CHIP MULTIPROCESSORS FOR SERVER WORKLOADS

NIKOLAOS HARDAVELLAS

CMU-CS-09-150

JULY 2009

SCHOOL OF COMPUTER SCIENCE

COMPUTER SCIENCE DEPARTMENT

CARNEGIE MELLON UNIVERSITY

Thesis Committee:

Babak Falsafi, co-Chair

Anastasia Ailamaki, co-Chair

David R. O'Hallaron

Todd C. Mowry

Luiz André Barroso (Google)

Submitted in partial fulfillment of the requirements

for the degree of Doctor of Philosophy

This research was sponsored by equipment donations from Intel, two Sloan research fellowships, an ESF European Young Investigator award, an IBM faculty partnership award, and the National Science Foundation under grants CCR-0205544, CCF-0702658, CCR-0509356, CCF-0845157, IIS-0133686, and IIS-0713409. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: computer architecture, cache, multicore, chip multiprocessors, data placement, chip design, NUCA, commercial server workloads, performance modeling, design-space exploration.

to our family

to Jasmine

to Kivanc
where you are, is paradise

Abstract

We stand on the cusp of the giga-scale era of chip integration. Technological advancements in semiconductor fabrication yield ever-smaller and faster devices, enabling billion-transistor chips with multi-gigahertz clock frequencies. To utilize the abundant transistors on chip, modern processors pack an exponentially increasing number of cores on chip, multi-megabyte caches, and large interconnects to facilitate intra-chip data transfers. However, the growing on-chip resources do not directly translate into a commensurate increase in performance. Rather, they come at the cost of increased on-chip data access latency, while thermal considerations and pin constraints limit the parallelism that a multicore chip can support.

To mitigate the increasing on-chip data access latency, cache blocks on chip should be placed close to the cores that use them. We observe that cache access patterns can be classified at run time into distinct classes with different on-chip block placement requirements. Based on this observation, we propose Reactive NUCA (R-NUCA), a distributed cache design which reacts to the class of each access to place blocks close to the requesting cores. We then explore the design space of physically-constrained multicore processors, and find that future multicores should utilize low-operational-power transistors even for time-critical components (e.g., cores) to ease the power wall, employ novel on-chip block placement techniques to utilize efficiently large caches, while techniques like 3D-stacked memory can mitigate the off-chip bandwidth constraint even for peak-performance designs. Moving forward, we find that heterogeneous multicores hold great promise in improving designs even further.

Acknowledgements

Any success I have had, I owe to my advisors, Babak Falsafi and Anastasia Ailamaki. They taught me how to think and do research, how to speak and write. Their penetrating insights have been a source of inspiration for this work. But, foremost, they have been my dear friends; always eager to help, always there when I need them the most. Thank you!

I am grateful to my thesis committee, Dave O'Hallaron, Todd Mowry, and Luiz André Barroso. Their support has been unwavering, and their comments and suggestions have improved this work tremendously. I thank you all for your collaboration and gracious help, and for making the thesis process such a pleasurable experience. I also owe special thanks to Andreas Moshovos for generously devoting a lot of his time to me.

I wouldn't have gone into research, if it wasn't for Marios Mavronicolas, an amazing teacher and great friend who took me under his wings early in my student life and offered me my first research experience. Thankfully for the field of theoretical computer science, but sadly for computer architecture, later I decided to switch research areas.

There are a lot of people who supported me at Carnegie Mellon, and without their help this work would never be possible. They have all been my mentors and my dear friends. Brian Gold tolerated me as an office-mate, and provided invaluable input and support, both for my work as well as outside the office. Stephen Somogyi and Mike Ferdman put countless hours towards helping me decipher results and polish papers and talks. Tom Wenisch made our research infrastructure

possible. Jared Smolens often put his incredible skills to the help of others. Eric Chung was always ready for long and deep research discussions. Thank you all for everything.

If it was not for my wife, Kivanc Sabirli, I would have truly lost my mind at various stages of my thesis work. Her unconditional support kept me going; her infinite love put a purpose in my life and reminded me of what truly matters; her smile gave me power to hike the thesis mountain. Kivanc, I couldn't have done it without you. You are my home. Long time [sic].

I've watched countless dawns with Jasmine and Lilly after a night of hard work, listening to birds upon the first break of the morning light. They kept me company in the wee hours, Jasmine sitting on my chair or on the warm laptop, Lilly laying on the keyboard when it was time for a break. Jasmine and Lilly, thank you for your warm company. You'll always be with us.

This section would never be complete without expressing my sincere gratitude to our family. The unconditional love and support of Yiannis, Yiota, and Themis pulled me through rough seas and taught me the value of education. This dissertation is as much their accomplishment, as it is mine. The enormous hospitality and love of Hikmet, Semra, Isik and Erhan warmed my heart when I needed it the most. I can only aspire sometime to be as gentle and thoughtful as they are.

I am grateful to Leonidas Kontothanassis and Cindy Arens. Their friendship and encouragement has been instrumental throughout my life. They are among the most generous people I have ever met, and I owe them more than I can ever put in words. Special thanks are also due to Alexandros Labrinidis. His hospitality allowed me to proceed undeterred in the last stages of the thesis, and his friendship, mentoring and support have helped me tremendously throughout my career.

Sotiris Damouras has been the ideal house-mate. His friendship and support has filled our life in Pittsburgh with warm memories, and our breaks at the porch provided a much-needed

respite. Vagelis Vlachos and Michael Papamichael helped me put work aside once in a while and enjoy life, and supported me a lot at the last stages of this effort. Takis, Maria, Tina, Panos, Areti, Kostas, Ioanna, Manos, Liz, Eleni, Tania, Yiannis, Pantelis and Ippokratis have always opened their houses to us and hosted parties and cook-outs that cleared my head and recharged my mind. Thank you all for helping me keep my sanity.

I wish to thank the roasters at Vivace for their amazing coffee, the people at Isomac for their great coffee machines, and at Mazzer for their grinders. Matt Koeske for introducing me to the culture of espresso aficionados and fine teas. The A-level faculty for making sure my cup was always full. The Brothers K Coffee House in Evanston, IL for providing me with an office during the last month of the thesis writing, and for making sure my cup remained bottomless. James Hoe for the Friday lunch series that kept us all connected, and for his advise and support whenever requested.

There are a lot more people to thank, and a hundred pages would not suffice to list them all. Please, permit me to thank you all at once and take this acknowledgement as an invitation to toast yourselves, wherever you might be.

I sincerely hope you enjoy reading.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Commercial server software systems (e.g., database management systems, web servers) are at the center of a multi-billion dollar server industry, utilizing state-of-the-art processors to maximize performance. Over the past decades, processor designs focused primarily on improving performance by exploiting instruction-level parallelism (ILP). The resulting wide-issue out-of-order (OoO) processors overlap both computation and memory accesses to increase performance, but fall short of realizing their full potential when running server workloads. Many important server workloads exhibit large instruction footprints and tight data dependencies that reduce instruction-level parallelism and incur data and instruction transfer delays [2,79,93,101].

Thus, increasingly aggressive OoO techniques yield diminishing returns in performance, while their power dissipation is reaching prohibitive levels [14]. The shortcomings of out-of-order processors, along with the continued increase in the number of transistors available on chip, have encouraged most vendors to integrate multiple processors on a single chip, instead of simply increasing the complexity of individual cores. Major manufacturers already ship 8-core chip multiprocessors [61] with plans to scale to 100s of cores [8,86]. Specialized vendors already push the envelope further, with Cisco CRS-1 featuring 188 processing cores, Tilera's TILE64 with 64 cores, and Azul Vega 3 with 54 out-of-order cores.

The resulting chip multiprocessor (CMP) designs may increase processor stalls through promoting on-chip data sharing across cores and increasing contention for shared hardware resources. However, to date, there hasn't been a comprehensive study that examines how the emerging hardware trends affect the performance of server workloads.

1.1 Performance Analysis of Modern CMPs

Recent studies show that processors are far from realizing their maximum performance when executing commercial server workloads. Prior research [75, 101] indicates that the adverse memory access patterns in commercial server workloads result in poor cache locality and overall performance. Server workloads are known to spend at least half of their execution time on data-access stalls, implying that high-performance systems should focus on bringing data on chip [2, 93, 101], typically to the second-level (L2) cache found on chip in modern processors.

However, over the past decade, advancements in semiconductor technology have dramatically changed the landscape of on-chip caches. The increase in the number of transistors available on-chip has enabled on-chip cache sizes to increase exponentially across processor generations. The trend of increasing on-chip cache sizes is apparent in Figure 1 (a), which presents historic data on the on-chip cache sizes of several processors in the last two decades. The upward trend in cache sizes shows no signs of a slowdown. Industry advocates large caches as a microarchitectural technique that allows designers to exploit the available transistors efficiently to improve performance [14]. At the same time, the advent of multicore processors requires large on-chip caches to supply the ever-increasing number of on-chip cores with data. As a result, modern processors increasingly

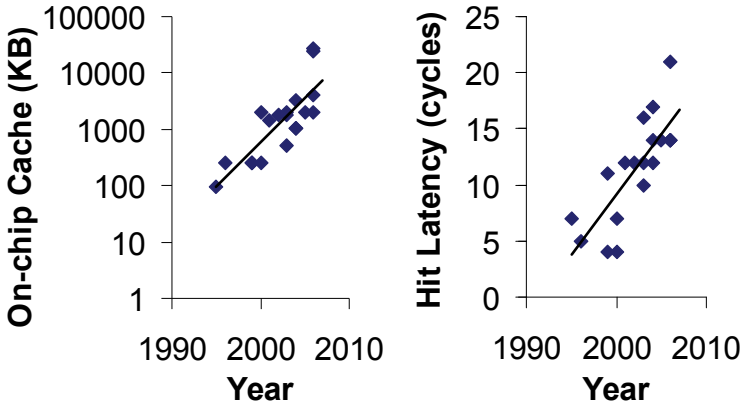


FIGURE 1: Historic trends of on-chip caches on (a) size, and (b) latency.

feature mega-caches on chip (e.g., 16MB in Dual-Core Intel Xeon 7100 [85], and 24MB in Intel Nehalem-EX [53] and Intel Itanium 2 [104]).

Large caches, however, come at the cost of high access latency. Figure 1 (b) presents historic data on the L2 cache access latency, indicating that on-chip L2 latency has increased between 3-fold to 6-fold during the past decade — e.g., from 4 cycles in Intel Pentium III (1995) to 26 cycles in Sun UltraSPARC T2 (2007) [95]. Caches enhance performance most when they capture fully the primary working set of the workload; otherwise, they provide only marginal improvements in the miss rate as size increases. Commercial server workloads typically have a small primary working set which can be captured on chip, and a large secondary set which is beyond the reach of on-chip caches for modern processors. Conventional wisdom dictates that large on-chip caches provide significant performance benefits as they eliminate off-chip memory requests. In reality, a large cache may degrade the performance of server workloads because the cache's high latency slows the common case (cache hits) and introduces stalls in the execution, while the additional capacity fails to lower the miss rate enough to compensate.

As a case study, we investigate the performance of database workloads on modern CMPs and identify data cache stalls as a fundamental performance bottleneck. Recent work in the database community [1, 2, 79] attributes most of the data stalls to off-chip memory accesses. In contrast to prior work, our results indicate that the current trend of increasing L2 latency intensifies stalls due to L2 hits[1], shifting the bottleneck from off-chip accesses to on-chip L2 hits.

Thus, merely bringing data on-chip is no longer enough to attain maximum performance and sustain high throughput. Rather, it is imperative that modern CMP designs optimize for low on-chip data access latencies as well, especially to the (typically larger) last-level cache (LLC).

1.2 Near-Optimal On-Chip Block Placement and Replication

The exponential increase in the number of cores on chip results in a commensurate increase in the on-chip cache size required to supply all these cores with data. At the same time, physical and manufacturing considerations suggest that future processors will be tiled: the last-level on-chip cache will be decomposed into smaller *slices,* and groups of processor cores and cache slices will be physically distributed together throughout the die area [8,107]. Tiled architectures give rise to varying access latencies between the cores and the cache slices spread across the die, naturally leading to a Non-Uniform Cache Access (NUCA) organization of the LLC, where the latency of a cache hit depends on the physical distance between the requesting core and the location of the cached data. Although increasing device switching speeds results in faster cache-bank access times, communication delay remains constant across technologies [20], and access latency of far away cache slices becomes dominated by wire delays and on-chip communication [60].

1. We refer to the time spent by the processor accessing a cache block that missed in L1D but was found in L2 as “L2 hit stalls”.

An ideal LLC allows for fast access to data, large aggregate capacity to store the ever-increasing applications' working sets, and has a simple and practical design. Ideally, the cache places blocks on chip close to the cores that access them. In the case of shared blocks, a mechanism is required to either place the shared blocks equally close to all sharers, or to replicate the blocks among all shares, thereby bringing them close to each one. In this thesis we propose *Reactive NUCA* (R-NUCA), a scalable, low-overhead, and low-complexity cache architecture that optimizes block placement for all cache accesses.

R-NUCA cooperates with the operating system to classify accesses at the page granularity, achieving negligible hardware overhead and avoiding complex heuristics that are prone to error, oscillation, or slow convergence [12,19,23]. The placement decisions in R-NUCA guarantee that each modifiable block is mapped to a single location in the aggregate cache, obviating the need for complex, area- and power-intensive coherence mechanisms prevalent in prior on-chip block placement proposals [12,19,23,107]. R-NUCA utilizes *Rotational Interleaving*, a novel lookup mechanism that matches the fast speed of address-interleaved lookup, without pinning blocks to a single location in the cache [23,107]. Rotational interleaving allows read-only blocks to be shared by neighboring cores and replicated at distant ones, ensuring low access latency while balancing capacity constraints. Overall, we find that R-NUCA allows for near-optimal block placement in a physically-distributed on-chip cache, attaining performance within 5% of an ideal cache design.

While it is imperative that future multicores optimize for on-chip data access latency, there are still many design decisions left to be answered. The capability of future CMPs to host exponentially larger numbers of cores do not directly translate into a commensurate increase in performance. Rather, the abundance of hardware resources comes not only at the cost of increased on-chip data access latencies, but also increased power consumption and off-chip bandwidth require-

ments. However, chips are physically constrained—cooling technology and thermal considerations limit the maximum power at almost constant levels across process technologies, manufacturing considerations and yield economics cap the maximum affordable chip size, while packaging constraints limit the number of pins and the available off-chip bandwidth. Thus, it is important that we optimize future multicore processors with all constraints in mind, in addition to performance. With a promising distributed cache design on hand, we then embark on exploring the design parameters of physically-constrained multicore processors to find the optimal designs for server workloads.

1.3 Optimal CMPs Across Technologies

Design parameters such as the number and type of cores, the supply voltage, the on-chip clock frequency, the size of the on-chip cache, and the chip's operational temperature are intertwined and affect each other non-linearly. Thus, to attain maximum performance and power-efficiency, it is imperative that all design parameters are jointly optimized to maximize their impact on performance while keeping overheads at bay.

Through first-order analytic modeling of server applications' dataset evolution, multicore performance, power, off-chip bandwidth, memory technology, and thermal constraints, we propose multicore designs for commercial server workloads that attain the highest performance at a given power budget. These models conform to ITRS projections across process technologies and jointly optimize supply and threshold voltage, on-chip clock frequency, core count and technology, cache size, and memory technology to result in multicore designs that lie on the pareto frontier of peak-performance designs. We find that ideal future multicores will be based on heterogeneous cores built with low-operational-power devices, employ large multi-megabyte caches, utilize 3D-die

stacking to alleviate the pin bandwidth limitations, and rely on novel on-chip data placement mechanisms that encourage localized data transfers.

1.4 Contributions

In this thesis we investigate multicore designs for commercial server workloads. Through a combination of analytic modeling, execution trace analysis and cycle-accurate full-system simulation using FLEXUS [42,103] of multicore processors running a range of unmodified commercial server applications and multiprogrammed workloads, we demonstrate:

- **On-chip cache latency dominates execution time**. The large on-chip cache sizes of modern CMPs and the high levels of data sharing in commercial workloads increase the cache hit rate, but the higher data access latencies penalize each hit with extra cycles. The combined effects result in server workloads spending up to 35% of their execution time stalled on cache hits, amounting to an increase on the time spent in the on-chip cache by a factor of 5-7 over traditional symmetric multiprocessors.

- **On-line classification of cache accesses**. We show that cache accesses in server workloads can be classified at run time into classes that exhibit distinct characteristics, leading to different on-chip cache block placement policies.

- **Reactive NUCA**. We leverage the characteristics of each access class to design Reactive NUCA (R-NUCA), a low-overhead, low-latency mechanism for block placement in distributed caches. R-NUCA improves performance by allocating cache blocks close to the cores that access them, replicating or migrating them as necessary without the overhead of a hardware coherence mechanism.

- **Rotational Interleaving**. To make block replication practical, we propose Rotational Interleaving, a mechanism for fast lookup in distributed caches with replicated blocks. Rotational Interleaving enables cache block replication in the distributed cache without wasting space and without coherence overheads. It balances capacity constraints with access latency by distributing blocks to nearest neighbors to relieve capacity pressure and ensuring that all allocation decisions among neighbors are mutually consistent, while indexing blocks by always probing only the cache slice that may have the requested block (single-probe lookup). Thus, it provides lookup speeds comparable to conventional address interleaving, without pinning blocks to a single location in the cache.

- **Performance Robustness**. We find that R-NUCA provides performance stability across workloads, as on a per-workload basis it either matches the performance of competing cache designs or improves upon them by 17% on average and by 32% at best, while performing within 5% of an ideal design.

- **CMP Design-Space Exploration Across Process Technologies**. We explore the design space of physically-constrained CMPs across technologies and find that heterogeneous multicores is a viable alternative that holds great promise in optimizing the chip design. Contrary to conventional wisdom, we find that the cores can be implemented using low-operational-power devices without loss in performance, but with significant reduction in power. In contrast to prior research findings, we find that 3D-stacked memory can mitigate the bandwidth wall, making peak-performance designs purely power-constrained. Finally, we find that cache sizes will continue to grow at an exponential rate, so future multicore processors will require novel on-chip data placement mechanisms that encourage localized data transfers.

The rest of the thesis is organized as follows. In Chapter 2, we analyze the performance of chip multiprocessors running database workloads as a case study that quantifies the impact of modern hardware trends on commercial server workloads. In Chapter 3, we introduce Reactive NUCA and Rotational Interleaving and evaluate their performance. In Chapter 4, we introduce first-order analytical models that we use to explore the design space of physically-constrained multicore processors across technologies. We comment on related research in Chapter 5 and conclude in Chapter 6.

The performance analysis in Chapter 2 was previously presented in CIDR 2007 [41]. The Reactive NUCA and Rotational Interleaving designs in Chapter 3 were previously presented in ISCA 2009 [40]. In this document, we provide a more detailed description of Reactive NUCA and Rotational Interleaving that was not possible in the limited space of a conference publication. Material presented in Chapter 4 has not yet appeared in other venues.

Chapter 2

Performance Analysis of CMPs

2.1 Introduction

The commercial server market forms a multi-billion dollar industry [36,37] that penetrates deeply into our everyday life, from ATM and credit card transactions, to travel, to commerce, to government services. High-end servers employ state-of-the-art processors and software to maximize performance. Any failure to realize their maximum potential directly translates into lost investment and impacts the end user who demands low response times and high availability at minimum cost. At the same time, improving the performance of commercial server software will free computational resources for additional services besides the absolutely necessary for a transaction (e.g., it will allow augmenting transactions with on-line fraud detection). Unfortunately, current technological trends make it increasingly difficult to attain high processor utilization and reach maximum performance.

While prior research advocates that high-end server systems and software should focus on bringing data on chip to mitigate the processor-memory performance gap [2], the advent of multi-core processors has resulted in an exponential increase in cache sizes and on-chip interconnects which shift the dominant performance bottlenecks. Large caches and interconnects come at the expense of slow on-chip data access times, which increases the contribution of on-chip cache hits

to execution time. At the same time, the relative contribution of off-chip accesses to execution time dwindles as clock frequency increases have stagnated [7] and DRAM access latency continues to improve. Additionally, multicore processors may increase processor stalls through promoting on-chip data sharing across cores and increasing contention for shared hardware resources.

In this study, we quantify the effects of current hardware trends on the performance of commercial server workloads using database applications as a case study. We demonstrate that the growing L2 hit latency largely determines the performance of commercial software servers. In essence, server workloads are running into an "on-chip cache wall", which has become the new fundamental bottleneck. Our results present a departure from prior research findings that identify off-chip accesses as the dominant barrier to high performance [2, 1, 79]. In contrast to prior work, our results indicate that the current trend of increasing L2 latency intensifies stalls due to L2 hits[1], shifting the bottleneck from off-chip accesses to on-chip L2 hits. Thus, merely bringing data on-chip is no longer enough to attain maximum performance and sustain high throughput.

In this study we recognize that chip multiprocessor designs follow two distinct schools of thought, and present a taxonomy of processor designs and DBMS workloads to distinguish the various combinations of workload and system configuration. We divide chip multiprocessors into two "camps." The *fat camp* employs wide-issue out-of-order processors and addresses data stalls by exploiting instruction-level parallelism (e.g., Intel Core Duo [52], IBM Power 5 [57]). The *lean camp* employs heavily multithreaded in-order processors to hide data stalls across threads by overlapping data access latencies with useful computation (e.g., Sun UltraSPARC T1 [61]). Even though LC is heavily multithreaded, it is a much simpler hardware design than the complex out-of-order FC. We divide database applications into *saturated* workloads, in which idle processors

1. We refer to the time spent by the processor accessing a cache block that missed in L1D but was found in L2 as "L2 hit stalls".

always find an available thread to run, and *unsaturated* workloads, in which processors may not always find threads to run, thereby exposing data access latencies. We characterize the performance of each database workload and system configuration pair within the taxonomy through cycle-accurate full-system simulations using FLEXUS [42,103] of on-line transactional processing (OLTP) and decision support system (DSS) workloads on a commercial DBMS. Our results indicate that:

- High on-chip cache latencies shift the data stall component from off-chip data accesses to L2 hits, to the point where up to 35% of the execution time is spent on L2 hit stalls for our workload and CMP configurations. The increased cache latency results in workloads losing as much as half of their potential performance.

- High levels of on-chip core integration increase L2 hit rates, improving performance by 12-15% and increasing the relative contribution of L2 hit stalls to 10% and 25% of execution time, respectively, for DSS and OLTP.

- The combined effects of high L2 latency and higher L2 hit rates due to on-chip core integration increase the contribution of L2 hit stalls on execution time by a factor of 5 for DSS and a factor of 7 for OLTP over traditional symmetric multiprocessors, explaining the observed departure from prior research findings.

- Conventional DBMS hide stalls only in one out of four combinations of chip designs and workloads. Despite the significant performance enhancements that stem from chip-level parallelism, the fat camp still spends 46-64% of execution time on data stalls. The lean camp efficiently overlaps data stalls when executing saturated workloads, but exhibit up to 70% longer response times than the fat camp for unsaturated workloads.

The remainder of this chapter is structured as follows. Section 2.2 proposes a taxonomy of chip multiprocessor technologies and workloads. Section 2.3 presents our experimental methodology and Section 2.4 analyzes the behavior of a commercial database server on chip multiprocessors, as a function of hardware designs and workloads. Section 2.5 discusses the effects of hardware parameters on data stalls. Finally, Section 2.6 presents a summary of our analysis.

