Gestión de Contenidos en Caches Operando a Bajo Voltaje

Alexandra Ferrerón[1], Jesús Alastruey Benedé[1], Darío Suárez Gracia[1], Teresa Monreal Arnal[2], Pablo Ibáñez[1] y Víctor Viñals Yúfera[1]

Resumen— **La eficiencia energética de las caches en chip puede mejorarse reduciendo su voltaje de alimentación (V_{dd}). Sin embargo, este escalado de V_{dd} está limitado a una tensión $V_{dd_{min}}$ por debajo de la cual algunas celdas SRAM (*Static Random Access Memory*) puede que no operen de forma fiable.**

***Block disabling* (BD) es una técnica microarquitectónica que permite operar a tensiones muy bajas desactivando aquellas entradas que contienen alguna celda que no opera de forma fiable, aunque a cambio de reducir la capacidad efectiva de la cache. Se utiliza en caches de último nivel (LLC), donde el ahorro potencial es mayor. Sin embargo, para algunas aplicaciones, el incremento de consumo debido a los accesos a memoria fuera del chip no compensa el ahorro energético conseguido en la LLC.**

Este trabajo aprovecha recursos existentes en los multiprocesadores, como son la jerarquía de memoria en chip y el mecanismo de coherencia, para mejorar las prestaciones de BD. En concreto, proponemos explotar la redundancia natural existente en una jerarquía de cache inclusiva para mitigar la pérdida de rendimiento debida a la reducción en la capacidad de la LLC.

También proponemos una nueva política de gestión de contenidos consciente de la existencia de entradas de cache defectuosas. Utilizando la información de reúso, el algoritmo de reemplazo asigna entradas de cache operativas a aquellos bloques con más probabilidad de ser referenciados.

Las técnicas propuestas llegan a reducir el MPKI hasta en un 36.4 % respecto a block disabling, mejorando su rendimiento entre un 2 y un 13 %.

Palabras clave— **Computación cerca de la tensión umbral, tolerancia a fallos, gestión de contenidos en LLC**

I. Introducción

La densidad de potencia es el principal factor que limita las prestaciones de las tecnologías CMOS actuales. La ley de Moore se sigue cumpliendo, por lo que cada nueva tecnología de fabricación duplica el número de transistores y la escala de integración. Sin embargo, el escalado de Dennard ha llegado a su fin [1], y ya no es posible mantener la densidad de potencia constante de generación en generación. Las restricciones en el consumo impiden la activación simultánea de todos los transistores disponibles, lo que se denomina *dark silicon* [2].

Durante años, la industria ha confiado en la reducción de la tensión de alimentación (V_{dd}) para disminuir el consumo de energía. Pero, desde la generación de 90 nm, esta tendencia ha cambiado, debido principalmente al incremento de las corrientes de fuga. Al reducir V_{dd} hay que reducir asimismo la tensión umbral (V_{th}) para mantener la fiabilidad y velocidad en la conmutación. Sin embargo, una V_{th} baja incrementa el consumo estático de energía en mayor medida que los ahorros obtenidos al disminuir V_{dd}. Reducir la tensión de operación de los transistores a valores cercanos a la tensión umbral sin pérdida de fiabilidad permitiría minimizar los consumos estático y dinámico de energía (corrientes de fuga y conmutación, respectivamente). La reducción de consumo resultante podría usarse para activar más recursos del chip y potencialmente obtener mejores prestaciones [3].

Desafortunadamente, el escalado de la tensión de alimentación está limitado por los ajustados márgenes de operación que tienen las celdas SRAM utilizadas para construir las caches en chip. Las variaciones en los parámetros de las celdas SRAM limitan la reducción del voltaje de alimentación a una tensión, $V_{dd_{min}}$, por debajo de la cual las celdas SRAM no tienen un funcionamiento fiable. $V_{dd_{min}}$ normalmente determina la tensión mínima del procesador, que en las tecnologías actuales es del orden de 0.7–1.0 V cuando se utilizan celdas SRAM convencionales (de 6 transistores o 6T). La fluctuación aleatoria de dopantes (*random dopant fluctuation*, RDF) dentro del dado es una de las principales causas de los fallos en las celdas que funcionan a voltajes ultra bajos, lo que resulta en diferentes tensiones umbral en los dispositivos SRAM considerados. Esto da lugar a una distribución aleatoria de celdas defectuosas cuyo comportamiento a tensiones inferiores a $V_{dd_{min}}$ es equivalente al de un fallo de operación (*hard fault*).

En la literatura se han propuesto diferentes soluciones a los problemas que aparecen cuando la cache opera a baja tensión. A nivel circuital, puede aumentarse la resiliencia de una celda SRAM aumentando el tamaño de sus transistores o añadiendo circuitería especial [4], [5]. Su principal desventaja es que aumentan el área y el consumo del dispositivo. Procesadores comerciales, como por ejemplo los de la familia Intel Nehalem, utilizan celdas con circuitería adicional (celdas SRAM 8T) en las caches de primer nivel para hacerlas más resilientes [6]. Las caches de primer nivel en los multiprocesadores en chip (*chip multiprocessors*, CMPs) ocupan poca superficie para no limitar la frecuencia del procesador. Por otra parte, las caches de último nivel (*last-level caches*, LLCs) tienen tamaños y asociatividades mucho mayores, por lo que ocupan gran parte de la superficie del chip. Por tanto, para construir las LLCs son preferibles

[1]gaZ. Dpto. de Informática e Ing. de Sistemas. Inst. de Investigación en Ing. de Aragón. Universidad de Zaragoza, e-mail: {ferreron, jalastru, dario, imarin, victor}@unizar.es; [2]Dpto. de Arquitectura de Computadores. Universitat Politècnica de Catalunya, email: teresa@ac.upc.edu

las estructuras SRAM formadas por celdas 6T, ya que son más densas que las formadas por celdas más grandes o con circuitería adicional.

A nivel microarquitectónico, los diseños de caches tolerantes a fallos se basan en la deshabilitación de los recursos defectuosos usando distintas granularidades [7], en la corrección de bits defectuosos mediante complejos códigos de corrección de error (*error correction codes*, ECCs) [8], o en la replicación distribuida de bloques [9], [10]. *Block disabling* (BD) es una técnica sencilla que desconecta toda una entrada de cache cuando en ella se detecta al menos un bit defectuoso[1]. Esta técnica se implementa en procesadores modernos para protegerlos contra los *hard faults* [11]. Sin embargo, debido a que las celdas defectuosas se distribuyen de forma aleatoria, la capacidad de la cache queda comprometida. Técnicas complejas basadas en sofisticados ECCs, o la combinación de recursos defectuosos, permiten utilizar más capacidad de cache, pero requieren costes adicionales de almacenamiento y a veces complicados mecanismos de mapeo de direcciones de memoria que penalizan la latencia de acceso a cache.

Este trabajo propone una nueva estrategia para mitigar el impacto de los fallos que causan las variaciones de parámetros en las celdas SRAM de las LLCs. Se basa en la técnica *block disabling* y utiliza estructuras ya existentes en un CMP. Hemos identificado una fuente natural de redundancia de datos dentro del chip que surge por la replicación de bloques en la jerarquía de cache en inclusión y la hemos aprovechado mediante nuevas técnicas de gestión de contenidos.

