Análisis dinámico de bloques iterativos

Marcos R. de Alba, David R. Kaeli
Dept. of Electrical and Computer Engineering
Northeastern University
Boston, MA, USA
mdealba,kaeli@ece.neu.edu

Eun-Sung Kim
Div. of Information Technology Engineering
Soonchunhyang University
Asan, Chongham, Korea
eskim@sch.ac.kr

22 de julio de 2001

Resumen

En las últimas dos décadas se han propuesto diversas técnicas para mejorar el desempeño de microprocesadores. La técnica de predicción de saltos y bifurcaciones ha sido una de las más estudiadas. Su importancia radica en proveer anticipadamente la dirección de la siguiente instrucción que debe ser leída de memoria. Sin embargo, a pesar de que dicha técnica provee ahorro de ciclos de reloj, aún se puede obtener mayor ganancia si se cargan instrucciones a las unidades de ejecución mientras se espera por instrucciones provenientes de memoria. Una forma de extraer tal ganancia es aprovechando la localidad existente en bloques iterativos. Un análisis extenso de bloques iterativos en tiempo de ejecución proporciona la capacidad de detectar sus patrones de comportamiento. La información contenida en los patrones, establece los fundamentos para diseñar un predictor de ciclos eficiente que ayude a mejorar el uso de la técnica de hardware *loop unrolling*. En este trabajo se desarrolla un análisis dinámico de ciclos recolectando y procesando la información existente en los patrones. A partir de ésta se generan resultados importantes que sirven como base de los parámetros de diseño de un predictor de ciclos eficiente.

1 Introducción

La técnica de predicción de saltos o de predicción de bifurcaciones, conocida en inglés como *branh prediction*, ha sido estudiada desde el principio de la década de los 80 [1, 2]. La finalidad de dicha técnica consiste en predecir la siguiente instrucción a ser ejecutada por el procesador. Cuando la predicción es correcta se ahorra el tiempo que cuesta determinar la siguiente instrucción; cuando la predicción es incorrecta, se agrega un tiempo de penalización al tiempo de determinación de la instrucción correcta y además se debe reestablecer el flujo correcto del programa. Afortunadamente, los niveles de precisión alcanzados por predictores actuales superan el 95 porciento [3]-[7].

Otra forma de reducir el tiempo total de ejecución de un programa es incrementando la utilización de las unidades de ejecución. Para lograr tal objetivo, la etapa de búsqueda de instrucciones, $fetch$, debe maximizar el número de instrucciones que libera a las unidades de procesamiento. Sin embargo, el proceso de solicitud y recepción de instrucciones entre la unidad $fetch$ y memoria toma determinados ciclos de reloj que limitan la velocidad con la cual pueden ser llenadas las unidades procesadoras. Para reducir el tiempo de ocio de las unidades de procesamiento, se debe explotar al máximo el paralelismo a nivel instrucciones existente en los bloques iterativos.

Los bloques iterativos de instrucciones o ciclos, conocidos en inglés como *loops*, incluyen un alto nivel de paralelismo. Diferentes iteraciones de un bloque iterativo pueden ser ejecutadas simultáneamente, técnica conocida en inglés como *loop unrolling*. Dicha técnica es empleada altamente en compiladores modernos y ha demostrado mejorar el desempeño de microprocesadores, en [8] son presentadas diversas técnicas de compilación que explotan la localidad de los datos mejorando la forma en la que se accesan las líneas de la memoria cache. Sin embargo, a pesar de que dichas metodologías reducen el tiempo total de ejecución, cuando el programa se encuentra ejecutando muchos datos cambian, situación imposible de detectar en tiempo de compilación.

Haciendo un análisis dinámico del comportamiento de los bloques de iteración se puede detectar con precisión cuales son los registros, localidades de memoria, e instrucciones accesadas por cada ciclo. De tal forma que si esta información es utilizada en tiempo de ejecución, se pueden predecir diversas características como: el número de iteraciones por ejecución y el número de instrucciones por iteración. Tales valores, ayudan a establecer una mejor utilización de los recursos del microprocesador, además de facilitar el incremento del paralelismo a nivel instrucciones. Aunado a esto, el contar con dicha información permite optimizar en hardware la aplicación de *loop unrolling*. Una gran ventaja de aplicar tal método en hardware es que puede determinarse de forma rápida si conviene o no desenrollar un bloque iterativo y si así fuese, cuantas veces hay que hacerlo. Tal ventaja permite también la optimización en el llenado de la ventana de instrucciones y la reducción del tiempo en el cual las unidades de procesamiento o ejecución permanecen inútiles en espera de instrucciones.

2 Trabajo previo

La técnica de predicción de saltos puede ser clasificada en dos grandes grupos. El primero, conocido como predicción estática de saltos [1, 2], consiste en predecir en tiempo de compilación que instrucciones de control (saltos y bifurcaciones condicionales) brincarán a la dirección destino. Este grupo fue el originalmente utilizado, sus niveles de precisión oscilaban entre 60 y 70 porciento dependiendo de la aplicación. Hoy en día, dicho método es utilizado en ciertas arquitecturas, una de ellas es [9]. En tal arquitectura, el compilador asigna ciertas claves (*hints*) como parte de la instrucción. En tiempo de ejecución el procesador observa las claves y, basado en ellas, determina la siguiente instrucción. Los niveles de precisión alcanzados por esta técnica alcanzan el 85 porciento. El segundo grupo, consiste en predecir de forma dinámica cual es la dirección de la siguiente instrucción. Para ello, el procesador cuenta con hardware específico, conocido en inglés como *branch predictor*, que mantiene historia de la ocurrencia de las instrucciones de control, *branches*. La historia de cada instrucción de bifurcación se almacena en una celda y a partir de ella se establece su predicción. Varias técnicas han sido propuestas, destacándose el predictor de bifurcaciones de dos niveles [4] que almacena información no sólo de la historia de ocurrencias individuales, sino también de ocurrencias en forma global. De tal forma que el predictor permite la detección de patrones de ocurrencia para un número determinado de instrucciones de bifurcación. Otro método importante dentro de este grupo, consiste en la predicción de saltos indirectos. La dificultad en predecir tal tipo de saltos radica en que no sólo se debe predecir si el salto brincará o no a la dirección destino, sino también en determinar cual es el valor de la dirección destino [7, 10, 11, 12]. En [7] se demuestra que la mayoría de los saltos indirectos normalmente saltan arbitrariamente a una de dos direcciones destino. Sin embargo, existe un número considerable de saltos indirectos que brincan a más de dos direcciones distintas, lo cual limita enormemente las posibilidades de una predicción exacta.

