Computación y Sistemas. Vol. 7 No. 2 pp. 130 - 147

Diseño de Sistemas de Memoria Cache de Alto Rendimiento aplicando Algoritmos de Acceso Seudo-Especulativo

Design of High Performance Cache Memory Systems applying Pseudo-Speculative Access Algorithm

Graduated: Oscar Camacho Nieto

Centro de Investigación en Computación
Juan de Dios Bátiz s/n esq. Miguel Othón de Mendizábal
Unidad Profesional Adolfo López Mateos
Del. Gustavo A. Madero, México, D. F.
e-mail: oscarc@cic.ipn.mx

Advisors: Luis Alfonso Villa Vargas. Instituto Mexicano del Petróleo
Juan Luis Díaz de León Santiago. Centro de Investigación en Computación
Cornelio Yáñez Márquez. Centro de Investigación en Computación

Resumen

La diferencia que existe entre el tiempo de ciclo de operación del procesador y el tiempo de acceso a memoria cada vez es mayor. El rendimiento de los procesadores se ha venido incrementando aproximadamente un 60% cada año debido a la reducción del tiempo de ciclo de reloj y al incremento del número de instrucciones ejecutadas por ciclo (IPC). Sin embargo, el tiempo de acceso a las memorias DRAMS sólo mejora un 10% por año, aunque la capacidad se duplica cada año y medio, según la Ley de Moore. Para reducir esta diferencia de tiempos se utiliza una organización de memoria jerarquizada con el objetivo de que el nivel cercano al procesador (cache) almacene temporalmente el contenido de la memoria principal que se prevé pronto será utilizado. Los factores que afectan el rendimiento son: el tiempo necesario para obtener un dato de la cache y el número de accesos que se resuelven directamente desde la cache. Este trabajo se centra en incrementar la frecuencia de aciertos y reducir el tiempo medio de acceso en la cache sin incrementar el tiempo de ciclo del procesador, manteniendo dentro de límites razonables la latencia de acceso. Usando la capacidad de predicción que presentan las referencias a memoria para guiar la gestión y acceso al primer nivel en caches de acceso secuencial, proponemos un esquema dinámico e inteligente para acceder a la cache de datos del primer nivel en un sistema jerarquizado. La evaluación muestra que el esquema propuesto, con respecto a la cache convencional de mapeo directo, reduce el tiempo promedio de acceso en 14.71%, 11.47% y 12.80% en capacidades de 8k, 16k y 32k, respectivamente. Asimismo, nuestro esquema mantiene una tasa de fallos similar a la de una cache convencional asociativa de dos vías.
Palabras clave: Memoria cache, acceso seudo-especulativo.

Abstract

The gap between the cycle time of processor and the access time to memory go on being eminent. The processor performance increases about 60% by year because of reduction of cycle time and the rise in the number of instruction processed by cycle. Nevertheless, DRAM memories access time only reach an improvement about 10% by year although capacity is duplicate every one and a half year, according to Moore's Law . To reduce this contrast of times, a hierarchical memory organization is used, with the purpose that the level near to the processor holds the content of main memory that is foreseen to be referenced. The factors that affect the performance are the necessary time to get one data to the first level L1 the cache and the fraction of references satisfied from the cache. Our work consists in increase the frequency of hits and reduces the average memory-access time in cache, without increase the cycle time of processor supported inside limit fair the access latency. Using address prediction at capacity to guide the management and access to the firs level cache, in sequential access caches, we propose a dynamic and clever scheme to access caches memories. The evaluation shows that this scheme can achieve an average improvement in the average memory-access time of 14.71% 11.47% and 12.80% in caches of 8kb, 16kb and 32kb of capability respectively, over conventional directed mapped caches memories. At the same time, our scheme maintains the frequency of miss similar to the conventional 2-way associative cache memory.
Keywords: Memoria cache, acceso seudo-especulativo.

1. Introducción

Este trabajo se enfoca en mejorar el rendimiento del sistema de memoria de los procesadores de propósito general, particularmente la memoria cache. En una memoria cache convencional, para leer un dato podemos distinguir varias fases: calcular, a partir de la dirección efectiva, un índice para acceder a la cache; leer las etiquetas; comprobar las etiquetas; y leer el bloque. Dependiendo de la función de mapeo utilizada en la cache, algunas fases pueden efectuarse en paralelo, aunque otras son estrictamente secuenciales.

En una cache de mapeo directo, el bloque referido sólo puede encontrarse en una entrada; pero en una cache asociativa por conjuntos, varias entradas son candidatas para almacenar el bloque. Grados elevados de asociatividad disminuyen la frecuencia de fallos, pero esto incrementa el tiempo medio de acceso a los datos debido al tiempo que se requiere para la conmutación de las posibles ubicaciones donde puede estar ubicada la información. Ahora bien, el tiempo de acceso y el grado de asociatividad del primer nivel de la jerarquía son un compromiso de diseño, ya que el acceso al primer nivel de cache está incluido en el camino crítico del procesador.

La búsqueda del bloque entre las posibles entradas de la memoria cache puede efectuarse de forma paralela o secuencial. En el caso de búsqueda paralela, después de leer las etiquetas correspondientes a los bloques del conjunto, su comparación con la etiqueta de la dirección efectiva permite determinar la posición en que se almacena el bloque en caso de acierto. De forma concurrente a la lectura de etiquetas se pueden leer los correspondientes bloques de datos. En búsqueda secuencial, las distintas posiciones del conjunto de entradas en que puede estar ubicado un bloque se recorren de forma secuencial; entonces, el tiempo de acceso en caso de acierto no es constante: depende de la posición del bloque referido en la secuencia de búsqueda.

Debido a la propiedad de localidad temporal que presentan las referencias a memoria, la ubicación más recientemente usada presenta mayor importancia en el orden de búsqueda; por lo tanto, en un esquema de búsqueda secuencial, es importante determinar el orden de búsqueda de la ubicación más recientemente usada (MRU) antes de acceder a la cache.

Al efectuar una búsqueda secuencial de las distintas posiciones en que puede estar ubicado un bloque, el tiempo medio de acceso en caso de acierto es superior al tiempo de acceso a cache; por lo tanto, para reducir el tiempo medio de acceso puede predecirse la primera entrada que debe comprobarse. Por ejemplo, B. Calder (1996) propone la "*Predictive Sequential Associative Cache*", donde se tiene una cache de acceso secuencial con un esquema de predicción que predice la primera entrada que debe ser accesada. En su esquema usa varias fuentes de predicción para accederlo. Calder encuentra que la fuente de predicción que muestra mayor fiabilidad para accesar al esquema de predicción de primera entrada es la dirección efectiva, la cual se calcula en la etapa de ejecución del procesador para accesar a la cache. Debido a que el cálculo de la dirección efectiva se hace justo antes de accesar a la cache, el tiempo de ciclo del procesador se incrementa si se utiliza la dirección efectiva para accesar al predictor de primera entrada y posteriormente a la cache. El acceso al predictor y a la cache son dos accesos secuenciales. Las referencias a memoria pueden caracterizarse en función de la secuencia de direcciones que generan; por lo tanto, pueden utilizarse reconocedores de patrones dinámicos que permitan predecir la dirección (efectiva) que será accesada en una referencia determinada. A diferencia de los trabajos previos, para soportar el tiempo de acceso al predictor de primera entrada, en este trabajo **proponemos predecir con anticipación las direcciones efectivas** en relación a la etapa en que se calculan dichas direcciones, evitando así el incremento en el tiempo de ciclo del procesador.

En los esquemas de búsqueda secuencial propuestos a la fecha, podemos encontrar dos formas distintas para determinar el orden de búsqueda. La primera se distingue por acceder a la cache con un orden de búsqueda determinado dinámicamente (*Predictive-Sequential-Associative*). En la segunda, accedemos a la cache con un orden de búsqueda determinado estáticamente (*Hash-Rehash* y *Column-Associative*); en este caso se requiere intercambiar los bloques entre sus posibles ubicaciones para mantener el bloque MRU en la ubicación determinada estáticamente.

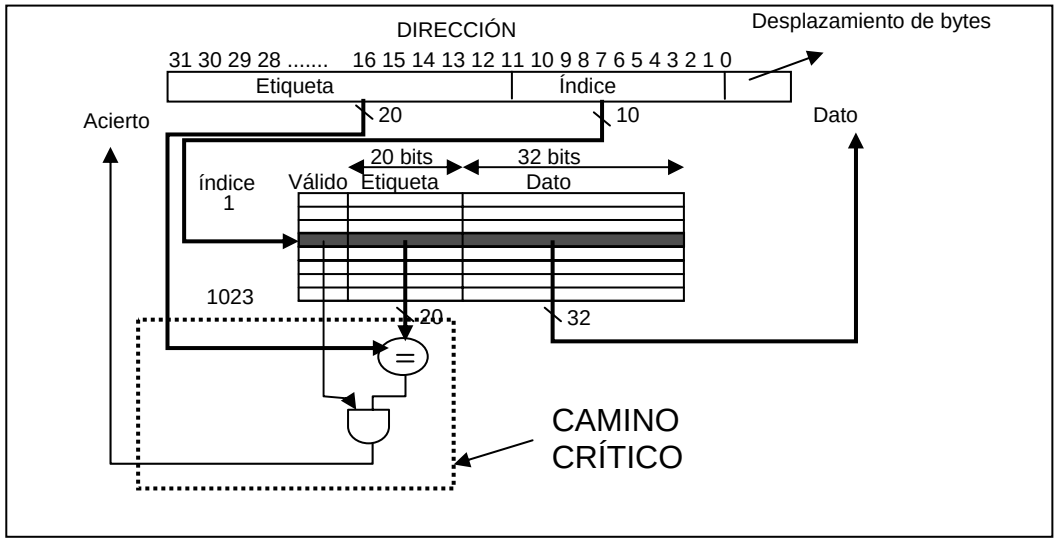
En el esquema que se propone en este trabajo, cuando las direcciones no sean predecibles, la primera entrada que debe comprobarse será determinada estáticamente. Para reducir el tiempo medio de acceso, se actuará diseñando un mecanismo de ubicación y reemplazo que priorice la utilización de la primera entrada. En caso de una instrucción de carga (*load*), el dato no se suministra a instrucciones dependientes hasta que no se haya comprobado que la predicción de la dirección efectiva sea correcta; es decir, no se evalúa como un mecanismo especulativo, sino como uno **seudo-especulativo**.