2.2 CMP Camps and Workloads

In this section we propose a taxonomy of chip multiprocessor technologies and database workloads, and analyze their characteristics. To our knowledge, this is the first study to provide an analytic taxonomy of the behavior of database workloads in such a diverse spectrum of current and future chip designs. A recent study [30] focuses on throughput as the primary performance metric to compare server workload performance across chip multiprocessors with varying processor granularity, but has stopped short of a detailed performance characterization and breakdown of where time is spent during execution. The taxonomy we propose enables us to concentrate on each segment separately and derive the reasons behind our performance observations. Through a series of simulations we find that the behavior of database systems varies as a function of hardware and workload type, and that conventional database systems fail to provide high performance across the entire spectrum.

Table 1: Chip multiprocessor camp characteristics.

Core Technology	Fat Camp (FC)	Lean Camp (LC)
Issue Width	Wide (4+)	Narrow (1 or 2)
Execution Order	Out-of-order	In-order
Pipeline Depth	Deep (14+ stages)	Shallow (5-6 stages)
Hardware Threads	Few (1-2)	Many (4+)
Core Size	Large (3 x LCsize)	Small (LC size)

2.2.1 Fat Camp vs. Lean Camp

Hardware vendors adopt two distinct approaches to chip multiprocessor design. One approach uses cores that target maximum single-thread performance through sophisticated out-of-order execution and aggressive speculation (fat-camp or FC). Representative chip multiprocessors from this camp include Intel Core Duo [52] and IBM Power5 [57]. The second approach favors much simpler designs with cores that support many thread contexts[1] in hardware (lean-camp or LC). Such cores overlap stalls in a given thread with useful computation by other threads. Sun UltraSPARC T1 [61] and Compaq Piranha [10] fall into this camp. Table 1 summarizes the characteristics of each technology camp.

Integrating multiple cores on a chip multiprocessor exhibits similar effects within each camp (e.g., increase in shared resource contention). In this chapter we study the increasing performance differences between fat and lean camps when running identical database workloads, assuming that both camps are supported by the same memory hierarchy. Thus, it suffices to analyze the characteristics of each camp by focusing on the characteristics of the different core technologies within each camp.

1. We refer to hardware threads as “hardware contexts” to distinguish them from software (operating system) threads.

Because LC cores are heavily multithreaded, we expect them to hide stalls efficiently and provide high and scalable throughput when there is enough parallelism in the workload. However, when the workload consists of a few threads, the LC cores cannot find enough threads to overlap stalls, leaving long data access latencies exposed. On the other hand, the FC cores are optimized for single-thread performance through wide pipelines that issue/complete multiple instructions per cycle, and out-of-order speculative execution. These features exploit instruction-level parallelism within the workload to hide stalls.

Thus, we expect LC cores to outperform FC cores when there is enough parallelism in the workload, even with much lower single-thread performance than that of an FC core. However, when the workload consists of few threads, we expect the response time of the single-thread optimized FC cores to be significantly lower than the corresponding response time of their LC counterparts.

In addition to the performance differences when comparing single cores, an LC CMP can typically fit three times more cores in one chip than an FC CMP, resulting in roughly an order of magnitude more hardware contexts in the same space. In this study we do not apply constraints on the chip area. Keeping a constant chip area would favor the LC camp because it would have a larger on-chip cache than the FC camp, allowing LC to attain even higher performance in heavily multithreaded workloads, because LC is able to hide L2 stalls through multithreading.

2.2.2 Unsaturated vs. Saturated Workloads

Database performance varies with the number of requests serviced. Our unsaturated workload highlights single-thread performance by assigning one worker thread per query (or transaction) it receives. A conventional DBMS can increase the parallelism through partitioning, but in

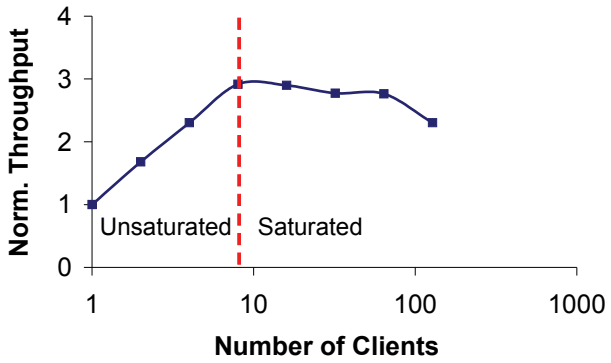


FIGURE 2: Unsaturated vs. saturated workloads.

the context of this study we can treat this as having multiple clients (instead of threads). The reader should also keep in mind that not all query plans are trivially parallelizable.

We observe that the performance of a database application falls within one of two regions, for a given hardware platform, and characterize the workload as unsaturated or saturated. A workload is unsaturated when processors do not always find threads to run. As the number of concurrent requests increases, performance improves by utilizing otherwise idle hardware contexts. Figure 2 illustrates throughput as a function of the number of concurrent requests in the system when running TPC-H queries on a commercial DBMS on a real 4-core IBM Power5 (FC) server. Increasing the number of concurrent requests eventually results in a saturated workload, where there are always available threads for idle processors to run. Peak performance occurs at the beginning of the saturated region; increasing the number of concurrent requests too far overwhelms the hardware, reducing the amount of useful work performed by the system and lowering performance.

2.3 Experimental Methodology

We use FLEXUS [42,103] to provide accurate simulations of chip multiprocessors and symmetric multiprocessors running unmodified commercial database workloads. FLEXUS is a cycle-accurate full-system simulator that simulates both user-level and operating system code. We use the SimFlex statistical sampling methodology [103]. Our samples are drawn over an interval of 10 to 30 seconds of simulated time (as observed by the operating system in functional simulation) for OLTP, and over the complete workload execution for DSS. We show 95% confidence intervals on performance measurements using paired measurement sampling. We launch measurements from checkpoints with warmed caches and branch predictors, then run for 100,000 cycles to warm queue and interconnect state prior to collecting measurements of 50,000 cycles. We use the aggregate number of user instructions committed per cycle (i.e., committed user instructions summed over the simulated processors divided by total elapsed cycles) as our performance metric, which is proportional to overall system throughput [103].

We characterize the performance of database workloads on an LC CMP and an FC CMP with the UltraSPARC III instruction set architecture running the Solaris 8 operating system. The LC CMP employs four 2-issue superscalar in-order cores. The LC cores are 4-way multithreaded, for a total of 16 hardware contexts on the LC CMP. The hardware contexts are interleaved in round-robin fashion, issuing instructions from each runnable thread in turn. When a hardware context stalls on a miss it becomes non-runnable until the miss is serviced. In the meantime, the LC core executes instructions from the remaining contexts.

The FC CMP employs four aggressive out-of-order cores that can issue four instructions per cycle from a single hardware context. The two CMP designs have identical memory subsystems

and clock frequencies and feature a shared on-chip L2 cache with size that ranges from 1MB to 26MB.

We estimate cache access latencies using Cacti 5.3 [97]. Cacti is an integrated cache access time, cycle time, area, leakage, and dynamic power model. By integrating all these models together, cache trade-offs are all based on the same assumptions and, hence, are mutually consistent. In some experiments we purposefully vary the latency of caches beyond the latency indicated by Cacti to explore the resulting impact on performance or to obtain conservative estimates.

Our workloads consist of OLTP (TPC-C) and DSS (TPC-H) benchmarks running on a commercial DBMS. The saturated OLTP workload consists of 64 clients submitting transactions on a 100-warehouse database. The saturated DSS workload consists of 16 concurrent clients running four queries from the TPC-H benchmark, each with random predicates. We select the queries as follows [88]: Queries 1, 6 are scan-dominated, Query 16 is join-dominated and Query 13 exhibits mixed behavior. To achieve practical simulation times we run the queries on a 1GB database. We corroborate recent research that shows that varying the database size does not incur any microarchitectural behavior changes [88]. Unsaturated workloads use the above methodology running only a single client, with intra-query parallelism disabled to highlight single-thread performance. We tune both the OLTP and DSS workloads to minimize I/O overhead and maximize CPU and memory system utilization.

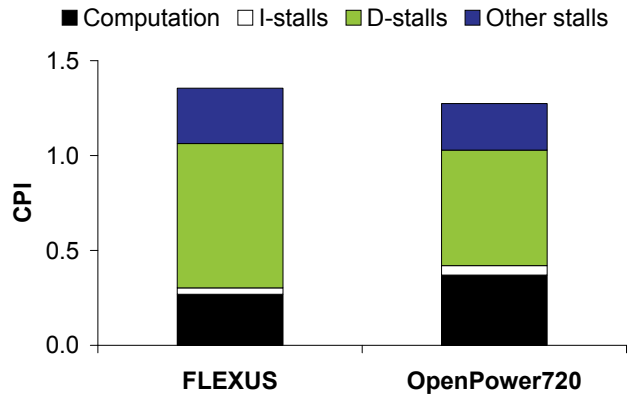


FIGURE 3: validation using the saturated DSS workload.

We validate by comparing against an IBM OpenPower720 server that runs the same workloads. We calculate the cycles per instruction (CPI) on OpenPower720 by extracting Power5's hardware counters through pmcount [27], post-processing the raw counters using scripts kindly provided by IBM, and comparing the results with a simulation that approximates the same IBM server. Figure 3 presents the absolute CPI values and their respective breakdowns. The overall simulated CPI is within 5% of the measured CPI for both OLTP and DSS workloads. The computation component for OpenPower720 is 10% higher, which we attribute to Power5's instruction grouping and cracking overhead. The data stall component is 15% higher due to the absence of a hardware prefetcher mechanism.

While employing a stride prefetcher will not change the performance trends that are the focus of our study, it is instructive to discuss its performance implications on our workload mix. Prior research [93] measures the impact of hardware prefetching on the performance of OLTP and DSS workloads and finds that even complex hardware prefetchers that subsume stride prefetchers yield less than 10% performance improvement for OLTP workloads and scan-dominated DSS queries. Join-dominated DSS queries do see as much as 50% improvement, but contribute relatively

little to total execution time in our DSS query mix. Even if a stride prefetcher could match the performance improvements of [93], we estimate that the performance improvement due to a stride prefetcher on our OLTP workload will be less than 10%, while the performance improvement on our scan-dominated DSS workload will be less than 20%. However, the performance trends due to the increasing L2 latencies will remain the same.

2.4 DBMS Performance on CMPs

In this section we characterize the performance of both CMP camps on a commercial DBMS running unsaturated and saturated DSS and OLTP workloads. For unsaturated workloads the performance metric of interest is response time, while for saturated workloads the performance metric of interest is throughput. Figure 4 (a) presents the response time of the LC CMP normalized to the FC CMP when running unsaturated (single-thread) workloads. Figure 4 (b) presents the throughput of the LC CMP normalized to the throughput of the FC CMP when running saturated workloads.

The LC CMP suffers up to 70% higher response times than FC when running unsaturated (single-thread) DSS workloads and up to 12% higher when running unsaturated OLTP workloads, corroborating prior results [79]. The performance difference between FC and LC on unsaturated OLTP workloads is narrower due to limited ILP. Even though FC exhibits higher single-thread per-

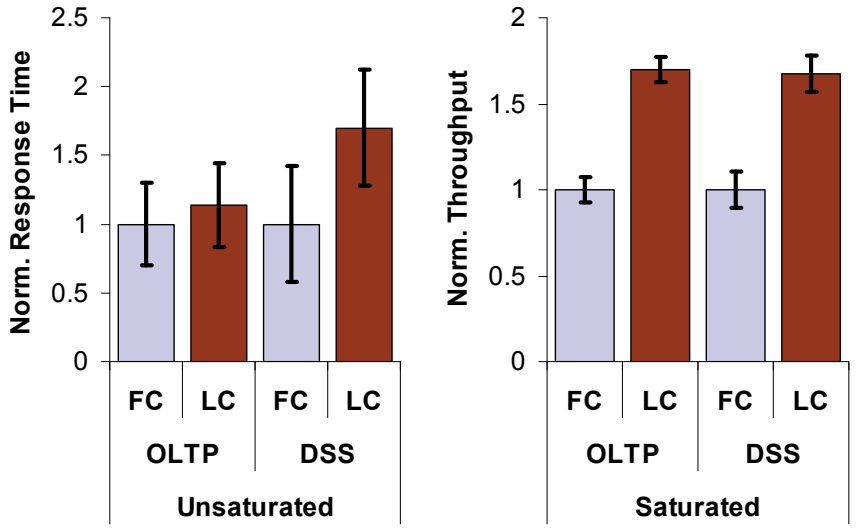


FIGURE 4: (a) Response time and (b) throughput of LC normalized to FC.

formance than LC, the LC CMP achieves 70% higher throughput than its FC counterpart when running saturated workloads (Figure 4 b).

Figure 5 shows the execution time breakdown for each camp and workload combination. Although we configure the CMPs with an unrealistically fast 26MB shared L2 cache, data stalls dominate execution time in three out of four cases. While FC spends 46% - 64% of execution time on data stalls, LC spends at most 13% of execution time on data stalls when running saturated workloads, while spending 76-80% of the time on useful computation. The multiple hardware contexts in LC efficiently overlap data stalls with useful computation, thereby allowing LC to outperform significantly its FC counterpart on saturated workloads.

Despite prior work [2] showing that instruction stalls often dominate memory stalls when running database workloads, our CMP experiments indicate that data stalls dominate the memory access component of the execution time for all workload/camp combinations. Both camps employ instruction stream buffers [56], a simple hardware mechanism that automatically initiates

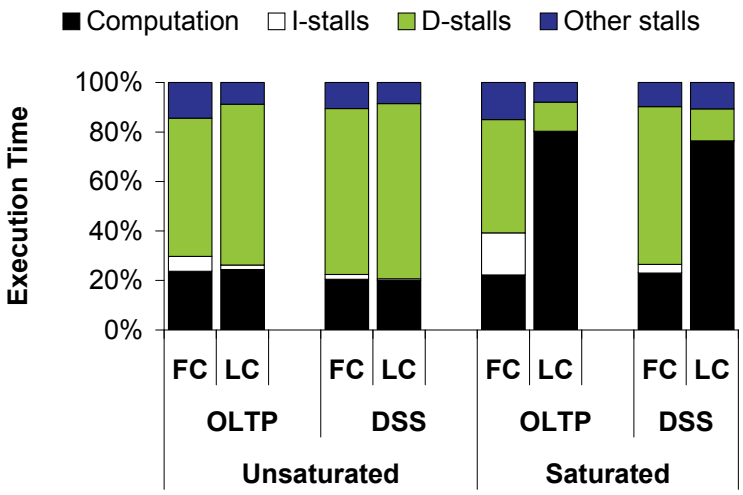


FIGURE 5: Breakdown of execution time.

prefetches to successive instruction cache lines following a miss. Our results corroborate prior research [79] that demonstrates instruction stream buffers efficiently reduce instruction stalls. Because of their simplicity, instruction stream buffers can be employed easily by the majority of chip multiprocessors, thus we do not further analyze instruction cache performance.

We conclude that the abundance of threads in saturated workloads allows LC CMPs to hide data stalls efficiently. The multiple hardware contexts available on the LC CMP allow it to perform useful computation while some of the contexts are stalled on long latency data access operations, thereby improving overall throughput. In contrast, the FC CMP fails to utilize fully its hardware resources because database workloads exhibit limited ILP. FC processors would also benefit from multithreaded operation, but their complexity limits the number of hardware contexts they can employ. Our calculations show that each FC core would require more than 15 hardware contexts to fully overlap data stalls, which is infeasible due to the complexity and power implications it entails. Thus, FC CMPs cannot hide data stalls the way context-rich LC CMPs can.

However, we expect that in spite of less than ideal performance on database workloads, FC CMPs will still claim a significant market share due to their unparalleled single-thread performance and optimized execution on a variety of other workloads (e.g., desktop, scientific computing). Thus, database systems must be designed to perform well on both CMP camps, independent of workload type. To maximize performance across hardware and workload combinations, database systems must exhibit high thread-level parallelism across and within queries and transactions, and improve data locality/reuse. Increased parallelism helps exploit the abundance of on-chip thread and processor execution resources when the workload is not saturated. Data locality helps eliminate stalls independent of workload type.

Figure 5 shows that in six out of eight combinations of hardware and workloads, data stalls dominate execution time even with unrealistically fast and large caches. In Section 2.5 we analyze the data stall component of execution time to identify dominant subcomponents and trends, that will help guide the implementation and optimization of future database software. In the interest of brevity, we analyze data stalls by focusing on saturated database workloads running on FC CMPs, but the results of our analysis are applicable across all combinations of hardware and workloads that exhibit high data stall time.

2.5 Analysis of Data Stalls

In this section we analyze the individual sub-components of data cache stalls and identify the emerging importance of L2 hit stalls, which account for up to 35% of execution time for our hardware configurations and workloads. This represents a a 7-fold increase as compared to traditional symmetric multiprocessors with small caches running the same workloads.

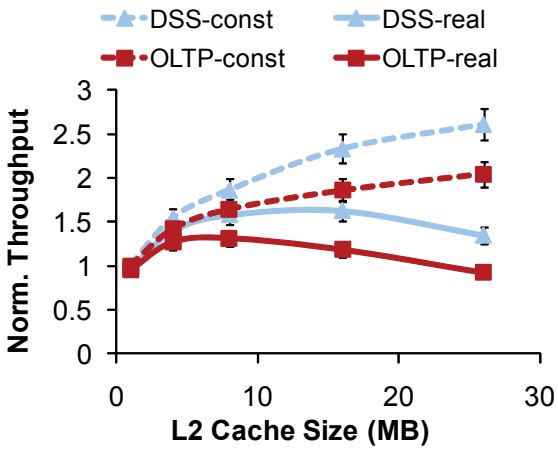


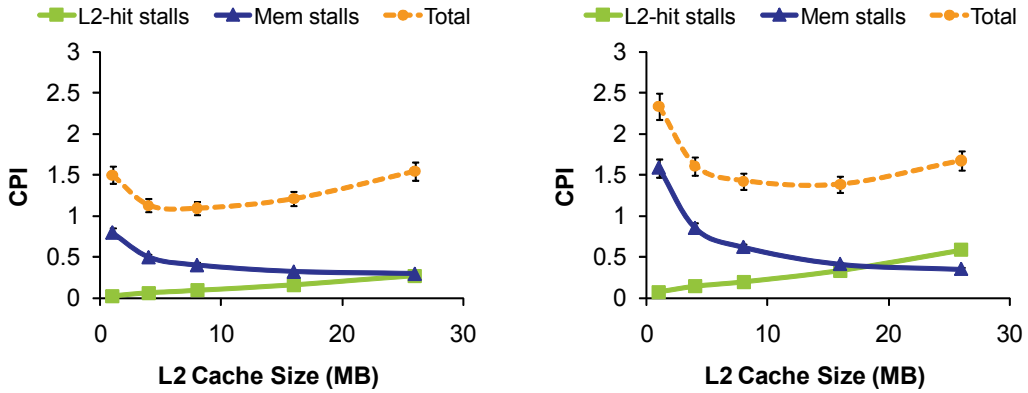
FIGURE 6: Effect of cache size and latency on throughput.

Section 2.5.1 explores the impact of increased on-chip cache sizes on the breakdown of data stalls. In Section 2.5.2 we analyze the impact of moving from traditional (multi-chip) multiprocessors to single-chip multiprocessors. Finally, in Section 2.5.3 we study the effects of high levels of on-chip core integration.

2.5.1 Impact of On-Chip Cache Size

Large on-chip L2 caches shift the data stall bottleneck in two ways. First, large caches exhibit high hit rates. As more requests are serviced by the cache, data stalls shift from memory to L2 hits. Second, rising hit latencies penalize each hit and increase the number of stalls caused by L2 hits without changing the number of accesses to other parts of the memory hierarchy.

Figure 6 presents the impact of increasing cache size on DBMS performance. We simulate both OLTP and DSS workloads on a FC CMP, with cache sizes ranging from 1MB to 26MB. To separate the effect of hit rates from that of hit latencies, we perform two sets of simulations. The upper (dotted) pair of lines shows the performance increase achieved when the hit latency remains



(a) CPI contributions for OLTP (b) CPI contributions for DSS

FIGURE 7: Effect of cache size and latency on CPI for (a) OLTP and (b) DSS.

fixed at an unrealistically low 4 cycles. The lower (solid) pair of lines shows the performance under the more reasonable hit latencies estimated using CACTI for each cache configuration. These estimates are conservative because hit latencies estimated by CACTI are typically lower than the ones achieved in commercial products.

In all cases, increasing the cache size significantly improves performance as more of the primary L2 working set fits in the cache. However, the realistic-latency and constant-latency performance curves quickly begin to diverge, even before the cache captures a significant fraction of the entire working set. Even though there is no cycle penalty for increasing L2 sizes in the constant-latency case, we see diminishing returns because even the biggest cache fails to capture the large secondary L2 working set. In contrast, realistic hit latencies further reduce the benefit of larger caches, and the added delay begins to outweigh the benefits of lower miss rates. The adverse effects of high L2 hit latency reduce the potential performance benefit of large L2 caches by up to 2.2x for OLTP and 2x for DSS.

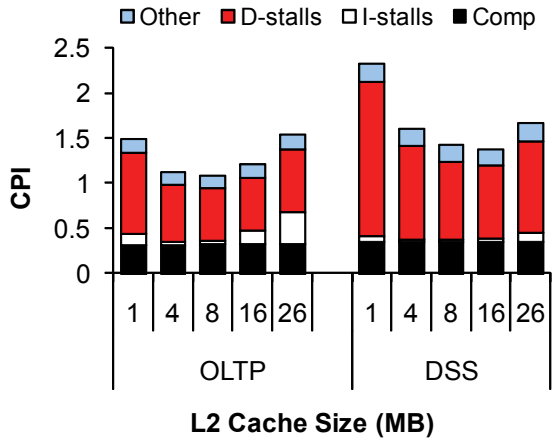


FIGURE 8: CPI breakdown with increasing L2 size.

Figure 7 (a) and (b) show the effect of realistic L2 hit latencies and sizes to L2 hit stalls, memory stalls and overall CPI for OLTP and DSS respectively. In the constant-latency case (not shown) the stall component due to L2 hits quickly stabilizes at less than 5% of the total CPI. On the other hand, realistic L2 latencies are responsible for a growing fraction of L2 hit stalls to the overall CPI, especially in DSS, where they become the single largest component of execution time. Conversely, the larger L2 sizes cause a decrease in off-chip requests, with a subsequent reduction of memory stalls. These figures demonstrate that the growing L2 sizes result in L2 hits contributing significantly to execution time, while the contribution of off-chip memory requests drops.

The remainder of the CPI increase comes from instruction stalls due to L2 hits, again an artifact of larger (and slower) caches. Instruction stalls due to L2 are especially evident in the OLTP workload, where they account for roughly half of the overall CPI increase. This is evident in Figure 8 which shows the breakdown of CPI for OLTP and DSS with increasing cache size.