La combinación de *branch prediction* con la explotación del paralelismo disponible en los bloques iterativos representa una forma más agresiva para obtener beneficios en el desempeño.

La detección de bloques iterativos ha sido estudiada en [13, 14] con el objeto de mejorar la ejecución especulativa en procesadores multihilos. En [13], se propone un método efectivo para la detección dinámica de ciclos y también se presenta una política de control de especulación basada en el comportamiento repetitivo de los bloques iterativos. En [14] se ejecutan diferentes iteraciones de un mismo bloque iterativo (independientes y/o dependientes) en diferentes hilos con el fin de incrementar el ancho de banda de la unidad de *fetch*.

En [15], se propone el uso de dos técnicas para mejorar la precisión de las predicciones de los predictores de saltos. Una de ellas es *terminaci'on* de *bloques iterativos*, en inglés *loop termination*, su objetivo es detectar instrucciones de bifurcación que están relacionadas con bloques iterativos. La forma de detectar tal relación consiste en identificar dinámicamente secuencias de occurrencias; por ejemplo, si la secuencia de ocurrencias de una bifurcación es *tomada*, $no - tomada$, *tomada* (donde, *tomada* representa que la instrucción de bifuración saltó a la dirección destino y $no - tomada$ indica que la instrucción de bifurcación no saltó a la dirección destino), 101 es la historia almacenada que representa tales eventos. Por lo tanto, usando *loop termination* es posible detectar bifurcaciones de bloques iterativos que no pueden ser captadas con un predictor de dos niveles. La otra técnica propuesta en [15] propone dividir bloques iterativos con un gran número de iteraciones en uno o más bloques iterativos de menor tamaño. El objeto de tal división es poder registrar la historia de cualquier bloque iterativo con el hardware disponible. Por ejemplo, tal técnica permite que un ciclo con un millón de iteraciones sea dividido en dos ciclos, cada uno con mil iteraciones; en tal situación, el ciclo original requeriría al menos una tabla con una altura de 20 bits para poder almacenar la historia de las ocurrencias, si la división es aplicada, el tamaño es reducido a 10 bits. Combinando ambas técnicas con predicción de bifurcaciones los autores obtuvieron altos niveles de predicción para bloques iterativos.

En [8], se presentan diversas técnicas de compilación para optimizar código en presencia de bloques iterativos. Se propone un modelo para mejorar el acceso de las líneas de la memoria cache por medio del cálculo de localidad temporal y espacial en tiempo de compilación. En dicho trabajo se evalúan diferentes núcleos de programas científicos y se demuestran mejoramientos en el desempenó para diferentes arquitecturas.

La originalidad de este trabajo radica en la determinación dinámica de patrones de comportamiento en bloques iterativos. Dichos patrones incluyen información específica del número de veces que cada loop es ejecutado, así como de cuantas y cuales fueron las iteraciones en cada ejecución y cuantas y cuales fueron el número de instrucciones por cada iteración. Para el diseño de un predictor de bloques iterativos eficiente es necesario recolectar la mayor cantidad de información posible. Una vez que la información es recolectada, entonces se obtienen las medias aritmética y ponderada de diversas características, detalladas en la sección de experimentación y resultados.

3 Metodología

En este trabajo se realiza la instrumentación de diversas aplicaciones del estándar SPEC 2000 para recolectar patrones de comportamiento en bloques iterativos. La obtención de dichos patrones demuestra la gran localidad de datos y temporal que existe en dichas estructuras.

Para detectar los bloques iterativos se sigue la metodología propuesta en [13] con la diferencia de que aquí se recolecta toda la información existente en tiempo de ejecución. Tal que es posible detectar exactamente como un *loop* se comporta durante todas y cada una de sus ejecuciones

a lo largo del programa. Asimismo, se determina con precisión cuantas veces se repiten sus iteraciones y el número de instrucciones en cada una de ellas. El contar con tal fuente, permite determinar que bloques iterativos se comportan de forma estable y cuales varian notablemente. El uso de dicha información permitirá acentar los parámetros de disño de un predictor de ciclos que proveerá ganancia en tiempo de ejecución. La cual será producida por la utilización eficiente de las unidades de procesamiento con instrucciones provenientes de diversas iteraciones de bloques iterativos.

En la siguiente sección se describen los experimentos realizados y se presentan resultados con su respectivos análisis.

4 Experimentación y resultados

4.1 Experimentación y recolección de información de bloques iterativos

Para recolectar la información de los bloques iterativos se escribieron rutinas de análisis e intrumentación para la herramienta ATOM [16]. Los experimentos fueron llevados a cabo en una computadora Alpha 21264 con dos procesadores. En la parte de instrumentación se identifican de forma estática los diferentes grupos de instrucciones de bifurcación y saltos y se calculan las direcciones destino. En la parte de análisis se lleva el control y cuenta de cada una de las instrucciones de bifurcación y salto, distinguiéndo aquellas relacionadas con bloques iterativos. En el conjunto de experimentos se tomaron en cuenta todos y cada uno de los bloques iterativos. Se observó que existe un gran número de bloques iterativos con sólo una ejecución y una iteración. La información recolectada en cada experimento incluye lo siguiente:

- Análisis y caracterización estática de instrucciones de bifurcación.
- Tamaño promedio de bloques básicos.
- Porcentaje de instrucciones que cambian el control de flujo del programa respecto al total.
- Caracterización dinámica de instrucciones de bifurcación, llamadas a subrutina y retornos.
- Patrones de ejecuciones, iteraciones e instrucciones.
- Número de ejecuciones para cada bloque iterativo.
- Media aritmética, media ponderada, varianza y desviación estándar del número de iteraciones.
- Media aritmética, media ponderada, varianza y desviación estándar del número de instrucciones ejecutadas por iteración.
- Media aritmética, media ponderada, varianza y desviación estándar de la probabilidad de que un bloque iterativo itere más de una vez.
- Media aritmética, media ponderada, varianza y desviación estándar de la probabilidad de que un bloque ejecute el mismo número de instrucciones en iteraciones contiguas.