En este trabajo se hacen las siguientes contribuciones. En primer lugar, evaluamos varias técnicas de deshabilitación de bloques en un CMP coherente, con una jerarquía de memoria compartida de dos niveles, y que ejecuta cargas de trabajo multiprogramadas con un completo y detallado modelo de memoria. En segundo lugar, presentamos una técnica de baja complejidad que, basándose en el mecanismo de coherencia de cache existente, mantiene operativas las etiquetas de las entradas de cache deshabilitadas, y también las funcionalidades de control del directorio de coherencia. De esta forma, un bloque que físicamente no está almacenado en la LLC, puede estar presente, y disponible, en las caches privadas. Finalmente, proponemos una técnica de gestión de contenidos consciente de las entradas defectuosas. Esta técnica predice la utilidad de un bloque basándose en su patrón de uso y controla la asignación de bloques a entradas de cache, por lo que reduce la tasa de fallos de cache sin ningún coste añadido.

El resto de este artículo está organizado como sigue. La sección II presenta el modelo de fallos de celdas SRAM cuando operan a bajo voltaje. La sección III introduce las técnicas de deshabilitación de entradas de cache (*block disabling*) y su impacto en la LLC. En la sección IV proponemos aprovechar la organización de la jerarquía de memoria en inclusión para operar a voltajes bajos y en la sección V exploramos técnicas de gestión de contenidos en este contexto. La sección VI describe la metodología utilizada. En la sección VII se detallan los resultados obtenidos. La sección VIII recoge el trabajo relacionado y, finalmente, la sección IX concluye.

[1]Distinguimos entre bloque y entrada de cache. En este trabajo, bloque es la unidad de correspondencia y transferencia, junto con su contenido; entrada es el conjunto de celdas físicas que almacenan un bloque.

II. Modelo de fallos de celdas SRAM

Las estructuras SRAM son especialmente vulnerables a fallos causados por variaciones paramétricas cuando operan a tensiones de alimentación muy bajas o cercanas al umbral. Normalmente, las estructuras SRAM tienen un voltaje mínimo que garantiza la operación de las celdas de forma fiable, $V_{dd_{min}}$. En la actualidad este voltaje se sitúa entre los 0.7 y 1.0 V cuando se utilizan celdas 6T.

Diversos autores han analizado la fiabilidad de estas celdas por debajo de $V_{dd_{min}}$. En concreto, Zhou *et al.* estudian seis tamaños de celdas SRAM 6T en una escala de integración de 32 nm y sus probabilidades de fallo conforme disminuye V_{dd} [5]. Según este estudio, a 0.5 V la probabilidad de fallo de una celda SRAM (P_{fail}) varía entre 10^{-3} y 10^{-2}. El uso de celdas más grandes reduce la probabilidad de fallo, ya que se incrementan los márgenes de lectura y escritura. Por contra, celdas más grandes tienen peor densidad y se incrementa el consumo de potencia y energía.

La Tabla I recoge las seis celdas SRAM de este estudio (C1, C2, C3, C4, C5 y C6) con su área relativa a la celda más pequeña (C1), así como el porcentaje de entradas sin fallos de una cache implementada con cada celda a 0.5 V, asumiendo que las entradas tienen un tamaño de 64 bytes. Se considera que una entrada falla si al menos una de sus celdas no es operativa.

Como muestra la Tabla I, menos del 10 % de las entradas no tienen fallos cuando se utilizan las celdas C1 o C2 a 0.5 V. Si se utilizan celdas más robustas (e.g., C6), el porcentaje de entradas operativas sube al 60 %, pero con un coste añadido de incrementar el área un 58 % (relativo a C1), lo que no es una opción viable para una estructura tan grande como la LLC.

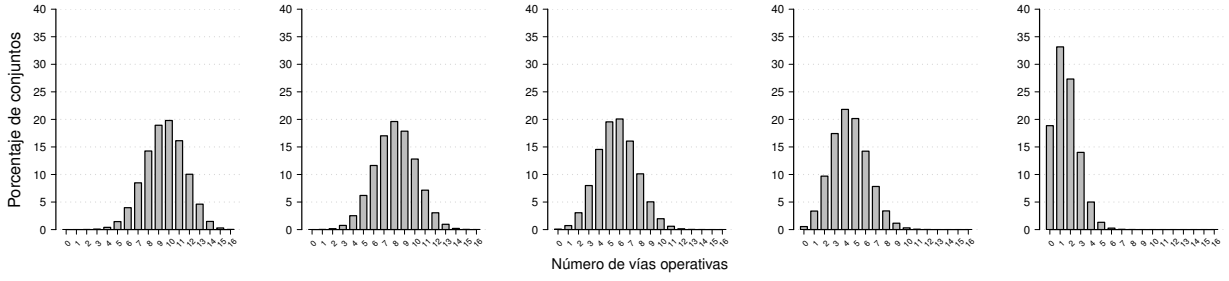
En este artículo utilizamos este estudio como referencia y nos centramos en las celdas C2-C6 a 0.5 V, ya que prácticamente todas las entradas de una cache con celdas C1 son defectuosas.

TABLA I
Porcentaje de entradas sin fallos en una cache a 0.5 V (entradas de 64 bytes) para los distintos tamaños de celdas y su área relativa al tamaño de la celda C1.

Celda	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**
Área relativa	1.00	1.12	1.23	1.35	1.46	1.58
% sin fallos	0.0	9.9	27.8	35.8	50.6	59.9

III. Block Disabling aplicado a las caches de último nivel

Una opción sencilla para mitigar los fallos de operación consiste en deshabilitar aquellos recursos que



(a) C6. (b) C5. (c) C4. (d) C3. (e) C2.

Fig. 1. Vías operativas, no defectuosas, en un banco de cache de 16 vías (celdas C6-C2) con tamaño de bloque 64 bytes a 0.5 V.

no operan de forma fiable. La técnica *block disabling* (BD), deshabilita estos recursos usando granularidad de bloque. Si se detecta un fallo en una celda de memoria, se desactiva la entrada de cache en la que se encuentra, por lo que no podrá almacenar bloque alguno. Esta técnica ya está implementada en procesadores actuales como protección contra *hard faults* [11].

Debido a su sencilla implementación y bajo coste, se ha estudiado la aplicación de BD a caches operando a tensiones bajas [12]. Desde el punto de vista de su implementación, un único bit basta para marcar una entrada de cache como defectuosa. La principal desventaja, tal como se mostró en la Tabla I, es que a medida que la probabilidad de fallo aumenta, la capacidad de la cache disminuye de forma sustancial. La cantidad total de celdas defectuosas en la cache no es elevada (menos del 1 %), pero dada la aleatoriedad en su distribución, la capacidad efectiva de la cache disminuye rápidamente. Utilizar BD como técnica de tolerancia a fallos a tensiones bajas da lugar a caches con una asociatividad variable, que viene determinada por el número y distribución de los fallos.

La interacción entre BD y la organización de la jerarquía de cache en el chip también juega un papel importante en el rendimiento del sistema. Actualmente, algunos procesadores comerciales, como Intel Core i7, implementan jerarquías en inclusión para gestionar la coherencia de una forma sencilla. En tales jerarquías, todos los bloques almacenados en los niveles privados también lo están en la LLC. La información de coherencia va integrada en la propia LLC, es decir, para cada bloque se guarda su estado y un vector binario de presencia para identificar la ubicación de todas sus copias compartidas. Para garantizar la propiedad de inclusión, cuando se expulsa un bloque de la LLC, se invalidan todas la copias del bloque presentes en las caches privadas. A estos bloques invalidados se les llama víctimas por inclusión [13].

