%.

Después de los predictores perceptron le sigue el McFarling (96.36%), luego el gshare (96.02%), el

TABLA III

COSTO HARDWARE APROXIMADO DE CADA UNO DE LOS PREDICTORES DE SALTO.

Predictor	**Presupuesto Hardware**
Bimodal	$BHT = 16K$; $Presupuesto = 16K * 2bits = 32Kbits$ $Presupuesto = 4KB$ (Figura 2, página 4).
GAg	$GBHR = 14bits$, $GPHT = 16K * 2bits = 32Kbits$; $Presupuesto = (32K + 14)bits \approx 32Kbits$ $Presupuesto \approx 4KB$ (Figura 3, página 4). k=14 (Tabla I, página 3).
PAg	$PBHT = 1K * 13bits = 13Kbits$, $GPHT = 8K * 2bits = 16Kbits$; $Presupuesto = (29K + 10)bits \approx 3.625KB$ (Figura 4, página 4) (k=13 , b=1K) (Tabla I, página 3).
gshare	$GBHR = 14bits$, $GPHT = 16K * 2bits = 32Kbits$; $Presupuesto = (32K + 14)bits \approx 4KB$ (Ver Figura 5).
McFarling	Predictor 1 (GAg): $GBHR = 12bits$, $GPHT = 4K * 2bits = 8Kbits$; $Presupuesto1 \approx 1KB$ (Figura 3, página 4). k=12 (Tabla I, página 3). Predictor 2 (PAg): $PBHT = 1K * 10bits = 10Kbits$, $GPHT = 1K * 2bits = 2Kbits$; $Presupuesto2 \approx 1.5KB$ (Figura 4, página 4) (k=10 , b=1K) (Tabla I, página 3). Predictor Selector (Bimodal): $BHT = 4K * 2bits = 8Kbits$; $Presupuesto\,Selector = 1KB$ (Figura 2, página 4). $Presupuesto = (28K + 12)bits \approx 3.5KB$
Perceptron	$H = 7bits$, $N = 4K$; $Presupuesto = (32K + 8)bits$ $Presupuesto \approx 4KB$ (Figura 7, página 6).
Perceptron Rapido	$H = 7bits$, $N = 4K$; $Presupuesto = (32K + 8)bits$ $Presupuesto \approx 4KB$ (Figura 7, página 6).

TABLA VI

PORCENTAJE DE PRECISIÓN DE LA PREDICCIÓN DE SALTO PARA LOS SIETE BENCHMARKS SPECINT00.

	Benchmark SPECint00							
Predictor	**bzip2**	**gcc**	**gzip**	**mcf**	**parser**	**vortex**	**vpr**	**promedio**
Bimodal	96.28	92.48	90.23	94.51	93.91	98.69	93.38	94.21
GAg	96.44	94.34	92.43	97.73	95.85	95.57	97.21	95.65
PAg	96.43	92.94	91.73	96.99	96.09	97.20	97.97	95.62
Gshare	95.29	94.91	92.43	97.26	96.19	98.06	97.97	96.02
McFarling	96.66	94.62	92.85	97.86	96.18	98.06	98.32	96.36
Perceptron	96.80	95.85	93.01	97.63	95.88	99.25	96.92	96.48
Perceptron Rápido	95.38	96.50	93.07	97.67	96.15	99.12	97.42	96.47

GAg (95.65%), el *PAg* (95.62%) y por último el Bimodal (94.21%) en orden decreciente en cuanto a precisión de la predicción.

En la Tabla VII (página 13) se muestra el porcentaje de precisión de la predicción de salto para los tres benchmarks SPECfp00 y en la Figura 10 (página 13) se muestra un diagrama de barras que representa dichos porcentajes de precisión de los predictores.

En estos resultados se observa lo siguiente: Los mejores predictores de salto en promedio son el *PAg* y el MCFarling con porcentajes de predicción de 99.26% y 99.20% respectivamente.

La menor precisión se observa en el predictor gshare con un valor de 94.54% en la aplicación ammp y en promedio un 96.42%.

El orden de los predictores desde el mejor hasta el menos bueno es el siguiente: El *PAg* (99.26%), el McFarling (99.20%), el *GAg* (98.07%), el perceptron rápido (97.93%), el perceptron (97.90%), el Bimodal (96.54%) *PAg* y por último el gshare (96.42%).

