UNIVERSIDAD DE MURCIA

FACULTAD DE INFORMÁTICA

Mejora de la Eficiencia Energética de Sistemas de Memoria Transaccional Hardware para Arquitecturas Multinúcleo

D. Epifanio Gaona Ramírez

2015

Universidad de Murcia
Facultad de Informática

Mejora de la Eficiencia Energética de Sistemas de Memoria Transaccional Hardware para Arquitecturas Multinúcleo

Tesis propuesta para la
obtención del grado de

Doctor en Informática

Presentada por
Epifanio Gaona Ramírez

Supervisada por
Manuel Eugenio Acacio Sánchez
Juan Fernández Peinador

Murcia, Septiembre de 2015

UNIVERSIDAD DE MURCIA
Facultad de Informática

Improving Energy Efficiency in Hardware Transactional Memory Systems for Multicore Architectures

A dissertation submitted in fulfillment of
the requirements for the degree of

DOCTOR EN INFORMÁTICA

By
Epifanio Gaona Ramírez

Advised by
Manuel Eugenio Acacio Sánchez
Juan Fernández Peinador

Murcia, Sept 2015

A Elena y Alejandro

Agradecimientos

Muchas han sido las experiencias que he vivido y me han enriquecido desde que decidiera comenzar los estudios de doctorado. Todas y cada una de ellas han sido siempre en contacto con personas con las que estoy muy agradecido y siento que sin ellas la culminación de esta tesis no hubiera sido posible. Es para mi un verdadero placer aprovechar este espacio para expresarles mi más sincera gratitud.

Deseo empezar agradeciendo a mis padres, Epi y Paqui, y a mi hermano Antonio por todo su amor y cariño y por esos momentos de desconexión que me han permitido continuar con esta larga aventura doctoral con gran tranquilidad y paz mental.

Agradezco a mis directores de tesis, Juan y Manolo por aceptarme como doctorando, por su guía y su consejo. No han sido pocos los callejones sin salida que esta tesis ha esquivado gracias a su buen criterio y dedicación. A Manolo me gustaría agradecerle especialmente su psicología y gran trato humano que ha permitido la finalización de esta tesis, literalmente.

A mis amigos y compañeros de doctorando y departamento: Ana, Antonio, Toni, Ginés, Dani, Juanma, Chema, Alberto y Ricardo. Sin ellos no hubiera sido posible el gran ambiente casi familiar que he tenido la suerte de vivir en el departamento DITEC. Quiero agradecer además a José Luis por ser un gran apoyo, ya incluso desde el proyecto fin de carrera, por ser una fuente de ideas y por su inestimable ayuda como coautor de GCommit. No puedo dejar pasar tampoco a Rubén. No sólo eres una excelente persona sino que gracias a ti la curva de aprendizaje del simulador se hizo más suave y además has tenido un rol protagonista durante el desarrollo de esta tesis, también como coautor. Llegados a este punto me acuerdo de Joaquín, gran amigo y compañero durante la carrera e inicios de estudios doctorales.

Mi estancia en Göteborg ha sido sin duda una experiencia tremendamente

enriquecedora. Doy las gracias al profesor Per Stenström por acogerme en su departamento en Chalmers, lo que a su vez me ha permitido conocer Suecia y disfrutarla de una manera única.

A mis actuales compañeros de trabajo Juanjo, Antonio y Fran por soportar con resignación mi estrés estas ultimas semanas.

Agradezco también de corazón a mis amigos: Andrea, Peter, José Miguel, María, Toro, Esther, Julián, Jorge y Marta por servirme de distracción y apoyo durante estos años.

Quiero agradecer también a la Fundación Séneca (Agencia de Ciencia y Tecnología de la Región de Murcia) por su vital apoyo económico mediante la Beca-Contrato Predoctoral de Formación de Personal Investigador (FPI) 09503/FPI/08 y especialmente a Viviane.

Por último dar mi más profundo agradecimiento a mi mujer Elena y por supuesto a nuestro pequeño Alejando. Gracias Elena, por acompañarme en este largo viaje (y los que restan), por tu amor, apoyo y comprensión sin los que no hubiera sido posible la culminación con éxito de esta tesis. Gracias Alejandro, eres la luz de cada mañana.

Resumen

Las actuales arquitecturas de procesador con múltiples núcleos de ejecución (arquitecturas multinúcleo o CMP) se presentan con un gran reto difícil de superar para la mayoría de los programadores. El clásico modelo de programación secuencial sigue siendo aplicable en un CMP, sin embargo mediante su empleo no se estarían aprovechando los múltiples núcleos de procesamiento de que disponen estos chips. Para poder extraer todo el rendimiento posible de un CMP es necesario recurrir a la programación paralela y al uso de varios hilos de ejecución. A través de ellos se consigue exprimir el paralelismo a nivel de hilo (TLP) que estas arquitecturas explotan.

El modelo de Memoria Transaccional (TM) está siendo promovido actualmente como un nuevo paradigma mediante el cual los programadores pueden desarrollar sus aplicaciones de forma cercana al paradigma secuencial al que están acostumbrados, pero habilitando los beneficios del uso de varios núcleos de ejecución. La identificación en el código de las secciones críticas es la única condición nueva que debe satisfacerse. Enmarcando estas secciones en bloques de código denominados transacciones, el sistema de Memoria Transaccional (TM) se encarga de asegurar las propiedades de atomicidad y aislamiento a pesar de la ejecución de los mismos en paralelo. Todo ello de forma totalmente transparente al programador, que no debe preocuparse por los problemas típicos achacados al uso de los cerrojos clásicos (interbloqueo, convoying e inversión de prioridades). En esta tesis doctoral se presta atención a las implementaciones hardware del modelo de Memoria Transaccional (HTM).

Por otra parte, la búsqueda del rendimiento de forma aislada ha dejado de ser la principal motivación en el desarrollo de procesadores para pasar a serlo el rendimiento por watio. El rendimiento sigue siendo importante pero no a costa de un consumo desmesurado de energía. El objetivo de esta tesis es abordar el diseño de sistemas HTM teniendo en cuenta tanto rendimiento como consumo de

energía. Para ello, se analiza con detalle en primer lugar el rendimiento y consumo energético que las implementaciones hardware actuales ofrecen (*eager-eager* y *lazy-lazy*). Los resultados de este estudio comparativo permiten identificar las ventajas e inconvenientes de cada uno de estos sistemas de memoria transaccional y son el punto de partida para proponer técnicas encaminadas a mejorar los aspectos problemáticos que cada sistema presenta.

Por un lado, para los sistemas *eager-eager* se ha observado que una fracción importante de la energía consumida es debida a los conflictos que surgen entre transacciones. Se han desarrollado, implementado y evaluado diversas políticas que permiten serializar en tiempo de ejecución transacciones que entran en conflicto sólo cuando éste se produce sin que se produzcan ciclos de que den lugar a interbloqueos. El resultado final ha sido un sistema *eager-eager* más eficiente desde el punto de vista del consumo de energía sin pérdida de prestaciones. Por otro lado, el principal problema en los sistemas *lazy-lazy* se encuentra en la etapa de confirmación de las transacciones, más concretamente en las operaciones que son necesarias para garantizar que una transacción ha finalizado sin conflictos. Para solucionarlo se ha implementado y evaluado una técnica hardware que permite acelerar, mejorar la eficiencia energética y simplificar la principal propuesta diseñada hasta ahora en sistemas HTM escalables. Concretamente se ha propuesto emplear una red específica de bajo coste y un protocolo implementado en hardware.

A lo largo de la tesis se hace uso de un entorno de desarrollo y pruebas ampliamente aceptado, conformado por la herramienta de simulación SIMICS+GEMS y la suite de benchmarks STAMP. Los resultados de esta tesis ofrecen una visión amplia sobre el compromiso entre rendimiento-energía de sistemas HTM sofisticados.

Abstract

Current microprocessor architectures with multiple execution cores (Chip MultiProcessors or CMP) present a great challenge difficult to overcome for most programmers. Although, the classic model of sequential programming remains applicable in CMPs, developing applications in such way would hinder taking advantage of the multiple processor cores available in these chips. To extract all the possible performance of a CMP it is necessary to use parallel programing and multiple threads, through which we are able to squeeze the thread-level parallelism (TLP) that these architectures exploit.

Transactional Memory (TM) is currently being promoted as a new paradigm that provides applications' programmers a programming model that is closest to the sequential paradigm they are familiar with, and at the same time enabling the benefits of using multiple execution cores. The identification of critical sections in code is the only new condition that must be satisfied. By framing these sections in blocks of code called transactions, the underlying Transactional Memory (TM) system is able to ensure the atomicity and isolation properties despite transactions are executed in parallel. All of this is completely transparent to the programmer, who should not worry about the typical problems blamed on the use of classical locks (deadlock, convoying and priority inversion). This thesis focuses on hardware implementations of the Transactional Memory model (HTM).

Moreover, the pursuit of performance in isolation is no longer the main motivation in the development of processors, and nowadays, performance per watt constitutes the new target. Performance remains important but not at the cost of excessive energy consumption. The goal of this thesis is to address the design of HTM systems considering both performance and energy consumption. To do so, we first analyze in detail performance and power consumption that current HTM implementations offer (*eager-eager* and *lazy-lazy*). The results of

this comparative study allow us to identify the benefits and drawbacks of these two transactional memory systems, and they are the starting point to propose techniques aimed at alleviating the main problems each system has.

In *eager-eager* systems, we have noticed that a substantial fraction of the energy consumed is due to conflicts between transactions. In order to minimize such conflicts, we have developed, implemented and evaluated several policies that identify and serialize transactions that will probably conflict, all of this avoiding cycles leading to potential deadlocks. The final result has been a more efficient *eager-eager* system from the energy consumption point of view without suffering any performance loss. On the other hand, the main problem in the *lazy-lazy* systems is at the commit stage of the transactions, specifically in the operations that are necessary to ensure that a transaction can be completed without any conflicts. To solve this, we have implemented and evaluated a hardware technique that speeds up performance, improves energy efficiency and simplifies the main proposal designed so far in scalable HTM systems. In particular, we have proposed to use a low cost specific network and a protocol implemented in hardware that uses it.

