TESIS DE LA UNIVERSIDAD DE ZARAGOZA

2011

40

Darío Suárez Gracia

# A Tiled Cache Organization

Departamento

Informática e Ingeniería de Sistemas

Director/es

Monreal Arnal, Teresa

Viñals Yúfera, Víctor



ISSN 2254-7606

Prensas de la Universidad Universidad Zaragoza

Tesis Doctoral

# A TILED CACHE ORGANIZATION

Autor

Darío Suárez Gracia

Director/es

Monreal Arnal, Teresa
Viñals Yúfera, Víctor

**UNIVERSIDAD DE ZARAGOZA**

Informática e Ingeniería de Sistemas

2011

Universidad Zaragoza
1542

# A TILED CACHE ORGANIZATION

*Author:*
Darío SUÁREZ GRACIA

*Supervisors:*
Dr. Teresa MONREAL ARNAL
Dr. Víctor VIÑALS YÚFERA

DISSERTATION
Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Universidad de Zaragoza

Grupo de Arquitectura de Computadores
Dpto. de Informática e Ingeniería de Sistemas
Instituto de Investigación en Ingeniería de Aragón
Universidad de Zaragoza

September, 2011

# Acknowledgements

Large works are seldom the result of the effort of a single person and this one is not an exception. Thanks to each and every one of the people who have supported me during this work. Undoubtedly, without your help Sisyphus would have become a close friend.

First, I would like to acknowledge my advisors, Víctor and Teresa, for their teachings and support. Also, the rest of people at the Computer Architecture Group (gaZ) of the University of Zaragoza have been helpful and willingness all the time with an special mention to Ana, Chus, Jorge, María, Quique, Pablo, and Rubén. Also, José María Llabería and Ramón Beivide have given me good advice and right solutions to tough questions.

Second, I'm also indebted with all the professors who have welcomed me in their labs: Andreas Moshovos at the University of Toronto, Josep Torrellas at the University of Illinois at Urbana-Champaign, and Manolis G.H. Katevenis at FORTH—Hellas. Thanks also to the people who made me fell at home abroad, especially María Jesús Garzarán, Brian Greskamp, Shelley Chen and Pablo Montesinos Ortego, Eleni Mpitsaki, and Giorgos Passas.

Last but not least, I want to acknowledge to the DIISasters, my runner fellows at Zaragoza, and the Baltasar Gracián crew with whom I have spent so many pleasant moments. Also, to Jorge Silvestre, who have always been ready to take me out to the mountains, to my parents, because their support and encouragement has pushed me further away than any wind, and to Ana Cris whose very smile turns problems into petty trifles.

# Project Framework

This thesis has been developed within the Computer Architecture Group at the University of Zaragoza inside the framework of the projects: Computación de altas prestaciones IV. Jerarquía de memoria de altas prestaciones (TIN2004-07739-C02-02), Jerarquías de memoria de alto rendimiento (TIN2007-66423) and scholarships from the Gobierno de Aragón and the Spanish Ministry of Education and Science (FPI ref. BES-2005-10385).


Part of this work is the result of the several visits to other research groups: the AENAO group lead by Prof. Andreas Moshovos at the University of Toronto in Canada (February–August 2007), the IaCOMA group lead by Prof. Josep Torrellas at the University of Illinois at Urbana Champaign in USA (March–July 2008), and the CARV Laboratory lead by Prof. Manolis G. H. Katevenis at the Institute of Computer Science, FORTH in Greece (July–October 2009). These research stays have been also supported by the FPI Scholarship and the European Network of Excellence HiPEAC.

# Executive Summary

The gap between logic and DRAM speed has wided with technology scaling. As a result, current processors include a complex memory hierarchy to minimize the cost of accesses to main memory. A few years ago, only top-end microprocessors include such hierarchies, but now smartphones, ultra-portable computers, and related devices demand powerful memory hierarchies with much lower energy consumption. The most common organization comprises several on-chip cache levels. The Last Level Cache (LLC) is optimized for density (size) and the first one for latency. As larger LLCs are incorporated we can note a growing latency gap between them, what we call the on-chip inter-level latency gap. This work assumes this fact, recognizes it as a potential problem and proposes a new cache organization able to deal with the inter-level gap.

This dissertation proposes Light NUCA (Non-Uniform Cache Architecture), a tiled cache organization made of small caches connected with very specialized Networks-in-Cache. L-NUCA improves performance and reduces energy at the same time by capturing temporal locality at a finer granularity than other cache organizations, and solves the inter-cache latency gap. Besides, we have proved with a layout in 90 nm that its regular organization has potential to easy verification and reduce time-to-market.

To ensure the accuracy of our results, we have build a simulation platform, which models not only the processors but also the interconnection networks with routing, control flow, and back-pressure. On the platform, we run representative and up to date benchmarks. Our results shown that the proposed organization shows benefits in several environments, namely, high-performance uniprocessors, high-performance low-power embedded, and simultaneous multithreading.

For the high-performance low-power embedded domain, we have extended Light NUCA with several proactive and reactive techniques to reduce dynamic energy consumption without impact on performance. These techniques leverage from the Networks-in-Cache, and are very easy to implement. This Light Power NUCA adapts the cache latency very well to the variations in working set of programs because it tracks temporal locality at a very fine granularity. Nevertheless, its power consumption is indepented of the cache hit rate. To provide a more adaptive behaviour, we have proposed a learning based controller detecting when the cache is not providing blocks and drops them. To further reduce energy, the same mechanism changes the cache access for parallel to serial in the tag and

data arrays. The cost of this novelty is almost negligible because it also leverages from the Networks-in-Cache congestion mechanisms.

Since the simulation of simultaneous multithreading workloads is very costly in terms of time, we have proposed a statistical-based mechanism to select representative combinations of benchmarks. With the generated workloads, we have shown that Light NUCA with minimum changes overpasses other cache organizations even if simultaneous multithreading helps to tolerate memory latency.

As a summary, this Ph.D. thesis proposes a cache organization for the first levels of the hierarchy, providing area, energy, and performance advantage over other cache alternatives. Most important, we have verified from layout that the proposals are easy to implement.

# Resumen Ejecutivo

La reducción en la escala de integración se ha traducido de modo desigual entre los procesadores y los chips de memoria DRAM. La velocidad de los primeros ha aumentando considerablemente más rápido causando un grave problema que se conoce por el "Memory gap". La manera más habitual de paliar este problema ha sido la inclusión de complejas jerarquías de memoria multinivel que intentan que el procesador no se detenga por no tener datos e instrucciones disponibles. Hace unos pocos años, estas jerarquías eran exclusivas de los procesadores de alta gama, sin embargo, actualmente, los teléfonos inteligentes, ordenadores ultraportátiles, y dispositivos semejantes también requieren jerarquías muy potentes pero que a la vez tengan un consumo muy bajo de energía. La mayor parte de las jerarquías de memoria están organizadas en varios niveles de memoria cache. El último nivel, Last Level Cache (LLC), esta optimizado para maximizar su densidad (tamaño) mientras que los primeros niveles intentan reducir al máximo la latencia. Conforme las LLC se agrandan, se vuelven más lentas creando un nuevo salto dentro del chip.

Esta tesis propone una nueva organización de cache basada en pequeñas teselas de memoria conectadas mediante redes muy especialadas a las que denominamos *Networks-in-Cache*. La nueva organización mejora el rendimiento a la par que reduce el consumo energético porque es capaz de capturar la localidad temporal a una granularidad más fina que otras organizaciones lo que le permite paliar sino resolver el problema del santo entre caches dentro del chip. Además, se ha verificado con un *layout* en 90 nm que su estructura regular tiene mucho potencial tanto para reducir el tiempo de verificación del diseño como el del lanzamiento del producto.

Para asegurar la precisión de nuestros resultados ha sido construida una infraestructura de simulación que modela no sólo el procesador sino también las redes de interconexión incluyendo enrutamiento, control de flujo y congestión. Sobre la plataforma ejecutamos programas de prueba representativos y actuales como SPEC CPU2006. Los resultados muestras que la organización propuesta, denominada Light NUCA [1], ofrece ventajas en diversos segmentos del mercado como los uniprocesadores de altas prestaciones, procesadores para dispositivos ultraportátiles y multihilo, en concreto *Simultaneous Multithreading*.

[1] Non Uniform Cache Architecture.

En el segmento de los procesadores ultraportátiles que requiren un muy bajo consumo, hemos extendido Light NUCA con varias técnicas tanto proactivas como reactivas para reducir el consumo de energía dinámica sin disminuir el rendimiento. Estas técnicas se aprovechan de las *Networks-in-Cache* y se ha verificado que son fácilmente implementables. Esta cache, Light Power NUCA, es capaz de adaptar su tamaño, y latencia, a las variaciones en el *working set*[2] gracias a su capacidad de captura de localidad temporal. Sin embargo, su consumo energético es independiente de la tasa de aciertos que alcance. Para permitir un comportamiento más adaptativo, proponemos un controlador basado en aprendizaje que detecta cuando la Light Power NUCA no esta sirviendo bloques al procesador y los descarta antes de que incrementen el consumo energético. Para reducirlo todavía más, el mismo mecanismo cambia el modo de acceso a los *arrays* de datos y direcciones de paralelo a serie. El coste de esta mejora es muy pequeño porque también aprovecha los mecanismos existentes de gestión de la congestión de las *Networks-in-Cache*.

Debido a que la simulación de cargas de trabajo multihilo es muy costosa en tiempo, esta tesis propone un método basado en muestreo estadístico para elegir combinaciones representativas de programas. Con estas cargas hemos verificado que Light NUCA con pequeñas mejoras ofrece ventajas en términos de rendimiento incluso teniendo en cuenta que los procesadores multihilo ayudan a tolerar altas latencias de memoria.

En resumen, esta tesis propone una organización de cache para los primeros niveles de la jerarquía de memoria que permiten aumentar el rendimiento y reducir el area y el consumo energético respecto otras organizaciones. Igualmente importante, se ha visto como nuestras propuestas son viables mediante la implementación VLSI realizada.

[2]Conjunto de posiciones de memoria accedidas en un intervalo de tiempo acotado.

# Contents

# Chapter 1

# Introduction


### Summary

*This chapter analyzes some of the requirements of today computers with a brief historical evolution. Then, it introduces the memory hierarchy as the field of study of this dissertation and comments on two recents proposals in which this work is based: Networks-on-Chip and Non-Uniform Memory Architectures. The last part of the chapter describes the rest of the document and summarizes the contributions of this thesis.*

By a slave memory I mean one which automatically accumulates to itself words that come from a slower main memory, and keeps them available for subsequent use without it being necessary for the penalty of main memory access to be incurred again.

Maurice V. Wilkes [143]

## 1.1 Rationale

**Figure 1.1:** M. Wilkes with the EDSAC © Computer History Museum

Computers execute programs consisting of instructions that in turn code operations and their respective data. Both, instructions and data are read and written from memories. When the speed of the memories is slower than that of the processors, system performance reduces, and most time the processor stalls waiting for instructions or data from memory. This problem has many names: von Neumann bottleneck, memory-bottleneck, memory-wall, memory-processor gap, ... , has been heavily studied [7, 121, 145, 144, 52], and unless a novel computation approach or a disruptive technology appear, it will continue being an infinite field of study in computer architecture.

Maurice Wilkes first observed that an small yet fast memory located in between the processor and the main memory could reduce the processor stall time. He coined them as slave memories [143]. A few years later, IBM introduced a cache in a commercial design, the System 360/85 [84]. Nowadays, in order to satisfy society performance requirements, caches are ubiquitous in mostly every computing device regardless its application domain; from high-end petascale computers such as the IBM Power 7 [71], including 3 cache levels of 32 KB, 256 KB, and 4 MB per core, to budged digital cameras such as the Canon PowerShot A470, including an ARM946-S processor with instruction and data caches of 8KB[2].

Current societies claim for more capable devices but also for a sustainable and environmentally friendly computing. Reducing activity is one of the most straightforward paths to save energy because it is directly proportional to the amount of capacity charged and discharged by transistors. The energy cost of accessing off-chip main memory DRAM chips requires activity in multiple components; namely, the processor, the memory chips, and the interconnections between them. Therefore, increase the on-chip cache hit ratio saves energy because it lowers the number of off-chip accesses [59]. To cut off even more energy consumption, there are plenty of proposals, specially in the embedded domain, that specifically reduce the cache energy to reduce even more the total system energy [78, 73, 1, 60, 74, 33].

Performance and energy are deeply rooted because given a technology a faster operation requires more energy. Nevertheless, all changes among integration technologies have taken place to reduce energy consumption at the cost of lowering performance [9], but at this time there is not a clear substitute for CMOS, and the "power-wall" appears as the main thread for keeping the performance rising pace of the last decades [70]. The power-wall has two faces. On one hand,

the social face of being sustainable. On the other hand, the engineering face of power density because microprocessors now dissipate around 100 W/cm$^2$—as a comparison, induction cooking appliances have a power density 3 times lower—and higher values entail complex and expensive cooling system only affordable for supercomputers.

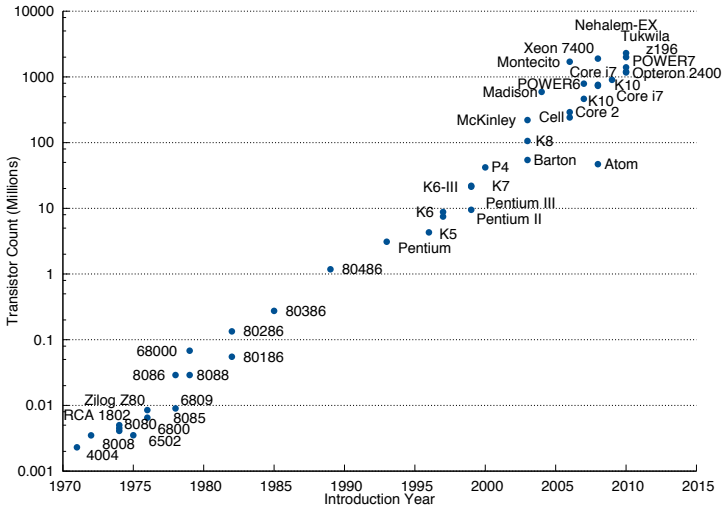



**Figure 1.2:** Microprocessor number of transistors over time. Data from Wikipedia

So far we have seen two requirements for future caches, performance and energy, but the rules dictating the economics of processors set their own requirements as well. In general terms, two well-known papers have driven the CMOS microprocessor industry for the last forty years. One established that the number of transistors would double first every year and then every two years, the Moore's Law [97], which result can be seen in Figure 1.2. Since 1974 up to now the number of transistors inside microprocessor has been rising exponentially, and only entry level processors, such as the Intel Atom, remain outside the trend. The other paper by Dennard, Gaensslen, Rideout, Bassous, and LeBlanc sets the scaling principles to simultaneously make transistors occupy less area, switch faster, and use less energy [28]. Ideally, following Dennard's scaling power density remains constant across generations as shown in Table 1.1; however, deep submicron technologies have forced to increase power density to maintain the delay improvements. Voltage cannot continuously decreases because transistors stop switching. In fact, nowadays high-performance technologies set $V_{dd}$ to 1 V, and by 2024 the International Technology Roadmap for Semiconductors estimates that $V_{dd}$ will only lower to 0.6 V [4].

Processors have profit from Dennard's scaling, but architects have also taken advantage of Moore's law for introducing multiple on-chip caches and many other features such as out-of-order execution, branch and memory dependences prediction, pipelining, superscalar execution, prefetching, ... to fuel performance

**Table 1.1:** Scaling results for circuit performance (From Dennard *et al.* [28])

| Device or Circuit Parameter | Scaling Factor |
|---|---|
| Device dimension $t_{ox}$, $L$, $W$ [a] | $1/\kappa$ [b] |
| Doping concentration $N_a$ | $\kappa$ |
| Voltage $V$ | $1/\kappa$ |
| Current $I$ | $1/\kappa$ |
| Capacitance $\epsilon A/t$ [c] | $1/\kappa$ |
| Delay time/circuit $VC/I$ | $1/\kappa$ |
| Power dissipation/circuit $VI$ | $1/\kappa^2$ |
| Power density $VI/A$ | 1 |

[a] $t_{ox}$, $L$, and $W$ stands for gate insulator thickness, length, an width of the device, respectively
[b] Industry have normally scalled with $\kappa = \sqrt{2}$
[c] $\epsilon$, $A$, and $t$ stands for permitivity, area, and thickness, respectively

(mostly by extracting instruction level parallelism) [17]. Nevertheless, current deep-submicron technologies, (90nm and below) introduce additional problems such as process variation increasing development costs and harding the inclusion of novel features. To prove the correctness of circuits becomes terribly complex, and more modular designs could ease this task [18]. Since often manufacturers sell the same processor with different cache sizes, more modular cache architectures could reduce their cost and time-to-market.

We have hitherto seen three important requirements for caches: performance, energy consumption, and modularity, but what really forces a continuous reevaluation of memory hierarchies and caches in particular are applications. Users and industry relentless feed this loop with new requirements and possibilities. Last generation mobile handsets rival in performance with low-end general purpose processors, and the barriers between these segments are melting. Therefore, it is preferable a cache organization suitable for multiple processing domains.

The fact that power does not scale well together with the difficulties for wringing out more instruction level parallelism (ILP) have forced the industry to trade ILP extraction features for the ability of simultaneously executing multiple programs [1] either with chip-multiprocessors, or simultaneous multithreading, or a combination of both techniques. For example, the Sun UltraSPARC T2 chip includes 8 processors each one executing 8 threads concurrently [103], and the IBM POWER7 also has 8 cores that simultaneously execute 4 threads [71]. Both approaches demand more bandwidth in the hierarchy, tend to require larger caches, and in the case of private caches require coherence management mechanism.

Previous paragraphs have described some desirable design goals for caches; namely, high-performance, low-energy consumption, modularity, multithread-

[1] Hereafter thread and program may be used interchangeably.

ready. Next, we continue to describe in more detailed how caches are integrated into the memory hierarchy and introduce Networks-on-Chip and Non-Uniform Cache Architectures since this dissertation is mostly based on them.

## 1.2 Memory Hierarchy Organization

Most modern instruction set architectures define instructions either to compute or to load and store data from memory. The former group reads and writes its operands from/to the register file, and the latter loads values from memory into the registers and stores their content back to the memory. This two storages, the register file and the main memory, are the starting and ending levels of memory hierarchies [2] and caches are in between.

Figure 1.4 represents a simplified view of the memory hierarchy present in any modern processor including separate instruction and data first level caches. The first computer including split caches was the IBM Automatic Sequence Controlled Calculator (ASCC)–Harvard Mark I. This machine coined the Harvard architecture term for processors with separated caches.

**Figure 1.3:** Harvard Mark I © IBM

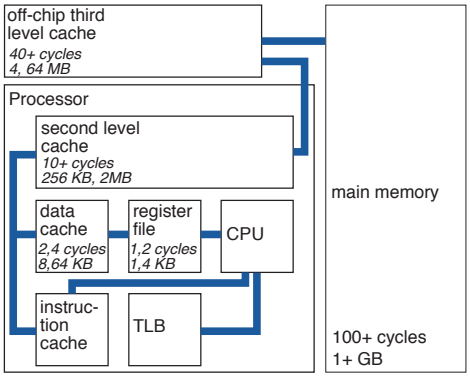



**Figure 1.4:** Memory Hierarchy in Uniprocessors. CPU and TLB stands for Central Processing Unit and Translation Lookaside Buffer, respectively. A reasonable access time and size is represented inside each box with italic

Oversimplifying, when a processor is turned on, the program initial address is written into the program counter (PC) register. Just after, the PC virtual address is translated into a physical address by the Translation Lookaside Buffer, which caches most referenced memory pages [3]. The translated physical address of the instruction will be requested to the first level instruction cache that services the CPU when it contains the requested instructions. When not, it will repeat the request to the next cache level, in our case an on-die second level cache (L2) .

[2] Secondary storage in hard drives or network is not considered in this work.

[3] For a complete description of Virtual Memory and its interaction with the Operating System please refer to part III of the classical text by Silberschatz, Galvin and Gagne [119] or to the Appendix C.4 of the no less classic book by Hennessy and Patterson [52].

Typically, it takes a tens of cycles to access an L2 cache and its storage ranges between 256KB and 2MB. When the L2 is the last on-chip cache level its size is larger. If the request misses again in the L2, the off-chip third level cache (L3) will look-up for it, and if in turn it also misses, the request reaches the main memory. The cost will be hundred of cycles, and the requested block will be written into the L3, L2, and L1 instruction caches depending on their allocation policy.

As soon as the instruction arrives to the CPU, the fetch stage completes and decoding starts. Several instruction bits code the logical register entry of the source and destination operands. The register file is accessed with these indexes and returns the data in 1 or 2 cycles. The drawback of such speed is the size limitation because register files can only store between 1 and 4 KBytes. Load and Store instructions interchange data between the register file and main memory. As in the case of instructions, there are several caches trying to shorten the memory access time. In our particular case, L1 data cache misses are served in the same way than L1 instruction ones, either by the L2, L3, or in the worst case by the main memory.

Above explanations also apply to multiprocessor systems, but in this case caches can be either private, they service only one processor, or shared, they service multiple processors. Normally all levels but the last are private, and the last is shared. Cache sharing is an open study area remaining outside the scope of this dissertation [13, 61, 149, 19, 51, 85].

Cache aims to present the main memory as an infinite and fast storage to the processor and success when programs exhibit locality [29, 30].

## 1.3 Networks-on-Chip

The miniaturization of all kind of devices forced the rethinking of intercommunications among them. With current technologies, it is possible to embed components such as GPS, accelerators, graphics cards, microprocessors, ... in a single chip. Communication mechanism followed the same path than components and the Networks-on-Chip appeared [26, 14, 34].

Networks-on-Chip have different requirements and substrate than off chip networks, especially in buffers and wires. While the cost of large buffers was not an issue in off-chip networks, Networks-on-Chip have to minimize them in order to reduce area and improve yield. On the contrary, wires were rare in off-chip environments, but with more than 10 metal layers in current technologies, they are affordable in Networks-on-Chip.

This work follows the Networks-on-Chip paradigm, but takes the basics and gives them a new twist because communications are done inside a component, Networks-in-Cache, and not among them as Networks-on-Chips normally do. Section 3.2.2 provides more details in the context of NoCs for Non-Uniform Cache Architectures.

## 1.4 Non-Uniform Cache Architectures

The first cache organization exploiting the Networks-on-Chip paradigm was the Non-Uniform Cache Architectures (NUCA) [77]. NUCA exposed to the microarchitecture the physical organization of the cache, so the latency of cache banks depends on their location in the chip. The objective was to reduce the wire delay problem [58]. In large Last Level Caches (LLC), the latency cost of driving a signal from and to the banks may be larger than that of the bank itself. The NUCA ideas have generated much attention in Academia and as a result there have been multitude of proposals based on them. Most of these works are commented in Section 3.2.1.

NUCA caches exploit the Networks-on-Chip paradigm mostly stitching cache banks with well-known router designs without exploiting the possible synergies from a more glued design as this thesis does. The coupling includes the first level cache and leaves apart the LLC because close to the execution unit the latency and bandwidth requirements are much larger so NoC advantages can be fully exploited. LLCs tend to be more optimized for capacity [110]. In summary, this work proposes, implements, and evaluates a tiled cache organization built over three specialized Networks-in-Cache. The idea is to keep a large set of L1 data cache evicted blocks accessible at very low latency and with a small amount of energy independently of the number of threads in execution. Throughout the following chapters, we will see how introducing a modular L-NUCA either to a conventional or NUCA hierarchy gives benefits in terms of performance, energy consumption, and area.

## 1.5 Dissertation Structure

The rest of this document is as follows. Chapter 2 describes our evaluation environment including our simulator, workloads, and methodology. Chapter 3 introduces our tile cache proposal: the Light NUCA, and evaluates its performance and energy consumption in a high performance uniprocessor system. Chapter 4 depicts the placed & routed design of a Light NUCA tile and then based on the extracted power consumption proposes and evaluates several mechanisms to reduce dynamic power for high-performance low-power embedded processors, the LP-NUCA. Chapter 5 describes a learning-based approach to adapt the dynamic energy consumption of the Light Power NUCA to phases of programs. Chapter 6 analyzes the requirements set by the simultaneous execution of multiple threads in first level caches and proves how Light NUCA meets them better that other cache alternatives. Chapter 7 qualitatively analyzes the impact of supporting coherency and real time by Light-NUCA. Finally, Chapter 8 concludes and point out several possible future lines.

## 1.6 Contributions

To sum-up, the contributions of this Ph.D. thesis are as follows:

- We have observed an *on-chip latency gap* threatening the microprocessor performance. As integration scale shrinks, the latency of the first level caches remain constant while those from last level caches rises creating this new gap.
- As a possible solution for the on-chip latency gap we proposed an *alternative cache organization* for the first levels of the cache hierarchy based on the Networks-on-Chip and Non-Uniform Cache Architectures paradigms.
- The key aspect from the proposal is their *Networks-in-Cache* specifically designed to convey cache traffic.
- As energy consumption is another limiting performance factor, our approach is extended with *static and dynamic techniques to save energy* leveraging the Networks-in-Cache.
- To prove the viability of the design, including its modularity, this work includes a fully placed & routed *VLSI implementation*.
- Design has been *tested in two processor domains*: high performance general purpose and low-power high-end embedded. In both cases, the evaluation comprises single threaded and Simultaneous Multithreading processors.
- In order to ensure the accuracy of our results, this work proposes a *statistical based methodology* for simulating multiprogrammed workloads as well.
- Setup of an *experimental framework* including state-of-the art benchmarks that has been successfully released for institutions that we collaborate with.

# Chapter 2

# Experimental Framework


### Summary

*This Chapter describes the simulation infrastructure built for carrying our experiments. It includes a discusion on microarchitecture of the reference processor and the advanced techniques included to extract more instruction level parallelism. Topics related to the memory hierarchy such us the interaction between the execution core and miss status holding registers are covered in detail. Then, we analyze the Simultaneous Multithreading extension and conclude with the description of our workloads; namely, SPEC CPU2000 and CPU2006.*

## 2.1 The SMTScalar Simulator

SMTScalar [1] is an execution-based simulator based on SimpleScalar 3.0d for Alpha ISA [5]. Aimed to accurately model data cache hierarchies, it supports conventional, NUCA, and Light NUCA caches executed single thread and multiprogrammed workloads. The starting line for the microarchitectural model was the Alpha 21264 [75], but also with significant influences from the IBM Power4 and Power5 [137, 130, 72]. The change from the Register Update Unit SimpleScalar based model to a Reorder Buffer was accomplished with code by Enrique Torres Moreno and Jesús Alastruey Benedé from the Universidad de Zaragoza who provide much support during the first stages of development.

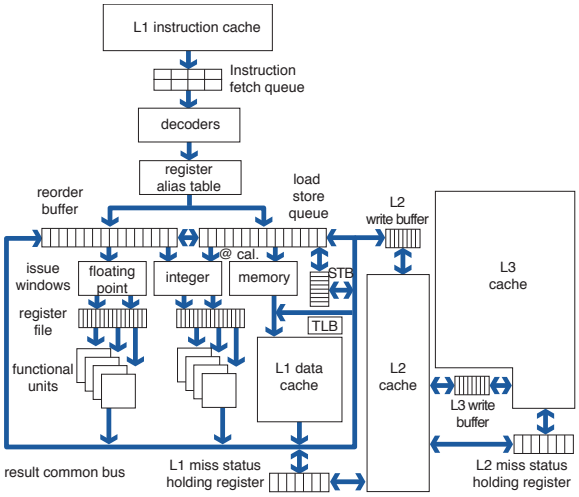



**Figure 2.1:** Organization overview of the baseline simulated processor with a conventional three level cache hierarchy

Figure 2.1 shows the main blocks of the simulated processor. The upper part represents the fetch unit. In this work, we assume a perfect instruction cache because our focus is data caches. Instructions stay in the instruction fetch queue until being decoded, then they are renamed and stored in the reorder buffer and in the load store queue if necessary. Instruction issue from three different windows, one for floating point, another for integer including address computation instructions, and the last one for loads and stores. Note that with this organization, memory instructions have not separated ALUs for address calculation, and the outputs of the integer functional units also feeds the data cache. The decoupling of address calculation and cache access allows to compute more addresses in parallel reducing the stalls due to memory dependencies miss-predictions.

Apart from the perfect fetch, we configure SMTScalar with infinite number of physical registers to minimize all processor stalls but cache ones. Other sources

[1] SMTScalar was named by Jesús Alastruey Benedé.

**Table 2.1:** Speculative issue and recovery implementation choices

| | | Issue | | |
|---|---|---|---|---|
| | | fixed | | variable |
| | | always[a] | never[b] | |
| Recovery | full | — | no speculation | 21264[c] |
| | selective | Pentium 4 | — | — |

[a] Assumes always hit, 100% hit rate
[b] Assumes always miss, 100% miss rate
[c] Prediction based

of stalling in heavily pipelined out-of-order cores are memory dependencies and load latency miss-prediction.

*memory disambiguation*

This simulator supports a perfect memory disambiguation scheme and a conservative policy in which loads are issued unless there is a prior store to an unknown address or to the same address that the load without being completed. From our experiments we have not seen a large difference in performance between both approaches, and SMTScalar sets the conservative policy as default. The load–store disambiguation code has been adapted from an implementation by Andreas Moshovos from the University of Toronto.

*load hit prediction*

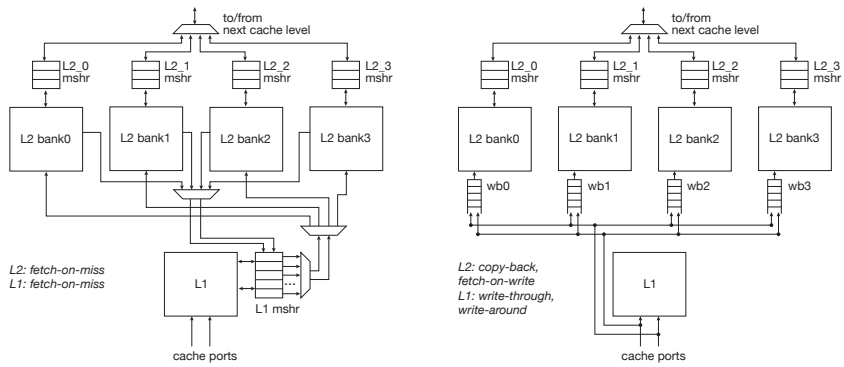
In deep pipelined superscalar processors, the delay between a load instruction leaves the issue window until the data becomes available takes several cycles. If load dependent instructions are not speculatively issued, the processor looses the ability of back-to-back execution, called load latency miss-prediction. Besides, the cache outcome is unknown, so when dependent instructions are speculatively issued—assuming cache hit—they may need to be recovered and then re-executed. Therefore, we have two related policies regarding load latency miss-prediction whether the issue window speculatively launches load dependant instructions, and if so whether the recovery drains all the pipeline or only dependant instructions are drained. If the processor frequency is very high, the delay from the miss-prediction detection to the issue window can be several cycles. By default, SMTscalar assumes that the issue window is notified 2-cycles after the load arrives to the write-back stage, but this value can be changed by the user. During the setup of the simulation, we carried out several experiments tuning this value, and with selective recovery it does not affect too much performance.

Table 2.1 shows the possible alternatives for issue and recovery and includes the policies of the 21264 and the Pentium 4 [57]. The options are as follows: a fix issue policy, either always or never, or a variable policy in which dependant instructions issue based on a prediction. The never issue policy does not require recovery at the cost of a larger cache latency. On the contrary, always and variable may require recovery that can be either full, in which all younger instructions are drained from the pipeline, or selective, in which only dependant instructions are drained. SMTScalar implements all combinations. By default, it emulates the

Pentium 4 with always issue and selective recovery.

### 2.1.1 Cache Hierarchy

This simulation infrastructure supports multiple data cache hierarchies configurations: conventional multibanked with 2 or 3 levels, L1 plus dynamic NUCA with or without an L3, and Light NUCA backed by a conventional or D-NUCA [77].




**(a)** Loads **(b)** Stores

**Figure 2.2:** Conventional L1 and 4-bank L2 cache organizations

As shown in Figures 2.1 and 2.2, the modeling includes cycle accurate models for Miss Status Holding Registers (MSHR) and Write Buffers [79, 69, 120]. For the sake of briefness, we are going to detail the cache behaviour with a conventional L1 backed by a 4-bank L2. The L3 is not depicted because its organization and policies have the same parameters and choices than the L2. Differences in other cache organizations are explained when necessary in the rest of this document.

Starting with loads, Figure 2.2(a), SMTScalar models high-bandwidth L1 caches, either with physical banks, true multiporting, or with multiple banks. This work assumes true multiported desings, with 1 and 2 ports for 2 and 4 issue width processors. When a load misses in the L1 cache, the replacement way is marked as invalid, unless the miss is secondary[2], and goes to the MSHR; there, the insertion of the miss can be successful or not. On the former case, the request will be sent to the L2 cache for primary misses or will wait until the block becomes ready for secondary ones. Right after a block arrives from the l2, the MSHR notifies the memory issue window that the block is ready so the primary miss and the secondaries if any can issue. These refills have priority over load and stores during cache port arbitration. On the latter case, failed insertion, there are more causes. For primary misses: MSHR full and no replacement way, and for secondary misses: MSHR secondary full and entry already served for request that are in transit of being refilled.

[2]Secondary misses are those address have been already requested by the Miss Status Holding Register.

When a miss can not be inserted into the Miss Status Holding Register, it is reexecuted after a configurable number of cycles. Normally, we set this value to 7 similarly to the IBM Power4 [130].