Increasing cache sizes and their commensurate increase in latency can have dramatic effects on the fraction of time spent on L2 hit data stalls. For our workloads running on a FC CMP we

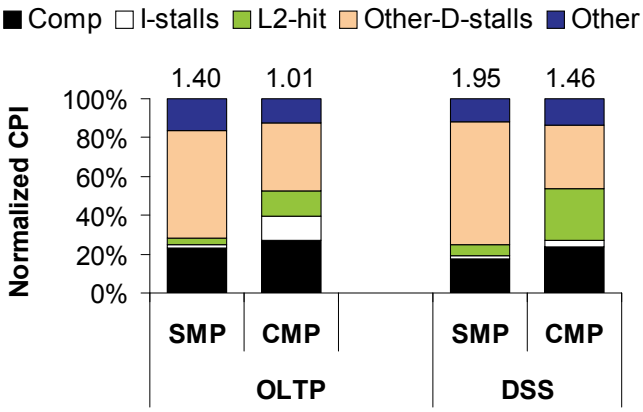


FIGURE 9: Impact of chip multiprocessing on CPI.

measure a 12-fold increase in time spent in L2 hit stalls when increasing the cache size from 1MB to 26MB; rising hit latencies are responsible for up to 78% of this increase.

2.5.2 Impact of Core Integration on Single Chip

In this section we study the outcome of integrating multiple processing cores on a single chip. We compare the performance of a commercial database server running OLTP and DSS workloads in two variants of our baseline system: (a) a 4-processor SMP with private 4MB L2 caches at each node, and (b) a 4-core CMP with a single shared 16MB L2.

Figure 9 presents normalized CPI breakdowns for the two systems, with labels indicating the actual CPI. We observe that the performance of the CMP systems is higher. The difference in the performance between the SMP and the CMP systems can be attributed to the elimination of coherence traffic. Data accesses that result in long-latency coherence misses in the SMP system are converted into L2 hits on the shared L2 cache of the CMP and fast L1-to-L1 on-chip data transfers. Thus, the L2 hit stall component of CPI increases by a factor of 7 over the corresponding SMP designs, explaining the disparity of our results as compared to prior research findings [2,88].

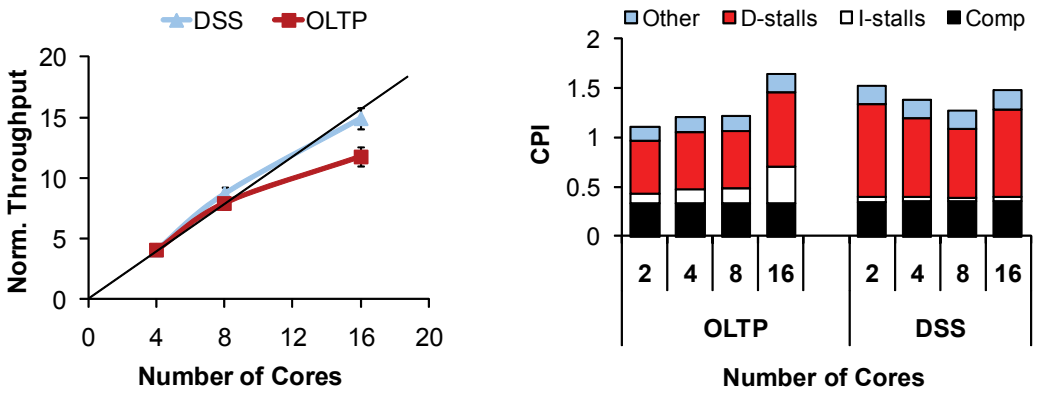


FIGURE 10: Impact of core count on (a) throughput, and (b) CPI breakdown.

2.5.3 Impact of On-Chip Core Count

Chip multiprocessors integrate multiple cores on a single chip, which promotes sharing of data through the common L2 cache. At the same time, contention for shared hardware resources my offset some of the benefits of fast on-chip data sharing. To study the impact of high levels of core integration on chip we simulate saturated OLTP and DSS workloads on a FC chip multiprocessor with a 16MB shared L2 as we increase the number of cores from 4 (the baseline) to 16.

Figure 10(a) presents the change in performance as the number of processing cores increases. The diagonal line shows linear speedup as a reference. We observe a 9% super-linear increase in throughput at 8 cores for DSS, due to an increase in sharing, after which pressure on the L2 cache adversely affects performance for both workloads. OLTP, in particular, realizes about 74% of its potential linear performance improvement. The pressure on the cache is not due to extra misses — in fact, the L2 miss rate continues to drop due to increased sharing as more cores are added. Rather, physical resources such as cache ports and status registers induce queueing delays

during bursts of misses. These correlated misses are especially common in OLTP and are largely responsible for the sub-linear speedup when adding more cores.

Figure 10(b) shows the CPI breakdown of a CMP running our OLTP and DSS workloads as the core count increases. The breakdown shows that the contribution of instruction stalls for OLTP increases dramatically with increasing core count. This increase is due to contention for the instruction cache blocks. As we will see in the next chapter, the instruction stream for commercial server workloads is universally shared among all cores, so the introduction of more cores does not result in more misses; on the contrary, the hit rate for instructions increases as all cores request the same blocks. However, the increased contention for instruction blocks creates hot spots in the cache, causing the cores to queue their requests behind others already in the cache controller and stall until the requests can be serviced. This observation shows that while it is important for future multicore architectures to alleviate the growing on-chip cache access latency, it is also important to invent techniques that will reduce contention for hot cache blocks.

2.5.4 Ramifications

Our analysis of data stalls and the impact of current hardware trends on CMP performance have several ramifications. The bottleneck shift to L2 hit stalls arises primarily from the combination of higher L2 hit rates, increased L2 hit latencies and increased cache block contention due to on-chip core integration. The emerging L2 hit stalls indicate that simply bringing data on chip no longer suffices to attain maximum performance. In the future it will become increasingly important to utilize techniques that lower the L2 access latency. This will be our focus in Chapter 3.

The growing L2 hit latencies can be alleviated by bringing data beyond L2 and closer to L1, for example through streaming and prefetching. Several streaming and prefetching techniques

have been proposed by the architectural community [55,78,93,101,21,48,83,90,99,92,59, 102,70,72,84,71], where complicated hardware, the compiler or a hardware/software mechanism strive to recognize the workloads' access patterns and stream data ahead of the request. However, these techniques typically focus on predicting off-chip requests, where the large size and relative inactivity of L2 make them tolerable to mispredictions, and the processor-memory gap provides a large opportunity for improvement. In on-chip streaming, however, the small L1s and stream buffers are prone to pollution, and the L1-to-L2 distance is relatively small, so none but the most accurate techniques may effectively stream blocks within a chip. Unfortunately, commercial server workloads exhibit adverse and arbitrarily complex data sharing and access patterns [21,94,101] that hamper most efforts to build prefetching engines that attain both high accuracy and coverage.

Second, increasing the number of cores that share an on-chip L2 cache does not cause an inordinate number of additional cache misses for commercial server workloads. In fact, these workloads exhibit significant sharing, so the cores benefit from each other's requests (constructive interference). However, the extra cores do cause contention for hot blocks that can degrade performance in spite of the lower miss rate. We expect that future CMP designs will feature specially-designed L2 caches to reduce this pressure, allowing workloads to benefit from the effects of sharing. Our proposed cache design in Chapter 3 also helps alleviate most of the cache block contention by replicating the hottest blocks across the die area.

Third, incorporating large (slow) caches on chip can have detrimental effects to the performance of commercial server workloads. Optimized future CMP designs should incorporate caches large enough to capture the primary L2 working set, but not larger, so they can maintain reasonably low hit latencies. This observation runs counter to the conventional wisdom that larger caches

are always a good way to use extra transistors [14]. We will investigate the trade-off between hit rate and cache access latency through first-order analytic modeling of CMPs in Chapter 4.

Finally, the software system can be restructured to intelligently balance parallelism with locality and work synergistically with hardware. Conventional software is optimized for outdated hardware architectures, assuming private resources and coarse-grain OS-managed threads with no coordination of resource usage among them. However, modern CMPs employ shared hardware resources and facilitate fine-grain communication between software threads. To utilize efficiently the available hardware resources, modern software systems can be restructured to split otherwise single-threaded operations into many smaller parallel tasks, and simultaneously optimize for data affinity to processors to minimize cross-chip data transfers. Such software architectures can expose the data access and sharing patterns to the execution system, allowing simple hardware techniques to eliminate the remaining on-chip data access stall times. We will discuss our current efforts in this area when we present future work in Chapter 6.

2.6 Summary

High levels of integration have enabled the advent of chip multiprocessors and increasingly large (and slow) on-chip caches. These trends of increasing core counts and larger on-chip caches pose new performance challenges to the architecture and software communities. In this study we present a performance characterization of a commercial database servers in a number of representative chip multiprocessor technologies. Our results indicate that on-chip cache accesses are the new performance bottleneck, with L2 hit stalls rising from oblivion to become the dominant single execution time component. Future processors must employ novel cache architectures that keep access latency and block contention at bay without sacrificing capacity. At the same time, software

systems and not ready for such dramatic shifts in hardware design and must be restructured to utilize efficiently the new hardware.

In the next chapter, we introduce Reactive NUCA, our proposal for a novel distributed cache architecture that allows for fast access and low contention for hot cache blocks by placing blocks on chip close to the cores that access them. Later, in Chapter 6, we present our on-going work on novel software architectures that enhance parallelism and locality, and show promise in utilizing efficiently the hardware resources of modern multicore processors.

Chapter 3

Reactive NUCA

The exponential increase in the number of cores on chip results in a commensurate increase in the on-chip cache size required to supply all these cores with data and the on-chip interconnect that facilitates cross-chip data transfers. At the same time, physical and manufacturing considerations suggest that future processors will be tiled: the last-level on-chip cache will be decomposed into smaller *slices,* and groups of processor cores and cache slices will be physically distributed throughout the die area [8,107].

Tiled architectures result in varying access latencies between the cores and the cache slices spread across the die, as the latency of a cache hit depends on the physical distance between the requesting core and the location of the cached data. Although increasing device switching speeds results in faster cache-bank access times, communication delay remains constant across technologies [20], and access latency of far away cache slices becomes dominated by wire delays and on-chip communication [60]. Thus, tiled architectures naturally lead to a Non-Uniform Cache Access (NUCA) [60] organization of the LLC.

From an access-latency perspective, an LLC organization where each core treats a nearby LLC slice as a private cache is desirable. Although a private organization results in fast local hits, it requires area-intensive, slow and complex mechanisms to guarantee the coherence of shared data, which are prevalent in many multicore workloads [11,41]. At the same time, the growing

application working sets render private caching designs impractical due to the inefficient use of cache capacity, as cache blocks are independently replicated in each private cache slice.

At the other extreme, a shared organization where blocks are statically address-interleaved in the aggregate cache offers maximum capacity by ensuring that no two cache frames are used to store the same block. Because static interleaving defines a single, fixed location for each block, a shared LLC does not require a coherence mechanism, enabling a simple design and allowing for larger aggregate cache capacity. However, static interleaving results in a random distribution of cache blocks across the LLC slices, leading to frequent accesses to distant cache slices and high average access latency.

An ideal LLC enables the fast access of the private organization and the design simplicity and large capacity of the shared organization. Recent research advocates hybrid and adaptive mechanisms to bridge the gap between the private and shared organizations. However, prior proposals require complex, area-intensive, and high-latency lookup and coherence mechanisms [12,19,23,107], waste cache capacity [12,107], do not scale to high core counts [19,39], or optimize only for a subset of the cache accesses [12,19,24]. In this thesis we propose *Reactive NUCA* (R-NUCA), a scalable, low-overhead, and low-complexity cache architecture that optimizes block placement for all cache accesses, at the same time attaining the fast access of the private organization and the large aggregate capacity of the shared organization.

R-NUCA cooperates with the operating system to classify accesses at the page granularity, achieving negligible hardware overhead and avoiding complex heuristics that are prone to error, oscillation, or slow convergence [12,19,23]. The placement decisions in R-NUCA guarantee that each modifiable block is mapped to a single location in the aggregate cache, obviating the need for complex, area- and power-intensive coherence mechanisms of prior proposals [12,19,23,107].

R-NUCA utilizes *Rotational Interleaving*, a novel lookup mechanism that matches the fast speed of address-interleaved lookup, without pinning blocks to a single location in the cache [23,107]. Rotational interleaving allows read-only blocks to be shared by neighboring cores and replicated at distant ones, ensuring low access latency while balancing capacity constraints.

The work presented in this chapter makes the following contributions:

- Through execution trace analysis, we show that cache accesses for instructions, private data, and shared data exhibit distinct characteristics, leading to different replication, migration, and placement policies.
- We leverage the characteristics of each access class to design R-NUCA, a novel, low-overhead, low-latency mechanism for block placement in distributed caches.
- We propose rotational interleaving, a novel mechanism for fast nearest-neighbor lookup with one cache probe, enabling replication without wasted space and without coherence overheads.
- Through full-system cycle-accurate simulation of multicore systems, we show that R-NUCA provides performance stability across workloads. On a per-workload basis, R-NUCA either matches the performance of competing cache designs or improves upon them by 17% on average. More specifically, R-NUCA attains a maximum speedup of 32%, and an average speedup of 14% across all workloads over the private design (17% for server workloads) and 6% over the shared design (17% for multi-programmed workloads and OLTP), while achieving performance within 5% of an ideal cache design.

The rest of this chapter is organized as follows. Section 3.1 presents background on distributed caches and tiled architectures. Section 3.2 presents our classification and offers a detailed

empirical analysis of the cache-access patterns of server, scientific, and multi-programmed workloads. We describe R-NUCA in Section 3.3 and evaluate it in Section 3.4. We summarize our findings in Section 3.5.

3.1 Background

3.1.1 Non-Uniform Cache Architectures

The exponential increase in the cache sizes of multicore processors (CMPs) renders uniform access latency impractical, as capacity increases also increase access latency [41]. To mitigate the rising access latency, recent research [60] advocates decomposing the cache into slices. Each slice may consist of multiple banks to optimize for low access latency within the slice [13], and all slices are physically distributed on the die. Thus, cores realize fast accesses to nearby slices and slower accesses to physically distant ones.

Just as cache slices are distributed across the entire die, processor cores are similarly distributed. Economic, manufacturing, and physical design considerations [8,107] suggest *tiled architectures*, with cores coupled together with cache slices in tiles that communicate via an on-chip interconnect. Tiled architectures are attractive from a design and manufacturing perspective, enabling developers to concentrate on the design of a single tile and then replicate it across the die [8]. They are also economically attractive, as they can easily support families of products with varying number of tiles and power/cooling requirements. Finally, their scalability to high core counts make them suitable for large-scale CMPs.

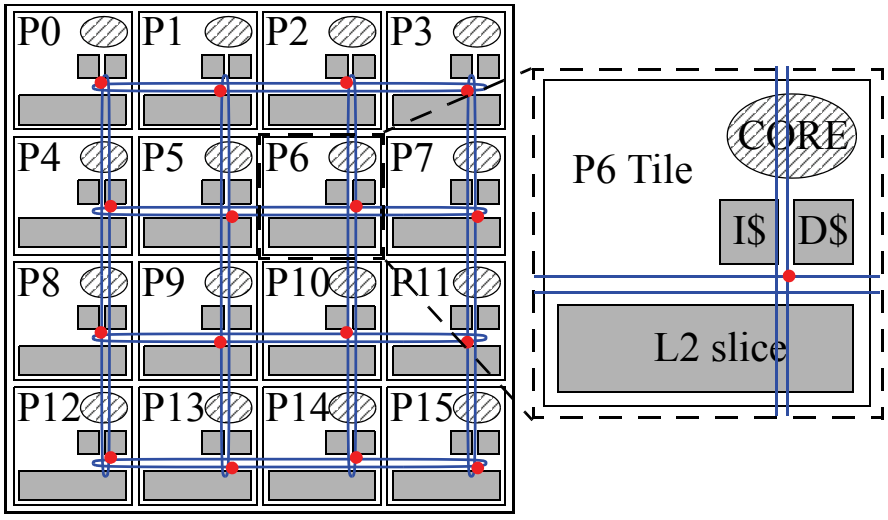


FIGURE 11: Typical tiled architecture and floorplan of folded 2D-torus.

3.1.2 Tiled Architectures

Figure 11 presents a typical tiled architecture. Multiple tiles, each comprising a processor core, caches, and network router/switch, are replicated to fill the die area. Each tile includes private L1 data and instruction caches and an L2 cache slice. Each L1 cache miss probes an on-chip L2 cache slice via an on-chip network that interconnects the tiles. Depending on the L2 organization, the L2 slice can be either a private L2 cache or a portion of a larger distributed shared L2 cache. Also depending on the cache architecture, the tile may include structures to support cache coherence such as L1 duplicate tags [10] or sections of the L2-cache distributed directory. The tiles are connected through an on-chip folded 2D-torus interconnect.

Private L2 organization. Each tile's L2 slice serves as a private second-level cache for the tile's core. Upon an L1 miss, the L2 slice located in the same tile is probed. On a write miss in the local L2 slice, the coherence mechanism (a network broadcast or access to an address-interleaved

distributed directory) invalidates all other on-chip copies. On a read miss, the coherence mechanism either confirms that a copy of the block is not present on chip, or it obtains the data from an existing copy. With a directory-based coherence mechanism, a typical coherence request is performed in three network traversals. A similar request in token-coherence [73] requires a broadcast followed by a response from the farthest tile.

Enforcing coherence requires large storage and complexity overheads. For example, a full-map directory for a 16-tile CMP with 64-byte blocks, 1MB L2 slices, and 64KB split I/D L1 caches requires 288K directory entries, assuming two separate hardware structures to keep the L1 caches and the L2 slices coherent (on each request, both structures are searched in parallel). With a 42-bit physical address space, and a 16-bit sharers bit-mask and 5-bit state per block to account for intermediate states, the directory size per tile is 1.2MB, exceeding the L2 capacity. Thus, full-map directories are impractical for the private L2 organization. Limited-directory mechanisms are smaller, but may require complex, slow, or non-scalable fall-back mechanisms such as full-chip broadcast. In this work, we optimistically assume a private L2 organization where each tile has a full-map directory with zero area overhead.

Shared L2 organization. Cache blocks are address-interleaved among the slices, which service requests from any tile through the interconnect. On an L1 cache miss, the miss address dictates the slice responsible for caching the block, and a request is sent directly to that slice. The target slice stores both the block and its coherence state. Because each block has a fixed, unique location in the aggregate L2 cache, the coherence state must cover only the L1 cache tags; following the example above, a full-map directory for a shared L2 organization requires only 152KB per tile.

3.1.3 Requirements for Intelligent Block Placement

A distributed cache presents a range of latencies to each core, from fast access to nearby slices, to several times slower access to slices on the opposite side of the die. Intelligent cache block placement can improve performance by placing blocks close to the requesting cores, allowing fast access.

We identify three key requirements for intelligent block placement in distributed caches. First, the block address must be decoupled from its physical location, allowing to store the block at a location independent of its address [22]. Decoupling the physical location from the block address complicates lookup on each access. Thus, the second requirement for intelligent block placement is a lookup mechanism capable of quickly and efficiently locating the cached block. Finally, intelligent block placement must optimize for all accesses prevalent in the workload. A placement policy may benefit some access classes while penalizing others [108]. To achieve high performance, an intelligent placement algorithm must react appropriately to each access class.

3.2 Characterization of L2 References

3.2.1 Methodology

We analyze the cache access patterns using trace-based and cycle-accurate full-system simulation in FLEXUS [42, 103] of a tiled CMP executing unmodified applications and operating systems. FLEXUS extends the Virtutech Simics functional simulator with timing models of processing tiles with out-of-order cores, NUCA cache, on-chip protocol controllers, and on-chip interconnect. We simulate a tiled CMP similar to Section 3.1.2, where the LLC is the L2 cache.

Table 2: System parameters for the 8-core and 16-core CMPs.

CMP Size	16-core for server and scientific workloads 8-core for multi-programmed workloads
Processing Cores	UltraSPARC III ISA; 2GHz, OoO cores 8-stage pipeline, 4-wide dispatch/retirement 96-entry ROB and LSQ, 32-entry store buffer
L1 Caches	split I/D, 64KB 2-way, 2-cycle load-to-use, 3 ports 64-byte blocks, 32 MSHRs, 16-entry victim cache
L2 NUCA Cache	16-core CMP: 1MB per core, 16-way, 14-cycle hit 8-core CMP: 3MB per core, 12-way, 25-cycle hit 64-byte blocks, 32 MSHRs, 16-entry victim cache
Main Memory	3 GB memory, 8KB pages, 45 ns access latency
Memory Controllers	one per 4 cores, round-robin page interleaving
Interconnect	2D folded torus (4x4 for 16-core, 4x2 for 8-core) 32-byte links, 1-cycle link latency, 2-cycle router

We run server and scientific workloads on CMPs optimized for high core counts [30], so we run these workloads on a 16-core CMP with 1MB L2 cache per core. Similar to [19], we believe that multi-programmed desktop workloads are likely to run on CMPs with fewer cores that leave enough die area for a larger on-chip cache, required to store the larger private working sets of each software process. Thus, we run our multi-programmed mix on an 8-core tiled CMP with 3MB L2 cache per core. To estimate the L2 cache size for each configuration, we assume a die size of 180mm^2 in 45nm technology and calculate component sizes following ITRS guidelines [87]. We account for the area of the system-on-chip components, allocating 65% of the die area to the tiles [30]. We estimate the area of the cores by scaling a micrograph of the Sun UltraSPARC II processor. We summarize our tiled architecture parameters in Table 2.

We simulate systems running the *Solaris 8* operating system and executing the workloads listed in Table 3. We include a wide range of server workloads: online transaction processing (TPC-C on IBM DB2 v8 ESE and Oracle 10g), decision support systems (TPC-H queries 6,8, and 13 on IBM DB2 v8 ESE), and a web server workload (SPECweb running on Apache HTTP Server

Table 3: Workload parameters for R-NUCA evaluation.

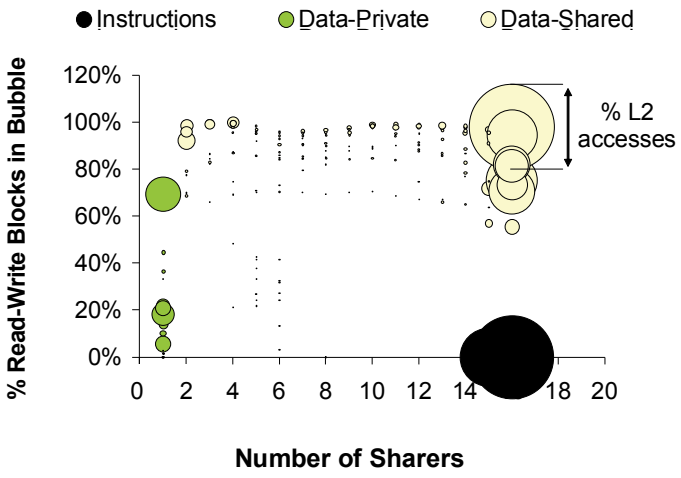
OLTP – Online Transaction Processing (TPC-C v3.0)	
DB2	*IBM DB2 v8 ESE,* 100 warehouses (10 GB), 64 clients, 2 GB buffer pool
Oracle	*Oracle 10g Enterprise Database Server* 100 warehouses (10 GB), 16 clients, 1.4 GB SGA
Web Server (SPECweb99)	
Apache	*Apache HTTP Server v2.0.* 16K connections, fastCGI, worker threading model
DSS – Decision Support Systems (TPC-H)	
Qry 6, 8, 13	*IBM DB2 v8 ESE*, 480 MB buffer pool, 1GB database
Scientific	
em3d	768K nodes, degree 2, span 5, 15% remote
Multi-programmed (SPEC CPU2000)	
MIX	2 copies from each of gcc, twolf, mcf, art; reference inputs

v2.0). For the TPC-C and TPC-H workloads we follow the scaling rules of the TPC specifications. We include a multiprogrammed workload (MIX), consisting of SPEC CPU2000 applications with one application per core. Finally, even though we tailor the design of R-NUCA to server workloads, we include a scientific application (em3d, which models electromagnetic wave propagation through 3D-space) as a frame of reference.

