Universidad Zaragoza

Trabajo Fin de Grado

Caracterización del Envejecimiento en los Bancos de Registros de Microprocesadores x86

Autor

Miguel Lasaosa Isac

Directores

Alejandro Valero Bresó

Rubén Gran Tejero

ESCUELA DE INGENIERÍA Y ARQUITECTURA
2020

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Miguel Lasaosa Isac,

con nº de DNI 18063859-G en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Caracterizaci on del Envejecimiento en los Bancos deRegistros de Microprocesadores x86

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 22/09/2020

Fdo: Miguel Lasaosa Isac

AGRADECIMIENTOS

A Alejandro y a Rubén, porque gracias a sus consejos, disponibilidad continuada y confianza he podido llevar a buen término este proyecto.

A mi familia, por apoyarme durante la realización de este trabajo.

RESUMEN

La continua miniaturización del transistor compromete la fiabilidad de los circuitos digitales. Entre los efectos que degradan el transistor a lo largo del tiempo, destacan los efectos *Bias Temperature Instability* (BTI) y *Hot Carrier Injection* (HCI). Ambos efectos aumentan el voltaje umbral de los transistores, lo cual puede provocar fallos en la ejecución de las aplicaciones, acortando el tiempo de vida útil de los microprocesadores.

Los componentes de un procesador más susceptibles a los efectos BTI y HCI son aquellos que integran un mayor número de transistores, se utilizan continuamente y con frecuencia, y tienen un impacto directo en el rendimiento del sistema. En este trabajo, se caracterizan ambos efectos de envejecimiento sobre el banco de registros de un procesador x86 atendiendo a los valores almacenados en esta estructura de memoria. Para ello, se instrumenta un simulador ciclo-a-ciclo que permite emular el comportamiento de un procesador Skylake y extraer estadísticas de uso relacionadas con la degradación de los transistores a partir de la ejecución de un conjunto de aplicaciones científicas.

Los resultados experimentales muestran diferentes patrones de degradación dependiendo de la aplicación ejecutada, aunque se pueden extraer características comunes que contribuyen a acelerar la degradación del banco de registros. Por ejemplo, la mayoría de aplicaciones almacenan una gran cantidad de bytes nulos y direcciones de memoria que afectan a celdas específicas de los registros. A partir de un modelo analítico, se comprueba que los transistores de estas celdas sufren el mayor aumento de voltaje umbral al cabo de un uso continuado durante tres años.

El estudio de caracterización presentado en este trabajo permitirá diseñar mecanismos arquitectónicos capaces de mitigar o distribuir el desgaste de los transistores de manera homogénea por todo el banco de registros, alargando sustancialmente el tiempo de vida del procesador.

Índice

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

Este primer capítulo introduce al lector el contexto, motivación, objetivos, metodología y tareas principales que componen el presente trabajo, así como una descripción de la estructura general del documento.

1.1. Motivación

Los avances en la industria del semiconductor están dictados por una continua demanda de mayor rendimiento por parte de los usuarios. A lo largo de los años, esta demanda se ha satisfecho con la continua miniaturización del transistor y consecuente cumplimiento de la Ley de Moore. Esta ley predice que el número transistores por unidad de superficie se dobla aproximadamente cada dos años, lo que permite que sucesivas generaciones de microprocesadores contengan una mayor funcionalidad.

Sin embargo, la continua miniaturización del transistor no está exenta de problemas. Por un lado, con el fin de la Ley de Dennard hacia 2005, la densidad de potencia en el chip ya no se mantiene constante en sucesivas generaciones de procesadores, lo que ha supuesto la aparición de nuevos paradigmas como los procesadores multinúcleo, el uso de procesadores gráficos de propósito general o la consolidación de técnicas como *Dynamic Voltage and Frequency Scaling* (DVFS), entre otras. Por otro lado, la continua reducción del transistor también compromete la fiabilidad del mismo. En este sentido, la fiabilidad se ha convertido en un aspecto crítico de diseño para los fabricantes de chips debido a que la cuota de mercado, incluyendo procesadores utilizados en dispositivos como servidores, CPUs, tablets y smartphones, depende en gran medida de la fiabilidad del producto.

Existe una gran variedad de efectos que comprometen la fiabilidad de los procesadores actuales. Entre ellos, destacan aquellos que provocan un continuo envejecimiento o desgaste del transistor debido al uso del mismo. Estos efectos pueden acortar el tiempo de vida útil de los procesadores hasta el punto de provocar fallos en

la ejecución de las aplicaciones.

Los componentes de un procesador que presentan una mayor probabilidad de fallo por envejecimiento son aquellos que, por un lado, contienen un mayor número de transistores, y por otro, se utilizan con mayor frecuencia. En este sentido, la jerarquía de memoria *Static Random-Access Memory* (SRAM) *on chip* en general, y los bancos de registros y la cache de primer nivel (L1) en particular, presentan una mayor susceptibilidad a sufrir fallos por envejecimiento frente a otros componentes. Además, el banco de registros y las cache L1 se consideran unas estructuras críticas debido a que tienen una incidencia directa sobre el rendimiento del sistema. Por esta razón, resulta imprescindible mitigar los efectos de envejecimiento en estas estructuras.

Entre los efectos de envejecimiento destacan los conocidos como *Bias Temperature Instability* (BTI) y *Hot Carrier Injection* (HCI). BTI afecta a los transistores que implementan celdas de memoria cuando estas almacenan un valor lógico '0' o '1' durante largos periodos de tiempo. Por su parte, HCI afecta a los transistores cuando el valor almacenado en una celda transita de '0' a '1' y viceversa. Por lo tanto, para caracterizar los efectos BTI y HCI en una memoria del procesador, es importante conocer con exactitud los datos almacenados a lo largo del tiempo en cada celda para así extraer conclusiones acerca de la degradación que estas sufren y posteriormente diseñar mecanismos arquitectónicos que combatan el desgaste.

1.2. Objetivos y Alcance

El objetivo principal de este trabajo es caracterizar el envejecimiento que sufre el banco de registros de un procesador x86 al ejecutar aplicaciones científicas reales. Para ello, se tendrá en cuenta no sólo los valores y tipos de datos que almacenan los registros, sino también la estrategia de renombre de registros presente en procesadores modernos.

Como caso de uso se considerará un procesador Intel Skylake con arquitectura x86. Las características y componentes principales de este procesador se modelarán con el simulador ciclo-a-ciclo Multi2sim, ampliamente utilizado tanto en la industria como en la academia para validar diseños de procesador. Más allá de modelar las características y componentes principales del procesador, se instrumentará el código fuente de Multi2Sim para obtener las estadísticas necesarias acerca del uso de los registros, incluyendo el número de transiciones de valor lógico o *flips* en las celdas, ciclos almacenando un valor lógico '0' o '1', tiempo total de ejecución en ciclos, etcétera.

Las estadísticas mencionadas se obtendrán mediante la ejecución de aplicaciones científicas reales (SPEC CPU 2006) sobre el procesador modelado en Multi2Sim. Finalmente, las estadísticas obtenidas se introducirán en un modelo analítico a partir

del cual se cuantificará el envejecimiento inducido por cada aplicación.

Para lograr estos objetivos se llevarán a cabo las siguientes tareas:

- Tarea 1: Estudio del renombre y del repertorio de registros arquitectónicos de la arquitectura x86.
- Tarea 2: Familiarización con el entorno de simulación: puesta a punto del entorno, lanzamiento de experimentos y estudio de la implementación de la estrategia de renombre y del uso de registros en el simulador Multi2Sim.
- Tarea 3: Instrumentación de Multi2Sim para la obtención de estadísticas y análisis de resultados.
- Tarea 4: Comprensión del modelo analítico de envejecimiento, cuantificación de los efectos BTI y HCI en el banco de registros y análisis de resultados.
- Tarea 5: Propuesta de conclusiones y presentación del trabajo.

1.3. Estructura del Documento

El resto del documento se organiza como sigue. El Capítulo 2 presenta una serie de conceptos y definiciones para una mejor comprensión del trabajo. El Capítulo 3 recoge detalles de la metodología que se han tenido en cuenta y que permiten validar el estudio y los resultados obtenidos. En el Capítulo 4 se muestran y se describen un conjunto de resultados que suponen la caracterización del envejecimiento en el banco de registros x86. Finalmente, en el Capítulo 5 se exponen las conclusiones más relevantes, la dedicación al proyecto en horas y una serie de líneas de trabajo futuro.

Capítulo 2

Antecedentes

Este capítulo expone una serie de conceptos y definiciones para una mejor comprensión del trabajo como es el caso de los efectos BTI y HCI en una celda de memoria SRAM, así como algunos detalles de la arquitectura x86 como el conjunto de registros lógicos y la técnica de renombre de registros.

2.1. Efectos BTI y HCI en una Celda SRAM

BTI y HCI son dos de los principales efectos que limitan el tiempo de vida de los transistores de un circuito digital. En concreto, ambos efectos aumentan el voltaje umbral (V_{th}) del transistor a lo largo del tiempo. Esta degradación de voltaje provoca que el transistor sufra una conmutación más lenta, lo cual a su vez puede provocar una violación del tiempo y fallos de operación si el camino crítico sobrepasa el tiempo de ciclo del procesador. Más aún, si V_{th} sobrepasa la tensión de alimentación del circuito (V_{dd}) se produce un fallo de operación permanente en el transistor. Este problema resulta especialmente patente cuando se reduce V_{dd} de manera agresiva a efectos de mitigar el consumo energético del circuito.

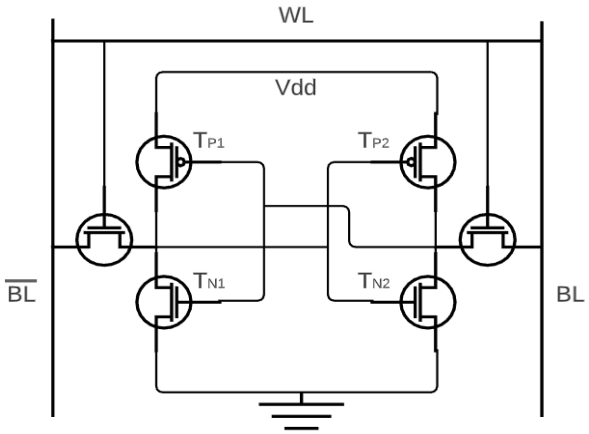


Figura 2.1: Implementación de una celda 6T SRAM.

Como se muestra en la Figura 2.1, una celda de memoria SRAM consta de 6 transistores. Los 4 transistores etiquetados forman un bucle de inversores que contiene el valor lógico almacenado. Los dos transistores nMOS restantes, denominados transistores de paso, están controlados por la señal *wordline* (WL) y permiten operaciones de lectura y escritura en la celda a través de la línea *bitline* (BL) y su complementaria $\overline{BL}$.

Cuando la celda SRAM se encuentra bajo lo que denominamos un *duty cycle* '0', es decir, cuando la celda almacena de manera estable el valor lógico '0', la pareja de transistores pMOS T_{P1} y nMOS T_{N2} sufren los efectos *Negative* BTI (NBTI) y *Positive* BTI (PBTI), respectivamente. Por el contrario, si la celda se encuentra en un *duty cycle* '1', la pareja de transistores T_{P2} y T_{N1} estarán afectados por NBTI y PBTI, respectivamente.

Es importante destacar que el efecto NBTI resulta más perjudicial que el efecto PBTI. En otras palabras, los transistores pMOS son más sensibles a la degradación (sufren un mayor estrés y aumento de V_{th}) cuando en la puerta se encuentra un '0' frente a los transistores nMOS cuando en la puerta se encuentra un '1'. Finalmente, nótese que el desgaste inducido por cada tipo de *duty cycle* es complementario, es decir, para un *duty cycle* dado, el par de transistores que no están sometidos a estrés se recuperan parcialmente de la degradación de V_{th}. Así pues, los *duty cycles* ideales de una determinada celda son del 50 % a '0' y 50 % a '1', igualando de esta manera el desgaste inducido por el efecto BTI, puesto que así se evita que una pareja de transistores envejezca más rápido que la otra. Sin embargo, esta situación rara vez tiene lugar en las celdas de memoria de un procesador.

Por su parte, el efecto HCI afecta por igual a los 4 transistores que forman el bucle de inversores de las celdas SRAM. Sin embargo, su incidencia es más limitada frente a NBTI porque tan sólo afecta a los transistores durante el instante de tiempo en el que la celda cambia su valor lógico almacenado de '0' a '1' y viceversa. A estos cambios de valor lógico nos referiremos como *flips* a lo largo del documento. Para minimizar el efecto HCI, los flips deberían distribuirse uniformemente entre todas las celdas de memoria de un array, lo cual tampoco es habitual en las celdas de memoria de un procesador.

En resumen, de acuerdo con la implementación de una celda SRAM, sus transistores en conjunto se encuentran permanentemente envejeciendo, ya sea manteniendo un valor lógico a lo largo del tiempo como transitando de valor lógico.