Las jerarquías en inclusión no se comportan bien cuando la capacidad agregada de las caches privadas es similar a la de la LLC [14]. BD agrava este problema al disminuir de manera significativa la asociatividad y capacidad de la LLC. La Figura 1 muestra el porcentaje de vías operativas (no defectuosas) en un banco de cache de 16 vías y tamaño de bloque 64 bytes, usando las celdas descritas a 0.5 V. La cantidad de vías defectuosas por conjunto sigue una distribución binomial $B(n, p)$, donde n es la asociatividad p la probabilidad de fallo de una entrada de cache. Esta pérdida de asociatividad se traduce directamente en un aumento importante del número de víctimas por inclusión. Como ejemplo, el número de invalidaciones se multiplica por 10 a 0.5 V en relación a una cache implementada con celdas SRAM libres de fallos.

Este último dato sugiere que las jerarquías en inclusión no son óptimas en sistemas que implementan BD como técnica de tolerancia a fallos en situaciones en las que se dan un gran número de entradas defectuosas. Además, para gestionar la coherencia, sólo es estrictamente necesario disponer de inclusión de directorio: basta con conservar en la cache compartida la etiqueta y el vector de presencia de los bloques existentes en las caches privadas, sin necesidad de tener una copia de los datos [13].*para aquellos bloques ya almacenados en alguna de las caches privadas, basta con guardar su etiqueta en la cache compartida, sin necesidad de tener replicados sus datos* [13]. Esta observación es la base de la propuesta que planteamos en este trabajo.

IV. Aprovechar la coherencia para operar de manera eficiente a bajo voltaje

Block disabling utiliza un bit por entrada de cache para identificar las entradas defectuosas en el *array* de datos (una o más celdas defectuosas). Las entradas defectuosas se excluyen de la búsqueda asociativa en etiquetas y del reemplazo, lo que conlleva una reducción en el número de vías útiles y el incremento de víctimas por inclusión.

Sin embargo, desde el punto de vista de la coherencia, identificar los bloques que se encuentran en caches privadas mediante el almacenamiento de su etiqueta en el *array* de etiquetas de la LLC es suficiente para garantizar la inclusión de directorio. Esta idea es la base de nuestra primera técnica, a la que llamamos *Block Disabling with Operational Tags* (BDOT). Para mantener la inclusión de directorio utilizamos las etiquetas de las entradas defectuosas, de forma que se incluyen en las operaciones de búsqueda y reemplazo. Al habilitar estas etiquetas, la asociatividad vista por los niveles privados de la jerarquía se mantiene. La etiqueta de una entrada defectuosa, si es válida, nos indica que un bloque puede estar presente en las

caches privadas, pero que no se puede almacenar en la cache compartida.

Para poder implementar este esquema, debemos garantizar que no existan defectos en el *array* de etiquetas. Esto puede hacerse por ejemplo utilizando celdas robustas (e.g., celdas con más transistores) o mediante sofisticados mecanismos ECC. Como las etiquetas ocupan un área relativamente pequeña en comparación con los datos (en nuestro diseño, Tabla II, alrededor del 6 %), incrementar el tamaño de la celda un 33 % (si asumimos celdas 8T) tiene un impacto en el área total de la cache del 2 %. Nos decantamos por esta solución porque utilizar complejos ECCs puede incrementar la latencia de acceso a las etiquetas, mientras que utilizar celdas más robustas implica un menor coste añadido. Otros trabajos también utilizan esta alternativa [9], [10].

El protocolo de coherencia también tiene que adaptarse a esta nueva situación, en donde entradas defectuosas sólo contienen etiqueta, información de presencia y estado de coherencia. Desde el punto de vista de la implementación, sólo se necesita un bit que indique si la entrada es defecutosa o no (como en el caso de *block disabling*). Aparecen dos tipos de entradas en la LLC: (a) sólo etiqueta (T), en donde la entrada de datos que le corresponde es defectuosa y sólo se almacena etiqueta, y (b) etiqueta y dato (D), en donde la entrada de datos correspondiente no tiene fallos y se almacena el bloque. Cuando llega una petición de un bloque almacenado en una entrada tipo T, la petición debe enviarse al siguiente nivel de la jerarquía (en este caso fuera del chip). Lo mismo ocurre con bloques sucios, que deben enviarse a memoria una vez se expulsan de la cache privada.

Protegiendo la información del directorio, el incremento en el número de víctimas por inclusión desaparece, ya que la asociatividad de la cache se mantiene. De todas formas, la capacidad de la cache se sigue viendo comprometida, con el consecuente aumento del tráfico fuera del chip.

La principal limitación de este esquema es que la asignación de bloques a entradas de cache no tiene en cuenta el tipo de entrada, es decir, si puede almacenar sólo etiqueta (T) o etiqueta y dato (D). En el caso de que un bloque con un patrón de reúso (es decir, que tras ser expulsado de la L1 el procesador vuelva a solicitarlo en un periodo corto de tiempo) sea asignado a una entrada T, posteriores peticiones a la LLC se redirigen a memoria. Por tanto, es necesario implementar BDOT junto con una política de gestión de contenidos capaz de identificar dichas situaciones, así como de aprovechar las diferencias en cuanto a los dos tipos de contenedores existentes.

V. Asignación de vías basada en reúso operando a voltajes muy bajos

Las políticas de reemplazo para caches convencionales (libres de fallos) no cumplen los requerimientos impuestos por BDOT, ya que implicitamente asumen que todas las entradas pueden almacenar bloques. Esta asunción es falsa en caches con deshabilitación de entradas pero etiquetas operacionales (que implementan BDOT).

En BDOT, cada conjunto de la cache con asociatividad N contiene entradas T que solo pueden almacenar la etiqueta del bloque y entradas D capaces de almacenar tanto etiquetas como datos, siendo $Num_T + Num_D = N$. Para mejorar las prestaciones del sistema es necesario acomodar de manera distinta bloques en ambos tipos de entradas. En concreto, proponemos incorporar a la política de reemplazo dos objetivos relacionados con la gestión diferenciada de entradas T y D:

1. Asignar entradas D a los bloques con mayor probabilidad de ser usados en el futuro.
2. Entre los bloques con alta probabilidad de volver a ser usados en el futuro, asignar entradas D preferiblemente a aquellos bloques no presentes en caches privadas (L1). Una petición sobre un bloque alojado en una entrada T pero presente en una cache L1 puede ser servido dentro del chip. Sin embargo, una petición sobre un bloque alojado en una entrada T no presente en ninguna L1 siempre es servida por la memoria principal, con la consecuente penalización en tiempo y consumo.

Estos nuevos objetivos pueden ser añadidos a cualquier política de reemplazo. En este artículo usaremos *Not Recently Reused*, NRR, como algoritmo de reemplazo base ya que es simple, efectivo, y nuestra propuesta puede aprovechar la información existente en NRR. A continuación describimos el algoritmo NRR. En la subsección V-B se detalla la implementación de nuestra propuesta sobre NRR.