Throughout this thesis we make use of a widely accepted development and evaluation environment, consisting in the simulation tool known as SIM-ICS+GEMS and the STAMP benchmarks suite. The results reported in this thesis provide a broad view of the trade-off between performance and energy consumption in sophisticated HTM systems.

Índice general

Capítulo 1

Introducción

La modalidad escogida para la realización de la presente tesis doctoral es la de compendio de publicaciones. Tres trabajos aparecidos en revistas internacionales indexadas en *Journal Citation Reports (JCR)* constituirán los capítulos centrales de esta tesis doctoral. Este capítulo de introducción pretende tanto facilitar al lector el seguimiento de las ideas expuestas en los mencionados capítulos, como cumplir con la normativa vigente de la Universidad de Murcia en lo relativo a la estructura de tesis por compendio de publicaciones. En primer lugar se presentarán los antecedentes y el estado actual del campo de investigación, para pasar después a describir los objetivos alcanzados e introducir de forma resumida los tres trabajos que componen esta tesis por compendio, y cómo la misma representa una unidad científica.

1.1 Antecedentes y estado del arte

A lo largo de las últimas cuatro décadas, la Ley de Moore ha alimentado un extraordinario incremento en el rendimiento de los procesadores, ya que ha permitido doblar la cantidad de transistores en un chip aproximadamente cada dos años. Transistores cada vez más pequeños han posibilitado frecuencias de reloj cada vez más altas, al tiempo que los arquitectos de computadores han venido aprovechando la cada vez mayor cantidad de recursos en el silicio para diseñar *pipelines* más sofisticados y explotar mejor el paralelismo a nivel de instrucción (ILP) presente en los programas secuenciales. Esta combinación de avances en la tecnología de semiconductores y la microarquitectura de los

procesadores ha sostenido décadas de crecimiento exponencial en el rendimiento de dichas aplicaciones secuenciales. Un hito sin precedentes que sin embargo llegó a su fin a comienzos de la pasada década, provocando un punto de inflexión en la arquitectura de los procesadores.

1.1.1 La era multinúcleo

Como antes comentábamos, la escala de integración de los transistores ha sido la fuerza líder que ha dirigido el rápido crecimiento en el rendimiento de los procesadores en las pasadas décadas. En cada iteración de la tecnología, la integración de los transistores era doblada, los circuitos eran más rápidos y el consumo de energía permanecía constante [6]. Con el doble de transistores disponibles en cada nueva generación, los diseñadores tenían una enorme cantidad de recursos para crear microarquitecturas más complejas, incluyendo jerarquías de caché más grandes y sofisticadas así como poder usar la creciente velocidad de operación de los transistores para incrementar la frecuencia de reloj. Desafortunadamente, conforme los transistores se acercaban a dimensiones atómicas, nuevos desafíos han incrementado la dificultad de continuar con esta tendencia. Mientras la Ley de Moore continúe viva [31, 47] y a pesar de parecer que no goza de tan buena salud-–los tiempos de desarrollo de los nuevos nodos de fabricación por debajo de los 20nm han sido superiores a lo habitual—podemos esperar que la cantidad de transistores continúe doblándose en las próximas generaciones. Recientes investigaciones de IBM llevan al optimismo de poder impulsar esta famosa ley más que nunca en la próxima década [39] gracias a la introducción de nanotubos de carbono en la construcción de chips. Este hecho permitiría de nuevo un aumento de la frecuencia o bien una reducción drástica de la energía necesaria en los chips al mismo tiempo que se reduce aún más la escala de integración. No obstante, actualmente problemas como la energía consumida y el enfriamiento de los chips se han convertido en los primeros factores limitantes que restringen el incremento en la frecuencia de reloj. Este hecho ha causado el abandono de microarquitecturas con un gran cauce de ejecución [32]. La baja eficiencia energética, diseños de alta complejidad y costes de verificación muy caros, y el escaso ILP que queda por ser extraído, han parado los esfuerzos de diseño de procesadores superescalares muy agresivos con enormes ventanas de instrucciones. Consecuentemente, la dificultad de mejorar el rendimiento de los procesadores secuenciales ha ido en incremento debido a lo que se ha conocido como los 3 Muros: Energía, Memoria e ILP (*Power, Memory and ILP Walls*) [4]. En respuesta al estancamiento en el rendimiento, potencia y eficiencia energética y al

alto coste de verificación, la industria ha encontrado una manera de esquivar esta crisis basándose en chips con más de un procesador, conectándolos a través de memoria compartida [22]. Estos nuevos procesadores paralelos en un solo chip son conocidos como multiprocesadores-en-chip (*chip-multiprocessors* o CMPs) o arquitecturas multinúcleo (o *multicores*).

En una década, las arquitecturas multinúcleo se han vuelto ubicuas, especializándose en un amplio rango de diferentes sistemas: servidores [23], ordenadores portátiles y de escritorio [19,50], consolas de videojuegos [11] y teléfonos inteligentes o *smartphones* [3]. Cada nuevo desarrollo ha ido haciendo crecer rápidamente el número de núcleos integrados en un chip en un intento por incrementar el rendimiento a través de la explotación del paralelismo a nivel de hilo (TLP). Algunos investigadores predicen que en un futuro cercano podrán diseñarse chips con miles de núcleos de procesamiento [4]. La restricción en cuanto a la eficiencia energética no ha sido sólo responsable de esta transición a la era multinúcleo, sino de también estar forzando a los arquitectos de computadores a moverse desde procesadores multinúcleo con núcleos de procesamiento (pocos) complejos, a procesadores multinúcleo con un gran número de núcleos de procesamiento más simples (*manycores* [40]).

1.1.2 La programación concurrente

El giro producido hacia arquitecturas multinúcleo trae consigo el difícil reto de desarrollar aplicaciones paralelas que puedan aprovecharlas. Si bien la programación secuencial sigue siendo posible, se estarían desperdiciando la principal característica de este tipo de arquitecturas y un amplio número de recursos hardware disponibles: los múltiple núcleos de ejecución. Desafortunadamente, la programación concurrente se ha presentado como un paradigma poco conocido para los programadores secuenciales y simplemente era usado en pequeñas comunidades donde la supercomputación primaba por encima del resto. Ahora que las microarquitecturas hardware han cambiado su paradigma, así debe actuar el desarrollo de software, cambiando los desarrollos secuenciales por aquellos basados en la programación concurrente. En este viaje existen nuevos desafíos a los que los programadores deben enfrentarse. Escribir código eficiente y paralelo es una tarea difícil que supone orquestar la ejecución concurrente de todas las partes para mejorar el rendimiento mientras que al mismo tiempo se debe garantizar la corrección. Defectos únicos de software complejos y difíciles de encontrar en las aplicaciones multihilo como condiciones de carrera y interbloqueos (*deadlocks*) pueden rápidamente dar al traste con un proyecto software [10]. La realidad es

que a día de hoy la gran mayoría de los desarrollos software siguen haciendo uso de un único hilo de ejecución [24].

La programación concurrente es una tarea mucho más desafiante que la programación secuencial. Un programa paralelo es indudablemente más difícil de diseñar, escribir y depurar. Organizar la ejecución concurrente de la todas las partes para incrementar el rendimiento mientras que al mismo tiempo se garantiza la corrección no es para nada una tarea fácil. Para enfatizar el problema, la extracción de paralelismo a nivel de hilo (TLP) no es una tarea sencilla [1] o una propiedad local del código, requiere de la organización del código de formas a menudo no intuitivas. Durante una ejecución concurrente, los diferentes hilos entrelazan su ejecución de formas impredecibles y aleatorias, dando lugar a soluciones no deterministas. De esta forma los programadores se encuentran con que no sólo deben razonar sobre cómo coordinar los hilos de forma correcta sino que además en caso de error, su depuración es compleja en el mejor de los casos. La probabilidad de conseguir exactamente la misma ejecución que en una ocasión anterior es prácticamente cero.

La comunicación entre los diferentes hilos se suele hacer en base a dos paradigmas: el paso de mensajes y la memoria compartida. En el primero, el desarrollador es el que de forma explícita coordina el intercambio de información entre los hilos. Debe razonar cuidadosamente comunicación y sincronización para evitar las condiciones de carrera y llevar la concurrencia de los hilos a una finalización correcta. Por otra parte y como su propio nombre indica, la memoria compartida se basa en simplemente accesos concurrentes a direcciones de memoria que en realidad son compartidas y visibles a todos los hilos de ejecución. Esta tesis se centra en el paradigma de memoria compartida, considerado como el más intuitivo [35] ya que es el más cercano a la programación secuencial y además es el más extendido.

En el contexto de la memoria compartida, la sincronización en el acceso a las estructuras de datos compartidas es clave para conseguir programas correctos y eficientes. A menudo se recurren a mecanismos que aseguran la exclusión mutua en el acceso a estas secciones críticas. Son los denominados cerrojos (*locks*). Se trata de primitivas de bajo nivel que pueden ser usadas de dos formas distintas. Los cerrojos de grano fino permiten conseguir un gran rendimiento del hardware multinúcleo a costa de incrementar enormemente la complejidad de la programación concurrente. El desarrollador debe razonar de forma muy cuidadosa sobre el uso casi masivo de estas estructuras en pequeños bloques de código. No sólo debe conseguirse evitar los interbloqueos en los accesos a una misma sección crítica sino entre todas ellas. Por contra, los cerrojos de grano

grueso son fáciles de usar ya que engloban un código completo como sección crítica y por tanto, la complejidad de sincronización en el intercalamiento de los hilos es menor. El problema es obvio, el rendimiento se degrada, al producirse serialización de los hilos más a menudo de lo que se puede considerar deseable.