El análisis estático incluye el conteo de cada tipo de instrucción de control por medio de las rutinas de instrumentación. Se identifica cuantas instrucciones de bifurcación condicional existen en cada aplicación, cuantas de ellas tienen una dirección destino con desplazamiento positivo y cuantas con

desplazamiento negativo. También se cuenta cuantas instrucciones son saltos, dividiendo entre aquellos con desplazamiento positivo y aquellos con negativo. Se cuenta además el número de llamadas a subrutina debidas a instrucciones de salto y a instrucciones de bifurcación. Por último se cuenta el número de instrucciones de retorno. La recopilación de esta información es importante para establecer parámetros de relación entre el código fuente de la aplicación y su comportamiento en tiempo de ejecución.

El tamaño de los bloques básicos refleja la densidad del número de instrucciones de bifurcación en una aplicación. Además, se considera como un indicador de la cantidad de paralelismo existente en una aplicación dada. Mientras más grande sea el tamaño de los bloques básicos, mayor es la probabilidad de extraer paralelismo en ella. Mientras más instrucciones de bifurcación estén presentes en un programa, mayor es el número de dependencias de control y mayor es la dificultad para extraer el paralelismo en él.

El porcentaje de instrucciones de control respecto al total está inversamente relacionado con el tamaño de los bloques básicos y viceversa. Mientras mayor sea el número de instrucciones de control menor es el de los bloques básicos.

La caracterización dinámica de instrucciones de bifurcación, llamadas a subrutina y retornos es importante para saber con precisión de que tamaño deben de ser las tablas de los predictores y para determinar como deben ser interconectados los predictores de instrucciones de salto y bifurcaciones con los detectores de bloques iterativos.

Los patrones de ejecuciones, iteraciones e instrucciones por iteración se obtienen para determinar de forma precisa como el conjunto de bloques iterativos pertenecientes a un programa se comporta. Estos patrones permiten reconocer como en algunas aplicaciones como 176.*gcc* existen ciclos con un número pequeño de ejecuciones, iteraciones, e instrucciones por iteración que se repiten una y otra vez. También, tales patrones permiten identificar la existencia de ciclos que contienen un gran número de ejecuciones, en las cuales el número de iteraciones y el número de instrucciones por iteración cambia en cada ejecución. De igual manera, se encontró que un número de bloques iterativos contiene patrones distintos en cuanto a las iteraciones, pero similares en cuanto al número de instrucciones por iteración. Estos patrones representan un mapa del comportamiento de los bloques iterativos de cualquier programa.

El número de ejecuciones de cada bloque iterativo es obtenido para saber que ciclos ocurren con mayor frecuencia, además de ayudar a detectar correlaciones entre diferentes ciclos que se encuentre anidados. Como ejemplo, si un bloque iterativo se ejecuta 10 veces e itera 5 en cada ejecución y dentro de él incluye a otro bloque iterativo; entonces el bloque interior se ejecutará 50 veces y cada una de ellas tendrá un número dado de iteraciones, que puede ser fijo o variable dependiendo de las condiciones del control de ese ciclo.

Las medidas estadísticas del número de ejecuciones, iteraciones, e instrucciones ejecutadas por iteración sirven para determinar la cantidad de recursos que deben ser asignados a cada ciclo. De igual forma, sirven para establecer los parámetros de diseño de un predictor de ciclos y para aplicar de manera óptima la técnica de *loop unrolling*. Si dichas estadísticas son conocidas con precisión en tiempo de ejecución, entonces es posible determinar el número de veces que un ciclo determinado iterará y el número de instrucciones que ejecutará. Con estos datos, es posible cargar instrucciones pertenecientes a diferentes iteraciones de un mismo bloque iterativo en la ventana de instrucciones y enviarlas a las unidades de procesamiento. Tal situación beneficia el desempeño del procesador debido a que se reduce el tiempo que permanecen inútiles las unidades de procesamiento cuando la unidad de $fetch$ se encuentra esperando por instrucciones provenientes de la memoria.

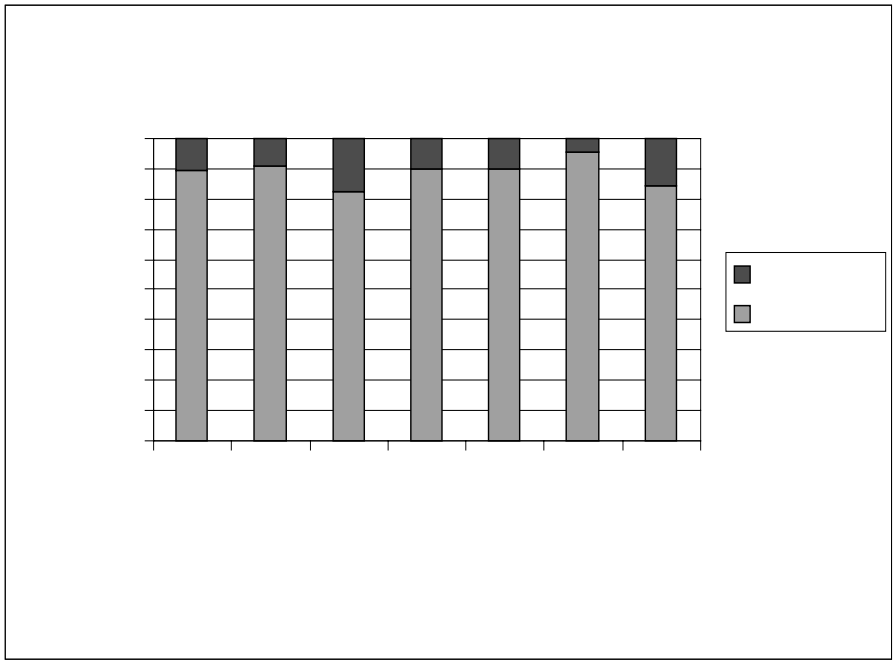

Figura 1: Grupos de instrucciones.