2.1.1. Degradación de V_{th}

El envejecimiento máximo que sufre una celda de memoria durante un periodo de tiempo, en cuanto a degradación de V_{th} se refiere (dV_{th}), se cuantifica como el mayor aumento de tensión umbral (en mV) entre los cuatro transistores que implementan el bucle de inversores de la celda. El valor total de dV_{th} por cada transistor se calcula como la suma de dV_{th}_BTI y dV_{th}_HCI, siendo cada uno de estos términos la contribución de degradación separada por los efectos BTI y HCI. La Ecuación 2.1 permite obtener el primer término.

$$\begin{aligned} dV_{th}\_BTI = A_{BTI} \times t_{ox} \times \sqrt{C_{ox} \times (V_{dd} - V_{t_0})} \times (1 - \frac{V_{ds}}{\alpha \times (V_{dd} - V_{t_0})}) \times \\ e^{\frac{V_{dd}}{t_{ox} \times E_{BTI}} - \frac{Ea}{k \times T}} \times {t_{stress}}^{0,25} \times (1 - \sqrt{etha \times \frac{t_{rec}}{t_{stress} + t_{rec}}}) \end{aligned} \tag{2.1}$$

Los parámetros t_{stress} y t_{rec} se refieren al tiempo de estrés y de recuperación, respectivamente, del transistor. Por ejemplo, en el caso del transistor T_{P1}, t_{stress} y t_{rec} hacen referencia a los *duty cycle* '0' y '1' de la celda, respectivamente. A_{BTI} es una constante que depende si el tipo de efecto es NBTI o PBTI sobre el transistor, t_{ox} es el grosor del óxido (0,65 nm), C_{ox} es la capacitancia de la puerta por unidad de área, α es el factor de actividad, V_{t_0} es el V_{th} inicial y T es la temperatura. Para más detalles se refiere al lector a [2].

$$dV_{th}\_HCI = A_{HCI} \times \alpha \times f \times e^{\frac{V_{dd} - V_{t_0}}{t_{ox} \times E_{HCI}}} \times \sqrt{t} \tag{2.2}$$

La Ecuación 2.2 muestra la degradación debida al efecto HCI, donde t es el tiempo durante el cual se producen *flips* en la celda, f es la frecuencia y A_{HCI} es una constante de envejecimiento de tipo HCI. El resto de parámetros representan lo mismo que en la ecuación anterior. Para más detalles se refiere al lector a [3].

Normalmente, dV_{th} se cuantifica tras un periodo de tiempo de tres años de uso de las celdas. Para simular este periodo de tiempo, se asume la ejecución repetida en el tiempo de ciertas aplicaciones científicas (véase la Sección 3.4) hasta cubrir el periodo de tres años [4].

2.1.2. Trabajos Relacionados

La mitigación de los efectos de envejecimiento, en especial el efecto NBTI, ha suscitado un creciente interés en los últimos 15 años. Algunos trabajos proponen técnicas de balanceo en las escrituras de los registros, incluyendo la inversión periódica

de los datos a escribir[1], algoritmos de compresión de los datos, o esquemas alternativos de decodificación de los registros. Estos trabajos se centran en arquitecturas como MIPS [5], SPARC [6, 7] o Alpha [8]. Penelope [9] es un diseño de procesador x86 que mitiga el efecto NBTI tanto en bloques combinaciones como secuenciales, incluyendo el banco de registros. Sin embargo, el presente trabajo muestra una caracterización de los efectos de envejecimiento más detallada mediante el estudio de aplicaciones individuales.

Otras propuestas han atacado los efectos NBTI y HCI en las memorias cache del procesador mediante diseños alternativos de celdas de memoria, la reducción de V_{dd} o el apagado de regiones de memoria infrautilizadas [10, 11, 12, 13], así como en los bancos de registros de las arquitecturas GPU [14, 15, 16].

2.2. Arquitectura x86

La familia x86 reagrupa los microprocesadores compatibles con el juego de instrucciones Intel 8086. Por tanto, x86 representa a ese conjunto de instrucciones, siendo también una denominación genérica dada a los correspondientes microprocesadores.

La arquitectura x86 cuenta con varios tipos de registros que son visibles al programador: registros de propósito general, registros índice, registros apuntadores, registros flags y registros de segmento.

2.2.1. Registros Lógicos Visibles al Programador

– **Registros de propósito general:** registros de propósito general son el EAX, EBX, ECX y EDX, cada uno de 32 bits. Se puede acceder únicamente a los 16 bits menos significativos con los registros llamados AX, BX, CX y DX, y a su vez, cada uno de ellos se divide en dos registros de 8 bits, llamados AH y AL, BH y BL, CH y CL y DH y DL, H significando High (parte alta del registro) y L significando Low (parte baja del registro) correspondiente a 16 bits (Figura 2.2). Aparte del uso general de los registros para hacer cálculos aritméticos y lógicos, existen instrucciones que usan estos registros con un uso particular especializado, como se indica a continuación:

- **Registro EAX:** el registro AX es el registro acumulador, es utilizado para operaciones que implican entrada/salida, multiplicación y división (estas dos últimas en conjunto con el registro EDX).

[1]Se refiere a la función que asocia el valor lógico '0' con el valor de tensión *Gnd* y valor lógico '1' con el valor de tensión V_{dd}. En caso de inversión se asocia el valor '0' con V_{dd} y valor '1' con *Gnd*.

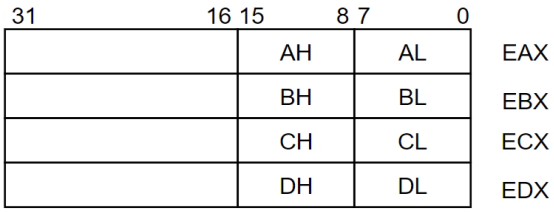


Figura 2.2: Registros de propósito general.

- **Registro EBX:** es el registro base y el único de propósito general que puede ser un índice para direccionamiento indexado.
- **Registro ECX:** es conocido como el registro contador. Puede contener un valor para controlar el número de veces que un bucle se repite o un valor para desplazamiento de bits.
- **Registro EDX:** es el registro de datos. En algunas operaciones se indica mediante este registro el número de puerto de entrada/salida, y en las operaciones de multiplicación y división de 16 bits se utiliza junto con el acumulador EAX.

– **Registros índice:** los registros ESI y EDI están disponibles para direccionamiento indexado y para operaciones de cadenas de caracteres.

– **Registros apuntadores:** los registros ESP (apuntador de pila) y EBP (apuntador base) están asociados con el registro SS y permiten al sistema acceder a datos en el segmento de la pila.

– **Registro de flags:** es un registro de 32 bits, de los cuales nueve sirven para indicar el estado actual de la máquina y el resultado del procesamiento. Muchas instrucciones aritméticas y de comparación cambian el estado de los flags y apoyándose en ellos se pueden tomar decisiones para determinar la acción subsiguiente. Los bits de los flags son los siguientes: OF (desbordamiento), DF (dirección), IF (interrupción), TF (captura), SF (signo), ZF (cero), AF (acarreo auxiliar), PF (paridad) y CF (acarreo).

– **Registros de segmento:** se trata de registros de 16 bits. Definen áreas dentro del espacio de direcciones de la memoria del procesador. Estas áreas pueden solaparse total o parcialmente. No es posible acceder a una posición de memoria no definida por algún segmento: si es preciso, habrá que cambiar de segmento. Existen cuatro registros de segmento: CS, DS, SS y ES.

2.2.2. Técnica de Renombre

En los procesadores con varios caminos de ejecución de latencia variable, las dependencias entre instrucciones del flujo de programa puede requerir la parada de procesamiento para asegurar una correcta ejecución. El inevitable reuso de los registros lógicos de la arquitectura crea dos tipos de dependencias: dependencias verdaderas provocadas por la necesidad de comunicar valores entre una instrucción productora y una consumidora a través de un registro, y falsas dependencias en las que no debe comunicarse valor alguno entre dos instrucciones pero reutilizan el mismo registro lógico como operando. Estas últimas provocan riesgos de datos que pueden exigir la parada del procesamiento: WAW (*write after write*) y WAR (*write after read*).

Como alternativa a la parada del procesador, se pueden renombrar [17] los operandos de las instrucciones y eliminar las falsas dependencias. Para el renombre, existe un conjunto de registros físicos (no visible al programador) más numeroso que el conjunto de registros lógicos, y cada registro lógico del flujo de programa se renombra en uno de estos registros físicos. En definitiva, se almacena cada versión de un registro lógico en un registro físico, a las instrucciones consumidoras (dependencias verdaderas) se les anota en el registro físico del cual deben leer y este registro no se libera mientras queden instrucciones consumidoras que lo vayan a leer. Para que este mecanismo funcione correctamente es fundamental el uso de una tabla de renombre (RAT), situada en una etapa previa a la lectura del banco de registros. La tabla RAT contiene el identificador del registro físico, asignado a la última versión de cada registro lógico y se modifica cada vez que se crea una nueva versión del registro lógico, tal como se aprecia en la Figura 2.3. Los fabricantes no especifican cual es la política de renombre que utilizan, aunque normalmente se presupone que es una política cíclica de tipo Round-Robin gestionada mediante una cola FIFO [8].

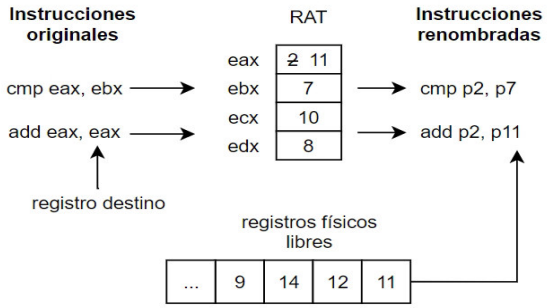


Figura 2.3: Ejemplo de renombre de registros.

Capítulo 3

Metodología

Este capítulo recoge detalles de la metodología que se han tenido en cuenta y que permiten validar el estudio y los resultados obtenidos en este trabajo, incluyendo una breve descripción de la herramienta de simulación de procesadores, el procesador modelado, las modificaciones más importantes realizadas sobre el simulador y las aplicaciones científicas utilizadas.

3.1. Multi2Sim

En este trabajo se ha instrumentado el simulador Multi2Sim para obtener estadísticas de uso del banco de registros y caracterizar el envejecimiento [18]. Multi2Sim es un software ampliamente utilizado tanto en la industria como en la academia para modelar y validar nuevos diseños de procesador. Escrito en C y para sistemas operativos Linux, Multi2Sim simula diferentes arquitecturas tanto de CPU como GPU, entre ellas x86. Permite ejecutar cualquier aplicación para distintos modelos de procesador y así obtener estadísticas de rendimiento.

La simulación de un programa se puede dividir en dos módulos principales: simulación funcional y simulación detallada. La simulación funcional se encarga de que el programa se ejecute de la misma forma como si se ejecutara de forma nativa en la máquina. Por su parte, la simulación detallada proporciona un modelo de las estructuras hardware del procesador. De esta manera, la simulación detallada es la que proporciona estadísticas acerca del rendimiento de la ejecución de una determinada aplicación en el procesador.

El código de Multi2Sim puede descargarse de su página web[1]. En este trabajo se ha utilizado la versión 5.0. Se ha configurado el simulador para modelar un procesador lo más semejante posible a un Intel Skylake. En este sentido, se han modificado parámetros como el número de registros físicos, la organización de la jerarquía de memoria o el

[1] www.multi2sim.org

tamaño del *reorder buffer* (ROB), entre otros, tal como resume la Tabla 3.1.

Caches L1	32 KB 4 vías 128 conjuntos 64 B × bloque	**Cache L2**	256 KB 4 vías 1024 conjuntos 64 B × bloque
Cache L3	2 MB 16 vías 2048 conjuntos 64 B × bloque	**# entradas ROB** **# entradas LSQ** **# regs enteros** **Frecuencia**	256 entradas 256 entradas 180 3000 MHz

Tabla 3.1: Configuración de los parámetros principales del procesador modelado.

3.2. Intel Skylake

Intel Skylake es un procesador fabricado con tecnología FinFet de 14 nm, diseñado para computadoras de sobremesa y dispositivos móviles. Skylake ofrece la marca de procesadores Intel Core i3, Core i5 y Core i7 de sexta generación. Es compatible con los sistemas operativos Windows, Linux, Chromiun y VxWorks.

En general, Skylake se basa en la microarquitectura anterior de Intel, Broadwell, pero incluye una interfaz más amplia y reforzada, un motor de ejecución más optimizado, entre otras mejoras. Intel diseñó Skylake para abarcar una amplia gama de dispositivos y aplicaciones con un gran énfasis en dispositivos móviles con modelos con un consumo desde 4,5 W hasta 100 W.

Para este trabajo se ha utilizado como referencia el core de Skylake. El core se puede dividir en tres partes principales como se puede apreciar en la Figura 3.1: *front end*, *execution engine* y *memory subsystem*.

– *Front end*: Su propósito es proveer de un flujo constante y suficiente de instrucciones para que el *execution engine* mantenga ocupadas todas sus unidades funcionales. Las principales tareas son la lectura del programa de memoria, la selección del camino a ejecutar y la decodificación de las instrucciones.

  Las instrucciones pueden seguir dos rutas: la ruta de la cache de μOPs o la ruta tradicional. En esta última las instrucciones se buscan en la cache de instrucciones L1, se almacenan en la cola de instrucciones y se decodifican en instrucciones de longitud fija. La otra alternativa es la ruta de cache de μOPs, mediante la cual una cache que contiene μOPs ya decodificadas recibe un acierto que permite que estas μOPs se envíen directamente a la cola de decodificación, evitando que sean almacenadas en la cola de instrucciones y que tengan que ser pre-decodificadas.