A. Política de reemplazo base

Los algoritmos de reemplazo basados en reúso han demostrado ser eficientes en caches compartidas de último nivel [15]. La localidad de reúso dice que si un bloque ha sido usado por lo menos dos veces, este bloque tiende a ser reusado muchas veces en el futuro cercano; además, indica que los bloques más recientemente reusados son más útiles que los que fueron reusados con anterioridad.

Por otra parte, en las jerarquías inclusivas los bloques solicitados por el procesador se almacenan tanto en las caches privadas como en la cache compartida (LLC). El reemplazo de un bloque de la LLC implica su invalidación en las caches privadas, lo que suele conllevar una merma del rendimiento porque estos bloques están siendo usados por el procesador.

NNR se basa en estas dos observaciones y almacena el estado de todos los bloques presentes en la cache LLC según el diagrama de la Figura 2. Cuando la memoria sirve un bloque al procesador, su estado en LLC pasa a ser NR-C (NonReused, Cached). Si el bloque es expulsado de la cache privada, su estado en LLC pasa a NR-NC (NonReused, NonCached). Y si un procesador vuelve a pedir el bloque, éste se copia a su cache privada y su estado en LLC pasa a R-C (Reused, Cached). Finalmente, cuando el

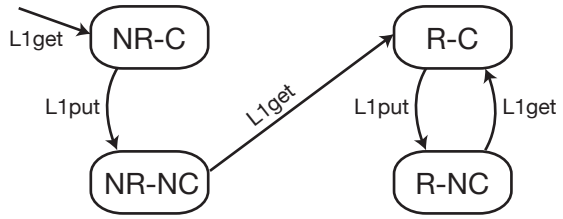


Fig. 2. Estados según reúso e inclusión de un bloque en LLC. NR, R, C, y NC representan: no reusado, reusado, presente en caches privadas, no presente en caches privadas, respectivamente.

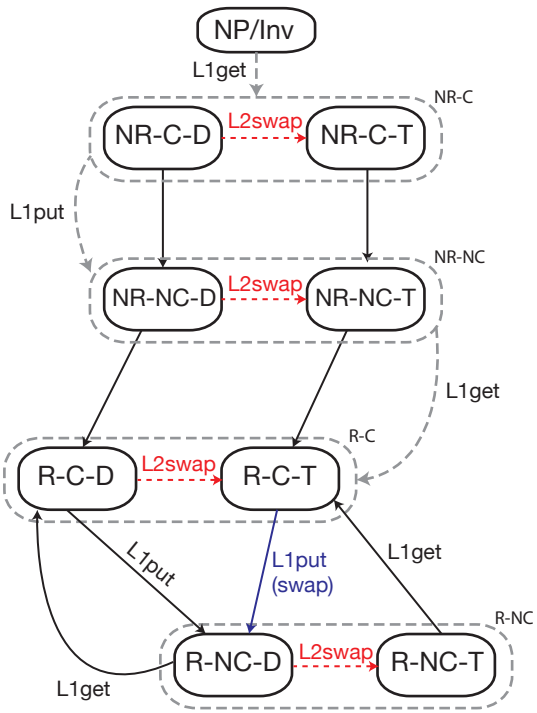


Fig. 3. Estados según reúso e inclusión de un bloque en LLC con BDOT.

bloque es expulsado de la cache privada su estado en LLC pasa a R-NC (Reused, NonCached). Si continua siendo pedido por algún procesador, el bloque seguirá cambiando entre los estados R-NC y R-C.

NRR asigna la máxima protección a los bloques presentes en las caches privadas y entre los no presentes, a aquellos que han demostrado reúso, lo que se traduce en el siguiente orden de reemplazo: NR-NC, R-NC, NR-C, R-C.

La informacion de presencia en caches privadas se obtiene a partir del vector de presencia que gestiona el protocolo de coherencia. La política aquí descrita se suele conocer como política NRR y requiere un bit por cada bloque de la cache para recordar su reúso [15].

B. Política de reemplazo consciente de los fallos

Para dar soporte a BDOT, es necesario desdoblar en dos cada estado de la política de reemplazo base: uno para entradas de etiquetas, T, y otro para entradas de etiquetas y datos, D, tal y como muestra la Figura 3. Mientras un bloque recorre los estados NR-C, NR-NC y R-C, puede estar alojado en una entrada T o D, en función de la vía que inicialmente le asigna el algoritmo de reemplazo. Sin embargo, para proteger a los bloques con reúso, cuando un bloque en estado R-C es expulsado de la cache privada, le aseguramos una entrada no defectuosa tipo D (siempre pasa a estado R-NC-D). Es decir, si residía en una entrada defectuosa tipo T (estado R-C-T), el bloque se mueve a una entrada D.

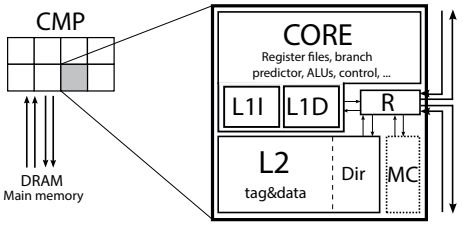


Fig. 4. Multiprocesador de 8 núcleos modelado.

Esta transición normalmente requiere que otro bloque sea degradado de una entrada D a la entrada T que queda libre, lo que implica la pérdida del bloque en la cache de último nivel. Por lo tanto, se requiere otro algoritmo de reemplazo que seleccione un bloque en D para ser degradado a T (transición *swap* en la Figura 3). El objetivo de este algoritmo es maximizar los contenidos dentro del chip, degradando prioritariamente a bloques presentes en caches privadas. Como objetivo secundario degradará preferiblemente a los bloques sin reúso. Por tanto, el orden de mayor a menor probabilidad de ser degradado es: NR-C-D, R-C-D, NR-NC-D, R-NC-D.

VI. Metodología

En esta sección presentamos la metodología utilizada, incluyendo las características del sistema modelado, el entorno de simulación, y las métricas utilizadas para cuantificar los beneficios de nuestros mecanismos.

A. Sistema modelado

Nuestro sistema de referencia es un chip multiprocesador (CMP) teselado con una jerarquía de cache inclusiva de dos niveles. El segundo nivel (L2 o LLC) es compartido y esta distribuido entre los ocho núcleos. Las teselas están interconectadas por medio de una malla. Cada tesela tiene un núcleo y un primer nivel de cache (L1) dividido en instrucciones y datos, y un banco de la cache compartida, todo ello conectado al router (Figura 4). El CMP incluye dos controladores de memoria colocados en los bordes del chip. La Tabla II recoge los parámetros de configuración del sistema de referencia, la jerarquía de memoria y la red de interconexión.

Asumimos una frecuencia de 1 GHz a 0.5 V (nuestro V_{dd} cerca del umbral). El voltaje del módulo de DRAM no escala con el resto del sistema, y por tanto la velocidad relativa de la memoria principal con respecto al chip incrementa conforme se baja el voltaje. Este modelo es consistente con trabajos previos [9], [10].

TABLA II
CARACTERÍSTICAS PRINCIPALES DEL CMP MODELADO.

