UNIVERSITAT POLITÈCNICA DE CATALUNYA

Departament d'Arquitectura de Computadors

RECURSOS ANCHOS: UNA TÉCNICA DE BAJO COSTE PARA EXPLOTAR PARALELISMO AGRESIVO EN CÓDIGOS NUMÉRICOS

Autor: David López Alvarez
Directores: Mateo Valero Cortés
Josep Llosa i Espuny

Capítulo 3.
Límites en ILP y técnicas para alcanzarlos

3.1 Introducción

En este capítulo ilustraremos los factores que limitan la explotación del paralelismo en bucles numéricos donde se aplica segmentación software, así como las técnicas que nos permitirán alcanzar dichos límites. Como ya hemos visto, los ciclos necesarios para la ejecución de un bucle están limitados por las recurrencias de dicho bucle y por los ciclos necesarios para la ejecución de sus operaciones en una arquitectura dada. Así pues, veremos técnicas para reducir cada uno de estos límites, estudiando pros y contras de cada una de ellas. También se presentará un sistema de clasificación de bucles que será utilizado con los bucles del juego de pruebas.

Antes de entrar a estudiar los límites, veamos una perspectiva histórica de trabajos relacionados con el estudio del paralelismo.

Los primeros experimentos llevados a cabo para medir paralelismo se realizaron sobre bloques básicos. Los resultados obtenidos en estos experimentos mostraron que, aun con

hardware infinito, no se podía explotar un grado de paralelismo superior a un factor de 2 ó 3 instrucciones por ciclo.

Nicolau y Fisher [NiFi84] usaron el compilador Bulldog [Ell86] para evaluar el paralelismo de un conjunto de programas científicos. El scheduling de estos programas fue hecho con un oráculo, es decir, se calculó el paralelismo máximo que se podría encontrar con las trazas generadas durante la ejecución de los programas. El incremento de velocidad (*speed-up*) encontrado variaba entre 4 y 988, siendo la media de 90. Cuando los autores repitieron el experimento utilizando paralelismo entre bloques básicos, la media fue de 2.5.

Jouppi y Wall [JoWa89] realizaron un estudio de paralelismo para procesadores superescalares y supersegmentados, encontrando un paralelismo entre bloques básicos entre 1.5 y 2 para programas no numéricos y entre 2 y 3 para programas numéricos. Es interesante destacar que en este trabajo también se estudió el incremento de paralelismo en el código numérico cuando se aplicaba *unrolling* (técnica que emplearemos en los capítulos siguientes), encontrando que el grado de paralelismo se incrementa hasta llegar a 6 cuando se aplicaba un grado de *unrolling* de 10.

Smith, Johnson y Horowitz [SJH89] realizaron un estudio para procesadores superescalares basándose en las trazas generadas por el procesador MIPS R2000. No fue propiamente un estudio del paralelismo máximo, sino del alcanzable con una limitación de los recursos de una máquina, probando varios códigos no numéricos con diversas configuraciones de unidades funcionales. Los rangos de paralelismo encontrados variaban entre 2 y 4, dependiendo de lo realista de la configuración.

Wall [Wal91] realizó un extenso estudio teniendo en cuenta técnicas como ejecución especulativa, desambiguación de memoria, etc. Con un juego de pruebas que comprendía 17 programas (algunos de los SPEC y programas reales usados en el *Western Research Laboratory* de Digital), compilados para un MIPS R3000 se creó un elaborado modelo donde

se estudió el límite de ILP para diversas configuraciones. El paralelismo encontrado variaba entre 7 y 60 para las configuraciones más agresivas.

Lam y Wilson [LaWi92] realizaron un experimento para evaluar diferentes métodos de eliminar problemas con el control de flujo. Encontraron un grado de paralelismo entre 18 y 400 para código no numérico y entre 196 hasta casi infinito en código numérico. Con las configuraciones usadas por Wall, los resultados eran prácticamente iguales.

Todos estos estudios dan resultados bastante diferentes debido a que han sido realizados con gran variación de parámetros, además de con juegos de pruebas diferentes. En este capítulo presentaremos un estudio de los límites del ILP de los bucles del Perfect Club que nos permitirá proponer métodos para alcanzar dichos límites.

3.2 Limitaciones: recursos y recurrencias

Amo las limitaciones, porque son la causa de la inspiración.

Susan Sontag

Como vimos en el capítulo anterior, el número mínimo de ciclos entre el inicio de la ejecución de dos iteraciones consecutivas (llamado *Minimum Initiation Interval* o *MII*) en un bucle donde se aplica planificación software, depende tanto del límite debido a las recurrencias (*Recurrence-constrained Minimum Initiation Interval, RecMII*) como del límite debido a los recursos disponibles (*Resource-constrained Minimum Initiation Interval, ResMII*), siendo *MII=max(ResMII, RecMII)*. Así dada una arquitectura, podríamos dividir los bucles en limitados por recurrencias (*recurrence bound*) y limitados por recursos (*resource bound*).

Bucles *recurrence bound*

Consideraremos un bucle *recurrence bound* cuando tenga el *RecMII* mayor o igual que el *ResMII*. En este caso la cantidad de unidades funcionales no afecta al rendimiento, pues las operaciones de la recurrencia no pueden ejecutarse en paralelo y aun con infinitas unidades funcionales la ejecución del bucle no puede ser más rápida. ¿Cómo podemos pues incrementar

el rendimiento de estos bucles?. La solución está en reducir el número de ciclos que se necesita para ejecutar las operaciones de la recurrencia. Esto puede solucionarse reduciendo la latencia de las unidades funcionales o bien realizando operaciones más complejas con la misma latencia.

Bucles *resource bound*

Los bucles limitados por recursos (con *ResMII>RecMII*) pueden incrementar su rendimiento a base de incrementar el número de operaciones que se pueden realizar por ciclo. Una planificación con segmentación software óptima aprovecha el recurso más escaso de manera que esté al 100% de ocupación (véase sección 2.3). Si suponemos unidades funcionales completamente segmentadas, el *ResMII* depende del número de operaciones y de la cantidad de unidades funcionales, pero es independiente de la latencia de las mismas.

Los grafos limitados por recursos se pueden dividir en los siguientes grupos:

- *memory bound* (limitados por memoria): en los que el recurso más escaso es el recurso de acceso a memoria.
- *compute bound* (limitados por cálculos): en los que el recurso más escaso es alguno de los que resuelve operaciones aritméticas.
- *balanced* (balanceado): en los que el *ResMII* debido a las unidades de memoria es igual al *ResMII* debido a las unidades aritméticas.

Clasificación de un bucle

No se puede afirmar a qué tipo de los anteriores pertenece un grafo "per se". Sólo se puede hacer una clasificación de un grafo en función de la arquitectura en que se quiere ejecutar. Así por ejemplo, un bucle memory bound en una arquitectura con 1 bus entre el banco de registros y el primer nivel de memoria cache, puede estar limitado por recurrencias, operaciones aritméticas o ser balanceado en una arquitectura con dos buses. Para ilustrar estos

conceptos utilizaremos un código ejemplo cuyo grafo de dependencias puede observase en la figura 3.1a.

En el grafo ejemplo tenemos un total de 7 operaciones: 3 operaciones de memoria (loads L0 y L1 y store S6) y 4 operaciones aritméticas (productos *2, *3 y *4 y suma +5); para simplificar no dibujaremos los arcos de invariantes que deberían entrar en los nodos *3 y +5. Se puede observar que existe una recurrencia dado que la operación *4 usa como uno de los datos de entrada el resultado generado por la operación +5 en la iteración anterior.