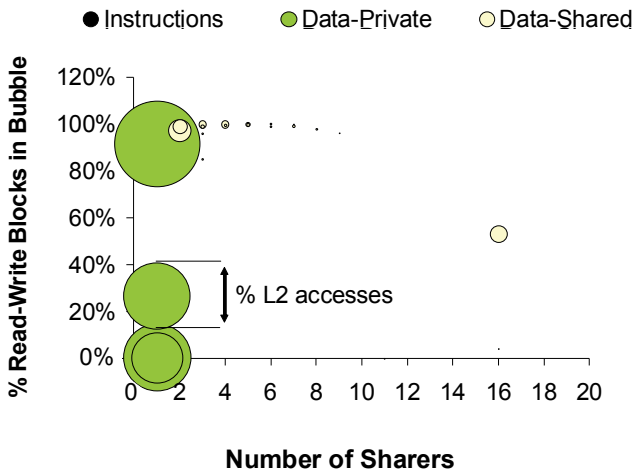
With one exception, we focus our study on the workloads described in Table 3. To show the wide applicability of our L2 reference clustering observations, Figure 12 includes statistics gathered using a larger number of server workloads (TPC-C on DB2 and Oracle, TPC-H queries 6, 8, 11, 13, 16, and 20 on DB2, SPECweb on Apache and Zeus), scientific workloads (em3d, moldyn, ocean, sparse), and the multi-programmed workload from Table 3.

3.2.2 Categorization of Cache Accesses

We analyze the L2 accesses at the granularity of cache blocks along two axes: the number of cores sharing an L2 block and the percentage of blocks with at least one write request during the



(a) Server Workloads



(b) Scientific and Multi-Programmed Workloads

FIGURE 12: L2 Access Categorization and Clustering.

workload's execution (*read-write blocks*). Each bubble in Figure 12 represents blocks with the same number of sharers (1-16). For each workload, we plot two bubbles for each number of sharers, one for instruction and one for data accesses. The bubble diameter is proportional to the number of L2 accesses. We indicate instruction accesses in black and data accesses in yellow (shared) or green (private), drawing a distinction for private blocks (accessed by only one core).

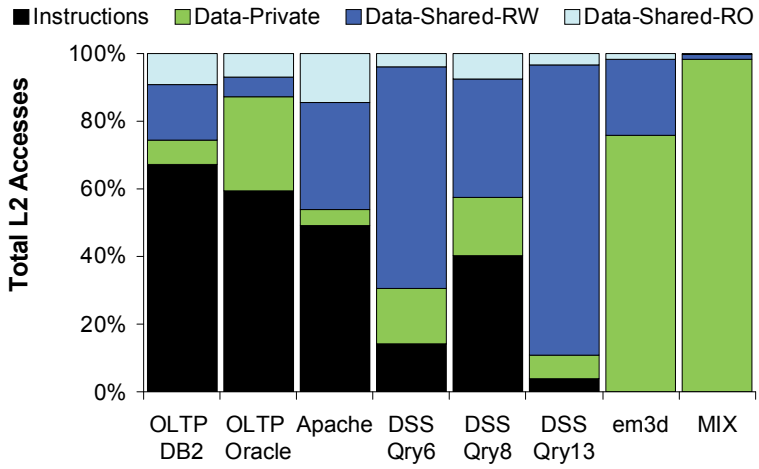


FIGURE 13: L2 Reference breakdown.

We observe that, in server workloads, L2 accesses naturally form three clusters with distinct characteristics: (1) instructions are shared by all cores and are read-only, (2) shared data are shared by all cores and are mostly read-write, and (3) private data exhibit a varying degree of read-write blocks. We further observe that scientific and multi-programmed workloads access mostly private data, with a small fraction of shared accesses in data-parallel scientific codes exhibiting producer-consumer (two sharers) or nearest-neighbor (two to six sharers) communication. The instruction footprints of scientific and multi-programmed workloads are effectively captured by L1-I.

The axes of Figure 12 suggest an appropriate L2 placement policy for each access class. Private data blocks are prime candidates for allocation near the requesting tile; placement at the requesting tile achieves the lowest possible access latency. Because private blocks are always read or written by the same core, coherence is guaranteed without requiring a hardware mechanism. Read-only universally-shared blocks (e.g., instructions) are prime candidates for replication across multiple tiles; replicas allow the blocks to be placed in the physical proximity of the requesting cores, while the blocks' read-only nature obviates coherence. Finally, read-write blocks with many

sharers (shared data) may benefit from migration or replication if the blocks exhibit reuse at the L2. However, migration requires complex lookup and replication requires coherence enforcement, and the low reuse of shared blocks does not justify such complex mechanisms (Section 3.2.3.3). Instead, shared blocks benefit from intelligent block placement on chip.

Although server workloads are dominated by accesses to instructions and shared read-write data, a significant fraction of L2 references are to private blocks (Figure 13). The scientific and multi-programmed workloads are dominated by accesses to private data, but also exhibit some shared data accesses. The varying importance of the cache accesses categories underscores a need to react to the access class when placing blocks at L2, and emphasizes the opportunity loss of addressing only a subset of the access classes.

3.2.3 Characterization of Access Classes

3.2.3.1 Private Data

Accesses to private data, such as stack space and thread-local storage, are always initiated by the same processor core. As a result, replicating private data at multiple locations on chip only wastes cache capacity [106]. Although some private data are read-write, having only one requestor eliminates the need for a cache coherence mechanism for private blocks. Therefore, the only requirement for private data is to be placed close to its requestor, ensuring low access latency[1]. R-NUCA places private data in the slice close to the requesting core, ensuring minimum latency.

To evaluate whether private data fit in a single slice, Figure 14 shows the CDF of L2 accesses to private data, instructions, and shared data as a function of the footprint of each access

1. The operating system may migrate a thread from one core to another. In these cases, coherence can be enforced by the OS by shooting down the private blocks upon a thread migration.

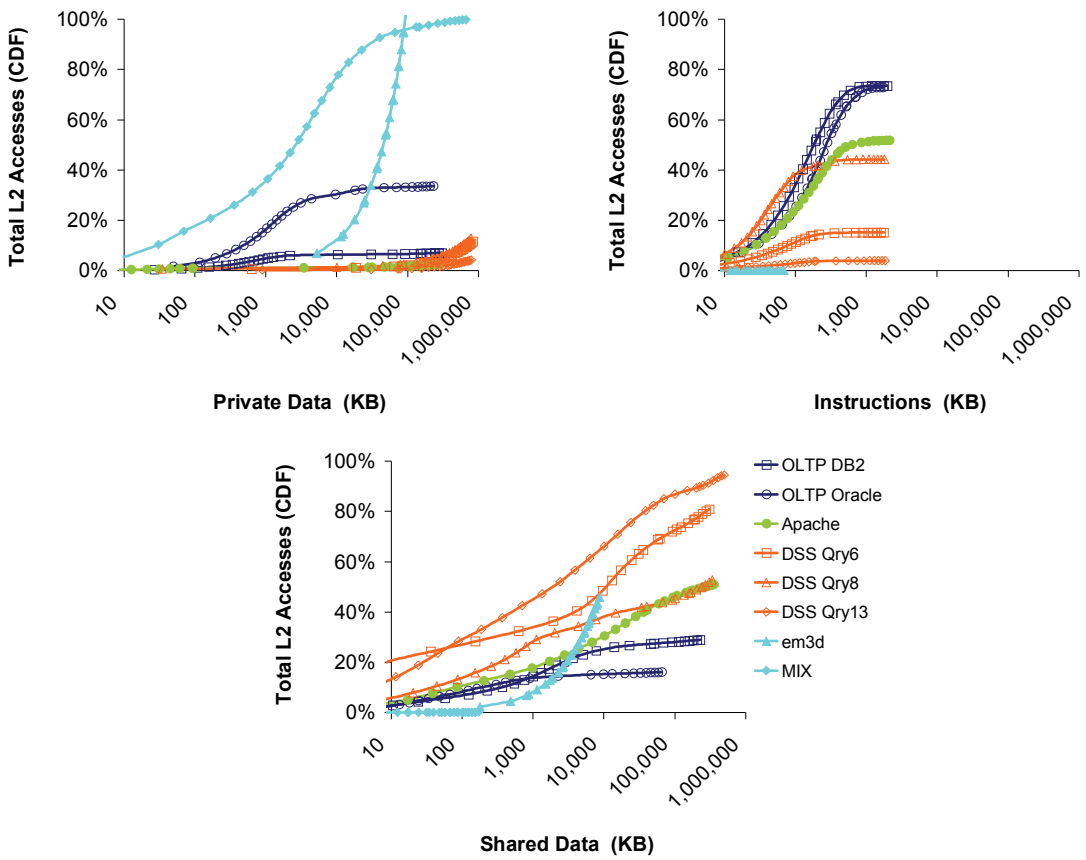


FIGURE 14: L2 working set sizes for private data, instruction and shared data.

class (in log-scale). Accesses are normalized to the total L2 cache accesses for each workload. As shown in the figure, the private-data working set for OLTP fits into a single, local L2 slice for each core. The total size of the private data is about 3MB, but the private data are spread among 16 cores, thereby requiring only about 192KB per slice. Similarly, the primary working set for private data for the web workload is about 6KB per slice. The secondary working set is too large to fit in any reasonable L2 slice; however, it accounts for only 1.5% of L2 accesses, thus its impact on performance is expected to be small. The private-data working set for the multi-programmed MIX barely fits into a single cache slice, while the working set for the DSS and scientific workloads is

too big to fit. DSS workloads scan multi-gigabyte database tables and scientific workloads operate on large data sets, both exceeding any reasonable L2 capacity.

To accommodate large private data working sets, prior proposals advocate migrating (spilling) these blocks to neighbors [24]. Spilling may be applicable to multi-programmed workloads composed of applications with a range of private-data working set sizes; however, it is inapplicable to server or scientific workloads. All cores in a typical server or balanced scientific workload run similar threads, with each L2 slice experiencing similar capacity pressure. Migrating private data blocks to a neighboring slice is offset by the neighboring tiles undergoing an identical operation and spilling in the opposite direction. Thus, cache pressure remains the same, but requests incur higher access latency.

3.2.3.2 Instructions

Instruction blocks are typically written once when the operating system loads an application binary or shared library into memory. Once in memory, instruction blocks remain read-only for the duration of execution. Figure 12 indicates that instruction blocks in server workloads are universally shared among the processor cores. All cores in server workloads typically exercise the same instruction working set, with all cores requiring low-latency access to the instruction blocks with equal probability. Instruction blocks are therefore amenable to replication. By caching multiple copies of the blocks on chip, replication enables low-latency access to the instruction blocks from multiple locations on chip.

In addition to replication, in Figure 15 (left) we examine the utility of instruction-block migration toward a requesting core. We present the percentage of L2 accesses which constitute the 1st, 2nd, and subsequent instruction-block accesses by one core without intervening L2 accesses for the same block by a different core. The grey and higher portions of the bars represent reuse

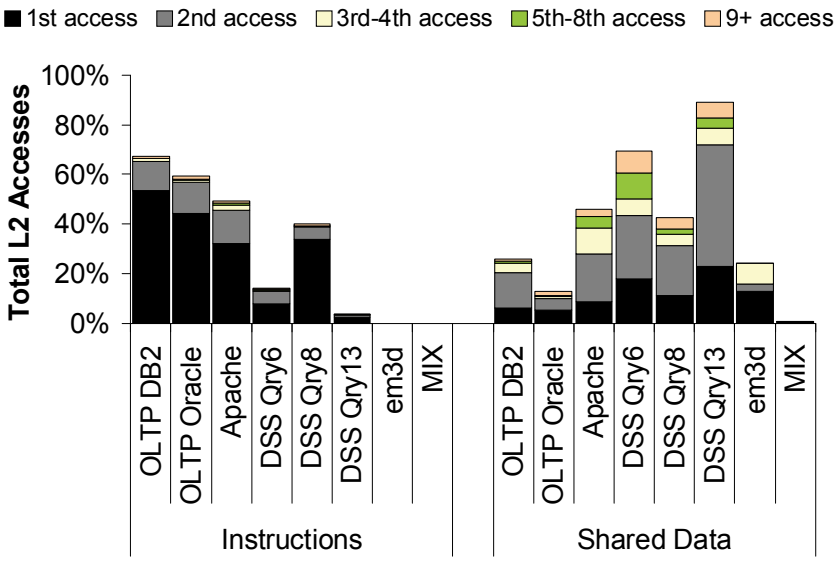


FIGURE 15: Instruction and shared data reuse by same core.

accesses that could experience a lower latency if the instruction block was migrated to the requesting core after the first access. The results indicate that accesses to L2 instruction blocks are finely interleaved between participating sharers, with minimal opportunity of instruction block migration. On the contrary, migration may reduce performance, as it increases contention in the on-chip network.

Figure 14 (right) shows that the instruction working set size for some workloads approximates the size of a single L2 slice. Indiscriminate replication of the instruction blocks at each slice creates too many replicas and increases the capacity pressure and the off-chip miss rate. At the same time, replicating a block in adjacent L2 slices offers virtually no latency benefit, as multiple replicas are one network hop away from a core, while having just one copy nearby is enough. Thus, replication should be done at a coarser granularity: R-NUCA logically divides the L2 into clusters of neighboring slices, replicating instructions at the granularity of a cluster rather than in individual L2 slices. While an application's working set may not fit comfortably in an individual

L2 slice, it fits into the aggregate capacity of a cluster. Each slice participating in a cluster of size *n* should store *1/n* of the instruction working set. By controlling the cluster size, it is possible to smoothly trade off instruction-block access latency for cache capacity: many small clusters provide low access latency while consuming a large fraction of the capacity of each participating slice; a few large clusters result in higher access latency but with a small number of replicas.

For our system configurations and workloads, clusters of 4 slices are appropriate. Clusters of size 4 ensure that instruction blocks are at most one network hop away from the requesting core while storing only a quarter of the instruction working set at each slice.

3.2.3.3 Shared Data

Shared data comprise predominantly read-write blocks containing data and synchronization structures. Replication or migration of shared blocks can provide low-latency access for the subsequent references to the same block from the local or nearby cores. However, on each write, complex coherence mechanisms are necessary to invalidate the replicas or to update the migrating blocks. Figure 15 (right) shows the number of L2 accesses to a shared data block issued by the same core between consecutive writes by other cores. In most cases, a core accesses a block only once or twice before a write by another core. Thus, an invalidation will occur nearly after each replication or migration opportunity, eliminating the possibility of accessing the block at its new location in most cases, and rendering both techniques ineffective for shared data.

Not only do the access characteristics shown in Figure 15 (right) indicate a small opportunity for the replication or migration of shared data, the complexity and overheads of these mechanisms entirely overshadow their benefit. The replication or migration of shared data blocks at arbitrary locations on chip require the use of directory- or broadcast-based mechanisms for lookup and coherence enforcement, as each block is likely to have different placement requirements.

However, to date, there have been only few promising directions to provide fast lookup [80], while the area and latency overheads of directory-based schemes (Section 3.1.2) discourage their use, and broadcast-based mechanisms do not scale due to the bandwidth and power overheads of probing multiple cache slices per access. Also, replicating or migrating shared data would increase the cache pressure and the off-chip requests due to the shared data's large working sets (Figure 14, middle).

Placing shared read-write data in a NUCA cache presents a challenging problem because their coherence requirements, diverse access patterns, and large working sets render migration and replication policies undesirable for these data. The challenge has been recognized by prior studies in NUCA architectures. However, the problem remained largely unaddressed, with the best proposals completely ignoring shared read-write blocks [12] or ignoring them once their adverse behavior is detected [23].

Instead of relying to migration or replication, R-NUCA places the shared read-write data close to the requestors by distributing them evenly among all participating sharers. Shared data blocks in server workloads are universally accessed (Figure 12), with every core having the same likelihood to be the next accessor [94]. Therefore, R-NUCA distributes shared data across all tiles using standard address interleaving. By placing the blocks at the address-interleaved locations, R-NUCA avoids replication. Thus, it eliminates wasted space and obviates the need for a coherence mechanism by ensuring that, for each shared block, there is a unique slice to which that block is mapped by all sharers. At the same time, R-NUCA utilizes a trivial and fast lookup mechanism, as a block's address uniquely determines its location. Because the access latency depends on the network topology, accesses to statically-placed shared data benefit most from a topology that avoids hot spots and affords best-case (average) access latency for all cores (e.g., torus).

Other on-chip interconnect topologies are also possible. Recently, Balfour and Dally [9] concluded that while folded 2D-torus networks compare favorably against 2D-meshes for on-chip interconnects, other topologies (e.g., concentrated 2D-meshes) exhibit better area-delay and energy-delay characteristics. However, an exhaustive evaluation of on-chip interconnect topologies is beyond the scope of this thesis. Without loss of generality, we simulate CMPs employing a folded 2D-torus interconnect, but our techniques are not restricted to a particular topology.

3.2.4 Characterization Conclusions

The diverse cache access patterns of server workloads make each access class amenable to a different placement policy. R-NUCA is motivated by this observation, and its design is guided by the characteristics of each class. More specifically, we find that:

- An intelligent placement policy is sufficient to achieve low cache access latency.
- L2 hardware coherence mechanisms in a tiled CMP running server workloads are unnecessary and should be avoided.
- Private blocks should be placed in the local slice of the requesting core.
- Instruction blocks should be replicated in clusters (groups) of nearby slices.
- Shared data blocks should be placed at fixed address-interleaved cache locations.

3.3 R-NUCA Design

We base our design on a CMP with private split L1 I/D caches and a distributed shared L2 cache. The L2 cache is partitioned into slices, which are interconnected by an on-chip folded 2D-

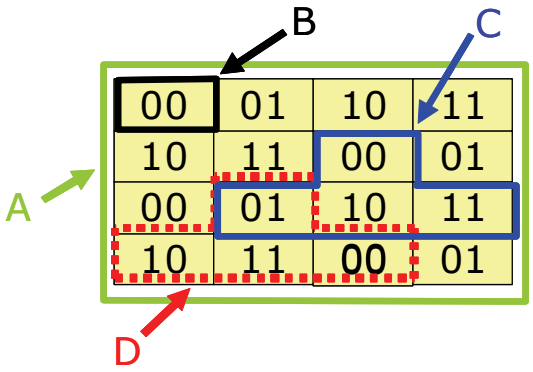


FIGURE 16: Example of R-NUCA clusters and Rotational Interleaving.

torus network. We assume that cores and L2 slices are distributed on the chip in tiles, forming a tiled architecture similar to the one described in Section 3.1.2. This assumption is not a limitation, as the mechanisms we describe apply to alternative organizations, for example, groups of cores assigned to a single L2 slice.

Conceptually, R-NUCA operates on overlapping *clusters* of one or more tiles. R-NUCA introduces *fixed-center* clusters, which consist of the tiles logically surrounding a core. Each core defines its own fixed-center cluster. For example, Figure 16 shows a tiled multicore processor where each rectangle represents a tile with the lines surrounding some of the tiles representing cluster boundaries. Clusters C and D in Figure 16 each consist of a center tile and the neighboring tiles around it. Clusters can be of various power-of-2 sizes. Clusters C and D in Figure 16 are size-4. Size-1 clusters always consist of a single tile (e.g., cluster B). In our example, size-16 clusters comprise all tiles (e.g., cluster A). As shown in Figure 16, clusters may overlap. Data within each cluster are interleaved among the participating L2 slices, and shared among all cores participating in that cluster.

3.3.1 Indexing and Rotational Interleaving

R-NUCA indexes blocks within each cluster using either standard address interleaving or rotational interleaving. In standard address interleaving, an L2 slice is selected based on the bits immediately above the set-index bits of the accessed address. In rotational interleaving, each core is assigned a *rotational ID* (RID) by the operating system. The RID is different from the conventional *core ID* (CID) that the OS assigns to each core for process bookkeeping.

RIDs in a size-*n* cluster range from *0* to *n-1*. To assign RIDs, the OS first assigns the RID *0* to a random tile. Consecutive tiles in a row receive consecutive RIDs. Similarly, consecutive tiles in a column are assigned RIDs that differ by $log_2(n)$ (along rows and columns, *n-1* wraps around to *0*). An example of RID assignment for size-4 fixed-center clusters is shown in Figure 16, where the binary numbers in the rectangles denote each tile's RID.

To index a block in its size-4 fixed-center cluster, the center core uses the 2 address bits $<a_1, a_0>$ immediately above the set-index bits. The core compares the bits with its own RID $<c_1, c_0>$ using a boolean function; the outcome of the comparison determines which slice caches the accessed block. The general form of the boolean indexing function for size-*n* clusters with the rotational-interleaving bits starting at offset *k* is:

$$R = (Addr[k + \log_2(n) - 1 : k] + \overline{RID} + 1) \wedge (n - 1)$$

For size-4 clusters, the 2-bit result *R* indicates that the block is in, to the right, above, or to the left of the requesting tile, for binary results *<0,0>, <0,1>, <1,0>* and *<1,1>* respectively.

In the example of Figure 16, when the center core of cluster C accesses a block with address bits *<0, 1>*, the core evaluates the indexing function and access the block in the slice to its left. Similarly, when the center core of cluster D accesses the same block, the indexing function indi-

cates that the block is at the slice above. Thus, each slice stores exactly the same *1/n*-th of the data on behalf of any cluster to which it belongs.

Rotational interleaving allows clusters to replicate data without increasing the capacity pressure in the cache, and at the same time enable fast nearest-neighbor communication. The implementation of rotational interleaving is trivial, requiring only that tiles have RIDs and that indexing is performed through simple boolean logic on the tile's RID and the block's address. Although for illustration purposes we limit our description here to size-4 clusters, rotational interleaving is simply generalized to clusters of any power-of-two. An in-depth discussion of rotational interleaving can be found at Chapter 3.3.5

3.3.2 Placement

Depending on the access latency requirements, the working set, the user-specified configuration, or other factors available to the OS, the system can smoothly trade off latency, capacity, and replication degree by varying the cluster sizes. Based on the cache block's classification presented in Section 3.2.2, R-NUCA selects the cluster and places the block according to the appropriate interleaving mechanism for this cluster.

In our configuration, R-NUCA utilizes only clusters of size-1, size-4 and size-16. R-NUCA places core-private data in the size-1 cluster encompassing the core, ensuring minimal access latency. Shared data blocks are placed in size-16 clusters which are fully overlapped by all sharers. Instructions are allocated in the most size-appropriate fixed-center cluster (size-4 for our workloads), and are replicated across clusters on chip. Thus, instructions are shared by neighboring cores and replicated at distant ones, ensuring low access latency for surrounding cores while balancing capacity constraints. Although R-NUCA forces an instruction cluster to experience an off-

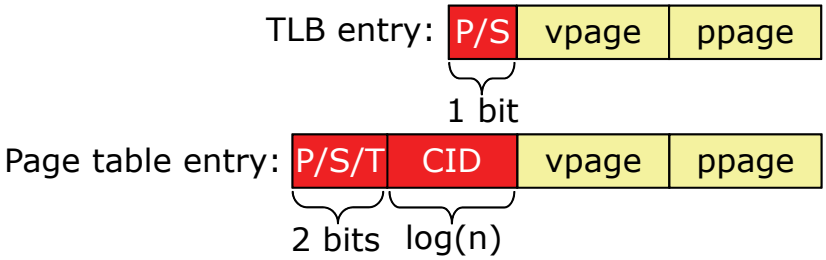


FIGURE 17: Page table entry and TLB extensions.

chip miss rather than retrieving blocks from other on-chip replicas, the performance impact of these "compulsory" misses is negligible.