Además de los interbloqueos, los cerrojos traen otras situaciones indeseables como la inversión de prioridad (cuando un hilo de alta prioridad no puede adquirir el cerrojo porque otro hilo de baja prioridad lo tiene en posesión), convoy (cuando múltiples hilos de ejecución compiten por el mismo cerrojo continuamente) y falta de tolerancia a fallos (cuando un hilo en la sección crítica modifica los datos y aborta, causando que el programa entero falle). Además, los cerrojos rompen el principio de abstracción, ya que los programadores que utilizan un módulo deben ser conscientes de qué cerrojos se usan en él. Finalmente, los cerrojos ponen en peligro la propiedad de composición del código. Dos módulos individualmente correctos pueden sufrir interbloqueo (*deadlock*) cuando son combinados.

1.1.3 La abstracción transaccional

El compromiso entre una programación fácil y el rendimiento que puede alcanzarse haciendo uso de los cerrojos todavía persiste como un desafío clave para los programadores y los arquitectos de computadores en la era multinúcleo. La Memoria Transaccional (*Transactional Memory* o TM) [26,28] se ha propuesto como un modelo de programación conceptualmente más simple que puede ayudar a acelerar la productividad de desarrollo eliminando la compleja tarea de articular sabiamente los cerrojos de grano fino. La memoria transaccional coge prestado el concepto de transacción del mundo de las bases de datos, y lo aplica al dominio de la programación de memoria compartida en un intento de simplificar la sincronización de hilos. Las transacciones en el mundo de la programación multihilo son bloques de código que garantizan ser ejecutados atómicamente y en aislamiento con respecto al resto del código. A alto nivel, un programador o compilador marca las secciones críticas de código como bloques atómicos o transacciones. El sistema subyacente ejecuta estas transacciones especulativamente en un intento de explotar tanta concurrencia como sea posible. Los sistemas de Memoria Transaccional emplean generalmente un acercamiento optimista al control de la concurrencia para permitir a múltiples transacciones ejecutarse en paralelo, al tiempo que todavía preservan las propiedades de atomicidad y aislamiento y garantizan la corrección. Usando transacciones para acceder de forma segura a los datos, los programadores no necesitan razonar sobre un

intercalamiento seguro de los hilos o la posibilidad de interbloqueos para escribir código multihilo correcto. Por lo tanto, TM se presenta como un paradigma que permite a los desarrolladores software programar con una complejidad similar a los cerrojos de grano grueso mientras que el sistema subyacente se encarga de conseguir un rendimiento equivalente al del uso de cerrojos de grano fino, y siempre de forma transparente al programador. Además los sistemas TM solucionan otras limitaciones de la sincronización basada en cerrojos. El código transaccional es robusto tanto en errores de hardware como de software, ya que el sistema siempre puede deshacer (*rollback*) las actualizaciones especulativas a su estado pretransaccional en caso de que un hilo falle en medio de una transacción. Al contrario que los cerrojos, las transacciones son componibles, y pueden ser anidadas de forma segura sin el riesgo de interbloqueos [41].

1.1.4 Sistemas de memoria transaccional

Las transacciones son una abstracción prometedora que puede facilitar la programación paralela y hacerla más accesible al programador común. Las semánticas transaccionales pueden ser completamente implementadas en software, hardware o como una combinación de ambos. De acuerdo a esto, podemos clasificar la sistemas de Memoria Transaccional en Memoria Transaccional Software (*Software Transactional Memory* o STM), Memoria Transaccional Hardware (*Hardware Transactional Memory* o HTM) y como sistemas de Memoria Transaccional Híbridos. Las implementaciones STM [20, 29, 38, 48] permiten ejecutar cargas de trabajo transaccional en los sistemas existentes sin requerir un soporte especial de hardware, proveyendo un alto grado de flexibilidad con un pequeño coste. Por otro lado, el hecho de que prácticamente toda la gestión se realiza es software supone imponer una sobrecarga muy alta y por tanto estos sistemas no se comportan bien en términos de rendimiento comparados con las aproximaciones tradicionales basadas en cerrojos. Para que este nuevo paradigma sea viable a la alternativa de los cerrojos, los mecanismos clave que proveen la semántica transaccional deben ser implementados a nivel hardware. Los sistemas TM Híbridos [5, 17, 44, 49, 51] intentan aprovechar la flexibilidad de los sistemas STM al tiempo que usan soporte hardware simple para acelerar el rendimiento de las operaciones críticas de una implementación STM. Los sistemas híbridos usan la Memoria Transaccional Software como un respaldo para manejar situaciones donde el hardware no puede ejecutar las transacciones con éxito [27]. La semántica transaccional puede también ser soportada en gran parte en hardware [2, 7, 8, 25, 45, 52], consiguiendo un buen rendimiento con diversas variaciones de complejidad que cambian

considerablemente de un sistema HTM a otro. Todo depende de qué tipo de transacciones es capaz de graduar (*commit*) sin caer en mecanismos de retroceso. Esquemas de HTM simples [9,12,30] adoptan la solución del "mejor esfuerzo" que no puede garantizar que todas las transacciones podrán eventualmente confirmarse o graduarse usando solamente soporte hardware, principalmente debido a las limitaciones impuestas por las estructuras hardware involucradas. Propuestas más sofisticadas de HTM [2,25,45] tratan con esta limitación en el tamaño de la transacción, garantizando que ciertas transacciones "acotadas" pueden ser completamente ejecutadas en hardware. Sin embargo, ni las soluciones acotadas ni las del mejor esfuerzo pueden graduar transacciones que sufren eventos que son muy complicados de manejar en hardware, como los cambios de contexto, fallos de página, excepciones de entrada/salida o interrupciones [33]. En estas circunstancias las transacciones son invariablemente abortadas. Por otro lado, también existen elaborados esquemas HTM que han sido diseñados [2,46] para tratar todas las transacciones en hardware, asegurando que la misma transacción no sea abortada indefinidamente debido a sus tamaño, duración o eventos que pueda encontrar. El precio a pagar es precisamente el coste de implementación para los fabricantes de microprocesadores.

Comercialmente algunos fabricantes han presentado diseños con soporte para memoria transaccional. IBM [21] apostó por añadir hardware en el diseño de Blue Gene/Q (BG/Q) convirtiéndose en el primer procesador comercial en disponer de este soporte. Cada chip de cómputo de BG/Q dispone de 16 núcleos, 4 hilos por núcleo, caché L1 privada de 16KB con asociatividad de 8 vías y línea de caché de 64B, y 32MB compartidos de caché de nivel 2 asociativa de 16 vías. La coherencia transaccional se mantiene únicamente en la caché L2 aprovechando cada vía para almacenar un estado especulativo transaccional. BG/Q detecta los conflictos que se producen vigilando los accesos de escritura en modo transaccional (conflictos *read-write*, *write-read*, *write-write*) o cuando un acceso transaccional es seguido de una escritura no transaccional. En caso de conflicto el hardware envía una interrupción a los hilos transaccionales involucrados. BG/Q presenta en definitiva soporte hardware para transacciones limitadas del mejor esfuerzo debido a que en caso de no poder almacenar en una nueva vía de la caché L2 un nuevo estado especulativo, se produciría un fallo de capacidad que provocaría que la ejecución transaccional falle. Por otra parte, Intel también ha dado soporte de memoria transaccional a través de una extensión del ISA denominada *Transactional Synchronization Extensions (TSX)* [13]. Esta extensión permite el uso de semántica transaccional en los programas para los chips de Intel con arquitectura Haswell, Broadwell o Skylake. Además le da al programador dos modos de

aplicar TM en sus programas, bien mediante el denominado *Hardware Lock Elision (HLE)* que permite compatibilidad con procesadores sin soporte para TSX; o bien mediante *Restricted Transactional Memory (RTM)*, más avanzado y flexible. Aunque la implementación hardware de este soporte se mantiene relativamente opaca, Intel especifica en sus manuales de optimización y del desarrollador que Haswell mantiene los conjuntos de escritura y lectura en la granularidad de la línea de caché, rastreando los accesos a la caché de datos de nivel 1 [34]. También indica que los conflictos son detectados gracias al protocolo de coherencia de caché [14].

1.1.5 Características de la memoria transaccional

La Memoria Transaccional (TM) [26,28] se presenta como un nuevo paradigma de programación con el que construir estructuras de datos de memoria compartida escalables, superando las limitaciones impuestas por los cerrojos de grano fino. Bajo el modelo de TM, el programador simplemente declara qué regiones de código deben ser ejecutadas aislada y atómicamente (transacciones), confiando en que el sistema subyacente garantizará ambas propiedades. Para ello, es necesaria una extensión en el conjunto de instrucciones (ISA). Todas las implementaciones HTM introducen un par de nuevas instrucciones tales como "empezar transacción" y "confirmar transacción" (*commit*). Por un lado, la ejecución de la instrucción "empezar transacción" causa que el procesador entre en "modo transaccional" y realice algunas acciones comunes relacionadas con la inicialización de mecanismos transaccionales básicos, como respaldar los registros del procesador en un banco de registros virtual. Estos registros, junto con la memoria, forman el estado preciso del procesador, y por tanto el estado de los registros necesita ser restaurado a un estado previo conocido en caso de aborto. Por otro lado, la instrucción "confirmar transacción" intenta graduar o confirmar (*commit*) el estado especulativo de las actualizaciones de la transacción haciéndolas visibles al resto del sistema. En caso de éxito se produce una vuelta del procesador a un estado no transaccional al tiempo que se descarta el respaldo de los registros. La Memoria Transaccional ejecuta de forma optimista las transacciones, deteniéndolas o abortándolas cuando ocurren conflictos en los datos en tiempo real entre las transacciones concurrentes. Para ello se garantizan dos propiedades básicas: atomicidad y aislamiento. La atomicidad dictamina que la transacción es ejecutada hasta completarse o no se ejecuta en absoluto. Si la transacción se gradúa exitosamente, todos sus cambios se hacen globalmente visibles al mismo tiempo. Por el contrario, si la transacción aborta, todas sus tentativas de actualización son descartadas para volver la máquina a un estado pretransaccional,

como si nunca se hubiera iniciado la ejecución de la transacción. Por su parte, la propiedad de aislamiento requiere que el estado intermedio especulativo se mantenga oculto al resto del sistema. Satisfaciendo estas dos propiedades, las transacciones parecen ejecutarse en algún orden secuencial global. Para proveer estas dos propiedades, los sistemas TM deben implementar dos mecanismos básicos llamados versionado o gestión de versiones (*data version* o *version management*) y detección o tratamiento de conflictos (*conflict management/detection*). Las políticas e implementaciones de estos dos mecanismos constituyen las dos dimensiones fundamentales del espacio de diseño de la Memoria Transaccional.

