UNIVERSIDAD DE ZARAGOZA

Tesis Doctoral

Zaragoza, Octubre de 2009

Renombre de Registros Especulativo

Jesús Alastruey Benedé

Directores

Dra. Teresa Monreal Arnal
Dr. Mateo Valero Cortés

Universitat Politècnica de Catalunya

Dr. Víctor Viñals Yúfera

Universidad de Zaragoza

DEPARTAMENTO DE INFORMÁTICA E INGENIERÍA DE SISTEMAS

Renombre de Registros Especulativo

Memoria presentada por

Jesús Alastruey Benedé

para obtener el título de Doctor por la Universidad de Zaragoza

Departamento de Informática e Ingeniería de Sistemas

Zaragoza, Octubre de 2009

Directores

Dra. Teresa Monreal Arnal
Dr. Mateo Valero Cortés

Universitat Politècnica de Catalunya

Dr. Víctor Viñals Yúfera

Universidad de Zaragoza

Departamento de Informática e Ingeniería de Sistemas

Universidad de Zaragoza

Agradecimientos

A mi familia, mi padre, mi madre y mi hermano, por todo el apoyo que me han brindado siempre y por tantas otras cosas que no se pueden resumir en un par de líneas.

A Natalia, por su cariño y su comprensión a los largos días delante de la pantalla, a las miradas perdidas cuando estábamos juntos y a las conversaciones con alguien que estaba bastante lejos.

A Teresa, entre otras muchas virtudes, su trabajo sistemático me ha ayudado a ordenar las ideas en mi cabeza.

A Víctor, su creatividad buscando soluciones y su claridad y precisión explicando sus ideas han logrado desatascar este trabajo varias veces.

A Mateo, por sus ánimos y su trato cercano.

A Kike, por toda su ayuda, especialmente con el trabajo de simulación.

A Darío, con quien he descubierto el placer por el detalle y por su trabajo con SPEC 2006.

A Pablo, por su capacidad para hacer las cosas fáciles.

A los miembros del gaZ, con quienes he compartido viajes.

A Guti, por todo el soporte de simulación, desde la gestión de los nodos hasta la escritura de scripts para procesar resultados.

A Antonio Sanz, por su dedicación a hermes.

A Natalia, mi compañera de despacho, por aguantarme a diario.

A José Luis y Jorge, con quienes comparto miles de kilómetros al año.

Y a otros tantos que no puedo recordar en este momento.

Agradecimientos

Este trabajo ha sido financiado por distintas administraciones e instituciones a través de proyectos de investigación:

- Gobierno de Aragón
  - gaZ: Grupo Consolidado de Investigación
- Ministerio de Ciencia y Tecnología / Ministerio de Ciencia e Innovación
  - Computación de Altas Prestaciones III. Jerarquía de Memoria de Altas Prestaciones. CICYT-TIC-2001-0995.
  - Computación de Altas Prestaciones IV. Jerarquía de Memoria de Altas Prestaciones. CICYT-TIN2004-07739.
  - Jerarquías de Memoria de Alto Rendimiento. CICYT-TIN2007-66423.
  - TIN2007-60625 (Barcelona)
- Unión Europea
  - HiPEAC: European Union Network of Excellence on High-Performance Embedded Architecture and Compilation
  - HiPEAC-2: European Union Network of Excellence on High-Performance Embedded Architectures and Compilers, FP7/ICT 217068

Asimismo, se ha contado con los recursos del Departamento de Informática e Ingeniería de Sistemas de la Universidad de Zaragoza (DIIS) y del Instituto de Ingeniería de Aragón (I3A).

Índice

Agradecimientos

Resumen 1

CAPÍTULO 1

Introducción 3

1.1 Procesador con ejecución fuera de orden **3**
1.2 Problemática del Banco de Registros **5**
1.2.1 Modificación de la organización y estructura del Banco de Registros 7
1.2.2 Modificación del mecanismo de renombre de registros 10
1.2.3 Modificación de la política de distribución de registros 11
1.3 Contribuciones de la Tesis **12**
1.4 Organización de la Memoria **13**
1.5 Referencias **14**

CAPÍTULO 2

Estado del Arte 19

2.1 Introducción **19**
2.1.1 Dependencias falsas de datos 19
2.1.2 Gestión de interrupciones 20
2.1.3 Renombre de Registros 21
2.2 Implementación del mecanismo de renombre de registros **22**
2.2.1 Ubicación de los almacenes de renombre 22
2.2.2 Ubicación de la correspondencia entre registros lógicos y almacenes de renombre 24
2.2.3 Recuperación frente a fallos de predicción de salto y excepciones 25
2.2.4 Renombre de Registros convencional 28
2.2.5 Estructuras para el renombre de registros convencional 29
2.2.6 Ineficiencias del renombre de registros convencional 30
2.3 Modificaciones al renombre de registros **32**
2.3.1 Renombre basado en Reorder Buffer y Registros de Renombre 32
2.4 Modificaciones al Renombre Convencional de Registros **33**
2.4.1 Asignación de registros 33
2.4.2 Liberación de registros 38
2.4.3 Recuperación tras saltos mal predichos y excepciones 43
2.4.4 Cache del Banco de Registros 43

2.4.5 Otras propuestas 44
2.5 Contribuciones 45
2.6 Referencias 47

CAPÍTULO 3

Metodología y Entorno de Experimentación 51

3.1 Introducción 51
3.2 Procesador base 52
3.2.1 Renombre de registros 52
3.2.2 Núcleo fuera de orden (back-end) 53
3.2.3 Parámetros del procesador modelado 53
3.3 Simuladores 54
3.3.1 simplescalar 54
3.3.2 smtsim 55
3.4 Cargas de Trabajo 55
3.5 Métricas 57
3.6 Coste de una implementación 58
3.7 Plataformas de Simulación 58
3.8 Referencias 59

CAPÍTULO 4

Renombre de Registros Especulativo basado en Predicción de Último Uso 61

4.1 Introducción 61
4.2 Renombre de Registros Especulativo 64
4.3 Renombre Especulativo basado en la Predicción de Último Uso 65
4.4 Último Uso: Definiciones y Caracterización 65
4.4.1 Patrones de uso 66
4.4.2 Frecuencia dinámica de patrones de uso 66
4.4.3 Predecibilidad de los patrones de uso 67
4.5 Predicción de Último Uso (LUP) 68
4.5.1 Características de la predicción de Último Uso 68
4.5.2 Efectos de la ejecución fuera de orden y la especulación de control en la predicción de último uso 69
4.6 Banco de registros auxiliar (XRF) 71
4.7 Potencial del Renombre de Registros Especulativo 72
4.7.1 Sensibilidad de las prestaciones a la política de gestión de UUs 73
4.7.2 Desglose de la ganancia de prestaciones por OPRA y ERPR 74
4.7.3 Sensibilidad de las prestaciones al tamaño del XRF 74
4.7.4 Comparativa XRF vs. Checkpointed Register File 76
4.7.5 Reducción del número de entradas del PRF 77
4.8 Conclusiones 79
4.9 Referencias 80

CAPÍTULO 5

Soporte Microarquitectónico para Renombre de Registros Especulativo 81

5.1 Introducción 81
5.2 Microarquitectura para Renombre de Registros Especulativo 83
5.2.1 Idea del Renombre Especulativo 83
5.2.2 Gestión de dependencias y de localización de registros 83
5.2.3 Liberación de Registros Virtuales, Físicos y Auxiliares 86
5.3 Predictor de Último Uso 87
5.3.1 SLUP: Sticky Last Use Predictor 88
5.3.2 DULUP: Degree of Use Last-Use Predictor 89
5.3.3 Evaluación de los predictores SLUP y DULUP 90
5.4 Sensibilidad de las prestaciones de SR-LUP al diseño del XRF 93
5.4.1 Efectos de limitar el número de entradas del XRF 93
5.4.2 Efectos de limitar el número de puertos del XRF 93
5.4.3 Selección del número de entradas y puertos del XRF 94
5.5 Prestaciones de SR-LUP (IPC e IPS) 95
5.5.1 Costes de implementación 96
5.6 Conclusiones 97
5.7 Referencias 98

CAPÍTULO 6

Implementación de un Predictor de Último Uso con Decaimiento . . . 99

6.1 Introducción 99
6.2 Implementación de Estructuras Decaídas 101
6.2.1 Decaimiento con celdas SRAM-6T 102
6.2.2 Decaimiento con Celdas RAM-4T 102
6.3 Predictor de Último Uso Sticky Decaído 103
6.3.1 Último Uso de Registros 103
6.3.2 Predictor de Último Uso Sticky (SLUP) 104
6.3.3 Diseños de SLUP decaído 104
6.4 Evaluación de SLUPd-6T y SLUPd-4T 106
6.4.1 Entorno Experimental 106
6.4.2 Sensibilidad al intervalo de decaimiento 106
6.4.3 Comparación de implementaciones de SLUP 108
6.5 Conclusiones 111
6.6 Referencias 112

CAPÍTULO 7

Selección del Tamaño del Banco de Registros y de la Política de Asignación de Recursos en Procesadores SMT 115

7.1 Introducción 115
7.2 Políticas de Asignación de Recursos 117

7.3 Experimentación **121**
7.3.1 Modelo de procesador 121
7.3.2 Implementación de las políticas de asignación de recursos evaluadas 122
7.3.3 Cargas de trabajo 122
7.3.4 Métricas 123
7.4 Resultados **124**
7.4.1 IPS 124
7.4.2 Hmean-wIPC 126
7.4.3 Selección del tamaño del PRF y de la política de asignación de recursos 126
7.5 Conclusiones **129**
7.6 Referencias **129**

CAPÍTULO 8

Conclusiones y Líneas Abiertas 131

8.1 Conclusiones **131**
8.1.1 Potencial del Renombre Especulativo 132
8.1.2 Microarquitectura con soporte de Renombre de Registros Especulativo 132
8.1.3 Aplicación de Técnicas de Decaimiento en el Diseño de Predictores Sticky 132
8.1.4 Gestión del Banco de Registros Físico en Procesadores SMT 133
8.1.5 Otras contribuciones 133
8.2 Líneas abiertas **133**
8.3 Referencias **134**

ANEXO A

Análisis de Patrones de Uso en SPEC CPU2000 135

ANEXO B

Aspectos del Diseño de Predictores de Último Uso 141

ANEXO C

Detalles del Diseño del Banco de Registros Auxiliar 145

Listado de Referencias 149

Resumen

La ejecución de instrucciones fuera de orden aumenta el rendimiento de los procesadores de altas prestaciones pero también complica la gestión del Banco de Registros Físico (Physical Register File, PRF). La técnica del renombre de registros solventa los riesgos asociados a este agresivo modelo de ejecución y permite soportar especulación de control e interrupciones precisas. Sin embargo, el renombre convencional no hace un uso eficiente de los registros físicos ya que, durante gran parte del tiempo, éstos contienen valores que no van a ser leídos. En consecuencia, los procesadores requieren más registros de los estrictamente necesarios para almacenar los valores que van a ser leídos en el futuro. Este sobredimensionamiento del PRF puede aumentar su tiempo de acceso a dos ciclos, o limitar la frecuencia del procesador, con el consiguiente efecto negativo en el rendimiento del procesador. Además, cada vez son necesarios bancos de registros con más entradas para aprovechar el paralelismo a nivel de instrucción. En los modernos procesadores Simultaneous Multithreading (SMT) este requerimiento se ve agravado por la necesidad de almacenar los valores generados por varios hilos de ejecución.

El principal objetivo de esta Tesis es proponer nuevas estrategias de renombre de registros que mejoren la gestión del PRF y así aumentar su rendimiento en términos de prestaciones, velocidad, consumo o área. En concreto, se pretende mejorar la utilización de los registros, reduciendo el tiempo que un registro físico permanece asignado a uno lógico. Existen trabajos previos en esta misma línea, pero ninguno ha llevado esta mejora al límite de reutilizar un registro físico tan pronto como no tenga consumidores pendientes. Esta Tesis busca ese límite mediante una nueva estrategia de renombre basada en la predicción del último uso de un registro físico. Este trabajo analiza la viabilidad de este nuevo tipo de predicción y propone dos diseños de predictores de último uso. De forma original, a uno de los diseños se le aplican técnicas de decaimiento (decay) para reducir su consumo estático de energía y a la vez tratar de mejorar sus prestaciones aprovechando su carácter "pegajoso" (sticky). Para explotar la predicción de último uso, se propone una microarquitectura capaz de soportar políticas de omisión de asignación y de liberación anticipada de registros físicos. Se detallan los cambios microarquitectónicos necesarios para la correcta gestión de dependencias, localización de operandos y recuperación por fallo de predicción. La microarquitectura propuesta también puede soportar otro tipo de políticas de renombre especulativo. Finalmente, como paso previo a la aplicación de estas ideas a procesadores SMT, se propone un procedimiento para obtener la política más eficiente distribuyendo los registros físicos entre los distintos threads en ejecución.

Palabras Clave: Microarquitectura, Jerarquía de Memoria, Banco de Registros, Predictor de Último Uso, Omisión de asignación, Liberación anticipada, Consumo Estático de Energía, Decay, Políticas de asignación de recursos en procesadores SMT.

CAPÍTULO 1

Introducción

El objetivo de este capítulo es presentar el contexto de la Tesis. Para ello, primero se describe el procesador con ejecución fuera de orden. Después, se detalla la parte de este procesador en la que esta centrada la Tesis: el Banco de Registros. A continuación se exponen los problemas a los que se enfrenta el diseñador del Banco de Registros, y que motivan las propuestas de esta Tesis. Finalmente se resumen las contribuciones del trabajo y se describe la organización de esta Memoria.

1.1 Procesador con ejecución fuera de orden

Los primeros procesadores implementaban un modelo de ejecución secuencial. No procesaban una instrucción hasta haber completado la anterior. Posteriormente, en la década de los 80, se utiliza la segmentación para aumentar el rendimiento. De esta forma, comienza a explotarse el paralelismo a nivel de instrucción (ILP por sus siglas en inglés), que puede definirse como el potencial de solapar la ejecución de distintas instrucciones [HePa03]. Así, surgen los procesadores escalares, capaces de iniciar una instrucción por ciclo, aunque las dependencias de control (saltos) y las de datos (productor-consumidor) reducen el número efectivo de instrucciones ejecutadas por ciclo, y por tanto, las prestaciones. Durante estos años, los arquitectos trabajan en técnicas que permitan acercarse al límite de una instrucción ejecutada por ciclo.

En los años 90 se produce un avance en la explotación de ILP con la aparición de los procesadores superescalares, capaces de buscar y ejecutar más de una instrucción por ciclo. El agrupamiento de las instrucciones difiere según el tipo de máquina. En los procesadores superescalares con planificación dinámica [HePa03], el *hardware* decide en tiempo de ejecución qué instrucciones se ejecutan cada ciclo, mientras que las máquinas VLIW optan por una solución estática basada en *software* en la que los grupos de instrucciones a ejecutar se determinan en tiempo de compilación [ShLi04]. La solución dinámica, basada en el hardware, es la preferida por la mayoría de fabricantes.

En los años siguientes, los avances tecnológicos (capacidad de integración) y arquitectónicos (microarquitecturas mejoradas) propician la aparición de procesadores superescalares con capacidad para ejecutar instrucciones fuera de orden[1] y de forma especulativa.

La ejecución fuera de orden permite que una instrucción preparada para ejecutarse adelante a otra previa que está detenida, modificando por tanto el orden original del programa. De esta forma, se puede explotar más ILP, y por tanto, mejorar el rendimiento, aunque a costa de aumentar la complejidad del procesador. Además de otros recursos, se necesita una estructura de reordenación (*Reorder Buffer*, ROB) para que las instrucciones actualicen el estado del procesador en orden de programa. De esta manera, se gestionan las excepciones de forma precisa, es decir, de forma consistente con el modelo de ejecución secuencial. También se requieren técnicas que eliminen las dependencias falsas de datos (*scoreboard* [Tho64], Tomasulo [Tom67]), bastante frecuentes en este agresivo modelo de ejecución.

La ejecución especulativa es otra técnica utilizada en los procesadores para aumentar las prestaciones. Se predice el destino de un salto condicional y se ejecutan las instrucciones de dicha rama para evitar la latencia asociada a la resolución del salto. En caso de que la predicción sea fallida, hay que restaurar el estado del procesador en la instrucción de salto y retomar la ejecución por la rama correcta.

En la década actual, los procesadores superescalares utilizan recursos hardware cada vez más complejos para seguir extrayendo ILP, pero obtienen incrementos de rendimiento moderados. La razón de esta desaceleración en la ganancia de prestaciones es la dificultad para encontrar más instrucciones independientes en un único flujo de ejecución. El ILP ya no parece suficiente para mantener activas las cada vez más numerosas unidades funcionales del procesador. Así, se propone explotar el paralelismo a nivel de thread (*Thread Level Paralellism*, TLP). El aprovechamiento de esta nueva fuente de paralelismo da lugar a la aparición de dos nuevos paradigmas microarquitectónicos donde se invierten las cada vez mayores cantidades de transistores disponibles[2]: múltiples threads simultáneos (*Simultaneous Multithreading,* SMT) [TEL95] y multiprocesadores en un chip o procesadores multinúcleo (*Chip MultiProcessing,* CMP*,* o *multicore)* [HNO97].

Los procesadores SMT extienden la ejecución superescalar mediante la ejecución de instrucciones de distintos threads en el mismo ciclo. La compartición de recursos del procesador por los distintos threads permite mejorar la productividad, sin embargo, también puede ralentizar algún thread respecto su ejecución en solitario por problemas de contención en el uso de alguna estructura. Varios procesadores comerciales han aplicado este modelo -Intel Pentium 4 (*Netburst*) [MBH+02], Intel Core (*Nehalem*), IBM Power4 [TDF+02], Power5 [KST03], Power6[3], Fujitsu SPARC64 VII-, siempre con soporte para 2 threads excepto en el caso del recientemente anunciado IBM Power7, que es capaz de ejecutar 4 threads.

1. Anteriormente, sólo procesadores escalares para *mainframes* (CDC6600, IBM 360/91) eran capaces de ejecutar instrucciones fuera de orden.
2. Más de 1.700 millones de transistores en el caso del Intel Itanium 2 Montecito.
3. Procesador con ejecución fuera de orden limitada a las instrucciones de coma flotante.

Los procesadores multinúcleo (*multicore)* integran varios núcleos de ejecución en el mismo chip, por lo que se incrementa la productividad (*throughput*). Este modelo de microarquitectura ha tenido aplicación industrial en procesadores fuera de orden comerciales, primero en el formato de doble-núcleo (IBM Power4 [TDF+02], Power5 [KST03]; Fujitsu SPARC64 VI; AMD Opteron; Intel Pentium 4, Core, Itanium2 Montecito; HP-PA 8800 y 8900), y más recientemente en modelos con cuatro núcleos (Fujitsu SPARC64 VII, AMD Opteron K10, Intel Core) o incluso con 6 (Intel Xeon, serie Dunnington de la microarquitectura Core). Existen también procesadores con 8 y 16 núcleos (Ultra T1, Ultra T2), pero no soportan ejecución fuera de orden, su objetivo principal es la productividad, renunciando en parte a las prestaciones en la ejecución de un thread en favor de la reducción del consumo de energía.

Los dos paradigmas descritos pueden combinarse, de hecho, existen procesadores CMP en los que cada núcleo superescalar fuera de orden puede ejecutar varios threads simultáneamente. Es el caso del IBM Power4 [TDF+02], Power5 [KST03]; Fujitsu SPARC64 VII.

Esta Tesis se centra en los procesadores superescalares fuera de orden, en concreto se propone mejorar una parte del mecanismo que gestiona la ejecución fuera de orden: el renombre de registros. El objetivo de la propuesta es mejorar la eficiencia de las políticas de asignación y liberación de registros físico. También se analizan diversas estrategias de gestión del banco de registros en procesadores SMT como preparación antes de aplicar un mecanismo alternativo de renombre de registros en este tipo de procesadores.

1.2 Problemática del Banco de Registros

El Banco de Registros (*Register File*, RF) puede considerarse como el primer nivel en la jerarquía de memoria en cuanto a los datos se refiere. Almacena los resultados generados por todas las instrucciones que producen un valor.

Con el objetivo de poder explotar más ILP, cada nueva generación de procesadores soporta más instrucciones en vuelo y tiene un mayor ancho de emisión que la anterior, por lo que requiere generalmente RFs con un mayor número de entradas y de puertos. Ambas tendencias puede comprobarse en la Tabla 1.1. Además, los modernos procesadores *Simultaneous Multithreading* (SMT) necesitan muchas entradas para almacenar los valores generados por varios hilos de ejecución y muchos puertos para dar salida a las instrucciones preparadas de todos los hilos. Un ejemplo que ilustra esta última afirmación es el de la familia IBM Power. Incorporar la capacidad de ejecución SMT en el Power5 causó un considerable incremento en el número de entradas de los RFs respecto a los de su predecesor, el Power4 (de 80 a 120 entradas para el RF entero y de 72 a 120 entradas en el RF de coma flotante, ver Tabla 1.1). En el caso del Intel Itanium, el soporte de ejecución SMT en el núcleo Montecito aumentó sensiblemente el número de entradas del RF entero (de 128 a 144, ver Tabla 1.1), pero afectó de forma más importante al número de puertos (de 8 a 10 para el RF entero y de 6 a 10 en el RF de coma flotante).

Este incremento en el número de entradas y de puertos afecta negativamente a la latencia, consumo y área de un RF que tiene que integrarse en procesadores con frecuencias altas de funcionamiento y restricciones térmicas [RDK+00]. Más detalladamente, según el modelo de Rixner y para una tecnología de 180 nm, el área es proporcional al número de registros y al cuadrado

Tabla 1.1 Tamaños de los Bancos de Registros de distintos procesadores comerciales. En el caso de procesadores multinúcleo se refiere a número de registros en cada núcleo. nd: no disponible.

Procesador	Año	Banco de Registros Entero		Banco de Registros de Coma Flotante		Estructura de Reordenación	
		#reg. físicos	#puertos	#reg. físicos	#puertos	#entradas	nombre
MIPS R10000	1996	64 x 64b	7L + 3E	64 x 64b	5L + 3E	32	*Active List*
MIPS R12000	1998					48	*Active List*
Alpha 21264	1998	(80 x 64b)x2[a]	(4L + 6E)x2[a]	72 x 64b	6L + 4E	80	*In-Flight Window*
AMD K7 [GHS+99][GoPa99]	1999	24 x 32b	9L + 8E	88 x 90b	5L +5E	72[b]	*Instruction Control Unit*
Intel Pentium4	2000	128 x 32b[c]	nd[d]	128[e]	nd	126	*Reorder Buffer*
IBM Power4 [TDF+02][BMK+05]	2001	80	8L + 4E	72 x 64b	12L + 4E	100[f]	*Group Completion Table*
Intel Itanium 2 [FeOr02]	2002	128 x 65b	12L + 8E[g]	128 x 82b	8L + 6E	-	-
AMD K8 [Vri03]	2003	40[h]	9L + 8E	120[i]	5L + 5E	72[b]	*Reorder Buffer*
IBM Power5 [KST04]	2004	120	¿8L + 4E?	120 x 64b	¿12L + 4E?	100[f]	*Global Completion Table*
Intel Itanium 2 Montecito [FDL+05]	2005	144 x 65b	12L +10E	128 x 82b	8L + 10E	-	-
Intel Core	2006	nd	nd	nd	nd	96[b]	*Reorder Buffer*
Fujitsu SPARC VI	2007	(156x2[j] + 32[k] + 48[l]) x 64b	8L + 2E (CWR)	(64x2[j]+ 48[k]) x 64b	¿?L + 4E (FPR)	64	*Commit Stack Entry*
Fujitsu SPARC VII	2008	(156x2[j] + 48[k]) x 64b	nd		nd		
Fujitsu SPARC VIIIfx	2009?	(188[j] + 32[k]) x 64b	nd	(256[j] + 48x2[k]) x 64b	nd	48	

a. Replicado para reducir el número de puertos.

b. Almacena los resultados generados por las instrucciones enteras en vuelo.

c. El acceso al banco de registros está completamente segmentado y cuesta dos ciclos.

d. Funciona al doble de velocidad para reducir el número de puertos a la mitad [Vri00].

e. Registros del ISA x87, MMX, SSE y SSE-II.

f. Puede almacenar 20 grupos de hasta 5 instrucciones.

g. Físicamente hay 12 puertos (*wordlines*), 4 de lectura y 8 que pueden hacer una lectura y una escritura en el mismo ciclo.

h. Banco de Registros de la Arquitectura (ARF, 16 registros de la arquitectura, valores consolidados) + *Future File* (16 registros, valores más recientes de cada registro) + 8 registros para almacenar resultados generados por microcódigo.

i. Banco de registros mezclado (PRF) que combina registros de la arquitectura y de renombre (SS2, SSE, x87, 3DNow, MMX).

j. Registros de la ventana de SPARC (GPR -int- y FPR -fp-), repartidos en 2 bancos en el caso del SPARC VI y VII.

k. Registros de renombre para ejecución fuera de orden: *General Register Update Buffer* (GUB) para enteros y *Floating-Point Register Update Buffer* (FUB) para coma flotante.

l. *Current Window Register*: subconjunto de los registros de la ventana (GPR)

de puertos, el término dominante de la latencia de acceso crece con el número de puertos y la raíz cuadrada del número de entradas, y la energía consumida por acceso aumenta con el cuadrado de los puertos y el número de entradas. A modo de ejemplo, la Figura 1.1a muestra la variación de la latencia y del consumo de energía con el número de entradas para un RF con 15 puertos, y la Figura 1.1b muestra la variación de esas dos mismas magnitudes al aumentar el número de puertos en un RF de 80 entradas.

El acceso al RF puede segmentarse a lo largo de varios ciclos. De esta forma es posible mantener su tasa de lecturas y escrituras (*throughput*) sin degradar la frecuencia del procesador. Sin embargo, las etapas añadidas degradan las prestaciones ya que incrementan uno de los bucles más críticos del procesador [BTM+02], amplificando los efectos negativos causados por los fallos de especulación de control (predicción de saltos) y por la planificación especulativa (predicción de latencia) [HSU+01]. Además, la segmentación del acceso al RF complica el diseño de la red de cortocircuitos (*bypass network*).

Por todo lo anterior, se plantea un interesante compromiso en el diseño de un procesador en cuanto al dimensionado del RF. Por una parte, se requieren grandes RF para explotar el paralelismo a nivel de instrucción (ILP) y en el caso procesadores SMT también para explotar el paralelismo a nivel de thread (TLP) y así obtener altas prestaciones medidas en instrucciones por ciclo (IPC). Por otra parte, RFs más ajustados pueden permitir mayores frecuencias del procesador, lo que resulta en mayores prestaciones en términos de instrucciones por segundo (IPS). Además, ocupan menos área y consumen menos energía. A lo largo de esta Tesis se analiza con detalle el dimensionado del RF en procesadores superescalares sin y con capacidad SMT.

Toda esta problemática asociada a los grandes bancos de registros ha incentivado la investigación de mejoras en el diseño y gestión de esta estructura. Las propuestas pueden agruparse en tres grandes líneas que se van a detallar en las siguientes subsecciones:

- Modificación de la organización y estructura del banco de registros.
- Modificación del mecanismo de renombre de registros.
- Modificación de la política de distribución de registros en procesadores SMT.

Las dos primeras líneas están centradas en los RFs de procesadores superescalares con ejecución fuera de orden. La última es de aplicación únicamente a procesadores *Simultaneous Multithreading* (SMT).

1.2.1 Modificación de la organización y estructura del Banco de Registros

En lo que a esta línea se refiere, la industria ha optado por reducir el número de puertos. La Tabla 1.1 recoge varias soluciones en este sentido:

- Alpha 21264: el banco de registros enteros es replicado. De esta forma, en lugar de un RF monolítico con 8 puertos de lectura se tienen dos RFs con 4 puertos.
- Intel Pentium 4: el RF funciona al doble de la frecuencia del procesador para reducir el número de puertos a la mitad.

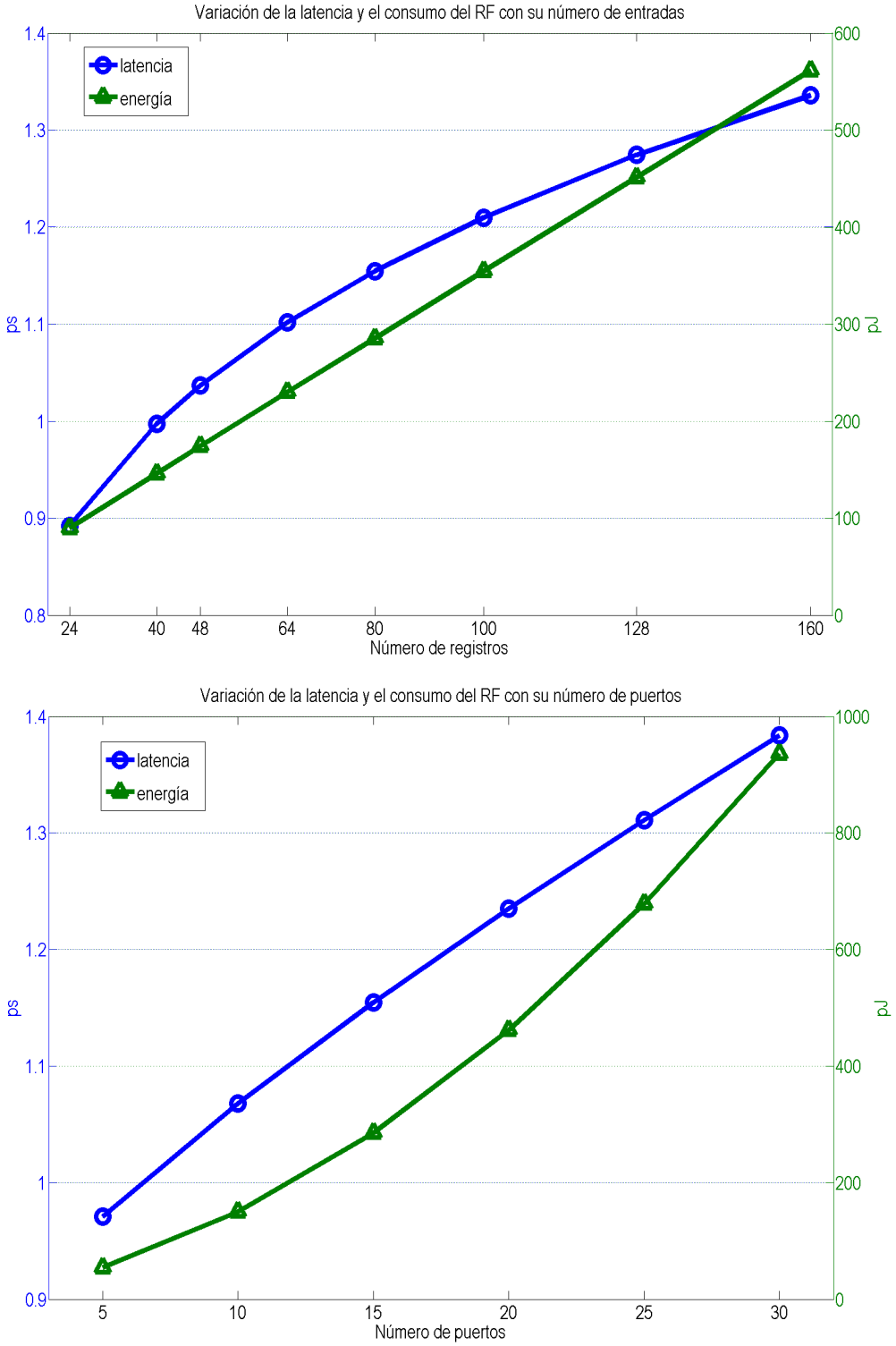


Figura 1.1 Variación de la latencia y del consumo de energía por acceso al banco de registros con el número de registros (arriba) y con el número de puertos (abajo) según el modelo de Rixner [RDK+00]. En la gráfica superior, el número de puertos (constante) es de 15, en la gráfica inferior, el número de registros (constante) es de 80.

- Intel Itanium 2: 8 de los 12 puertos existentes pueden realizar una lectura y una escritura en el mismo ciclo.

Desde el ámbito académico también se han propuesto alternativas de diseño orientadas a reducir el tiempo efectivo de acceso al banco de registros, su área y su consumo.

Un primer grupo de trabajos plantea microarquitecturas distribuidas [SBV95][FCJ+97][PJS97][ZyKo01][STR02][BTM+02]. Por ejemplo, la arquitectura Multicluster particiona la ruta de datos en varios *clusters*, cada uno compuesto por una unidad funcional y un RF que contiene un subconjunto de los registros lógicos [FCJ+97]. Estos RFs particionados tienen menor latencia, consumo y área que el RF monolítico original ya que requieren menos entradas y puertos. Borch y Tune plantean otra arquitectura basada en *clusters*, pero en este caso las unidades funcionales están ligadas a pequeñas caches de un RF principal que es reubicado en el camino de decodificación a lanzamiento, de esta forma se reduce el camino crítico en las etapas de lanzamiento a ejecución [BTM+02]. Estas microarquitecturas distribuidas requieren una compleja lógica de control para distribuir las instrucciones entre los clusters y para gestionar las dependencias entre distintos clusters.

Otra serie de aportaciones mantiene una microarquitectura centralizada y reducen el número de puertos del RF a través de otro tipo de estrategias como la modificación de la estructura interna del RF (por ejemplo, dividiéndolo en varios bancos entrelazados con menos puertos por banco [WaBa96][BDA01b][TsAs03]), la modificación de las etapas del segmentado dedicadas a la lectura de operandos (por ejemplo, dedicando un ciclo a la identificación de operandos que se van a leer de la red de cortocircuitos [PPV02]) o añadiendo pequeñas estructuras auxiliares que reducen la actividad del RF principal (por ejemplo, colas de escritura retardadas y buffers de operandos en [KiMu03]).

Otra línea de trabajos plantea cambios más profundos en la organización del RF, como por ejemplo, jerarquías de varios niveles que gestionan sus contenidos por exclusión [BDA01a], varios RFs con anchuras adaptadas a distintos tipos de datos [GCO+04] o diseños con varios RFs basados en las arquitecturas con *Future File* [PGC+04][Rei05].

Finalmente, un grupo de diseños añade una pequeña estructura para que trabaje como cache de un RF convencional [CGV+00][BuSo04][ZeGh06]. De esta forma, se producen menos accesos al RF y puede reducirse su número de puertos. La diferencia principal entre estas propuestas radica en la gestión de contenidos. En la primera, se escriben en la cache aquellos valores que no han sido leídos de la red de cortocircuitos [CGV+00]. En la segunda, la cache contiene los registros físicos para los que se haya predicho más referencias pendientes [BuSo04]. La inserción y el reemplazo se controlan por medio de un predictor de grado de uso (referencias) que es muy preciso para registros con un reducido número de consumidores [BuSo02b]. En la tercera, la cache está dividida en dos estructuras, una que contiene los últimos valores generados y otra que retiene los valores de los registros físicos más recientemente leídos (política de reemplazo LRU) [ZeGh06]. Además, en la primera propuesta, cuando se produce un fallo de esta cache de registros, se copia el valor del registro solicitado desde el RF principal, pero con menos puertos que el original. En las otras, el RF suministra directamente a la unidad funcional correspondiente el valor del registro físico referenciado.

1.2.2 Modificación del mecanismo de renombre de registros

Los procesadores con ejecución fuera de orden renombran los registros lógicos que determinan el flujo de datos de un programa a unos buffers de renombre o almacenes físicos implementados en la máquina. Los objetivos del renombre de registros son eliminar las dependencias falsas de datos (*Write-After-Read* y *Write-After-Write*) que limitan artificialmente la capacidad de un procesador para extraer paralelismo, y proporcionar un mecanismo que permita recuperar el estado preciso en caso de que la especulación de control ejecute instrucciones de camino incorrecto. El renombre de registros se describe detalladamente en el Capítulo 2.

La mayoría de los procesadores comerciales actuales implementan alguna variante del mecanismo de renombre de registros. Básicamente, las distintas implementaciones difieren en:

- La estructura donde se almacenan las versiones especulativas de los registros lógicos, que puede ser la estructura de reordenación (*Reorder Buffer*, ROB), un banco de registros de renombre (*Rename Register File*, RRF) o un banco de registros físico mezclado o centralizado (*Physical Register File*, PRF).

- La ubicación de las correspondencias entre registros lógicos y almacenes de renombre, por ejemplo en una tabla de mapeo (*Mapping Table*, MT), un banco de registros especulativo (*Future File*, FF), o junto a los almacenes de renombre

- El mecanismo de recuperación por especulación de control o excepciones, que puede estar basado en la salvaguarda de tablas de mapeo (*checkpointing)*, en una tabla de mapeo consolidada (RRAT), o en la reversión de las acciones de renombre (*roll-back*)

- La granularidad a la hora de hacer la asignación y la liberación de registros físicos, a nivel de instrucción o de grupo de instrucciones.

El Capítulo 2 analiza distintas variantes en la implementación del mecanismo de renombre de registros y su utilización en procesadores comerciales. Esta Tesis se centra en la denominada implementación de renombre convencional de registros, que se detalla en el Capítulo 2.

Los procesadores actuales necesitan más registros de los estrictamente necesarios para almacenar los valores que se van a leer en el futuro (versiones vivas) ya que el mecanismo de renombre convencional es conservador. Un registro físico es asignado demasiado pronto, varios ciclos antes de que vaya a contener un valor, mientras que su liberación se realiza demasiado tarde, puede que muchos ciclos después de haber sido leído por última vez.

Por otra parte, varios estudios han observado que existe redundancia en los valores almacenados en el banco de registros (valores repetidos, valores con pocos bits significativos), por lo que existen oportunidades para aprovechar mejor su capacidad de almacenamiento.

Desde el ámbito académico se han planteado modificaciones al renombre convencional, tanto en la parte de asignación como en la de liberación de registros. En unos casos, su objetivo es reducir el tiempo que un registro físico permanece asignado a uno lógico, en otros, el objetivo es evitar la redundancia de los valores almacenados en el banco de registros. En ambos casos, el propósito es reducir el número de registros físicos necesario para evitar las detenciones por no

disponer de registros físicos en el renombre. Esto puede conseguirse liberando anticipadamente los registros físicos [MRF97][BDA01b][MVG+02][MRH+02][ARS03][LMG04][KoNa05][JOA+05][BSP+06a][EBP+06][ShPo07], retrasando [WaBa96][MGV+99a] o incluso omitiendo su asignación [BuSo02b][BSP+06b], compartiendo registro físico por parte de distintas instrucciones [JRB+98][TuSe99][PBR02][BaSo03][TNN+04][PSR05] o robando los registros físicos a instrucciones que están bloqueadas por un fallo de memoria de latencia larga [SRA+04]. De esta manera puede reducirse el número de registros físicos mediante una gestión más eficiente de los mismos. El Capítulo 2 describe las aportaciones más relevantes de esta línea de investigación. El núcleo de esta Tesis se integra dentro de este conjunto de propuestas.

1.2.3 Modificación de la política de distribución de registros

Los modernos procesadores *Simultaneous Multithreading* (SMT) tratan de mejorar la utilización de sus estructuras mediante la ejecución simultánea de varios threads que comparten determinados recursos de cálculo y almacenamiento [KoMa03][KST04]. Entre estos recursos se encuentra el RF, una estructura especialmente crítica ya que debe almacenar los estados consolidados y especulativos de todos los threads, lo que requiere un elevado número de entradas.

Para evitar RFs muy grandes, y en especial, sus efectos adversos asociados, pueden aplicarse las propuestas de diseño expuestas en las subsecciones anteriores (organización y renombre). Además, en un procesador SMT puede mejorarse la eficiencia de su RF actuando sobre la política de asignación de recursos, la cual determina el modo en que los recursos compartidos del procesador se distribuyen entre los threads que se ejecutan. Al igual que en un grupo de propuestas de la subsección anterior, el objetivo es reducir el tiempo que un registro físico permanece asignado a uno lógico, pero en este caso, en lugar de buscarlo modificando las políticas de asignación y liberación de registros, se puede conseguir mediante un reparto de registros entre los threads que favorezca su reutilización, por ejemplo, priorizando a los threads de alto IPC.

Los procesadores SMT comerciales implementan mecanismos más bien sencillos de distribución de los recursos compartidos entre los distintos threads [KoMa03][KST04]. Por otra parte, desde el ámbito académico se han propuesto políticas más elaboradas con el objetivo de mejorar la utilización de recursos [TEE+96][TuBr01][CFR+03][RaRe03][CRV+04b][ChYe06][EyEc07][SPL08]. Uno de los objetivos de estas estrategias es realizar un reparto de recursos adecuado cuando aparecen loads de latencia larga. Esta situación es la que peor gestionan políticas sencillas como *Round Robin* o *Icount*, ya que permiten que un thread detenido acumule recursos que van a tardar en ser liberados, impidiendo su utilización por parte de otros threads activos [TEE+96]. Para evitar este problema, los procesadores comerciales imponen topes al uso de recursos por parte de un thread o incluso particionan determinadas estructuras, práctica que en algunos casos puede limitar las prestaciones. El Capítulo 7 recoge las aportaciones más relevantes de esta línea de investigación, y señala que no se ha publicado ningún análisis sobre el efecto combinado en el rendimiento de la política de asignación de recursos y del tamaño del PRF. Un grupo de contribuciones de esta Tesis se integra en esta línea de trabajo.

1.3 Contribuciones de la Tesis

En esta Tesis se ha diseñado un mecanismo de renombre de registros especulativo que permite reducir el tiempo en que un registro físico permanece asignado a uno lógico. Este mecanismo se basa en unas políticas de asignación y liberación de registros físicos que utilizan la predicción de último uso de un registro para mejorar la gestión del RF. De esta forma puede reducirse el número de registros, y por tanto, el tiempo de acceso al RF. Se ha detallado asimismo una microarquitectura capaz de soportar este nuevo mecanismo de renombre especulativo. También se ha analizado una serie de políticas de asignación de recursos en procesadores SMT buscando el tamaño del RF que maximiza sus prestaciones.

El primer conjunto de contribuciones hace referencia al concepto de *patrón de uso* de una instrucción y su aplicación al renombre de registros. El Capítulo 4 define este concepto y propone aplicar la predicción de patrones último uso para mejorar el mecanismo de renombre de registros. Como pasos previos a esta aplicación, se verifica la viabilidad de la predicción del patrón de uso de una instrucción y se analiza el potencial de la idea. Este trabajo se publicó en [AMV+06].

El segundo grupo de contribuciones está relacionado con el Renombre de Registros Especulativo, en concreto, con el soporte microarquitectónico para la ejecución de políticas de renombre de registros basadas en predicción que permiten la omisión de asignación y la liberación anticipada de registros físicos. El Capítulo 5 describe las estructuras hardware necesarias para implementar dichas políticas y los cambios en la gestión de dependencias, localización de operandos y las acciones de recuperación ligadas a la predicción. Creemos que esta es la primera propuesta de una microarquitectura que soporta cualquier política de omisión de asignación y de liberación anticipada de registros físicos. Para evaluar esta nueva microarquitectura se han diseñado dos predictores de último uso de complejidad y prestaciones incrementales que controlan políticas de asignación y liberación de registros basadas en predicción de último uso de un registro. Estas ideas se publicaron en [AMV+07].

La tercera contribución principal es la aplicación de técnicas de decaimiento (*decay*) en el diseño de un predictor *sticky* de último uso. Añadiendo transistores de paso (*gated-Vdd* [PYF+00]) o construyendo dicho predictor con celdas RAM cuasi-estáticas de 4 transistores [HJD+02] se puede reducir el consumo y área del predictor sin afectar a sus prestaciones, o incluso mejorándolas. Estas ideas están publicadas en [AMV+09].

Como antesala a la aplicación del Renombre de Registros Especulativo a procesadores SMT, se ha analizado la sensibilidad del rendimiento de un procesador SMT al tamaño del PRF y a la política de reparto de registros físicos entre threads. Creemos que este es el primer análisis que realiza una comparación entre las políticas de asignación más importantes para un amplio rango de tamaños del PRF. También se propone un procedimiento sencillo que, dada una política de asignación de recursos, selecciona un tamaño de PRF de acuerdo con un compromiso entre productividad y equidad. Este trabajo se publicó en [AMC+08].

Finalmente, y como contribuciones secundarias, puede destacarse la clasificación de las aportaciones más relevantes en el campo de la gestión del banco de registros desde una perspectiva diferente a la tomada por los trabajos previos (Capítulo 2). También se ha modificado el simu-

lador sim-outorder de simplescalar para modelar un procesador con renombre de registros convencional y con las políticas de renombre especulativo basadas en la predicción de último uso de un registro. Por último, se han modelado con detalle distintas políticas de asignación de recursos en el simulador smtsim, algunas muy recientes.

1.4 Organización de la Memoria

La Memoria de esta Tesis tiene la siguiente estructura:

- El Capítulo 1 presenta primero el objeto de estudio en esta Tesis: el procesador con ejecución fuera de orden, más en concreto, la problemática asociada a la evolución de uno de sus componentes clave: el banco de registros (*Register File*, RF). Después describe las dos grandes líneas de trabajo centradas en el RF: (a) modificación de la organización y/o estructura y (b) modificación de la gestión del RF. Tras situar esta Tesis en la segunda línea, se resumen las contribuciones de este trabajo y se listan las publicaciones que las recogen.

- En el Capítulo 2 se describe el Estado del Arte en el campo de la gestión del banco de registros. Se detalla el renombre de registros convencional (línea base que se adopta a lo largo de esta Tesis) y las diversas alternativas que se han planteado.

- El Capítulo 3 muestra el entorno experimental utilizado en la evaluación de las propuestas planteadas: se detallan las características del modelo de procesador, las cargas de trabajo utilizadas y las métricas empleadas.

- En el Capítulo 4 se presenta el renombre de registros basado en predicción de último uso. Primero se analizan las características de este nuevo tipo de predicción con el objetivo de mostrar su viabilidad. Después se analiza el potencial de esta alternativa al renombre de registros convencional.

- El Capítulo 5 propone una microarquitectura que soporta mecanismos arbitrarios de Renombre Especulativo. En primer lugar detalla las estructuras hardware necesarias para implementar políticas especulativas de asignación y liberación de registros. Después se diseñan dos predictores de último uso que se utilizan para evaluar las prestaciones de un tipo de Renombre Especulativo sobre la microarquitectura propuesta. Finalmente se realiza un análisis para reducir su complejidad.

- En el Capítulo 6 se diseña un predictor de último uso de baja complejidad. Aplicando técnicas de decaimiento a dicho componente puede reducirse tanto su consumo energético como su área. Se han evaluado dos implementaciones decaídas de un predictor de último uso de registro, una basada en la conmutación de alimentación (*gated-Vdd* en celdas SRAM-6T) y otra basada en celdas de memoria cuasi-estáticas (RAM-4T).

- El Capítulo 7 analiza distintas políticas de distribución de recursos entre threads y su interacción con el tamaño del banco de registros y el número de threads en ejecución. También se propone un procedimiento para seleccionar el tamaño de banco de registros que maximiza las prestaciones para una política de asignación de recursos dada y para facilitar la com-

paración de prestaciones entre distintas políticas. Este trabajo prepara la aplicación del renombre especulativo a procesadores SMT.

- Finalmente, el Capítulo 8 presenta las conclusiones de la Tesis y sugiere líneas de trabajo futuro.

1.5 Referencias

[ARS03] H. Akkary, R. Rajwar, S.Srinivasan. Checkpoint Processing and Recovery: Towards Scalable Large Instruction Window Processors. Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2003). pp. 423. Diciembre 2003.

[AMV+06] J. Alastruey, T. Monreal, V. Viñals and M. Valero. "Speculative Early Register Release". *Proc. of the 3rd conference on Computing frontiers* (CF), May 2006, pp 291-302.

[AMV+07] J. Alastruey, T. Monreal, V. Viñals and M. Valero. "Microarchitectural Support for Speculative Register Renaming". *IEEE Int. Parallel & Distributed Processing Symp.* (IPDPS), March 2007.

[AMC+08] J. Alastruey, T. Monreal, F.J. Cazorla, V. Viñals y M. Valero. Selection of the Register File Size and the Resource Allocation Policy on SMT Processors. *Proceedings of the 20th International Symposium on Computer Architecture and High Performance Computing* (SBAC-PAD), Campo Grande, Brasil, Oct 29-Nov 1, 2008. IEEE Computer Society, pp. 63-70, ISBN 978-0-7695-3423-7.

[AMV+09] J. Alastruey, T. Monreal, M. Valero y V. Viñals. "Implementación de un Predictor de Último Uso con Decaimiento". XX Jornadas de Paralelismo, La Coruña (España). 17-18 Sept. 2009.

[BaSo03] S. Balakrishnan and G. Sohi. Exploiting Value Locality in Physical Register Files. *In Proceedings of the 36th Annual IEEE/ACM international Symposium on Microarchitecture* (MICRO 03), pp. 265. San Diego (USA), December 03- 05, 2003.

[BDA01a] R. Balasubramonian, S. Dwarkadas, and D.H. Albonesi, Dynamically Allocating Processor Resources between Nearby and Distant ILP. *Proceedings of the 28th International Symposium on Computer Architecture* (ISCA 01), pp. 26-37, June 2001.

[BDA01b] R. Balasubramonian, S. Dwarkadas, and D. H. Albonesi, Reducing the Complexity of the Register File in Dynamic Superscalar Processors. *Proceedings of the 34th International Symposium on Microarchitecture* (MICRO 01), pp. 237-249, December 2001.

[BSP+06a] D. Balkan, J. Sharkey, D. Ponomarev and A. Aggarwal, "Address-Value Decoupling for Early Register Deallocation", Proc. 35th Int'l Conf. on Parallel Processing (ICPP-06), Aug. 2006, pp. 337-346.

[BSP+06b] D. Balkan, J. Sharkey, D. Ponomarev and K. Ghose, "SPARTAN: Speculative Avoidance of Register Allocation to Transient Values for Performance and Energy Efficiency", *Proc. 15th Int'l Conf. Parallel Architectures and Compilation Techniques (PACT 06)*, Sept. 2006, pp. 265-274.

[BTM+02] E. Borch, E. Tune, S. Manne and J. Emer, "Loose Loops Sink Chips", *Proc. 8th Int'l Symp. High-Performance Computer Architecture (HPCA 02)*, Feb. 2002, pp. 299-310.

[BMK+05] T. N. Buti, , R. G. McDonald, Z. Khwaja, A. Ambekar, H. Q. Le, W. E. Burky, and B. Williams. Organization and implementation of the register-renaming mapper for out-of-order IBM POWER4 processors. *IBM J. Res. Dev. 49, 1* , pp. 167-188, Jan. 2005

[BuSo02a] J.A. Butts and G. Sohi, "Dynamic dead-instruction detection and elimination", *Proc. 10th Int'l Conf. Architectural Support for Programing Languages and Operating Systems (ASPLOS 02)*, Oct. 2002, pp. 199-210.

[BuSo02b] J.A. Butts and G. Sohi, "Characterizing and Predicting Value Degree of Use". Proc. 35th Int'l Symp. Microarchitecture (MICRO 02), Nov. 2002, pp. 15-26.

[BuSo04] J.A. Butts and G. Sohi, “Use-Based Register Caching with Decoupled Indexing”, Proc. 31st Int’l Symp. Computer Architecture (ISCA 04), June 2004.

[CFR+03] F.J. Cazorla, E.Fernández, A.Ramírez and M.Valero. Improving Memory Latency aware fetch policies for SMT processors. *Int’l Symp. High Performance Computing*, Oct. 2003.

[CRV+04b] F. J. Cazorla, A. Ramirez, M. Valero and E. Fernandez. Dynamically Controlled Resource Allocation in SMT Processors. *Int’l Symp. on Microarchitecture (MICRO)*, pp. 171–182. IEEE Computer Society, December 2004.

[ChYe06] S.Choi and D.Yeung. Learning-Based SMT Processor Resource Distribution via Hill-Climbing. *Int’l Symp. on Computer Architecture (ISCA)*, 2006, pp. 239-251.

[CGV+00] J. Cruz, A. González, M. Valero, and N.P. Topham, Multiple-Banked Register File Architectures. *Proceedings of the 27th International Symposium on Computer Architecture* (ISCA 00), pp. 316-325, June 2000.

[ElAl03] A.El-Moursy and D.Albonesi. Front-End Policies for Improved Issue Efficiency in SMT processors. *HPCA*, pp. 31-40, February 2003.

[EBP+06] O. Ergin, D. Balkan, D. Ponomarev and K. Ghose, “Early Register Deallocation Mechanisms Using Checkpointed Register Files”, *IEEE Transactions on Computers*, vol. 55, no. 9, Sept. 2006, pp. 1153-1166.

[EyEc07] S.Eyerman, L.Eeckhout. A Memory-Level Parallelism Aware Fetch Policy for SMT Processors, *HPCA*, 2007, pp. 240-249.

[FCJ+97] K. Farkas, P. Chow, N. Jouppi, and Z. Vranesic, The Multicluster Architecture: Reducing Cycle Time Through Partitioning. *Proceedings of the 30th International Symposium on Microarchitecture* (MICRO97), pp. 149-159, December 1997.

[FeOr02] E. Fetzer and T. Orton, “A Fully-Bypassed 6-Issue Integer Datapath and Register File on an Itanium Microprocessor”. ISSCC Dig. Tech. Papers, pp. 418-419, 2002.

[FDL+05] E. Fetzer, D. Dahle, C. Little and Kevin Safford. The Multi-Threaded, Parity-Protected, 128-Word Register Files on a Dual-Core Itanium®-Family Processor. ISSCC Dig. Tech. Papers, Paper 20.5, pp. 382-383, Feb., 2005.

[GHS+99] M. Golden, S. Hesley, A. Scherer, M. Crowley, S.C. Johnson, S. Meier, D. Meyer, J.D.Moench, S. Oberman, H. Partovi, F. Weber, S. White, T. Wood, J. Yong. A seventh-generation x86 microprocessor. *IEEE Journal of Solid-State Circuits*, Volume 34, Issue 11, Nov. 1999, pp. 1466-1477.

[GoPa99] M. Golden, H. Partovi, A 500 MHz, write-bypassed, 88-entry, 90-bit register file. *Symposium on VLSI Circuits, Digest of Technical Papers*, pp. 105-108, 1999.

[GCO+04] R. González, A. Cristal, D. Ortega, A. Veidenbaum and M. Valero. A Content Aware Integer Register File Organization. *In Proceedings of the 31st Annual international Symposium on Computer Architecture* (ISCA 04), p. 314. München, Germany, June 19-23, 2004.

[HNO97] L. Hammand, B. A. Nayfeh, and K. Olukotun. A single-chip multiprocessor. *IEEE Computer*, 30(9):79–85, Sep. 1997.

[HePa03] J.L. Hennessy, and D.A. Patterson. *Computer Architecture, A Quantitative Approach.* Morgan Kaufmann Publishers, San Francisco, Third Edition 2003.

[HSU+01] G. Hinton, D. Sager, M. Upton, D. Boggs, D. Carmean, A. Kyker, and P. Roussel.. The Microarchitecture of the Pentium 4 Processor. *Intel Technology Journal Q1*, February 2001.

[HJD+02] Z.Hu, P. Juang, P. Diodato, S. Kaxiras, Kevin Skadron, M. Martonosi, D.W. Clark. “Managing Leakage for Transient Data: Decay and Quasi-Static Memory Cells”. ISLPED, 2002.

[JOA+05] T.M. Jones, M.F.P. O'Boyle, J. Abella, A. González, and O. Ergin, "Compiler Directed Early Register Release", *Proc. 14th Int'l Conf. on Parallel Architectures and Compilation Techniques* (PACT 05), Sept. 2005, pp. 110-122.

[JRB+98] S. Jourdan, R. Ronen, M. Bekerman, B. Shomar, A. Yoaz. A Novel Renaming Scheme to Exploit Value Temporal Locality through Physical Register Reuse and Unification", *in Proc. of 31st. Int. Symp. on Microarchitecture* (MICRO 98), pp. 216-225, 1998.

[KST03] R. Kalla, B. Sinharoy, and J. Tendler. Power5: Ibm's next generation power microprocessor. *Proc. 15th Hot Chips Symp*, pp 292–303, August 2003.

[KST04] R. Kalla, B. Sinharoy, and J. Tendler. IBM Power5 Chip: A Dual-Core Multithreaded Processor. *IEEE Micro*, vol. 24, no. 2, pp. 40-47, Mar./Apr. 2004.

[KiMu03] N. Kim, and T. Mudge, T. Reducing register ports using delayed write-back queues and operand pre-fetch. *In Proceedings of the 17th Annual international Conference on Supercomputing* (ICS 03). San Francisco (USA), June 23-26, 2003.

[KoNa05] M. Kondo and H. Nakamura. A Small, Fast and Low-Power Register File by Bit-Partitioning. *In Proceedings of the 11th international Symposium on High-Performance Computer Architecture* (HPCA 2005), pp. 40-49. February 12-16, 2005.

[KoMa03] D.Koufaty, D.Marr, Hyperthreading technology in the netburst microarchitecture, *IEEE Micro,* vol 23, no 2, Mar.-Apr. 2003, pp.56-65.

[LMG04] M. Lipasti, B. Mestan and E. Gunadi, Physical Register Inlining. *In Proceedings of the 31st Annual international Symposium on Computer Architecture* (ISCA 04), pp. 325-335. München, Germany, June 19-23, 2004.

[MBH+02] D. T. Marr, F. Binns, D. L. Hill, G. Hinton, D. A. Koufaty, J. A. Miller, and M. Upton. Hyper-threading technology architecture and microarchitecture. *Intel Technology Journal*, 6(1):4–15, Feb. 2002.

[MRF97] M. Martin, A. Roth, C. Fisher. Exploiting Dead Value Information. *Proceedings of the 30th Annual International Symposium on Microarchitecture* (MICRO 97), pp. 125-135. 1997.

[MRH+02] J. Martinez, J. Renau, M. Huang, M. Prvulovich, J. Torrellas, "Cherry: Checkpointed Early Resource Recycling in Out-of-order Microprocessors", *Proc. 35nd Int'l Symp. Microarchitecture (MICRO 02)*, Nov. 2002, pp. 3-14.

[MGV+99a] T. Monreal, A. González, M. Valero, J. González, and V. Viñals, Delaying Physical Register Allocation Through Virtual-Physical Registers. *Proceedings of the 32nd International Symposium on Microarchitecture* (MICRO 99), pp. 186-192, November 1999.

[MVG+02] T. Monreal, V. Viñals, A. González and M. Valero. Hardware Schemes for Early Register Release. *Proc. Int'l Conf. Parallel Processing* (ICPP 02), Aug. 2002, pp. 5-13.

[PPV02] I. Park, M. D. Powell, and T. N. Vijaykumar. Reducing register ports for higher speed and lower energy. In MICRO-35, Istanbul, Turkey, November 2002.

[PJS97] A.S. Palacharla, N.P. Jouppi, and J.E. Smith, Complexity-Effective Superscalar Processors. *Proceedings of the 24th International Symposium on Computer Architecture* (ISCA 97), pp. 206-218, June 1997.

[PGC+04] M. Pericàs, R. Gonzalez, A. Cristal, A. Veidenbaum and M. Valero. An Optimized Front-End Physical Register File with Banking and Writeback Filtering. *Workshop on Power Aware Computer Systems*. December 2004.

[PBR02] V. Petric, A. Bracy and A. Roth. Three extensions to register integration. *In Proceedings of the 35th Annual ACM/IEEE International Symposium on Microarchitecture* (MICRO 02). Istanbul, Turkey, November 18 - 22, 2002.

[PSR05] V. Petric, T. Sha and A. Roth. RENO: a rename-based instruction optimizer. *In Proceedings of the 32nd Annual International Symposium on Computer Architecture* (ISCA 05), pp. 98-109. Madison (USA), June 4-8, 2005.

[PYF+00] M. Powell, S. Yang, B. Falsafi, K. Roy, T.N. Vijaykumar. "Gated-Vdd: a circuit technique to reduce leakage in deep-submicron cache memories". ISLPED, 2000.

[RaRe03] S.E.Raasch, S.K.Reinhardt, The impact of resource partitioning on SMT processors, *PACT*, pp. 15-25, Sept. 2003.

[Rei05] G. Reinman. Using an operand file to save energy and to decouple commit resources. IEE Proceedings: Computers and digital techniques, ISSN 1350-2387, Vol. 152, Nº 5, 2005 , pags. 666-678.

[RDK+00] S. Rixner, W.J. Dally, B. Khailany, P. Mattson, U.J. Kapasi, and J.D. Owens, Register Organization for Media Processing. P*roceedings of the 6th International Symposium on High-Performance Computer Architecture* (HPCA 00), pp. 375-386, January 2000.

[STR02] A. Seznec, E. Toullec, and O. Rochecouste. Register write specialization register read specialization: A path to complexity-effective wide-issue superscalar processors. In *MICRO-35*, Istanbul, Turkey, November 2002.

[ShPo07] J. Sharkey and D. Ponomarev. An L2-miss-driven early register deallocation for SMT processors. *In Proceedings of the 21st Annual international Conference on Supercomputing* (ICS 07), pp. 138-147. Seattle (USA), June 17-21, 2007.

[ShLi04] J. P. Shen, M. H. Lipasti. *Modern Processor Design.* McGraw-Hill, First Edition, 2004.

[SBV95] G. S. Sohi, S. Breach, and T. N. Vijaykumar. Multiscalar processors. In ISCA-22, 1995.

[SRA+04] S. Srinivasan, R. Rajwar, H. Akkary, A. Gandhi and M. Upton. Continual flow pipelines. *In Proceedings of the 11th international Conference on Architectural Support For Programming Languages and Operating Systems* (ASPLOS 04), pp. 107-119. Boston, (USA), October 07-13, 2004.

[SPL08] S.Subramaniam, M.Prvulovic and G. Loh. PEEP: Exploiting Predictability of Memory Dependences in SMT Processors. *HPCA*, pp. 137-148, Feb. 2008.

[TDF+02] J. M. Tendler, J. S. Dodson, J. S. Fields, Jr., H. Le, and B. Sinharoy, POWER4 System Microarchitecture, *IBM J. Res.& Dev. 46, No. 1, 5–25*, January 2002.

[Tho64] J.E. Thornton. Parallel operation in the Control Data 6600. *In Proc. AFIPS Fall Joint Computer Conf.* pp 33-40, 1964.

[Tom67] R.M. Tomasulo. An Efficient Algorithm for Exploiting Multiple Arithmetic Units. IBM Journal of Research and Development, vol 11, nº1, pp 25. 1967.

[TNN+04] L. Tran, N. Nelson, F. Ngai, S. Dropsho and M. Huang. Dynamically reducing pressure on the physical register file through simple register sharing. *In Proceedings of the 2004 IEEE international Symposium on Performance Analysis of Systems and Software* (ISPASS 04), pp. 78-87. March 10-12, Austin (USA), 2004.

[TsAs03] J. Tseng, and K. Asanovic. Banked multiported register files for high-frequency superscalar microprocessors. In Proceedings of the 30th Annual International Symposium on Computer Architecture (ISCA 03), pp. 62. 2003.