El resto del trabajo está organizado como sigue: en la sección 2 se hace una descripción resumida de los microprocesadores de propósito general, haciendo una breve descripción de sus características técnicas y de la manera en que trabaja el sistema de memoria, y una caracterización de los esquemas de memoria cache conocidos; la sección 3 incluye la presentación de algunos modelos analíticos existentes para complementar la evaluación con herramientas de simulación que finalmente derivan en un modelo experimental eficiente. En la sección 4 se describe la propuesta central de este trabajo que hemos denominado *"Múltiples Disciplinas de Acceso a Cache" (MDAC)*, y se muestran las disquisiciones experimentales de este esquema, el cual se basa en la experiencia adquirida del estudio realizado sobre la capacidad de predicción que presentan las referencias a memoria. Finalmente, en la sección 5 se presentan las conclusiones de este trabajo y las propuestas para trabajos futuros.

2. Microprocesadores y Cache

Desde la aparición del transistor en diciembre de 1947 hasta nuestros días, la arquitectura de las computadoras ha experimentado un desarrollo a pasos agigantados. Mientras que el primer circuito integrado contenía aproximadamente 2000 transistores, el actual Pentium IV reúne en una sola pieza de silicio cerca de 55 millones de transistores; es 250 mil veces más poderoso, 50 mil veces más barato y mucho más pequeño. A los procesadores modernos se les llama comúnmente procesadores superescalares, y algunos de ellos son: el Pentium IV de Intel, el 21264C de Alpha, el UltraSparc III de SUN, Power 4 de IBM, el Athlon XP de AMD, el R14000 de MIPS, el PA-8700 de HP. Los diseñadores de este tipo de procesadores mantienen el compromiso, adquirido desde hace cinco décadas, de continuar buscando mecanismos que mejoren su rendimiento. La enorme cantidad de transistores presente en los procesadores superescalares nos ha permitido incorporar técnicas de procesamiento cada vez más complejas; de igual forma, se ha integrado en el dispositivo el primer nivel de memoria, comúnmente llamado memoria cache (la mayoría de procesadores actuales disponen de un primer nivel de memoria *cache*, la cual se compone de dos módulos independientes, uno para datos y otro para instrucciones).

Actualmente todos los procesadores de propósito general son procesadores segmentados con una estructura superescalar.



Cache de Mapeo Directo

La segmentación (*Pipelining*) surge con la idea de mejorar la productividad (en cuanto al número de instrucciones ejecutadas se refiere) de los procesadores, y consiste en dividir la ejecución de una instrucción entre las distintas etapas que intervienen en su proceso de ejecución, de tal forma que cuando la instrucción *X* libera la etapa *N* para continuar con la etapa *N*+1, la siguiente instrucción inicia su ejecución en la etapa *N*. Esta idea se puede ver como una línea de ensamble de automóviles donde finalmente, cuando la cadena de montaje se encuentra llena, en cada etapa se produce un automóvil, y de esta forma se pueden estar construyendo varios automóviles a la vez. De igual manera, con la segmentación es posible ejecutar varias instrucciones a la mismo tiempo; esto acuñó el nombre de Paralelismo a Nivel de Instrucción (*Instruction Level Parallel: ILP*).

Desde que surgió la idea de construir una arquitectura con programa almacenado hasta la fecha, las computadoras actuales mantienen una estructura muy similar, la cual consiste de una parte donde se almacenan los datos e instrucciones (memoria), una sección que ejecuta las instrucciones y manipula los datos (unidad de procesamiento), un área que se

encarga del contacto con el exterior (los dispositivos de entrada y salida) y el elemento que controla las actividades de todo el conjunto (unidad de control). Las aportaciones de los arquitectos de computadoras se han enfocado principalmente a obtener el máximo rendimiento posible de la estructura de programa almacenado.

De igual manera, la forma básica de ejecutar las instrucciones no ha sufrido grandes cambios; un programa consta de un grupo de instrucciones y datos, el procesador obtiene las instrucciones y las ejecuta. Hay varias etapas involucradas en la ejecución de una instrucción: la búsqueda de la instrucción en la memoria, decodificación de la instrucción y obtención de sus registros fuente, ejecución de la operación indicada por la instrucción, acceso a la memoria en caso de una instrucción de referencia a ésta y por último la escritura del resultado de la instrucción.

El tiempo en ciclos de reloj del procesador requerido para la ejecución de una instrucción, multiplicado por el número de instrucciones que conforman el programa, nos proporciona el tiempo que requiere un procesador para procesar dicho programa, el cual depende de tres factores: número de instrucciones necesarias para procesar el programa, ciclos de reloj necesarios para ejecutar una instrucción y período del ciclo de reloj.

Reduciendo cualquiera de los tres factores en la ejecución de un determinado programa, se mejora el rendimiento del procesador. Por ejemplo, la segmentación (*Pipelining)* es una innovación que permitió reducir el período del ciclo de reloj. Debido al comportamiento natural de los programas, las instrucciones y datos que se presentan de forma estática (código que entrega el compilador) en el código del programa al ser ejecutadas son accedidas más de una vez en forma dinámica (en ejecución); tal es el caso de las instrucciones de referencia a memoria. Debido a que la mayoría de los programas no accesan a todo el código o a los datos de memoria uniformemente, es decir, favorecen una parte de su espacio de direcciones en cualquier instante de tiempo -por ejemplo, el cálculo de una matriz- es necesario mantener más cercana al procesador la información que se prevé que será utilizada en el corto tiempo y la información que se encuentre contigua a la información que se está procesando.

Cuando se accesa al mismo espacio de direcciones en períodos de tiempo relativamente pequeños, estamos hablando de la propiedad de localidad temporal; cuando se accesa a espacios contiguos de direcciones en períodos de tiempo relativamente pequeños, estamos hablando de la propiedad de localidad espacial. Estas propiedades, bien utilizadas, evitan un viaje demasiado largo (en tiempo de procesamiento) hasta el sistema principal de memoria, donde se encuentra almacenado todo el programa.

Con el objetivo de reducir el tiempo necesario para accesar a la memoria principal, tomando en cuenta las propiedades de localidad, se ha propuesto una memoria jerarquizada donde el nivel más cercano al procesador es más pequeño, más rápido, aunque de mayor costo y que usa una tecnología distinta a los niveles más lejanos. A este nivel más cercano al procesador dentro de la jerarquía se le denomina **memoria cache**.

En los procesadores modernos (por ejemplo el Pentium III de INTEL), en el mismo circuito integrado se tienen los niveles de memoria cache, y esto ha mejorado el rendimiento general, pero aun así no se consigue cerrar la brecha que existe entre la velocidad de procesamiento y el tiempo de acceso al sistema de memoria. La cache convencional es un buffer que almacena información; dependiendo de la forma en que se escribe o se lee dicha información, ésta puede ser de **mapeo directo** o **asociativa de *n* vías**; cuando *n* alcanza su valor máximo se dice que es **completamente asociativa**.

La cache se presenta como un arreglo de dos dimensiones; la primera dimensión contiene el conjunto donde se encuentra el bloque de datos y la segunda, contiene el grado de asociatividad con que cuenta dicho conjunto (número fijo de posiciones donde puede ubicarse cada bloque); para asociatividad igual a uno, se dice que tenemos una cache de *mapeo directo*. Para asociatividades mayores a uno se dice que tenemos una cache *asociativa por conjuntos de n vías.* Cuando *n* alcanza su máximo posible se dice que la cache es completamente asociativa.

Para mejorar el rendimiento de la cache, se puede especular entre cuál es la **capacidad** óptima y cuál es el grado de asociatividad que mejor rendimiento presenta, entre otras características. Para medir estos parámetros se usan como figuras de mérito: la tasa de fallos y el tiempo medio de acceso. Un fallo se presenta cuando la información que se busca no se encuentra en la cache y es necesario ir al siguiente nivel en la jerarquía de memoria para traer el dato requerido.

La tasa de fallos es una relación que involucra al número de veces que se busca un dato en la cache y no se encuentra, dividido por el número total de referencias hechas a la cache. El tiempo medio de acceso se mide en ciclos de reloj del

procesador y es el tiempo que tarda éste en obtener un acierto en memoria más la tasa de fallos por la penalización. La cache se compone de Memorias Estáticas de Acceso Aleatorio (*Static Random Access Memory*).