Comparando los resultados obtenidos en las simulaciones de los programas de prueba para los enteros con respecto a los programas en punto flotante, se observa que los predictores de salto para los programas de aplicación en punto flotante funcionan mejor, en tal medida que para el peor desempeño promedio del predictor en punto flotante que corresponde al Bimodal (96.54%), este porcentaje es

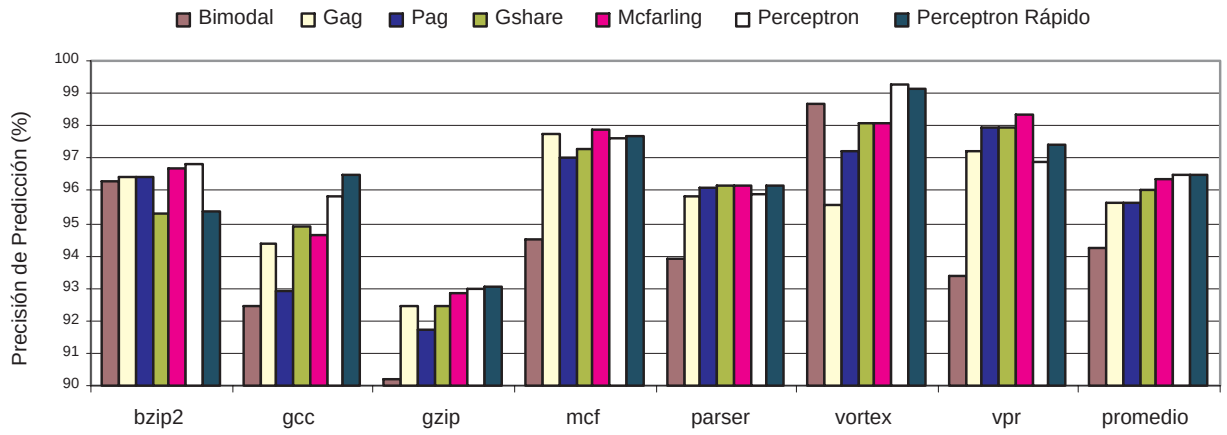

Fig. 9. Porcentaje de Precisión de Predicción para los Benchmarks Specint00.

TABLA IV
BENCHMARK SPEC2000 UTILIZADO PARA LA EVALUACIÓN DE LAS ESTRATEGIAS DE PREDICCIÓN DE SALTO.

Aplicación	Descripción
*bzip*2	Compresión de archivos
gcc	Compilación de programas en Lenguaje C
gzip	Compresión de archivos
mcf	Optimización Combinatorial
parser	Procesamiento de palabras
vortex	Base de datos orientada a objetos
vpr	Ruteo y colocamiento en Circuito FPGA
ammp	Cálculos Químicos
art	Redes Neuronales/Reconocimiento Imágenes
mesa	Biblioteca Gráfica 3-D

TABLA V
NÚMERO DE SALTOS EJECUTADOS EN LAS SIMULACIONES DE CADA APLICACIÓN DE PRUEBA.

Aplicación	Saltos Ejecutados
*bzip*2	217729346
gcc	262888371
gzip	250236219
mcf	431263980
parser	340437835
vortex	226049671
vpr	258055437
ammp	454606550
art	232123999
mesa	192650472

superior al mejor desempeño promedio del predictor en entero que corresponde al perceptron (96.48%). En otras palabras, el peor resultado promedio en aplicaciones punto flotante es aún superior al mejor resultado promedio en aplicaciones enteras.

VI. Trabajo Futuro

En este momento es posible obtener medidas de precisión de la predicción para las distintas estrategias de predicción con la modificación del simulador SimpleScalar, lo cual garantizará la validación de los resultados obtenidos en hardware. En otras palabras, se tiene desarrollada una estructura de simulación para predictores dinámicos de salto integrada a Simplescalar.

Por otro lado estan codificados en VHDL (lenguaje descriptor de hardware) los mismos esquemas de predicción antes descritos. Con ellos se procederá a realizar simulaciones funcionales y temporales para validar la correcta operativilidad de los mismos, luego se sintetizarán a un FPGA objetivo (previo estudio de selección) y se obtendrán métricas de consumo de energía, consumo de área y latencia, lo que vendria a representar la implementación de la plataforma de comparación de los predictores dinámicos de salto.