4.2 Resultados

En la figura 1 se muestran los porcentajes de instrucciones que cambian el control de flujo del programa, *totcontrol*, y aquellas que no lo hacen *nonbranch*. Es decir, se muestran las proporciones de saltos y bifurcaciones en relación a instrucciones que no realizan ningún tipo de cambio del flujo de control. Se analizaron 6 aplicaciones del estándar SPECINT2000 y una del SPECINT95. En tal figura se aprecia como las instrucciones de salto y bifurcación varían entre un 4.36 y un 17.45 porciento. De estos resultados se espera que el número de bloques iterativos sea mayor en proporción para las aplicaciones con mayor número de saltos y bifurcaciones, en inglés *branches*.

En la figura 2 se demuestra la distribución de las instrucciones de salto y bifurcación para cada una de las aplicaciones. Tales instrucciones son clasificadas de acuerdo a los siguientes tipos:

- Retornos de subrutina, *ret*
- Llamadas a subrutina debido a bifurcaciones, *bsrcall*
- Llamadas a subrutina debido a saltos, *jsrcall*
- Saltos con desplazamiento negativo, *bwdjmp*
- Saltos con desplazamiento positivo, $fwdjmp$
- Bifurcaciones no condicionales con desplazamiento negativo, *bwduncond*
- Bifurcaciones no condicionales con desplazamiento positivo, $fwduncond$
- Bifurcaciones condicionales con desplazamiento negativo, *bwdcond*
- Bifurcaciones condicionales con desplazamiento positivo, $fwdcond$

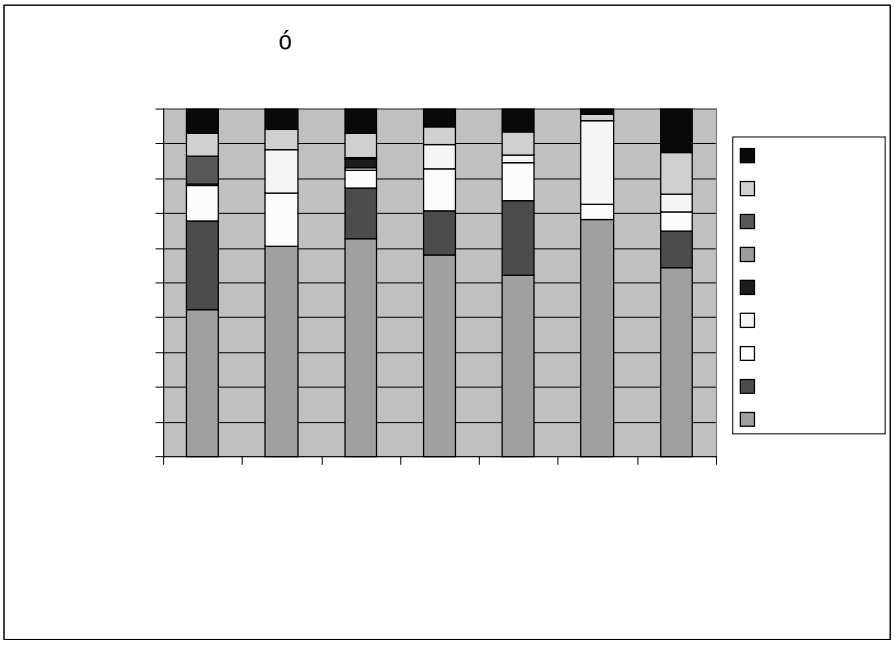

Figura 2: Distribución de instrucciones.

En la figura se aprecia como las bifurcaciones condicionales con desplazamiento positivo oscilan entre el 40 y 60 porciento del total. También se puede observar como las bifurcaciones con desplazamiento negativo (condicionales y no condicionales) varian entre 25 y 40 porciento. Este último grupo es de interés en este trabajo porque en este grupo se encuentran los bloques iterativos.

En la figura 3 se presenta el número de instrucciones ejecutadas para cada una de las aplicaciones. Como se puede observar, la cantidad de información a ser analizada es muy variable dependiendo del estándar en turno. Aquí se ve como la aplicación 300.$twolf$ ejecuta 260 millones de instrucciones, en cambio 256.$bzip2$ ejecuta 60,000 millones de instrucciones.

En la figura 4 se indica el número de bloques iterativos distintos que fueron detectados en cada aplicación. Como se comentó en párrafos anteriores, las aplicaciones con mayor porcentaje de instrucciones *branch* (saltos y bifurcaciones) tienen el mayor número de ciclos. Esto puede observarse claramente tomando como referencia la *benchmark* 176.gcc, la cual cuenta con un 17.45 porciento de instrucciones *branch* y demuestra 1,416 ciclos distintos.

Respecto a la proporción de ciclos que se deben a saltos y bifurcaciones, la figura 5 demuestra tal información. En ésta se aprecia una gran diversidad en cuanto a los origenes de los ciclos, es importante hacer notar como aplicaciones que realizan tareas similares (tal es el caso de 164.$gzip$, $gzip$(95) y 256.$bzip2$) presentan distribuciones completamente diferentes. Esto se debe a que el código fuente de cada aplicación es muy diverso, en especial, los ciclos utilizados en una y otra son distintos. Para la aplicación con un mayor número de bifurcaciones condicionales el número de ciclos de tipo *while* es sobresaliente; por el contrario, para la aplicación con un mayor número de bifurcaciones no condicionales el tipo de bloque iterativo con mayor frecuencia es de tipo for. En ciclos de tipo *while* una bifurcación condicional es evaluada al principio del bloque y si la condición se cumple el ciclo repite. En el caso de un ciclo for, existe una bifurcación condicional al inicio del ciclo y otra no condicional al final que hace brincar de regreso a la evaluación de la condición. Al recopilar los patrones de comportamiento de los bloques iterativos se observó que con mayor frecuencia los ciclos repiten sus iteraciones e instrucciones por iteración cuando su nivel