Si suponemos que las unidades funcionales están totalmente segmentadas y que se realiza una planificación óptima, el número de ciclos dedicados a operaciones de cualquier tipo dependerá del número de operaciones y del número de recursos. Por ejemplo, si suponemos una arquitectura con una unidad de memoria (l/s) y una unidad aritmética de coma flotante (fpu), la limitación de memoria será de 3 ciclos pues hay 3 operaciones de memoria; asimismo la limitación por operaciones aritméticas será de 4 ciclos, al haber 4 operaciones aritméticas.

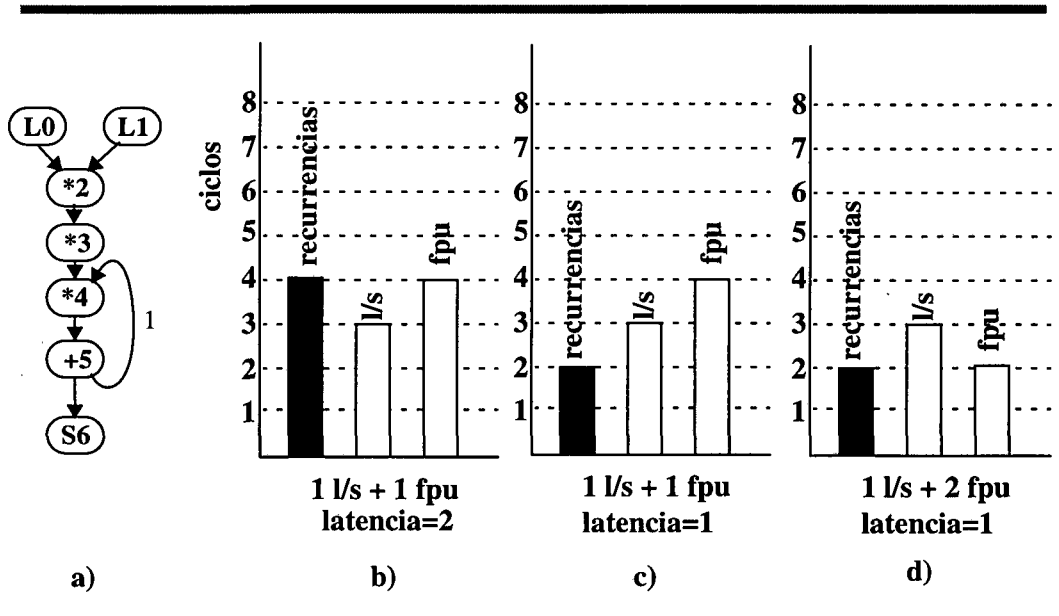


Figura 3.1: Grafo de dependencias ejemplo y sus límites para diversas configuraciones

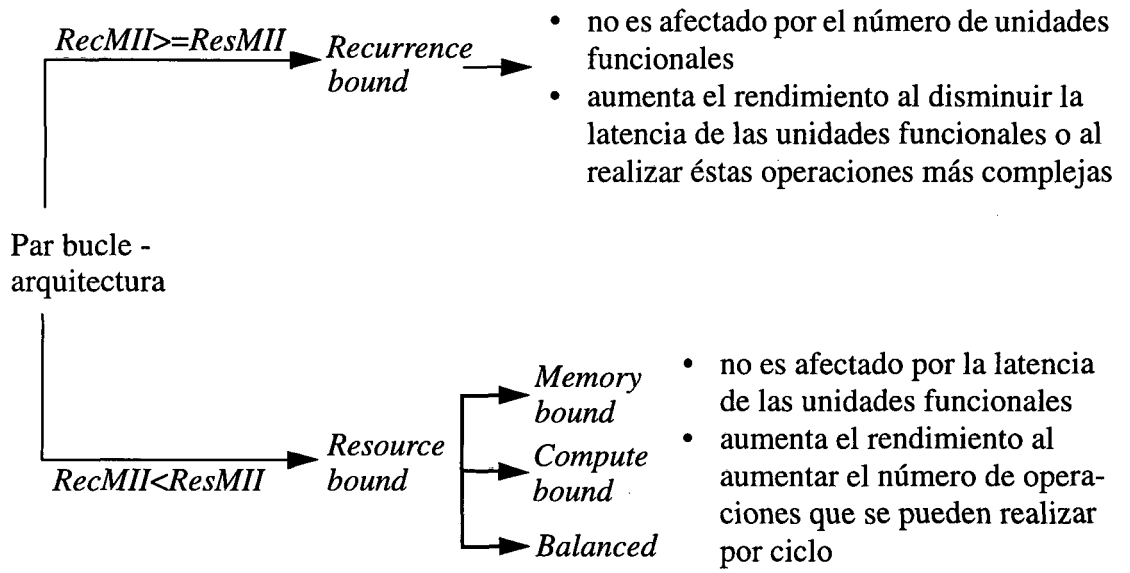


Figura 3.2: Clasificación de un par bucle-arquitectura en función de sus limitaciones

En el caso de las recurrencias pasa lo contrario: afecta la latencia de las operaciones, mientras que el número de unidades funcionales no afecta al rendimiento. La figura 3.1b muestra los límites para 1 l/s y 1 fpu de latencia 2 ciclos. Al tener la recurrencia dos operaciones implicadas, no se puede lanzar la operación *4 de la iteración *i* hasta 4 ciclos después, cuando se dispone del resultado de la operación +5 de la iteración *i-1*. En este caso, el grafo es *recurrence bound*. Sin embargo, si la latencia fuera de 1 ciclo (figura 3.1c), la recurrencia ocuparía 2 ciclos y el grafo estaría limitado por las operaciones aritméticas (*compute bound*). En la figura 3.1d, en que tenemos latencia igual a 1 pero donde hemos añadido una unidad aritmética, las limitaciones por operaciones aritméticas ocupan 2 ciclos. En este caso, quien limita el rendimiento son las operaciones de memoria (*memory bound*).

A modo de resumen, la figura 3.2 nos muestra las posibles clasificaciones de un par bucle-arquitectura. Como hemos visto, el rendimiento teórico de un bucle depende de la arquitectura en la que se ejecuta.

3.3 Incrementando el rendimiento de bucles limitados por los recursos

P: ¿Hay límites para las necesidades del hombre?
R: Sí, claro, los recursos de la tierra.

Jacint Ros Hombravella. Interviú en La Vanguardia.

El rendimiento de los bucles limitados por los recursos se puede mejorar incrementando el número de operaciones que se pueden lanzar a ejecutar por ciclo en un procesador. Este objetivo ha sido alcanzado en muchos de los microprocesadores actuales replicando el número de unidades funcionales disponibles (técnica de replicación o *replication technique*).

Dicha técnica tiene el inconveniente de que resulta muy costosa; en este trabajo proponemos una alternativa de menor coste: hacer los recursos más anchos (*widening technique*). A continuación se describen ambas técnicas y sus costes.

3.3.1 Replicación

-You are a replicant. I design your eyes.
-Chew, if only you could see what I've seen with your eyes.

Diálogo entre Chew y Roy Batty. Blade Runner, 1982.

Para explicar la técnica de replicación, tomemos una arquitectura básica como la de la figura 3.3a. En esta arquitectura disponemos de un banco de registros (*Register File*, RF) y dos recursos: una unidad funcional en coma flotante multifunción (FPU) y un bus que conecta el banco de registros y el primer nivel de memoria cache. Al tener dos recursos, seríamos capaces de enviar a ejecutar dos instrucciones por ciclo, una ocupando cada recurso. Por tanto, podríamos enviar a ejecutar una instrucción aritmética y una de memoria por ciclo.