A continuación se listan otras consideraciones que surgen a futuro sobre la base del presente trabajo.

- Ratificar que el presupuesto de bits no es una medida o métrica efectiva para realizar comparaciones entre predictores, el objetivo es encontrar números que lo indiquen. Se propone contrastar el predictor propuesto por Jiménez vs los clásicos.

- Investigar cuál es la predicción de cada uno de los predictores en estudio, sobre la base de un mismo presupuesto hardware. Mostrar dicha información de manera gráfica.

- Encontrar el costo asociado en Hardware (presupuesto) para cada uno de los predictores en estudio, asumiendo las mismas condiciones de funcionamiento es decir con el mismo índice o

TABLA VII

PORCENTAJE DE PRECISIÓN DE LA PREDICCIÓN DE SALTO PARA LOS TRES BENCHMARKS SPECFP00.

	Benchmark SPECfp00			
Predictor	**ammp**	**art**	**mesa**	**promedio**
Bimodal	99.03	95.08	95.50	96.54
GAg	99.48	97.67	97.06	98.07
PAg	99.55	99.49	98.75	99.26
Gshare	94.54	97.41	97.30	96.42
McFarling	99.62	99.03	98.96	99.20
Perceptron	99.50	97.20	97.01	97.90
Perceptron Rápido	99.53	97.19	97.06	97.93

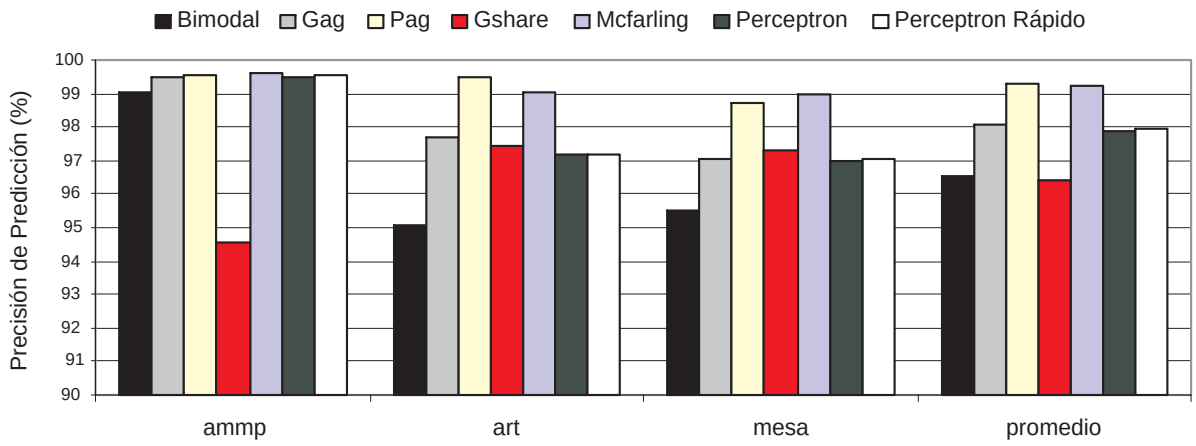

Fig. 10. Porcentaje de Precisión de Predicción para los Benchmarks Specfp00.

porcentaje de precisión.

- Inferir medidas o proponer medidas confiables tales como Unidad de Hardware por Unidad de predicción, $\frac{\%Chip}{\%precision}$, establecer la relación con respecto al tiempo como métrica.

- Investigar cuál es la predicción de cada uno de los predictores en estudio, sobre la base de un mismo consumo de energía. Mostrar dicha información de manera gráfica.

- Encontrar el costo asociado en consumo de energía para cada uno de los predictores en estudio, asumiendo las mismas condiciones de funcionamiento es decir con el mismo índice o porcentaje de precisión.

- Establecer cuál esquema ofrece mayor potencial para la reducción de consumo de energía. Este estudio es de manera independiente.

- Determinar el impacto de cada predictor al ser conectado a microprocesadores superscalares.

- Determinar el impacto de cada predictor al ser conectado a microprocesadores VLIW.