[TEL95] D.M. Tullsen, S. Eggers, and H. Levy, Simultaneous Multithreading: Maximizing On-Chip Parallelism. *Proceedings of the 22nd International Symposium on Computer Architecture* (ISCA 95), pp. 392-403, June 1995.

[TEE+96] D.Tullsen, S.Eggers, J.Emer, H.Levy, J.Lo, and R. Stamm. Exploiting choice: Instruction fetch and issue on an implementable simultaneous multithreading processor. *ISCA*, pp. 191–202, Apr. 1996.

[TuSe99] D. Tullsen and J. Seng. Storageless value prediction using prior register values. *In Proceedings of the 26th Annual International Symposium on Computer Architecture* (ISCA 99), pp.270-279. Atlanta (USA), May 01-04, 1999.

[TuBr01] D.Tullsen and J.Brown. Handling long-latency loads in a simultaneous multithreaded processor. *Int'l Symp. on Microarchitecture (MICRO)*, pp. 318–327. Dec. 2001.

[Vri00] H. Vries. An analysis of the newly presented Pentium 4 core, *http://www.chip-architect.com/*. Consultada el 5 de febrero de 2009.

[Vri03] H. Vries. Understanding the detailed Architecture of AMD's 64 bit Core, *http://www.chip-architect.com/*. Consultada el 5 de febrero de 2009.

[WaBa96] S. Wallace and N. Bagherzadeh. A Scalable Register File Architecture for Dynamically Scheduled Processors. *Proceedings of the 5th International Conference on Parallel Architectures and Compilation Techniques* (PACT 96), pp. 179-184, October 1996.

[ZeGh06] H. Zeng, and K. Ghose. Register file caching for energy efficiency. *In Proceedings of the 2006 international Symposium on Low Power Electronics and Design* (ISLPED 06), pp. 244-249. Tegernsee (Germany), October 04-06, 2006.

[ZyKo01] V. V. Zyuban and P. M. Kogge. Inherently lower-power high-performance superscalar architectures. *IEEE Trans. on Computers*, 50(3):268–285, March 2001.

CAPÍTULO 2

Estado del Arte

En este capítulo se presenta el Estado del Arte en el campo de la gestión del banco de registros. En primer lugar, se describe la técnica del renombre de registros y se describen las distintas implementaciones adoptadas por procesadores comerciales. A continuación, se detalla la implementación de renombre de registros basada en un banco de registros físico mezclado, escogida como línea base en esta Tesis, y se señalan sus limitaciones principales. Finalmente, se clasifican y describen las diversas alternativas que se han planteado a la estrategia de renombre de registros convencional y se sitúan en este contexto las contribuciones de esta Tesis.

2.1 Introducción

El renombre de registros es una técnica empleada para incrementar el paralelismo a nivel de instrucción (ILP) explotado en procesadores con ejecución fuera de orden. Permite eliminar las dependencias falsas de datos y gestionar las interrupciones como si de una máquina con ejecución secuencial se tratara. En las dos siguientes subsecciones se detallan estos dos problemas y en la tercera se describen de forma general las acciones asociadas al renombre de registros.

2.1.1 Dependencias falsas de datos

Uno de los últimos pasos en el proceso de compilación es la asignación de registros. En esta fase, se pasa de un lenguaje de asignación única en el que se dispone de un número infinito de registros a un lenguaje máquina con un número finito de registros (registros lógicos o de la arquitectura). Esta limitación en el número de registros conlleva la reutilización de registros durante la ejecución de un programa. En un modelo de ejecución secuencial, este reciclaje de registros no supone ningún problema, sin embargo, en procesadores segmentados y en especial, en los fuera de orden, sí que provoca riesgos por dependencias falsas de datos.

Figura 2.1 Dependencias falsas de datos: WAR (a) y WAW (b).

Las dependencias falsas de datos ocurren cuando el registro destino de una instrucción no está disponible, ya sea porque existe una instrucción previa con el mismo registro que no ha sido leído (WAR, *Write After Read*, Figura 2.1a) o porque existe una instrucción previa con el mismo registro que todavía no ha sido escrito (WAW, *Write After Write*, Figura 2.1b).

En ambos casos no existe una dependencia real entre las instrucciones i1 e i2, sino que existe un riesgo por la especificación del mismo contenedor para dos versiones distintas de un mismo registro lógico (r2 en el caso de la Figura 2.1). Si la instrucción i2 utilizara un contenedor diferente al especificado, el riesgo desaparecería.

La técnica del renombre de registros elimina las dependencias falsas de datos mediante la asociación dinámica entre los registros referenciados en las instrucciones (lógicos) y los registros realmente utilizados para almacenar los datos (físicos). De esta forma, es posible que varias versiones de un mismo registro lógico convivan simultáneamente, lo que evita posibles detenciones del procesador y aumenta su capacidad de explotar paralelismo. La sección 2.2 describe en detalle la técnica del renombre de registros utilizada por los procesadores actuales.

2.1.2 Gestión de interrupciones

Cuando se produce una interrupción, se salva el estado de un proceso, definido como el contador de programa, los registros lógicos de la arquitectura y la memoria. Una vez gestionada la interrupción, el estado del proceso se restaura para continuar su ejecución. En este contexto, se define interrupción *precisa* a aquella en la que el estado salvado es coherente con el del modelo secuencial de ejecución[1] [SmPl88].

En un procesador fuera de orden, dado que varias instrucciones se ejecutan concurrentemente, puede ocurrir que el estado del procesador se modifique en un orden diferente al de programa. Si se produce una interrupción, y se salva un estado del proceso que no se corresponde con el del modelo secuencial, se dice que la interrupción es *imprecisa*. Para algunas arquitecturas, determinados tipos de interrupciones no pueden ser imprecisas, ya que impedirían reanudar la ejecución de un proceso (por ejemplo, los fallos de página en sistemas con memoria virtual, [SmPl88]).

El método más sencillo para soportar interrupciones precisas consiste en forzar que todas las instrucciones modifiquen el estado del procesador en orden de programa aunque la ejecución de dichas instrucciones haya sido en desorden. Esto suele implementarse mediante una *estruc-*

1. En el modelo secuencial de ejecución, una instrucción debe completarse antes de iniciar la ejecución de la siguiente.

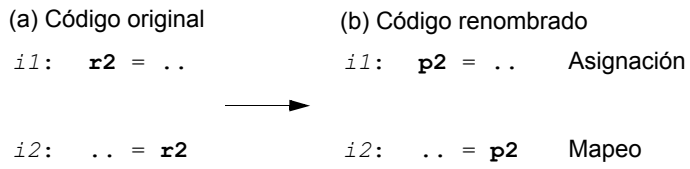


Figura 2.2 Renombre de registros lógicos destino y fuente

tura de reordenación que almacena todas las instrucciones en vuelo. Así, cuando se detecta una excepción, se permite progresar a la instrucción que la ha producido y a las previas en orden de programa. Cuando todas estas instrucciones previas se han jubilado[2], se salva el estado del procesador, que es el estado preciso asociado al contador de programa de la instrucción que ha producido la excepción. El proceso ahora puede interrumpirse para permitir la ejecución de la rutina de servicio. Para reanudar la ejecución del proceso original, se restaura el proceso salvado y se reejecuta desde la dirección correspondiente a la instrucción que produjo la excepción.

2.1.3 Renombre de Registros

El renombre de registros es una técnica que elimina dependencias falsas de datos asignando dinámicamente una ubicación física (*buffer* de renombre) al registro lógico destino de cada instrucción que genera un valor [Yea96][Gwe97][Kes99][HSU+01]. Dependiendo de la implementación, el resultado de una instrucción se puede almacenar en un banco de registros específico para renombre (*Rename Register File*, RRF), en la estructura de reordenación (*Re-Order Buffer*, ROB) o en un banco de registros físico que combina registros de la arquitectura y de renombre (*Physical Register File*, PRF) [Sim00]. Para mantener las dependencias, las instrucciones posteriores deben leer sus operandos fuente de los almacenes asignados a sus registros lógicos. Por tanto, hay que obtener el almacén de renombre asignado a los registros lógicos fuente. La Figura 2.2 muestra un ejemplo de estas dos acciones, asignación a registro lógico destino y mapeo de registro lógico fuente, que se efectúan en la etapa de renombre. Otra acción importante es la liberación de un almacén de renombre, que permite su reutilización durante la ejecución del código. Las condiciones para liberar un almacén asignado a un registro lógico dependen de la implementación concreta del mecanismo de renombre. Finalmente, es necesario articular un procedimiento de recuperación tras producirse una predicción de salto incorrecta. Hay que restaurar el estado de las estructuras hardware implicadas en el renombre deshaciendo los cambios que hayan podido realizar las instrucciones de camino incorrecto. Dependiendo de la implementación, este procedimiento puede integrarse junto al mecanismo para recuperar el estado preciso tras producirse una excepción.

En la sección 2.2.4 se explican detalladamente las acciones asociadas al renombre de registros para la implementación basada en un banco de registros físico que combina registros de la arquitectura y de renombre.

2. Una instrucción jubilada o retirada es aquélla que ha ejecutado la última etapa (*commit*) del segmentado.

2.2 Implementación del mecanismo de renombre de registros

Existen múltiples variables de diseño que hay que considerar en la implementación del mecanismo de renombre. El diseñador se enfrenta a decisiones de tipo cualitativo, como por ejemplo, dónde ubicar los almacenes de renombre, y otras de tipo cuantitativo, como por ejemplo, el dimensionado de la estructura que alberga los almacenes de renombre. Algunas de estas decisiones pueden determinar parámetros del procesador tan importantes como el tiempo de ciclo. En las siguientes subsecciones analizamos las tres variables principales del espacio de diseño: la ubicación de los almacenes de renombre, la ubicación de las correspondencias entre registros lógicos y almacenes de renombre, y el procedimiento de recuperación frente a fallos de predicción de salto y excepciones. No consideramos variables de diseño como por ejemplo, el tipo de instrucciones a las que se aplica el renombre, ya que en la actualidad todos los procesadores con ejecución fuera de orden renombran todas las instrucciones[3].

2.2.1 Ubicación de los almacenes de renombre

Existen tres alternativas principales para implementar los *buffers* de renombre:

1. Banco de Registros Físico (*Physical Register File*, PRF): la misma estructura contiene los registros de la arquitectura y los registros de renombre. Durante la ejecución del código, los registros de la arquitectura y los de renombre son dinámicamente asociados a los registros físicos. Cada registro físico del PRF puede estar en cuatro estados [Lip92]:

   a) Libre: cuando no está asociado a un registro lógico.

   b) Asociado a un registro de la arquitectura: cuando contiene el valor generado por una instrucción consolidada.

   c) Asociado a un registro de renombre, inválido: cuando todavía no se ha escrito el valor del registro lógico destino asociado.

   d) Asociado a un registro de renombre, válido: cuando ha sido escrito el valor del registro lógico destino asociado.

2. Banco de Registros de Renombre (*Rename Register File*, RRF): una estructura hardware dedicada contiene los *buffers* de renombre. Otra estructura, el banco de registros de la arquitectura (*Architected Register File*, ARF[4]) almacena los valores de los registros de la arquitectura, es decir, aquellos producidos por instrucciones jubiladas.

3. Estructura de Reordenación (*Reorder Buffer*, ROB): la estructura de reordenación almacena los resultados generados por las instrucciones en vuelo. Al igual que en el diseño anterior, existe un banco de registros de la arquitectura que contiene los valores consolidados (los generados por instrucciones ya jubiladas).

3. Los primeros procesadores que implementaron el renombre de registros lo hicieron de forma parcial. El POWER1 (1990) renombraba sólo los loads de coma flotante. Su sucesor, el POWER2 (1993), renombraba todas las instrucciones de coma flotante.

4. A esta estructura se le llama *Retirement Register File* en el Pentium III, lo cual genera confusión ya que tiene el mismo acrónimo que *Rename Register File* (Banco de Registros de Renombre).

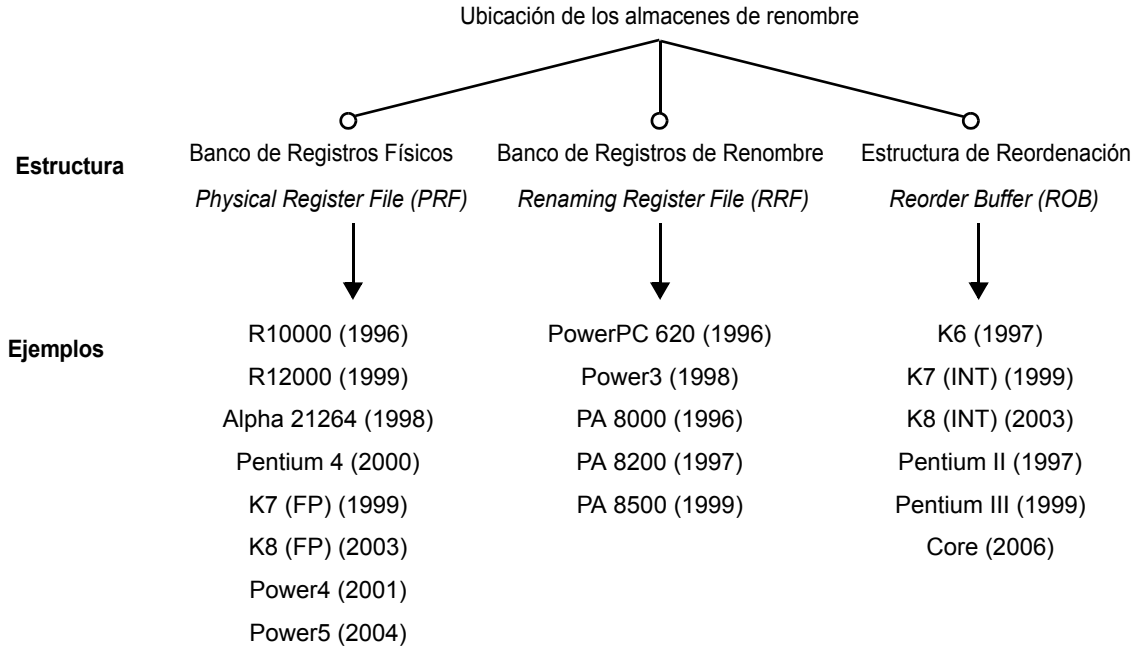


Figura 2.3 Estructuras donde se alojan los almacenes de renombre y ejemplos de su aplicación en procesadores comerciales. Diagrama adaptado de [Sim08].

Las dos primeras implementaciones utilizan distintos bancos de registros para almacenar los datos enteros y los de coma flotante. En el primer caso, existe un banco de registros físico entero y otro de coma flotante; en el segundo, existen bancos de registros de renombre y de la arquitectura separados para cada tipo de datos. Para la implementación basada en ROB, el campo reservado para la escritura del valor que produce una instrucción debe poder acomodar tanto valores enteros como de coma flotante, por lo que su anchura debe adaptarse al tipo de dato de mayor tamaño (generalmente, los datos de coma flotante).

Hay una diferencia reseñable entre la implementación basada en PRF y las otras dos. En los diseños basados en RRF y ROB, cuando una instrucción se jubila hay que transferir el resultado que ha generado desde el almacén de renombre correspondiente al banco de registros de la arquitectura (ARF). En el diseño basado en PRF se evita dicha escritura mediante el cambio de estado del registro físico que almacena dicho resultado (pasa de estar asociado a un registro de renombre -válido- a estar asociado a un registro de la arquitectura).

La Figura 2.3 recoge estas opciones de diseño y su aplicación en varios procesadores recientes. En los últimos años, los almacenes de renombre se han integrado en el PRF y en el ROB. En el caso de los AMD K7 y K8, puede observarse que han combinado ambas opciones: almacena los registros enteros renombrados en el ROB, mientras que los registros de coma flotante renombrados están alojados en el PRF junto a los registros de la arquitectura.

2.2.2 Ubicación de la correspondencia entre registros lógicos y almacenes de renombre

Las opciones principales para almacenar las asociaciones entre registros lógicos y almacenes de renombre son las siguientes:

1. Tabla de Mapeo (*Mapping Table,* MT): dado un registro lógico, permite conocer el **identificador** del almacén de renombre más reciente asociado a dicho registro lógico. Lógicamente (conceptualmente), es una tabla que indexada por registro lógico devuelve el almacén de renombre asociado. Físicamente, esta estructura puede implementarse con una memoria RAM (*Random Access Memory*, memoria de acceso aleatorio) o con una memoria CAM (*Content-Addressable Memory*, memoria direccionable por contenido). En el primer caso, la tabla tiene tantas entradas como registros lógicos, y cada entrada contiene el identificador del almacén de renombre asociado al identificador del registro lógico que selecciona dicha entrada. Esta es la opción escogida en los MIPS R10000, R12000 y el Intel Pentium 4[5]. En el caso de la CAM, la tabla tiene tantas entradas como almacenes de renombre existan, y cada entrada contiene el identificador de registro lógico (si lo hay) al que está asociado cada almacén de renombre, por lo que se requiere una búsqueda asociativa para obtener la asociación entre registro lógico y almacén de renombre. También es necesario un bit de validez por entrada ya que un registro lógico puede estar asociado a varios almacenes de renombre, pero solamente es válida la asociación más reciente. Esta implementación se encuentra en el Alpha 21264 y en los IBM Power 4 y Power5.

2. Banco de Registros Especulativo (*Future File*, *FF*): esta estructura contiene un subconjunto de los almacenes de renombre, más concretamente, los **valores** más recientes asociados a cada registro lógico [SmPl88]. Tiene, por tanto, tantas entradas como registros lógicos.

3. Integrada junto a los almacenes de renombre: junto a cada **valor** de los almacenes de renombre, se guarda el **identificador** del registro lógico al que está asociado. Hay que efectuar una búsqueda asociativa en un campo de la estructura (generalmente, *RRF* o *ROB*) para encontrar el identificador del registro lógico buscado.

La Figura 2.4 clasifica varios procesadores recientes según la opción de diseño escogida. La implementación más frecuente es mediante tabla de mapeo. Al igual que para ubicar los almacenes de renombre, los procesadores AMD K7 y K8 han optado por soluciones distintas para los datos enteros y de coma flotante: un banco de registros especulativo contiene las versiones más recientes de cada registro lógico entero mientras que una tabla de mapeo permite obtener el identificador del registro físico asociado a cada registro lógico de coma flotante. Según la información que disponemos, estos procesadores son los únicos que han implementado un *Future File* desde el año 2000. El Intel Pentium 4, además de la tabla de mapeo convencional que se actualiza en la etapa de renombre, utiliza una tabla de mapeo adicional, denominada *Retirement Register Alias Table* (RRAT), que se actualiza en la etapa de jubilación y que refleja las asociaciones consolidadas entre registros lógicos y físicos. La RRAT se utiliza para recuperar el estado de la unidad de renombre tras fallos de predicción de salto o excepciones.

5. En este procesador la MT es llamada *Register Alias Table* (RAT).

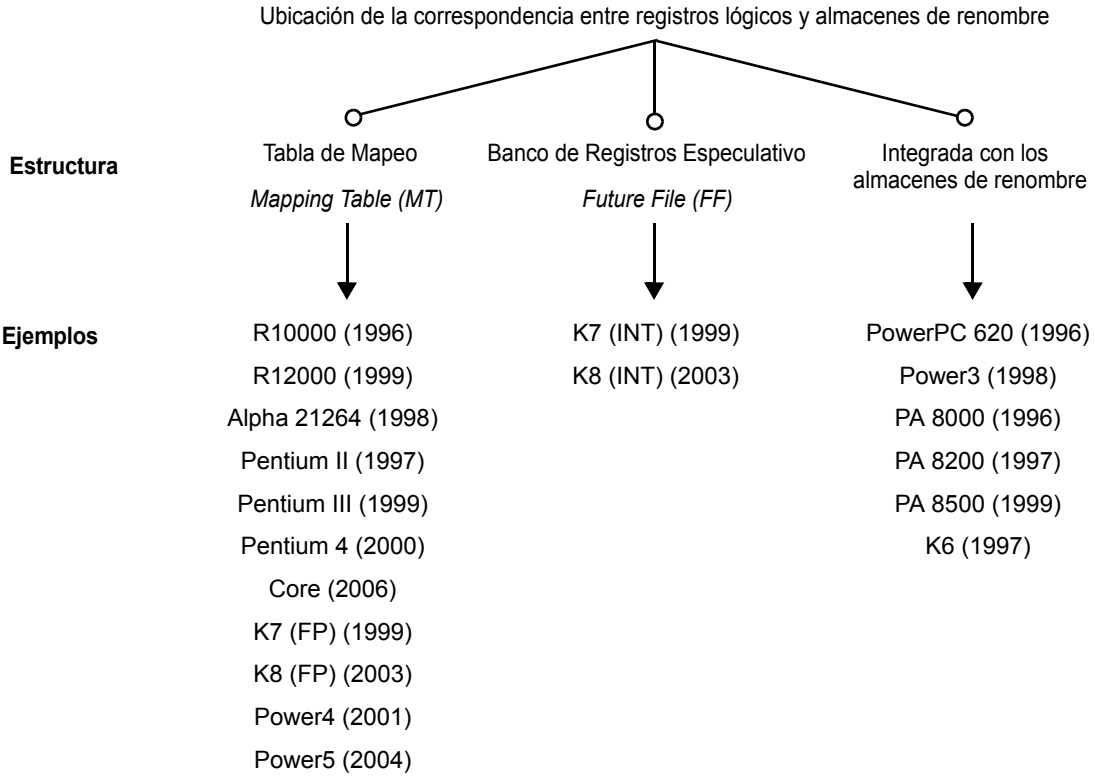


Figura 2.4 Estructuras donde se alojan las correspondencias entre registros lógicos y almacenes de renombre y ejemplos de su aplicación en procesadores comerciales. Diagrama adaptado de [Sim08].

2.2.3 Recuperación frente a fallos de predicción de salto y excepciones

Una instrucción renombrada puede ser anulada, ya sea por ser de un camino de control incorrecto (salto mal predicho) o por suceder a una instrucción que provoca una excepción. Ante tales eventos, hay que deshacer las asignaciones de almacén de renombre al registro destino de cada instrucción anulada. Por tanto, las dos tareas del mecanismo de recuperación son:

1. Recuperar las asociaciones entre registros lógicos y almacenes de renombre existentes antes de las acciones de renombre efectuadas por la instrucción anulada más vieja.
2. Liberar los almacenes de renombre asignados a todas las instrucciones anuladas.

Los métodos más comunes que permiten restaurar dichas asociaciones entre registros lógicos y almacenes de renombre son:

1. *Checkpointing*: se realizan salvaguardas (*checkpoints)* de la estructura que guarda dichas asociaciones. La restauración del estado del renombre es más rápida que en los otros métodos, pero tiene una limitación por el coste de almacenamiento que conllevan las distintas copias. Para disminuir la sobrecarga de las copias, en lugar de llevarse a cabo una

salvaguarda previa al renombre de cada instrucción, se realiza una antes de renombrar cada grupo de instrucciones (un máximo 20 *shadow maps* en el Power4) ó antes del renombre de cada salto (con un límite de 4 en el R10000 y de 20 en el 21264). Por otra parte, el uso de una CAM para la MT permite que cada salvaguarda (*checkpoint*) se realice guardando solamente los bits de validez [PJS96]. En el caso concreto del Power4, cada *shadow map* se realiza guardando dos bits de estado por registro físico [BMK+05].

2. *Roll-back desde MT especulativa*: se recorre el ROB, desde la instrucción renombrada más joven (última instrucción renombrada) hasta la instrucción más vieja que hay que anular[6]. Partiendo de la estructura que contiene las asociaciones más recientes entre registros lógicos y almacenes de renombre, se deshacen los renombres del registro destino de cada instrucción. Para ello, cada instrucción, antes de ser renombrada, guarda en la estructura de reordenación (ROB) el mapeo **previo** de su registro lógico destino (*old_pd*). Esta aproximación permite recuperar las asociaciones entre registro lógico y almacén de renombre existentes en el instante previo al renombre de **cualquier** instrucción, pero tiene una limitación en el número de instrucciones que puede procesar cada ciclo (generalmente el mismo número de instrucciones que el procesador es capaz de renombrar en un ciclo), por lo que su penalización es proporcional al número de instrucciones anuladas.

3. *Roll-back desde MT no especulativa*: este método es similar al anterior. Difiere en la estructura de la que se parte para obtener el mapeo restaurado y en el sentido del recorrido por el ROB. Este esquema parte de una estructura que almacena las asociaciones entre registros lógicos y de renombre consolidadas (por ejemplo la RRAT) y recorre el ROB en el orden de programa, desde la instrucción más vieja hasta la instrucción previa a la instrucción más joven que hay que anular, incorporando las cambios en el mapeo realizados por cada instrucción. Al igual que el método anterior, este esquema está limitado en el número de instrucciones que puede procesar cada ciclo, lo que implica una penalización proporcional al número de instrucciones cuyos cambios en el mapeo hay que incorporar.

4. *MT no especulativa*: el recorrido por el ROB del esquema anterior puede evitarse esperando a que se retiren todas las instrucciones previas a la instrucción más vieja a anular. En ese momento, dicha instrucción estará en la cabeza del ROB y la MT no especulativa (por ejemplo la RRAT) contendrá el estado preciso. Esta alternativa puede suponer muchos ciclos de penalización si una instrucción más vieja a la que provoca la excepción es de latencia larga y bloquea su avance en el ROB (por ejemplo, un load que falla en L2). Se diferencia del anterior método en que en lugar de rehacer los cambios en el renombre realizados por las instrucciones previas a la instrucción más vieja a anular, se espera a que dichas instrucciones se jubilen.

La Figura 2.5 describe con un ejemplo el funcionamiento de estos cuatro métodos de recuperación del estado del renombre de registros. Si suponemos que la (i1) es la instrucción más vieja sin retirar (está en la cabeza del ROB) y que la última instrucción renombrada es (i5), entonces los mapeos previos al renombre de (i1) y de (i6) se corresponderían con el estado de las asociaciones consolidadas y más recientes, respectivamente (por ejemplo, RRAT y tabla de mapeo).

6. Puede ser la instrucción posterior a un salto mal predicho o a una instrucción que provoca una excepción.

a) Código original

```
i1:  add r1 = ..
i2:  add r2 = ..
i3:  bz  r2, i6
i4:  ld  r1 = ..
i5:  add r2 = ..
i6:  ..
```

b) Código renombrado

```
i1:  add p21 = ..; old_pd = p1
i2:  add p22 = ..; old_pd = p2
i3:  bz  p22, i6
i4:  ld  p23 = ..; old_pd = p21
i5:  add p24 = ..; old_pd = p22
i6:  ..
```

c) Contenido del ROB tras el renombre de las instrucciones (i1) a (i5).

inst.	nuevo mapeo	mapeo previo
i1	r1 <- p21	old_pd = p1
i2	r2 <- p22	old_pd = p2
i3	--	--
i4	r1 <- p23	old_pd = p21
i5	r2 <- p24	old_pd = p22
i6	..	..

ROB

d) Asociaciones especulativas/consolidadas entre registros lógicos y almacenes de renombre en el instante previo al renombre/jubilación de las instrucciones del código anterior. La instrucción i3 no cambia ninguna correspondencia lógica-física.

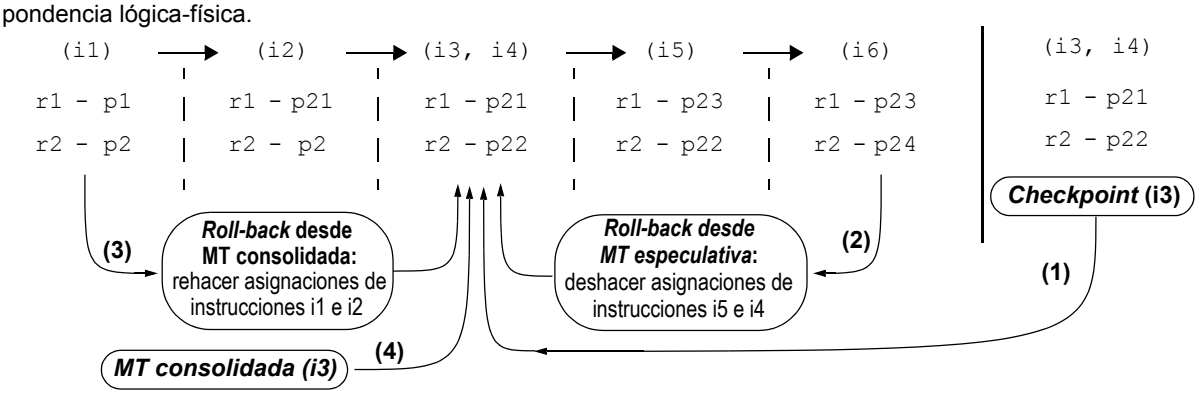


Figura 2.5 Métodos de recuperación del estado del renombre de registros: *checkpointing* (1), *roll-back* desde MT especulativa (2), *roll-back* desde MT consolidada (3) y MT consolidada (4).

Tras detectarse una predicción de salto incorrecta para (i3), podría recuperarse el estado del renombre previo a dicho salto de cuatro formas:

1. *Checkpointing*: directamente se sobreescribe la estructura que mantiene el mapeo con la copia que se hizo en el instante previo al renombre del salto.
2. *Roll-back desde MT especulativa*: partiendo de las asociaciones etiquetadas como (i6), y gracias a la información almacenada en los campos "mapeo previo" del ROB, pueden **deshacerse** las asignaciones de almacén de renombre a los registros destinos de las instrucciones (i5) e (i4).
3. *Roll-back desde MT consolidada*: partiendo de la tabla etiquetada como (i1), y utilizando la información contenida en las entradas "nuevo mapeo" del ROB, pueden **rehacerse** las asignaciones realizadas a los registros lógicos destino de las instrucciones (i1) e (i2). De este modo se puede restaurar el conjunto de correspondencias entre registros lógicos y almacenes de renombre previo al renombre del salto (i3).
4. *MT consolidada*: en lugar de rehacerse las asignaciones realizadas a los registros lógicos destinos (i1) e (i2), se espera a que dichas instrucciones consoliden y de esta forma la MT consolidada ya refleja el mapeo previo al renombre del salto.

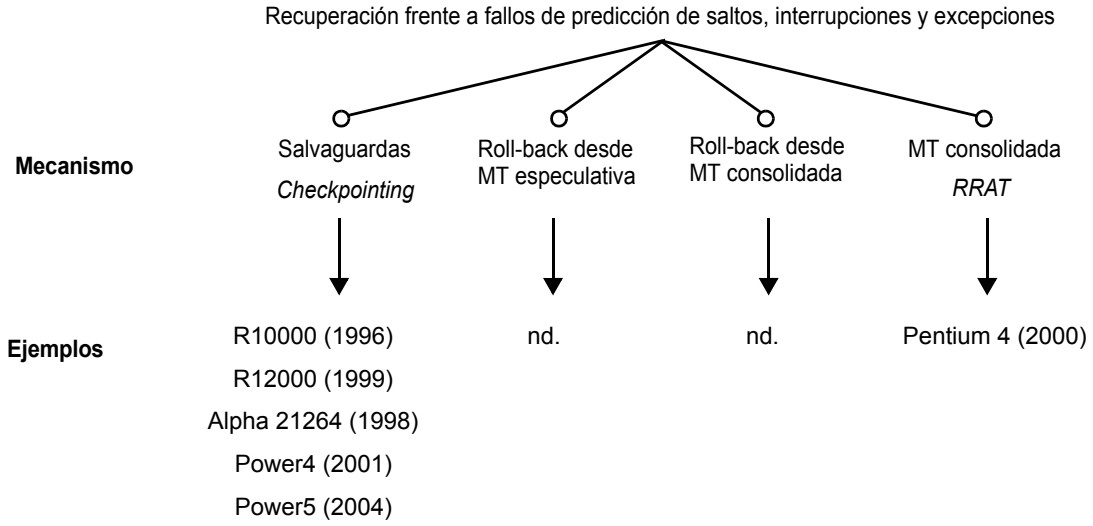


Figura 2.6 Mecanismos de recuperación frente a fallos de predicción de salto, excepciones e interrupciones. nd: no disponible.

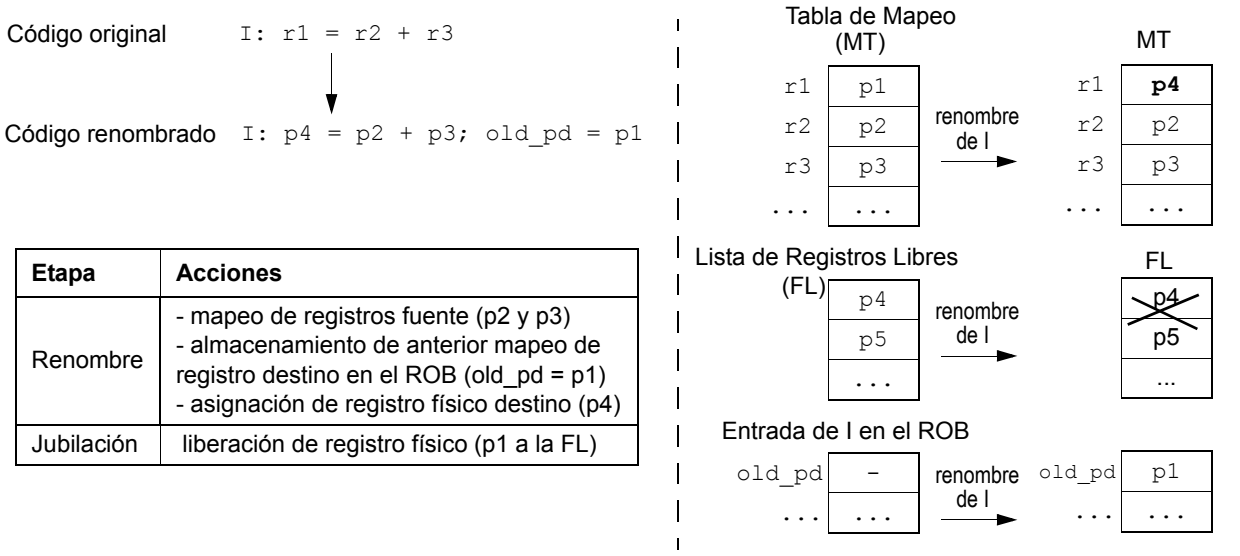
Si suponemos que el salto es predicho correctamente pero el load (i4) provoca un fallo de página, se procedería del mismo modo que el descrito anteriormente, pero anulando las instrucciones (i5) y las siguientes más jóvenes en orden de programa. Para el caso de *roll-back* desde la MT consolidada (por ejemplo, RRAT), podría esperarse a que el load llegara a la cabeza del ROB, en cuyo caso, la RRAT se correspondería con el estado preciso del renombre.

Pueden combinarse varios métodos, por ejemplo, se puede utilizar *checkpointing* para restaurar el renombre tras un error de predicción (caso frecuente, recuperación rápida) y *roll-back* para recuperarse de una excepción (caso menos frecuente, recuperación más costosa). La Figura 2.6 enumera estas opciones de diseño y su aplicación en varios procesadores recientes.

2.2.4 Renombre de Registros convencional

Esta Tesis está centrada en el renombre de registros con un banco de registros físico que contiene tanto los registros físicos como los de la arquitectura (*Physical Register File*, PRF) y con asociaciones entre registros lógicos y físicos almacenadas en una tabla de mapeo (*Mapping Table*, MT). La recuperación es por medio de *checkpoints* en el caso de saltos mal predichos y mediante *roll-back* a partir de la tabla de mapeo especulativa en el caso de las excepciones.

En las siguientes secciones se describen la estructuras utilizadas en esta implementación. También se detallan las acciones de asignación, mapeo y liberación de registros para este diseño. A lo largo de esta tesis nos referiremos a este mecanismo y su implementación como renombre de registros *convencional*.



Etapa	Acciones
Renombre	- mapeo de registros fuente (p2 y p3) - almacenamiento de anterior mapeo de registro destino en el ROB (old_pd = p1) - asignación de registro físico destino (p4)
Jubilación	liberación de registro físico (p1 a la FL)

Figura 2.7 Mapeo de registros fuente, asignación de registro físico destino y liberación de registro físico. Se muestra el estado de la Tabla de Mapeo (MT), de la Lista de Registros Libres (FL) y de la entrada de la instrucción I en en ROB antes y después de la etapa de renombre de la instrucción I (izquierda y derecha respectivamente).

2.2.5 Estructuras para el renombre de registros convencional

En la etapa de renombre, para cada instrucción se realiza en primer lugar el mapeo de sus registros fuente y posteriormente se asigna un registro físico libre a su registro destino (si lo tiene).

Para el mapeo de registros fuente se utiliza la Tabla de Mapeo (*Mapping Table*, MT), una estructura que almacena las asociaciones entre registros lógicos y físicos. Implementada como una RAM, al acceder a la entrada indexada por el identificador de un registro lógico se obtiene su registro físico asociado. Por ejemplo, en la Figura 2.7, la lectura de las entradas r2 y r3 de la MT permite conocer sus registros físicos asociados (p2 y p3, respectivamente).

La asignación de registro físico implica a la lista de registros físicos libres (*Free List*, FL), a la MT y a la estructura de reordenación (*Reorder Buffer*, ROB). La lista de registros físicos libres es una estructura que almacena los registros físicos disponibles para asignar al registro lógico destino de una instrucción. Cuando se renombra un registro destino, se lee un identificador de la FL. Por ejemplo, en la Figura 2.7, al registro lógico r1 se le asigna el registro físico p4, leído de la FL. La nueva correspondencia entre registro lógico y físico queda registrada en la MT, mientras que la anterior correspondencia se guarda en el ROB. De esta forma en la etapa de jubilación se podrá liberar el registro físico asociado a la anterior versión del registro lógico destino, y también podrá recuperarse el estado de la MT en caso de que se anule la instrucción por una excepción. Siguiendo con el ejemplo de la Figura 2.7, en la etapa de renombre de I se almacenaría p1 en la entrada del ROB correspondiente a I, así cuando ésta se retire, liberará p1 escribiéndolo en la FL.

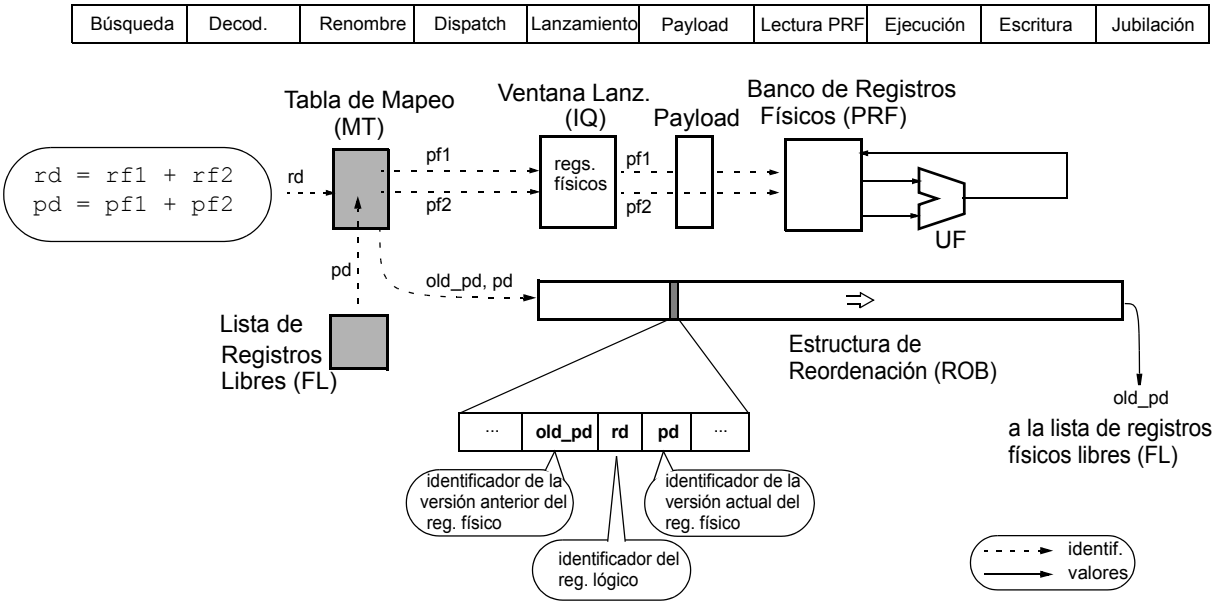


Figura 2.8 Esquema simplificado de la microarquitectura de un procesador con ejecución fuera de orden. Las estructuras directamente implicadas en el renombre de registros se muestran sombreadas. Los componentes se han situado debajo de la etapa en la que realizan actividad.

Asumiendo un ISA con dos operandos fuente y uno destino por instrucción, un procesador capaz de renombrar en un ciclo N instrucciones necesitará una MT con 2*N puertos de lectura para el mapeo de registros fuente y N puertos de escritura para las asignaciones de N registros destino. Además, dado que antes de escribir las nuevas asignaciones hay que salvaguardar el anterior registro físico, son necesarios N puertos adicionales de lectura. Así pues, en total la MT necesita 3*N puertos de lectura y N puertos de escritura. En procesadores con una gran grado de superescalaridad, este elevado requerimiento de puertos hace que esta estructura consuma una cantidad no despreciable de energía [KEP+05][San06][SAM+07].

En la Figura 2.8 se representa un esquema de la microarquitectura en el que se muestran las relaciones entre las estructuras implicadas en el renombre. El modelo de procesador se describe en la sección 3.2.

2.2.6 Ineficiencias del renombre de registros convencional

El renombre de registros convencional mantiene un registro físico asignado hasta la jubilación de la siguiente instrucción que redefine su registro lógico asociado. Este mecanismo, usado en procesadores como el Alpha 21264, MIPS R10K/12K e Intel Pentium 4, tiene una implementación sencilla que soporta tanto especulación de control como recuperación frente a excepciones. Sin embargo, libera registros tarde, causando detenciones innecesarias si el procesador se queda sin registros físicos mientras varios registros almacenan valores que no van a ser leídos por ninguna instrucción.

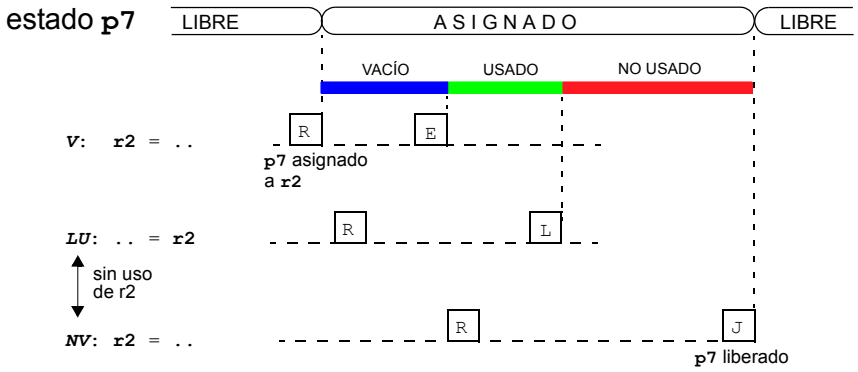


Figura 2.9 Ejemplo de la evolución del estado del registro físico `p7` con el mecanismo de renombre convencional. R = Renombre, L = Lectura del PRF, E = Escritura, J = Jubilación.

En la Figura 2.9, las instrucciones V y NV (*Version* y *Next Version*) definen dos versiones consecutivas del mismo registro lógico destino (`r2`). LU es la última instrucción intermedia en orden de programa que tiene a `r2` como operando. Con el mecanismo de renombre convencional, el registro físico `p7` es asignado a `r2` en la etapa de renombre de V y no es liberado hasta la etapa de jubilación de NV.

La Figura 2.9 también muestra los estados del registro físico `p7` con el mecanismo de renombre convencional. `p7` cambia de Libre a Asignado cuando se renombra la instrucción V, y vuelve al estado Libre cuando la instrucción NV se jubila. Esta asignación en orden de programa evita bloqueos fatales (*deadlocks*) por la imposibilidad de encontrar un registro físico libre.

El registro físico `p7` pasa por tres subestados durante los ciclos en que permanece Asignado. Hasta que V se ejecuta, `p7` no contiene ningún valor y decimos que está en estado Vacío. Después de la escritura de la instrucción V, `p7` contiene el valor que será leído por sus instrucciones consumidoras. `p7` está ahora en el estado Usado. Finalmente, el estado No-Usado comienza con la lectura de la instrucción último uso y finaliza cuando NV se retira. En esta fase, el registro está muerto, pero la política de liberación convencional espera hasta la jubilación de NV para garantizar que ninguna otra instrucción va a leer el valor que contiene.

Para cuantificar la ineficiencia de la política de renombre convencional, hemos medido el número medio de registros asignados que están en los estados Vacío, Usado y No-Usado en un procesador superescalar de grado 8 con 80 registros físicos enteros y otros tantos de coma flotante que ejecuta SPEC CPU2000 (las características del procesador están detalladas en el Capítulo 3). También hemos contabilizado el número de ciclos en los que se detiene el renombre por falta de registros físicos. La Figura 2.10 muestra el resultado del experimento. Puede observarse la intensa presión sobre los bancos de registros, tanto para los códigos enteros (CINT2000) como para los de coma flotante (CFP2000). El número medio de registros Libres es muy bajo (11.3% de los registros enteros para CINT2000, y 10.7% de los registros de coma flotante para CFP2000), siendo prácticamente nulo para la mitad de los programas de coma flotante. Esta escasez de registros libres se traduce en continuas detenciones del renombre por falta de registros físicos. En media, estas detenciones ocurren alrededor del 60% de los ciclos,

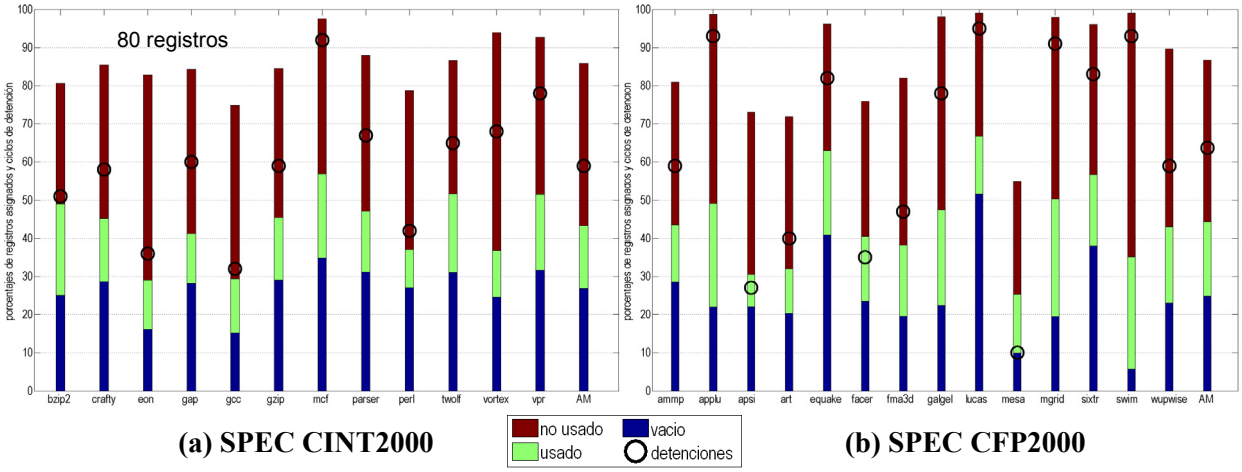


Figura 2.10 Porcentaje de ciclos en los que se detiene la etapa de renombre por falta de registros físicos y porcentaje medio de registros físicos Asignados en los estados Vacío, Usado y No-Usado para un procesador con un PRF de 80 entradas (80int+80fp) que ejecuta SPEC CINT2000 (a) y CFP2000 (b). La figura (a) presenta datos correspondientes a registros enteros y la figura (b) lo hace para registros de coma flotante. La última columna de cada figura es la media aritmética (AM).

superando el 90% en programas como mcf, applu, lucas, mgrid o swim. Una de las causas de esta pérdida de rendimiento es que la liberación convencional mantiene en estado No-Usado, en media, casi la mitad de los registros Asignados. En el caso de eon, gcc, vortex y swim, más del 60% de los registros asignados están en estado No-Usado. También es reseñable que en torno a un 20% de los registros Asignados están en estado Vacío, con un máximo del 52% para lucas. Se ha realizado el mismo experimento con SPEC CINT2006 y se han observado resultados muy similares a los de SPEC CINT2000. Actualmente estamos trabajando en obtener los intervalos de simulación de SPEC CFP2006, por lo que todavía no tenemos resultados de dichos *benchmarks*.

2.3 Modificaciones al renombre de registros

Generalmente, los trabajos que plantean cambios al mecanismo de renombre de registros están centrados en una implementación concreta. En una primera aproximación, pueden diferenciarse propuestas de variantes para mecanismos de renombre basados en Banco de Registros Físico (PRF), Banco de Registros de Renombre (RRF) y Reorder Buffer (ROB) -ver Figura 2.3-. A continuación se repasan las propuestas más relevantes para las implementaciones de renombre basadas en RRF y en ROB. Los trabajos relacionados con el que hemos denominado mecanismo convencional de renombre de registros se revisan con más detalle en la sección 2.4.

2.3.1 Renombre basado en Reorder Buffer y Registros de Renombre

Se han publicado varios trabajos que comparten el objetivo de evitar las escrituras en el ARF de valores de vida corta alojados en el ROB o en los registros de renombre.

En [LoGa95] se propone modificar el ISA para que el compilador pueda comunicar al hardware aquellos valores que serán leídos del ROB por todas sus instrucciones consumidoras (*short-lived values*). Los objetivos son dos:

1. Evitar las escritura de dichos valores en el ARF: esto permitiría reducir el número de puertos de escritura del ARF sin perder prácticamente prestaciones.
2. Evitar la asignación de registros lógicos a dichos valores: esta aplicación permite reducir el número de *spills*, y por tanto, mejorar las prestaciones.

En [SRG02] se presenta un mecanismo hardware que trata de evitar escrituras en el ARF con el objetivo de ahorrar energía. La idea -*Lazy Retirement*- consiste en retrasar la escritura desde el ROB al ARF de un valor generado por una instrucción que se jubila hasta el momento en que va a ser sobreescrito por una instrucción más joven. La escritura del valor en el ARF puede evitarse si el registro lógico que estaba asociado a dicho valor ha sido sobreescrito por una instrucción ya jubilada.

También con el objetivo de reducir el consumo energético, [PKE+03] propone eliminar las escrituras tanto en el ARF como en el ROB de los valores de vida corta (*short-lived values*). A diferencia del trabajo de [LoGa95], en [PKE+03] se define un valor de vida corta como aquél cuyo registro lógico ya ha sido renombrado en el momento en que la instrucción que lo produce llega a la etapa de escritura de registros (*write-back*). Un pequeño banco de registros dedicado (*Short-lived Register File*, SRF) almacenará este tipo de valores para permitir la recuperación frente a fallos de predicción de salto y soportar de excepciones precisas.

2.4 Modificaciones al Renombre Convencional de Registros

Se han propuesto diversas modificaciones a la implementación convencional del mecanismo de renombre de registros, es decir, de la basada en un Banco de Registros Físico mezclado que combina los registros de la arquitectura y los de renombre. A continuación se presentan los trabajos más importantes agrupados según planteen cambios en la parte de asignación, en la de liberación, o en la de recuperación del estado preciso. Por su interés, también se han incluido en esta sección las propuestas que añaden una cache del banco de registros (ver Figura 2.11).

2.4.1 Asignación de registros

Las propuestas de modificación de la política convencional de asignación de registros se pueden dividir en tres grupos:

1. Asignación tardía: se retrasa la asignación de registros físicos que el mecanismo de renombre convencional efectúa en la etapa de renombre. Estas propuestas pretenden eliminar el estado Vacío de un registro físico, de forma que cuando pase de libre a asignado, entre directamente al estado Usado. Como en estas propuestas la asignación se realiza ahora en el núcleo de ejecución fuera de orden (*back-end*), hay que contemplar acciones para evitar que una instrucción quede bloqueada sin posibilidad de conseguir un registro físico y se produzca un *deadlock*.

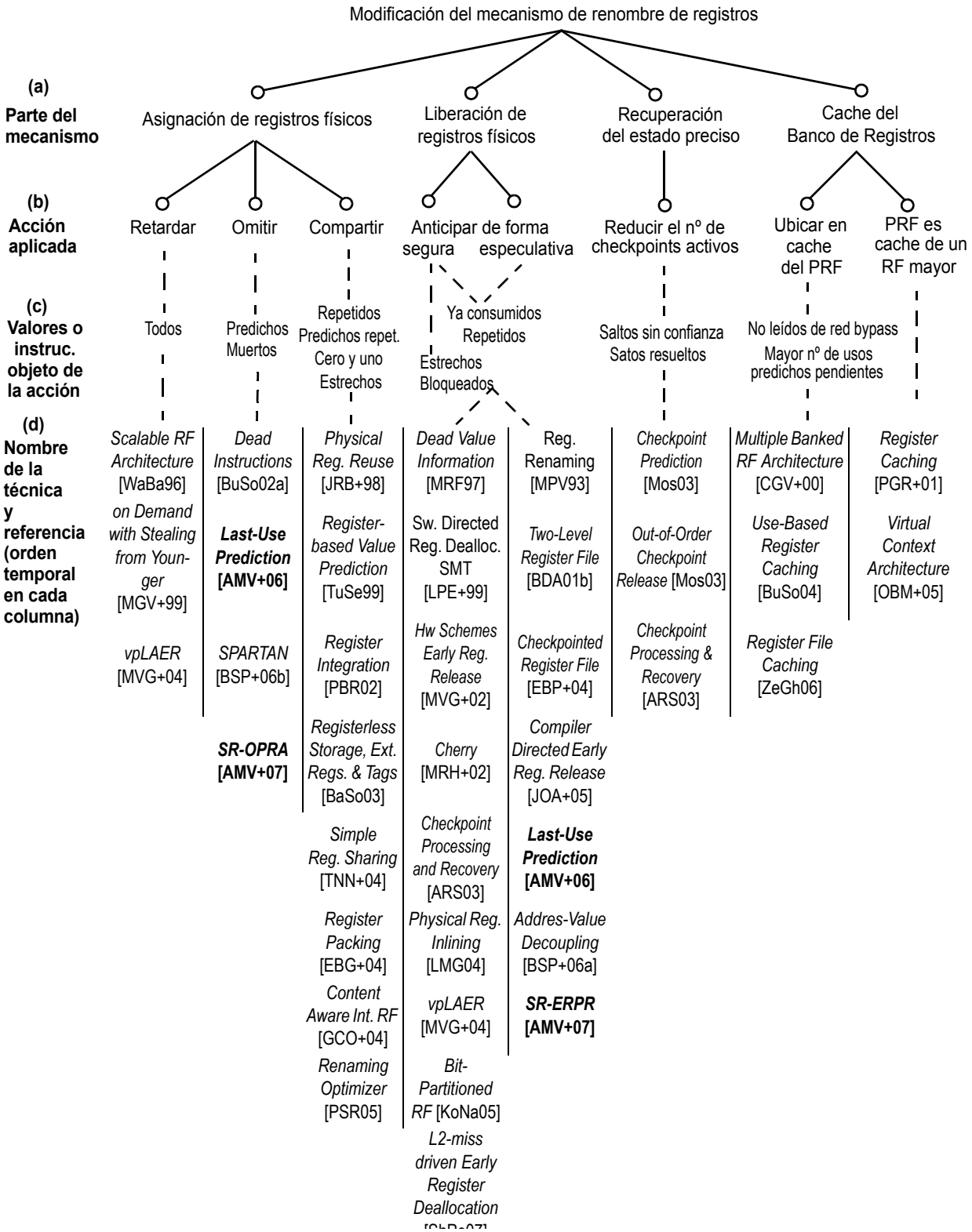


Figura 2.11 Clasificación de los distintos trabajos que proponen modificaciones al mecanismo de renombre de registros. Las contribuciones de esta Tesis están resaltadas en negrita. Abreviaturas: RF: Register File, Reg: Register.

2. Omisión de asignación: bajo determinadas circunstancias se omite la asignación de registro físico a un registro lógico. En caso de que se realice de forma especulativa, como ocurre en todas las propuestas dinámicas -basadas en hardware-, hay que prever un mecanismo de recuperación.
3. Compartición de registro físico: la idea común de estos trabajos es la asociación del mismo registro físico a distintos registros lógicos. Las primeras aproximaciones tratan de que valores repetidos compartan un mismo registro físico, mientras que otras técnicas más sofisticadas pretenden que un registro físico almacene o bien partes comunes de distintos valores o bien varios valores estrechos (de pocos bits significativos).

Asignación tardía. Estas propuestas eliminan el estado Vacío de un registro físico a cambio de prever un mecanismo para evitar bloqueos fatales (*deadlock*) por la asignación de registros físicos fuera del orden de programa.

[WaBa96] particiona el PRF en varios bancos cuyos registros son asignados en la etapa de escritura. Con este esquema puede ocurrir que una vez finalizada la ejecución de una instrucción, una unidad funcional no encuentre un registro físico disponible para escribir su resultado, en cuyo caso tendrá que detenerse. Esta detención puede provocar *deadlock* si se bloquea el lanzamiento de la instrucción en vuelo más vieja, la única que puede liberar un registro físico. Para evitar esta situación indeseable, cuando la instrucción más vieja esté bloqueada porque no hay registros físicos disponibles, se pasa a un modo especial de ejecución en el que sólo se permite el lanzamiento de la instrucción más vieja en vuelo. Una vez finalizada su ejecución, se le permitirá escribir en el registro físico que en principio iba a liberar en la etapa de jubilación (su *old_pd*, el asociado a la anterior versión de su registro lógico).

Monreal et al. proponen retrasar la asignación de registros físicos hasta el último ciclo de la etapa de ejecución [MGV+99][MGV+00a]. El uso de registros virtuales permite desacoplar las acciones de seguimiento de dependencias y asignación de registros físicos. Para evitar *deadlocks*, se propone un esquema de asignación en el cual, si una instrucción no encuentra registro físico, se lo roba a la instrucción más joven que lo tenga asignado (bajo demanda con robo a la más joven, *on-Demand with Stealing from Younger*, DSY). Un inconveniente de este mecanismo es que la instrucción robada tiene que reejecutarse. En una publicación posterior se detalla un mecanismo de renombre (*Virtual-Physical registers based Late Allocation and Early Release*, VP-LAER) capaz de realizar asignación retardada y liberación anticipada [MVG+04].

Omisión de asignación. Como estas técnicas están basadas en predicción, todas articulan mecanismos de recuperación en caso de errores de predicción que puedan afectar a la corrección del programa.

Butts y Sohi sugieren que no se asigne registro físico a una instrucción cuando se prediga dinámicamente que el valor que produce nunca va a ser leído por una instrucción (valores muertos), e incluso que no se ejecute este tipo de instrucciones denominadas instrucciones muertas [BuSo02a].

En [AMV+06], además de una política de liberación anticipada de registros físicos, propusimos la omisión de asignación de registro físico a un resultado si un predictor de último uso prevé que no va a tener instrucciones consumidoras. Estos valores identificados como muertos son

escritos en un almacén auxiliar para permitir su posterior recuperación en caso de que la predicción de último uso resulte ser incorrecta. Esta es una de las contribuciones principales de esta Tesis.

Balkan et al. evitan la asignación de registro físico a valores predichos como transitorios (*transients values*, TV) [BSP+06b]. Se define un valor como transitorio cuando su instrucción productora (V), su única consumidora (U) y la instrucción que renombra el registro lógico del valor transitorio (NV) cumplen una serie de condiciones:

- V: en el instante de la escritura del TV, el registro lógico destino de la instrucción productora ya ha sido renombrado (condición de *short-lived values* en [PKE+03]). Además, la instrucción V sólo debe tener un único consumidor.

- U: el único consumidor de un valor transitorio realiza su lectura de la red de cortocircuitos. Asimismo, el consumidor no puede ser anulado por predicciones de latencia incorrectas asociadas a operaciones de memoria anteriores.

- NV: no hay saltos condicionales entre las instrucciones V y NV, es decir, entre la que genera el valor transitorio y la siguiente instrucción que renombra su registro lógico destino.

El cumplimiento de todas estas condiciones, en especial el de la ausencia de saltos entre la instrucción productora de un valor y la instrucción que redefine el registro lógico asociado a dicho valor, limita el número de valores susceptibles de ser identificados como transitorios. Los valores predichos como transitorios se escriben en una pequeña estructura auxiliar (*Transient Value Buffer*, TVB) para poder gestionar las predicciones de valor transitorio incorrectas. Además, este mecanismo requiere copias periódicas del PRF para soportar excepciones precisas.

En la misma línea que un trabajo previo, en [AMV+07] se presenta una microarquitectura que soporta renombre especulativo, es decir, tanto la omisión en la asignación como la liberación de registros físicos de forma especulativa. Los registros virtuales resuelven los problemas que surgen con estas agresivas estrategias de renombre (actualización de mapeo, mecanismo de recuperación...). La novedad del trabajo estriba en que la microarquitectura detallada soporta políticas arbitrarias de omisión de asignación y de liberación de registros físicos. Por tanto, es posible integrar distintos esquemas de detección o predicción de valores a los que no se va a asignar registro físico o valores cuyo registro físico va a liberarse anticipadamente. También sería factible integrar esquemas de compartición de registro físico. Esta es otra de las contribuciones principales de esta Tesis.

Compartición de registro físico. El denominador común de estos trabajos es evitar la redundancia en el banco de registros, ya sea en forma de valores repetidos, valores con gran similitud o valores con secuencias largas de ceros y unos. La idea general es que los resultados (todos sus bits o parte) de distintas instrucciones compartan un mismo registro físico, ya sea porque son el mismo valor, o porque su anchura permite empaquetarlos sin pérdida de información. La decisión de compartir un registro físico se puede tomar de forma segura una vez detectado un valor adecuado (por ejemplo, un valor repetido) o mediante predicción (por ejemplo, apostando a que una instrucción generará un valor repetido). En el segundo caso, hay que contemplar mecanismos que gestionen los fallos en la predicción.

El esquema propuesto por Jourdan et al. aprovecha el hecho de que el número de valores producidos por las instrucciones es bastante limitado y que muchos de los valores producidos se repiten [JRB+98]. En concreto, se propone el reúso de un registro físico (*Physical Register Reuse*, PRR) cuando se detecte que un resultado es el mismo que uno previo. Si esta detección se realiza antes del renombre, a la instrucción que produce el valor ya contenido en un registro físico puede asignársele directamente dicho registro. Si la detección es tras el renombre, puede liberarse inmediatamente el registro que se le había asignado y asignarle el registro físico que contiene el valor repetido. Las detecciones pueden ser seguras o especulativas. Por otra parte, el reúso de registros físicos también permitiría evitar la ejecución de ciertas instrucciones (por ejemplo, *mov*).

Tullsen y Seng presentan una técnica llamada predicción de valor de registro (*Register Value Prediction*, RVP) para reutilizar registros físicos [TuSe99]. A diferencia de [JRB+98], más centrado en las ideas, este trabajo detalla la implementación, en concreto, el predictor de valor y el mecanismo de recuperación en caso de fallo de predicción.