Cuando se proponen nuevos esquemas de memoria cache, su evaluación generalmente parte de un modelo de procesador convencional y consiste en buscar la mejor configuración del esquema, de forma que se minimice el tiempo de ejecución de ciertos programas de prueba. Podemos definir este tiempo como:

$$T = nct$$

donde T es el tiempo de ejecución, n es el número de instrucciones, c es el número de ciclos por instrucción y t es el tiempo de ciclo.

Las figuras de mérito usadas en este trabajo para medir el rendimiento de la cache son la tasa de fallos y el tiempo medio de acceso. La tasa de fallos se obtiene de la siguiente forma:

$$Tf = Nf/Ni = 1 - Ta$$

donde Tf es la tasa de fallos, Nf es el número total de fallos, Ni es el número total de instrucciones y Ta es la Tasa de aciertos. Se considera un fallo cuando se intenta acceder a la cache y no se encuentra el dato; por lo tanto, es necesario pagar una penalización en tiempo (latencia de fallo) para traer el dato del siguiente nivel de la jerarquía de memoria. El tiempo medio de acceso se obtiene de la siguiente forma:

$$Tma = Tac + Tf*Pf$$

donde Tma es el tiempo medio de acceso, Tac es el tiempo de acirto, Tf es la tasa de fallos y Pf es la penalización de fallos. El tiempo de acierto es el que se requiere para solicitar y obtener un dato desde el primer nivel de la jerarquía de memoria.

Las caches de mapeo directo presentan una característica: el tiempo medio de acceso es el menor; el problema con las caches de mapeo directo se presenta cuando dos direcciones de referencia a memoria entran en conflicto, al tratar de acceder a la misma localidad en memoria, y esto incrementa la tasa de fallos, es decir: la dirección de referencia *A* escribe en la localidad que antes escribió la dirección *B*. Esto significa que se elimina de la cache el dato que escribió la dirección *B*, y si *B* nuevamente quiere leer el dato que escribió se encontrará con que falla, debido a que el dato que ahora se encuentra en esa localidad pertenece a la dirección de referencia *A*; si esto se hace de forma intermitente se estará fallando constantemente; es decir, ni la dirección de referencia *A* ni la dirección de referencia *B* tendrán acierto al acceder a la cache.

Para evitar este problema se implementó la cache convencional asociativa por conjuntos de *n* vías; en este tipo de cache se cumple que a medida que crece *n*, disminuye la posibilidad de conflictos, pero como se incrementa la ruta crítica, esto a su vez incrementa el tiempo medio de acceso. Finalmente, lo anterior presenta un dilema sobre qué conviene mantener: el tiempo medio de acceso de las caches de mapeo directo aun cuando su tasa de fallos es muy alta, o elegir una menor tasa de fallos de las caches asociativas pero con un tiempo de acceso que se incrementa conforme se aumenta el grado de asociatividad.

Para mejorar el rendimiento de la memoria cache se buscan caches alternativas a las convencionales, donde una de las mejores aportaciones son las caches híbridas, las cuales se comportan como una cache de mapeo directo pero implícitamente manejan asociatividad de hasta dos vías. Este tipo de caches cambian la forma en que se accede a los datos, haciendo un acceso secuencial y buscando que el dato se encuentre en el primer acceso.

Uno de los problemas con los que tratan los diseñadores de sistemas de memoria cache de alto rendimiento es la elección del tipo de direccionamiento (real o virtual) que se usará para accesar a la memoria.
Para reducir el tiempo efectivo de acceso a memoria, el acceso debe realizarse tan pronto sea válida la dirección efectiva de la referencia a memoria. En principio, en la mayoría de las computadoras la memoria cache se accede con la dirección real (física) de memoria, en tanto que la Unidad Aritmético Lógica (ALU) genera direcciones de memoria virtual. Puesto que es necesario hacer una traducción de memoria virtual a memora física, se necesita acceder a las tablas de páginas de memoria principal, lo que afecta el rendimiento del sistema de memoria debido al retardo provocado por la traducción. Para disminuir

este retardo, se ha implementado otra cache que almacenará las tablas de páginas usadas. Estamos hablando del buffer de traducción anticipada (*Translation Lookaside Buffer* TLB).

Cuando se está trabajando con caches de dirección física, se usan métodos para minimizar la traducción de dirección virtual a física. Debido a que sólo se deben traducir los bits que indican el número de página (TLB), los bits restantes (no traducidos) están inmediatamente disponibles para iniciar el acceso; por lo tanto, si los bits no traducidos pueden usarse para seleccionar el conjunto de la cache, para páginas de tamaño 2^n podemos traslapar la selección del conjunto con la traducción y después hacer la búsqueda asociativa de k-vías entre los elementos del conjunto para caches de tamaño k x 2^n. Sin embargo, la factibilidad de este método es limitada, debido a que aun cuando al incrementar la asociatividad entre los conjuntos de la cache se provoca una disminución en la tasa de fallos, se incrementa al mismo tiempo la complejidad del *hardware* e inversamente se afecta el tiempo de acceso. Con esta misma idea, existen otros métodos que se encargan de incrementar el número de bits disponibles antes de la traducción. Por ejemplo, aumentar el tamaño de página; sin embargo, para un gran número de arquitecturas de computadoras el tamaño de página es fijo y agrandarlo requeriría cambios sustanciales tanto en la arquitectura como en el sistema de software, y no obstante que el retardo provocado por la traducción es parcialmente traslapado, no se evita completamente.

Otro método propuesto para obtener más bits disponibles antes de la traducción se basa en la predicción de los bits de dirección adicionales. Un ejemplo de un buen predictor es el contenido del registro base usado para calcular la dirección efectiva. En la etapa donde se genera la dirección, los bits de orden inferior de la dirección de la página se usan para acceder a la tabla de historia para obtener los bits de dirección necesarios, por lo que el contenido del registro base es un acertado predictor de los bits de dirección física necesarios, debido a que el desplazamiento para calcular la dirección efectiva usualmente es muy pequeño y páginas que fueron recientemente usadas tienden a ser referenciadas nuevamente (principio de localidad temporal). El MIPS R6000 contiene un *TLB-slice* que esencialmente es un predictor basado en los bits de orden inferior de una página de dirección virtual.

En vez de usar bits de la dirección virtual para predecir la dirección física, otro método sería usar la dirección virtual para acceder directamente a la cache; esto evita el retardo por la traducción y además, en paralelo con el acceso a cache el TLB puede realizar otras funciones como protección de página, actualización, y otras más.

Las caches de direccionamiento virtual no requieren traducción de dirección durante el acceso, pero el hecho de que múltiples páginas virtuales puedan estar mapeadas con la misma página física, puede complicar enormemente su diseño. Cuando la cache es accedida con una dirección virtual y las páginas están compartidas entre programas que pueden acceder con diferentes direcciones virtuales, existen los sinónimos; éstos se presentan cuando el mismo objeto tiene dos nombres; es decir, existen dos direcciones virtuales para la misma página, lo cual crea un problema ya que una palabra en alguna página puede tener dos posiciones diferentes en la cache, cada una correspondiente a una dirección virtual diferente, permitiendo a un programa escribir el dato sin que el otro programa esté seguro que el dato ha cambiado. El mejor camino para detectar sinónimos es corresponder la dirección virtual requerida con su dirección física y ver si alguna de las direcciones virtuales en la cache se corresponden con la misma dirección física. Para que este método sea posible, se generó el *Reverse Translation Buffer* (RTB), que se accede con la dirección física e indica las líneas de cache que se corresponden con esa dirección física. Así, se mantiene una etiqueta de dirección física para cada una de las líneas presentes en la cache de dirección virtual, entendiéndose que una línea de cache debe ser invalidada si su etiqueta de dirección física es reemplazada.

Otro método sencillo para reducir la complejidad al tratar con sinónimos es tener la seguridad de que los bits de índice usados para seleccionar el conjunto de la cache son los mismos para ambas direcciones, tanto física como virtual. En este caso, el mapeo inverso de la línea de cache se realiza simplemente asociando las etiquetas de dirección física y virtual con cada línea de cache. Sin embargo, el restringido mapeo de páginas puede resultar en más fallos de página. Puesto que los bits de índice son los más importantes para el acceso a cache, un aprovechamiento combinado utilizaría índice virtual con etiquetas físicas. En este caso, los sinónimos son mapeados hacia un pequeño número de conjuntos de entradas, determinados por el número de bits del índice. También, basándose en software, se han propuesto métodos para eliminar los sinónimos.

Una combinación de las técnicas antes mencionadas, así como otras soluciones de hardware, se usan para evadir las restricciones del direccionamiento. Por ejemplo, la UltraSPARK-IIi tiene una cache de datos y de instrucciones de 16KB en el primer nivel de cache. La cache de instrucciones es asociativa de dos vías, físicamente indexada y etiquetada; sin embargo, para tener un acceso rápido, la cache de datos es de mapeo directo con índice virtual y etiqueta física. En el R10000, se tienen caches independientes de datos e instrucciones de primer nivel con una capacidad de 32KB y

asociatividad de dos vías. Ambas caches son virtualmente indexadas y físicamente etiquetadas, y el nivel dos de memoria cache es físicamente indexado y etiquetado con la idea de detectar sinónimos y tratar la coherencia de cache; para soportar un tamaño de página de 4KB, dos bits de dirección virtual son necesarios para indexar el nivel uno de la cache.