Es necesario aclarar que para lograr determinar el impacto de los predictores al ser conectados a microprocesadores con arquitectura VLIW se debe desarrollar con anterioridad un entorno de validación como el que se tiene para los superscalares en Simplescalar. Para ello, se seleccionara un microprocesador VLIW especifico y a partir de alli se construira este entorno.

En relación al predictor de saltos propuesto por Jiménez [9] se tiene como trabajo a futuro el disminuir la latencia que presenta, a través de la optimización del algoritmo de cálculo de la salida del perceptron.

REFERENCIAS

[1] M. S. Lam and R. P. Wilson, *"Limits on Control Flow Paralelism". In Proceedings of the 25th Annual International Symposium on Computer Architecture* , pp 46-57, 1992.

[2] C. Young and M. D. Smith, *"Improving the Accuracy of Static Branch Prediction Using Branch Correlation". In Proceedings of the 6th Annual International Conference on Architectural Support for Programming Languages and Operating Systems* , pp 232-241, 1994.

[3] A. González, *"A Survey of Branch Techniques in Pipelined Processors". In Euromicro Journal* , pp 243-257, 1993.

[4] C. H. Perleberg and A. J. Smith, *"Branch Target Buffer Design and Optimization". In IEEE Transactions on Computers* , Volume 42, Number 4, pp 396-412, April 1993.

[5] C. Lee, K. Chen and T. N. Mudge, *"The Bi-mode Branch Predictor". In Proceedings of the 30th Annual Interna-*

tional Symposium on Microarchitecture, pp 4-13, December 1997.

[6] J. L. Hennessy and D. A. Patterson, *"Computer Architecture A Quantitative Approach". Tercera Edición. Morgan Kaufmann Publishers* , 2003.

[7] S. McFarling, *"Combining Branch Predictors". In Tech. Report TN-36, Digital Western Research Laboratory*, June 1993.

[8] D. A. Jiménez and C. Lin, *"Neural Methods for Dynamic Branch Prediction". In ACM Transactions on Computer Systems* , Volume 20, Number 4, pp 369-397, November 2002.

[9] D. A. Jiménez, *"Fast Path-Based Neural Branch Prediction". In Proceedings of the 36th International Symposium on Microarchitecture* , (MICRO-36 2003).

[10] J. Lee and A. J. Smith, *"Branch Prediction Strategies and Branch Target Buffer Design". In Computer* , Volume 17, Number 1, pp 6-22, 1984.

[11] J. E. Smith, *"A Study of Branch Prediction Strategies". In Proceedings of the 8th Annual International Symposium on Computer Architecture* , pp 135-148, April 1981.

[12] T. Yeh and Y. N. Patt, *"Two Level Adaptative Branch Prediction". In Proceedings of the 24th Annual International Symposium on Microarchitecture* , pp 51-61, 1991.

[13] S. Pan, K. So and J. Rahmeh, *"Improving the Accuracy of Dynamic Branch Prediction Using Branch Correlation". In Proceedings of the 5th Annual Conference on Architectural Support for Programming Languages and Operating Systems* , pp 76-84, 1992.

[14] T. Yeh and Y. N. Patt, *"Alternative Implementations of Two-Level Adaptative Branch Prediction". In Proceedings of the 19th Annual International Symposium on Computer Architecture* , pp 124-134, 1992.

[15] T. Yeh and Y. N. Patt, *"A Comparasion of Dynamic Branch Predictors that use Two Levels of Branch History". In Proceedings of the 20th Annual International Symposium on Computer Architecture* , pp 257-266, 1993.

[16] J. Silc, B. Rabic and T. Ungerer, *"Processor Architecture from Data Flow to Superscalar and Beyond"*. pp 134-145, Springer-Verlag 1999.

[17] P. Chang, E. Hao and Y. N. Patt, *"Alternative Implementations of Hybrid Branch Predictors". In Proceedings of the 28th Annual International Symposium on Computer Microrchitecture* , pp 252-257, 1995.

[18] D. A. Jiménez, *"Dynamic Branch Prediction with Perceptrons". In Proceedings of the 7th International Symposium on High Performance Computer Architecture* , pp 197-206, January 2001.

[19] L. Vintan and M. Iridon, *"Towards a High Performance Neural Branch Predictor". In Proceedings of the 9th International Joint Conference on Neural Networks* , pp 868-873, July 1999.