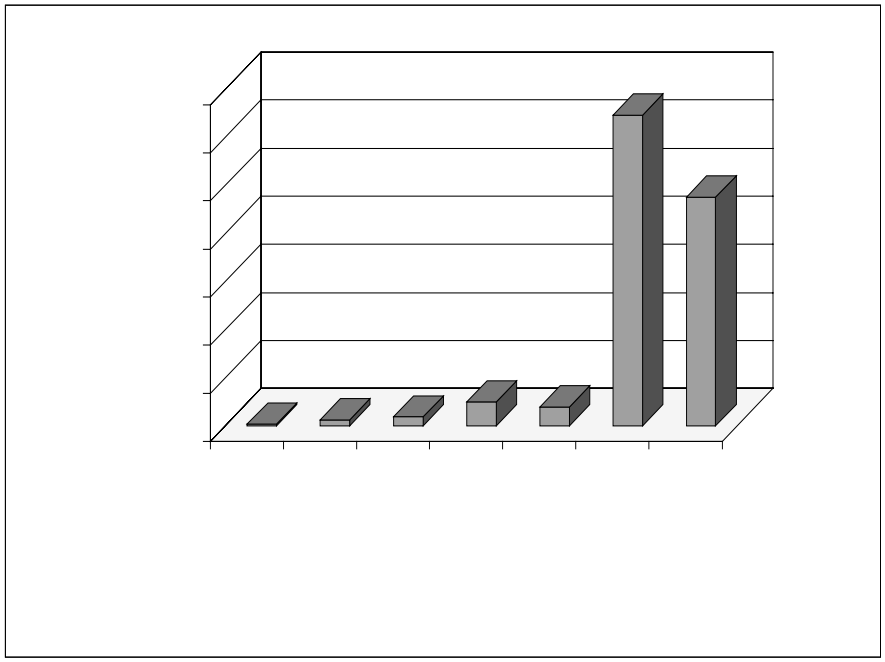

Figura 3: Instrucciones dinámicas totales

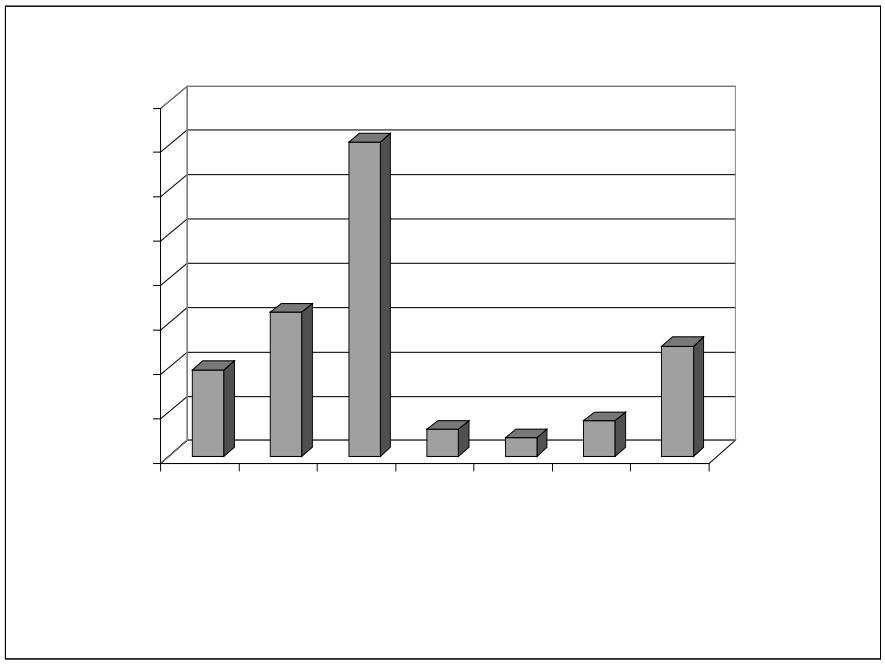

Figura 4: Número de bloques iterativos

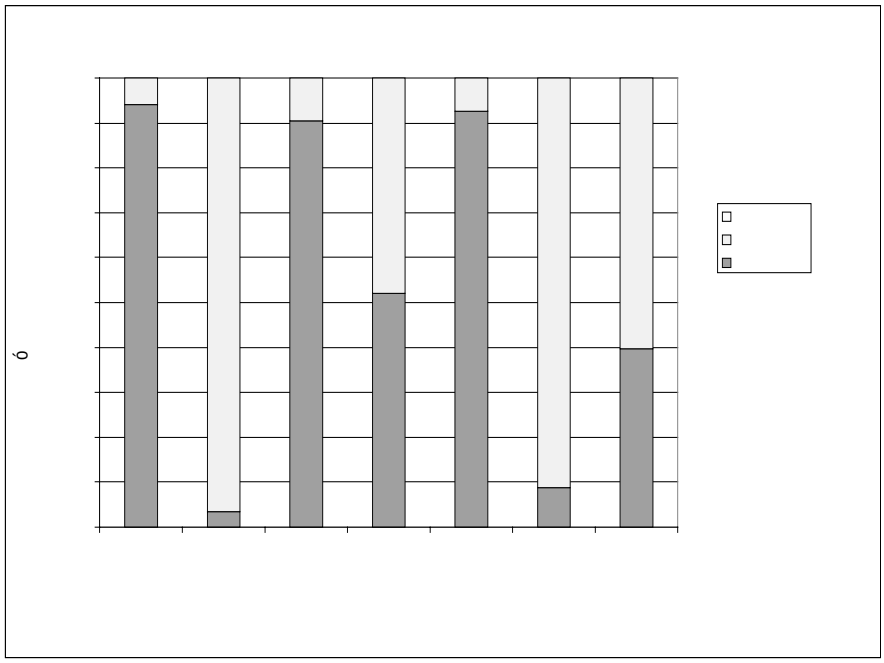

Figura 5: Distribución de saltos y bifurcaciones en bloques iterativos

de anidación es menor o igual a dos. Esto es, cuando los bloques iterativos contienen cero, uno, dos bloques anidados. Cuando este número es mayor, los patrones de ejecución son muy diversos y el grado de predicción de este tipo de ciclos se torna muy complejo.