3. Metodología de Evaluación

En esta sección se describen los modelos analíticos en que se fundamenta la nueva propuesta; además, se mencionan las herramientas de simulación y la forma en que se usan para llegar a un modelo experimental eficiente.

3.1. Modelos Analíticos

El tiempo de ejecución de un programa está dado por el número total de ciclos de CPU necesitados para ejecutar un programa, N_{Total}, y el tiempo que dura el ciclo de CPU, t_{CPU}. Una forma de comparar la importancia de la variación, aparentemente independiente de los parámetros, es la relación que existe entre ellos para minimizar el tiempo de ciclo, T_{L1}, y como consecuencia el tiempo de ejecución. En realidad, la falta de continuidad entre estas variables se debe al lazo existente entre tiempo de ciclo de la memoria cache y el tiempo de ciclo del CPU. Un cambio en la organización de la memoria cache podría o no afectar el tiempo de ciclo del procesador, dependiendo de los caminos críticos en el diseño.

Claramente, si el tiempo de la cache no está determinado por el tiempo de ciclo del CPU, cualquier cambio en su organización puede mejorar el conteo total de ciclos en forma genérica, proporcionando un nuevo tiempo de ciclo de cache menor o igual que un tiempo de ciclo de CPU sin cambio. Dado que los obstáculos en la cache son claros y poco interesantes cuando el tiempo de ciclo de cache es menor que el tiempo de CPU, podemos consistentemente asumir que el tiempo de ciclo de sistema es determinado por la cache. El tiempo de ciclo de CPU y del sistema son, por consecuencia, una parte mínima del tiempo de ciclo de la cache, y los términos son usados como sinónimos.

Para un sistema de memoria con un nivel de cache, la cuenta total de ciclos es una función de la velocidad de memoria y el porcentaje de error de la cache. El porcentaje de error de una memoria cache **C** se representa por m (**C**). Las características primarias de la organización de la memoria que determinan el porcentaje de error de la cache son el tamaño de la cache C, su nivel de asociatividad A, el número de conjuntos S, y el tamaño de los bloques B. Las opciones de las estrategias de búsqueda y escritura introducen otros factores en la ecuación del porcentaje de error, que pueden despreciarse. En consecuencia, el porcentaje de error está dado como una función de estos cuatro parámetros.

$$m(\mathrm{C}) = f(S, C, A, B) \qquad [1]$$

El tamaño de bloque es el único entre todos los parámetros mencionados, que según su tamaño puede llegar a minimizar el porcentaje de error. Para bloques grandes, el incremento de tamaño afecta en forma negativa al porcentaje de error.

La porción del ciclo total medido atribuible a la memoria principal N_{MM}, depende de las características de la memoria: el tamaño del bloque y el tiempo de ciclo de CPU. Esta última relación resulta del factor de que algunos de los atributos de la memoria son típicamente especificados en segundos. El más significativo de éstos es la latencia (LA) entre el inicio de la búsqueda de la memoria y el inicio de la transferencia de datos solicitados. El número de ciclos desperdiciados por la espera en la memoria principal depende de la latencia expresada en ciclos la, la cual es dada por el valor más alto del porcentaje de latencia expresado en segundos y el tiempo de ciclo. La otra característica dominante de la memoria es el porcentaje en el cual los datos pueden ser transferidos hacia y desde la memoria principal. El porcentaje de transferencia tr en *words* por ciclo y TR en palabras por segundo, es actualmente el más pequeño de los tres porcentajes o tasas: la tasa en la cual la memoria puede presentar un dato, la velocidad en la cual la cache acepta el dato, y la tasa máxima de transferencia determinada por las características físicas y eléctricas del diseño (material y construcción). El tiempo utilizado en transferir un dato en una sencilla búsqueda es denominado período de transferencia. Esto es igual al porcentaje o tasa del tamaño de búsqueda en *words* y la tasa de transferencia.

La función que relaciona los parámetros de la memoria y la cache en un conteo total de ciclos es lineal en el porcentaje de error, latencia y período de transferencia.

$$N_{Total} = g(m(\mathbf{C}), B, LA, TR, t_{CPU}) \qquad [2]$$

La relación entre el tiempo de ciclo y los parámetros de la cache es la más difícil de cuantificar en los niveles más bajos de la jerarquía. La declaración más fuerte que puede ser hecha es que el tiempo de ciclo es invariante, aunque no en un sentido totalmente estricto, en cada uno de los cuatro parámetros básicos: tamaño, asociatividad, número de conjuntos y tamaño de conjunto:

$$t_{CPU} = h(C, S, A, B) \qquad [3]$$

El tiempo de ciclo mínimo es también influenciado por la complejidad de la lógica de control utilizada en las políticas de búsqueda, escritura y reemplazo. La poca predicción de esta función h, ocasiona que se dificulte determinar cuál es la configuración óptima de la cache sin realizar un número de diseños individuales para acercarse a un grado de detalle.
Ahora examinemos la naturaleza de las ecuaciones 1, 2 y 3, para así relacionar el comportamiento de los distintos parámetros del sistema de memoria en sus distintos niveles.

El tiempo total de ejecución es el producto del tiempo de ciclo y el conteo total de ciclos:

$$T_{Total} = t_{CPU} \text{ X } N_{Total} = t_{L1}(\mathbf{C}) \text{ X } N_{Total} \qquad [4]$$

El tiempo de CPU está dado por el tiempo de ciclo de la cache, el cual es una función de sus parámetros organizacionales.

Dado que el tiempo de ejecución es una función convexa de las variables organizacionales y temporales, el tiempo de ejecución mínimo es obtenido cuando la derivada parcial con respecto a algunas variables es igual a cero.

Muchos de los parámetros de la cache no son continuos: están restringidos a valores enteros o binarios. Para estas variables discretas, la ecuación tiene una forma diferente – una que iguala el cambio relativo en el tiempo de ciclo y el conteo de ciclos a través de los cambios de una organización de la cache a una adyacente (esta forma discreta fue usada por Hill (1987) para observar los obstáculos en la asociatividad por conjuntos):

$$\frac{1}{t_{L1}} \text{ x } \frac{\Delta_{L1}}{\Delta C} = -\frac{1}{N_{Total}} \text{ x } \frac{\Delta N_{Total}}{\Delta C} \qquad [5]$$

Dado un cambio particular de una organización u otra, si el lado izquierdo y el lado derecho de la ecuación son iguales, el cambio es de comportamiento neutral; sin embargo, si el lado izquierdo de la ecuación es mayor (cercano a $+\infty$), el cambio se incrementa sobre todo el tiempo de ejecución, y si el lado derecho de la ecuación es más positivo, hay una ganancia neta en rendimiento.

Una estimación comúnmente utilizada para el conteo total de ciclos, $g(m(\mathbf{C}), B, LA, TR, t_{CPU})$, es la suma de ciclos utilizados en la espera en cada uno de los niveles de la jerarquía de memoria. Para un nivel sencillo o un solo nivel de cache, el número total de ciclos está dado por el número de ciclos utilizados en referencias a memoria no medidas ($N_{Execute}$), más el número de ciclos utilizados en realizar instrucciones de búsqueda, carga y almacenamiento, más el tiempo utilizado de espera en la memoria principal.

Para una máquina RISC con un ciclo de ejecución sencillo, el número de ciclos en el cual ninguna referencia de flujo es activa, $N_{Execute}$, es cero. Podemos colapsar todas las instrucciones de búsqueda y carga paralelas en un término común y esto nos deja una forma más simple para el conteo total de ciclos. Luego, para cada lectura hay un ciclo sencillo de máximo valor, además de las penalizaciones por error en la cache por referencias.

$$N_{Total} = N_{Read}\ (1 + m_{Read}(\mathbf{C}) \text{ X } \grave{o}_{MMread}) + N_{Store} \text{ X } \grave{o}_{L1write} \qquad [6]$$

El tiempo para recuperar un bloque de la memoria principal, $\grave{o}_{MMread,}$ tiene dos componentes: el tiempo actual para buscar un bloque y, ocasionalmente, algunos retardos debido a que la memoria está ocupada al momento de la petición. Si no hay prebúsqueda o tráfico del bus de I/O, la única fuente posible para retardo es una escritura en progreso. El promedio de la cantidad de retardos por referencia de memoria es denotada por $\grave{o}_{ReadDelay,}$ El número de ciclos utilizados en la búsqueda de un bloque es la suma de la latencia del período, *la*, y el período de transferencia, *B/tr*. Cuando tratamos con obstáculos que envuelven el tamaño de bloque, es frecuentemente necesario asumir que el promedio de retardo en la memoria es dado por una escritura en progreso, y $\grave{o}_{ReadDelay,}$ es pequeño con respecto a la suma de los períodos de latencia y transferencia. En la

práctica, este es generalmente el caso si la estrategia de escritura es efectiva y los ciclos utilizados en la espera en la memoria principal no dominan el tiempo de ejecución.

$$N_{Total} = N_{Read} \left[1 + m_{Read}(\mathbf{C}) \times (\text{la} + B/tr + \grave{o}_{ReadDelay})\right] + N_{Store} \times \grave{o}_{L1write}$$

$$\approx N_{Read} \left[1 + m_{Read}(\mathbf{C}) \times (\text{la} + B/tr)\right] + N_{Store} \times \grave{o}_{L1write}$$

El número de ciclos para completar una escritura, $\grave{o}_{L1write,}$ incluye el tiempo para realizar la escritura y cualquier ciclo adicional necesario para poner el dato en el *buffer* de escritura. Este último componente también incluye cualquier tiempo de espera para la escribir en el *buffer* cuando está lleno. Para una cache de escritura en línea, el dato escrito siempre requiere ser colocado en el *buffer*, cuando para una cache de postescritura, esta potencial penalización ocurre solamente cuando un bloque no utilizado está siendo retirado. Si una lectura causa el reemplazo de un bloque no válido, se asume que el tiempo necesario para reemplazarlo es ocultado por el tiempo de búsqueda en la memoria, contabilizado en un ciclo de lectura. Para caches de postescritura el número de ciclos para realizar una escritura, $\grave{o}_{L1write,}$ puede ser aproximado por el número de ciclos necesarios para realizar una escritura en la cache, $n_{L1write}$, cuando la frecuencia de postescrituras ocasionada por escrituras es baja.