3.3.3 Page Classification

R-NUCA classifies memory accesses at the time of a TLB miss. Classification is performed at the OS-page granularity, and communicated to the processor cores using the standard TLB mechanism. Requests from L1 instruction caches are immediately classified as *instructions* and a lookup is performed on the size-4 fixed-center cluster centered at the requesting core. All other requests are classified as data requests, and the OS is responsible for distinguishing between private and shared data accesses.

To communicate the private or shared classification for data pages, the operating system extends the page table entries with a two-bit state (one bit denotes the current classification while the other bit is used for a temporary poison state, *"T"*), and a log(n)-bits field to record the CID of the last core to access the page (Figure 17). Similarly, the TLB entry is extended with one bit that denotes whether the page is classified as *private* or *shared-data*.

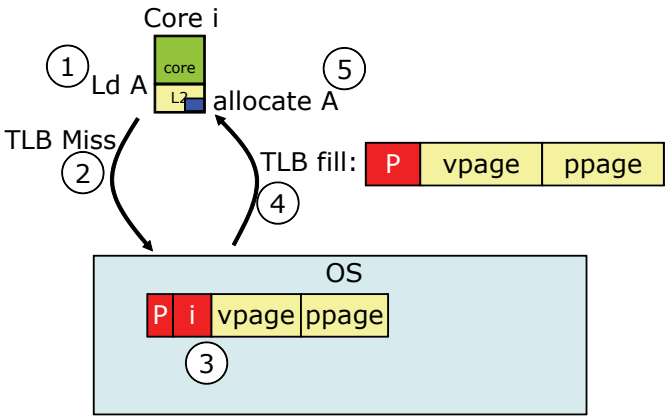


FIGURE 18: Time-line of private page classification.

Figure 18 shows the time-line of classifying a page as *private*. Upon the first access to a page (1), a core encounters a TLB miss and traps to the OS (2). The OS locates the page table entry for the faulting page, marks is as *private* and the CID of the accessor is recorded (3). The accessor receives a TLB fill with an additional *Private* bit set (4) and the corresponding cache block is allocated at the local L2 slice (5). On any subsequent request, during the virtual-to-physical translation, the requesting core examines the Private bit and looks for the block only in its own local L2 slice.

On a subsequent TLB miss to the page, the OS compares the CID in the page table entry with the CID of the core encountering the TLB miss. In the case of a mismatch, either the thread accessing this page has migrated to another core and the page is still private to the thread, or the page is shared by multiple threads and must be re-classified as *shared*. Because the OS is fully aware of thread scheduling, it can precisely determine whether or not thread migration took place, and correctly classify a page as private or shared.

If a page is actively shared, the OS must re-classify the page from private to shared. Figure 19 shows the time-line of a page classification as *shared-data.* The first access to a page,

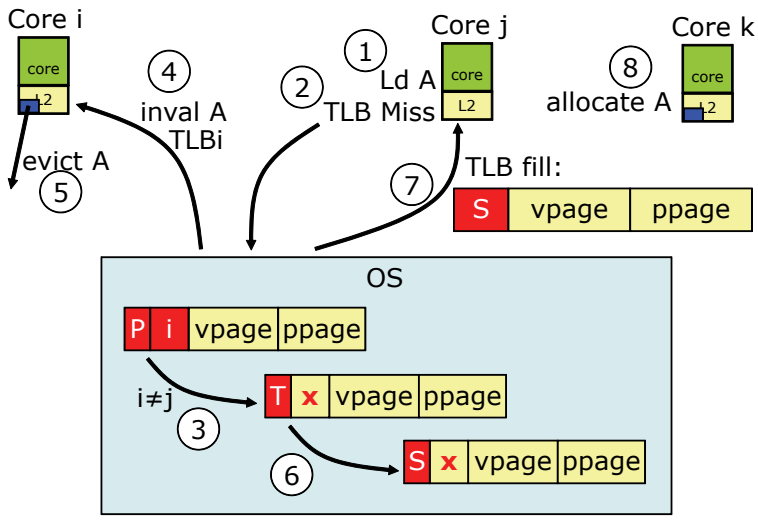


FIGURE 19: Time-line of shared-data page classification.

e.g., by core *i*, causes the page to be classified as *private* to core *i*. Upon a subsequent access by another core *j* (1), that core encounters a TLB miss and traps into the OS (2). The OS compares the CID in the page table entry with the CID of the core encountering the TLB miss and discovers a mismatch (3). Thus, it needs to re-classify the page as shared.

Upon a re-classification, the OS first sets the page to a temporary *poisoned* state (3). TLB misses for this page by other cores are delayed until the poisoned state is cleared. Once the *Poisoned* bit is set, the OS shoots down the TLB entry (4) and invalidates any cache blocks belonging to the page at the previous accessor's tile[1] (5). When the shoot-down completes, the OS classifies the page as shared by clearing the Private bit in the page table entry (6), removes the poisoned state, and services any pending TLB requests, including the one from core *j*. Because the Private bit is cleared, any core that receives a TLB fill (7) will treat accesses to this page as shared, apply-

1. Block invalidation at the previous accessor is required to guarantee coherence when transitioning from a private to shared classification. The invalidation can be performed by any shoot-down mechanism available to the OS, such as scheduling a special shoot-down kernel thread at the previous accessor's core; instructions or PAL code routines to perform this operation already exist in many of today's architectures.

ing the standard address interleaving over the size-16 cluster (entire aggregate cache) to locate the shared block. Thus, after the TLB fill, the corresponding block is allocated at the L2 slice corresponding to the address interleave of the requested address (e.g., L2 cache slice *k*) (8). Any subsequent accesses to this block will be serviced by that L2 slice.

If a page is private but the thread has migrated from one core to another, a procedure similar to re-classification is employed. The only difference being that after the invalidation of the previous accessor, the page retains its private classification, and the CID in the page table entry is updated to the CID of the new owner.

It is possible that pages have blocks of different categories. For example, a page may have blocks for private data and blocks for shared data; yet, R-NUCA classifies an entire page with a single category. Classifying pages with both private and shared data as *shared-data* results in correct execution, with only minor implications to performance (as we will see in Chapter 3.4.2). Similarly, pages may hold both data and instructions. As long as cache blocks within these pages store only a single class (i.e., within a page, some blocks store only instructions while other blocks store only data, but no blocks store both classes) then R-NUCA classifies blocks correctly.

For convenience, we assume in this work that the operating system separates code from data at the granularity of cache blocks. This separation allows R-NUCA to replicate instructions without requiring hardware coherence mechanisms. It is important to note that this assumption is done for convenience only and has no correctness implications. Even if the operating system did not separate instructions from data at the cache-block granularity, classifying these blocks as *shared-data* results in correct execution, with only minor implications to performance (Chapter 3.4.2).

3.3.4 Extensions

Although our configuration of R-NUCA utilizes only clusters of size-1, size-4 and size-16, the techniques can be applied to clusters of different types and sizes. For example, R-NUCA can utilize fixed-boundary clusters, which have a fixed rectangular boundary and all cores within the rectangle share the same data. The regular shapes of these clusters make them appropriate for partitioning a CMP into equal-size non-overlapping partitions, which may not always be possible with fixed-center clusters. The regular shapes come at the cost of allowing a smaller degree of nearest-neighbor communication, as tiles in the corners of the rectangle are farther away from the other tiles in the cluster.

The indexing policy is orthogonal to the cluster type. Indexing within a cluster can use standard address interleaving or rotational interleaving. The choice of interleaving depends on the block replication requirements. Rotational interleaving is appropriate for replicating blocks while balancing capacity constraints. Standard address interleaving is appropriate for disjoint clusters. By designating a *center* for a cluster and communicating it to the cores via the TLB mechanism in addition to the Private bit, both interleaving mechanisms are possible for any cluster type of any size.

Our configuration of R-NUCA employs fixed-center clusters only for instructions; however, alternative configurations are possible. For example, heterogeneous workloads with different private data capacity requirements for each thread (e.g., multi-programmed workloads) may favor a fixed-center cluster of appropriate size for private data, effectively spilling blocks to the neighboring slices to lower cache capacity pressure while retaining fast lookup.

3.3.5 Generalized Form of Rotational Interleaving

As discussed in Chapter 3.3.1, indexing with rotational interleaving requires that the requesting core evaluates a simple boolean function. Conceptually, this boolean function is a compound function of three boolean functions. Let *Addr* be the address of the block to be placed in a size-*n* cluster of a tiled multicore processor, where the cache index and the block offset use the lowest *k* bits of the address *Addr*. Then the three boolean functions are:

1.The first function ***IL*** is the interleaving function: it maps the address *Addr* of the block to a destination slice RID_{dest} within the cluster. Typically, the *IL* function maps the address bits right above the cache index to the destination *RID*:.

$$IL\colon Addr[k+\log_2(n)-1:k] \rightarrow RID_{dest}$$

2.The next function ***RV*** is the relative-vector function: it maps the tuple $<RID_{center}, RID_{dest}>$ to a destination vector *D*, denoting the location of RID_{dest} relative to the center of the cluster RID_{center}.

$$RV\colon \langle RID_{center}, RID_{dest}\rangle \rightarrow \vec{D}$$

3.The third function ***GM*** is the global-mapping function: it maps the tuple $<CID_{center}, D>$ to the system-wide ID of the destination slice CID_{dest}. The slice CID_{dest} is the location where the block with address *Addr* is placed.

$$GM\colon \langle CID_{center}, \vec{D}\rangle \rightarrow CID_{dest}$$

The *IL* function is similar to the one used in traditional address-interleaving. The *RV* function determines which direction from the center of the cluster (i.e., from the requesting core) to send the access request. It's general form is

$$\vec{D} = RV(\langle RID_{center}, RID_{dest} \rangle) = (RID_{dest} + \overline{RID_{center}} + 1) \;\&\; (n-1)$$

where the & operator denotes the bit-wise AND and the bar operator denotes the power-of-2 complement. The *RV* function is the one that guarantees that overlapping clusters have mutually-consistent interleavings. By construction, the outcomes *D* of the indexing function *RV* and the destination slice RIDs are the same when the function is applied to clusters centered at a slice with RID *0*.

Finally, the *GM* function is topology-dependent. It is relatively easy to construct it for regular network topologies (e.g., in a 2D-torus with *n* tiles and *m* tiles per row, the tile below CID_a is the tile $CID_a + m$, where *n-1* wraps around to *0*). Irregular network topologies may have more complex functions or need to resort to lookup tables. Similarly, the *GM* function provides an additional degree of freedom in the mapping of the cluster to physical tiles. While a size-4 fixed-center cluster logically consists of 4 neighboring tiles, these tiles need not necessarily map to 4 physically neighboring ones. This freedom of mapping allows Rotational Interleaving to share blocks among any tiles in a multicore processor, provided there is a function or lookup table to map a logical cluster to a physical one. It is important to note that, conventional address interleaving is a special case of rotational interleaving, where

$$\forall \text{slice } X, \text{vector } D: \begin{cases} CID_X = RID_X \\ IL\left(Addr\left[k+\log_2(n)-1:k\right]\right) = Addr\left[k+\log_2(n)-1:k\right] \\ RV\left(\langle RID_{center}, RID_X \rangle\right) = CID_X \\ GM\left(CID_{center}, D\right) = D \end{cases}$$

Rotational interleaving can be generalized to any cluster of size a power of 2. The RID assignment algorithm presented in Chapter 3.3.1 provides correct assignments when the size of the

0	1	2	3	4	5	6	7
3	4	5	6	7	0	1	2
6	7	0	1	2	3	4	5
1	2	3	4	5	6	7	0
4	5	6	7	0	1	2	3
7	0	1	2	3	4	5	6
2	3	4	5	6	7	0	1
5	6	7	0	1	2	3	4

0	1	2	3	4	5	6	7
4	5	6	7	8	9	10	11
8	9	10	11	12	13	14	15
12	13	14	15	0	1	2	3
0	1	2	3	4	5	6	7
4	5	6	7	8	9	10	11
8	9	10	11	12	13	14	15
12	13	14	15	0	1	2	3

FIGURE 20: RIDs and examples for size-8 and size-16 fixed-center clusters.

cluster is less than *n/2*. For bigger clusters (i.e., size-*n/2* and size-*n*) the RIDs of tiles in a column differ by *m*, where *m* is the number of tiles in a row. Figure 20 (left) shows the RID assignments and examples of a size-8 fixed-center clusters in a 64-tile processor, while Figure 20 (right) shows the RID assignments and example of a size-16 fixed-center cluster in a 64-tile processor.

3.4 Evaluation

3.4.1 Methodology

For both CMP configurations, we evaluate four NUCA designs: private (P), ASR (A), shared (S), and R-NUCA (R). The shared and private designs are described in Section 3.1.2. ASR [12] is based on the private design and adds an adaptive mechanism that probabilistically allocates clean shared blocks in the local L2 slice upon their eviction from L1. If ASR chooses not to allocate a block, the block is cached at an empty cache frame in another slice, or dropped if no empty L2 frame exists or another replica is found on chip. ASR was recently shown to outperform all prior proposals for on-chip cache management in a CMP[12].

Although we did a best-effort implementation of ASR, our results did not match with [12]. We believe that the assumptions of our system penalize ASR, while the assumptions of [12] penalize the shared and private cache designs. The relatively fast memory system (90 cycles vs. 500

cycles in [12]) and the long-latency coherence operations due to our directory-based implementation ([12] utilizes token broadcast) leave ASR with a small opportunity for improvement. We implemented six versions of ASR: an adaptive version following the guidelines in [12], and five versions that statically choose to allocate an L1 victim at the local slice with probabilities 0, 0.25, 0.5, 0.75 and 1, respectively. In our results for ASR, we report, for each workload, the highest-performing of these six versions.

For the private and ASR designs, we optimistically assume an on-chip full-map distributed directory with zero area overhead. In reality, a full-map directory occupies more area than the aggregate L2 cache, and yet-undiscovered approaches are required to maintain coherence among the tiles with a lower overhead. Such techniques are beyond the scope of this study. Similarly, we assume that ASR mechanisms incur no area overhead. Thus, the speedup of R-NUCA compared to a realistic implementation of the private or ASR designs will be higher than reported in this study.

Our on-chip coherence protocol is a four-state MOSI protocol modeled after Piranha [10]. The cores perform speculative load execution and store prefetching [25,38,79]. We simulate one memory controller per four cores, each controller co-located with one tile, assuming communication with off-chip memory through flip-chip technology. Tiles communicate through the on-chip network. We list other relevant parameters in Table 2.

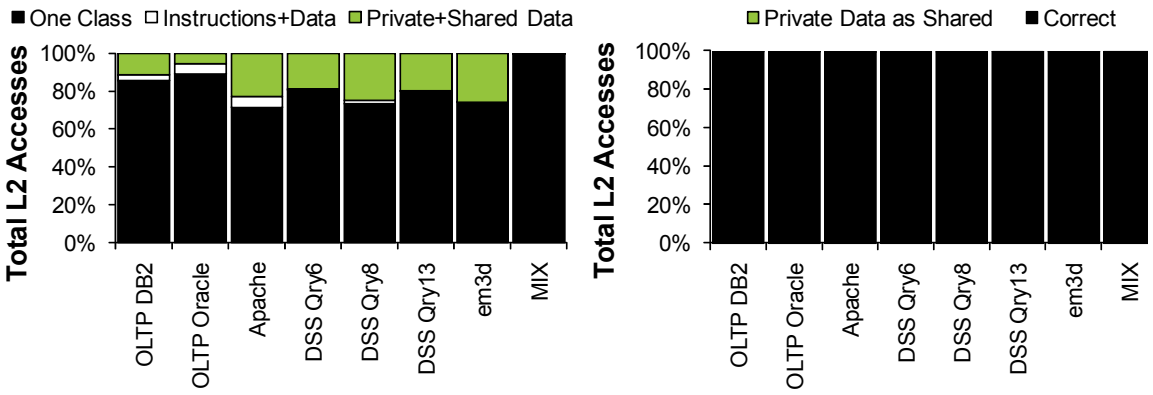
We simulate a 2D-folded torus [29] on-chip interconnection network. While prior research typically utilizes mesh interconnects due to their simple implementation, meshes are prone to hot spots and penalize tiles at the network edges. In contrast, torus interconnects have no edges and treat nodes homogeneously, spreading the traffic across all links and avoiding hot spots. 2D-tori can be built efficiently in modern VLSI by following a folded topology [98] which eliminates long links. While a 2D-torus is not planar, each of its dimensions is planar, requiring only two metal

layers for the interconnect [98]. With current commercial products already featuring 11 metal layers, and favorable comparisons of tori against meshes with respect to area and power overheads [98], we believe 2D-torus interconnects are a feasible and desirable design point.

We measure performance using the SimFlex multiprocessor sampling methodology [103]. Our samples are drawn over an interval of 10 to 30 seconds for OLTP and web server applications, the complete query execution for DSS, one complete iteration for the scientific application, and the first 10 billion instructions after the start of the first main-loop iteration for MIX. We launch measurements from checkpoints with warmed caches, branch predictors, TLBs, on-chip directories, and OS page tables, then warm queue and interconnect state for 100,000 cycles prior to measuring 50,000 cycles. We use the aggregate number of user instructions committed per cycle (i.e., committed user instructions summed over all cores divided by total elapsed cycles) as our performance metric, which is proportional to overall system throughput [103].

3.4.2 Classification Accuracy

Although in Section 3.2 we analyzed the workloads at the granularity of cache blocks, R-NUCA performs classification at page granularity. Pages may simultaneously contain blocks of multiple classes; part of a page may contain private data, while the rest may contain shared data. For our workloads, 6% to 26% of L2 accesses are to pages with more than one class (Figure 21 (a)). However, the accesses issued to these pages are dominated by a single class; if a page holds both shared and private data, accesses to shared data dominate. Classifying these pages as *shared-data* effectively captures the majority of accesses. Similarly, in pages that hold both data and instructions, accesses to instructions dominate.



(a) Access breakdown by block types per page. (b) Access misclassifications.

FIGURE 21: Page-grain access types and misclassifications.

As discussed in Chapter 3.3.3, in this work we assume that the operating system separates code from data at the granularity of cache blocks. This separation allows R-NUCA to replicate instructions without requiring hardware coherence mechanisms. However, it is important to note that this assumption is done for convenience only and has no correctness implications and only minor implications to performance. In the worst case, the white bars in Figure 21 (a) correspond to accesses to cache blocks that store both instructions and data (as opposed to accesses to pages that have blocks storing only instructions and blocks storing only data). Even if the operating system did not separate instructions from data at the cache-block granularity, classifying these blocks as *shared-data* would result in correct execution, at the cost of misclassifying some instructions as shared data (which amounts to less than 6% of total L2 accesses).

Overall, we find that the classification at page granularity results in the misclassification of less than 0.75% of L2 accesses (Figure 21 (b)).

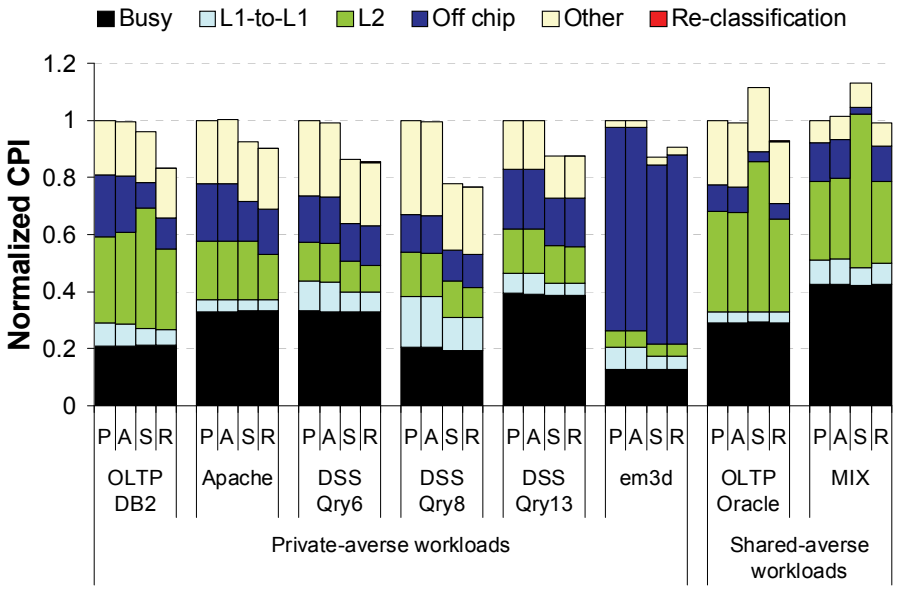


FIGURE 22: Total CPI breakdown for L2 designs. The CPI is normalized to the total CPI of the private design.

3.4.3 Impact of R-NUCA Mechanisms

Because different workloads favor a different cache organization, we split our workloads into two categories: private-averse and shared-averse, based on which design has a higher cycles-per-instruction (CPI). The private design may perform poorly if it increases the number of off-chip accesses, or if there is a large number of L1-to-L1 or L2 coherence requests. Such requests occur if a core misses in its private L1 and L2 slice, and the data are transferred from a remote L1 (*L1-to-L1 transfer*) or a remote L2 (*L2 coherence transfer*). The private and ASR designs penalize such requests, because each request accesses first the on-chip distributed directory, which forwards the request to the remote tile, which then probes its L2 slice and (if needed) its L1 and replies with the data. Thus, such requests incur additional network traversals and accesses to L2 slices. Similarly, the shared design may perform poorly if there are many accesses to private data or instructions,

which the shared design spreads across the entire chip, while the private design services through the local and fast L2 slice.

Figure 22 shows the CPI (normalized to the private design) due to useful computation (busy), L1-to-L1 transfers, L2 loads and instruction fetches (L2), off-chip requests, other delays (e.g., front-end stalls), and the CPI overhead due to R-NUCA page re-classifications. Load latency is affected the most by high on-chip data access latency because commercial server workloads have tight dependencies resulting in cores stalling for data to arrive. While load latency is exposed to the application, store latency may also get exposed when store buffers become full and processing cores stall waiting for the store buffer to drain. However, recent proposals for store-wait-free architectures promise to minimize the store latency [18,100]. Thus, we account for store latency in the *other* category.

Figure 22 confirms that the re-classification overhead of R-NUCA is negligible. Page re-classifications happen only once per shared page, and at steady state amount to less than 0.4% of the total CPI for each workload. Overall, R-NUCA outperforms the competing designs, lowering the L2 hit latency exhibited by the shared design, and eliminating the long-latency coherence operations of the private and ASR designs, while at the same time approximating the low off-chip miss rate of the shared organization.