Previous paragraphs assumed that the cache fetches, or allocates, the requested block on misses, but this option is configurable individually for loads and stores. In general, we assume a write-through, write-no-allocate L1 and copy-back, write-allocate for the L2. Write buffers (WB) coalesce entries by default and track whether each individual byte has been written during coalescing. They do not service data, so when the address of a load request matches with the address of a WB entry, the cache controller inverts access priorities until the entry has been written into the cache, so the load request can be serviced.

As this dissertation targets the design of first cache levels the memory model is less accurate, and we do not model the DRAM timing in detail. Nevertheless, this choice is not at the cost of results representativity because the miss rates of the Last Level Caches with SPEC CPU2000 and CPU2006 in general are small.

#### Non-Uniform Cache Architectures

SMTScalar accurately simulates dynamic and static NUCA caches with some variations with regards to the original proposal [77]. The most important one is the inclusion of virtual channels, VC, to improve the throughput of the network. The current implementation assumes 4 VC by default, but this value can be adjusted by the user. Besides, routers prioritizes packages based on their type and if equals based on their age. Deadlock is avoided by restringing the assignation of the last available VC to demotion messages.

Another aspect that deserved attention is the interface between the NUCA and the previous cache in the hierarchy. While in the original model there is a single injection point from the L1 to the NUCA, in multiprogrammed workloads, we observed a high congestion on this single channel and added a crossbar connecting all the first banks of each sparse set evenly distributing the traffic and reducing the processing delay at the first router.

Finally, instead of inserting misses in the tail banks, those further apart from the NUCA cache controller, we insert them into the head tile. The reason behind this change is that in the original work, the write policy of the L1 most probably was write-allocate copy-back and in our case is write-no-allocate write-through, so it is important to have write misses blocks close to the cache controller because otherwise their messages traverses all the NUCA fabric.

### 2.1.2 Simultaneous Multithreading Support

SMTScalar supports the simultaneous execution of multiple independent threads. The aim is to model multiprogrammed workloads because they tend to stress more the cache hierarchy than parallel workloads, and the complexity of its implementation is lower because coherence support is not required.

Table 2.2 summarizes the main features of the implementation. Currently, SMTScalar only supports the Icount fetch policy [134] and shares all the resources among threads without limitations except in the post-commit write buffer. During the setup, we observed that stalling a thread is only required when it monopolizes the post-commit write buffer; the release of entries takes up to hundred of cycles. When a thread occupies a number of entries equals to a threshold, 3/4 by default, it stops inserting more entries until the number decreases.

**Table 2.2:** SMT implementation details

| Policy | Description |
|---|---|
| Fetch | Icount [135] |
| Resource Allocation | None |
| Sharing | All resources are shared among threads including the reorder buffer and the issue windows |
| Stalling | When a thread have allocated too many entries, 3/4 of the total, in the post-commit write buffer, it is stalled until some of its entries has been released |

Since SimpleScalar does not provide support for virtual memory, SMTScalar assumes a fixed memory offset of 1048583 × the page size [3], 8KB, among threads. This simple partitioning minimizes the possibilities of conflict misses for example with the stack addresses at the beginning of the execution. Address collisions among threads are impossible because address always include the thread id.

All caches are shared among threads based on Marr's Ph.D. dissertation [91], and because private per-thread caches almost requires that the processor always executes the same number of threads contrary to commercial designs [57, 130, 72, 82, 71].

## 2.2 Methodology and Workloads

Two benchmark suites, SPEC CPU2000 and SPEC CPU2006 [53, 54] make up our collection of programs to evaluate our proposals. In both cases to reduce the simulation time we follow the SimPoint approach [50] and run traces of 100 millions of instructions. The SPEC CPU 2000 traces are the same than in Torres *et al.* [132], and Tables 2.3 and 2.4 show the computed SimPoint for SPEC CPU 2006, Integer and Floating Point benchmarks, respectively. In all the cases, we took the first SimPoint, so that the number of required system calls to implement was minimized. For those SPEC CPU 2006 benchmarks including several inputs, we select the representative set proposed by Phansalkar *et al.* [109].

SPEC CPU 2006 does not support Alpha machines and we were unable to successfully run 483.xalancbmk, which produced a well known stack overflow problem.

*warm-up*

[3] 1048583 is a prime value making the offset close to 8 Gigabytes.

**Table 2.3:** SPEC CINT2006 benchmarks with their respective 100M SimPoint

| Name | Input | SimPoint |
|---|---|---|
| 400.perlbench | -I./lib checkspam.pl 2500 5 25 11 150 1 1 1 1 | 14399 |
| 401.bzip2 | input.program 280 | 1070 |
| 403.gcc | 166.i | 255 |
| 429.mcf | inp.in | 907 |
| 445.gobmk | –mode gtp -i trevord.tst | 503 |
| 456.hmmer | –fixed 0 –mean 500 –num 500000 –sd 350 –seed 0 retro.hmm | 149 |
| 458.sjeng | ref.txt | 8221 |
| 462.libquantum | 1397 8 | 2370 |
| 464.h264ref | -d foreman_ref_encoder_main.cfg | 3828 |
| 471.omnetpp | omnetpp.ini | 6834 |
| 473.astar | rivers.cfg | 2207 |
| 483.xalancbmk | — | — |

**Table 2.4:** SPEC CFP2006 benchmarks with their respective 100M SimPoint

| Name | Input | SimPoint |
|---|---|---|
| 410.bwaves | — | 16688 |
| 416.gamess | triazolium.config | 29807 |
| 433.milc | su3imp.in | 8976 |
| 434.zeusmp | — | 17939 |
| 435.gromacs | -silent -deffnm gromacs -nice 0 | 5887 |
| 436.cactusADM | benchADM.par | 18497 |
| 437.leslie3d | leslie3d.in | 6372 |
| 444.namd | –input namd.input –iterations 38 –output namd.out | 12 |
| 447.dealII | 23 | 419 |
| 450.soplex | — | 674 |
| 453.povray | SPEC-benchmark-ref.ini | 1686 |
| 454.calculix | -i hyperviscoplastic | 10995 |
| 459.GemsFDTD | — | 31713 |
| 465.tonto | — | 31713 |
| 470.lbm | 3000 reference.dat 0 0 100_100_130_ldc.of | 179 |
| 481.wrf | — | 27497 |
| 482.sphinx3 | ctlfile . args.an4 | 17404 |

To avoid inaccuracies due to cold misses, traces include the previous 200M instructions of the SimPoint for the warm-up and 900M after in case a longer run are required. Both branch predictors and caches can individually use the warm-up instructions.

Our multithreading simulation methodology, described later, does not complete the simulation until all threads have committed 100M instructions. When a thread finishes its first 100M interval, it is restarted. All write buffers and MSHR belonging to the thread are flushed, and its cache blocks are invalidated without update the LRU position. Statistics are only gathered for the first execution interval of each thread.

SMTScalar verifies at user defined intervals and at the end of the simulation that the status of structures such as caches, MSHR, write buffers, ... are correct and consistent. Also instructions and cache requests require to pass several assertion checks for finding abnormal behaviours; i.e., a request that stays in a MSHR more than ten thousand cycles triggers an error.

*validation*

# Chapter 3

# Light NUCA Organization


### Summary

*This Chapter describes the inter cache latency gap as a thread for performance and proposes the Light NUCA, L-NUCA, cache organization as a possible solution for it. Then, it continues with a detailed description of the L-NUCA and analyses its viability with high level modeling tools. The core of the L-NUCA that are its Networks-in-Cache are characterized including the topologies, routing, and back-pressure mechanism. At the end of the chapter, we present some performance, energy, and area results of Light NUCAs paired with multiple Last Level Cache organizations running SPEC CPU2006 benchmarks with the SMTScalar framework.*

## 3.1 Introduction

High performance microprocessors rely upon memory hierarchy to achieve outstanding performance. To cope with slow DRAM main memories, computers include multiple cache levels shorting the latency gap between the processor and main memory. For example, the Intel Itanium 2 and the IBM Power6 include three cache levels [92, 82]. Unfortunately, the latency gap widens as technology scales, forcing a size increment in large Last Level Caches (LLC) that negatively impacts their latency. So, at the same time these LLCs reduce the latency to main memory, they widen an inter-cache latency gap between them and fast L1 caches.

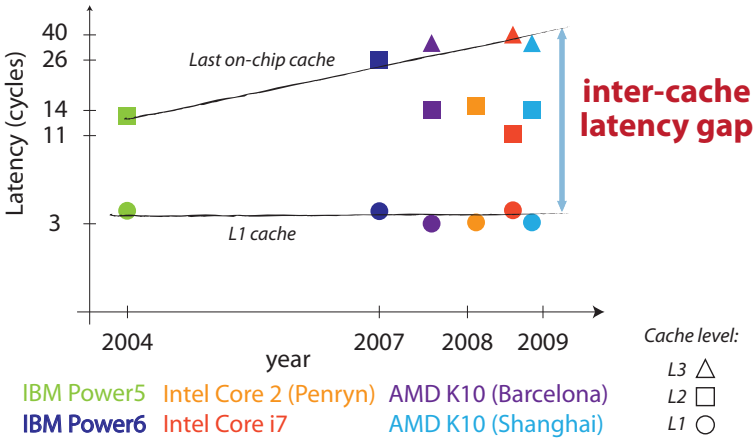



**Figure 3.1:** Inter-cache latency gap among three processor families

Figure 3.1 shows the grow of the inter-cache latency gap for the IBM Power, AMD K10, and Intel Core families during the last years. While the latency of the L1 cache has remained almost constant, that of the LLC has been multiplied between two and three. To bridge this new gap, two main approaches can be followed. The first one is based on reducing the latency of secondary caches. The second, consists of increasing the size of first level caches without compromising their latency, bandwidth, and pipeline integration.

Within the first approach, Kim *et al.* proposed Non-Uniform Cache Architectures (NUCA) to deal with the wire-delay [1] impact on latency of multi-megabyte caches [77]. NUCA connects cache banks in a 2D-mesh. Banks are individually accessible at latencies which are proportional to their distance from the processor. NUCA authors pioneered inter-bank block migration techniques, but their papers have solely focused on large secondary caches. In respect to the second approach, Balasubramonian *et al.* provided evidence of the latency/size trade-off

[1] Processor area has remain almost constant with technology scaling. On one hand, the advantages of this approach have been the inclusion of new features and large caches thanks to the immense number of available transistors and the increment in their speed. On the other hand, the distances among transistors have not scaled making the wire-delay a performance limiting factor.

between L1 and L2 caches. They proposed a reconfigurable cache able to dynamically adjust its size to the working set [10]. However, this scheme only supports single-ported cells, and it may not be able to provide the high bandwidth that superscalar processors require.

The present work tries to close the inter-cache latency gap by enlarging the cache accessible by the processor at low latencies. This is done without degrading bandwidth and without requiring any complex change in the critical processor execution core. Our proposal is based on a light dynamic NUCA that benefits from the fine granularity and working set adaptability of the Balasubramonian approach and from the non-uniform access time and block-migration techniques of original NUCAs. To make feasible this idea, we have to fight against the reasons that, up to now, have prevented NUCAs to be used as first level caches. Some of them follow.

NUCA latencies can be high. For example, an optimal bank delay of 17 cycles is suggested in [99] for a 32MB NUCA cache with 2MB banks. NUCA employs a 2D-mesh network with wormhole routing requiring at least one routing cycle before and after accessing any bank. NUCA links can suffer from contention making latency even higher and complex to predict. NUCAs employ a single-injection point and shared banks with multiple cycle initiation rate that can stall the network in miss bursts. Hence, it is easy to conclude that a fast networked cache tightly coupled to the processor would require other interconnection mechanisms such as operand networks [129, 113].

The main contribution of this dissertation is the proposal of a new Light NUCA design (L-NUCA), removing all the previous drawbacks. We will demonstrate through detailed simulation and benchmarking that L-NUCAs are valid candidates for bridging the inter-cache latency gap. As we will see, attaching an L-NUCA to a conventional hierarchy gives noticeable benefits in terms of performance, area, and energy consumption. In the same way, attaching a small L-NUCA to an original NUCA hierarchy is also beneficial for performance and energy consumption with a negligible area overhead.

Since NUCA latency and bandwidth mostly depends on its networking, L-NUCA focuses on improving topologies, routing, and packet delivery. In an L-NUCA hierarchy, the first cache levels are replaced by a set of small cache tiles (light NUCA banks) which surround the L1 cache. Tiles are connected by three on-chip networks specially tuned for different cache operations. The L-NUCA hierarchy manages a cache access and one hop routing within a single cycle. This allows for placing blocks at latencies inversely proportional to their temporal locality at a finer granularity than conventional or NUCA hierarchies. In addition, the L1 ability of servicing multiple loads with a small initiation rate is extended to the whole L-NUCA structure no matter its size. This reduces the negative impact that L1 miss bursts have in conventional or NUCA hierarchies with multi-cycle initiation rates.

This Chapter is organized as follows. Section 3.2 describes the background on Non-Uniform Cache Architectures and a bit on Networks-on-Chip. Section 3.3

introduces the L-NUCA organization. Section 3.4 describes the Networks-in-Cache. Section 3.5 evaluates the performance, energy, and area of L-NUCA with conventional and D-NUCA LLCs executing uniprocessor workloads, and Section 3.6 summarizes the chapter.

## 3.2 Background and Related Work

### 3.2.1 Non-Uniform Cache Architectures

Kim *et al.* introduced the Non-Uniform Cache Architecture (NUCA) organization [77]. Their focus was to reduce the impact of wire delay in large last level caches (LLC), so they replace the global wires between the banks and the cache controller with a conventional 2D-mesh and wormhole routers forming the static NUCA (S-NUCA). They extend the network with the ability of inter-bank block migration, dynamic NUCA (D-NUCA), and implement them in the TRIPS processor [42].

Later, many authors have focused on improving NUCA caches, mostly in two aspects: networks and content management in chip-multiprocessors. In both cases the target was LLCs. On the former group (NUCA networks), Jin *et al.* proposed a novel router for efficient multicast, a replacement algorithm, and a heterogeneous halo topology [67]. Muralimanohar and Balasubramonian introduced heterogeneity in the wires and in the topology with a mixed point-to-point bus network [99]. The same authors with N. Jouppi extended the Cacti tool to support NUCA caches and add multiple kinds of wires such as low-swing buses [100]. Foglia *et al.* proposed Triangular D-NUCA (TD-NUCA) for reducing power in large caches of embedded processors [35]. Chou *et al.* have proposed a single-cycle ring interconnection for Multi-Core L1-NUCA on 3D Chips [24]. They connect all the L1 caches with two counter-rotating rings and a global arbiter for granting permissions. This approach enables to efficiently share the L1 caches at the cost of increasing the load-use latency and complexity because remote accesses require multiple cycles (at least 4 plus the cache access), the TLB and the cache have to be serially accessed, and either an extra cache port or extra arbitration in the critical issue logic is required for handling local and remote cache accesses.

On the latter group (content management), Beckmann and Wood showed the complexity of block migration with requests coming from multiple processors [13]. With a similar layout, Lira *et al.* improves the bank replacement policy of D-NUCA based on the observation that some types of data are diferently accessed depending on which bank they reside in [85]. Merino *et al.* dynamically partition NUCA banks at set level in private and shared to reduce access latency and improve core isolation [94]. Recently, other proposals have departed from S-NUCA to extend them with OS directed placement in order to get the advantages of D-NUCA without the migration complexity [22, 6, 20, 51].

**Table 3.1:** Routing delay, bank latency and size, and total size of several NUCA proposals. L-NUCA does not target LLC and is placed between L1 and a large LLC

| | Routing delay (cycles) | Bank latency (cycles) | Bank size (KB) | Total size (MB) |
|---|---|---|---|---|
| S/D-NUCA [77] | 1 | 3 | 64 | 16 |
| TD-NUCA [35] | 1 | 3 | 64 | 8 |
| Cho *et* Jin [22] | 2 | 8 | 512 | 8 |
| Jin *et al.* [67] | 3 (min) | 4/5 | 256 | 16 |
| Muralimanohar *et al.* [99] | 3/4 (link) | 17 | 2048 | 32 |
| SP-NUCA [94] | 5 | 5 | 512 | 16 |
| Awasthi *et al.* [6] | 5 | 3 | 512 | 4/16 |
| PageNUCA [20] | 2/4 | 10 | 1024 | 16 |
| R-NUCA [51] | 3 | 14 | 1024 | 16/24 |
| LRU-PEA and Lira *et al.*[a] [85, 86] | 1 | 4 | 32 | 8 |
| ESP-NUCA [93] | 5 | 5 | 256 | 8 |
| L-NUCA[b] [124] | < 1 | < 1 | 8/32 | 0.125/0.5 |

[a] Associativity 8 and frequency 1.5 GHz
[b] Routing plus bank latency equals 1

Most of the previous proposals target multi-megabyte LLCs, and are made of large cache banks connected with conventional routers. Table 3.1 shows the routing delay, the bank latency and size, and the total NUCA size of several state-of-the-art proposals. In NUCA's seminal work [77], small bank sizes were employed in order to keep routing delay along a bank just within a clock cycle. Subsequent designs increased the bank size and thus the corresponding routing delay for improving performance in this large LLCs environment [22, 99, 100, 94, 6, 20, 51, 85, 93, 86]. L-NUCA places a small yet fast distributed victim cache between L1 and Last Level Caches to exploit temporal locality. Its aim is to close the latency gap between speed-optimized L1 cache and capacity-optimized LLCs, offering at the same time low-latency access to recently used blocks, virtually behaving as an L1 capacity extension mechanism. Hence, the L-NUCA cache works equally well whether the LLC is a conventional or a NUCA cache [124].

### 3.2.2 Networks-on-Chip

Most previous NUCA designs rely on conventional routers. Since communication delay significantly determines application performance, routers become a crucial component in the fabric. In fact, their importance rises as the number of cache banks and cores on-chip increases. Three components are mostly responsible for NoC routing latency: link delay, switch traversal delay, and control delay [34].

Since link delay is mostly determined by the technology, and switch traversal depends roughly on the degree of the network, many authors have focused on reducing control delay by speculatively routing flits. For example, Mullins *et al.* have proposed a single-cycle virtual channel router [98], and Michelogiannakis

*et al.* achieve the same objective for routers without virtual channels by adapting the mad-postman technique for on-chip communications [95]. Kumar *et al.* also presented a low-latency router with a low-complexity virtual channel allocator and a shared buffer [80].

In general, NoC routers are kept simple in order to minimize their area overhead. However, low-complexity routing policies such as oblivious routing may degrade performance—especially in bandwidth demanding scenarios. While previous designs were traffic agnostic, Cho *et al.* have published a traffic-aware router that changes the direction of the links depending on the demand to reduce congestion [21]. The MIT RAW and the TILERA chips also include some traffic-aware scalar operand networks (SON) for efficiently communicating register values [129, 141]. Since SONs convey register values instead of cache blocks as L-NUCA does, they employ smaller links ($\approx$ [8, 64] versus $\approx$ [128, 256] bits) and require very fast operation-operand matching [129, 124].

Another important aspect in NoC design is energy consumption. For 2D-meshes, Banerjee *et al.* implemented circuit-switched, wormhole, and virtual-channel routers in a 90 nm process [11]. Their study concludes that in future routers, spending some extra energy in elaborated control policies will decrease the system's overall energy consumption.

L-NUCA Networks-in-Cache exploit the fact that traffic patterns within caches are quite regular by using three specialized networks that simultaneously reduce control overhead and energy consumption through the minimization of the network activities.

## 3.3 Light NUCA Basic Operation

The processor interfaces to the L-NUCA through the *root-tile* (r-tile), which is a conventional L1 cache extended with the flow control logic required for sending and receiving blocks as shown in Figure 3.2. The rest of tiles surround the r-tile and are interconnected side by side only through local links in order to minimize wire delay. To simplify block migration, all the tiles share the same block size. Block search is efficiently performed by grouping tiles into growing size levels. For example, the L-NUCA of Figure 3.2 has 4 levels, named Le$i$. The r-tile forms the first level, the 5 tiles surrounding it the second one, and so on...

L-NUCA operation is simple yet efficient. Tiles only have knowledge of their local cache contents; hence, on a miss, the r-tile forward the miss request outwards to the Le2 tiles. If any of them hits, it sends the requested block back to the r-tile and stops propagating the miss. At the same time, because there is no time left in the cycle to communicate the hit, the remaining Le2 missing tiles are sending out the miss request to their Le3 leaf tiles. In the next cycle, the requested Le3 tiles will miss and propagate the request to Le4. Eventually, when all Le4 tiles miss, the request is forwarded to the L3 cache. Incoming blocks from the L3 cache are directly sent to the r-tile. This partial broadcast of miss requests has

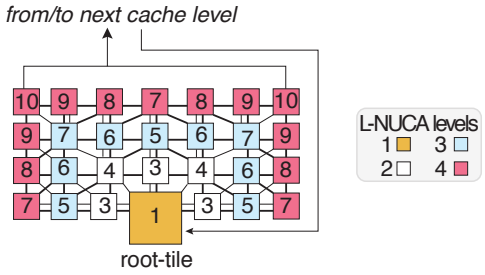



**Figure 3.2:** A 4-level L-NUCA. The numbers inside tiles and their color represent the tile hit latency seen by the processor assuming single-cycle tiles and their level;i.e., cyan represents the third level tiles (Le3), respectively.

some impact on dynamic energy consumption, but as we will see it is negligible because the number of levels and tile read energy are quite small.

After completing enough requests the r-tile becomes filled. In order to refill an incoming block from the L3, the r-tile will evict a victim block to an Le2 tile. The Le2 destination tile will repeat the eviction operation if the corresponding set is full; therefore, L-NUCA tiles act like a distributed victim cache similarly to the L2 from the Piranha CMP [68, 12].

Because tile latency is set to one processor cycle, each L-NUCA level can be looking up for a different request during the same cycle. Hence, the number of in-flight requests increases with the number of L-NUCA levels, contrary to conventional caches where size increments do not easily translate in supporting a higher number of parallel requests.

## 3.4 Networks, and Routing in L-NUCA

All the L-NUCA networks, their rules of routing, and its integration with the tile cache banks has been made looking at several broad goals, namely (i) maximize the hit ratio, (ii) minimize the hit time, (iii) minimize the miss determination time, and (iv) match the miss rate bandwidth of the r-tile and keep it even though L-NUCA size increases.

Concerning goal (i), L-NUCA capacity is maximized by managing tile contents in exclusion. Furthermore, the number of conflict misses is reduced by placing no restriction in block mapping into tiles. In particular, any L-NUCA set has local associativity (the tile associativity) and global associativity (local associativity times the number of tiles). The remaining goals impact the implementation of the three basic operations in L-NUCA: block search, block transport or service, and block replacement. Search operation requires a low-latency miss propagation network, goal (ii), along with a fast method for determining global misses, goal (iii). Both requirements together with goal (iv), call for integrating the tile cache access and one-hop routing within a single processor cycle. Transport operation requires a quick block delivery to the r-tile, avoiding delays due to contention (ii).

The replacement operation must contribute to goal (i), exploiting as much as possible the temporal locality. Finally, the proposed topologies and the interactions among networks supporting the basic operations must contribute to goal (iv).

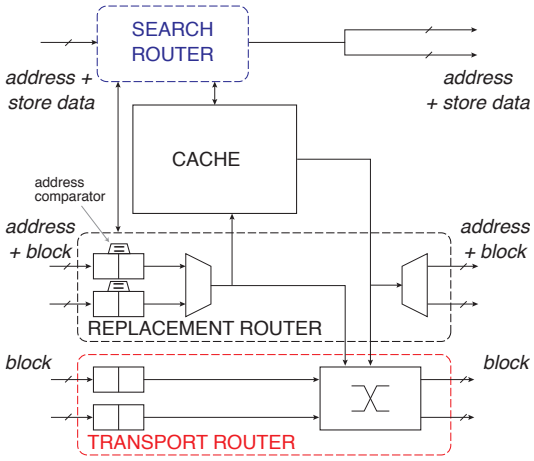



**Figure 3.3:** L-NUCA tile main components

Figure 3.3 shows the main components of an L-NUCA tile. It includes the three routers interfacing with the specialized Networks-in-Cache. All these mechanisms are independent of the caches they connect, and the only requirement is that caches have to support load hits under previous non-completed store hits as conventional caches do.

The rest of this section discuses topologies, routing strategies, and the integration of cache access with routing. Finally, we give some additional words on writing policies in L-NUCAs.

### 3.4.1 Topologies

On-Chip networks present some advantages compared to off-chip ones; e.g., the number of parallel wires can be larger [26]. Existing tiled architectures such as the MIT Raw or TRIPS already leverage from this wire-availability [129, 113]. Since L-NUCA tiles (4 to 32KB) are much smaller than D-NUCA ones (64KB to 2MB), the inter-tile distance is shorter, and smaller pitch metal layers can be used to route a very large number of wires. For example, using Cacti 4.2 [128], we have estimated that one side of a 2-way set associative 8KB cache in a 70 nm technology is less than 0.5 mm wide, and in this length there is room for more than 1600 wires in the M4 layer of an Intel production technology [8]. To benefit from this wire-availability, we advocate to replace the 2D-Mesh of NUCA caches with 3 dedicated point-to-point networks with unidirectional links, one for each cache operation: the *search*, *transport*, and *replacement* networks.

The three Networks-in-Cache employ unidirectional links, and a different topology adapted to each cache operation. As depicted in Figure 3.4, the *Search*

network uses a broadcast-tree. This topology propagates miss requests very fast through the fabric and requires the minimum number of links. In the shown 3-level L-NUCA, the distance from the r-tile to the rest of tiles is 2 or fewer hops with only 14 links in total. Besides, the maximum distance is only increased by one hop when another L-NUCA level is added. As a comparison, a NUCA 2D-mesh Search network would double the number of required hops to reach all the tiles, would increase the number of links by more than 50%, and would add 2 hops to the maximum distance when adding a new level.

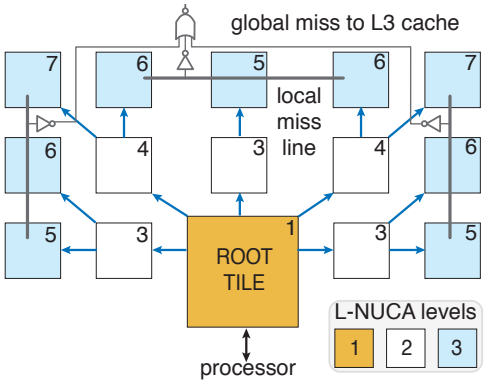



**Figure 3.4:** Broadcast tree search network topology

The Search network also collects global misses and forwards them to the next cache level. Global miss determination requires only to gather the miss status of all last-level tiles because when a tile experiences a hit it stops propagating the Search message. As shown in Figure 3.4, global misses may be efficiently determined with a segmented miss-line similar to the hierarchical bit-lines of SRAMs [147, 110]. We assume that this operation takes one-cycle after the last-level search. While this approach is not as scalable as Search network itself, global miss determination does not represent a problem because L-NUCA targets small caches where the lengths of segmented miss-lines are always small.

The *Transport* operation, shown in Figure 3.5, employs a 2D-mesh that offers multiple return paths to the r-tile ensuring path diversity and low contention even when the same tile hits during consecutive cycles.

To take profit of temporal locality, L-NUCAs place recently used blocks in low latency tiles. When evicting blocks, tiles place them as close as possible to the r-tile in terms of their distance when traveling through the transport network. So, except for the r-tile, the *Replacement* network connects tiles whose latencies differ in one cycle by means of an irregular topology with the lowest possible degree[2], see Figure 3.6. This topology tries to keep blocks in the L-NUCA as much as possible; hence, when a level is added the distance from the r-tile to the upper

[2]The degree of a node is the sum of all its input and output links. Low degrees reduce network complexity.

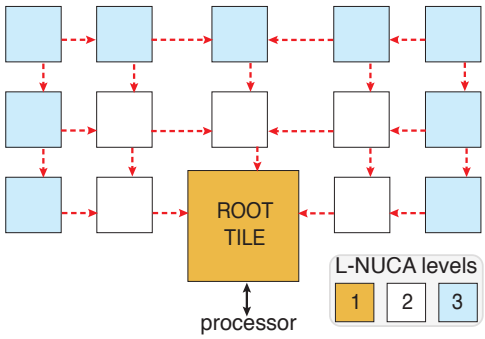



**Figure 3.5:** 2D-Mesh transport network topology

corner tiles—the only tiles that evict blocks to the next cache level— increases by 3 hops.

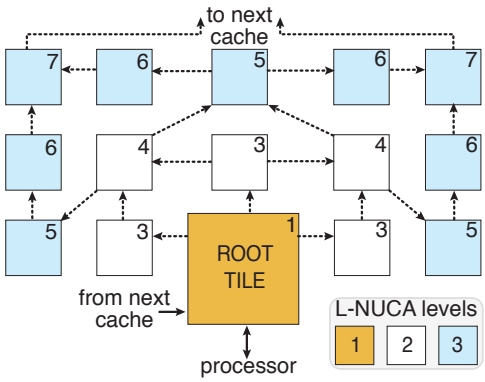



**Figure 3.6:** Latency-driven replacement network topology. Numbers inside tiles represent the tile latency assuming 1-cycle tiles. Tile latency includes search, tile access, and transport delay

### 3.4.2 Headerless Messages, Distributed Routing, and Flow Control

Each network transfers its own message, detailed in Table 3.2. Since the message destination is implicit in all the networks and their topologies ensure that all output links are valid for every message, L-NUCA employs headerless messages reducing routing delay (there is no need to read and manage headers) and the size of buffers and crossbars.

Routing a Search message simply consists in sending it to all tile leaves. This “wired" multicast is much simpler than multicast in conventional routers because it avoids the complexity of message replication [67].

Transport and Replacement networks route headerless messages with a dynamic distributed algorithm in which every node randomly selects an output link

**Table 3.2:** Network messages. Last column is computed assuming 32-Byte blocks, 41-bit addresses, and 16-entry r-tile Miss Status Holding Register (MSHR) and represents the link width of each network

| Operation | Message contents | Source | Destination | Size (bits) |
|---|---|---|---|---|
| Search | address + MSHR entry number + store data + control | r-tile | rest of tiles | 41+4+64+2=111 |
| Transport | block data + MSHR entry number | hit tile | r-tile | 256+4=260 |
| Replacement | block data + address + status | tile $j$ | a neighbour of tile $j$ | 256+41+2=299 |

(all output links are valid for all messages). Replacement messages are single-hop, so the routing operation is carried once for every message. On the contrary, Transport messages that always go to the r-tile are multi-hop, so the dynamic routing sends multiple messages with the same source through different paths. This reduces contention in comparison to dimensional order routing where all the messages with the same source and destination take the same route. Note that L-NUCAs, and NUCAs in general, have a single block return channel to the L1, so in order to avoid congestion it is very convenient to provide multiple routes to this single destination.

In respect to flow control, it is easy to apply on the Search network because Search messages cannot be blocked, so contention cannot arise. On the contrary, the Transport and Replacement networks can suffer from contention and rely on buffered flow control to avoid message dropping. Since links are message-wide, the flow control digits (flits) are the messages themselves; therefore, L-NUCAs use store-and-forward flow control with a two-entry buffer per link because round-trip delay between tiles is two cycles. L-NUCA signals the buffer availability with an On/Off back-pressure protocol [27]. Since hit blocks move down toward the r-tile and evicted ones move up, we call respectively downstream (D buffers), and upstream (U buffers), to the Transport and Replacement flow control buffers.

In L-NUCA, transport and replacement operations are completely decoupled. When a tile hits, the block is directly transported to the r-tile, which can start a “domino” replacement operation. The r-tile can send an evicted block to an Le2 tile that in turn will repeat the operation if it does not have an empty way. Eventually, an empty way will be found or a block will be evicted out of the L-NUCA to the next cache level.

The lack of cyclic dependencies among messages with the guarantee of message consumption ensures that L-NUCAs are deadlock-free by construction, not requiring virtual channels.

### 3.4.3 Parallel Cache Access and One-Hop Routing in a Single-Cycle

Now, we analyze the critical paths of search, transport, and replacement operations, looking carefully at the most demanding timing path: the integration of a cache access with an one-hop transport routing within a single processor cycle.

Figure 3.7 shows the timing of a conventional router, similar to those used in all previous NUCA designs, and of an L-NUCA tile. A Network-in-Cache made from conventional routers would raise the tile latency to 9 cycles: 4 to process the search request, 1 to access the cache, and 4 more for the transport message. High-performance routers can process a message in a single cycle in the best case [98, 80], so with conventional routers tile latency can not be lower than 3 cycles. Clearly, it is very hard to fit a cache access and one hop of routing in a single cycle without removing some router stages.