Para duplicar el número de operaciones emitidas (*issued*) por ciclo usando la técnica de replicación, deberíamos añadir otra FPU y otro bus, de manera que podríamos emitir dos instrucciones aritméticas y dos de memoria por ciclo (figura 3.3b).

Definiremos el grado de replicación como el número de veces que se han replicado los recursos de la arquitectura base. Aplicar replicación tiene una serie de costes, tanto dentro como fuera del procesador, que apuntaremos a continuación. El estudio en profundidad del cálculo de costes está descrito en el capítulo 6.

- A nivel del núcleo del procesador, incrementar los recursos requiere incrementar el número de puertos del banco de registros [GAB+88, LMM92], lo que incrementa el tiempo de acceso al banco de registros [WeEs88, LLVA98c]. Además, incrementar el número de puertos tiene un gran impacto en el área requerida, ya que usando tecnología CMOS, el área de un recurso multipuerto es proporcional al cuadrado del número de puertos [Jol91, Lee92, LLVA98a].

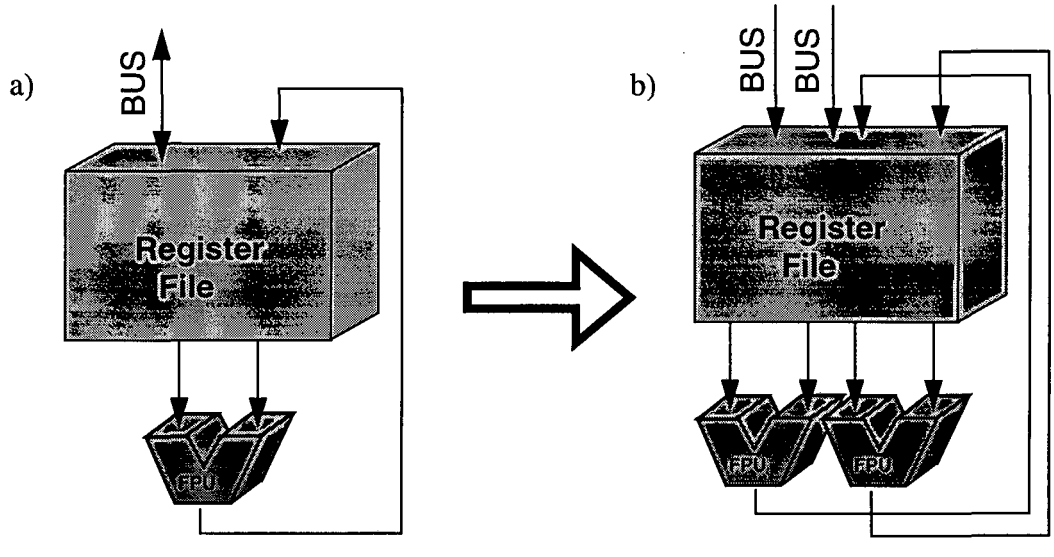


Figura 3.3: Aplicación de la técnica de replicación a una arquitectura básica

- A nivel de la cache interna del procesador, se requiere que sea multipuerto para poder implementar diversos accesos por ciclo, y hacer que una cache sea multipuerto incrementa su coste en área y su tiempo de acceso [JNT97]. Una cache multipuerto se puede implementar replicando la cache y manteniendo copias idénticas de cada cache [ERPR95] lo que multiplica el área requerida para la cache. Otra solución es fraccionar la cache en varios bancos independientes [SoFr91], aunque los conflictos de acceso pueden reducir el ancho de banda efectivo.

- Para traducir direcciones de memoria virtual a memoria física de una manera rápida (a baja latencia), la mayoría de los microprocesadores actuales utilizan un hardware llamado *Translation Lookaside Buffer* (TLB). Un TLB es una cache, altamente asociativa, que contiene entradas de la tabla de páginas de memoria virtual y cada entrada describe la dirección física de una página de memoria virtual, así como sus permisos de acceso y estatus. Traducir diversas instrucciones por ciclo requiere que el TLB sea multipuerto, lo que incrementa sus costes [CLC+95, AS96].

- A nivel de la conexión fuera del *chip*, incrementar el número de buses (direcciones, datos y control) incrementa el número de pins requeridos por el procesador encareciendo el empaquetamiento del mismo. Además, se aumenta la complejidad de las caches *off-chip* en un aspecto similar a las *on-chip* caches.

Los costes expuestos serían los principales para un procesador de tipo VLIW, que es el tipo de arquitectura sobre el que basamos nuestro trabajo. Un procesador superescalar tendría unos costes adicionales, ya que cada incremento en el número de instrucciones que se puede emitir (*issue*) complica el hardware que controla las condiciones de emisión, incrementando su coste en área y tiempo de respuesta (en ciertos casos, dicho hardware es tan complejo que es el que marca el tiempo de ciclo).

3.3.2 *Widening*

Wider is better.

Linley Gwennap. AltiVec Vectorizes PowerPC. Microprocessor Report V.12, N.6, 1998

Una alternativa a la técnica de replicación es la técnica de *widening* (o de arquitecturas anchas), en la que no se incrementa el número de recursos, sino que se incrementa su anchura, es decir, el número de datos que se pueden procesar en cada recurso. Definiremos el grado de *widening* (o anchura) de una arquitectura como la cantidad de datos que se pueden procesar simultáneamente en una única instrucción. La idea puede observarse en la figura 3.4, donde se aplica un grado 2 de *widening* al bus, al banco de registros y a las FPUs.

Tomemos por ejemplo el bus, que es el recurso que nos impondrá más restricciones a la hora de aplicar *widening*. Un bus ancho es capaz de mover más de un dato entre el banco de

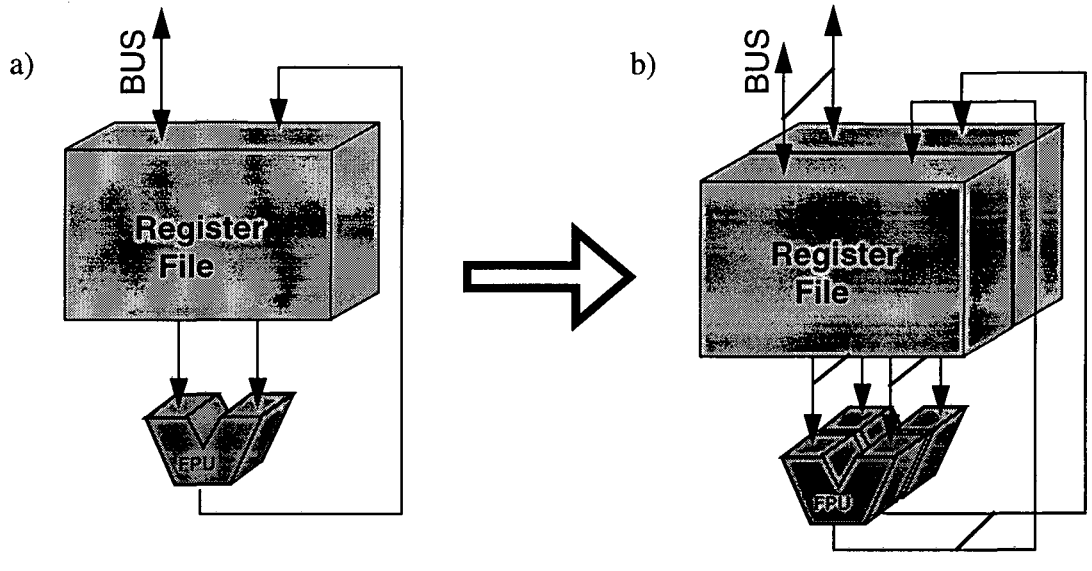


Figura 3.4: Aplicación de la técnica de widening a una arquitectura básica

registros y el primer nivel de memoria cache; se lanza una única dirección de memoria y se mueven dos (o más) datos de memoria, siempre y cuando la ubicación de estos datos sea consecutiva en memoria. Estos datos se almacenarían en un único registro del banco de registros, por lo que dichos registros deben ser de la misma anchura que los recursos disponibles. Igualmente se pueden ejecutar varias instrucciones aritméticas siempre y cuando sean la misma operación (suma, producto, etc) sobre datos anchos.