[PBR02] extiende un trabajo previo de reutilización de resultados de instrucciones anuladas para permitir un reúso más general de resultados previos mediante la compartición de registros físicos. En concreto, a una instrucción puede asignársele el mismo registro físico que se le asignó a una instancia previa de la misma instrucción (mismo PC) o a una instrucción con un mismo índice obtenido a partir del código de operación.

También con el objetivo de reusar resultados recientes mediante la compartición de registros físicos, Balakrishnan y Sohi analizan la localidad de los resultados producidos por las instrucciones en vuelo [BaSo03]. Los autores concluyen que un esquema hardware que permita el reúso de registros físicos por parte de valores arbitrarios requiere un hardware de gran complejidad. Por ello, proponen dos esquemas de reúso de registros físicos de bajo coste centrados en los valores cero y uno (*Registerless Storage* y *Extended Register and Tags*).

Al igual que el trabajo anterior, [TNN+04] aplica las ideas de reúso de registros físicos para el caso de los resultados cero y uno. Según los autores, una implementación de baja complejidad hardware optimizada para detectar dichos valores compartidos puede obtener casi la mitad de los beneficios que una técnica de compartición de resultados arbitrarios. En este trabajo se propone otro mecanismo que permite compartir el mismo registro físico a varias versiones de un registro lógico.

Una evolución de estos trabajos es la idea de utilizar un único registro físico para almacenar los bits de más peso comunes a varios valores y así utilizar registros de menos bits para almacenar los bits restantes en los que difieren [GCO+04]. Esta localidad parcial de valores (*partial value locality*) puede explotarse dividiendo el PRF monolítico en tres bancos distintos (Largo, Corto y Sencillo) con distinto número de entradas y anchuras adaptadas a los diferentes tipos de datos que van a contener. La evaluación concluye que una configuración con compromiso entre prestaciones y consumo de energía es la siguiente:

- Largo (*Long Register File*): 48 entradas que almacenan los valores sin localidad parcial (realmente almacenan sólo 51 bits, ya que los 13 bits restantes se almacenan en el RF sencillo).

- Corto (*Short Register File*): 8 entradas para guardar los 47 bits de más peso comunes.
- Sencillo (*Simple Register File*): 112 entradas de 22 bits. Tiene una entrada asignada a cada valor que se produce. Cada entrada consta de 2 bits para almacenar el tipo de valor y 20 bits con distinto uso según el valor sea con localidad parcial o no:
  - **a)** 3 bits para el puntero a la entrada del RF corto que almacena la parte común del valor y 17 bits para almacenar la parte única del valor.
  - **b)** 7 bits para un puntero a la entrada del RF largo que guarda los valores sin localidad y los restantes 13 bits para almacenar parte del dicho valor.

En el mismo año se publicó otro trabajo cuyo objetivo es explotar el hecho de que un considerable porcentaje de los valores que se generan son "estrechos", es decir, con pocos bits significativos si se comparan con la anchura de un registro físico [EBG+04]. Esta observación puede aprovecharse para reducir la presión sobre el PRF empaquetando distintos valores "estrechos" en un único registro físico.

Petric et al. presentan y detallan la implementación de un mecanismo de renombre (*Renaming Optimizer*, RENO) que integra la versión dinámica (en tiempo de ejecución) de varias y conocidas técnicas de optimización estáticas (en tiempo de compilación) [PSR05]. Mediante asignaciones especiales de registro físico en la etapa de renombre, RENO dinámicamente evita la ejecución de instrucciones move, elimina la ejecución de instrucciones redundantes (subexpresiones comunes), realiza cortocircuitos especulativos de memoria (la versión dinámica de la asignación estática de registros, sustituye cadenas de dependencias productor-store-load-consumidor por productor-consumidor) y simplifica expresiones comunes (*constant folding*[7]). Excepto la última, el resto de adaptaciones ya habían sido propuestas en trabajos anteriores [MoSs97][JRB+98][PBR02]. RENO las implementa mediante compartición de registros físicos. Para la simplificación dinámica de expresiones comunes, se extiende el mapeo lógico-físico (r1->p1) a otro que codifica una operación que implica al registro físico y a un valor inmediato (r1-> p1+4). De esta forma, resultados distintos también pueden compartir un mismo registro físico.

2.4.2 Liberación de registros

Prácticamente todas las propuestas van en la misma línea: anticipar la liberación de los registros físicos para reducir el tiempo que un registro está en estado No-Usado y permitir el adelanto de su reutilización.

Son numerosas las mejoras que se han propuesto con el objetivo de relajar las condiciones de liberación de un registro físico para poder reutilizarlo antes de que se jubile la instrucción que redefine su registro lógico asociado [MRF97][LPE+99][BDA01b][MVG+02][MRH+02][ARS03][LMG04][EBP+04][KoNa05][JOA+05][AMV+06][BSP+06a][AMV+07][ShPo07]. Podemos distinguir entre políticas de liberación *seguras* y *especulativas*. Llamamos políticas

7. La simplificación de expresiones constantes (*constant folding*) es una técnica de optimización utilizada en tiempo de compilación.

de liberación seguras a aquellas que no liberan prematuramente valores útiles, ni siquiera durante la ejecución por caminos de control incorrectos. Por tanto, tras una predicción de salto incorrecta, no se necesitan mecanismos especiales de recuperación por la posibilidad de que se haya liberado de forma incorrecta algún registro físico que sea referenciado posteriormente[8].

Políticas de liberación seguras. En este grupo existen tanto aproximaciones *software* como *hardware*. Las primeras están basadas en análisis estáticos en tiempo de compilación y requieren cambios en el juego de instrucciones (ISA). Las segundas son técnicas dinámicas que no necesitan extender el ISA.

Martin et al. definen el concepto de Información de Valor Muerto (*Dead Value Information*, DVI) como el conocimiento de ciertos registros que contienen valores que ya han sido consumidos y no van a ser leídos en el futuro [MRF97]. La DVI, obtenida a partir del análisis del compilador, puede explotarse por medio de un tipo especial de instrucción añadido al ISA (*kill register*) para liberar anticipadamente registros físicos que contienen valores muertos[9], así como para eliminar instrucciones *save* y *restore* innecesarias. Lo et al. aplican una técnica software similar a la anterior en el contexto de procesadores SMT [LPE+99].

Las soluciones hardware monitorizan ciertos aspectos de la ejecución para liberar un registro físico cuando se cumplen unas determinadas condiciones, como por ejemplo, dependencias resueltas de registros, resultado de saltos condicionales, la posibilidad de que se produzcan excepciones o la generación de resultados con pocos bits significativos [MVG+02][MRH+02] [ARS03][LMG04].

En [MVG+02] se presentan dos mecanismos hardware para liberar registros anticipadamente. La idea del mecanismo básico consiste en asociar la liberación de un registro a la etapa de jubilación de su última instrucción consumidora (*last-use*, LU) en lugar de a la siguiente instrucción que escriba en el mismo registro lógico (*next version*, NV) siempre que no haya saltos condicionales pendientes entre las instrucciones LU y NV. Un mecanismo extendido permite planificar liberaciones anticipadas para los casos en que la identificación de la instrucción LU sea especulativa (es decir, que haya saltos pendientes de resolver entre las instrucciones LU y NV). En concreto, se retrasa la liberación del mecanismo básico hasta el momento en que los saltos pendientes entre LU y NV estén resueltos.

El modo de ejecución Cherry permite anticipar la reutilización de varios recursos usados por las instrucciones [MRH+02]. En el caso de los registros físicos, se adelanta su liberación desde la jubilación de la instrucción que redefine el mismo registro lógico hasta el denominado Punto de No Retorno (*Point of No Return*, PNR), instante es que es seguro que dicha instrucción no va a ser anulada por eventos relativamente frecuentes como saltos mal predichos o traps de re-

8. Excluimos de esta definición las acciones derivadas de la reconstrucción del estado preciso tras las relativamente poco frecuentes (si se comparan con los fallos de predicción de salto) excepciones.

9. Los autores proponen insertar una instrucción kill register tras la instrucción que hace el último uso de un registro. Al jubilarse esta instrucción especial se haría efectiva la liberación del registro físico.

petición de memoria[10]. La recuperación tras eventos menos frecuentes como excepciones o interrupciones es posible gracias a los *checkpoints* realizados periódicamente.

La microarquitectura CPR (*Checkpoint Processing and Recovery*) se caracteriza por el soporte de una gran ventana de instrucciones [ARS03]. Para poder implementar esta estructura, entre otros cambios importantes, se necesita una política de liberación de registros más agresiva que la convencional. En este trabajo se propone liberar un registro físico cuando todos sus consumidores lo hayan leído y no esté almacenado en ningún checkpoint activo.

[LMG04] propone liberar anticipadamente aquellos registros asignados a valores que al tener pocos bits significativos pueden almacenarse directamente en la tabla de mapeo entre registros lógicos y físicos (*Map Table*, MT). Estos valores "estrechos" serán leídos por sus instrucciones consumidoras directamente de la MT.

Kondo et al. también explotan la frecuente generación de valores estrechos [KoNa05]. Su propuesta se basa en que una vez detectado un valor con pocos bits significativos, se libera el espacio de almacenamiento que no necesita para que pueda ser reasignado a otros valores generados por instrucciones posteriores. Esto es posible gracias a que el banco de registros se divide en varios bancos de menor anchura. El trabajo analiza con detalle las acciones asociadas a la actualización del mapeo de un valor estrecho, en concreto, la propagación del nuevo mapeo a las estructuras pertinentes y la gestión de los mapeos obsoletos que han podido ser leídos por instrucciones consumidoras de un valor estrecho. Estas acciones pueden resultar complejas.

Por último, en [ShPo07] se propone un mecanismo de liberación anticipada para procesadores SMT. Cuando una lectura de memoria (*load*) produce un fallo de L2[11], se inspeccionan las instrucciones bloqueadas independientes del load y se liberan los registros físicos mapeados a las anteriores versiones de sus registros lógicos destino (old_pd). Esta identificación de registros candidatos a ser liberados se realiza recorriendo el ROB desde las instrucciones más viejas a las más jóvenes, deteniéndose cuando se encuentra un salto condicional sin resolver.

Políticas de liberación especulativas. Por otra parte, se han propuesto políticas de liberación más agresivas, hasta el extremo de que pueden liberar un registro físico antes de que todos sus potenciales consumidores lo hayan leído, ya sea porque la ejecución por un camino de control incorrecto ha inducido a tal liberación prematura [MPV93][BDA01b][EBP+04][JOA+05][BSP+06a] (por ejemplo, una instrucción de camino incorrecto puede redefinir el registro lógico asociado a un valor y así cerrar la cuenta de sus potenciales consumidores) o porque un predictor ha errado al señalar un valor como candidato a ser liberado anticipadamente [AMV+06][AMV+07]. Por lo tanto, todas estas estrategias de liberación deben contemplar mecanismos que permitan recuperar los valores descartados incorrectamente.

La mayoría de estas políticas establecen un conjunto de condiciones bajo las cuales un registro físico puede ser liberado anticipadamente, por lo que realizan un seguimiento continuo de los registros físicos para detectar el cumplimiento de las condiciones fijadas.

10. Un trap de repetición de memoria (*memory replay trap*) ocurre cuando se descubre que un load se ha ejecutado antes que un store previo a la misma dirección de memoria.

11. En general, del nivel inferior de la jerarquía de cache.

Moudgill et al. proponen liberar un registro físico cuando su última instrucción consumidora se haya ejecutado [MPV93]. La identificación de dicha instrucción para cada registro físico se efectúa mediante un contador de consumidores pendientes y un bit que indica si su registro lógico asociado ha sido redefinido. Esta propuesta no soporta excepciones precisas, puesto que los contadores no se recuperan correctamente en caso de que una instrucción sea anulada. Los siguientes trabajos, además de un conjunto de condiciones para la liberación, dedican recursos hardware para recuperar los valores liberados prematuramente de forma incorrecta. Estas estructuras de respaldo pueden consistir en un Segundo Nivel de Banco de Registros [BDA01b] o una salvaguarda del PRF (*Checkpointed Register File*, CRF) [EBP+04][JOA+05][BSP+06a].

Balasubromanian et al. también proponen una política de liberación anticipada basada en la identificación de registros inactivos mediante contadores de consumidores pendientes [BDA01b]. Un segundo nivel de banco de registros a donde es transferido un registro que se considera inactivo (sin consumidores pendientes y con su registro lógico asociado redefinido) permite el soporte de excepciones precisas. Un valor puede volver al nivel de registros activos (primer nivel) cuando su liberación no ha sido correcta ya que todavía tiene consumidores pendientes, hecho que puede ocurrir tras una predicción de salto incorrecta.

Los contadores de consumidores pendientes también son utilizados en [EBP+04] para liberar un registro físico cuando la instrucción a la que está asignado se jubila, todos los consumidores del valor contenido en el registro físico ya lo han leído y el registro lógico asociado ha sido redefinido. Un banco de registros de respaldo (*Checkpointed Register File*, CRF) integrado en el PRF almacena los valores liberados anticipadamente para permitir el soporte de excepciones precisas. Como el tamaño del CRF está limitado por el del PRF, bancos de registros pequeños perderán muchas oportunidades de reutilizar sus entradas. Además, por construcción el CRF sólo puede contener como máximo un valor asociado a cada registro físico, lo que también restringe las oportunidades de liberar anticipadamente un registro físico.

Jones et al. utilizan el análisis del compilador para liberar un registro físico después de que haya sido leído por su único consumidor [JOA+05]. En este esquema, varios registros lógicos se reservan para almacenar los valores que el compilador ha identificado como de un único uso. Puesto que la instrucción que activa la liberación de uno de estos valores puede ser de camino incorrecto, es necesario un banco de registros de respaldo para recuperarse de la especulación de control incorrecta y soportar excepciones precisas. Se utiliza el CRF de [EBP+04]. Además de la restricciones asociadas a la organización del CRF, esta técnica tiene otra limitación debida a su naturaleza estática: el conocimiento limitado del comportamiento del programa en tiempo de ejecución impide la identificación de todos los valores con un único consumidor.

En [AMV+06] buscamos los límites que puede alcanzar una política especulativa de liberación anticipada de registros físicos. Se propone utilizar la predicción de último uso de un registro físico para activar su liberación anticipada. De esta forma y a diferencia del resto de los esquemas propuestos anteriormente, se puede liberar un registro físico aunque la siguiente instrucción que redefine su registro lógico asociado no se haya renombrado. En este trabajo, también se propone una técnica que permite la lectura por parte de sus consumidores de un valor almacenado en un registro físico que ha sido liberado anticipadamente e incluso ha podido ser reasignado a otra instrucción. Este rescate, posible hasta que el valor es sobreescrito, permi-

te reducir el número de acciones de recuperación provocadas por fallos en la predicción de último uso. Este trabajo es una de las contribuciones de esta Tesis.

El mismo año, Balkan et al. explotan también el hecho de que después de la liberación de un registro físico, el valor que contiene está disponible durante varios ciclos hasta que es sobreescrito por la instrucción a la que se ha reasignado el registro físico [BSP+06a]. Esta técnica, llamada Desacoplo de Dirección y Valor de un Registro (*Address-Value Decoupling*, AVD), libera un registro físico cuando ha sido escrito y su registro lógico asociado ha sido renombrado. La instrucción que recibe un registro físico que ha sido liberado anticipadamente espera en una estructura dedicada llamada Buffer Temporal de Instrucciones (*Temporary Instruction Buffer*, TIB) a que el valor que contiene dicho registro físico sea leído por todos sus consumidores. Por otra parte, otra estructura integrada con el PRF (en concreto, el CRF de [EBP+04]) almacena los valores liberados anticipadamente por si se requieren tras un salto mal predicho (por ejemplo, si la instrucción que renombró el registro lógico asociado a un valor liberado anticipadamente era de camino incorrecto habrá que restaurar el valor en el registro físico correspondiente). Este mecanismo de recuperación tiene las limitaciones ligadas al CRF citadas anteriormente (un registro físico no puede tener dos liberaciones anticipadas a la primera liberación convencional) que limitan el número de liberaciones anticipadas que se pueden efectuar.

Como se ha citado anteriormente, en [AMV+07] detallamos una microarquitectura que soporta políticas arbitrarias de renombre especulativo, es decir, tanto la omisión en la asignación como la liberación de registros físicos de forma especulativa. Aunque la evaluación se realiza con políticas basadas en predicción de último uso [AMV+06], esta implementación del mecanismo de renombre podría incluir esquemas de liberación anticipada software y hardware, además de otras optimizaciones a la hora de asignar registros (retraso, omisión, compartición). Este trabajo está integrado dentro de esta Tesis.

A excepción de la estrategia software [JOA+05], de la escritura en la tabla de mapeo [LMG04], y de las basadas en predicción [AMV+06][AMV+07], ningún esquema de liberación anticipada de registros, seguro o especulativo, es capaz de liberar un registro físico hasta que la siguiente redefinición de su registro lógico asociado llegue a la etapa de renombre. Otros además requieren que la redefinición pierda la condición de especulativa [MPV93]. La primera condición retrasa la liberación anticipada de un registro en caso de que la instrucción que redefine su lógico asociado aparezca muchos ciclos tarde, algo que según nuestros experimentos ocurre con frecuencia. Por otra parte, y de nuevo con las mismas excepciones citadas anteriormente, todos estos trabajos necesitan llevar la cuenta de las lecturas que tiene pendientes cada registro físico[12], lo que implica un hardware complejo por la necesidad de restaurar su estado correcto tras cada salto incorrectamente predicho y tras cada excepción. La complejidad de este mecanismo basado en contadores se pone de manifiesto en [Rot07], el primer trabajo que describe una implementación detallada del mismo. Según el diseño propuesto por [Rot07], los contadores pueden implementarse mediante una matriz de bits, con tantas filas como número de entradas del ROB y tantas columnas como registros físicos. Con un ROB de 256 entradas, 192 registros enteros y otros tantos de coma flotante, la implementación de estos contadores requie-

12. En algunos casos se emplean incluso dos contadores por registro físico [BSP+06a].

re 96 kilobits (12 kilobytes). En la microarquitectura sin ROB propuesta por [ARS03] (CPR (*Checkpoint Processing and Recovery*), con el mismo número de registros físicos, una ventana de lanzamiento (*scheduling window*) de 128 entradas y 8 checkpoints, la implementación de los contadores necesarios para su política de liberación anticipada requiere 51.4 kilobits (6.4 kilobytes).

2.4.3 Recuperación tras saltos mal predichos y excepciones

En esta sección reseñamos dos propuestas que pretenden reducir la complejidad del mecanismo de *checkpointing*.

Moshovos propone las siguientes acciones para reducir los recursos y el consumo de energía del mecanismo de recuperación del renombre tras saltos mal predichos y excepciones [Mos03]:

- Permitir la liberación de checkpoints fuera de orden (*out-of-order checkpoint release*).
- Realizar copias (*checkpoints*) sólo en los saltos para los que hay poca confianza en su predicción (*anyweak checkpoint prediction*).
- Sobreescribir checkpoints de saltos con bastante confianza en su predicción (*lazy anyweak checkpoint prediction*)

[ARS03] analiza el coste de la recuperación tras saltos mal predichos en procesadores con grandes ventanas de lanzamiento. Por una parte, la realización de un *checkpoint* en cada salto condicional o de forma periódica supone un coste hardware que lo hace inviable. Por otra parte, la latencia asociada a la recuperación vía *roll-back* es muy alta. Su propuesta se basa en una microarquitectura CPR (*Checkpoint Processing and Recovery*) que, al igual que [Mos03], realiza *checkpoints* de forma selectiva, solamente en los saltos con alta probabilidad de fallo de predicción. Este mecanismo será usado también para la recuperación del estado preciso tras una excepción.

2.4.4 Cache del Banco de Registros

Existe un grupo de trabajos que propone un banco de registros multinivel. La mayoría de las propuestas se basan en un nivel inferior (grande y lento) y en otro nivel superior más cercano al núcleo de ejecución (más pequeño y rápido) que generalmente se encarga de suministrar los operandos a las instrucciones. Por su similitud con la jerarquía de memoria, al primer nivel del banco de registros se le suele llamar cache de banco de registros (*register file cache*).

Dentro de este grupo de trabajos podemos distinguir entre (1) las propuestas que plantean el uso de una cache del PRF (pequeña, rápida y con los puertos necesarios para suministrar los operandos a las instrucciones) para acelerar un RF gestionado de forma convencional y (2) las propuestas que usan el propio PRF como la cache de un banco de registros mucho mayor.

Dentro del primer grupo de trabajo, destacamos tres aportaciones.

En [CGV+00] se proponen dos políticas de gestión de contenidos para esta cache:

- No-cortocircuito (*non-bypass*): se almacenan en el primer nivel (cache) solamente aquellos valores que no se leen de la red de cortocircuitos. Por tanto, estos valores se escriben en ambos niveles mientras que el resto solamente en el segundo nivel

- Preparado (*ready*): se almacenan en cache aquellos resultados que convierten a una instrucción esperando a ser lanzada en preparada.

Butts y Sohi proponen una gestión de contenidos de la cache de registros basada en la predicción de usos pendientes [BuSo04]. Así, las políticas de inserción y reemplazo tratarán de ubicar y mantener en la cache aquellos registros físicos que tengan un mayor número de consumidores pendientes. Otra contribución del trabajo es el indexado desacoplado (*decoupled indexing*), un esquema de ubicación de registros en la cache que no depende del identificador del registro físico y que permite minimizar fallos de conflicto.

En [ZeGh06] se propone una cache que trata de separar los valores de duración corta y larga. Para ello, se plantea un diseño formado por dos estructuras: una cola FIFO que contiene los valores más recientemente generados y una cache asociativa que trata de retener los valores almacenados en los registros físicos más recientemente leídos y aquellos valores a los que se ha predicho un tiempo de vida largo. Cuando ninguna de ambas caches da servicio a una petición de lectura de un registro físico, la instrucción implicada y sus dependientes son anuladas y relanzadas con una planificación que tiene en cuenta el acceso de la primera al banco de registros principal (más lento).

En cuanto a los trabajos que proponen que el PRF trabaje como cache de un banco de registros mayor, destacamos un par de aportaciones. [PGR+01] plantea un diseño en el que el PRF trabaja como cache de un banco de registros lógico (*Logical Register File*, LRF) con un elevado número de entradas (256 en la evaluación). Un valor es escrito en el LRF cuando la instrucción que lo produce consolida.

La arquitectura de contexto virtual (*Virtual Context Architecture*, VCA) utiliza el PRF como una cache del banco de registros, que en este caso, es un gran espacio de registros lógicos mapeado en memoria (hasta miles de registros) [OBM+05]. El PRF contiene el subconjunto más activo de registros lógicos gracias a la lectura (*fill*) y escritura (*spill*) en memoria con grano de registro. Esta arquitectura puede utilizarse para implementar de forma eficiente bancos de registros de procesadores SMT o ventanas de registros.

2.4.5 Otras propuestas

En este apartado incluimos propuestas que no han podido clasificarse dentro de las categorías anteriores.

La microarquitectura con flujo continuo en el segmentado (*Continual Flow Pipeline*, CFP) libera los recursos asignados a una instrucción de latencia larga y a sus instrucciones dependientes [SRA+04]. También expulsa a dichas instrucciones del segmentado. Al liberar los registros físicos y entradas en la ventana de lanzamiento (*scheduling window*) asignadas a estas instrucciones bloqueadas (por ejemplo, un load de latencia larga -fallo en L2- y a su cadena de instrucciones dependientes), estos recursos pueden ser reasignados a instrucciones independientes

posteriores. Posteriormente, una vez que la instrucción de latencia larga esté preparada para ejecutarse, ella y sus dependientes se reinsertarán en el segmentado y se les reasignarán los registros físicos que requieren para su ejecución.

[KEP+05] propone dos técnicas para reducir el consumo de energía de la MT mediante la eliminación de lecturas de dicha estructura.

1. Deshabilitar la lectura de la MT en el caso de un registro físico fuente que ha sido asignado a una instrucción previa en el mismo ciclo
2. Almacenar en una cache de la MT los mapeos lógico-físico recientes

Sangireddy advierte que el número de ciclos de ejecución en los que es necesario renombrar instrucciones con dos registros fuente es muy limitado (6% para un procesador de 4 vías y 8% para uno de 8 vías) [San06]. Esta observación puede explotarse para reducir el número de puertos de lectura de la tabla de mapeo, y por tanto su latencia, consumo energético y área, sin perder apenas prestaciones. Este diseño simplificado de la MT también permite reducir la complejidad de la lógica de verificación de dependencias (*dependency check logic*, DCL)[13]. La evaluación concluye que una MT con capacidad para renombrar en un ciclo una única instrucción con dos operandos fuente (8 puertos de lectura en lugar de 5 para un procesador de 4 vías, 9 puertos en lugar de 16 para otro de 8 vías) obtiene el mejor compromiso entre complejidad y prestaciones.

En un trabajo posterior se evalúa cuantitativamente la latencia, energía y área de una tabla de mapeo en función del número de checkpoints que integra [SAM+07]. Se analizan dos implementaciones *full-custom* de una MT para un procesador de 4 vías construido con tecnología de 130nm . La diferencia entre ambos diseños radica en la gestión de los *checkpoints*:

- MT convencional: *checkpoints* organizados como registros de desplazamiento mediante buffers serie (*Serial Access Buffer*, SAB). La latencia de la recuperación de un *checkpoint* es proporcional a su distancia al MT.
- MT alternativo: los *checkpoints* se implementan con un buffer de acceso aleatorio (*Random Access Buffer*, RAB). La latencia de la recuperación es la misma para todas las copias. A diferencia del anterior, este diseño permitiría la gestión de checkpoints fuera de orden.

2.5 Contribuciones

Nuestro principal objetivo es incrementar el tamaño efectivo del PRF sin afectar a su tiempo de acceso y sin incrementar su consumo de energía por acceso. Para conseguir este propósito es necesario utilizar los registros físicos de una manera más eficiente, por lo que nos centramos en el mecanismo de renombre de registros (políticas de asignación y liberación de registros físi-

13. En procesadores superescalares, puede ocurrir que sea renombrada una instrucción consumidora en el mismo ciclo en que se le ha asignado registro físico a su instrucción productora. Como la MT no refleja una nueva asociación lógico-físico hasta el ciclo siguiente de la asignación, la instrucción consumidora obtiene su mapeo correcto gracias a la lógica de verificación de dependencias (DCL).

cos). Hemos observado que muchos valores almacenados en el PRF no se van a leer. Estos valores o no tienen consumidores (valores muertos), o se leen de la red de cortocircuitos (bypass), o ya han sido leídos por todos sus consumidores. En todos los casos, son valores candidatos a ser almacenados en una estructura auxiliar ubicada fuera del cualquier camino crítico del procesador con el objetivo de reducir el tamaño del PRF.

Esta gestión mejorada se puede aplicar de dos formas diferentes:

- Puede reducirse el número de entradas del PRF y así reducir también su área, consumo de energía y tiempo de acceso. Este último efecto puede permitir a su vez un incremento de la frecuencia del procesador, lo que repercutirá en una mayor cantidad de trabajo realizada por unidad de tiempo -instrucciones por segundo, IPS-.

- Puede mantenerse el tamaño del PRF y así aumentar las prestaciones por la reducción del número de detenciones en la etapa de renombre debidas a la carencia de registros físicos. De esta forma puede incrementarse el número de instrucciones ejecutadas por ciclo -instrucciones por ciclo, IPC-. En este caso se produce un incremento efectivo del tamaño del PRF pero sin aumentar su área, consumo energético por acceso ni tiempo de acceso.

En este marco de trabajo, nuestra primera contribución al renombre de registros es un mecanismo hardware de renombre especulativo de registros con liberación anticipada y omisión de asignación de registros físicos basada en la predicción de último uso [AMV+06]. A diferencia del resto de los esquemas propuestos anteriormente, este nueva estrategia de renombre puede liberar un registro físico aunque la siguiente instrucción que redefine su registro lógico asociado no se haya renombrado. En este trabajo, también se propone la lectura por parte de sus consumidores de un valor almacenado en un registro físico que ha sido liberado anticipadamente e incluso ha podido ser reasignado a otra instrucción. Este rescate, posible hasta que el valor es sobreescrito, permite reducir el número de acciones de recuperación provocadas por fallos en la predicción de último uso. A excepción del mecanismo propuesto en [MVG+04], este nuevo esquema de renombre es el único que modifica conjuntamente las políticas convencionales de asignación y liberación de registros físicos. El resto de trabajos se centra en sólo una parte del mecanismo de renombre.

Como paso previo al diseño de una arquitectura de complejidad limitada, se ha analizado el potencial de un esquema de renombre que libera anticipadamente aquellos registros físicos que ya han sido leídos por todos sus consumidores (y que por tanto, no van a ser referenciados en el futuro) y que no asigna registro físico a valores sin consumidores. Un mecanismo con conocimiento perfecto del uso futuro de todos los registros físicos marcará el límite de prestaciones que un sistema real podrá alcanzar. En [AMV+06] se define el concepto de *patrón de uso* de una instrucción como la información del último uso de sus operandos, tanto fuente como destino. Se propone explotar la predecibilidad de la información de último uso de un registro en el renombre de registros para acercarse a las prestaciones que obtendría un sistema con conocimiento perfecto del uso futuro de los registros físicos. El concepto de patrón de uso puede considerarse relacionado con la idea de grado de uso presentada en [BuSo02b]. Las diferencias estriban en el rango de predicciones (un número para cada valor generado por una instrucción en el caso de [BuSo02b] y una combinación de hasta 3 valores binarios ("sí es último uso" o "no es último uso" para cada uno de los operandos fuente y destino de una instrucción en

[AMV+06]). Se han diseñado dos predictores capaces de anticipar con gran precisión el uso futuro de los registros de una instrucción, tanto fuentes como destino. Estos predictores permiten alcanzar un nivel de prestaciones muy similar al obtenido por un predictor oráculo.

En [AMV+07] detallamos el diseño de dos predictores de último uso con prestaciones cercanas a las de un predictor perfecto. Las prestaciones del predictor sencillo se tratan de mejorar en [AMV+09] con una original aplicación de técnicas de decaimiento (*decay*), originalmente pensadas para reducir el consumo estático de estructuras de memoria.

En [AMV+07] también se detalla una microarquitectura que soporta políticas arbitrarias de renombre especulativo, es decir, políticas que permiten la omisión de asignación y de liberación anticipada de registros físicos. Un predictor de último uso que controla políticas de renombre especulativo basadas en la predicción de último uso se utiliza para evaluar las prestaciones de esta nueva microarquitectura con una aplicación concreta.

Por último, como trabajo previo a la aplicación del Renombre Especulativo a procesadores SMT, hemos analizado la interacción entre tamaño del PRF y política de asignación de recursos a threads [AMC+08]. Hemos obtenido el tamaño de PRF que mejores prestaciones obtiene para una serie de esquemas que controlan la distribución de registros entre threads.

2.6 Referencias

[ARS03] H. Akkary, R. Rajwar, S.Srinivasan. Checkpoint Processing and Recovery: Towards Scalable Large Instruction Window Processors. *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture* (MICRO 03). pp. 423. Diciembre 2003.

[AMV+06] J. Alastruey, T. Monreal, V. Viñals and M. Valero. “Speculative Early Register Release”. *Proc. of the 3rd conference on Computing frontiers* (CF 06), May 2006, pp 291-302.

[AMV+07] J. Alastruey, T. Monreal, V. Viñals and M. Valero. “Microarchitectural Support for Speculative Register Renaming”. *IEEE Int. Parallel & Distributed Processing Symp.* (IPDPS 07), March 2007.

[AMC+08] J. Alastruey, T. Monreal, F.J. Cazorla, V. Viñals y M. Valero. Selection of the Register File Size and the Resource Allocation Policy on SMT Processors. *Proceedings of the 20th International Symposium on Computer Architecture and High Performance Computing* (SBAC-PAD 08), Campo Grande, Brasil, Oct 29-Nov 1, 2008. IEEE Computer Society, pp. 63-70, ISBN 978-0-7695-3423-7.

[AMV+09] J. Alastruey, T. Monreal, M. Valero y V. Viñals. "Implementación de un Predictor de Último Uso con Decaimiento". *XX Jornadas de Paralelismo*, La Coruña (España). 17-18 Sept. 2009.

[BaSo03] S. Balakrishnan and G. Sohi. Exploiting Value Locality in Physical Register Files. *In Proceedings of the 36th Annual IEEE/ACM international Symposium on Microarchitecture* (MICRO 03), pp. 265. San Diego (USA), December 03- 05, 2003.

[BDA01b] R. Balasubramonian, S. Dwarkadas, and D. H. Albonesi, Reducing the Complexity of the Register File in Dynamic Superscalar Processors. *Proceedings of the 34th International Symposium on Microarchitecture* (MICRO 01), pp. 237-249, December 2001

[BSP+06a] D. Balkan, J. Sharkey, D. Ponomarev and A. Aggarwal, “Address-Value Decoupling for Early Register Deallocation”, Proc. 35th Int’l Conf. on Parallel Processing (ICPP 06), Aug. 2006, pp. 337-346.

[BSP+06b] D. Balkan, J. Sharkey, D. Ponomarev and K. Ghose. SPARTAN: Speculative Avoidance of Register Allocation to Transient Values for Performance and Energy Efficiency. *Proc. 15th Int’l Conf. Parallel Architectures and Compilation Techniques* (PACT 06), Sept. 2006, pp. 265-274.

[BMK+05] T. N. Buti, R. G. McDonald, Z. Khwaja, A. Ambekar, H. Q. Le, W. E. Burky, and B. Williams. Organization and implementation of the register-renaming mapper for out-of-order IBM POWER4 processors. *IBM J. Res. Dev. 49, 1* , pp. 167-188, Jan. 2005.

[BuSo02a] J.A. Butts and G. Sohi. "Dynamic dead-instruction detection and elimination". *Proc. 10th Int'l Conf. Architectural Support for Programing Languages and Operating Systems* (ASPLOS 02), Oct. 2002, pp. 199-210.

[BuSo02b] J.A. Butts and G. Sohi, Characterizing and Predicting Value Degree of Use", *Proc. 35th Int'l Symp. Microarchitecture* (MICRO 02), Nov. 2002, pp. 15-26.

[BuSo04] J.A. Butts and G. Sohi, Use-Based Register Caching with Decoupled Indexing. *Proc. 31st Int'l Symp. Computer Architecture* (ISCA 02), pp. 302. Munich (Alemania), Jun. 2004.

[CGV+00] J.L. Cruz, A. González, M. Valero and N.P. Topham. Multiple-Banked Register File Architectures. *Proc. 27th Int'l Symp. Computer Architecture* (ISCA 00), June 2000, pp. 316-325.

[EBP+04] O. Ergin, D. Balkan, D. Ponomarev and K. Ghose, "Increasing Processor Performance Through Early Register Release". *In Proceedings of the IEEE international Conference on Computer Design* (ICCD 04), pp. 480-487, San Jose (USA), October 11-13, 2004.

[EBG+04] O. Ergin, D. Balkan, K. Ghose, D. Ponomarev. Register Packing: Exploiting Narrow-Width Operands for Reducing Register File Pressure. *In Proceedings of 37th International Symposium on Microarchitecture* (MICRO 04), pp. 304-315, 04-08 Dec. 2004.

[EBP+06] O. Ergin, D. Balkan, D. Ponomarev and K. Ghose, "Early Register Deallocation Mechanisms Using Checkpointed Register Files", *IEEE Transactions on Computers*, vol. 55, no. 9, Sept. 2006, pp. 1153-1166.

[GCO+04] R. González, A. Cristal, D. Ortega, A. Veidenbaum and M. Valero. A Content Aware Integer Register File Organization. *In Proceedings of the 31st Annual international Symposium on Computer Architecture* (ISCA 04), p. 314. München, Germany, June 19-23, 2004.

[Gwe97] L. Gwennap, MIPS R12000 to Hit 300 MHz. *Microprocessor Report, Micro Design Resources*, vol. 11, no. 13, pp. 1-4, October 1997.

[HePa03] J.L. Hennessy, and D.A. Patterson, *Computer Architecture, A Quantitative Approach*, Morgan Kaufmann Publishers, San Francisco, Third Edition 2003.

[HSU+01] G. Hinton, D. Sager, M. Upton, D. Boggs, D. Carmean, A. Kyker, and P. Roussel.. The Microarchitecture of the Pentium 4 Processor. *Intel Technology Journal Q1*, February 2001.

[HwPa87] W.W. Hwu, and Y.N. Patt, Checkpoint Repair for Out-of-Order Execution Machines. *Proceedings of the 14th International Symposium on Computer Architecture* (ISCA 87), pp. 18-26, June 1987.

[JOA+05] T.M. Jones, M.F.P. O'Boyle, J. Abella, A. González, and O. Ergin, "Compiler Directed Early Register Release", *Proc. 14th Int'l Conf. on Parallel Architectures and Compilation Techniques* (PACT 05), pp. 110-122. Sept. 2005.

[JRB+98] S. Jourdan, R. Ronen, M. Bekerman, B. Shomar, A. Yoaz. A Novel Renaming Scheme to Exploit Value Temporal Locality through Physical Register Reuse and Unification", *in Proc. of 31st. Int. Symp. on Microarchitecture* (MICRO 98), pp. 216-225, 1998.

[Kes99] R.E. Kessler, The Alpha 21264 Microprocessor. *IEEE Micro*, vol. 19, no. 2, pp. 24-36, March/April 1999.

[KoNa05] M. Kondo and H. Nakamura. A Small, Fast and Low-Power Register File by Bit-Partitioning. *In Proceedings of the 11th international Symposium on High-Performance Computer Architecture* (HPCA 05), pp. 40-49. February 12-16, 2005.

[KEP+05] G. Kucuk, O. Ergin, D. Ponomarev, K. Ghose. Reducing power dissipation of register alias tables in high-performance processors. *IEE Proceedings on Computers and Digital Techniques*, Volume 152, issue 6, pp.739-746. Nov. 2005.

[LMG04] M. Lipasti, B. Mestan and E. Gunadi, Physical Register Inlining. *In Proceedings of the 31st Annual international Symposium on Computer Architecture* (ISCA 04), pp. 325-335. München, Germany, June 19-23, 2004.

[Lip92] J.S. Liptay. Design of the IBM Enterprise System/9000 high-end processor. *IBM Journal of Research and Development*, 36, 4, 713. 1992.

[LPE+99] J. Lo, S. Parekh, S. Eggers, H. Levy, D. Tullsen. Software-Directed Register Deallocation for Simultaneous Multithreaded Processors. *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 9, pp. 922-933, September, 1999.

[LoGa95] L. Lozano, G. Gao. Exploiting short-lived variables in superscalar processors. *Proceedings of the 28th Annual International Symposium on Microarchitecture* (MICRO 95), pp. 293-302. 1995.

[MRF97] M. Martin, A. Roth, C. Fisher. Exploiting Dead Value Information. *Proceedings of the 30th Annual International Symposium on Microarchitecture* (MICRO 97), pp. 125-135. 1997.

[MRH+02] J. Martinez, J. Renau, M. Huang, M. Prvulovich, J. Torrellas, Cherry: Checkpointed Early Resource Recycling in Out-of-order Microprocessors. *Proc. 35th Int'l Symp. Microarchitecture* (MICRO 02), Nov. 2002, pp. 3-14.

[MGV+99] T. Monreal, A. González, M. Valero, J. González and V. Viñals, "Delaying Physical Register Allocation Through Virtual-Physical Registers", *Proc. 32nd Int'l Symp. Microarchitecture* (MICRO 99), Nov. 1999, pp.186-192.

[MGV+00a] T. Monreal, A. González, M. Valero, J. González, and V. Viñals, Dynamic Register Renaming Through Virtual-Physical Registers. *In The Journal of Instruction-Level Parallelism*, vol. 2, May 2000. http://www.jilp.org/vol2.

[MVG+02] T. Monreal, V. Viñals, A. González and M. Valero. Hardware Schemes for Early Register Release. *Proc. Int'l Conf. Parallel Processing* (ICPP 02), Aug. 2002, pp. 5-13.

[MVG+04] T. Monreal, V. Viñals, J. González, A. González and M. Valero, "Late Allocation and Early Release of Physical Registers", *IEEE Transactions on Computers*, vol. 53, no. 10, Oct. 2004, pp. 1244-59.

[MoSs97] A. Moshovos and G. Sohi. Streamlining inter-operation memory communication via data dependence prediction. *In Proceedings of the 30th Annual ACM/IEEE international Symposium on Microarchitecture* (MICRO 97), pp. 235-245. North Carolina (United States), December 01-03, 1997.

[Mos03] A. Moshovos. Checkpointing Alternatives for High Performance, Power-Aware Processors. IEEE International Symposium on Low Power Electronic and Design (ISLPED 03), pp. 318 -321, Aug. 2003.

[MPV93] M. Moudgill, K. Pingali and S. Vassiliadis. Register Renaming and Dynamic Speculation: an Alternative Approach. *Proc. 26th Int'l Symp. Microarchitecture* (MICRO 93), Nov. 1993, pp. 202-213.

[OBM+05] D. Oehmke, N. Binkert, T. Mudge and S. Reinhardt. How to Fake 1000 Registers. *In Proceedings of the 38th Annual IEEE/ACM international Symposium on Microarchitecture* (MICRO 05), pp. 7-18. Barcelona, Spain, November 12-16, 2005.

[PJS96] S. Palacharla, N. P. Jouppi, and J. E. Smith. Quantifying the complexity of superscalar processors. Technical Report CSTR-1996-1328, Computer Sciences Dept., University of Wisconsin-Madison, 1996.

[PBR02] V. Petric, A. Bracy and A. Roth. Three extensions to register integration. *In Proceedings of the 35th Annual ACM/IEEE International Symposium on Microarchitecture* (MICRO 02). Istanbul, Turkey, November 18 - 22, 2002.

[PSR05] V. Petric, T. Sha and A. Roth. RENO: a rename-based instruction optimizer. *In Proceedings of the 32nd Annual International Symposium on Computer Architecture* (ISCA 05), pp. 98-109. Madison (USA), June 4-8, 2005.

[PKE+03] D. Ponomarev, G. Kucuk, O. Ergin, K. Ghose. Reducing Datapath energy Through the Isolation of Short-Lived Operands. *Proceedings of the 12th International Conference on Parallel Architectures and Compilation Techniques* (PACT 03), pp. 258.

[PGR+01] M. Postiff, D. Greene, S. Raasch and T. Mudge. Integrating superscalar processor components to implement register caching. *In Proceedings of the 15th international Conference on Supercomputing* (ISCS 01). Sorrento, Italia, 2001.

[Rot07] A. Roth, Physical Register Reference Counting. *IEEE Computer Architecture. Letters*, vol. 7, issue 1, pp. 9-12. Jan. 2008.

[SAM+07] E. Safi, P. Akl, A. Moshovos, A. Veneris, and A. Arapoyianni, On the latency, energy and area of checkpointed, superscalar register alias tables. *In Proceedings of the 2007 international Symposium on Low Power Electronics and Design* (ISLPED 07), pp. 379-382. Portland (USA), August 27-29, 2007.

[San06] R. Sangireddy. Reducing Rename Logic Complexity for High-Speed and Low-Power Front-End Architectures. IEEE Transactions on Computers, vol. 55, issue 6, pp- 672-685. June 2006.

[SRG02] G. Savransky, R. Ronen, A. Gonzalez. Lazy Retirement: A Power Aware Register Management Mechanism. WCED 2002.

[ShPo07] J. Sharkey and D. Ponomarev. An L2-miss-driven early register deallocation for SMT processors. *In Proceedings of the 21st Annual international Conference on Supercomputing* (ICS 07), pp. 138-147. Seattle (USA), June 17-21, 2007.

[ShLi04] J. P. Shen, M. H. Lipasti, *Modern Processor Design*, McGraw-Hill, First Edition, 2004.

[Sim00] D. Sima, The Design Space of Register Renaming Techniques. *IEEE Micro*, vol. 20, no. 5, pp. 70-83, September-October 2000.

[Sim08] D. Sima, Register Renaming Techniques. *Digital Systems and Aplications (segunda edición)*. Capítulo 2. Editor: V. Oklobdzija. CRC Press. ISBN: 978-0-8493-8619-0, 2008.

[SmPl88] J.E. Smith, and A.R. Pleszkun. Implementing Precise Interrupts in Pipelined Processors. *IEEE Transactions on Computers*, vol. 37, no. 5, pp. 562-573, May 1988.

[SRA+04] S. Srinivasan, R. Rajwar, H. Akkary, A. Gandhi and M. Upton. Continual flow pipelines. *In Proceedings of the 11th international Conference on Architectural Support For Programming Languages and Operating Systems* (ASPLOS 04), pp. 107-119. Boston, (USA), October 07-13, 2004.

[TNN+04] L. Tran, N. Nelson, F. Ngai, S. Dropsho and M. Huang. Dynamically reducing pressure on the physical register file through simple register sharing. *In Proceedings of the 2004 IEEE International Symposium on Performance Analysis of Systems and Software* (ISPASS 04), pp. 78-87. March 10-12, Austin (USA), 2004.

[TuSe99] D. Tullsen and J. Seng. Storageless value prediction using prior register values. *In Proceedings of the 26th Annual International Symposium on Computer Architecture* (ISCA 99), pp.270-279. Atlanta (USA), May 01-04, 1999.

[WaBa96] S. Wallace, N. Bagherzadeh. A Scalable Register File Architecture for Dynamically Scheduled Processors. *Proc. of the International Conference on Parallel Architectures and Compilation Techniques* (PACT 96). Oct. 1996.

[Yea96] K.C. Yeager, The MIPS R10000 Superscalar Microprocessor. *in IEEE Micro*, vol. 16, no. 2, pp. 28-40, April 1996.

[ZeGh06] H. Zeng, and K. Ghose. Register file caching for energy efficiency. *In Proceedings of the 2006 international Symposium on Low Power Electronics and Design* (ISLPED 06), pp. 244-249. Tegernsee (Germany), October 04-06, 2006.

CAPÍTULO 3

Metodología y Entorno de Experimentación

Este Capítulo describe la metodología y el entorno de trabajo utilizado para evaluar las propuestas de esta Tesis. Primero, se detallan las características del procesador base utilizado. Después, se expone la metodología de simulación, incidiendo en los simuladores utilizados, las cargas de trabajo ejecutadas y las métricas usadas para cuantificar prestaciones.

3.1 Introducción

El trabajo de experimentación de esta Tesis ha consistido principalmente en ejecutar una serie de programas de prueba sobre unos procesadores modelados con una herramienta de simulación. Este es el procedimiento habitual para evaluar propuestas en el campo de Arquitectura de Computadores, ya que por su elevado coste, la utilización de prototipos queda reservada a grandes corporaciones, y solamente en fases avanzadas de diseño.

Existen distintos tipos de herramientas de simulación de procesadores. Los simuladores funcionales modelan el funcionamiento de un procesador abstrayéndose de su comportamiento temporal. Los simuladores temporales, además del funcionamiento, permiten modelar la temporización de un procesador, generalmente a nivel de ciclo. Normalmente, el modelado se realiza con distintos niveles de abstracción, detallando con más precisión los eventos y componentes objeto de estudio. La simulación temporal de un procesador permite un análisis pormenorizado de aspectos como sus prestaciones, índices de rendimiento, ocupación de estructuras... En esta Tesis se ha utilizado simulación funcional para obtener determinadas estadísticas de los programas ejecutados y simulación temporal ciclo a ciclo para evaluar las diferentes modificaciones propuestas sobre el procesador base modelado.

En las siguientes secciones se describen los puntos principales del procedimiento de experimentación empleado en esta Tesis.

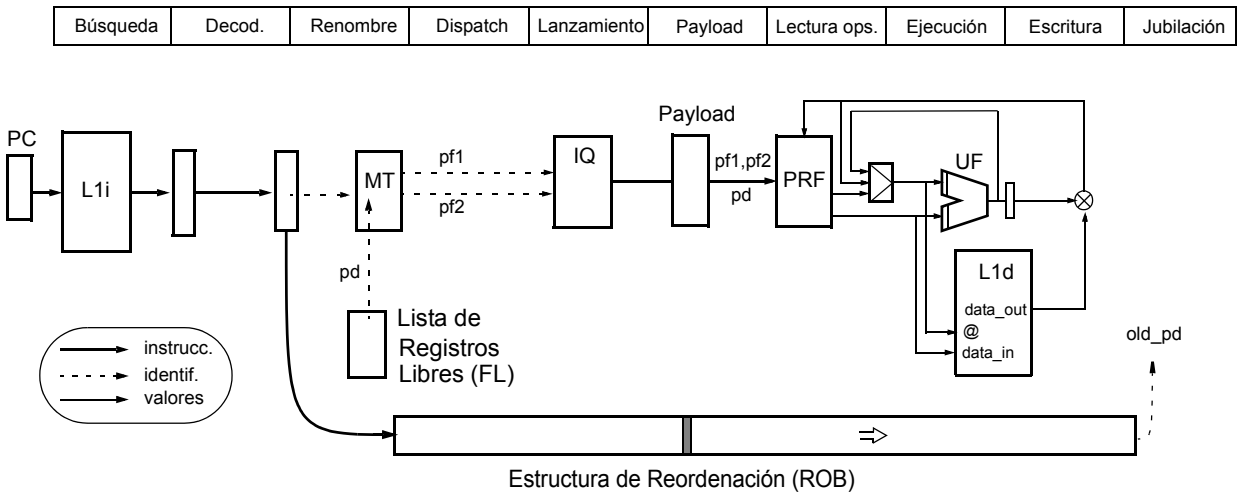


Figura 3.1 Esquema simplificado de la microarquitectura del procesador superescalar con ejecución fuera de orden modelado. Los componentes se han situado debajo de la etapa en la que realizan actividad. Sólo se ha dibujado la red de cortocircuitos correspondiente a un operando fuente.

3.2 Procesador base

Hemos modelado un procesador superescalar basándonos en procesadores comerciales existentes en el mercado y en la línea de otros trabajos de investigación. Consta de un segmentado de 10 etapas, donde se puede distinguir una parte en orden (*front-end*) y un núcleo de ejecución fuera de orden (*back-end)*. La Figura 3.1 muestra un esquema del procesador modelado.

La parte en orden consta de cuatro etapas: búsqueda de instrucciones, decodificación, renombre y emisión (*dispatch*) de las instrucciones al núcleo de ejecución fuera de orden. El núcleo de ejecución fuera de orden se compone de seis etapas: lanzamiento de instrucciones, lectura de información relevante para la ejecución (*payload*), lectura de operandos, ejecución, escritura y jubilación. A continuación se reseñan los aspectos cualitativos más importantes de la microarquitectura modelada.

3.2.1 Renombre de registros

El mecanismo de renombre de registros convencional se basa en un banco de registros físico (PRF) que contiene tanto los registros de la arquitectura como los de renombre, en una tabla de mapeo (MT) y en una lista de registros físicos libres (FL). En cada salto se realiza una copia (*checkpoint*) de la tabla de mapeo, lo que permite su recuperación en caso de que un salto sea mal predicho. La estructura de reordenación (ROB) almacena la información necesaria para restaurar el estado preciso en caso de que se produzca alguna excepción.

Los registros liberados en un ciclo no pueden reasignarse hasta el ciclo siguiente, es decir, los registros liberados en el ciclo i no son visibles en la FL hasta el ciclo i+1.

3.2.2 Núcleo fuera de orden (*back-end*)

El lanzamiento de una instrucción consta de dos fases: despertar de instrucciones preparadas (*wake-up*) y selección de instrucciones (*select*). En la primera fase, una matriz de dependencias basada en identificadores de registros físicos permite identificar las instrucciones que tienen sus operandos fuente listos, y que por tanto, podrían comenzar su ejecución. En la segunda fase, la lógica de selección determina cuáles de esas instrucciones van a ser lanzadas. La política convencional de selección escoge las instrucciones preparadas más viejas.

Para que el procesador soporte ejecución encadenada (*back-to-back*) es necesario añadir una red de cortocircuitos (*bypass network*) que suministra los resultados que todavía no han sido escritos en el PRF. Además, el despertar de una instrucción no debe realizarse cuando sus registros fuente ya han sido escritos, sino que se adelanta para que cuando una instrucción lanzada llegue a la etapa de lectura del banco de registros pueda leer de la red de cortocircuitos el operando producido en el mismo ciclo o en el anterior. Con nuestro modelo de segmentado, esto implica que la red de cortocircuitos debe suministrar los resultados generados al final de la etapa de ejecución y al final de la etapa de escritura en el PRF (ver Figura 3.1).

Las instrucciones dependientes de una instrucción con latencia no determinista, se despiertan de forma especulativa, apostando a que sus operandos fuente serán suministrados a tiempo. Por ejemplo, cuando una lectura de memoria (load) se lanza, sus instrucciones dependientes son planificadas considerando que el acceso a la cache de primer nivel de datos (L1D) será un acierto. A este esquema se le llama predicción de latencia y exige que un load y sus instrucciones dependientes permanezcan en la ventana de lanzamiento (IQ) hasta que se confirme el acierto de L1D. En caso de fallo de cache, y por tanto, fallo en la predicción de latencia, un mecanismo selectivo de recuperación en cadena anula tanto el load como sus instrucciones dependientes [TIV+03]. Las instrucciones anuladas son posteriormente relanzadas cuando estén preparadas.

La RAM accedida en la etapa de *payload* tiene una entrada por cada instrucción que puede ser seleccionada para ejecutarse. Cada entrada contiene la información que requiere una instrucción para ejecutarse (código de operación, registros físicos fuente y destino, operandos inmediatos...).

3.2.3 Parámetros del procesador modelado

El procesador simulado es un superescalar de grado 8 con capacidad para ejecutar instrucciones fuera de orden. Las características principales de la microarquitectura modelada se muestran en la Tabla 3.1.

Hemos escogido una ventana de lanzamiento razonablemente grande para que los resultados no se vean limitados por el tamaño de la ventana de lanzamiento.

Tabla 3.1 Configuración del procesador base modelado.

Estructura Parámetro	Configuración Valor	Estructura Parámetro	Configuración Valor
Ancho búsqueda, decodificación y renombre	8 instrucciones.	L1 I-cache	64 KB, asociatividad 4, líneas de 32 bytes, tiempo de acierto: 1 ciclo
Predictor de saltos	Predictor híbrido: bimodal + gshare 16-bits con actualización especulativa	L1 D-cache	32 KB, asociatividad 2, líneas de 64 bytes, tiempo de acierto: 2 ciclos
Unidades Funcionales (*latencia*)	8 int sencillas (*1*), 2 mult int (*7*), 4 load/store, 4 fp sencillas (*4*), 4 fp mult/div (*4/16*)	L2 U-Cache	256 KB, asociatividad 8 líneas de 128 bytes, tiempo de acierto: 10 ciclos
Reorder Buffer	256 entradas	L3 U-Cache	4 MB, asociatividad 4, líneas de 128 bytes, tiempo de acierto: 16 ciclos
Ventana Lanzam.	int+mem: 64 entradas; fp: 32 entradas		
Cola Load/Store	128 entradas, store-load *forwarding*	Memoria princ.	ilimitada, 200 ciclos
Ancho lanzamiento	8int + 4fp	TLB	256 entradas, asociatividad 4
Mecanismo lanzamiento	Fuera de orden. Los loads se lanzan cuando se conocen las direcciones de todos los stores previos	PRF	36-288 int / 36-288 fp (32 int / 32 fp lógicos)
Ancho jubilación	8 instrucciones	XRF	32-288 int /32-288 fp tiempo de acceso: 3 ciclos

3.3 Simuladores

Se han utilizado dos herramientas de simulación: *simplescalar* [BuAu97] para la evaluación de modelos de procesador fuera de orden *single-thread* y *smtsim* [TEL95] para la evaluación de procesadores con capacidad para ejecutar múltiples threads de forma simultánea (*Simultaneous Multithread,* SMT).

3.3.1 simplescalar

El entorno de trabajo para la evaluación y el análisis de procesadores *single-thread* se ha basado en simplescalar v3.0d, una herramienta de simulación guiada por ejecución que soporta el juego de instrucciones Alpha. A partir del simulador funcional *sim-safe* se han generado dos simuladores, uno para analizar características de los códigos ejecutados y otro para generar las trazas con los operandos último uso de cada instrucción ejecutada. El simulador temporal *sim-outorder* ha sido la base sobre la que se ha modelado el procesador de referencia y las modificaciones que se propone incorporar.

El simulador detallado ciclo a ciclo se ha modificado para incluir las estructuras y mecanismos que se han reseñado en la sección 3.2 (banco de registros físico entero y de coma flotante, mecanismo de renombre de registros, ventana de lanzamiento desacoplada del ROB, predicción de latencia para despertar especulativo, mecanismo selectivo de recuperación en cadena para fallos de predicción de latencia con reejecución selectiva...).

3.3.2 smtsim

El procesador SMT se ha modelado a partir de una versión mejorada del simulador smtsim desarrollada en la Universidad Politécnica de Cataluña (UPC) y el *Barcelona Supercomputing Center* (BSC) por F. J. Cazorla. Las diferencias respecto a la versión original son:

- Es guiado por traza en lugar de por ejecución. Sin embargo, tiene capacidad para ejecutar instrucciones por caminos incorrectos.
- Se ha eliminado soporte de memoria compartida.

Como parte del trabajo de esta Tesis, se ha modificado el simulador recibido. Se ha implementado el mecanismo de renombre convencional. También se han modelado con detalle varias política de asignación de recursos. En el Capítulo 7 se describen los cambios realizados.

3.4 Cargas de Trabajo

Como carga de trabajo se ha seleccionado SPEC CPU2000, un conjunto de programas de cálculo intensivo. Se han utilizado todos los programas de la serie de cálculo intensivo con datos enteros (SPEC **C**PU **INT** 2000, abreviado como SPEC CINT2000) y todos los de cálculo intensivo con datos de coma flotante (SPEC **C**PU **FP**, abreviado como SPEC CFP2000). En las ejecuciones de cada *benchmark* se emplea su entrada de referencia (*ref*). Se ha escogido una única configuración de ejecución en aquellos programas para los cuales las reglas de SPEC CPU2000 definen varias ejecuciones con distintas entradas de referencia o parámetros. El listado de programas con las entradas seleccionadas se muestra en la Tabla 3.2.

Se han utilizado unos binarios Alpha compilados y validados por Chris Weaver (Universidad de Michigan). Estas versiones de los programas de SPEC CPU2000 están enlazadas desde el sitio web de simplescalar:

`http://www.simplescalar.com/benchmarks.html`

Los binarios se compilaron con el máximo nivel de optimización permitido por las reglas de SPEC CPU2000 (*peak*) y se verificó que su ejecución sobre simplescalar cumple dichas normas[1]. Las características principales del entorno de compilación se muestran en la Tabla 3.3.

Para reducir el tiempo de simulación se ha explotado la repetitividad temporal en el comportamiento de los programas. En lugar de ejecutar los programas completos de principio a fin, se ha ejecutado su fase más representativa. Este procedimiento permite realizar análisis de programas y simulaciones de modelos de procesador de forma eficiente y precisa. En esta Tesis, se ha simulado la ejecución de secuencias de 100 millones de instrucciones previo calentamiento de caches y predictores durante 200 millones de instrucciones. El inicio de estos intervalos de simulación en los binarios Alpha está marcado por los *Simpoints* enumerados por Sherwood et al. [SPH+02].

1. SPEC define un procedimiento para verificar que los binarios compilados son válidos. Básicamente, se trata de comprobar que la salida del programa sea correcta.

Tabla 3.2 Listado de programas de SPEC CPU2000 junto con sus entradas seleccionadas.

	Programa	Entrada
SPEC CINT2000	bzip	input.program
	crafty	crafty.in
	eon	chair.control.cook
	gap	ref.in
	gcc	166.i
	gzip	input.graphic
	mcf	inp.in
	parser	ref.in
	perl	diffmail.pl
	twolf	ref
	vortex	lendian1.raw
	vpr	net.in arch.in place.in -route_only
SPEC CFP2000	ammp	ammp.in
	applu	applu.in
	apsi	apsi.in
	art	c756hel.in
	equake	inp.in
	facerec	ref.in
	fma3d	fma3d.in
	galgel	galgel.in
	lucas	lucas2.in
	mesa	mesa.in
	mgrid	mgrid.in
	sixtrack	inp.in
	swim	swim.in
	wupwise	wupwise.in

La simulación bajo Simplescalar de los intervalos de ejecución definidos por los *Simpoints* para los binarios Alpha constituye un procedimiento de análisis y evaluación de propuestas ampliamente utilizado por la comunidad investigadora, lo que produce una cierta consistencia en los resultados generados por distintos trabajos.

Para poder utilizar de forma eficaz los *Simpoints*, es necesario que las herramientas de simulación permitan comenzar la ejecución de un programa en un punto específico del mismo. A partir de la versión 3.0, todos los simuladores de Simplescalar soportan la ejecución de un programa a partir de un único fichero que captura el estado del programa en un punto determinado (*checkpoint* de memoria y registros). Este fichero, que recibe el nombre de traza EIO (traza de entrada/salida externa, *external I/O trace*), puede incluir también un registro con las

Tabla 3.3 Características principales del entorno de compilación de los benchmarks SPEC CPU 2000 para la arquitectura Alpha AXP.

Hardware	AlphaServer ES40, 21264@500MHz, 4GB RAM, 6 GB swap
Sistema Operativo	Digital UNIX V4.0F (Rev. 1229)
Compilador C	cc DEC C V5.9-008
Compilador C++	cxx Compaq C++ V6.2-024
Compilador Fortran 90	f90 Compaq Fortran V5.3-915
Compilador Fortran 77	f77 Compaq Fortran V5.3-915

interacciones entre el programa y el sistema operativo. Esto último permite que las ejecuciones a partir de trazas EIO sean reproducibles, pues las fuentes de posibles variaciones como por ejemplo, llamadas al sistema operativo, son capturadas y servidas por la traza EIO. Las simulaciones funcionales y temporales de esta Tesis se han realizado con las trazas EIO generadas por E. Torres durante el trabajo de experimentación de [Tor05].

Actualmente, en el seno del grupo de investigación (gaZ) se está en proceso de validación de las trazas EIO correspondientes a todos los programas de SPEC CPU2006.

También se ha utilizado Olden en algunos experimentos, como por ejemplo, en la caracterización de patrones de uso. Los resultados han sido muy similares a los obtenidos con SPEC CPU2000.

3.5 Métricas

Se han utilizado varias métricas para cuantificar las prestaciones:

- Instrucciones Por Ciclo (IPC): número medio de instrucciones *jubiladas* por ciclo. Se calcula dividiendo el número total de instrucciones jubiladas entre el número de ciclos de ejecución. Este número es menor que el número de instrucciones *ejecutadas* por ciclo a causa de las instrucciones anuladas por la especulación de control o latencia.

- Instrucciones Por Segundo (IPS): número medio de instrucciones *jubiladas* por segundo. Como su número es elevado se suele expresar como BIPS (Billones[2] de Instrucciones por Segundo).

Como ambas métricas hacen referencia a velocidad (trabajo realizado por unidad de tiempo), y todos los programas se ejecutan el mismo número de instrucciones, el promedio de IPC e IPS para varios programas se calcula mediante la media armónica [Smi88]. Otras estadísticas, por ejemplo, número de instrucciones con una determinada característica, son promediadas mediante la media aritmética.

