Proyecto Fin de Carrera
Ingeniería en Informática

Coherent vs. non-coherent last level on-chip caches: an evaluation of the latency and capacity trade-offs

Alexandra Ferrerón Labari

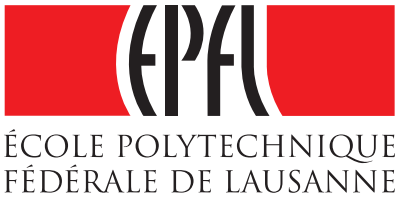
Director: Babak Falsafi
Ponente: Darío Suárez Gracia

Parallel Systems Architecture Lab.
Faculté Informatique et Communications
École Polytechnique Fédérale de Lausanne (Switzerland)

Departamento de Informática e Ingeniería de Sistemas
Centro Politécnico Superior
Universidad de Zaragoza

Curso 2010/2011
Noviembre 2010

A mis padres.



Proyecto Fin de Carrera
Ingeniería en Informática

Coherent vs. non-coherent last level on-chip caches: an evaluation of the latency and capacity trade-offs

Alexandra Ferrerón Labari

Director: Babak Falsafi
Ponente: Darío Suárez Gracia

Parallel Systems Architecture Lab.
Faculté Informatique et Communications
École Polytechnique Fédérale de Lausanne

Departamento de Informática e Ingeniería de Sistemas
Centro Politécnico Superior
Universidad de Zaragoza

Curso 2010/2011
Noviembre 2010

Resumen ejecutivo

El desorbitado consumo energético de los centros de datos actuales y la creciente preocupación por el medio ambiente han llevado a que las tecnologías de la información deban plantearse cómo reducir costes, a la vez que preservar el medio ambiente, en futuros centros de datos.

ARM, en un consorcio con Nokia, IMEC, EPFL (Escuela Politécnica Federal de Lausanne) y UCY (Universidad de Chipre), lidera el proyecto EuroCloud, en donde se pretende desarrollar una nueva generación de servidores-on-chip con tecnología 3D y de bajo consumo para servicios de computación en nube (cloud computing). EuroCloud propone un nuevo servidor-on-chip de muy bajo consumo, utilizando procesadores ARM, aceleradores de hardware y memoria DRAM en chip integrada en 3D.

En este proyecto hemos estudiado uno de los componentes principales del chip del proyecto EuroCloud, la jerarquía de memoria cache en chip, haciendo una comparación entre diferentes opciones para su organización. La configuración de la jerarquía de memoria cache en chip afectará al tiempo medio de acceso a memoria y, en consecuencia, influenciará el rendimiento global.

El chip que hemos estudiado está compuesto por dos clusters. Cada cluster contiene dos procesadores con sus respectivas caches de nivel uno privadas y una porción del segundo nivel de memoria cache (en este caso el segundo nivel de cache es el último nivel de la jerarquía). Este último nivel de cache se encuentra, por tanto, físicamente distribuido entre los clusters y puede ser configurado de forma distinta. En concreto, admite dos organizaciones: caches compartidas o caches privadas.

En este proyecto hemos analizado dos organizaciones: una organización Compartida, en la que los dos clusters comparten el último nivel de la memoria cache, y que pretende conseguir aprovechar al máximo la capacidad efectiva de la cache, y una organización en Cluster, en donde el último nivel de cache es privado para cada cluster. En este último caso, damos prioridad a un acceso más rápido (menor latencia) a este nivel de la jerarquía. Dentro de una organización en Cluster, hemos estudiado la posibilidad de introducir un mecanismo de coherencia para este nivel.

Tras una extensa labor de investigación sobre el estado del arte del tema y sobre la organización del chip y su arquitectura, hemos modelado los dos diseños antes mencionados en nuestra plataforma de simulación y simulado cargas de trabajo representativas. Hemos analizado en detalle los resultados obtenidos para distintos tamaños de memoria cache y concluido que una organización en Cluster, en general, funciona mejor. Un diseño Cluster se beneficia de una latencia de acceso más baja a la vez que proporciona en la mayoría de los casos la capacidad de cache necesaria para obtener un buen rendimiento. En los casos en que capacidad es más crítica que acceso o en cargas de trabajo con poca localidad, el diseño Compartido aventaja al diseño Cluster. En cuanto a los mecanismos de coherencia para este nivel de la jerarquía, creemos que, para el tipo de servidor estudiado y el tipo de aplicaciones consideradas, son innecesarios. Adicionalmente, hemos extendido el entorno de simulación utilizado, así como profundizado en la metodología de simulación para conseguir unos resultados más ajustados.

Índice general

Índice de tablas

Índice de figuras

Capítulo 1
Introducción

Las tecnologías de la información comienzan a considerar el coste energético y los problemas medioambientales como una variable importante en el diseño de chips. Actualmente, el consumo energético es uno de los factores principales en el coste total de los centros de datos [7], en donde microprocesadores y sistema de memoria son los componentes más costosos y que más energía consumen en un servidor [13]. En las últimas décadas, los diseñadores de microprocesadores han confiado en escalar el voltaje (*voltage scaling*) mediante la reducción del voltaje suministrado para así rebajar el consumo. Sin embargo, esta técnica implica un incremento de la corriente de fuga, limitando hasta dónde podemos reducir el voltaje sin incurrir en un aumento del consumo. Parece que integrar un gran número de procesadores de bajo consumo en chip o utilizar una nueva clase de arquitecturas de servidores basados en sistemas embebidos son formas de reducir el problema del consumo de energía en los futuros centros de datos.

ARM, en un consorcio con Nokia, IMEC, EPFL (Escuela Politécnica Federal de Lausanne) y UCY (Universidad de Chipre), lidera el proyecto EuroCloud, un proyecto cuyo objetivo es desarrollar una nueva generación de servidores *on-chip* con tecnología 3D y de bajo consumo para servicios de computación en nube (*cloud computing*) [26]. La computación en nube aparece como un nuevo paradigma que propone computación basada en Internet, en donde los recursos compartidos, software e información se proporciona a los computadores y otros dispositivos bajo demanda. El proyecto EuroCloud propone un servidor 3D en chip de muy bajo consumo utilizando procesadores ARM, aceleradores de hardware e integrando DRAM en 3D.

Las aplicaciones comerciales (por ejemplo bases de datos y servidores web) y aplicaciones emergentes de computación en nube (como por ejemplo *streaming* de música o vídeo, reconocimiento de canciones o *data mining*) presentan altos niveles de paralelismo a nivel de thread y de memoria, y pueden sacar muy poco partido a los complejos procesadores fuera de orden, que están especializados en extraer el paralelismo a nivel de instrucción. Este tipo de aplicaciones se pueden beneficiar mucho más de procesadores más simples en donde la latencia individual a nivel de thread es menos importante que el rendimiento global agregado.

En este proyecto estudiamos un componente clave del chip de ARM para el proyecto EuroCloud, desde una perspectiva de rendimiento y de consumo. Nos centramos en la jerarquía de memoria en chip, comparando diferentes configuraciones de memoria para el chip multiprocesador (CMP). Estudiamos si es preferible tener un último nivel de cache privado o compartido, y en el caso de privado, si las caches deberían ser coherentes o no coherentes.

1.1 Contexto del proyecto

Este proyecto ha sido desarrollado en el Laboratorio de Arquitectura de Sistemas Paralelos (PARSALab) en la Escuela Politécnica Federal de Lausanne (Suiza) y en el Grupo de Arquitectura de Computadores de la Universidad de Zaragoza.

El multiprocesador estudiado está formado por dos clusters de dos cores cada uno. Para cada procesador el primer nivel en la jerarquía de cache es privado. El último nivel de la jerarquía de cache (en nuestro caso el nivel dos [1]) está físicamente distribuido entre los clusters y puede ser configurado de diferentes formas. Dependiendo de esa configuración, el tiempo medio de acceso a memoria (que depende de tres parámetros principales: latencia, ancho de banda y tasa de aciertos) se verá influenciado de un modo u otro: una configuración de cache privada implica una menor latencia, mientras que caches compartidas incrementan la tasa de aciertos. Además, en el caso de una organización privada de caches, tendremos que determinar si se debería incluir una gestión explícita de la coherencia.

En este proyecto analizamos las diferentes configuraciones de este último nivel de memoria cache buscando la mejor opción, de acuerdo con la simulación de cargas de trabajo representativas. Como entorno de trabajo usamos *FLEXUS* [33]. *FLEXUS* es un simulador desarrollado por el grupo de Arquitectura de Computadores de la Universidad Carnegie Mellon (CALCM) que modela la arquitectura SPARC y puede ejecutar aplicaciones comerciales y sistemas operativos.

1.2 Objetivos

El objetivo de este proyecto es analizar cuál es la mejor opción para la jerarquía de memoria del chip de ARM para el proyecto EuroCloud. Las tareas principales en las que se puede dividir el proyecto son:

1. Estudio de la arquitectura ARM, la organización del chip y trabajos relacionados con el tema (estado del arte).
2. Extensión del entorno de simulación para modelar el multiprocesador de ARM.
3. Modelo de las diferentes organizaciones de memoria cache en chip:
   - Caches compartidas (distribución estática de bloques).
   - Caches privadas coherentes.
   - Caches privadas no coherentes.
4. Estudio de la metodología de simulación y simulación de cargas de trabajo representativas.
5. Análisis de resultados.