El mecanismo de gestión de versiones (*version management*) se encarga del almacenamiento simultáneo tanto de los datos especulativos (nuevos valores que serán visibles si la transacción se gradúa) como de los datos pretransaccionales (valores antiguos conservados por si la transacción aborta). Sólo uno de estos dos valores puede ser almacenado *in-situ*, es decir, en la correspondiente dirección de memoria, mientras que el otro necesita ser alojado en otro lugar. Dependiendo de qué valor, antiguo o nuevo, se mantenga en la dirección real, la política de manejo de versiones puede ser clasificada como agresiva (*eager*) o perezosa (*lazy*). Una política de manejo de versiones perezosa (*lazy*) mantiene *in-situ* los valores antiguos hasta la fase de confirmación (*commit*), almacenando las actualizaciones especulativas al margen mientras tanto. Sólo si la transacción se confirma, los valores antiguos se sobrescriben con los nuevos. Ya que los valores antiguos se mantienen inalterados, un sistema *lazy* ejecuta los abortos rápidamente, pues sólo es necesario descartar los datos especulativos. Por el contrario, la aproximación agresiva (*eager*) al versionado de datos, usa un *log* transaccional por hilo para almacenar el valor antiguo de cada posición de memoria previo a una escritura y después escribe el nuevo valor en la dirección de memoria original. Esta política favorece la confirmación de las transacciones, ya que los nuevos valores se encuentran en la dirección de memoria correcta, mientras que penaliza los abortos al tener que desenrollar el *log* transaccional para poder restaurar el estado original.

Cuando dos transacciones concurrentes acceden a la misma dirección de memoria, y al menos uno de los accesos es una operación de escritura, decimos que hay un conflicto o carrera entra ellas. Todos los sistemas TM han de implementar mecanismos de manejo de conflictos (*conflict management*) para detectar y resolver estas situaciones. Para este propósito, tanto los datos leídos como escritos por cada transacción deben ser rastreados. Esta tarea se realiza gracias al protocolo de coherencia de caché especialmente adaptado a la memoria transaccional. El conjunto de direcciones de datos que una transacción modifica durante su

ejecución es conocido como el conjunto de escritura (*write set*). De forma análoga, el conjunto de lectura (*read set*) se refiere al grupo de direcciones de memoria leídas por una transacción. En estos términos, un conflicto entre dos transacciones concurrentes ocurre cuando el conjunto de escritura de una transacción se solapa con el conjunto de lectura o escritura de la otra. Dependiendo de los metadatos usados para almacenar ambos conjuntos de direcciones (*book-keeping*), la detección de conflictos puede tener lugar a diferentes niveles de granularidad, desde objetos, a líneas de caché o incluso a direcciones a nivel de byte. Las estrategias de detección de conflictos varían dependiendo de cuándo el procesador examina los conjuntos de lectura y escritura. En sistemas con detección de conflictos agresiva (*eager*)–a veces definidos también como pesimistas–los conflictos son detectados tan pronto como ocurren, es decir, en cada acceso individual a una dirección de memoria. En el acercamiento contrario, llamado detección de conflictos perezoso (*lazy*) u optimista, esta comprobación se retrasa hasta la fase final de la transacción o *commit*, y la resolución se realiza generalmente siguiendo el esquema de que gana el primero en graduarse. La transacción graduada publica entonces su conjunto de escritura al resto del sistema para que las demás transacciones puedan comprobar si sus respectivos conjuntos (lectura y escritura) han colisionado con él, y abortar si es necesario. En el ámbito de la memoria compartida, el protocolo de coherencia de caché es el responsable de asegurar que las escrituras a la memoria compartida son eventualmente visibles al resto de núcleos [15]. Los protocolos basado en invalidación (en lugar de los basados en actualización) son más atractivos para los arquitectos de computadores ya que hacen un menor uso de la red de interconexión. Como ejemplo, en esta tesis se hace uso de un protocolo basado en invalidación MESI que emplea un directorio distribuido para mantener la coherencia a través de una red punto a punto (que no garantiza la ordenación de los mensajes). Cada línea de caché es asignada a un *tile* origen (*home tile*) que mantiene la entrada de directorio para las líneas mapeadas a su banco de caché compartida de nivel 2. En cada fallo de caché se envía una petición al correspondiente banco origen de nivel 2, que determina la acción de coherencia que es necesario realizar (si este fuese el caso): por ejemplo, reenviar la petición a la caché que debe proveer el bloque de datos, o enviar mensajes de invalidación a los compartidores en caso de un fallo de escritura.

Atendiendo tanto al manejo de versiones como a la detección de conflictos, podemos clasificar las dos tendencias más importantes en el desarrollo de sistemas HTM: *eager-eager* y *lazy-lazy*. Como sus descriptivos nombres indican, un sistema HTM *eager-eager* realiza tanto una gestión de versiones de datos agresiva junto con una política de detección de conflictos también agresiva. Por el

contrario un sistema *lazy-lazy* confía en que no se producirán conflictos (sistema optimista) y se basa tanto en un versionado perezoso como en una detección de conflictos tardía. La combinación de sistemas *eager-lazy* (para el versionado de datos y detección de conflictos respectivamente) es imposible. Las transacciones no pueden escribir los datos especulativos en la dirección final de memoria si no se comprueba activamente su uso por otra transacción antes. En caso contrario no se cumpliría la propiedad de aislamiento.

1.1.6 Eficiencia energética

El rendimiento siempre ha sido considerado, desde el principio del desarrollo de la arquitectura de computadores, como el factor clave a mejorar en cada nueva generación. Una microarquitectura era considerada como "buena" si conseguía mejorar ampliamente a su predecesora en términos de prestaciones. Este comportamiento se ha repetido constantemente hasta la aparición de las tres barreras clave que cambiaron el diseño de procesadores en la década pasada: El muro de la Energía, el muro de la Memoria y el muro del ILP. El primero establece que el consumo de energía crece exponencialmente con respecto al crecimiento de la frecuencia que el procesador experimenta en cada nueva escala de integración. En este punto, el consumo energético dejó de ser una consideración menor para ser un limitante en el desarrollo de las microarquitecturas. Además de dar lugar a la era multinúcleo, la frecuencia máxima de funcionamiento de un procesador se estancó (alrededor de los 4 GHz para sistemas de escritorio) debido al consumo energético. La ubicuidad de los procesadores posteriores dio lugar a una explosión en el diseño de microarquitecturas. Los sistemas embebidos o los actuales dispositivos móviles carecen de sofisticados sistemas de refrigeración y en el caso de los segundos, la batería no puede disponer de energía al sistema indefinidamente. Los requerimientos energéticos de un procesador pasaron de estar limitados por el muro de Energía a tener que contar con unas limitaciones muchos más fuertes. Hoy en día podemos afirmar que el desarrollo de procesadores basado únicamente en la búsqueda de más prestaciones ha girado hacia la mejora del rendimiento por watio, es decir, el rendimiento y la eficiencia energética al mismo tiempo. Casos actuales los podemos encontrar por ejemplo en la estrategia de desarrollo que sigue Intel en las últimas generaciones. Si bien existe una mejora de rendimiento de una generación a otra, esta es cada vez menor, siendo hoy día en torno a 5-7 % de media entre rendimiento secuencial y multinúcleo [16]. No obstante, sigue habiendo una evolución en el desarrollo de microarquitecturas, consiguiendo que los consumos que homologan

los procesadores actuales (en TDP) sean menores que sus equivalentes de hace cuatro generaciones. Esta tendencia no se restringe únicamente a los sistemas de escritorio sino también en procesadores de ámbito específico como son las GPUs.

1.2 Motivación y objetivos

A pesar de las ventajas que los sistemas de Memoria Transaccional Software e Híbridos ofrecen en términos de complejidad, la realidad es que implementaciones rápidas de programación transaccional son necesarias para que la TM gane en visibilidad. Algunos fabricantes de procesadores ya se han aventurado en la inclusión de hardware que dé soporte a TM en sus últimos diseños [9,12,21]. A pesar de que nunca ha visto la luz comercialmente, el procesador Rock [9], desarrollado por Sun Microsystems fue el primer chip multinúcleo de propósito general con capacidades de HTM utilizando una aproximación de mejor esfuerzo. IBM también apostó por incluir soporte hardware en los procesadores del Blue Gene/Q [21], siendo estos los primeros chips comerciales con soporte de HTM. De hecho, dada la gran cantidad de transistores disponibles en los chips de hoy día, uno de los desafíos más importantes para los arquitectos de computadores es entender qué abstracciones pueden mejorar la productividad del desarrollo de software paralelo e introducir el hardware apropiado que les dé soporte [18]. Las transacciones son un buen candidato para esa abstracción. Por otra parte, la búsqueda del rendimiento de forma aislada ha dejado de ser la principal motivación en el desarrollo de procesadores para serlo el rendimiento por watio. El rendimiento sigue siendo importante pero no a costa de un consumo desmesurado de energía. El objetivo global de esta tesis es por tanto abordar el diseño de un sistema HTM teniendo en cuenta tanto rendimiento como consumo de energía. Para ello, nos planteamos los siguientes objetivo concretos:

- Analizar en un marco común el rendimiento y consumo de energía de los dos sistemas HTM más ampliamente estudiados: *eager-eager* y *lazy-lazy*.
- Proponer mejoras arquitectónicas con las que se consigan mejorar el rendimiento por watio de cada uno de estos sistemas.