2. Se refiere a los billones americanos, es decir, 1000 millones y no a los billones europeos.

Otra métrica que se ha empleado, y que es de utilización extensiva en el campo Arquitectura de Computadores es la ganancia de velocidad, más conocida como *speedup*. El *speedup* muestra la mejora de IPC relativa a un IPC de referencia:

$$\text{speedup} = \frac{\text{IPC}}{\text{IPC}_{\text{ref}}}$$

Esta métrica se utiliza para comparar el rendimiento de una configuración o propuesta con el del procesador base, que sirve de referencia. Generalmente se usa en forma de porcentaje:

$$\text{speedup}(\%) = \frac{\text{IPC} - \text{IPC}_{\text{ref}}}{\text{IPC}_{\text{ref}}} \text{x} 100$$

Si el porcentaje es positivo, la propuesta supone una mejora respecto el procesador base. En caso contrario, las prestaciones se degradan al incorporar la propuesta.

Para caracterizar las prestaciones de un procesador SMT se utilizan otro tipo de métricas más elaboradas, como por ejemplo IPCs agregados o medias armónicas de IPCs relativos. La descripción de dichas métricas se realiza en el Capítulo 7.

3.6 Coste de una implementación

Las propuestas de mejora de una microarquitectura suelen suponer un coste en términos de almacenamiento, área y energía. Por otra parte, las estructuras añadidas o modificadas pueden tener un tiempo de acceso que afecte al tiempo de ciclo del procesador.

El análisis de los tiempos de acceso a estructuras como el PRF, así como sus costes de área y energía se han obtenido a partir del modelo de banco de registros centralizado de Rixner et al. [RDK+00]. Este trabajo presenta una serie de modelos para distintas organizaciones de bancos de registros (centralizado, SIMD, distribuido, jerárquico, *stream*) que permiten calcular área, latencia y consumo dinámico para una tecnología de 180 nm. en función de parámetros como el número de entradas y puertos.

3.7 Plataformas de Simulación

Los experimentos se han realizado sobre los *clusters* de PCs del Departamento de Informática e Ingeniería de Sistemas (atpc, atps[3]) y del Instituto de Investigación en Ingeniería de Aragón (hermes). Condor ha sido la herramienta utilizada para repartir la carga de trabajo entre las máquinas de los citados *clusters* [LLM88].

3. El lector malpensado podrá descifrar ambos acrónimos sin ninguna dificultad.

3.8 Referencias

[BuAu97] D. Burger and T.M. Austin, The Simplescalar Tool Set v2.0, Technical Report 1342, Computer Science Dept., University of Wisconsin-Madison, June 1997.

[HePa03] J.L. Hennessy, and D.A. Patterson, *Computer Architecture, A Quantitative Approach*, Morgan Kaufmann Publishers, San Francisco, Third Edition 2003.

[LLM88] M. Litzkow, M. Livny, and M. Mutka. Condor - A Hunter of Idle Workstations. *Proceedings of the 8th International Conference of Distributed Computing Systems*, pages 104-111, June, 1988.

[RDK+00] S. Rixner, W.J. Dally, B. Khailany, P. Mattson, U.J. Kapasi, and J.D. Owens, Register Organization for Media Processing. P*roceedings of the 6th International Symposium on High-Performance Computer Architecture* (HPCA 00), pp. 375-386, January 2000.

[SPH+02] T. Sherwood, E. Perelman, G. Hamerly and B. Calder. Automatically Characterizing Large Scale Program Behavior. *Proc. 10th Int'l Conf. Architectural Support for Programing Languages and Operating Systems* (ASPLOS 02), pp. 45-57, Oct. 2002.

[Smi88] J.E. Smith. Characterizing computer performance with a single number. Communications of the ACM, vol 31, issue 10, pp. 1202-1206. October 1988.

[TIV+03] E. Torres, P. Ibáñez, V. Viñals and J.M. Llabería. Counteracting Bank Misprediction in Sliced First-Level Caches. *In proc. 9th Int'l Conf. on Parallel and Distributed Computing (Euro-Par 03)*, pp. 586-596. Aug. 2003.

[Tor05] E. Torres. Alternativas de Diseño en Memoria Cache de Primer Nivel Multibanco. Tesis Doctoral. Dpto. Informática e Ingeniería de Sistemas, Universidad de Zaragoza. Junio 2005.

[TEL95] D.Tullsen, S. Eggers and H. Levy. Simultaneous multithreading: Maximizing on-chip parallelism. Int'l Symp. on Computer Architecture (ISCA 95), pp. 392-403, 1995.

CAPÍTULO 4

Renombre de Registros Especulativo basado en Predicción de Último Uso

El objetivo de este Capítulo es analizar el potencial de un nuevo mecanismo de renombre de registros basado la predicción de último uso. Primero, se señalan las ineficiencias de la estrategia de renombre convencional. A continuación, se presentan varias ideas que permiten mejorar la gestión de los registros físicos y que constituyen la base de un agresivo mecanismo de renombre de registros con políticas de asignación y liberación de registros físicos basadas en la predicción de último uso de un registro. Seguidamente se muestra de forma experimental la viabilidad de la predicción de último uso y se describen las características de este tipo de predicción y sus efectos en la ejecución fuera de orden. Después se detalla el banco de registros auxiliar, la otra estructura clave que da soporte a la nueva estrategia de renombre. Finalmente se evalúa el potencial del mecanismo cuantificando la mejora del rendimiento que obtiene un predictor oráculo de último uso.

4.1 Introducción

El mecanismo convencional de renombre de registros asigna un registro físico en la etapa de renombre a cada instrucción con registro lógico destino y no lo libera hasta la etapa de jubilación de la siguiente instrucción que escribe en el mismo registro lógico. Esta estrategia, usada en procesadores como el Alpha 21264, MIPS R10K/12K e Intel Pentium 4, tiene una implementación sencilla que soporta tanto especulación de control como recuperación frente a excepciones. Sin embargo, la política de liberación es ineficiente ya que retrasa la reutilización de registros físicos. Esto causa detenciones innecesarias si el procesador se queda sin registros físicos mientras algunos de ellos almacenan valores que no van a ser leídos nunca. De la misma forma, la política de asignación también es ineficiente ya que asigna registros físicos a valores que no van a ser leídos nunca.

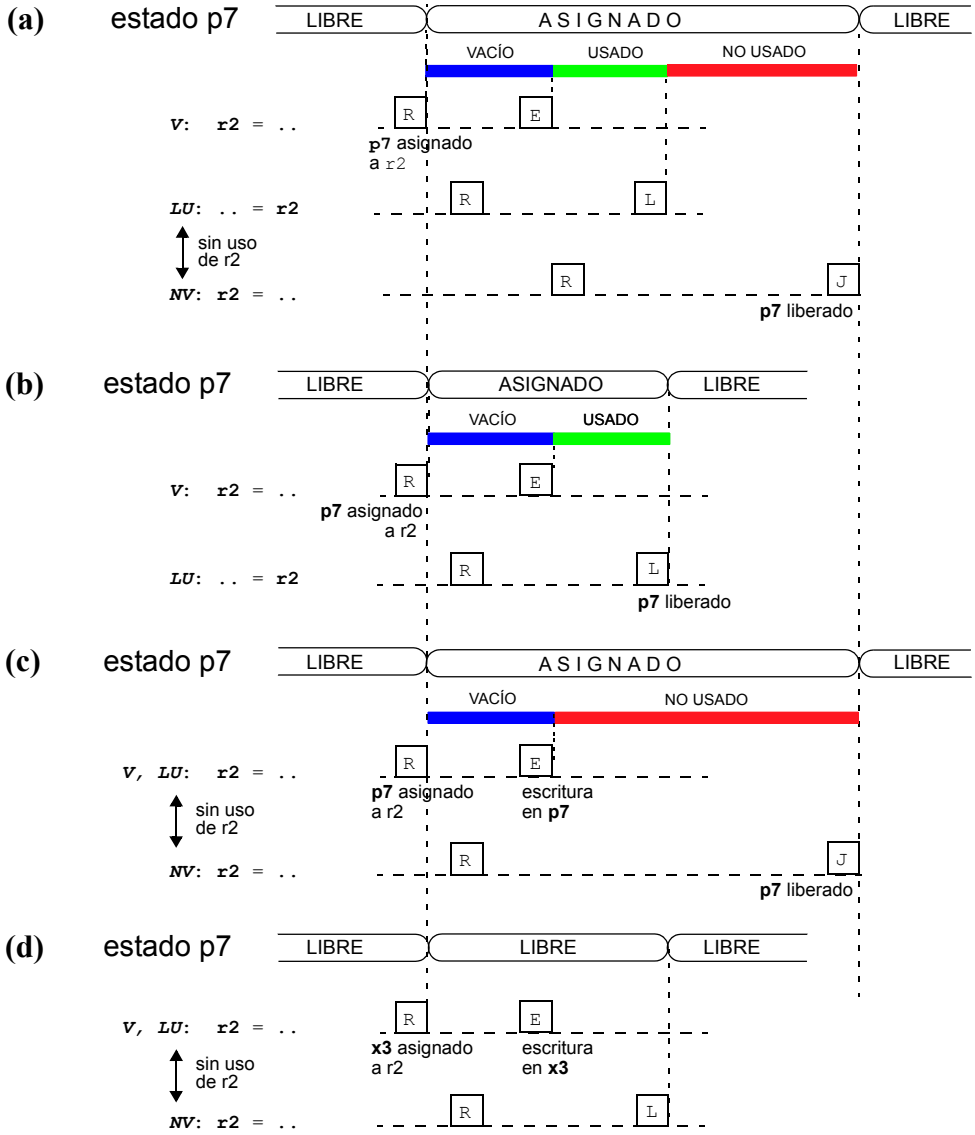


Figura 4.1 Ejemplo de la evolución del estado del registro físico `p7` con el mecanismo de renombre convencional (a) y (c), con una política de liberación anticipada (b) y con otra de omisión de asignación de registro físico (d).
R = Renombre, L = Lectura del PRF, E = Escritura, J = Jubilación.

La Figura 4.1a y la Figura 4.1c muestran dos casos que ilustran las ineficiencias anteriores. Las instrucciones V y NV (*Version* y *Next Version*) definen dos versiones consecutivas del mismo registro lógico destino (r2). LU (*Last Use*) es la instrucción último uso de la versión de r2 creada por V. Decimos que una instrucción es el último uso (*Last Use*, LU) de uno de sus registros cuando es la última instrucción en orden de programa que referencia un valor, ya sea para consumirlo (Figura 4.1a y Figura 4.1b) o para producirlo (Figura 4.1c y Figura 4.1d). Según las políticas convencionales de asignación y liberación, el registro físico p7 es asignado a r2 en la etapa de renombre de V y no es liberado hasta la etapa de jubilación de NV, tal y como muestran la Figura 4.1a y la Figura 4.1c. En ambas figuras también pueden observarse los estados

del registro físico p7 con el mecanismo de renombre convencional. p7 cambia de libre a asignado cuando se renombra la instrucción V, y vuelve al estado libre cuando NV se jubila. Durante los ciclos en que p7 permanece asignado, pasa por otros subestados: está Vacío mientras espera a que sea escrito el resultado de su instrucción productora (V), pasa a Usado desde que deja de estar Vacío hasta que es leído por su última instrucción consumidora (LU), y se mantiene No-Usado el tiempo restante hasta su liberación (ver Figura 4.1a). En el caso de registros físicos sin instrucciones consumidoras, no hay paso por el estado Usado, se produce una transición directa entre los estados Vacío y No-usado (ver Figura 4.1c). Según nuestros experimentos (ver Capítulo 2, sección 2.2.6), el porcentaje medio de registros físicos en estado libre es del 11% para un procesador superescalar de grado 8 ejecutando SPEC CPU2000 con un PRF de 80 entradas (80int+80fp). Además, casi la mitad de los registros asignados están en estado No-Usado, es decir, están almacenando un valor que ninguna instrucción va a leer.

Tal y como se ha explicado en el Capítulo 2, son muchos los trabajos que proponen adelantar la liberación de un registro físico que ha sido leído por todos sus consumidores. Sin embargo, ninguno ha tratado de abstraerse de una implementación concreta y mostrar el potencial de una política de liberación anticipada llevada al límite de reusar un registro físico tan pronto como haya sido leído por su última instrucción consumidora (LU), incluso antes de que haya aparecido la siguiente instrucción que escriba en el mismo el registro lógico destino (NV)[1]. Esta agresiva política de liberación anticipada de un registro físico elimina su estado No-usado, reduciendo el tiempo medio que está asignado un registro físico a uno lógico (ver Figura 4.1b). Para el caso de valores sin instrucciones consumidoras, una acción natural (ya planteada en trabajos previos) es omitir la asignación de registro físico. Esta política permite que continúe en estado libre un registro físico que con la política convencional de asignación se mantendría asignado a un valor muerto durante bastantes ciclos (ver Figura 4.1d).

En este Capítulo vamos a desarrollar uno de los objetivos de esta Tesis: analizar el potencial de un nuevo y agresivo mecanismo de Renombre Especulativo basado en la predicción del último uso de un registro físico (*Speculative Renaming based on Last-Use Prediction*, SR-LUP). Llamamos SR a una estrategia de renombre de registros que permite la omisión en la asignación y la liberación anticipada de registros físicos basándose en algún tipo de predicción. En este caso, la predicción de último uso de un registro, es decir, el intento de identificar aquellos registros cuya siguiente referencia va a ser una escritura, controla tanto la nueva política especulativa de asignación como la de liberación anticipada de registros físicos, y permite reducir el tiempo en que un registro físico está asignado un lógico mediante la eliminación de su estado No-usado.

La naturaleza predictiva de las políticas de SR requiere acciones de recuperación en caso de que se produzca algún fallo de predicción. Una de las estructuras claves de nuestra propuesta es un banco de registros auxiliar (*Auxiliary Register File*, XRF). Cuando un registro es identificado como último uso, su valor asociado se envía al XRF. Más en detalle, las instrucciones a las que no se les asigna registro físico escriben su resultado directamente en el XRF (Figura 4.1d), mientras que los valores liberados anticipadamente son copiados del PRF al XRF (Figura 4.1b). De esta forma, pueden recuperarse en caso de que sean referenciados inesperadamente.

1. Notar que en tal caso, el mapeo entre el registro físico a liberar (p7) y su registro lógico (r2) sigue siendo válido.

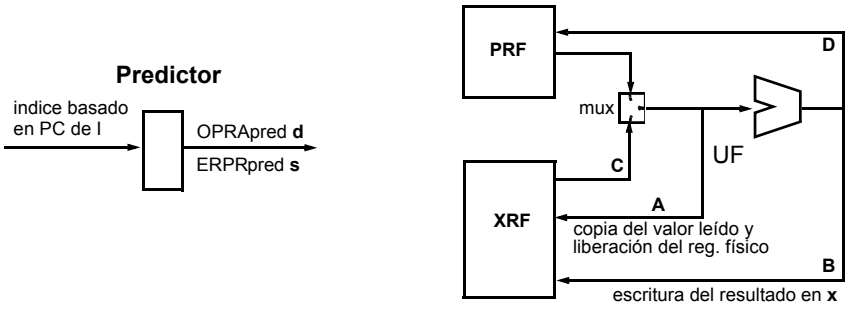


Figura 4.2 Componentes básicos y funcionamiento de un mecanismo de renombre especulativo de registros.

4.2 Renombre de Registros Especulativo

En un esquema de SR, uno o varios registros de una instrucción pueden ser etiquetados con una predicción dirigida a actuar sobre las políticas de asignación y liberación de registros. Más concretamente, los registros fuente pueden recibir una predicción de liberación anticipada de registro físico (*Early Release of Physical Register prediction*, ERPRpred) y el registro destino puede recibir una predicción de omisión de asignación de registro físico (*Omission of Physical Register Allocation*, OPRApred).

En la Figura 4.2 podemos observar el funcionamiento básico de un esquema de SR. Supongamos que el predictor marca al registro de una instrucción I. Si el registro marcado es un operando fuente, una vez leído su valor del PRF[2] se copia en el XRF (circuito A en la Figura 4.2) y se devuelve su identificador físico asociado a la lista de registros libres. Si el registro marcado es un operando destino, el resultado de la instrucción I no se almacena en el PRF (circuito D en la Figura 4.2) y se escribe directamente en el XRF (circuito B en la Figura 4.2). En este último caso, la política de asignación asigna al registro lógico destino de I un registro auxiliar en lugar de un registro físico como haría el esquema convencional.

Un fallo de predicción puede hacer que una instrucción referencie un valor sin registro físico o asociado a un registro físico liberado prematuramente. A estas instrucciones las llamamos usos inesperados (*Unexpected Uses*, UUs). En nuestro diseño, un UU que referencia un valor al que se ha omitido la asignación de registro físico (a causa de una OPRApred incorrecta) leerá su operando del XRF (circuito C en la Figura 4.2), mientras que un UU que referencia un registro físico liberado anticipadamente (por causa de una ERPRpred incorrecta) leerá su operando del PRF si el valor no ha sido sobreescrito o del XRF en caso contrario (circuito C en la Figura 4.2). Para gestionar la ubicación de un valor en varios almacenes es necesario modificar el esquema convencional de gestión de dependencias y localización de operandos. Como en este capítulo estamos interesados en el potencial de la idea, posponemos el diseño de este esquema extendido para el siguiente capítulo, donde se describe detalladamente una microarquitectura con soporte de renombre especulativo.

2. Aunque el esquema sólo refleja lectura del PRF, el operando también puede ser leído de la red de cortocircuitos.

4.3 Renombre Especulativo basado en la Predicción de Último Uso

Proponemos una estrategia de renombre especulativo basada en la predicción de último uso de un registro (*Speculative Renaming based on Last-Use Prediction*, SR-LUP). Una instrucción contiene un último uso de un registro si la siguiente referencia a dicho registro en orden de programa va a ser una escritura. Un registro último uso puede ser fuente (ver Figura 4.1a) o destino (ver Figura 4.1c). En caso de un registro fuente último uso, la instrucción que lo referencia es su última consumidora en orden de programa. Si una instrucción tiene un registro destino último uso, el resultado que genera no va a tener consumidores.

Para cada registro de una instrucción, un predictor señala si es la última instrucción en orden de programa que lo usa. Cuando una instrucción recibe una predicción de último uso para un registro fuente (ERPRpred), SR-LUP etiqueta dicho registro. Posteriormente, en su etapa de lectura de operandos, esa misma instrucción lee el registro etiquetado para enviarlo a la unidad funcional correspondiente. Esa lectura es aprovechada para enviar el valor leído al XRF y liberar anticipadamente el identificador físico del registro identificado como último uso. De esta manera, la copia en el almacén auxiliar no requiere puertos adicionales de lectura en el PRF. Si suponemos que la instrucción LU recibe una predicción de último uso para su registro r2, la Figura 4.1b muestra un ejemplo de esta liberación anticipada. Según nuestros experimentos con SPEC CPU2000, este anticipo tiene un valor medio de 41 y 64 ciclos para los programas enteros y de coma flotante respectivamente. Según medidas realizadas por [BuSo04], para un procesador ejecutando SPEC CINT2000, el valor mediano del adelanto es de 17 ciclos.

Cuando se produzca un uso inesperado de un registro físico liberado anticipadamente, el valor referenciado puede recuperarse del XRF con una penalización en dicho acceso, o del PRF sin penalización dependiendo de si el registro físico ha sido sobreescrito o no.

Una instrucción que recibe una predicción de último uso para su registro destino (OPRApred) escribe su resultado directamente en el XRF y no se le asigna ningún registro físico. De esta forma se evitan escrituras inútiles en el PRF. También se reduce la presión sobre los registros físicos, especialmente en los códigos enteros. Si suponemos que la instrucción LU recibe una predicción de último uso para su registro r2, la Figura 4.1d muestra un ejemplo de esta omisión de asignación. Cuando surja un uso inesperado de un valor al que no se ha asignado registro físico, sólo se podrá recuperar del XRF.

Como consecuencia de esta estrategia de omisión de asignación y de liberación anticipada, los registros físicos ya no identifican inequívocamente un valor, y por tanto, no pueden utilizarse para gestionar las dependencias productor-consumidor. Además, se genera incertidumbre acerca de la ubicación de un valor marcado para ser liberado anticipadamente. En el Capítulo 5 se detalla un mecanismo modificado que permite gestionar las dependencias y la localización de operandos en una microarquitectura que soporta renombre especulativo.

4.4 Último Uso: Definiciones y Caracterización

En esta sección se definen conceptos relativos al último uso (*Last-Use*, LU) de un registro. También se analizan a nivel de instrucción determinadas propiedades relacionadas con el últi-

```
                                          último-uso
                                          s1 s2 d (s=sí, n=no)
      addq      r1, r2 -> r3    ;     n  n  n
      stl       r1 -> 0(r3)     ;     n  n  -
      ldl       0(r3)-> r3      ;     s  -  s
      bne       r2, A           ;     n  -  -     salto tomado
       ...
A:    ldl       8(r1) -> r3     ;     s  -  n
      addq      r3, r2 -> r1
```

Figura 4.3 Ejecución de un fragmento de código cuyas instrucciones muestran distintos patrones de uso. Notar que si el código se ejecuta de nuevo siguiendo el camino no-tomado, el patrón de uso de `ldl 0(r3)-> r3` puede cambiar.

mo uso de sus registros fuente y destino (frecuencia dinámica y predecibilidad) con el objetivo de mostrar la viabilidad de su predicción.

4.4.1 Patrones de uso

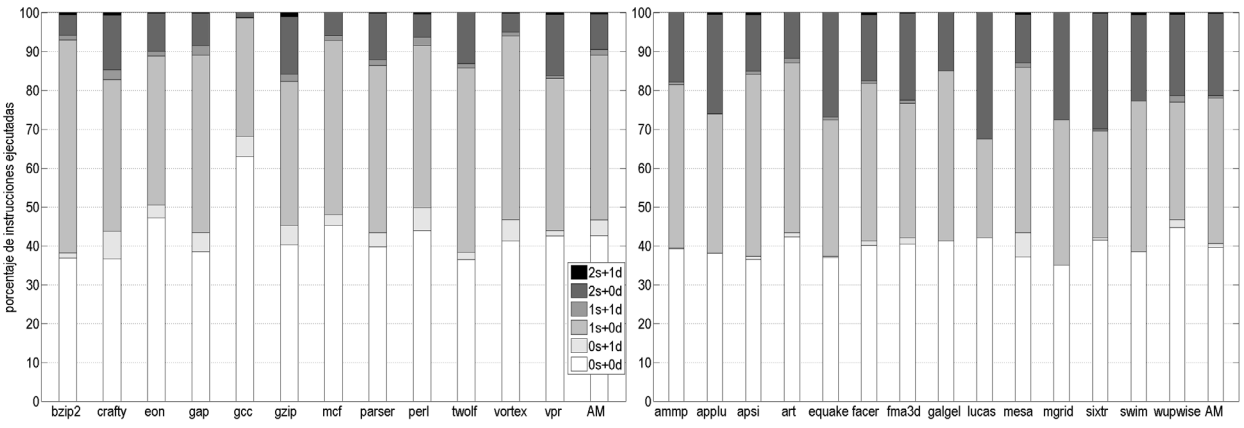
Durante la ejecución de un programa, las instrucciones pueden clasificarse según el uso futuro de sus registros fuente y destino. Un registro fuente (Figura 4.1a) o destino (Figura 4.1c) cuya siguiente referencia sea una escritura puede ser etiquetado como "último-uso". Y al contrario, un registro que tiene varias referencias de lectura antes de ser escrito de nuevo se puede etiquetar como "no es último uso". Utilizando la terminología definida en [BuSo02b], el último uso de un registro fuente se corresponde con el último consumidor de todos los grados de uso mayores que cero, mientras que el último uso de un registro destino se corresponde con el grado de uso cero.

Definimos patrón de uso de una instrucción como el conjunto de etiquetas de todos sus registros, o dicho de otra forma, como la combinación de la información sobre el último uso de sus distintos registros. Diferentes instancias de una misma instrucción estática pueden tener distintos patrones de uso, como ilustra la Figura 4.3.

4.4.2 Frecuencia dinámica de patrones de uso

La Figura 4.4 muestra la distribución de los patrones de uso encontrados en SPEC CPU2000. De acuerdo al número de registros fuente último-uso (0, 1 ó 2) y al número de registros destino último-uso (0 ó 1), distinguimos 6 distintos patrones de uso: 0f+0d, 0f+1d, 1f+0d, 1f+1d, 2f+0d, 2f+1d. No hacemos distinción entre el último uso del registro fuente 1 y 2.

Nuestros experimentos muestran que en media, casi el 60% de las instrucciones tienen un patrón con algún registro último-uso. El "último uso de un registro fuente" (patrones 1f+1d y 1f+0d) se da en el 43% y 38% de las instrucciones ejecutadas (CINT2000 y CFP2000, respectivamente), mientras que el "último uso de ambos registros fuente" se observa en el 9% y 21% de las instrucciones ejecutadas. También es reseñable que para los programas enteros, un 9% de los registros destino almacenan un resultado que no será leído por ninguna instrucción.



(a) SPEC CINT2000 **(b) SPEC CFP2000**

Figura 4.4 Frecuencia dinámica de los patrones de uso en SPEC CPU2000: distribución de las instrucciones ejecutadas según sus patrones de uso. La última columna de cada figura es la media aritmética (AM, *arithmetic mean*).

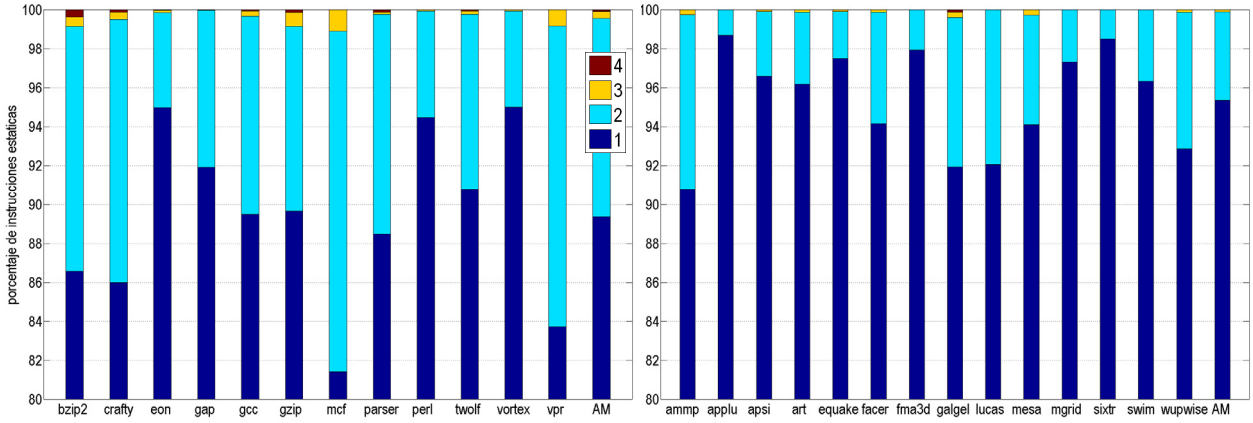
Estos números dan una idea de la continua renovación de los valores almacenados en los registros físicos. También, puede concluirse que la liberación de al menos un registro físico puede avanzarse en más de la mitad de las instrucciones (alrededor del 60%).

4.4.3 Predecibilidad de los patrones de uso

Como se ha mostrado en la Figura 4.3, distintas instancias de la misma instrucción estática pueden cambiar el patrón de uso dependiendo del camino de ejecución. La Figura 4.5 muestra el número de patrones de uso distintos que se han observado para cada instrucción estática. Aquí sí se distingue entre el último uso del registro fuente 1 y 2 (por ejemplo, para instrucciones con "último uso de un registro fuente" se distinguen dos patrones de uso diferentes). Sólo se han contabilizado las instancias jubiladas.

Tanto para CINT2000 como para CFP2000, casi el 100% de las instrucciones estáticas presenta como máximo 2 patrones de uso distintos, y casi el 90% y el 95% de las instrucciones estáticas presenta siempre el mismo patrón de uso para programas enteros y de coma flotante, respectivamente. Por tanto, podemos concluir que los saltos condicionales no afectan demasiado al patrón de uso de las instrucciones estáticas. Este hecho anima al diseño de predictores de último uso de baja complejidad, que pueden basarse en el asociación de patrones de uso a instrucciones estáticas en lugar de a instancias dinámicas. En el siguiente capítulo (Capítulo 5) se presentan y evalúan las prestaciones de dos diseños de predictor de último uso, uno de ellos un diseño sencillo capaz de capturar el comportamiento de los registros con un coste limitado.

El Anexo A extiende el estudio de las propiedades de los patrones de uso. En él se analiza la frecuencia dinámica y predecibilidad de los patrones de uso de ciertas instrucciones (loads, stores). Este estudio completa el análisis de viabilidad de un predictor de último uso sencillo. También puede inspirar predictores más elaborados, por ejemplo, un diseño que considere el tipo de instrucción a la hora de realizar la predicción del patrón de uso de una instrucción.



(a) SPEC CINT2000 **(b) SPEC CFP2000**

Figura 4.5 Predecibilidad de los patrones de uso en SPEC CPU2000: distribución de las instrucciones estáticas según el número de distintos patrones de uso observados en sus instancias jubiladas. La última columna de cada figura es la media aritmética (AM, *arithmetic mean*). Notar el desplazamiento en el eje y.

4.5 Predicción de Último Uso (LUP)

En esta sección se presentan las características de la predicción de último uso y los efectos en la predicción de último uso de la ejecución fuera de orden y de la especulación de control. El diseño de un predictor de último uso (*Last-Use Predictor*, LUP) se realiza en el Capítulo 5.

4.5.1 Características de la predicción de Último Uso

La predicción es una técnica muy utilizada en los procesadores actuales. La literatura recoge muchas propuestas de predictores dedicados a anticipar información que generarán las instrucciones durante su ejecución: sentido de salto, dirección de salto, resultados de operaciones, etc. La predicción de último uso (*Last-Use, LU)* presenta similitudes y diferencias con otros tipos de predicciones. Una gran diferencia reside en las instrucciones objeto de predicción: la mayoría de los predictores sólo almacena información para un tipo concreto de instrucción (saltos, loads...), mientras que el LUP debe asociar un patrón de uso a todas las instrucciones con un registro (las que consumen y/o generan un valor), lo que supone un 97.5% y un 99.5% de las instrucciones que se ejecutan en SPEC CINT 2000 y CFP2000 respectivamente. Por otra parte, de forma similar a los predictores de salto, que generan uno de dos posibles valores (Tomado/No Tomado), el LUP debe generar un patrón compuesto por la indicación de último uso (Sí/No) de cada uno de los registros de una instrucción. Por último, la instrucción que verifica la predicción de último uso de un registro no es la objeto de la misma, sino que depende de si la predicción es correcta o no.

La comprobación de una predicción LU correcta es realizada por la instrucción que redefine el registro lógico objeto de la predicción (Figura 4.6a) mientras que la detección de una predicción incorrecta la realiza la primera instrucción consumidora del registro lógico señalado por la

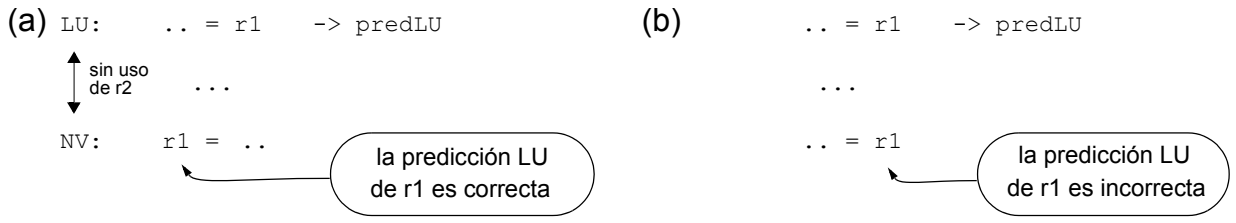


Figura 4.6 Verificación de una predicción de último uso de un operando: verificación de predicción correcta (a) y detección de fallo de predicción (b).

predicción (Figura 4.6b). Por tanto, en el caso general, la verificación de la predicción de un patrón de uso puede ser realizada por varias instrucciones, en concreto por las que redefinen o consumen los registros lógicos cuyos valores han sido objeto de predicción de último uso. En ambos casos, si las verificaciones se efectúan en la etapa de jubilación se evitan detecciones de error o comprobaciones de acierto incorrectamente realizadas por instrucciones de camino incorrecto. El proceso de comprobación de predicciones LU puede llevar muchos ciclos, lo que dificulta el entrenamiento del predictor. Por ejemplo, si suponemos que se realiza una predicción LU correcta en la etapa de decodificación de una instrucción, su comprobación se realizará transcurrido un número medio de ciclos mayor de 41 y 64 para CINT2000 y CFP2000 respectivamente (recordar que ese es el número de ciclos que según nuestros experimentos transcurre desde la lectura de un registro hasta la jubilación de la siguiente instrucción que escribe en el mismo registro lógico, un intervalo menor que el existente entre una predicción LU y su comprobación).

4.5.2 Efectos de la ejecución fuera de orden y la especulación de control en la predicción de último uso

Predecir erróneamente que un registro de una instrucción es último uso fuerza el almacenamiento en el XRF de un valor que puede ser referenciado por alguna instrucción posterior. Estos usos inesperados de un valor (*unexpected uses*, UUs) no son los únicos que pueden aparecer en un procesador con ejecución fuera de orden y especulación de control. Pueden surgir otro tipo de UUs incluso en ausencia de fallos de predicción de último uso. Por otra parte, la ejecución fuera de orden también puede ocultar efectos adversos ligados a los errores de predicción de último uso. A continuación, y con ayuda de la Figura 4.7 y de la Figura 4.8, se describen los efectos del lanzamiento de instrucciones fuera de orden y de la ejecución especulativa en la predicción de último uso.

En la Figura 4.7a, una instrucción correctamente predicha como último uso del registro r1 (predLU-r1) se lanza fuera de orden antes de que una instrucción previa (U) y consumidora del registro r1 lo lea. La instrucción U, cuando se lanza, encontrará que su registro fuente r1 ha sido liberado, lo que la convierte en un uso inesperado fuera de orden (*out-of-order unexpected use*, oooUU). Y al contrario del caso anterior, puede ocurrir que la ejecución fuera de orden oculte fallos de predicción de último uso. En el código de la Figura 4.7b, una instrucción que contiene un registro incorrectamente identificado como último uso (*predLU*) puede que se lance después de otras instrucciones consumidoras de dicho registro (*U*). En tal caso, el fallo de predicción LU no tendría efectos negativos.

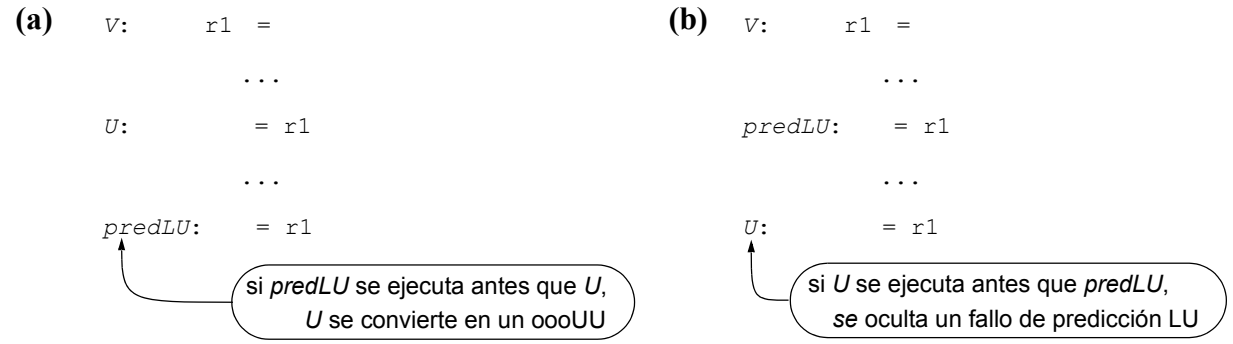


Figura 4.7 Ejemplos del efecto de la ejecución fuera de orden en la predicción de último uso: uso inesperado fuera de orden (*out-of-order unexpected use*, oooUU) (a) y ocultación de fallo de predicción de último uso (b).

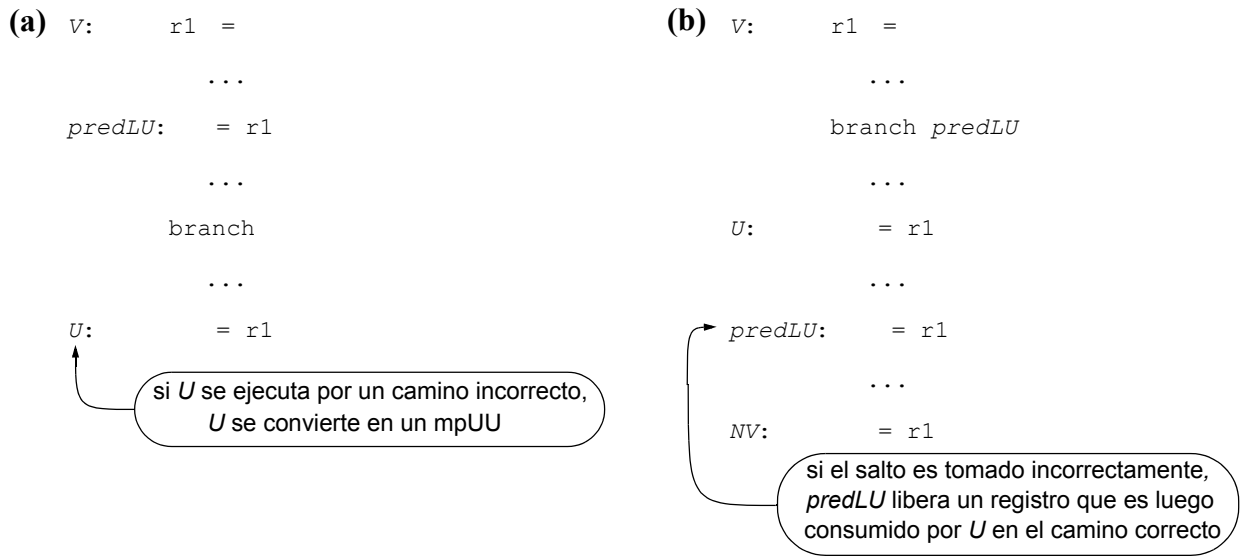


Figura 4.8 Ejemplos del efecto de la especulación de control en la predicción de último uso: uso inesperado por camino incorrecto (*mispredicted path unexpected use*, mpUU) (a) y uso inesperado por liberación anticipada por camino incorrecto (b).

La Figura 4.8a muestra el caso en que un fallo de predicción de salto dirige la ejecución de un programa hacia un camino que usa un valor liberado anticipadamente. A este caso lo llamamos uso inesperado por camino incorrecto (*mispredicted-path unexpected use*, mpUU). En principio, este UU no degrada las prestaciones, ya que afecta solamente a instrucciones de camino incorrecto. Otro caso de uso inesperado puede encontrarse dentro de rutinas de gestión de excepciones, pero su impacto en las prestaciones es despreciable.

La especulación de control también puede convertir predicciones aparentemente correctas en incorrectas. En la Figura 4.8b, puede observarse como un salto tomado lleva el flujo de control a una instrucción que recibe una predicción de último uso supuestamente correcta para uno de sus registros fuente. Sin embargo, si el salto fue incorrectamente predicho, se efectuará la liberación de un registro que será referenciado tras la pertinente recuperación del camino correcto.

Por tanto, el predictor de último uso ha errado, ya que ha liberado un registro físico prematuramente, antes de su última referencia. Estos fallos en la predicción de último uso inducidos por errores en la predicción de salto pueden afectar a las prestaciones, ya que ubican un valor en el XRF, un almacén con mayor latencia de acceso que el PRF. Cuanto mayor sea la precisión del predictor de salto, menor será la frecuencia de este tipo de errores de predicción de último uso.

En la sección 4.7 se muestran las lecturas del XRF que causan las instrucciones oooUUs y mpUUs en un procesador con un esquema de renombre especulativo controlado por un predictor de último uso perfecto.

4.6 Banco de registros auxiliar (XRF)

El XRF tiene dos funciones: almacenar los valores predichos como último uso y suministrar los valores referenciados inesperadamente, aunque a una latencia mayor que la del PRF.

El almacenamiento de registros predichos como último uso puede realizarse mientras existan entradas libres en el XRF. En caso contrario, las acciones asociadas a las predicciones de último uso se anularán, y el mecanismo de renombre actuará como el convencional. En la sección 4.7 se analiza el impacto en las prestaciones de la limitación del número de entradas del XRF. Por otra parte, el número de puertos de escritura del XRF limita la cantidad de instrucciones con operandos último uso que se pueden lanzar en un ciclo. Un XRF con un número de puertos de escritura reducido puede retrasar la ejecución de instrucciones con operandos último uso.

La función de suministro de operandos a las instrucciones UU también está limitada por el número de puertos del XRF. En el caso de un número reducido de puertos de lectura del XRF, se tendría que restringir el número de instrucciones UU que pueden lanzarse en un ciclo. Pero la mayor complejidad asociada al suministro de operandos referenciados inesperadamente reside en el esquema de localización de registros. Un valor liberado anticipadamente no puede ser localizado en el XRF mediante el identificador físico al que estaba asociado. La Figura 4.9a muestra un ejemplo que ilustra esta afirmación. El valor1, que ha sido enviado al XRF, ya no puede referenciarse con el identificador p1, por lo que la instrucción UU ha perdido el enlace al valor1. Tampoco puede ser referenciado con un identificador físico un resultado al que no se le ha asignado registro físico porque su operando destino ha recibido una predicción de último (Figura 4.9b). Es necesario extender el mecanismo de mapeo para permitir la referencia a valores almacenados en el XRF. Así mismo, actualizar la ubicación de un valor cuyo registro ha sido marcado para ser liberado anticipadamente es una tarea que debe ser realizada cuidadosamente. Por ejemplo, si la instrucción predLU actualiza la ubicación de valor1 antes de que sea lanzada a ejecutarse, la instrucción UU buscará el valor1 en el XRF sin encontrarlo, ya que todavía no ha sido realizada la liberación anticipada, y por tanto, no se ha copiado el valor1 en el XRF. Todas estas limitaciones a la hora del diseño del mecanismo de gestión de dependencias y de localización de operandos son consideradas en el Capítulo 5.

En este Capítulo, dado que estamos interesados en el potencial de la idea, supondremos un mecanismo ideal capaz de localizar en el XRF los valores referenciados inesperadamente y suministrarlos con una latencia superior a la del PRF. En el Capítulo 5 se describe un mecanismo de complejidad acotada que permite mantener los enlaces necesarios entre valores liberados anti-

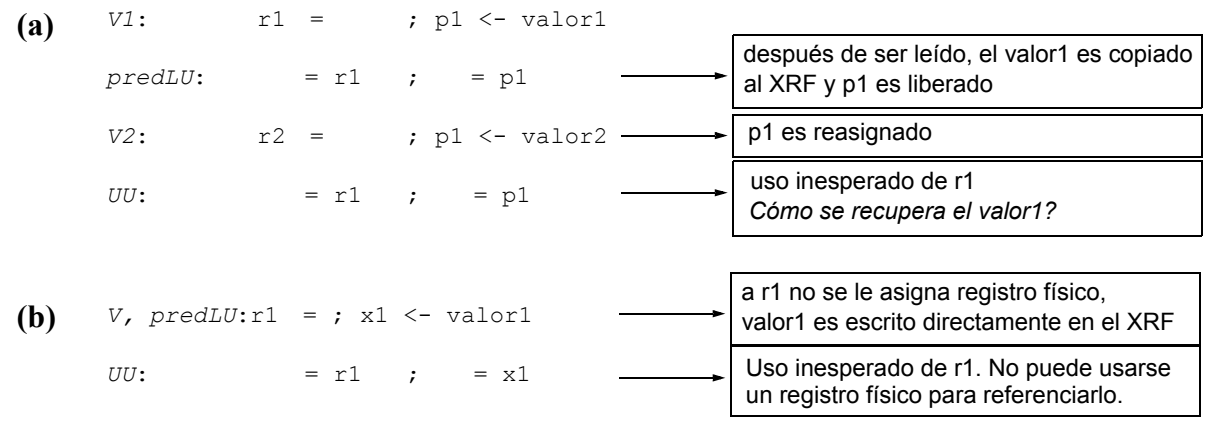


Figura 4.9 La instrucción *UU* quiere leer el registro lógico `r1`, pero su registro físico asociado (`p1`) no contiene el valor correcto, que fue guardado en el XRF.

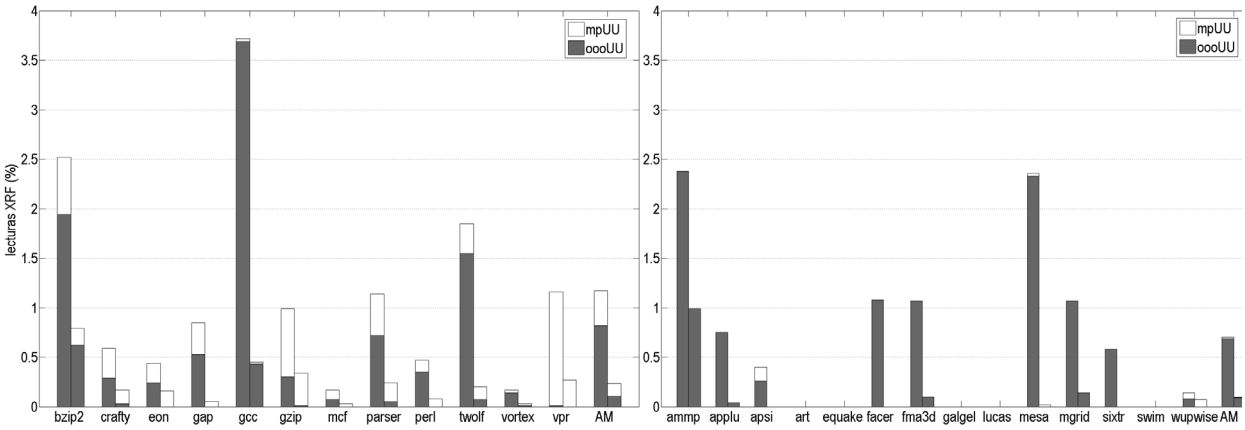
cipadamente y sus posibles usos inesperados. También asumimos un XRF con un número suficiente de puertos tanto de lectura como de escritura para suministrar valores a las instrucciones UU y para almacenar los valores identificados como último uso. Para modelar la ocupación del XRF, supondremos que un registro auxiliar es asignado cuando un registro es predicho último uso y que se libera cuando la siguiente instrucción que escribe en el mismo registro se jubila.

Es posible reducir el número de lecturas del XRF aprovechando el hecho de que pueden transcurrir muchos ciclos desde el instante en que un registro físico es liberado anticipadamente hasta que es sobreescrito. Una implementación mejorada podría recuperar del PRF un registro liberado anticipadamente que todavía no ha sido sobreescrito, ya sea porque no se han reasignado, o porque se ha reasignado a una instrucción que todavía no ha escrito su resultado.

4.7 Potencial del Renombre de Registros Especulativo

En el siguiente análisis, un procesador con renombre convencional se compara con un procesador mejorado con nuestra propuesta SR-LUP. Usamos un PRF con el mismo número de entradas para registros enteros y de coma flotante. Para obtener límites de prestaciones, supondremos un mecanismo de renombre controlado por un predictor de último uso ideal. Tal predictor es un oráculo con conocimiento perfecto de la última instrucción que referencia un registro en orden de programa. Este predictor oráculo no libera anticipadamente ningún registro por caminos incorrectos de ejecución, sin embargo pueden aparecer tanto oooUUs como mpUUs. A esta configuración la llamamos *SR-Oracle Based*. (SR-OB). Consideraremos un XRF con un número ilimitado de entradas y puertos.

Vamos a evaluar dos políticas de gestión de usos inesperados. En la primera, el XRF suministra todos los valores liberados anticipadamente que posteriormente son referenciados por instrucciones UU (los debidos a fallos de predicción, oooUU y mpUU). En la segunda, el XRF sólo suministra aquellos valores referenciados inesperadamente que no se encuentran en el PRF. Llamamos SR-OB+ a esta política mejorada de gestión de UUs, capaz de rescatar del PRF registros liberados prematuramente que todavía no han sido sobreescritos.



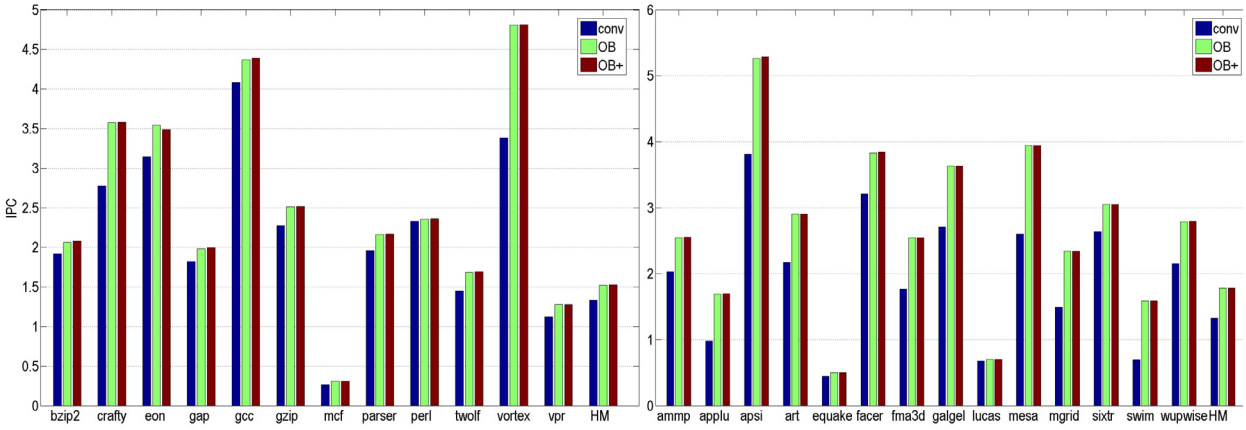
(a) SPEC CINT2000 **(b) SPEC CFP2000**

Figura 4.10 Porcentaje de operandos leídos del XRFint (a) y del XRFfp (b) en un procesador con un PRF de 80 registros (80int+80fp). Para cada programa, las dos barras representan el porcentaje de operandos leídos del XRF por el esquema SR-OB (izquierda) y por SR-OB+ (derecha). Cada barra muestra el desglose de lecturas debidas a oooUUs y a mpUUs. La última pareja de barras de cada gráfica es la media aritmética (AM).

4.7.1 Sensibilidad de las prestaciones a la política de gestión de UUs

La Figura 4.10 compara los porcentajes de lecturas del XRF realizadas por las políticas SR-OB y SR-OB+ gestionando un PRF de 80 entradas (80int+80fp) y un XRF sin límite de entradas (288int+288fp para la configuración simulada). Como ambos esquemas disponen de un predictor de último uso oráculo, las lecturas son debidas a usos inesperados por ejecución fuera de orden (oooUU) o por especulación de control (mpUU). Puede observarse que en media, los porcentajes de lecturas del XRF debidas a mpUUs o a oooUUs por separado no llegan al 1%. Solamente dos programas enteros (bzip2 y gcc) y dos de coma flotante (ammp y mesa) alcanzan el 2% para SR-OB. Para SR-OB+ el agregado de lecturas del XRF por mpUUs y oooUUs no llega al 1%. Un mpUU no afecta a las prestaciones ya que ralentiza la ejecución por camino incorrecto. Por tanto, en principio se espera que los programas más afectados por las lecturas del XRF sean aquéllos que sufren un mayor número de oooUUs. Si estos usos inesperados están en caminos críticos, su efecto negativo en las prestaciones se incrementará.

La Figura 4.11 muestra el número de instrucciones jubiladas por ciclo (IPC) para el renombre convencional, SR-OB y SR-OB+ ejecutando SPEC CPU 2000 con un PRF de 80 entradas (80int+80fp). La barra de más a la izquierda de cada grupo representa el renombre convencional, y las siguientes barras representan SR-OB y SR-OB+ con un XRF sin límite de entradas (288 para la configuración simulada) y una penalización por suministro de usos inesperados (UUs) de 2 ciclos. Puede observarse que, tanto para códigos enteros como de coma flotante, tanto SR-OB como SR-OB+ superan significativamente al renombre convencional. SR-OB+ obtiene unas aceleraciones medias del 14.3% y 34.1% para enteros y coma flotante, respectivamente. En concreto, las mejoras obtenidas por swim y applu son impresionantes (129% y 74%, respectivamente). Estos eran programas con un número muy reducido de registros libres y un elevado número de registros asignados en estado no-usado (ver Figura 2.10 del Capítulo 2).



(a) SPEC CINT2000 **(b) SPEC CFP2000**

Figura 4.11 IPC obtenidos con un PRF de 80 registros (80int+80fp). Para cada programa, cada grupo de tres barras representa el IPC del sistema con renombre convencional, SR-OB y SR-OB+, respectivamente. El último grupo de barras de cada gráfica es la media armónica (*harmonic mean*, HM).

La Figura 4.11 también muestra que SR-OB y SR-OB+ alcanzan IPCs prácticamente idénticos, siendo en media ligeramente superiores para SR-OB+. Para el resto de tamaños de PRF, los resultados son muy similares, con ventajas para SR-OB+ que no superan el 0.3%. Posiblemente, la lecturas adicionales del XRF en SR-OB debida a los oooUUs (con SR-OB cada oooUU provoca una lectura del XRF) no están en caminos críticos y por tanto no afectan al rendimiento. Es de esperar que con un predictor de último uso real se pueda sacar más provecho de los rescates de valores liberados anticipadamente pero que todavía no han sido sobreescritos en el PRF. A partir de ahora, sólo se presentan resultados para la política SR-OB+.

4.7.2 Desglose de la ganancia de prestaciones por OPRA y ERPR

La Figura 4.12 muestra el IPC para cuatro esquemas de renombre de registros: convencional, y oráculos que realizan omisión de asignación de registros físicos (OB+OPRA), liberación anticipada de registros físicos (OB+ERPR) y oráculo que controla tanto la asignación como la liberación de registros físicos (SR-OB+). El resultado más evidente es que la mayor parte de las ganancia media de SR-OB+ se debe a la liberación anticipada (OB+ERPR). Las mejoras por SR-OPRA son mayores para códigos enteros (3.1%) que para códigos de coma flotante (1.1%). En algunos programas enteros, la relativa alta frecuencia de registros destino sin consumidores permite ganancias significativas (7% para crafty, 6% para gcc y 10% para vortex). Entre los programas de coma flotante, art, fma3d y mesa aceleran la ejecución un 5%, mientras que el resto de programas no llegan al 2% de mejora. La regularidad de este tipo de códigos permite al compilador reducir el número de registros sin consumidores.

4.7.3 Sensibilidad de las prestaciones al tamaño del XRF

El límite superior en el número de entradas necesarias para cada XRF (int y fp) está determinado por el tamaño del ROB y por el número de registros lógicos, en concreto por su suma (288

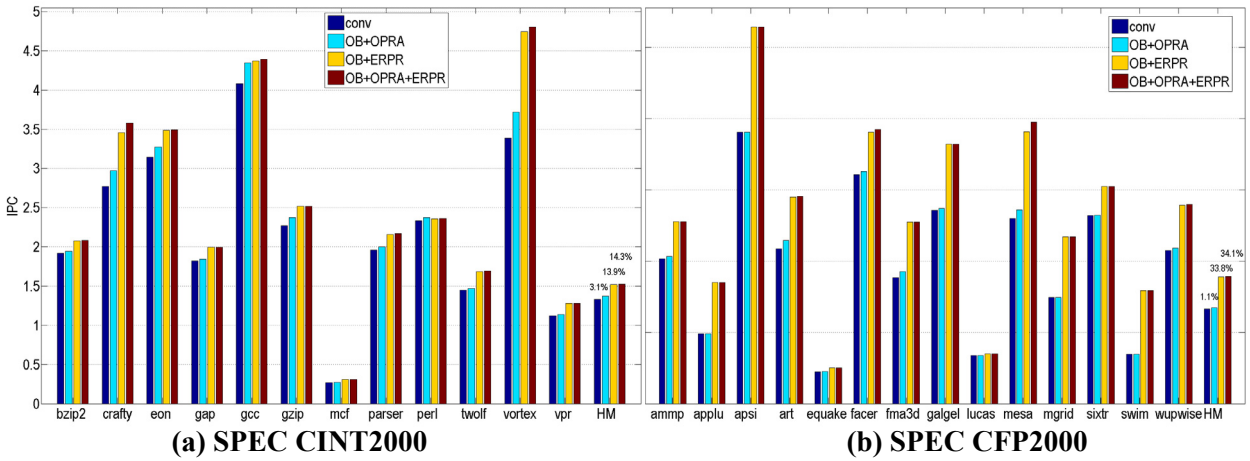


Figura 4.12 IPCs obtenidos con un PRF de 80 registros (80int+80fp). Para cada programa de SPEC CPU2000, se representa el IPC del sistema con renombre convencional (conv), oráculo que controla la omisión de asignación de registros físicos (OB+OPRA), oráculo que dirije la liberación anticipada de registros físicos (OB+ERPR), y oráculo que combina OPRA y ERPR (OB+OPRA-ERPR, es decir, SR-OB+). En todos las configuraciones SR, el XRF es de 288 entradas. El último grupo de barras de cada gráfica es la media armónica (*harmonic mean*, HM).

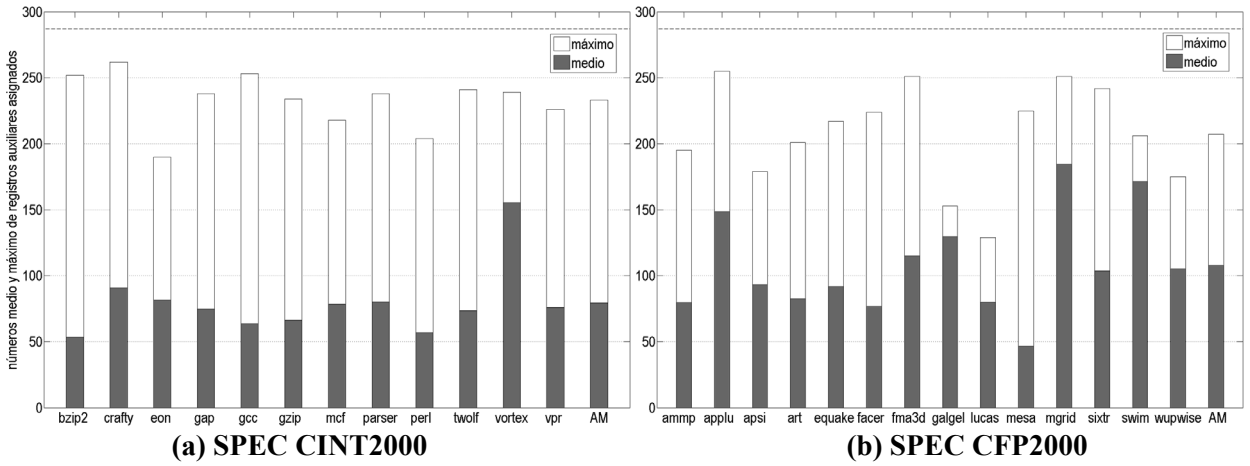
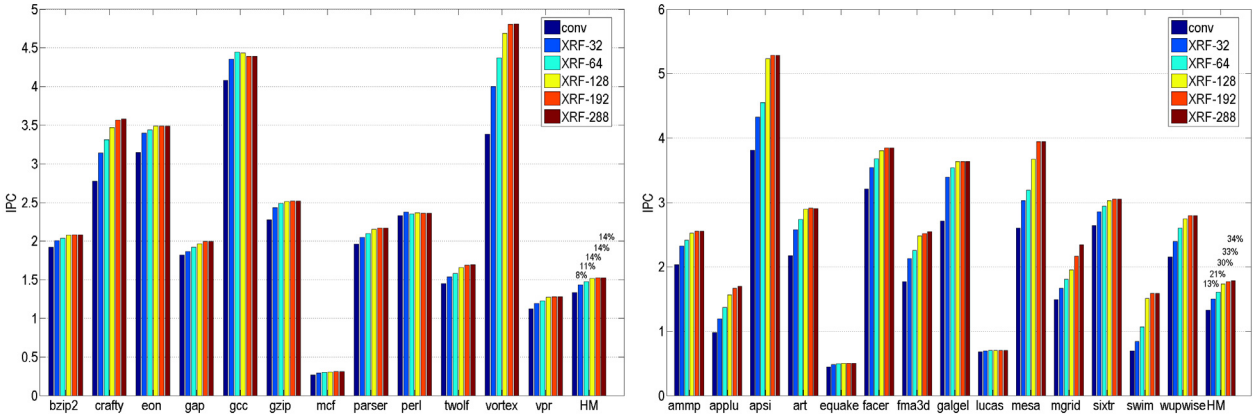


Figura 4.13 Números medio y máximo de entradas asignadas del XRFint (a) y XRFfp (b) para un PRF de 80 registros (80int+80fp) gestionado con SR-OB+. La última barra de cada gráfica es la media aritmética (*arithmetic mean*, AM).

en nuestra configuración). La Figura 4.13 muestra los números medio y máximo de entradas del XRF asignadas para SR-OB+. El número medio de entradas asignadas es de 79 y 108 para los programas enteros y de coma flotante, respectivamente. Ambos valores están muy distantes del número máximo de entradas. Estos datos sugieren que XRFs de tamaños más ajustados podrían mantener buenos niveles de prestaciones, ya que no se perderían muchas oportunidades de omitir la asignación o de liberar anticipadamente registros físicos por la falta de entradas en el XRF.



(a) SPEC CINT2000 **(b) SPEC CFP2000**

Figura 4.14 IPC para un PRF de 80 registros (80int+80fp). Para cada programa, cada grupo de barras representa el IPC del sistema con renombre convencional y SR-OB+ con XRFs de 32, 64, 128, 182 y 288 entradas. El último grupo de barras de cada gráfica es la media armónica (*harmonic mean*, HM), y los porcentajes indican el speedup respecto el renombre convencional..

Para poder analizar la sensibilidad al tamaño del XRF, la Figura 4.14 muestra el IPC obtenido con PRFs de 80 entradas (80int+80fp) gestionados de forma convencional y con el esquema SR-OB+. Para cada programa, la barra de la izquierda de cada grupo representa el renombre convencional y las siguientes barras representan SR-OB+ con XRFs de 32, 64, 128, 192 y 288 entradas. De nuevo puede observarse que cualquier configuración de SR-OB+ supera significativamente la gestión convencional tanto para códigos enteros como de coma flotante. La Figura 4.14 también muestra que, en media, los programas enteros no mejoran sus prestaciones al incrementar el tamaño del XRF por encima de 128 entradas. Solamente crafty, gap y vortex parecen aprovechar XRFs con mayor cantidad de registros. Sin embargo, ocurre lo contrario con los programas de coma flotante, donde todos los programas excepto dos (equake y lucas) son capaces de sacar partido de XRFs con hasta 192 entradas. Estos resultados sugieren un diseño asimétrico con XRFs de distinto tamaño para las rutas de datos enteros y de coma flotante.