Núcleos	8, Ultrasparc III Plus, en orden, 1 instr/cycle, mono-hilo, 1 GHz a V_{dd} 0.5 V
Protocolo de Coherencia	MESI, directorio (full-map) distribuído en los bancos de L2
Modelo de Consistencia	Secuencial
L1 cache	Privada, 64 KB datos e inst., 4-way, 64 B tamaño de bloque, LRU, 2 ciclos acceso en acierto
L2 cache	Compartida, inclusiva, entrelazado por dirección de bloque, 1 banco/tesela, 1 MB/banco, 16-way 64 B tamaño de bloque, NRU, 8 ciclos acceso en acierto (4 ciclos acceso etiquetas)
Memoria	2 controladores de memoria (en los bordes del chip); Double Data Rate (DDR3 1333 MHz) 2 canales, 8 Gb/canal, 8 bancos, 8 KB tamaño de página, política *open page*; acceso *raw* 50 ciclos
Red on-chip	Malla, 2 redes virtuales (VN): peticiones y respuestas; 16-byte *flit*, 1 ciclo latencia salto, routing en dos etapas

El protocolo de coherencia se implementa sobre un directorio *full-map* con estados MESI (Modificado, Exclusivo, Compartido e Invalido). Utilizamos notificación explícita de expulsión de bloques tanto compartidos como exclusivos. Las caches L1 están construidas con celdas robustas y funcionan a voltages bajos sin errores inducidos por variaciones paramétricas [6]. Los bancos de L2 están construidos con celdas convencionales (6T SRAM) y, por lo tanto, son sensibles a fallos. Como en estudios previos [9], [10], asumimos que el *array* de etiquetas es robusto utilizando, por ejemplo, celdas 8T [16].

B. Entorno de simulación y cargas de trabajo

Con respecto al entorno de simulación, usamos Simics [17] con GEMS para modelar la jerarquía de memoria on-chip y la red de interconexión [18], y DRAMSim2 para un modelo detallado de DDR3 DRAM [19]. Para modelar latencias utilizamos McPAT [20].

Utilizamos un conjunto de 20 cargas de trabajo multiprogramadas que son combinaciones aleatorias de los 29 programas que componen SPEC CPU 2006 [21], sin distinguir entre los intensivos en cálculo con enteros y con coma flotante. Cada aplicación aparece en media 5.5 veces con una desviación estándar de 2.5. Para localizar el final de la fase de inicialización de cada aplicacion, los programas corrieron con la entrada de referencia en una máquina real usando contadores de rendimiento. Para asegurarnos que ninguna aplicación se encuentra en su fase de inicialización, cada carga multiprogramada ejecutó tantas instrucciones como la fase de inicialización más larga, y se creó un *checkpoint* en ese lugar. A partir de ahí, corremos simulaciones detalladas que incluyen 300 millones de ciclos para calentar la jerarquía de memoria y 700 millones de ciclos de generación de estadísticas.

Creamos mapas de fallos aleatorios y ejecutamos simulaciones Monte Carlo para asegurarnos que los resultados se encuentran dentro de un error del 5 % con un nivel de confianza $(1-\alpha) = 0.95$. El número de muestras se incrementa si es necesario para alcanzar el error. Calculamos si una celda falla o no de forma aleatoria e independientemente del resto de las celdas. De esta manera obtenemos mapas de fallos con un número similar de celdas defectuosas, pero en diferentes localizaciones. Para poder compararnos con *block disabling* en un escenario poco favorable (C2 ó C3), asumimos que al menos una de las vías es robusta (no tiene fallos), aunque nuestras técnicas funcionan incluso si no hay ninguna entrada operativa.

C. Métricas

Por último, para cuantificar los beneficios de nuestros mecanismos, vamos a utilizar dos métricas de rendimiento: fallos de la LLC por cada mil instrucciones (MPKI) y *speedup*.

Nuestro sistema de referencia ($Base$) es un sistema donde la LLC está construida con celdas robustas, es decir, que no presentan fallos al operar a bajo voltaje, y que no tienen ninguna penalización con respecto a las celdas C2. Podemos decir que esta configuración es "no realista", ya que para conseguir una LLC con celdas robustas necesitaríamos incrementar el tamaño de las celdas o utilizar celdas con más transistores. Esta configuración representa una cota máxima del rendimiento alcanzable.

El MPKI nos muestra el aumento del tráfico a la memoria fuera de chip. Para calcular el MPKI relativo de cada mezcla, sumamos el número total de fallos (peticiones a memoria) en la LLC para una configuración dada y lo dividimos entre el MPKI del sistema base.

Para entender cómo afectan las diferentes técnicas al rendimiento global del sistema, calculamos el *speedup* para cada mezcla como se muestra en la Ecuación 1, donde IPC_i^A es el número de instrucciones ejecutadas por ciclo para el programa i cuando corre en el sistema A, e IPC_i^{Base} es el número de instrucciones ejecutadas por ciclo cuando corre en el sistema de referencia $Base$.

$$Speedup = \sqrt[n]{\prod_{i=1}^{n} \frac{IPC_i^A}{IPC_i^{Base}}} \quad (1)$$

Como las diferentes cargas de trabajo ejecutan el mismo número de ciclos tanto en el sistema Base como en las distintas configuraciones, utilizamos la media aritmética para calcular el rendimiento medio (MPKI y *speedup*) de todas las mezclas para una configuración dada.

VII. RESULTADOS

En esta sección presentamos los resultados de la evaluación de nuestras técnicas de gestión de contenidos utilizando la metodología expuesta en la sección VI.

Como ya hemos comentado en la sección VI, ofrecemos datos relativos a un sistema base que posee una

LLC ideal, implementada con celdas robustas, capaces de operar sin fallos a muy bajo voltaje sin penalización de ningún tipo. Las figuras muestran la media de los resultados obtenidos para cada una de las 20 mezclas. Añadimos como referencia los resultados obtenidos por *block disabling* y reemplazo NRU [22], solución con coste similar a nuestras propuestas. También nos comparamos con *Word disabling* [10], solución más compleja y de mayor coste.

En la subsección VII-A se presentan los datos de MPKI y en la subsección VII-B se analiza el *speedup*.

En cada sección mostramos datos de los siguientes modelos:

- BASE: sistema con LLC construida con celdas robustas. Es nuestra base de comparación (todos los datos son relativos a este sistema).
- BD-NRU: sistema real con *block disabling*, reemplazo NRU [22] (sección III).
- BDOT-NRU: sistema real con *block disabling with operational tags*, reemplazo NRU (sección IV).
- BDOT-NRR: sistema real con *block disabling with operational tags*, reemplazo NRR [15](sección V-A).
- BDOT-NRR-FA: sistema real con *block disabling with operational tags*, reemplazo NRR Fault Aware (sección V-B)
- WD: Word disabling [10].

A. Análisis de resultados en términos de MPKI

En esta sección presentamos, para cada configuración de LLC, los datos de MPKI normalizados al sistema BASE. La Figura 5 muestra la media para las 20 mezclas descritas en sección VI.

BD-NRU puede ser una solución válida para caches con pocos bloques defectuosos como las construidas con celdas C6, en las que produce una penalización media en términos de MPKI del 21,4 %. Sin embargo, esta penalización aumenta rápidamente al aumentar el número de bloques defectuosos llegando a un 126 % para celdas C2. El uso de las entradas defectuosas de LLC para guardar la información de coherencia de bloques presentes en los niveles privados, permite a BDOT-NRU disminuir significativamente el número de fallos respecto a BD-NRU en caches con muchos bloques defectuosos (C2). Pero, BDOT-NRU va peor que BD-NRU para el resto de celdas.