Más concretamente, una vez se ha estudiado con detalle el rendimiento y consumo de energía que las aproximaciones *eager-eager* y *lazy-lazy* ofrecen, hemos procedido a desarrollar técnicas hardware que han permitido evolucionar ambos sistemas, ya sea mejorando el consumo manteniendo el rendimiento o

incluso mejorando ambos aspectos. El objetivo de esta tesis no es por tanto determinar qué versión de los sistemas HTM es mejor desde el punto de vista del rendimiento sino ofrecer una comparación en igualdad de condiciones tanto del rendimiento como de la eficiencia energética y el tráfico de red. Al mismo tiempo, se identifican las ineficiencias que impactan tanto en el rendimiento como en los consumos de algunos patrones de acceso a datos. Para ello se hace uso de un entorno de desarrollo y pruebas ampliamente aceptado como es la herramienta de simulación SIMICS+GEMS [37, 42]. En concreto se ha ampliado el módulo de GEMS encargado de la abstracción de memoria conocido como Ruby. El estudio de los resultados se realiza para sistemas de propósito general con todas sus particularidades. La suite de benchmarks para las pruebas es también un estándar en la comunidad TM: STAMP [43]. La descripción concreta del entorno de evaluación es presentada en cada uno de los artículos que conforman los capítulos 2, 3 y 4 de esta tesis. A la vista de los resultados no se puede catalogar de forma absoluta que un sistema sea preferible sobre otro, dado que diversos patrones de acceso a datos benefician a los sistemas HTM *eager-eager* sobre los *lazy-lazy* y viceversa para otros tantos. No obstante, esta tesis y los artículos que la componen ofrecen una visión amplia sobre el compromiso entre rendimiento-energía de sistemas HTM sofisticados. Al mismo tiempo se diseñan técnicas que mejoran a los dos sistemas base de partida. Las principales contribuciones se pueden resumir en la siguiente estructura de capítulos de esta tesis:

- Capítulo 2: *"On the Design of Energy-Efficient Hardware Transactional Memory Systems"*. En este trabajo se realiza una comparación exhaustiva entre las dos alternativas de diseño más populares de sistemas hardware de memoria transaccional: *lazy-lazy* (LL) y *eager-eager* (EE). Esta comparación se ha realizado en términos de rendimiento, prestando especial atención al tiempo de ejecución de las aplicaciones transaccionales bajo cada uno de los sistemas, y energía consumida en el subsistema de memoria, siendo esta última la primera comparativa de este estilo que se ha publicado. Para ello se incluyó en la herramienta de simulación modelos de consumo de energía ampliamente utilizados por la comunidad científica para la jerarquía de caché y la red de interconexión del CMP. Los resultados de este estudio comparativo han servido para identificar las ventajas e inconvenientes de cada uno de los sistemas de memoria transaccional (LL y EE) y servir de punto de partida para proponer técnicas encaminadas a mejorar los aspectos problemáticos que cada sistema presenta.
- Capítulo 3: *"Selective Dynamic Serialization for Reducing Energy Consumption*

in Hardware Transactional Memory Systems". En el Capítulo 2 se demuestra que una fracción importante de la energía consumida por los sistemas EE es debida a los conflictos que surgen entre transacciones. Profundizando más, se ha observado que en algunas aplicaciones transaccionales podrían predecirse los conflictos entre las transacciones en tiempo de ejecución. En estos casos es preferible desde el punto de vista energético que las transacciones involucradas en el conflicto se ejecuten en serie con el objetivo de evitar la pérdida de energía a causa de los abortos que surgen por los mismos conflictos. Esta serialización permite a los sistemas EE mejorar su eficiencia energética sin degradar las prestaciones. En este trabajo se han desarrollado, implementado y evaluado diversas políticas que permiten serializar en tiempo de ejecución transacciones que entran en conflicto sólo cuando éste se produce sin que se produzcan ciclos de que den lugar a interbloqueos. El resultado final es un sistema *eager-eager* más eficiente desde el punto de vista del consumo de energía sin pérdida de prestaciones y, por lo tanto, mejorando el rendimiento por watio.

- Capítulo 4: *"Fast and Efficient Commits for Lazy-Lazy Hardware Transactional Memory"*. En el Capítulo 2 se descubre que el principal problema de los sistemas LL se encuentra en la etapa de confirmación o graduación de las transacciones. Más concretamente en las operaciones que son necesarias para garantizar que una transacción ha finalizado sin conflictos. En este trabajo se ha implementado y evaluado una técnica hardware que permite acelerar, mejorar la eficiencia energética y simplificar la principal propuesta diseñada hasta ahora para la etapa de confirmación en sistemas HTM escalables: el mecanismo de reserva de bancos de directorio previo a que una transacción pueda realizar su confirmación (*commit*). Para ello se ha propuesto emplear una red específica de bajo coste. Se han considerado dos implementaciones de esta red, una conformada a partir de *g-lines* [36] y otra que utiliza tecnología estándar. Los resultados de este trabajo demuestran la conveniencia este tipo de redes específicas.

1.3 Publicaciones derivadas de la tesis

Las siguientes publicaciones son resultado directo del trabajo realizado para poder completar esta tesis doctoral. Además de las tres publicaciones en revistas internacionales que han dado lugar a esta tesis por compendio de publicaciones, se presentan también varias contribuciones en congresos internacionales y

nacionales:

Revistas internacionales

1. Epifanio Gaona, Rubén Titos-Gil, Juan Fernández y Manuel E. Acacio. *"On the Design of Energy-Efficient Hardware Transactional Memory Systems"*. Concurrency and Computation: Practice & Experience, Vol. 25, No. 6, Abril 2013, pp. 862-880, Wiley, ISSN: 1532-0626. Área JCR: Science Edition 2013, Índice de Impacto: 0.78, Tercio: Segundo (50/102)

2. Epifanio Gaona, Rubén Titos-Gil, Juan Fernández y Manuel E. Acacio. *"Selective Dynamic Serialization for Reducing Energy Consumption in Hardware Transactional Memory Systems"*. Journal of Supercomputing, Vol. 68, No. 2, Mayo 2014, pp. 914-934, Springer, ISSN: 0920-8542. Área JCR: Science Edition 2014, Índice de Impacto: 0.858, Tercio: Segundo (25/50)

3. Epifanio Gaona, José L. Abellán y Manuel E. Acacio. *"Fast and Efficient Commits for Lazy-Lazy Hardware Transactional Memory"*. Journal of Supercomputing, DOI: 10.1007/s11227-015-1523-8, Springer, ISSN: 0920-8542. Área JCR: Science Edition 2014, Índice de Impacto: 0.858, Tercio: Segundo (25/50)

Congresos internacionales

4. Epifanio Gaona, Rubén Titos, Juan Fernández y Manuel E. Acacio. *"Characterizing Energy Consumption in Hardware Transactional Memory Systems"*. 22nd Int'l Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2010), Petrópolis (Brasil), Octubre 2010. Publicación: IEEE Computer Society Press, ISBN: 978-0-7695-4216-4, pp. 9–16.

5. Epifanio Gaona, Rubén Titos-Gil, Manuel E. Acacio y Juan Fernández. *"Dynamic Serialization: Improving Energy Consumption in Eager-Eager Hardware Transactional Memory Systems"*. 20th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP-2012), Garching (Alemania), Febrero 2012. Publicación: IEEE Computer Society Press, ISBN: 978-0-7695-4633-9, pp. 221–228.

6. Epifanio Gaona, José L. Abellán, Manuel E. Acacio y Juan Fernández. *"Deploying Hardware Locks to Improve Performance and Energy Efficiency of Hardware Transactional Memory"*. 26th Int'l Conference on Architecture of Computing Systems (ARCS 2013), Praga (República Checa), Febrero 2013. Publicación: Springer, ISBN: 978-3-642-36423-5, pp. 220–231 .

Congresos nacionales

7. Epifanio Gaona, Rubén Titos, Manuel E. Acacio y Juan Fernández. *"Caracterización del Rendimiento y Consumo de Energía de dos Sistemas de Memoria Transaccional Hardware"*. Actas de las XXI Jornadas de Paralelismo, Valencia, Septiembre 2010. Publicación: IBERGARCETA PUBLICACIONES, S.L., Madrid, ISBN: 978-84-92812-50-9.

Además de las anteriores publicaciones, que como se indica, son resultado directo de la tesis doctoral, se han realizado la siguientes publicaciones adicionales durante la etapa de postgrado:

Congresos internacionales

8. Epifanio Gaona, Juan Fernández y Manuel E. Acacio. *"Fast and Efficient Synchronization and Communication Collective Primitives for Dual Cell-based Blades"*. Euro-Par 2009, Delft (Holanda), Agosto 2009. Publicación: Lecture Notes in Computer Science 5704, Springer, ISBN: 978-3-642-03868-6, pp. 900–911.

Congresos nacionales

9. Epifanio Gaona, Juan Fernández y Manuel E. Acacio. *"Implementation and Evaluation of Collectives Primitives for Dual Cell-based Blades"*. Actas de las XX Jornadas de Paralelismo, La Coruña, Septiembre 2009. Publicación: Servizo de Publicacións, Universidade da Coruña, ISBN: 84-9749-346-8.