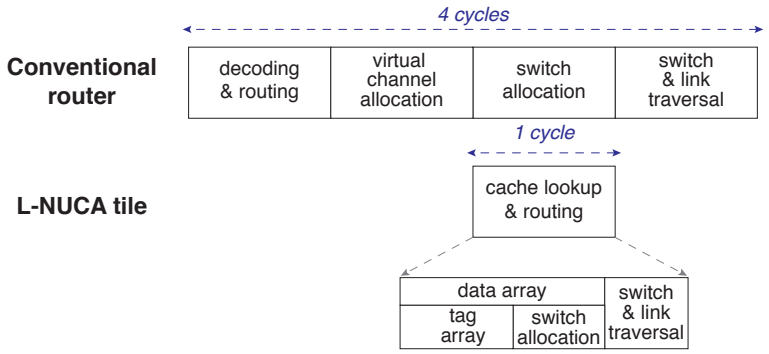
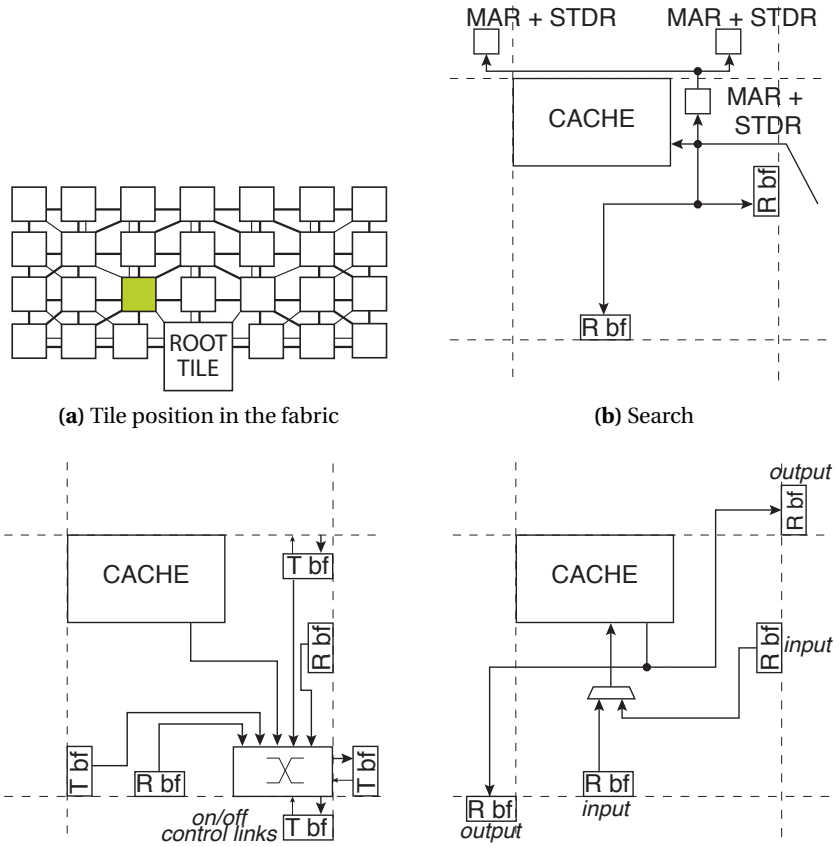



**Figure 3.7:** Conventional router and L-NUCA timing diagram

In the following, we will describe how it is possible to perform a cache access and a single hop in a processor cycle. For the sake of completeness, we will focus on the upper left corner tile in the second L-NUCA level as it has the maximum number of links a tile can require. Figure 3.8(a) shows its position in the fabric. Figures 3.8(b), 3.8(c) and 3.8(d) show respectively the Search, Transport, and Replacement network components.

**Search operation** Fig. 3.8(b). It begins when a tile receives a miss request in its Miss Address register (MA). At the beginning of the cycle, the requested address is looked up simultaneously in the tag array and in the U buffers that include an address comparator per entry (up to four comparators per tile: 2 U buffers × 2 entries/buffer). Looking up in the U buffers is enough for finding blocks in transit across the Replacement network (if any) avoiding false misses.

If the tile hits, a Transport operation starts. Otherwise, if the block is not present in the cache and U buffers, the request is propagated to the MA registers of the neighbor leaf tiles. Tile look-up and miss propagation is done in a single cycle because cache miss determination takes less time than the whole cache access. To be confident with that, we have computed the delay of the tag and



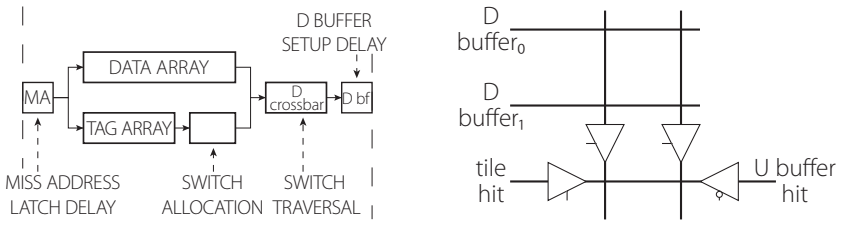

**(a)** Tile position in the fabric

**(b)** Search

**(c)** Transport

**(d)** Replacement

**Figure 3.8:** Tile organization with the components involved in each operation for an example tile having the highest degree. MA, T bf, and R bf stand for Miss Address register, Transport buffer, and Replacement buffer, respectively




**(a)** Critical path

**(b)** D segmented cut-through crossbar.

**Figure 3.9:** Timing critical path and output transport network crossbar

data paths (parallel access to tag and data arrays) for multiple caches with Cacti 4.2. For the small caches and low-associativity employed in L-NUCA tiles, the delay until the tag comparison represents roughly 60% of the total delay.

Note that to avoid false misses, conventional NUCA designs require either to prioritize replacement messages over search ones during routing arbitration (delaying them) or to check in every virtual channel that can contain an evicted block when performing block search operations. In fact, conventional virtual channel (VC) routers would require a comparator per virtual channel (a 4-port router with 4 VC per channel requires up to 16 comparators that have to be checked during the already critical virtual channel allocation stage) and additional logic to re-route the message.

**Transport operation** Fig. 3.8(c). It consists on routing hit blocks either from the tile D buffers or from the cache. In both cases, blocks are sent through the D crossbar and stored into a D buffer of a neighbour tile. In the rare event that a tile hits and all its output D channels are Off, the tile sends a contention-marked message through the Search network. When the marked Search message arrives to the global miss logic, it is returned to the r-tile that restarts the Search operation. We have observed in the simulations that this event rarely occurs due to the low contention of the transport network.

**Replacement operation** Fig. 3.8(d). It is only carried out during Search idle cycles: i.e., in the cycles when a tile has not received a miss request. Replacement requires two, non necessarily consecutive, cycles. In the first one, the control logic checks whether any of the output U channels are On, in which case a victim block (either clean or dirty) is read out from the tile cache, sent through the selected output U channel, and stored in the corresponding output U buffer. In the second cycle, the incoming block is written in the cache from the input U buffer, and if this buffer becomes empty, the tile will notify its neighbour that the channel is again On (remember that L-NUCA employs On/Off flow control). Replacement operations finish when a tile has an empty way in its cache or when a block is evicted from the L-NUCA.

From the previous paragraphs, we can deduce that a hit search followed by a transport operation is the critical timing path for a tile operation in L-NUCAs. Existing caches and routers already accomplish both task sequentially in a very low number of FO4s[3]; e.g., a sequential cache access to a 8KB subarray of the 8-way 64KB L1 data cache of the IBM Power6 plus one-hop routing in a 5x5 virtual-channel router takes less than 21 FO4s (9 the cache access and 12 the router) [110, 80]. This is roughly the cycle time of recent commercial microprocessors such as the Intel Core 2 Quad Q9550 [62]. Since our network subsystem is much simpler and allows to perform some routing tasks in parallel with the cache access, we

[3]A fanout-of-4 (FO4) is the delay of an inverter driving four identical copies of itself. FO4 is one way of referring to delay in a process-independent metric.

believe that both tasks fit perfectly in a single cycle. To verify this assumption, let's consider the timing diagram of L-NUCAs, see Figure 3.8(a), and compare it with the multiple stages that conventional virtual channel router performs, namely *decoding* and *routing*, *virtual channel allocation*, *switch allocation*, and *switch traversal* [107].

*Decoding* and *routing*. This stage is mainly removed because destinations are implicit and all the output links are valid for all the messages; hence, this state only requires to check if D buffers contain any entry and if the tile hits.

*Virtual channel allocation*. This stage is completely avoided because L-NUCAs do not employ virtual channels.

*Switch allocation*. This stage assigns output channels to input requesters. Basically, its delay depends upon how many resources have to be assigned and how complex is the assignment algorithm. Employing multiple networks reduce the maximum number of output channels to two. Therefore, random routing is fast in this case because it avoids the use of slow queue-based or matrix arbiters. Because switch allocation depends only on the number of occupied D buffers and on the results of the miss address comparison (tag array access and U comparators), it can be performed in parallel with the data array acess; therefore, its delay can be overlapped with the cache access.

*Switch traversal*. This stage sends messages through the crossbars. It is a demanding stage in the Transport network due to its larger number of inputs, 5 (2 D buffers, 2 U buffers, and the cache bank). Because L-NUCAs ensure that only a single copy of each block exists(content exclusion is enforced among tiles), hits can not happen simultaneously in the cache and in the U buffers and the number of inputs can be reduced to 3. This favors the use of a cut-through crossbar (Figure 3.8(b)) that reduces the number of inputs (from 5 to 3), latency, and dynamic energy [139]. For measuring its delay, we have modeled this crossbar with HSPICE and found that is below 5FO4s, which is in line with previously published results [76].

Summarizing, L-NUCAs only add a low-latency *switch traversal* stage to the critical path, so we conclude that a cache access and one-hop routing can both fit within one processor clock cycle. Besides, this objective is achieved without requiring any complex routing mechanism such as speculation or look-forward routing [107, 27].

### 3.4.4 Write Policy

Many current processors implement write-through (WT), no-write allocation (NWA) L1 caches to simplify coherence because it guarantees that dirty blocks can be supplied directly from L2/L3 caches in a multiprocessor environment without requiring neither to disturb nor to add additional ports to the L1 cache. On the other hand, a burst of processor writes can fill the interface between L1 and L2 (we assume in this work a Coalescing Write Buffer, CBW), occasionally stalling the processor and hurting performance.

L-NUCA tiled fabric offers a range of write policies balancing the trade-off between the updating degree of the upper cache levels and write bandwidth. To equilibrate both aspects, in this work we employ the following mixed write policy, supported by the Search network in the form of Search + Write messages:

- r-tile write hits are forwarded to the next cache level (L3/NUCA) through a Coalescing Write Buffer (CWB) by means of a dedicated path. Hits in any other tiles only make a local update, setting the dirty bit. The written block neither changes its replacement status nor is it moved to the r-tile (no-write transport).

- global write misses, like global read misses, end up in the last L-NUCA level and are sent to the CWB from one of the upper edge tiles, see Figure 3.8(d).

- when evicting a dirty block from the L-NUCA, it is sent to the next cache level through the same CWB, which also acts as a copy-back buffer.

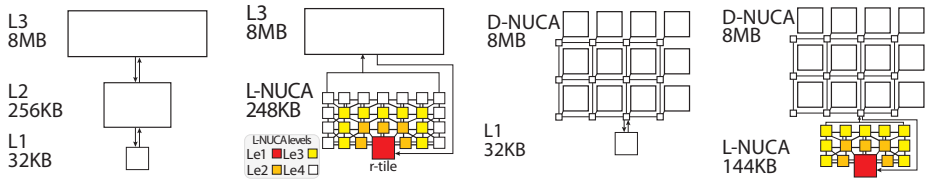
Hence, in contrast with a conventional WT-NWA write policy, our option hides to the next cache level the write stream hitting in the all tiles but the r-tile. Of course other policies can be devised, ranging from pure write-through to pure copy-back, but for the uniprocessor workload tested in this work, the previous policy has proven adequate.

### 3.4.5 Touch bit

When the L-NUCA expels a block to the next cache level, and this level is not a victim cache, it has to choose whether it drops clean blocks or not. Dropping saves bandwidth at the cost of losing potential hit candidates. L-NUCA follows a intermediate solution called touching. The idea is to mark those blocks that have been serviced, at least one time, from all tiles but the r-tile, and then drop only the rest of blocks. The implementation is straightforward, tags include an extra sticky bit that is reset when the block is inserted into the r-tile from the next level cache and is set every time the block arrives to the transport buffers of the r-tile. This simple policy filters streaming and all other low locality benchmarks.

## 3.5 L-NUCA Evaluation in Uniprocessor Environments

We have studied L-NUCA in two representative environments shown in Figure 3.10: a conventional 3-level hierarchy in which an L-NUCA replaces the L2 cache (Figures 3.10(a) and 3.10(b)) and a D-NUCA hierarchy in which an L-NUCA is placed between the L1 and the D-NUCA (Figures 3.10(c) and 3.10(d)). In the D-NUCA environment the added L-NUCA is smaller because D-NUCA partially shortens the latency gap.




**(a)** Conventional **(b)** L-NUCA + L3 **(c)** D-NUCA **(d)** L-NUCA with D-NUCA

**Figure 3.10:** Cache hierarchies studied in this work. L-NUCA levels from Le2 to Le4 are made of 8KB tiles.

### 3.5.1 Baseline Processor Description

All simulations have been run with SMTScalar running the SPEC CPU2006 traces described in Chapter 2. All caches use LRU replacement except L-NUCA, which employ LRF (Least Recently Filled), and the 8MB L3 is similar to the Intel Core 2 [38]. D-NUCAs are modelled based upon the SS-performance configuration in [77]. For the rest of parameters see Table 3.3.

As regards delay, energy, and area, we assume a cycle time of 19 FO4s similar to the Intel Core2 Duo E8600 in a 32 nm technology [63]. Cache area, delay, and energy are estimated with Cacti 5.3 [131] improved to support access time as an optimization objective. L3 caches employ Low Operating Power (LOP) transistors and the rest of caches High Performance (HP) ones. Within our cycle time constraints, the largest configuration found for the one-cycle L-NUCA tile was an 8KB-2Way-32B cache. The area and energy of routers have been estimated with Orion [138]. The transport crossbar delay was modelled with HSPICE to verify it satisfies the cycle time constraints.

### 3.5.2 L-NUCAs vs. Conventional Hierarchies

The selected baseline configuration (*L2-256KB*) was the one performing the most in our exploration of the L2 design space for three-level conventional hierarchies. This baseline is compared to three L-NUCAs (2, 3, and 4 levels).

Regarding area (Table 3.4), the L-NUCA low network overhead with their small size keep area under control; e.g., *LN3-144KB* saves a 5.5% compared to *L2-256KB* while overpassing its performance.

Regarding performance, Fig. 3.11 shows the IPC for all configurations. All L-NUCAs overpass *L2-256KB* with average gains ranging from 5.4% (*LN2-72KB*) to 6.22% (*LN4-248KB*) in Integer and from 14.3% (*LN2-72KB*) to 15.4% (*LN4-248KB*) in Floating Point. Integer results are biased by *462.libquantum*, which shows a constant and very low IPC of 0.27. When *462.libquantum* is removed gains increase from 6.7% to 9.0% for *LN2-72KB* and *LN4-248KB*, respectively.

Concerning the optimal number of L-NUCA levels, the fourth level and beyond is not worthy because it increases area 81% compared to *LN3-144KB* and

**Table 3.3:** Architectural and network parameters.

| | | | | | |
|---|---|---|---|---|---|
| Fetch/Decode width | 4, up to 2 taken branches | Issue width | 4(INT or MEM)+4 FP | Store Buffer size | 48 |
| Branch predictor | bimodal + gshare, 16 bit | Commit width | 4 | L2/L3 Write Buffer size | 32/32 |
| ROB/LSQ size | 128/64 | MSHR L1/L2/L3 size | 16/16/8 | TLB miss latency | 30 |
| INT/FP/MEM IW size | 32/24/16 | MSHR sec. misses | 4 | Branch misspred. delay | 8 |

| | |
|---|---|
| L1 cache / r-tile | 32KB, 4 Way, 32B block size, parallel access mode, 2-cycle completion, 1-cycle initiation, write-through, 2 ports, read hit ener.: 21.2 pJ, leakage power: 12.8 mW |
| L2 cache | 256KB, 8 Way, 64B block size, 4 cycle completion, 2 cycle initiation, serial access mode, copy-back, 1 port, read hit ener.: 47.2 pJ, leakage power: 66.9mW |
| L-NUCA cache tile | 8KB, 2 Way, 32B block size, parallel access mode, 1-cycle completion and initiation, copy-back, 1 port, read hit ener.: 14 pJ, leakage power: 2.2 mW |
| L3 cache | 8MB, 16 Way, 128B block size, 20-cycle completion, 15-cycle initiation, copy-back, 1 port, read hit ener.: 20.9 pJ, leakage power: 600 mW |
| D-NUCA cache | 8MB, 8 sparse sets, 4 rows. Banks: 256KB, 2 Way, 128B block size, parallel access mode, 3-cycle completion and initiation, 1 port, read hit ener.: 131.2 pJ, leakage power: 33.5 mW. Links: 256 bits |
| Main memory | First chunk: 200 cycles, 4-cycle inter chunk, 16B wires |

| | D-NUCA | L-NUCA |
|---|---|---|
| Flit Size | 32B | 32B |
| Buffer size per channel | 4 (virtual) | 2 (physical) |
| Routing latency | 1 | 1 |

| | D-NUCA | L-NUCA |
|---|---|---|
| Flits per message | 1-5 | 1 |
| Virtual channels | 4 | — |
| MSHR size | 16 | 16 |

**Table 3.4:** Conventional and L-NUCA area. The r-tile of L-NUCA configurations is the same 16KB-32BS-2W L1

| | L1 + L2 /L-NUCA area ($mm^2$) | tile area ($mm^2$) | network area percentage (%) |
|---|---|---|---|
| *L2-256KB* | 0.91 | — | — |
| *LN2-72KB* | 0.46 | 0.06 | 14.01 |
| *LN3-144KB* | 0.86 | 0.06 | 18.8 |
| *LN4-248KB* | 1.59 | 0.06 | 19.02 |

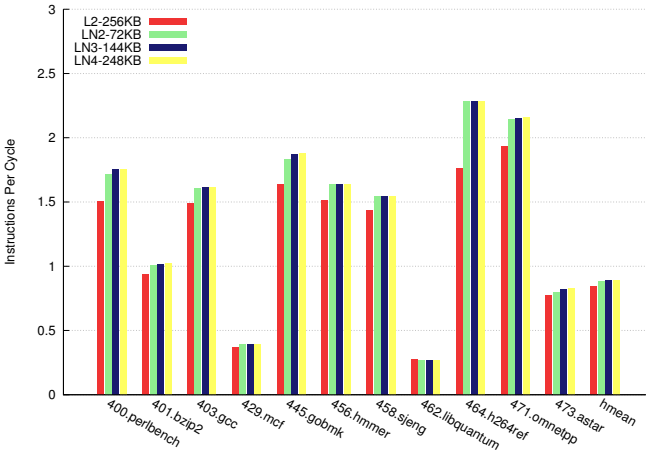
only increases performance in a 0.13% and 1.1% for Integer and Floating point benchmarks.

Digging into the individual behaviour of benchmarks, those with bigger Le-to-L2 hit-ratios improve more. Focusing on *LN3-144KB*, in the 2 Integer benchmarks (*458.sjeng* and *464.h264ref*) and in the 7 floating point ones (*416.gamess*, *437.leslie3d*, *453.povray*, *454.calculix*, *459.GemsFDTD*, *465.tonto*, and *481.wrf*) with an Le2-to-L2 bigger than 90%, gains are over 7% (*437.leslie3d*) and rise up to 30.3% (*464.h264ref*). The benchmark *410.bwaves* does not reach 90% Le2-to-L2 hit-ratio, but repeatedly experiences an access pattern just like this: (L1 read, L1 replacement, write, L1 read miss). Note that the second read in a row is serviced by the L2 in the conventional case, but in L-NUCA it is feed by the lower latency tiles. For those benchmarks without locality to exploit such as *429.mcf*, L-NUCA also provide some advantage because their miss time is smaller than the miss time of the L1 plus the L2.

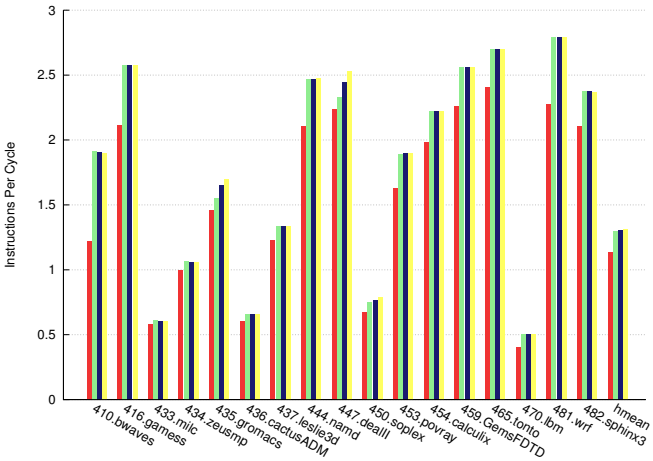
**Table 3.5:** Percentage of the number of read hits in each L-NUCA level relative to the number of read hits in the L2 of *L2-256KB*

| | Le2 / L2 (%) | | Le3 / L2 (%) | | Le4 / L2 (%) | | All levels / L2 (%) | |
|---|---|---|---|---|---|---|---|---|
| | Int. | FP. | Int. | FP. | Int. | FP. | Int. | FP. |
| *LN2-72KB* | 58.7 | 40.9 | — | — | — | — | 58.7 | 40.9 |
| *LN3-144KB* | 59.9 | 41.0 | 21.2 | 29.4 | — | — | 81.2 | 70.3 |
| *LN4-248KB* | 60.1 | 41.0 | 21.1 | 27.1 | 7.4 | 19.5 | 88.6 | 87.7 |
| Average | 59.6 | 41.0 | 21.2 | 28.3 | 7.4 | 19.5 | — | — |

L-NUCA gives consistent gains across benchmarks because it reduces the average latency of cache hits. To give an insight into this reasoning, left part of Table 3.5 shows the average percentage of L-NUCA read hits relative to the *L2-256KB* ones. On average, 59.6% and 41% of L2 hits are serviced by the 5 Le2 tiles at a smaller latency, for Integer and Floating Point respectively. Besides, these hits do not suffer from contention as the right part of Table 3.5 proves. These two columns show the ratio between the average transport latency (delay between a tile experiences a hit until the Issue Window is notified about the data availability) and the minimum transport latency (no contention) for all tested benchmarks. These values (less than 1.5% of increment) prove the effectiveness




**(a)** Integer




**(b)** Floating Point

**Figure 3.11:** Instructions Per Cycle for conventional and L-NUCA configurations per benchmark

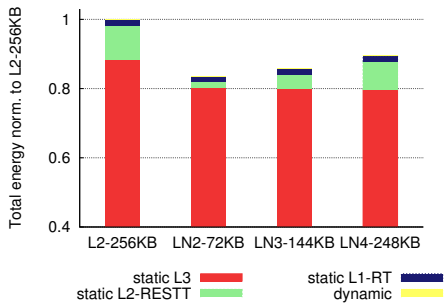



**Figure 3.12:** Total Energy consumption for conventional and L-NUCA configurations. RT stands for r-tile and RESTT for rest of tiles. Network energy consumption is included for the L-NUCA configurations

of the proposed topologies and the distributed dynamic routing.

**Table 3.6:** Average-to-Minimum transport latency ratio. A value of 1 represents no congestion

| | Integer | Floating Point |
|---|---|---|
| *LN2-72KB* | 1.014 | 1.009 |
| *LN3-144KB* | 1.008 | 1.005 |
| *LN4-248KB* | 1.005 | 1.004 |

Regarding energy, cache consumption is dominated by static energy in current technologies [146]. Therefore, L-NUCA performance improvements also entail energy savings because the savings in static consumption overpass the small increment in dynamic energy caused by inter-tile block migrations and search broadcasts.

Fig. 3.12 shows the average energy consumption for all benchmarks normalized to the *L2-256KB* configuration. As expected, L3 static energy stands out above the rest, and the execution time reduction coming from L-NUCA saves roughly 10% of static L3 energy. In the overall picture, all L-NUCA configurations beat the baseline with savings ranging from 10.5% (*LN4-248KB*) to 16.5% (*LN2-72KB*). We do not plot energy results per benchmark because the shape is similar to that of IPC because static energy is proportional to execution time.

Putting together area, performance, and energy results, the replacement of a 256KB L2 by an 144KB L-NUCA saves a 5.5% in area, improves IPC by 6.1% and 15% for Integer and Floating Point benchmarks, respectively, and reduces energy by 14.24%.

### 3.5.3 Integrating L-NUCAs with D-NUCAs

The 8MB D-NUCA baseline (*DN-4x8*) was selected after exploring its design space varying, among others, bank size, link width, number of sparse sets, number of rows, associativity, insertion policy, and injection bandwidth. *DN-4x8* includes 32 banks as described in Table 3.3.

**Table 3.7:** L1 + D-NUCA and L-NUCA + D-NUCA configurations. Bank column shows the bank size (in KBytes), block size (in Bytes), and latency (in cycles)

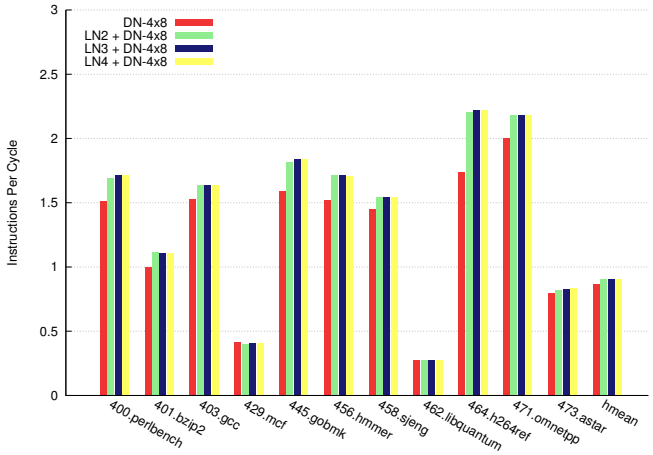
| | D-NUCA | | | |
|---|---|---|---|---|
| | sparse sets | rows | bank | size (MB) |
| *DN-4x8* | 8 | 4 | 256KB-128B BS-3c | 8 |

| | | L-NUCA + D-NUCA | | | |
|---|---|---|---|---|---|
| | L-NUCA | sparse sets | rows | bank | size (MB) |
| *LN2 + DN-4x8* | LN2-72KB | | | | 8.04 |
| *LN3 + DN-4x8* | LN3-144KB | 8 | 4 | 256KB-128B BS-3c | 8.11 |
| *LN4 + DN-4x8* | LN4-248KB | | | | 8.23 |

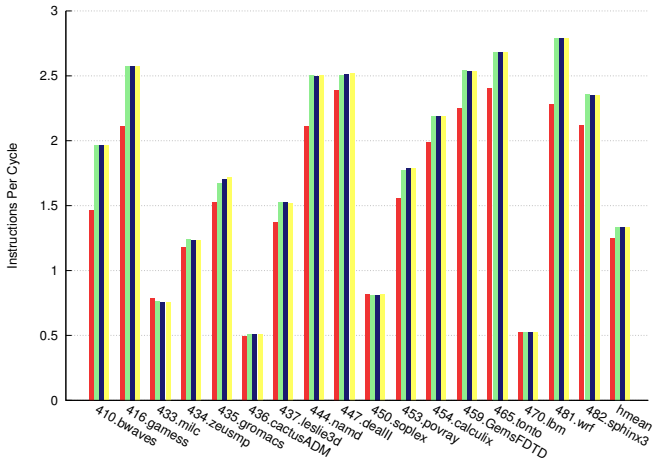
Table 3.7 presents all the configuration under test in this section. The increment in size also increases area. We also test L-NUCA with other D-NUCA organization, but DN-4x8 was the best configuration in both cases, mostly due to its higher bandwidth from the 8 rows.

Regarding performance, Fig. 3.13(a) compares Integer and Floating Point IPC of all configurations. The combination of L-NUCA + D-NUCA always improves performance, being the achieved gains almost flat across the tested L-NUCAs, roughly 4.5% and 7% for Integer and Floating Point benchmarks, respectively. Since D-NUCA organizations reduce the latency gap when compared to conventional hierarchies, the number of L-NUCA levels required to maximize performance decreases; therefore, two levels are enough, and with a slight 1.2% area increment, *LN2 + DN-4x8* improves *DN-4x8* by 4.2% and 6.8% for Integer and Floating Point, respectively. Besides, gains are consistent across benchmarks and in 54% of them IPC improves more than 10%. In 6 of 11 Integer (*400.perlbench*, *401.bzip2*, *445.gobmk*, *456.hmmer*, *464.h264ref* and *471.omnetpp*) and in 11 of 17 Floating Point (*410.bwaves*, *416.gamess*, *435.gromacs*, *437.leslie3d*, *444.namd*, *453.povray*, *454.calculix*, *459.GemsFDTD*, *465.tonto*, *481.wrf*, *482.sphinx3*) IPC improvements are larger than 10%.

Regarding energy, Fig. 3.14 shows the total energy consumption relative to *DN-4x8*. Once again, the execution time reduction helps to decrease the energy consumption from 4.25% (*LN2 + DN-4x8*) to 0.2% (*LN4 + DN-4x8*). Interestingly, *LN2 + DN-4x8* saves 19.8% of dynamic energy with regards to *DN-4x8* because the added activity in the L-NUCA (8KB tiles, simple networking) requires less energy than the dynamic activity removed in the D-NUCA (256KB banks, virtual channel routing).




**(a)** Integer




**(b)** Floating Point

**Figure 3.13:** Instructions Per Cycle for L1 + D-NUCA and L-NUCA + D-NUCA configurations

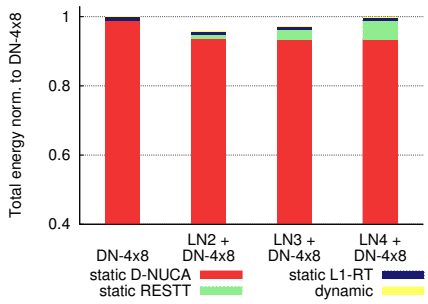



**Figure 3.14:** Total Energy for L1 + D-NUCA and L-NUCA + D-NUCA configurations

Summarizing, with a negligible 1.2% area increment, adding an LN2-72KB to a D-NUCA hierarchy improves IPC by 4.2% and 6.8% for Integer and Floating Point benchmarks, respectively and saves 4.25% in total energy consumption.

### 3.5.4 Hierarchy Performance Comparison

Figure 3.15 shows the average IPC for the four organization studied in this chapter: conventional (*L2-256KB*), D-NUCA (*DN-4x8*), L-NUCA (*LN3 + L3*), and L-NUCA + D-NUCA (*LN2 + NC-4x8*. Not surprisingly, *DN-4x8* beats the conventional organization. L-NUCA configurations overcomes both conventional and D-NUCA thanks to the implementation of the principles underlying the non-uniform cache architecture concept, but in a range of latencies and sizes smaller than those affordable in conventional NUCA designs.

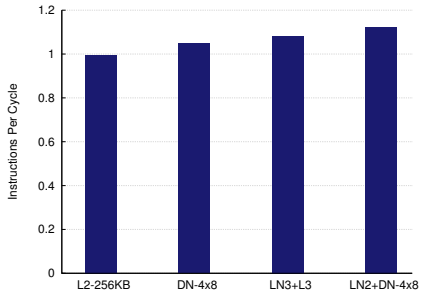



**Figure 3.15:** Average performance of the different studied organizations.

Previous works reported larger gains of D+NUCA versus conventional hierarchies [77]. However, LLCs L3 block size and cycle time are larger in our evaluations, 128 versus 64 KB and 16 versus 8 FO4s, but most important we choose their best configuration as baseline that was tested in SPEC CPU 2000 benchmarks instead of 2006 ones. In SPEC CPU 2000, our infrastructure reports a 15.9% IPC gain of D+NUCA versus the 3-level conventional hierarchy that is very similar to theirs.

The most interesting point to comment in terms of performance is that L+D-NUCA is the best of all the considered organizations. With regards to *L2-256KB*, *LN2 + NC-4x8* improves average performance by 13.1% and shows the advantages of these flattened hierarchies.

## 3.6 Final Remarks

The inclusion of large secondary on-chip caches for closing the latency gap between the processor and main-memory causes another latency gap between those large caches and the fast and small first level caches. This work tackles this problem by extending the cache size reachable by the processor at very low

latencies. This objective is achieved by adapting NUCA caches to size-reduced caches.

One of the main novelties of L-NUCAs is their interconnection system. Three different networks have been conceived for the basic cache operations: search, transport, and replacement. All of them are based on short and scalable local links. Besides, it supports fast lookup and block delivery in a fully-associate structure maximizing hit ratios. Routing is implicit in all the networks minimizing cost and increasing message delivery. With this interconnection fabric, L-NUCA performs in parallel a cache access and one-hop routing in a single cycle.

Our detailed simulations show that, in general, L-NUCA improves simultaneously performance, energy, and area when integrated into both conventional or D-NUCA hierarchies for single thread workloads.

# Chapter 4

# Light NUCA VLSI Implementation


### Summary

*This Chapter describes the VLSI Implementation of the Light NUCA to prove its feasibility. All three proposed Networks-in-Cache incur minimal latency, area, and energy overhead. Since the 90 nm technology available for the implementation was low-power oriented, this chapter also compares L-NUCA performance in the domain of high-performance low-power embedded processors, which are the heart of smartphones or netbooks. With a collection of benchmarks for this domain, L-NUCA classes first against a conventional cache hierarchy similar to that of a representative high-performance low-power embedded processor and a Static NUCA because it does not have any energy consumption overhead due to block migration. Since energy consumption is critical in embedded applications, we propose two network-wide techniques (Miss Wave Stopping and Sectoring) that together reduced L-NUCA dynamic energy 35% without degrading performance. Finally, we scale the improved design, named Light Power NUCA (LP-NUCA), to a 32 nm technology with high level tools and observe that L/LP-NUCA scales well and continue overpassing the rest of organizations.*

## 4.1 Introduction

The complexity and variety of embedded applications are constantly increasing demanding systems with high computing capacities, low power consumption, and many specific analog and digital devices. In order to execute these applications and reduce costs embedded systems are integrated into System-on-Chips (SoC). Many manufacturers offer SoCs meeting these stringent requirements. To reach the performance goal, SoCs rely on concepts and techniques previously proposed for the high-performance computing segment, but tuned for minimizing energy consumption.