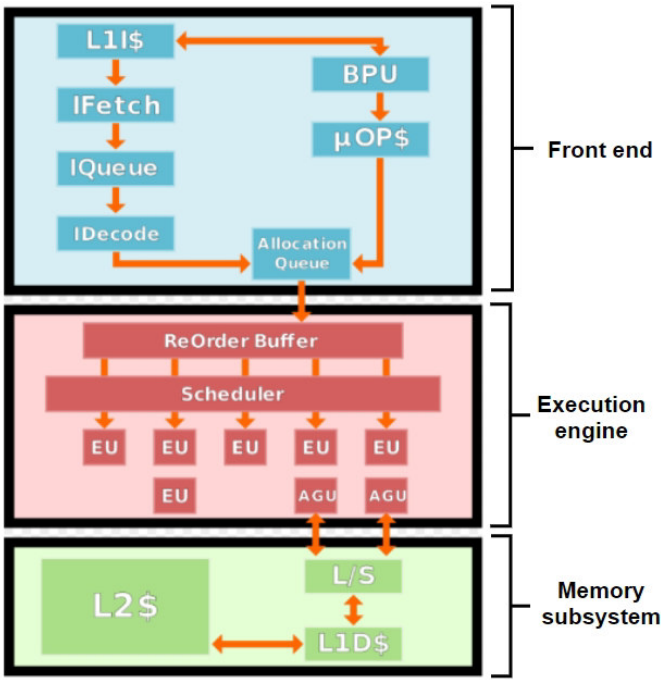


Figura 3.1: Ruta de datos de un núcleo de Skylake [1].

– *Execution engine*: En esta parte, las micro-operaciones se introducen en el ROB. En este punto, entre otras acciones, es donde traducen los registros lógicos de la instrucción a registros físicos, renombre de registro destino de la instrucción y la liberación de los registros sin usar. Skylake cuenta con 180 registros físicos para datos de tipo entero. Desde el ROB, las μOPs se envían a un *scheduler* unificado que tiene varios puertos de salida y cada uno de ellos está conectado a un conjunto de unidades diferentes de ejecución.

– *Memory subsystem*: Algunas μOPs como las *loads* y *stores* generan accesos a memoria. Estas micro-operaciones se envían a puertos dedicados, los cuales gestionan las operaciones de memoria asociadas. Skylake cuenta con caches de datos e instrucciones L1 separadas, cada una con 32 KB. También cuenta con una cache L2 de 256 KB privada y compartida por ambas caches L1.

3.3. Modificaciones del Simulador

Como se ha comentado en la Sección 3.1, Multi2sim ofrece una serie de estadísticas por defecto acerca del rendimiento del procesador para una determinada aplicación. Sin embargo, estas estadísticas no son suficientes para los objetivos de este trabajo, ya que no muestran todos los datos necesarios para determinar el envejecimiento de las celdas de memoria del banco de registros. Por esta razón, se realizan una serie de modificaciones en el código del simulador que nos permitan obtener las estadísticas de

duty cycle '0', *duty cycle* '1' y número de *flips*, descritas en la Sección 2.1, por cada celda de memoria del banco de registros.

El principal inconveniente que se encuentra un programador en Multi2Sim es que el simulador se divide en las dos partes funcional y detallada. La funcional se encarga de trabajar con los valores reales de cada aplicación, mientras que la detallada simula el comportamiento de los componentes hardware a lo largo del tiempo, es decir, modela detalladamente las etapas del procesador, renombra registros, tiene en cuenta las dependencias, el número de ciclos de ejecución, etcétera. Sin embargo, la simulación detallada no tiene en consideración el valor almacenado en los registros. Por esta razón, en este trabajo se ha instrumentado ambas partes del simulador para que la parte funcional transmita de forma coherente al simulador detallado los valores que maneja cada registro. Además, se han añadido una serie de estructuras al simulador detallado para no sólo almacenar estos valores sino también obtener las estadísticas específicas de uso de cada celda de memoria. El código fuente del simulador instrumentado se encuentra disponible en un repositorio de GitHub [2].

3.4. SPEC CPU 2006

Standard Performance Evaluation Corporation (SPEC) es un consorcio sin ánimo de lucro que incluye a empresas informáticas de ámbito global como AMD, Dell o Intel, entre otras, además de universidades y grupos de investigación de todo el mundo. El objetivo principal de SPEC es el de crear conjuntos de benchmarks estándar para medir el rendimiento de computadoras y controlar y publicar los resultados de estos tests.

Entre los conjuntos de benchmarks que ofrece SPEC, en este trabajo se hace uso del conjunto SPEC CPU 2006, y más concretamente del subconjunto SPECint, formado por 12 benchmarks que trabajan principalmente con datos de tipo entero. El Anexo A muestra una breve descripción de estos benchmarks. El otro subconjunto que completa SPEC CPU 2006 hace uso principalmente de datos reales y se conoce como SPECfp. Este subconjunto no se ha tenido en consideración en el presente trabajo debido a que el formato de representación IEEE-754 para reales compuesto por signo, exponente y mantisa genera patrones irregulares que normalmente no afectan tanto al envejecimiento como la representación entera [13]. Un estudio detallado de las aplicaciones reales se deja como trabajo futuro.

Nótese que SPEC CPU ofrece una suite de benchmarks más reciente publicada en el año 2017. Sin embargo, en este estudio se decide caracterizar las aplicaciones de la suite de 2006 debido a que ejercen un mayor estrés sobre el banco de registros frente a

[2] https://github.com/miguellasaosa/m2sModificado.git

la suite de 2017 [19].

3.5. Recopilación de Datos

Para obtener resultados representativos de los benchmarks, se han ejecutado 500 millones de instrucciones de cada uno de ellos en el simulador instrumentado, eliminando los primeros 1000 millones de instrucciones.

Mediante estructuras de datos matriciales, se ha almacenado tanto el número de ciclos en los que cada celda de memoria almacena un '0' lógico ($matriz\_duty$), lo cual nos permitirá inferir tanto el *duty cycle* '0' como el '1' en porcentaje de tiempo, así como el número de *flips* ($matriz\_flips$). Cada matriz tiene unas dimensiones de 180 filas (número de registros físicos) $\times$ 32 columnas (número de celdas por registro). De esta manera, si por ejemplo se desea conocer el *duty cycle* '0' que ha sufrido la celda 8 del registro 1, nos referiremos al elemento $matriz\_duty(1, 8)$. Dividiendo el valor de este elemento por el número total de ciclos ejecutados se obtiene el porcentaje de *duty cycle* '0' de la celda de memoria.

Capítulo 4

Resultados Experimentales

Este capítulo presenta y analiza los resultados experimentales más relevantes. En primer lugar se clasifican los benchmarks en clusters para una mejor presentación de los resultados. En base a esta clasificación, en las siguientes secciones se exponen resultados que permiten identificar patrones de uso y desgaste del banco de registros. Puesto que la extensión del documento no permite exponer todos los experimentos realizados, en este apartado se comenta un subconjunto de los mismo. El lector puede encontrar la totalidad de los experimentos en el Anexo B.

4.1. Clasificación de Benchmarks

Debido al elevado número de benchmarks enteros, se decide clasificar las aplicaciones en seis clusters en función de su *duty cycle* '0' (o el complementario *duty cycle* '1') medio. Esta métrica se refiere a la media aritmética de *duty cycle* '0' de todas las celdas de memoria que componen el banco de registros. Se ha decidido utilizar esta métrica frente al número de *flips* medio porque el *duty cycle* tiene una incidencia directa en el efecto NBTI, que a su vez se trata del efecto más perjudicial de envejecimiento.

La Tabla 4.1 muestra el *duty cycle* '0' medio y el número de millones de ciclos de ejecución de cada aplicación. La clasificación en clusters se expone en la Tabla 4.2. Entre paréntesis se muestra el valor de *duty cycle* '0' por cada benchmark, mientras que en negrita se hace referencia a la aplicación representativa del cluster en secciones posteriores.

4.2. Duty Cycle Acumulado

Esta sección analiza el *duty cycle* acumulado, el cual se define como la media aritmética de *duty cycle* de los 180 registros físicos para cada una de sus celdas. La siguiente ecuación muestra cómo se realiza el cálculo de esta métrica para la celda

Benchmark	*Duty cycle* '0' medio	Millones de ciclos de ejecución
astar	67,7 %	1.460
bzip2	64,6 %	5.847
gcc	63,2 %	1.372
gobmk	64,2 %	311
h264ref	76,4 %	1.327
hmmer	44,1 %	592
libquantum	64,3 %	2.469
mcf	59,2 %	625
omnetpp	55,5 %	376
perlbench	64,1 %	776
sjeng	70,7 %	5.826
xalancbmk	55,8 %	941

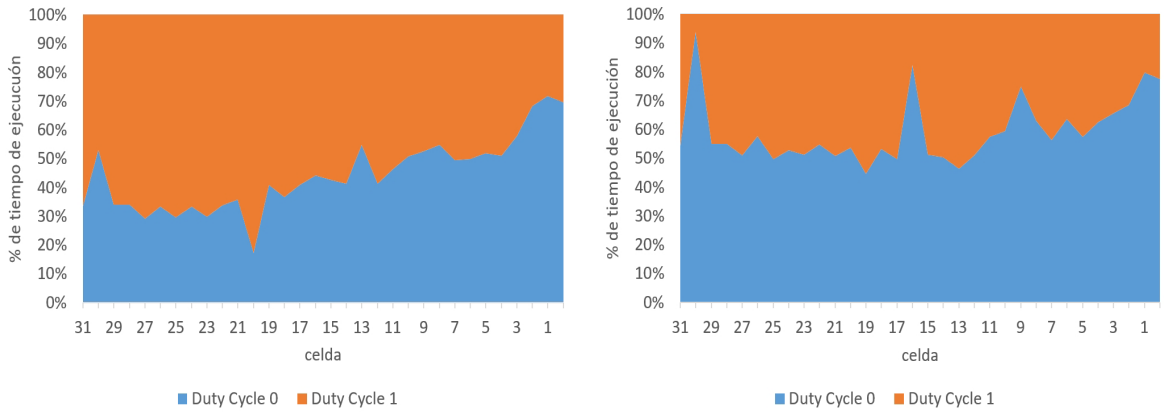
Tabla 4.1: *Duty cycle* '0' medio y millones de ciclos de ejecución por cada aplicación.

Cluster 1	Cluster 2	Cluster 3
hmmer (44,1 %)	**mcf (59,1 %)**	astar (67,7 %)
	omnetp (55,4 %)	bzip2 (64,5 %)
	xalancbmk (55,8 %)	**gcc (63,2 %)**
		gobmk (64,2 %)
		perlbench (64,1 %)
Cluster 4	Cluster 5	Cluster 6
libquantum (64,3 %)	**sjeng (70,7 %)**	**h264ref (76,4 %)**

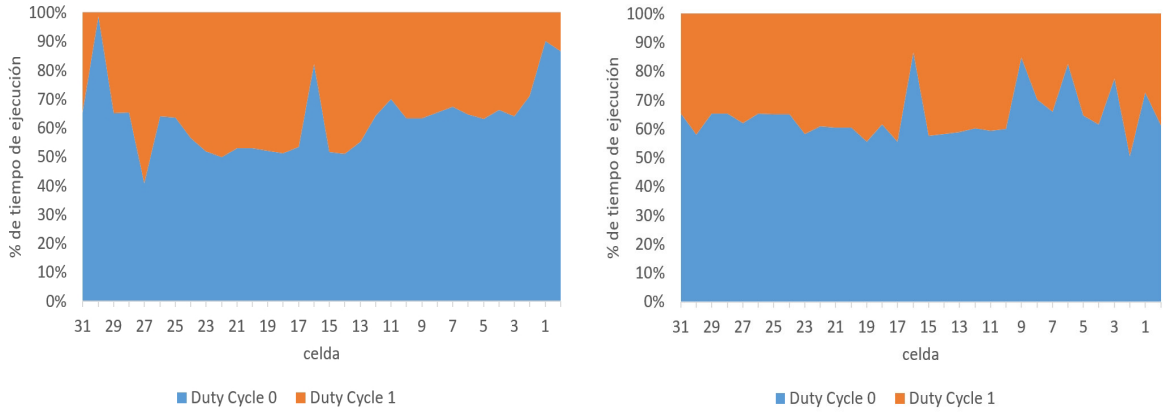
Tabla 4.2: Clusters de benchmarks en función de su *duty cycle* '0' medio (entre paréntesis). Las aplicaciones en negrita son las representativas por cada cluster.

j-ésima:

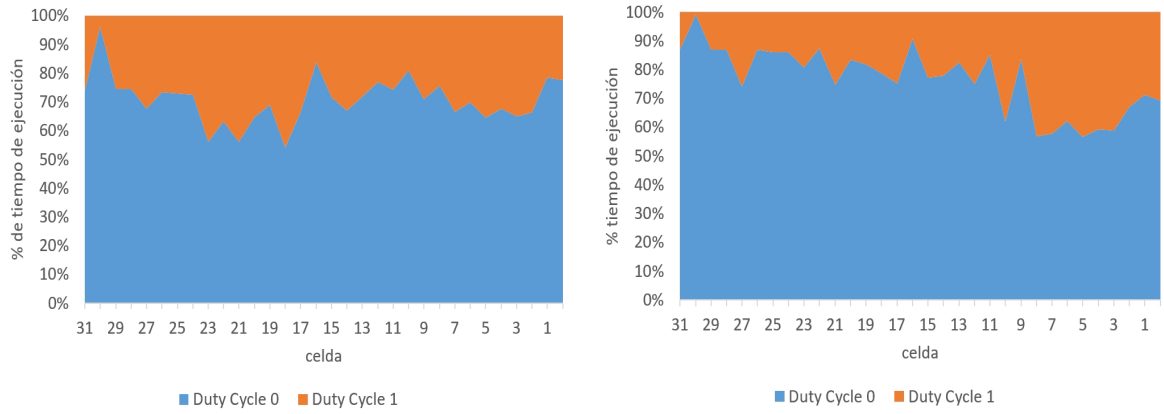
$$Duty\ cycle\ acumulado\ celda\ j = \frac{\sum_{i=0}^{179} matriz\_duty(i,j)}{180\ registros}$$



(a) *hmmer* (b) *mcf*



(c) *gcc* (d) *libquantum*



(e) *sjeng* (f) *h264ref*

Figura 4.1: *Duty cycle* acumulado en cada celda de los registros: (a) Cluster 1, (b) Cluster 2, (c) Cluster 3, (d) Cluster 4, (e) Cluster 5 y (f) Cluster 6.