Capítulo 2

On the design of energy-efficient hardware transactional memory systems

Epifanio Gaona, Rubén Titos-Gil, Juan Fernández y Manuel E. Acacio. *"On the Design of Energy-Efficient Hardware Transactional Memory Systems"*. Concurrency and Computation: Practice & Experience, Vol. 25, No. 6, Abril 2013, pp. 862-880, Wiley, ISSN: 1532-0626. Área JCR: Science Edition 2013, Índice de Impacto: 0.78, Tercio: Segundo (50/102) URL:http://onlinelibrary.wiley.com/doi/10.1002/cpe.2866/abstract

Abstract *Transactional memory is currently being advocated as a promising alternative to lock-based synchronization because it simplifies multithreaded programming. In this way, future many-core chip multiprocessor architectures may need to provide hardware support for transactional memory. On the other hand, energy consumption constitutes nowadays a first class consideration in multicore processor designs. In this work, we characterize the performance and energy consumption of two well-known hardware transactional memory systems that employ opposite policies for data versioning and conflict management. More specifically, we compare a LogTM-SE eager-eager system and a version of the Scalable Transactional Coherence and Consistency lazy-lazy system that enable parallel commits. To do so, we extended the Multifacet GEMS simulator to estimate the energy consumed in the on-chip caches according to CACTI and used the interconnection network energy model given by Orion 2. Results show that the energy*

consumption of the eager-eager system is 38 % higher in average than in the lazy-lazy case, whereas performance differences between the two systems are 26 % in average. We found that even though lazy-lazy beats eager-eager on average, there are considerable deviations in performance depending on the particular characteristics of each application and the settings of both systems. Finally, from this characterization, we observe that a significant part of the energy consumed in some applications in eager-eager is spent on the back-off delay phase and explore more energy-efficient hardware back-off mechanisms. For lazy-lazy systems, the way in which memory lines are assigned to the L2 cache banks affects the number of parallel commits in some applications, and we study an alternative fine-grained assignment.

Aportación del doctorando *Para la elaboración de este artículo he participado en las etapas de implementación, evaluación y redacción del mismo. La implementación ha consistido en aumentar el simulador SIMICS+GEMS para que soporte el HTM lazy-lazy STCC tanto con el protocolo de commit SEQ como SEQPRO. Además ha sido necesaria la modificación del protocolo de coherencia para el rastreo de los accesos a memoria realizados así como de los mensajes generados con el objetivo de medir adecuadamente su consumo energético. Siempre siguiendo el modelo presentado por CACTI para el consumo de la jerarquía de caché y por Orion 2.0 para el consumo de la red de interconexión tanto en los enlaces como en los routers de cada tile. También se han ampliado las subfases de ejecución disponibles en GEMS con la inclusión de precommit así como la identificación de las estructuras donde se produce el consumo energético. Esto permite asignar el gasto energético a las estructuras donde se producen. Además se ha asignado una subfase de ejecución a cada consumo producido en el sistema de memoria de modo que es distinguible el consumo de cada fase y facilita la extracción de conclusiones al disponer de las mismas subfases que se estudian en la evaluación del rendimiento. Para esta fase de implementación también se ha desarrollado los algoritmos hardware de backoff de LogTM-SE así como los distintos mapeos de dirección físicas de memoria. Durante la evaluación he realizado los experimentos y también he participado en la extracción de las conclusiones de los mismos. Finalmente he sido el principal escritor del artículo, apoyado por los coautores.*

Capítulo 3

Selective dynamic serialization for reducing energy consumption in hardware transactional memory systems

Epifanio Gaona, Rubén Titos-Gil, Juan Fernández y Manuel E. Acacio. "*Selective Dynamic Serialization for Reducing Energy Consumption in Hardware Transactional Memory Systems*". Journal of Supercomputing, Vol. 68, No. 2, Mayo 2014, pp. 914-934, Springer, ISSN: 0920-8542. Área JCR: Science Edition 2014, Índice de Impacto: 0.858, Tercio: Segundo (25/50) URL:http://link.springer.com/article/10.1007%2Fs11227-013-1072-y

Abstract *In the search for new paradigms to simplify multithreaded programming, Transactional Memory (TM) is currently being advocated as a promising alternative to deadlock-prone lock-based synchronization. In this way, future many-core CMP architectures may need to provide hardware support for TM. On the other hand, power dissipation constitutes a first class consideration in multicore processor designs. In this work, we propose Selective Dynamic Serialization (SDS) as a new technique to improve energy consumption without degrading performance in applications with conflicting transactions by avoiding wasted work due to aborted transactions. Our proposal, which is implemented on top of a hardware transactional memory (HTM) system with an eager conflict management policy, detects and serializes conflicting*

transactions dynamically (at run-time). In its simplest form, in case of conflict, one transaction is allowed to continue whilst the rest are completely stalled. Once the executing transaction has finished, it wakes up several of the stalling transactions. More elaborated implementations of SDS try to delay this behavior until serialization of transactions is profitable, achieving the best trade-off between performance, energy savings and network traffic. SDS implementations differ from each other in the condition that triggers the serialization mode. We have evaluated several SDS schemes using GEMS, a full-system simulator implementing the LogTM-SE Eager–Eager HTM system, and several benchmarks from the STAMP suite. Results for a 16-core CMP show that SDS obtains reductions of 6 % on average in energy consumption (more than 20 % in high contention scenarios) in a wide range of benchmarks without affecting, on average, execution time. At the same time, network traffic level is also reduced by 22 %.

Aportación del doctorando *En este artículo mi aportación ha consistido en la participación del diseño de Selective Dynamic Serialization (SDS), su implementación, evaluación y redacción del artículo. Mi aportación principal al diseño de SDS ha consistido en el sistema de herencia de transacciones serializadas, es decir, la cesión de responsabilidad de despertar a otras transacciones durante el aborto de una transacción. También en los requerimientos hardware y su dimensionamiento. Por otra parte la implementación ha consistido en modificar el protocolo de coherencia de caché de LogTM-SE para evitar el reenvío masivo de peticiones de memoria conflictivas y el posterior reenvío de mensajes NACK. Se ha realizado también la implementación del modo serizalizado así como de su mecanismo de activación basada en contadores de saturación y el más complejo sistema de herencia de transacciones serializadas en caso de aborto. Durante la evaluación he participado en la extracción de resultados, contextualizándolos y definiendo las conclusiones. También he sido el principal redactor del artículo.*

Capítulo 4

Fast and efficient commits for lazy-lazy hardware transactional memory

Epifanio Gaona, José L. Abellán y Manuel E. Acacio. *"Fast and Efficient Commits for Lazy-Lazy Hardware Transactional Memory"*. Journal of Supercomputing, DOI: 10.1007/s11227-015-1523-8, Springer, ISSN: 0920-8542. Área JCR: Science Edition 2014, Índice de Impacto: 0.858, Tercio: Segundo (25/50) URL: http://link.springer.com/article/10.1007%2Fs11227-015-1523-8

Abstract *Transactional memory (TM) is a compelling alternative to simplify multithreaded programming that traditionally relies on error-prone lock-based synchronization for implementing cooperative tasks. Lazy-Lazy hardware TM is one of the most efficient schemes in today's hardware TM systems. Nonetheless, the commit protocol in these systems has severe impact on performance and energy. The SEQ in Scalable-TCC implementation (STCC-SEQ) is the most popular and efficient commit protocol to date. In this paper, we propose GCommit, a cost-effective hardware-based STCC-SEQ protocol. GCommit employs a G-Arbiter microarchitecture for achieving minimal-latency and high-efficient commits. We implement G-Arbiter with a standard 45 nm cell library. For a target 16-core CMP, a G-Arbiter just represents 0.07 % of the whole on-chip area, requiring marginal energy consumption. Full-system simulations of the target system with the STAMP benchmarks show that GCommit achieves average reductions of 15.7 and 13.7 % in execution time and energy, respectively, when compared with STCC-SEQ.*

Aportación del doctorando *Para este artículo, además de ser el principal redactor del mismo he realizado su implementación y, con el resto de coautores, he participado en la evaluación de los resultados. La implementación ha consistido en la implementación hardware de los G-Arbiters así como del protocolo de GCommit usado por las transacciones. Esto incluye el mapeo de direcciones a cada G-Arbiter en caso de haber varios. También he realizado las pruebas para la extracción de resultados y participado en la evaluación de los mismos y las conclusiones.*

Capítulo 5

Conclusiones y vías futuras

La Memoria Transaccional se presenta como un nuevo paradigma de programación que enfrenta algunos de los desafíos asociados con la programación concurrente. El rendimiento de las aplicaciones secuenciales ha alcanzado un punto en el que su mejora es marginal y los programadores deben considerar los algoritmos paralelos como la alternativa más viable para acelerar sus aplicaciones haciendo uso de los múltiples núcleos de ejecución disponibles en los CMPs actuales. En este contexto, la memoria transaccional permite simplificar la difícil tarea de sincronizar los diferentes hilos de ejecución de una forma fácil e intuitiva bajo el modelo de memoria compartida. Esta tesis se centra en el estudio, desarrollo y mejora de los sistemas de memoria transaccional hardware más prominentes hoy en día. Además lo hace no sólo teniendo en cuenta el rendimiento, sino también dándole al consumo de energía un papel protagonista, al igual que ocurre en la industria de fabricación de microprocesadores actual.