Así pues, la técnica de *widening* se aplica sobre instrucciones que operan con varios datos, por lo que se necesita *compactar* diversas operaciones en una única instrucción ancha (nótese que estamos modificando el lenguaje máquina); en cierto aspecto es como si especificamos varias instrucciones escalares en una instrucción similar a una vectorial. Las técnicas necesarias para compactar se estudiarán en profundidad en el capítulo 4; veamos ahora un simple ejemplo de compactación.

En la figura 3.5a se muestra el grafo de dependencias extendido para nuestro ejemplo; los nodos corresponden a operaciones y los arcos a dependencias entre operaciones. Los arcos punteados muestran *stride* entre operaciones de memoria (por ejemplo, el load L0 tiene un *stride* 1 consigo mismo en la siguiente iteración). Todas las dependencias están dentro de una iteración (distancia 0) excepto la dependencia entre operaciones (*5, *5) en la cual el resultado de la operación *5 es usado por la misma operación en la siguiente iteración (recurrencia de distancia 1).

Los nodos que compactaremos serán copias de la misma operación en iteraciones consecutivas, por lo que será necesario desenrollar el bucle para poder compactar dichas copias. La figura 3.5b muestra el bucle ejemplo una vez se ha desenrollado el bucle y tenemos dos copias del grafo (grado de *unrolling* u=2), así que las operaciones de dos iteraciones son consideradas a la hora de hacer la planificación. Los subíndices indican a qué iteración pertenece cada nodo.

En la misma figura 3.5b se muestran unidas por bandas grises las operaciones que son compactables. Dos operaciones aritméticas son compactables si no existe ninguna dependencia entre ellas, de manera que todas las operaciones aritméticas de la figura son compactables menos el par ($*5_0$, $*5_1$) que tiene una recurrencia. Respecto a las operaciones de memoria, son compactables siempre y cuando no haya dependencia entre ellas y haya un arco de *stride* 1 entre las mismas. Así en el ejemplo, los loads L0 y L1 son compactables mientras que el store S6 no lo es, dado que no tiene *stride* 1. La figura 3.5c muestra el grafo compactado.

Aplicando un grado de *widening* **n**, obtenemos el mismo rendimiento máximo teórico que aplicando el mismo grado de replicación, siempre y cuando todas la operaciones sean compactables. Como desgraciadamente no todas las operaciones serán compactables, la técnica *widening* tendrá un rendimiento inferior a la técnica de replicación. Sin embargo, podemos observar cómo *widening* tiene unos costes menores que replicación. La discusión

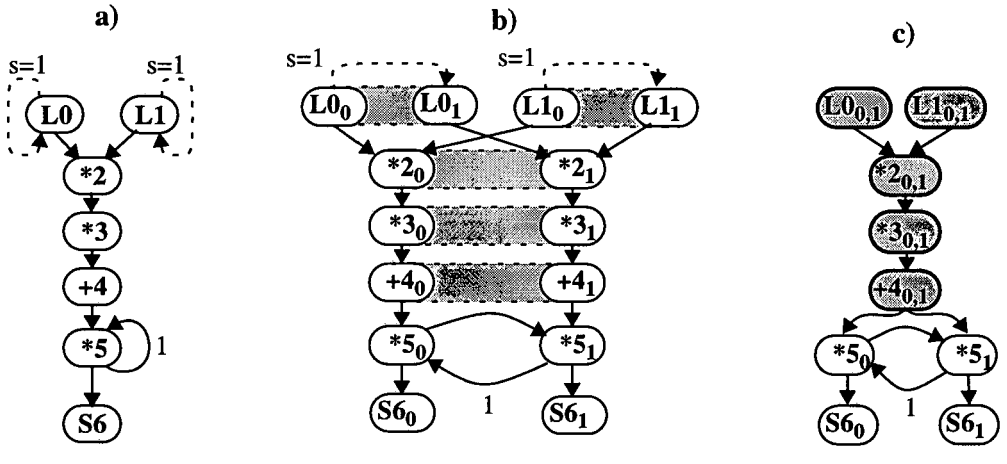


Figura 3.5: Ejemplo de compactación para arquitecturas con un grado de widening=2. a) Grafo ejemplo b) Grafo después de aplicar un grado de unrolling u=2 (las bandas grises marcan operaciones compactables) c) Grafo compactado

	replicación	*widening*
buses: direcciones control datos	2 2 2	1 1 2
TLB	multipuerto	puerto simple
Cache	multipuerto	puerto simple
puertos del banco de registros	duplicados	no se modifica
ancho de los registros	x1	x2
FPU hardware	duplicado	duplicado
analisis estático	no necesario	necesario
ancho de banda real	x2	entre 1 y 2
instrucciones especiales	no	sí

Tabla 4.1: costes de duplicar el rendimiento máximo teórico de la arquitectura base usando replicación y *widening*

completa de los costes se presentará en el capítulo 6, pero podemos ver en la tabla 4.1 la diferencia entre una configuración donde tiene un ancho de banda teórico duplicado respecto a la arquitectura original (1 bus, 1 fpu) usando replicación y usando *widening*.

En la tabla 4.1 se puede observar las diferentes necesidades de hardware de ambas técnicas cuando aplicamos un grado, tanto de replicación como de *widening* de 2. La técnica de replicación requiere dos buses de direcciones, datos y control, mientras que *widening* requiere duplicar únicamente el bus de datos. Al generarse dos direcciones por ciclo cuando se usa replicación, se requiere hacer multipuerto el TLB y la memoria cache, mientras que con *widening* sólo se genera una dirección, con los que el TLB y la cache no varían.

Respecto al banco de registros, *widening* implica que los registros sean el doble de anchos, con la consiguiente penalización respecto al coste en área. Sin embargo se dispone de más capacidad de almacenamiento en dicho banco, lo que puede ser una ventaja si se necesita

generar *spill code*. Además replicación incrementa el número de unidades funcionales, por lo que el número de puertos de lectura/escritura de cada celda de memoria del banco de registros se incrementa, aumentando a su vez el área requerida.

Respecto a las FPUs, el coste apenas varía, dado que si se quieren hacer dos operaciones, sea usando replicación o usando *widening*, hace falta el hardware necesario para realizar estas dos operaciones. Como ya hemos dicho, el capítulo 6 de este trabajo está dedicado a la evaluación de los costes de ambas técnicas y su comparación.

3.3.3 Clasificación de los bucles

Retomemos el ejemplo de la figura 3.5 y supongamos que queremos planificarlo en una arquitectura con 1 unidad aritmética en coma flotante (FPU) y una unidad de memoria (L/S). Asumamos que todas las operaciones están completamente segmentadas y tienen una latencia de 2 ciclos.

Tal como ya hemos visto en la sección 2.3, la técnica de segmentación software planifica conjuntamente operaciones de diversas iteraciones, así que las limitaciones de un bucle son las recurrencias del mismo y los recursos disponibles. Para cada par bucle/arquitectura es posible calcular el rendimiento máximo posible de una iteración. En nuestro ejemplo consideraremos 3 límites separados (véase figura 3.6d1):

- Recurrencias: En el bucle 3.6b hay una recurrencia ($*5_0$, $*5_1$) que limita la ejecución del bucle a 4 ciclos por iteración, dado que la suma de las latencias de las operaciones de la recurrencia es de 4 ciclos y la suma de las distancias de la recurrencia es 1. En este punto, consideramos los límites por recurrencias como insalvables.