Hemos alcanzado los objetivos principales haciendo una comparación entre las configuraciones propuestas mediante la simulación de cargas de trabajo representativas. Debido a la falta de soporte por parte del entorno de simulación para modelar multiprocesadores heterogéneos, modelamos un multiprocesador simétrico de 16 cores desde donde podemos extrapolar los resultados.

[1] En este proyecto cuando hablamos del último nivel de cache (LLC) del chip multiprocesador estudiado, nos referimos al nivel dos (L2)

Además, hemos extendido el entorno de simulación permitiendo la simulación de multiprocesadores con dos cores.

1.3 Organización de la memoria

El resto de este documento se organiza de la siguiente manera: el capítulo 2 presenta el estado del arte y trabajos relacionados; el capítulo 3 explora las alternativas propuestas; el capítulo 4 explica la metodología que se ha seguido; el capítulo 5 contiene los resultados principales y el capítulo 6 concluye y presenta el trabajo futuro.

Se incluyen como apéndices (en inglés):

A. Project management (Gestión del proyecto): incluye calendarios y control de esfuerzos.

B. Statistical sampling simulation methodology: la metodología de simulación basada en muestreo estadístico para reducir el tiempo de simulación y cómo se aplica a nuestro proyecto.

C. Complementary (additional) results (Resultados complementarios): estudio detallado de los resultados obtenidos.

D. English Report: la versión en inglés del presente documento entregada en la EPFL (no incluye apéndices).

1.4 Agradecimientos

Me gustaría agradecer a mi director de proyecto Babak Falsafi el darme la oportunidad de tomar contacto con el mundo de la investigación durante este último año. También a todos los chicos del PARSALab, especialmente a Mutaz, Mike, Pejman y Mammad, que siempre estuvieron disponibles para resolver mis dudas sobre el entorno de simulación y sobre este proyecto en general. Gracias también a Mehdi, un buen *inteconnects-guy* con el que compartir despacho.

Gracias especialmente a Darío, que siempre ha estado ahí para resolver mis problemas y ayudarme a revisar y revisar este proyecto. Muchísimas gracias por tu ayuda.

A todos mis amigos, por todos los cafés, tanto aquí en Zaragoza como en Lausanne. A Markus, por todos los momentos en los que me has aguantado y ayudado.

Finalmente, quiero expresar mi sincera gratitud a toda mi familia, que siempre me ha apoyado en todas las decisiones que he tomado. Especialmente dedico este proyecto a mis padres. Sin vuestro apoyo nada de esto hubiera sido posible. Si hoy estoy donde estoy, es gracias a vosotros.

Capítulo 2
Trabajo relacionado

El escalado CMOS ha propiciado que los fabricantes de procesadores elijan, por todas las ventajas que ofrecen, chip multiprocesadores (CMP) como la arquitectura común para aprovechar el gran número de transistores disponibles y alcanzar, a la vez, alto rendimiento. Sin embargo, conforme el número de procesadores integrados en chip aumenta, también aumenta la presión en la memoria en chip, originada por la petición de datos por parte de esos procesadores. Al mismo tiempo, los CMPs requieren acceso rápido a los datos. El último nivel de la jerarquía de cache en chip (LLC) constituye un nuevo cuello de botella en la jerarquía de memoria que, no sólo necesita utilizar su capacidad limitada de una forma eficiente, sino que además tiene que mitigar las latencias que siguen incrementandose debido a los retrasos introducidos por los cables [18].

Hasta hace unos años parecía que incrementar el tamaño de este nivel de la jerarquía de cache era una buena manera de aprovechar los transistores disponibles. Incrementando la capacidad de la cache se pretendía conseguir mejor rendimiento. Algunos ejemplos de mega-caches en chip son el Dual-Core Intel Xeon 7100 con 16MB [28] o el Dual-Core Intel Itanium 2 con 24MB [36]. Sin embargo, aumentar el tamaño de la cache viene de la mano de un incremento en la latencia de acceso. Aumentar la latencia de la cache penaliza cada acceso y además incrementa el número de paradas en la ejecución causadas por aciertos en L2 (se incrementa el número de ciclos que debemos detener el pipeline hasta que el dato está disponible) sin cambiar el número de accesos a otras partes de la jerarquía de memoria.

Ailamiki *et al.* [17] demostraron que aplicaciones comerciales de bases de datos no consiguen ninguna mejora en el rendimiento (e incluso el rendimiento empeora hasta un 30 %) cuando se aumenta la capacidad del último nivel de cache de 4MB a 26MB. Si las caches son capaces de capturar el *working set* primario de este tipo de aplicaciones[1], entonces el rendimiento incrementa. Si continuamos incrementando el tamaño de las caches, entonces el mayor tiempo de acceso penaliza el caso común (acierto) introduciendo paradas en la ejecución, mientras que la capacidad adicional de la cache no es capaz de rebajar la tasa de fallos suficiente como para compensarlo.

Si nos centramos en la organización de la LLC, encontramos dos opciones para CMPs: caches compartidas o caches privadas. Una cache compartida tiene sólo una copia de cada bloque y permite a los procesadores compartir la capacidad de la cache. Sin embargo, las caches compartidas son lentas, debido a los retrasos asociados con caches grandes e interconexiones. Caches privadas son más rápidas porque son más pequeñas y se pueden colocar más cerca de cada procesador, pero su capacidad es limitada. Por tanto, caches compartidas o privadas proporcionan capacidad

[1]Estas aplicaciones tienen un *working set* primario (orden de MBs) que puede ser capturado en chip, y un *working set* secundario más grande (orden de GBs) que está fuera del alcance de las caches en chip actuales.

o acceso rápido, pero no ambos.

Recientemente, se ha hecho un gran esfuerzo en tratar de combinar las ventajas de diseños basados en caches privadas y compartidas, proponiendo diseños híbridos. En general, los diseños híbridos utilizan replicación selectiva para balancear latencia y capacidad [37, 6, 10]. Chang *et al.* presentan en [9] un marco unificado para manejar la capacidad agregada de los recursos de cache en CMPs, mediante la formación de una cache compartida a través de cooperación entre caches privadas. En la misma línea, Reactive-NUCA [16] propone clusterizar los procesadores en grupos de compartición para minimizar los fallos que suponen comunicación con memoria principal. R-NUCA realiza también una clasificación de los bloques que se beneficiarían de ser privados, compartidos o clusterizados, en lo que se refiere a referencias a la LLC. Basándose en esta observación, los autores proponen una diseño de cache que, cooperando con el sistema operativo, reacciona a las distintas clases de accesos y coloca los bloques en el lugar adecuado.

Por otro lado, los complejos procesadores fuera de orden están especializados en extraer el paralelismo a nivel de instrucción. Sin embargo, las aplicaciones comerciales siguen un patrón diferente. Para estos tipos de aplicaciones la latencia individual de un thread es menos importante que el rendimiento global agregado. Podemos incrementar el rendimiento agregado con procesadores multithread, de manera que eventos que normalmente paran el procesador, como fallos de cache, se ocultan, aumentando así la utilización. Además, usando procesadores escalares sencillos reducimos la complejidad del diseño, además del consumo de energía.

Un trabajo de la Universidad de Michigan y ARM propone una nueva arquitectura llamada PicoServer [20]. Este trabajo afirma que integrando 3D DRAM en chip podemos prescindir de la cache L2. El rendimiento se mantiene mediante el uso de buses muy anchos, una pequeña memoria DRAM en chip e incrementando el número de cores, mientras que ahorramos una considerable cantidad de energía. Esta hipótesis la soporta la idea de que aplicaciones para servidores en el futuro se ejecutarán en CMPs con un gran número de pequeños cores simples de ejecución en orden [12]. De todas formas, incrementar el número de cores implica una alta demanda de datos para alimentar esos cores, por lo que el rendimiento puede empeorar si no hay suficiente ancho de banda. La situación en la que el ancho de banda con la memoria fuera del chip se convierte en un cuello de botella para el rendimiento se conoce como el *bandwidth wall problem* [27]. Además, los buses son una fuente significante de pérdida de energía, especialmente los buses *interchip*, que son normalmente muy largos y anchos.

El proyecto EuroCloud sigue la idea de incluir muchos procesadores de bajo consumo en chip y usar DRAM integrada en 3D. La integración de DRAM en 3D ha recibido gran atención en Arquitectura de Computadores en los últimos años, ya que la interconexión en vertical permite tener sistemas con latencias más bajas y anchos de banda mayores [23]. Estos aspectos son muy atractivos para servidores y procesadores de alto rendimiento por sus altos requerimientos de memoria y niveles de paralelismo a nivel de thread. La memoria tipo DRAM estándar (fuera de chip) tiene un ancho de banda limitado debido a la limitación en el número de pins, es lenta porque para acceder a ella hay que recorrer todo el chip, y consume mucha energía debido a la circuitería y entrada/salida [26]. La integración 3D de DRAM sobre la lógica, en combinación con aceleradores de hardware, puede eliminar esas ineficiencias.

Nuestro trabajo se centra en un componente clave (la jerarquía de memoria en chip) de una nueva clase de servidores-on-chip basados en sistemas embebidos. Desde nuestro conocimiento, es la primera vez que se estudia el rendimiento de aplicaciones comerciales en sistemas embebidos.

Capítulo 3
Alternativas para la organización de cache en multiprocesadores embebidos

Este capítulo presenta las diferentes alternativas para la jerarquía de memoria en chip analizadas en este proyecto. Exponemos las principales características de cada alternativa para concluir formulando nuestra hipótesis.