Los algoritmos de reemplazo basados en reúso han demostrado ser eficientes en caches compartidas de último nivel [15]. Sin embargo, y salvo para C2, NRR no consigue mejorar los resultados de NRU en el contexto de caches con entradas defectuosas (BDOT-NRR vs. BDOT-NRU). Recordemos que, en este modelo, la asignación de bloques a entradas es ciega, solo depende del algoritmo de reemplazo NRR y no es consciente de qué entradas están sanas y cuales son defectuosas. La asignación de un bloque con reúso a una entrada de LLC defectuosa provoca que todas las peticiones a dicho bloque dirigidas a LLC se conviertan en peticiones a memoria principal. Además, debido al reemplazo basado en reúso, este bloque permanecerá mucho tiempo en la entrada defectuosa.

BDOT-NRR-FA soluciona este problema añadiendo la información de bloques defectuosos al algoritmo de reemplazo/colocación. La penalización en términos de MPKI disminuye drásticamente respecto a BD-NRU, pasando de 21,4 %, 27,6 %, 40,3 %, 52,1 % y 126 % en BD-NRU para las celdas C6, C5, C4, C3 y C2 respectivamente, a 4,6 %, 9,3 %, 18,7 %, 25 % y 43,8 % en BDOT-NRR-FA.

El MPKI conseguido por WD es casi invariante entre las distintas celdas, pese a la gran diferencia de celdas defectuosas en cada una de ellas. Dos razones explican este comportamiento: i) una sola celda defectuosa clasifica al bloque entero como defectuoso y obliga a combinarlo con otro, y ii) el número de celdas defectuosas por bloque es siempre relativamente pequeño (tres en media para la peor celda: C2 [23]), por lo que siempre se puede conseguir un bloque bueno combinando dos defectuosos. Por consiguiente, el número medio de vías por conjunto conseguido por WD es de 8. En la comparación con WD, BDOT-NRR-FA consigue menor MPKI con celdas C3, C4, C5 y C6 con diferencias de 19 %, 15,3 %, 8 % y 3,2 % respectivamente. WD solo supera a BDOT-NRR-FA en caches con muchos bloques defectuosos como las fabricadas con celdas C2. Además, la complejidad y el coste en área de BDOT-NRR-FA son mucho menores que los de WD.

B. Análisis de resultados en términos de speedup

En esta sección presentamos, para cada configuración de LLC, los datos de *speedup* normalizados al sistema BASE. La Figura 6 muestra la media para las 20 mezclas descritas en la seccion VI. Los resultados en términos de *speedup* correlan perfectamente con los de MPKI analizados en la sección previa. BD provoca pérdidas respecto a la cache ideal que varían entre el 3,3 % para C6 y el 17.7 % para C2. BDOT-NRR-FA reduce las pérdidas hasta el 1,3 %, 2 %, 3,4 %, 4,3 % y 6,9 % para las celdas C6, C5, C4, C3 y C2 respectivamente. Al igual que en MPKI, WD solo mejora a BDOT-NRR-FA con la celda C2 en un 2 %.

VIII. Trabajo Relacionado

Las celdas SRAM con 6 transistores no funcionan de forma fiable cuando el voltaje escala a valores cerca de la tensión umbral, ya que no se pueden garantizar los márgenes necesarios para la estabilidad de escrituras y lecturas, especialmente en un contexto de V_{th} variable. Las propuestas previas para mitigar el impacto de los fallos de SRAM debidos a variaciones paramétricas se pueden dividir en dos grupos: técnicas a nivel de circuito y técnicas a nivel arquitectónico.

Las soluciones a nivel de circuito incluyen métodos que hacen que la celda sea más fiable incrementando su tamaño [5], o añadiendo circuitería adicional (i.e., más transistores) [16], [4]. El principal problema de incrementar el tamaño de los transistores o el número de transistores por celda es el coste en área (disminuye la densisdad) y el aumento del consumo. Por tanto,

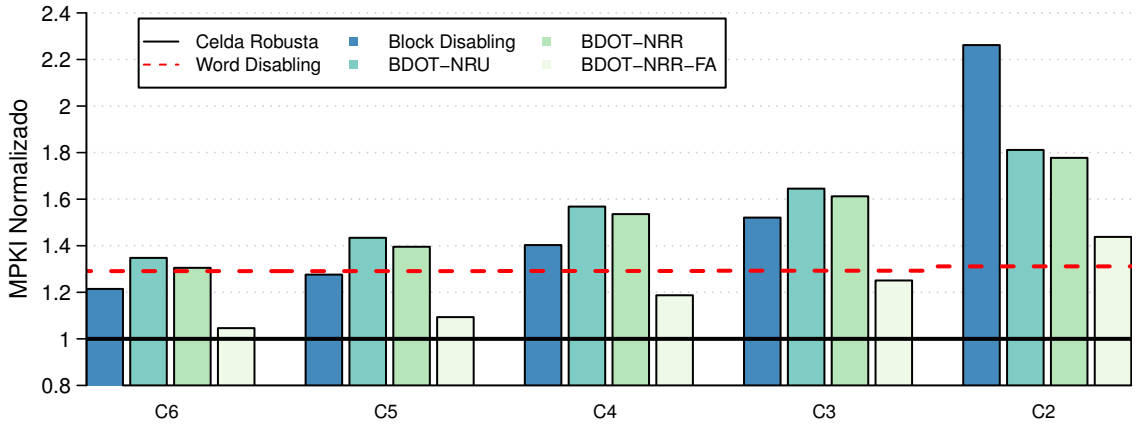


Fig. 5. MPKI normalizado (media cargas SPEC) con respecto a una celda robusta para las distintas propuestas.

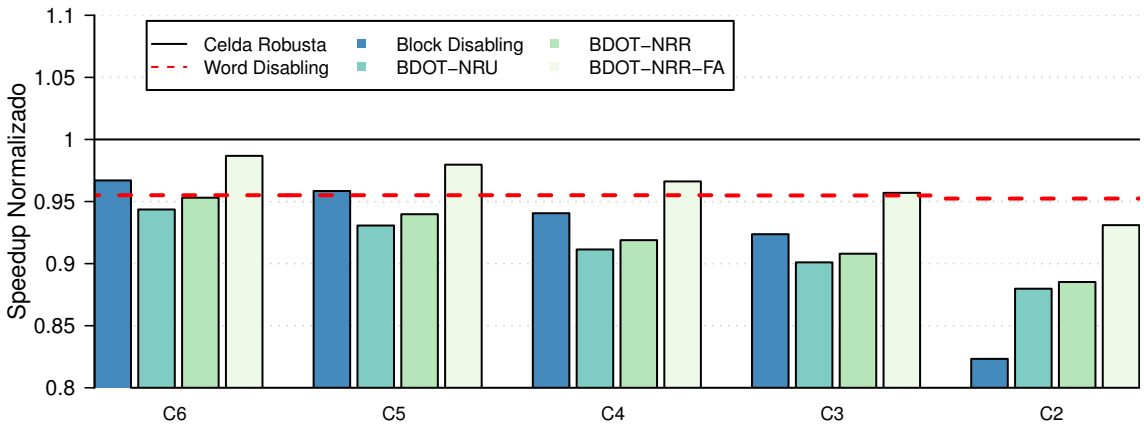


Fig. 6. Speedup normalizado (media cargas SPEC) con respecto a una celda robusta para las distintas propuestas.

estás técnicas no se suelen aplicar en la LLC.