La Figura 4.1 muestra el *duty cycle* acumulado para las aplicaciones representativas

de cada cluster. El eje X representa a cada una de las celdas de los registros con un identificador de acuerdo a la posición del bit que almacenan, desde la posición 0 hasta la 31 de derecha a izquierda. El eje Y muestra el porcentaje de tiempo de ejecución en el que cada celda almacena el valor lógico '0' o el valor '1'. Se refiere al lector al Anexo B para resultados de *duty cycle* acumulado para el resto de aplicaciones.

Como cabía esperar de acuerdo con la clasificación anterior, *hmmer* (Figura 4.1a) y *h264ref* (Figura 4.1f) son las aplicaciones con menor y mayor *duty cycle* '0' acumulado, respectivamente. A excepción de *hmmer*, el resto de benchmarks presentan un *duty cycle* '0' predominante frente al complementario *duty cycle* '1'. Este hecho pone de manifiesto que la mayoría de aplicaciones enteras generan una cantidad importante de bytes nulos, lo cual puede deberse a que normalmente el valor de los enteros es cercano o igual a cero.

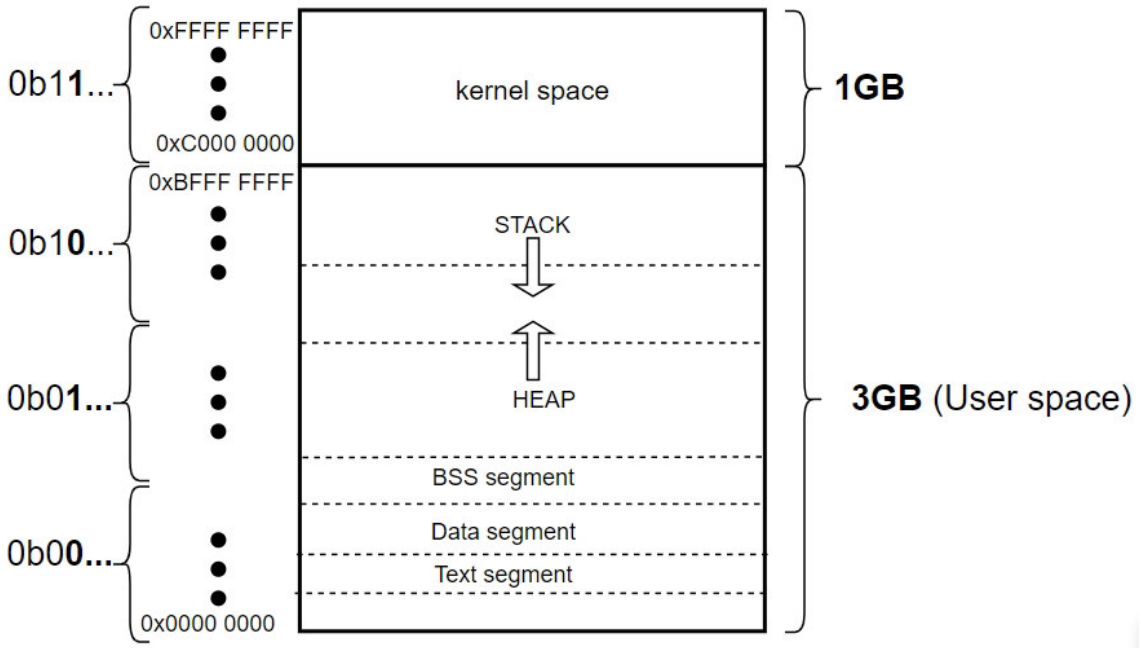


Figura 4.2: Espacio de direccionamiento de un proceso en sistemas operativos Linux.

Por otra parte, en todos los benchmarks excepto *libquantum* (Figura 4.1d), se observa en mayor o menor medida un pico de *duty cycle* '0' en la celda 30. Una posible explicación a este efecto es el almacenamiento de no sólo valores enteros sino también de direcciones lógicas en el banco de registros. Para todos los benchmarks, se inspeccionó detalladamente el contenido almacenado en los registros. Se observó que las instrucciones que cargan la dirección efectiva de las variables en un registro siempre cargaban un '0' en el bit 30. Esto nos permitió conjeturar sobre qué estaba ocurriendo. La Figura 4.2 muestra la imagen de memoria de un proceso en un sistema operativo Linux. La imagen de proceso se puede dividir en cuatro bloques de 1 GB

cada uno. El primero de estos bloques de arriba hacia abajo es una zona reservada para el kernel, mientras que el resto pertenece al espacio de usuario, es decir, se trata de una región accesible por el programa. En la figura se aprecia como dos de los 3 GB del espacio de usuario son direccionados siempre con un '0' en el bit 30. Además, este espacio contiene regiones muy utilizadas por la mayoría de programas como la pila (*stack*) o el segmento de datos. El espacio de pila, como se indica en la figura, comienza en la dirección 0xBFFFFFFF[1]. Desde el punto de vista del envejecimiento, el desbalanceo que muestra el bit 30 es un problema que se debe resolver, ya que en este caso el desgaste de las celdas 30 de los registros por la presencia del efecto NBTI puede comprometer el funcionamiento del procesador al completo.

4.3. Flips Acumulados

De manera similar al duty cycle acumulado, esta sección estudia el número de flips acumulados. Esta métrica se define mediante la siguiente ecuación:

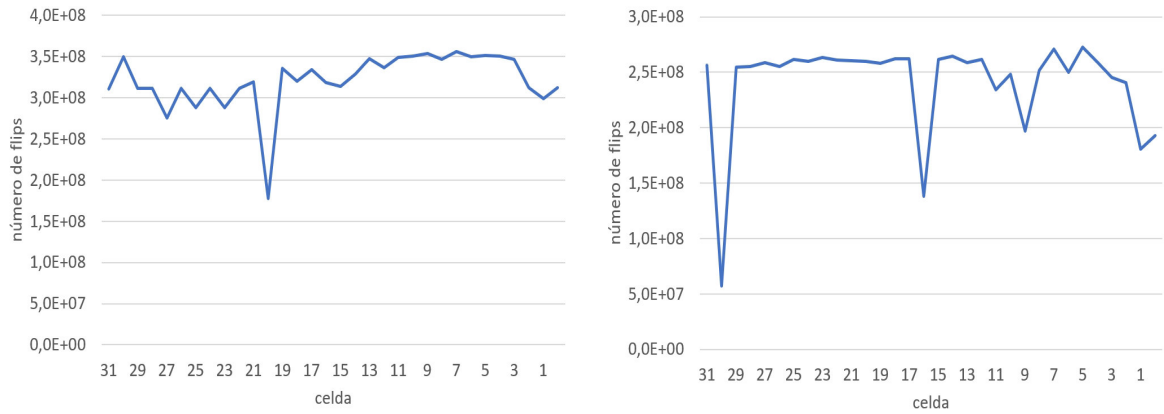
$$Flips\ acumulados\ celda\ j = \sum_{i=0}^{179} matriz\_flips(i,j)$$

La Figura 4.3 muestra los resultados para los benchmarks seleccionados. En este caso, el eje Y representa el número de *flips* acumulado que ha sufrido cada una de las celdas. Los resultados ponen de manifiesto la relación existente entre los *duty cycles* y los *flips*. Aquellas celdas donde se observan picos de *duty cycle* '0' sufren normalmente un menor número de flips. Esto se observa especialmente en la celda 30 de la mayoría de las aplicaciones. En aplicaciones como *sjeng* y *h264ref*, se aprecia claramente como la mayoría de valores almacenados son cercanos a cero debido a que la mayoría de picos, en cuanto a número de *flips* se refiere, se concentran en los bits de menor peso de los registros. En el caso de *h264ref*, estos resultados también se pueden confirmar atendiendo al *duty cycle* '0' medio (Figura 4.1f). Estas aplicaciones, por tanto, sufren un mayor desgaste por efecto HCI en las celdas que contienen los bits menos significativos.

4.4. Duty Cycle por Celda

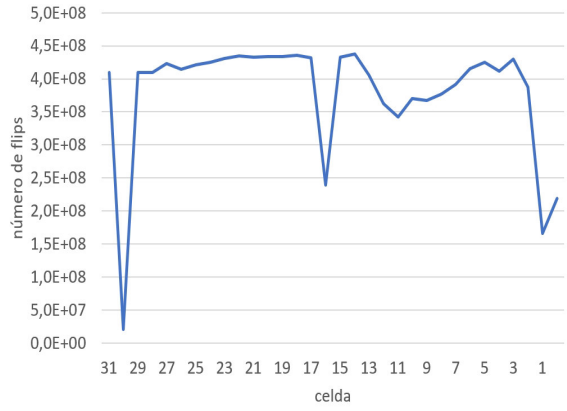
En esta sección se estudia el *duty cycle* a nivel de celda de memoria, es decir, de manera más detallada frente al estudio de la Sección 4.2. La Figura 4.4 muestra los resultados. En este caso, por razones ilustrativas, se presentan tan sólo dos aplicaciones representativas del comportamiento del *duty cycle* por celda. estas aplicaciones

[1]La ubicación del resto de segmentos de espacio de usuario en la Figura 4.2 es orientativa.

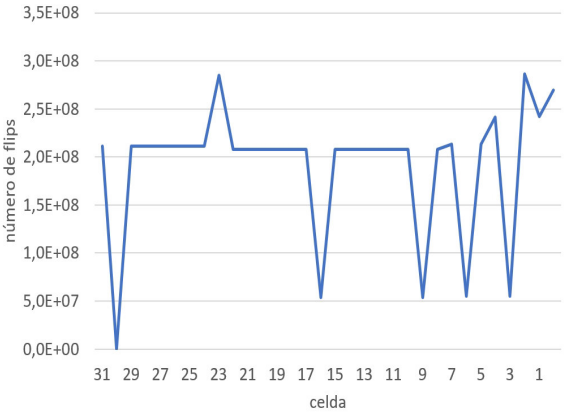


(a) *hmmer*

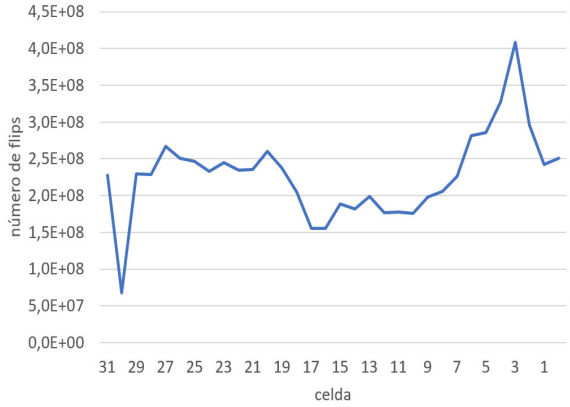
(b) *mcf*



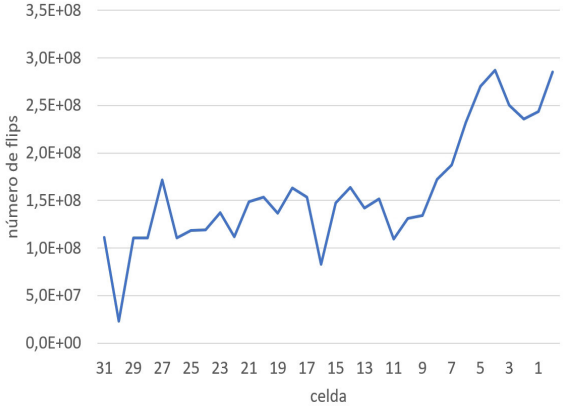
(c) *gcc*



(d) *libquantum*



(e) *sjeng*

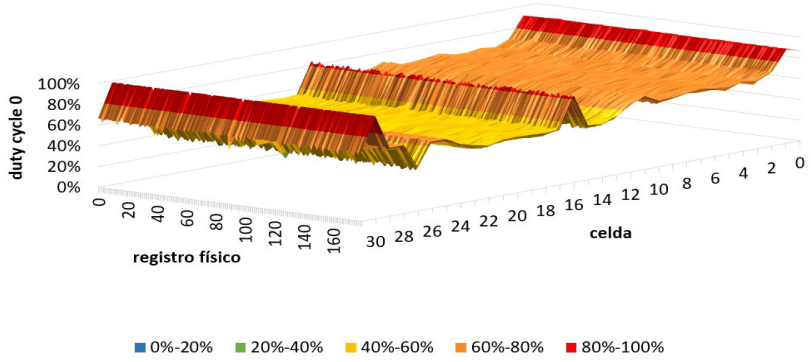


(f) *h264ref*

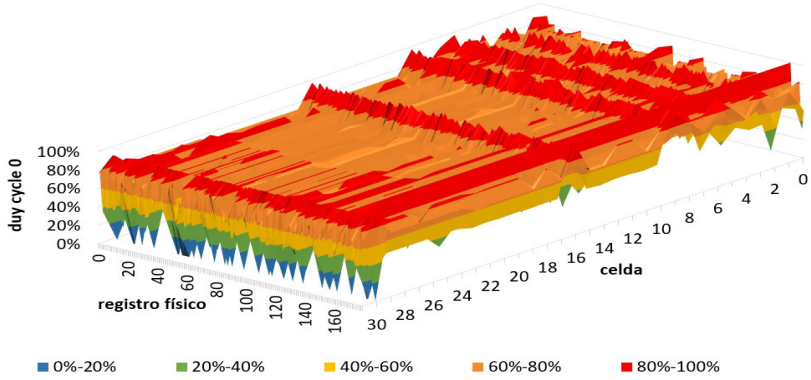
Figura 4.3: Número de *flips* acumulados en cada celda de los registros: (a) Cluster 1, (b) Cluster 2, (c) Cluster 3, (d) Cluster 4, (e) Cluster 5 y (f) Cluster 6.

pertenecen a los clusters 2 y 3. Nótese que para la representación tridimensional se añade un eje adicional correspondiente al identificador de registro físico desde el 0 hasta el 179.