4.7.4 Comparativa XRF vs. *Checkpointed Register File*

Varios de los trabajos revisados en el Capítulo 2 proponen que un banco de registros de respaldo (*Checkpointed Register File*, CRF) almacene los valores liberados anticipadamente [EBP+04][JOA+05][BSP+06a]. Esta estructura tiene las siguientes características:

- Está integrada en el PRF, por tanto el tamaño del CRF está limitado por el del PRF. Esto provoca que con bancos de registros pequeños se pierdan muchas oportunidades de realizar liberaciones anticipadas por falta de entradas en el CRF. Esta limitación vamos a denotarla como L1.

- Por construcción, el CRF sólo puede contener un valor asociado a cada registro físico, lo que también restringe las oportunidades de liberar anticipadamente un registro físico. Esta limitación vamos a referirla como L2.

Tabla 4.1 Porcentajes de mejora del IPC de SR-OB+ gestionando un CRF, un XRF del mismo tamaño del PRF, y un XRF ilimitado (288 entradas, no provoca anulaciones de predicciones de último uso).

		Número de registros físicos									
		36	**40**	**48**	**64**	**80**	**96**	**128**	**160**	**192**	**224**
SPEC CINT 2000	**CRF**	106,9	59,5	35,5	15,9	9,2	5,4	2,8	1,1	0,4	0,1
	#XRF = #PRF	178,4	98,1	50,8	21,3	11,9	6,9	3,2	1,2	0,4	0,1
	XRF-288	228,5	126,5	65,5	26,9	14,3	7,7	3,3	1,2	0,4	0,1
SPEC CFP 2000	**CRF**	114,0	57,3	33,0	24,6	18,1	12,6	9,5	6,0	1,8	0,6
	#XRF = #PRF	158,2	90,1	55,6	33,0	22,7	17,1	10,9	6,4	1,8	0,6
	XRF-288	238,5	139,4	84,5	52,2	34,1	21,1	11,4	6,4	1,8	0,6

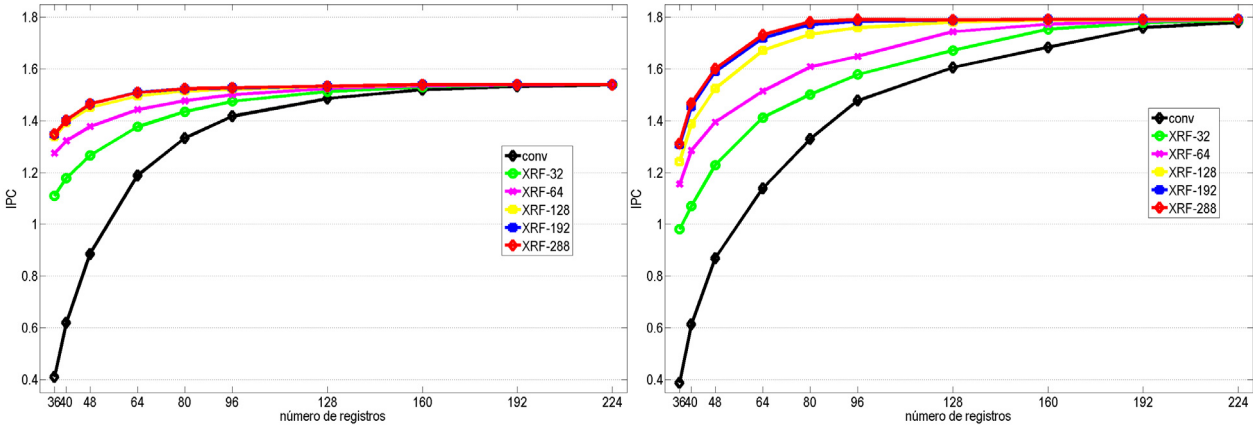
Para cuantificar la pérdida de prestaciones que provocan estas dos limitaciones del CRF, hemos obtenido el IPC obtenido por SR-OB+ gestionando un CRF, un XRF del mismo tamaño que el PRF, y un XRF con un número de entradas que no provoque anulaciones de predicciones de último uso (288 para nuestra configuración). La diferencia entre el esquema que usa un CRF y el que utiliza un XRF limitado por el número de entradas del PRF es la pérdida de rendimiento causada por la restricción de un valor liberado anticipadamente por registro físico (L2). La diferencia entre los esquemas que usan un XRF con un número limitado de entradas y el de XRF ilimitado es la pérdida de prestaciones causada por las anulaciones de predicciones de último uso por no disponer de registros auxiliares en el XRF (L1).

La Tabla 4.1 compara las mejoras en prestaciones respecto el mecanismo de renombre convencional de SR-OB+ gestionando un CRF, un XRF con el mismo número de entradas que el PRF y un XRF sin límite de entradas (288 entradas en la configuración simulada). Puede observarse que en el rango de tamaño del PRF donde las mejoras son más significativas (hasta 96 y 128 registros para CINT2000 y CFP2000), las mejoras se reducen más por causa de la limitación L2 que por la L1 en los códigos enteros, mientras que para códigos de coma flotante, la limitación que más afecta a las prestaciones es la L1.

Como SR-OB+ es capaz de liberar un registro físico antes que cualquiera de las propuestas presentadas en [EBP+04], [JOA+05] y [BSP+06a], SR-OB+CRF podría considerarse una cota superior del rendimiento de dichos esquemas.

4.7.5 Reducción del número de entradas del PRF

También es interesante analizar el efecto de modificar el tamaño del PRF. La Figura 4.15 ilustra como varía el IPC con el número de registros físicos para 5 configuraciones (convencional, SR-OB+ con XRFs de 32, 64, 128, 192 y 288 entradas). Podemos observar como SR-OB+ hace que las prestaciones del procesador sean menos dependientes del tamaño del PRF, permitiendo alcanzar el 98% del IPC máximo (el obtenido con el PRF de mayor tamaño -288 entradas-) con PRFs mucho más ajustados: 64 registros para los códigos enteros y 80 para los de coma flotan-



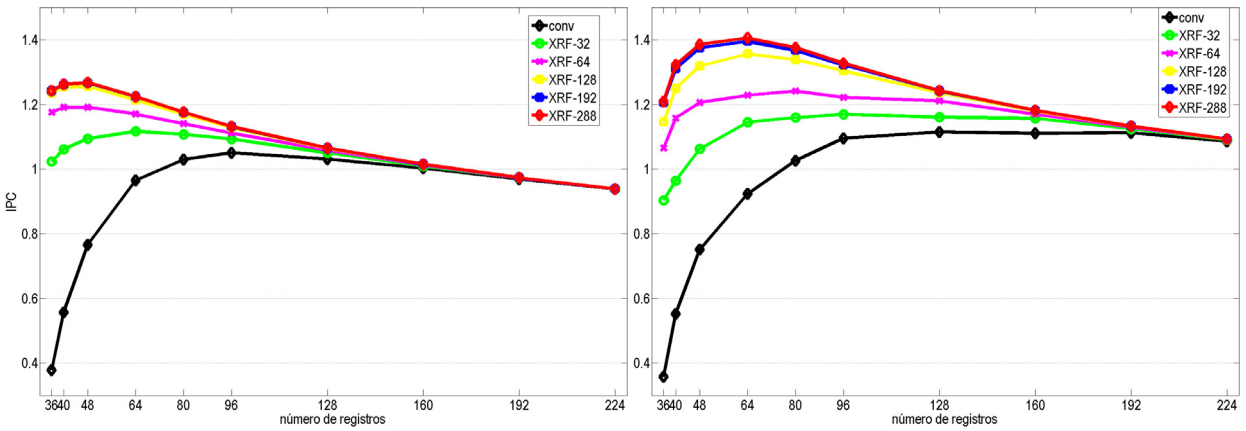
(a) SPEC CINT2000 **(b) SPEC CFP2000**

Figura 4.15 IPC medio vs. número de registros físicos para el mecanismo de renombre convencional y SR-OB+ con XRFs de 32, 64, 128, 182 y 288 entradas.

te. Por otra parte, la Figura 4.15 muestra que para códigos enteros y en todo el rango de entradas del PRF, SR-OB+ no mejora las prestaciones al aumentar el tamaño del XRF por encima de 128 entradas (de hecho, las curvas correspondientes a XRFs de 128, 192 y 288 entradas se superponen). Con esos tamaños de XRF, un PRF de 36 registros (el menor tamaño de PRF simulado) alcanza casi el 90% del IPC máximo. Para códigos de coma flotante, SR-OB+ es capaz de explotar XRFs de hasta 192 entradas con PRFs ajustados (menos de 96 registros). Combinando el XRF de 192 entradas con PRFs de 48 y 64 entradas se obtiene el 89% y el 96% del IPC máximo, respectivamente. Con PRFs menos ajustados (96 ó más entradas), el XRF de 128 entradas es suficiente para alcanzar más del 98% del IPC máximo.

Si el tiempo de ciclo del procesador está limitado por el tiempo de acceso al PRF, una reducción en el número de registros puede causar un incremento de la frecuencia del procesador y una mejora de rendimiento. La Figura 4.16 muestra los billones de instrucciones por segundo (BIPS) ejecutadas por procesadores con renombre convencional y SR-OB+ con XRFs de distintos tamaños. Para los programas enteros, el renombre convencional obtiene sus mejores prestaciones con un PRF de 96 registros. Pero cuando se aplica SR-OB+ a un XRF de 32 entradas, el tamaño óptimo del PRF se reduce a 64. Con XRFs de 64, 128, 192 y 288 entradas, los máximos IPSs se obtienen con PRFs de solamente 48 entradas. Para los programas de coma flotante, las mejores prestaciones del renombre convencional se alcanzan con un PRF de 128 registros. Cuando se utiliza SR-OB+, los tamaños óptimos del PRF se reducen, haciéndolo en mayor medida cuanto mayor es el tamaño del XRF. Así, con XRFs de 32, 64, y 128 entradas, él los IPS máximos se obtienen con PRFs de 96, 80 y 64 entradas, respectivamente. El tamaño óptimo del PRF no cambia con XRFs mayores de 128 entradas.

A partir de estos números, podemos concluir que SR-OB+ puede conseguir una gran reducción en el tamaño del PRF a la vez que un incremento de prestaciones. Por ejemplo, considerando tanto códigos enteros como de coma flotante, un procesador con un PRF de 96 entradas (96int+96fp) gestionado de forma convencional puede ser reemplazado por un PRF de 64 entradas (64int+64fp) gestionado por SR-OB+ con un XRF de 64 entradas (64int+64fp), pero obteniendo un incremento del 2% de IPC y de más del 11% de IPS.



(a) SPEC CINT2000 **(b) SPEC CFP2000**

Figura 4.16 Miles de millones de instrucciones por segundo ejecutadas (*Billions of Instructions Per Second*, BIPS) vs. número de registros físicos para el mecanismo de renombre convencional y SR-OB+ con XRFs de 32, 64, 128, 182 y 288 entradas.

4.8 Conclusiones

En este Capítulo hemos analizado el potencial de un nueva y agresiva estrategia de renombre de registros basada en la predicción de último uso de un registro llamada SR-LUP. Este esquema de renombre modifica las políticas convencionales de asignación y liberación de registros. A los registros destino identificados como último uso no se les asigna registro físico, mientras que los registros fuentes marcados como último uso son liberados anticipadamente aprovechando su lectura por parte de la última instrucción consumidora en orden de programa. Un banco de registros auxiliar (XRF) almacena los valores marcados por el predictor de último uso por si son referenciados inesperadamente por posteriores instrucciones y no es posible su suministro desde el PRF. La ubicación de este almacén auxiliar fuera de los caminos críticos supone una penalización en su acceso.

A diferencia de otras propuestas previas, SR-LUP puede liberar un registro físico incluso cuando la redefinición de su registro lógico no ha sido siguiera buscada, permitiendo así la reutilización de registros físicos en situaciones en que otros mecanismos de renombre deberían detener la etapa de renombre por carecer de registros disponibles. El predictor de uso también reemplaza el complejo hardware necesario para verificar las condiciones de liberación anticipada propuestas por otros autores, en especial, los contadores de usos pendientes.

Además de demostrar el potencial de la idea, en este capítulo hemos analizado también la viabilidad de la predicción de patrones de uso. Hemos mostrado que casi el 90% y el 95% de las instrucciones estáticas de SPEC CINT2000 y CFP2000 presentan en todos sus instancias el mismo patrón de uso. Este es el paso previo al diseño de un predictor de último uso que se aborda en el Capítulo 5.

El nuevo mecanismo de renombre puede mejorar notablemente las prestaciones alcanzadas por un PRF. Por ejemplo, un PRF de 80 entradas (80int+80fp) gestionado con un SR-LUP perfecto (predictor de último uso oráculo) obtiene unas aceleraciones medias del 14.3% y 34.1% para SPEC CINT2000 y CFP2000, respectivamente. En su mayor parte, estas ganancias provienen de la liberación anticipada. También hemos observado que no es necesario un XRF con el máximo número de entradas (288 en la configuración analizada) para alcanzar el rendimiento máximo de SR-LUP. Para SPEC CINT2000 y CFP2000, unos XRFs de 128 y 192 entradas son suficientes para explotar las políticas de omisión de asignación y liberación anticipada de registros físicos. En el Capítulo 5 se analiza el tamaño del XRF requerido por predictores reales. También se analiza el efecto en la limitación de puertos.

Otra opción de diseño que permite SR-LUP es reducir el número de entradas del PRF sin perder IPC. Además, en este caso el menor tiempo de acceso al PRF podría permitir un aumento de la frecuencia del procesador y por tanto un incremento de las instrucciones ejecutadas por segundo (IPS). Para el procesador evaluado, puede reducirse un 50% un 33% el tamaño del PRF a la vez que se obtienen mejoras de más del 11% de IPS.

4.9 Referencias

[AMV+06] J. Alastruey, T. Monreal, V. Viñals and M. Valero. “Speculative Early Register Release”. *Proc. of the 3rd conference on Computing frontiers* (CF 06), May 2006, pp 291-302.

[AMV+07] J. Alastruey, T. Monreal, V. Viñals and M. Valero. “Microarchitectural Support for Speculative Register Renaming”. *IEEE Int. Parallel & Distributed Processing Symp.* (IPDPS 07), Marzo 2007.

[BSP+06a] D. Balkan, J. Sharkey, D. Ponomarev and A. Aggarwal, “Address-Value Decoupling for Early Register Deallocation”, Proc. 35th Int’l Conf. on Parallel Processing (ICPP 06), Agosto 2006, pp. 337-346.

[BSP+06b] D. Balkan, J. Sharkey, D. Ponomarev and K. Ghose. SPARTAN: Speculative Avoidance of Register Allocation to Transient Values for Performance and Energy Efficiency. *Proc. 15th Int’l Conf. Parallel Architectures and Compilation Techniques* (PACT 06), Sept. 2006, pp. 265-274.

[BuSo02a] J.A. Butts and G. Sohi. "Dynamic dead-instruction detection and elimination". *Proc. 10th Int’l Conf. Architectural Support for Programing Languages and Operating Systems* (ASPLOS 02), Oct. 2002, pp. 199-210.

[BuSo02b] J.A. Butts and G. Sohi, Characterizing and Predicting Value Degree of Use”, *Proc. 35th Int’l Symp. Microarchitecture* (MICRO 02), Nov. 2002, pp. 15-26.

[BuSo04] J.A. Butts and G. Sohi, “Use-Based Register Caching with Decoupled Indexing”, Proc. 31st Int’l Symp. Computer Architecture (ISCA 04), June 2004.

[EBP+04] O. Ergin, D. Balkan, D. Ponomarev and K. Ghose, "Increasing Processor Performance Through Early Register Release". *In Proceedings of the IEEE international Conference on Computer Design* (ICCD 04), pp. 480-487, San Jose (USA), October 11-13, 2004.

[JOA+05] T.M. Jones, M.F.P. O’Boyle, J. Abella, A. González, and O. Ergin, “Compiler Directed Early Register Release”, *Proc. 14th Int’l Conf. on Parallel Architectures and Compilation Techniques* (PACT 05), pp. 110-122. Sept. 2005.

CAPÍTULO 5

Soporte Microarquitectónico para Renombre de Registros Especulativo

En este Capítulo se detalla y evalúa una microarquitectura que soporta estrategias de renombre especulativo, es decir, políticas de omisión de asignación y de liberación anticipada de registros físicos. El objetivo es implementar una microarquitectura que soporte políticas arbitrarias de renombre especulativo con unas estructuras de complejidad limitada. Primero se presenta el diseño general, basado en un predictor y un banco de registros auxiliar. Después se describen las principales modificaciones y extensiones de la microarquitectura, entre las que destaca la gestión de dependencias basada en registros virtuales, la localización de operandos y la recuperación por fallo de predicción. A continuación, se proponen dos diseños de predictores de último uso de complejidad y prestaciones incrementales y que pueden dirigir tanto las políticas de omisión de asignación como las de liberación anticipada de registros físicos. Por último, se evalúa el mecanismo de renombre especulativo asistido por los predictores de último uso planteados y se muestra que es posible acercarse al límite ideal de prestaciones con un predictor de último uso y un banco de registros auxiliar de complejidad limitada. La mejora en la gestión del PRF también puede aprovecharse para reducir su tamaño buscando frecuencias de procesador más elevadas.

5.1 Introducción

En este Capítulo proponemos una microarquitectura que soporta distintas estrategias de renombre especulativo (*Speculative Renaming*, SR), es decir, esquemas de renombre con políticas de omisión de asignación de registros físicos y con políticas de liberación anticipada. Ambas políticas son especulativas porque tienen naturaleza predictiva y requieren acciones de recuperación en caso de errores de predicción. Nuestra propuesta se basa en la gestión de un banco de registros auxiliar (XRF) ubicado fuera de los caminos críticos del procesador. Los resultados de las instrucciones a las que no se ha asignado registro físico son escritos directamente en el XRF, mientras que los registros físicos liberados anticipadamente se transfieren del PRF al XRF.

Este soporte de SR es ortogonal con las políticas específicas que identifican los valores que se envían al XRF. Creemos que esta es la primera propuesta de una microarquitectura que soporta cualquier política de omisión de asignación y de liberación anticipada de registros físicos, independientemente de que estén controladas por información suministrada por el compilador o por un predictor hardware (el caso que evaluamos en este Capítulo).

Las principales modificaciones y extensiones de la microarquitectura para soportar SR son:

- Banco de Registros Auxiliar (XRF): el XRF mantiene valores hasta que es seguro descartarlos, soportando por tanto excepciones precisas además de especulación de control. El XRF suministra a las instrucciones que leen (de forma inesperada) tanto el resultado de una instrucción sin registro físico asignado como un valor liberado anticipadamente y que ya ha sido sobreescrito en el PRF, pero sin reintegrarlos al PRF. El acceso al XRF supone una penalización de varios ciclos.

- Gestión de dependencias: bajo SR, los operandos pueden ser leídos del PRF o del XRF. Para resolver esta doble ubicación de operandos, la gestión de dependencias se desacopla de los registros físicos mediante el uso de registros virtuales [MGV+99].

- Despertar y lanzamiento especulativo: para beneficiar el caso frecuente, predeciremos para todas las instrucciones con registros físicos fuente que sus operandos todavía residen en el PRF. En paralelo con el acceso al PRF, se verifica que el registro físico leído no ha sido prematuramente liberado y después sobreescrito por una instrucción más joven que lo ha reusado. Los operandos fuente sin registro físico asignado se leen directamente del XRF.

- Recuperación de fallos de predicción: cuando una instrucción lee un registro físico y comprueba que ha sido sobreescrito por una instrucción más joven, aquella instrucción y todas sus dependientes que hayan sido lanzadas son anuladas y posteriormente relanzadas. Estas acciones son realizadas por un mecanismo de recuperación selectiva en cadena propuesto para gestionar fallos de predicción de latencia [TIV+03].

Para evaluar la microarquitectura SR escogemos un predictor de Último Uso (*Last-Use Predictor*, LUP) porque puede dirigir tanto las políticas de omisión de asignación como las de liberación anticipada de registros físicos. Son posibles otras opciones, pero como se ha mostrado en el Capítulo 4, el potencial de este predictor es muy atractivo cuando se consideran componentes ideales (oráculo LUP, XRF ilimitado).

La organización de este Capítulo es la siguiente. La Sección 5.2 presenta los componentes y el funcionamiento de esta microarquitectura, y describe la utilización de registros virtuales para la gestión de dependencias y localización de operandos. En la Sección 5.3 se diseñan dos predictores de último uso, la estructura que va a controlar las políticas de omisión de asignación y liberación anticipada de registros. La Sección 5.4 analiza la sensibilidad de las prestaciones al diseño del XRF, con el objetivo de reducir la complejidad de esta estructura. En la Sección 5.5 se muestran los resultados de la evaluación de la microarquitectura con los predictores de último uso (SR-LUP). Finalmente, se resumen las conclusiones del Capítulo en la Sección 5.6. Los contenidos de este Capítulo son la continuación del trabajo publicado en [AMV+06], y se corresponden con la aportación publicada en [AMV+07].

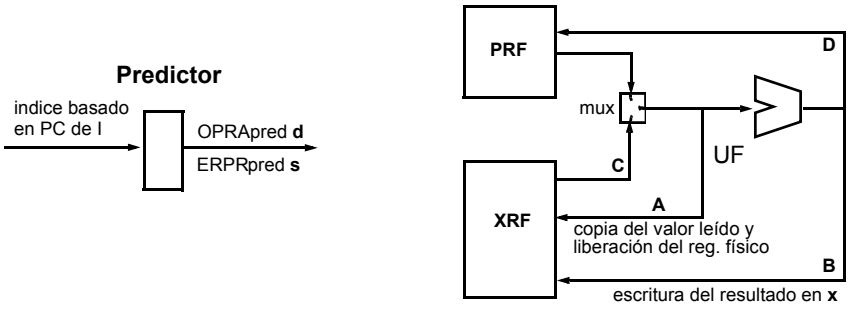


Figura 5.1 Componentes básicos y funcionamiento de un mecanismo de renombre especulativo de registros.

5.2 Microarquitectura para Renombre de Registros Especulativo

5.2.1 Idea del Renombre Especulativo

En una microarquitectura que soporta Renombre Especulativo (SR), uno o varios registros de una instrucción pueden ser etiquetados con una predicción dirigida a actuar sobre las políticas de asignación y liberación de registros. En concreto, un registro fuente puede recibir una predicción de liberación anticipada de registro físico (*Early Release of Physical Register prediction*, ERPRpred) y un registro destino puede recibir una predicción de omisión de asignación de registro físico (*Omission of Physical Register Allocation*, OPRApred).

La Figura 5.1 muestra los componentes y el funcionamiento de la microarquitectura SR. Un registro fuente marcado por un predictor será copiado en el XRF aprovechando su acceso en la etapa de lectura de operandos (circuito A en la Figura 5.1), y su identificador físico será enviado a la lista de registros libres. Un registro destino señalado por el predictor recibirá un identificador físico, por lo que no se escribirá en el PRF (circuito D en la Figura 5.1), sino que se escribe directamente en el PRF (circuito B en la Figura 5.1). Un fallo de predicción o una lectura retardada por la ejecución fuera de orden pueden requerir un valor sin registro físico o liberado prematuramente. Un uso inesperado (*Unexpected Use*, UU) de un valor sin registro asignado leerá el operando del XRF, mientras que un uso inesperado de un registro liberado anticipadamente leerá su operando del PRF (si el valor no ha sido sobreescrito) o del XRF si el en caso contrario (circuito C en la Figura 5.1). Los registros virtuales permiten gestionar tanto las dependencias como la localización de un operando en estos dos almacenes [MGV+99]. A continuación, detallamos nuestra propuesta de uso de registros virtuales para la gestión de dependencias y localización de valores en el PRF y XRF.

5.2.2 Gestión de dependencias y de localización de registros

Consideremos dos instrucciones A y B, A ejecutándose antes que B, cuyos registros lógicos destino son asignados consecutivamente al mismo registro p1 (ver Figura 5.2). Supongamos también que una tercera instrucción I, localizada en orden de programa entre A y B, libera especulativa y anticipadamente el registro p1. Después de que la instrucción A se ejecute, el re-

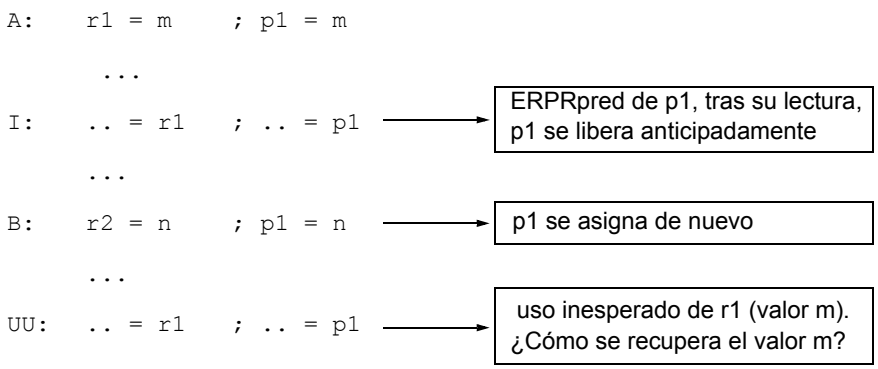


Figura 5.2 Ejemplo de instrucción (UU) que ha perdido el enlace con el valor de su registro fuente r1 porque su registro físico p1 ha sido liberado prematuramente y enviado al XRF.

sultado que genera (`m`) se encuentra en el registro `p1` del PRF, pero después de la etapa de lectura de operandos de I, dicho resultado se encuentra también en el XRF. El valor `m` permanecerá en el PRF hasta la que la instrucción B complete su ejecución y lo sobreescriba con el nuevo resultado `n`. Tras ese momento, el registro `pd` del PRF almacena el valor `n`, mientras que `m` ya sólo puede encontrarse en el XRF. Por tanto, con renombre especulativo (SR), el resultado de una instrucción puede estar sólo en el PRF, en ambos el PRF y el XRF, o solamente en el XRF. Además, puede observarse que la instrucción UU ha perdido su enlace con `m` a través del identificador físico `p1`.

Esta característica del SR impide que los identificadores de registros físicos puedan utilizarse para localizar los operandos en registros. Proponemos solucionar este problema mediante la utilización de los registros virtuales, utilizados anteriormente en el contexto de la asignación tardía de registros físicos [MGV+99].

En la etapa de renombre se asigna a cada registro lógico destino un par de identificadores: un registro físico (p) y un registro virtual (v). La Figura 5.3 presenta las estructuras que implementan el circuito SR. El nuevo doble mapeo es soportado por varias estructuras que gestionan el renombre especulativo y proporcionan la manera de localizar valores. La Tabla de Mapeo Global (Global Map Table, GMT) soporta el doble mapeo y la gestión de dependencias. Una Tabla de traducción Físico a Virtual (Physical to Virtual Table, P2V), una tabla de traducción Virtual a Auxiliar (Virtual to Auxiliar, V2X), un sencillo circuito lógico de comparación y un multiplexor que selecciona entre el PRF y el XRF (integrado en la red de cortocircuitos o anticipación de operandos) permiten la localización de operandos. La tabla P2V es accedida en paralelo con el PRF y devuelve el registro virtual asociado a un registro físico dado. La tabla P2V y los comparadores verifican que un registro físico leído se corresponde con la versión deseada. La tabla V2X funciona como tabla de mapeo auxiliar y almacena las correspondencias entre un identificador virtual y un registro auxiliar que contiene un valor sin registro físico asignado o liberado anticipadamente. Las listas de registros virtuales y auxiliares libres (*Virtual Free List*, VFL y *auXiliary Free List*, XFL) gestionan los identificadores virtuales y auxiliares libres.

Para cada instrucción, sus predicciones (OPRApred y/o ERPRpred) se almacenan en la RAM del *payload*. En la etapa de renombre, una instrucción lee del GMT la pareja <p, v>correspon-

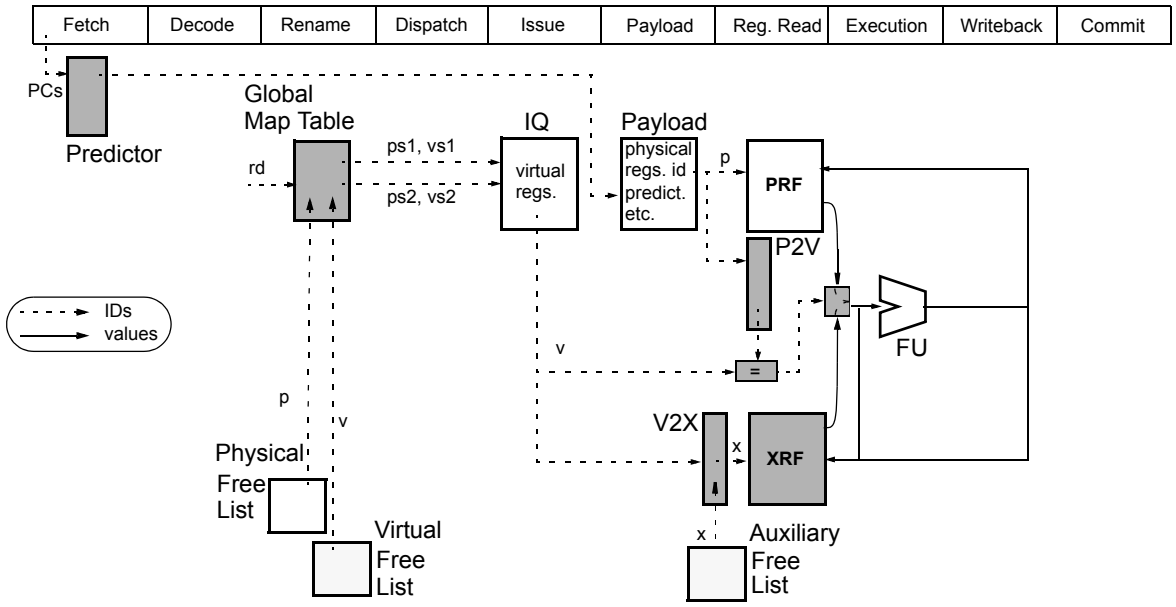


Figura 5.3 Microarquitectura con soporte de renombre especulativo. Las estructuras hardware modificadas o añadidas (sombreadas) se colocan debajo de su correspondiente etapa del segmentado.

diente a sus registros fuente. También en la etapa de renombre, la GMT es actualizada con una nueva pareja de identificadores <p, v> que son asignados a su registro lógico destino. Una OPRApred para el registro destino fuerza la asignación de sólo un identificador virtual, no se asigna registro físico alguno. Para cada registro fuente y destino con una predicción (ERPRpred y OPRApred, respectivamente), se pide un registro auxiliar x a la XFL. La tabla V2X almacena dichas asociaciones (traducción de identificador virtual a auxiliar). Después de eso, las instrucciones son enviadas a la ventana de lanzamiento (IQ) y toda la información relevante es almacenada en la memoria RAM de la etapa de payload (código de operación, identificadores de registro físicos, predicciones, etc.). Con SR, la gestión de dependencias en la IQ es gestionada mediante identificadores virtuales.

Al mismo tiempo que un valor es escrito en el PRF (registro p), su identificador virtual es almacenado en la tabla P2V (P2V[p]=v). Respecto a la liberación especulativa, una instrucción con algún registro marcado con una ERPRpred realiza tres acciones en la etapa de lectura de operandos:

- leer el valor del registro del PRF,
- escribir el valor en el XRF
- liberar el identificador físico enviándolo a la lista de registros físicos libres (FL)

En caso de un registro destino marcado con una OPRApred, la salida de la ALU o de cache (L1D) es escrita directamente en el XRF.

Con SR, excepto para los operandos que no tienen registro físico asignado, en tiempo de lanzamiento no se conoce la localización de los registros fuente (PRF o XRF), por tanto, cada instrucción seleccionada para ejecutarse usa la pareja <p, v> para buscar sus operandos. El identificador p se usa par acceder en paralelo al PRF y a la tabla P2V, buscando el identificador virtual del valor escrito en p (v_prf=P2V[p]). El identificador v es comparado con v_prf; si coinciden, el valor contenido en p es el adecuado, pero cuando la verificación de identificadores virtuales falla, el operando debe leerse del XRF y comenzar las acciones de recuperación. Leer el XRF requiere una lectura de la tabla V2X para buscar el identificador auxiliar que contiene el valor requerido. Respecto las acciones de recuperación, se utiliza el esquema de recuperación selectiva en cadena empleado en la predicción de latencia de instrucciones de latencia variable (por ejemplo, loads) [TIV+03]. La instrucción que no ha encontrado su operando en el PRF es anulada junto a sus instrucciones dependientes que han sido lanzadas. Después es relanzada y sus instrucciones dependientes son replanificadas considerando la nueva latencia de acceso del valor que consumen (latencia del XRF). Hay que tener presente que las acciones de recuperación solamente son necesarias cuando el valor liberado anticipadamente ha sido sobreescrito en el PRF por la instrucción que ha reutilizado su identificador p.

5.2.3 Liberación de Registros Virtuales, Físicos y Auxiliares

Con SR, hasta tres tipos de identificadores de registros pueden volver a sus correspondientes Listas de Registros Libres (Listas de Registros Libres Virtuales, Físicos y Auxiliares en la Figura 5.3). Tanto los identificadores virtuales como los auxiliares se gestionan de forma convencional, siendo liberados en la etapa de jubilación de las instrucciones que redefinen un registro lógico. Por otra parte, un identificador físico puede ser liberado de forma anticipada o convencional. Para distinguir entre ambos tipos de liberación, se necesitan dos vectores de bits (en concreto, dos vectores con NVR entradas de 1 bit, siendo NVR el número de registros virtuales). Estos vectores tienen varias funcionalidades: permitir la liberación convencional de registros físicos no etiquetados con predicciones ERPR, controlar que los registros liberados anticipadamente no son liberados dos veces, y cancelar las predicciones ERPR de registros que ya han recibido una predicción ERPR previa.

Llamamos lista de Predicciones de Renombre Especulativo (*Speculative Renaming Prediction* List, SRP) al vector de bits que indica los valores que han recibido una predicción OPRA o ERPR. La lista SRP es indexada por identificador virtual y el bit correspondiente se activa en la etapa de renombre de cada instrucción que recibe una predicción OPRA o ERPR. Por tanto, para cada instrucción, un bit activo en la entrada de la SRP correspondiente a su registro destino activa la liberación convencional del identificador auxiliar e inhibe la liberación convencional de un registro físico que no ha sido asignado. Un bit SRP activo en un registro fuente también activa la liberación convencional de su identificador auxiliar, y además cancela la liberación convencional de su identificador físico e inhibe nuevas predicciones ERPR que pueda recibir dicho registro. Un bit del SRP es desactivado cuando su registro virtual es asignado. Al igual que con la GMT, debe hacerse una copia del SRP en cada salto predicho.

La lista de Liberaciones Anticipadas Consumadas (*Committed Early Release List*, CER) es un vector de bits usado para recuperar de forma correcta la lista SRP. La CER indica si cada una de las liberaciones anticipadas planificadas durante un camino incorrecto de ejecución ha finaliza-

do o no (el valor está en el XRF o todavía se encuentra en el PRF). La lista CER es también indexada por identificador virtual. Una entrada se activa cuando el valor asociado a su identificador virtual es escrito en el XRF. Solamente aquellas entradas del SRP que no están activas en la lista CER se recuperan tras una predicción de salto incorrecta. La recuperación de un bit de la lista SRP implica la anulación de las acciones de liberación anticipada planificadas para el registro virtual correspondiente, es decir, la liberación del registro auxiliar asignado y la planificación convencional de su registro físico.

5.3 Predictor de Último Uso

Vamos a evaluar la microarquitectura con soporte de renombre especulativo con un predictor de último uso (*Last-Use*, LU) que controlará tanto la política de omisión de asignación como la de liberación anticipada de registros físicos. Vamos a proponer dos diseños de predictor LU. Ambos se organizan de manera similar a una cache asociativa, se indexan por el contador de programa (PC) y utilizan etiquetas (*tags*) parciales. Para cada instrucción que acierta en el predictor (la etiqueta de la entrada indexada coincide con la suya), el predictor devuelve un patrón de uso, un vector de bits que prevé si la siguiente referencia a cada uno de sus registros será una escritura. A partir de ahora, llamaremos predicción de último uso (*Last-Use Prediction*, LU-pred) a la predicción que indica a un registro que no va a volver a ser usado.

SR-LUP aprovecha una predicción LU de un registro fuente (ERPRpred) para liberarlo anticipadamente después de haberlo leído. Una predicción LU de un registro destino (OPRApred) se procesa antes, en la etapa de renombre, omitiendo la asignación de registro físico. El patrón de uso de una instrucción recorre el segmentado, almacenándose en la RAM del Payload cuando la instrucción es enviada a la IQ (*dispatch*).

Un predictor de último uso puede cometer dos tipos de fallos de predicción. Un fallo al predecir “no es último uso” equivale a una pérdida de oportunidad en la liberación anticipada o en la omisión de asignación de un registro físico (ERPR o OPRA, respectivamente). Un fallo al predecir “es último uso” es más costoso si el valor excluido del banco de registros tiene que ser recuperado de un banco de registros auxiliar (*Auxiliar Register File*, XRF) con latencia de acceso superior a la del banco de registros principal. La sección 5.3.3 explica con más detalles esta clasificación y cuantifica los fallos de cada tipo en los predictores diseñados.

Proponemos dos predictores de complejidad y prestaciones incrementales. El primero es un predictor de último uso sticky (*Sticky Last-Use Predictor*, SLUP). El segundo predictor explota el concepto de grado de uso presentado por Butts y Sohi [BuSo02b], y lo denominamos predictor de último uso basado en Grado de Uso (*Degree of Use Last-Use Predictor*, DULUP). Ambos predictores LU se pueden beneficiar de mejoras, como por ejemplo, el reemplazo de etiquetas (*tags*) por firmas (*signatures*) dependientes de la historia de saltos, lo que permitiría almacenar varios patrones de uso para una misma instrucción estática dependiendo del camino de control previo; o como por ejemplo, el uso de bits de confianza. Sin embargo, con el objetivo de limitar la complejidad de esta estructura, usaremos los dos diseños simplificados que se describen en las siguientes secciones.

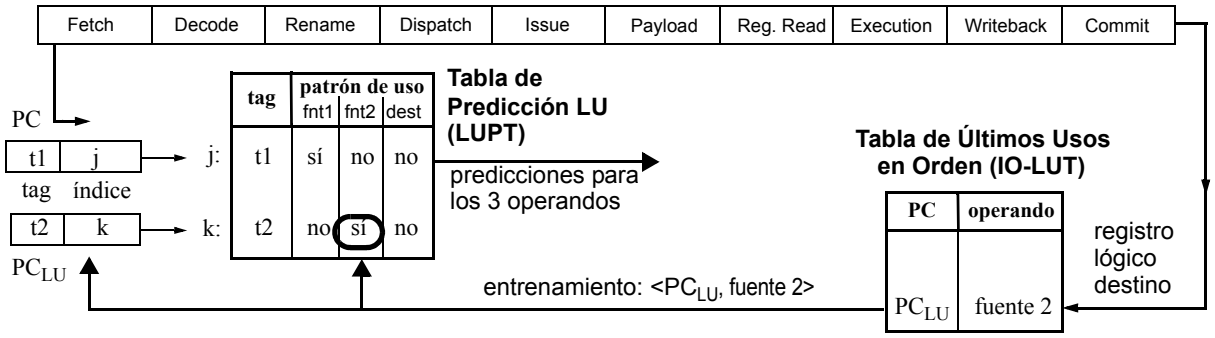


Figura 5.4 Implementación de Predictor de Último Uso Sticky (SLUP): Tabla de Predicción LU y Tabla de Últimos Usos en Orden IO-LUT.

5.3.1 SLUP: Sticky Last Use Predictor

El predictor SLUP está formado por dos estructuras: la Tabla de Predicción LU (Last-Use Prediction Table, LUPT) que almacena las predicciones de último uso y la Tabla de Últimos Usos en Orden (In-Order Last-Use Table, IO-LUT) que se encarga de entrenar la LUPT. La Figura 5.4 detalla la organización de ambas estructuras.

La LUPT puede organizarse como una cache indexada por el contador de programa (PC). Cada entrada contiene tres bits (fnt1, fnt2 y dest, uno por operando) que codifican el patrón de uso de una instrucción que ha tenido al menos un operando último uso en alguna de sus ejecuciones. Además, contiene la etiqueta (*tag*) de la dirección de la instrucción objeto de predicción. La etiqueta puede ser total, parcial o incluso puede omitirse. Cuando la LUPT no tiene un patrón de uso para una instrucción, se asume como predicción por defecto "no es último uso" para todos los operandos.

Por otro lado, la IO-LUT guarda el último uso consolidado de cada registro lógico. Cada entrada almacena el contador de programa (PC_{LU}) y el tipo de operando (destino, fuente 1 ó 2) de la última instrucción jubilada que ha referenciado un registro lógico. Cuando una instrucción con registro destino se jubila (*commit*), los contenidos de la entrada correspondiente a dicho registro se envían a la LUPT (ver Figura 5.4). De este modo, en cada entrenamiento, la LUPT registra el último uso de un operando de una instrucción, por lo que aunque posteriores ejecuciones de dicha instrucción no tengan el mismo patrón, con este esquema no es posible revocar una predicción (predictor *sticky*). Sólo se realiza desentrenamiento cuando una entrada es reemplazada. El Anexo B amplía la información referente a los procesos de determinación de patrones de uso y de entrenamiento de la LUPT.

Así pues, el comportamiento *sticky* mantiene en la LUPT patrones de uso obsoletos que pueden causar un aumento de los costosos fallos prediciendo "es último uso". En [ChEm98] y [YER+99] se sugiere el borrar periódicamente una tabla de predicción de dependencias store-load para eliminar la información obsoleta, aún a expensas de eliminar también la reciente. Nuestros experimentos muestran que las mejores prestaciones se obtienen borrando la LUPT cada 128 Kciclos. Otra opción sería utilizar técnicas de decaimiento (*decay*) en el diseño de la

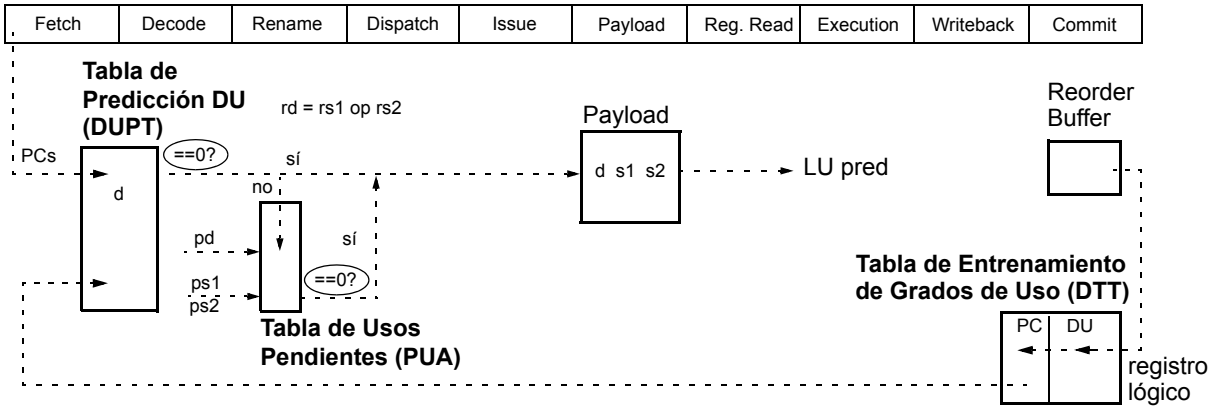


Figura 5.5 Implementación del predictor LU basado en predicción de grado de uso (*Degree of Use Last-Use Predictor*, DULUP): Tabla de Predicción de Grado de Uso (*Degree-of-Use Prediction Table*, DUPT), Tabla de Usos Pendientes (*Pending Uses Array*) y Tabla de Entrenamiento de Grados de Uso (*Degree of Use Training Table*, DTT).

tabla de predicción. El decaimiento podría implementar de forma natural este borrado periódico. En el Capítulo 6 se proponen dos diseños que aplican técnicas de decaimiento a un SLUP.

La precisión del SLUP puede mejorarse mediante un sencillo filtrado de sus predicciones. Este filtrado consiste en descartar el patrón de uso predicho para una instrucción cuando no se corresponde con los operandos de la misma. Por ejemplo, si hay predicción de último uso de un registro destino para una instrucción que no lo tiene, puede considerarse que se ha leído una entrada del SLUP correspondiente a otra instrucción (a causa del *aliasing* por el uso de etiquetas parciales) y obviarse la predicción completa. Hemos observado que este filtrado mejora las prestaciones de SR-SLUP.

5.3.2 DULUP: Degree of Use Last-Use Predictor

Un predictor de Grado de Uso (*Degree of Use*, DU) predice los usos que va a tener un registro físico [BuSo02b]. A partir de este componente puede derivarse un predictor de último uso que llamamos Predictor LU basado en Grados de Uso (*Degree of Use Last-Use Predictor*, DULUP).

El predictor DULUP está formado por tres estructuras: la Tabla de Predicción de Grado de Uso (*Degree of Use Prediction Table*, DUPT) que almacena las predicciones de grado de uso, la Tabla de Entrenamiento de Grado de Uso (*Degree of Use Training Table*, DTT) que se encarga de actualizar las predicciones almacenadas en la DUPT, y la Tabla de Usos Pendientes (*Pending Uses Array*, PUA), almacena los usos pendientes de los registros físicos. La Figura 5.5 detalla la organización de estos componentes. Las dos primeras estructuras son las propias del predictor de Grado de Uso, mientras que la tercera adapta el predictor de grado de uso para funcionar como predictor de Último Uso.

Al igual que la LUPT, la DUPT puede organizarse como una cache indexada por el contador de programa (PC). Para una instrucción con registro destino, una entrada de la DUPT contiene el número esperado de lecturas de dicho registro destino. También contiene la etiqueta (*tag*) de la

dirección de la instrucción objeto de predicción. Cada vez que se jubila una instancia de la misma instrucción, se entrena la DUPT, sobreescribiendo la predicción anterior. Cuando la DUPT no tiene el grado de uso para una instrucción, se asume como predicción por defecto un número ilimitado de usos para su registro destino.

La consulta de la DUPT en la etapa de búsqueda de instrucciones (*fetch*) permite conocer el número de usos esperado para los registros destino de las instrucciones buscadas. Estas predicciones se almacenan posteriormente en un la Tabla de Usos Pendientes (PUA), un vector de contadores indexado por registro físico (pd en la Figura 5.5). A partir de este momento, cuando una instrucción se renombra, se decrementan los contadores de la PUA correspondientes a sus registros físicos fuente (ps1 y ps2 en la Figura 5.5).

Cuando tras renombrar un registro fuente, el contador asociado a su registro físico está a cero, esto implica que se espera que sea el último consumidor, por tanto recibirá una predicción LU (una ERPRpred, para ser más precisos), y la instrucción correspondiente procederá a realizar la liberación de dicho registro. Notar que el decremento de los contadores en la PUA se realiza en la etapa de renombre, por lo que una gestión precisa de éstos requiere su restauración tras un fallo de predicción de salto.

Al igual que con el SLUP, una predicción de grado de uso cero para un registro destino indica a la etapa de renombre de registros que evite la asignación de un registro físico y a la etapa de escritura (*writeback*) que escriba solamente en el XRF.

El entrenamiento del la DUPT se lleva a cabo por parte de la DTT, que contabiliza el número de lecturas de cada registro lógico.

5.3.3 Evaluación de los predictores SLUP y DULUP

A continuación, vamos a evaluar la precisión de los predictores SLUP y DULUP. Más en detalle, vamos a analizar la variación de los fallos de predicción de último uso al cambiar parámetros de diseño de los predictores como su tamaño y asociatividad.

Los fallos de predicción LU se dividen en dos tipos:

- Fallos cuando se predice "es último uso de un registro" (falsos positivos): un registro es marcado como último uso cuando en realidad no lo es. Según sea una OPRApred o una ERPRpred, este error conduce a la omisión de asignación de registro físico a un valor que tiene instrucciones consumidoras o a la liberación prematura de un registro físico. En ambos casos, las prestaciones pueden degradarse debido a los costosos accesos al XRF por parte de los usos inesperados que pueden aparecer[1].

- Fallos cuando se predice "no es último uso de un registro" (falsos negativos): ocurren cuando un registro que realmente es último uso no es identificado como tal. Según sea una OPRApred o una ERPRpred, este error causa la asignación de registro físico a un valor que

1. Decimos "*pueden* aparecer" porque la ejecución fuera de orden o la red de cortocircuitos pueden evitar que los usos inesperados tengan que acceder al XRF.

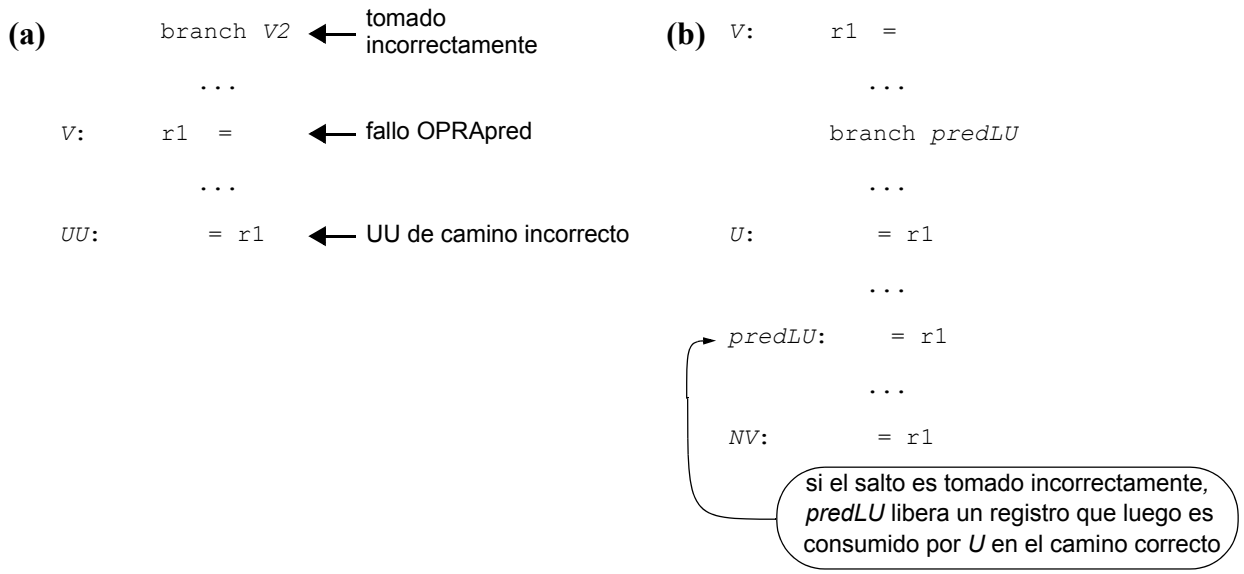


Figura 5.6 Ejemplos de fallos de predicción LU por camino incorrecto que afecta a un valor generado por camino incorrecto (a) y a un valor generado por camino correcto (b).

no tiene instrucciones consumidoras o perder la oportunidad de que la última consumidora de un registro físico lo libere anticipadamente. En ambos casos, no se degradan las prestaciones respecto al renombre convencional, sino que se pierde una posibilidad de mejorarlas.

Un fallo prediciendo "es último uso de un registro" puede surgir por camino correcto o incorrecto. En principio, un fallo de una OPRApred o de una ERPRpred que afecta a un valor generado por un camino incorrecto no degradaría las prestaciones. Más bien al contrario, pueden reducir la profundidad de especulación por caminos incorrectos de ejecución y reducir el tiempo de recuperación por el error de predicción de salto. La Figura 5.5a muestra un ejemplo de un fallo de OPRApred a un valor generado por camino incorrecto. Sin embargo, la liberación prematura de un valor generado por el camino correcto sí que puede degradar las prestaciones[2]. Una ERPRpred puede liberar un registro que, tras la recuperación del fallo de predicción de salto, sea referenciado por una instrucción de camino correcto, pudiendo generar un acceso al XRF por liberación prematura (ver Figura 5.5b). Cuanto mayor sea la precisión del predictor de salto, menor será la cantidad de fallos de predicción LU debidos a esta causa. La dificultad de evitar este tipo de fallos estriba en que atendiendo al flujo de instrucciones de un camino incorrecto, la liberación anticipada puede ser aparentemente correcta (ver Figura 5.5b).

La Figura 5.7 muestra el desglose de aciertos y fallos de predicción para distintos tamaños de los predictores SLUP y DULUP. Para cada tamaño de predictor, de izquierda a derecha, las tres primeras barras representan las tasas de aciertos y fallos para predictores SLUP con asociatividades 1, 2 y 4, y las tres siguientes muestran las tasas de aciertos y fallos para predictores DU-

2. Notar que no puede haber fallos de OPRApred por camino incorrecto que afecten a valores de camino correcto.

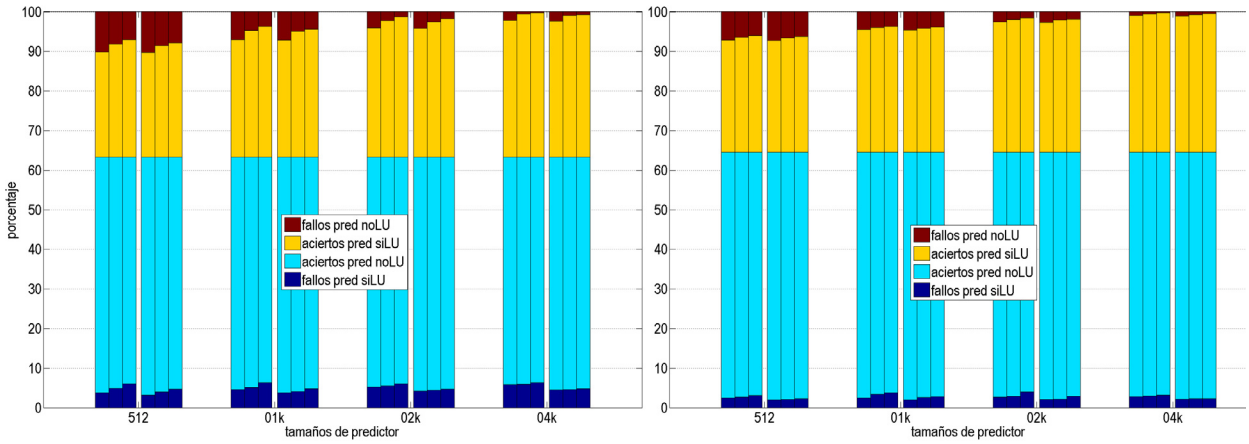


Figura 5.7 Tasas de aciertos y fallos de predicción de Último Uso para varios tamaños y asociatividades de SLUP y DULUP. En todos los casos se ha usado una combinación de un PRF con 80 entradas (80int+80fp) y un XRF de 288 entradas (ilimitado).

LUP con la misma secuencia de asociatividades. Los segmentos de las barras que representan los fallos están en los extremos superior (falsos negativos) e inferior (falsos positivos). En el caso del predictor SLUP, sus predicciones son descartadas en caso de discordancia entre operandos y la LUPT es borrada cada 128K ciclos.

Los falsos negativos (pérdidas de oportunidad) disminuyen conforme aumenta el número de entradas de ambos predictores. Con el incremento de capacidad, disminuyen los conflictos y por tanto hay más predicciones LU disponibles. Por otra parte, los falsos positivos (aquellos que pueden generar reejecuciones y lecturas de XRF) aumentan con el tamaño del predictor: cuantas más entradas, más predicciones "sí es último uso" se efectúan y más fallos de este tipo se producen. Estos efectos son más pronunciados para SLUP, a pesar de las dos mejoras aplicadas para reducir falsos positivos (el borrado de la LUPT los reduce en más del 20% para los predictores de 2k y 4k). Como era de esperar, el aumento de asociatividad se comporta como el aumento de tamaño: disminuyen los falsos negativos y aumentan los falsos positivos. Este efecto es más acusado para códigos enteros.

Si comparamos ambos predictores, podemos observar como SLUP sufre más falsos positivos que DULUP para todas las configuraciones del predictor. Esto es debido a que SLUP mantiene una predicción "sí es último uso" para una instrucción que ha tenido un registro último uso en alguna de sus ejecuciones anteriores, incluso cuando en sus últimas ejecuciones su patrón de uso no contenga ningún registro que sea último uso[3]. En cambio, DULUP es capaz de capturar cambios en el patrón de uso de una instrucción gracias a su distinto mecanismo de actualización y predicción. En lo que a falsos negativos se refiere, las pérdidas de oportunidad son más frecuentes para DULUP, especialmente para los predictores más grandes. Esto se debe a que hay más instrucciones susceptibles de ser almacenadas en el DULUP (hay más instrucciones con registro destino que con algún registro último uso).

3. En el caso que hemos evaluado, las predicciones son mantenidas durante períodos de 128K ciclos.

En las siguientes secciones se usan predictores LU de mapeo directo, con 2048 entradas que contienen patrones de uso y etiquetas de 5 bits. En el caso del DULUP, la DUPT y la PUA contienen contadores de 3 bits y la última estructura no se restaura cuando se recupera un salto mal predicho. Hemos seleccionado estas configuraciones sencillas (presupuesto hardware de alrededor de 2 Kbytes para los predictores y de 0.1 Kbytes para sus mecanismos de entrenamiento) como unos diseños con un buen compromiso entre complejidad y prestaciones.

5.4 Sensibilidad de las prestaciones de SR-LUP al diseño del XRF

El objetivo de esta sección es encontrar una configuración del XRF que se acerque al máximo de prestaciones alcanzables por SR-LUP, pero con un coste hardware acotado. Hemos considerado un XRF con un tiempo de acceso de un ciclo y dos ciclos adicionales para encaminar los datos hacia y desde dicha estructura. Por tanto, hemos descartado todos las configuraciones del XRF (entradas, puertos) que exceden el tiempo de acceso del PRF considerado en cada caso. A continuación, se detallan los efectos de limitar el número de entradas y puertos del XRF y se evalúan varias configuraciones del XRF que permitan encontrar diseños que exploten el potencial de SR-LUP.

5.4.1 Efectos de limitar el número de entradas del XRF

En la etapa de renombre de registros se tienen que suministrar tantos registros auxiliares como número de registros último uso identificados por el predictor LU. La falta de registros auxiliares supone la cancelación de predicciones LU, con la consiguiente pérdida de oportunidad de omitir la asignación de un registro físico o de planificar una liberación anticipada. Para explotar al máximo el potencial de SR-LUP, el número de registros auxiliares debe ser el suficiente para que la cantidad de cancelaciones de predicciones LU sea despreciable. El tamaño máximo teórico del XRF para nuestra configuración es de 288 entradas, número bastante cercano al máximo observado (274 para XRFint y 270 para XRFfp).

5.4.2 Efectos de limitar el número de puertos del XRF

Los puertos de escritura del XRF permiten la copia de los registros identificados como LU por el predictor. Por tanto, su número está determinado por el número de registros LU fuente (instrucciones en la etapa de lectura de operandos) y destino (instrucciones en la etapa de escritura -writeback-). La falta de puertos de escritura puede causar contención a la hora de seleccionar instrucciones con registros predichos como LU, y por tanto, el retraso en la ejecución de instrucciones sin registro físico asignado o con una liberación anticipada planificada. Para evitar esta contención, el número de puertos de escritura no debe limitar el lanzamiento de instrucciones con predicciones OPRA y ERPR (un máximo de 30 puertos para el XRFint, 16+14 para registros fuente+destino).

Los puertos de lectura del XRF suministran los valores que no están en el PRF a las unidades funcionales que ejecutan instrucciones con usos inesperados. El número de puertos necesarios viene fijado por la cantidad de lecturas del XRF debida tanto a valores liberados anticipadamente que ya no se encuentran en el PRF como a aquellos valores sin registro físico y que sus

Tabla 5.1 Configuraciones ajustadas de XRFint y XRFfp que mejor explotan SR-LUP. Los PRFs con más de 160 entradas no se muestran porque no se benefician de SR-LUP.

	Número de entradas del PRF							
	36	**40**	**48**	**64**	**80**	**96**	**128**	**160**
Tamaño XRF	160	160	192	224	224	224	256	256
Puertos lectura XRF	2	2	2	3	4	5	6	8
Puertos escritura XRF (fuente+destino)	4f+1d	5f+1d	5f+1d	5f+2d	7f+2d	8f+2d	10f+2d	11f+2d

consumidores no han podido leer de la red de cortocircuitos (*bypass network*). La falta de puertos de lectura puede causar contención a la hora de lanzar instrucciones UU, y por tanto retrasar el avance de este tipo de instrucciones que ya han sido penalizadas con dos ciclos por su acceso al XRF. Para evitar esta contención, el número de puertos de lectura debería permitir a todos los posibles UUs a leer el XRF en un mismo ciclo (un máximo de 16 puertos para el XRFint).

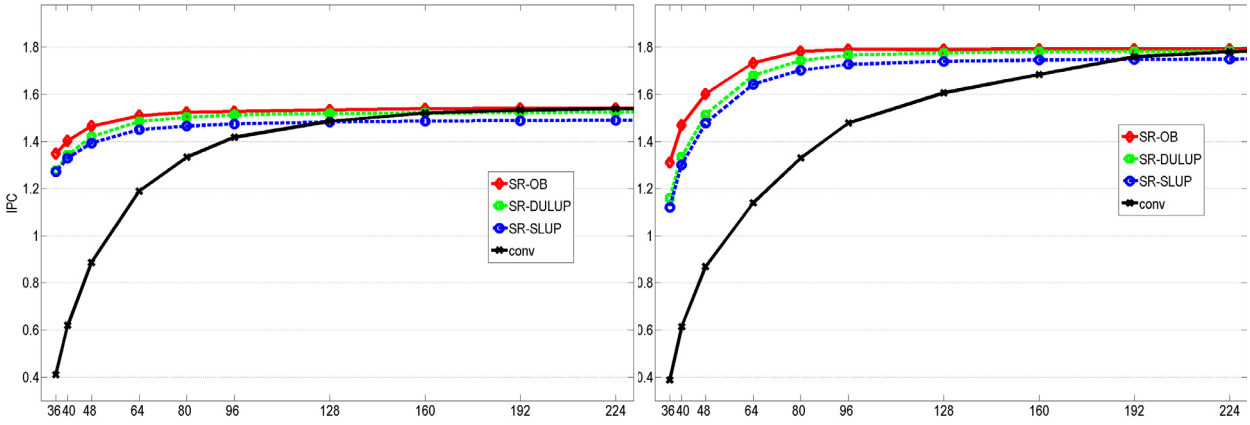
Según nuestros experimentos, el uso real de los puertos de lectura y escritura sugiere que su número puede ser significativamente reducido con tasas de contención desdeñables.