- Memoria (MU): en el bucle 3.6b hay 6 operaciones de memoria ($L0_0$, $L0_1$, $L1_0$, $L1_1$, $S6_0$ y $S6_1$). Como disponemos de una sola unidad de memoria completamente segmentada, necesitamos al menos 6 ciclos para ejecutar las 6 operaciones.

- Aritmética (FPU): En el bucle 3.6b hay 8 operaciones en coma flotante ($+2_0$, $+2_1$, $*3_0$, $*3_1$, $+4_0$, $+4_1$, $*5_0$ y $*5_1$). Como hay una única FPU completamente segmentada, son necesarios al menos 8 ciclos para ejecutar las operaciones aritméticas.

Como conclusión, el factor que más limita el bucle ejemplo son las operaciones aritméticas, que requieren 8 ciclos por iteración del bucle desenrollado (lo que corresponde a 4 ciclos por iteración del bucle original), como se puede ver en la figura 3.6d1, en que cada barra marca uno de los factores que limitan el rendimiento.

En la figura 3.6d1 se han dividido las barras que marcan las limitaciones (tanto de memoria como aritméticas) en:

- debidas a operaciones compactables (la parte blanca)

- debidas a operaciones no compactables (parte rayada)

Si aplicamos la técnica de *widening* (obteniendo una arquitectura con 1 FPU y 1 MU, ambas de ancho 2) podemos reducir los límites, pero sólo la parte debida a operaciones compactables. En nuestro ejemplo, la parte blanca de las barras puede ser reducida a la mitad, mientras que la parte rayada (correspondiente a operaciones no compactables) no varía (figura 3.6d2), obteniendo un rendimiento final de 2.5 ciclos por iteración.

Se puede lograr más paralelismo empleando replicación en lugar de *widening*. La figura 3.6d3 muestra los límites para una arquitectura con 2 FPUs y 2 MUs. Nótese que en este caso, ambas partes de la barra, tanto las pertenecientes a operaciones compactables como las pertenecientes a operaciones no compactables, se reducen a la mitad. En este caso se consigue un rendimiento de 2 ciclos por iteración. Nótese también que usando replicación hemos alcanzado el límite impuesto por las recurrencias, de manera que no podemos explotar más paralelismo incluso poniendo infinitos recursos.

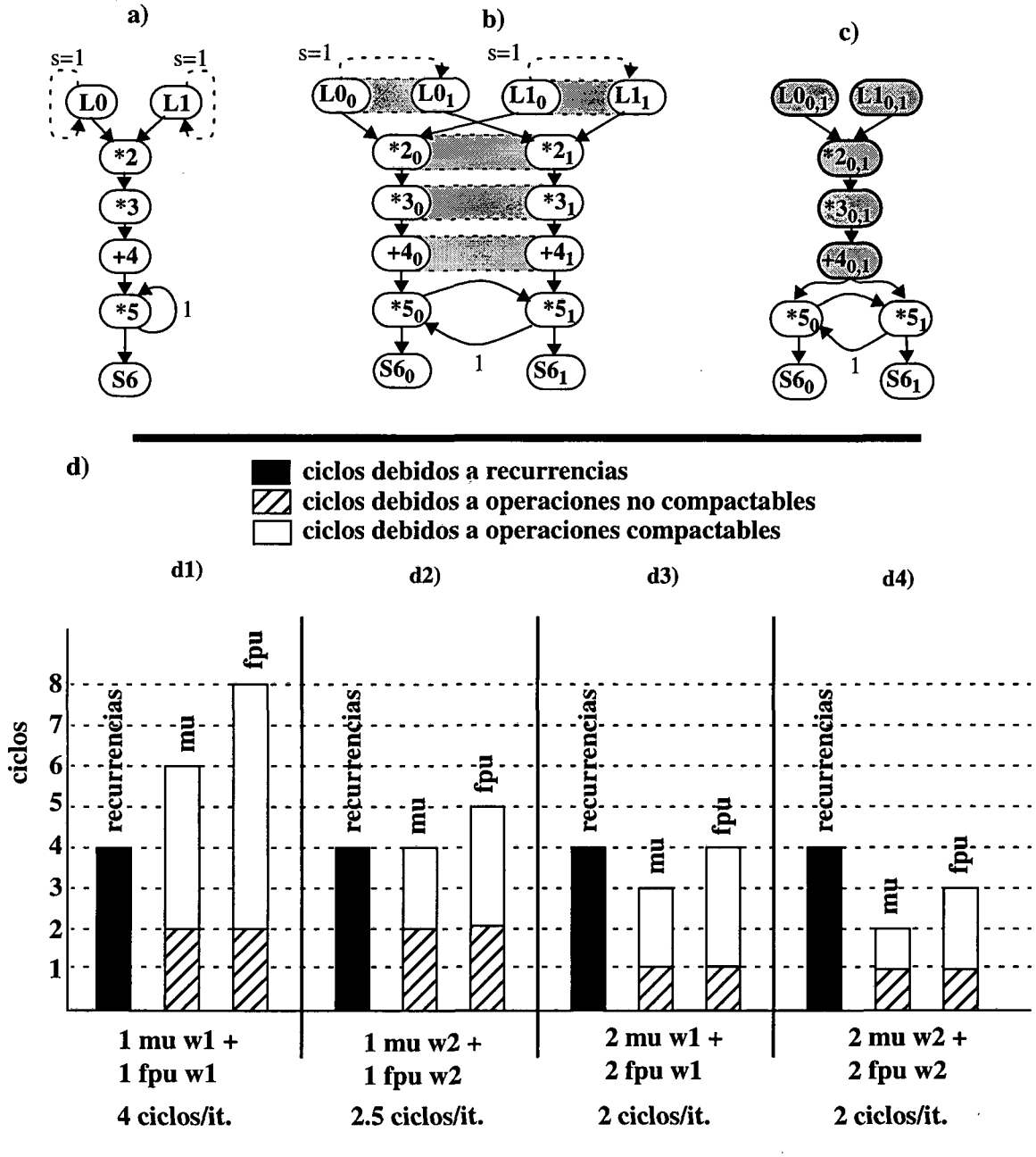


Figura 3.6: a) Grafo de dependencias ejemplo b) Grafo tras aplicar un grado de unrolling u=2 (la marca gris indica operaciones compactables) c) Grafo compactado d) Ciclos necesarios para ejecutar dos iteraciones en diversas arquitecturas

Las dos técnicas pueden combinarse. La figura 3.6d muestra los límites con una arquitectura con 2 FPUs y 2 MUs, ambas de ancho 2. En este caso, las operaciones no compactables se han reducido a la mitad, mientras que las compactables se han reducido a la cuarta parte (hemos doblado tanto el grado de replicación como el de *widening*). Desafortunadamente, el bucle sigue requiriendo dos ciclos por iteración debido al límite de la recurrencia.

De este ejemplo se pueden extraer unas primeras conclusiones que nos permitirán entender mejor la clasificación de los bucles que se especificará más tarde:

- los bucles con recurrencias están limitados por la latencia de las operaciones implicadas en el bucle. Una vez se alcanza este límite, no podemos aumentar el paralelismo con ninguna de las dos técnicas descritas.