Mientras *gcc* muestra un comportamiento bastante regular con valores homogéneos en todos los registros físicos, por el contrario, *libquantum* es irregular y los valores de



(a) *gcc*



(b) *libquantum*

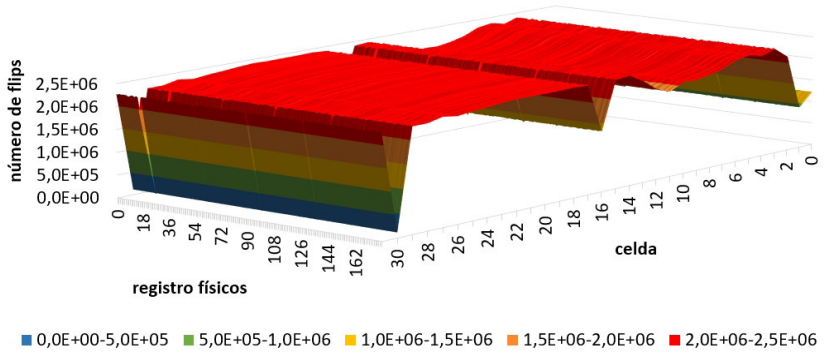
Figura 4.4: *Duty cycle* '0' por cada celda del banco de registros.

duty cycle en cada registro físico son dispares. En otras palabras, comparando estos resultados con las Figuras 4.1c y 4.1d, el patrón de *gcc* se replica en los resultados por celda, mientras que con *libquantum* resulta difícil establecer una relación entre las figuras. En el caso de *gcc*, la gestión cíclica del renombre de registros uniformiza claramente los valores escritos en cada registro del banco.

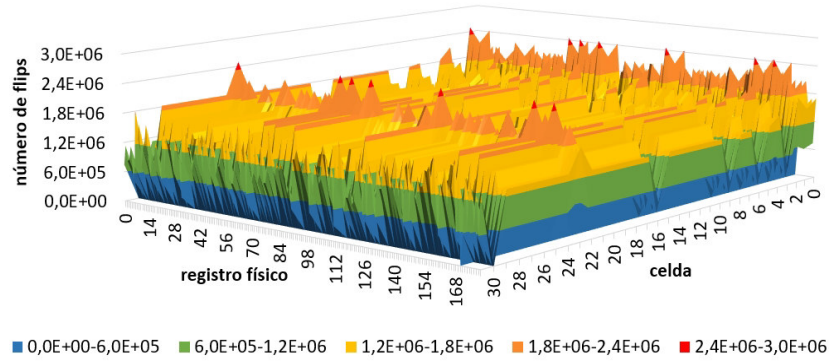
4.5. Flips por Celda

A diferencia de la sección anterior, en este estudio se analiza el número de *flips* por cada celda del banco de registros. La Figura 4.5 muestra los resultados. En este caso, el eje Z muestra el número de *flips* por cada aplicación. De manera similar al estudio previo, para *gcc* el patrón de *flips* de la Figura 4.3f se reproduce en el análisis de número de *flips* a nivel de celda de memoria. Por el contrario, en el caso de *libquantum* es difícil establecer una relación con la Figura 4.3c.

A la vista de los resultados, el comportamiento homogéneo de *gcc* podría explicarse por el hecho de que esta aplicación concentra sus 500 millones de instrucciones en una estructura de control repetitiva (p.e. un bucle), las instrucciones que forman el



(a) *gcc*



(b) *libquantum*

Figura 4.5: Número de *flips* por cada celda del banco de registros.

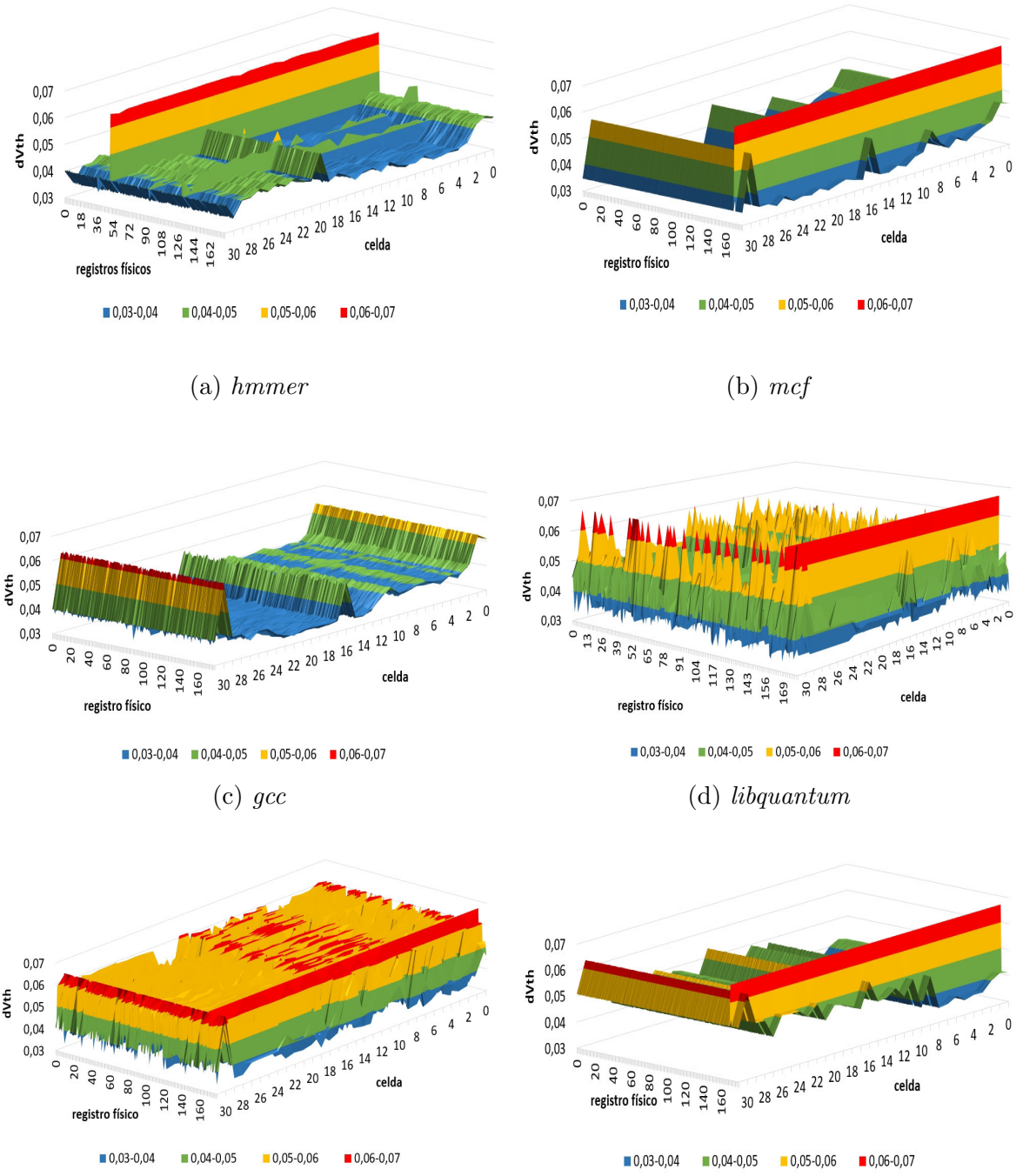
cuerpo de esta estructura repetitiva tienen un patrón de desgaste [2] que la política de renombre (FIFO) se va a encargar de esparcir a lo largo del banco de registros físico. Por el contrario, la ejecución de *libquantum* muestra un patrón de desgaste que varía a lo largo de la ejecución de las 500 millones de instrucciones.

4.6. dV_{th} por Celda

El análisis de los resultados anteriores permite inferir ciertos patrones de envejecimiento en el banco de registros. En esta sección se cuantifica el envejecimiento en el banco de registros de acuerdo a la obtención del aumento de V_{th} por cada celda atendiendo a las Ecuaciones 2.1 y 2.2.

La Figura 4.6 muestra los resultados de dV_{th} por cada aplicación representativa de un cluster. Los picos más altos indican una mayor degradación en la celda correspondiente. Recuérdese también que los resultados por cada celda se refieren al transistor en la misma con mayor dV_{th}, el cual se define como la suma de dV_{th}_BTI y

[2] En sucesiva ejecuciones, esta porción de código escribe valores que provocan los mismos *flips* (en media) en cada celda del registro escrito



(a) *hmmer* (b) *mcf*

(c) *gcc* (d) *libquantum*

(e) *sjeng* (f) *h264ref*

Figura 4.6: dV_{th} por cada celda del banco de registros: (a) Cluster 1, (b) Cluster 2, (c) Cluster 3, (d) Cluster 4, (e) Cluster 5 y (f) Cluster 6.

dV_{th}_HCI.

Existe cierta similitud entre los resultados de dV_{th} y los resultados de *duty cycle* acumulado presentados en la Figura 4.1. Este hecho pone de manifiesto el impacto que tiene el efecto NBTI sobre la degradación de voltaje umbral. Además, cabe destacar que, para los *duty cycle*, tanto los picos altos como los picos bajos se traducen en picos altos de dV_{th} en la Figura 4.6. En este sentido, recuérdese que los *duty cycle* '0'

y ‘1’ generan el efecto NBTI en los transistores pMOS T_{P1} y T_{P2}, respectivamente (Sección 2.1). Pese a estos resultados, estudiar el efecto HCI también reviste de interés, puesto que algunos trabajos recientes inciden en que HCI cobrará una mayor importancia en futuros nodos tecnológicos [20].

Finalmente, nótese que en todas las aplicaciones, a excepción de *gcc*, existen algunos registros donde, para todas las celdas del registro, el valor de dV_{th} es máximo. Esto se a que los registros implicados mantienen un mismo valor durante gran parte de los 500 millones de instrucciones ejecutadas. Los cual fuerza a que sus celdas mantienen un valor constante a ’0’ o a ’1’ durante todo ese tiempo, es decir *duty cycle* ‘0’ o ‘1’ cercana al 100 %. Un patrón de uso relacionado con este caso patológico son por ejemplo las constantes o invariantes de bucle almacenadas en un registro y que son leídas por las instrucciones del cuerpo del bucle en el espacio de iteración del mismo. En cualquier caso, estos registros, así como los bits 30 en muchos casos, y otras celdas de memoria, precisarían de mecanismos arquitectónicos para distribuir la degradación de manera uniforme entre el resto de celdas de memoria.

Capítulo 5

Conclusiones y Trabajo Futuro

El presente capítulo resume las conclusiones principales de este trabajo, muestra posibles direcciones futuras y un diagrama de Gantt con las principales tareas del trabajo y las horas de dedicación.

5.1. Conclusiones

En este trabajo se ha realizado un estudio sobre cómo afectan los valores almacenados en el banco de registros de un procesador x86 al envejecimiento de las celdas de memoria SRAM que lo forman. En este sentido, una degradación prematura de las celdas del banco de registros compromete la vida útil de esta estructura, y por extensión, del procesador.

Para ello, se han modelado las características principales de un procesador Intel Skylake en el simulador ciclo-a-ciclo Multi2Sim. Asimismo, se ha instrumentado esta herramienta para obtener estadísticas de uso del banco de registros ejecutando aplicaciones científicas representativas para, finalmente, cuantificar la degradación de cada celda de memoria del banco mediante un modelo analítico y en términos de aumento del voltaje umbral de los transistores.

El análisis de los resultados experimentales confirma que la fuente de desgaste más importante en el banco de registros se debe al efecto NBTI, el cual aparece bajo el denominado *duty cycle*, es decir, el mantenimiento a lo largo del tiempo de un valor lógico en una celda de memoria. En este sentido, se ha observado que las celdas de memoria almacenando el bit 30-ésimo de los registros concentran una gran degradación por el hecho de almacenar, entre otros datos, direcciones lógicas de memoria. Asimismo, algunos registros específicos concentran una gran degradación en todas las celdas, posiblemente debido a que almacenan valores constantes y por lo tanto no van a ser renombrados hasta que dicha constante deje de ser utilizada.

Por otro lado, también se ha tenido en cuenta el efecto de degradación HCI

producido por la frecuencia de cambio del valor lógico almacenado en una celda de memoria. Aunque el impacto de este efecto es menor en la degradación de las celdas frente a NBTI, su estudio resulta necesario, puesto que se prevé que en futuros nodos tecnológicos HCI adquiera una mayor relevancia. En este sentido, algunas aplicaciones muestran una mayor concentración de cambios de valor lógico en las celdas que almacenan los bits menos significativos de los registros.

Finalmente, los resultados experimentales también han mostrado que el envejecimiento del banco de registros en procesadores x86 depende en gran medida de la aplicación que se esté ejecutando. Esto significa que aplicar medidas paliativas independientes de la aplicación tendrá un menor impacto frente a otras técnicas que sean conscientes de la actividad interna en el banco de registros según la aplicación.