5.4.3 Selección del número de entradas y puertos del XRF

En resumen, se necesita un XRF con suficientes entradas para no cancelar predicciones LU y con suficientes puertos para escribir registros LU y leer registros de UUs. Para explorar una cantidad razonable de puntos de diseño, hemos seleccionado varios tamaños de XRF y varias combinaciones de puertos de lectura y escritura con tasas de contención similares. Después hemos cruzado tamaños y puertos para obtener diferentes configuraciones de XRF. Para cada tamaño de PRF, hemos descartado los XRFs cuyo tiempo de acceso es mayor que el del PRF considerado. Como resultado de este procedimiento, se obtienen diferentes configuraciones de XRF para cada PRF, desde XRFs con pocas entradas y muchos puertos hasta XRFs con muchas entradas y pocas puertos. Para reducir los puntos de experimentación, para cada tamaño de XRF se ha escogido el XRF con más puertos. Del mismo modo, para cada número de puertos de XRF, se ha escogido el mayor número de entradas. Es decir, hemos seleccionado los XRFs de mayores prestaciones. Hemos considerado diseños XRFint y XRFfp simétricos. Finalmente, hemos evaluado las configuraciones resultantes del proceso de selección para buscar la que mejor explota SR-LUP. El Anexo C amplía los detalles de este proceso de selección.

Para cada tamaño de PRF, la Tabla 5.1 muestra la configuración de XRF que obtiene las mejores prestaciones (IPC) de entre las muestras seleccionadas. La Tabla 5.1 sólo muestra PRFs con hasta 160 entradas porque los mayores PRFs no se benefician de SR-LUP. El número de puertos de escritura se ha descompuesto en puertos para registros fuente y destino (f,d).

Observamos que, en proporción, el número de puertos del XRF ha sido más reducido que el número de entradas. Por ejemplo, el XRF con 160 entradas y 7 puertos sufre una reducción del 85% del número de puertos y del menos del 45% del número de entradas. A partir de estos resultados, y con el objetivo de las prestaciones presente, es preferible retrasar el lanzamiento de instrucciones UU o con predicciones LU antes que cancelar predicciones de último uso. Aun-



(a) SPEC CINT2000 **(b) SPEC CFP2000**

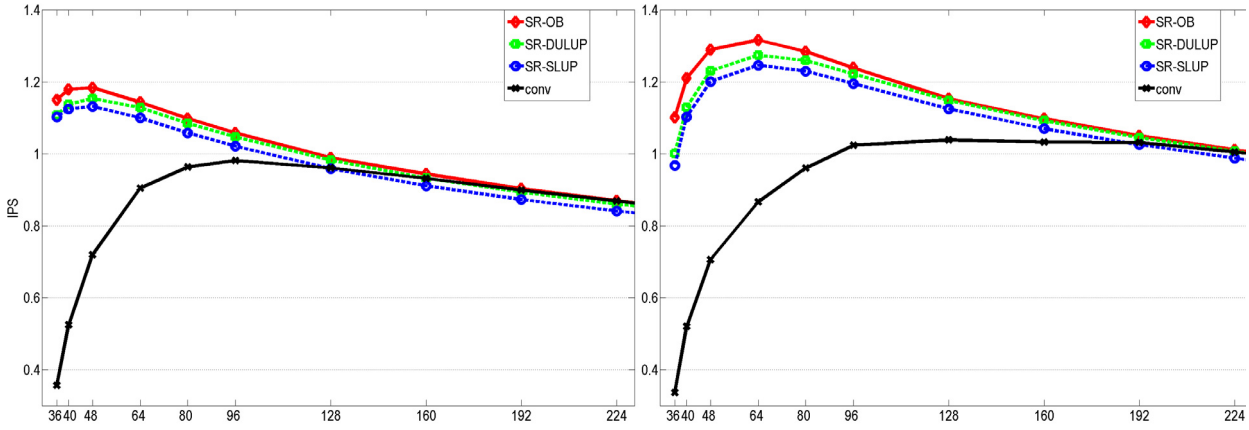
Figura 5.8 IPC vs. tamaño de PRF para renombre convencional, SR-SLUP, SR-DULUP y SR-OB. Para cada PRF de SR-SLUP y SR-DULUP, la configuración del XRF es la mostrada en la Tabla 5.1. El XRF del oráculo no está limitado por entradas ni puertos.

que la contención en la lectura y escritura del XRF evita el avance de instrucciones con usos inesperados y con registros predichos LU, respectivamente, la etapa de lanzamiento no se bloquea, puesto que otras instrucciones puede avanzar. Y al contrario, la cancelación de una predicción LU supone una pérdida de oportunidad de evitar la asignación de un registro o de planificar una liberación anticipada. Con PRFs ajustados en tamaño, esto puede favorecer las detenciones en renombre por falta de registros físicos, que afectan negativamente a las prestaciones. Para los PRFs con menor número de entradas (hasta 80 registros), puede observarse que 2 ó 3 puertos de lectura en el XRF son suficientes para lecturas por parte de UUs, mientras que los registros predichos como LU (fuente y destino) pueden escribirse con sólo de 5 a 8 puertos. Estas cantidades de puertos distan mucho de las teóricamente necesarias para evitar contención en el XRF (un máximo de 16 y 30 puertos para el XRFint).

5.5 Prestaciones de SR-LUP (IPC e IPS)

La Figura 5.8 muestra para cada tamaño de PRF, el IPC obtenido por los XRFs listados en la Tabla 5.1 con los predictores SLUP y DULUP. Las prestaciones del renombre convencional (conv) y del SR-LUP con un predictor de último uso oráculo (SR-OB) se muestran como cotas de prestaciones inferior y superior. Podemos observar como los beneficios de SR-LUP en códigos de coma flotante son mucho más significativos que en los códigos enteros. Este es un resultado esperado puesto que los programas de coma flotante causan mayores demandas de registros físicos. SR-LUP mejora manifiestamente al renombre convencional y alcanza ganancias cercanas al límite. Para PRFs ajustados (36 y 40 registros), y con cualquier programa, las prestaciones se multiplican por más de dos tanto para SR-SLUP como para SR-DULUP.

Una combinación de un PRF con 80 entradas respaldado por un XRF de 224 entradas con 4 puertos de lectura y 9 de escritura obtiene con SR-SLUP una mejora de IPC del 9.9% y 27.9% para programas enteros y de coma flotante, respectivamente. Con la misma configuración, SR-DULUP, las mejoras son del 12.6% y 31.0%. Con PRFs de más de 128 ó 160 entradas, y con



(a) SPEC CINT2000 **(b) SPEC CFP2000**

Figura 5.9 IPS vs. tamaño de PRF para renombre convencional, SR-SLUP, SR-DULUP y SR-OB. Para cada PRF, la configuración del XRF (también el del oráculo) es la mostrada en la Tabla 5.1.

cualquier configuración SR-LUP, las mejoras se reducen o incluso se vuelven negativas. Al respecto de los límites, SR-DULUP gestionando un PRF de 80 entradas y un XRF de 224 entradas con 4 puertos de lectura y 9 de escritura alcanza el 99% y el 98% del IPC alcanzado por SR-OB, para códigos enteros y de coma flotante, respectivamente. Además, supera el 97% del IPC que se obtendría con un número ilimitado de registros físicos.

Cuando el tiempo de ciclo del procesador está limitado por el tiempo de acceso al PRF, una reducción del tamaño del PRF podría permitir un incremento en la frecuencia y la consecuente mejora de prestaciones. La Figura 5.9 muestra la media de instrucciones por segundo (IPS) que ejecutan procesadores con bancos de registros gestionados por los esquemas analizados (convencional, SR-SLUP, SR-DULUP y SR-OB, todos con los XRFs listados en la Tabla 5.1). Los tiempos de ciclo se han obtenido a partir del modelo de Rixner para bancos de registros centralizados implementados con tecnología de 0.18µ [RDK+00].

El renombre convencional alcanza su cota máxima de prestaciones ejecutando códigos enteros con un PRF de 96 entradas. Con SR-DULUP y un XRF de 192 entradas con 8 puertos, puede reducirse el PRF a 48 entradas obteniendo el mismo IPC y un 17% de incremento en IPS. Con SR-SLUP se obtiene casi el mismo IPC y una mejora del 15% en el IPS. Cuando ejecuta códigos de coma flotante, las mejores prestaciones del renombre convencional son ahora con un PRF de 128 entradas. Combinando un PRF de 64 entradas con un XRF de 224 entradas y 10 puertos, se obtienen mejoras del 2% y 4% de IPC y ganancias de IPS del 20% y 22% (SR-SLUP y SR-DULUP).

5.5.1 Costes de implementación

En este subapartado se compara el coste (en términos de almacenamiento, área y energía) de las configuraciones que maximizan la métrica IPS en la ejecución de códigos enteros (96PRF para renombre convencional vs. 48PRF+192XRF para SR-LUP) y en la ejecución de códigos de coma flotante (128PRF vs. 64PRF+224XRF).

Tabla 5.2 Comparativa de tiempos de acceso, consumo y áreas del PRF y del XRF.

			Área (mm^2)	Consumo (pJ)	Tiempo de acceso (ns)
IPSint máximo	**conv.**	**PRF-96**	1.50	1118	1.44
	SR-LUP	**PRF-48**	0.75	570	1.23
		XRF-192	0.35	251	1.22
IPSfp máximo	**conv.**	**PRF-128**	1.99	1480	1.55
	SR-LUP	**PRF-64**	1.00	753	1.31
		XRF-224	0.57	410	1.31

Al sustituir un PRF con 96 entradas gestionado de forma convencional por una configuración 48PRF+192XRF gestionada con SR-LUP, se requieren unas estructuras adicionales añadidas (registros extra, predictor de último uso, tablas y vectores de bits) con un coste de almacenamiento de 6.10 Kilobytes. Cambiar un PRF de 128 entradas por una conjunción 64PRF+224XRF supone un incremento de almacenamiento que no llega a 6.50 Kilobytes. Ambos costes *hardware* son razonables para un procesador actual.

La Tabla 5.2 recoge las áreas, consumos de energía y tiempos de acceso correspondientes a los PRFs y XRFs de las cuatro configuraciones analizadas. En el cálculo se ha utilizado el modelo de área, consumo de energía y tiempo de acceso de Rixner et al. para un banco de registros centralizado y una tecnología de 180 nm. [RDK+00]. Puede observarse como SR-LUP reduce el área dedicada al banco de registros (la combinación PRF-XRF ocupa menos área que el PRF original). Asimismo, un acceso al PRF de SR-LUP consume la mitad de energía que un acceso al PRF original, y un acceso al XRF supone un ahorro todavía mayor, en torno al 75%. A pesar de estos números, el balance energético global puede ser desfavorable a SR-LUP debido al coste de las reejecuciones por fallos de predicción, al consumo de otras estructuras como el predictor de salto, V2X...

5.6 Conclusiones

Este Capítulo presenta un diseño de microarquitectura que soporta renombre especulativo (SR), proponiendo todas las estructuras y control necesarios para explotar tal alternativa al renombre convencional. Con SR, los valores de los registros que se espera no vayan a tener más consumidores se envían a un almacén auxiliar, el XRF, ubicado fuera de los caminos críticos del procesador. El XRF suministra aquellos valores referenciados inesperadamente que no están en el PRF. Un doble mapeo de registros físicos y virtuales permite la localización de operandos y la gestión de dependencias sin incrementar la complejidad de la ventana de lanzamiento (IQ).

Un predictor de último uso controla las políticas de omisión de asignación y de liberación anticipada de registros físicos utilizadas en la evaluación de la microarquitectura SR. Este diseño microarquitectónico soporta también otro tipo de políticas de renombre especulativas que puedan proponerse, tanto hardware como software.

La microarquitectura ha sido analizada con dos predictores de último uso de complejidad y prestaciones incrementales. Con el predictor sencillo (SLUP), un procesador con una estructura de reordenación de 256 entradas y un PRF con 80 entradas (80int+80fp) puede mejorar su IPC un 8.6% y un 25.3% ejecutando programas enteros y de coma flotante, respectivamente, aplicando SR-LUP con un XRF de 256 entradas. Estas mejoras aumentan al 11.5% y 29% con el mejor predictor (DULUP). En cuanto a los límites, y para las configuraciones anteriores, SR-DULUP alcanza para programas enteros el 98% del IPC obtenido por un predictor oráculo de último uso. Ejecutando códigos de coma flotante, un PRF de 128 entradas (128int+128fp) gestionado de forma convencional, puede reemplazarse por un PRF de 64 entradas (64int+64fp) y un XRF de 224 entradas gestionados mediante SR-DULUP, con una mejora del 1.2% de IPC. Si el PRF limita la frecuencia del procesador, se obtienen ganancias de IPS del 16% y 19% con SLUP y DULUP, con un coste de almacenamiento inferior a 6.5 Kilobytes.

5.7 Referencias

[AMV+06] J. Alastruey, T. Monreal, V. Viñals and M. Valero. “Speculative Early Register Release”. *Proc. of the 3rd conference on Computing frontiers* (CF 06), May 2006, pp 291-302.

[AMV+07] J. Alastruey, T. Monreal, V. Viñals and M. Valero. “Microarchitectural Support for Speculative Register Renaming”. *IEEE Int. Parallel & Distributed Processing Symp.* (IPDPS 07), March 2007.

[AMV+09] J. Alastruey, T. Monreal, M. Valero y V. Viñals. "Implementación de un Predictor de Último Uso con Decaimiento". *XX Jornadas de Paralelismo*, La Coruña (España). 17-18 Sept. 2009.

[BSP+06a] D. Balkan, J. Sharkey, D. Ponomarev and A. Aggarwal, “Address-Value Decoupling for Early Register Deallocation”, Proc. 35th Int’l Conf. on Parallel Processing (ICPP 06), Aug. 2006, pp. 337-346.

[BuSo02a] J.A. Butts and G. Sohi. "Dynamic dead-instruction detection and elimination". *Proc. 10th Int’l Conf. Architectural Support for Programing Languages and Operating Systems* (ASPLOS 02), Oct. 2002, pp. 199-210.

[BuSo02b] J.A. Butts and G. Sohi, Characterizing and Predicting Value Degree of Use”, *Proc. 35th Int’l Symp. Microarchitecture* (MICRO 02), Nov. 2002, pp. 15-26.

[BuSo04] J.A. Butts and G. Sohi, Use-Based Register Caching with Decoupled Indexing. *Proc. 31st Int’l Symp. Computer Architecture* (ISCA 02), pp. 302. Munich (Alemania), Jun. 2004.

[ChEm98] G. Z. Chrysos, J.S. Emer. “Memory dependence prediction using store sets”. *Proc. International Symposium on Computer Architecture* (ISCA). 1998, pp. 142 - 153.

[MGV+99] T. Monreal, A. González, M. Valero, J. González and V. Viñals, "Delaying Physical Register Allocation Through Virtual-Physical Registers". *Proc. 32nd Int’l Symp. Microarchitecture* (MICRO 99), Nov. 1999, pp.186-192.

[RDK+00] S. Rixner, W.J. Dally, B. Khailany, P. Mattson, U.J. Kapasi, and J.D. Owens, Register Organization for Media Processing. P*roceedings of the 6th International Symposium on High-Performance Computer Architecture* (HPCA 00), pp. 375-386, January 2000.

[TIV+03] E. Torres, P. Ibáñez, V. Viñals and J.M. Llabería. Counteracting Bank Misprediction in Sliced First-Level Caches. *In proc. 9th Int’l Conf. on Parallel and Distributed Computing (Euro-Par 03)*, pp. 586-596. Aug. 2003.

[YER+99] A. Yoaz, M. Erez, R. Ronen, S. Jourdan. Speculation Techniques for Improving Load Related Instruction Scheduling. ISCA 1999: pp. 42-53.

CAPÍTULO 6

Implementación de un Predictor de Último Uso con Decaimiento

Las técnicas de decaimiento permiten reducir el consumo estático de energía. Este Capítulo plantea su aplicación a predictores de tipo sticky, ya que, por su especial funcionamiento, sus prestaciones no se degradan e incluso pueden mejorar con la descarga de sus celdas de memoria. En concreto, hemos evaluado dos implementaciones decaídas de un predictor sticky que trata de identificar los registros último uso de una instrucción (Sticky Last Use Predictor, SLUP). El primer diseño está basado en la técnica de conmutación de alimentación (gated-Vdd con celdas SRAM-6T) mientras que el segundo utiliza celdas de memoria cuasi-estáticas (RAM-4T). Hemos observado que ambas implementaciones alcanzan, para todos los tamaños de predictor analizados, el mismo rendimiento que el diseño SLUP convencional pero con un porcentaje medio de entradas activas que va desde el 31% para un predictor de 4k entradas hasta un 67% para un predictor de 512 entradas. Por otra parte, se ha observado que en la tecnología evaluada (160 nm.), la energía necesaria para recargar las celdas RAM-4T puede anular la reducción de consumo estático obtenida por dicha técnica.

6.1 Introducción

La reducción del consumo de energía y potencia del procesador se ha convertido en los últimos años en un tema de investigación de gran interés, tanto para la comunidad académica como para la industria. En el segmento de los dispositivos portátiles, aumentar el tiempo de autonomía es un objetivo prioritario. En cuanto a los procesadores de altas prestaciones, su mejora de rendimiento ha acarreado un considerable incremento de la energía consumida y del calor a disipar, lo que requiere sistemas de refrigeración cada vez más complejos.

La energía consumida por un procesador tiene dos componentes: la dinámica, causada por la actividad de conmutación, y la estática, que tiene su origen en las corrientes de fuga de los transistores. Los avances en la capacidad de integración han permitido mantener más o menos

constante el consumo dinámico, ya que en cada generación se ha escalado la tensión de alimentación. Sin embargo, al reducirse igualmente la tensión umbral de conmutación (Vt), las corrientes de fuga han ido aumentando, y se espera que en próximas generaciones se conviertan en la principal fuente de disipación energética en los procesadores CMOS de altas prestaciones.

Por todo lo anterior, se han propuesto varios mecanismos a nivel circuital para reducir el consumo estático. Un primer grupo de trabajos propone incrementar la tensión umbral del transistor (por ejemplo, alargando el canal del transistor) para disminuir la corriente de fuga y por tanto, el consumo estático [MDM+95]. Como también reduce la velocidad de conmutación, este mecanismo se utiliza en circuitos que no están en caminos críticos. Por ejemplo, se aplica en el tercer nivel de cache en-chip del Intel Itanium 2 [StRu03]. En una versión reciente del Intel Xeon (8 núcleos Nehalem de 45nm), alrededor del 58% de los transistores de cada núcleo son de canal efectivo largo (*long-Le*) [RTM+09]. En el IBM Power6, los circuitos lógicos se implementan con tensión umbral alta, normal o baja, según se encuentren en caminos con suficiente margen temporal, cerca del camino crítico o en caminos muy críticos. Debido a su alto consumo estático, sólo el 1% de los transistores del núcleo son de baja tensión umbral [CFP+07]. El UltraSPARC T2 también utiliza un proceso CMOS con triple tensión umbral.

Otros trabajos plantean la deshabilitación de partes del procesador que no están siendo usadas, pero que contribuyen a gran parte del consumo estático. De mayor a menor granularidad encontramos desde la desconexión de un determinado núcleo en procesadores multinúcleo [KuHi09] hasta la deshabilitación selectiva de partes de estructuras de memoria ubicadas dentro del chip, como caches y predictores de salto [PYF+00][KHM01][FKM+02][HJD+02][JSM+04]. En unos casos se preserva el estado de la zona deshabilitada, por ejemplo mediante el volcado del estado arquitectónico del núcleo a memoria principal [KuHi09] o mediante el suministro de una tensión de alimentación reducida que mantiene los contenidos de las líneas de memoria cache no usadas durante un determinado período de tiempo (*drowsy caches*) [FKM+02]. En otros casos se desconecta la alimentación de las celdas de memoria, por lo que no se mantiene su estado lógico, pero a cambio se logran mayores ahorros de consumo estático de energía. La desconexión puede implementarse mediante la conmutación de un transistor de paso (*gated-Vdd,* ver Figura 6.1b) [PYF+00] o con celdas RAM cuasi-estáticas de cuatro transistores (4T, ver Figura 6.1c) [HJD+02]. *Gated-Vdd* se utilizó originalmente para implementar una cache de instrucciones con ajuste dinámico de tamaño (DRI) [PYF+00] y posteriormente en las llamadas caches decaídas (*decay caches*) [KHM01]. Las celdas 4T se han propuesto para implementar predictores de salto decaídos [JSM+04].

Algunos de estos mecanismos pueden aplicarse conjuntamente, como por ejemplo, en el tercer nivel de cache del Intel Xeon (Nehalem). Cuando se detecta que un *slice* de L3 no ha sido usado durante un determinado período de tiempo, su tensión de alimentación se reduce de 0.90V a 0.75V, lo que permite disminuir el consumo estático en torno al 35%. Si se constata mayor inactividad, la alimentación se reduce hasta 0.36V, con lo que puede ahorrarse el 83% de consumo estático [RTM+09].

Todas las propuestas anteriores disminuyen el consumo estático del procesador, pero también reducen su rendimiento, ya que la estructura modificada es más lenta o menos eficaz. En concreto, *dual-Vt* reduce la velocidad de conmutación del transistor*;* en las *drowsy caches*, la reac-

tivación de las líneas con alimentación reducida retarda su acceso; y las caches y los predictores de salto con decaimiento tienen mayores tasas de fallos por la pérdida de información en las celdas descargadas.

Nuestro objetivo es disminuir el consumo estático sin afectar al rendimiento de la estructura modificada. Esto es posible en los predictores tipo *sticky* por sus características especiales. Con el objetivo de simplificar su implementación, estas estructuras especulativas almacenan información asociada a una instrucción que posteriormente no actualizan. Por ejemplo, si un predictor *sticky* registra una colisión entre un load y un store previo, posteriores ejecuciones del load siempre se señalarán como dependientes del store, aunque no lo sean. En las implementaciones convencionales de este tipo de predictores, la única forma de suprimir dicha información es por el reemplazo de la entrada. Para forzar la eliminación de contenidos obsoletos acumulados en un predictor de dependencia store-load, se ha sugerido un borrado periódico [ChEm98] [YER+99]. La pérdida de contenidos asociada al decaimiento, ya sea por conmutación de alimentación o celdas cuasi-estáticas, puede implementar de forma natural este borrado periódico.

Las contribuciones de esta Tesis en esta línea de trabajo son las siguientes:

- Aplicación de técnicas de decaimiento a un predictor de tipo *sticky* que trata de anticipar si los operandos de una instrucción son los últimos en referenciar un registro (*Sticky Last-Use Predictor*, SLUP) [AMV+07].

- Evaluación de dos predictores SLUP con decaimiento, uno implementado con el mecanismo de conmutación de alimentación (*gated-Vdd*) y otro con celdas de memoria cuasi-estáticas de 4 transistores (RAM-4T).

- Análisis de la sensibilidad al intervalo de decaimiento de los predictores anteriores.

La estructura de este Capítulo es la siguiente: la Sección 6.2 explica los mecanismos circuitales utilizados para soporte de estructuras decaídas. La Sección 6.3 describe el predictor *sticky* de último uso de un registro (SLUP) y sus implementaciones decaídas con *gated-Vdd* y con celdas cuasi-estáticas. La Sección 6.4 detalla la metodología experimental y analiza los resultados de la evaluación de los predictores SLUP decaídos. La Sección 6.5 resume las conclusiones del Capítulo.

6.2 Implementación de Estructuras Decaídas

La desconexión de la alimentación de una celda de memoria disminuye la intensidad de sus corrientes de fuga, lo que permite reducir gran parte de su consumo estático de energía. Sin embargo, la diferencia de tensión almacenada en una celda sin alimentación decae y se acaba perdiendo su valor lógico. Por esta razón, estas técnicas son apropiadas para estructuras que almacenan réplicas de datos (caches) [KHM01] o información de carácter predictivo (predictores de salto) [JSM+04]. El decaimiento puede implementarse mediante conmutación de alimentación con celdas SRAM de 6 transistores (*gated-Vdd*) [PYF+00] o con celdas RAM de 4 transistores [HJD+02]. Ambas opciones se describen a continuación.

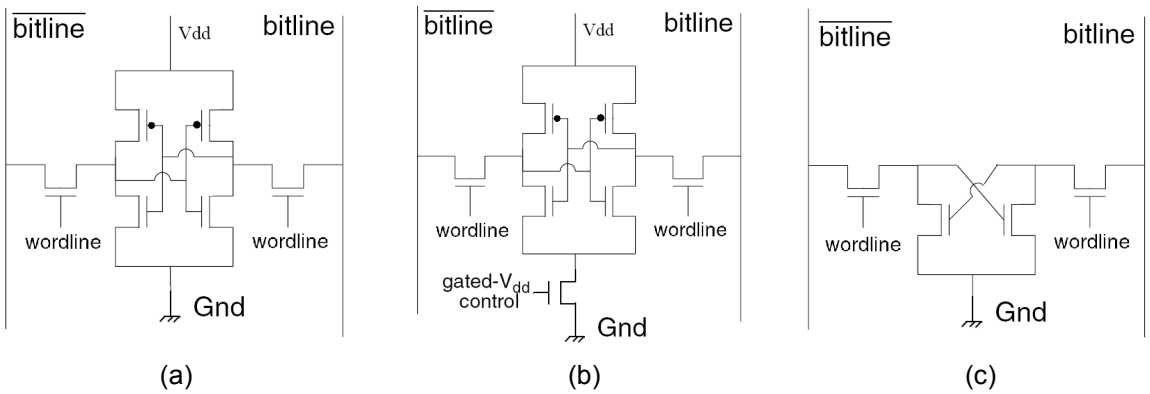


Figura 6.1 Celda SRAM de 6 transistores (a), celda SRAM con transistor de paso NMOS (b) y celda RAM cuasi-estática de 4 transistores (c). Diagramas adaptados de [HJD+02] y [PYF+00]

6.2.1 Decaimiento con celdas SRAM-6T

Las propuestas tradicionales de apagado de celdas de memoria SRAM de 6 transistores (SRAM-6T) se implementan con el mecanismo de conmutación de alimentación (*gated-Vdd*). Un transistor de paso añadido controla la conexión de la celda SRAM a la fuente de alimentación (Figura 6.1b). Cuando el transistor de paso está activo, la celda funciona como una convencional. Cuando se desconecta, corta la alimentación de la celda, que se descarga y pasa a un estado lógico indefinido. El transistor de paso puede ser compartido por varias celdas, lo que aumenta la granularidad de la desconexión.

Las caches decaídas utilizan este mecanismo para desconectar aquellas líneas que no han sido accedidas en un determinado período de tiempo, perdiendo por tanto sus contenidos [PYF+00].

La conmutación de alimentación tiene varios inconvenientes. En primer lugar, los transistores de paso añadidos para la conmutación de las celdas pueden incrementar el área de la estructura que se pretende decaer. Además, se requieren estructuras adicionales para detectar las celdas que no están siendo accedidas, identificar las desconectadas y controlar dinámicamente los transistores de paso. Este hardware añadido supone un consumo adicional (estático y dinámico), lo que podría minimizar el ahorro estático de energía obtenido al desconectar celdas.

6.2.2 Decaimiento con Celdas RAM-4T

Una implementación alternativa y más elegante de una estructura decaída se basa en el uso de celdas de memoria cuasi-estáticas de 4 transistores (RAM-4T). A diferencia de las celdas convencionales de 6 transistores (SRAM-6T), estas celdas carecen de los dos transistores de conexión a la alimentación (ver Figura 6.1a y c). Las celdas 4T se cargan en cada acceso (lectura o escritura) y después su tensión va decayendo debido a las corrientes de fuga (*leakage*). Pasado el denominado tiempo de retención no es posible recuperar el valor lógico almacenado en una celda decaída. La velocidad de descarga depende de varios parámetros tecnológicos de la celda y de su temperatura de funcionamiento.

En lo concerniente a la regeneración de los contenidos de una celda 4T, la activación de la línea de selección de fila (*wordline*) permite que el circuito que precarga las líneas de bit (*bitlines*) refresque los contenidos de todas las celdas pertenecientes a dicha fila [HJD+02]. Por consiguiente, la localidad espacial en los accesos puede aumentar el tiempo efectivo de retención. Mediante líneas de selección de fila segmentadas podrían conseguirse refrescos más selectivos.

Las celdas RAM-4T ocupan menos área (aproximadamente 2/3) que las celdas SRAM convencionales de 6 transistores y consumen menos energía ya que, una vez descargadas, prácticamente no existe corriente de fuga al no tener conexión a alimentación. La latencia de lectura de las celdas 6T y 4T es similar puesto que el efecto causado por no disponer de conexión a alimentación es compensado por la menor capacidad de la celda. A diferencia de una técnica de decaimiento basada en celdas SRAM-6T (*gated-Vdd*), una técnica basada en celdas de memoria cuasi-estáticas no necesita hardware añadido para detectar la inactividad de las celdas ni para forzar su descarga.

6.3 Predictor de Último Uso *Sticky* Decaído

Proponemos aplicar estas técnicas para descarga de celdas de memoria en la implementación de predictores *sticky*. El característico funcionamiento de este tipo de predictores sugiere que la pérdida de memoria por la desconexión de entradas podría mejorar las prestaciones del predictor y a la vez reducir el consumo estático de energía. Como aplicación de uso concreta, evaluaremos dos implementaciones decaídas de un predictor de último uso *sticky* (SLUP), una basada en *gated-Vdd* y otra en celdas de memoria cuasi-estáticas. El predictor de último uso se utiliza para dirigir políticas especulativas de asignación y liberación de registros físicos en la microarquitectura con renombre especulativo descrita en el Capítulo anterior [AMV+06][AMV+07].

6.3.1 Último Uso de Registros

Una instrucción contiene un operando último uso (*Last Use*, LU) de un registro lógico si es la última instrucción de camino correcto y en orden de programa que utiliza un valor contenido en dicho registro lógico, ya sea para consumirlo o para producirlo.

En el Capítulo 4 se ha definido el patrón de uso de una instrucción como la combinación de información sobre el último uso de sus operandos [AMV+06]. También se han analizado varias propiedades de estos patrones de uso (frecuencia dinámica y predecibilidad) para motivar el diseño de un predictor sencillo de patrones de uso (*Last Use Predictor*, LUP). En el Capítulo 5 se propone una microarquitectura que utiliza un LUP como estructura clave para controlar sus agresivas políticas de asignación y liberación de registros físicos [AMV+07]. Por un lado, las instrucciones con predicción “es último uso” para alguno de sus registros lógicos fuente liberan anticipadamente el correspondiente registro físico tras consumirlo (*Early Release of Physical Registers*, ERPR). Por otro lado, no se asigna registro físico a las instrucciones que reciben una predicción “es último uso” para su registro destino (*Omission of Physical Register Allocation*, OPRA). En el Capítulo 5 se pueden consultar más detalles de esta microarquitectura así como las características de dos diseños de LUP. En este capítulo nos centramos en mejorar el más sencillo: el predictor de último uso *sticky* (*Sticky Last-Use Predictor*, SLUP).

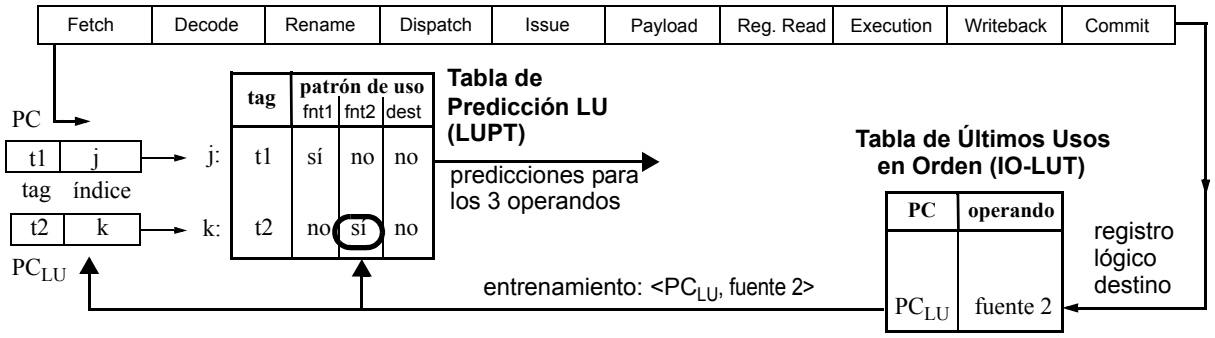


Figura 6.2 Implementación de Predictor de Último Uso Sticky (SLUP): Tabla de Predicción LU y Tabla de Últimos Usos en Orden IO-LUT.

6.3.2 Predictor de Último Uso Sticky (SLUP)

Tal como se ha descrito en el Capítulo 5, el predictor SLUP está formado por dos estructuras: la Tabla de Predicción LU (LUPT) que almacena las predicciones de último uso y la Tabla de Últimos Usos en Orden (IO-LUT) que se encarga de entrenar la LUPT. La Figura 6.2 detalla la organización de ambas estructuras.

Un predictor de último uso puede cometer dos tipos de fallos de predicción. Un fallo al predecir "no es último uso" supone perder la oportunidad de omitir la asignación de un registro físico o de liberarlo anticipadamente (OPRApred o ERPRpred, respectivamente). Un fallo al predecir "es último uso" puede degradar las prestaciones si el valor excluido del banco de registros físico (PRF) tiene que ser recuperado del banco de registros auxiliar (XRF), ya que su latencia de acceso es superior a la del PRF (para más detalles, consultar el Capítulo 5).

En cada entrenamiento, el predictor registra el último uso de un operando de una instrucción (ver Figura 6.2). Aunque posteriores ejecuciones de dicha instrucción no tengan el mismo patrón de uso, con este mecanismo no es posible revocar una predicción (predictor *sticky*). Una predicción de último uso de un registro sólo puede anularse si su entrada en el SLUP es reemplazada. Por tanto, el comportamiento *sticky* mantiene en la LUPT patrones de uso obsoletos que pueden causar un aumento de los costosos fallos prediciendo "es último uso". En [ChEm98] y [YER+99] se sugiere el borrado periódico de un predictor de dependencias store-load para eliminar la información obsoleta, aún a expensas de eliminar también la reciente. Una tabla de predicción decaída puede implementar de forma natural este borrado periódico.

6.3.3 Diseños de SLUP decaído

A continuación describimos dos implementaciones de un SLUP decaído (SLUPd), una con la técnica de conmutación de alimentación de celdas SRAM-6T (*gated-Vdd*) y otra con celdas RAM de 4 transistores (RAM-4T). En ambos diseños, que llamamos SLUPd-6T y SLUPd-4T, consideraremos una disposición física de la LUPT lo más cuadrada posible para minimizar área y latencia. Por ejemplo, un SLUP de 2048 entradas con 3 bits para almacenar el patrón de uso y 5 bits para las etiquetas tendrá una estructura física de 128 filas de 16 entradas (128 x 128 bits).

SLUPd-6T. El funcionamiento de un predictor con decaimiento implementado con celdas SRAM-6T es el detallado en [JSM+04]. Cuando una entrada no ha sido accedida en un determinado período de tiempo, se considera que ha decaído y se desconecta de la alimentación. Si se trata de acceder a una entrada desconectada, ésta se vuelve a conectar y se reinicializa su estado a uno prefijado, en nuestro caso, "no es último uso" para todos los operandos.

Si el período de tiempo, llamado intervalo de decaimiento, es muy corto, habrá más posibilidades de desactivar entradas, pero también se acorta la vida de las predicciones almacenadas, lo que puede afectar al rendimiento del predictor. El intervalo de decaimiento es fijado por unos contadores globales [JSM+04].

El control de activación de celdas a nivel de entrada de predictor requiere una excesiva sobrecarga de estructuras de control añadidas. Una implementación menos costosa controla la activación a nivel de fila [JSM+04]. Las entradas de una misma fila comparten 2 bits de estado: uno para indicar si alguna entrada de la fila ha sido accedida o no en el último intervalo de decaimiento y otro para indicar si la fila está activa o decaída.

SLUPd-4T. En esta implementación de predictor decaído, las celdas se descargan de forma natural tras su último acceso, sin necesidad de ninguna estructura de control añadida. Pasado el tiempo de retención, una lectura de la celda encontrará una diferencia de tensión en sus nodos por debajo del umbral de detección del *sense amplifier*, por lo que no podrá recuperarse su valor lógico. Por tanto, el intervalo de decaimiento queda en este caso fijado por el tiempo de retención. Para 0.16μ, el tiempo de retención es de 1700 ns. en celdas 4T con transistores que trabajan a 1.5V y de 57200 ns. en celdas 4T con transistores a 3.3V. Debido al aumento de las corrientes de fuga, estos tiempos decrecen con cada nueva generación tecnológica. Sin embargo, la técnica tiene potencial de aplicación para tecnologías futuras, ya que los tiempos de retención medidos en ciclos de procesador disminuyen en menor medida por el aumento de la frecuencia de procesador [JSM+04].

Cuando se lee una entrada, se selecciona una fila del predictor y el circuito de precarga regenera todos los contenidos de dicha fila. Si existe localidad espacial en los accesos al SLUPd, el tiempo efectivo de retención aumenta.

Existen varios problemas asociados a esta implementación:

- La lectura de una celda decaída produce un valor eléctricamente estable, aunque lógicamente aleatorio [JSM+04]. Para evitar el uso del valor leído de una celda decaída, en [JSM+04] se propone añadir al predictor una columna de celdas conectadas a un comparador de tensión en lugar de a un *sense amplifier*. Para cada fila, esta columna permite detectar si la diferencia de tensión entre los nodos de una celda de dicha fila se encuentra por debajo del umbral de detección del *sense amplifier*.

- Al leer una fila cuyas celdas se han descargado, la precarga puede escribir un valor diferente al que había en las celdas. Además, la celda de referencia será cargada, por lo que posteriores lecturas pueden dar valores aparentemente válidos. Para evitar esto, proponemos un vector de bits que indique la validez de los contenidos de una fila. Para una fila determinada, su bit de validez se desactiva cuando se realiza la lectura de una fila decaída.

- La escritura en una celda de una fila descargada provoca que en una fila convivan celdas cargadas y decaídas. Para evitar esta situación, proponemos que cuando se escriba en una celda de una fila decaída, se escriba asimismo en el resto de celdas de la fila la predicción "no es último uso". De este modo, la columna de referencia mantiene una diferencia de tensión coherente con la de todas las celdas de la fila.

Para forzar refrescos selectivos, habría que segmentar las líneas de selección de palabra (*wordlines*). En este caso y a diferencia del anterior, la localidad espacial no aumenta el tiempo efectivo de retención. En una primera aproximación, implementar este esquema requiere una columna de referencia ligada a cada columna del predictor y un bit de validez por entrada. El coste añadido de estas estructuras hace poco atractivo este diseño.

6.4 Evaluación de SLUPd-6T y SLUPd-4T

6.4.1 Entorno Experimental

Se ha usado la metodología y el entorno experimental descritos en el Capítulo 3. Las configuraciones de los bancos de registros (64int+64fp para el PRF, y 224int+224fp para el XRF, con 3 puertos de lectura y 7 de escritura) se corresponden con uno de los puntos de diseño que mejores prestaciones alcanza y que se ha obtenido en el Capítulo anterior. En los dos diseños SLUPd, las etiquetas son de 5 bits y la tabla de últimos usos en orden (IO-LUT) es decaída.

Para convertir a ciclos de procesador los tiempos de retención antes citados, hemos aproximado el tiempo de ciclo con el tiempo de acceso al PRF, que para una tecnología de 0.18µ es de 1.34 ns. [RDK+00]. Por tanto, los tiempos de retención son de 1300 y 42700 ciclos para las celdas alimentadas a 1.3V y 3.3V respectivamente. Una tecnología de 0.16µ permitiría un tiempo de ciclo menor, y por tanto incrementaría los tiempos de retención. Este aspecto se ha considerado en la selección de valores de intervalo de decaimiento explorados.

6.4.2 Sensibilidad al intervalo de decaimiento

La *tasa de actividad* es el porcentaje medio de entradas activas del predictor (entradas no decaídas). Esta medida puede usarse para aproximar la reducción de consumo estático, de hecho, marca el límite inferior de consumo estático que obtendría una técnica de decaimiento si las estructuras de control añadidas no consumieran energía. La Figura 6.3 muestra la variación de la tasa de actividad con el intervalo de decaimiento para SLUPd-6T de varios tamaños (512, 1k, 2k y 4k entradas). Se muestran las tasas de actividad de las dos estructuras que forman el predictor (LUPT e IO-LUT).

Como era de esperar, la tasa de actividad de la LUPT crece con el intervalo de decaimiento para todos los tamaños de predictor. Cuanto mayor es el intervalo, existe mayor probabilidad de que una fila sea accedida en ese período de tiempo, y por tanto habrá menos oportunidades de desconectar filas. Para un intervalo de decaimiento dado, la tasa de actividad de la LUPT crece al disminuir el tamaño del predictor. Este efecto es debido a la menor probabilidad de encontrar una fila no accedida. En el caso de la IO-LUT, la tasa de actividad no cambia con el tamaño de predictor ya que el número de filas de la IO-LUT es fijo (sólo cambia ligeramente el número de

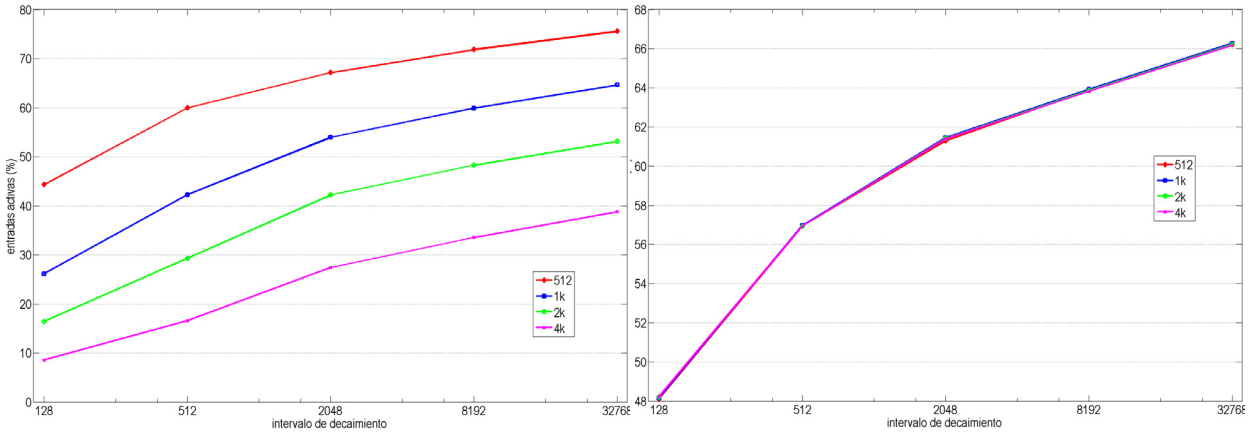


Figura 6.3 Tasa de actividad vs. intervalo de decaimiento para distintos tamaños de SLUPd-6T (512, 1k, 2k y 4k entradas) para las estructuras LUPT (a) e IO-LUT (b).

bits de sus entradas). En los programas enteros se observan menores tasas de actividad gracias al decaimiento de las entradas correspondientes a los registros de coma flotante. Los resultados para SLUPd-4T son muy similares, tanto para la LUPT como para la IO-LUT.

La Figura 6.4 muestra la sensibilidad del IPC al intervalo de decaimiento para varios tamaños de SLUPd-6T (512, 1k, 2k y 4k entradas). El IPC está normalizado al IPC que obtiene un oráculo último uso. A su vez, la sensibilidad del IPC depende de los fallos y aciertos de predicción (Figura 6.5). Las gráficas correspondientes a la implementación SLUPd-4T son muy similares por lo que no se muestran.

Como se puede observar, las prestaciones obtenidas por SLUPd-6T varían con el intervalo de decaimiento. Con el menor intervalo de decaimiento (128 ciclos[1]), ningún tamaño de predictor alcanza el 85% del IPC máximo. Ni la localidad temporal ni la localidad espacial de los accesos logran mantener activas muchas entradas con información útil, incluso en el caso de predictores relativamente pequeños (512 entradas). Todos los predictores mejoran su IPC conforme aumenta el intervalo de decaimiento hasta 2048 ciclos, excepto el predictor de 4k entradas, que continua mejorando hasta 8192 ciclos y alcanza el 96% del IPC máximo.

El desglose de fallos de SLUPd-6T en la Figura 6.5 muestra que al incrementar el intervalo de decaimiento dentro del primer rango (2048-8192 ciclos), se mejora apreciablemente la precisión identificando patrones con últimos usos (disminuyen los falsos negativos), especialmente en los predictores de mayor tamaño. Por encima de 2048-8192 ciclos, esta mejora prácticamente desaparece mientras que siguen aumentando los fallos prediciendo "es último uso" (falsos positivos). Para grandes intervalos de decaimiento, el efecto de estos costosos fallos ya no compensa las pérdidas de oportunidad causadas por los falsos negativos.

1. Este intervalo es inviable como tiempo de retención para el SLUPd-4T construido con tecnología de 0.16µ.

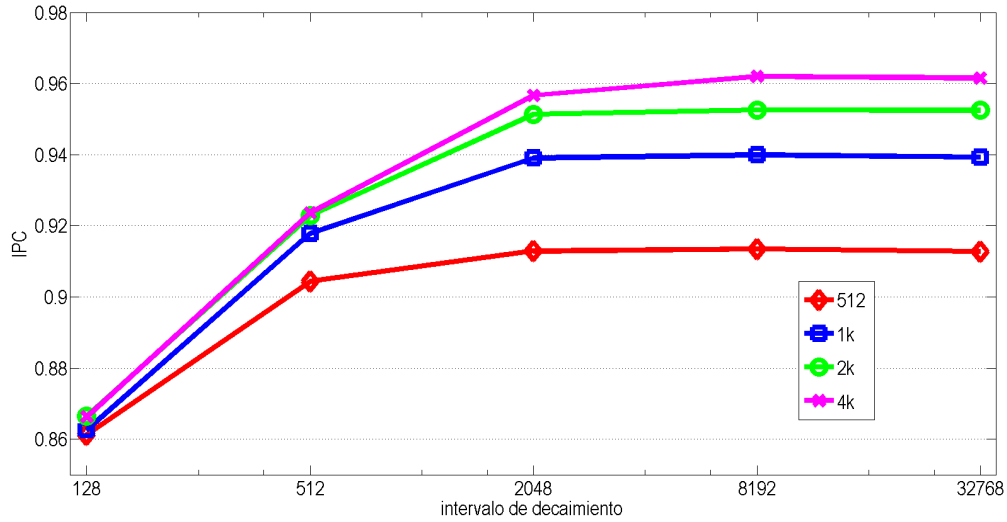


Figura 6.4 IPC medio en SLUPd-6T vs. intervalo de decaimiento para distintos tamaños de predictor (512, 1k, 2k y 4k entradas). El IPC está normalizado al obtenido con un predictor oráculo de último uso.

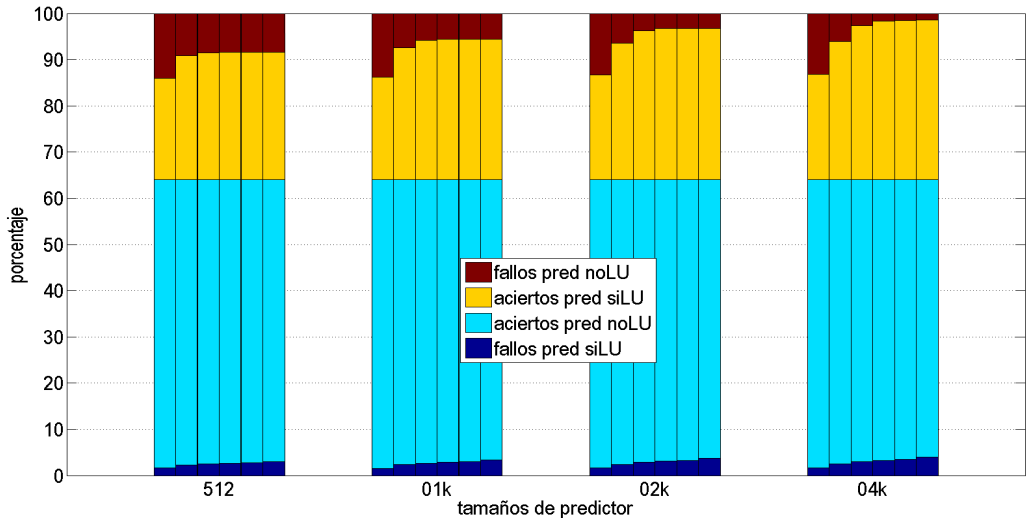


Figura 6.5 Porcentajes de aciertos y fallos de predicción para varios predictores SLUPd-6T y SLUPc. Para cada tamaño de predictor (512-4k entr.) se presentan los porcentajes para SLUPd con varios intervalos de decaimiento (128, 512, 2048, 8192 y 32768, de izq. a der.) y para SLUPc (sin decaimiento). Los segmentos de las barras que representan los fallos están arriba y abajo (falsos negativos y falsos positivos).

6.4.3 Comparación de implementaciones de SLUP

La Figura 6.6 compara las prestaciones de varias implementaciones de SLUP: convencional (SLUPc), convencional con borrado periódico cada 128 kciclos (SLUPreset), SLUPd-4T y SLUPd-6T. La comparativa se efectúa para varios tamaños de predictor (512, 1k, 2k y 4k entradas). Cada tamaño de SLUPd está decaído con el menor intervalo con el que obtiene el 99.5%

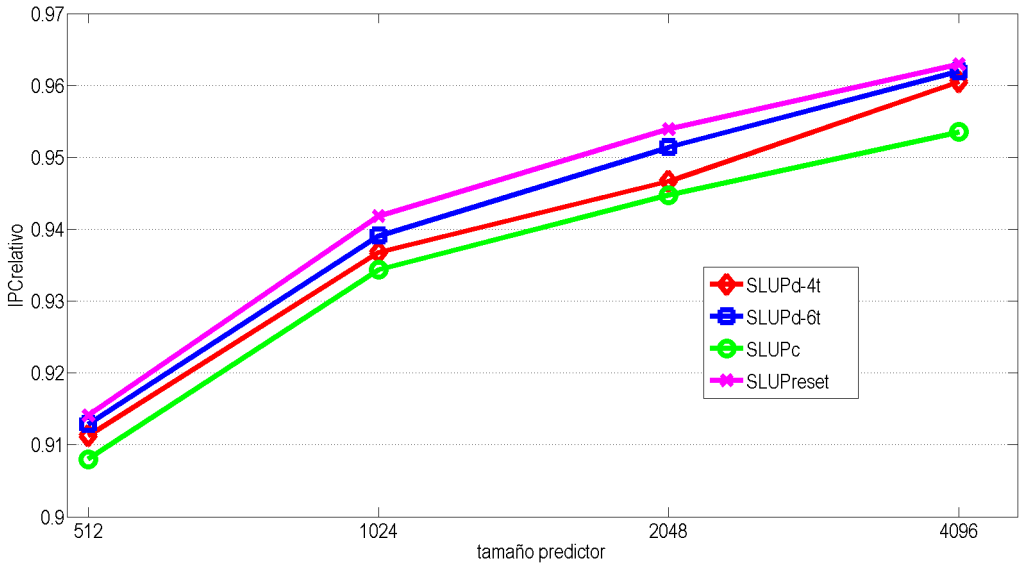


Figura 6.6 IPC medio para distintos tipos de SLUP (convencional, reset, 6T y 4T) con los intervalos de decaimiento mostrados en la Tabla 6.1. El IPC es relativo al obtenido por un predictor oráculo.

Tabla 6.1 Tasa de actividad para distintos tamaños de SLUPd-6T y SLUPd-4T con el mejor intervalo de decaimiento (en ciclos) para cada tamaño.

	Configuración del predictor (entradas/intervalo decaimiento)			
	512/2048c	**1024/2048c**	**2048/2048c**	**4096/8192c**
SLUPd-6T	67%	54%	42%	34%
SLUPd-4T	65%	50%	38%	31%

del IPC máximo alcanzado por dicho tamaño, de esta forma se obtiene un cierto compromiso entre prestaciones y ahorro de consumo estático. También se ha considerado la viabilidad tecnológica del intervalo escogido para SLUPd-4T. Los puntos de diseño seleccionados se listan en la Tabla 6.1.

Las prestaciones que obtienen las distintas implementaciones de SLUP son muy similares. Para todos los tamaños de predictor analizados, SLUPd-4T y SLUPd-6T alcanzan IPCs mayores que SLUPc y un poco menores que SLUPreset. Para el predictor de 2k entradas, esto se consigue con sólo un 40% de entradas activas, porcentaje que se reduce hasta en torno al 30% para SLUPd con 4k entradas (ver Tabla 6.1).

Para cuantificar el ahorro neto en el consumo estático de energía, se combina la reducción de consumo estático observada en la estructura decaída con el consumo añadido (estático y dinámico) para el funcionamiento de cada técnica de decaimiento. En concreto, para cada diseño de predictor decaído se han considerado los siguientes consumos:

- SLUPd-6T: consumo estático de los bits de estado añadidos.

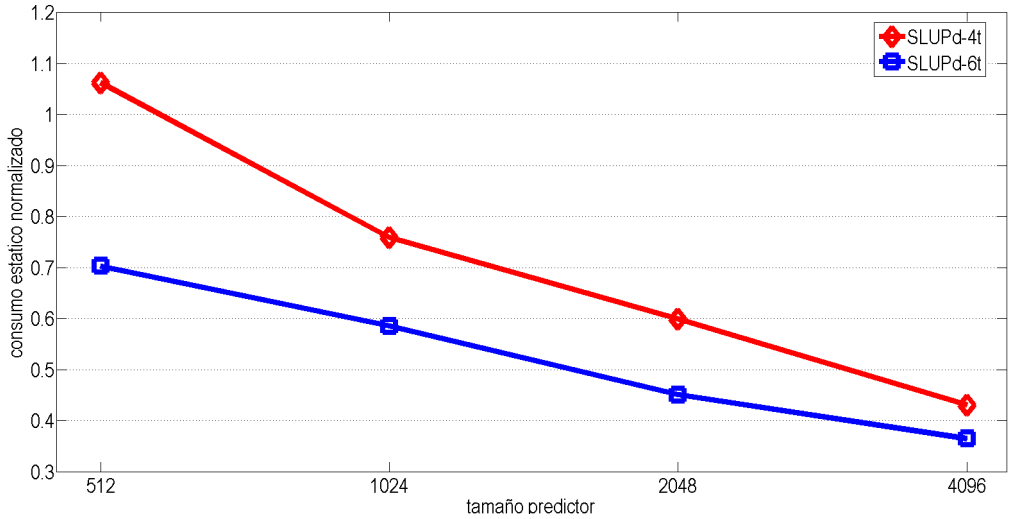


Figura 6.7 Consumo estático de energía normalizado al de SLUP convencional para distintos tamaños de SLUPd-4T y SLUPd-6T. Los intervalos de decaimiento son los mostrados en la Tabla 6.1.

- SLUPd-4T: consumo estático del bit de validez añadido, consumo dinámico extra originado por la recarga de las celdas de una fila en cada lectura, y consumos estático y dinámico de la celda de referencia.

Se ha despreciado el consumo dinámico de los bits añadidos ya que cambian a muy baja frecuencia (como máximo, una vez por intervalo de decaimiento).

El consumo estático de energía normalizado se obtiene dividiendo el consumo estático neto por el consumo estático del predictor convencional [JSM+04], y permite cuantificar el ahorro en el consumo estático con más precisión que la tasa de actividad. Hemos calculado esta métrica para una tecnología de 160 nm. a partir de los parámetros tecnológicos publicados en [JSM+04] (consumo estático por celda de memoria y ciclo, consumo por lectura de celdas 4T y 6T). Dado que la contribución de la componente estática al consumo de energía aumenta con cada generación tecnológica, el ahorro de consumo estático de energía será más significativo para tecnologías más recientes.

La Figura 6.7 muestra el consumo estático de energía normalizado de SLUPd-4T y SLUPd-6T para varios tamaños de predictor (512, 1k, 2k y 4k entradas). Para todos los tamaños, SLUPd-6T alcanza mayores reducciones de consumo que SLUPd-4T. SLUPd-6T obtiene unas reducciones de consumo muy similares a las que pueden aproximarse a partir de sus tasas de actividad. Los dos bits añadidos a cada fila del predictor escasamente afectan al ahorro de energía. Sin embargo, SLUPd-4T se queda lejos del límite marcado por sus tasas de actividad. La energía necesaria para recargar las celdas en cada acceso afectan al balance neto de energía, especialmente en los predictores pequeños, ya que son los que mayores tasas de actividad presentan. Por ejemplo, el consumo estático neto de un SLUPd-4T con 512 entradas es un 6% mayor que el de un SLUP sin decaimiento (convencional).

A la vista de estos resultados, el diseño SLUPd-6T es más atractivo que SLUPd-4T para la tecnología que ha sido evaluada (160 nm.). Sin embargo, dado que en cada nueva generación tecnológica aumenta la fracción de consumo estático sobre el gasto total de energía, es posible que con tecnologías recientes, una implementación basada en celdas RAM-4T consuma menos energía total. En tal caso, suponiendo que las celdas 4T y 6T activas fugan de forma similar, se cumpliría la siguiente desigualdad:

$$\text{consumo_extra_6t} > \text{consumo_extra_4t}$$

Siguiendo la siguiente notación:

- Ce : consumo estático de energía por celda 6T y ciclo.
- Cd : consumo dinámico por lectura de una celda 6T.
- k: fracción de Cd necesaria para leer una celda 4T completamente descargada (peor caso).
- Ta : tasa de actividad.

Y asumiendo que la estructura dominante en el SLUPd es la LUPT, compuesta por nfilas, y cuyas entradas son de 8 bits, entonces se tiene:

$$\text{nfilas} \times \text{nciclos} \times \text{Ce} \times 2 > \text{nfilas} \times \text{nciclos} \times \text{Ce} \times (1 + \text{Ta}) + \text{lecturas_slup} \times \text{Cd} \times \text{k} \times 8$$

Operando, se llega a la siguiente relación entre consumo dinámico (Cd) y estático (Ce) de una celda 6T que tiene que cumplirse para que el diseño SLUPd-4T consuma menos energía:

$$\frac{\text{Cd}}{\text{Ce}} < \frac{\text{nfilas} \times \text{nciclos} \times (1 - \text{Ta})}{\text{lecturas_slup} \times \text{k} \times 8}$$

Suponiendo que k no varía con cada generación tecnológica, puede aplicarse su valor para 160 nm. (0.27, obtenido de [JSM+04]). Y sustituyendo el resto de parámetros por los observados en nuestros experimentos para un predictor de 2k entradas, se llega a que la relación entre consumo dinámico y estático tiene que ser menor que 10 para que SLUPd-4T ahorre mayor cantidad de energía que SLUPd-6T[2]. Según [JSM+04], el valor de este cociente para 160 nm. es 23, de ahí que SLUPd-6T sea la mejor alternativa. Para las tecnologías actuales (45 nm), se barajan cocientes entre consumo dinámico y estático en torno a 2.5, por lo que SLUPd-4T se convertiría en la mejor opción de diseño.

6.5 Conclusiones

En este capítulo se propone la aplicación de técnicas de decaimiento a predictores de tipo *sticky*. En concreto, se han diseñado dos predictores de último uso decaídos, uno implementado con transistores de paso (*gated-Vdd*, SLUPd-6T) y otro con celdas de memoria cuasi-estáticas (RAM-4T, SLUPd-4T).

2. Los cocientes para los predictores de 512, 1k, 2k y 4k entradas son 5, 14, 10 y 22, respectivamente.

Se ha comprobado que, para todos los tamaños evaluados de SLUPd-6T, existen intervalos de decaimiento con un gran potencial de ahorro de consumo estático de energía sin perder rendimiento respecto al SLUP convencional. Por ejemplo, un SLUPd-6T de 2k entradas con un intervalo de decaimiento de 2048 ciclos obtiene prácticamente las mismas prestaciones que SLUP con borrado cada 128k ciclos (más del 95% del IPC alcanzado por un oráculo), pero con un 42% de las entradas inactivas, que para una tecnología de 160 nm. se traduce en un consumo estático neto del 46% respecto al SLUP convencional.

Por otra parte, también se ha observado que, para todos los tamaños analizados de SLUPd-4T, pueden encontrarse intervalos de decaimiento con los que alcanzar buenas prestaciones con tasas de actividad muy similares a las de SLUPd-6T. Sin embargo, la evaluación del consumo estático neto de energía para 160 nm. muestra que para los predictores pequeños, la energía necesaria para regenerar los contenidos de las celdas reduce e incluso llega a superar a la ahorrada por el decaimiento. En los predictores más grandes, las bajas tasas de actividad sí que permiten compensan el consumo extra. Por ejemplo, SLUPd-4T de 4k entradas decaído a 8192 ciclos alcanza más del 96% del IPC máximo con un 31% de entradas sin decaer, lo que permite un consumo estático del 44% respecto al SLUP convencional.

Como trabajo futuro, parece interesante evaluar el consumo de energía para tecnologías actuales y determinar si SLUPd-4T se convierte en la mejor opción de diseño.

6.6 Referencias

[AMV+06] J. Alastruey, T. Monreal, V. Viñals and M. Valero. “Speculative Early Register Release”. *Proc. of the 3rd conference on Computing frontiers* (CF 06), May 2006, pp 291-302.

[AMV+07] J. Alastruey, T. Monreal, V. Viñals and M. Valero. “Microarchitectural Support for Speculative Register Renaming”. *IEEE Int. Parallel & Distributed Processing Symp.* (IPDPS 07), March 2007.

[AMC+08] J. Alastruey, T. Monreal, F.J. Cazorla, V. Viñals y M. Valero. Selection of the Register File Size and the Resource Allocation Policy on SMT Processors. *Proceedings of the 20th International Symposium on Computer Architecture and High Performance Computing* (SBAC-PAD 08), Campo Grande, Brasil, Oct 29-Nov 1, 2008. IEEE Computer Society, pp. 63-70, ISBN 978-0-7695-3423-7.

[AMV+09] J. Alastruey, T. Monreal, M. Valero y V. Viñals. "Implementación de un Predictor de Último Uso con Decaimiento". *XX Jornadas de Paralelismo*, La Coruña (España). 17-18 Sept. 2009.

[BuAu97] D. Burger, and T.M. Austin, *The Simplescalar Tool Set v2.0*, Technical Report 1342, Computer Science Dept., University of Wisconsin-Madison, June 1997.

[ChEm98] G. Z. Chrysos, J.S. Emer. “Memory dependence prediction using store sets”. *Proc. International Symposium on Computer Architecture* (ISCA). 1998, pp. 142 - 153

[CFP+07] B. Curran, E. Fluhr, J. Paredes, L. Sigal, J. Friedrich, Y.-H. Chan, C. Hwang. “Power-constrained high-frequency circuits for the IBM POWER6 microprocessor”. *IBM J. Res. & Dev.* vol. 51 no. 6, Nov. 2007.

[FKM+02] K.Flautner, N.S. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy caches: simple techniques for reducing leakage power. *Proceedings of the 29th Annual International Symposium on Computer Architecture* (ISCA 02).

[HJD+02] Z. Hu, P. Juang, P. Diodato, S. Kaxiras, K. Skadron, M. Martonosi, and D.W. Clark. Managing leakage for transient data: decay and quasi-static 4T memory cells. *In Proceedings of the 2002 International Symposium on Low Power Electronics and Design* (ISLPED 02), pp.52-55. Monterey (USA), August 12-14, 2002.