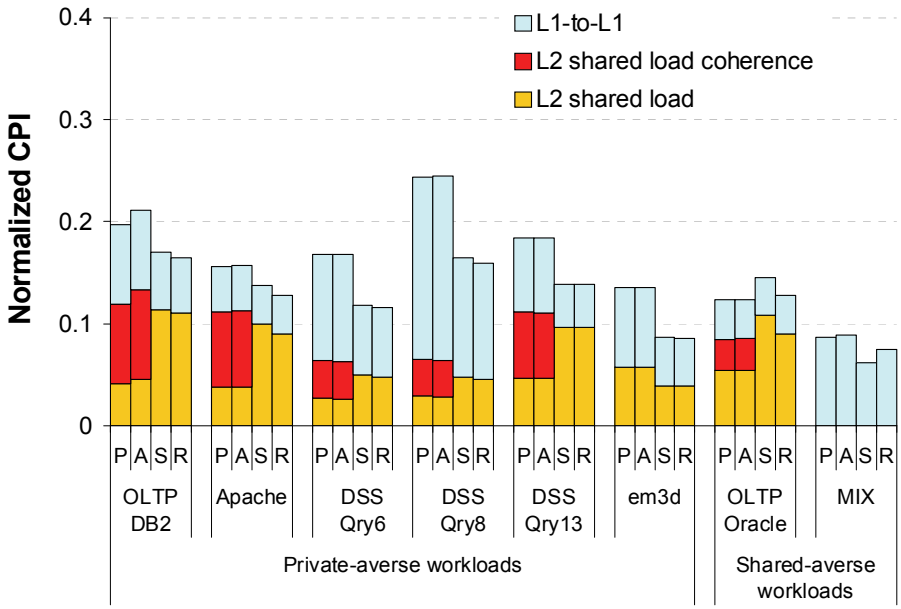


FIGURE 23: CPI breakdown of L1-to-L1 and L2 load accesses. The CPI is normalized to the total CPI of the private design.

Impact of L2 coherence elimination. Figure 23 shows the portion of the total CPI due to accesses to shared data, which may engage the coherence mechanism. Shared data in R-NUCA and in the shared design are interleaved across all L2 slices, with both designs having equal latency. The private and ASR designs replicate blocks, alternating between servicing requests from the local slice (*L2 shared load*) or a remote slice (*L2 shared load coherence*). Although local L2 slice accesses are fast, remote accesses engage the on-chip coherence mechanism, requiring an additional network traversal and two additional tile accesses compared to the shared or R-NUCA designs. Thus, the benefits of fast local reuse for shared data under the private and ASR designs are quickly outweighed by long-latency coherence operations. On average, eliminating L2 coherence requests in R-NUCA results in 18% lower CPI contribution of accesses to shared data. Similarly, the private and ASR designs require accessing both local and remote L2 slices to complete an L1-to-L1 transfer, whereas the shared and R-NUCA designs use only one access to an L2 slice

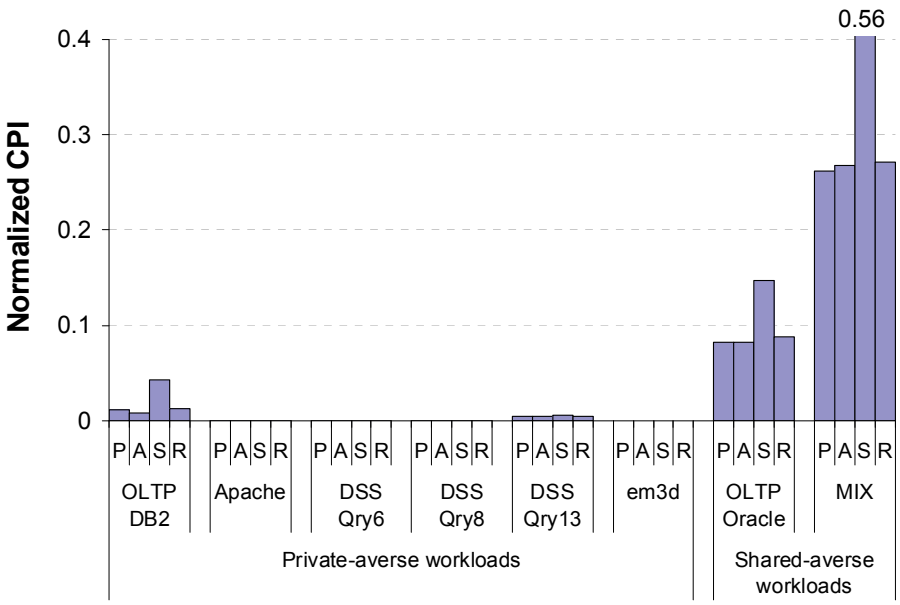


FIGURE 24: CPI contribution of L2 accesses to private data. The CPI is normalized to the total CPI of the private design.

before requests are sent to an L1. By eliminating the additional remote L2 slice access, R-NUCA lowers the latency for L1-to-L1 transfers by 27% on average. Overall, eliminating coherence requirements at L2 lowers the CPI due to shared data accesses by 22% on average compared to the private and ASR designs.

Impact of local allocation of private data. Similar to the private and ASR designs, R-NUCA allocates private data at the local L2 slice for fast access, while the shared design increases latency by distributing private data across all L2 slices. Figure 24 shows the impact of allocating the private data locally. R-NUCA reduces the access latency of private data by 42% compared to the shared design, matching the performance of the private organization.

To accommodate large private data working sets, prior proposals advocate migrating (spilling) these blocks to neighbors [24]. Spilling may be applicable to multi-programmed workloads composed of applications with a range of private-data working set sizes. Figure 25 shows the per-

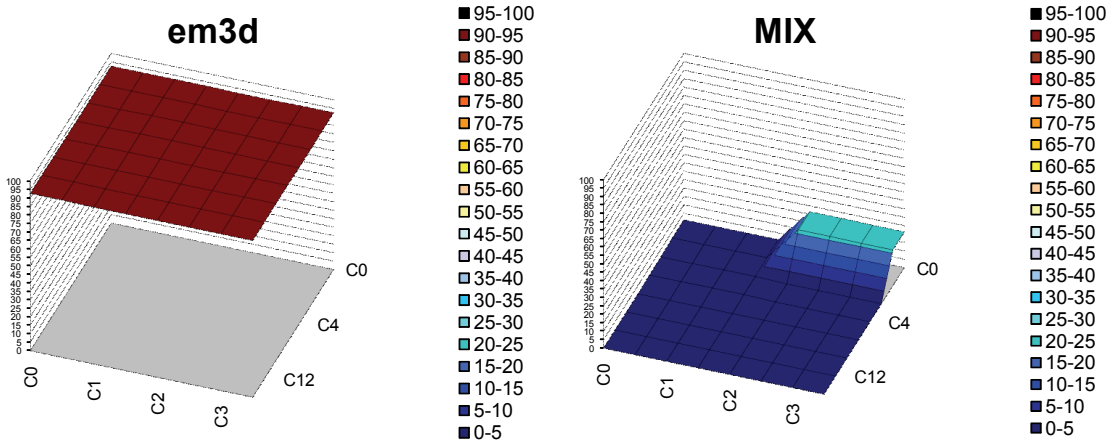


FIGURE 25: Per-core miss rates for scientific and multiprogrammed workloads.

core miss rates of the scientific and multiprogrammed workloads in our suite when running on a CMP with private L2 organization. Scientific workloads typically exhibit high miss rates as they run large models requiring vast amounts of memory. Yet, the miss rate is uniform across cores, as typically the dataset is blocked and distributed among the cores, with each core exerting the same pressure on its local L2 cache slice. Similarly, spilling provides little advantage to server workloads. All cores in a typical server workload run similar threads, with each L2 slice experiencing similar capacity pressure (Figure 26). Migrating private blocks to a neighboring slice is offset by the neighboring tiles undergoing an identical operation and spilling in the opposite direction. Thus, cache pressure remains the same, but requests incur higher access latency.

On the other hand, multiprogrammed workloads run disjoint processes on each core, each with varying capacity requirements. Thus, L2 slices often exhibit significantly higher miss rate than neighboring ones (e.g., the SPEC CPU2000 mcf application running on cores 2 and 3 in Figure 25, right). Such workloads could benefit from spilling to neighboring slices. While in this work we do not investigate private-data spilling, it is easy to extend R-NUCA to utilize rotational interleaving to spill private data to neighboring slices when it provides a performance advantage.

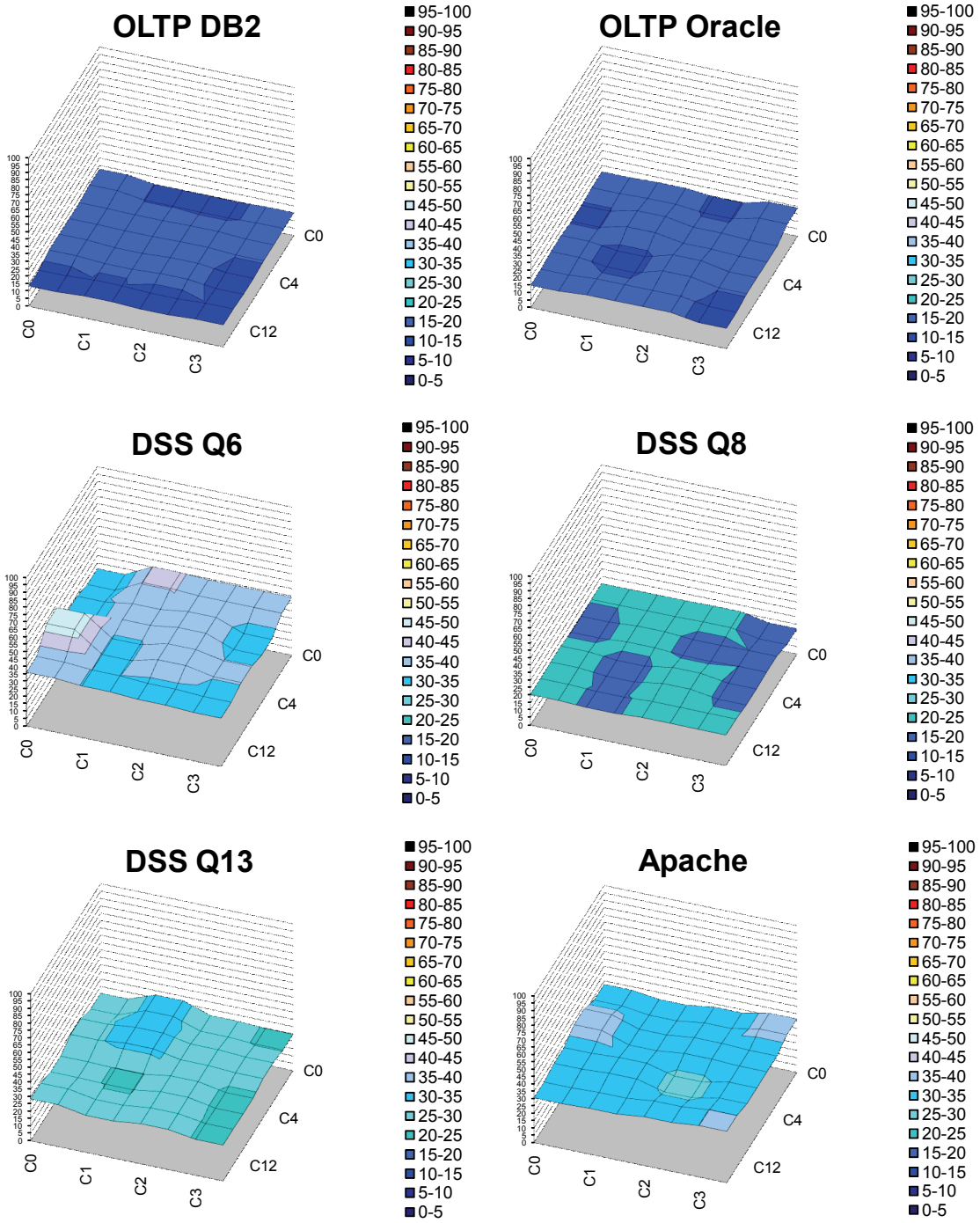


FIGURE 26: Per-core miss rates for server workloads.

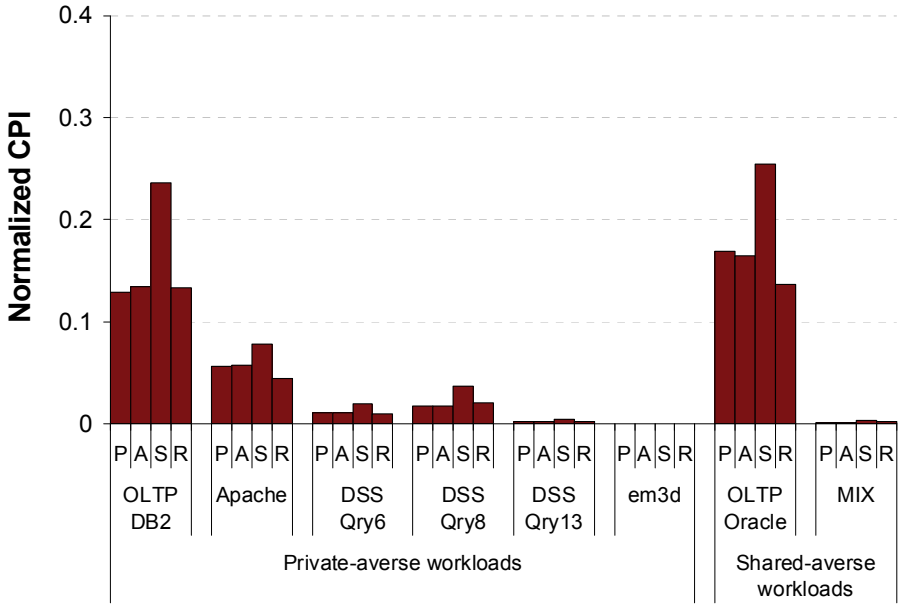


FIGURE 27: CPI contribution of L2 instruction accesses. The CPI is normalized to the total CPI of the private design.

Impact of instruction clustering. R-NUCA's clustered replication distributes instructions among neighboring slices. Replication ensures that instruction blocks are only one hop away from the requestor, and rotational interleaving ensures fast lookup that matches the speed of a local L2 access. In contrast, the shared design spreads instruction blocks across the entire die area, requiring significantly more cycles for each instruction L2 request (Figure 27). As a result, R-NUCA obtains instruction blocks from L2 on average 40% faster than the shared design. In OLTP-Oracle and Apache, R-NUCA even outperforms the private design by 20%, as the latter accesses remote tiles to fill some requests.

While the private design enables fast instruction L2 accesses, the excessive replication of instruction blocks causes evictions and an increase in off-chip misses. Figure 28 compares the performance of instruction clusters of various sizes. We find that storing instructions only in the local L2 slice (size-1) increases the off-chip CPI component by 62% on average over a size-4 cluster,

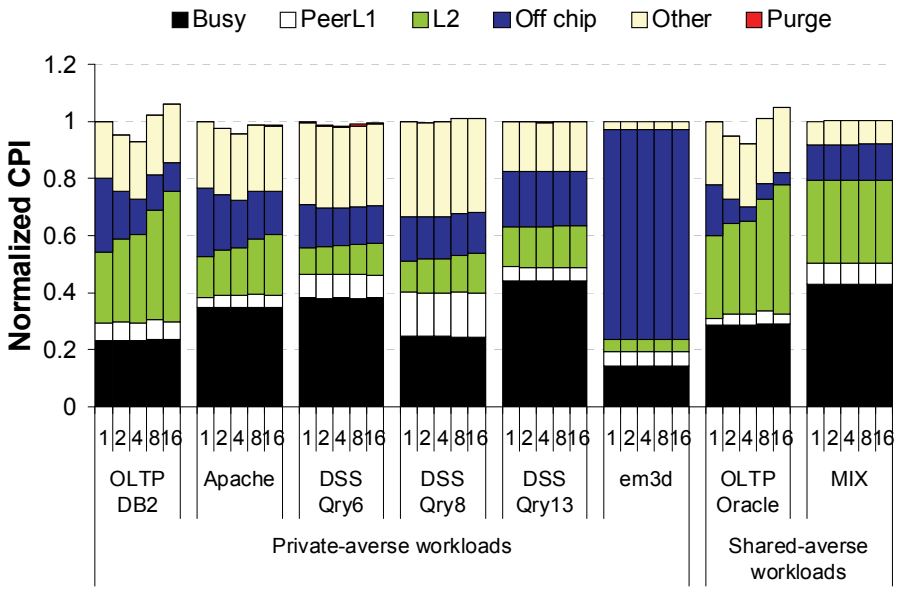


FIGURE 28: CPI breakdown of instruction clusters with various sizes. The CPI is normalized to size-1 clusters.

resulting in reduced performance. At the same time, clusters larger than size-4 spread instruction blocks to a larger area, increasing instruction access latency by 34% to 69% for size-8 and size-16 clusters respectively. We find that, for our workloads and system configurations, size-4 clusters offer the best balance between L2 hit latency and off-chip misses.

3.4.4 Performance Improvement

R-NUCA lowers the CPI contribution of L2 hits by 18% on average compared to the private design, and by 22% on average compared to the shared design. At the same time, like the shared design, R-NUCA is effective at maintaining the large aggregate capacity of the distributed L2. The CPI due to off-chip misses for R-NUCA is on average within 17% of the shared design's for server workloads, while the private design increases the off-chip CPI by 72%. Thus, R-NUCA delivers both the fast local access of the private design and the large effective cache capacity of the shared

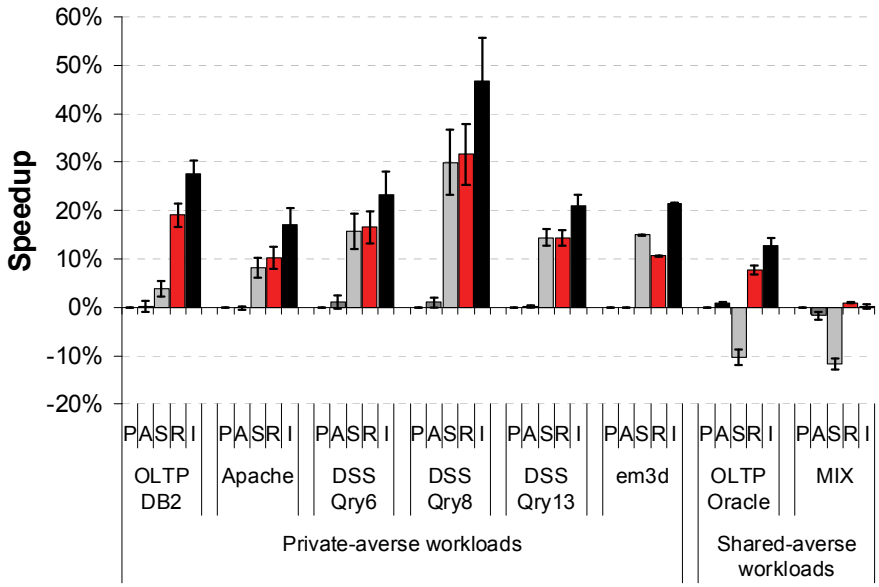


FIGURE 29: Performance Improvement of R-NUCA. Speedup is normalized to the private design.

design, bridging the gap between the two. R-NUCA attains an average speedup of 14% over the private and 6% over the shared organizations, and a maximum speedup of 32%. Figure 29 shows the corresponding speedups, along with the 95% confidence intervals produced by our sampling methodology. The performance difference between the shared design and R-NUCA for em3d stems from the higher off-chip miss rate of R-NUCA for this workload. While R-NUCA compares favorably against the private design because it eliminates L2 coherence for shared data and achieves lower off-chip miss rate, its miss rate is still higher than the shared design's.

In Figure 29, we also show an *Ideal* design (I) that offers to each core the capacity of the aggregate L2 cache at the access latency of the local slice. The ideal design assumes a shared organization with direct on-chip network links from every core to every L2 slice, where each slice is heavily multi-banked to eliminate contention. We find that R-NUCA achieves near-optimal block placement, as its performance is within 5% of the ideal design (Figure 30).

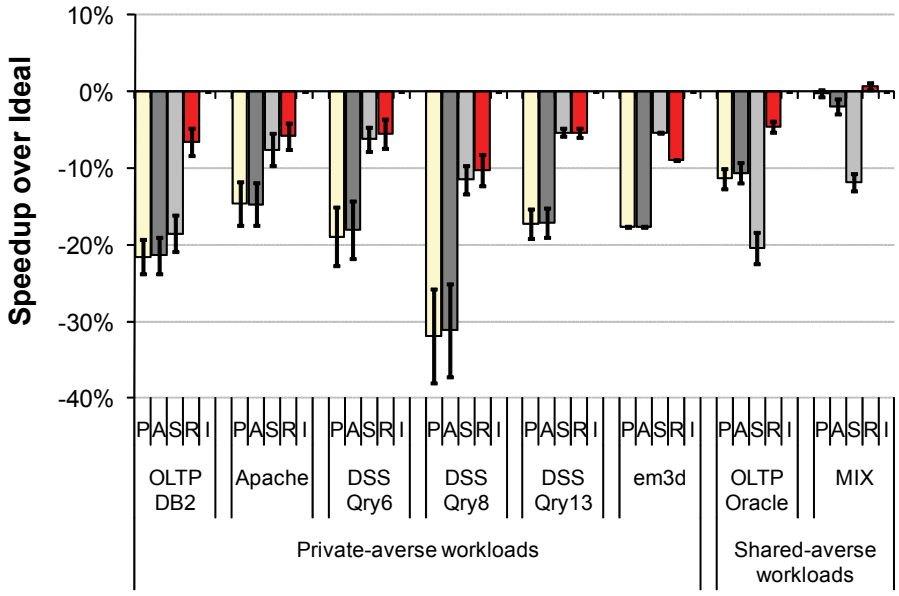


FIGURE 30: Performance of R-NUCA relative to Ideal.

3.4.5 Impact of Technology

As Moore's Law continues and the number of cores on chip continue to grow, the on-chip interconnect and the aggregate cache will grow commensurately. This will make the shared design even less attractive, as cache blocks will be spread over an ever increasing number of tiles. At the same time, the coherence demands of the private and private-based designs will grow with the size of the aggregate cache, increasing the area and latency overhead for accesses to shared data. R-NUCA eliminates coherence among the L2 slices, avoiding the private design's overheads, while still exhibiting fast L2 access times. Moreover, by allowing for the local and nearest-neighbor allocation of blocks, R-NUCA will continue to provide an ever-increasing performance benefit over the shared design. Finally, we believe that the generality of R-NUCA's clustered organizations will allow for the seamless decomposition of a large-scale multicore processor into virtual

domains, each one with its own subset of the cache, where each domain will experience fast and trivial cache lookup through rotational interleaving with minimal hardware and operating system involvement.

3.5 Summary

Wire delays are becoming the dominant component of on-chip communication; meanwhile, the increasing device density is driving a rise in on-chip core count and cache capacity, both factors that rely on fast on-chip communication. Although distributed caches permit low-latency accesses by cores to nearby cache slices, the performance benefits depend on the logical organization of the distributed LLC. Private organizations offer fast local access at the cost of substantially lower effective cache capacity, while address-interleaved shared organizations offer large capacity at the cost of higher access latency. Prior research proposes hybrid designs that strike a balance between latency and capacity, but fail to optimize for all accesses, or rely on complex, area-intensive, and high-latency lookup and coherence mechanisms.

In this work, we observe that accesses can be classified into distinct classes, where each class is amenable to a different block placement policy. Based on this observation, we propose R-NUCA, a novel cache design that optimizes the placement of each access class. By utilizing novel rotational interleaving mechanisms and cluster organizations, R-NUCA offers fast local access while maintaining high aggregate capacity, and simplifies the design of the multicore processor by obviating the need for coherence at the LLC. With minimal software and hardware overheads, R-NUCA improves performance by 14% on average, and by 32% at best, while achieving performance within 5% of an ideal cache design.