5.2. Trabajo Futuro

En cuanto a trabajo futuro, existen varias direcciones posibles:

- Ampliar el estudio para los benchmarks SPECfp 2006, analizando el impacto que tiene la representación de reales constituyendo un signo, exponente y mantisa.
- Profundizar en las razones que explican algunos detalles de la caracterización propuesta como la degradación observada en el bit 30 o los registros con una alta degradación en todas las celdas.
- Proponer nuevos mecanismos anti-envejecimiento en el banco de registros x86 que tengan en cuenta los patrones de cada aplicación, por ejemplo a través de propuestas alternativas de renombre de registros o la transformación de los datos almacenados para mantener a las celdas en valores próximos a los ideales de *duty cycle* y número de *flips* durante todo el tiempo de ejecución de las aplicaciones.

5.3. Diagrama de Gantt

En esta sección se muestra un diagrama de Gantt que ordena temporalmente las tareas descritas en la Sección 1.2. Nótese que se añade una tarea adicional consistente en la documentación del trabajo, la cual ha tenido lugar durante toda la realización del proyecto.

Finalmente, la Figura 5.2 muestra las horas dedicadas a cada una de las tareas, siendo aquellas involucradas en el simulador ciclo-a-ciclo las que han supuesto una mayor dedicación en horas.

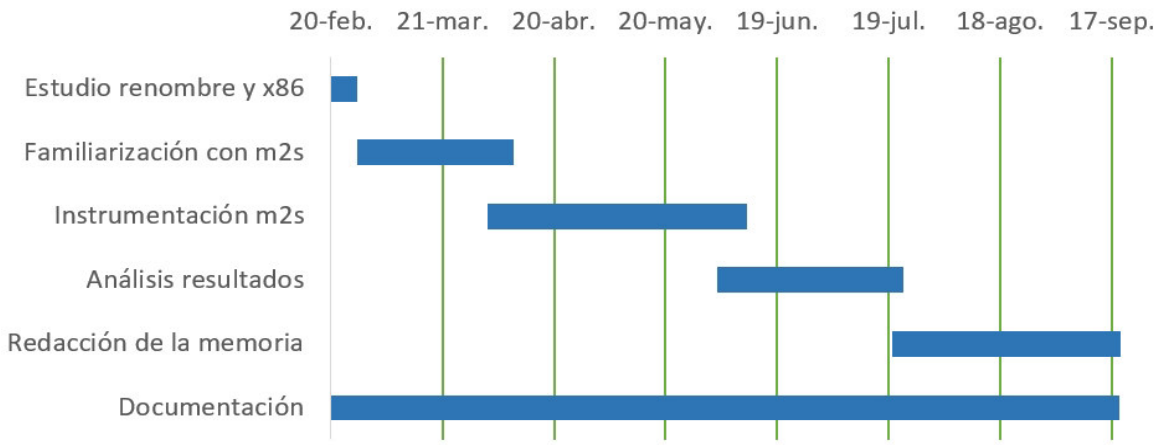


Figura 5.1: Diagrama de Gantt del proyecto.

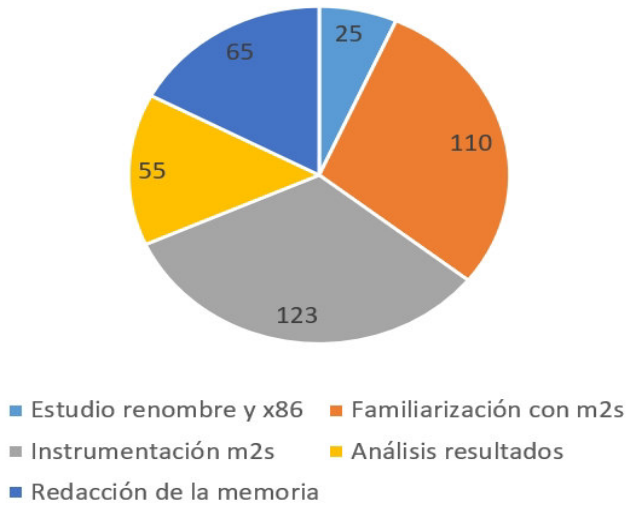


Figura 5.2: Horas dedicadas a cada tarea del trabajo.

Capítulo 6

Bibliografía

[1] Skylake (client), Intel Microarchitectures, `https://en.wikichip.org/wiki/intel/microarchitectures/skylake_(client)`.

[2] R. Vattikonda, W.Wang, and Y. Cao. Modeling and Minimization of PMOS NBTI Effect for Robust Nanometer Design. In *Proceedings of the 43rd ACM/IEEE Design Automation Conference*, pages 1047–1052, 2006.

[3] A. Tiwari and J. Torrellas. Facelift: Hiding and Slowing Down Aging in Multicores. In *Proceedings of the 41st IEEE/ACM International Symposium on Microarchitecture*, pages 129–140, 2008.

[4] E. Mintarno, V. Chandra, D. Pietromonaco, R. Aitken, and R. W. Dutton. Workload Dependent NBTI and PBTI Analysis for a Sub-45nm Commercial Microprocessor. In *Proceedings of the IEEE International Reliability Physics Symposium*, pages 1–6, 2013.

[5] H. Amrouch, T. Ebi, and J. Henkel. Stress Balancing to Mitigate NBTI Effects in Register Files. In *Proceedings of the 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 1–10, 2013.

[6] S. Kothawade, D. M. Ancajas, K. Chakraborty, and S. Roy. Mitigating NBTI in the Physical Register File through Stress Prediction. In *Proceedings of the IEEE 30th International Conference on Computer Design*, pages 345–351, 2012.

[7] S. Kothawade, K. Chakraborty, and S. Roy. Analysis and Mitigation of NBTI Aging in Register File: An End-To-End Approach. In *2011 12th International Symposium on Quality Electronic Design*, pages 1–7, 2011.

[8] E. Gunadi, A. A. Sinkar, N. S. Kim, and M. H. Lipasti. Combating Aging with the Colt Duty Cycle Equalizer. In *Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 103–114, 2010.

[9] J. Abella, X. Vera, and A. Gonzalez. Penelope: The NBTI-Aware Processor. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 85–96, 2007.

[10] J. Shin, V. Zyuban, P. Bose, and T. M. Pinkston. A Proactive Wearout Recovery Approach for Exploiting Microarchitectural Redundancy to Extend Cache SRAM Lifetime. *SIGARCH Computer Architecture News*, 36(3):353–362, 2008.

[11] A. Calimera, M. Loghi, E. Macii, and M. Poncino. Dynamic Indexing: Leakage-Aging Co-Optimization for Caches. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(2):251–264, 2014.

[12] S. Wang, G. Duan, C. Zheng, and T. Jin. Combating NBTI-Induced Aging in Data Caches. In *Proceedings of the 23rd ACM International Conference on Great Lakes Symposium on VLSI*, page 215–220, 2013.

[13] A. Valero, N. Miralaei, S. Petit, J. Sahuquillo, and T. M. Jones. On Microarchitectural Mechanisms for Cache Wearout Reduction. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(3):857–871, 2017.

[14] M. Namaki-Shoushtari, A. Rahimi, N. Dutt, P. Gupta, and R. K. Gupta. ARGO: Aging-aware GPGPU Register File Allocation. In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*, pages 1–9, 2013.

[15] J. Tan, M. Chen, Y. Yi, and X. Fu. Mitigating the Impact of Hardware Variability for GPGPUs Register File. *IEEE Transactions on Parallel and Distributed Systems*, 27(11):3283–3297, 2016.

[16] A. Valero, F. Candel, D. Suárez-Gracia, S. Petit, and J. Sahuquillo. An Aging-Aware GPU Register File Design Based on Data Redundancy. *IEEE Transactions on Computers*, 68(1):4–20, 2019.

[17] R. M. Tomasulo. An Efficient Algorithm for Exploiting Multiple Arithmetic Units. *IBM Journal of Research and Development*, 11(1):25–33, 1967.

[18] R. Ubal, J. Sahuquillo, S. Petit, and P. Lopez. Multi2Sim: A Simulation Framework to Evaluate Multicore-Multithreaded Processors. In *Proceedings of the 19th International Symposium on Computer Architecture and High Performance Computing*, pages 62–68, 2007.

[19] A. Limaye and T. Adegbija. A Workload Characterization of the SPEC CPU2017 Benchmark Suite. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software*, pages 149–158, 2018.

[20] F. Gabbay and A. Mendelson. Asymmetric Aging Effect on Modern Microprocessors. *arXiv:2009.03945*, pages 1–11, 2020.

Anexos

Anexos A

Benchmarks SPECint 2006

Este anexo presenta una breve descripción de cada uno de los benchmarks utilizados en el proyecto.

- perlbench: se trata de una versión reducida del lenguaje de programación Perl. Su carga de trabajo es la ejecución de los programas SpamAssian, MHonARc y Specdiff, escritos en Perl.

- bzip2: se trata de una versión modificada de la herramienta de compresión/descompresión de archivos bzip2 escrita en ANSI C.

- gcc: se trata de un compilador para el lenguaje C que traduce a lenguaje maquina. Su carga de trabajo de se compone de 9 programas escritos en C.

- mcf: escrito en ANSI C, es un programa cuya finalidad es el cálculo y optimización de rutas de transporte público.

- gobmk: es un programa escrito en C, ideado para jugar al Go. Partiendo de un fichero, calcula la secuencia de movimientos óptima.

- hmmer: escrito en C, utiliza modelos ocultos de Markov para buscar coincidencias en las cadenas genéticas presentes en una base de datos. Como carga de trabajo utiliza una base de datos y un fichero compuesto por modelos de Markov.

- sjeng: se trata de un programa escrito en ANSI C que mediante inteligencia artificial calcula las mejores jugadas a partir de posiciones iniciales dadas.

- libquantum: este benchmark escritro en C simula un computador cuántico que utiliza el algoritmo descubierto por Peter Shor para factorizar números en tiempo polinomial.

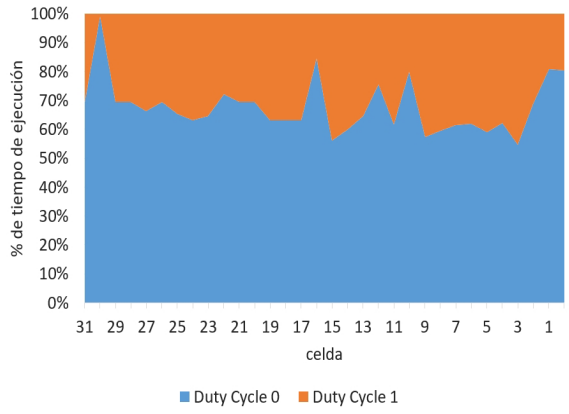
- omnetpp: simula una gran red Ethernet utilizando OMNeT++. Como carga de trabajo recibe un fichero que contiene la topología completa de la red. El programa, al igual que el simulador, está escrito en C++.
- xalancbmk: es una versión modificada de XalanC++, un procesador XSLT para la transformación de XSL, convirtiendo documentos XML a HTML, texto plano u otros tipos de ficheros.

Anexos B

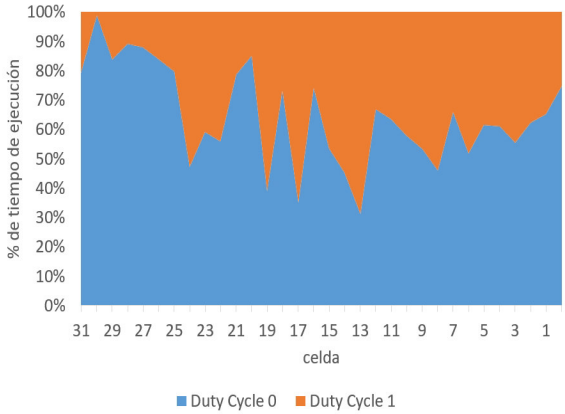
Resultados

En este anexo se adjuntan las figuras de los benchmarks que no han sido expuestas en el Capítulo 4, siguiendo la misma estructura: duty cycle acumulado, flips acumulados, duty cycle por celda, flips por celda y dV_{th} por celda.