La base del conteo total de ciclos es el número de instrucciones buscadas.

$$N_{Total} = N_{Ifetch} + N_{Ifetch} \times m(\mathbf{C}_{L1I}) \times \grave{o}_{MMread} + N_{Load} \times m(\mathbf{C}_{L1D}) \times \grave{o}_{MMread} + N_{Store} \times (\grave{o}_{L1write}^{-1}) \qquad [7]$$

La ecuación 6 puede ser usada en lugar de la 7 para esta clase de sistemas si el número de lecturas, $N_{Read,}$ el porcentaje de error de lectura, $m_{Read}(\mathbf{C})$, y el número principal de ciclos por escritura, $\grave{o}_{L1write,}$ son definidos apropiadamente. El número de lecturas se define como el número de instrucciones buscadas, el porcentaje de error de lectura es el porcentaje de error de la cache de instrucciones más el porcentaje de error de la cache de datos soportado por la tasa de número de instrucciones buscadas y cargadas, y el tiempo de escritura es menor que el tiempo para realizar la escritura.

Si el número de ciclos libres de cualquier actividad de memoria, $N_{Execute}$, no es cero como en una máquina CISC, el modelo representado por la ecuación 6 puede aún ser usado sin modificarlo si el número de lecturas y la tasa de de error de lecturas son localmente definibles.

Para encontrar el tiempo de ejecución, el conteo total de ciclos debe de ser multiplicado por el tiempo de duración del ciclo. El tiempo de duración del ciclo está en función de un tiempo de ciclo, t_{L1}, el porcentaje de error, $m(\mathbf{C})$, el tiempo de acceso de lectura de la memoria principal, • $_{MMread}$, el promedio de número de ciclos para retirar una escritura, $\grave{o}_{L1write,}$ y el número de escritura y lecturas.

Cuando tratamos con una jerarquía de memoria cache multinivel, el promedio de tiempo para satisfacer una petición de lectura en cada uno de los niveles es una función del tiempo de acceso de lectura de la cache, tasa de error y penalización por error. Cada penalización de error es exactamente el tiempo más significativo utilizado para satisfacer una lectura en el siguiente nivel en la jerarquía de memoria. Esta recursión se extiende hasta todos los niveles de la jerarquía, donde la penalización por error en la cache está dada por el tiempo de acceso a lectura de la memoria principal. El tiempo total de acceso de lectura, n_{Li}, esta expresado en ciclos de CPU.El porcentaje de error global, M_{Li}, esta definido como el número de lecturas erróneas en el nivel i dividido por el número de referencias de lectura generados por el CPU. El conteo total de ciclos llega a depender de la suma de los productos del porcentaje de error global de cache y el próximo tiempo de ciclo de la cache.

Para investigar el lazo entre los parámetros organizacionales y el tiempo de ejecución, necesitamos investigar la tasa de error como una función dada por los parámetros de la cache, expresada como la función $f(S, C, A, B)$.

El modelo de Agarwal (1987) está basado en la descomposición de los errores en diferentes clases. Un error no estacionario es inevitablemente causado por las referencias a memoria de un programa en su inicio. Intrínsecamente los errores son producidos por hacer referencia a una misma localidad de memoria; si el número de distintas direcciones mapeadas en un conjunto dado es mayor que el tamaño del conjunto, entonces éstas competirán entre sí por el espacio disponible, chocando

entre sí, y ocasionando que queden fuera de la memoria cache. Extrínsecamente los errores son causados por los efectos de la multiprogramación: entre espacios de tiempo, un programa tendrá bloqueados algunos conjuntos mientras que otro requiere hacer uso de los datos en esos bloques.

El porcentaje total de error cubre el t considerando las τ referencias dadas como la suma de tres términos, uno para cada tipo de error: no estacionario, intrínseco y extrínseco.

La tasa de error está definida en términos del número de bloques solamente tocados en el fragmento de referencias, u(B), el tamaño total de la traza, T, el número total de bloques tocados en la traza, U, la tasa de colisión de la traza, c(A), y la probabilidad de que a bloques de un u(B) sean mapeados en el mismo conjunto, $P(a)$. La tasa de colisión está dentro del intervalo de $(0, \tau/u)$.

Asumiendo un reemplazo aleatorio, la probabilidad P(a) está dada por una distribución binomial, la cual puede ser aproximada por una distribución de *Poisson* para cache grandes (una S grande) y un número grande de referencias por evento (una u grande). Finalmente, el porcentaje de error también depende de la fracción de los bloques, $f(t)$, perteneciente a un proceso dado, que son purgados por otros procesos en espacios de tiempo consistentes. Para una cache de mapeo directo, la fracción puede ser aproximada a una simple forma exponencial del número de bloques, u(B), y el número de bloques utilizados por otros procesos en sistemas multiprogramados, u'(B):

$$f(S, B, t) = 1 - e^{-u(B)/S}\,(1 + u'(B)/S)$$

La siguiente sección versará sobre las herramientas de simulación que fueron utilizadas para la evaluación del presente trabajo.

3.2. Herramientas de Simulación

Durante muchos años, la herramienta de simulación fue una herramienta limitada, costosa, tediosa y poco rentable. Aún así, los proyectos de simulación se completaban si el costo de la inversión lo ameritaba. En la actualidad, el costo de simular ha disminuido significativamente por diferentes factores, entre ellos: la facilidad de obtener la información del sistema a simular, el costo del software, el costo del hardware, la rapidez en hacer los modelos, la optimización, la integración entre modelos, entre otros factores.

La metodología que usaremos para evaluar nuestra propuesta se basa principalmente en la simulación. Los arquitectos de computadoras necesitan herramientas que les permitan variar los parámetros de los diseños propuestos hasta obtener el rendimiento esperado en el producto terminal, evitando así la necesidad de construir prototipos que finalmente no funcionen como se esperaba.

Existen varios tipos de simuladores: de cache de datos, de monoprocesador, de multiprocesador, y otros ; de igual forma se cuenta con diferentes técnicas de simulación, por ejemplo: simulación dirigida por traza, dirigida por emulación, dirigida por ejecución, y algunas otras; estas difieren en complejidad de acuerdo con las estadísticas que se requieren de la simulación (de consumo, fallos de cache, de tiempo de ejecución…). En el presente trabajo se adecuaron las herramientas de simulación Atom (Eustace & Srivastava, 1994) y Simplescalar (Burger and Austin, 1997), mismas que se han validado en un gran número de publicaciones de investigadores que se encuentran desarrollando propuestas tendientes a mejorar el rendimiento del sistema de memoria.

3.3. Modelo Experimental

La herramienta Atom permitió una evaluación de la memoria evitando el ruido de factores externos a ésta. Esto no quiere decir que no sea importante validar la propuesta en un entorno general, pero para una primera validación de la propuesta, los resultados del comportamiento específico de la memoria cache de datos fueron de vital importancia con miras a continuar con el proyecto. Posteriormente la herramienta Simplescalar permitió la evaluación de la propuesta incluyendo el resto de la arquitectura.

El Atom se usó para obtener trazas de loads y stores junto con sus direcciones efectivas; con estas trazas se midió el porcentaje de predicción de las instrucciones de referencia a memoria en cargas de trabajo representativas (SPEC'95). El Atom nos permite obtener desde bloques básicos de instrucciones de un programa hasta una sola instrucción de cualquier

tipo con todo y sus registros operandos, por lo que es posible añadir a cada instrucción, el código que nos proporcione estadísticas de comportamiento tales como número de loads y stores; con estos datos y con la misma herramienta se procesaron las trazas de las cargas de trabajo representativas modelando el diseño del esquema propuesto diseñado con el lenguaje de alto nivel "C".

Con el simulador generado se obtuvieron estadísticas de comportamiento tales como número de aciertos y número de instrucciones ejecutadas por programa para distintos algoritmos de reemplazo en capacidades de cache de 8k, 16k y 32k, que son las capacidades típicas de caches de primer nivel en arquitecturas de propósito general. Con esta misma herramienta se simularon esquemas de cache convencional para las mismas capacidades. Con la información obtenida se procedió a calcular la tasa de fallos y el tiempo medio de acceso, usando las fórmulas de la sección 2 y posteriormente se realizaron gráficas de resultados donde se muestra la mejora obtenida entre el esquema propuesto y los esquemas convencionales.