En la figura 6 se muestran los valores de las medias aritmética y ponderada que fueron obtenidas a partir de estos patrones. Es importante señalar que las variaciones entre ambos valores reflejan el comportamiento tan diverso que existe en los ciclos de la aplicación. Por ejemplo, en 256.*bzip*2, la media aritmética del número de iteraciones es aproximadamente 5,000 mientras que su media ponderada es de 6.2. Tal situación ilustra como en 256.*bzip*2 existe un gran número de ciclos con un número pequenó de iteraciones y a la vez existe un número importante de ciclos con un gran número de iteraciones. Esta observación es de vital importancia para el diseño de un predictor de ciclos, debido a que debe determinarse durante ejecución cual ciclo iterará un número pequenó y cual lo hará con mayor dimensión.

La figuras 7 y 8 incluyen las medias aritmética y ponderada, obtenidas a partir de los patrones, del número de instrucciones por iteración. En este experimento se observó como ciclos anidados dominan la frecuencia de instrucciones por iteración. De igual manera, se detectó como un número limitado de ciclos son ejecutados un gran número de veces y permanecen en la pila dinámica de ciclos en ejecución por un largo periodo.

Al recolectar la información se midió la frecuencia con la que los ciclos repiten el número de instrucciones por iteración en forma consecutiva. La figura 9 indica las medias aritmética y ponderada de esta medición. En general, la media aritmética indica que los porcentajes de repetición oscilan entre un 38 y 63 porciento, lo cual representa que existe un gran número de ciclos que repiten sus patrones de iteración, pero que hay otro número que cambia constantemente. La media ponderada indica repeticiones de patrones de iteración entre 10 y 45 por ciento, lo que refleja de forma más acertada como es que los ciclos varian durante el transcurso del programa. Además, se determinó la probabilidad de que cualquier ciclo dado itere dos o más veces. Los