En primer lugar se han evaluado los sistemas de memoria transaccional hardware más representativos de las dos vertientes actuales más prominentes: *LogTM Signature Edition (LogTM-SE)* como máximo exponente de los sistemas *eager-eager* actuales, y *Scalable Transactional Coherence and Consistency (STCC)* con *Sequential Commit (STCC-SEQ)* como algoritmo de confirmación por parte de la vertiente *lazy-lazy*. La primera conclusión que podemos extraer de los resultados presentados en esta tesis es que ninguna de las dos aproximaciones resulta claramente mejor que la otra en todos los casos de estudio, ni teniendo en cuenta el rendimiento ni la energía consumida. Las distintas particularidades y patologías intrínsecas de cada aplicación pueden afectar negativamente a los sistemas *eager-eager*, a los

lazy-lazy o a ambos. No obstante en media, los sistemas *lazy-lazy* obtienen un rendimiento un 26 % mejor, al tiempo que su eficiencia energética es en media un 38 % superior. La segunda conclusión es, por tanto, que los sistemas *lazy-lazy* y su optimismo resultan, a igualdad de rendimiento, más eficientes desde el punto de vista energético. O de otra manera, un pequeño incremento del rendimiento da lugar a una reducción mayor en el consumo de energía. Particularmente para los sistemas *eager-eager*, la implementación del mecanismo de *backoff* se realiza en software mediante un bucle de espera activa y por tanto, resulta muy ineficiente en términos de energía al producirse continuos accesos a memoria. Simplemente cambiando esta implementación a una versión hardware, se consigue un ahorro energético de entre el 10 % y el 30 % en aplicaciones con una fracción importante del tiempo final empleado en la fase de *backoff*. Además, desde el punto de vista energético, la principal patología de los sistemas *eager-eager* y en particular de LogTM-SE es que en caso de contención, el comportamiento por defecto consiste en detener la ejecución de las transacciones involucradas e intentar el acceso a la dirección de memoria de forma continuada. Mientras el conflicto persista, todas las peticiones serán rechazadas por el protocolo de coherencia aumentado para la memoria transaccional mediante el mensaje NACK. Esta fase, conocida como *stall*, es la causante de la generación de una gran cantidad de tráfico de red y múltiples accesos a la jerarquía de memoria, dando lugar a un consumo desmesurado de energía que no es traducible directamente en trabajo útil. Esta es otra de las conclusiones que se extraen directamente del segundo capítulo y a su vez introduce nuestro segundo trabajo, expuesto en esta tesis como el Capítulo 3.

Selective Dynamic Serialization (SDS) es la técnica hardware que se ha desarrollado para paliar las grandes pérdidas energéticas que supone la fase activa conocida como *stall*, donde continuamente se intenta el acceso a una dirección conflictiva de memoria. La solución pasa por convertir esta fase en pasiva y a la transacción en durmiente. Para evitar interbloqueos, la transacción en posesión de la dirección solicitada se convierte en la responsable de despertar a la primera. Sólo cuando una transacción se confirma puede despertar a las que de forma dinámica han colisionado con su conjunto de lectura o escritura previamente. Esta técnica supone un sofisticado sistema de herencia de transacciones serializadas en caso de que una transacción abortante debiera despertar a otras. El resultado es que en ausencia de contención, los sistemas *eager-eager* se ejecutan con total normalidad y en caso de contención, no son serializadas excepto en caso real de conflicto, realizando trabajo útil hasta el mismo. La sobrecarga introducida en tiempo de ejecución por los procesos de serialización (dormir) y des-serialización (despertar) de las transacciones es marginal comparado con el tiempo real de la

fase de *stall* en caso de contención. Además, en caso de una contención baja o moderada, SDS introduce un periodo de carencia donde se retrasa la entrada de una transacción en modo serializado mediante el uso de contadores de saturación. De este modo se permite o bien un número pequeño de abortos consecutivos, o bien una pequeña cantidad de mensajes NACK antes de activar el modo de serialización. Los resultados confirman que sin afectar al rendimiento de las transacciones, se produce en media un ahorro energético del 6 % siendo de un 42 % en escenarios con alta contención, y una reducción del tráfico de red del 22 % con unas necesidades hardware mínimas.

Por otra parte, a partir de los resultados del Capítulo 2 se observa que la política de mapeo de direcciones afecta de forma significativa en los sistemas *lazy-lazy* con un árbitro de confirmación descentralizado basado en los bancos de directorio de la caché de nivel 2. El mecanismo de obtención de privilegios para hacer la confirmación (*commit*) de las transacciones pasa por el árbitro distribuido que realiza la reserva de los bancos de nivel 2. Si diferentes transacciones mapean sus conjuntos de lectura y escritura no coincidentes a los mismos bancos de memoria, entonces competirán durante la fase de confirmación intentando pedir permisos a los mismos bancos de directorio de nivel 2 a pesar de que nunca habrá conflicto real entre ellas. De forma local para estas dos transacciones, el árbitro se comportaría como si fuese centralizado, alargando innecesariamente esta subfase de obtención de privilegios para lograr confirmación (*precommit*). Es la identificación del cuello de botella que supone esta subfase la que origina nuestro tercer trabajo desglosado en el Capítulo 4 de la presente tesis.

En esta tesis también se ha investigado el espacio de diseño de los sistemas *lazy-lazy* en profundidad. Si bien podemos concluir que la aproximación de estos sistemas es más eficiente en escenarios de alta contención, su escalabilidad con baja contención depende en gran medida de su habilidad para confirmar transacciones no conflictivas en un corto espacio de tiempo. Para ello es necesario acelerar la denominada fase de *precommit* donde se comprueba la existencia de conflictos previos durante su ejecución optimista. El protocolo de *commit* SEQ y su versión avanzada SEQPRO, consiguen esto mismo mediante la habilitación de confirmaciones paralelas de las transacciones. GCommit se presenta en el Capítulo 4 de esta tesis como una aproximación con un enfoque diferente. En lugar de intentar el máximo grado de paralelismo de *commits*, GCommit acelera dramáticamente la subfase de *precommit* valiéndose de una pequeña red de interconexión global y un conjunto de controladores (*G-Arbiters*) exclusivamente diseñados para esta tarea. El resultado es tajante, la fase de *precommit* se reduce a menos de 10 ciclos, lo que incrementa a su vez el número de confirmaciones por

periodo de tiempo. La implementación de esta red dedicada se puede realizar bien con el uso de tecnología no convencional como es la basada en GLines, o bien con tecnología estándar para los enlaces. En este Capítulo 4 se desarrolla el diseño de la red mediante tecnología estándar resultando en un consumo de energía marginal y en área de chip del 0,07 %. En términos generales de rendimiento y energía, GCommit consigue un 15,7 % y un 13,7 % de mejora con respecto a SEQ respectivamente gracias a un impresionante 68,5 % de aceleración en media de la fase de *precommit*. Además, GCommit también habilita la posibilidad de *precommits* paralelos mediante el uso de los *G-Arbiters* en la misma medida en que SEQ utiliza los bancos de directorios de nivel 2. Sin embargo las pruebas realizadas demuestran que cuando la fase de adquisición de privilegios se reduce tanto, la ventaja adicional de confirmaciones paralelas se compensa con la sobrecarga que se introduce al tener que alargar la fase de *precommit* teniendo que reservar más de un único *G-Arbiter*. Se concluye, por tanto, que GCommit sólo necesita de un único *G-Arbiter* para obtener su máximo rendimiento, con la ventaja que supone en el coste hardware.

Como parte de vías futuras de trabajo, planeamos extender GCommit implementando "*Parallel Reader Optimization*" para extraer también paralelismo durante el *precommit*. Si bien GCommit se comporta excelentemente sin paralelismo en esta fase y hemos comprobado que introduciendo más *G-Arbiters* no se consigue una mejora, esto sólo se produce en caso de imitar el compartimiento del protocolo de commit SEQ, no el de SEQPRO con la optimización paralela de lectores. El desafío se encuentra especialmente en dar con una configuración que mantenga los requisitos hardware al mínimo mientras que se consigue una mejora en rendimiento y energía en el *precommit*.

Por otra parte se ha comprobado que factores ajenos al programador afectan de manera crucial sobre el rendimiento de la memoria transaccional. Por ejemplo, el mapeo de direcciones físicas a bancos de directorio ha demostrado ser clave en el rendimiento de los sistemas *lazy-lazy*. Este es un factor que el programador no controla ni debería hacer. Sin embargo una mayor integración del software a nivel de compilador y la plataforma hardware subyacente puede conseguir esta optimización implícita y no intuitiva de los programas. Para ello nos proponemos desarrollar una serie de principios que den lugar primero a técnicas implementables en compiladores actuales y segundo a la implementación propia de un precompilador que analice el código de las transacciones y lo reordene consiguiendo explotar el máximo rendimiento posible que permanezca oculto en el programa.

Bibliografía

[1] Saman Amarasinghe. The looming software crisis due to the multicore menace. In *http://groups.csail.mit.edu/commit/papers/06/MulticoreMenace.pdf*, 2007. 1.1.2

[2] C. Scott Ananian, Krste Asanovic, Bradley C. Kuszmaul, Charles E. Leiserson, and Sean Lie. Unbounded transactional memory. In *HPCA-11*, Feb 2005. 1.1.4

[3] ARM. Cortex-a72 processor. In *https://www.arm.com/products/processors/cortex-a/cortex-a72-processor.php*, 2015. 1.1.1

[4] Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, and Katherine A. Yelick. The landscape of parallel computing research: A view from berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, December 2006. 1.1.1

[5] Lee Baugh, Naveen Neelakantam, and Craig B. Zilles. Using hardware memory protection to build a high-performance, strongly-atomic hybrid transactional memory. In *ISCA-35*, pages 115–126. IEEE Computer Society, 2008. 1.1.4

[6] Shekhar Borkar and Andrew A. Chien. The future of microprocessors. *Commun. ACM*, 54:67–77, May 2011. 1.1.1

[7] Luis Ceze, James Tuck, Josep Torrellas, and Calin Cascaval. Bulk disambiguation of speculative threads in multiprocessors. In *ISCA-33*, Jun 2006. 1.1.4

[8] Hassan Chafi, Jared Casper, Brian D. Carlstrom, Austen McDonald, Chi Cao Minh, Woongki Baek, Christos Kozyrakis, and Kunle Olukotun. A scalable, non-blocking approach to transactional memory. In *HPCA-13*, Feb 2007. 1.1.4