A review of current advanced embedded SoCs (40-45nm) clearly shows how their CPUs follow close behind their high-performance siblings. Among other advanced features (see Table 4.1), embedded processors profit from out-of-order wide issue (e.g., the Xeon LC3528, XLP832, and PowerPC 476FP issue 3, 4, and 5 instructions per cycle, respectively [64, 49, 89]) or multithreaded execution (e.g., MIPS32-1004K and Xeon LC3582 support 2 concurrent threads while the XLP832 supports 4 [96, 64, 48]).

**Table 4.1:** Representative processor samples of current high-end SoCs (40-45 nm). Shaded cells show the Last Level Cache (LLC)

| **Model**, *Brand* | # threads / CPU | L1 / CPU (KB) I | L1 / CPU (KB) D | L2 / CPU | L3 (MB) |
|---|---|---|---|---|---|
| **QorIQ P5010**, *Freescale* [47] | 1 | 32 | 32 | 512KB | 1 |
| **PowerPC Proccessor (476FP) Embedded core**, *IBM–LSI* [89] | 1 | 32 | 32 | 512KB | 2 shared + 4 eDRAM |
| **Xeon LC3528**, *Intel* [64] | 2 | 32 | 32 | 256KB ECC | 4 shared |
| **Cortex-A9 MPCore**, *ARM* [49] | 1 | 32 | 32 | up to 8MB shared | no |
| **MIPS32-1004K Coherent Processing System**, *MIPS* [96] | 2 | 32 | 32 | up to 1MB shared | no |
| **XLP832**, *NetLogic* [48] | 4 | 64 | 32 | 512KB | 8 shared |

These techniques (wide issue, multithreading, ...) put pressure on the cache hierarchy, so its design also inherits concepts and techniques from the high-performance segment. For example, in the first level of the hierarchy (L1) we can see multi-ported data caches optimized for speed, as the dual-ported cache of the XLP832 [48], and close to the main memory we find relatively big Last Level Caches (LLCs) optimized for size; highlighted with shaded cells in Table 4.1.

An appealing proposal that has received much attention for improving the performance of the cache hierarchy has been the Non-Uniform Cache Architecture (NUCA) [77, 67, 99, 100, 35, 13, 22, 6, 20, 51, 94, 124, 85, 24, 93]. The rationale behind NUCA is the following: as technology scales, large Last Level Caches (LLC) suffer from wire delay; driving a signal from the cache controller to the banks takes more time than the bank access itself. Originally, Non-Uniform Cache Architectures tackle this problem by merging the L2 and the L3 into a mesh of cache banks and enabling inter-bank block migration; therefore, blocks located

in the banks close to the cache controller have a lower latency that those located further apart [77].

Nevertheless, there are almost no examples of NUCA caches in high-performance embedded SoCs. A close approach is the L3 cache of NetLogic XLP832 [48]. This LLC is divided into 8 independent banks and each bank is attached to a bidirectional ring that also communicates with 4 DDR ports and 8 private L2 caches. Since L3 cache lines are statically mapped into banks, the NetLogic solution resembles the static NUCA proposal, but with a topological change: from the original mesh to a ring.

According to our knowledge, there is little published work attempting to bring NUCA caches into SoCs. Foglia *et al.* took a first step in that direction proposing a low-power NUCA for large embedded LLC (triangular NUCA) [35], and Chou *et al.* continue it with a ring for sharing L1 caches in 3D multi-core chips [24]. Following this path, in this chapter we extend Light NUCAs for deployment in embedded environments.

Light NUCA (L-NUCA) is our proposal for closing the latency gap between speed-optimized L1 caches and capacity-optimized LLC [124]. They try to smooth this gap by retaining L1 Most Recently Evicted blocks in a fabric of very small tiles (capacity: 8-32KB, latency: 1 processor cycle). Tiles behave as a very large distributed victim cache [68] leveraging short-term temporal locality. The key novel feature of L-NUCAs is their three specialized Networks-in-Cache. These networks exploit the ready availability of on-chip wires to minimize the network overhead in terms of delay, area, and power. The resulting cache offers a reduced average memory access time and a lower energy consumption when incorporated into either a 3-level cache hierarchy (replacing the L2) or into a NUCA hierarchy [77] (small L-NUCA + Dynamic NUCA).

To prove the scalability and benefits of L-NUCA and its Networks-in-Cache, we have implemented L-NUCA in a 90nm ASIC technology. From the implementation, we draw several conclusions. First, only two types of tiles are required to build L-NUCAs of any size. Second, all the Networks-in-Cache proposed in L-NUCA involve minimum overhead and all tile operations, including cache access plus routing fit in a single core clock cycle [1]. Third, on-chip wire availability can be used to reduce router complexity and, hence, latency. Fourth, cache accesses (SRAM cells and cache control logic) account for most of the total tile power, around 75%. Since energy consumption is crucial in embedded processors we propose two network enhancements for reducing it based on the previous fact. From the implementation, we have observed that there is time to propagate a few global signals among tiles to reduce the cache accesses, which we call *Miss Wave Stopping*. With subsequent simulations we have also observed that block placement can be restricted (which we call *Sectoring*) without degrading performance. Both techniques taken together form Light-Power NUCA (LP-NUCA), a specialization of L-NUCA for high-performance embedded processors.

[1] Tile latency is mainly dominated by the cache access, which, in turn, is due to its SRAM macros.

In order to evaluate the performance and energy attained, we have fed a detailed cycle-by-cycle simulator with the energy costs from the different layout variants, finding that LP-NUCA reduces the dynamic power consumption of L-NUCAs by 35% without performance losses when running representative embedded workloads. Compared to a slightly larger conventional L2 cache that mimics the memory hierarchy of a representative high-end embedded processor, both L-NUCA and LP-NUCA, besides power savings, enable also faster execution. Similar conclusions have been derived after testing the energy behavior of LP-NUCA at 32nm using Cacti simulations.

The rest of this chapter is organized as follows: Section 4.2 comments on some points related to the implementation that where not presented on Chapter 3 Section 4.3 presents on its VLSI implementation. The performance of L-NUCA is evaluated and compared to previous state-of-the-art second level cache organizations in Section 4.4. The low power enhancements of the basic L-NUCA are described in Section 4.5 along with the energy savings achieved. Finally, Section 4.6 concludes the chapter.

## 4.2 Initial Observations about the VLSI Implementation

This section comments some interesting details from the implementation. Most of them related on the architecture of several latency critical components. First, we start with the cache inside the tiles and with the generation of Search Network control signals. Then, we continue with the management of load requests that hit in the write buffers, and we end with the transport router, which is the last element in the critical path of a tile.

As previously stated, the cache size of a tile is chosen so that the L-NUCA cycle time (cache access plus one hop routing) is the same as the processor cycle. Since the L1 cache SRAM macros often dominate the critical path of high-end processors [110], we can make two key observations: First, caches inside the L-NUCA tiles will have lower size, or lower associativity, or lower number of ports, or a combination of all previous conditions than their corresponding L1[2]. Second, L-NUCA network delay will be constrained to the remainder of the cycle after the cache access completes.

The organization of the cache inside tiles does not differ from the implementation of conventional caches as shown in Figure 4.1. The valid, LRF, and dirty bits are stored in flip-flops to enable back-to-back request injection. One-cycle tile latency requires single-cycle loads and stores. The data array is simultaneously read with the tags, and in case of hit the valid way is routed to the transport network through the data_out wires. With stores, data can only be written into the data array if the block is present in the cache; hence, the store data is saved in the write buffer, and then written in the first non load search cycle or served to

[2]Also tile cache control is simpler because tiles only provide whole blocks and not precise words as L1 caches.

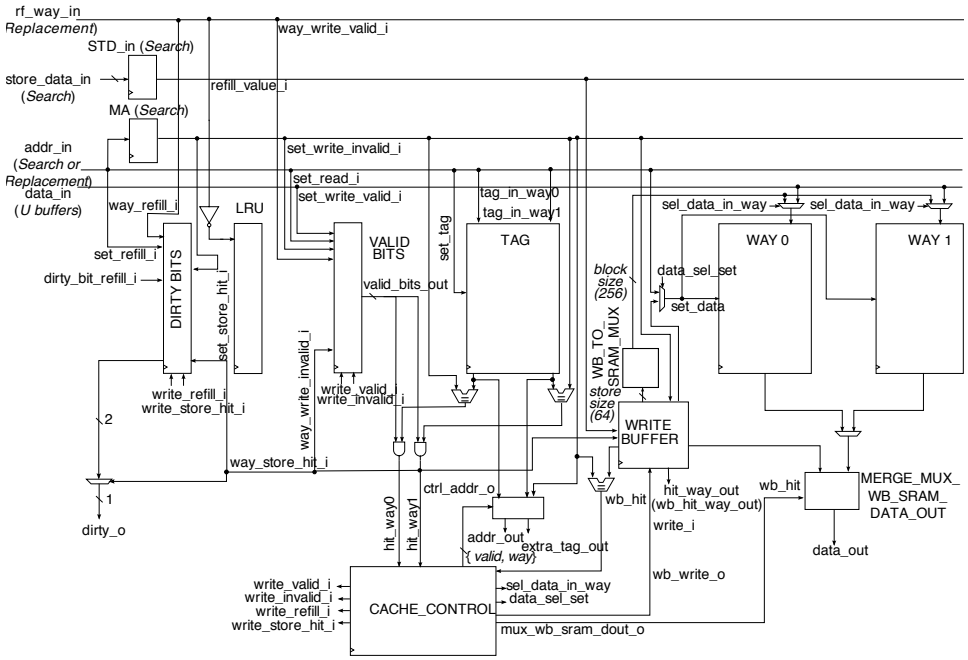



**Figure 4.1:** Cache main components. Many wires have been removed for the sake of clarity, and signals arriving from the left correspond to the Search Network

the processor in case a later load request hits in the write buffer that require an special handling because as for store hits, data has to be aligned.

Many architectures support multiple store sizes; e.g., the Alpha ISA includes four store instructions: `STB`, `STW`, `STL`, and `STQ` with 1, 2, 4, and 8 bytes operands, respectively. When the r-tile is write-through, the rest of tiles require to align the store data within the cache block when the write operation occurs. The circuit that handles this operation is `WB_TO_SRAM_MUX` in Figure 4.1. Its output is connected to both ways because WEB (write enable low) signals enable in which way the data is written. Figure 4.2 depicts a simplified version of this circuit, supporting 1, 2, 4, and 8 byte operands as input and 8 byte cache lines. For larger cache lines the connections are replicated in 8-byte blocks. For example, for 32-byte lines, the input bytes 0 and 3 will be connected to every output and to output bytes 3, 7, 11, 15, 19, 23, 27, and 31.

Store search operations require the previous aligner, but load hits in the same block that a previous non-complete store hit need also a similar circuit merging the outputs of the data ways and the write buffer content. Task performed by the `MERGE_MUX_WB_SRAM_DATA_OUT` block. This circuit is on the critical path of the cache, so to reduce cache delay the circuit has been replicated and placed before than the way selection multiplexer. Thus, the merging is done in parallel with the tag comparison. We observed from synthesis that the area overhead was very small, but the reduction in latency was noticeable.

Input Search Network signals are directly connected to the SRAM macros

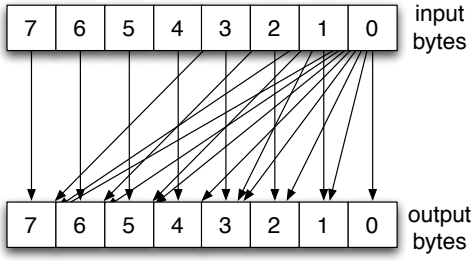



**Figure 4.2:** Aligner/Merger circuit located at the output of the data array

of the cache (`TAG`, `WAY0`, and `WAY1`) because they are synchronous. Figure 4.3 shows how cache control signals are generated in the previous cycle of the tile access with information from both the tile itself and the parent tile in the Search network in order to avoid an extra cycle. In this way, the MAR and SRAM blocks are accessed in parallel and the MAR is used with the cache tag comparators as depicted in Figure 4.1. To reduce the tile control activity, the output enable (OE) signals are left high. The WEB signals of the tag and data array are controlled in a different way because they can be set to low either for a refill (both tag and data array) and a store hit (only data array).

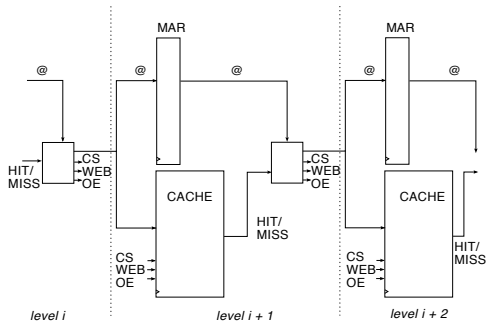



**Figure 4.3:** Tile search control signals propagation with synchronous SRAM macros

Moving from the cache to the Networks-in-Cache we focus on the Transport routers that are serially accessed after caches, so their delay is critical. Also, they need to route messages among links evenly to avoid congestion and allocate channels fairly to the requesters to prevent starvation. Their delay is divided between two components: the allocator and the switch. The first assigns outputs to requests, and the second routes the granted inputs to the outputs.

Figure 4.4 shows the basic organization of the transport switch allocator. Request signals are represented by the "r_" prefix, and dc, r0, r1, t0, and t1 correspond to the cache, replacement, and transport buffers, respectively. `Avail_out0` and `avail_out1` represent the on/off control signals from the destination transport buffers. Allocation and routing priorities are managed with a round-robin policy in order to minimize the allocator area (`RR_last_out` and `RR_last_in`

keep track of the last output link and service input employed, respectively).

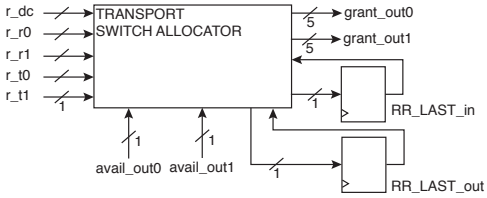



**Figure 4.4:** Transport switch allocator diagram

The complexity of the allocator can be reduced based on two architectural observations. First, since cache content among tiles is in exclusion, the lookups in the cache and in the two replacement buffers always return zero or one hit. In other words, only one signal among `r_dc`, `r_r0`, `r_r1` can be active in the same cycle; therefore, only 3 requesters can be active for the two outputs at maximum. Second, the allocator prioritizes Search (younger) requests that hit in the tile (cache or replacement buffers) over transport buffers (older requests). Otherwise, it would require an extra buffer for the hit and another input port in the allocator to route the hit in a subsequent cycle. This priority scheme does not starve requesters because in the worst-case scenario, older requests will proceed when the Miss Status Holding Register becomes full. At that time, the insertion of new misses into the L-NUCA is stopped.

Output grants (`grant_out0` and `grant_out1`) are 5-bit one-hot vectors to remove decoding in the control switch. Besides, in this way, grant signal bits are almost directly driven by request signals, and the critical path from `r_dc` to `grant_out`*X* only requires 3 gate levels with a maximum delay of 0.36 ns. As we will see in next section, this delay is short enough to proceed in parallel with the data array access.

## 4.3 VLSI Implementation

In this section we present a VLSI implementation of the L-NUCA concept in order to evaluate three main aspects: complexity, feasibility of wide links, and overhead introduced by the Networks-in-Cache. To do so, we focus on the most common tile, further analyzing its area, power consumption, and timing.

In each tile, we consider a two-way set associative 32KB cache with a block size of 32B and LRF (least-recently-filled) replacement. LRF works as follows: when a tile receives an evicted block, it evicts the oldest block in the same set or the least recently filled one. Parallel access to tag and data arrays is employed to shorten the critical path delay. This is a configuration commonly seen for a first level cache in the high-end embedded market, except for the associativity, which is usually higher in commercial caches; e.g., 4 ways in the IBM PowerPC 476FP and 8 ways in the Freescale e500mc/e5500 [49, 47].

The design flow employed the following tools: Synopsys Design Compiler (synthesis), Cadence SOC-Encounter (placement and routing), and Mentor Graphics ModelSim (simulation). The technology library is a low-power 90 nm technology available to European Universities, featuring 9 metal layers and 1 V supply voltage.

L-NUCAs of any size can be built with two tile types, differing only in the transport network. One type is employed in the middle column (3 input and 1 output transport links), and the other one in the rest of tiles (2 input and 2 output transport links). In both tile designs, most connections match when tiles are placed side by side. Therefore, building an L-NUCA with the desired number of levels is very easy. Since both tile types involve the same design complexity, in this Section we focus on the second and the most common one. Table 4.2 shows a summary of placement and routing statistics, and Figure 4.5 a layout of a tile located at the left of the r-tile. Note that the layout for a tile to the right of the r-tile can be obtained by mirroring its left counterpart.

**Table 4.2:** Placement & Routing summary

| | |
|---|---|
| Clock freq. | 333 MHz |
| IO pins | 2498 |
| Gate count | 396516 |
| Gate area | 0.9326 $mm^2$ |
| Macro area | 0.7048 $mm^2$ |
| Total area of Core | 1.8797 $mm^2$ |
| Aspect ratio | 1.0 |

The Networks-in-Cache require 2458 data pins and 40 more to enable permissions and flow control. Each side of a tile contains up to 677 pins routed in M3 and M4 layers. To place all these pins is not an issue because there is room for more than 2000 pins with 0.6 $\mu$m pin pitch in each side of a tile.

For the cache memory, we chose single-port SRAM macros with 512 entries of 64 bits, which can be written at byte granularity. The two cache ways are mapped onto 8 SRAM macros, whereas the whole tag directory fits in a single one (assuming 43-bits addresses, a tag has 29 bits and there are still three free bits to implement an hypothetical coherence protocol). The SRAM macros occupy the largest portion of the tile. The replacement and status bits (LRF, dirty, and valid) are stored in separate flip-flops. This allows trivial multi-ported access to the tags, for example to invalidate a block sent to the r-tile in the previous cycle while simultaneously probing another access this cycle.

During floorplaning, only the SRAM macros were placed and the tool chose the location for the rest of modules. For example, the Write Buffer was placed close to the data ways to minimize cache and tile delay.

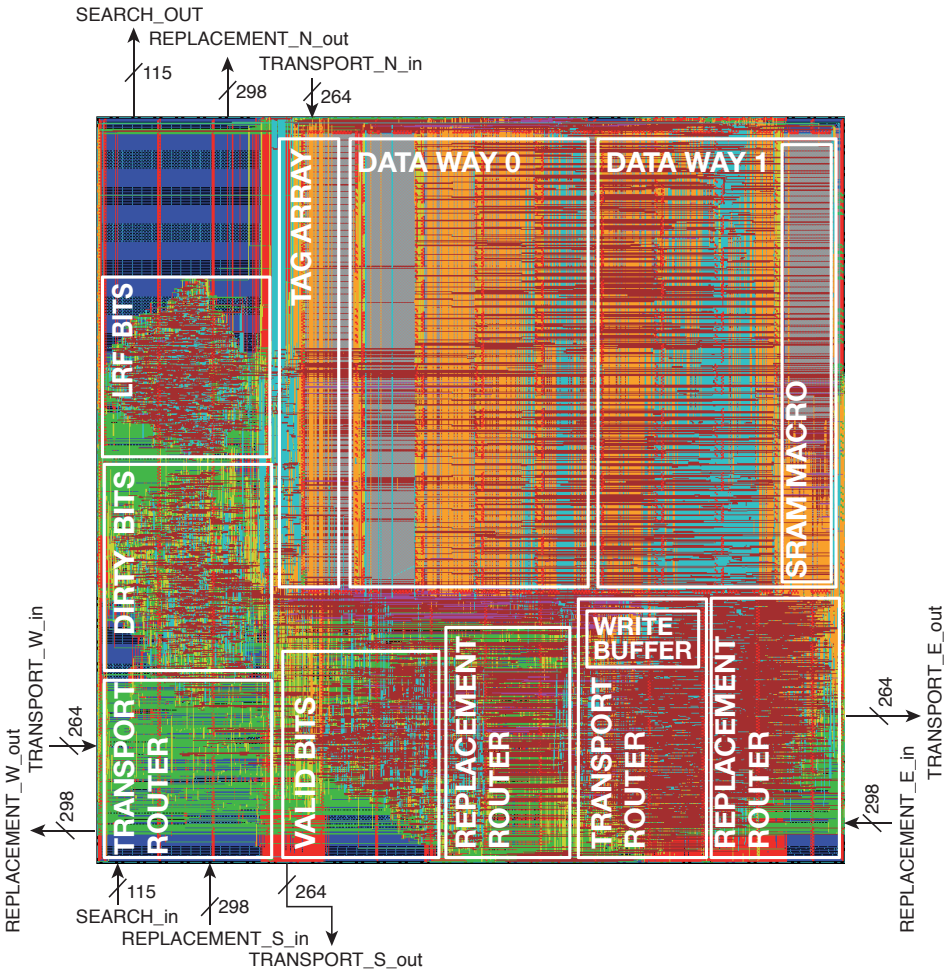



**Figure 4.5:** L-NUCA tile layout with pin placement

### 4.3.1 Area

Figure 4.6 shows the area breakdown by module. The 9 SRAM macros of the data and tag array dominate, accounting respectively for 68% and 8% of the total. The write buffer, valid, lrf, and dirty bits occupy 10%.

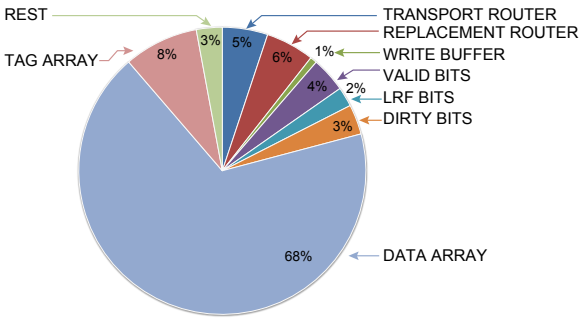



**Figure 4.6:** Main modules area distribution

The Network-in-Cache overhead is small, representing only 11% of the total area (5% for the Transport and 6% for the Replacement network). Even though the Transport router includes the output switch, the Replacement router area is larger because its buffers are bigger and it contains 4 address comparators and 4 write mergers for handling load and store hits in the R buffers.

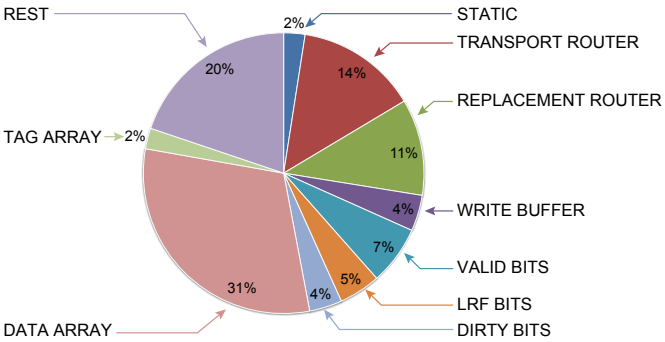



**Figure 4.7:** Statistical power distribution

### 4.3.2 Power

The global static power and the dynamic power for each main module is reported in Figure 4.7. The pie chart comes from a statistical-based power distribution assuming an activity factor of 0.2 at the maximum operation frequency and typical case conditions. The total average power is 32.3 mW, comprising 22.67 mW switching power, 8.89 mW internal power, and 0.7 mW leakage, which represents only 2% of the total thanks to the used low-power technology.

The cache memory access represents most of the dynamic consumption (53%, including data and tag arrays, LRF, status bits and write buffers), while the Replacement and Transport networks consume only 25%. The cache control logic is mostly responsible of the Rest piece that also includes a small fraction due to the Search network (miss propagation to the child tiles). This breakdown implies that in order to reduce the overall L-NUCA power it is necessary to reduce the number of tile accesses triggered by r-tile misses. Section 4.5 presents two techniques aimed at this objective.

### 4.3.3 Timing

As expected, the longest paths are two: evicting the victim block through the Replacement network, and sending the hit block to the transport buffer of a neighbor tile. Figure 4.8 shows the second one, which is the critical path.

The critical path closed timing at 3 ns with 0.116 ns of slack. Data delay involves SRAM reading followed by write merging, while tag delay involves SRAM reading followed by address comparison. The data array delivers a block 2.7 ns after the rise of the clock, while the hit/miss signal is available 0.5 ns earlier (2.7-2.19). The early arrival of the hit/miss signal enables switch allocation to occur in parallel with the data access. After allocation, traversing the switch and writing the transport buffer requires 0.173 and 0.011ns, respectively, accounting for less than 10% of the cycle time.

So, from a practical standpoint, we conclude that the proposed Networks-in-Cache enable both cache-access and single hop routing in one processor cycle.

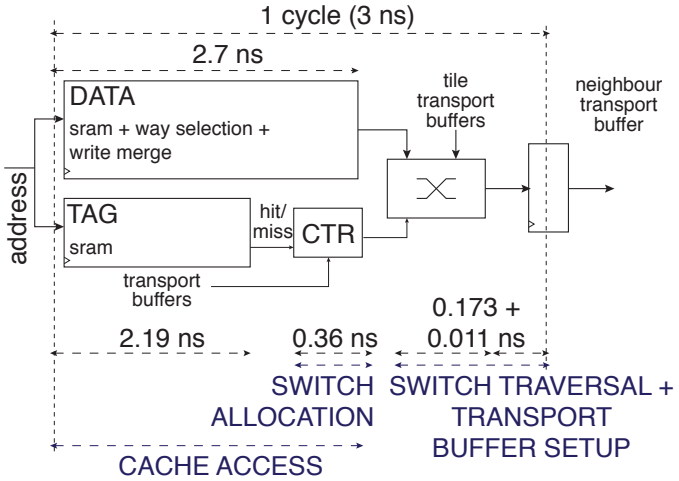



**Figure 4.8:** Tile Critical Path

## 4.4 Performance Comparisons

In this Section, we describe our detailed simulation environment and compare the performance of the implemented L-NUCA cache with an L2 cache representative of the high-end embedded marked and a Static NUCA (S-NUCA) [77] as a low-power alternative.

### 4.4.1 Experimental Framework

We have employed the SMTScalar framework. As a sample of a high-performance embedded processor, we simulate an out-of-order core with a 2-way pipeline similar to the IBM/LSI PowerPC 476FP [49, 89, 65]. The processor includes a reorder buffer, three issue windows (IW) for integer, floating point, and memory instructions, speculative wake-up support and selective recovery, one-cycle register file stage, accurate timing models for non-blocking caches, write buffers, buses, network contention, flow control, and request arbitration.

Table 4.3 summarizes the simulation parameters for the reference processor and cache hierarchy, which includes the same L1 and L3 caches and either a conventional L2, a S-NUCA, or an L-NUCA. All caches use LRU replacement except L-NUCA that employs LRF, least-recently-filled, and they all have a single read/write port.

Besides the 512 KB reference L2, and the 448 KB L-NUCA, we add to the comparison a 512 KB S-NUCA because it does not migrate blocks and may offer better energy results [77].

The S-NUCA has 4 banks of 128KB organized in a 2 by 2 2D-Mesh. This configuration was selected after exploring the S-NUCA design space. Larger number of banks do not improve performance because network overhead becomes very high even if the latency of the wormhole router is set to 1 processor cycle. Bidirectional links were doubled from the original 16 bytes to 32 to match the size of L-NUCA transport links.

Unfortunately, the latency and the initiation rate of the PowerPC 476FP L2 and L3 caches are not available, so we estimate them with the High Performance model of Cacti 6.5 [101]. The latency assumed for the reference L2 is optimistic. For example, the dedicated L2 of the ARM Cortex-A8 has a cache latency of 6 cycles—50% larger than the L2 used in the comparisons [44]. Additionally, for the L2 cache we assume a serial access mode, where the tag array is accessed before the data array similar to the one used by the L2 cache of PowerPC 476FP [65]. Using a parallel access L2 cache does not improve the performance significantly when compared to the serial one. We have computed with Cacti 6.5 the latency and the initiation rate of both a serial and a parallel access L2 cache. A parallel access cache reduces the latency from 4 to 3 cycles, while it increases the initiation from 2 to 3 cycles. This behavior translates to a 0.6% IPC improvement that does not payoff for the increase in the initiation rate and the 19% and 4.5% increased dynamic and static energy consumption, respectively.

**Table 4.3:** Simulator Micro-architectural parameters. BS, AM, lat, and init stand for block size, access mode, latency, and initiation rate, respectively

| | |
|---|---|
| fetch/decode width | 2 |
| branch predictor | bimodal + gshare, 16 bit |
| ROB/LSQ entries | 32/16 |
| INT/FP/MEM IW entries | 8/8/8 |
| Clock Frequency | 333 MHz |

| | |
|---|---|
| issue width | 2(INT+MEM) + 2 FP |
| commit width | 2 |
| r-tile/L-NUCA/L3 MSHR entries | 8/8/4 |
| MSHR secondary misses | 4 |

| | |
|---|---|
| Store Buffer size | 8 |
| L3-Coalescing WB size | 4 |
| TLB miss latency | 30 |
| Miss-predicted branch delay | 6 |

Reference L2

| | |
|---|---|
| L1 | 32KB, 4 Way, 32B BS, write-through, 2-cycle lat, 1-cycle init |
| L2 | 512KB, 8 Way, 32B BS, serial AM, 4-cycle lat, 2-cycle init, copy-back |

S-NUCA

| | |
|---|---|
| L1 | 32KB, 4 Way, 32B BS, write-through, 2-cycle lat, 1-cycle init |
| S-NUCA | 2×2 128KB, 2 Way, 32B BS parallel AM, 3-cycle lat, 3-cycle init, copy-back |

L-NUCA

| | |
|---|---|
| r-tile | 32KB, 4 Way, 32B BS, copy-back and write-around 2-cycle lat, 1-cycle init |
| tiles | 32KB, 2 Way, 32B BS parallel AM, copy-back, levels: 3, total size: 448 KB |

| | |
|---|---|
| L3 cache | 4MB eDRAM, 16 Way, 128B BS, 14-cycle lat, 7-cycle init, copy-back |
| Main memory | 100 cycles/4 cycle inter chunk, 16 Byte wires |

The simulator was also used to validate the HDL description of the L-NUCA tiles. We extracted the memory references from the simulator and fed the Verilog simulations to verify the outputs were equal. The traces were up to 150,000 references long.

The workload selection includes benchmarks from SPEC CPU2000 and CPU-2006 [53, 54], focusing on those related to the embedded domain: gaming, image recognition, or compression. The subset includes 13 benchmarks, 8 integer and 5 floating point from 3 different languages, as shown in Table 4.4. Table 4.4 also includes Misses Per Kilo-Instruction (MPKI) for the L1 cache shown in Table 4.3. MPKI is a useful metric that allows the architect to get an insight on which applications LP-NUCA is expected to show its full potential. Those with a higher L1 MPKI fit better to LP-NUCA. Simulations consist of a 100 million instruction run after a warm-up of 200 million instructions following the SimPoint guidelines [50].

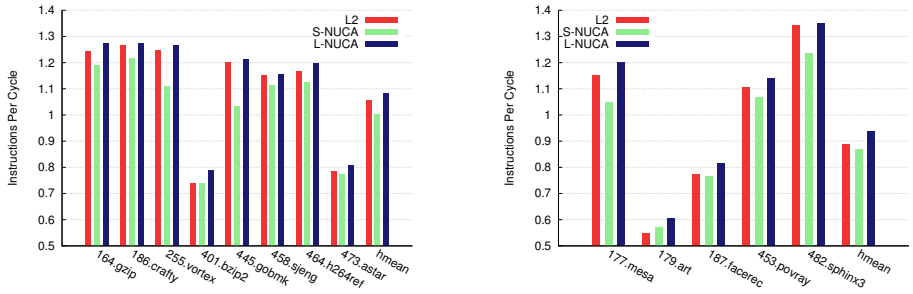
**Table 4.4:** Benchmark selection. The L1 misses per kiloinstruction correspond to the L1 caches of Table 4.3

| | Name | Lang. | Description | L1 MPKI |
|---|---|---|---|---|
| *CINT2000* | 164.gzip | C | Compression | 20.08 |
| | 186.crafty | C | Game Playing: Chess | 3.90 |
| | 255.vortex | C | Object-oriented Database | 8.34 |
| *CINT2006* | 401.bzip2 | C | Compression | 45.20 |
| | 445.gobmk | C | Artificial Intelligence: Go | 22.88 |
| | 458.sjeng | C | Artificial Intelligence: chess | 4.70 |
| | 464.h264ref | C | Video Compression | 65.79 |
| | 473.astar | C++ | Path-finding Algorithms | 27.90 |
| *CFP2000* | 177.mesa | C | 3-D Graphics Library | 19.40 |
| | 179.art | C | Image Recognition / Neural Networks | 143.42 |
| | 187.facerec | F90 | Image Processing: Face Recognition | 53.88 |
| *CFP2006* | 453.povray | C++ | Image Ray-tracing | 30.13 |
| | 482.sphinx3 | C | Speech recognition | 3.56 |

### 4.4.2 IPC and AMAT Comparison

Figure 4.9 shows the IPC for all benchmark under test and the harmonic mean (hmean) of Integer and Floating Point for the reference L2 cache, S-NUCA, and L-NUCA.