Las soluciones a nivel de micro-arquitectura incluyen técnicas que añaden redundancia a través de códigos de corrección de errores, deshabilitar contenido, o mecanismos basados en la duplicación (y combinación). Nuestra propuesta se enmarca en esta categoría.

Los códigos de correción de errores (ECC) se utilizan para protección ante errores *soft*. Algunos trabajos proponen utilizar ECC para protección ante errores *hard* cuando las caches operan a voltajes ultrabajos [24], [8]. Los ECC se optimizan normalmente para minimizar el espacio que ocupan a costa de una lógica más compleja en las fases de detección y corrección de errores. Modificar el ECC para que sea capaz de detectar y corregir más errores implica lógica más compleja en la fase de decodificación, o incrementar el número de bits almacenados [8]. Nuestra propuesta es orotogonal al uso de ECC para proporcionar más entradas funcionales; de hecho se adapta al número de entradas de datos funcionales y no funcionales.

En cuanto a *block disabling*, Lee *et al.* estudian la degradación en rendimiento al deshabilitar entradas de cache, conjuntos, vías, puertos, o el banco de cache al completo, en un entorno mono procesador [7]. Ladas *et al.* proponen añadir una cache víctima para compensar por la asociatividad perdida [12]. Nuestro mecanimo también se basa en *block disabling*, pero no requiere ninguna estructura adicional.

Ghasemi *et al.* utilizan tamaños de celda heterogéneos, de modo que cuando la cache opera a voltajes bajos las vías o conjuntos con celdas SRAM pequeñas se desactivan cuando empiezan a fallar [25]. En la misma línea, Khan *et al.* proponen el diseño de una cache en donde una pequeña partición se implementa con celdas robustas y el resto con celdas no robustas (de menor tamaño). Las celdas robustas se utilizan para almacenar bloques sucios, para lo que modifican la política de reemplazo, guiando la inserción de los bloques en función de si la petición es un *load* o un *store* [26]. Zhou *et al.* combinan celdas de repuesto, tamaños de celdas heterogéneos y ECC para crear un diseño híbrido que mejora la eficiencia de dichas técnicas aplicadas de forma individual [5]. Al contrario que estos trabajos, nosotros no asumimos que existen celdas robustas y guiamos la inserción de bloques basandonos en el reúso.

Las entradas de la cache se pueden deshabilitar a una granularidad más fina (por ejemplo, a nivel de palabra), a cambio de incrementar la complejidad de

acceso a cache. *Word disabling* monitoriza fallos a nivel de palabra y combina dos entradas consecutivas para obtener una entrada sin fallos, lo que divide entre dos la asociatividad y la capacidad de la cache resultante [10]. Abella *et al.* también deshabilitan contenido a nivel de palabra, redirigiendo los accesos a palabras deshabilitadas al siguiente nivel de la jerarquía; esta técnica tiene sentido sólo en el primer nivel de la jerarquía en donde los accessos son a nivel de palabra [27]. Palframan *et al.* siguen una política similar buscando palabras almacenadas en celdas defectuosas en otras estructuras micro-arquitectónicas, como por ejemplo la *store queue* o el *miss status holding register* [28]. Ferreron *et al.* proponen comprimir los bloques y almacenarlos en entradas con fallos, lo que permite utilizar la totalidad de la cache [23]. Esquemas más complejos combinan entradas con defectos utilizando un mecanismo de remapeo que se basa en la ejecución de un algoritmo para encontrar entradas que se puedan combinar y las correspondientes estructuras para almacenar el remapeo [9], [29], [30]. Los mecanismos de remapeo añaden un nivel de indirección al acceso a cache (incrementando la latencia), y combinar entradas para obtener un bloque añade complejidad. Además, se necesitan varios accesos a la cache para obtener un bloque completo, lo que aumenta el consumo de energía y/o la latencia por acceso a bloque. Nuestra propuesta no necesita ninguna estructura adicional o mecanismo de remapeo, sólo cambios mínimos en el protocolo de coherencia y la política de reemplazo.

En el contexto de ejecución a voltajes ultra-bajos, Keramidas *et al.* usan un predictor espacial indexado por PC para las decisiones de reemplazo entre entradas con o sin fallos en el primer nivel de la jerarquía [31]. Nuestro mecanismo de inserción no necesita ninguna predicción, lo que simplifca el hardware de manera considerable, y no contempla el uso de entradas parcialmente deshabilitadas.

IX. Conclusiones

Uno de los mayores desafíos para continuar mejorando el rendimiento y la eficiencia energética de los procesadores es reducir V_{dd}. El mayor freno a la hora de hacerlo son las celdas de memoria ya que pequeñas disminuciones de V_{dd} rápidamente las vuelven inestables e inválidas para almacenar cualquier contenido.

Disminuir la capacidad de las memorias caches mediante la invalidación de celdas permite reducir V_{dd} a costa de empeorar el rendimiento y la eficiencia energética del procesador porque aumenta considerablemente la actividad de la memoria principal. Sin embargo si se explota adecuadamente la replicación de contenidos en jerarquías de memoria inclusiva se puede paliar este problema. Además, con politicas de gestión de contenidos conscientes de los fallos en celdas se puede casi alcanzar el rendimiento de un sistema trabajando con un V_{dd} mayor a la par que se reduce el consumo energético. Este trabajo fusiona ambas propuestas con un aprovechamiento inteligente de los mecanismos de coherencia existentes para permitir operar a la memoria a un V_{dd} bajo.

En un entorno de un multiprocesador homogeneo las técnicas propuestas consiguen mejorar *block disabling* reduciendo los MPKI hasta un 36,4 %, lo que se traduce en una ganancia de rendimiento del 13 %. Respecto a una celda ideal, la pérdida de rendmiento se encuentra entre el 1,2 y el 6,9 %, frente al 3,3 y el 17.7 % de *block disabling*.

Agradecimientos

Este trabajo ha sido financiado en parte por los proyectos gaZ: T48 grupo de investigación (Gobierno de Aragón y Unión Europea) y TIN2013-46957-C2-1-P, TIN2015-65316-P, y Consolider NoE TIN2014-52608-REDC (Gobierno de España y Unión Europea).

Referencias

[1] R. H. Dennard, F. H. Gaensslen, Hwa-Nien Yu, V. L. Rideout, E. Bassous, and A. R. Leblanc, "Design of ion-implanted MOSFET's with very small physical dimensions," *Proceedings of the IEEE*, vol. 87, no. 4, pp. 668–678, April 1999.

[2] M.B. Taylor, "A landscape of the new dark silicon design regime," *IEEE Micro*, vol. 33, no. 5, pp. 8–19, Sept. 2013.

[3] R.G. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge, "Near-threshold computing: Reclaiming moore's law through energy efficient integrated circuits," *Proc. of the IEEE*, vol. 98, no. 2, pp. 253–266, Feb. 2010.