While it is imperative that future multicore processors optimize for on-chip data access latency, it is important that we optimize future multicore processors with all constraints in mind, in addition to performance. The abundance of hardware resources comes not only at the cost of increased on-chip data access latencies, but also increased power consumption and off-chip bandwidth requirements. To achieve optimal performance at a given power envelope, it is important to consider all design parameters. With a promising distributed cache design on hand, we embark on exploring the design parameters of physically-constrained multicore processors in the following chapter.

Chapter 4

Optimal CMPs Across Technologies

As Moore's Law continues and the number of transistors on chip rises exponentially, there is enough die real estate to fabricate large-scale CMPs with potentially hundreds of cores. Such CMPs are able to execute several billions of instructions per second under ideal conditions. Unfortunately, this massive processing capability is throttled by the latency gap between the memory subsystem and the processor, and for many commercial server applications, only a fraction of the peak performance can be achieved [2,11,30,41]. Judiciously growing the on-chip cache allows for more data to be serviced from the faster cache rather than the slower main memory, but cache and cores compete for die area. At the same time, power and thermal considerations limit the number of cores that can run concurrently, while leakage current limits the amount of cache that can be employed. While scaling the supply voltage allows for lower overall power consumption, it does so only at the expense of performance. Concurrently, bandwidth constraints impede the ability to feed all cores with data, raising yet another wall that computer architects must consider [82].

Without careful balancing of all constraints and design parameters, the end result is multi-core designs that do not reach their full potential. To reach the required performance levels they dissipate too much power, raising their cost of ownership and operation, especially for large data centers. Enterprise IT is already in shock from the astronomically high budgets required to build, cool and operate warehouse-sized "computers". The high barrier of entry pushes many smaller

businesses to off-load their computational demands to the cloud, simply transferring their IT problem to the cloud provider. Inefficiencies in the datacenter's power infrastructure, along with inefficient computation, are pushing current data centers to an "economic meltdown" [15]. While the datacenter infrastructure is responsible for almost half of the energy expended, servers are responsible for the other half, with processors and memories occupying the largest slice (37%) [34]. Thus, maximizing the overall performance of the processor for a given power budget may hold promise in alleviating some of the energy burden of modern warehouse-style computing.

Parameters such as the supply voltage, clock frequency, core count, cache size, and the workload's stress of the memory hierarchy affect the performance and viability of a chip design in non-linear ways. Thus, it is important to optimize jointly all the parameters of a CMP design. However, complexity and run-time requirements make it impractical to rely on full-system simulation for such a large-scale study. Instead, for our study we build first-order analytical models of performance, power, area, and bandwidth of dominant components, and investigate the effects of technology scaling on optimizing physically-constrained CMPs for commercial workloads. The goal of this effort is not to provide absolute values on how many cores a CMP should have. Rather, our intent is to uncover trends and devise guidelines for promising CMP designs of the future.

4.1 First-Order Analytical Modeling

4.1.1 Technology Model

We model multicore processors across four process technologies: 65nm (in large-scale commercial production since 2007), 45nm (to be used by the majority of new products by 2010), 32nm (due in 2013) and 20nm (due in 2017). For each technology process, we utilize parameters and projections from the International Technology Roadmap for Semiconductors (ITRS) 2008 Edition

[7]. While the ITRS date projections may differ from the projections of other sources forecasting technology, we follow ITRS to ensure our technology assumptions are mutually consistent.

When scaling across technologies, we follow ITRS forecasts on new device types that are expected to replace older ones that don't scale beyond some technology node. In agreement with ITRS, we assume bulk planar CMOS for the 65nm and 45nm nodes, ultra-thin-body fully-depleted MOSFETs for 32nm technology [31], and double-gate FinFETs [91] for the 20nm node. Such devices are already under development at industrial labs and several working prototypes exist.

4.1.2 Hardware Model

4.1.2.1 Core Model

We assume that the cores are built in one of three ways: general purpose cores (GPP), embedded cores (EMB) or ideal cores (Ideal-P). The GPP cores are similar to the cores in Sun's UltraSPARC chips [61,64]. We assume 4-way fine-grain multi-threaded scalar in-order cores, as similar cores have been shown to optimize performance for commercial workloads [41,30]. However, general-purpose cores may still consume an inordinate amount of power and area as compared to carefully tuned embedded processors. Thus, we also run our models on cores similar to ARM11 MPCore [47,4], assuming that the ARM core can deliver the same performance as a single-threaded GPP when the rest of the chip parameters are identical.

Finally, we also evaluate ideal cores (Ideal-P) that have ASIC-like properties, consuming 140x less power and delivering 7x the performance of a general purpose core [16]. This design point provides an upper bound on what cores could achieve. The evaluation of Ideal-P cores is useful for designs in the deep-nanometer regime, where CMPs could house hundreds to thousands of cores. With so much real estate at hand, the chip may offer sets of cores that are heavily optimized

for different sets of functions, in addition to some amount of reconfigurable logic. An application running on such a heterogeneous CMP could switch on only the cores that most closely match the requirements of the work on hand, and run critical parts of the code on the reconfigurable logic, in addition to using some general-purpose cores for the less-critical or complex/uncommon parts of the program. Thereby, many of the cores running could exhibit near-ASIC properties.

4.1.2.2 Cache Model

Each core in our model has 64KB private split data/instruction caches that are 4-way set-associative with 64-byte cache blocks. We assume that micro-architectural techniques hide the L1 access latency in all cases, so the L1 caches in our model simply filter data and instruction accesses to the second-level cache. Our L1 caches are physically-tagged and write-back, and we model a 16-entry victim buffer between L1 and L2.

All cores in our model share a second-level cache ranging in size from 1MB to 512 MB. As caches grow in size and become slower, their optimal organization changes from monolithic caches with a single access latency to a NUCA organization with variable access latency. A NUCA cache is split into multiple slices [60], physically distributed across the die area and connected to each other and to the cores through an on-chip interconnect. Prior research shows that a NUCA organization outperforms any multi-level cache design [60], thus we assume a NUCA second-level cache and do not evaluate deeper on-chip cache hierarchies.

We model L2 caches with 64-byte blocks and 16-way set-associativity. We optimize each L2 cache configuration for each technology node with CACTI 6.0 [77] and use the tool's average access latency estimate in our models. CACTI models the access time, cycle time, area and power characteristics of caches, and optimizes their design for a given technology and configuration. CACTI 6.0 improves upon prior versions of the tool by introducing the ability to model large

Table 4: Performance Model Parameters.

R	Process technology (65nm, 45nm, 32nm, or 20nm)
N	Number of cores
H	Type of cores (general purpose-GPP, embedded-EMB, Ideal-P)
H_0	Baseline GPP processor: 4-way MT, scalar, in-order
M	Size of L2 cache in MB
F	Frequency of on-chip clock in GHz
A	Server application running (OLTP-DB2, DSS, Apache)

NUCA caches through accurate modeling of the on-chip interconnect, and the ability to model different types of wires (RC-based wires with different power, delay, and area characteristics, and differential low-swing buses). In addition to the NUCA organization and interconnect, each slice is also independently optimized and multi-banked for performance [77, 13].

4.1.3 Area Modeling of Hardware Components

We assume a 310 mm^2 die, which corresponds to the die size that high-performance processors can be economically fabricated on [7]. We model proportional-scaling for the cores and cache area [30], allocating 72% of the die for cores and cache, while the remaining area is used by the on-chip interconnect, memory controllers and other system-on-chip components.

We estimate the core area by scaling existing designs. For GPP cores, we scale the cores of the Sun UltraSPARC T1 processor [64]. It is built in 90nm, with an area estimated at 13.67 mm^2 per core in 65nm assuming 64KB split L1 I/D caches. For EMB cores the starting point is ARM11 MPCore, with an estimated core size of 2.48 mm^2 at 65nm. Finally, we assume the size of Ideal-P cores is similar to the EMB ones. We scale the cores across technologies by following ITRS guidelines on transistor size, logic and SRAM density, area efficiency, and SRAM cell area factor for each technology. Finally, we calculate the area required for the L2 cache by calculating the area of the tag and data arrays following ITRS projections and assuming ECC-protected caches.

4.1.4 Performance Modeling

Assume a multicore processor with the characteristics shown at Table 4 The processor's performance when running application *A* is given by Amdahl's Law:

$$Perf(N \text{ cores}, M \text{ MB L2 cache}, F \text{ GHz}, H \text{ core type}, R \text{ process}, A \text{ app}) =$$
$$= \frac{Perf(1, M, F, H, R, A)}{\left(1 - f_{parallel}(A)\right) + \frac{f_{parallel}(A)}{N}}$$

where $f_{parallel}$ is the fraction of the application that can be fully parallelized. We assume that 99% of the application can be parallelized, which is reasonable for commercial server workloads. It is important to consider Amdahl's Law when investigating massive parallelism, as even a small serial portion can severely limit the speedup obtained by throwing more cores at the problem. For example, with a 99% parallelizable application, 128 cores yield a speedup of only 56, while 1024 cores achieve a speedup of only 91, which is an order of magnitude less than linear speedup!

The performance of a single core running application *A* can be estimated by the aggregate number of user instructions committed per cycle (*IPC*), as this metric proportional to overall system throughput [103]:.

$$Perf(1, M, F, H, R, A) = \frac{instructions}{time}$$
$$= \frac{cycles}{time} \times \frac{instructions}{cycles}$$
$$= F \times IPC(M, F, H, R, A)$$

where:

$$IPC(M, F, H, R, A) = IPC(M, F, H_0, R, A) \times CorePerfFactor(H, H_0, A)$$

In the equation above, $CorePerfFactor(H,H_0,A)$ represents the performance of core *H* relative to the performance of core H_0. This factor is 1.7 for a GPP core, as a 4-way multithreaded core

achieves speedup of 1.7 over a single-threaded one when running server workloads [41]. Based on the relative performance of the core models outlined in Chapter 4.1.2.1, we estimate this factor to be 1 for EMB cores, and 7 for Ideal-P cores.

Then, the *IPC* is simply estimated by summing the expected number of cycles an instruction needs to execute assuming an ideal pipeline (one cycle for the execution, and some fractional cycles to account for the probability this instruction accesses the L2 cache or main memory):

$$IPC(M, F, H_0, R, A) = \\ = \frac{1}{1 + L2HitCyclesPerInst(M, F, R, A) + L2MissCyclesPerInst(M, F, R, A)}$$

where:

$$L2HitCyclesPerInst(M, F, R, A) = \Pr(L2hit, M, A) \times LatencyInCycles(L2hit, M, F, R)$$
$$L2MissCyclesPerInst(M, F, R, A) = \Pr(L2miss, M, A) \times LatencyInCycles(L2miss, M, F, R)$$

The probability an instruction accesses the L2 cache (hit or miss) is proportional to the fraction of load/store dynamic instructions in the application and the L1 miss rate, multiplied by the miss rate for the L2 cache. These probabilities along with the corresponding latencies are:

$$\Pr(L2hit, M, A) = \frac{1}{InstructionsBetweenLdSt(A)} \times MissRate_{L1}(A) \times \left(1 - MissRate_{L2}(M, A)\right)$$

$$\Pr(L2miss, M, A) = \frac{1}{InstructionsBetweenLdSt(A)} \times MissRate_{L1}(A) \times MissRate_{L2}(M, A)$$

$$LatencyInCycles(L2hit, M, F, R) = \frac{L2AccessTime(M, R)}{\frac{1}{F}}$$

$$LatencyInCycles(L2miss, M, F, R) = \frac{L2AccessTime(M, R) + MemAccessTime(F, R)}{\frac{1}{F}}$$

The L2 access time is calculated using CACTI 6.0. The memory access time is calculated assuming that DRAM latencies improve at a rate of 7% per year and calculate the speed of a tech-

nology node relative to the speed of main memory in 2007. For the latter, we conservatively assume a DRAM latency of 53ns in 2007 (65nm node), even though similar products were common in the marketplace since 2005 (e.g., PC-533 and PC-667 [26]).

For a memory access, we assume the CMP has an on-chip memory controller with a 2-cycle latency, while the DRAM modules are located on the board about 5cm from the processor chip. The memory access time then is given by:

$$MemAccessTime(F,R) = MCAccess(F) + DRAMAccess(R) + 2 \times Dist(core,mem) \times SignalSpeed$$

where:

$$MCAccess(F) = \text{on-chip memory controller latency} = 2 \text{ cycles} = 2 \times F \text{ ns}$$

$$DRAMAccess(R) = MemAccessTime(0{,}65nm) \times 0.93^{(year(R)-2007)}$$

$$Dist(core,mem) = \text{physical distance between CMP and memory modules } (\sim 5\text{cm})$$

$$year(R) = \text{year of introduction of process } R \text{ according to ITRS'08}$$

With the hardware components in place, we then estimate the application-dependent components (e.g., the application's memory reference intensity, given by the number of instructions between consecutive loads and stores—*InstructionsBetweenLdSt(A)*, and the L1 and L2 cache miss rates). *InstructionsBetweenLdSt(A)* is a characteristic of the application that doesn't depend on the CMP configuration. We measure it for each application in FLEXUS [42,103] by simulating a 16-core CMP with the configuration and application parameters shown in Table 2 and Table 3 respectively. Similarly, through the same simulation runs we measure the L1 miss rate—$MissRate_{L1}(A)$.

4.1.5 Miss Rate Model and Application Dataset Evolution

To estimate the miss rate of the second-level cache ($MissRate_{L2}(A)$) we rely on a combination of modeling and full-system simulations. We simulate our workloads again on FLEXUS on a 16-core CMP and measure each workload's miss rate as a function of the L2 size, which ranges in

Table 5: Miss Rate Model Parameters.

X-Shifted Power Law: $y = \alpha (x + \beta)^{\gamma}$					
	α	**β**	**γ**	mean error	max error
OLTP-DB2	0.5785	0.4750	-0.589	1.3%	8.2%
DSS	0.5925	0.5154	-0.327	0.5%	6.5%
Apache	1.0081	2.1104	-0.503	1.2%	4.9%

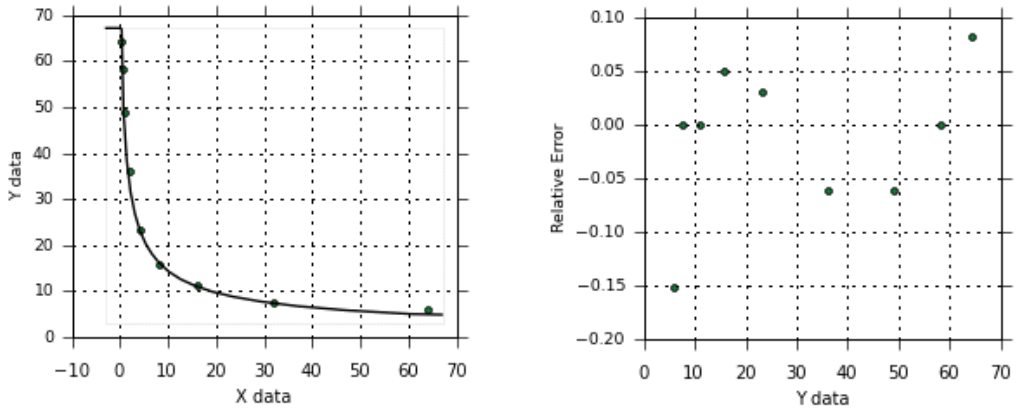
size from 256KB to 64MB. Then, we perform curve-fitting [110] on the simulations' miss rate measurements to find a function that best approximates the measured miss rates.

We evaluate a large pool of over 900 candidate functions with at most three coefficients to control smoothness and over-fitting. The function candidates include polynomials, logarithmic functions, exponentials, hyperbolas, x-shifted and y-shifted power laws, reciprocal functions, and functions prominent in the scientific literature (e.g., Weibull, Steinhart-Hart), along with several variations (e.g., adding linear growth to a function, or exponential decay). The parameters of each function are individually fitted to provide the lowest sum of absolute values of relative errors, a robust curve-fitting technique albeit a slow one. Then, the functions are ranked according to their average error for each of the workloads, and the function with the lowest sum of absolute values of relative errors across workloads is the one used in the model.

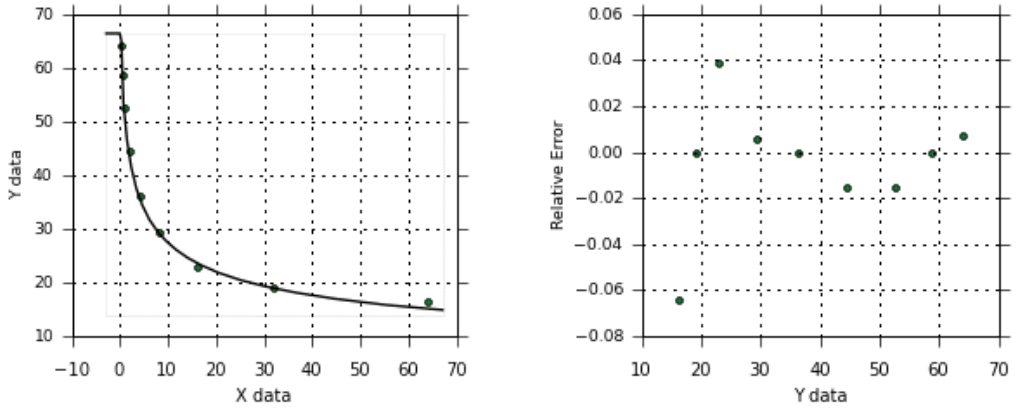
Following this procedure, we find that the function most accurately predicting a cache's miss rate for our workloads is an x-shifted power law, of the form:

$$y = \alpha(x + \beta)^{\gamma}$$

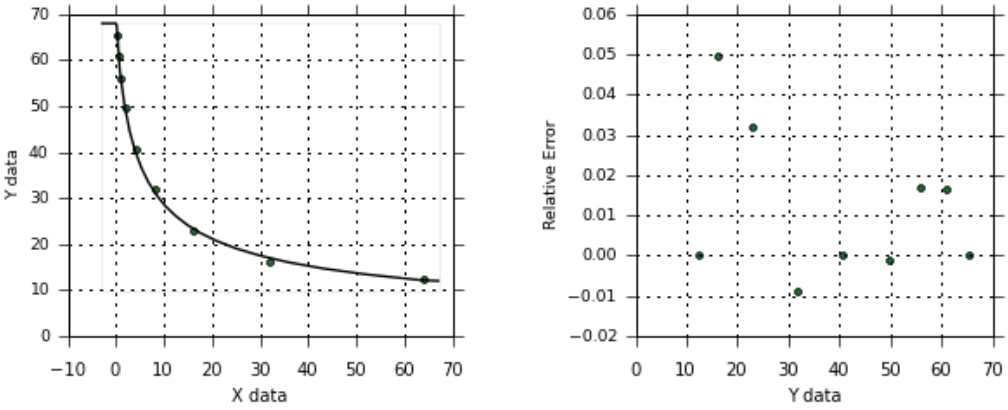
where y is the target miss rate and x is the size of the cache in MB. The parameters for each workload are fitted, and are shown in Table 5, along with the average and maximum errors of the fitted function. Figure 31 shows the agreement between the model and the experimental data on the left, and the relative errors plot on the right for each one of our workloads.



OLTP-DB2 miss rate model fitting (*X* represents cache size, *Y* the miss rate).



DSS miss rate model fitting (*X* represents cache size, *Y* the miss rate).



Apache miss rate model fitting (*X* represents cache size, *Y* the miss rate).

FIGURE 31: Miss rate model fitting (left) and relative error plots (right).

We find that the average error of the x-shifted power law is less than 1.3% across our workloads, while the maximum error is 8.2%. We find that a traditional power law of the form $y = \alpha x^{\gamma}$ which has been used in prior studies to model miss rates [82] fails to capture accurately the miss rate characteristics of commercial server workloads. The best fit for a traditional power law attains average errors of 11% for Apache and 6.4% for OLTP-DB2, with maximum errors of 58% and 23.6% respectively. It is worth noting that the commonly-used rule-of-thumb that quadrupling the cache size halves the miss rate is subsumed by the traditional power law. While the rule-of-thumb is among the most accurate functions with one coefficient, the additional degrees of freedom offered by the x-shifted power law allow it to more accurately capture the miss rate behavior.

To predict a workload's miss rate, we can simply substitute x for the cache size and apply the appropriate fitted function. However, doing so across technologies implicitly assumes that applications do not change over time. However, applications evolve and their datasets grow at an exponential rate. To accurately estimate the miss rate of an application across technologies that span a decade, it is imperative that we consider the application's dataset evolution as well.

Nathan Myhrvold observed that the Microsoft Windows operating system grows at a rate of 33% per year, and argued that software is a gas that expands to fill its container ("Myhrvold's Law") [63]. We corroborate this result by measuring the dataset growth of the Transactional Processing Council's TPC-A, -B, -C, and -E [28] benchmarks' datasets since 1994. These benchmarks are frequently updated to accurately represent the computational demands of online transactional processing and data warehousing in large-scale database management systems deployed at customer sites. Our results indicate that these benchmarks' datasets grow by 29.13% per year, which approximates Myhrvold's Law. Thereby, when estimating the miss rate of a cache we lower its effective size by the expected growth of the application's dataset in the considered time frame.

The workloads we simulate for the miss rate estimation follow the dataset scaling rules in effect circa 2007, so this date is the starting point for the dataset evolution of our analytic modeling. The complete formula to calculate the L2 cache miss rate accounting for the application's dataset evolution by derating the effective cache size is given by:

$$MissRate_{L2}(M, A, R) = \alpha \times \left(M_{effective} + \beta\right)^{\gamma}$$

$$= \alpha \times \left(\frac{M}{DatasetChange(A, R)} + \beta \right)^{\gamma}$$

$$DatasetChange(A, R) = 1.2913^{(year(R)-2007)}$$

With a complete performance model on hand, we then proceed to model the power of the hardware designs we evaluate.

4.1.6 Power Models

The total power is given by the sum of the power of the individual components:

$$Power = P_{cores} + P_{L2,dynamic} + P_{net} + P_{static} + P_{I/O} + P_{misc}$$

where $Power \leq Power_{max}(R)$ as given by ITRS for technology process R. The core power is obtained by scaling the active power $P_{H'}$ of a reference core H' to a new technology R:

$$P_{cores}(N \text{ cores}, H \text{ core type}, R \text{ process}, F \text{ GHz}, T \text{ temperature}, f_{V_{dd}} \text{ scaling factor}) =$$

$$= N \times P_{H'} \times \frac{GateCapacitance(R)}{GateCapacitance(R_{H'})} \times \frac{\left(f_{V_{dd}} \times V_{dd}(R)\right)^2}{\left(V_{dd,H'}\right)^2} \times \frac{F}{F_{H'}}$$

The reference cores used for power estimates are the ones used for the area modeling (Chapter 4.1.2.1). In the equation above, $V_{dd,H'}$ and $F_{H'}$ represent the supply voltage and frequency of the reference core H' in technology node $R_{H'}$, while $V_{dd}(R)$ is the nominal supply voltage of technology R. The scaling factor f_{Vdd} is used to perform voltage-frequency scaling in our

models to trade-off clock frequency for lower power. The supply voltage scales such that $2.3 \times V_{th}(R,T) \leq f_{Vdd} \times V_{dd}(R) \leq V_{dd}(R)$ at a temperature of $T\ ^oK$ [6]. The scaling factor f_{Vdd} is quantized in steps of 10%. Finally, the frequency F also scales with the supply voltage, such that $F \leq F_{max}(R) \times f_{Fmax}(f_{Vdd})$. Frequency doesn't scale exactly linearly with voltage, so we curve-fit published data [17] to describe $f_{Fmax}(\cdot)$.