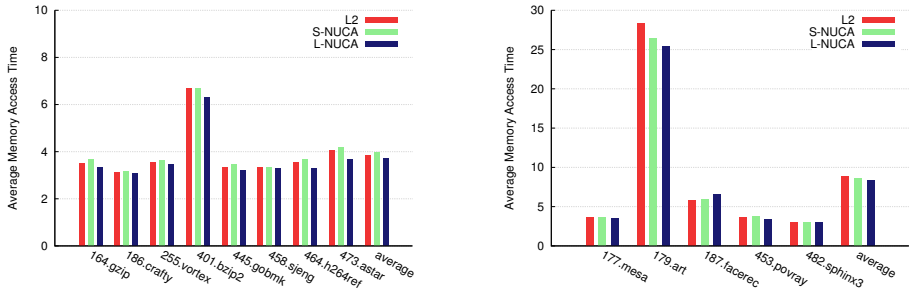
L-NUCA performs better than L2 and S-NUCA regardless of the benchmark group. In Integer benchmarks, L-NUCA provides 2.5% and 7.7% IPC benefit over the L2 and the S-NUCA, respectively. Floating Point benchmarks follow the same trend with 5.7% and 8.1% L-NUCA improvements over L2 and S-NUCA. Individual results does not differ from the mean, and L-NUCA ranks the first in all of them. Network overhead is the cause behind S-NUCA results. As shown in Section 3.2.1, S-NUCA network mechanism was designed for large last level multimegabyte caches, and the 3 required hops per access (on average) double the effective cache latency.




**(a)** Integer **(b)** Floating Point

**Figure 4.9:** IPC comparison of L2, S-NUCA and 3-level L-NUCA

L-NUCA gains are larger in Floating Point benchmarks because their average L1 MPKI doubles that from Integer ones. Relative to L2, L-NUCA improvements are consistent among benchmarks ranging between 0.5% (458.sjeng *CINT2006*) and 10.3% (179.art *CFP2000*). Please note that 458.sjeng and 179.art are among the benchmarks having the lowest and highest L1 MPKI, as shown in Table 4.4.




**(a)** Integer **(b)** Floating Point

**Figure 4.10:** AMAT comparison of L2, S-NUCA, and 3-level L-NUCA

Figure 4.10 shows the Average Memory Access Time (AMAT) for all configurations and benchmarks. Although AMAT combines miss rate, miss penalty, and hit latency, it should be analyzed with care in out-of-order processors because AMAT does not reflect the latency overlap among misses [52]. When computing the AMAT we only include the committed loads. In this way, the minimum load-to-use latency, which reflects the time between a load issues and the requested data is ready for dependent instructions, is averaged. AMAT reveals two interesting facts: First, S-NUCA, despite having a lower AMAT has an IPC worse than L2. Two benchmarks cause this effect: 482.sphinx3 and 177.mesa. IPC decreases 8% and 9.2% for 482.sphinx3 and 177.mesa because the write buffer of the S-NUCA is filled during a larger fraction of time, stalling the commit of stores even though loads complete faster. Second, L-NUCA exhibit the lowest AMAT in all

benchmarks but one, 187.facerec, and again stores are the cause. To summarize, L-NUCA reduces AMAT about 5% relative to L2 and S-NUCA in both benchmark groups.

The IPC and AMAT gains offered by L-NUCA mostly come from hit and miss time reductions than from miss rate decrements. In Integer programs, S-NUCA and L2 have a miss rate of 2.9% while L-NUCA has 3%, and in Floating Point programs, L2, S-NUCA, and L-NUCA experiment a miss rate of 17.9%, 15.8%, and 15.2%, respectively.

AMAT differences among configurations are larger than their IPC counterpart because our processor model exploits memory level parallelism with its out-of-order core and the large Miss Status Holding Registers. With simpler in-order cores, gains would be larger.

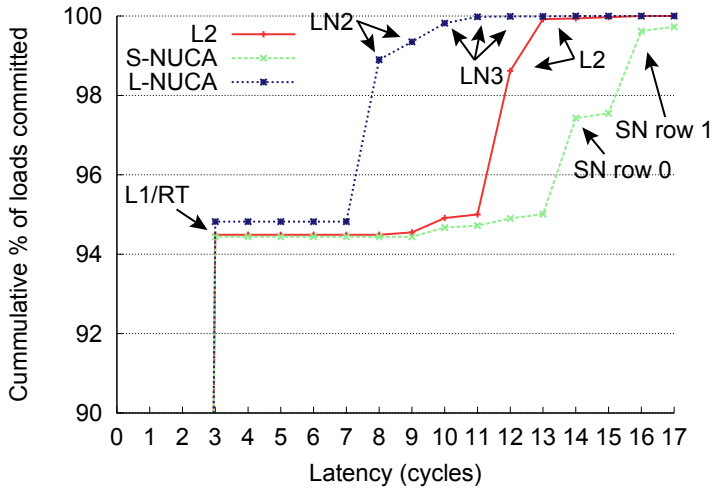



**Figure 4.11:** Cumulative load-to-use latency distribution for 464.h264ref. LN2 and LN3 refers to hits in the 2nd and 3rd L-NUCA levels

To get deep insight into the results, Figure 4.11 shows the cumulative load-to-use latency distribution of a representative benchmark, 464.h264ref. The first remarkable point is cycle 3 representing the L1/RT (r-tile) load-to-use latency, 1 cycle to compute the address plus 2 L1 cache cycles. L1/RT hit in more than 94.5% of accesses, values differ in 0.35% due to the different execution ordering that results from receiving L1 misses at different latencies. The next "flat" cycles correspond to the cost of the next cache level latency plus the re-issue cost. On one hand, if we focus on the L2 line, L2 hits are not ready for dependant instructions until cycle 12 and successive[3]. Hits between cycles 9 to 11 correspond to L1 secondary misses that do not experiment the whole L2 latency. Continuing with the SN line, hits are serviced in cycle 14, two more than the L2. The S-NUCA

[3]L1 cache misses expend 1 cycle in the L1 MSHR, then another cycle in the L2 arbiter (cache port arbitration among reads, writes, and refills), 4 more cycles in the L2 cache, 1 cycle between the L1 MSHR and the IW, and 2 more to issue the refill and the dependant instructions, if any.

bank latency is 1 cycle less than the L2 but the 2D-mesh adds two routing cycles. The next row of banks, row 1, requires another two extra cycles resulting in a 16 cycle latency. On the other hand, LN2 tile hits are serviced in cycle 8[4]. So in this cycle, 8, 94.5% of loads have completed in the L2 and S-NUCA cases, while 98.9% in the L-NUCA. This 4.1% improvement reduces AMAT from 3.55 and 3.66 to 3.28 cycles and increases IPC 2.46% and 6.3% regarding L2 and S-NUCA.

Finally, the use of embedded DRAM in the LLC matches up previous reported gains for conventional and D-NUCA LLCs [124]. Therefore, we can conclude that L-NUCA provides performance improvements no matter the LLC organization because it exploits the short term temporal locality.

## 4.5 Low-Power Enhancements

This section first describes Light-Power NUCA (LP-NUCA), the improved low-power L-NUCA. Next, it quantifies its dynamic energy savings with estimations from the 90 nm VLSI implementation. Finally, in two possible 32 nm application domains, it compares LP-NUCA energy consumption and energy-delay against an L2 cache, representative of the high-end embedded market, and a Static NUCA [77], as another low-power alternative.

Reducing dynamic energy consumption in L-NUCAs requires lowering the number of tile accesses. L-NUCA can do this *reactively* by recognizing and stopping unnecessary activity already in progress, and *proactively* by trying to minimizing activity before it starts. We present one reactive technique, Miss Wave Stopping, and another proactive, Sectoring. They both have low complexity and take advantage of the network organization to save energy without degrading performance.

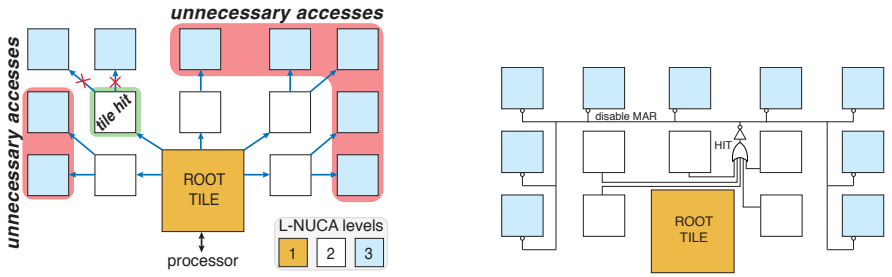
### 4.5.1 Miss Wave Stopping

One source of energy waste is the unnecessary lookups performed by the search operation after a hit in a lower level. In the base L-NUCA, tiles do not share hit information, and they only stop miss waves to their child tiles. For example, Figure 4.12(a) shows a hit in a level-2 tile. Seven of the nine level-3 tiles look for a block that they surely do not have (because data is exclusive across all tiles and one tile has already hit). These extra lookups can be avoided if tiles propagate the search request only when all the tiles of a level miss. Fortunately, as shown in Figure 4.8, hit/miss signals are ready after around two thirds of the tile latency. The remaining time can be used for computing a wave-stop signal by "ORing" all the hit signals of a level. Figure 4.12(b) shows a 3-level LP-NUCA with a hit detection gate in level 2. When the wave-stop signal is set, all the level-3 tiles

[4]RT misses expend one cycle in the LN2 tiles and the RT MSHR, then another cycle to notify the RT controller and the MSHR, one more cycle between the RT MSHR and the IW, and 2 more to issue the refill and the dependant instructions, if any.

discard the search operation. Note that this mechanism does not interfere with the global miss detection, which sends a miss to the next cache level only when all the last-level tiles miss.

At first glance, Miss Wave Stopping seems to reduce the modularity and scalability of L-NUCA because it introduces wires that connect multiple tiles. Nevertheless, their length—and delay—scales when technology shrinks [58], and increasing the number of levels beyond 4 does not get additional benefits [124]. So this technique does not increase the complexity of the L-NUCA implementation. Finally, we have verified from synthesis that Miss Wave Stopping is viable at least for 5-level LP-NUCA.




**(a)** Extra accesses in the 3rd level

**(b)** Hit detection gate between 2nd and 3rd levels

**Figure 4.12:** L-NUCA unnecessary accesses after a second level hit and LP-NUCA with a level-2 hit detection gate

Note that this technique cannot affect performance because the number of hits is the same and only unnecessary accesses are removed.

### 4.5.2 Sectoring

Sectoring is a proactive technique limiting the activity in the Search and Replacement networks by restricting block placement. One end, a cache block can be mapped in a fixed unique NUCA tile, such as in Static NUCA [77]. The other end, blocks can be mapped in any tile such as L-NUCA.

Sectoring is in between and maps blocks to clusters of tiles, sector, based on the Search network branches. The idea is similar to the D-NUCA sparse sets [77], but differs in that if $n$ sparse sets share a single bank, then all cache banks are $n$-way set associative while the L-NUCA sectors do not impose any restriction on the cache configuration. Blocks are mapped to sectors based on a hash function of their addresses. As a simple and efficient hash function, we use that of the skewed-associative caches [116].

Figure 4.13 shows a 3-level sectored LP-NUCA with 4 sectors. When the r-tile misses, it hashes the block address to obtain a 2-bit sector number. Depending on the value, the search request is propagated to all the second level tiles (sectors 01 and 10), or only to the two tiles on the left (sector 00) or on the right (sector 11).

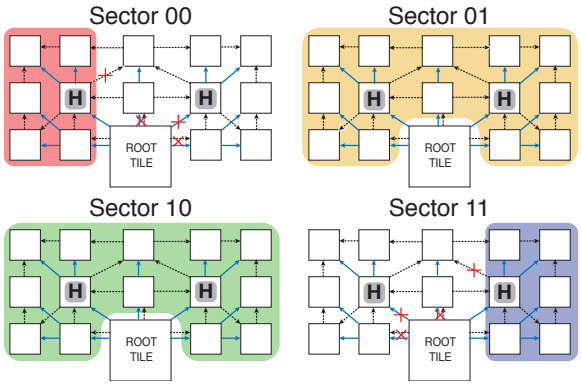



**Figure 4.13:** A 3-level sectored LP-NUCA. For each sector, red crosses mark the disabled links in the Search and Replacement networks; "H" tiles have replicated the r-tile hash function to ensure blocks do not leave their sector.

The miss wave propagation proceeds as usual except in the tiles marked with "H", which compute again the hash function and forward the miss request according to the sector number. In this way, sectors 00 and 11 will experience at most 6 lookups instead of 14. The r-tile also evicts blocks according to sector number.

Sectoring changes global associativity. For instance, in the implemented 3-level LP-NUCA, we depart from the homogeneous 28-way set associative structure (tile associativity × number of tiles) to end up with different associativities: 28 and 12 for sectors 01-10 and 00-11, respectively. In any case, the associativity is still big, and the key factor for maintaining performance is to equally evict blocks among sectors. Assuming the hash function evenly distributes blocks among sectors, the eviction policy has to equalize the traffic among the three replacement links: all 00-blocks leave the r-tile through its West replacement link (West-rl), all 11-blocks through its East-rl, and for the remainder 50% blocks, two thirds are evicted through the North-rl and one third is halved between West-rl and East-rl. The extra hardware for this technique is only a counter and two 3-bit flip-flops instead of the current 2-bit round-robin flip-flop.

Finally, for multiported root tiles, sectoring offers the additional advantage that two simultaneous misses to sectors 00 and 11 can enter the LP-NUCA at a time.

### 4.5.3 Energy Estimation in 90 nm

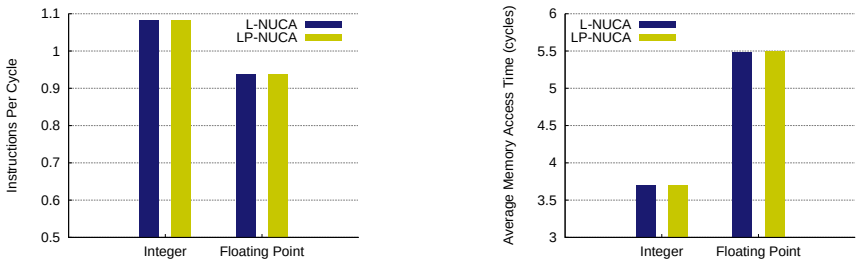
Tile energy consumption is estimated by extracting the post-layout netlist and simulating the tile behavior for at least 1024 consecutive operations with random data. Operations include: read hit, read miss, write hit, write miss, fill with eviction, fill without eviction, and transport. Then, for each tile and each operation, the average energy per operation is multiplied by the corresponding tile activity counter, obtained previously with the microarchitectural simulator.

As previously stated, sectoring is an effective technique when it is able to distribute blocks evenly among sectors and reduce tile accesses. Table 4.5 shows the total number of tile accesses for a 3-level LP-NUCA with sectoring and an L-NUCA without it. Accesses are reduced by 36.5% and 35.3% for integer and floating point, respectively. More important, the hash function distributes blocks almost perfectly for integer benchmarks—a perfect hashing would be 37.5, 25, and 37.5—and a bit worse for floating point, where 45.3% of all searches proceed through the East channel. We have observed that results for different numbers of levels are quite similar.

**Table 4.5:** Sectoring impact on traffic distribution for the whole workload in a 3-level LP-NUCA

| | L-NUCA | | | | | | LP-NUCA (with sectoring) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Int | | | FP | | | Int | | | FP | | |
| # of tile accesses | 250.7M | | | 311.8M | | | 159.1M | | | 201.8M | | |
| r-tile channel | W | N | E | W | N | E | W | N | E | W | N | E |
| search requests % | 100 | 100 | 100 | 100 | 100 | 100 | 37.0 | 25.0 | 37.9 | 34.0 | 20.8 | 45.3 |

Miss Wave Stopping is effective because most LP-NUCA hits occur in the second level tiles, namely, 78% and 62% for Integer and Floating Point programs. As a consequence, most time only 5 of the 14 tiles perform any activity, and these 5 tiles ensure that the network consumption is low: requests do not propagate to all fabric (Search), hit service only requires one or two hops (Transport), and the empty ways leaved by these hits are almost immediately filed with r-tile evictions stopping the domino effect (Replacement). Finally, although the number of requests to the second level caches is small—on average the r-tile injects a request in 3 out of 100 cycles—its service latency is crucial in performance because out-of-order cores tolerate the latencies of first level caches but not those of the second one.




**(a)** Instructions Per Cycle **(b)** Average Memory Access Time

**Figure 4.14:** Performance comparison of L-NUCA and LP-NUCA

Figure 4.14 shows that LP-NUCA maintains L-NUCA performance level in both type of benchmarks. Neither Miss Wave Stopping nor Sectoring hurt performance. On one hand, Miss Wave Stopping cannot affect performance by

construction. On the other hand, Sectoring causes a negligible hit ratio reduction, but the extra latency is hidden by out of order execution.

As far as the power consumption of our design is concerned, Figure 4.15 shows the results for a baseline L-NUCA (L-NUCA), an L-NUCA enhanced with Miss Wave Stopping (MWS), an L-NUCA enhanced with Sectoring (Sect), and an LP-NUCA that comprises both techniques (LP-NUCA). The plot represents the sum of the energies spent for all the benchmarks. Each bar represents the leakage consumption stacked over the dynamic, which is the target of our techniques.

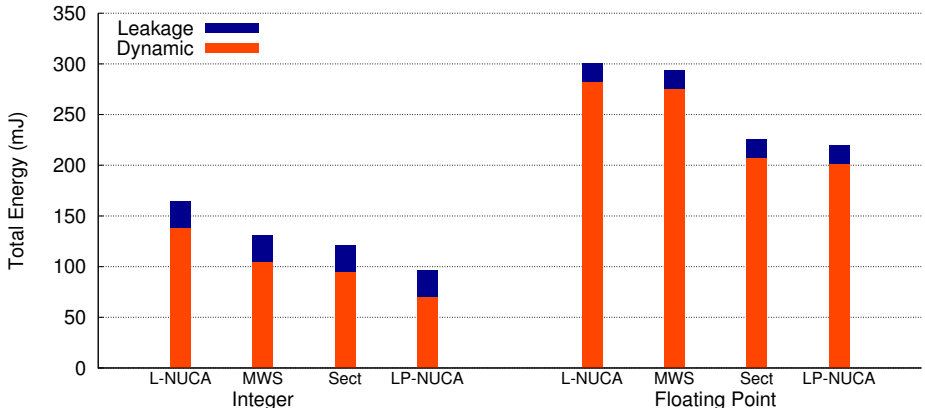



**Figure 4.15:** LP-NUCA energy savings in 90 nm. L-NUCA represents the original organization, MWS and Sect add Miss Wave Stopping and Sectoring, respectively, and LP-NUCA combines both

MWS is effective whether it applies to L-NUCA or Sectoring. Relative gains in dynamic energy are very similar in both cases with reductions ranging between 25.9% and 2.6% for integer and floating point benchmarks, respectively. It is more effective for integer programs since they have a hit ratio larger than floating point and this technique is useless for misses. Nevertheless, the absolute energy value saved by MWS in L-NUCA is higher than that in Sectoring: 40.4 vs. 31.4 mJ, respectively for integer and floating point benchmarks together. This is because in Sectoring, MWS only avoids lookups in two tiles for sectors 00 and 11, instead of the seven tiles avoided in L-NUCA.

With respect to Sectoring, it can be seen that it saves more energy than Miss Wave Stopping, whether it applies to L-NUCA or MWS. Again, the relative gains in dynamic energy are very similar with reductions ranging between 32.9% and 26.5% for integer and floating point benchmarks, respectively. Sectoring saves more energy than Miss Wave Stopping for two reasons. First, it reduces the energy in two operations: search (lookup is performed in fewer tiles), and replacement (blocks pass through fewer tiles to leave the LP-NUCA), while Miss Wave Stopping only decreases the search energy. Second, Sectoring is effective for both hits and misses in contrast with Miss Wave Stopping, which only targets hits.

If we look separately at dynamic and static energy, two observations can be

made. On one hand, the contribution of static energy is not negligible as one could expect from Figure 4.7. Averaging integer and floating point programs, the static energy represents 9.4% of the total in the L-NUCA configuration because tile activity is not very high. For instance, if we assume a r-tile miss rate of 10%, only one in ten cycles there is dynamic consumption, whereas leakage is active the 10 cycles. Of course, as the dynamic activity decreases, the relative static contribution increases because the execution time is almost constant in all configurations; thus, in the LP-NUCA executing integer benchmarks, we found the maximum static contribution of 26.6%.

Finally, we can consider the full benchmark set (including both integer and floating point) to quantify the impact of Wave Miss Stopping and Sectoring on the total energy consumption. Thus, with respect to L-NUCA, Miss Wave Stopping saves 8.7%, Sectoring saves 25.4%, and their combination, LP-NUCA, rises the savings up to 32.1%.

We do not include any energy measurements regarding the conventional L2 cache and the S-NUCA organization, because we do not have any place & routed designs available for L2 and S-NUCA in our 90 nm implementation technology. Also, mixing real implementation measurements with Cacti estimations at 90 nm is not a good option because our 90 nm technology does not match well with the 90 nm technology assumptions of Cacti. Instead, we chose to perform the energy comparison of the three cache organizations under a common framework employing Cacti and 32 nm technology models. Next, the obtained results along with the technology parameters employed are discussed.

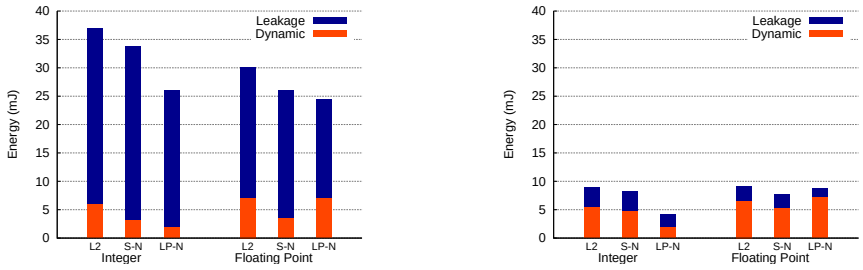
### 4.5.4 Technology Scaling to 32 nm

LP-NUCA offers three main advantages over conventional or S-NUCA second level cache organizations. First, its tiled organization allows the design of any size caches by just replicating the LP-NUCA tiles as many times as needed. The tile's area, delay, and power characteristics, and thus its scaling to smaller technologies are mainly determined by the size and the characteristics of the SRAM banks inside each tile. For acceptable performance, the total SRAM size per tile can be as low as 8 KB [124]. Second, it requires smaller total memory size compared to a conventional or S-NUCA L2 cache implementation in order to perform equally well, in terms of IPC and average memory access time as shown in Figures 4.9 and 4.14. From our experiments, equal performance is achieved when the total size of LP-NUCA is 12.5% less than L2 state-of-the-art. Third, the proposed Networks-in-Cache that connect the LP-NUCA tiles rely on simple customized switches and on local wire connections linking only neighbor tiles. The length of the corresponding wires do not exceed the layout area of the SRAM banks used in each tile. For example, in the 90 nm implementation with 32 KB of SRAM banks the longest inter-tile wires span 1.37 mm.

The two main issues that can arise when scaling to even smaller technologies are the wire delay and the power consumption. As far as wire delay is concerned

LP-NUCA is expected to scale well to smaller technologies, assuming a constant capacity per tile. All inter-tile wires are short, and their length is solely determined by the area of small SRAM banks used per tile. Therefore, if capacity per tile is small enough, LP-NUCA is guaranteed to behave in terms of latency the same as the 90 nm implementation.

Additionally, if we keep the total cache area constant, instead of the the total cache size, which is 26.3 $mm^2$ for a 3-level LP-NUCA at 90 nm (r-tile not included), then the empty space left can be filled with more LP-NUCA levels. In this case, increased size conventional L2 caches have to face some serious issues arising from long wires [58] that may negatively affect their initiation rate. On the contrary, by construction, the initiation rate in LP-NUCA remains constant. The only problem that LP-NUCA may face in a constant-area scaling scenario, is the delay of Miss Wave Stopping. This can be easily solved by limiting Miss Wave Stopping only to the levels close to the r-tile. Although this technique limits the full potential of Miss Wave Stopping, it still offers significant energy savings, since the $2^{nd}$ and the $3^{rd}$ level tiles are responsible of most hits and would continue to stop the miss propagation to higher LP-NUCA levels.




**(a)** mains-powered **(b)** battery-powered

**Figure 4.16:** Energy comparison of conventional L2 (L2), Static NUCA (S-N), and LP-NUCA (LP-N) in the two application domains at 32 nm

The power consumption of LP-NUCA is also expected to scale well in smaller technologies, assuming again a constant capacity per tile and without changing the number of tiles. LP-NUCA would benefit from the smaller capacitances offered by smaller devices, while at the same time its activity reduction techniques would again reduce any unnecessary power consumption. Also, LP-NUCA implicitly saves static power when compared to conventional or S-NUCA caches, since (a) it provides a similar level of performance with lower cache sizes [124] and (b) it reduces the total execution time.

To verify these qualitative arguments, we have simulated an LP-NUCA, a conventional L2, and a S-NUCA at 32 nm using Cacti 6.5, keeping cache sizes constant. At 32 nm there are a lot of options regarding the operating voltage, the transistor's threshold as well as the $I_{on}/I_{off}$ per $\mu$m current ratio offered by the technology [16]. All these parameters can be tuned depending on the application

domain. For high-performance low-power processors, we can identify two main domains: mains-powered and battery-powered. To the first category we find high-end networking systems, storage servers, or set-top boxes requiring the maximum performance with limited power consumption, while the second category covers systems that are designed for delivering desktop-level performance with very low energy consumption such as smartphones, eReaders, or mobile internet devices.

At 32 nm, for high-performance cores the voltage ranges between 0.8 and 1 V, the transistor's threshold is between 0.2 and 0.3 V while the $I_{on}/I_{off}$ per $\mu$m ratio approaches 15 [16, 43, 106]. The model of Cacti 6.5 that best matched these cases is the Low Operational Power (LOP) one [101] after changing the voltage from 0.6 to 0.9 V[5].

Therefore, we employed the modified LOP model in the mains-powered domain, whereas we use the LSTP (Low STandby Power) model of Cacti for the SRAM core and the modified LOP for the rest in the battery-powered domain. In the LSPT model both threshold voltage and $I_{on}/I_{off}$ ratio are increased in order to guarantee that the static (leakage) power is always around 10-20% of the total.

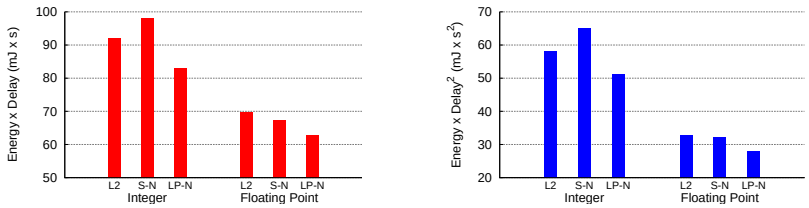
Since Cacti does not model the proposed Networks-in-Cache, in order to estimate their consumption we assume the following: (1) tile capacity (32 KB) remains constant, (2) SRAM, logic transistors, and wires scale equally well. Hence, the relative module consumption does not change with scaling. For example, the transport router consumes 14% of the total in both 90 and 32 nm, and the cost of a read hit in 32 nm is the Cacti's read hit estimation divided by 0.73 that is the percentage of the cache consumption in tiles at 90 nm. Due to scaling, tile and processor frequency increase from 333 MHz to 1.2 GHz and 1 GHz for the mains-powered and the battery-powered domains, respectively. Finally, as a lower bound on the power consumption of the S-NUCA interconnection network we take that of the LP-NUCA transport network.

Figure 4.16 shows the total energy consumption of the three cache organizations for the two application domains at 32 nm. A first observation regarding energy consumption of LP-NUCA implementation at 90 nm, Figure 4.15, and the 32 nm Cacti estimations is that in the mains-powered domain the total energy consumption of LP-NUCA reduces by 3× and 9× for Integer and Floating Point programs, respectively, while the savings grows up to 23× and 24×, respectively, in the battery-powered domain.

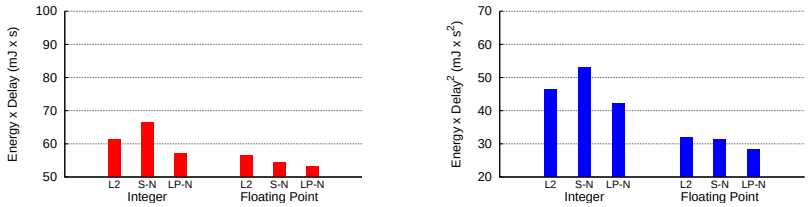
Concerning the energy comparison among caches, LP-NUCA always requires the smallest static power consumption regardless the domain due to the reduction in the cache size and in the execution time. As far as the total consumption is concerned, in Integer benchmarks, LP-NUCA overpasses L2 and S-NUCA with gains ranging between 23% and 111%. In Floating Point programs, LP-NUCA is

[5]When caches operate at a voltage close to 0.5 V special circuit techniques are required, such the migration from 6T to 8T cells, that would guarantee SRAM cell stability. The high performance (HP) model of Cacti is not used in the comparisons since it is far more optimistic regarding the threshold voltage scaling and $I_{on}$ current per $\mu$m relative to current technologies from industry [16, 43, 106]. At the same time, this model significantly overestimates the $I_{off}$ per $\mu$m current.

the best option in the mains-powered domain, whereas S-NUCA is the best in the battery-powered domain. Although LP-NUCA significantly saves dynamic power of more than 30% in the majority of the Floating Point benchmarks there is one benchmark, 179.art, that stresses the energy behavior of LP-NUCA. This benchmark has the highest L1 MPKI and causes a lot of activity in the intermediate cache level. Its dynamic power represents 50% of the total dynamic power in L2 and S-NUCA organizations, while it reaches 88% in the LP-NUCA case. LP-NUCA reacts well to 179.art demands and services more hits that the L2 and the S-NUCA; however, 93% of them are serviced from the last level tiles[6]. Regarding total energy consumption, in both domains, LP-NUCA wastes less energy with gains ranging from 18% to 39%.




**(a)** Mains-powered




**(b)** Battery-powered

**Figure 4.17:** Energy-Delay and Energy-Delay$^2$ for the conventional L2 (L2), Static NUCA (S-N), and Light Power NUCA (LP-N). Both ED and ED$^2$ comprises level 1, L2 or S-N or LP-N, and level 3 cache energies.

Finally, Figure 4.17 compares the three cache organizations using two system level metrics: Energy-Delay (ED) and Energy-Delay$^2$ (ED$^2$). Both metrics have been computed by taking the product of the total energy and execution time for all Integer and Floating Point programs. Please note that the numbers reported in Figure 4.17 also include the energy contribution of L1 and L3 caches. With this addition we want to show that even if LP-NUCA reduces the energy of the second level cache, it still does not increase the energy of the remaining cache hierarchy, offering a complete energy-efficient solution.

Figure 4.17 shows that LP-NUCA behaves well both in the mains-powered and battery-powered domains, where its savings are larger. LP-NUCA gains

[6]On average, each hit in 179.art causes 14 tile look-ups (both tag and data arrays), transport messages of 3 hops on average, and 4 or 5 domino replacements, all of which increases the overall power consumption.

are bigger in Integer programs ranging between 2% (battery-powered, Floating Point relative to S-NUCA) and 21% (mains-powered, Integer relative to S-NUCA). While S-NUCA saves energy with respect to L2, it only improves ED and $ED^2$ in Floating Point programs. Overall, we can conclude that LP-NUCA achieves both better performance and less energy consumption than conventional and S-NUCA caches.

## 4.6 Final Remarks

In this chapter we propose, implement, and evaluate Light-Power NUCA (LP-NUCA), a large distributed victim cache made of small cache tiles. It aims at high-end low-power processors for System-On-Chips and is based on Light NUCA. LP-NUCA leverages three specialized Networks-in-Cache featuring block-wide links for exploiting short-term temporal locality. To prove its feasibility, we have fully implemented a tile in a 90nm standard-cell based technology, drawing several conclusions.

First, it is very easy to build LP-NUCA with different sizes by just replicating two tile layouts, and most connections match when tiles are placed side by side. Managing the different cache operations in separate networks enables low-complexity router designs.

Second, all the Networks-in-Cache proposed in LP-NUCA involve minimum overhead. The area, energy, and delay of LP-NUCAs are dominated by the SRAM macros filling each tile and not by the interconnection itself. In a 32B block size, 32 KB tile, LP-NUCA's low-complexity routing and migration policies require 10% of the tile cycle time, 11% of its area, and consume at most 25% of the tile power.

Third, LP-NUCA introduces two techniques for reducing dynamic energy consumption, namely *Miss Wave Stopping* and *Sectoring*. The former reduces the number of lookups after a hit, and the later restricts block placement and search. In order to evaluate performance and energy we run representative embedded workloads and feed a detailed cycle-by-cycle simulator with the energy costs from the different layout variants in 90 nm and models in 32 nm. Both techniques together decrease one third the dynamic consumption with regards to L-NUCA without degrading performance. LP-NUCA enables both faster execution and lower energy consumption compared to a slightly larger conventional or S-NUCA L2 cache. Hence, LP-NUCA offers a complete energy-delay efficient solution for high-performance low-power embedded processors.

# Chapter 5

# An Adaptive Controller to Save Dynamic Energy


### Summary

*Last portable device generation demands powerful processors to run computing intensive applications such as video playing or gaming and ultra low energy consumption to extend device uptime. Such conflicting requirements are hard to fulfil and appeal for adaptive hardware that only consumes energy when required to satisfy the user experience; i.e., to evict video playback stallings.*

*LP-NUCA has two main dynamic energy wasting sources: (a) blocks are continuously migrating inter tiles even in low locality phases, (b) to reduce cache latency, the tag and data arrays or the tiles are always accessed in parallel.*

*This chapter proposes a learning-based controller that dynamically tunes the migration of blocks and the access policy to the tag and data arrays. During low locality phases the controller drops blocks from the cache and forces a sequential access to the tag and data arrays in the tiles, so that the two wasting sources are reduced. With a cycle-accurate simulator and layout derived energy estimations, the proposed controller reduces almost 20% on average dynamic energy for single and multi-thread workloads.*

## 5.1 Introduction

The way people use computers is partially shifting from personal computers with local data to mobile devices with data on the cloud. This "platform" move has not carried along "applications" changes. Users almost demand the same performance in mobile devices that they we used to experiment in desktop computers. Giving the same performance level with the tight energy constraints of mobile environments appeals for adaptive hardware that judiciously detects whether it is profitable to waste energy in order to satisfy the user.