[JSM+04] P.Juang, K. Skadron, M. Martonosi, Z. Hu, D.W. Clark, P.W. Diodato, and S. Kaxiras. Implementing branch-predictor decay using quasi-static memory cells. *ACM Trans. Archit. Code Optimization* (TACO 04), vol 1, n.2, pp. 180-219. Jun. 2004.

[KHM01] S.Kaxiras, Z. Hu, M. Martonosi. Cache decay: exploiting generational behavior to reduce cache leakage power. ISCA 2001, pp- 240-251.

[KuHi09] R. Kumar, G. Hinton A family of 45nm IA Processors. *International Solid-State Circuits Conference* (ISSCC 09), Feb. 8-12, 2009. San Francisco (USA).

[MDM+95] S.Mutoh, T. Douseki, Y. Matsuya, T. Aoki, S. Shigematsu, and J. Yamada. 1-V power supply high-speed digital circuit technology with multithreshold-voltage CMOS. *IEEE Journal of Solid-State Circuits*, 1995, vol. 30, n.8, pp. 847-854.

[PYF+00] M.Powell, S. Yang, B. Falsafi, K. Roy, T. N. Vijaykumar. Gated-Vdd: a circuit technique to reduce leakage in deep-submicron cache memories. *In Proceedings of the 2000 International Symposium on Low Power Electronics and Design (*ISLPED 00).

[RDK+00] S. Rixner, W. Dally, B. Khailany, P. Mattson, U. Kapasi and J.Owens, Register Organization for Media Processing, *HPCA* 2000.pp. 375-386.

[RTM+09] S. Rusu, S. Tam, H. Muljono, J. Stinson, D. Ayers, J. Chang, R. Varada, M. Ratta, S. Kottapalli. A 45nm 8-Core Enterprise Xeon® Processor. *International Solid-State Circuits Conference* (ISSCC 09), Feb. 8-12, 2009. San Francisco (USA)..

[SPH+02] T. Sherwood, E. Perelman, G. Hamerly and B. Calder. Automatically Characterizing Large Scale Program Behavior. ASPLOS 02, pp. 45-57.

[StRu03] J.Stinton et S. Rusu. A 1.5GHz Third Generation Itanium® 2 Processor. *In Proceedings of Design Automation Conference* (DAC 03), pp. 706-709. 2-6 June 2003.

[YER+99] A. Yoaz, M. Erez, R. Ronen, S. Jourdan. Speculation Techniques for Improving Load Related Instruction Scheduling. ISCA 1999: pp. 42-53.

CAPÍTULO 7

Selección del Tamaño del Banco de Registros y de la Política de Asignación de Recursos en Procesadores SMT

El impacto del tamaño del Banco de Registros Físico (Physical Register File, PRF) en las prestaciones de un procesador Simultaneous Multithreading (SMT) no ha sido detalladamente estudiado a pesar de que el PRF es un recurso compartido crítico. En este Capítulo analizamos el efecto del número de entradas del PRF en las prestaciones para un amplia selección de políticas de asignación de recursos (Icount, Stall, Flush, Flush++, Static, Dcra y Hill-climbing). En la evaluación se usan dos métricas: instrucciones por segundo (IPS) para cuantificar la productividad (throughput) y media armónica de IPCs relativos (harmonic mean of weighted IPCs, Hmean-wIPC) para comprobar la equidad (fairness) de la distribución de recursos. Hemos observado que la política de asignación de recursos y el tamaño del PRF deben considerarse juntos para poder obtener los mejores resultados en las métricas propuestas. Por ejemplo, para las cargas de trabajo analizadas (2 y 4 threads de SPEC CPU2000), la mejor gestión de PRFs con pocas entradas corresponde a Flush, mientras que para los PRFs con más entradas, Hill-climbing y Static llevan a los mejores valores de productividad y equidad, respectivamente. La segunda contribución de este trabajo es un procedimiento sencillo que, para una política de asignación de recursos dada, selecciona el tamaño de PRF que maximiza IPS y obtiene un valor de Hmean-wIPC cercano a su máximo. Según nuestros resultados, Hill-climbing con un PRF de 320 entradas obtiene los mejores resultados con cargas de 2 threads. Cuando se ejecutan 4 threads, Hill-Climbing gestionando un PRF de 384 entradas alcanza la mayor productividad mientras que Static obtiene el mejor balance entre productividad y equidad.

7.1 Introducción

Los procesadores *Simultaneous Multithreading* (SMT) extienden la ejecución superescalar permitiendo que sean lanzadas en el mismo ciclo instrucciones pertenecientes a distintos hilos de ejecución. En los procesadores SMT modernos, este modelo de ejecución implica la comparti-

ción de varios recursos entre los threads, por ejemplo, el PRF [KoMa03][KST04]. Por tanto, el PRF debe ser suficientemente grande para almacenar los estados consolidados y especulativos de todos los threads, y también debe suministrar cada ciclo todos los operandos requeridos por las instrucciones que se lanzan. En consecuencia, los procesadores SMT requieren PRFs con un elevado número de entradas y puertos, características que incrementan su consumo de energía, área y tiempo de acceso. Esto último puede afectar tiempo al tiempo de ciclo del procesador, limitando su frecuencia y reduciendo sus prestaciones.

Por todo lo anterior, al equipo de diseño de un procesador se le plantea un compromiso en cuanto al dimensionado del PRF. Por una parte, grandes PRFs pueden reducir el número de detenciones en renombre por falta de registros físicos, mejorando las prestaciones medidas en instrucciones por ciclo (IPC). Por otra parte, los PRFs más ajustados puede permitir mayores frecuencias del procesador, lo que resulta en más prestaciones medidas en instrucciones por segundo (IPS).

Otra decisión importante en el diseño de un procesador SMT es la política de asignación de recursos, la cual determina el modo en que los recursos compartidos del procesador se distribuyen entre los threads que se ejecutan. Varias políticas propuestas reivindican mejoras sobre las estrategias previas, pero todas sus evaluaciones se han realizado para un tamaño del PRF fijo [TEE+96][TuBr01][CFR+03] [RaRe03][CRV+04b][ChYe06][EyEc07][SPL08].

Las principales contribuciones de este Capítulo son:

- Un análisis del efecto combinado en las prestaciones del tamaño del PRF y de la política de asignación de recursos. Hemos considerado un amplio rango de tamaños del PRF y un variado conjunto de políticas de asignación de recursos (*Icount* [TEE+96], *Stall*, *Flush* [TuBr01], *Flush++* [CFR+03], *Static* [KoMa03][RaRe03], *Dcra* [CRV+04b], y *Hill-climbing* [ChYe06]). Hemos observado que las prestaciones relativas entre políticas dependen estrechamente del tamaño del PRF. Por ejemplo, *Flush* es la mejor política para gestionar PRFs pequeños pero también la peor con PRFs grandes.

- Un procedimiento sencillo que, dada una política de asignación de recursos, selecciona un tamaño de PRF de acuerdo con un compromiso de prestaciones entre productividad y equidad. Para esto, hemos utilizado dos métricas, instrucciones por segundo (IPS) para medir prestaciones y a la vez considerar el efecto de acceder a PRFs de distintos tamaños, y la media armónica de IPCs relativos (Hmean-wIPC) para cuantificar la equidad en la distribución de recursos entre threads [LGF01]. Hemos aplicado este procedimiento a varias políticas de asignación de recursos y hemos obervado que, en todos los casos, existe un tamaño de PRF que maximiza IPS y a la vez alcanza valores de Hmean-wIPC cercanos al máximo. Finalmente, hemos comparado la productividad, equidad y la eficiencia energética para cada pareja tamaño óptimo de PRF y política de asignación de recursos. Esto permite seleccionar el mejor punto de diseño de acuerdo con un objetivo particular.

El resto del Capítulo se estructura de la siguiente manera: en la Sección 7.2 se describen y clasifican varias políticas de asignación de recursos. La Sección 7.3 detalla el entorno experimental y la metodología aplicada. La Sección 7.4 discute los resultados y la Sección 7.5 resume las conclusiones del Capítulo.

7.2 Políticas de Asignación de Recursos

Varios recursos de los procesadores SMT, como por ejemplo, la ventana de lanzamiento (*Issue Queue*) o el PRF, suelen ser compartidos por los distintos threads en ejecución. La distribución de estos recursos puede ser indirectamente controlada mediante la política de búsqueda de instrucciones. De esta forma, la asignación de recursos es dirigida por la distribución del ancho de banda entre threads.

La Figura 7.1 muestra distintas políticas de búsqueda de instrucciones. Excepto *Round Robin*, que reparte equitativamente la unidad de búsqueda entre todos los threads, todos los esquemas dan prioridad a los threads siguiendo diferentes heurísticas como el número de saltos pendientes de resolver (*Brcount*), el número de fallos de cache de datos pendientes (*Misscount*), el número de fallos pendientes de L1D (*DWarn*), el número de instrucciones en las etapas previas al lanzamiento (*Icount*) o la edad de sus instrucciones más viejas (*IQPOSN*) [TEE+96] [CRV+04a]. De entre todas estas políticas, se considera a *Icount* como la que logra las mejores prestaciones, por eso ha sido adoptada como base de casi todas las propuestas académicas de políticas de asignación de recursos.

Como se ha comentado antes, la política de búsqueda *Icount* otorga mayor prioridad a los threads con menos instrucciones en las etapas del *front-end* [TEE+96]. Obtiene buenos resultados con threads de alto ILP, sin embargo, cuando un thread sufre muchas lecturas de memoria de larga latencia (lecturas que fallan en el nivel más grande de cache o en el TLB), *Icount* sigue asignando recursos a este thread detenido. En consecuencia, puede acaparar muchos recursos, atascando el segmentado y reduciendo las prestaciones del procesador [KST04].

Existen propuestas más complejas que tratan de superar las desventajas de las política de búsqueda de instrucciones. Algunos de estos esquemas reaccionan cuando un thread sufre una operación de larga latencia, mientras que otros tratan de anticipar su reacción mediante la predicción de ciertos eventos (por ejemplo, prediciendo fallos de L2). El objetivo en ambos casos es distribuir los recursos entre los threads para maximizar prestaciones. Llamamos políticas de asignación de recursos a esas políticas de búsqueda mejoradas. La Figura 7.1 muestra una clasificación varias propuestas surgidas del ámbito académico y de la industria. Aunque todas las políticas de asignación de recursos puede basarse en cualquier política de búsqueda de instrucciones, todas menos una (*FPG*) se asientan sobre *Icount* debido a sus mejores prestaciones.

Un primer grupo de políticas (G1) lidia con la escasez de recursos producida por los threads limitados por memoria acometiendo determinadas acciones cuando un load de latencia larga es detectado. *Stall* es una mejora de *Icount* que impide la búsqueda de instrucciones por parte de threads que tengan un load de latencia larga pendiente [TuBr01]. Sin embargo, cuando un load de latencia larga es detectado, puede ser demasiado tarde para evitar que su thread acumule recursos. Y al contrario, un thread puede ser detenido a pesar de que existan muchos recursos disponibles, provocando la infrautilización de recursos y quizá evitando la explotación de paralelismo a nivel de memoria (*memory level parallelism*, MLP).

Flush es una extensión de *Stall* que anula un thread que sufre un load de latencia larga. Todas las instrucciones más jóvenes que el load implicado se anulan para liberar los recursos que se les había asignado [TuBr01]. Sin embargo, estos recursos liberados puede que no sean requeri-

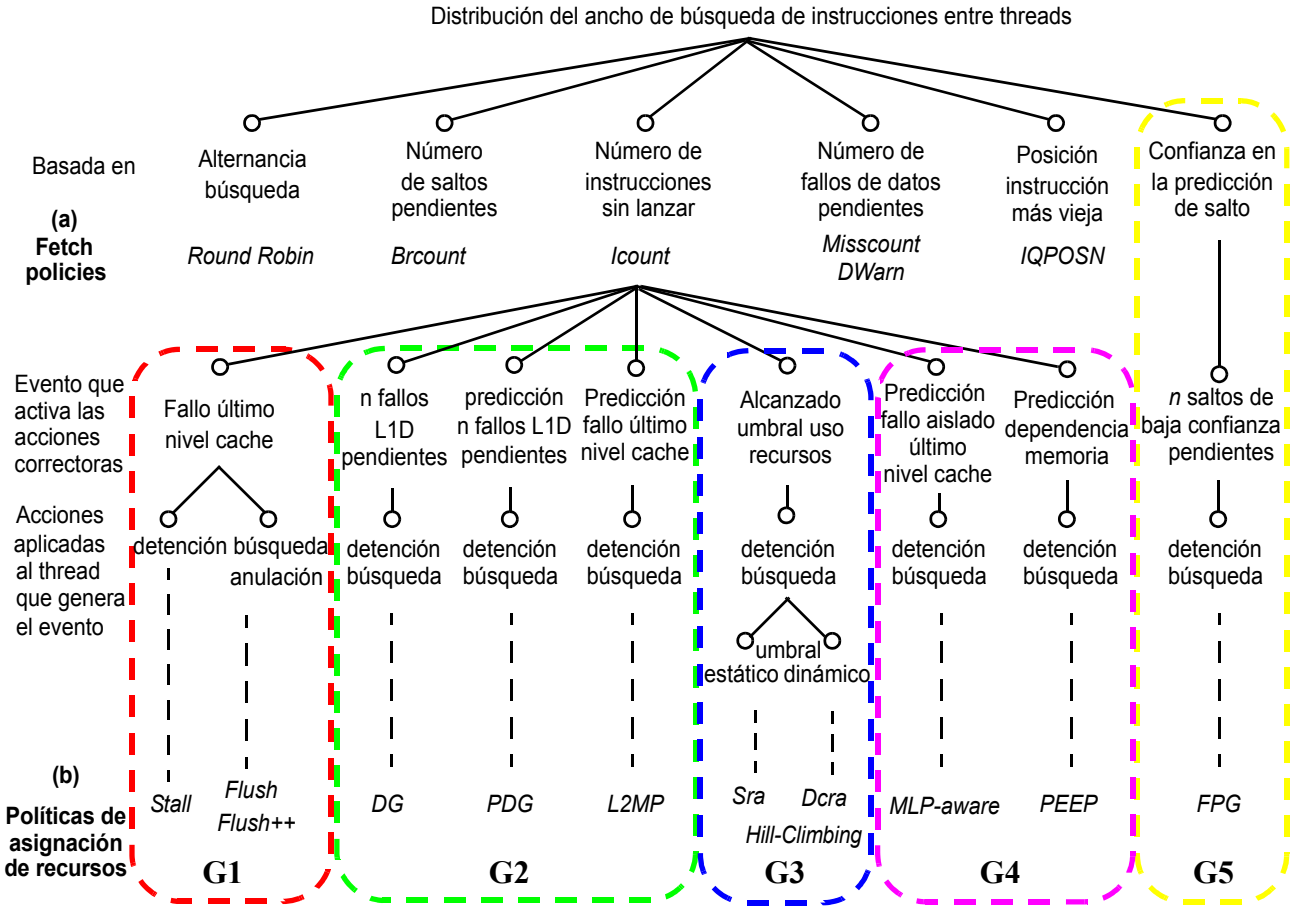


Figura 7.1 Clasificación de políticas de búsqueda de instrucciones y de asignación de recursos.

dos por los otros threads. Además, el consumo de energía aumenta como consecuencia de la reejecución de las instrucciones anuladas.

Flush++ se basa en el hecho de que *Stall* funciona mejor que *Flush* para cargas de trabajo que no requieres muchos recursos (pocos loads de latencia larga) y de que *Flush* funciona mejor que *Stall* para cargas de trabajo que requieren muchos recursos (muchos loads de latencia larga) [CFR+03]. Por tanto, *Flush++* trata de combinar lo mejor de *Flush* y *Stall* para incrementar las prestaciones. El número de threads detenidos se usa para conmutar entre *Stall* y *Flush*. Un thread que sufre un load de latencia larga sólo se anula si hay otros threads sin detener.

Un segundo grupo de propuestas (G2) trata de anticipar los loads de latencia larga mediante distintas aproximaciones. *Data Miss Gating* (*DG*) detiene los threads con frecuentes fallos del primer nivel de cache de datos (L1D), en concreto los que acumulan *n* fallos pendientes [ElAl03]. De este modo, los recursos pueden ser liberados poco después de ser asignados, favoreciendo su reutilización. De una manera similar, *Predictive Data Miss Gating* (*PDG*) detiene un thread con más de *n* fallos pendientes de la cache de datos, ya sea detectados o predichos [ElAl03]. Puesto que la predicción de un fallo se realiza cuando un load se busca, los threads pueden ser detenidos antes que con *DG*. El objetivo principal de estas políticas es reducir la

ocupación de la ventana de lanzamiento (*issue queue*), no mejorar prestaciones. Por esta razón, pueden degradar las prestaciones de threads limitados por memoria. Finalmente, *L2MP* usa un predictor de fallos de loads para anticipar fallos del nivel más grande de la jerarquía de caches [YER+99]. Un thread con un load predicho de larga latencia se detiene hasta que se resuelva el acceso de dicho load a memoria [CFR+03].

Un tercer grupo de políticas (G3) controla explícitamente la distribución de recursos fijando un umbral máximo de uso de recursos por parte de un thread. Por consiguiente, se impide que un thread busque instrucciones cuando ha consumido todos los recursos de su cuota. Estas estrategias pueden ser clasificadas como estáticas o dinámicas dependiendo de si el umbral es fijo o varía durante la ejecución.

La Asignación Estática de Recursos (*Static Resource Allocation, Sra*) asigna una cantidad estática de recursos a cada uno de los *n* threads que se ejecutan. Estas cuotas pueden variar entre cantidades proporcionales que no permiten la compartición de recursos (1/n, particionado) y todos los recursos (compartición total) [RaRe03]. Por ejemplo, el Intel Pentium 4 particiona las colas de búsqueda, micro-ops y jubilación, usa un esquema con umbral máximo para las colas de lanzamiento (IQs) y comparte totalmente sus caches [KoMa03]. El microprocesador IBM Power5 usa una política con umbrales para la Tabla de Finalización Global (*Global Completion Table*) y la cola de loads que han fallado en cache (*Load Miss Queue*). Esta lógica de reparto de recursos también detecta cuando un thread alcanza un límite de fallos de L2 y TLB [KST04].

Dynamically Controlled Resource Allocation (*Dcra*) distribuye dinámicamente los recursos del procesador entre threads dependiendo de su interacción con la cache: a los threads con fallos frecuentes de L1D (los que probablemente sufran más fallos de larga latencia) se les asigna una cuota limitada de recursos para prevenir que los acaparen en caso de un fallo de larga latencia [CRV+04b]. Este umbral es mayor que la parte proporcional de manera que a estos threads se les permite explotar paralelismo más allá de los loads de latencia larga. Además, *Dcra* calcula las cuotas para cada thread basándose en su necesidad de recursos, permitiendo que un thread tome prestados recursos que otros threads no está usando.

Hill-climbing evalúa distintas distribuciones de recursos tratando de encontrar el reparto que optimiza las prestaciones [ChYe06]. Usa un algoritmo de aprendizaje que dinámicamente selecciona la distribución que obtuvo las mejores prestaciones en tiempo de ejecución durante el último período de prueba. La monitorización de prestaciones se lleva a cabo continuamente, por esto *Hill-climbing* es capaz de reaccionar ante las necesidades de recursos cambiantes en el tiempo por parte de los threads.

Dos políticas recientes explotan predicciones relacionadas con memoria (G4). *Memory-Level Parallelism aware* (*MLP-aware*) trata de solapar la ejecución de loads de larga latencia independientes, ocultando de este modo las penalizaciones de acceso a memoria [EyEc07]. Su implementación se basa en dos mecanismos: uno detecta o predice loads de latencia larga y el otro trata de determinar si dicho fallo de lectura es aislado o no (predicción MLP). En el último caso, el predictor debe también determinar el número adicional de instrucciones que deberían ser buscadas para solapar múltiples loads de latencia larga. Un thread que sufre un load de latencia larga es detenido, e incluso puede que se anulen las instrucciones posteriores a dicho load. Comparado con *Icount*, esta política es mejor cuando ejecuta cargas de trabajo intensivas en

MLP, pero degrada las prestaciones cuando ejecuta cargas de trabajo de alto paralelismo (ilp), en algunos casos, con ralentizaciones de más del 20%. Además, las predicciones de loads de larga latencia y de MLP son relativamente complicadas. En una aproximación diferente, una propuesta posterior explota la alta predecibilidad de las dependencias de memoria. *Proactive Exclusion-Early Parole* (*PEEP*) detiene un thread cuando se predice que un load buscado tiene una dependencia con un store previo [SPL08]. Otro mecanismo de predicción trata de que el thread detallado reanude su ejecución tan pronto como la dependencia haya sido resuelta.

Finalmente (G5), *Fetch Prioritization and Gate* (*FPG*) fija una prioridad a un thread según su número de saltos de baja confianza pendientes [LFM+01]. Además, impide que un thread busque instrucciones cuando alcanza un umbral de saltos de baja confianza pendientes. Esta es la única política descrita que no está basada sobre *Icount*.

Ninguna de las referencias previas realiza un análisis del tamaño del PRF. En todos los casos utilizan un tamaño fijo del PRF. En este Capítulo se evalúan las prestaciones de varias de estas políticas para un rango amplio de tamaños del PRFs con dos objetivos en mente:

- Analizar el efecto combinado en las prestaciones del tamaño del PRF y de la política de asignación de recursos
- Encontrar el tamaño del PRF que optimiza las prestaciones para una política de asignación de recursos dada.

Por una parte, este trabajo advierte al diseñador de procesadores que la política de asignación de recursos y el tamaño del PRF deberían considerarse juntos para maximizar prestaciones. Por ejemplo, la política que mejor gestiona un determinado tamaño de PRF podría ser la peor con otro tamaño de PRF. Por otra parte, proponemos un procedimiento sencillo que, para una política de asignación dada, selecciona el tamaño del PRF que obtiene el mejor compromiso entre productividad y equidad. Puesto que lo hemos aplicado a una amplia selección de políticas de asignación de registros, puede también ayudar al diseñador de un procesador SMT a encontrar la mejor política de asignación para un tamaño de PRF dado.

No es nuestra intención determinar cuál es la política de asignación de recursos que obtiene las mejores prestaciones para cada tamaño de PRF. Varias políticas tienen opciones de diseño que afectan significativamente a sus prestaciones (por ejemplo, el dimensionado de un predictor y su entrenamiento). Por este motivo, una comparación precisa entre las prestaciones requeriría el ajuste fino de muchos parámetros para muchas políticas. Además, requeriría una evaluación exhaustiva con varias configuraciones de procesador y distintas cargas de trabajo. Ese objetivo se encuentra fuera del alcance de este trabajo. Por tanto, nos hemos centrado en las políticas clásicas (G1) y en las basadas en reparto con umbrales (G3) para dar evidencia empírica de nuestras contribuciones.

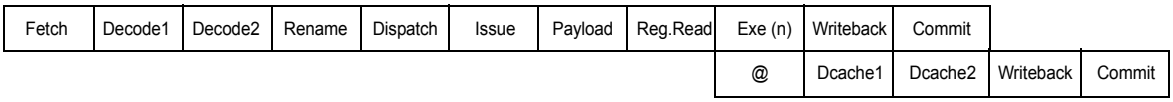


Figura 7.2 Segmentación de la microarquitectura modelada.

Tabla 7.1 Parámetros del procesador SMT simulado.

Parámetro	Valor	Parámetro	Valor
Número de threads	2 y 4	L1 Icache	64 KB, asociatividad 4, 4 bancos, líneas de 64 bytes, latencia: 1 ciclo
Ancho de búsqueda	8 instrucciones (máx: 2 threads)	L1 Dcache	64 KB, 8-way, 4 bancos, líneas de 64 bytes, latencia: 2 ciclos
Ancho de decodificación y jubilación	8 instrucciones	L2 Ucache	2048 KB, 8-way, 8 bancos, líneas de 64 bytes, latencia: 15 ciclos
Ancho lanzam.	6 int + 3 fp+ 4 mem	Memoria principal	Tiempo de acceso: 300 ciclos
Reorder Buffer (ROB)	512 entradas, compartido, jubilación por thread	TLB	ITLB: 64 entradas, DTLB: 128 entradas, penalización por fallo: 360 ciclos
Ventanas de Lanzamiento (IQ)	int: 80 entradas, fp: 80 entradas, mem: 80 entradas	Unidades Funcionales (*latencia*)	6 int (*simple: 1, mult/div: 4*), 4 load/store (*2*), 3 fp (*simple/cmp/mult: 4,div: 17, sqrt: 19*)
Predicción de salto	gshare: 16K entradas BTB: 256, 4-way, RAS: 256	PRF	int: 80-640 entradas, 16r+10w puertos, fp: 80-640 entradas, 6r+7w puertos tiempo de acceso: 1 ciclo

7.3 Experimentación

7.3.1 Modelo de procesador

Se ha utilizado un simulador temporal detallado ciclo a ciclo consistente en un *front-end* dirigido por traza y un *back-end* basado en una versión mejorada de smtsim [TEL95]. Un diccionario de bloques básicos que contiene todas las instrucciones estáticas permite la ejecución por caminos de control incorrectos.

Hemos modelado un procesador SMT con un alto ancho de lanzamiento y un segmentado de 11 etapas (ver Figura 7.2). Los detalles de la microarquitectura simulada se muestran en la Tabla 7.1. Las colas de lanzamiento, el ancho de lanzamiento y el ROB se han sobredimensionado razonablemente para adaptarse a los tamaños más grandes de PRF. De este modo, el efecto de variar el tamaño del PRF puede analizarse mejor. Una red de cortocircuitos suministra operandos a las instrucciones consumidoras que se han lanzado encadenadas a sus productoras, evitando así lecturas del PRF. La cache L1D está compuesta por cuatro bancos de 16 KB entrelazados a nivel de línea, y cuyos conflictos son adecuadamente modelados y penalizados.

Tabla 7.2 Parámetros principales de las políticas de asignación de recursos implementadas.

Política	Parámetro	Valor
Stall, Flush, Flush++	Retardo de L2 para suponer un fallo	20 ciclos
Sra	Límites por thread [RaRe03]	2 threads: 50%, 4 threads: 30%
Hill-climbing	Duración de época	64k ciclos
	Delta	#registros renombre / 64

7.3.2 Implementación de las políticas de asignación de recursos evaluadas

Los parámetros principales usados en la implementación de las políticas de asignación de recursos consideradas se muestran en la Tabla 7.2.

Al respecto del grupo G1, hemos implementado *Stall*, *Flush* y *Flush++* con una mejora denominada *Continue the Oldest Thread* (*COT*) [CFR+03]. Cuando todos los threads están detenidos por loads de latencia larga, el thread con el load más antiguo que ha fallado en cache reanuda la búsqueda de instrucciones. La idea es avanzar la ejecución del thread con más posibilidades de ser el primero en resolver su fallo de cache.

En lo que a las políticas que controlan explícitamente la asignación de recursos se refiere (G3), todas se asientan sobre *Icount*. Nuestra implementación de *Dcra* distribuye las entradas de la ventanas de lanzamiento (entera, coma flotante y memoria) y de los bancos de registros físicos enteros y de coma flotante, mientras que nuestras implementaciones de *Sra* y *Hill-climbing* distribuyen las entradas de la ventana de lanzamiento entera, del PRFint y del ROB. Los límites por thread de *Sra* se corresponden con particionado puro (sin compartición de recursos) para 2 threads y una compartición muy limitada para 4 threads. Según [RaRe03], estos umbrales proporcionan buenas prestaciones y favorecen la equidad (*fairness*). Hemos explorado otros límites de recursos y no hemos encontrado mejores números de productividad y equidad. Hemos elegido el IPC como métrica que controla la realimentación de prestaciones SMT en el algoritmo *Hill-climbing* (opción denominada HILL-IPC en [ChYe06]) porque permite la implementación con mejor compromiso entre complejidad y prestaciones. El resto de métricas consideradas en [ChYe06] necesitan evaluar el IPC de cada thread por separado, por eso tienen que ser periódicamente ejecutados en solitario durante ciertos períodos de tiempo, mientras que la medida del IPC agregado puede realizarse sin deshabilitar la ejecución SMT. Con el objetivo de adaptar su valor a un rango amplio de tamaños del PRF, el parámetro Delta (valor constante value de 4 en [ChYe06]), ha sido reemplazado por un valor proporcional al número de registros de renombre. Finalmente, la mínima cantidad de recursos compartidos asignados a cada thread ha sido fijado a Delta.

7.3.3 Cargas de trabajo

Se han usado todos los programas de SPEC CPU2000 excepto *fma3d* que no ha podido ser ejecutado en nuestro entorno de trabajo. Se han utilizado trazas de 300 millones de instrucciones siguiendo las ideas presentadas en [SPH+02]. Los programas han sido clasificados en alto ILP

Tabla 7.3 Cargas de trabajo *multithreaded*.

		Tipo de carga de trabajo		
		alto ILP	**bajo ILP**	**mezcla**
2 threads	**int**	bzip2-eon, gzip-gcc	vpr-mcf, vortex-twolf	perlbmk-vortex, gcc-gap
	int-fp	perlbmk-apsi, crafty-galgel	gap-swim, parser-mgrid	crafty-art, gzip-mgrid
		bzip2-mesa, eon-sixtrack	vpr-lucas, mcf-equake	twolf-galgel, parser-ammp
	fp	mesa-sixtrack, ammp-wupwise	lucas-equake, applu-art	apsi-applu, facerec-swim
4 threads	**int**	bzip2-eon-gzip-gcc	vpr-mcf-vortex-twolf	perlbmk-vortex-gcc-gap
	int-fp	perlbmk-apsi-crafty-galgel	gap-swim-parser-mgrid	crafty-art-gzip-mgrid
		bzip2-mesa-eon-sixtrack	vpr-lucas-mcf-equake	twolf-galgel-parser-ammp
	fp	mesa-sixtrack-ammp-wupwise	lucas-equake-applu-art	apsi-applu-facerec-swim

y bajo ILP según el valor de sus IPCs cuando son ejecutados en solitario. De este modo, se pueden distinguir tres tipos de cargas de trabajo: alto ILP (todos los threads son de alto ILP), *low-ILP* (todos los threads son de bajo ILP) y mezcla (compuesta por threads de alto y bajo ILP). Para cada tipo de carga de trabajo, se han seleccionado ocho combinaciones de dos programas y cuatro combinaciones de cuatro programas (ver Tabla 7.3). Se ha intentando que todos los benchmarks aparezcan el mismo número de veces y que el número de combinaciones de programas enteros y de coma flotante esté equilibrado. Las simulaciones terminan cuando todos los threads han finalizado su ejecución al menos una vez (metodología *last*, [VCP+07]). Cuando un thread finaliza su ejecución antes de que termine la simulación, es reejecutado. De este modo, la carga de trabajo siempre está compuesta por el número de threads especificado.

7.3.4 Métricas

Se han considerado dos métricas para evaluar las prestaciones. IPS considera el efecto de los diferentes tiempos de acceso al PRF en las prestaciones. Hmean-wIPC es muy usada para caracterizar las prestaciones en procesadores SMT [LGF01][CFR+03][CRV+04a][CRV+04b] [ChYe06][EyEc07][SPL08].

- IPS: cuantifica las prestaciones (*throughput*) obtenidas por una microarquitectura SMT. Es el cociente entre IPC y tiempo de ciclo del procesador (T_c). El IPC se obtiene como la suma del IPC alcanzado por cada thread en ejecución SMT (IPC_i) dividido por el número de threads (N). Asumimos que el tiempo de ciclo del procesador está limitado por el tiempo de acceso al PRF, calculado para cada tamaño de PRF según el modelo de Rixner para 180 nm [RDK+00].

$$IPS = \frac{IPC}{T_c} = \frac{\frac{\sum_i IPC_i}{N}}{T_c}$$

- Hmean-wIPC: considera el IPC relativo de cada thread ($wIPC_i$), es decir, el cociente entre su IPC en ejecución SMT y su IPC en solitario (*single-threaded IPC*, $IPCst_i$), y los promedia con la media armónica. Combina prestaciones y equidad en la distribución de recursos entre threads [LGF01], por lo que penaliza a las políticas que obtienen altas prestaciones acelerando threads con alto ILP a costa de ralentizar threads de bajo ILP. Esta métrica no depende del tiempo de ciclo del procesador y puede también ser expresada como la media armónica de las instrucciones relativas por segundo (wIPS) de cada thread:

$$Hmean_wIPC = \frac{N}{\sum_i (wIPC_i)^{-1}} = \frac{N}{\sum_i \frac{IPCst_i}{IPC_i}} =$$

$$= \frac{N}{\sum_i \frac{IPSst_i}{IPS_i}} = \frac{N}{\sum_i (wIPS_i)^{-1}} = Hmean_wIPS$$

7.4 Resultados

La Figura 7.3 muestra IPS y Hmean-wIPC para todas las cargas de trabajo de 2 threads (columna de la izquierda) y para todas las cargas de trabajo de 4 threads (columna de la derecha). Para que las gráficas sean más claras, no se han representado los datos de *Flush++* (muy similares a los de *Flush*).

Para cada métrica, primero describimos su comportamiento cuando el tamaño del PRF varía (igual tamaño de PRFint y PRFfp). A continuación, comparamos las prestaciones obtenidas por las políticas de asignación de recursos consideradas. Finalmente, comparamos las gráficas correspondientes a las cargas de trabajo de 2 y 4 threads.

7.4.1 IPS

Para evaluar esta métrica, hemos considerado que el acceso al PRF está en el camino crítico, por lo que el tiempo de acceso al PRF limita la frecuencia del procesador.

Las gráficas superiores de la Figura 7.3 muestran que los PRFs más pequeños obtienen bajas prestaciones. El efecto negativo del elevado número de detenciones en renombre domina sobre las ventajas de las altas frecuencias que permitirían los reducidos tiempo de acceso de estos PRFs. Por otra parte, las altas latencias de los PRFs más grandes limitan la frecuencia del procesador, afectando negativamente a las prestaciones. Un punto de diseño interesante es el tamaño de PRF que maximiza IPS. Este número de entradas varía con la política de asignación de recursos: para cargas de trabajo de 2 threads, el máximo IPS se alcanza con entre 224 y 320 registros, mientras que para cargas de trabajo de 4 threads, se alcanza con entre 320 y 448 registros. La presión sobre los recursos es mayor en el último caso, por eso todas las políticas explotan PRFs más grandes.

Si comparamos las políticas de asignación de recursos, podemos observar que para cargas de trabajo de 2 threads, *Flush* obtiene las mejores prestaciones con PRFs pequeños (por debajo de

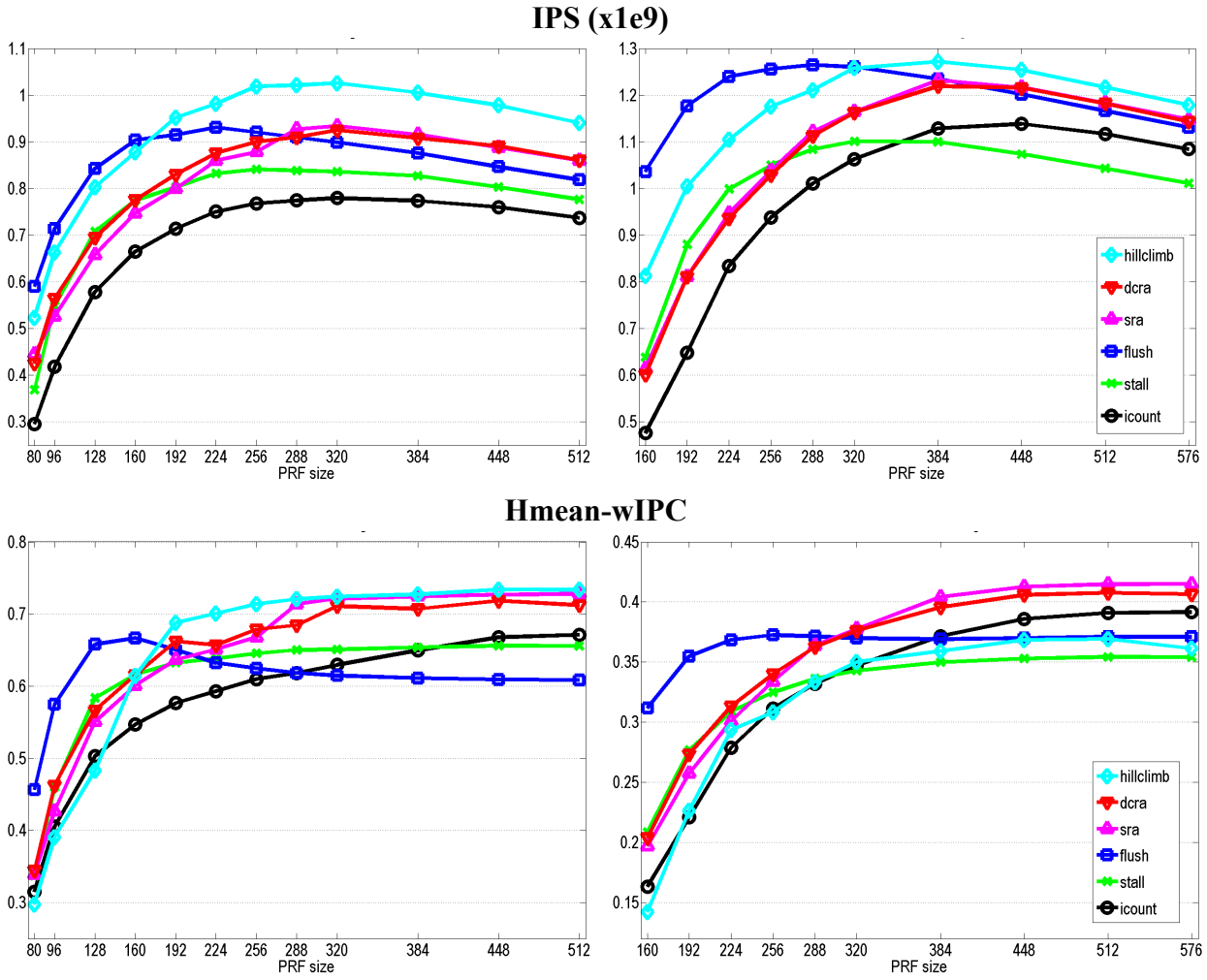


Figura 7.3 IPS (arriba) y Hmean-wIPC (abajo) vs. tamaño de PRF para las políticas *Icount*, *Stall*, *Flush*, *Sra*, *Dcra* y *Hill-climbing*. Las dos gráficas de la izquierda y las dos de la derecha se corresponden a las cargas de trabajo de 2 y 4 threads, respectivamente. Notar las diferentes escalas de tanto los ejes x e y en las figuras de 2 y 4 threads.

160 registros). Esto es debido a que evita las detenciones en renombre anulando instrucciones de threads detenidos para que sus registros asignados puedan ser utilizados por otros threads activos. Para PRFs más grandes, *Hill-climbing* supera al resto de políticas: su algoritmo adaptativo de asignación de recursos es capaz de encontrar la mejor distribución de los mismos. Las anulaciones de instrucciones por parte de *Flush* son menos efectivas debido que existen menos detenciones por falta de registros. *Icount* (la política más sencilla) es la peor opción para todos los tamaños.

Cuando se ejecutan cargas de trabajo de 4 threads, los PRFs por debajo de 320 registros alcanzan el mejor IPS con *Flush.* La presión sobre los recursos es mayor que con 2 threads, por eso está política ahora es más efectiva hasta con PRFs de tamaño bastante mayor. Más allá de esa cantidad de registros, *Hill-climbing* obtiene los mejores resultados, pero las aceleraciones res-

pecto al resto de políticas son menores que con cargas de trabajo de 2 threads. *Icount* es la peor opción para PRFs pequeños, pero es la política capaz de mejorar sus prestaciones hasta los tamaños más grandes de PRF (448 registros), superando a *Stall* por encima de 320 registers y obteniendo valores de IPS similares al del resto de políticas.

Es interesante señalar que *Dcra* y *Sra* se comportan de manera muy parecida tanto para cargas de trabajo de 2 threads como de 4 threads y para todos los tamaños del PRF, a pesar de fijar umbrales de uso de recursos de manera diferente (dinámica y estática).

Finalmente, comparamos las productividades (throughput) obtenidas por una política determinada con cargas de trabajo de 2 y 4 threads. Si estamos interesados en el punto de máximo IPS, en todos los casos es alcanzado por cargas de trabajo de 4 threads, pero a cambio de PRFs más grandes. Si comparamos la productividad obtenida por un PRF de tamaño fijo con 2 y 4 threads, puede observarse que son necesarios al menos 224 registros para conseguir mejoras por ejecutar cargas de trabajo con 4 threads en lugar de con 2. Con PRFs más pequeños, *Flush* es la única política que explota paralelismo a nivel de thread.

7.4.2 Hmean-wIPC

Esta métrica es baja con pequeños PRFs e incrementa hasta saturar conforme aumenta el tamaño de PRF para todas las políticas excepto para *Flush*, que alcanza su máximo valor con muchos menos registros (160 y 256 registers para cargas de trabajo de 2 y 4 threads).

Cuando se ejecutan cargas de trabajo de 2 threads, los PRFs ajustados (hasta 160 registros) son mejor gestionados con *Flush*. Al igual que ocurre con el IPS, *Flush* es capaz de reducir las detenciones de renombre mediante la liberación de los recursos asignados a threads. Los PRFs de tamaño medio (hasta 288 registros) obtienen el mejor Hmean-wIPC con *Hill-climbing* y los PRFs más grandes alcanzan los mejores valores de equidad con políticas del grupo G3, con *Hill-climbing* superando ligeramente a *Sra* y *Dcra*.

Para cargas de trabajo de 4 threads, *Flush* es de nuevo la mejor política para PRFs de tamaño pequeño y medio (hasta 288 registros), mientras que *Dcra* y *Sra* son los mejores esquemas con PRFs grandes. En este caso, *Hill Climbing* puede verse afectado por el uso del IPC como métrica de prestaciones en su algoritmo de aprendizaje. Si el objetivo es maximizar la equidad, Hmean-wIPC debería ser la métrica utilizada en su algoritmo de distribución de registros (esquema denominado HILL-HWIPC en [ChYe06]).

Finalmente, los valores de Hmean-wIPC son mucho mayores para cargas de trabajo de 2 threads. Proporciona hasta un 100% de mejora relativa a las cargas de 4 threads en el caso de *Hill-climbing*. Si un diseño requiere un cierto nivel de equidad, debe tenerse en cuenta que, ejecutando 2 threads, cualquiera de los tamaños evaluados de PRF obtienen valores de Hmean-wIPC que son inalcanzables cuando se ejecutan 4 threads.

7.4.3 Selección del tamaño del PRF y de la política de asignación de recursos

En las subsecciones previas hemos observado que el tamaño del PRF y la política de asignación de recursos no deben seleccionarse de forma separada. Ahora, nuestro objetivo es seleccionar

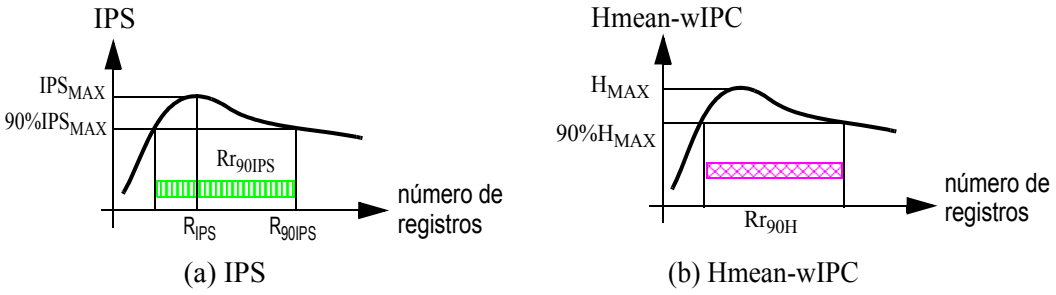


Figura 7.4 Rangos Rr_{90IPS}, Rr_{90H} y parámetros R_{IPS} y R_{90IPS} (a, b).

el tamaño de PRF y la política de asignación de recursos que maximizan las prestaciones de un procesador SMT en términos de productividad y equidad. El procedimiento de selección se lleva a cabo en dos pasos. Primero, para cada política de asignación de recursos, se busca el tamaño de PRF que obtiene el mejor compromiso de las dos métricas consideradas en este Capítulo: IPS y Hmean-wIPC. Este análisis simplifica la comparación entre productividad y equidad para distintas políticas de asignación de recursos. El segundo paso consiste en comparar los puntos seleccionados en el primer paso usando métricas de prestaciones y eficiencia energética.

De acuerdo a nuestros resultados, Hmean-wIPC incrementa con el número de registros, saturando con tamaños próximos a los que se obtiene el pico de IPS. Tales sensibilidades al tamaño de PRF sugieren que el IPS puede maximizarse a la vez que se obtienen valores cercanos al máximo de Hmean-wIPC.

De este modo, para una política de asignación de recursos dada, nuestro procedimiento considera los tamaños del PRF que obtienen al menos el 90% de los valores máximos de ambas métricas (ver Figura 7.4):

- Rr_{90IPS}: rango de registros necesario para obtener el 90% del valor máximo de IPS. También definimos R_{90IPS} como el valor máximo en dicho rango y R_{IPS} como el número de registros necesario para alcanzar el máximo IPS.

- Rr_{90H}: rango de registros necesario para obtener el 90% del máximo Hmean-wIPC.

La Tabla 7.4 recoge estos rangos y R_{IPS} para cada política de asignación de recursos y para cargas de trabajo de 2 y 4 threads. Puede observarse que en todos los casos, los dos rangos se solapan y que R_{IPS} cae en esa intersección.

En resumen, para cada política de asignación de recursos, el procedimiento busca un tamaño de PRF que obtiene simultáneamente al menos el 90% del máximo IPS y Hmean-wIPC. Para todas las políticas analizadas, hemos encontrado varios tamaños de PRF que cumplen estos requerimientos. Entre estos puntos de diseño y con las prestaciones como objetivo principal, hemos seleccionado el tamaño de PRF que maximiza IPS. Estos tamaños están resaltados en la Tabla 7.4 con el fondo sombreado. El procedimiento puede ser fácilmente modificado para cumplir otros objetivos de diseño. Por ejemplo, si el coste hardware fuera el objetivo principal, podríamos seleccionar el PRF de menor tamaño que queda en la intersección de los rangos Rr_{90IPS} y Rr_{90H} .

Tabla 7.4 Para cada política de asignación de recursos, rangos de registros que obtienen el 90% del máximo IPS (Rr_{90IPS}), el 90% del máximo Hmean-wIPC (Rr_{90H}), y cantidad de registros que obtiene el máximo IPS (R_{IPS}).

	2-threads			4-threads		
	Rr_{90IPS}	Rr_{90H}	R_{IPS}	Rr_{90IPS}	Rr_{90H}	R_{IPS}
Hill-climbing	192-512	192+	320	256+	288+	384
Dcra	224-512	192+	320	288+	320+	384
Sra	224-512	256+	320	288+	320+	384
Flush	128-448	128+	224	192-576	192+	288
Stall	160-512	160+	256	224-576	256+	320
Icount	192+	256+	320	320+	384+	448

Tabla 7.5 Para cada política de asignación de recursos, tamaños de PRF (R) seleccionados por el procedimiento descrito y valores de prestaciones para dichos tamaños de PRF: IPS (miles de millones), Hmean-wIPC y energía dinámica por instrucción (EpI, medida en nJ) consumida por el PRF.

	2 threads				4 threads			
	R	IPS	Hmean	EpI	R	IPS	Hmean	EpI
Hill-climbing	320	1.03	0.72	2.5	384	1.27	0.36	2.7
Dcra	320	0.92	0.71	2.5	384	1.22	0.40	2.8
Sra	320	0.93	0.72	2.5	384	1.23	0.40	2.8
Flush	224	0.93	0.63	2.2	288	1.27	0.37	2.7
Stall	256	0.84	0.65	2.1	320	1.10	0.34	2.4
Icount	320	0.78	0.63	2.5	448	1.14	0.39	3.3

Finalmente, para estas parejas de tamaño de PRF y política de asignación de recursos seleccionadas, la Tabla 7.5 recoge su IPS y Hmean-wIPC (se resaltan los mejores valores). También recoge la energía dinámica por instrucción (EpI) consumida por el PRF como aproximación de primer orden de la eficiencia energética. La energía total se ha calculado multiplicando el número de accesos al PRF por la energía de cada acceso, obtenida para cada tamaño a partir del modelo de Rixner [RDK+00]. Se usa la EpI porque el número de instrucciones ejecutadas por cada política puede ser muy diferente. Hemos observado que los valores de EpI para los mejores puntos de diseño son muy similares. También hemos encontrado que *Flush* no aprovecha el menor coste energético de acceder a un PRF más pequeño debido a la energía consumida por las reejecución de instrucciones anuladas.

Hill-climbing con un PRF de 320 entradas parece la mejor opción cuando se ejecutan cargas de trabajo de 2 threads. Si el objetivo es un procesador de 4 threads, podría optimizarse la productividad (*throughput*) o la equidad (*fairness*). Si se está más interesado en la productividad, la mejor opción es *Hill-climbing* con un PRF de 384 entradas porque obtiene los mejores valores

de IPC e IPS. Sin embargo, si la prioridad es el compromiso prestaciones-equidad, *Sra* y *Dcra* son las mejores políticas.

7.5 Conclusiones

Este capítulo analiza el impacto combinado del tamaño del PRF y de la política de asignación de recursos en las prestaciones de un procesador SMT, mostrando que un análisis desacoplado no es una elección correcta. Hemos explorado un rango amplio de tamaños de PRF y una extensa serie de políticas de asignación de recursos y hemos encontrado que para pequeños PRFs, *Flush* es la política que obtiene mejores valores de productividad (IPS) y equidad (Hmean-wIPC). Para PRFs más grandes, nuestros resultados muestran que *Hill-climbing* siempre consigue los mejores valores de productividad y equidad, excepto para los procesadores que soportan 4 threads que tienen que ejecutarse de una manera equilibrada. En tal caso, *Sra* y *Dcra* son las mejores políticas.

En lo que a obtener los mejores valores de productividad y equidad se refiere, proponemos un procedimiento sencillo para encontrar el mejor tamaño de PRF para una política de asignación dada. Aplicando tal procedimiento a un conjunto de políticas de asignación de recursos, nuestros resultados muestran que es posible encontrar un tamaño de PRF que maximice IPS a la vez que alcanza valores cercanos al máximo de Hmean-wIPC. Entre los puntos de diseño seleccionados, *Hill-climbing* con 320 registros es el mejor esquema para cargas de trabajo de 2 threads. Para cargas de trabajo de 4 threads, un PRF de 384 registros es la mejor opción, ya sea con *Hill-climbing* (la mejor productividad) o con *Sra* o *Dcra* (la mejor equidad). Este último punto de diseño supone una mejora en IPS del 23% con respecto al procesador de 2 threads, pero con un incremento del 17% en el tamaño del PRF.

7.6 Referencias

[CFR+03] F.Cazorla, E.Fernández, A.Ramírez and M.Valero. Improving Memory Latency aware fetch policies for SMT processors. *Int'l Symp. High Performance Computing*, Oct. 2003.

[CRV+04a] F.Cazorla, A.Ramírez, M.Valero and E.Fernández. DCache Warn: an I-Fetch Policy to Increase SMT Efficiency. *IPDPS*, pp. 74-83, 2004.

[CRV+04b] F. J. Cazorla, A. Ramirez, M. Valero and E. Fernandez. Dynamically Controlled Resource Allocation in SMT Processors. *Int'l Symp. on Microarchitecture (MICRO)*, pp. 171–182. IEEE Computer Society, December 2004.

[ChYe06] S.Choi and D.Yeung. Learning-Based SMT Processor Resource Distribution via Hill-Climbing. *Int'l Symp. on Computer Architecture (ISCA)*, 2006, pp. 239-251.

[ElAl03] A.El-Moursy and D.Albonesi. Front-End Policies for Improved Issue Efficiency in SMT processors. *HPCA*, pp. 31-40, February 2003.

[EyEc07] S.Eyerman, L.Eeckhout. A Memory-Level Parallelism Aware Fetch Policy for SMT Processors, *HPCA*, 2007, pp. 240-249.

[KST04] R.Kalla, B.Sinharoy and J.Tendler, IBM Power5 Chip: A Dual-Core Multithreaded Processor, *IEEE Micro*, vol. 24, no. 2, Mar.-Apr. 2004, pp. 40-47.

[KoMa03] D.Koufaty, D.Marr, Hyperthreading technology in the netburst microarchitecture, *IEEE Micro,* vol 23, no 2, Mar.-Apr. 2003, pp.56-65.

[LFM+01] K.Luo, M.Franklin, S.Mukherjee and A.Seznec. Boosting SMT Performance by Speculation Control. *IPDPS*, pp. 9-16, April 2001.

[LGF01] K.Luo, J.Gummaraju and M.Franklin. Balancing throughput and fairness in SMT processors. *ISPASS*, Nov. 2001.

[RaRe03] S.E.Raasch, S.K.Reinhardt, The impact of resource partitioning on SMT processors, *PACT*, pp. 15-25, Sept. 2003.

[RDK+00] S. Rixner, W. Dally, B. Khailany, P. Mattson, U. Kapasi and J.Owens, Register Organization for Media Processing. *HPCA*, pp. 375-386, Jan. 2000.

[SPH+02] T.Sherwood, E.Perelman, G.Hamerly and B.Calder. Automatically Characterizing Large Scale Program Behavior. *ASPLOS*, pp. 45-57, Oct. 2002.

[SPL08] S.Subramaniam, M.Prvulovic and G. Loh. PEEP: Exploiting Predictability of Memory Dependences in SMT Processors. *HPCA*, pp. 137-148, Feb. 2008.

[TuBr01] D.Tullsen and J.Brown. Handling long-latency loads in a simultaneous multithreaded processor. *Int'l Symp. on Microarchitecture (MICRO)*, pp. 318–327. Dec. 2001.

[TEL95] D.Tullsen, S. Eggers and H. Levy. Simultaneous multithreading: Maximizing on-chip parallelism. *Int'l Symp. on Computer Architecture (ISCA)*, pp. 392-403, 1995

[TEE+96] D.Tullsen, S.Eggers, J.Emer, H.Levy, J.Lo, and R. Stamm. Exploiting choice: Instruction fetch and issue on an implementable simultaneous multithreading processor. *ISCA*, pp. 191–202, Apr. 1996.

[YER+99] A.Yoaz, M.Erez, R.Ronen and S.Jourdan. Speculation techniques for improving load related instruction scheduling. *ISCA*, May 1999.

[VCP+07] J.Vera, F.Cazorla, A.Pajuelo, O.Santana, E.Fernandez and M.Valero. FAME: FAirly MEasuring Multithreaded Architectures. *PACT*. September 2007.

CAPÍTULO 8

Conclusiones y Líneas Abiertas

8.1 Conclusiones

Cada nueva generación de procesadores requiere bancos de registros con un mayor número de entradas. Además, los modernos procesadores *Simultaneous Multithreading* (SMT) necesitan muchos registros para almacenar los valores generados por varios hilos de ejecución. Este incremento en el número de entradas afecta negativamente a la latencia, consumo y área de un banco de registros que tiene que integrarse en procesadores con frecuencias altas de funcionamiento y restricciones térmicas.

Esta Tesis aborda esta problemática del diseño del banco de registros en procesadores superescalares con ejecución fuera de orden. El objetivo es mejorar el mecanismo de renombre de registros implementado con un banco de registros físico (*Physical Register File*, PRF), de forma que se reduzca el número de detenciones por no disponer de registros físicos. Los procesadores actuales necesitan más registros de los estrictamente necesarios para almacenar los valores con consumidores pendientes, ya que el mecanismo de renombre convencional es conservador. Un registro físico se asigna demasiado pronto, varios ciclos antes de que contenga un valor, y su liberación se realiza demasiado tarde, en algunos casos muchos ciclos después de haber sido leído por última vez. Nuestro propósito es que los registros del PRF se gestionen de una manera más eficiente, manteniendo solamente aquellos valores que van a ser consumidos por alguna instrucción. Un banco de registros auxiliar (auXiliary Register File, XRF), almacenará los valores necesarios para soportar especulación de control y excepciones precisas. Esta mejora tiene dos aplicaciones: puede mantenerse el tamaño del PRF y ganar prestaciones, y también puede reducirse el número de registros físicos sin perder rendimiento por la gestión más eficiente de los mismos.

En todas las soluciones propuestas, se ha tratado de limitar la complejidad y el coste hardware de las estructuras y el control necesario para su implementación.

8.1.1 Potencial del Renombre Especulativo

Se ha definido el concepto de *patrón de uso* de una instrucción y se ha propuesto aplicar la predicción de patrones con último uso de un registro para mejorar tanto la política de asignación como la de liberación de registros físicos (Speculative Renaming based on Last-Use Prediction, SR-LUP). Se ha demostrado la viabilidad de la predicción del patrón de uso de una instrucción por parte de una estructura sencilla en términos de lógica de control y requerimientos de memoria. También se ha evaluado el potencial de este mecanismo alternativo analizando los límites de prestaciones que obtendría un predictor de último uso oráculo (SR-Oracle Based, SR-OB). Se ha observando que el potencial de mejora de prestaciones es mucho mayor para programas de coma flotante que para enteros. Para éstos últimos códigos, se ha mostrado que para un procesador superescalar de grado 8 con un ROB de 256 entradas, un PRF con 160 entradas gestionado de forma convencional obtiene el mismo rendimiento que otro con la mitad de entradas gestionado por un mecanismo SR-OB. Con el cambio pueden incluso ganarse prestaciones en términos de IPS si la reducción del tamaño del PRF permite un incremento de la frecuencia del procesador. El grueso de este trabajo está publicado en [AMV+06].

8.1.2 Microarquitectura con soporte de Renombre de Registros Especulativo

Se ha diseñado una microarquitectura para la ejecución de políticas arbitrarias que permitan la omisión de asignación y la liberación anticipada de registros físicos. Se ha tratado de reducir la complejidad de las estructuras hardware añadidas y de los cambios introducidos. La nueva microarquitectura para Renombre de Registros Especulativo se ha evaluado con políticas de asignación y liberación de registros físicos basadas en la predicción de último uso de un registro. Para la evaluación se han diseñado dos predictores de último uso, uno sencillo pero que es capaz de capturar gran parte de los patrones de uso y otro más elaborado que permite obtener más prestaciones. Asimismo, se ha analizado el espacio de diseño del XRF (entradas y puertos), y se ha encontrado que con los predictores anteriores, una configuración del XRF con un número de puertos reducido permite explotar gran parte del potencial de SR-LUP. Estos contenidos están publicados en [AMV+07].

8.1.3 Aplicación de Técnicas de Decaimiento en el Diseño de Predictores Sticky

Se ha propuesto la aplicación de técnicas de decaimiento (*decay*) en el diseño de predictores *sticky* con el objetivo de reducir el consumo estático de energía sin perder prestaciones. Por una parte, puede reducirse el consumo ligado a las corrientes de fuga desconectando las celdas de memoria mediante transistores de paso (*gated-Vdd*) o implementándolas con celdas RAM cuasi-estáticas de 4 transistores. Esta última técnica permite también reducir el área. Por otra parte, la descarga de las celdas de memoria que se produce con estas técnicas de decaimiento permite eliminar contenidos obsoletos que causan fallos de predicción. Se ha evaluado esta idea con un predictor *sticky* de último uso (SLUP) y se ha observado una gran reducción en el consumo estático de energía (hasta casi el 50%) sin degradar las prestaciones gracias a la reducción de los costosos falsos positivos identificando registros último uso. Estas ideas están publicadas en [AMV+09].

8.1.4 Gestión del Banco de Registros Físico en Procesadores SMT

Como paso previo a la aplicación del Renombre de Registros Especulativo a procesadores SMT, se ha analizado la sensibilidad del rendimiento de un procesador SMT al tamaño del banco de registros y a la política de asignación de recursos. Se ha modelado con detalle el mecanismo de renombre convencional en un procesador SMT así como una serie de mecanismos que distribuyen los recursos del procesador (como por ejemplo, las entradas del banco de registros) entre los distintos hilos hardware. Creemos que este es el primer trabajo que realiza una comparativa detallada entre distintas políticas de asignación de recursos para un rango amplio de tamaños del PRF. También parece el primer trabajo que analiza la interacción entre tamaño del PRF y política de asignación de recursos. Se ha mostrado que la mejor política de asignación de recursos depende del tamaño del PRF y del número de threads que se ejecutan. Este trabajo se publicó en [AMC+08].

8.1.5 Otras contribuciones

Entre las contribuciones secundarias me gustaría señalar la clasificación de trabajos relacionados con el diseño del banco de registros. Creo que se ha realizado una revisión del estado del arte bastante concienzuda, cuyo resultado permite aproximarse al trabajo previo desde una perspectiva diferente a la tomada por publicaciones anteriores.

8.2 Líneas abiertas

Para finalizar, citamos algunas líneas de investigación que pueden derivarse del trabajo realizado:

- La microarquitectura con soporte de renombre especulativo podría integrar otras estrategias de renombre, no necesariamente restringidas a políticas de asignación y liberación de registros. Por ejemplo, podría aprovecharse el coste hardware añadido, en especial, el mecanismo de registros virtuales, para implementar propuestas de compartición de registros físicos.

- La ejecución SMT provoca una gran demanda de registros físicos. La aplicación de SR-LUP a un procesador SMT podría o bien aumentar el número de registros físicos disponibles para distribuir entre los threads o bien reducir los requerimientos de registros físicos. Sería interesante investigar si la aplicación de SR-LUP a SMT podría mejorar las prestaciones de políticas sencillas de asignación de recursos o incluso propiciar la aparición de alguna nueva estrategia de distribución de recursos.

- Las técnicas de decaimiento (*decay*) han mostrado un gran potencial de ahorro energético en las estructuras a las que se ha aplicado (caches y predictores de salto), y unas sorprendentes prestaciones en los predictores de tipo sticky. Podría extenderse su aplicación a otros componentes de procesadores actuales que tengan problemas de consumo de energía y a otros predictores de tipo *sticky*.

8.3 Referencias

[AMV+06] J. Alastruey, T. Monreal, V. Viñals and M. Valero. “Speculative Early Register Release”. *Proc. of the 3rd conference on Computing frontiers* (CF), May 2006, pp 291-302.

[AMV+07] J. Alastruey, T. Monreal, V. Viñals and M. Valero. “Microarchitectural Support for Speculative Register Renaming”. *IEEE Int. Parallel & Distributed Processing Symp.* (IPDPS), March 2007.

[AMC+08] J. Alastruey, T. Monreal, F.J. Cazorla, V. Viñals y M. Valero. Selection of the Register File Size and the Resource Allocation Policy on SMT Processors. *Proceedings of the 20th International Symposium on Computer Architecture and High Performance Computing* (SBAC-PAD), Campo Grande, Brasil, Oct 29-Nov 1, 2008. IEEE Computer Society, pp. 63-70, ISBN 978-0-7695-3423-7.