Con la herramienta de simulación Simplescalar se simuló una arquitectura de un procesador superescalar fuera de orden (similar al Alpha 21264) y una cache de primer nivel convencional (mapeo directo y asociativa de dos vías) en capacidades de 8k, 16k y 332k y algoritmo de reemplazo LRU. Los datos obtenidos de esta simulación **manifiestan el mismo comportamiento que los datos obtenidos con la simulación hecha con el Atom.**

Cuando se requiere evaluar un nuevo diseño, se adecua la herramienta de simulación para que simule una arquitectura convencional, y se le aplican cargas de trabajo representativas (*benchmarks*) para medir su rendimiento; posteriormente se adecua la herramienta de simulación para simular el nuevo diseño y se le aplican los mismos *benchmarks.*

Benchmark	Description	Input Set
Go	Artificial Intelligence, game of Go.	Reference
M88ksim	Moto 88K Chip simulator.	Reference
Gcc	New Ver. of GCC, builds SPARC code.	Lamptjp.i
Compress	Compresses and uncompresses file.	Bigtest.in
Li	Lisp interpreter.	Reference
Ijpeg	Graphic compression and decompression	Penguin.ppm
Perl	Manipulate strings and prime numbers (Perl).	Primes.in
vortex	A database program.	Reference
Tomcatv	Is a mesh generation program	Tomcatv.in
Swim	Shallow Water Model 513 x 513 grids.	Swim2.in
Su2cor	Quantum physics. Monte Carlo.	Su2cor.in
hydro2d	Astrophysics. Hydrodynamical equations.	Hydro2d.in
Mgrid	Multi-grid solver in 3D potential field.	Mgrid.in
Applu	Parabolic/elliptic partial differential eqtns.	Applu.in
turb3d	Simulate isotropic turbulence in a cube.	Turb3d.in
apsi	Solve temp., wind, velocity of pollutants.	Apsi.in
fpppp	Quantum chemistry.	Natoms.in
Wave5	Plasma physics. Electromagnetic particle.	Wave5.in

Los resultados de ambas simulaciones, tanto de la arquitectura convencional como de la del nuevo diseño, se comparan para medir el rendimiento obtenido con el nuevo diseño; si los resultados no son alentadores, se plantea otro diseño o se realizan adecuaciones al diseño propuesto. Los *benchmarks* utilizados se presentan en la tabla de abajo; hay 10 *benchmarks* de

enteros y 8 *benchmarks* de punto flotante. Los SPEC'95 de punto flotante y de enteros se usaron como cargas de trabajo representativas.

Los *benchmarks* se compilaron en una computadora basada en el procesador Alpha 21164, con sistema operativo OSF1 V4.0. Para los *benchmarks* de enteros usamos la opción -04 del compilador "cc" propietario. Los *benchmarks* de punto flotante se compilaron con la opción -05 del compilador "fortran" propietario y los resultados se corresponden con una ejecución completa de éstos.

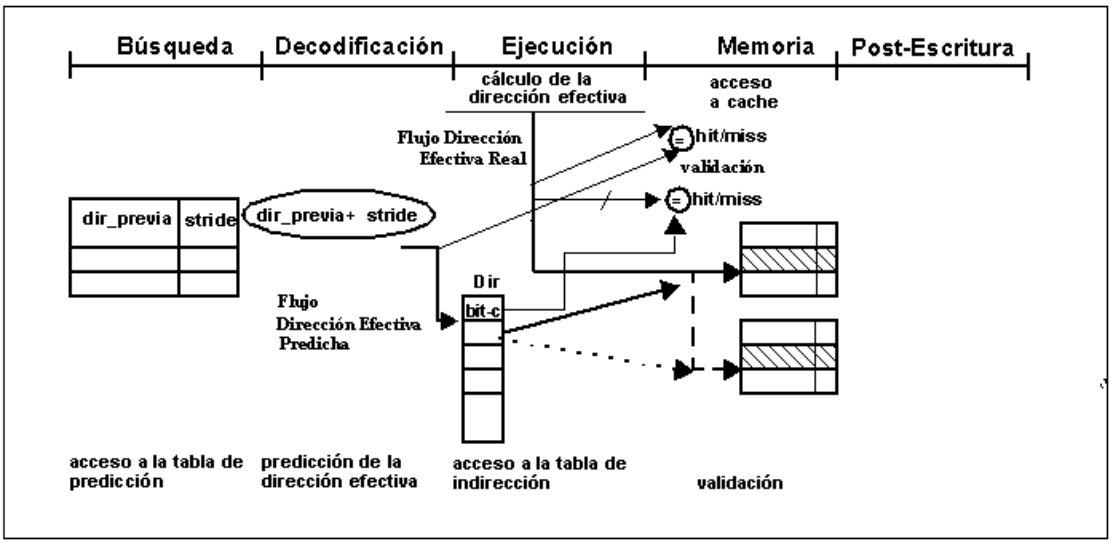
En una etapa previa a la simulación y evaluación de la propuesta, se realizó un análisis de predicción de las referencias a memoria usadas en los SPEC`95, se simuló un autómata de predicción basándose en el stride, y para su evaluación se usaron cargas de trabajo representativas. La dinámica del experimento fue la siguiente: primero se desarrolló el autómata de predicción con el lenguaje de programación "C", éste se simuló con cargas de trabajo representativas y se obtuvieron estadísticas del compor-tamiento de las referencias a memoria (instrucciones de carga y almacenamiento).

4. MDAC

En esta sección se describe la propuesta central de este trabajo que hemos denominado "*Múltiples Disciplinas de Acceso a Cache" (MDAC)*, y se muestran y discuten las disquisiciones experimentales.

4.1. Descripción de MDAC

Con el propósito de contar con un esquema que explote la capacidad de predicción de las referencias a memoria, se propone una cache que trabaja de la siguiente forma: la cache es de acceso secuencial y cuenta con dos posibles ubicaciones para seleccionar o ubicar el bloque (ubicación primaria y alternativa). El tiempo de ciclo del procesador se incrementa si se utiliza la dirección efectiva para acceder al predictor de primera entrada; los accesos a predictor y cache son secuenciales. Por tanto, para soportar el tiempo de acceso al predictor de primera entrada, proponemos predecir las direcciones efectivas con suficiente anticipación a la etapa en que se calculan. En la siguiente figura se muestran los posibles flujos para acceder a los elementos del conjunto. Por un lado, tenemos el flujo de la dirección efectiva real y por otro lado el flujo de la dirección efectiva predicha. La dirección efectiva real se calcula en la etapa de ejecución del procesador. Como se usa el campo índice para acceder a los elementos del conjunto, se dice que el orden de acceso se obtiene estáticamente; la ubicación alternativa del bloque se obtiene invirtiendo el bit de orden superior del campo índice. La dirección efectiva predicha se obtiene con la ayuda del autómata de predicción basándose en el stride.



Después de predecir la dirección efectiva, nos auxiliamos de una tabla de indirección (Dir) para saber a qué elemento del conjunto se debe acceder; en este caso se dice que el orden de acceso se obtiene dinámicamente. Dir tiene un tamaño de 16 veces la capacidad de la cache.

Cuando accedemos a los elementos del conjunto por el flujo de la dirección efectiva real se dice que tenemos una referencia *no_predecible*. Cuando accedemos a los elementos del conjunto por el flujo de la dirección efectiva predicha se dice que tenemos una referencia *predecible*. En ambos flujos se accede al primer bloque (ubicación primaria) y se comparan las etiquetas si las etiquetas no coinciden se accede al segundo bloque (ubicación alternativa), y se tiene un fallo al no coincidir las etiquetas en el segundo acceso. El objetivo es que el bloque referido sea el primero que se busca.

La evaluación se basa en gestionar el esquema con cuatro algoritmos distintos en cuanto a la política de reemplazo. Esta política contempla dos criterios de gestión, el optimista para los dos primeros algoritmos, y el conservador para los siguientes dos algoritmos. La diferencia principal entre los algoritmos optimistas y conservadores, es que en los últimos se agregan bits a cada entrada de la cache con el fin de mantener información temporal entre los elementos de un conjunto y para saber si el bloque se encuentra en su ubicación primaria o alternativa. Cuando un bloque se encuentra en su posición alternativa se dice que está mal colocado. A continuación se describe la forma de trabajo de cada algoritmo:

A.- Optimista 1

Con la idea de que en futuras referencias del bloque éste se acierte en el primer acceso, para el flujo de referencias no predecibles se coloca en la ubicación primaria. Al hacer esto, prácticamente estamos usando la mitad de la cache, ya que se comporta como una cache de mapeo directo.
Para el flujo de referencias predecibles, la tabla de indirección apunta al elemento del conjunto que se debe acceder; en este caso, además de usar toda la cache, ésta se comporta como una cache asociativa de grado dos, por lo que tenemos un conjunto de dos bloques (primario y secundario). Cuando se falla en este flujo, el reemplazo se hace en el bloque secundario, el cual se encuentra invirtiendo el bit (bit-tc) de la tabla de indirección. Es importante notar que si el primer acceso se hace al bloque secundario, en caso de fallo el reemplazo se hará en el bloque primario; es decir, al acceder por el flujo de referencias predecibles no se sabe si se está accediendo al bloque primario o alternativo del flujo de referencias no predecibles. En cada reemplazo se actualiza Dir para procurar que se acierte en el primer acceso en futuras referencias del bloque. Esta forma de trabajo sigue una política de reemplazo LRU.