One of the most energy-efficient mechanism to achieve high-performance is the memory hierarchy [17], where several small caches pretend to be an infinite and fast storage thanks to the locality of programs. Non-Uniform Cache Architectures, NUCA, exploit locality at a finer granularity than conventional caches because they enable inter bank block migrations [77]. Light Power NUCA, LP-NUCA, is a variant of Light NUCA (L-NUCA) for high-performance low-power embedded processors, such as those of mobile devices, that conveys blocks through three specialized Networks-in-Cache as L-NUCA does [124, 41], but also includes two static techniques for saving dynamic energy, Miss Wave Stopping and Sectoring. These techniques together with LP-NUCA ad-hoc network mechanism enable to outperform conventional and static NUCA organizations in terms of energy and performance.

The organization of LP-NUCA consists of many small tiles behaving as a very large distributed victim cache [68]. Blocks remain ordered by temporal locality (TL), so L1 (renamed r-tile) recently evicted blocks have a lower service latency than those previously evicted. LP-NUCA incorrectly assumes that all programs exhibit temporal locality across all their execution; hence, when the r-tile evicts a block, it triggers a chain of dominoes replacement for maintaining the TL block ordering. The problem arises during low locality phases. During them, the r-tile floods the rest of tiles with blocks that will be seldom requested and degrade older blocks that may be re-referenced in the near future. Moreover, LP-NUCA always accesses in parallel tag and data arrays to reduce cache latency. Since a data array access roughly consumes more than 5× the energy of the tag in LP-NUCA [41], this parallel access is a mayor waste of energy for request with high likelihood of miss. Ideally, we would like to detect low locality phases to prevent the insertion of low locality blocks and to dynamically switch between parallel and serial access in the tiles.

LP-NUCA was conceived for single-thread processors; however, to increase their $^{\text{performance}}/_{\text{energy}}$ ratio, current advanced embedded processors rely on extracting parallelism from multiple threads rather than from a single one. For example, the Intel Xeon LC3528, the MIPS MIPS32-1004K, or the Netlogic XLP832 simultaneously execute between 2 and 4 threads [64, 96, 48]. Traditionally, multi-threaded processors (MT) have shared all the cache hierarchy [135] increasing the chances of polluting the cache with useless blocks and evicting useful blocks from other threads. LP-NUCA in MT environments would suffer from this problem and

would benefit from a controller able to drop low locality blocks and to retain high locality ones. Finally, in this case we can expect performance improvements together with energy ones because high locality threads will experiment more hits in the LP-NUCA.

This chapter extends LP-NUCA in several significant ways. First, we identify that LP-NUCA wastes dynamic energy during low locality phases by continuously degrading blocks among tiles and by accessing the tag and data arrays in parallel even when the likelihood of miss is high. Second, we propose a learning based mechanism based on local search methods that dynamically selects when dropping blocks from the cache will harm neither performance nor energy. Third, we employ the same controller to dynamically adjust between parallel and serial access to the cache tag and data arrays leveraging from the congestion management support from L-NUCA. Fourth, we show that the proposed controller requires minimal hardware for improving energy consumption with marginal gains in performance.

The rest of the chapter is organized as follows. Section 5.2 presents the adaptive controller. Section 5.3 describes our methodology and simulation environment. Section 5.4 evaluates the results. Section 5.5 comments on the related work, and Section 5.6 concludes the chapter.

## 5.2 Adaptive Drop Ratio Controller

Figure 5.1 shows the organization of a LP-NUCA cache. When the r-tile misses, it evicts a block to a neighbour tile with the minimum latency difference. The destination tile, with a transport latency of 3, will repeat the operation to a tile with transport latency of 4, and this dominoes operation will continue until any tile has an empty slot or a block is evicted from the LP-NUCA.

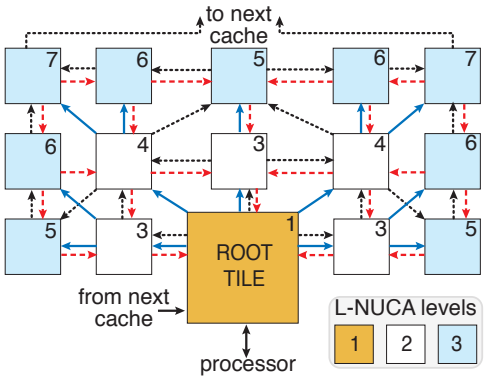



**Figure 5.1:** LP-NUCA basic organization with its three Networks-in-Cache: Search in blue, Transport in red, and Replacement in black. The number in the right upper corner of each tile represents its service latency assuming single-cycle tiles

This chain of evictions is useful to keep blocks ordered by temporal locality

but wastes a lot of energy when blocks leave the LP-NUCA before being requested again. The aim of this work is to find a controller minimizing the insertion of these useless blocks from the r-tile into the rest of tiles. The controller has to dynamically adapt itself to the phases of programs and requires to choose among the best drop ratios for multiple threads.

If we define a set of states for each thread $ST = [[dr_0, dir_0], ..., [dr_{n-1}, dir_[0]]]$ such that $0 \leq dr_i \leq 1$ where 0 means that all blocks are inserted into the rest of tiles from the r-tile and 1 means that all blocks are dropped [1], and $dir_i$ represents the direction of the thread, if in the next trial it will drop more or less blocks. Our problem is to find an algorithm that periodically adjust the drop rates and directions accordingly to a target function; i.e., the total number of hits in the rest of tiles or throughput, number of committed instructions during the epoch. Then, we can use a local search method such as hill climbing to explore the solution space[2]. hill climbing has already been used with good results for SMT fetch policies [23].

**Algorithm 1:** Hill-Climbing algorithm of the ADR controller

```
computeEpochStatistics();
if n_epoch % n_threads == 0 then
    foreach thread do
        if not isExempted(thread) then
            if ifTrialBetterThanRef(thread) then
                ref[thread].dir = trial[thread].dir;
            else
                ref[thread].dir = !trial[thread].dir;
            end
        end
    end
    if bestTrialBetterThanRef() or maxEpochsWoutChange() then
        refSt.dr = bestTrialSt.dr;
    end
end
n_epoch++;
setTrialState();
```

Algorithm 1 shows the proposed implementation of the ADR controller based on hill climbing. At the end of each epoch, it computes the epoch stats such as hits, dropped and evicted blocks, committed instructions, ... Then, every number of thread epochs the $ST$ is reevaluated. When a thread experiences a few number of misses, it is counterproductive his evaluation because we will be moving the

[1] Dirty blocks require to be sent to the next cache level in copy-back configurations.

[2] Other algorithms such as simulated annealing may reach better results at the cost of much larger computational cost and hardware complexity

controller in a flat zone and not towards the steep areas [3].

For those threads under evaluation, ADR checks if its trial have performed better than the reference state. If so, the direction of the thread, upwards (↑ drops) or downwards (↓ drops) is maintained and otherwise it is reversed. For example, if during the last epoch we have dropped 50% of blocks from a thread, and the global metric has improved, in its next trial it will try evicting more blocks. After establishing the direction, the ADR tries to update the reference state. From our experiments, we observe that it is worthy its update either when a trial is better or after a given number of epochs without change. In this work, we empirically fixed this value to 4× the number of threads in execution.

Finally, we update the epoch counter and set the trial state. If the direction is upwards the drop rate will increment and decrement otherwise (downwards direction).

**Table 5.1:** Possible organizations for the Adaptive Drop Ratio Controller

| Evaluation Trigger | Time | 1-512K cycles |
|---|---|---|
| | # misses | ¼-5× r-tile blocks |
| # Automata States | From 2 to 9 | |
| Target Metric | IPC, # hits, reuse rate | |
| Auxiliary Tags | From 0 to 128 entries | |

As stated in the previous paragraphs, there many strategies for tunning the drop ratio controller. Apart from the learning algorithm, hill-climbing in our case, we have to choose among many other parameters summarized in Table 5.1.

The first big choice is how to trigger a new epoch, either at fix intervals of time or after a fix number of r-tile misses. On the former, we have experimented with epochs from 1K to 512K cycles and, on the later from ¼ to 5× the number of blocks the r-tile stores[4]. The second big choice is the number of automata states or drop ratios; i.e., with 3 states a thread can drop all, half, or none of the evicted blocks. Finally, when a thread reaches the all drop state, $dr_i = 1$, the controller requires an heuristic for returning the injection of evicted blocks into the rest of LP-NUCA tiles. One option is to force the return to a state that does not drop all the blocks after a given number of epochs. Another cheap option a bit more clever can be the introduction of a small auxiliary tags keeping track of the last $n$ dropped blocks and lookup for misses in this structure. When updating the controller state if several requests have matched in the auxiliary tags, automatically that thread leaves the all drop state.

Finally, Figure 5.2 shows the behaviour of the controller executing 255.vortex with 179.art during 2 millions of cycles. ADR synchronously reevaluates after 4096 cycles, has 3 dropping states, and when the drop ratio is 1, cache arrays are serially

[3] We could also reevaluate after a number of epochs equals to the no exempted threads, but the hardware complexity will be higher.

[4] Assuming a 32KB r-tile organized in blocks of 32B, there are 1024 blocks in total.

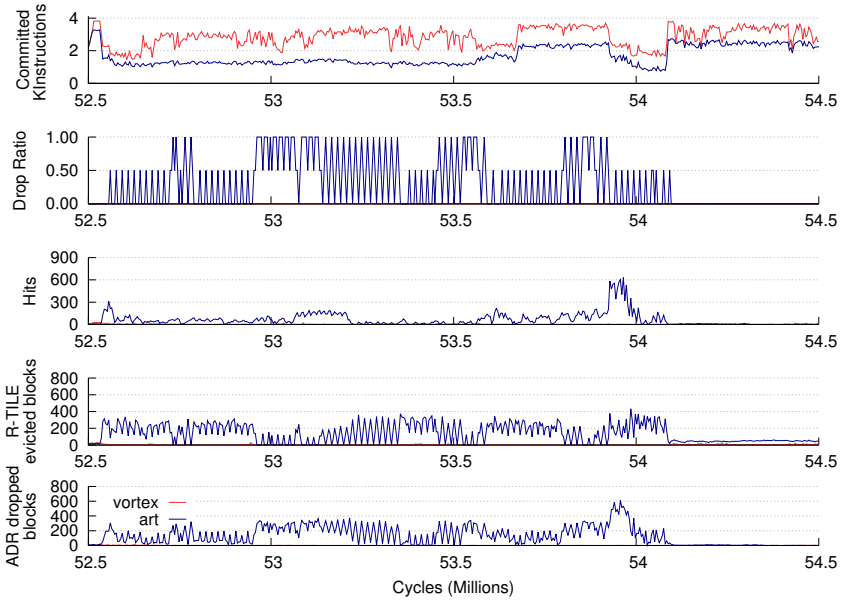



**Figure 5.2:** SMT execution of 255.vortex and 179.art with an ADR of 4Kcycles epochs and 2 automata states

accessed. It includes an auxiliary tag array of 512 entries. The plot includes from top to bottom the number of committed instructions, the drop ratio indexes, the number of rest of tiles hits, the number of evicted (inserted) blocks into the rest of tiles, and the number of dropped blocks. 255.vortex, red lines, is a example of of benchmarks that is better to exempt from the controller. Its miss rate is very low, and by dropping blocks we could only reduce its performance and increase the accesses to the next cache level. On the contrary, 179.art experiences program phases in which it pollutes the cache. For example, before the 54M point, the controller drop blocks and keeps useful blocks inside the cache that are served, and then when the miss rate drop again below the exemption threshold, it evicts all the blocks inside the rest of tiles.

### 5.2.1 Hardware Cost

The hardware implementation of the Adaptive Drop Ratio controllers requires minimal overhead. Most current processors already include the performance counters for the target function and we only require to store the reference state for all the threads, 1 bit for the direction plus $\log_2(drop\ ratios)$, and the trial configuration. The partial tags for the single thread mode only require an small SRAM array, consuming little energy, and if necessary it could be easily replaced by a bloom filter.

One novelty of this work is the proposal of switching between parallel and serial access to the cache arrays. At first glance, this feature could be hard to im-

plement, just the opposite is true. The key observation is that when the likelihood of cache miss is high only the tag array is accessed. So we can add an extra bit in the Search Network disabling the accesses to the data arrays in the tiles. Miss will propagate back-to-back over the fabric. Nevertheless, in the rate case that a tile hits during a serial access, the data array have to be accessed. Since we can not stop the request propagation in the Search network because it does not have any control flow mechanism, we need to re-inject the request in parallel mode. This re-injection phenomenon is already supported by LP-NUCA in case of congestion of the Transport network, when a tile hits and does not have any output transport link available. Therefore, a serial request hitting in a tile will reset the serial and set the congestion bits, so that the request be be reinserted.

## 5.3 Methodology and Simulation Environment

We employ the same simulation environment, energy estimations, and cache hierarchy organizations that previous LP-NUCA work [41]. The baseline processor resembles the IBM/LSI PowerPC 476FP [49, 89] and executes 1 or 2 threads simultaneously. Table 5.2 summarizes the main parameters for the reference processor and cache hierarchy, including the same L1 and L3 caches and either a conventional L2, a S-NUCA, or an L-NUCA. All caches use LRU replacement except L-NUCA that employs LRF, least-recently-filled, and they all have a single read/write port.

**Table 5.2:** Simulator Micro-architectural parameters. BS, AM, lat, and init stand for block size, access mode, latency, and initiation rate, respectively

| | | | |
|---|---|---|---|
| Clock Frequency | 1 GHz | Fetch/Decode/ Commit width | 2 |
| Issue width | 2(IN+ME) +2FP | ROB / LSQ entries | 32 / 16 |
| INT/FP/MEM IW entries | 8 / 8 / 8 | branch predictor | bimodal + gshare, 16 bit |
| Miss. branch penalty | 6 | Instruction Cache | perfect |
| L1/L2/L3 MSHR entries | 8 / 8 / 4 | TLB miss latency | 30 |
| MSHR secon. misses | 4 | Store Buffer/ L2/ L3 WB size[a] | 8 / 4 / 4 |

| | |
|---|---|
| L1/r-tile[b] | 32KB–4Way–32B BS, write-through, 2-cycle lat, 1-cycle init |
| L2 | 512KB–8Way–32B BS, serial AM, 4-cycle lat, 2-cycle init, copy-back |
| S-NUCA | 2×2 128KB–2Way–32BS, parallel AM, 3-cycle lat, 3-cycle init, copy-back |
| L-NUCA rest of tiles | 32KB–2Way–32B BS, parallel AM, copy-back, levels: 3, total size: 448KB |

| | |
|---|---|
| L3 | 4MB eDRAM–16Way–128B BS, 14-cycle lat, 7-cycle init, copy-back |
| Main Memory | 100 cycles/4 cycle inter chunk, 16 Byte bus |

[a] L2, S-NUCA, L-NUCA, and L3 Write Buffers coalesce entries

[b] In r-tile, copy-back and write-around

Our workload comprises the same embedded domain oriented application

than previous work [41]. Nevertheless, To get deeper insights from the results, we divide the benchmarks in two groups: low MPKI and high MPKI as table 5.3 shows. For the two SMT experiments we presents the results in three groups: low MPKI, medium MPKI, and high MPKI when both, one, and none benchmarks of the combination exhibit a low misses per kilo instruction rate.

**Table 5.3:** Workload selection. *I*, *F*, *0*, *6* refer to Integer, Floating Point, SPEC CPU2000, and SPEC CPU2006, respectively

| | |
|---|---|
| **Low MPKI**[a] | 186.crafty *I0*, 255.vortex *I0*, 177.mesa *F0*, 458.sjeng *I6*, 482.sphinx3 *F6* |
| **High MPKI** | 164.gzip *I0*, 179.art *F0*, 187.facerec *F0*, 401.bzip2 *I6*, 445.gobmk *I6*, 464.h264ref *I6*, 473.astar *I6*, 453.povray *F6* |

[a] L1 MPKI rate lower than 20 in the baseline processor

We used a similar energy estimations than the previous work for the battery powered domain in 32 nm [41]. In the single thread execution, the simulator warms-up caches and branch-predictor for 200M instructions before starting the cycle-accurate simulation. We follow the same approach that Li *et al.* for energy and delay measurements in SMT environments [83], and account for all the energy consumed until the last thread commits 100M instructions.

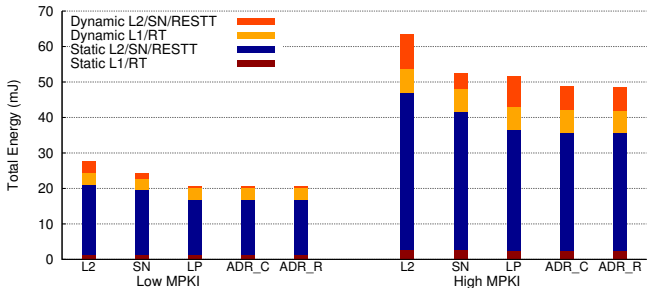
## 5.4 Results Evaluation

The section presents compares the cache organizations presented in previous section, namely, conventional L2 (L2), Static NUCA (SN), LP-NUCA (LP), and two LP-NUCAs enhanced with two adaptive drop ratio controllers: one with synchronous epochs (ADR_C) and another with epoch based on the number of request for evictions (ADR_R). Both ADR_C and ADR_R have been selected after exhaustively exploring the controlling design space with the options shown in Table 5.1. ADR_C re-evaluates the drop ratios and directions every 4096, while in ADR_R occurs after 1024 attempts of r-tile eviction (or r-tile primary misses). Both controllers share the rest of parameters, namely, 3 dropping states, 512-entry auxiliary tags, and an exemption threshold of 50 MPKI in the rest of tiles.

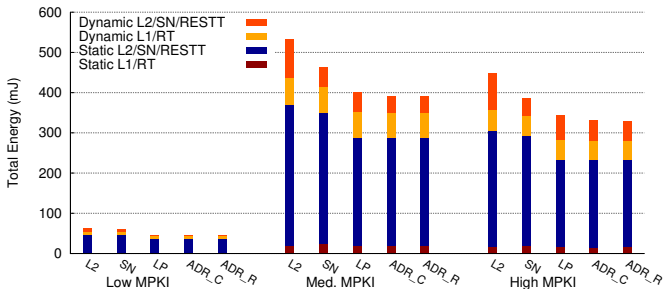
First, we compare the energy consumption, then we continue with the execution time, and we finish with energy-delay results. For the sake of brevity, we only show overall and averaged results, but the individual behaviours do not differ from the presented one.

### 5.4.1 Energy

Figure 5.3 shows the total energy consumed by all configurations. LP-NUCA with the ADP have the best results regardless the benchmark group and the number of threads.




**(a)** Single Thread




**(b)** 2 SMT

**Figure 5.3:** Energy consumption comparison

If we focus on the last three bars to compare the performance of the ADR we observe that the synchronous one performs slightly better than the based on the number of replacements. In the more energy demanding benchmarks, high MPKI, the controller reduces energy by 20% on average.

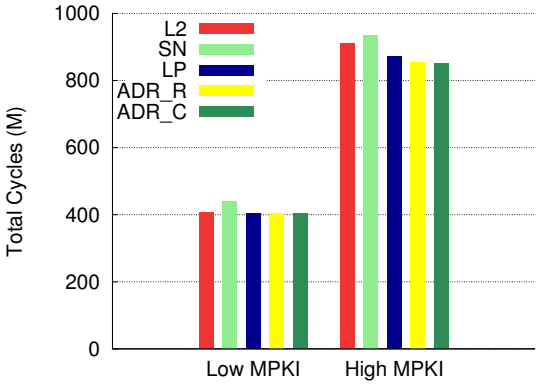
### 5.4.2 Execution Time

Dropping blocks may reduce LP-NUCA hit rates and increase execution time, EX. To verify that the proposed ADRs do not affect EX Figure 5.4 shows the total execution time.

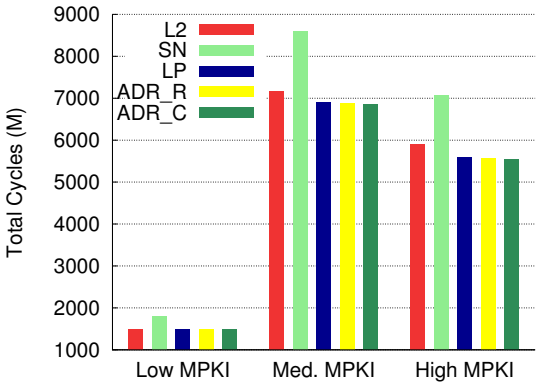
LP-NUCA shows performance improvements over L2 and SN, and both controllers slightly reduce execution time because they keep inside the cache blocks that otherwise would be expelled. ADC_R and ADR_R reduced total execution time by 2.14% and 2.29%, respectively, in the single-thread environment. Improvements become marginal in the 2 SMT and reduce to 0.62% and 0.89% for ADC_R and ADC_C, respectively, because the multi-threading execution covers the memory stalls.

### 5.4.3 Overall System Impact

Finally, we present the sum of the energy-delay of the tested benchmarks. Figure 5.5 includes the L3 energy to show that dropped blocks are not requested to



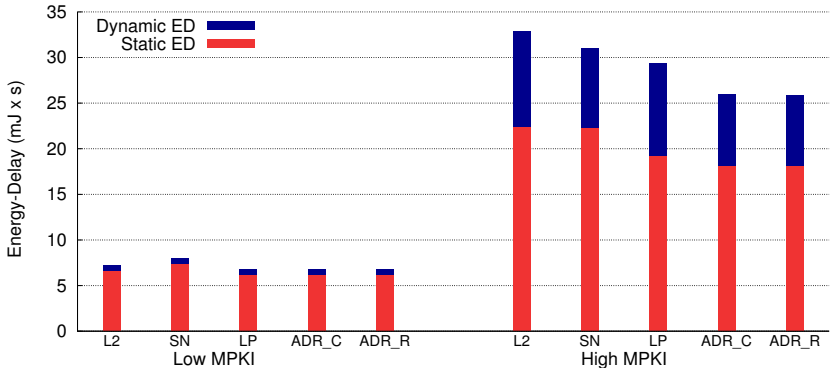

**(a)** Single Thread



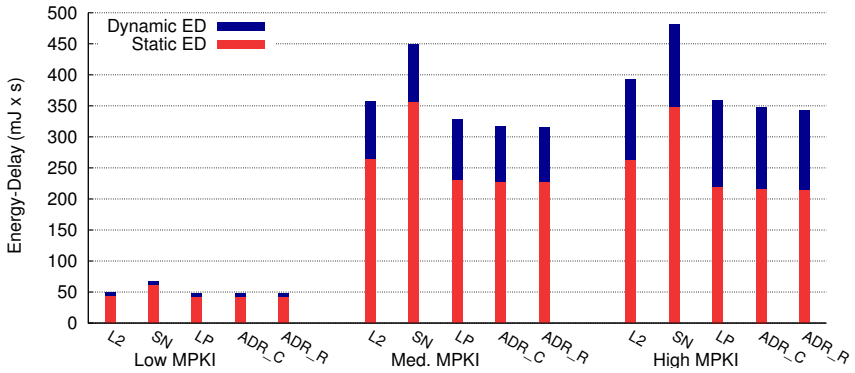

**(b)** 2 SMT

**Figure 5.4:** Total Execution Time for the different configurations

the L3 increasing the overall energy.




**(a)** Single Thread




**(b)** 2 SMT

**Figure 5.5:** Energy-Delay. This figure includes L3 cache consumption as well

Again ADR_R and ADR_C are the winners in all categories. LP-NUCA improvements in execution time reduces the static component with regards to L2 and SN, and the controllers reduce on average dynamic energy-delay, their target, 7.6% for all but low MPKI workloads.

## 5.5 Related Work

Architects have proposed a plethora of designs to save cache energy through reconfigurable caches that change their way, set of both configurations at run time [1, 10, 148, 127]. For an updated state of art please refer to Sundararajan *et al.* [127]. Previous works adapt the cache at a finer granularity than this work, and most proposed techniques can be easily applied to the LP-NUCA. Contrary to the original LP-NUCA design [41], this work proposes a proactive dynamic technique to save energy while previous ones, Sectoring and Miss Wave Stopping, were completely static and application agnostic. Besides, this work analyzes SMT

workloads which have not been extensively studied.

Regarding the learning based approach, the Hill Climbing algorithm has been employed for distributing resources in SMT processors [23], but not for cache reconfiguration.

## 5.6 Final Remarks

Ultra-portable mobile devices demand quasi-desktop performance with a fraction of energy consumption. Since application behaviour changes during execution, processors require adaptive mechanisms wasting the minimum amount of energy when necessary.

This chapter proposes an adaptive controller for LP-NUCA, a tiled organization for high-performance low-power processors, that automatically decides when cache blocks are not reused and can be dropped reducing the cache activity. Besides, during high dropping phases, the controller is able to change the cache array access from parallel to serial further reducing the energy consumption.

With representative workloads, a cycle-accurate simulator, and implementation based energy estimations, we observe that the proposed controller reduces dynamic energy on average by 20% for single-thread and 2-threaded workloads without increasing the execution time.

# Chapter 6

# Light NUCA for Simultaneous Multithreaded Processors


### Summary

*This chapter starts briefly introducing multithreading as a technique to tolerate the latency gap. Current multithreaded processors include large last level caches to fit the working sets of multiple programs at the same time, but first level caches have remain almost constant is size to maintain their latency. This scenario is suitable for Light NUCA as this chapter proves. Besides, it describes a novel sampling based methodology to reduce the simulation time without lossing accuracy.*

## 6.1 Introduction

Multithreading (MT) is supported by an ample spectrum of current processors devoted to uneven computing segments such as: embedded, high throughput, or high performance. Examples of representatives from these segments are Netlogic XLP832 (4-way multithreading, 8-cores), Oracle SPARC T3 (8-way multithreading, 16-cores), or IBM POWER7 (4-way multithreading, 4-6-8 cores), respectively [48, 118, 71].

All previous examples share a powerful multilevel cache hierarchy with large Last Level Caches (LLC), and only the XLP832 departs from the conventional organization including a ring for communicating the private L2 caches, the eight L3 cache banks, and the four DDR ports. While these LLCs seem able to accommodate the multiple working sets of SMT execution, sharing in the levels close to the processors proves to be more complex. On one hand, L1 and L2 caches deal with the latency-power vs. size trade-off. On the other hand, MT architectures add a new trade-off, number of threads in execution vs. miss rate. With many threads, cache misses can be tolerated executing instructions from other threads, but as the number of threads grows, the collective working set becomes larger and more changing, resulting in miss ratios potentially harmful to performance. The larger their number, the larger and size changing the collective working set becomes and the larger the miss rate. So when the miss rate reaches a critical value in which threads execution fails to overlap, the processor stalls and has the same problem that single thread machines.

SMT architectures may be favored by caches designed to support working set awareness such us the L-NUCA [124]. L-NUCAs belong to a family of cache organizations that has received much attention for improving cache performance: Non-Uniform Cache Architecture (NUCA) [77]. The seminal NUCA work targets the wire delay problem[1], and proposes the melting of the L2 and L3 caches into a meshed array of cache banks. Nevertheless, to the best of our knowledge, there is little work on evaluating NUCA with simultaneous multithreading processors (SMT).

Part of the complexity of assessing multiple cache hierarchies lies in the required simulation framework. So to carry out the experiments, we propose a simple yet efficient MT simulation methodology ensuring the accuracy of the results abreast with a sort simulation time. The methodology is based on statistical sampling, and contrary to other alternatives does not require a prior long profiling of the applications.

This work gives two main contributions. The first one is introducing a powerful methodology to evaluate MT architectures. The second one is the comparison and evaluation of several state-of-the-art cache hierarchy organizations driven by the SPEC CPU2006 benchmark suite. From the results, we conclude that re-

[1] The wire delay is longer than the bank delay and represents most part of the total cache latency in LLCs.

gardless the number of threads L-NUCAs outperform conventional multibanked and dynamic NUCA organizations, both in terms of throughput and fairness.

The rest of the chapter is organized as follows. Section 6.2 elaborates on previous work. Section 6.3 presents the proposed evaluation methodology. Section 6.4 describes our experimental framework and the hierarchies under test. Section 6.5 comments on the results, and Section 6.6 concludes the chapter.

## 6.2 Background

Tullsen, Eggers, and Levy in their SMT seminal work compare the performance of private and shared L1 caches (for both instruction and data) and observe that regardless the number of threads (from 1 to 8) shared data caches are the best choice and private instruction ones are only preferable for the 8-thread case [135]. Also they point out that shared caches do not require any special hardware for coherence support. Then, Tullsen and Brown observed that in many cases when a thread experiences a very long latency operation, such as a cache miss, it is better to flush the resources of the stalled thread rather than keeping them ready [133].

Hily and Seznec studied how secondary cache bandwidth limited SMT performance in a trace-driven environment [55]. They also point out that the larger the number of simultaneously executed threads the larger the L1 cache size has to be. Besides, when the number of threads increases, the memory references generated by the simultaneous execution of independent threads exhibit less spatial locality than that of a single thread, increasing conflict misses[2]. Block size is more critical than associativity and as the number of threads rise, it is preferable to keep small block sizes (16 to 32 BS).

To improve SMT performance, Settle *et al.* define a cache partitioning scheme based on column caching [115]. Two policies can control the partitioning: (a) synchronous in which each 1 million cycles, the partition is heuristically set for the next interval. (b) asynchronous in which the LRU algorithm is affected by some thread reuse counters.

Nemirovsky and Yamamoto analyzed the effect of varying cache capacity, associativity, and line size on miss rate for multistreamed architectures [104]. They observe that increasing both cache capacity and associativity reduces miss ratio specially for small caches and that large block size increase miss ratio.

The Multithreaded Virtual Processor (MVP) is a coarse-grain multithreaded system with software support that explicitly forces context switching on long latency events such as cache misses, I/O, or synchronization [81]. The evaluation comprised parallel workloads, and they show that when threads share a few data, the increment in miss rate affects performance.

Garcia *et al.* studied several data cache organizations for multithreaded processors using a trace-driven environment [37]. In accordance with other authors [55, 56], they observe that large associativities reduce inter-thread misses

[2] Larger associativities reduce conflict misses.

and that XOR-based placement reduces inter-thread miss rate in some cases. Besides, they proposed several organizations combining the hash-rehash caches and static cache splitting.

Sarkar and Tullsen proposed two strategies to minimize inter-object data cache misses at compilation time [114]. Lopez *et al.* studied the best control strategies for reconfigurable caches in SMT Globally Asynchronous, Locally Synchronous processors with a limited set of SPEC CPU2000 benchmarks. They conclude that the best control strategy to maximize performance is the harmonic mean of the per-thread weighted access time [87, 88].

Several authors have proposed SMT methodologies for selecting representative mixes of programs. Raasch and Reinhardt propose to profile individual applications and with the extracted characteristics choose the most representative pairs [111]. By combining the pairs, they also generate 4-thread workloads. Van Biesbrouck *et al.* proposed a methodology considering all starting phases and points but it requires a complex profiling and an advanced work flow to gather the results [15].

On the contrary our approach neither require profiling nor complex output information gathering, but it does not consider multiple starting points by default. Nevertheless, since we employ last simulation policy[3], faster programs execute multiple times until the slower ones finishes, so they run together starting at different points.

## 6.3 A Simple Statistic-based Methodology for Multiprogrammed Workloads

The generation of simulation traces is always a time consuming and costly process. Single program traces extracted with solid approaches such as SimPoints or SimFlex provide accurate simulation results [50, 140]; however, no guidelines are found in the literature about how to obtain a representative set of thread mixes from representative samples of thread individuals. Our goal is to provide a simple method to efficiently simulate multiprogramed workloads for multiple metrics.

The key idea is instead of executing all possible combinations for a given number of threads, to execute a representative sample based on statistical sampling [66, 140] so that simulation time reduces by several orders of magnitude while keeping measurements below a given error within a confidence level large enough. In this work, we focus on mixes composed of different threads (combinations without repetition), but the proposed methodology can be used with repetition as well.

Figure 6.1 shows the required steps to obtain the sample:

**Selection of metrics of interest**: For example if we want to compare the impact of different cache hierarchies on SMT processors we can take adjusted

[3]Simulation finishes when all threads have executed at least 100M instructions.

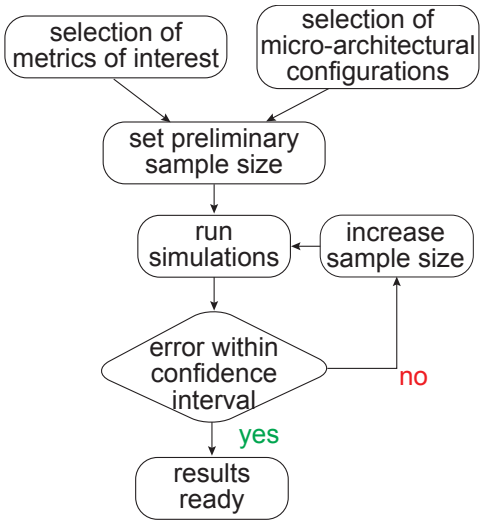



**Figure 6.1:** Flowchart of the proposed methodology

STP and ANTT, IPC throughput[4], and fairness [36, 32].

$$STP = \sum_{i=1}^{n} \frac{CPI_i^{SP}}{CPI_i^{MP}}, \; ANTT = \frac{1}{n} \sum_{i=1}^{n} \frac{CPI_i^{MP}}{CPI_i^{SP}}$$

$$IPC\ throughput = \sum_{i=1}^{n} IPC_i, \; fairness = \frac{\min_i \left( \frac{CPI_i^{MP}}{CPI_i^{SP}} \right)}{\max_i \left( \frac{CPI_i^{MP}}{CPI_i^{SP}} \right)}$$

$MP$ and $SP$ refer to the multithreaded and singlethreaded execution of a program.