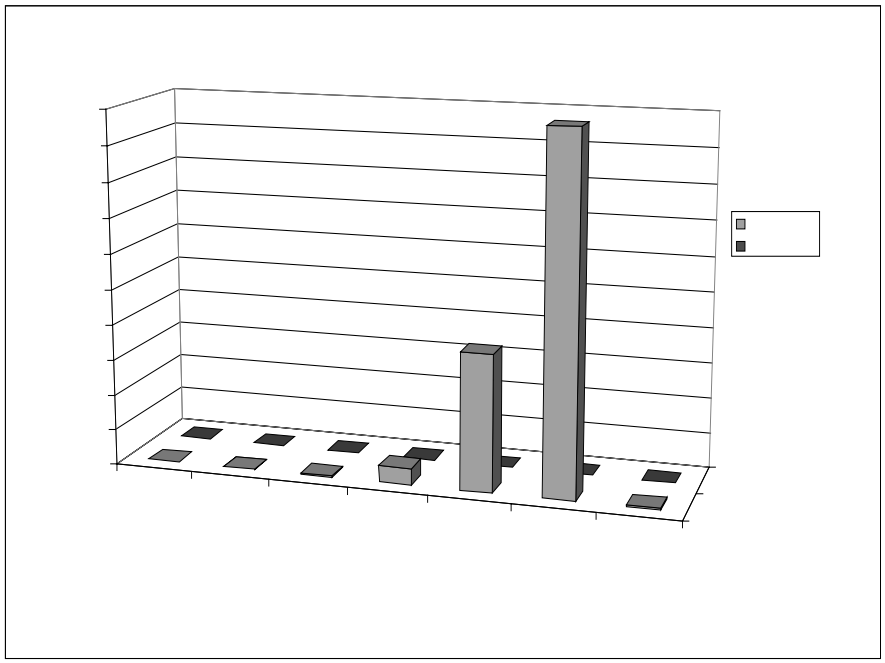

Figura 6: Medias aritmética y ponderada de iteraciones

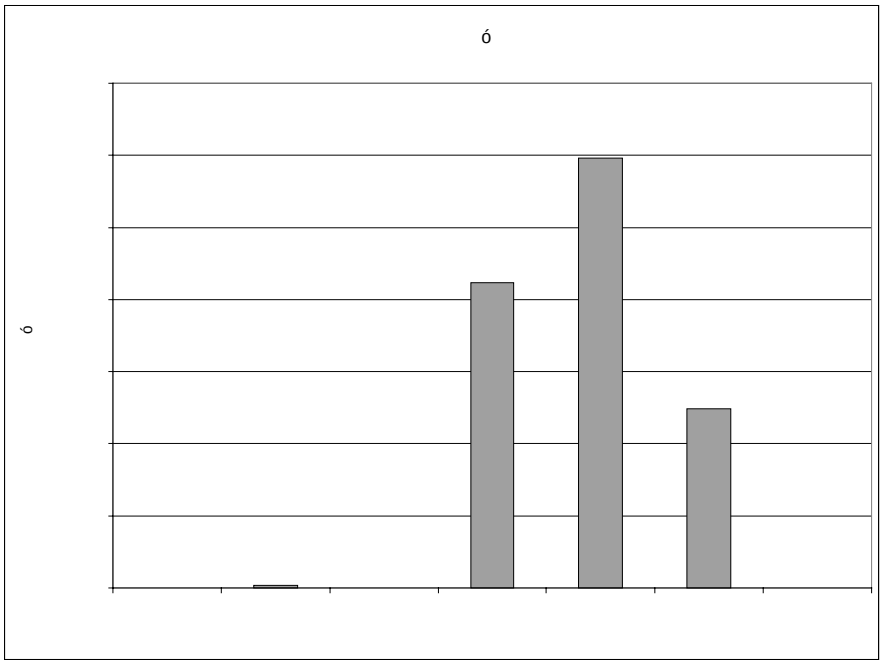

Figura 7: Media aritmética de instrucciones por iteración

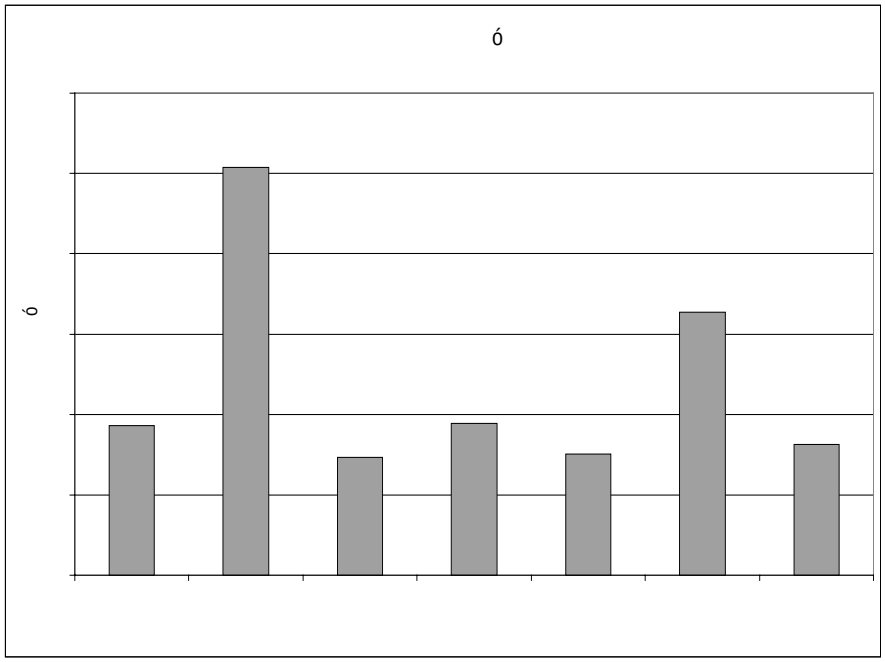

Figura 8: Media ponderada de instrucciones por iteración

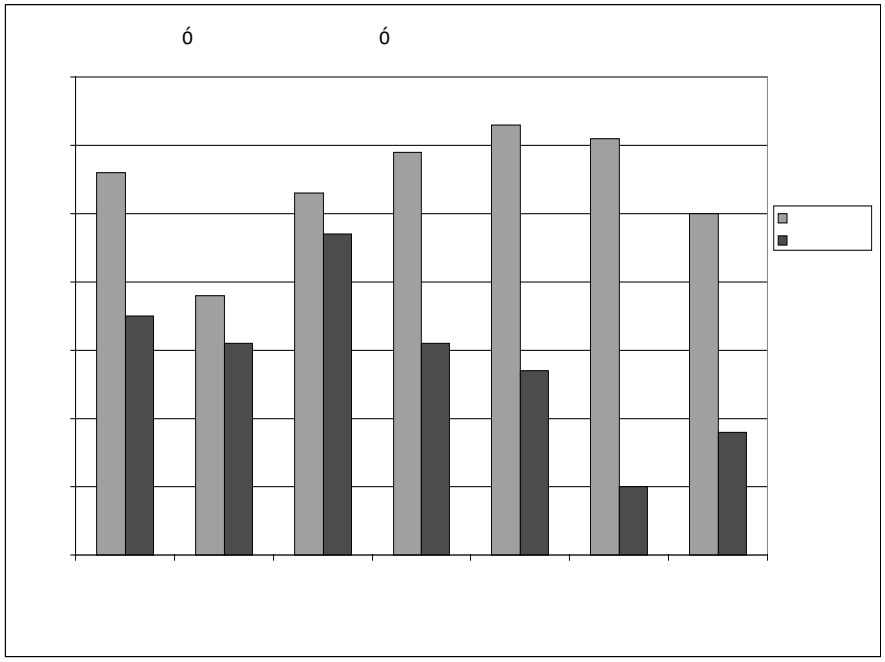

Figura 9: Probabilidad de repetición en número de instrucciones por iteración

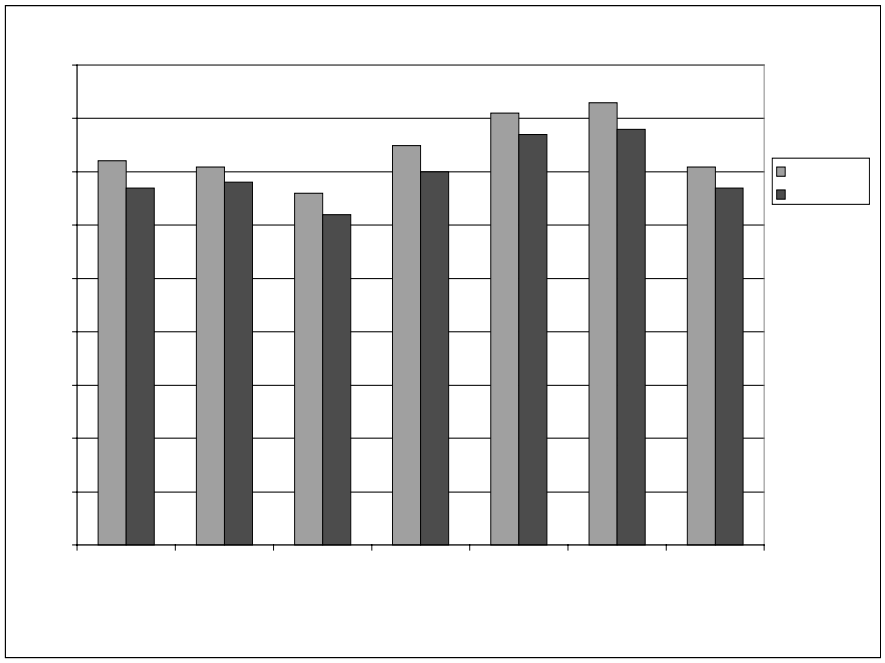

Figura 10: Probabilidad de que un bloque itere dos o más veces

resultados se muestran en la figura 10. Con estos resultados, se ve que un predictor de ciclos puede alcanzar altos niveles de precisión, debido a que errores de predicción iniciales serán corregidos con dos o más occurrencias del mismo ciclo.