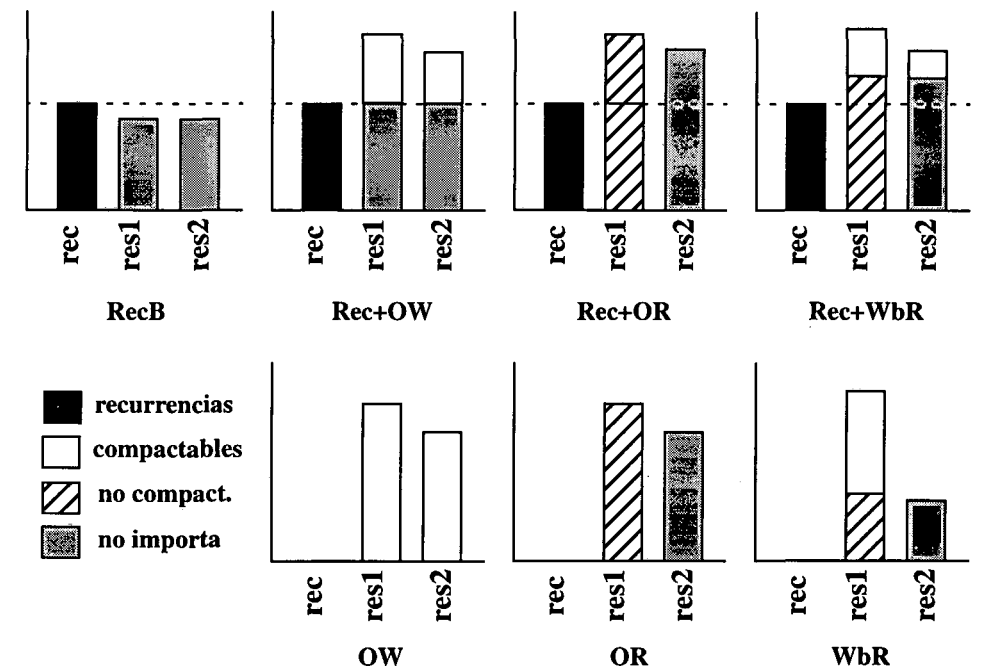
- la técnica de replicación reduce los límites impuestos por los recursos en un factor igual al grado de replicación aplicado.

- la técnica de *widening* reduce los límites impuestos por los recursos, pero sólo afecta a los ciclos debidos a operaciones compactables, que se reducen en un factor igual al grado de *widening*, mientras que los ciclos debidos a operaciones no compactables continúan igual, incluso con un grado de *widening* infinito.

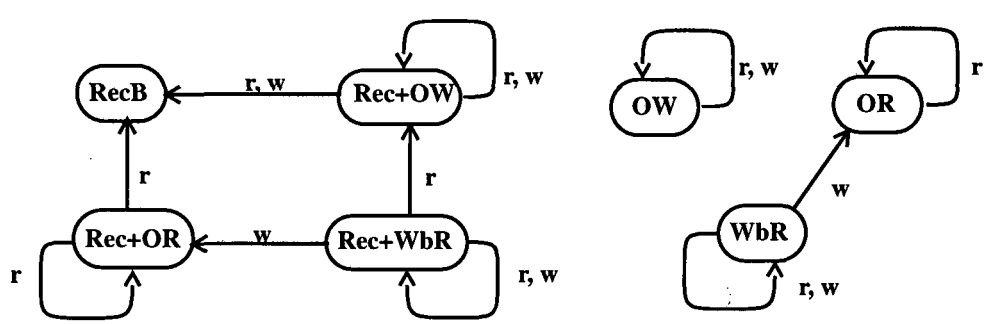
La figura 3.7a muestra una clasificación de los bucles en función de los factores que los limitan. La clasificación se divide en:

- RecB (*Recurrence Bound)*: son bucles cuyo rendimiento está limitado por las recurrencias (la barra negra) y no puede ser incrementado ni usando replicación ni usando *widening*.

- Rec+OW (*Only Widening with Recurrences*): son bucles con recurrencias cuyo factor limitador más importante son los recursos, y el límite debido a la recurrencia puede ser



a)



b)

*Figura 3.7: a) Tipos de bucles b) Transiciones posibles entre tipos de bucles aplicando replicación (**r**) o widening (**w**)*

alcanzado usando la técnica de *widening*. El mismo nivel de rendimiento puede alcanzarse usando únicamente la técnica de replicación.

- Rec+OR (*Only Replication with Recurrences*): son bucles con recurrencias cuyo factor limitador más importante son los recursos y en los que la única manera de incrementar el rendimiento es usando replicación (no hay operaciones compactables).

- Rec+WbR (*Widening but Replication with Recurrences*): bucles con recurrencias cuyo factor limitador más importante son los recursos, en los que se puede incrementar el rendimiento usando *widening*, pero que no se puede alcanzar el máximo rendimiento usando sólo esta técnica.

- OW (*Only Widening*): bucles sin recurrencias cuyas operaciones son todas compactables (el mismo rendimiento puede alcanzarse aplicando sólo replicación).

- OR (*Only Replication*): bucles sin recurrencias y sin ninguna operación compactable. Sólo podemos incrementar el rendimiento usando replicación.

- WbR (*Widening but Replication*): bucles sin recurrencias donde hay operaciones compactables y operaciones no compactables, de manera que aplicando *widening* podemos incrementar el rendimiento, pero replicación ofrece mayor incremento de rendimiento.

La figura 3.7b muestra las posibles transiciones entre los diferentes tipos de bucles de la clasificación anterior cuando se aplica replicación o *widening*. Por ejemplo, un bucle Rec+WbR puede convertirse en Rec+OR después de aplicar un grado de *widening* de 2. Puede mantenerse como Rec+OR aplicando la técnica de replicación (pero con rendimiento cada vez más cercano al límite) y convertirse finalmente en RecB después de un cierto grado de replicación. Una vez el bucle es RecB, su rendimiento se mantendrá constante, independientemente de cuántas veces se le vuelva a aplicar replicación o *widening*.

3.3.4 Caracterización del juego de pruebas

La tabla 4.2 muestra los bucles de nuestro juego de pruebas, según la clasificación mostrada en el punto anterior, para una arquitectura con 1 bus y una FPU multifunción, con latencias de 2, 3 y 4 ciclos. Como era de esperar, no hay un patrón de comportamiento en los programas. A continuación describimos ciertas características que pueden ser observadas en los programas:

- respecto a las recurrencias, tenemos desde un programa donde el 99% del tiempo se emplea en bucles con recurrencias (BDNA) hasta un programa donde sólo el 3% del tiempo se dedica a bucles con recurrencias (FLO52).

- respecto a la técnica de *widening*, hay programas donde puede ofrecer grandes ventajas, como FLO52, donde un 89% del tiempo de ejecución se emplea en bucles que pueden alcanzar su máximo rendimiento empleando únicamente la técnica de *widening* (86% OW + 3% R+OW) y otro 10% se dedica a bucles que pueden obtener ventajas de *widening* (WbR). En el otro extremo, MG3D utiliza el 78% de su tiempo de ejecución en bucles donde sólo se puede incrementar el rendimiento usando replicación (R+OR).

- la latencia de las unidades funcionales también tiene una gran influencia. Aumentar la latencia hace aumentar los requerimientos por recurrencias, pudiendo convertir bucles *resource-bound* en *recurrence-bound* (como el programa DYFESM, que dedica el 19% de su tiempo de ejecución en bucles limitados por las recurrencias cuando tenemos una latencia de 2 ciclos, mientras que el porcentaje crece hasta el 26% cuando la latencia es de 3 ciclos y hasta el 38% cuando la latencia es de 4 ciclos). Otro efecto interesante que se produce al aumentar la latencia es la reclasificación de ciertos bucles. Por ejemplo, en el programa OCEAN tenemos un 6% del tiempo en R+OW y un 63% en R+WbR cuando tenemos una latencia de 2 ciclos. Con una latencia de 3 ciclos, estos porcentajes son, respectivamente, de 37% R+OW y 33% R+WbR. Ello es debido a que un bucle R+WbR puede sacar ventaja de aplicar *widening*, pero no puede alcanzar su máximo paralelismo (que viene limitado por los ciclos de la recurrencia) sólo con esta técnica.