B.- Optimista 2

En este algoritmo, la gestión de las referencias no predecibles es igual que el algoritmo anterior. En cuanto a las referencias predecibles, el reemplazo se hace en la ubicación alternativa del flujo de referencias no predecibles. En este caso, no se sigue un algoritmo de reemplazo LRU, ya que cuando la tabla de indirección indica acceder a la ubicación alternativa como primera búsqueda, y en caso de fallo, se reemplaza el bloque más recientemente usado.

C.- Conservador 1

Con la idea de mantener en la cache la mayor cantidad de bloques no predecibles, el siguiente algoritmo hace una gestión más conservadora, guardando información sobre la última referencia (predecible/no predecible) hecha a los elementos del conjunto. Con un fallo en una referencia no_predecible, se reemplaza en primera ubicación siempre y cuando en esta ubicación se encuentre un bloque referido previamente como predecible; si no es el caso, se reemplaza el bloque LRU, y cuando tenemos un fallo en referencias predecibles, se reemplaza el LRU actualizando Dir según sea el caso.

D.- Conservador 2

Con este algoritmo, además de tratar de mantener en la cache las referencias no predecibles, también se intenta mantener en la cache los bloques que se encuentran en su posición primaria (bien colocados). Con un fallo en una referencia no_predecible, se reemplaza en primera ubicación siempre y cuando en esta ubicación se encuentre un bloque referenciado previamente como predecible o si en esta ubicación tenemos un bloque mal colocado; si no es el caso, se reemplaza el LRU. Con un fallo en una referencia predecible, se reemplaza el LRU actualizando Dir según sea el caso.

4.2. Disquisiciones Experimentales

Consideramos las tasas de fallos (en referencias predecibles y no predecibles) y de aciertos en el segundo acceso (en referencias predecibles y no predecibles) obtenidas al procesar los *benchmarks* de punto flotante y de enteros con los cuatro algoritmos (A, B, C y D), en capacidades de 8k, 16k y 32k y comparamos los resultados con tasas de fallos obtenidas en caches convencionales de mapeo directo y asociativa de dos vías. Al realizar las disquisiciones experimentales observamos lo siguiente: la distribución de fallos predecibles y no predecibles es proporcional al número de referencias que se tienen de cada caso. En los benchmarks de punto flotante predominan los fallos en referencias predecibles puesto que estos programas tienen mas capacidad de predicción que los benchmarks de enteros; por lo contrario, en los benchmarks de enteros se observa una mejor distribución entre las tasas de fallos predecibles y no predecibles.

En cada algoritmo tenemos dos flujos de referencias, predecibles y no predecibles; en las referencias no predecibles la cache trabaja como mapeo directo por lo que tenemos más fallos por conflicto, y estos fallos crecen en proporción con el número de referencias no predecibles. En cuanto a las referencias predecibles, la cache trabaja como asociativa de dos vías, por lo que se reducen los fallos por conflicto, de igual forma en proporción con el número de referencias predecibles. Para el caso del algoritmo "A", el flujo de referencias predecibles cuenta con una política de reemplazo LRU, por lo que al sumar las tasas de fallos predecibles más no predecibles (tasa total de fallos), el resultado es muy similar a la tasa de fallos de una cache convencional asociativa de dos vías. Lo anterior se cumple sólo en los *benchmarks* de punto flotante, ya que predomina el flujo de referencias predecibles. Para el caso de los enteros se incrementa el flujo de referencias no predecibles y éstas se tratan como mapeo directo, por lo que tenemos un incremento en los fallos por conflicto, que se refleja en la tasa total de fallos.

Para el caso del algoritmo "B", la tasa de fallos no predecible permanece constante para casi todos los *benchmarks*. En cambio, en las referencias predecibles, la tasa de fallos se incrementa debido a que la política de reemplazo no sigue al LRU, ésto provoca un incremento en los fallos por conflicto, que se observa en la tasa total de fallos; por lo tanto, al comparar la tasa total de fallos con las tasas de fallos de las caches convencionales, se observa que para casi todos los *benchmarks* de punto flotante se tiene una tasa total de fallos por encima de la tasa de fallos de una cache de mapeo directo; no siendo así para los *benchmarks* de enteros, donde la tasa total de fallos se sitúa por debajo de la tasa de fallos de una cache de mapeo directo. Nuevamente se observa cómo influye la capacidad de predicción de los *benchmarks* de punto flotante con respecto de los *benchmarks* de enteros.

En el algoritmo "C", al tratar de mantener las referencias no predecibles en la cache, se mejora la tasa de fallos en referencias no predecibles en todos los benchmarks, por lo que al comparar la tasa total de fallos con la tasas de fallos de una cache convencional de dos vías se observa que son casi iguales. De la misma forma se observa una mejora al comparar las tasas de fallos en referencias no predecibles del algoritmo "A" con las tasas de fallos en referencias no predecibles de este algoritmo. En cuanto a la tasa de aciertos en el segundo acceso en referencias no predecibles, notamos que en el algoritmo "C" se incrementa notablemente con respecto a los valores obtenidos en "A". Esto se debe a que la política de reemplazo en el flujo de referencias no predecibles en "C" en algunos casos sigue al LRU, provocando alternancia entre las dos posibles ubicaciones del conjunto. Por último, con el algoritmo "D", se consiguió una tasa total de fallos muy parecida a la tasa de fallos de una cache convencional asociativa de dos vías. Se puede decir que una diferencia entre el algoritmo "C" y "D", se observa en la tasa de aciertos en el segundo acceso en referencias no predecibles, siendo más pequeña en "D" excepto en el *tomcatv* en 32k. Esta diferencia se debe a que la política de reemplazo en el flujo de referencias no predecibles se aproxima aún más al LRU.

Con la finalidad de tener una apreciación más global de la forma de trabajo de cada algoritmo, en dos tablas siguientes se muestran los promedios de las tasas totales de fallos de los cuatro algoritmos y el promedio de las tasa de fallos de las caches convencionales de mapeo directo y asociativa de dos vías.El promedio mostrado en cada algoritmo contempla la suma de las tasas de fallos en referencias predecibles y no predecibles.

Capacidad	Asociativa (dos vías)	Mapeo Directo	Alg. A	Alg. B	Alg. C	Alg. D
8k	20.70	22.81	20.78	22.88	20.76	20.75
16k	12.33	13.99	12.67	14.01	12.38	12.38
32k	7.22	10.10	7.70	10.06	7.26	7.25

Benchmarks de punto flotante

Capacidad	Asociativa (dos vías)	Mapeo Directo	Alg. A	Alg. B	Alg. C	Alg. D
8k	5,79	9.37	6.69	8.15	6.02	6.04
16k	3.46	5.97	4.24	5.06	3.59	3.60
32k	2.36	4.60	2.91	3.42	2.42	2.43

Benchmarks de enteros

Después de calcular el tiempo medio de acceso que se presenta en cada algoritmo y en las caches convencionales de mapeo directo y asociativa de dos vías, se obtuvo el promedio de los resultados obtenidos para cada una de las capacidades de cache analizadas.

Suponemos que la cache asociativa requiere de dos ciclos de reloj para acceder a los elementos del conjunto. El cálculo se realizó para dos valores distintos de tiempo de fallo, 10 y 15 ciclos.

En la tabla siguiente tenemos los promedios del tiempo medio de acceso presentado por los *benchmarks* de punto flotante, y en este caso el cálculo se hizo con un tiempo de fallo igual a 10 ciclos.

Capacidad	Asociativa (dos vías)	Mapeo Directo	Alg. A	Alg. B	Alg. C	Alg. D
8k	3.63	3.05	3.09	3.31	3.10	3.10
16k	2.98	2.25	2.29	2.43	2.27	2.26
32k	2.57	1.90	1.80	2.04	1.77	1.76

Benchmarks de punto flotante

Al comparar el promedio obtenido en la cache convencional de mapeo directo con el promedio de cada algoritmo, tenemos que en el algoritmo "D" se observa una reducción (en ciclos de reloj) del 7.36%, pero sólo en la capacidad de 32k. En cambio, en la tabla donde tenemos los promedios de los *benchmarks* de enteros, con el mismo tiempo de fallo, se observa una reducción generalizada en las tres capacidades de cada algoritmo, excepto el algoritmo "B" en capacidades de 8k y 16k.

Capacidad	Asociativa (dos vías)	Mapeo Directo	Alg. A	Alg. B	Alg. C	Alg. D
8k	2.46	1.84	1.70	1.86	1.71	1.67
16k	2.27	1.53	1.46	1.55	1.49	1.44
32k	2.18	1.41	1.32	1.40	1.36	1.31

Benchmarks de enteros

En el caso de los *benchmarks* de enteros, también se observa que el algoritmo "D" presenta el mejor promedio de tiempo medio de acceso, con una diferencia del 9% en 8k, 5% en 16k y 7% en 32K con respecto al promedio obtenido por una cache convencional de mapeo directo. Esto se debe principalmente a que los *benchmarks* de enteros presentan tasas de fallos más pequeñas que los *benchmarks* de punto flotante, lo cual favorece los resultados en enteros.