[AMV+09] J. Alastruey, T. Monreal, M. Valero y V. Viñals. "Implementación de un Predictor de Último Uso con Decaimiento". XX Jornadas de Paralelismo, La Coruña (España). 17-18 Sept. 2009.

ANEXO A

Análisis de patrones de uso en SPEC CPU 2000

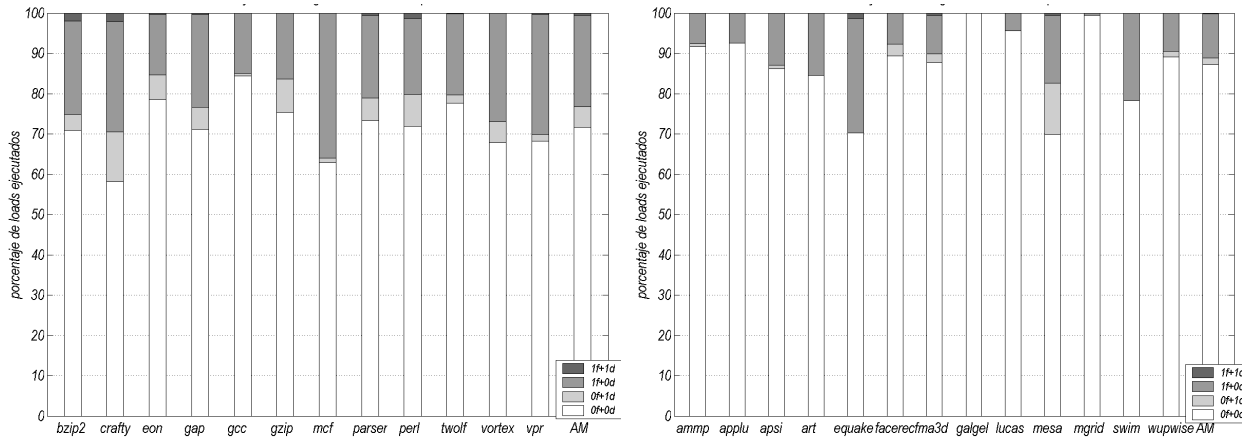
Este Anexo extiende el estudio del comportamiento de instrucciones último uso realizado en el Capítulo 4.

A.1 Frecuencia dinámica de patrones de uso

La Figura A.2 y la Figura A.1 muestran la distribución de los patrones de uso de las instrucciones load y store de camino correcto ejecutadas por SPEC CPU2000 (int y fp). Se distinguen las ocasiones en que hay último uso de m registros fuentes y n registros destino (2f+1d, 2f+0d, 1f+1d, 1f+0d, 0f+1d, 0f+0d).

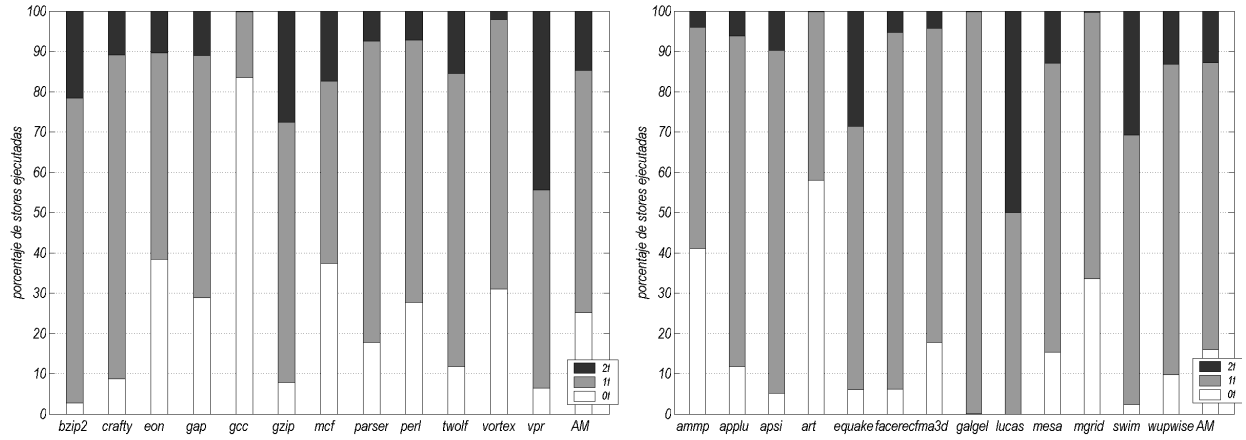
El patrón "último uso de un registro fuente" (1f) se da en más del 60% y del 70% de los stores enteros y de coma flotante respectivamente (en la mayor parte de las ocasiones es el dato que se escribe en memoria). El patrón "último uso de los dos registros fuente" (2f) se da en el 12% y 14% de los stores en los programas enteros y de coma flotante respectivamente. En cuanto a los loads, el 23% y el 12% de las ocasiones son los últimos consumidores de un valor (int y fp).

Muchas instrucciones tienen operandos nulos, de hecho, casi la mitad de las instrucciones no tiene registro fuente 1 -ver Tabla A.1-. Así que si analizamos el porcentaje de operandos válidos que son último uso, puede observarse que, en media, más del 50% de los registros fuente son último uso; en concreto, más del 70% de los registros fuente 1 y en torno al 40% de los registros fuente 2. Este desequilibrio es debido a que la arquitectura analizada -Alpha-, almacena el dato a escribir por un store en el registro fuente 1, que tiene mucha mayor probabilidad de ser último uso que el registro fuente 2, que contiene un valor a partir del cual se calcula la dirección de memoria donde se almacena el dato. Esta última distribución puede utilizarse para asesorar al predictor de último uso. Destacar también que para SPECint2000, casi el 9% de los registros destino referencian un valor que nunca se leerá. Esta medida concuerda con la publicada en [BuSo02b]. Para más detalles al respecto, consultar el Anexo, sección A.2.



(a) SPEC CINT2000 **(b) SPEC CFP2000**

Figura A.1 Distribución de los patrones de uso de las instrucciones load ejecutadas en SPEC CINT2000 (a) y CFP2000 (b). No hay registro fuente 1.



(a) SPEC CINT2000 **(b) SPEC CFP2000**

Figura A.2 Distribución de los patrones de uso de las instrucciones store ejecutadas en SPEC CINT2000 (a) y CFP2000 (b). No se distingue entre el registro fuente 1 y el fuente 2.

A.2 Distribución de últimos usos por operandos fuentes y destino

Se han analizado las trazas de ejecución simuladas (SPEC CPU2000, Alpha). En la Tabla A.3 se muestran características de los operandos fuente y destino. La variabilidad entre los operandos fuente es debida a la arquitectura analizada (Alpha). Los resultados son coherentes con la intuición: el registro fuente 2 almacena la dirección de memoria que se va a acceder (ya lea lectura -load- o escritura -store-), así que su contenido será más frecuentemente reutilizado, y por tanto, el operando fuente 2 será menos ocasiones último uso que el del operando fuente 1, utilizado para albergar el dato que se va a escribir en memoria.

Tabla A.1 Porcentaje de las instrucciones con operandos no nulos.

	int	fp	
Destino	**67,6**	**81,4**	**% de las instrucciones**
Fuente 1	**47,4**	**60,7**	
Fuente 2	**72,0**	**88,8**	

Tabla A.2 Distribución de últimos usos por operando

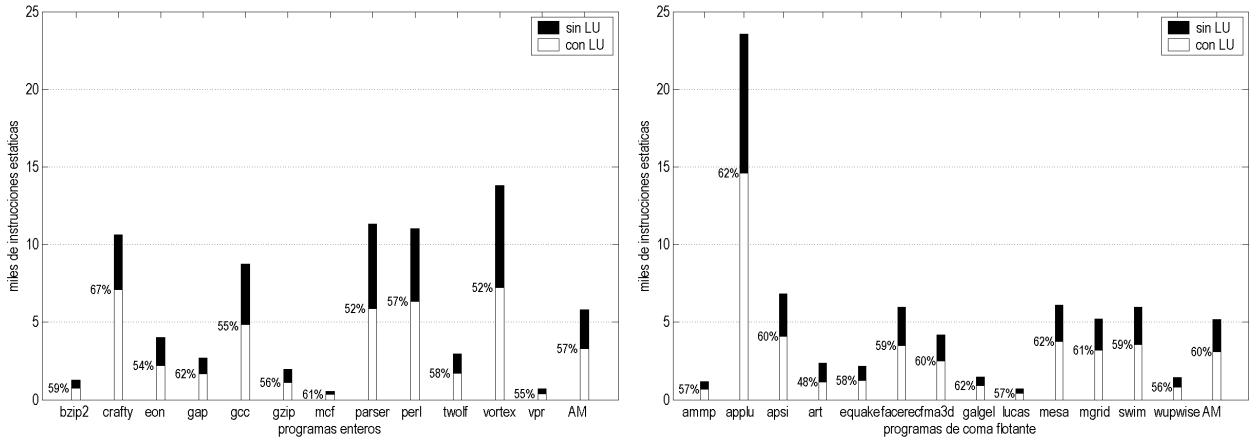
	int	fp	
Destino	**8,7**	**2,4**	**% de los operandos**
Fuente 1	**73,6**	**71,5**	
Fuente 2	**37,6**	**41,2**	
Fuentes 1 + 2	**51,6**	**53,3**	

Tabla A.3 Distribución de últimos usos por operando

	int	fp	
operandos destino LU	**5,7**	**1,9**	**% de las instrucciones**
operandos fuente 1 LU	**34,9**	**42,8**	
operandos fuente 2 LU	**27,0**	**36,8**	
operandos destino no nulos	**67,6**	**81,4**	**% de las instrucciones**
operandos fuente 1 no nulos	**47,4**	**60,7**	
operandos fuente 2 no nulos	**72,0**	**88,8**	
operandos destino LU	**8,7**	**2,4**	**% de los operandos**
operandos fuente 1 LU	**73,6**	**71,5**	
operandos fuente 2 LU	**37,6**	**41,2**	
operandos fuente LU	**51,6**	**53,3**	

A.3 Frecuencia estática de los patrones de uso

La Figura A.3 muestra el número de instrucciones estáticas de camino correcto ejecutadas por SPEC CPU2000 (int y fp) y la fracción de ellas que contiene algún patrón con último uso de un registro.



(a) SPEC CINT2000 **(b) SPEC CFP2000**

Figura A.3 Número de instrucciones estáticas con y sin algún registro último uso en alguna de sus instancias consolidadas para SPEC CINT2000 (a) y CFP2000 (b). La última columna de cada figura es la media aritmética (AM).

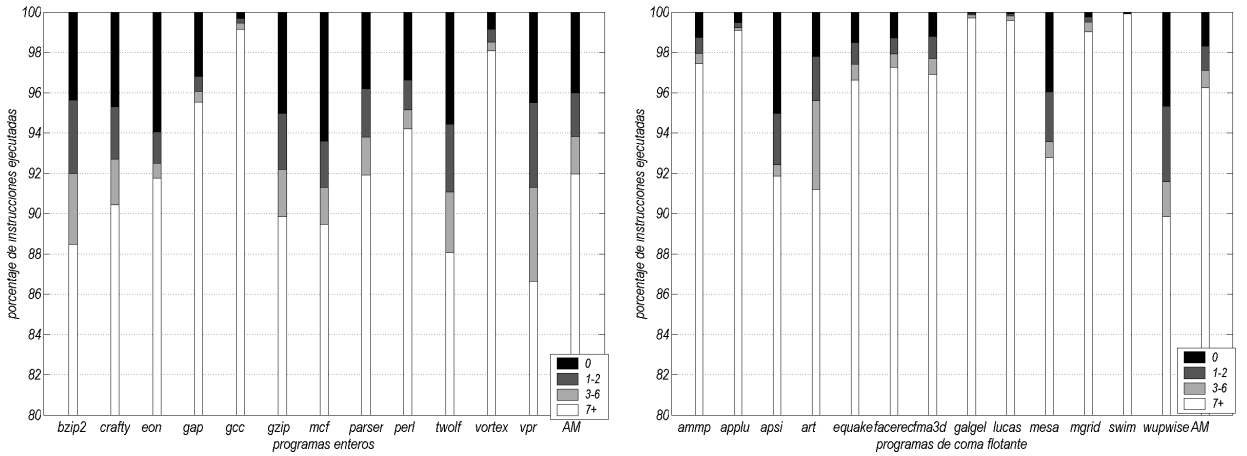
El primer valor cambia bastante con la aplicación (desde 549 instrucciones para mcf hasta 23.564 para applu), y aproxima la cantidad de entradas que necesitará un predictor de patrones de uso para dar cobertura a las instrucciones con último uso en alguna de sus ejecuciones.

La segunda medida es más homogénea entre aplicaciones (para enteros varía entre 52% y 67%, y para coma flotante entre 48% y 62%). En media, casi el 60% de las instrucciones estáticas tiene alguna instancia que presenta último uso de un registro. De estos datos se puede concluir que existe correlación entre el tamaño estático del código y el número de instrucciones que en alguna de sus ejecuciones presentan un patrón con último uso de un registro.

A.4 Repetitividad del patrón de uso

Una medida interesante en el diseño de un predictor de último uso es la repetitividad (localidad temporal) de los patrones de uso. Se trata de comprobar si instancias consecutivas de una instrucción estática presentan el mismo patrón de uso. A mayor repetitividad, más precisión tendrá la predicción asociada a una instrucción estática, de ahí el interés por esta propiedad.

Hemos medido el porcentaje de instrucciones consolidadas cuyo patrón de uso coincide con el de al menos sus 1, 3 y 7 instancias anteriores (se han escogido estos valores porque se ajustan a la historia que podrían resumir predictores de 1, 2 y 3 bits). Los resultados detallados por benchmark, así como la media, se muestran en la Figura A.4. Puede observarse como el 96%/98% (int/fp) del total de instrucciones ejecutadas presenta el mismo patrón de uso que su instancia anterior, el 94%/97% el mismo que sus 3 últimas instancias y el 92%/96% repite el de sus 7 instancias anteriores. Se constata una alta localidad en el patrón de uso exhibido por las instrucciones, mayor en el caso de los programas de coma flotante, por lo que la predicción de patrón de uso parece más fiable para estos códigos. Además, en los códigos de coma flotante hay poca diferencia entre los porcentajes de instrucciones que repiten el patrón de las 7, 3 y última instancia, por lo que no se espera que las prestaciones del predictor pudieran mejorar apreciablemente al aumentar los bits de historia.



(a) SPEC CINT2000 **(b) SPEC CFP2000**

Figura A.4 Repetitividad de los patrones de uso: distribución de las instrucciones ejecutadas en SPEC CINT2000 (a) y CFP2000 (b) según su patrón de uso coincida con el de al menos sus n instancias anteriores (n=0,1,3,7). Nótese el offset del eje y.

A.5 Referencias

[BuSo02b] J.A. Butts and G. Sohi, Characterizing and Predicting Value Degree of Use", *Proc. 35th Int'l Symp. Microarchitecture* (MICRO 02), Nov. 2002, pp. 15-26.

ANEXO B

Aspectos del Diseño de Predictores de Último Uso

En este Anexo se detallan aspectos relacionados con el diseño del Predictor de Último Uso.

B.6 Consideraciones generales

El predictor LU debe asociar un patrón de uso a todas las intrucciones con un registro (todas las que consumen o generan un valor). Esto supone un 96.7% y un 98.9% de las instrucciones estáticas de SPEC CINT2000 y CFP2000, respectivamente. En programas con gran tamaño estático de código, estos porcentajes conllevan la predicción de patrón de uso a un elevado número de instrucciones. Para reducir el espacio de almacenamiento del LUP, puede optarse por guardar solamente el estado de aquellas instrucciones que en alguna de sus instancias presentan un patrón con último uso de alguno de sus registros -en torno al 60% de las instrucciones estáticas, ver Figura A.3-. Así, cuando no haya predicción disponible para una instrucción, se asumirá para ella el patrón “no hay últimos usos de sus registros”.

B.7 Determinación de patrones de uso

Se necesita un mecanismo de observación de últimos usos de los operandos de las instrucciones tanto para entrenar el predictor de patrones de uso como para una posible verificación de sus predicciones.

El patrón de uso de una instrucción A se va conociendo en tiempo de ejecución conforme se jubilan las instrucciones que consumen y/o redefinen los registros lógicos fuente y destino de A. Hasta que no se jubilan todas las instrucciones que vuelven a referenciar cada uno de los registros de una instrucción A, no se conoce el patrón de uso de la instrucción A. Este proceso de de-

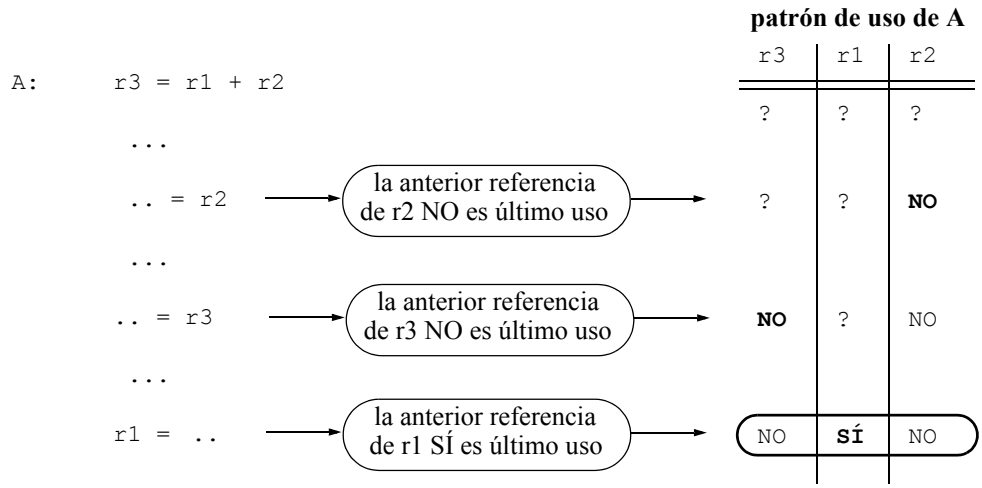


Figura B.1 Determinación del patrón de uso de una instrucción. La tabla va reflejando el conocimiento del patrón de uso de la instrucción A conforme se ejecutan las instrucciones posteriores.

terminación del patrón de uso de una instrucción puede durar muchos ciclos. La Figura B.1 muestra un ejemplo de la evolución del conocimiento del patrón de uso de una instrucción.

Cuando una instrucción B consolida, de cara a obtener el patrón de uso de cada una de las instrucciones que hicieron la última referencia a los registros de la instrucción B, se extrae distinta información según el tipo de sus operandos:

- fuente: la anterior referencia a ese registro lógico (fuente o destino) no es último uso.
- destino: la anterior referencia a ese registro lógico sí es último uso.

El patrón de uso de una instrucción se puede obtener a partir de la información de último uso de sus registros. Llamamos Tabla de Últimos Usos en Orden (*In-Order Last Use Table*, IO-LUT) a la estructura que identifica las instrucciones que han referenciado por última vez un registro lógico. Es una tabla indexada por registro lógico que almacena el PC de la última instrucción que usó un valor y el tipo de operando que referenciaba dicho valor (fuente 1, 2 ó destino). Para cada instrucción que consolida, se realizan las siguientes acciones, ilustradas en la Figura B.2:

1. Se apunta su PC y el tipo de operando (fuente 1 ó 2) en las entradas correspondientes a sus registros lógicos fuente. Las flechas con trazo continuo en la Figura B.2 marcan estas acciones.
2. Se inicializa la entrada correspondiente al registro lógico destino (PC y operando destino). Los anteriores valores de esta entrada se envían al predictor para indicarle la última instrucción que usó el registro y en qué operando estaba. Las flechas discontinuas en la Figura B.2 ilustran estas acciones. Notar que cada entrenamiento hace referencia al último uso del registro destino de la instrucción que consolida y que el predictor sólo almacenará información relativa a instrucciones que en alguna de sus instancias tengan un registro último uso. Con el objetivo de marcar la predicción para evitar *aliasing* (colisión en el predictor de distintas entradas), podría optarse por almacenar también en el predictor el

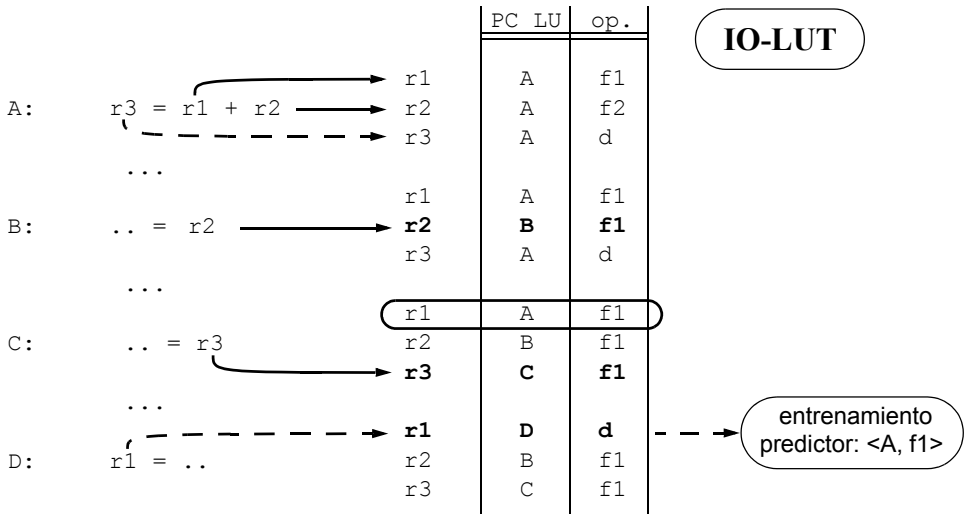


Figura B.2 Determinación de registros último uso mediante una tabla de últimos usos en orden (*In-Order Last-Use Table*, IO-LUT)

identificador del registro lógico último uso. Sin embargo esto supondría 5 bits por cada operando, por lo que incrementaría el coste hardware del predictor de forma significativa.

La IO-LUT tendrá 64 entradas que contendrán 2 bits para identificar el tipo de operando más los bits que se decidan almacenar del PC. Si por ejemplo, se opta por un predictor LU de 1024 entradas y mapeo directo sin tags, bastaría con guardar los 10 bits de menor peso del PC. Si J es el ancho de jubilación, la IO-LUT necesita 3 x J puertos de escritura para actualizar la información aportada por los registros fuentes y destino y J puertos de lectura para leer la información de entrenamiento del predictor. Como una de las entradas (la indexada por el registro lógico destino) debe ser leída antes de ser escrita, sería suficiente con 2 x J puertos de lectura y otros J de lectura/escritura.

Este mecanismo de entrenamiento tiene una limitación: si el predictor almacena el último uso de un registro de una instrucción, en posteriores ejecuciones de la misma instrucción se predecirá siempre último uso de dicho registro, aunque no vuelva a serlo nunca más. Esto se debe a que, con el mecanismo de entrenamiento que se ha explicado, el "desentrenamiento" para una instrucción dada sólo puede producirse si su entrada en el predictor es reemplazada por falta de espacio, y no porque sus siguientes instancias presenten patrones sin últimos usos. Este tipo de predictores, caracterizados por registrar predicciones no actualizables, son llamados predictores "pegajosos" o *sticky*.

Los experimentos realizados muestran que este mecanismo de entrenamiento causa muchos fallos predeciendo "sí es LU". Se necesita, por tanto, refinar el mecanismo de entrenamiento de forma que permita al predictor "olvidar" información obsoleta. A continuación se analizan varias alternativas:

- El proceso de desentrenamiento del predictor podría ser el siguiente: cada PC desalojado de la IO-LUT se busca en el predictor, y en caso de encontrarlo, se elimina la predicción de

```
V:    r1 =
        ...
U1:       = r1
        ...
BR:   br   V
        ...
U2:       = r1
        ...
end:  jump V
```

Figura B.3 Código ejemplo para mostrar una predicción de último uso incorrecta debida a los distintos caminos de ejecución.

último uso para el registro que corresponda (fuente o destino). Esto tiene un inconveniente: añade 2 x J puertos de lectura a la IO-LUT, y también aumenta el número de puertos de lectura y escritura del predictor (en el peor de los casos, habría que eliminar 2 x J predicciones de último uso). Los recursos utilizados para el desentrenamiento pueden reducirse de la siguiente forma: si una instrucción tiene entrada en el predictor de último uso (información que conoce en la etapa que recibe la predicción de patrón de uso), marca un bit en su entrada del ROB. Cuando dicha instrucción consolida, actualiza los campos mencionados de la IO-LUT y uno añadido: un bit que informa si la entrada estaba en el predictor. Sólo hay que desentrenar los PCs desalojados de la IO-LUT que tengan dicho bit activo, es decir, los que tienen entrada en el predictor. Notar que el bit asegura que una instrucción estaba o no en el predictor cuando se realizó la predicción de último uso, no el instante del desentrenamiento (puede haber sido desalojado en ese intervalo por otra instrucción).

- Invalidación de entrada del predictor tras su lectura. Esta estrategia elimina información que pueda dar lugar a fallos de predicción. Vamos a ilustrar esta idea con el código de la Figura B.3. Si suponemos que en la primera ejecución del bucle el salto condicional br es tomado, el predictor registrará una predicción de último uso del registro fuente de U1. Si en las posteriores ejecuciones del bucle el salto condicional br no es tomado, la predicción de último uso a U1 será incorrecta, ya que hay un consumidor posterior en orden de programa (U2). Aunque el predictor asocie a U2 una predicción de último, esta no tendrá efecto por ser posterior a U1. Si tras cada lectura de una predicción anulamos la información almacenada en el predictor, evitamos que se acumulen informaciones obsoletas de último uso.

- Diseño de predictor con decaimiento: existen opciones de diseño del predictor que permiten que la información almacenada vaya decayendo con el tiempo si no es accedida. Este decaimiento puede implementarse con transistores de paso o con celdas de memoria de 4 transistores. En el Capítulo 6 se analizan la aplicación de ambas implementaciones a un predictor LU.

ANEXO C

Detalles del Diseño del Banco de Registros Auxiliar

En este capítulo se amplía la información relativa al diseño de un Banco de Registros Auxiliar (Auxiliary Register File, XRF) con un número limitado de entradas y puertos. El objetivo es encontrar un XRF cuyo tiempo de acceso no sea mayor que el del PRF pero que permita explotar SR-LUP como si de un XRF sin límite de entradas y puertos se tratase.

C.1 Selección del número de entradas y puertos del XRF

Tal y como se ha razonado en el Capítulo 5, se necesita un XRF con suficientes entradas para no se cancelen predicciones de último uso (*Last Use*, LU) y con suficientes puertos para evitar la contención en la escritura de registros LU y lectura de Usos Inesperados (*Unexpected Uses*, UUs). A continuación se detalla el procedimiento de selección de número de entradas y puertos que se ha seguido en la Sección 5.4.3.

En primer lugar, hemos impuesto la condición de que el tiempo de acceso al XRF sea menor o igual que el tiempo de acceso al PRF. Esta restricción permite acotar también el consumo por acceso y el área de esta estructura auxiliar. Como resultado de este requisito, se descartan XRFs con muchas entradas y puertos. Para cada tamaño del PRF, tenemos una serie de configuraciones del XRF, desde XRFs con pocas entradas y muchos puertos, hasta XRFs con muchas entradas y pocas puertos. Se ha reducido el número de puntos de experimentación escogiendo, para cada tamaño de XRF, el XRF con más puertos. Del mismo modo, para cada número de puertos de XRF, se ha escogido el XRF con mayor número de entradas. Es decir, hemos seleccionado los XRFs de mayores prestaciones. Por ejemplo, para un PRF de 40 entradas, se va a evaluar desde un XRF de 96 entradas y 14 puertos hasta un XRF con 224 entradas y tan sólo 5 puertos (ver Tabla C.1). Para simplificar el proceso de diseño (al menos, en una primera aproximación), hemos considerado diseños simétricos para el XRFint y el XRFfp.

Tabla C.1 Configuraciones del XRF que tienen un tiempo de acceso similar al del PRF. Se muestran solamente algunos puntos de diseño a modo de ejemplo.

	Número de entradas del PRF				
	36	**40**	**48**	**64**	**80**
Número de registros (R) y puertos (P) del XRF	96R, 12P	96R, 14P	96R, 17P	96R, 21P	x
	128R, 9P	128R, 10P	128R, 13P	128R, 17P	128R, 21P
	160R, 7P	160R, 8P	160R, 10P	160R, 14P	160R, 17P
	192R, 5P	192R, 6P	192R, 8P	192R, 12P	192R, 15P
	x	224R, 5P	224R, 7P	224R, 10P	224R, 13P
	x	x	256R, 6P	256R, 9P	256R, 11P
	x	x	288R, 5P	288R, 8P	288R, 10P

Tabla C.2 Cantidad media de puertos del XRFint que se utilizan en el 98%, 99% y 99.9% de los ciclos. También se muestra el máximo de utilización y el programa que marca dicho máximo.

		Lecturas XRFint				Escrituras XRFint			
	PRF	**98%**	**99%**	**99.9%**	**máx (prog.)**	**98%**	**99%**	**99.9%**	**máx (prog.)**
SLUP	**36**	2	3	5	11 (twolf)	5	6	8	15 (vpr)
	40	2	3	5	11 (twolf)	5	6	8	15 (vpr)
	48	2	3	5	11 (twolf)	6	6	9	15 (bzip2)
	64	2	3	5	11 (twolf)	6	7	9	16 (crafty)
	80	2	3	5	11 (twolf)	6	7	9	16 (crafty)
	96	2	2	4	11 (twolf)	6	7	10	16 (crafty)
	128	2	2	4	11 (twolf)	6	7	10	16 (crafty)
	160	2	2	4	11 (twolf)	6	7	10	16 (crafty)
DULUP	**36**	1	1	3	10 (bzip2)	5	6	8	14 (vpr)
	40	1	1	4	11 (twolf)	5	6	8	15 (bzip2)
	48	1	2	4	11 (twolf)	6	6	9	15 (vpr)
	64	1	2	4	11 (twolf)	6	7	9	16 (crafty)
	80	1	2	4	11 (twolf)	6	7	10	16 (crafty)
	96	1	1	3	10 (bzip2)	6	7	10	16 (crafty)
	128	1	1	3	10 (bzip2)	6	7	10	16 (crafty)
	160	1	1	3	10 (bzip2)	6	7	10	16 (crafty)

Tabla C.3 Cantidad media de puertos del XRFfp que se utilizan en el 98%, 99% y 99.9% de los ciclos. También se muestra el máximo de utilización y el programa que marca dicho máximo.

		Lecturas XRFfp				Escrituras XRFfp			
	PRF	**98%**	**99%**	**99.9%**	**máx (prog.)**	**98%**	**99%**	**99.9%**	**máx (prog.)**
SLUP	36	0	1	4	8 (mesa)	5	6	8	9 (art)
	40	0	2	4	9 (mesa)	5	6	8	9 (ammp)
	48	1	2	4	10 (mesa)	6	6	8	10 (art)
	64	1	2	4	11 (mesa)	6	7	8	11 (ammp)
	80	1	2	4	12 (mesa)	6	7	8	12 (ammp)
	96	1	2	4	13 (mesa)	6	7	8	13 (ammp)
	128	1	2	4	14 (mesa)	6	7	8	14 (ammp)
	160	1	2	4	15 (mesa)	6	7	8	15 (ammp)
DULUP	36	0	0	1	8 (fma3d)	5	6	8	9 (art)
	40	0	0	1	6 (facer)	5	6	8	9 (equake)
	48	0	0	1	7 (fma3d)	6	6	8	11 (art)
	64	0	0	1	7 (fma3d)	6	7	8	10 (art)
	80	0	0	1	8 (mesa)	6	7	8	10 (fma3d)
	96	0	0	1	7 (fma3d)	6	7	8	10 (fma3d)
	128	0	0	1	7 (mesa)	6	7	8	10 (ammp)
	160	0	0	0	8 (mesa)	6	7	8	10 (ammp)

El siguiente paso consiste en distribuir los puertos disponibles del XRF en puertos de lectura y escritura. Los puertos de escritura, a su vez, se desglosan en puertos de escritura por último uso de registro fuente y registro destino. Para asistir esta tarea, hemos analizado la utilización del número de puertos de lectura y escritura del XRF en una estructura sin límite de entradas y puertos. La Tabla C.2 muestra el número de puertos del XRFint utilizados en el 98%, 99% y 99.9% de los ciclos de ejecución. También refleja el máximo número de puertos utilizados y el programa que ha marcado dicho máximo. La Tabla C.3 muestra los mismos datos para el XRFfp. A partir de dichos valores, seleccionamos varias combinaciones de puertos de lectura y escritura. La Tabla C.4 muestra algunas de las configuraciones evaluadas. Reseñar que es necesario un mínimo de 2 puertos de lectura, ya que en caso de no disponerlos, una instrucción con dos registros fuente en el XRF se quedaría bloqueada sin posibilidad de lanzarse. En el desglose de puertos de escritura se ha combinaciones que no incluyen puertos de escritura por último uso de registro destino, es decir, que las predicciones de último uso de registro destino se anulan (se tiene un esquema sólo con predicciones de último uso para registros fuente, ERPR-preds). Se han probado asimismo configuraciones de puertos diferentes para XRFint y XRFfp.

Finalmente, hemos evaluado las configuraciones resultantes del proceso de selección para buscar la que mejor explota SR-LUP en términos de IPC e IPS. Las diferencias de prestaciones entre los puntos de diseño no son muy apreciables, sobre todo entre los XRFs que respaldan los

Tabla C.4 Configuraciones de XRFint y XRFfp evaluadas. Se muestran solamente algunos puntos de diseño a modo de ejemplo. En negrita se resaltan las configuraciones que mejores prestaciones

	Número de entradas del PRF				
	36	**40**	**48**	**64**	**80**
Número de registros (R), puertos de lectura (L) y de escritura (E) del XRF	128R 2L+7E	128R 3L+7E	x	x	x
	160R 2L+5E	**160R 2L+6E**	160R 3L+7E	x	x
	192R 2L+3E	192R 2L+4E	**192R 2L+6E**	192R 4L+8E	192R 5L+10E
	x	x	224R 2L+5E	**224R 3L+7E** 2L+8E	**224R 4L+9E** 3L+10E
	x	x	x	256R 2L+7E	256R 3L+8E
	x	x	x	x	288R 3L+7E

PRFs de más de 40 entradas. Para los XRFs asociados a los PRFs más ajustados (36 y 40 registros), las diferencias son más notables, ya que es más apreciable el efecto de añadir un puerto a un XRF con un número muy reducido de puertos que a un XRF con un número más holgado de puertos. En la Tabla C.4 se resaltan los XRFs que mejores prestaciones obtienen para el rango de PRF mostrado. El listado completo de configuraciones del XRF se muestra en el Capítulo 5.

Listado de Referencias

[ARS03] H. Akkary, R. Rajwar, S.Srinivasan. Checkpoint Processing and Recovery: Towards Scalable Large Instruction Window Processors. Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2003). pp. 423. Diciembre 2003.

[AMV+06] J. Alastruey, T. Monreal, V. Viñals and M. Valero. “Speculative Early Register Release”. *Proc. of the 3rd conference on Computing frontiers* (CF), May 2006, pp 291-302.

[AMV+07] J. Alastruey, T. Monreal, V. Viñals and M. Valero. “Microarchitectural Support for Speculative Register Renaming”. *IEEE Int. Parallel & Distributed Processing Symp.* (IPDPS), March 2007.

[AMC+08] J. Alastruey, T. Monreal, F.J. Cazorla, V. Viñals y M. Valero. Selection of the Register File Size and the Resource Allocation Policy on SMT Processors. *Proceedings of the 20th International Symposium on Computer Architecture and High Performance Computing* (SBAC-PAD), Campo Grande, Brasil, Oct 29-Nov 1, 2008. IEEE Computer Society, pp. 63-70, ISBN 978-0-7695-3423-7.

[AMV+09] J. Alastruey, T. Monreal, M. Valero y V. Viñals. "Implementación de un Predictor de Último Uso con Decaimiento". XX Jornadas de Paralelismo, La Coruña (España). 17-18 Sept. 2009.

[BaSo03] S. Balakrishnan and G. Sohi. Exploiting Value Locality in Physical Register Files. *In Proceedings of the 36th Annual IEEE/ACM international Symposium on Microarchitecture* (MICRO 03), pp. 265. San Diego (USA), December 03- 05, 2003.

[BDA01a] R. Balasubramonian, S. Dwarkadas, and D.H. Albonesi, Dynamically Allocating Processor Resources between Nearby and Distant ILP. *Proceedings of the 28th International Symposium on Computer Architecture* (ISCA 01), pp. 26-37, June 2001.

[BDA01b] R. Balasubramonian, S. Dwarkadas, and D. H. Albonesi, Reducing the Complexity of the Register File in Dynamic Superscalar Processors. *Proceedings of the 34th International Symposium on Microarchitecture* (MICRO 01), pp. 237-249, December 2001.

[BSP+06a] D. Balkan, J. Sharkey, D. Ponomarev and A. Aggarwal, “Address-Value Decoupling for Early Register Deallocation”, Proc. 35th Int’l Conf. on Parallel Processing (ICPP-06), Aug. 2006, pp. 337-346.

[BSP+06b] D. Balkan, J. Sharkey, D. Ponomarev and K. Ghose, “SPARTAN: Speculative Avoidance of Register Allocation to Transient Values for Performance and Energy Efficiency”, *Proc. 15th Int’l Conf. Parallel Architectures and Compilation Techniques (PACT 06)*, Sept. 2006, pp. 265-274.

[BTM+02] E. Borch, E. Tune, S. Manne and J. Emer, “Loose Loops Sink Chips”, *Proc. 8th Int’l Symp. High-Performance Computer Architecture (HPCA 02)*, Feb. 2002, pp. 299-310.

[BuAu97] D. Burger and T.M. Austin, The Simplescalar Tool Set v2.0, Technical Report 1342, Computer Science Dept., University of Wisconsin-Madison, June 1997.

[BMK+05] T. N. Buti, , R. G. McDonald, Z. Khwaja, A. Ambekar, H. Q. Le, W. E. Burky, and B. Williams. Organization and implementation of the register-renaming mapper for out-of-order IBM POWER4 processors. *IBM J. Res. Dev. 49, 1* , pp. 167-188, Jan. 2005

[BuSo02a] J.A. Butts and G. Sohi, “Dynamic dead-instruction detection and elimination”, *Proc. 10th Int’l Conf. Architectural Support for Programing Languages and Operating Systems (ASPLOS 02)*, Oct. 2002, pp. 199-210.

[BuSo02b] J.A. Butts and G. Sohi, “Characterizing and Predicting Value Degree of Use”. Proc. 35th Int’l Symp. Microarchitecture (MICRO 02), Nov. 2002, pp. 15-26.

[BuSo04] J.A. Butts and G. Sohi, “Use-Based Register Caching with Decoupled Indexing”, Proc. 31st Int’l Symp. Computer Architecture (ISCA 04), June 2004.

[CFR+03] F.J. Cazorla, E.Fernández, A.Ramírez and M.Valero. Improving Memory Latency aware fetch policies for SMT processors. *Int’l Symp. High Performance Computing*, Oct. 2003.

[CRV+04a] F.Cazorla, A.Ramírez, M.Valero and E.Fernández. DCache Warn: an I-Fetch Policy to Increase SMT Efficiency. *IPDPS*, pp. 74-83, 2004.

[CRV+04b] F. J. Cazorla, A. Ramirez, M. Valero and E. Fernandez. Dynamically Controlled Resource Allocation in SMT Processors. *Int’l Symp. on Microarchitecture (MICRO)*, pp. 171–182. IEEE Computer Society, December 2004.

[ChYe06] S.Choi and D.Yeung. Learning-Based SMT Processor Resource Distribution via Hill-Climbing. *Int’l Symp. on Computer Architecture (ISCA)*, 2006, pp. 239-251.

[CGV+00] J. Cruz, A. González, M. Valero, and N.P. Topham, Multiple-Banked Register File Architectures. *Proceedings of the 27th International Symposium on Computer Architecture* (ISCA 00), pp. 316-325, June 2000.

[ChEm98] G. Z. Chrysos, J.S. Emer. “Memory dependence prediction using store sets”. *Proc. International Symposium on Computer Architecture* (ISCA). 1998, pp. 142 - 153.

[CFP+07] B. Curran, E. Fluhr, J. Paredes, L. Sigal, J. Friedrich, Y.-H. Chan, C. Hwang. “Power-constrained high-frequency circuits for the IBM POWER6 microprocessor”. *IBM J. Res. & Dev.* vol. 51 no. 6, Nov. 2007.

[ElAl03] A.El-Moursy and D.Albonesi. Front-End Policies for Improved Issue Efficiency in SMT processors. *HPCA*, pp. 31-40, February 2003.

[EBP+04] O. Ergin, D. Balkan, D. Ponomarev and K. Ghose, "Increasing Processor Performance Through Early Register Release". *In Proceedings of the IEEE international Conference on Computer Design* (ICCD 04), pp. 480-487, San Jose (USA), October 11-13, 2004.

[EBG+04] O. Ergin, D. Balkan, K. Ghose, D. Ponomarev. Register Packing: Exploiting Narrow-Width Operands for Reducing Register File Pressure. *In Proceedings of 37th International Symposium on Microarchitecture* (MICRO 04), pp. 304-315, 04-08 Dec. 2004.

[EBP+06] O. Ergin, D. Balkan, D. Ponomarev and K. Ghose, “Early Register Deallocation Mechanisms Using Checkpointed Register Files”, *IEEE Transactions on Computers*, vol. 55, no. 9, Sept. 2006, pp. 1153-1166.

[EyEc07] S.Eyerman, L.Ecckhout. A Memory-Level Parallelism Aware Fetch Policy for SMT Processors, *HPCA*, 2007, pp. 240-249.

[FCJ+97] K. Farkas, P. Chow, N. Jouppi, and Z. Vranesic, The Multicluster Architecture: Reducing Cycle Time Through Partitioning. *Proceedings of the 30th International Symposium on Microarchitecture* (MICRO97), pp. 149-159, December 1997.

[FeOr02] E. Fetzer and T. Orton, "A Fully-Bypassed 6-Issue Integer Datapath and Register File on an Itanium Microprocessor". ISSCC Dig. Tech. Papers, pp. 418-419, 2002.

[FDL+05] E. Fetzer, D. Dahle, C. Little and Kevin Safford. The Multi-Threaded, Parity-Protected, 128-Word Register Files on a Dual-Core Itanium®-Family Processor. ISSCC Dig. Tech. Papers, Paper 20.5, pp. 382-383, Feb., 2005.

[FKM+02] K.Flautner, N.S. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy caches: simple techniques for reducing leakage power. *Proceedings of the 29th Annual International Symposium on Computer Architecture* (ISCA 02).

[GHS+99] M. Golden, S. Hesley, A. Scherer, M. Crowley, S.C. Johnson, S. Meier, D. Meyer, J.D.Moench, S. Oberman, H. Partovi, F. Weber, S. White, T. Wood, J. Yong. A seventh-generation x86 microprocessor. *IEEE Journal of Solid-State Circuits*, Volume 34, Issue 11, Nov. 1999, pp. 1466-1477.

[GoPa99] M. Golden, H. Partovi, A 500 MHz, write-bypassed, 88-entry, 90-bit register file. *Symposium on VLSI Circuits, Digest of Technical Papers*, pp. 105-108, 1999.

[GCO+04] R. González, A. Cristal, D. Ortega, A. Veidenbaum and M. Valero. A Content Aware Integer Register File Organization. *In Proceedings of the 31st Annual international Symposium on Computer Architecture* (ISCA 04), p. 314. München, Germany, June 19-23, 2004.

[Gwe97] L. Gwennap, MIPS R12000 to Hit 300 MHz. *Microprocessor Report, Micro Design Resources*, vol. 11, no. 13, pp. 1-4, October 1997.

[HNO97] L. Hammand, B. A. Nayfeh, and K. Olukotun. A single-chip multiprocessor. *IEEE Computer*, 30(9):79–85, Sep. 1997.

[HePa03] J.L. Hennessy, and D.A. Patterson. *Computer Architecture, A Quantitative Approach.* Morgan Kaufmann Publishers, San Francisco, Third Edition 2003.

[HSU+01] G. Hinton, D. Sager, M. Upton, D. Boggs, D. Carmean, A. Kyker, and P. Roussel.. The Microarchitecture of the Pentium 4 Processor. *Intel Technology Journal Q1*, February 2001.

[HJD+02] Z. Hu, P. Juang, P. Diodato, S. Kaxiras, K. Skadron, M. Martonosi, and D.W. Clark. Managing leakage for transient data: decay and quasi-static 4T memory cells. *In Proceedings of the 2002 International Symposium on Low Power Electronics and Design* (ISLPED 02), pp.52-55. Monterey (USA), August 12-14, 2002.

[HwPa87] W.W. Hwu, and Y.N. Patt, Checkpoint Repair for Out-of-Order Execution Machines. *Proceedings of the 14th International Symposium on Computer Architecture* (ISCA 87), pp. 18-26, June 1987.

[JOA+05] T.M. Jones, M.F.P. O'Boyle, J. Abella, A. González, and O. Ergin, "Compiler Directed Early Register Release", *Proc. 14th Int'l Conf. on Parallel Architectures and Compilation Techniques* (PACT 05), Sept. 2005, pp. 110-122.

[JRB+98] S. Jourdan, R. Ronen, M. Bekerman, B. Shomar, A. Yoaz. A Novel Renaming Scheme to Exploit Value Temporal Locality through Physical Register Reuse and Unification", *in Proc. of 31st. Int. Symp. on Microarchitecture* (MICRO 98), pp. 216-225, 1998.

[JSM+04] P.Juang, K. Skadron, M. Martonosi, Z. Hu, D.W. Clark, P.W. Diodato, and S. Kaxiras. Implementing branch-predictor decay using quasi-static memory cells. *ACM Trans. Archit. Code Optimization* (TACO 04), vol 1, n.2, pp. 180-219. Jun. 2004.

[KST03] R. Kalla, B. Sinharoy, and J. Tendler. Power5: Ibm's next generation power microprocessor. *Proc. 15th Hot Chips Symp*, pp 292–303, August 2003.

[KST04] R. Kalla, B. Sinharoy, and J. Tendler. IBM Power5 Chip: A Dual-Core Multithreaded Processor. *IEEE Micro*, vol. 24, no. 2, pp. 40-47, Mar./Apr. 2004.

[KHM01] S.Kaxiras, Z. Hu, M. Martonosi. Cache decay: exploiting generational behavior to reduce cache leakage power. ISCA 2001, pp- 240-251.

[Kes99] R.E. Kessler, The Alpha 21264 Microprocessor. *IEEE Micro*, vol. 19, no. 2, pp. 24-36, March/April 1999.

[KiMu03] N. Kim, and T. Mudge, T. Reducing register ports using delayed write-back queues and operand pre-fetch. *In Proceedings of the 17th Annual international Conference on Supercomputing* (ICS 03). San Francisco (USA), June 23-26, 2003.

[KoNa05] M. Kondo and H. Nakamura. A Small, Fast and Low-Power Register File by Bit-Partitioning. *In Proceedings of the 11th international Symposium on High-Performance Computer Architecture* (HPCA 2005), pp. 40-49. February 12-16, 2005.

[KoMa03] D.Koufaty, D.Marr, Hyperthreading technology in the netburst microarchitecture, *IEEE Micro,* vol 23, no 2, Mar.-Apr. 2003, pp.56-65.

[KEP+05] G. Kucuk, O. Ergin, D. Ponomarev, K. Ghose. Reducing power dissipation of register alias tables in high-performance processors. *IEE Proceedings on Computers and Digital Techniques*, Volume 152, issue 6, pp.739-746. Nov. 2005.

[KuHi09] R. Kumar, G. Hinton A family of 45nm IA Processors. *International Solid-State Circuits Conference* (ISSCC 09), Feb. 8-12, 2009. San Francisco (USA).

[LMG04] M. Lipasti, B. Mestan and E. Gunadi, Physical Register Inlining. *In Proceedings of the 31st Annual international Symposium on Computer Architecture* (ISCA 04), pp. 325-335. München, Germany, June 19-23, 2004.

[Lip92] J.S. Liptay. Design of the IBM Enterprise System/9000 high-end processor. *IBM Journal of Research and Development*, 36, 4, 713. 1992.

[LLM88] M. Litzkow, M. Livny, and M. Mutka. Condor - A Hunter of Idle Workstations. *Proceedings of the 8th International Conference of Distributed Computing Systems*, pages 104-111, June, 1988.

[LPE+99] J. Lo, S. Parekh, S. Eggers, H. Levy, D. Tullsen. Software-Directed Register Deallocation for Simultaneous Multithreaded Processors. *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 9, pp. 922-933, September, 1999.

[LoGa95] L. Lozano, G. Gao. Exploiting short-lived variables in superscalar processors. *Proceedings of the 28th Annual International Symposium on Microarchitecture* (MICRO 95), pp. 293-302. 1995.

[LFM+01] K.Luo, M.Franklin, S.Mukherjee and A.Seznec. Boosting SMT Performance by Speculation Control. *IPDPS*, pp. 9-16, April 2001.

[LGF01] K.Luo, J.Gummaraju and M.Franklin. Balancing throughput and fairness in SMT processors. *ISPASS*, Nov. 2001.

[MBH+02] D. T. Marr, F. Binns, D. L. Hill, G. Hinton, D. A. Koufaty, J. A. Miller, and M. Upton. Hyper-threading technology architecture and microarchitecture. *Intel Technology Journal*, 6(1):4–15, Feb. 2002.

[MRF97] M. Martin, A. Roth, C. Fisher. Exploiting Dead Value Information. *Proceedings of the 30th Annual International Symposium on Microarchitecture* (MICRO 97), pp. 125-135. 1997.

[MRH+02] J. Martinez, J. Renau, M. Huang, M. Prvulovich, J. Torrellas, "Cherry: Checkpointed Early Resource Recycling in Out-of-order Microprocessors", *Proc. 35nd Int'l Symp. Microarchitecture (MICRO 02)*, Nov. 2002, pp. 3-14.

[MGV+99] T. Monreal, A. González, M. Valero, J. González, and V. Viñals, Delaying Physical Register Allocation Through Virtual-Physical Registers. *Proceedings of the 32nd International Symposium on Microarchitecture* (MICRO 99), pp. 186-192, November 1999.

[MGV+00a] T. Monreal, A. González, M. Valero, J. González, and V. Viñals, Dynamic Register Renaming Through Virtual-Physical Registers. *In The Journal of Instruction-Level Parallelism*, vol. 2, May 2000. http://www.jilp.org/vol2.

[MVG+02] T. Monreal, V. Viñals, A. González and M. Valero. Hardware Schemes for Early Register Release. *Proc. Int'l Conf. Parallel Processing* (ICPP 02), Aug. 2002, pp. 5-13.

[MVG+04] T. Monreal, V. Viñals, J. González, A. González and M. Valero, "Late Allocation and Early Release of Physical Registers", *IEEE Transactions on Computers*, vol. 53, no. 10, Oct. 2004, pp. 1244-59.

[MoSs97] A. Moshovos and G. Sohi. Streamlining inter-operation memory communication via data dependence prediction. *In Proceedings of the 30th Annual ACM/IEEE international Symposium on Microarchitecture* (MICRO 97), pp. 235-245. North Carolina (United States), December 01-03, 1997.

[Mos03] A. Moshovos. Checkpointing Alternatives for High Performance, Power-Aware Processors. IEEE International Symposium on Low Power Electronic and Design (ISLPED 03), pp. 318 -321, Aug. 2003.

[MPV93] M. Moudgill, K. Pingali and S. Vassiliadis. Register Renaming and Dynamic Speculation: an Alternative Approach. *Proc. 26th Int'l Symp. Microarchitecture* (MICRO 93), Nov. 1993, pp. 202-213.

[MDM+95] S.Mutoh, T. Douseki, Y. Matsuya, T. Aoki, S. Shigematsu, and J. Yamada. 1-V power supply high-speed digital circuit technology with multithreshold-voltage CMOS. *IEEE Journal of Solid-State Circuits*, 1995, vol. 30, n.8, pp. 847-854.

[OBM+05] D. Oehmke, N. Binkert, T. Mudge and S. Reinhardt. How to Fake 1000 Registers. *In Proceedings of the 38th Annual IEEE/ACM international Symposium on Microarchitecture* (MICRO 05), pp. 7-18. Barcelona, Spain, November 12-16, 2005.

[PPV02] I. Park, M. D. Powell, and T. N. Vijaykumar. Reducing register ports for higher speed and lower energy. In MICRO-35, Istanbul, Turkey, November 2002.

[PJS96] S. Palacharla, N. P. Jouppi, and J. E. Smith. Quantifying the complexity of superscalar processors. Technical Report CSTR-1996-1328, Computer Sciences Dept., University of Wisconsin-Madison, 1996.

[PJS97] A.S. Palacharla, N.P. Jouppi, and J.E. Smith, Complexity-Effective Superscalar Processors. *Proceedings of the 24th International Symposium on Computer Architecture* (ISCA 97), pp. 206-218, June 1997.

[PGC+04] M. Pericàs, R. Gonzalez, A. Cristal, A. Veidenbaum and M. Valero. An Optimized Front-End Physical Register File with Banking and Writeback Filtering. *Workshop on Power Aware Computer Systems*. December 2004.

[PBR02] V. Petric, A. Bracy and A. Roth. Three extensions to register integration. *In Proceedings of the 35th Annual ACM/IEEE International Symposium on Microarchitecture* (MICRO 02). Istanbul, Turkey, November 18 - 22, 2002.

[PSR05] V. Petric, T. Sha and A. Roth. RENO: a rename-based instruction optimizer. *In Proceedings of the 32nd Annual International Symposium on Computer Architecture* (ISCA 05), pp. 98-109. Madison (USA), June 4-8, 2005.

[PKE+03] D. Ponomarev, G. Kucuk, O. Ergin, K. Ghose. Reducing Datapath energy Through the Isolation of Short-Lived Operands. *Proceedings of the 12th International Conference on Parallel Architectures and Compilation Techniques* (PACT 03), pp. 258.

[PGR+01] M. Postiff, D. Greene, S. Raasch and T. Mudge. Integrating superscalar processor components to implement register caching. *In Proceedings of the 15th international Conference on Supercomputing* (ISCS 01). Sorrento, Italia, 2001.

[PYF+00] M.Powell, S. Yang, B. Falsafi, K. Roy, T. N. Vijaykumar. Gated-Vdd: a circuit technique to reduce leakage in deep-submicron cache memories. *In Proceedings of the 2000 International Symposium on Low Power Electronics and Design (*ISLPED 00).

[RaRe03] S.E.Raasch, S.K.Reinhardt, The impact of resource partitioning on SMT processors, *PACT*, pp. 15-25, Sept. 2003.

[Rei05] G. Reinman. Using an operand file to save energy and to decouple commit resources. IEE Proceedings: Computers and digital techniques, ISSN 1350-2387, Vol. 152, Nº 5, 2005 , pags. 666-678.

[RDK+00] S. Rixner, W.J. Dally, B. Khailany, P. Mattson, U.J. Kapasi, and J.D. Owens, Register Organization for Media Processing. P*roceedings of the 6th International Symposium on High-Performance Computer Architecture* (HPCA 00), pp. 375-386, January 2000.

[Rot07] A. Roth, Physical Register Reference Counting. *IEEE Computer Architecture. Letters*, vol. 7, issue 1, pp. 9-12. Jan. 2008.

[RTM+09] S. Rusu, S. Tam, H. Muljono, J. Stinson, D. Ayers, J. Chang, R. Varada, M. Ratta, S. Kottapalli. A 45nm 8-Core Enterprise Xeon® Processor. *International Solid-State Circuits Conference* (ISSCC 09), Feb. 8-12, 2009. San Francisco (USA)..

[SAM+07] E. Safi, P. Akl, A. Moshovos, A. Veneris, and A. Arapoyianni, On the latency, energy and area of checkpointed, superscalar register alias tables. *In Proceedings of the 2007 international Symposium on Low Power Electronics and Design* (ISLPED 07), pp. 379-382. Portland (USA), August 27-29, 2007.

[San06] R. Sangireddy. Reducing Rename Logic Complexity for High-Speed and Low-Power Front-End Architectures. IEEE Transactions on Computers, vol. 55, issue 6, pp- 672-685. June 2006.

[SRG02] G. Savransky, R. Ronen, A. Gonzalez. Lazy Retirement: A Power Aware Register Management Mechanism. WCED 2002.

[STR02] A. Seznec, E. Toullec, and O. Rochecouste. Register write specialization register read specialization: A path to complexity-effective wide-issue superscalar processors. In *MICRO-35*, Istanbul, Turkey, November 2002.

[ShPo07] J. Sharkey and D. Ponomarev. An L2-miss-driven early register deallocation for SMT processors. *In Proceedings of the 21st Annual international Conference on Supercomputing* (ICS 07), pp. 138-147. Seattle (USA), June 17-21, 2007.

[ShLi04] J. P. Shen, M. H. Lipasti. *Modern Processor Design.* McGraw-Hill, First Edition, 2004.

[SPH+02] T. Sherwood, E. Perelman, G. Hamerly and B. Calder. Automatically Characterizing Large Scale Program Behavior. *Proc. 10th Int'l Conf. Architectural Support for Programing Languages and Operating Systems* (ASPLOS 02), pp. 45-57, Oct. 2002.

[Sim00] D. Sima, The Design Space of Register Renaming Techniques. *IEEE Micro*, vol. 20, no. 5, pp. 70-83, September-October 2000.

[Sim08] D. Sima, Register Renaming Techniques. *Digital Systems and Aplications (segunda edición)*. Capítulo 2. Editor: V. Oklobdzija. CRC Press. ISBN: 978-0-8493-8619-0, 2008.

[SmPl88] J.E. Smith, and A.R. Pleszkun. Implementing Precise Interrupts in Pipelined Processors. *IEEE Transactions on Computers*, vol. 37, no. 5, pp. 562-573, May 1988.

[Smi88] J.E. Smith. Characterizing computer performance with a single number. Communications of the ACM, vol 31, issue 10, pp. 1202-1206. October 1988.

[SBV95] G. S. Sohi, S. Breach, and T. N. Vijaykumar. Multiscalar processors. In ISCA-22, 1995.

[SRA+04] S. Srinivasan, R. Rajwar, H. Akkary, A. Gandhi and M. Upton. Continual flow pipelines. *In Proceedings of the 11th international Conference on Architectural Support For Programming Languages and Operating Systems* (ASPLOS 04), pp. 107-119. Boston, (USA), October 07-13, 2004.

[StRu03] J.Stinton et S. Rusu. A 1.5GHz Third Generation Itanium® 2 Processor. *In Proceedings of Design Automation Conference* (DAC 03), pp. 706-709. 2-6 June 2003.

[SPL08] S.Subramaniam, M.Prvulovic and G. Loh. PEEP: Exploiting Predictability of Memory Dependences in SMT Processors. *HPCA*, pp. 137-148, Feb. 2008.

[TDF+02] J. M. Tendler, J. S. Dodson, J. S. Fields, Jr., H. Le, and B. Sinharoy, POWER4 System Microarchitecture, *IBM J. Res.& Dev. 46, No. 1, 5–25*, January 2002.

[Tho64] J.E. Thornton. Parallel operation in the Control Data 6600. *In Proc. AFIPS Fall Joint Computer Conf.* pp 33-40, 1964.

[Tom67] R.M. Tomasulo. An Efficient Algorithm for Exploiting Multiple Arithmetic Units. IBM Journal of Research and Development, vol 11, nº1, pp 25. 1967.

[TIV+03] E. Torres, P. Ibáñez, V. Viñals and J.M. Llabería. Counteracting Bank Misprediction in Sliced First-Level Caches. *In proc. 9th Int'l Conf. on Parallel and Distributed Computing (Euro-Par 03)*, pp. 586-596. Aug. 2003.

[Tor05] E. Torres. Alternativas de Diseño en Memoria Cache de Primer Nivel Multibanco. Tesis Doctoral. Dpto. Informática e Ingeniería de Sistemas, Universidad de Zaragoza. Junio 2005.

[TNN+04] L. Tran, N. Nelson, F. Ngai, S. Dropsho and M. Huang. Dynamically reducing pressure on the physical register file through simple register sharing. *In Proceedings of the 2004 IEEE international Symposium on Performance Analysis of Systems and Software* (ISPASS 04), pp. 78-87. March 10-12, Austin (USA), 2004.

[TsAs03] J. Tseng, and K. Asanovic. Banked multiported register files for high-frequency superscalar microprocessors. In Proceedings of the 30th Annual International Symposium on Computer Architecture (ISCA 03), pp. 62. 2003.

[TEL95] D.M. Tullsen, S. Eggers, and H. Levy, Simultaneous Multithreading: Maximizing On-Chip Parallelism. *Proceedings of the 22nd International Symposium on Computer Architecture* (ISCA 95), pp. 392-403, June 1995.

[TEE+96] D.Tullsen, S.Eggers, J.Emer, H.Levy, J.Lo, and R. Stamm. Exploiting choice: Instruction fetch and issue on an implementable simultaneous multithreading processor. *ISCA*, pp. 191–202, Apr. 1996.

[TuSe99] D. Tullsen and J. Seng. Storageless value prediction using prior register values. *In Proceedings of the 26th Annual International Symposium on Computer Architecture* (ISCA 99), pp.270-279. Atlanta (USA), May 01-04, 1999.

[TuBr01] D.Tullsen and J.Brown. Handling long-latency loads in a simultaneous multithreaded processor. *Int'l Symp. on Microarchitecture (MICRO)*, pp. 318–327. Dec. 2001.

[VCP+07] J.Vera, F.Cazorla, A.Pajuelo, O.Santana, E.Fernandez and M.Valero. FAME: FAirly MEasuring Multithreaded Architectures. *PACT*. September 2007.

[Vri00] H. Vries. An analysis of the newly presented Pentium 4 core, *http://www.chip-architect.com/*. Consultada el 5 de febrero de 2009.

[Vri03] H. Vries. Understanding the detailed Architecture of AMD's 64 bit Core, *http://www.chip-architect.com/*. Consultada el 5 de febrero de 2009.

[WaBa96] S. Wallace and N. Bagherzadeh. A Scalable Register File Architecture for Dynamically Scheduled Processors. *Proceedings of the 5th International Conference on Parallel Architectures and Compilation Techniques* (PACT 96), pp. 179-184, October 1996.

[Yea96] K.C. Yeager, The MIPS R10000 Superscalar Microprocessor. *in IEEE Micro*, vol. 16, no. 2, pp. 28-40, April 1996.

[YER+99] A. Yoaz, M. Erez, R. Ronen, S. Jourdan. Speculation Techniques for Improving Load Related Instruction Scheduling. ISCA 1999: pp. 42-53.

[ZeGh06] H. Zeng, and K. Ghose. Register file caching for energy efficiency. *In Proceedings of the 2006 international Symposium on Low Power Electronics and Design* (ISLPED 06), pp. 244-249. Tegernsee (Germany), October 04-06, 2006.

[ZyKo01] V. V. Zyuban and P. M. Kogge. Inherently lower-power high-performance superscalar architectures. *IEEE Trans. on Computers*, 50(3):268–285, March 2001.