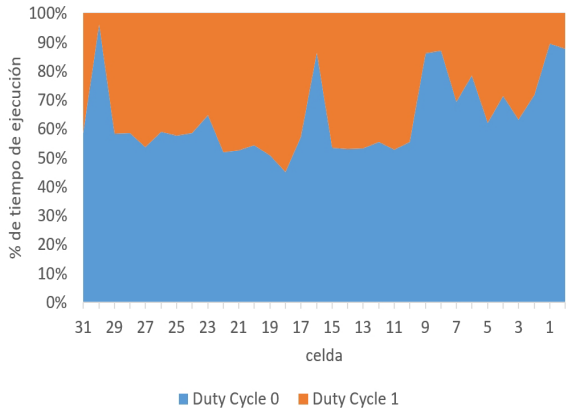
B.1. Duty Cycle Acumulado



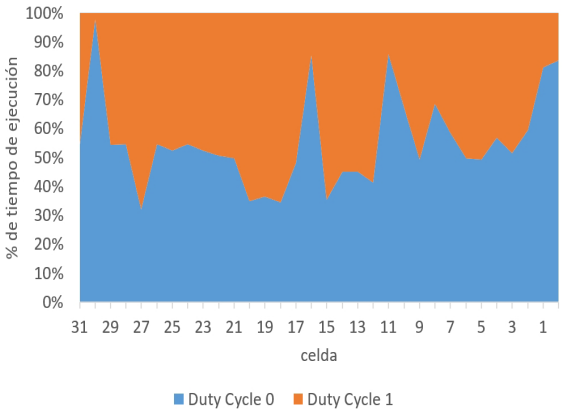
(a) *astar*



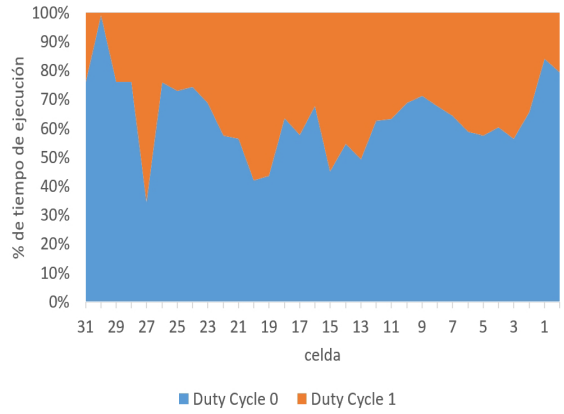
(b) *bzip2*



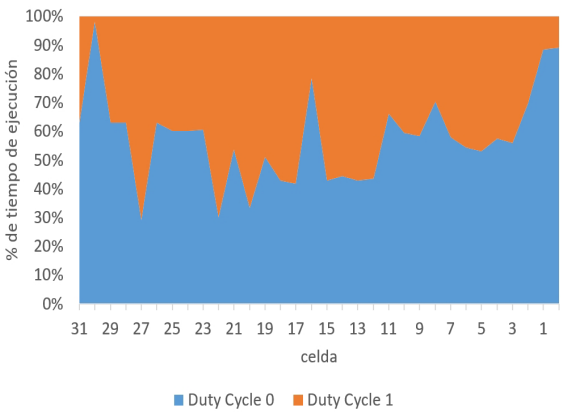
(c) *gobmk*



(d) *omnetp*



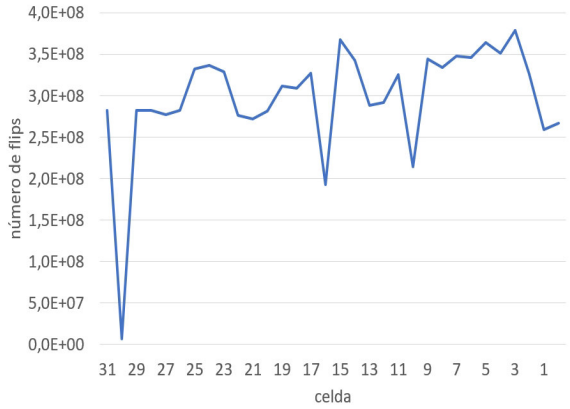
(e) *perlbench*



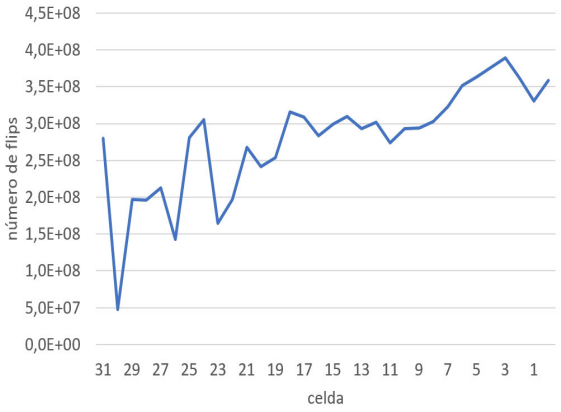
(f) *xalancbmk*

Figura B.1: *Duty cycle* acumulado en cada celda de los registros.

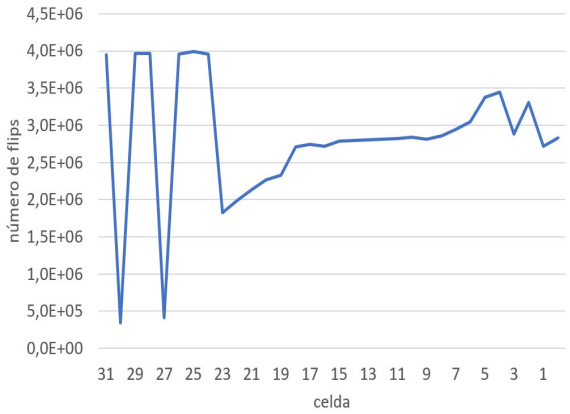
B.2. Flips Acumulados



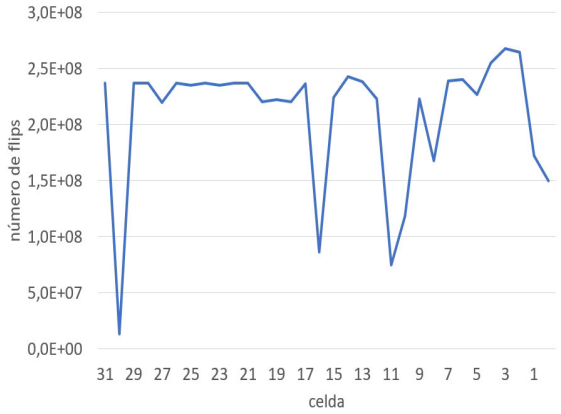
(a) *astar*



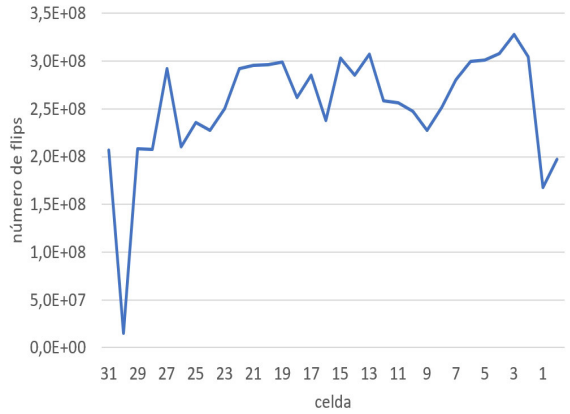
(b) *bzip2*



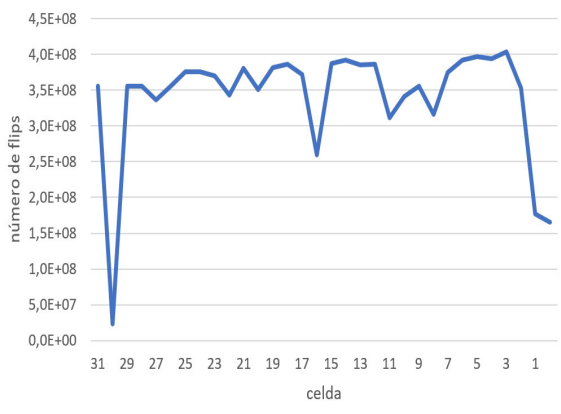
(c) *gobmk*



(d) *omnetp*



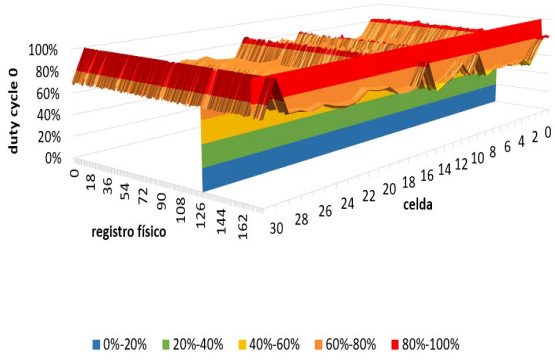
(e) *perlbench*



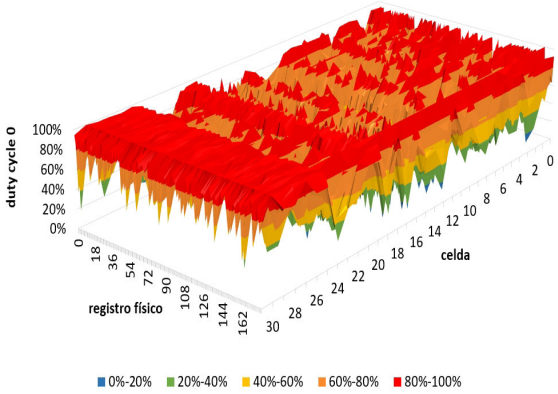
(f) *xalancbmk*

Figura B.2: Número de *flips* acumulados en cada celda de los registros.

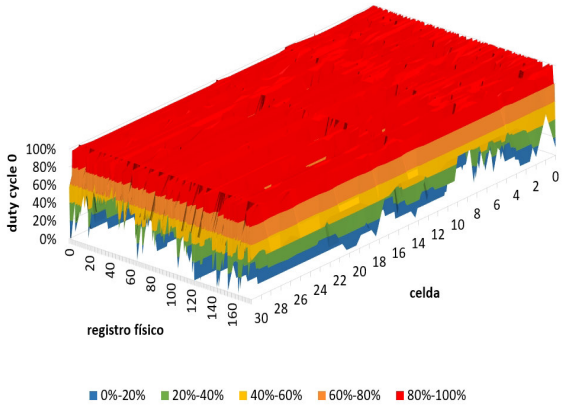
B.3. Duty Cycle por Celda



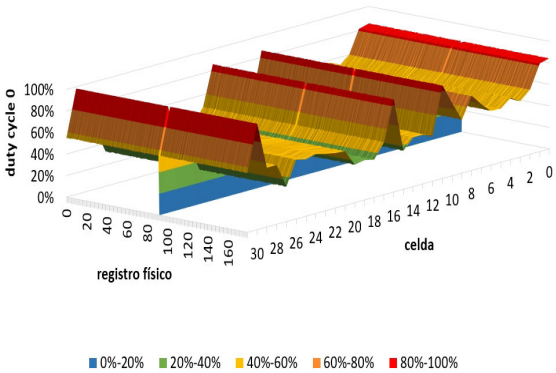
(a) *astar*



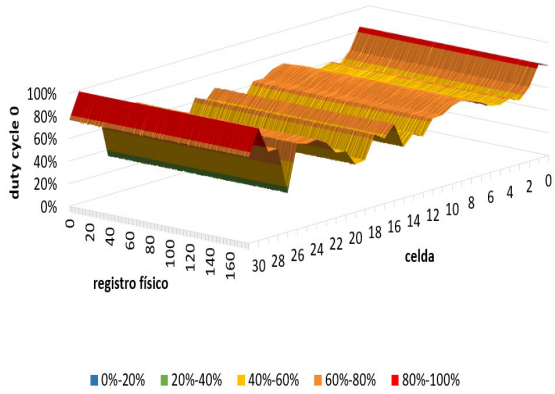
(b) *bzip2*



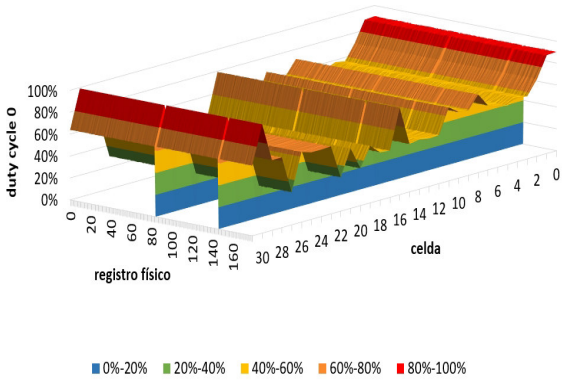
(c) *gobmk*



(d) *omnetpp*

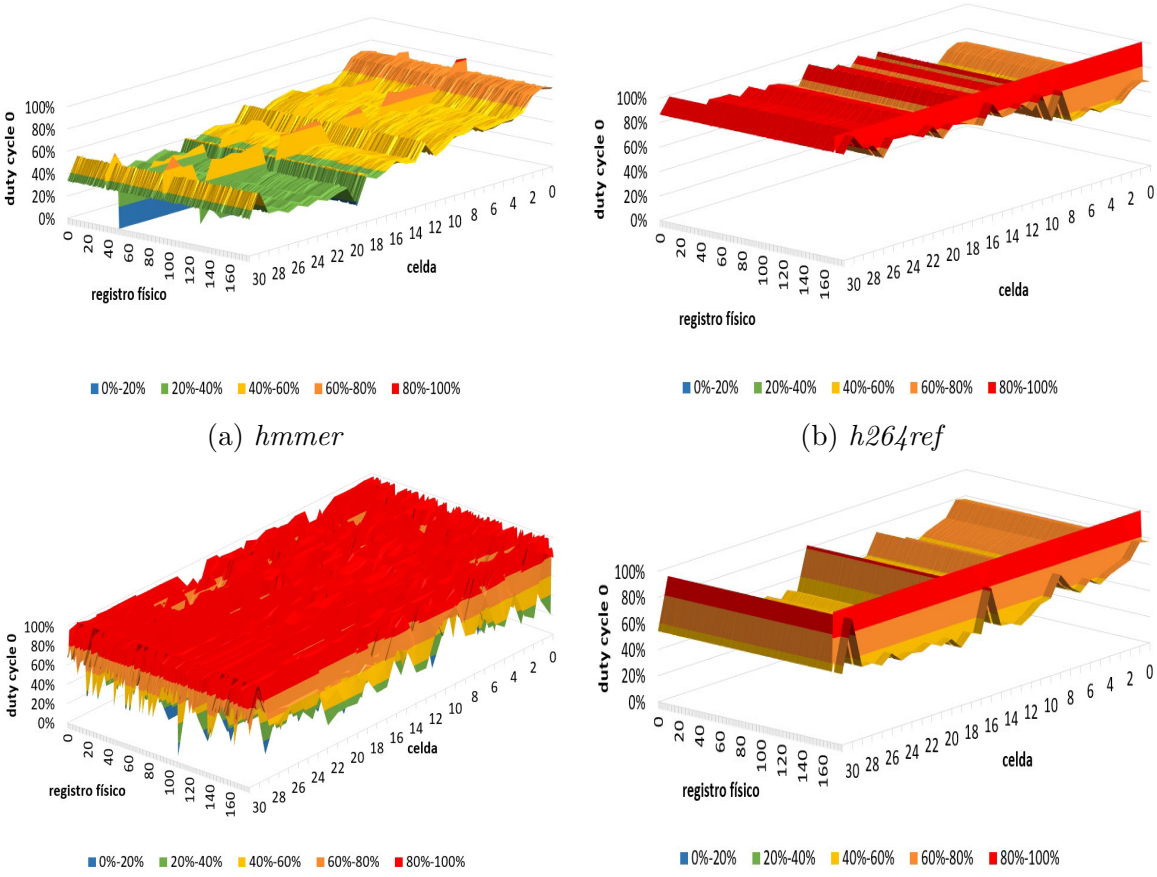


(e) *perlbench*



(f) *xalancbmk*

Figura B.3: *Duty cycle* '0' por cada celda del banco de registros (1).