Programa	Latencia	% tiempo de ejecución en bucles con recurrencias (# de bucles)					% tiempo de ejecución en bucles sin recurrencias (# de bucles)			
			RecB	R+OW	R+OR	R+WbR		OW	OR	WbR
ADM	2	50	20 (9)	6 (10)	22 (15)	2 (8)	50	40 (39)	0 (16)	10 (46)
	3	51	24 (13)	5 (7)	21 (15)	1 (7)	49	39 (39)	0 (16)	10 (46)
	4	53	27 (15)	5 (6)	20 (15)	1 (6)	47	37 (39)	0 (16)	10 (46)
QCD	2	56	0 (2)	1 (3)	0 (50)	55 (3)	44	43 (20)	1 (4)	0 (7)
	3	56	0 (2)	1 (3)	0 (50)	55 (3)	44	43 (20)	1 (4)	0 (7)
	4	56	0 (2)	56 (4)	0 (50)	0 (2)	44	43 (20)	1 (4)	0 (7)
MDG	2	39	1 (6)	37 (3)	0 (1)	1 (4)	61	20 (11)	0 (0)	41 (6)
	3	39	1 (7)	38 (4)	0 (0)	0 (3)	61	20 (11)	0 (0)	41 (6)
	4	39	1 (7)	38 (4)	0 (0)	0 (3)	61	20 (11)	0 (0)	41 (6)
TRACK	2	44	1 (2)	8 (2)	1 (2)	34 (6)	56	55 (32)	0 (1)	1 (4)
	3	45	1 (2)	24 (3)	1 (2)	19 (5)	55	54 (32)	0 (1)	1 (4)
	4	45	1 (2)	36 (4)	1 (2)	7 (4)	55	54 (32)	0 (1)	1 (4)
BDNA	2	99	4 (50)	0 (6)	16 (7)	79 (28)	1	1 (41)	0 (6)	0 (7)
	3	99	5 (51)	0 (6)	16 (7)	79 (27)	1	1 (41)	0 (6)	0 (7)
	4	99	6 (59)	2 (11)	16 (7)	76 (14)	1	1 (41)	0 (6)	0 (7)
OCEAN	2	74	2 (6)	6 (2)	3 (1)	63 (14)	26	8 (22)	1 (9)	17 (19)
	3	75	2 (6)	37 (5)	3 (1)	33 (11)	25	7 (22)	1 (9)	17 (19)
	4	75	3 (6)	36 (5)	3 (1)	33 (11)	25	7 (22)	1 (9)	17 (19)
DYFESM	2	28	19 (25)	1 (11)	1 (4)	7 (16)	72	70 (20)	1 (12)	1 (14)
	3	34	26 (31)	1 (7)	1 (4)	6 (14)	66	65 (20)	0 (12)	1 (14)
	4	40	38 (34)	1 (8)	1 (4)	0 (10)	60	59 (20)	0 (12)	1 (14)
MG3D	2	79	1 (4)	0 (1)	78 (40)	0 (6)	21	19 (21)	0 (3)	2 (4)
	3	79	1 (4)	0 (1)	78 (40)	0 (6)	21	19 (21)	0 (3)	2 (4)
	4	79	1 (4)	0 (1)	78 (40)	0 (6)	21	19 (21)	0 (3)	2 (4)
ARC2D	2	5	1 (2)	3 (4)	1 (2)	0 (5)	95	67 (50)	5 (13)	23 (63)
	3	5	1 (6)	3 (3)	1 (2)	0 (2)	95	67 (50)	5 (13)	23 (63)
	4	5	1 (6)	3 (3)	1 (2)	0 (2)	95	67 (50)	5 (13)	23 (63)
FLO52	2	3	0 (0)	3 (5)	0 (0)	0 (5)	97	86 (61)	1 (3)	10 (6)
	3	4	0 (1)	4 (5)	0 (0)	0 (4)	96	86 (61)	0 (3)	10 (6)
	4	4	0 (1)	4 (6)	0 (0)	0 (3)	96	86 (61)	0 (3)	10 (6)
TRFD	2	5	0 (0)	1 (2)	1 (2)	3 (3)	95	1 (8)	0 (0)	94 (10)
	3	6	0 (0)	4 (3)	1 (2)	1 (2)	94	0 (8)	0 (0)	94 (10)
	4	6	0 (0)	4 (3)	1 (2)	1 (2)	94	0 (8)	0 (0)	94 (10)
SPEC77	2	69	2 (16)	2 (15)	49 (16)	16 (29)	31	6 (52)	7 (44)	18 (53)
	3	69	3 (20)	16 (29)	49 (16)	1 (11)	31	6 (52)	7 (44)	18 (53)
	4	69	16 (40)	4 (12)	49 (16)	0 (8)	31	6 (52)	7 (44)	18 (53)

Tabla 4.2: Clasificación de los bucles del juego de pruebas

Sin embargo, al aumentar la latencia y por tanto incrementar el límite debido a las recurrencias, puede alcanzar su paralelismo máximo (que ahora es menor) utilizando solamente *widening*.

En los párrafos anteriores hemos remarcado ciertos efectos. En el capítulo 5 puede encontrarse un estudio más detallado sobre el rendimiento.

3.4 Unidades funcionales complejas para bucles limitados por las recurrencias

El progreso es la habilidad del hombre para hacer complejo lo que es sencillo.
Thor Heyerdahl.

En este trabajo se ha estudiado un tipo de unidad funcional compleja: aquella que implementa la operación *Fussed Multiply-and-Add* (FMA). Esta operación realiza una multiplicación y una suma asociada en una única instrucción. Por ejemplo, la operación R1=R2*R3+R4, que se resolvería en dos instrucciones en una fpu normal (MULf R1, R2, R3; ADDf R1, R1, R4), puede resolverse en una única instrucción (FMA R1, R2, R3, R4).

Esta operación ha sido implementada en las unidades funcionales de coma flotante de los microprocesadores IBM RS/6000 [IBM90] y POWER2 [HFH94, WhDh94] y en los MIPS R8000 [Hsu94] y R10000 [Hei94, Yea96]. La idea principal es encadenar el multiplicador y el sumador dentro de la propia unidad de coma flotante (FPU).

En algunas máquinas, la operación FMA no reduce el número de ciclos requerido para resolver las dos operaciones; por ejemplo, el MIPS R10000 requiere 2 ciclos para efectuar tanto la suma como la multiplicación, mientras que necesita 4 ciclos para realizar la operación FMA. La ventaja que explota esta máquina es el hecho de tener una instrucción menos, de manera que se ahorra un fetch, una decodificación e incrementa el número de instrucciones que han sido emitidas (*issued*) por ciclo.