3.1 Introducción

El objetivo de la jerarquía de memoria cache es minimizar la latencia de los datos a los que se accede de manera frecuente y así maximizar el rendimiento. En la jerarquía de cache de un uniprocesador convencional, conseguimos ese objetivo aprovechando la localidad espacial y temporal de los datos; esto quiere decir que movemos los bloques de datos en la jerarquía de memoria basándonos en la frecuencia de acceso. Los datos más referenciados se colocan cerca del procesador y por lo tanto se acceden más rápidamente. El mismo principio se puede aplicar a jerarquías de cache en multiprocesadores. La diferencia es que ahora tendremos que tener en cuenta también si los procesadores comparten o no un determinado nivel de la jerarquía, o si un nivel se implementa como un bloque físico con latencia de acceso uniforme a la cache (*uniform cache access* o UCA), o si por el contrario, se implementa como múltiples bancos físicamente distribuidos, en donde la latencia varía según qué procesador accede a los datos y los bancos de la cache se encuentran distribuidos en el chip, implicando que la latencia de acceso a los datos es no uniforme (es decir, *non-uniform cache access* o NUCA) [21].

Conforme el número de procesadores y bancos de cache aumenta, las caches físicamente distribuidas son más atractivas desde una perspectiva de diseño, manufactura y escalabilidad [3]. Pero también colocar una porción de la memoria cache cerca de un subconjunto de procesadores puede reducir la latencia de acceso a esa porción de la cache para esos procesadores, en lugar de ofrecer una latencia alta, pero igual, en toda la cache (diseños UCA).

Las arquitecturas tiled surgen como una solución para mitigar la creciente latencia de acceso al último nivel de cache [21]. En este tipo de arquitecturas, el chip se divide en un gran número de tiles idénticos (o casi idénticos) que se interconectan mediante una red de interconexión escalable y de bajo consumo. Cada tile contiene una porción de la LLC (que puede tener múltiples bancos), por lo que la cache se encuentra físicamente distribuida en el chip. De esta forma, los procesadores pueden acceder muy rápido a las porciones de cache que se encuentran cerca de ellos, aunque tienen que pagar la latencia de viajar a través de la red de interconexión si los

datos a los que quieren acceder se encuentran más lejos (NUCA).

Si el último nivel de cache (LLC) se distribuye en múltiples tiles, caches privadas y caches compartidas introducen distintos compromisos. En general, una cache compartida es preferible si lo que queremos es reducir el número colectivo de fallos de LLC, mientras que una organización con caches privadas es mejor si lo que queremos reducir es la latencia de acceso y la complejidad del diseño.

Caches compartidas incrementan la capacidad efectiva porque sólo una copia de cada bloque se encuentra en la cache. Si asumimos una distribución estática de los datos (basada en la dirección de bloque), los bloques pueden colocarse arbitrariamente lejos del procesador que los solicita, lo que penaliza el tiempo medio de acceso. Por otro lado, un diseño privado colocará los bloques cerca del procesador(es) que los soliciten, así que garantizamos un acceso rápido a esos datos. La desventaja es que puesto que bloques compartidos de sólo lectura estarán replicados en varios tiles, la capacidad efectiva de la cache disminuye y, en consecuencia, aumenta el tráfico fuera de chip.

El protocolo de coherencia puede jugar también un papel importante. Mientras que en un diseño compartido la coherencia es implícita por construcción, cuando hablamos de un diseño privado nos encontramos con dos posibilidades: caches coherentes o caches no coherentes. En caso de que el último nivel de cache sea coherente, el tráfico dentro del chip crece (mensajes para gestionar la coherencia), con los problemas de disipación de energía que esto conlleva [3]. Además, los mecanismos de coherencia reducen el área disponible y penalizan la compartición de datos. En el caso de caches no coherentes, éste tipo de diseño demandará tráfico extra fuera del chip, con los problemas de ancho de banda con memoria principal derivados [27]. Si elegimos un diseño privado coherente, los protocolos basados en directorio parecen la mejor opción para gestionar la compartición de bloques en el chip [8, 30].

3.2 Organización básica de servidores-on-chip y opciones para la jerarquía de memoria

En este proyecto evaluamos dos configuraciones diferentes para el último nivel de memoria cache para el chip del proyecto EuroCloud (figura 3.1). El multiprocesador que estudiamos está formado por dos clusters de dos procesadores cada uno, en donde para cada procesador el primer nivel de cache es privado. El último nivel de cache (en este caso nivel 2) está físicamente distribuido entre los clusters. La comunicación entre clusters se realiza a través de una red de interconexión. La *Snoop Control Unit* (SCU) es la encargada de la interconexión, arbitraje, comunicación, transferencias *cache-to-cache* y con el sistema de memoria y coherencia.

Nuestro objetivo es explorar las diferentes alternativas para la organización de la jerarquía de memoria en chip, proponiendo dos organizaciones para la LLC:

1. LLC Compartida: los dos clusters comparten la LLC en chip. En este caso suponemos que la distribución de los bloques es estática (basada en la dirección de bloque). Notar que este diseño es coherente por construcción.

2. LLC Privada: la LLC es privada para cada cluster. En este caso podemos considerar un diseño coherente o no coherente.

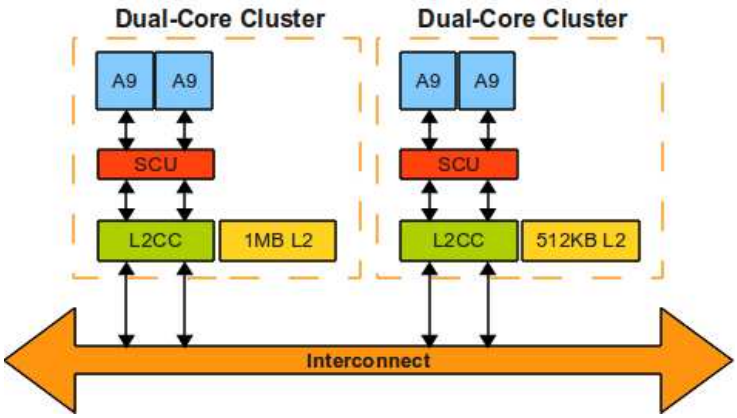
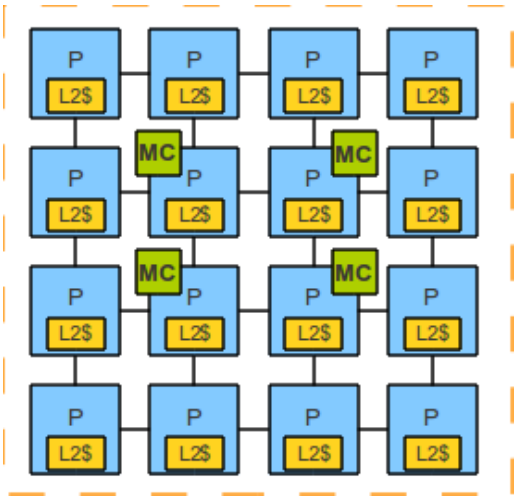


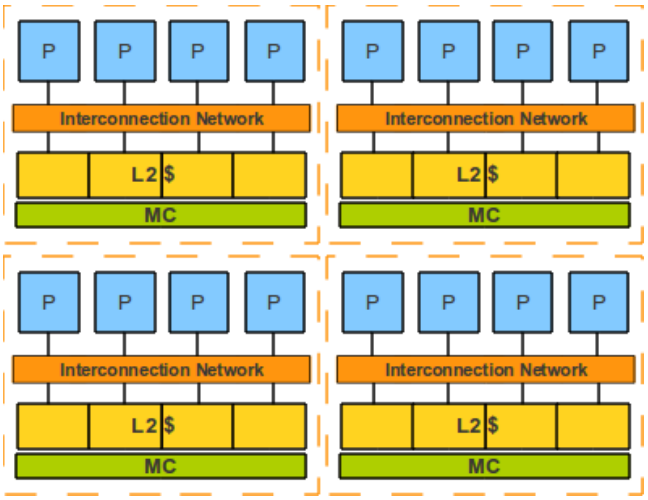
Figura 3.1: Diseño (aproximado) del chip de ARM. Sólo se incluye el detalle de la jerarquía de memoria en chip. L2CC son las iniciales de *L2 cache controller*.

3.2.1 Alternativas consideradas

Debido a las restricciones impuestas por el entorno de simulación, decidimos extender nuestro análisis a un multiprocesador de 16 cores, explorando dos diseños diferentes:



(a) Diseño Compartido de LLC (Baseline o Shared).



(b) Diseño Cluster de LLC (Cluster).

Figura 3.2: Alternativas de diseño consideradas.

1. Diseño Compartido (Baseline o Shared): consiste en un sistema de 16 nodos, con NUCA y una LLC (L2) compartida, interconectada por una red (*mesh*) 4 por 4 (figura 3.2a). En total hay cuatro controladores de memoria. Este diseño pretende sacar el máximo partido a la capacidad efectiva de la cache.

2. Diseño en Cluster (Cluster): también se trata de un sistema de 16 nodos, pero los procesadores se dividen en cuatro grupos (clusters) de cuatro procesadores cada uno; la cache L2 es una cache compartida por los procesadores dentro de cada grupo, pero es privada entre los cuatro clusters (figura 3.2b). La LCC es multibanco y se accede a través de una red de interconexión. Cada cluster tiene un controlador de memoria. Este diseño da prioridad a conseguir una latencia de acceso más baja.