**Selection of micro-architectural configurations**: Pick some of the configurations to be analyzed. In general those with lower performance are preferable because they tend to experiment the highest variances of the metrics. In our cache hierarchy comparison, we can take a conventional multibanked organization, a dynamic NUCA, and a light NUCA.

**Set preliminary sample size**: In this step, we have to choose a sample size—the larger the number of threads, the lower this value [31]—, and then, to randomly pick the combinations of programs for their simulation. This value should be big enough [66], larger than 30 at least, but tractable in the desired simulation environment.

**Run simulations**: Run the selected combinations and compute the sample

[4] IPC throughput is advantageous because it allows an absolute comparison among configurations. We can use IPC throughput whenever mixes are made from independent threads, because then no unpredictable instruction spinning can arise; e.g., before entering a critical section.

size for your given confidence interval with the next formula [66]:

$$n = \left( \frac{100 \times z \times s}{r \times \overline{x}} \right)^2 \tag{6.1}$$

where $n$, $z$, $s$, $r$, and $\overline{x}$ stand for the sample size, normal variate of the confidence interval, error (in %), sample standard deviation, and population mean, respectively.

**Error within confidence level?**: Check if the obtained $n$ is lower of equal than your preliminary sample size. If not, pick some different extra combinations, run them, and repeat the last steps until the results are ready.

Once a sample size has been established, all the configurations under test must be run in order to check that their results also fit within the confidence interval. Adding new configurations require the same procedure, so if the initial selection of micro-architectural configurations is carefully done, the sample size will not grow.

## 6.4 Comparing SMT cache Hierarchies

### 6.4.1 Common Baseline Parameters

We have heavily extended Simplescalar 3.0d [5] for Alpha with: Simultaneous Multithreading Support [135], Reorder buffer and three issue windows (IW) for integer, floating point, and memory instructions, speculative wake-up support and selective recovery (as in the Intel Pentium 4 [57]), one-cycle payload and register file stages, accurate timing models for non-blocking caches, write buffers, buses, network contention, flow control, and request arbitration. For the rest of parameters see Table 6.1.

**Table 6.1:** Baseline Processor Configuration

| | | | |
|---|---|---|---|
| Fetch | ICOUNT.2.8 | | |
| Decode / Commit Width | | | 4 |
| Branch Predictor | bimodal + gshare, 16 bit | ROB / LSQ | 198 / 96 |
| Issue Width | 4 (INT + MEM) + 4 (FP) | Issue Queues | 64(INT) / 32 (FP & MEM) |
| Functional Units | 4 INT + 4 FP ALU, 2 INT MULT + 4 FP MULT/DIV | | |
| L1 cache | 64KB-4Way-32B BS-2ports-LRU, Lat 2, Init. Rate 1 | | |

| | |
|---|---|
| L2 | 128KB:1MB-8Way-32B BS-1port-LRU |
| D-NUCA | 128KB:512KB-4Way-128B BS-1port-LRU, 8 columns |
| L-NUCA | 2:5Levels-16KB-4Way-32B BS-1port-LRF |

| | |
|---|---|
| L3 | 8MB-16W-128B BS-1port-LRU, 8 banks |
| Main Memory | First chunk: 200 cycles, 4-cycle inter chunk, 16B wires |

Thread fetch is prioritized based on the ICOUNT policy [134]. Structures such as ROB, IWs, STB, WB,... are shared among threads. To avoid starvation

at commit, a thread can not occupy more that three quarters of L2/L-NUCA/D-NUCA Write Buffers. Threads commit in Round Robin fashion with a maximum of 4 instructions committed by all the threads (RR.4.$X$), where $X$ corresponds to the number of threads in execution [134].

### 6.4.2 Cache Memory Organizations

Focussing in the first and second cache levels, we have selected three very distinct organizations. The first one is the baseline, corresponding to a conventional 3-level hierarchy with a 4-banked L2 cache, and an 8-banked, 8-MB L3 cache, see Figure 6.2.

In any cycle, the L2 cache can start servicing up to two L1 load misses from the MSHR, the rest of banks can simultaneously begin the processing of a write buffer entry, and two whole cache blocks can be in transit to the L1 MSHR. For this configuration, we tested L2 sizes from 128KB to 1MB organized in either 1, 2, 4, and 8 banks. The output crossbar has a fixed latency of 2 cycles with an initiation rate of 1 in configurations with 2 or more banks.

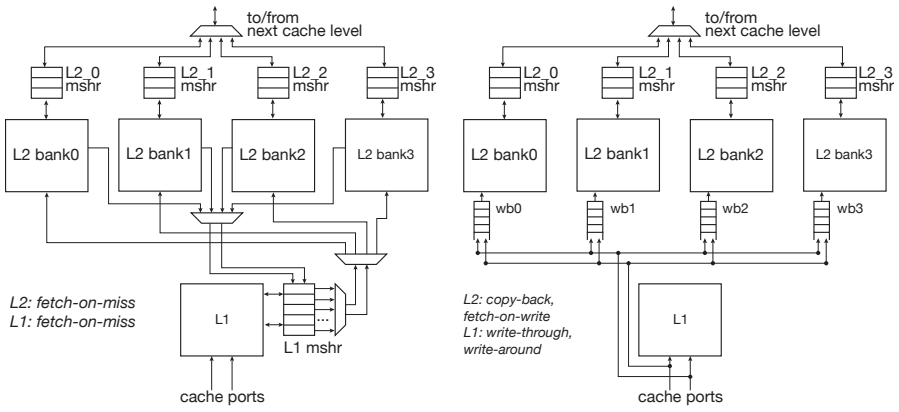



**Figure 6.2:** Baseline L2 cache organization with 4 banks

The second hierarchy replaces the L2 and L3 caches by a dynamic NUCA (D-NUCA) [77], Figure 6.3, in which its original interface with the L1 cache and shared mapping have been replaced by a crossbar to provide more bandwidth and by a simple mapping[5], respectively. As in the conventional organization, there is a write buffer for each column. In this way we can inject the same number of requests per cycle, so the conventional hierarchy can be considered as a particular case of a NUCA with a single row and without the partial tags [77].

The last one is Light NUCA (L-NUCA) [124], but also modified to provide more bandwidth, see Figure 6.4. Changes include widening the search links from

[5] A block can only reside in one column.

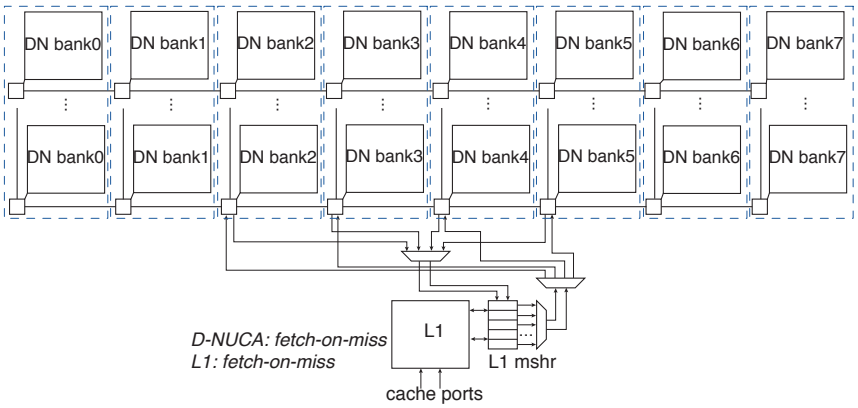
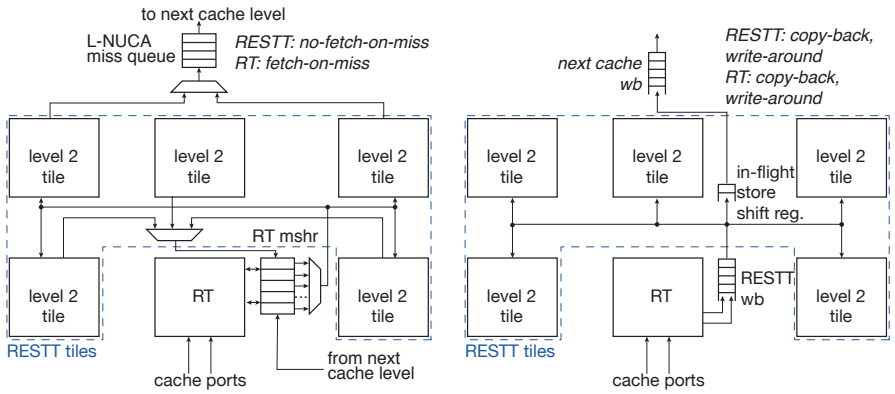



**Figure 6.3:** Multibanked Dynamic NUCA. A block can be mapped to any of the columns surrounded by a dashed line

word to block size (from 8 to 32B), increasing the write buffer size to match the rest of organizations (the buffer in between the r-tile and the rest of tiles). Besides the replacement buffers are managed with an improved back-pressure policy, so that the eviction of blocks from the r-tile does not stall until the replacement buffer of the r-tile becomes full.

We tested 2 organizations based on L-NUCA, one including the same L3 than the baseline organization, and another in which the LLC is a dynamic NUCA, similarly to previous work [124].




**(a)** Loads **(b)** Stores

**Figure 6.4:** 2-level L-NUCA load and store logical organization. For the sake of clarity, the Figure includes neither all network nor control flow links

During the setup or the organizations, we observed two interesting design details helping to maximize performance. First, regarding the priority of L1/r-tile misses, loads should have priority over writes except when a miss hits in the write

buffer[6] in which case priorities are reversed until the requested data can be served by the cache. Second, a centralized write buffer reduces performance except when the number of entries at each distributed write buffer is very small. As an approximate rule of thumb, each bank requires a coalescing write buffer with as many entries as threads are simultaneously executing. As stated by Hily and Seznec, bandwidth can be the limiting factor of the cache hierarchy in SMT [55], and the larger the number of banks (up to 8), the bigger the performance.

### 6.4.3 Workloads

Our multiprogrammed workload comprises combinations of 2, and 4, 6, and 8 programs from SPEC CPU2006 without repetition, namely, all benchmarks but 483.xalanbmk [54]. For this type of study, multiprogramed loads are preferable over parallel ones because they stress more the memory hierarchy since there is no sharing among threads. For each program, we have selected a representative trace consisting of 100 million instruction following the SimPoint guidelines [50]. Simulations terminate when all threads have executed at least 100M instructions, last policy, but program statistics are only gathered for the first 100M.

## 6.5 Experimental Results

### 6.5.1 Sample Sizes and Simulation Time

Table 6.2 shows the number of all combinations without repetition of our traces executing 2, 4, 6, and 8 threads, the average time required for running one combination of programs[7], and the sample size required to obtain accurate results, with an error less than 3% with a 97% confidence level for all configurations under test. This means for example that in 97 of 100 times a randomly chosen sample of 251 combinations will have an error than 3% in all the metric compared to the whole population of 6-threads combinations (37674). For the 2 threads case, we execute all combinations because total execution time is affordable. Nevertheless, as we increase the number of threads, our methodology obtains savings in simulation time of ×93, ×150, and ×9743, for 4, 6, and 8 threads, respectively.

**Table 6.2:** Sample sizes varying the number of threads

| | # Threads | | | |
|---|---|---|---|---|
| | 2 | 4 | 6 | 8 |
| Total combinations | 378 | 20475 | 37674 | 3108105 |
| Avrg. sim. time (min) | 18.5 | 45.7 | 60.4 | 90.5 |
| Sample size | — | 220 | 251 | 319 |

[6] Write buffers do not provide data.

[7] This value is for the L-NUCA + D-NUCA simulator, the slowest in our simulation framework, in an Intel Nehalem 2.33 GHz.

Table 6.2 provides the minimum sample size for all metrics of interest with an error less than 3%, in our case: STP, ANTT, IPT throughput, and fairness. Since fairness represents the ratio between the minimum and maximum slowdowns, it has the largest variance, and, hence, determines the required sample size.

If we focus on a single number of threads, for example 4, we can show the sample size of the best configurations from each organization as table 6.3 does. L2, NC-*I*x*J*, LN*I*, LN*I*-NC-*J*x*K* correspond to the conventional baseline, D-NUCA with *I* columns and *J* rows, L-NUCAs with *I* levels, and L-NUCAs combined with D-NUCAs organizations, respectively. Besides, each configuration includes its total size for the L2 and L-NUCAs and the size of their banks for D-NUCA.

**Table 6.3:** Sample sizes for the different metrics and configurations in 4SMT execution

| | Metric | | | |
|---|---|---|---|---|
| | STP | ANTT | IPC Throughtput | Fairness |
| L2-256KB-8Banks | 126 | 111 | 85 | 162 |
| L2-1MB-8Banks | 126 | 113 | 90 | 160 |
| NC-8x2-512KB | 111 | 132 | 119 | 218 |
| NC-8x4-256KB | 118 | 141 | 128 | 220 |
| LN3-240KB | 103 | 96 | 51 | 194 |
| LN4-448KB | 103 | 96 | 51 | 193 |
| LN2-NC-8x2 | 104 | 96 | 48 | 189 |
| LN2-NC-8x4 | 105 | 97 | 47 | 188 |
| LN3-NC-8x2 | 106 | 96 | 46 | 210 |
| LN3-NC-8x4 | 103 | 95 | 47 | 211 |

Irrespective of the organization, we observe that the required sample size follows a common trend: computing IPC throughput requires small sizes while fairness, due to its greater variance, requires doubling or even quadrupling the sample size. STP and ANNT lie in a middle ground. Across all organizations, IPC throughput is the metric giving the more uneven sample sizes, meaning that L-NUCA organizations achieve less IPC variability among individual thread combinations. From these results, we conclude that in general adding a different configuration to test does not require a large investment in new simulations for the preexisting configurations. Finally, fairness is the only metric that requires more samples as the number of threads rise. While many threads are able to keep all functional units busy, they tend to bother each other starving some of them. Conclusions are similar for other number of threads, and we do not show them for the sake of brevity.

### 6.5.2 STP, ANTT, IPC throughput, and Fairness

Figure 6.5 shows the results for the four metrics of interest for the best configuration of each hierarchy organization. STP and ANTT are metrics relative to the performance of a single thread system aimed at programs that may expend time in spin-lock loops, which is not our case [32]. In both metrics, L2 overpasses the

rest of hierarchies from 4 threads and beyond, and the L-NUCA ones dominates in the 2 thread case. This results are mostly due to the fact the L2 has the worst IPC in single-thread mode and the maximum IPC for all configurations is 4, so L2 relative slowdowns are smaller as the number of thread rises.

IPC throughput is an absolute metric representing the amount of committed instructions per unit of time. LN+NC achieves the best results regardless the number of threads close followed by LN. The small difference between them is mostly due to the NUCA partial tags not present in the L3 [77]. As the number of threads and bandwidth demand increase, the L2 overpasses the NC.

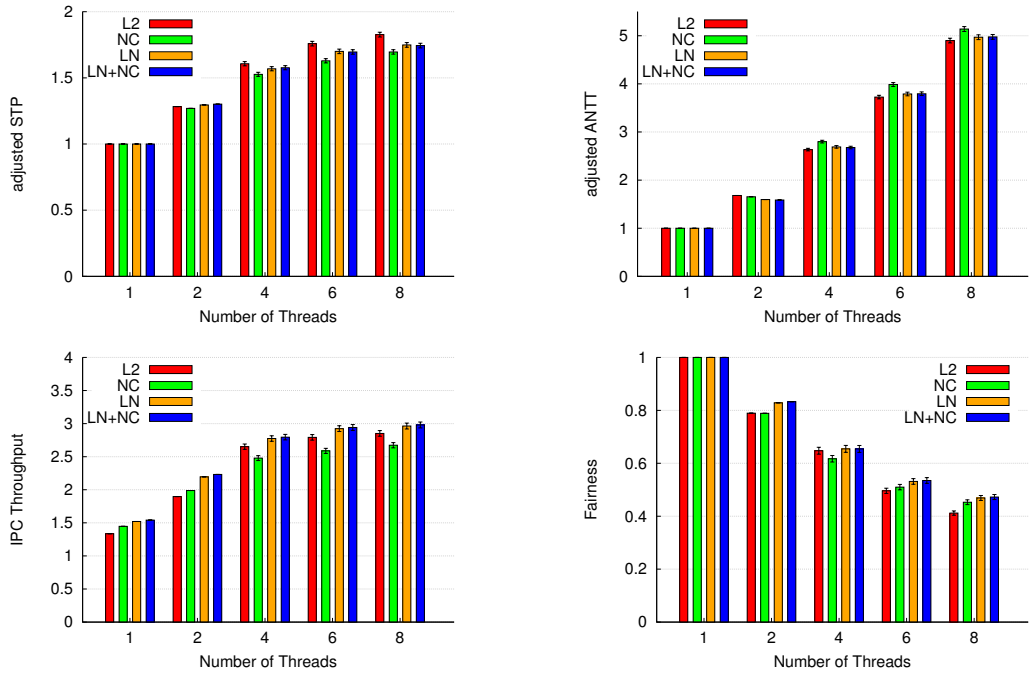



**Figure 6.5:** Results for the best configuration of each organization: L2, NC, LN, and LN+NC correspond to the L2-256KB, NC-8x4-256KB, LN3-240KB, and LN2-NC-8x4, respectively

Fairness is defined as the quotient between the programs that have suffered the lowest and the highest slowdowns. A value of zero means complete starvation of one thread,and a value of one means all threads experience the same slowdown.

## 6.6 Final Remarks

The adoption of thread level parallelism as the mainstream way to continue improving the performance pace of computers requires novel mechanism and the reevaluation of those which are well established such as cache hierarchies.

While large Last Level Caches have received a lot of attention in recent years, first and second levels have remained apart.

This work analyzes multiple state-of-the-art cache hierarchies executing multiprogramed workloads from 2 to 8 threads. In order to provide accurate results in a reasonable amount of time, we propose a sampling based methodology reducing simulation time by 2, 3, and 4 orders of magnitude for 4, 6, and 8 thread workloads, respectively. These savings do not occur at the cost of fidelity because their error is less than 3% for a 97% confidence level for a high variance metric such as fairness.

From the analysis we observe that regardless of the Last Level Cache, either conventional multibanked or dynamic NUCA, and the number of threads, L-NUCA provides the more efficient solution in terms of both throughput and fairness.

# Chapter 7

# Other Issues: Multicore–Coherency and Real-Time


### Summary

*To conclude the technical aspect of the Light NUCA, this chapter qualitatively analyzes two relevant aspects of memory hierarchies. The former is the adoption of L-NUCA caches in multiprocessors; namely, the requirements and advantages of L-NUCA in coherent system. The latter is real-time system because potentially the latency of L-NUCA can vary more than that of a conventional cache. Although this is certainly true, it is also the case that L-NUCA tiled organization offers multiple opportunities for locking and partitioning content, which are the two core techniques of worst case execution time computation algorithms.*

## 7.1 Multicore–Coherency

Next, we analyze the feasibility of integrating L-NUCA within a multicore chip in a coherent memory system, which essentially does not differ from that of conventional caches. Thereafter, we assume at least one private cache such an L-NUCA inside each core, addressing processor requests, and a single shared last-level cache (LLC) in charge of write serialization and coherence command handling. The LLC replies to processor requests by issuing coherence commands to a single, several, or all the processor cores accordingly to the content management policy among caches (inclusive, exclusive, noninclusive) and the scheme for locating cache blocks on cores (from full-map directories to snoopy-like management). These principles are commonly followed by high-end multicore chips[1].

An LLC can monitor alike the contents of a conventional L1, L2 pair of caches or an L-NUCA. In both cases the LLC controls block refills and evictions. Thus, regardless of contents management and block location policies, we can focus on the inner management of the external coherence commands entering an L-NUCA cache, such as invalidations, requests for ownership, or remote load misses. These external commands would be a new class of messages in the Search Network, and blocks would be evicted through the Replacement Network when necessary. In both cases, almost no extra wiring is required, and only the control logic has to be enhanced to process the new coherence messages.

Please note that by using the Search Network, successful commands in the r-tile do not propagate to the rest of tiles; however, other options for higher performance, complexity and consumption are possible. For instance, the r-tile can be enhanced with a duplicate tag port, devoted exclusively to coherence processing, such as in the exclusive L1 cache of the Opteron processors. Besides, by adding multiplexing and arbitration in the Search Network input it is possible to enter in parallel coherence commands into the r-tile duplicate tag and into the level 2 tiles. This may involve energy wasting because sometimes there are redundant checks, but coherence replies are faster for the L-NUCA blocks that are not in the r-tile.

Regarding Sectoring and Miss Wave Stopping, we can expect that coherence command processing will attain similar energy savings than processor demands. In both cases we are able to locate a block checking fewer tiles. Evaluating the trade-offs involved in designing a system with coherent L-NUCA is an interesting open work outside the scope of this dissertation. From our results we can guess that coherence traffic should not affect very much the performance of L-NUCA and even may be a cause of further improvement compared to a conventional

[1]The shared L2 cache of the UltraSPARC T2 System-on-a-Chip is an example of an inclusive shared LLC featuring a full-map scheme [117], the shared L2 cache of the low-power ARM Cortex-A9 MPCore processor is an example of snoopy last-level exclusive cache [49], and the shared L3 cache of the third-generation multi-core AMD Opteron processors is an example of a noninclusive cache featuring a limited directory scheme [25].

L1-L2 hierarchy. While on average the Search Network is idle[2] more than 97% of time (91% in the worst case, 179.art), the serial L2 cache is idle 83%, not accounting for the write misses refills that do not occur in the LP-NUCA because it is no-write-allocate.

## 7.2 Real-Time

In multitasking hard real-time systems, all tasks must complete execution before their deadlines. In order to perform a schedulability analysis is required to compute the worst case execution time (WCET) of each task and the effects of interferences between tasks as well. Several analysis tools have been proposed to compute useful upper WCET bounds in reasonable time based on modeling the dynamic cache behaviour, such as precise Worst Case Simulation [90], Integer Linear Programming methods [122], or Static Cache Simulation [112]. Nevertheless, computing tight WCET bounds of real-time applications executing with data cache hierarchies is still an open problem [142].

Regarding modeling their dynamic behavior, L-NUCAs resemble conventional high-associativity caches, with the added difficulty of the round-robin tile eviction. Sectored LP-NUCAs, by restricting block location may soften the modeling difficulty. Modeling and tunning the eviction policy is therefore one of the possible ways to use L-NUCAs in a real-time framework.

Regardless of the modeling tool two cache features may ease analysis and improve bounds: data locking and partitioning. *Data locking*, a widespread feature in embedded processors[3], disables cache replacement so that the cache content does not vary, avoiding both intra and inter-task interferences [136]. *Cache partitioning* assigns a portion of the cache to each task and restricts cache replacement to each partition, avoiding thus inter-task interferences [126].

LP-NUCAs accept in a natural way two partitioning and locking granularities, namely tile and sector. One or more tiles (or a sector) can be devoted to a given task, avoiding (or reducing) inter-task interferences. Furthermore, tiles can be preloaded and locked (at system start-up or at context switches) with blocks from one or more tasks. Both partitioning and locking in LP-NUCAs have a negligible hardware complexity and cost. Thus, either an automatic tool or the programmer could select, for instance, the data blocks to lock based in the latency offered by a given L-NUCA level. So, we think that LP-NUCAs with partitioning and locking capabilities are good candidates for real-time data caches, and their modeling and experimentation deserve further research.

[2] Number of cycles where the Search links between the r-tile and the second level tiles are idle divided by total number of cycles.

[3] Mainstream Instruction Set Architectures supporting whole cache-locking include, for instance, ARM (ARM940), MIPS (Integrated Device Technology IDT79RC64xxx), Motorola 68K (Freescale Coldfire), Power (Freescale MPC74xx, MPC8540, PowerPC 440 core, e300 core). Other processors support partial cache locking only (e.g. most ARM, IDT79RC46xx).

# Chapter 8

# Conclusions


### Summary

*This chapter concludes the dissertation putting together the main ideas of this work, which roughly correspond to each one of previous chapters. At the end of the chapter, we sketch some open questions deserving future work apart from the quantitive analysis of the topics from Chapter 7.*

Microprocessors have kept an incredible performance rise pace during the last forty years. Improvements in both technology and architecture have continuously fuelled this progress in what seemed an unstoppable process. Nevertheless, energy consumption and the extraction of more instruction level parallelism (ILP) have ballasted the pace, and emerged as the main factors to improve in order to return to the path of performance. Neither problems have a silver bullet solution and an answer for any of them requires a holistic approach in which diverse aspects such as programs, microarchitecture, implementation, and technology are taken into account.

In the microprocessor domain, the ultimate goal of most optimizations and features is to reduce execution time or to save energy without incurring in any penalty cost. Inside the processor, what they achieve is to keep busy functional units with correct operations for more time. The memory hierarchy is responsible for supplying both instructions and data to the control logic and the functionals units. Since the sixties, the memory hierarchy has been organized into levels that rise in latency and size as they get further apart from the processor.

The limits of ILP have driven the industry towards more coarse grained approaches such as Thread Level Parallelism (TLP) implemented in the form of on-chip multithreaded, or multiprocessing, or a combination of the two techniques. In both cases, the cache hierarchy plays a vital role, and, hence, cache hierarchies have been extensively studied during last years. The extraction of TLP requires the storage of instructions and data from multiple threads inside the processors because the cost of accessing main memory has turned too expensive due to the memory wall. Hence, many authors have focused on the organization and the management policies of large Last Level Caches (LLC). One approach for improving the performance of LLCs is the Non-Uniform Cache Architecture (NUCA) tha introduced a conventional 2D mesh inside the cache to migrate blocks intra banks.

In this work we have proposed a cache organization, named Light NUCA (L-NUCA), exploiting the non-uniform latency close to the processor. The idea is to reduce the latency gap between first and second cache levels and provide a fabric that offers a latency proportional to the size of the working set. We have observed that cache operations exhibit different communication patterns among them and that the differences can guide the design of specialized Networks-in-Cache. Three Networks-in-Cache are the core of the Light NUCA: one for lookups, another to convey hits to the execution core, and the last one to maintain blocks ordered by their locality.

We have proved the L-NUCA concept with a VLSI implementation. From the extracted layout, we have verified that: the Networks-in-Cache have a low overhead in terms of latency, area, and power, wide links (block sized) among tiles are not only desirable but possible, and to build L-NUCAs of any size just requires two tile layout variations, which connections match when placed side by side.

With a cycle-accurate simulation environment, energy and area estimations

from the extracted layout and well established models for current and future technologies, we have verified that L-NUCA provides latency, area, and energy advantages over conventional and static NUCA organizations regardless the LLC. These evaluations have been carried out in several application domains: uniprocessor, embedded, and simultaneous multithreading with equally good results in all three. To perform the simultaneous multithreading experiments, we have proposed a simple yet efficient methodology for accurately sampling mixes of threads in multiprogrammed workloads.

In order to fulfill the requirements of high-performance low power embedded processors, we have proposed a variant of the L-NUCA called Light Power NUCA (LP-NUCA) that includes proactive and reactive techniques that leverage the Networks-In-Cache to reduce dynamic consumption without degrading performance.

## 8.1 Publications

This section includes all publications of this dissertation, which have already been referenced in their corresponding chapters, with their collaborations where applicable, and also includes other works where we have been working at the same time.

### 8.1.1 Thesis Publications

- Darío Suárez, Teresa Monreal, and Víctor Viñals. Improvements on wire delay tolerant caches. In Proceedings of the 1$^{th}$ International Summer School on Advanced Computer Architecture and Compilation for Embedded Systems (ACACES'05).*Poster Abstracts.*July 2005. [125]

- Darío Suárez, Teresa Monreal, and Víctor Viñals. Improving Performance by Merging Cache Levels. Prooceedings of the 18$^{th}$ Jornadas de Paralelismo (JJPAR'07). September 2007. [123]

- Darío Suárez, Teresa Monreal, Fernando Vallejo, Ramón Beivide, and Víctor Viñals. Light NUCA: a proposal for bridging the inter-cache latency gap. In Prooceedings of the 12$^{th}$ Design, Automation and Test in Europe Conference and Exhibition (DATE'09), April 2009. In collaboration with the University of Cantabria. [124]

- Darío Suárez Gracia, Giorgos Dimitrakopoulos, Teresa Monreal Arnal, Manolis G.H. Katevenis, and Víctor Viñals Yúfera. LP-NUCA: Networks-in-Cache for high-performance low-power embedded processors. IEEE Transactions on Very Large Scale Integration (VLSI) Systems. May 2011. In collaboration with the Foundation for Research and Technology - Hellas. [41]

- Darío Suárez Gracia, Teresa Monreal Arnal, Víctor Viñals Yúfera. A Comparison of Cache Hierarchies for SMT Processors. Prooceedings of the 22th Jornadas de Paralelismo (JJPAR'11). September 2011. [40]

- Darío Suárez Gracia, Teresa Monreal Arnal, Víctor Viñals Yúfera. An Adaptive Controller to Save Dynamic Energy in LP-NUCA. Prooceedings of the 22th Jornadas de Paralelismo (JJPAR'11). September 2011. [39]

### Other Publications

Programmability:

- Abduallah Muzahid, Darío Suárez, Shanxiang Qi, and Josep Torrellas. SigRace: signature-based data race detection. Proceedings of the 36th Annual International Symposium on Computer architecture. June 2009. This work was performed during an internship at the University of Illinois at Urbana-Champaign. [102]

Power and Energy Education:

- Alicia Asín Pérez, Darío Suárez Gracia, and Víctor Viñals Yúfera. A proposal to introduce power and energy notions in computer architecture laboratories. Proceedings of the 2007 Workshop on Computer Architecture Education. *Best Paper Award*. June 2007. [3]

- Alicia Asín Pérez, Darío Suárez Gracia, and Víctor Viñals Yúfera. Introducing Energy and Power in Computer Architecture Laboratories. Prooceedings of the 18th Jornadas de Paralelismo (JJPAR'07). September 2007. [108]

- Sergio Gutiérrez Verde, Octavio Benedí Sánchez, Darío Suárez Gracia, Jose María Marín Herrero, and Víctor Viñals Yúfera. Forge: A Multi-purpose Platform for Measuring Energy and Temperature in Commodity PCs. Prooceedings of the 20th Jornadas de Paralelismo (JJPAR'09). September 2009. [45]

- Sergio Gutiérrez Verde, Octavio Benedí Sánchez, Darío Suárez Gracia, Jose María Marín Herrero, and Víctor Viñals Yúfera. Processor Energy and Temperature in Computer Architecture Courses: a hands-on approach. Proceedings of the 2009 Workshop on Computer Architecture Education. December 2009. [46]

Nanotechnologies:

- Nanotubos de Carbono para Conexiones en Caches: Arquitecturas más allá del CMOS. Jaime Ortiz, Darío Suárez, Víctor Viñals y María Villarroya. Proceedings of the 20th Jornadas de Paralelismo (JJPAR'09). September 2009. [105]

## 8.2 Future Work

As a future work, Light NUCAs offer plenty of options for continuing with such as:

- First, the inclusion of Light NUCAs in *chip multiprocessors*. Chapter 7 has qualitatively analyzed how symmetric multiprocessors can benefit from private L-NUCA, but a quantitative analysis would reinforce its conclusions. Another related subject of study is the sharing of a single L-NUCA because the current implementation only support one processor–r-tile.

- Second, *instruction caches*, current work has only addressed data caches leaving apart the fetch engine of the processor. Given that L-NUCA provides a lot of bandwidth at low latency, it will be interesting to study a dual root-tile organization, one for data and another for instructions. This organization is quite straight forward to implement because it only add a bit of complexity to the transport network in the r-tiles and to the miss status holding registers.

- Third, *fine grain tuning* with different purposes. This work has exploited neither per-level nor per-tile optimizations. As examples, we can think in proposing prefetchers that depending on the accuracy bring blocks to different L-NUCA levels or to look some data to a given tile in order to ensure a latency of service for hard real-time systems.

- Fourth, *root and rest tiles interface*, this work assumes that five tiles surround the root tile (second level tiles) and that only three of them service blocks. It would be very interesting to study the trade-off between number of tiles in the second L-NUCA level and their associated costs in terms of area, latency, and energy. For example, if all second level tiles had serviced blocks, the MSHR would have required more complex control policy and data multiplexer.

- Finally, *Fault Tolerance*, since the manufacturing costs of deep-submicron technologies are prohibitive, the liability of future processors may reduce and require fault tolerant organizations. From the current implementation of the Networks-in-Cache, only the transport network already can tolerate transitive and fix defects through the multiple paths that ensure path diversity. Hence, study other topologies and network policies for fault tolerance will add value to the L-NUCA.

# Bibliography


[1] David H. Albonesi. Selective cache ways: on-demand cache resource allocation. In *MICRO 32: Proceedings of the 32nd annual ACM/IEEE international symposium on Microarchitecture*, pages 248–259, Washington, DC, USA, 1999. IEEE Computer Society.

[2] ARM Limited. *ARM®946E-S Technical Reference Manual*, August 2000.

[3] Alicia Asín Pérez, Darío Suárez Gracia, and Víctor Viñals Yúfera. A proposal to introduce power and energy notions in computer architecture laboratories. In *WCAE '07: Proceedings of the 2007 workshop on Computer architecture education*, pages 52–57, New York, NY, USA, 2007. ACM.

[4] European Semiconductor Industry Association, Japan Electronics, Information Technology Industries Association, Korea Semiconductor Industry Association, Taiwan Semiconductor Industry Association, and Semiconductor Industry Association. International technology roadmap for semiconductors. 2010 update. executive summary. Technical report, INTERNATIONAL TECHNOLOGY ROADMAP FOR SEMICONDUCTORS, 2010.