[4] J.P. Kulkarni, Keejong Kim, and K. Roy, "A 160mV robust schmitt trigger based subthreshold SRAM," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 10, pp. 2303–2313, Oct. 2007.

[5] Shi-Ting Zhou, S. Katariya, H. Ghasemi, S. Draper, and Nam Sung Kim, "Minimizing total area of low-voltage SRAM arrays through joint optimization of cell size, redundancy, and ECC," in *IEEE Int. Conf. on Computer Design*, 2010, pp. 112–117.

[6] R. Kumar and G. Hinton, "A family of 45nm IA processors," in *IEEE Int. Solid-State Circuits Conf. Digest of Technical Papers*, 2009, pp. 58–59.

[7] Hyunjin Lee, Sangyeun Cho, and B.R. Childers, "Performance of graceful degradation for cache faults," in *IEEE Computer Society Annual Symp. on VLSI*, 2007, pp. 409–415.

[8] Zeshan Chishti, Alaa R Alameldeen, Chris Wilkerson, Wei Wu, and Shih-Lien Lu, "Improving cache lifetime reliability at ultra-low voltages," in *42nd Annual IEEE/ACM Int. Symp. on Microarchitecture*, 2009, pp. 89–99.

[9] A. Ansari, Shuguang Feng, S. Gupta, and S. Mahlke, "Archipelago: A polymorphic cache design for enabling robust near-threshold operation," in *IEEE 17th Int. Symp. on High Performance Computer Architecture*, 2011, pp. 539–550.

[10] Chris Wilkerson, Hongliang Gao, Alaa R. Alameldeen, Zeshan Chishti, Muhammad Khellah, and Shih-Lien Lu, "Trading off cache capacity for reliability to enable low voltage operation," in *35th Annual Int. Symp. on Computer Architecture*, 2008, pp. 203–214.

[11] Jonathan Chang, Ming Huang, J. Shoemaker, J. Benoit, Szu-Liang Chen, Wei Chen, Siufu Chiu, Raghuraman Ganesan, G. Leong, Venkata Lukka, S. Rusu, and D. Srivastava, "The 65-nm 16-MB Shared On-Die L3 Cache for the Dual-Core Intel Xeon Processor 7100 Series," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 4, pp. 846–852, Apr. 2007.

[12] N. Ladas, Y. Sazeides, and V. Desmet, "Performance-effective operation below Vcc-min," in *IEEE Int. Symp. on Performance Analysis of Systems Software*, 2010, pp. 223–234.

[13] J. L. Baer and W.H. Wang, "On the inclusion properties for multi-level cache hierarchies," in *15th Annual Int. Symp. on Computer Architecture*, 1988, pp. 73–80.

[14] Aamer Jaleel, Eric Borch, Malini Bhandaru, Simon C. Steely Jr., and Joel Emer, "Achieving non-inclusive cache performance with inclusive caches: Temporal locality aware (tLA) cache management policies," in *43rd Annual*

IEEE/ACM Int. Symp. on Microarchitecture, 2010, pp. 151–162.

[15] Jorge Albericio, Pablo Ibáñez, Víctor Viñals, and Jose María Llabería, "Exploiting reuse locality on inclusive shared last-level caches," *ACM Trans. Archit. Code Optim.*, vol. 9, no. 4, pp. 38:1–38:19, Jan. 2013.

[16] L. Chang, R.K. Montoye, Y. Nakamura, K.A. Batson, R.J. Eickemeyer, R.H. Dennard, W. Haensch, and D. Jamsek, "An 8T-SRAM for variability tolerance and low-voltage operation in high-performance caches," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 4, pp. 956–963, Apr. 2008.

[17] P.S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," *Computer*, vol. 35, no. 2, pp. 50–58, Feb. 2002.

[18] Milo M. K. Martin, Daniel J. Sorin, Bradford M. Beckmann, Michael R. Marty, Min Xu, Alaa R. Alameldeen, Kevin E. Moore, Mark D. Hill, and David A. Wood, "Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) toolset," *SIGARCH Comput. Archit. News*, vol. 33, no. 4, pp. 92–99, Nov. 2005.

[19] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A cycle accurate memory system simulator," *Computer Architecture Letters*, vol. 10, no. 1, pp. 16–19, Jan. 2011.

[20] Sheng Li, Jung Ho Ahn, Richard D Strong, Jay B Brockman, Dean M Tullsen, and Norman P Jouppi, "McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *42nd Annual IEEE/ACM Int. Symp. on Microarchitecture*, 2009, pp. 469–480.

[21] John L. Henning, "SPEC CPU2006 benchmark descriptions," *SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 1–17, Sept. 2006.

[22] Sun Microsystems, "UltraSPARC T2 supplement to the UltraSPARC architecture," Draft d1.4.3, Sun Microsystems Inc., 2007.

[23] A. Ferreron, D. Suarez, J. Alastruey, T. Monreal, and P. Ibañez, "Concertina: Squeezing in cache content to operate at near-threshold voltage," *IEEE Trans. on Computers*, vol. 65, no. 3, pp. 755–769, Mar. 2016.

[24] A.R. Alameldeen, I. Wagner, Z. Chishti, Wei Wu, C. Wilkerson, and Shih-Lien Lu, "Energy-efficient cache design using variable-strength error-correcting codes," in *38th Annual Int. Symp. on Computer Architecture*, 2011, pp. 461–471.

[25] H.R. Ghasemi, S.C. Draper, and Nam Sung Kim, "Low-voltage on-chip cache architecture using heterogeneous cell sizes for high-performance processors," in *IEEE 17th Int. Symp. on High Performance Computer Architecture*, 2011, pp. 38–49.

[26] Samira M. Khan, Alaa R. Alameldeen, Chris Wilkerson, Jaydeep Kulkarni, and Daniel A. Jimenez, "Improving multi-core performance using mixed-cell cache architecture," in *IEEE 19th Int. Symp. on High Performance Computer Architecture*, 2013, pp. 119–130.

[27] Jaume Abella, Javier Carretero, Pedro Chaparro, Xavier Vera, and Antonio González, "Low Vccmin fault-tolerant cache with highly predictable performance," in *42nd Annual IEEE/ACM Int. Symp. on Microarchitecture*, 2009, pp. 111–121.

[28] David J. Palframan, Nam Sung Kim, and Mikko H. Lipasti, "iPatch: Intelligent fault patching to improve energy efficiency," in *IEEE 21st Int. Symp. on High Performance Computer Architecture*, 2015, pp. 428–438.

[29] Cheng-Kok Koh, Weng-Fai Wong, Yiran Chen, and Hai Li, "Tolerating process variations in large, set-associative caches: The buddy cache," *ACM Trans. on Architecture and Code Optimization*, vol. 6, no. 2, pp. 8:1–8:34, July 2009.

[30] T. Mahmood, Soontae Kim, and Seokin Hong, "Macho: A failure model-oriented adaptive cache architecture to enable near-threshold voltage scaling," in *IEEE 19th Int. Symp. on High Performance Computer Architecture*, 2013, pp. 532–541.

[31] Georgios Keramidas, Michail Mavropoulos, Anna Karvouniari, and Dimitris Nikolos, "Spatial pattern prediction based management of faulty data caches," in *Conference on Design, Automation & Test in Europe*, 2014, pp. 60:1–60:6.