Vemos que los resultados basados en estos diseños pueden ser claramente extrapolables al CMP original (en todo caso los resultados obtenidos podrían degradar el rendimiento ya que aumentamos el número de procesadores y por tanto la presión en el chip).

3.2.2 Tamaño ideal del último nivel de memoria cache

Diferentes tipos de aplicaciones y diferentes tipos de paralelismo en esas aplicaciones presentan diferentes retos a nivel arquitectural y de diseño para la jerarquía de memoria a la hora de alcanzar objetivos que tienen que ver con el rendimiento, la escalabilidad y el ahorro de energía. En nuestro estudio nos centramos en cargas de trabajo comerciales, pues son el tipo de aplicaciones que los servidores de hoy en día ejecutan.

Hardavellas *et al.* en [16] hicieron un estudio sobre los requerimientos de aplicaciones comerciales en términos de capacidad y organización de LLC, proponiendo un nuevo esquema de distribución de bloques y organización de la cache. De ese estudio podemos extraer varias conclusiones:

- Las referencias a L2 forman de manera natural tres grupos con diferentes características: (1) las instrucciones se comparten entre todos los procesadores y son sólo lectura; (2) los datos compartidos se comparten entre todos los procesadores y son lectura-escritura en la mayoría de los casos; (3) los datos privados presentan distintos niveles de lectura-escritura.
- El *working set* de instrucciones para cargas de trabajo comerciales es aproximadamente del tamaño de una porción de la cache L2 (en su estudio, 1MB por porción).
- El primer nivel de memoria cache puede capturar el conjunto de datos (*data working set*).
- Los mecanismos de coherencia hardware para L2 en CMPs con arquitecturas tiled y ejecutando aplicaciones comerciales son innecesarios y deberían evitarse.

Basándonos en estas conclusiones podemos afirmar:

- Si la LLC es capaz de capturar el *working set* primario de la carga de trabajo [17], entonces el diseño que garantice la menor latencia tendrá mejores resultados.

Podemos formular entonces nuestra hipótesis afirmando que siempre que la capacidad de la cache sea mayor que un determinado tamaño (el tamaño que permita almacenar en cache el *working set* primario), esperamos que el diseño Cluster obtenga un rendimiento mejor.

Si el diseño Cluster no es capaz de capturar el *working set* primario de la carga de trabajo, entonces la petición de datos irá al siguiente nivel de la jerarquía, es decir, a la memoria fuera de chip, lo que incrementará el tiempo medio de acceso. Por otro lado, caches grandes con una distribución estática de los bloques pueden implicar accesos a porciones de cache arbitrariamente lejos del procesador que solicita el dato y, en consecuencia, una latencia mayor debido a la red de interconexión.

Por último, implementando mecanismos de coherencia en el último nivel de la jerarquía de cache, podríamos beneficiarnos de las transferencias *cache-to-cache* (pero pagando un alto precio en términos de tráfico en chip y hardware [9]). Las transferencias *cache-to-cache* se usan para reducir las peticiones de datos fuera del chip para fallos locales en LLC, pero estas operaciones requieren comunicación entre el tile que solicita el dato, el tile en el que se encuentra el directorio

(para la coherencia) y el tile que en ese momento es el dueño del dato. Esta operación es más costosa que un acierto remoto en una cache compartida (esto es el caso de un acierto en LLC en el que el dato se encuentra en otro tile), donde la transferencia *cache-to-cache* sólo se realiza si el bloque solicitado se encuentra en estado exclusivo. Además, el hardware que se requiere para implementar los mecanismos de coherencia reduce el área disponible para el LLC y consecuentemente limita la capacidad de la cache.

Siguiendo nuestra hipótesis (con la que esperamos que el diseño Cluster obtenga mejores resultados), la adición de un mecanismo de coherencia puede degradar el rendimiento, además de complicar el hardware e incrementar el consumo de energía. Por estas razones decidimos no explorar un diseño privado coherente.

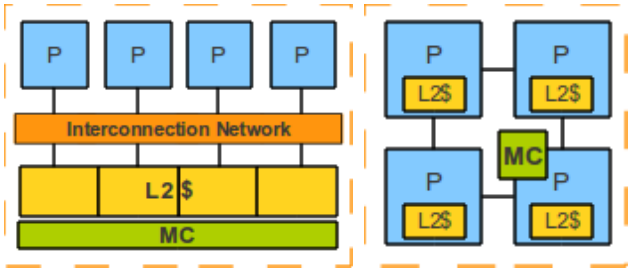
Para estimar el tamaño de la cache L2 (LLC) para nuestros diseños, asumimos un chip de 180 mm^2 en tecnología de 45 nm. Contamos que el 65 % del chip será utilizado para colocar los tiles. Asumimos un área de 3,1 mm^2 para cada Cortex A9, de forma que nos queda un tamaño por porción de L2 de 4,2125 mm^2, lo que nos permite sin problemas colocar una porción de hasta 1MB de LLC.

Respecto a la asociatividad de la cache, fijamos este valor a 16, puesto que es un valor adecuado y ampliamente utilizado para este nivel de la jerarquía de cache [28, 22].

Resumiendo, exploramos tamaños de cache desde 128KB hasta 1MB por procesador, lo que supone una capacidad total de cache desde 4MB hasta 16MB en chip. Esperamos que el diseño Compartido presente un rendimiento mejor hasta un determinado tamaño de cache, y que a partir de ese tamaño el diseño Cluster supere al diseño Compartido al aprovechar su menor tiempo de acceso. Estamos de acuerdo con la idea de que los mecanismos de coherencia para este nivel de jerarquía de cache y este tipo de cargas comerciales deberían evitarse, y no esperamos que introducir un mecanismo de coherencia vaya a incrementar el rendimiento.

3.2.3 Detalles de implementación

Finalmente comentar que en la práctica no implementamos el diseño Cluster tal y como se propone en la figura 3.2b.



(a) Diseño propuesto para Cluster. **(b)** Diseño implementado para Cluster.

Figura 3.3: Diseños propuesto e implementado para Cluster.

En la figura 3.2b, los procesadores se comunican con la memoria cache L2 a través de una red de interconexión. Para este tipo de conexiones, un *crossbar* parece ser la opción más adecuada.

Sin embargo, en vez de simular el diseño propuesto decidimos simular una arquitectura tiled interconectada mediante una red 2 por 2. La figura 3.3 muestra el diseño propuesto (3.3a) y el diseño implementado (3.3b).

Notar que *crossbars* con pocos puertos tienen una latencia de acceso media más baja [22, 11], así que podemos considerar el diseño que hemos implementado (red 2 por 2) como una cota inferior de rendimiento.

Capítulo 4
Metodología

Este capítulo presenta la metodología que se ha seguido en este proyecto. Se presenta brevemente la metodología de simulación y las cargas de trabajo (benchmarks) que hemos usado. Más información en los apéndices B (sobre metodología de simulación) y C (resultados detallados).

4.1 Simulador

Utilizamos el entorno de simulación *FLEXUS* [33] para conseguir simular de forma precisa chips multiprocesadores y multiprocesadores simétricos ejecutando cargas de trabajo comerciales. *FLEXUS* es un simulador capaz de simular código tanto a nivel de usuario como de sistema operativo. Utilizamos la metodología de *SimFlex statistical sampling* (muestreo estadístico) [33]. *FLEXUS* extiende el simulador funcional *Virtutech Simics* [24] con modelos de arquitecturas tiled con procesadores, caches NUCA, controladores de protocolos de coherencia e interconexiones en chip.

Lanzamos simulaciones desde checkpoints con caches y predictores de salto inicializados, ejecutamos durante 100.000 ciclos para inicializar la ventana de lanzamiento y la red de interconexión antes de recoger resultados de la ejecución del programa durante 50.000 ciclos. Utilizamos como métrica el número de instrucciones de usuario por ciclo (*user instructions per cycle* o UIPC), es decir, el número de instrucciones ejecutadas (*comitted*) por todos los procesadores dividido por el total de ciclos ejecutados, que es proporcional al rendimiento global del sistema [33].

Para estimar las latencias de las caches usamos Cacti 6.5 [34, 25]. Cacti es un modelo de caches que integra tiempo de acceso, tiempo de ciclo, área y consumo de energía. De esta forma todas las variables se basan en los mismos supuestos y son, por tanto, consistentes. Además las latencias que proporciona Cacti son normalmente más bajas que las del hardware comercial, por lo que nuestras suposiciones son conservativas.

4.2 Diseño del sistema

Para nuestros experimentos modelamos en *FLEXUS* el sistema que aparece en la tabla 4.1. Aunque *FLEXUS* modela la arquitectura UltraSPARC III, los parámetros del simulador se han ajustado para hacer el procesador lo más parecido posible al Cortex A9 de ARM [2].