Por último, se evaluó el nivel de anidación de las aplicaciones con el fin de establecer los parámetros de disenó preliminares para un predictor de ciclos. La figura 11 indica el máximo nivel obtenido en cada aplicación, además de las medias aritmética y ponderada.

Con la información recolectada, se observa que la pila de ciclos dinámicos no contiene más de 12 ciclos a la vez y que la tabla de predicción de ciclos debe tener 1500 o menos celdas para contener las estadísticas de todos los ciclos analizados. Estos parámetros se determinaron a partir del conjunto de *benchmarks* utilizadas, tomando los valores máximos necesarios en 176.*gcc*.

5 Conclusiones

Con el análisis dinámico realizado, se concluye que los bloques iterativos representan un punto de partida para la investigación y diseño de mejores técnicas de software y hardware para mejorar el desempeño de cualquier microprocesador superescalar. La localidad existente en este tipo de bloques estimula el desarrollo de mejores predictores de bifurcaciones y saltos. También es importante observar como este análisis es un indicador del potencial existente para proponer técnicas óptimas en la aplicación de hardware *loop unrolling*.

6 Trabajo futuro

A partir de este análisis se desarrollará un predictor de ciclos. Se evaluará el desempenó de un sistema con tal predictor, además será utilizado para aplicar la técnica de *loop unrolling* de forma eficiente.

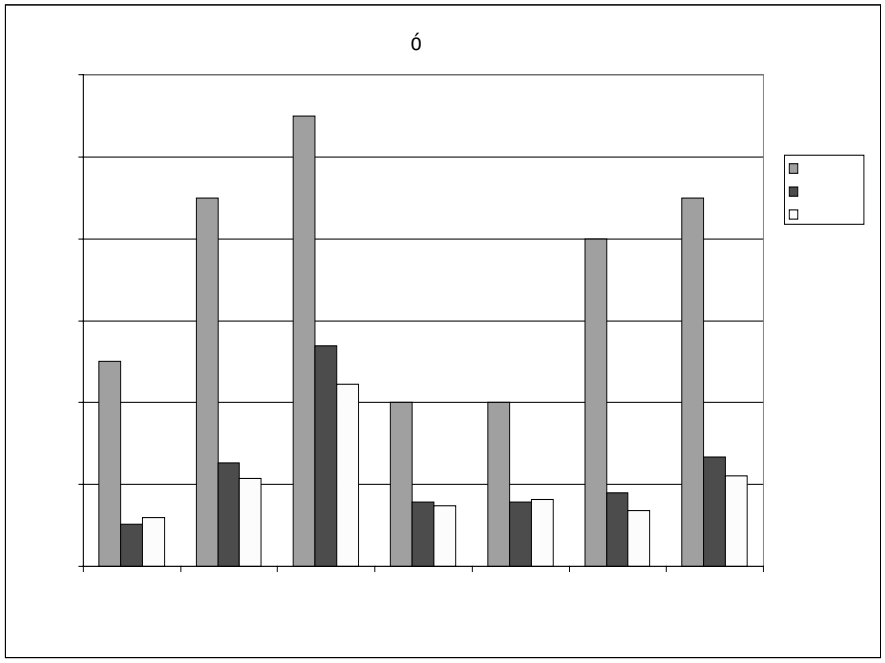

Figura 11: Nivel de anidación

Agradecimientos

Marcos de Alba es Profesor del Departamento de Electrónica de la Universidad Autónoma Metropolitana y es becario del programa Fulbright-Conacyt para la realización de estudios de Doctorado.

Referencias

[1] J. E. Smith. A study of branch prediction strategies. In *Proceedings of 8th Symposium in Computer Architecture*, pages 135–148, Minneapolis, MN, 1981.

[2] J. Lee and A. Smith. Branch prediction strategies and branch target buffer design. IEEE Computer 17(1), January 1984.

[3] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach, 2nd ed.* Morgan Kaufmann, Palo Alto, CA, 1995.

[4] T. Y. Yeh and Y.Ñ. Patt. A comparison of dynamic branch predictors that use two levels of branch history. In *The 20th Annual International Symposium on Computer Architecture*, pages 257–266, Goteborg, Sweden, 1993.

[5] John T. Coffey I-Cheng K. Chen and Trevor Mudge. Analysis of branch prediction via data compression. In *Proceedings of the seventh International Conference on Architectural support for programming languages and operating systems*, pages 128–137, Cambridge, MA, October, 1996.

[6] D. Kaeli and P. Emma. Improving the accuracy of history-based branch prediction. IEEE Transactions on Computers, 46(4):469–472, April 1997.

[7] John Kalamatianos. *Microarchitectural and Compile-Time Optimizations for Performance Improvement of Procedural and Object-Oriented Languages.* PhD thesis, Northeastern University, 2000.

[8] K. McKinley, S. Carr, and C-W Tseng. Improving data locality with loop transformations. *ACM Transactions in Programming Languages and Systems*, 18(4):424–453, 1996.

[9] Intel. *IA64 Application Developer's Architecture Guide*, May 1999.

[10] Y. Patt P-Y. Chang, E. Hao. Target prediction for indirect jumps. Proc. of the 24th Annual International Symposium on Computer Architecture, June 1997.

[11] K. Driesen and U. Hlzle. Accurate indirect branch prediction. Proceedings of The 25th International Symposium on Computer Architecture, July 1998.

[12] K. Driesen and U. Holzle. The cascaded predictor: Economic and adaptive branch target prediction. In Proceedings of the 31st International Symposium on Microarchitecture, November 1998.

[13] J. Tubella and A. Gonzalez. Control speculation in multithreaded processors through dynamic loop detection. In *Proceedings of the 4th International Symposium on High-Performance Computer Architecture*, Las Vegas, NV, January 1998.

[14] Antonio González and Pedro Marcuello. Dependence speculative multithreaded architecture. Technical report, Universitat Polytecnica de Catalunya, 1998.

[15] T. Sherwood and B. Calder. Loop termination prediction. In *Proceedings of the 3rd International Symposim on High Performance Computing.* Springer-Verlag, October 2000.

[16] A. Srivastava and A. Eustace. Atom: a system for building customized program analysis tools. DEC Western Research Laboratory Technical Report, 1994. Technical report.