[5] Todd Austin and Doug Burger. *SimpleScalar Tutorial (for tool set release 2.0)*. SimpleScalar LCC, 1997.

[6] Manu Awasthi, Kshitij Sudan, Rajeev Balasubramonian, and John Carter. Dynamic hardware-assisted software-controlled page placement to manage capacity allocation and sharing within large caches. In *In Proc. of the $15^{th}$ International Symposium on High-Performance Computer Architecture*, 2009.

[7] John Backus. Can programming be liberated from the von neumann style?: a functional style and its algebra of programs. *Commun. ACM*, 21:613–641, August 1978.

[8] Pend Bai, C. Auth, S. Balakrishnan, M. Bost, R. Brain, V. Chikarmane, R. Heussner, M.Hussein, J. Hwang, D. Ingerly, R. James, J. Jeong, C.Kenyon, E. Lee, S. H. Lee, N. Lindert, M. Liu, Z. Ma, T. Marieb, A. Murthy, R. Nagisetty, S. Natarajan, J. Neirynck, A. Ott, C.Parker, J. Sebastian, R. Shaheed,

S. Sivakumar, J. Steigerwald, S. Tyagi, C. Weber, B. Woolery, A. Yeoh, K. Zhang, and M.Bohr. A 65 nm logic technology featuring 35 nm gate length, enhanced channel strain, 8 Cu interconnect layers, low-k ILD and 0.57$\mu$m SRAM cell. In *Proceedings of the IEEE International Electron Devices Meeting*, 2004.

[9] H. B. Bakoglu. *Circuits, Interconnections, and Packaging for VLSI*. Addison-Wesley VLSI system series. Addison-Wesley, first edition, 1990.

[10] Rajeev Balasubramonian, David Albonesi, Alper Buyuktosunoglu, and Sandhya Dwarkadas. Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures. In *MICRO 33: Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture*, pages 245–257, New York, NY, USA, 2000. ACM Press.

[11] Arnab Banerjee, Robert Mullins, and Simon Moore. A power and energy exploration of network-on-chip architectures. In *Proceedings of the First International Symposium on Networks-on-Chip (NOCS'07)*, volume 00, pages 163–172, Los Alamitos, CA, USA, 2007. IEEE Computer Society.

[12] 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 *Proceedings of the 27th annual international symposium on Computer architecture*, ISCA '00, pages 282–293, New York, NY, USA, 2000. ACM.

[13] Bradford M. Beckmann and David A. Wood. Managing wire delay in large chip-multiprocessor caches. In *Proceedings of Micro37*, 2004.

[14] Luca Benini and Giovanni De Micheli. Networks on chips: A new soc paradigm. *IEEE Computer*, 35(1):70–78, January 2002.

[15] Michael Van Biesbrouck, Lieven Eeckhout, and Brad Calder. Representative multiprogram workloads for multithreaded processor simulation. In *IEEE Workload Characterization Symposium*, pages 193–203. IEEE Computer Society, September 2007.

[16] M. Bohr. The new era of scaling in an SoC world. In *Digest of Technical Papers of the Int'l Solid-State Circuits Conf. (ISSCC'09)*, pages 23 –28, 2009.

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

[18] B.H. Calhoun, Yu Cao, Xin Li, Ken Mai, L.T. Pileggi, R.A. Rutenbar, and K.L. Shepard. Digital circuit design challenges and opportunities in the era of nanoscale cmos. *Proc. of the IEEE*, 96(2):343 –365, 2008.

[19] 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.

[20] Mainak Chaudhuri. Pagenuca: Selected policies for page-grain locality management in large shared chip-multiprocessor caches. In *In Proceedings of the 15$^{th}$ IEEE International Symposium on High-Performance Computer Architecture*, pages 227–238, February 2009.

[21] Myong Hyon Cho, Mieszko Lis, Keun Sup Shim, Michel Kinsy, Tina Wen, and Srinivas Devadas. Oblivious routing in on-chip bandwidth-adaptive networks. In *PACT '09: Proceedings of the 2009 18th International Conference on Parallel Architectures and Compilation Techniques*, pages 181–190, Washington, DC, USA, 2009. IEEE Computer Society.

[22] Sangyeun Cho and Lei Jin. Managing distributed, shared l2 caches through os-level page allocation. In *MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 455–468, Washington, DC, USA, 2006. IEEE Computer Society.

[23] Seungryul Choi and Donald Yeung. Hill-climbing smt processor resource distribution. *ACM Trans. Comput. Syst.*, 27(1):1–47, 2009.

[24] Shu-Hsuan Chou, Chien-Chih Chen, Chi-Neng Wen, Yi-Chao Chan, Tien-Fu Chen, Chao-Ching Wang, and Jinn-Shyan Wang. No cache-coherence: a single-cycle ring interconnection for multi-core l1-nuca sharing on 3d chips. In *DAC '09: Proceedings of the 46th Annual Design Automation Conference*, pages 587–592, New York, NY, USA, 2009. ACM.

[25] Pat Conway, Nathan Kalyanasundharam, Gregg Donley, Kevin Lepak, and Bill Hughes. Cache hierarchy and memory subsystem of the amd opteron processor. *IEEE Micro*, 30:16–29, 2010.

[26] William J. Dally and Brian Towles. Route packets, not wires: on-chip inteconnection networks. In *Proc. of the 38$^{th}$ Conf. on Design Automation(DAC'01)*, 2001.

[27] William James Dally and Brian Patrick Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.

[28] R.H. Dennard, F.H. Gaensslen, V.L. Rideout, E. Bassous, and A.R. LeBlanc. Design of ion-implanted mosfet's with very small physical dimensions. *Solid-State Circuits, IEEE Journal of*, 9(5):256 – 268, October 1974.

[29] Peter J. Denning. The working set model for program behavior. *Commun. ACM*, 11:323–333, May 1968.

[30] Peter J. Denning. The locality principle. *Commun. ACM*, 48:19–24, July 2005.

[31] M. Ekman and P. Stenstrom. Enhancing multiprocessor architecture simulation speed using matched-pair comparison. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software, 2005*, pages 89–99, Washington, DC, USA, 2005. IEEE Computer Society.

[32] S. Eyerman and L. Eeckhout. System-level performance metrics for multiprogram workloads. *Micro, IEEE*, 28(3):42 –53, may. 2008.

[33] KrisztiÃąn Flautner, Nam Sung Kim, Steve Martin, David Blaauw, and Trevor Mudge. Drowsy caches: simple techniques for reducing leakage power. In *Proceedings of the 29th annual international symposium on Computer architecture*, pages 148–157. IEEE Computer Society, 2002.

[34] Jose Flich and Davide Bertozzi, editors. *Designing Network On-Chip Architectures in the Nanoscale Era*. Chapman & Hall/CRC Computational Science. CRC Press. Taylor & Francis Group, 2010.

[35] Pierfrancesco Foglia, Daniele Mangano, and Cosimo Antonio Prete. A nuca model for embedded systems cache design. In *Embedded Systems for Real-Time Multimedia, 2005. 3rd Workshop on*, September 2005.

[36] Ron Gabor, Shlomo Weiss, and Avi Mendelson. Fairness and throughput in switch on event multithreading. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 39, pages 149–160, Washington, DC, USA, 2006. IEEE Computer Society.

[37] Montse García, José González, and Antonio González. Data caches for multithreaded processors. In *Proc. of the Workshop on Multithreaded Execution, Architecture and Compilation*, 2000.

[38] Simcha Gochman, Avi Mendelson, Alon Naveh, and Efraim Rotem. Introduction to intel core duo processor architecture. *Intel Technology Journal*, 10:89–97, 2006.

[39] Darío Suárez Gracia, Teresa Monreal Arnal, , and Víctor Viñals Yúfera. An adaptive controller to save dynamic energy in lp-nuca. In *Proc. of the 22$^{th}$ Jornadas de Paralelismo (JJPAR'11)*, 2011.

[40] Darío Suárez Gracia, Teresa Monreal Arnal, and Víctor Viñals Yúfera. A comparison of cache hierarchies for smt processors. In *Proc. of the 22$^{th}$ Jornadas de Paralelismo (JJPAR'11)*, 2011.

[41] Darío Suárez Gracia, Giorgos Dimitrakopoulos, Teresa Monreal Arnal, Manolis G.H. Katevenis, and Víctor Viñals Yúfera. LP-NUCA: Networks-in-Cache for high-performance low-power embedded processors. *to appear in IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2011.

[42] Paul Gratz, Changkyu Kim, Robert McDonald, Stephen W. Keckler, , and Doug Burger. Implementation and evaluation of on-chip network architectures. In *Proceedings of ICCD'06*, 2006.

[43] B. Greene, Q. Liang, K. Amarnath, Y. Wang, J. Schaeffer, M. Cai, Y. Liang, S. Saroop, J. Cheng, A. Rotondaro, S.-J. Han, R. Mo, K. McStay, S. Ku, R. Pal, M. Kumar, B. Dirahoui, B. Yang, F. Tamweber, W.-H. Lee, M. Steigerwalt, H. Weijtmans, J. Holt, L. Black, S. Samavedam, M. Turner, K. Ramani, D. Lee, M. Belyansky, M. Chowdhury, D. Aime, B. Min, H. van Meer, H. Yin, K. Chan, M. Angyal, M. Zaleski, O. Ogunsola, C. Child, L. Zhuang, H. Yan, D. Permanaa, J. Sleight, D. Guo, S. Mittl, D. Ioannou, E. Wu, M. Chudzik, D.-G. Park, D. Brown, S. Luning, D. Mocuta, E. Maciejewski, K. Henson, and E. Leobandung. High performance 32 nm SOI CMOS with high-k/metal gate and 0.149 $mm^2$ SRAM and ultra low-k back end with eleven levels of copper. In *Proc. of the 2009 Symp. on VLSI Technology*, pages 140 –141, 2009.

[44] Richard Grisenthwaite. ARM cortex A8 processor. In *IEE Cambridge One Day Seminar. Processor Cores Putting Silicon IP to Work*, pages 1–25, 2005.

[45] Sergio Gutiérrez Verde, Octavio Benedí Sánchez, Darío Suárez Gracia, Jose María Marín Herrero, and Víctor Viñals Yúfera. Forge: A multi-purpose platform for measuring energy and temperature in commodity pcs. In *Proceedings of the $20^{th}$ Jornadas de Paralelismo (JJPAR'09)*, 2009.

[46] Sergio Gutiérrez Verde, Octavio Benedí Sánchez, Darío Suárez Gracia, Jose María Marín Herrero, and Víctor Viñals Yúfera. Processor energy and temperature in computer architecture courses: a hands-on approach. In *Proceedings of the 2009 Workshop on Computer Architecture Education*, 2009.

[47] Tom R. Halfhill. Freescale P5 raises QorIQ's I.Q. *Microprocessor Report*, 24(7):15–22, 2010.

[48] Tom R. Halfhill. Netlogic broadens XLP family. *Microprocessor Report*, 24(7):1–11, 2010.

[49] Tom R. Halfhill. The rise of licensable SMP. *Microprocessor Report*, 24(2):11–18, 2010.

[50] Greg Hamerly, Erez Perelman, Jeremy Lau, and Brad Calder. Simpoint 3.0: Faster and more flexible program analysis. In *Proceedings of Workshop on Modeling, Benchmarking and Simulation*, 2005.

[51] 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.

[52] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, fourth edition edition, 2007.

[53] John L. Henning. SPEC CPU2000: Measuring cpu performance in the new millennium. *Computer*, 33(7):28–35, 2000.

[54] John L. Henning. Spec cpu2006 benchmark descriptions. *SIGARCH Comput. Archit. News*, 34(4):1–17, 2006.

[55] Sébastien Hily and André Seznec. Contention on $2^{nd}$ level cache may limit the effectiveness of simultaneous multithreading. Technical Report 1086, IRISA, février 1997.

[56] Sébastien Hily and André Seznec. Standard memory hierarchy does not fit simultaneous multithreading. In *Proceedings of the $2^{nd}$ Workshop on MULTI-THREADED EXECUTION, ARCHITECTURE and COMPILATION (MTEAC-2)*, 1998.

[57] Glenn Hinton, Dave Sager, Mike Upton, Darrell Boggs, Doug Carmean, Alan Kyker, and Patrice Roussel. The microarchitecture of the Pentium® 4 processor. *Intel Technology Journal*, 1st quarter:1–13, 2001.

[58] Ron Ho, Kenneth W. Mai, and Mark A. Horowitz. The future of wires. *Proceedings of the IEEE*, 89(4):490–504, April 2001.

[59] M. Horowitz, T. Indermaur, and R. Gonzalez. Low-power digital design. In *Low Power Electronics, 1994. Digest of Technical Papers., IEEE Symposium*, pages 8 –11, October 1994.

[60] Michael Huang, Jose Renau, Seung-Moon Yoo, and Josep Torrellas. L1 data cache decomposition for energy efficiency. In *Proceedings of the 2001 international symposium on Low power electronics and design*, pages 10–15. ACM Press, 2001.

[61] Jaehyuk Huh, Changkyu Kim, Hazim Shafi, Lixin Zhang, Doug Burger, and Stephen W. Keckler. A nuca substrate for flexible cmp cache sharing. In *ICS '05: Proceedings of the 19th annual international conference on Supercomputing*, pages 31–40, New York, NY, USA, 2005. ACM Press.

[62] Intel Corporation. Intel® Core™2 Quad Processor Q9550 specifications, `http://processorfinder.intel.com/details.aspx?sSpec=SLAWQ`.

[63] Intel Corporation. Intel® Core™2 Duo Processor E8600, `http://ark.intel.com/cpu.aspx?groupId=35605`.

[64] Intel Embedded. Intel® Xeon® processor C5500/C3500 series. Datasheet–Volume 1, February 2010.

[65] International Business Machines Corporation. *PowerPC 476FP L2 Cache Core Databook. Version 1.3 Preliminary*, 2011.

[66] Raj Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley & Sons, Inc., April 1991.

[67] Yuho Jin, Eim Jung Kim, and Ki Hwan Yum. A domain-specific on-chip network design for large scale cache systems. In *High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on*, 2007.

[68] Norman P. Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *ISCA '90: Proceedings of the 17th annual international symposium on Computer Architecture*, pages 364–373, New York, NY, USA, 1990. ACM Press.

[69] Norman P. Jouppi. Cache write policies and performance. In *ISCA '93: Proceedings of the 20th annual international symposium on Computer architecture*, pages 191–201, New York, NY, USA, 1993. ACM Press.

[70] Norman P. Jouppi. The future evolution of high-performance microprocessors. keynote. In *Proceedings of the 38$^{th}$ annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 38, pages 155–, Washington, DC, USA, 2005. IEEE Computer Society.

[71] Ron Kalla, Balaram Sinharoy, William J. Starke, and Michael Floyd. Power7: Ibm's next-generation server processor. *IEEE Micro*, 30:7–15, 2010.

[72] Ron Kalla, Balaram Sinharoy, and Joel M. Tendler. IBM Power5 chip: A dual-core multithreaded processor. *IEEE Micro*, 24(2):40–47, March/April 2004.

[73] Milind B. Kamble and Kanad Ghose. Analytical energy dissipation models for low-power caches. In *Proceedings of the 1997 international symposium on Low power electronics and design*, pages 143–148. ACM Press, 1997.

[74] Stefanos Kaxiras, Zhigang Hu, and Margaret Martonosi. Cache decay: exploiting generational behavior to reduce cache leakage power. In *Proceedings of the 28th annual international symposium on Computer architecture*, pages 240–251. ACM Press, 2001.

[75] R. E. Kessler. The alpha 21264 microprocessor. *IEEE Micro*, 19(2):24–36, 1999.

[76] Brucek Khailany *et al.* Exploring the VLSI scalability of Stream processors. In *Proc. of the $9^{th}$ Int'l Symp. on High-Performance Computer Arch.(HPCA'03)*, 2003.

[77] Changkyu Kim, Doug Burger, and Stephen W. Keckler. An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches. In *Proceedings of the 10th international conference on architectural support for programming languages and operating systems (ASPLOS-X)*, pages 211–222. ACM Press, October 2002.

[78] Johnson Kin, Munish Gupta, and William H. Mangione-Smith. The filter cache: an energy efficient memory structure. In *Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture*, pages 184–193. IEEE Computer Society, 1997.

[79] David Kroft. Lockup-free instruction fetch/prefetch cache organization. In *ISCA '81: Proceedings of the 8th annual symposium on Computer Architecture*, pages 81–87. IEEE Computer Society Press, 1981.

[80] Amit Kumar, Partha Kundu, Arvind P. Singh, Li-Shiuan Pehy, and Niraj K. Jha. A 4.6tbits/s 3.6ghz single-cycle noc router with a novel switch allocator in 65nm cmos. In *Proceedings of the 25th International Conference on Computer Design*, 2007.

[81] Hantak Kwak, Ben Lee, Ali R. Hurson, Suk-Han Yoon, and Woo-Jong Hahn. Effects of multithreading on cache performance. *IEEE Transactions on Computers*, 48:176–184, 1999.

[82] H. Q. Le, W. J. Starke, J. S. Fields, F. P. O'Connell, D. Q. Nguyen, B. J. Ronchetti, W. M. Sauer, E. M. Schwarz, and M. T. Vaden. IBM POWER6 microarchitecture. *IBM Journal of Research and Development*, 51:639–662, 2007.

[83] Yingmin Li, David Brooks, Zhigang Hu, Kevin Skadron, and Pradip Bose. Understanding the energy efficiency of simultaneous multithreading. In *Proceedings of the 2004 international symposium on Low power electronics and design*, ISLPED '04, pages 44–49, New York, NY, USA, 2004. ACM.

[84] J. S. Liptay. Structural aspects of the system/360 model 85: Ii the cache. *IBM Syst. J.*, 7:15–21, March 1968.

[85] Javier Lira, Carlos Molina, and Antonio Gonzalez. Lru-pea: A smart replacement policy for non-uniform cache architectures on chip multiprocessors. In *IEEE International Conference on Computer Design, 2009. ICCD 2009.*, pages 275 –281, oct. 2009.

[86] Javier Lira, Carlos Molina, and Antonio González. The auction: optimizing banks usage in non-uniform cache architectures. In *ICS '10: Proceedings of*

*the 24th ACM International Conference on Supercomputing*, pages 37–47, New York, NY, USA, 2010. ACM.

[87] Sonia López, Steve Dropsho, David H. Albonesi, Oscar Garnica, and Juan Lanchares. Dynamic capacity-speed tradeoffs in smt processor caches. In *Proceedings of the 2nd international conference on High performance embedded architectures and compilers*, HiPEAC'07, pages 136–150, Berlin, Heidelberg, 2007. Springer-Verlag.

[88] Sonia Lopez, Oscar Garnica, David H. Albonesi, Steven Dropsho, Juan Lanchares, and Jose I. Hidalgo. Adaptive cache memories for smt processors. *Digital Systems Design, Euromicro Symposium on*, 0:331–338, 2010.

[89] LSI Corporation. PowerPC™ processor (476FP) embedded core product brief, `http://www.lsi.com/DistributionSystem/AssetDocument/PPC476FP-PB-v7.pdf`, January 2010.

[90] Thomas Lundqvist and Per Stenström. A method to improve the estimated worst-case performance of data caching. In *Proc. of the 6$^{th}$ Int'l Conf. on Real-Time Computing Systems and Applications (RTCSA'99)*, pages 255–262, 1999.

[91] Deborah T. Marr. *MICROARCHITECTURE CHOICES AND TRADEOFFS FOR MAXIMIZING PROCESSING EFFICIENCY*. PhD thesis, The University of Michigan (Computer Science and Engineering), 2008.

[92] Cameron McNairy and Rohit Bhatia. Montecito: A Dual-Core, Dual-Thread Itanium Processor. *IEEE Micro*, 25(02):10–20, 2005.

[93] J. Merino, V. Puente, and J.A. Gregorio. Esp-nuca: A low-cost adaptive non-uniform cache architecture. In *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, pages 1 –10, 2010.

[94] Javier Merino, Valentín Puente, Pablo Prieto, and José Ángel GregorioSP. Sp-nuca: a cost effective dynamic non-uniform cache architecture. *SIGARCH Comput. Archit. News*, 36(2):64–71, 2008.

[95] George Michelogiannakis, Dionisios Pnevmatikatos, and Manolis Katevenis. Approaching ideal noc latency with pre-configured routes. In *NOCS '07: Proceedings of the First International Symposium on Networks-on-Chip*, pages 153–162, Washington, DC, USA, 2007. IEEE Computer Society.

[96] MIPS Technologies. MIPS32® 1004K™coherent processing system (CPS), 2010.

[97] G.E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8):114–117, 1965.

[98] Robert Mullins, Andrew West, and Simon Moore. Low-latency virtual-channel routers for on-chip networks. In *ISCA '04: Proceedings of the 31st annual international symposium on Computer architecture*, page 188, Washington, DC, USA, 2004. IEEE Computer Society.

[99] Naveen Muralimanohar and Rajeev Balasubramonian. Interconnect design considerations for large nuca caches. In *ISCA '07: Proceedings of the 34th annual international symposium on Computer architecture*, pages 369–380, New York, NY, USA, 2007. ACM Press.

[100] Naveen Muralimanohar, Rajeev Balasubramonian, and Norm P. Jouppi. Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0. In *MICRO 40: Proceedings of the 40$^{nd}$ annual ACM/IEEE international symposium on Microarchitecture*. IEEE Computer Society, 2007.

[101] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P. Jouppi. CACTI 6.0: A tool to model large caches. Technical Report HPL-2009-85, HP Laboratories, April 2009.

[102] Abdullah Muzahid, Dario Suárez, Shanxiang Qi, and Josep Torrellas. Sigrace: signature-based data race detection. In *Proceedings of the 36th annual international symposium on Computer architecture*, ISCA '09, pages 337–348, New York, NY, USA, 2009. ACM.

[103] U. G. Nawathe, M. Hassan, K. C. Yen, A. Kumar, A. Ramachandran, and D. Greenhill. Implementation of an 8-core, 64-thread, power-efficient sparc server on a chip. *IEEE Journal of Solid-State circuits*, 43(1):6–20, 2008.

[104] Mario Nemirovsky and Wayne Yamamoto. Quantitative study of data caches on a multistreamed architecture. In *In Workshop on Multithreaded Execution, Architecture and Compilation*, 1998.

[105] Jaime Ortiz, Darío Suárez, nals Víctor Vi and María Villarroya. Nanotubos de carbono para conexiones en cachÃl's: Arquitecturas más allá del cmos. In *Proceedings of the 20$^{th}$ Jornadas de Paralelismo (JJPAR'09)*, 2009.

[106] P. Packan, S. Akbar, M. Armstrong, D. Bergstrom, M. Brazier, H. Deshpande, K. Dev, G. Ding, T. Ghani, O. Golonzka, W. Han, J. He, R. Heussner, R. James, J. Jopling, C. Kenyon, S.-H. Lee, M. Liu, S. Lodha, B. Mattis, A. Murthy, L. Neiberg, J. Neirynck, S. Pae, C. Parker, L. Pipes, J. Sebastian, J. Seiple, B. Sell, A. Sharma, S. Sivakumar, B. Song, A. St. Amour, K. Tone, T. Troeger, C. Weber, K. Zhang, Y. Luo, and S. Natarajan. High performance 32 nm logic technology featuring 2$^{nd}$ generation high-k + metal gate transistors. In *Proc. of the 2009 Int'l Electron Devices Meeting (IEDM'09)*, pages 1–4, 2009.

[107] Li-Shiuan Peh and William J. Dally. A delay model and speculative architecture for pipelined routers. In *HPCA '01: Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, page 255, Washington, DC, USA, 2001. IEEE Computer Society.

[108] Alicia Asín Pérez, Darío Suárez Gracia, and Víctor Viñals Yúfera. Introducing energy and power in computer architecture laboratories. In *Prooceedings of the 18th Jornadas de Paralelismo (JJPAR'07)*, 2007.

[109] Aashish Phansalkar, Ajay Joshi, and Lizy K. John. Analysis of redundancy and application balance in the spec cpu2006 benchmark suite. In *ISCA '07: Proceedings of the 34th annual international symposium on Computer architecture*, pages 412–423, New York, NY, USA, 2007. ACM.

[110] Donald W. Plass and Yuen H. Chan. IBM POWER6 SRAM arrays. *IBM Journal of Research and Development*, 51(6):747–756, November 2007.

[111] Steven E. Raasch and Steven K. Reinhardt. The impact of resource partitioning on smt processors. In *Proceedings of the 12th International Conference on Parallel Architectures and Compilation Techniques*, PACT '03, pages 15–, Washington, DC, USA, 2003. IEEE Computer Society.

[112] Harini Ramaprasad and Frank Mueller. Bounding preemption delay within data cache reference patterns for real-time tasks. In *Proc. of the 12th Real-Time and Embedded Technology and Applications Symp. (RTAS'06)*, pages 71–80, 2006.

[113] Karthikeyan Sankaralingam, Ramadass Nagarajan, Robert McDonald, Rajagopalan Desikan, Saurabh Drolia, M. S. Govindan, Paul Gratz, Divya Gulati, Heather Hanson, Changkyu Kim, Haiming Liu, Nitya Ranganathan, Simha Sethumadhavan, Sadia Sharif, Premkishore Shivakumar, Stephen W. Keckler, and Doug Burger. Distributed microarchitectural protocols in the trips prototype processor. In *MICRO '06: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 480–491, Washington, DC, USA, 2006. IEEE Computer Society.

[114] Subhradyuti Sarkar and Dean M. Tullsen. Data layout for cache performance on a multithreaded architecture. In Per Stenström, editor, *Transactions on high-performance embedded architectures and compilers III*, chapter Data layout for cache performance on a multithreaded architecture, pages 43–68. Springer-Verlag, Berlin, Heidelberg, 2011.

[115] Alex Settle, Dan Connors, Enric Gibert, and Antonio González. A dynamically reconfigurable cache for multithreaded processors. *J. Embedded Comput.*, 2(2):221–233, 2006.

[116] André Seznec. A case for two-way skewed-associative caches. In *ISCA '93: Proceedings of the 20th annual international symposium on Computer architecture*, pages 169–178, New York, NY, USA, 1993. ACM Press.

[117] M. Shah, J. Barren, J. Brooks, R. Golla, G. Grohoski, N. Gura, R. Hetherington, P. Jordan, M. Luttrell, C. Olson, B. Sana, D. Sheahan, L. Spracklen, and A. Wynn. Ultrasparc t2: A highly-treaded, power-efficient, sparc soc. In *Proc. of the Asian Solid-State Circuits Conference (ASSCC'07)*, pages 22 –25, 2007.

[118] J. L. Shin, K. Tam, D. Huang, B. Petrick, H. Pham, Changku Hwang, Hongping Li, A. Smith, T. Johnson, F. Schumacher, D. Greenhill, A. S. Leon, and A. Strong. A 40nm 16-core 128-thread cmt sparc soc processor. In *Proc. IEEE Int. Solid-State Circuits Conf. Digest of Technical Papers (ISSCC)*, pages 98–99, 2010.

[119] Abraham Silberschatz, Peter B. Galvin, and Greg Gagne. *Operating System Concepts*. Wiley, 2009.

[120] Kevin Skadron and Douglas W. Clark. Design issues and tradeoffs for write buffers. In *HPCA '97: Proceedings of the 3rd IEEE Symposium on High-Performance Computer Architecture*, page 144, Washington, DC, USA, 1997. IEEE Computer Society.

[121] Alan Jay Smith. Cache memories. *ACM Comput. Surv.*, 14(3):473–530, 1982.

[122] Jan Staschulat and Rolf Ernst. Worst case timing analysis of input dependent data cache behavior. In *Proc. of the 18$^{th}$ Euromicro Conf. on Real-Time Systems*, pages 227–236, 2006.

[123] Darío Suárez, Teresa Monreal, , and nals Víctor Vi˙ Improving performance by merging cache levels. In *Proc. of the 18$^{th}$ Jornadas de Paralelismo (JJPAR'07)*, 2007.

[124] Darío Suárez, Teresa Monreal, Fernando Vallejo, Ramón Beivide, and Víctor Viñals. Light NUCA: a proposal for bridging the inter-cache latency gap. In *Prooceedings of the 12$^{th}$ Design, Automation and Test in Europe Conference and Exhibition (DATE'09)*, April 2009.

[125] Darío Suárez, Teresa Monreal, and Víctor Viñals. Improvements on wire delay tolerant caches. In *Proceedings of ACACES 2005, poster abstracts*, July 2005.

[126] Vivy Suhendra and Tulika Mitra. Exploring locking & partitioning for predictable shared caches on multi-cores. In *Proc. of the 45$^{th}$ Annual Design Automation Conf. (DAC'08)*, pages 300–303, 2008.

[127] Karthik T. Sundararajan, Timothy M. Jones, and Nigel Topham. Smart cache: A self adaptive cache architecture for energy efficiency. In *Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, 2011.

[128] David Tarjan, Shyamkumar Thoziyoor, and Norman P. Jouppi. Cacti 4.0. Technical Report HPL-2006-86, HP Laboratories Palo Alto, June 2006.

[129] Michael Bedford Taylor, Walter Lee, Saman Amarasinghe, and Anant Agarwal. Scalar operand networks: On-chip interconnect for ilp in partitioned architectures. In *HPCA '03: Proceedings of the 9th International Symposium on High-Performance Computer Architecture*, page 341, Washington, DC, USA, 2003. IEEE Computer Society.

[130] J. M. Tendler, J. S. Dodson, J. S. Fields, Jr., H. Le, and B. Sinharoy. POWER4 system microarchitecture. *IBM Journal of Research and Development*, 46(1):5–25, 2002.

[131] Shyamkumar Thoziyoor, Naveen Muralimanohar, and Norman P. Jouppi. Cacti 5.0. Technical Report HPL-2007-167, HP Labs, October 2007.

[132] E. F. Torres, P. Ibanez, V. Vinals, and J. M. Llaberia. Store buffer design in first-level multibanked data caches. In *Proceedings of the 32nd annual international symposium on Computer Architecture*, ISCA '05, pages 469–480, Washington, DC, USA, 2005. IEEE Computer Society.

[133] Dean M. Tullsen and Jeffery A. Brown. Handling long-latency loads in a simultaneous multithreading processor. In *MICRO 34: Proceedings of the 34th annual ACM/IEEE international symposium on Microarchitecture*, pages 318–327, Washington, DC, USA, 2001. IEEE Computer Society.

[134] Dean M. Tullsen, Susan J. Eggers, Joel S. Emer, Henry M. Levy, Jack L. Lo, and Rebecca L. Stamm. Exploiting choice: instruction fetch and issue on an implementable simultaneous multithreading processor. In *Proceedings. 23nd Annual International Symposium on Computer Architecture*, pages 191–202, New York, NY, USA, 1996. ACM.

[135] D.M. Tullsen, S.J. Eggers, and H.M. Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In *Proceedings. 22nd Annual International Symposium on Computer Architecture*, pages 392–403, Jun 1995.

[136] Xavier Vera, Björn Lisper, and Jingling Xue. Data cache locking for higher program predictability. In *Proc. of the Int'l Conf. on Measurement and modeling of computer systems (SIGMETRICS'03)*, pages 272–282, 2003.

[137] Scott Vetter, Ste phen Behling, Pe ter Farrell, Holger Holthoff, Frank O'Connell, and Will Weir. *The POWER4 Processor Introduction and Tuning*

*Guide*. IBM International Technical Support Organization, Dept. JN9B Building 003 Internal Zip 2834. 11400 Burnet Road. Austin, Texas 78758-3493, 1 edition, November 2001.

[138] Hang-Sheng Wang, Xinping Zhu, Li-Shiuan Peh, and Sharad Malik. Orion: a power-performance simulator for interconnection networks. In *MICRO 35: Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture*, pages 294–305, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[139] Hangsheng Wang, Li-Shiuan Peh, and Sharad Malik. Power-driven design of router microarchitectures in on-chip networks. In *MICRO 36: Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, page 105, Washington, DC, USA, 2003. IEEE Computer Society.

[140] 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*, 26:18–31, July 2006.

[141] D. Wentzlaff, P. Griffin, H. Hoffmann, Liewei Bao, B. Edwards, C. Ramey, M. Mattina, Chyi-Chang Miao, J.F. Brown, and A. Agarwal. On-chip interconnection architecture of the tile processor. *Micro, IEEE*, 27(5):15–31, Sept.-Oct. 2007.

[142] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution-time problem–overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 7:36:1–36:53, 2008.

[143] M.V. Wilkes. Slave memories and dynamic storage allocation. *Electronic Computers, IEEE Transactions on*, EC-14(2):270–271, April 1965.

[144] M.V. Wilkes. High performance memory systems. *Computers, IEEE Transactions on*, 50(11):1105, November 2001.

[145] Wm. A. Wulf and Sally A. McKee. Hitting the memory wall: implications of the obvious. *SIGARCH Comput. Archit. News*, 23:20–24, March 1995.

[146] John Wuu, Don Weiss, Charles Morganti, and Michael Dreesen. The Asynchronous 24 MB On-Chip Level 3 Cache for a Dual-core Itanium architecture processor. In *Proceedings of the International Solid State Circuits Conf. Digest of Technical Papers*, 2005.

[147] Byung-Do Yang and Lee-Sup Kim. A low-power SRAM using hierarchical bit line and local sense amplifiers. *IEEE Journal of Solid-State circuits*, 40(6):1366–1376, June 2005.

[148] Chuanjun Zhang, Frank Vahid, and Walid Najjar. A highly configurable cache architecture for embedded systems. In *ISCA '03: Proceedings of the 30th annual international symposium on Computer architecture*, pages 136–146, New York, NY, USA, 2003. ACM Press.

[149] Michael Zhang and Krste Asanović. 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. Recomendado por JM.