La tabla 4.1 recoge los parámetros que no cambian entre los distintos diseños propuestos. Como ya hemos explicado en el capítulo 3, exploramos dos diseños cuya diferencia principal es la organización del último nivel de cache de la jerarquía (L2). Para el modelo Compartido (Shared) asumimos un sistema de 16 nodos donde cada procesador tiene una porción de la cache L2 (arquitectura tiled). Para nuestro diseño Cluster agrupamos los 16 procesadores en cuatro grupos o clusters de cuatro cores. Cada cluster comparte el último nivel de cache, y este nivel de cache es privado entre clusters. Exploramos tamaños de cache desde 128KB hasta 1MB por core, lo que supone un tamaño total entre 4MB y 16MB en chip. Esto corresponde, desde nuestro punto de vista, con un tamaño aceptable para la tecnología y el tipo de procesador estudiado (ver capítulo 3 para más detalles, incluyendo suposiciones sobre el área y el tamaño de la cache).

Nuestro protocolo de coherencia en chip es un protocolo MESI de cuatro estados basado en Piraña [5]. Simulamos un controlador de memoria por cada cuatro cores, cada controlador colocado en un tile, asumiendo comunicación con la memoria fuera de chip a través de tecnología *flip-chip*. Los tiles se comunican entre ellos a través de la red de interconexión en chip.

Parametros del CMP	
Tamaño CMP	16-core // 4x4-core clusters
Procesadores	UltraSPARC III ISA; 2GHz, OoO cores 8-stage pipeline, 2-wide dispatch/retirement 8-entradas ROB y LSQ, 4-entradas store buffer
Predictor de saltos	8K GShare + 16K bi-modal + 16K selector 2K entradas, 16-way, 2 saltos por ciclo
L1 Caches	Split I/D, 4-way 32KB 2-ciclos load-to-use, 2 puertos, LRU 64-byte bloques, 32 MSHRs, 8-entradas victim cache
L2 Cache	Tamaño variable dependiendo del diseño 16-way, NUCA 64-byte bloques, 32 MSHRs, 16-entradas victim cache
Memoria principal	3GB, 8KB tamaño de página, 75 cycles access time
Controladores memoria	Uno cada 4 cores, interleaving de páginas round robin
Interconexión en chip	Mesh (4x4 para Baseline, 2x2 para Cluster) 8-byte links, 3-cycles link latencia

Tabla 4.1: Parámetros del sistema.

4.3 Workloads (cargas de trabajo)

Simulamos sistemas bajo el sistema operativo Solaris 8 y ejecutamos cargas de trabajo comerciales formadas por un amplio rango de aplicaciones de servidores de distintos vendedores, incluyendo procesamiento de transacciones en línea, sistemas de soporte de decisiones y servidores web [4]. Los detalles técnicos de las aplicaciones se recogen en la tabla 4.2.

A continuación listamos las tres categorías de cargas de trabajo que hemos usado en este proyecto:

- **Servidores Web:** SPECWeb99 [29] sobre Apache y Zeus en este estudio. Estos benchmarks se pueden caracterizar por altos niveles de paralelismo a nivel de thread (*thread level parallelism* o TLP), alto número de fallos obligatorios en cache que detienen la máquina debido al tráfico con memoria, y un gran porcentaje de tiempo de kernel debido a manejo de interrupciones, trasmisión de paquetes y procesamiento de red. [20]

- **Procesamiento de transacciones en línea:** (*online transaction processing* o OLTP) las aplicaciones OLTP poseen uno de los segmentos más grandes en el mercado de servidores. Se usan en las operaciones comerciales del día a día (por ejemplo, reservas de vuelos) y se caracterizan por un gran número de clientes que acceden continuamente a una base de datos y realizan pequeñas modificaciones a través de transacciones. Los benchmarks de esta categoría que hemos incluido son TPC-C [31] sobre Oracle y DB2. Ambos utilizan un sistema gestor de bases de datos para realizar un gran número de pequeñas transacciones.

- **Sistema de soporte de decisiones:** (*decision suport systems* o DSS) los sistemas DSS se usan principalmente con propósitos de análisis comercial, en donde la información procedente del lado OLTP de un negocio se carga periódicamente en una base de datos DSS y se analiza. Al contrario que OLTP, DSS se caracteriza por consultas (queries) largas, principalmente sólo lectura, que pueden explorar una gran fracción de la base de datos. En este proyecto utilizamos benchmarks DSS que se basan en el benchmark TPC-H [32]. Consisten en queries y modificaciones de datos concurrentes en grandes bases de datos con el objetivo de dar respuesta a cuestiones de negocios, examinar grandes volúmenes de datos y ejecutar consultas mucho más complejas que la mayoría de las transacciones OLTP. Seleccionamos para nuestro estudio queries *scan-dominated* (query 1), *join-dominated* (query 2) y queries con compartamiento mixto o *mixed-behavior* (query 17).

Web Server (SPECWeb99)	
Apache	Apache HTTP Server v2.0. 16K conexiones, fastCGI, worker threading model
Zeus	16K conexiones, fastCGI
OLTP - Online Transaction Processing (TPC-C v3.0)	
DB2	IBM DB2 v8 ESE. 100 warehouses (10 GB), 64 clientes, 2 GB buffer pool
Oracle	Oracle 10g Enterprise Database Server. 100 warehouses (10 GB), 16 clientes, 1.4 GB SGA
DSS - Decision Support Systems (TPC-H)	
Queries 1, 2, 17	IBM DB2 v8 ESE, 480 MB buffer pool, 1 GB base de datos

Tabla 4.2: Parámetros de las aplicaciones.

Capítulo 5
Resultados

Este capítulo presenta un resumen de los resultados obtenidos en este proyecto. El apéndice C recoge todos los resultados de manera detallada, incluyendo más detalles sobre los benchmarks que hemos utilizado, así como sobre las métricas empleadas.

5.1 Visión general de resultados

Hemos simulado las cargas de trabajo que aparecen en la tabla 4.2 y presentamos como nuestra métrica de rendimiento el número total de instrucciones ejecutadas (*commited*) por ciclo (*user instructions per cycle* o UIPC), es decir, el número total de instrucciones ejecutadas por todos los procesadores dividido por el total de ciclos ejecutados. Esta métrica es proporcional al rendimiento global del sistema [33]. Exploramos los dos diseños presentados en el capítulo 3 y variamos el tamaño de la cache desde 128KB hasta 1MB por procesador (lo que supone un total de cache en chip desde 4MB hasta 16MB).

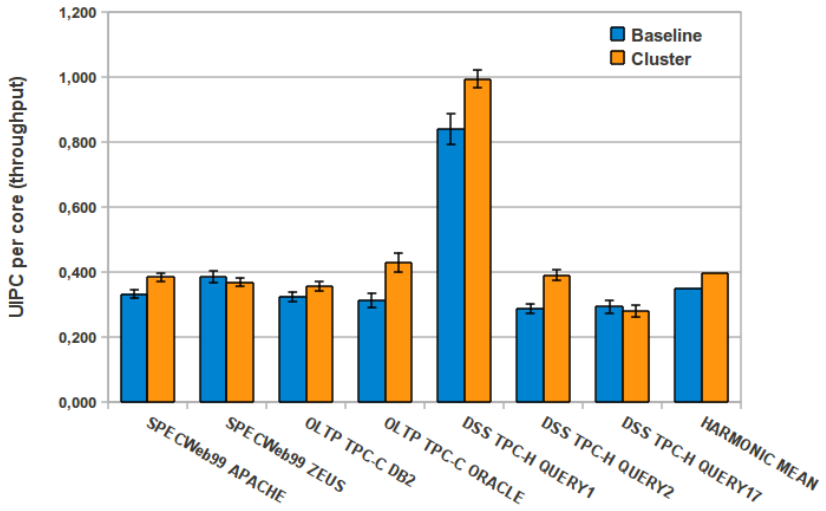


Figura 5.1: Mejor resultado (UIPC por core) para benchmarks comerciales.

La figura 5.1 muestra los mejores resultados para cada benchmark para los diseños Compartido (Shared) y Cluster, así como la media armónica. El tamaño del último nivel de la jerarquía de cache que proporciona el mejor rendimiento se puede ver en la tabla 5.1. En todos los casos, excepto para SPECWeb99 sobre Zeus y DSS TPC-H Query 17, el diseño Cluster funciona me-

Workload	Tamaño de cache (mejor)	
	Diseño Compartido (Shared)	Diseño Cluster
SPECWeb99 APACHE	16 MB	4 MB
SPECWeb99 ZEUS	16 MB	4 MB
OLTP TPC-C DB2	16 MB	4 MB
OLTP TPC-C ORACLE	16 MB	4 MB
DSS TPC-H QUERY 1	8 MB	512 KB
DSS TPC-H QUERY 2	8 MB	4 MB
DSS TPC-H QUERY 17	4 MB	2 MB

Tabla 5.1: Tamaño de la cache con el que se obtiene el mejor rendimiento para diferentes benchmarks/diseños. El tamaño de la cache corresponde al tamaño de cache visible para cada procesador (es decir, en el diseño Cluster una cache de 4MB corresponde a una cache de 16MB en chip).

jor. También tiene un mejor rendimiento en media. Los resultados en general siguen la misma tendencia para todos los benchmarks y tamaños de cache.

5.2 Resultados representativos

En esta sección exploramos algunos resultados representativos para cada tipo de benchmark, en concreto: OLTP TPC-C Oracle, Web server Zeus y DSS TPC-H Query 17. OLTP-C TPC-C Oracle es un benchmark representativo para el conjunto considerado, además de corroborar nuestra hipótesis inicial. Web server Zeus y DSS TPC-H Query 17 son los únicos benchmarks para los que el diseño Cluster no obtiene mejores resultados que el diseño Compartido. Las razones por lo que esto ocurre se presentan también en esta sección.