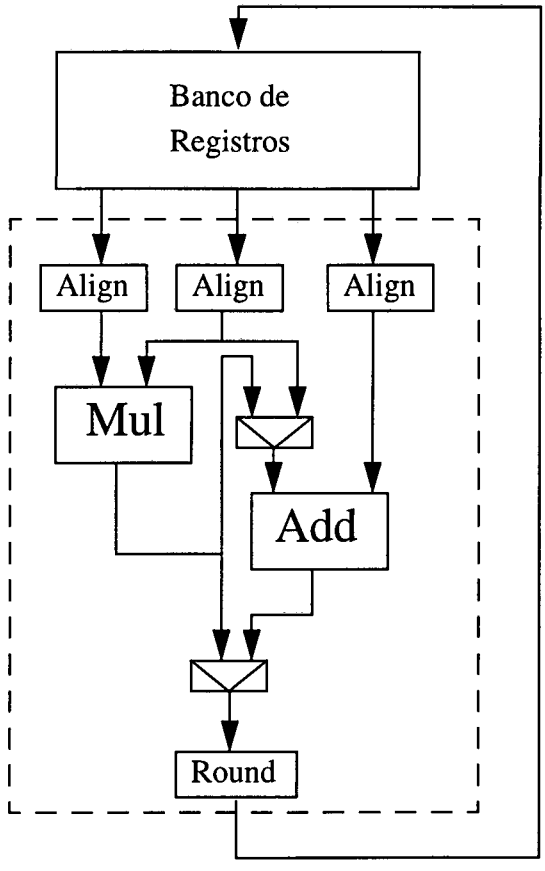


Figura 3.8: Esquema básico de una unidad fussed multiply-and-add

En cambio, el IBM POWER2 realiza las tres operaciones (suma, multiplicación y FMA) en el mismo número de ciclos: dos. La figura 3.8 muestra el esquema básico de la unidad del POWER2. Esta FPU consigue el mismo tiempo de ciclo debido a que los datos sólo se alinean una vez, y el resultado sólo se redondea una vez; además, la suma empieza a realizarse conforme va disponiendo de bits válidos de la multiplicación, sin que haga falta tener todo el resultado.

Si la operación FMA requiere menos ciclos que la suma de las latencias de las dos operaciones originales, y éstas están dentro de una recurrencia, los ciclos requeridos para

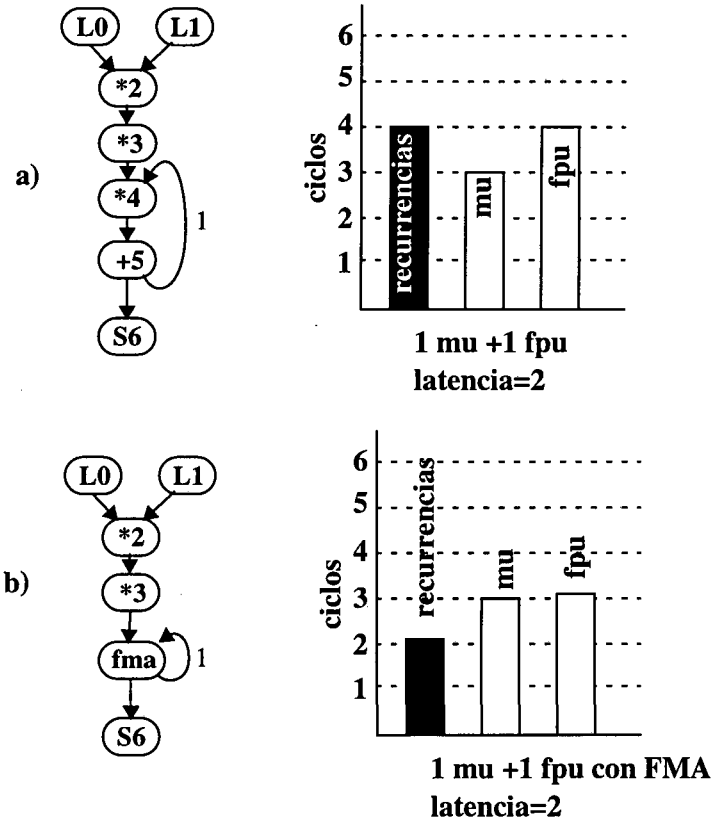


Figura 3.9:Grafo de dependencias y límites para a) Una fpu sin FMA y b) Una fpu con FMA

ejecutar la recurrencia disminuyen. Así, si el bucle estaba limitado por recurrencias, conseguimos decrementar su *RecMII*.

La figura 3.9 muestra un bucle ejemplo con dos operaciones, multiplicación y suma, asociadas dentro de una recurrencia. La figura 3.9a muestra los límites para una configuración con una FPU de latencia 2 que no implementa FMA mientras que la figura 3.9b muestra los límites para una configuración con una FPU que implementa la operación FMA, también con latencia 2. Puede observarse que la primera configuración es recurrence-bound, mientras que la segunda es compute-bound. Un efecto lateral del uso de FPUs que implementan la operación FMA es que, al reducir el número de operaciones, las limitaciones debidas al uso de las

unidades aritméticas también se reduce, como en la figura 3.9 donde el límite debido a operaciones aritméticas pasa de 4 a 3 ciclos. Asimismo, la operación FMA no debe guardar el resultado de la multiplicación en un registro para su uso en la suma posterior, por lo que también tendría un impacto en los requerimientos de registros.

La tabla 4.3 muestra el número de bucles del juego de pruebas que tienen operaciones suma y multiplicación que pueden ser encadenadas y el tiempo de ejecución sobre el total de dichos bucles. Como puede observarse, la operación FMA se da muy a menudo, por lo que el impacto de una FPU que la implemente sobre el rendimiento final será importante (como se verá en el capítulo 7).

Programa	# de bucles con FMA	% del tiempo gastado en estos bucles
ADM	88	85
QCD	7	28
MDG	17	62
TRACK	15	46
BDNA	46	96
OCEAN	30	58
DYFESM	43	99
MG3D	51	78
ARC2D	106	91
FLO52	47	83
TRFD	10	92
SPEC77	131	94

Tabla 4.3: Impacto de la operación fussed multiply-and-add en el juego de pruebas

3.5 Sumario y contribuciones

En este capítulo hemos visto una visión global del contenido de este trabajo. Hemos presentado los factores que limitan el ILP de bucles planificados con técnicas de segmentación software, que son los recursos de la arquitectura y las recurrencias de los bucles.

Para incrementar el rendimiento de bucles limitados por los recursos se debe incrementar el número de operaciones que se pueden ejecutar por ciclo. Este incremento ha sido

implementado habitualmente por la técnica de replicación, que consiste en incrementar el número de recursos (con unos grandes costes, como hemos visto en el capítulo). Una alternativa de menor coste es usar la técnica de ensanchamiento o *widening*, que consiste en no incrementar el número de recursos, sino incrementar el número de operaciones que pueden ejecutarse por recurso. Esta técnica ha sido utilizada en los buses de IBM POWER2 pero con un grado de *widening* pequeño (grado 2). Nosotros proponemos aplicarlo en grados mayores tanto a los buses como al banco de registros y a las FPUs.

Respecto a los bucles limitados por recurrencias, para incrementar su rendimiento debe reducirse el número de ciclos necesario para computar la recurrencia. Esto puede lograrse decrementando la latencia de las unidades funcionales (lo que está más allá del ámbito de esta tesis) o bien resolviendo operaciones más complejas en las unidades funcionales. Para ello proponemos estudiar unidades funcionales que implementen la operación *Fussed Multiply-and-Add*, que computa una multiplicación y una suma asociada como una única operación. Este tipo de unidades ya han sido implementadas en diversos microprocesadores.

Las contribuciones principales de este capítulo son:

- Propuesta de aplicación de la técnica *widening* a diversos recursos de la parte de coma flotante de un procesador VLIW (buses, banco de registros y FPUs), con diversos grados de *widening*.
- Clasificación de bucles en función de su comportamiento ante las técnicas estudiadas, así como la caracterización de nuestro juego de pruebas.
- Propuesta de combinar las técnicas anteriores con unidades funcionales que implementen la operación *Fussed Multiply-and-Add* para aumentar el rendimiento de bucles limitados tanto por recursos como por recurrencias.