[9] Shailender Chaudhry, Robert Cypher, Magnus Ekman, Martin Karlsson, Anders Landin, Sherman Yip, Håkan Zeffer, and Marc Tremblay. Rock: A high-performance sparc cmt processor. *IEEE Micro*, 29(2):6–16, 2009. 1.1.4, 1.2

[10] Ben Chelf. Ensuring code quality in multi-threaded applications. In *http://www.coverity.com/library/pdf/coverity_multi-threaded_whitepaper.pdf*. 1.1.2

[11] Thomas Chen, Ram Raghavan, Jason N. Dale, and Eiji Iwata. Cell broadband engine architecture and its first implementation - a performance view. *IBM Journal of Research and Development*, 51(5):559–572, 2007. 1.1.1

[12] Cliff Click. Azul's experiences with hardware transactional memory. In *http://sss.cs.purdue.edu/projects/tm/tmw2010/talks/Click-2010_TMW.pdf*, 2010. 1.1.4, 1.2

[13] Intel Corporation. Intel architecture instruction set extensions programming reference. In *https://software.intel.com/sites/default/files/m/9/2/3/41604*, 2012. 1.1.4

[14] Intel Corporation. Intel 64 and ia-32 architectures optimization reference manual. In *http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf*, 2013. 1.1.4

[15] David Culler, J.P. Singh, and Anoop Gupta. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann, 1998. The Morgan Kaufmann Series in Computer Architecture and Design. 1.1.5

[16] Ian Cutress. The intel 6th gen skylake review: Core i7-6700k and i5-6600k tested. In *http://www.anandtech.com/show/9483/intel-skylake-review-6700k-6600k-ddr4-ddr3-ipc-6th-generation/9*, 2015. 1.1.6

[17] Peter Damron, Alexandra Fedorova, Yossi Lev, Victor Luchangco, Mark Moir, and Daniel Nussbaum. Hybrid transactional memory. In *ASPLOS-XII*, pages 336–346, 2006. 1.1.4

[18] Koen De Bosschere, Wayne Luk, Xavier Martorell, Nacho Navarro, Mike O´Boyle, Dionisos Pnevmatikatos, Alex Ramirez, Pascal Sainrat, André Seznec, Per Stenström, and Olivier Temam. High-Performance Embedded Architecture and Compilation Roadmap. *Transactions on High-Performance Embedded Architecture and Compilation*, 1(1):5–29, January 2007. 1.2

[19] Advanced Micro Devices. Family 15h models 30h-3fh amd a-series accelerated processor product data sheet. In *http://support.amd.com/TechDocs/51590_15h_Models_30h-3Fh_A-Series_PDS.pdf*, 2014. 1.1.1

[20] David Dice, Ori Shalev, and Nir Shavit. Transactional locking II. In *DISC-20*, Sep 2006. 1.1.4

[21] Michael Feldman. Ibm specs out blue gene/q chip. In *http://www.hpcwire.com/hpcwire/2011-08-22/ibm_specs_out_blue_gene_q_chip.html*, 2011. 1.1.4, 1.2

[22] David Geer. Industry trends: Chip makers turn to multicore processors. *IEEE Computer*, 38(5):11–13, 2005. 1.1.1

[23] Robert Golla. Niagara2: A highly threaded server-on-a-chip. In *http://www.opensparc.net/pubs/preszo/06/04-Sun-Golla.pdf*, 2006. 1.1.1

[24] Tom Groenfeldt. Software programmers lag behind hardware developments. In *http://blogs.forbes.com/tomgroenfeldt/2011/04/21/software-programmers-lag-behind-hardware-developments*, 2011. 1.1.2

[25] Lance Hammond, Vicky Wong, Michael K. Chen, Brian D. Carlstrom, John D. Davis, Ben Hertzberg, Manohar K. Prabhu, Honggo Wijaya, Christos Kozyrakis, and Kunle Olukotun. Transactional memory coherence and consistency. In *ISCA-31*, Jun 2004. 1.1.4

[26] Tim Harris, Adrián Cristal, Osman S. Unsal, Eduard Ayguadé, Fabrizio Gagliardi, Burton Smith, and Mateo Valero. Transactional memory: An overview. *IEEE Micro*, 27(3):8–29, 2007. 1.1.3, 1.1.5

[27] Tim Harris, James R. Larus, and Ravi Rajwar. *Transactional Memory, 2nd edition*. Synthesis Lectures on Computer Architecture. 2010. 1.1.4

[28] M. Herlihy, J. Eliot, and B. Moss. Transactional memory: Architectural support for lock-free data structures. In *ISCA-20*, May 1993. 1.1.3, 1.1.5

[29] Maurice Herlihy, Victor Luchangco, Mark Moir, and William N. Scherer-III. Software transactional memory for dynamic-sized data structures. In *PODC-22*, Jul 2003. 1.1.4

[30] Maurice Herlihy and J. Eliot B. Moss. Transactional memory: Architectural support for lock-free data structures. *SIGARCH CAN*, 21(2):289–300, 1993. 1.1.4

[31] Arik Hesseldahl. Moore's law is alive and well, and intel will prove it today. In *allthingsd.com*, 2011. 1.1.1

[32] Glenn Hinton, Dave Sager, Mike Upton, Darrell Boggs, Desktop Platforms Group, and Intel Corp. The microarchitecture of the pentium-4 processor. *Intel Technology Journal*, 1:1–13, 2001. 1.1.1

[33] Owen S. Hofmann, Donald E. Porter, Christopher J. Rossbach, Hany E. Ramadan, and Emmett Witchel. Solving difficult htm problems without difficult hardware. In *TRANSACT '07: 2nd Workshop on Transactional Computing*, 2007. 1.1.4

[34] David Kanter. Analysis of haswell's transactional memory. In *http://www.realworldtech.com/haswell-tm/3*, 2012. 1.1.4

[35] Leonidas I. Kontothanassis and Michael L. Scott. Efficient shared memory with minimal hardware support. *SIGARCH Computer Architecture News*, 23:29–35, 1995. 1.1.2

[36] Tushar Krishna, Amit Kumar, Patrick Chiang, Mattan Erez, and Li-Shiuan Peh. Noc with near-ideal express virtual channels using global-line communication. In *Hot Interconnects-16*, Aug 2008. 1.2

[37] Peter S. Magnusson, Magnus Christensson, Jesper Eskilson, Daniel Forsgren, Gustav Hallberg, Johan Hogberg, Fredrik Larsson, Andreas Moestedt, and Bengt Werner. Simics: A full system simulation platform. *IEEE Computer*, 35:50–58, Feb 2002. 1.2

[38] Virendra J. Marathe, William N. Scherer-III, and Michael L. Scott. Adaptive software transactional memory. In *DISC-19*, Sep 2005. 1.1.4

[39] John Markoff. Ibm scientists find new way to shrink transistors. In *http://www.nytimes.com/2015/10/02/science/ibm-scientists-find-new-way-to-shrink-transistors.html?nytmobile=0&_r=2*, 2015. 1.1.1

[40] Ami Marowka. Back to thin-core massively parallel processors. *IEEE Computer*, 44(12):49–54, 2011. 1.1.1

[41] Milo M. K. Martin, Colin Blundell, and E. Lewis. Subtleties of transactional memory atomicity semantics. *Computer Architecture Letters*, 5(2), 2006. 1.1.3

[42] 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 CAN*, 33(4):92–99, 2005. 1.2

[43] Chi Cao Minh, JaeWoong Chung, Christos Kozyrakis, and Kunle Olukotun. STAMP: Stanford transactional applications for multi-processing. In *IISWC-4*, Sep 2008. 1.2

[44] Chi Cao Minh, Martin Trautmann, JaeWoong Chung, Austen McDonald, Nathan Grasso Bronson, Jared Casper, Christos Kozyrakis, and Kunle Olukotun. An effective hybrid transactional memory system with strong isolation guarantees. In *ISCA-34*, pages 69–80. ACM, 2007. 1.1.4

[45] Kevin E. Moore, Jayaram Bobba, Michelle J. Moravan, Mark D. Hill, and David A. Wood. LogTM: Log-based transactional memory. In *HPCA-12*, pages 254–265, 2006. 1.1.4

[46] Ravi Rajwar, Maurice Herlihy, and Konrad K. Lai. Virtualizing transactional memory. In *ISCA-32*, pages 494–505, 2005. 1.1.4

[47] Ben Rooney. Arm ceo: Moore's law "alive and well". In *http://blogs.wsj.com /tech-europe/2011/03/15/arm-ceo-moores-law-alive-and-well*, 2011. 1.1.1

[48] Nir Shavit and Dan Touitou. Software transactional memory. In *PODC '95: Proceedings of the fourteenth annual ACM symposium on Principles of distributed computing*, pages 204–213. ACM, 1995. 1.1.4

[49] Arrvindh Shriraman, Virendra J. Marathe, Sandhya Dwarkadas, Michael L. Scott, David Eisenstat, Christopher Heriot, William N. Scherer III, and Michael F. Spear. Hardware acceleration of software transactional memory. In *Workshop on Languages, Compilers, and Hardware Support for Transactional Computing (TRANSACT)*, 2006. 1.1.4

[50] Sthephen L. Smith. 32nm westmere family of processors. In *http://download. intel.com/pressroom/kits/32nm/westmere/32nm WSM Press.pdf*, 2009. 1.1.1

[51] Fuad Tabba, Mark Moir, James R. Goodman, Andrew W. Hay, and Cong Wang. Nztm: nonblocking zero-indirection transactional memory. In *SPAA*, pages 204–213. ACM, 2009. 1.1.4

[52] Luke Yen, Jayaram Bobba, Michael R. Marty, Kevin E. Moore, Haris Volos, Mark D. Hill, Michael M. Swift, and David A. Wood. LogTM-SE: Decoupling hardware transactional memory from caches. In *HPCA-13*, Feb 2007. 1.1.4