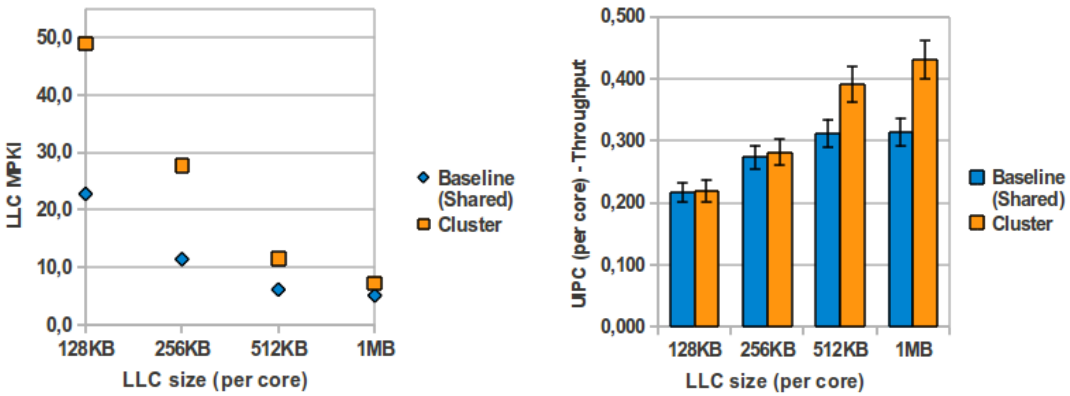
5.2.1 OLTP TPC-C: Oracle

Nuestros benchmarks OLTP includen TPC-C sobre DB2 y Oracle. En esta sección nos centramos en OLTP TPC-C sobre Oracle como un claro ejemplo de benchmark que corrobora nuestra hipótesis.

Ailamaki *et al.* [17] ya observaron en su estudio sobre aplicaciones de bases de datos los requerimientos de este tipo de aplicaciones. Siempre que el *working set* primario de la carga de trabajo no supere la capacidad del último nivel de cache (LLC), el rendimiento incrementa. A partir del punto en el que este *working set* cabe en la cache, caches más grandes pueden empeorar el rendimiento. Podemos ver esta tendencia claramente en la figura 5.2.

Es importante entender la diferencia entre nuestros diseños Baseline (Compartido) y Cluster. Aunque ambos diseños cuentan con la misma cantidad de cache en chip, la cantidad de cache que cada procesador ve es diferente. Es decir, si consideramos 1MB por core de LLC, en el caso del diseño Compartido, cada core puede acceder al total de la cache, 16MB; en el caso de Cluster, las caches son privadas para cada cluster, por lo que el tamaño de LLC visible para cada core es 4MB.

Oracle obtiene mejor rendimiento con un diseño Compartido con caches pequeñas. Una vez que los requerimientos de cache se satisfacen (a la vista de la figura 5.2b, 512KB por core), el diseño Cluster supera al diseño Compartido. De hecho, podemos ver como Oracle presenta una



(a) LLC MPKI para OLTP Oracle.

(b) UIPC para OLTP Oracle.

Figura 5.2: LLC MPKI y UIPC para OLTP Oracle.

diferencia de 0,1 UIPC (por core) cuando consideramos 1MB de cache por procesador.

La figura 5.2a muestra los fallos por 1000 instrucciones (MPKI) de LLC para Oracle. Esta gráfica sigue la misma tendencia: vemos como el número de fallos por 1000 instrucciones baja de manera considerable cuando pasamos de 128KB a 256KB por core (Cluster) pero apenas hay variación cuando aumentamos el tamaño de 512KB a 1MB por core (tanto Shared como Cluster).

Ilustrando todo lo que hemos comentado, la figura 5.3 muestra el desglose del tiempo de ejecución (de usuario) para Oracle (diseño Cluster). Podemos ver que una cache de 512KB no es capaz de capturar el *working set* primario del benchmark y por eso una gran parte del tiempo de ejecución se debe a comunicación con la memoria fuera del chip (*off-chip*). Con 1MB de cache por core vemos que ese tiempo se reduce desde el 50 % (128KB por core) a menos del 10 % del tiempo total de ejecución de usuario. Este decremento se traduce en un aumento de dos veces el rendimiento (figura 5.2b).

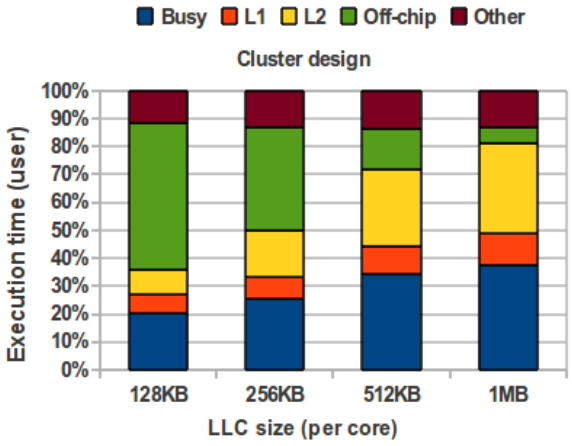
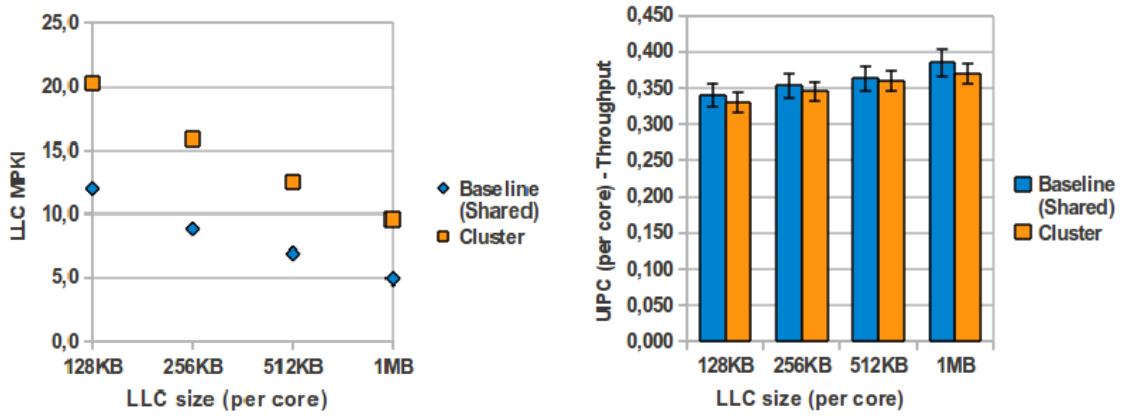


Figura 5.3: Desglose del tiempo de ejecución (usuario) para OLTP Oracle (Cluster).

5.2.2 Web server: Zeus

Nuestra categoría de benchmarks de servidores web incluye SPECWeb99 sobre Apache y Zeus. Ambas presentan una tendencia similar en lo que se refiere a rendimiento (UIPC aumenta conforme la capacidad de la LLC aumenta), pero Apache prefiere un diseño Cluster para todas las configuraciones consideradas, mientras que Zeus se beneficia de un diseño Compartido. La figura 5.4b muestra el rendimiento (UIPC) para Zeus. Podemos observar que la diferencia entre los diseños es muy pequeña.

El número de fallos por mil instrucciones (MPKI) en la figura 5.4a muestra un decremento en los MPKI de la LLC con caches más grandes. Si observamos los dos gráficos juntos (figura 5.4) podemos concluir que la reducción en la tasa de fallos compensa la latencia de acceso más alta de una cache más grande. Estos resultados indican que los benchmarks web se benefician de caches de último nivel con más capacidad. De todas formas, Zeus parece que saca provecho de la compartición de datos, y el tamaño de la cache es un parámetro más crítico que la latencia, que es por lo que el diseño Compartido funciona mejor. Para ambos benchmarks, el mejor resultado se obtiene con la cache más grande considerada (1MB por core).



(a) LLC MPKI para Web Zeus.

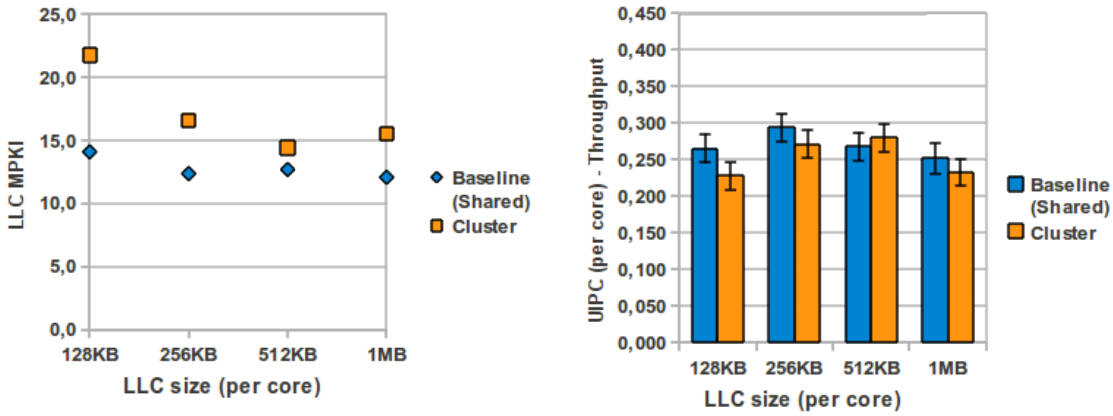
(b) UIPC para Web Zeus.