Es importante notar que nuestro esquema es mejor que una cache convencional de mapeo directo, en cuanto a tiempo medio de acceso se refiere en el algoritmo D, pero si la comparamos con una cache convencional asociativa de dos vías, la mejora es **sorprendente**; tal es el caso del 39.9% en capacidad de 32k, en el promedio de los *bechmarks* de enteros. Experimentos adicionales al aumenta el tiempo de fallo mostraron que el promedio se mejoró en los cuatro algoritmos de ambos tipos de *benchmarks.* Este comportamiento nos muestra que **el beneficio que presenta nuestro esquema es directamente proporcional al tiempo de fallo e inversamente proporcional a la tasa de fallos.**

5. Conclusiones y Propuestas de Trabajos Futuros

Se realizó un estudio sobre la capacidad de predicción de las referencias a memoria que se generan en cargas de trabajo representativas, utilizando un autómata de predicción basándose en el stride y un contador saturado de dos bits. Se obtuvieron resultados que muestran cómo las referencias a memoria son altamente predecibles. Lo anterior nos motivó a proponer nuevos esquemas de gestión de memoria cache basados en la predicción de la dirección efectiva, para mejorar el rendimiento del sistema de memoria, en lo que respecta a la reducción de tasa de fallos y tiempo medio de acceso.

Se propuso un esquema que explota la capacidad de predicción de las referencias a memoria, el esquema MDAC (Múltiples Disciplinas de Acceso a Cache). Se hicieron experimentos con 4 algoritmos de gestión, observándose que este esquema mejora en promedio, el tiempo medio de acceso hasta en un 12% con respecto al tiempo medio de acceso presentado por una cache convencional de mapeo directo. Es posible que para el caso de las referencias predecibles se pueda ampliar el nivel de asociatividad, puesto que contamos con tiempo suficiente para acceder a la cache sin alterar el ciclo de reloj. Para el caso de las referencias no_predecibles es necesario proponer un esquema más complejo para evitar que se altere la ruta crítica del procesador.
En este trabajo nos enfocamos en mejorar el rendimiento del sistema de memoria sin considerar el consumo de potencia que presenta esta mejora. Puesto que a finales de la última década la reducción del consumo de potencia en procesadores de propósito general viene a ser un compromiso de diseño (González, R. & M. Horowitz, 1996; Folegnani, D. & A. González, 2001) y dado que los accesos a cache consumen entre el 30 y 60% del total de energía consumida por el procesador, como trabajo futuro proponemos esquemas de gestión de memoria que mantengan el óptimo equilibrio *rendimiento-bajo consumo*, usando como base la experiencia adquirida en este trabajo.

Agradecimientos.- Los autores agradecen el apoyo de las siguientes instituciones: Banco de México, Secretaría Académica y COFAA del Instituto Politécnico Nacional y Sistema Nacional de Investigadores.

6. Referencias

1) **Calder, B.**, D.Grunwald and J. Emer. “Predictive Sequential Associative Cache”, Proc. 2nd HPCA, 1996.
2) **Agarwal, A.**, *Analysis of Cache Performance for Operating Systems and Multiprogramming.* PhD. Thesis, Stanford University, May, 1987. Available as Technical Report CSL- TR-87-332.
3) **Eustace, A.** and A. Srivastava. “ATOM: A Flexible Interface for Building High Performance Program Analysis Tools”. Technical Report TN-44, Digital Western Research Laboratory, 1994.
4) **Burger, D**. and T. M. Austin, "The SimpleScalar Tool Set, Version 2.0," University of Wisconsin-Madison Computer Sciences Department, Technical Report #1342, June, 1997.
5) **González, R.** and M. Horowitz. “Energy dissipation in general purpose microprocessors”. *IEEE Journal of Sold State Circuits*, 31(9):1277-1284, September 1996
6) **Folegnani, D.** and A. González. "Energy-Effective Issue Logic”, Proceedings of the 28th Int. Symposium on Computer Architecture, Goteborg (Swwden), 2001.
7) **Amrutur, B.** and M. Horowitz. “Techniques to reduce power in fast wide memories”. In Symposium on Low Power Electronics, vol. 1, pp 92-93, October 1994.
8) **VijayKrishnan, N.**, M. Kandemir, M. J. Irwin, H.S Kim, and W. Ye. “Energy-driven integrated hardware-software optimization using SimplePower”. In ISCA-27. Vancouver, Canada, June 2000.
9) **Albera, G.**, I. Bahar and S. Manne "Power and Performance trade-offs using various caching strategies”, Proc. of the ISLPED, 1998.
10) **Villa, L.**, M. Zhang and K. Asanovic, "Dynamic Zero Compression for Cache Energy Reduction”, 33rd International Symposium on Microarchitecture, Monterey, CA, December 2000.

Dr. Oscar Camacho Nieto. *Nacionalidad Mexicana. Grados de Maestro en Ciencias (1995) en ingeniería de cómputo con especialidad en sistemas digitales y doctor en ciencias de la computación (2003), obtenidos en el CINTEC – IPN (ahora CIDETEC) y el CIC - IPN respectivamente. Suficiencia de investigador otorgada por el Departamento de Arquitectura de Computadoras (1999) de la Universidad Politécnica de Cataluña, Barcelona-España. Actualmente se desempeña como profesor investigador adscrito al laboratorio de Sistemas Digitales del Centro de Investigación en Computación del Instituto Politécnico Nacional. Áreas de Interés: Arquitectura de Computadoras, Sistemas Digitales, Procesamiento Grafico y Análisis de Imágenes.* oscarc@cic.ipn.mx

Dr. Luis Alfonso Villa Vargas. *Nacionalidad Mexicana. Originario de los Mochis, Sinaloa, México. Ingreso al SIN en el año 2000 como investigador nivel "C". Realizó un postdoctorado en Arquitectura de Computadoras en el Instituto Tecnológico de Massachussets (MIT) el cual concluyó en enero de 2001. En 1999 obtuvo el grado de Doctor en Informática en la Universidad Politécnica de Cataluña en Barcelona, España. Actualmente forma parte del grupo de investigadores del PIAME del Instituto Mexicano de Petróleo. Áreas de Interés: Su línea de investigación está enfocada en el estudio de la Arquitectura de Microprocesadores de alto rendimiento y está particularmente interesado en la evaluación de nuevas técnicas de arquitecturas que permitan reducir el consumo de energía de los microprocesadores. lvilla@imp.mx*

Dr. Juan Luis Díaz de León Santiago. *Nacionalidad Mexicana. Grados de maestro en ciencias (1993) en control automático y de doctor en ciencias en morfología matemática (1996), obtenidos en el CINVESTAV-IPN. Investigador invitado (1996) y profesor visitante (1997) de la Universidad de Florida. Director de Tecnología (2000-2001) del Sistema Nacional de Seguridad Pública. Actualmente, es Director e investigador titular C del CIC-IPN y Director Fundador del Grupo de Robótica y Análisis de Imágenes (GRAI). Miembro del Sistema Nacional de Investigadores y Premio Nacional de Investigación y Desarrollo Tecnológico de Excelencia Luis Enrique Erro. Nombramiento de asociado honorario de la Asociación Boliviana para el Avance de la Ciencia (ABAC), perteneciente a la Academia Nacional de Ciencias de Bolivia. Otorgamiento oficial de testimonio de bienvenida y entrega de las llaves de la Ciudad de la Provincia de Cercado Cochabamba, Bolivia por su trayectoria profesional. Distinguido por el Departamento de Cochabamba, Bolivia, con el Reconocimiento "Honor al Mérito" por sus aportes a la investigación, en mayo de 2003. Presidente Fundador de la Fundación Ciencia y Tecnología de las Américas (CyT-Américas). Áreas de Interés: Morfología Matemática, Análisis de Imágenes, Redes Neuronales Morfológicas, Teoría de Control y Robótica Móvil. jdiaz@cic.ipn.mx*

Dr. Cornelio Yáñez Márquez. *Nacionalidad Mexicana. Licenciado en Física y Matemáticas (1989) por la ESFM-IPN. Grados de maestro en ciencias en ingeniería de cómputo (1995) y de doctor en ciencias de la computación (2002), obtenidos en el CIC-IPN. Actualmente, es profesor investigador titular C del CIC-IPN y miembro fundador del Grupo de Robótica y Análisis de Imágenes. Mención honorífica en el examen de grado de doctorado y Presea Lázaro Cárdenas 2002, recibida de manos del C. Presidente de la República. Miembro del Sistema Nacional de Investigadores. Nombramiento de asociado honorario de la Asociación Boliviana para el Avance de la Ciencia (ABAC), perteneciente a la Academia Nacional de Ciencias de Bolivia. Otorgamiento oficial de testimonio de bienvenida y entrega de las llaves de la Ciudad de la Provincia de Cercado Cochabamba, Bolivia por su trayectoria profesional, en mayo de 2003. Secretario Académico Fundador de la Fundación Ciencia y Tecnología de las Américas (CyT-Américas). Áreas de Interés: Memorias Asociativas, Redes Neuronales, Morfología Matemática, Análisis de Imágenes y Robótica Móvil. cyanez@cic.ipn.mx*