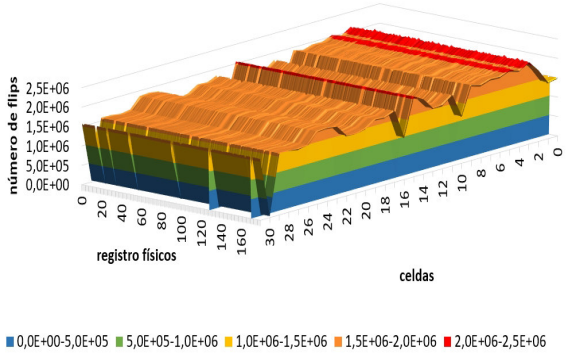


(a) *hmmer* (b) *h264ref*

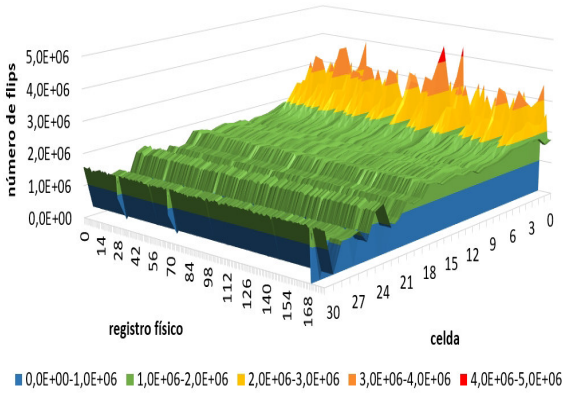
(c) *perlbench* (d) *mcf*

Figura B.4: *Duty cycle* '0' por cada celda del banco de registros (2).

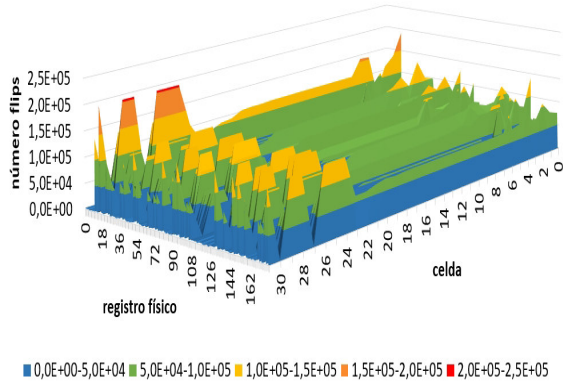
B.4. Flips por Celda



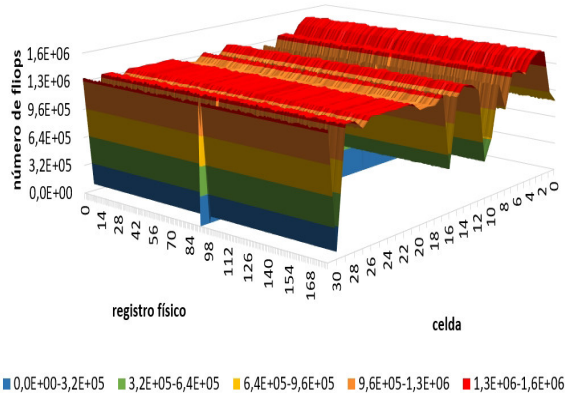
(a) *astar*



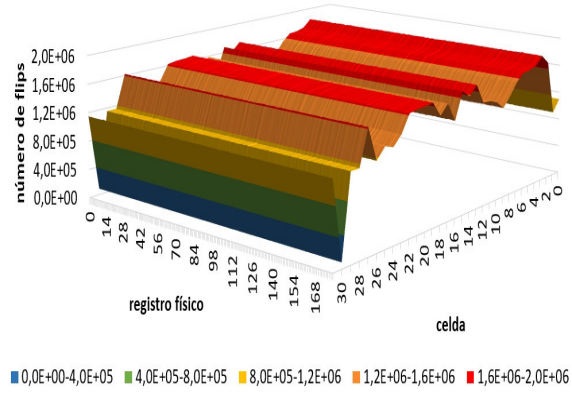
(b) *bzip*



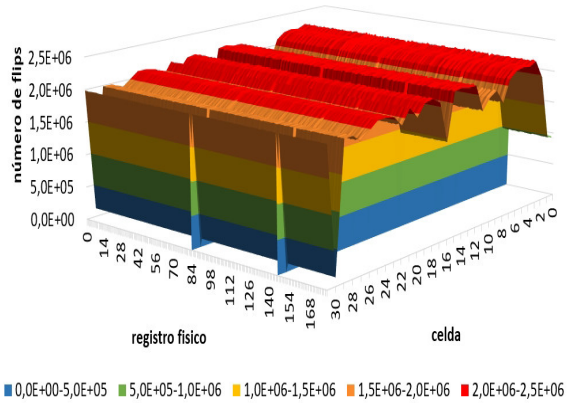
(c) *gobmk*



(d) *omnetpp*

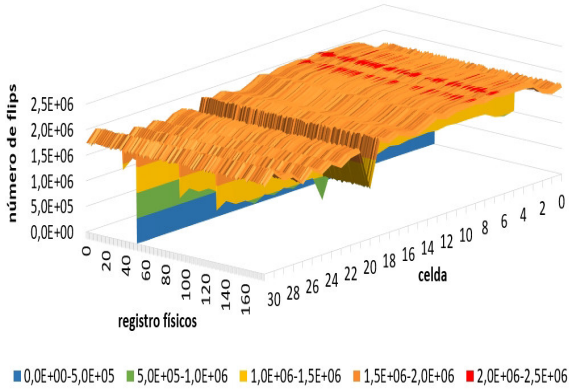


(e) *perlbench*

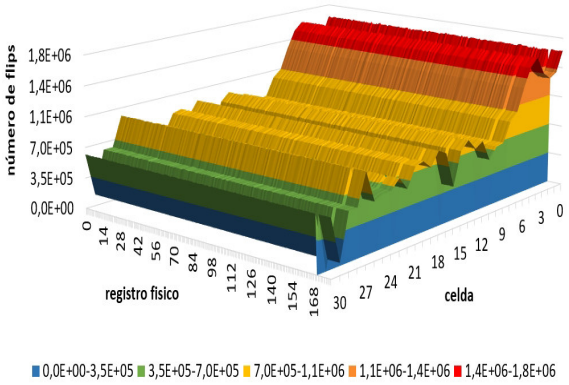


(f) *xalancbmnk*

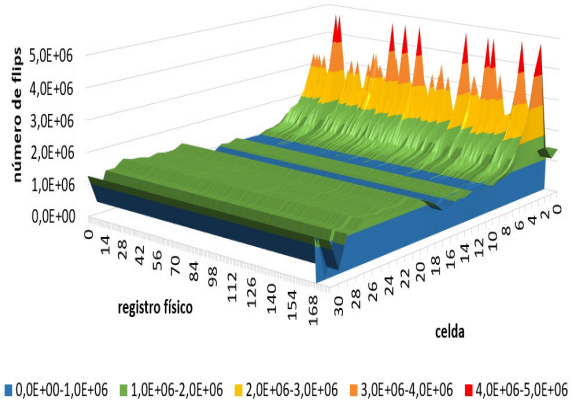
Figura B.5: Número de *flips* por cada celda del banco de registros (1).



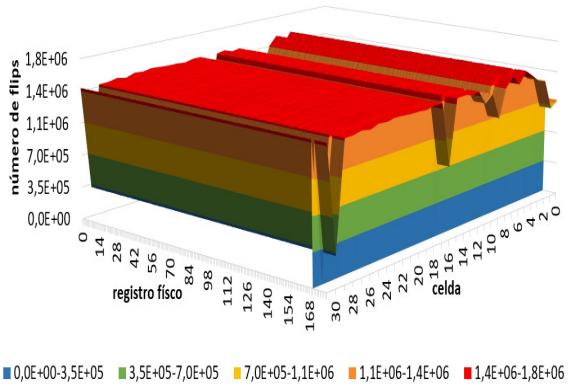
(a) *hmmer*



(b) *h264ref*



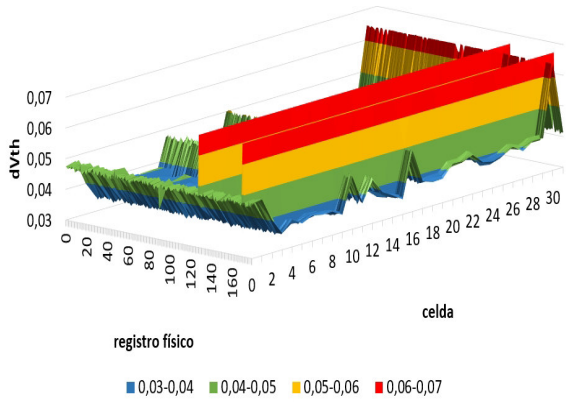
(c) *sjeng*



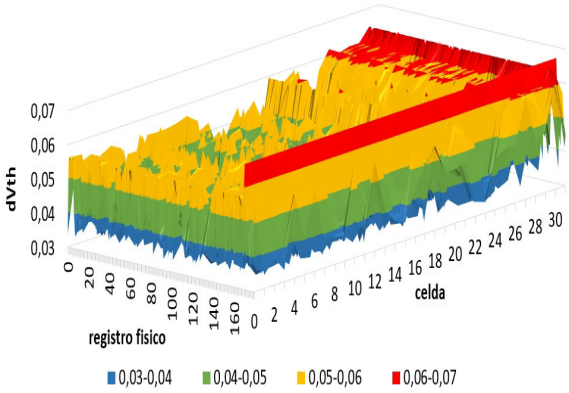
(d) *mcf*

Figura B.6: Número de *flips* por cada celda del banco de registros (2).

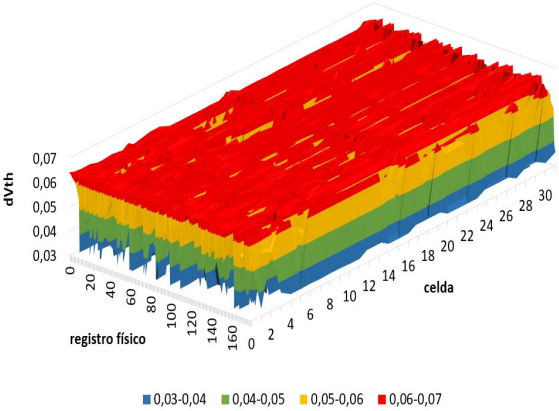
B.5. dV_{th} por Celda



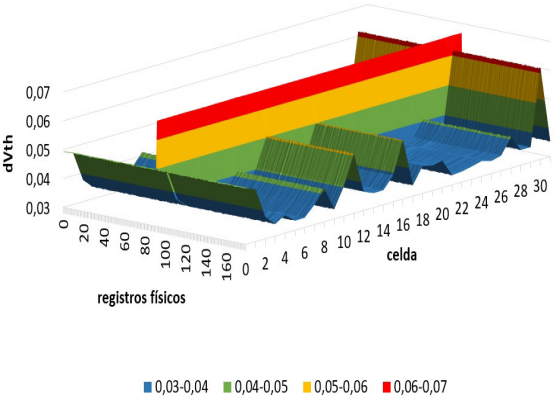
(a) *astar*



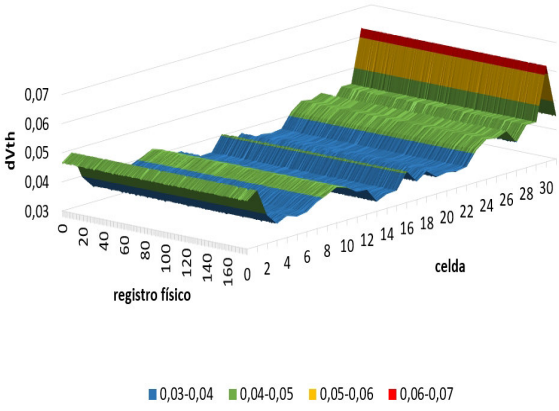
(b) *bzip2*



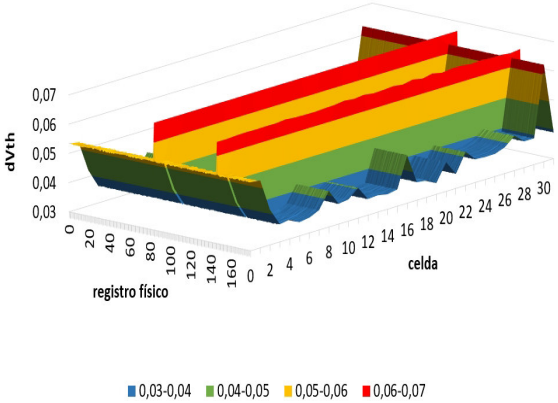
(c) *gobmk*



(d) *omnetpp*



(e) *perlbench*



(f) *xalancbmk*

Figura B.7: dV_{th} por cada celda del banco de registros.