Figura 5.4: LLC MPKI y UIPC para Web Zeus.

5.2.3 DSS TPC-H: Query 17

Nuestros benchmarks DSS son TPC-H queries 1, 2 y 17 sobre DB2. Estos benchmarks se corresponden con tres categorías: *scan-based* (query 1), *join-dominated* (query 2) y comportamiento mixto o *mixed-behavior* (query 17). En esta sección presentamos los resultados para la query 17 (se muestran en la figura 5.5: la figura 5.5b muestra el rendimiento y la figura 5.5a muestra el número de fallos por 1000 instrucciones para la LLC).

Para esta query una cache compartida es mejor en todos los casos excepto para tamaño 512KB por core. Si observamos el gráfico de rendimiento (UIPC), figura 5.5b, podemos ver cómo nuestro diseño Compartido presenta un incremento en el rendimiento cuando pasamos de una cache de 128KB a una cache de 256KB (por core), pero el rendimiento empeora si continuamos añadiendo capacidad a la cache. Para este diseño, 128KB por core de memoria cache proporcionan mejores



(a) LLC MPKI para DSS Query 17. **(b)** UIPC para DSS Query 17.

Figura 5.5: LLC MPKI y UIPC para DSS Query 17.

resultados que 1MB de cache por core. El diseño Cluster muestra la misma tendencia que el diseño Compartido, pero esta vez el decremento se presenta cuando pasamos de 512KB a 1MB (por core).

Ambos gráficos para UIPC y LLC MPKI sugieren que esta query tiene una localidad baja y, por tanto, caches grandes degradan el rendimiento (latencia de acceso más alta). El hecho de que el diseño Compartido proporcione mejores resultados también sugiere que esta query se beneficia de la compartición de datos entre los procesadores.

El desglose del tiempo de ejecución de usuario en la figura 5.6 (Shared) muestra que el tiempo correspondiente a comunicación fuera de chip está alrededor del 30 %.

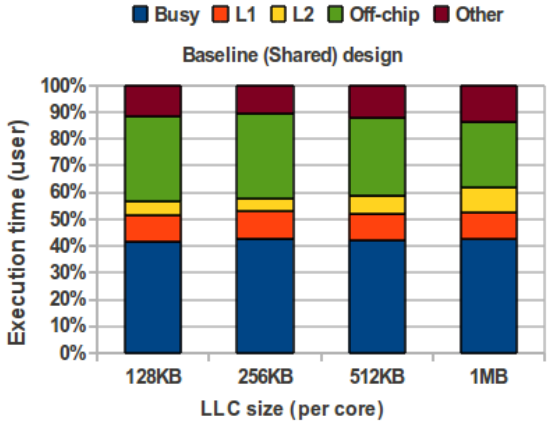


Figura 5.6: Desglose del tiempo de ejecución (usuario) para DSS Query 17 (Baseline).

Para las queries 1 y 2, el diseño Cluster consigue mejores resultados (figura 5.1). En estos casos, incrementar el tamaño de la cache no se traduce en un incremento del rendimiento. En el caso de la query 1, el primer nivel de cache es capaz de capturar prácticamente todos los accesos, por lo que el rendimiento no varía. La query 2 no se beneficia de caches grandes, e incluso el

rendimiento se degrada (ver apéndice C para más detalles).

5.3 Resumen

Hemos visto como diferentes tipos de aplicaciones presentan diferentes características en lo que se refiere a compartición, tamaño y organización del último nivel de memoria cache. En general, una organización privada (Cluster) es preferible para la mayoría de los benchmarks que hemos estudiado y en media.

En el caso de servidores web, éstos presentan un incremento en el rendimiento con caches de último nivel más grandes.

Los benchmarks OLTP tienen unos requerimientos mínimos de LLC. Una vez que esos requerimientos se han satisfecho, el diseño que permita un acceso más rápido (en nuestro caso, el diseño Cluster) tiene un rendimiento mejor.

Los benchmarks DSS tienen distinto comportamiento según la categoría a la que pertenecen: el rendimiento de la query *scan-dominated* (query 1) no cambia al variar el tamaño de la cache, ya que el primer nivel de la jerarquía captura prácticamente todos los accesos; la query *join-dominated* (query 2) se beneficia de un diseño privado (Cluster); finalmente, la query con comportamiento mixto (query 17) tiene una localidad muy baja, lo que se traduce en un rendimiento mayor con caches más pequeñas (al proporcionar éstas menor tiempo de acceso).

Capítulo 6
Conclusiones y trabajo futuro

La jerarquía de memoria es uno de los factores clave en el rendimiento de multiprocesadores, además de ser uno de los componentes que más energía consumen. Por lo tanto, la organización de la jerarquía de memoria y, especialmente, la jerarquía de memoria cache en chip juegan un papel fundamental en el rendimiento de chips multiprocesadores.

En este proyecto hemos analizado dos organizaciones para el último nivel de cache (en nuestro caso L2) del chip de ARM para el proyecto EuroCloud: un diseño Compartido, donde pretendemos sacar el máximo partido de la capacidad efectiva de la cache, y un diseño Cluster, en donde damos prioridad a colocar los datos cerca del procesador que los solicita y conseguir una latencia de acceso baja.

Hemos modelado los dos diseños propuestos en nuestro entorno de simulación *FLEXUS* [33]. Hemos simulado cargas de trabajo comerciales representativas de tres categorías distintas: servidores web, procesamiento de transacciones en línea (OLTP) y sistemas de soporte de decisiones (DSS). De los resultados obtenidos podemos concluir que nuestro diseño Cluster proporciona mejor rendimiento en media; nuestro diseño Compartido tiene un rendimiento mayor si el benchmark presenta baja localidad y/o el tamaño de la cache es pequeño.

En concreto distintos tipos de aplicaciones generan diferentes conclusiones.

- Aplicaciones Web presentan mejor rendimiento con caches de más capacidad.
- Aplicaciones OLTP tienen requerimientos mínimos en lo que se refiere al tamaño del último nivel de memoria cache de la jerarquía. Por encima de ese tamaño, organizaciones privadas y sus latencias más bajas proporcionan mejores resultados.
- Aplicaciones DSS tienen diferentes características dependiendo de la categoría. En general, el diseño Cluster funciona mejor que el diseño Compartido y los requerimientos de capacidad del último nivel de cache son pequeños.

Corroboramos las conclusiones en [15] sobre requerimientos mínimos en workloads de bases de datos, ya que nuestros resultados demuestran que este tipo de cargas de trabajo necesitan un mínimo de cache para obtener un rendimiento adecuado. A partir de un determinado tamaño, continuar incrementando la capacidad de la cache nos lleva a resultados similares o incluso peores debido a que el tiempo de acceso a la cache es mayor, mientras que la tasa de aciertos no aumenta lo suficiente para compensar.

Caches privadas ofrecen otras ventajas sobre diseños compartidos que deben ser considerados. Además de proporcionar latencias más bajas, el diseño en sí es más escalable. Las caches privadas son más independientes (en el sentido de *self-contained*) y es más fácil implementar mecanismos de prioridad o de calidad de servicio [37, 19].

Aunque inicialmente propusimos dos alternativas para caches privadas en Cluster (coherentes y no coherentes), sólo hemos explorado el modelo no coherente. Creemos que los mecanismos hardware para gestionar la coherencia en LLC (L2) en servidores de arquitectura tiled que ejecutan aplicaciones comerciales son innecesarios y deberían evitarse. Además de complicar el diseño hardware, mover bloques en chip debido a mecanismos de coherencia penaliza el tiempo medio de acceso a la cache y suponen un aumento en el consumo de energía.

De todas formas, las caches privadas presentan una desventaja fundamental frente a las caches compartidas, que es el gran numero de peticiones fuera de chip. Como parte del proyecto EuroCloud se pretende estudiar el impacto de la integración de DRAM en 3D dentro del chip. Esta integración 3D permitirá latencias más bajas en el acceso a memoria (ya que se colocan varios cientos de MBs de DRAM cerca de los procesadores). Creemos que una cantidad mínima de memoria cache LLC se debe mantener en chip, ya que la latencia de la DRAM en 3D es muy alta para un segundo nivel de memoria cache. Dejamos para trabajo futuro un estudio más detallado en el tema.

También creemos que estudios futuros deberían incluir datos sobre el consumo de energía para el chip considerado, ya que el ahorro de energía es una de las principales motivaciones para el proyecto EuroCloud. Pensamos que caches privadas (diseño Cluster) presentarán un menor consumo de energía que caches compartidas. Podríamos también, por ejemplo, estudiar la viabilidad de desconectar partes de la memoria como medida de ahorro de energía y el impacto que esto supondría en el rendimiento. Los diseños híbridos parecen una buena forma de obtener las ventajas de caches privadas y caches compartidas, pero deberíamos pensar en este tipo de diseños desde una perspectiva de ahorro de energía y si efectivamente son una opción para este tipo de chips multiprocesadores.

Finalmente, deberíamos considerar que, además del tipo de aplicaciones estudiadas, la computación en nube incluirá otro tipo de aplicaciones como *streaming* de vídeo y audio, reconocimiento de canciones o *data mining*. Cada tipo de aplicación presenta características y comportamiento únicos, así que benchmarks que incluyan este tipo de aplicaciones deberían implementarse e incluirse en el estudio de servidores para computación en nube.