[20] A. Seznec, *"Redundant History Skewed Perceptron Predictors: Pushing limits on global history branch predictors". In Technical Report 1554* , IRISA, September 2003.

[21] A. Seznec, *"Revisiting the Perceptron Predictor". In Technical Report 1620* , IRISA, May 2004.

[22] B. Calder and D. Grunwald, *"Fast and Accurate Instruction Fetch and Branch Prediction". In Proceedings of 21th Annual International Symposium on Computer Architecture*, pp 2-11, 1994.

[23] P. Hsu, *"Designing the TFP Microprocessor". In IEEE Micro* , Volume 14, Number 2, pp 23-33, 1994.

[24] K. Yeager, *"The R10000 Superscalar Microprocessor". In IEEE Micro* Volume 16, Number 2, pp 28-40, 1996.

[25] D. Hunt, *"Advanced Performance Features of the 64-bit PA-8000". In Proceedings of Compcon 95*, pp 123-128, 1995.

[26] J. H. Edmonson, P. Rubinfeld, R. Preston, V. Rajagoplan, *"Superscalar Instruction Execution in the 21164 Alpha Microprocessor". In IEEE Micro* , Volume 15, Number 2, pp 33-43, 1995.

[27] R. E. Kessler, E. J. McLellan, and D. A. Webb, *"The Alpha 21264 Microprocessor Architecture". In Proceedings of ICCD* , December, 1998.

[28] D. Christie, *"Developing the AMD-K5 Architecture". In IEEE Micro* , Volume 16, Number 2, pp 16-26, 1996.

[29] M. Johnson, *"Superscalar Microprocessor Design". Prentice Hall Inc* , 1991.

[30] Jam's X86, *"CPU Comparative Table"*. http://www.io.com/ kazushi/PC/X86-cmp.html, 1998.

[31] D. Gu, O. Zendra and K. Driesen, *"The Impact of Branch Prediction of Branch on Control Structures for Dynamic Dispatch in Java". Technical Report Number 4547* , Institut National de Recherche en Informatique et Automatique (INRIA), INRIA-Lorraine/LORIA, http://www.loria.fr., September 2002.

[32] A. Kaiser, *"K7 Branch Prediction"*. http://www.s.netic.de/ak/, December 1999.

[33] AMD, *"AMD Athlon Processor: x86 Code Optimization Guide". Publication no: 22007 K*, February 2002.

[34] Agner Fog, *"Branch Prediction in the Pentium Family". Microprocessor Resources*, http://x86.ddj.com/articles/branch/branchprediction.htm

[35] S. Petters, *"Worst Case Execution Time Estimation for Advanced Processor Architectures". PhD. thesis*, Technische Universität National München, August 2002.

[36] Intel, *"Intel Architecture Optimization Reference Manual". Document Number: 245127-001*, 1999.

[37] G. Hinton, D. Sager, M. Upton, D. Boggs, D. Carmean, A. Kyker and P. Roussel, *"The Microarchitecture of the Pentium 4 Processor". Intel Technology Journal*, Q1 2001.

[38] M. Tremblay and J.M. O'Connor, *"UltraSparc I: A Four-Issue Processor Supporting Multimedia". In IEEE Micro*, Volume 16, Number 2, pp 42-49, 1996.

[39] Sun Microsystems, *"UltraSPARC-IIi User's Manual". Part No: 805-0087-01*, 1997.

[40] A. Colin and I. Puaut, *"Worst Case Execution Time Analysis for a Processor with Branch Prediction". Journal of Real-Time Systems* , 18(2/3):249-274, May 2000.

[41] ARM Ltd, *"The ARM11 Microprocessor and ARM PrimeXsys Platform"*. White paper found at http://www.arm.com/armtech/ARM11, October 2002.

[42] P. Song, *"UltraSparc-3 Aims at MP Servers". Microprocessor Report*, October 27, 1997.

[43] D. Burger and T. Austin, *"The SimpleScalar Tool Set, Version 3.0". Technical Report* Computer Sciences Departament, University of Wisconsin-Madinson, 1999.

[44] D. Burger, T. Austin and S. Bennett, *"Evaluating Future Microprocessors: The SimpleScalar Tool Set ". Technical Report TR-1308*, Computer Sciences Departament, University of Wisconsin-Madinson, July 1996.