Bibliografía

[1] Alaa R. Alameldeen and David A. Wood. Variability in architectural simulations of multi-threaded workloads. In *HPCA '03: Proceedings of the 9th International Symposium on High-Performance Computer Architecture*, page 7, Washington, DC, USA, 2003. IEEE Computer Society.

[2] ARM. The ARM Cortex-A9 Processors, 2009. `www.arm.com/files/pdf/ARMCortexA-9Processors.pdf` (last access October 2010).

[3] M. Azimi, N. Cherukuri, D.N. Jayasimha, A. Kumar, P. Kundu, S. Park, I. Schoinas, and A. Vaidya. Integration Challenges and Tradeoffs for Tera-scale Architectures. *Intel Technology Journal*, 11(3):173–184, August 2007.

[4] Luiz André Barroso, Kourosh Gharachorloo, and Edouard Bugnion. Memory system characterization of commercial workloads. *SIGARCH Comput. Archit. News*, 26(3):3–14, 1998.

[5] Luiz André Barroso, Kourosh Gharachorloo, Robert McNamara, Andreas Nowatzyk, Shaz Qadeer, Barton Sano, Scott Smith, Robert Stets, and Ben Verghese. Piranha: a scalable architecture based on single-chip multiprocessing. In *ISCA '00: Proceedings of the 27th annual international symposium on Computer architecture*, pages 282–293, New York, NY, USA, 2000. ACM.

[6] Bradford M. Beckmann, Michael R. Marty, and David A. Wood. ASR: Adaptive Selective Replication for CMP Caches. In *MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 443–454, Washington, DC, USA, 2006. IEEE Computer Society.

[7] K.G. Brill. The Invisible Crisis in the Data Center: The Economic Meltdown of Moore's Law. White Paper, 2007. Uptime Institute.

[8] David Chaiken, Craig Fields, Kiyoshi Kurihara, and Anant Agarwal. Directory-Based Cache Coherence in Large-Scale Multiprocessors. *Computer*, 23(6):49–58, 1990.

[9] Jichuan Chang and Gurindar S. Sohi. Cooperative Caching for Chip Multiprocessors. In *ISCA '06: Proceedings of the 33rd annual international symposium on Computer Architecture*, pages 264–276, Washington, DC, USA, 2006. IEEE Computer Society.

[10] Zeshan Chishti, Michael D. Powell, and T. N. Vijaykumar. Optimizing Replication, Communication, and Capacity Allocation in CMPs. In *ISCA '05: Proceedings of the 32nd annual international symposium on Computer Architecture*, pages 357–368, Washington, DC, USA, 2005. IEEE Computer Society.

[11] William Dally and Brian Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[12] John D. Davis, James Laudon, and Kunle Olukotun. Maximizing CMP Throughput with Mediocre Cores. In *PACT '05: Proceedings of the 14th International Conference on Parallel Architectures and Compilation Techniques*, pages 51–62, Washington, DC, USA, 2005. IEEE Computer Society.

[13] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power Provisioning for a Warehouse-sized Computer. In *ISCA '07: Proceedings of the 34th annual international symposium on Computer architecture*, pages 13–23, New York, NY, USA, 2007. ACM.

[14] Richard A. Hankins, Trung Diep, Murali Annavaram, Brian Hirano, Harald Eri, Hubert Nueckel, and John P. Shen. Scaling and characterizing database workloads: Bridging the gap between research and practice. In *MICRO 36: Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, page 151, Washington, DC, USA, 2003. IEEE Computer Society.

[15] Nikolaos Hardavellas. *Chip Multiprocessors for Server Workloads.* PhD thesis, School of Computer Science, Carnegie Mellon University, July 2009.

[16] Nikos Hardavellas, Michael Ferdman, Babak Falsafi, and Anastasia Ailamaki. Reactive NUCA: near-optimal block placement and replication in distributed caches. In *ISCA '09: Proceedings of the 36th annual international symposium on Computer architecture*, pages 184–195, New York, NY, USA, 2009. ACM.

[17] Nikos Hardavellas, Ippokratis Pandis, Ryan Johnson, Naju Mancheril, Anastassia Ailamaki, and Babak Falsafi. Database Servers on Chip Multiprocessors: Limitations and Opportunities. In *Proceedings of the Biennial Conference on Innovative Data Systems Research*, 2007.

[18] Ron Ho, Kenneth W. Mai, Student Member, and Mark A. Horowitz. The future of wires. In *Proceedings of the IEEE*, pages 490–504, 2001.

[19] Ravi Iyer. CQoS: a framework for enabling QoS in shared caches of CMP platforms. In *ICS '04: Proceedings of the 18th annual international conference on Supercomputing*, pages 257–266, New York, NY, USA, 2004. ACM.

[20] Taeho Kgil, Shaun D'Souza, Ali Saidi, Nathan Binkert, Ronald Dreslinski, Trevor Mudge, Steven Reinhardt, and Krisztian Flautner. PicoServer: using 3D stacking technology to enable a compact energy efficient chip multiprocessor. *SIGARCH Comput. Archit. News*, 34(5):117–128, 2006.

[21] Changkyu Kim, Doug Burger, and Stephen W. Keckler. An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches. In *ASPLOS-X: Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, pages 211–222, New York, NY, USA, 2002. ACM.

[22] Poonacha Kongetira, Kathirgamar Aingaran, and Kunle Olukotun. Niagara: A 32-way multithreaded sparc processor. *IEEE Micro*, 25(2):21–29, 2005.

[23] Gabriel H. Loh. 3D-Stacked Memory Architectures for Multi-core Processors. In *ISCA '08: Proceedings of the 35th Annual International Symposium on Computer Architecture*, pages 453–464, Washington, DC, USA, 2008. IEEE Computer Society.

[24] 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. *Computer*, 35:50–58, 2002.

[25] Naveen Muralimanohar, Rajeev Balasubramonian, and Norm Jouppi. Optimizing NUCA Organizations and Wiring Alternatives for Large Caches with CACTI 6.0. In *MICRO 40: Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 3–14, Washington, DC, USA, 2007. IEEE Computer Society.

[26] Emre Ozer, Krisztian Flautner, Sachin Idgunji, Ali Saidi, Yiannakis Sazeides, Bushra Ahsan, Nikolas Ladas, Chrysostomos Nicopoulos, Isidoros Sideris, Babak Falsafi, Almutaz Adileh, Michael Ferdman, Pejman Lotfi-Kamran, Mika Kuulusa, Pol Marchal, and Nikolas Minas. EuroCloud: Energy-conscious 3D Server-on-Chip for Green Cloud Services. In *2nd Workshop on Architectural Concerns in Large Datacenters in conjunction with ISCA-2010*, June 2010.

[27] Brian M. Rogers, Anil Krishna, Gordon B. Bell, Ken Vu, Xiaowei Jiang, and Yan Solihin. Scaling the bandwidth wall: challenges in and avenues for CMP scaling. In *ISCA '09: Proceedings of the 36th annual international symposium on Computer architecture*, pages 371–382, New York, NY, USA, 2009. ACM.

[28] S. Rusu, S. Tam, H. Muljono, D. Ayers, and J. Chang. A Dual-Core Multi-Threaded Xeon Processor with 16MB L3 Cache. In *IEEE International Solid-State Circuits Conference Digest of Technical Papers*, pages 315 –324, feb. 2006.

[29] Standard Performance Evaluation Corporation (SPEC). Specweb99 benchmark, 2000. `http://www.spec.org/web99/` (last access October 2010).

[30] Per Stenstrom. A Survey of Cache Coherence Schemes for Multiprocessors. *Computer*, 23(6):12–24, 1990.

[31] Transaction Processing Performance Council (TPC). TPC Benchmark C Standard Specification, 2010. `http://www.tpc.org/tpcc/` (last access October 2010).

[32] Transaction Processing Performance Council (TPC). TPC Benchmark H (Decision Support) Standard Specification, 2010. `http://www.tpc.org/tpch/` (last access October 2010).

[33] Thomas F. Wenisch, Roland E. Wunderlich, Michael Ferdman, Anastassia Ailamaki, Babak Falsafi, and James C. Hoe. SimFlex: Statistical Sampling of Computer System Simulation. *IEEE Micro*, 26(4):18–31, 2006.

[34] Steven J. E. Wilton and Norman P. Jouppi. CACTI: An Enhanced Cache Access and Cycle Time Model. *IEEE Journal of Solid-State Circuits*, 31:677–688, 1996.

[35] Roland E. Wunderlich, Thomas F. Wenisch, Babak Falsafi, and James C. Hoe. SMARTS: accelerating microarchitecture simulation via rigorous statistical sampling. *SIGARCH Comput. Archit. News*, 31(2):84–97, 2003.

[36] J. Wuu, D. Weiss, C. Morganti, and M. Dreesen. The Asynchronous 24MB on-chip Level-3 cache for a dual-core Itanium®-Family Processor. In *Solid-State Circuits Conference, 2005. Digest of Technical Papers. ISSCC. 2005 IEEE International*, pages 488 –612 Vol. 1, feb. 2005.

[37] Michael Zhang and Krste Asanovic. Victim Replication: Maximizing Capacity while Hiding Wire Delay in Tiled Chip Multiprocessors. In *ISCA '05: Proceedings of the 32nd annual international symposium on Computer Architecture*, pages 336–345, Washington, DC, USA, 2005. IEEE Computer Society.