The L2 dynamic power is estimated similarly by scaling the UltraSPARC T1 cache from published data on power breakdown [64], but it is proportional to the cache activity (access rate):

$P_{L2,dynamic}$(N cores, H core type, R process, F GHz, T temp., $f_{V_{dd}}$ scaling factor, M MB cache, A app) =

$$= M \times P_{C_0} \times RelativeActivity(A,N,H,C_0) \times \frac{GateCapacitance(R)}{GateCapacitance(R_{C_0})} \times \frac{\left(f_{V_{dd}} \times V_{dd}(R)\right)^2}{\left(V_{dd,C_0}\right)^2} \times \frac{F}{F_{C_0}}$$

The activity factor is calculated by the relative L1 miss rates of our designs and workloads over the reference cache design:

$$RelativeActivity(A,N,H,C_0) = \frac{N \times CorePerfFactor(H,H_0,A) \times \frac{1}{InstrBetweenLdSt(A)} \times MissRate_{L1}(A)}{N_0 \times \frac{1}{InstrBetweenLdSt(A_{C_0})} \times MissRate_{L1}(A_{C_0})}$$

Similarly, the network power is activity based, and scaled over the same reference design. The activity of the network is the same as the activity of the cache on the number of messages injected to the network, with an additional factor reflecting the average hop count of each message assuming a 2D-torus on-chip interconnect:

P_{net}(N cores, H core type, R process, F GHz, T Temp., $f_{V_{dd}}$ scaling factor, A app) =

$$= P_{Net_0} \times RelativeActivity(A,N,H,Net_0) \times \frac{\frac{2\sqrt{N}}{4}}{\frac{2\sqrt{N_{Net_0}}}{4}} \times \frac{GateCapacitance(R)}{GateCapacitance(R_{Net_0})} \times \frac{\left(f_{V_{dd}} \times V_{dd}(R)\right)^2}{\left(V_{dd,Net_0}\right)^2} \times \frac{F}{F_{Net_0}}$$

The static power estimator for the cache is proportional to the size of the reference cache C_0, the ratio of the subthreshold leakage current of the target and the reference technologies for the corresponding temperatures, and the ratio of the average transistor's gate width. We model across technologies an average ratio of gate length to gate width of 3 [7]:

$$P_{static}(M \text{ MB cache}, R \text{ process}, f_{V_{dd}} \text{ scaling factor}, T \text{ temperature}) =$$
$$= \frac{M}{M_{C_0}} \times P_{static,C_0} \times \frac{f_{V_{dd}} \times V_{dd}(R)}{V_{dd,C_0}} \times \frac{I_{SD}(R,T)}{I_{SD}(R_{C_0},T_{C_0})} \times \frac{AvgWidth_{gate}(R)}{AvgWidth_{gate}(R_{C_0})}$$

We model only the subthreshold leakage and ignore the gate and junction leakage, as prior research shows they have only a small contribution to the overall leakage of a cache [81]. The leakage current is exponentially dependent on the chip's temperature, however the ITRS projections are given for a target temperature of only 25 ^{o}C. We estimate the change in the leakage current at a typical operating temperature of today's CMPs (66 ^{o}C [64]) similar to [65, 49]:

$$I_{SD}(R,T) = I_{SD}(R,T_0) \times \frac{g(T)}{g(T_0)}$$
$$g(T) = T^2 \times e^{\frac{-V_{th}(T)}{n(T) \times V_T(T)}}$$

$n(T)$ = subthreshold swing coefficient at T °Kelvin (fitted)
$V_T(T)$ = thermal voltage at T °Kelvin $= \frac{K \times T}{q}$
K = Boltzmann constant, q = electrical charge of electron

$V_{th}(T) = V_{th}(T_0) - k \times (T - T_0)$
k = threshold voltage temperature coefficient $= k_0 + \gamma \times (T - T_0)$ where k_0, γ fitted
$V_{th}(T_0)$ from ITRS'08

Because a large part of the chip could potentially be populated by cores, it is important in the leakage calculation to add the leakage of the cores. We calculate the leakage of the cores by

estimating the number of transistors in a core using ITRS logic transistor density projections, and assuming that, at any given time, ½ of the bits remain the same and the corresponding transistors do not switch and leak:

$$P_{static}(N \text{ cores}, H \text{ core type}, R \text{ process}, f_{V_{dd}} \text{ scaling factor}, T \text{ temperature}) =$$
$$= 0.5 \times N \times Area_H \times LogicTransistorDensity(R) \times AvgWidth_{gate}(R) \times I_{SD}(R,T) \times f_{V_{dd}} \times V_{dd}$$

The power of the I/O subsystem is also calculated proportionally to the reference GPP core design by calculating the relative I/O activity based on the estimated L2 miss rate the injection rate to the L2 cache by all participating cores. Because bandwidth is a limited resource, in the worst case the power would be the power expended when all the I/O pins are fully utilized. To account for that, we cap the bandwidth by the maximum allowed by the technology and the I/O subsystem we evaluate. In a conventional I/O subsystem, the memory modules are located off the die and communication is performed through the package's pins. The I/O power then is estimated by:

$$P_{I/O}(N \text{ cores}, H \text{ core type}, R \text{ technology}, Vdd \text{ Volts}, F \text{ GHz}, A \text{ application}, S \text{ I/O subsystem}) =$$
$$= MIN\left(\begin{array}{l} \left(\begin{array}{l} P_{I/O,H'} \times \dfrac{N \times CorePerfFactor(H, H_0, A)}{N_{H'} \times CorePerfFactor(H', H_0, A)} \times \dfrac{F}{F_{H'}} \times \dfrac{MissRate_{L_2}(H, A)}{MissRate_{L_2}(H', A)} \\ \times \dfrac{\text{GateCapacitance}(R)}{\text{GateCapacitance}(R_{H'})} \times \dfrac{V_{dd}(H,R)^2}{V_{dd}(H', R_{H'})^2} \times \dfrac{F_{off-chip}}{F'_{off-chip}} \end{array}\right), \\ \left(\begin{array}{l} P_{I/O,H'} \times \dfrac{BW\max(R,S)}{BW\max(R_{H_0}, S_0)} \\ \times \dfrac{\text{GateCapacitance}(R)}{\text{GateCapacitance}(R_{H'})} \times \dfrac{V_{dd}(H,R)^2}{V_{dd}(H', R_{H'})^2} \times \dfrac{F_{off-chip}}{F'_{off-chip}} \end{array}\right) \end{array}\right)$$

Finally, the power of the miscellaneous system-on-chip components is calculated similarly to the active power of the core by scaling the power of the reference design across technologies.

4.1.7 Off-Chip Bandwidth Model

We model the bandwidth a chip design requires again relative to the same reference design that we use to estimate power, by modeling the relative off-chip activity rate. The baseline bandwidth numbers are obtained through simulation of the application on the reference design. Thus, the bandwidth of a multicore chip is given by:

$$BW(N \text{ cores}, H \text{ core type}, F \text{ GHz}, A \text{ application}) =$$
$$= BW_{H'} \times \frac{N \times CorePerfFactor(H, H_0, A)}{N_{H'} \times CorePerfFactor(H', H_0, A)} \times \frac{F}{F_{H'}} \times \frac{MissRate_{L_2}(H, A)}{MissRate_{L_2}(H', A)}$$

4.1.8 Modeling 3D-Stacked Memory

Besides evaluating a conventional memory system, where memory is located on board close to the processor, we evaluate CMPs where memory is placed on a 3D-die stack on top of the processing cores and the L2 cache. We base our estimates of 3D-stacked memory on [69]. Each layer can host 8 Gbits of memory at 45nm technology, with a worst-case power consumption of 3.7W. We assume the 3D-die has 8 layers of memory arrays, for a total of 8GB. An additional layer is required in the stack to host controllers and logic, for a total of 9 layers. The additional 9 layers increase the average temperature of the chip by an estimated 6.5 ^{0}C. Thus, when we evaluate 3D-stacked designs, we account for the effects of the increased temperature on power.

Communication between the cores/cache and the 3D-stacked memory is done through vertical buses. Because each layer is only a few microns thick, and the on-chip clocks have stopped scaling exponentially, we estimate that a vertical bus can be traversed within a single cycle. We estimate the area for a 1Kbit vertical bus at 45nm is 0.32 mm^2, and model 8 such buses in our designs for a total of 2.56mm^2 area occupancy. With a total of 8Kbits in vertical buses connecting

the cores/cache to the 3D-stacked memory, and with a 1 GHz clock, the buses can deliver a stunning 1TB/s of bandwidth to the memory arrays in the stack, pushing the bandwidth wall far out.

In our models, we treat the 3D-stacked memory as a large L3 cache because the memory it houses is not enough for a full large-scale server software installation. Thus, we update our analytical models accordingly, in effect employing 2 memory subsystems: one that extends from the L2 cache to the 3D-stack, and one that extends from the 3D-die to the memory modules on board. For the 3D-stack on chip, we assume that memory access time improves an additional 32.5% due to more efficient communication between the cores the memory in the 3D-stack [69]. We model the miss rate of the 3D-stack using the same x-shifted power law we employ for the L2 cache.

4.2 Peak-Performing Designs Under Physical Constraints

There is already a large gap between the area and power envelopes in modern multicore processors, which is aggravated as process technology progresses. The “Area” curve in Figure 32 shows the core count-cache size area trade-off in a sample multicore processor with GPP cores at 20nm running our DSS workload, while the “Power” curve shows the power trade-off assuming maximum on-chip clock frequency. While we can fit potentially hundreds of cores, we can only power a handful of them at maximum frequency due to inordinate power requirements. This observation is typically called the power wall, which prohibits the chip from being fully populated.

In reality, however, the power wall is a power-performance trade-off. Lowering the voltage lowers the power consumption so a design can power a few more cores and cache, as shown by the curves between the power and area envelopes which assume lower supply voltage than the ITRS projections. It is even possible to fully populate the chip, as shown by the intersection of the area and 0.27V curves. However, this comes at the expense of lowering frequency and thus perfor-

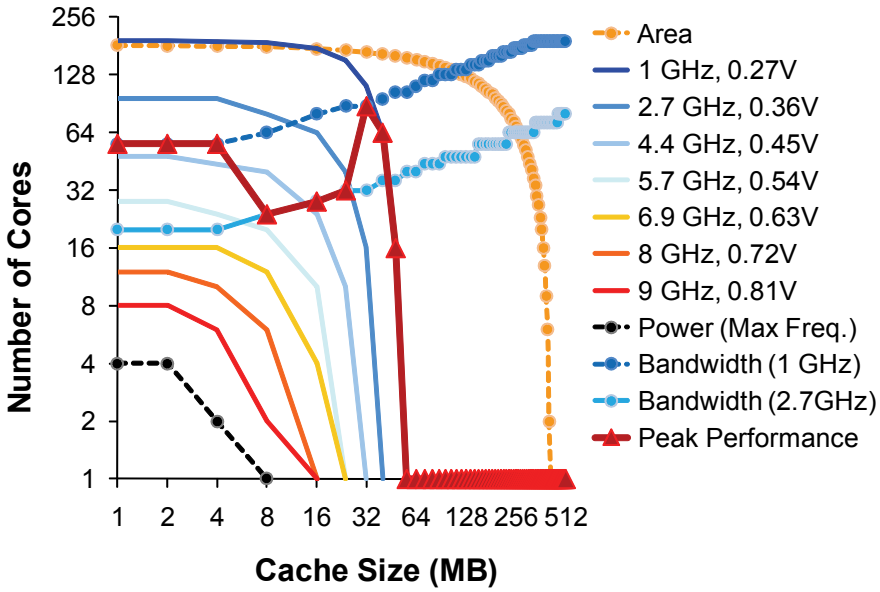


FIGURE 32: Core count-cache size trade-off subject to physical constraints.

mance. While the technology supports frequencies in excess of 10GHz, a 0.27V supply voltage confines the chip to a mere 1GHz clock.

At the same time, pin bandwidth presents another obstacle. While transistor counts double every three years, pin counts remain constant and their speed doubles every six years [7]. Thus, off-chip bandwidth falls behind and severely limits multicore chips. Because the bandwidth required by a CMP is a function of its performance, the bandwidth constraint pushes designs towards the bottom-right corner in Figure 32, favoring designs with fewer cores but larger caches which filter out more memory requests. For clarity, Figure 32 shows only the bandwidth limits corresponding to 1GHz and 2.7GHz clocks for each core count-cache size configuration.

The requirements imposed by the physical constraints may run contrary to each other. While bandwidth limitations favor larger caches, power constraints favor smaller ones that leak less. At the same time, performance may impose conflicting requirements. More and faster cores translate

into higher performance, but the power and bandwidth walls favor fewer and slower ones. The discovery of the best design, thus, is subject to a careful balance between the conflicting requirements imposed by the physical constraints, and the desire for higher performance of the chip.

We discover the peak-performance designs by fixing one design parameter at a time, eliminating from the resulting designs the ones that are not viable because they exceed a physical constraint, and then selecting the highest performing design from the remaining set. The design we select at each step is a potential candidate for the overall best design point. The overall best design is selected after comparing all potential candidates at the end of the selection process.

The peak-performance curve in Figure 32 shows the progression of our algorithm in the core count-cache size plot. In essence, our algorithm walks each time across the most limiting physical constraint, while exploring the slack afforded by the other constraints to improve performance. In the example of Figure 32, power limitations result in lowering the clock rate and voltage initially to 1GHz / 0.27V, at which point the most limiting constraint is bandwidth. The algorithm walks along the bandwidth limit, and when it reaches a larger 8MB cache that can support more cores or fewer faster ones, it discovers it is better to utilize fewer but faster cores. Thus, it increases the clock rate and jumps to the 2.7GHz bandwidth line with fewer cores.

When it reaches a large enough cache (32MB) it meets the power wall for the corresponding voltage of 0.36V. To increase performance, it lowers the voltage back to 0.27V / 1GHz which allows it to power 88 cores instead of the previous 32. Shortly thereafter, though, it meets the power wall again, and from that point on the chip remains power constrained until there are no designs left to explore. The highest performing design among all the candidates lies at the intersection of the power and bandwidth limits for 2.7GHz / 0.36V, where both constraints are balanced.

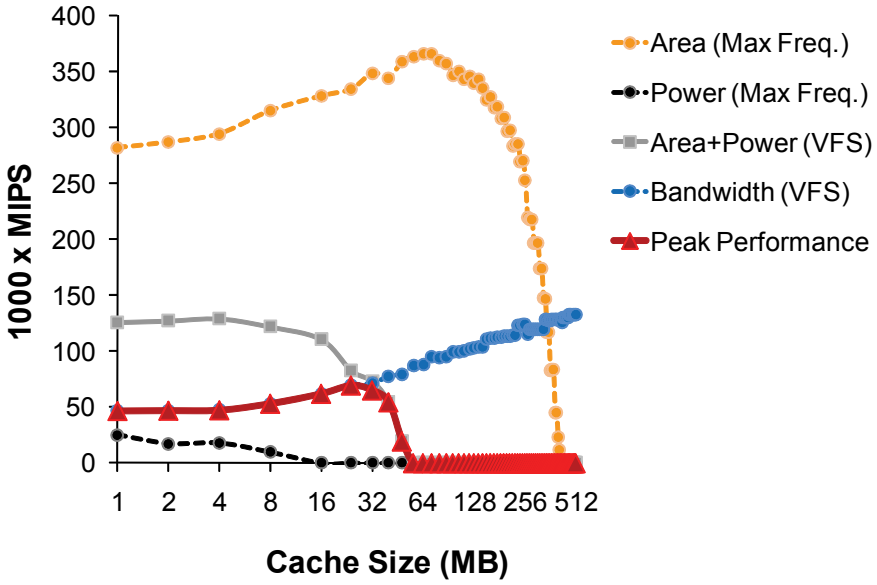


FIGURE 33: Performance of physically-constrained designs.

The performance of multicore processors with GPP cores at 20nm running our DSS workload is presented in Figure 33. The Area curve shows the performance of area-constrained designs at maximum frequency assuming unlimited power and bandwidth, while the Power curve shows the performance of power-constrained designs at maximum frequency with unlimited area and bandwidth. The area and power constraints are combined in the Area+Power curve, which performs voltage-frequency scaling (VFS) to find the highest-performing designs assuming unlimited bandwidth. The bandwidth constraint is represented by the Bandwidth curve, which performs VFS to find the highest-performing design subject only to bandwidth constraints. Figure 33 clearly shows that the peak-performing designs are initially bandwidth and later power constrained, while the best design lies at the intersection of the constraints.

It is important to note that while the Area+Power VFS curve in Figure 33 combines the area and power constraints, the designs it represents are only power-constrained and don't reach the

area limit. Thus, the curve represents the real power wall. The power wall doesn't mean we cannot power the entire chip; rather, it means that power constraints impose a limit on performance.

4.3 Physically-Constrained Designs Across Technologies

To mitigate the power wall, some processors utilize high-V_{th} transistors for non-time-critical components to lower the leakage current. Such low-operational-power (LOP) transistors achieve orders of magnitude lower subthreshold leakage current, while retaining 54%-68% of the switching speed of their high-performance (HP) counterparts [7]. Caches are a prime candidate for using LOP transistors, as their activity level is significantly lower than the cores' and their high density results in high aggregate leakage.

In Figure 34 and Figure 35 we explore CMPs with GPP cores that utilize (i) HP transistors for the entire chip—left column, (ii) HP transistors for the cores and LOP for the cache—middle column, and (iii) LOP transistors for the entire chip—right column. In the interest of brevity, Figure 34 shows results only for OLTP across technologies; the trends are similar for the other workloads and core technologies. The transistors we evaluate are described in more detail in [7].

Because full-HP designs are severely power-limited across technologies (Figure 34, left), they can only power a few cores. While the chip at 20nm can fit 180 cores, HP designs can hardly power more than 32 (Figure 35). Utilizing LOP transistors for the cache enables larger caches that can support more cores and yield higher performance (Figure 34 and Figure 35, middle). At 20nm, HP/LOP designs support around 64 cores, twice the HP count. But, cores leak as well because, on average, only half of the transistors switch every cycle. At 64 cores, about 20% of the chip power is dissipated due to leaky cores.

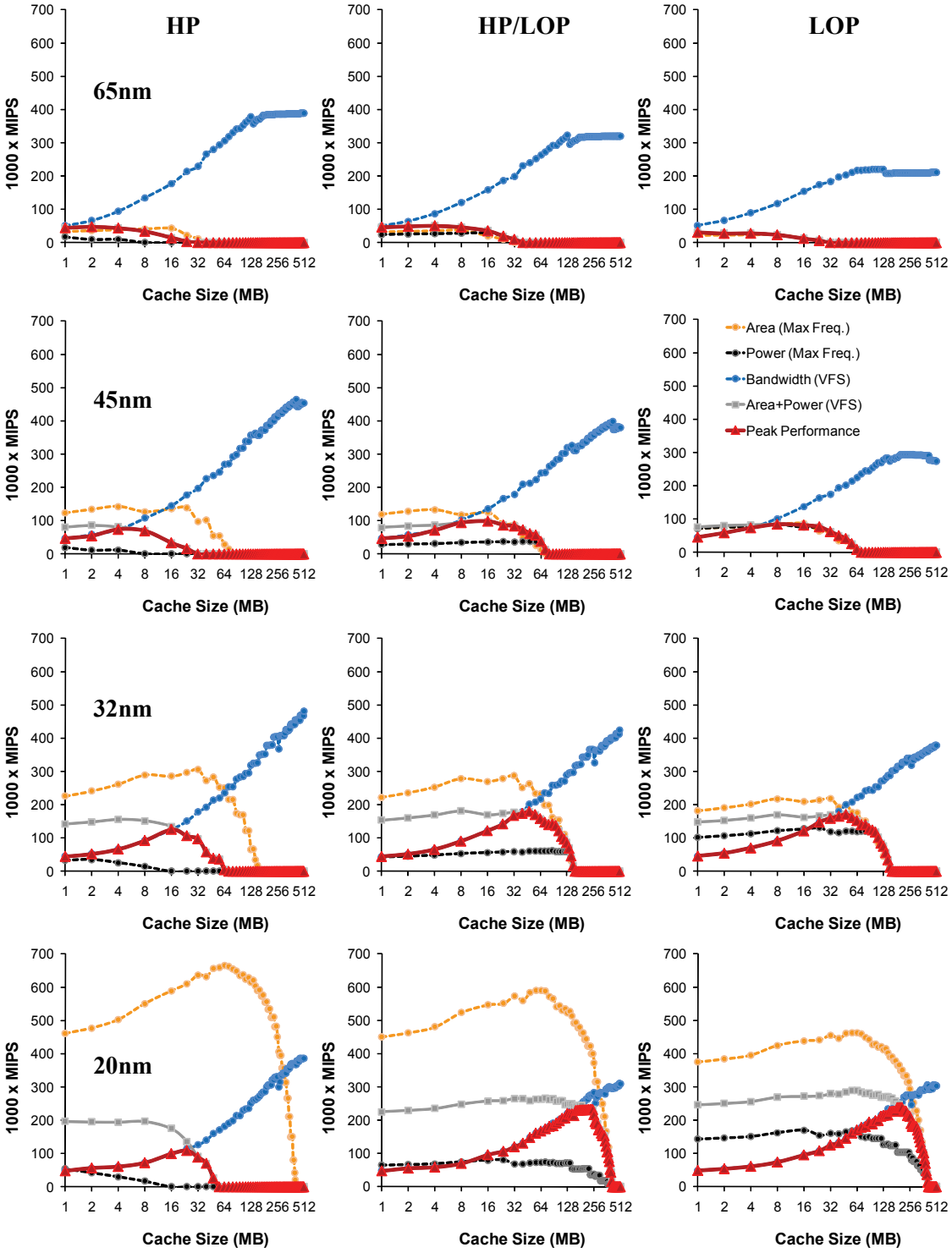
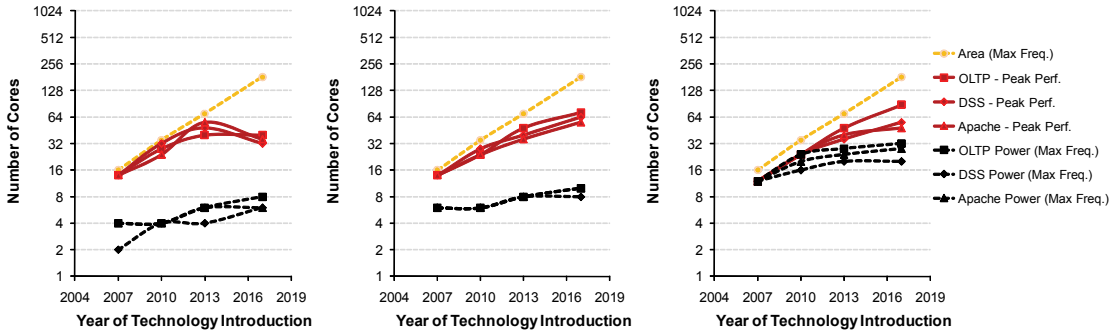


FIGURE 34: Performance of GPP CMPs across technologies and device types.



HP HP/LOP LOP

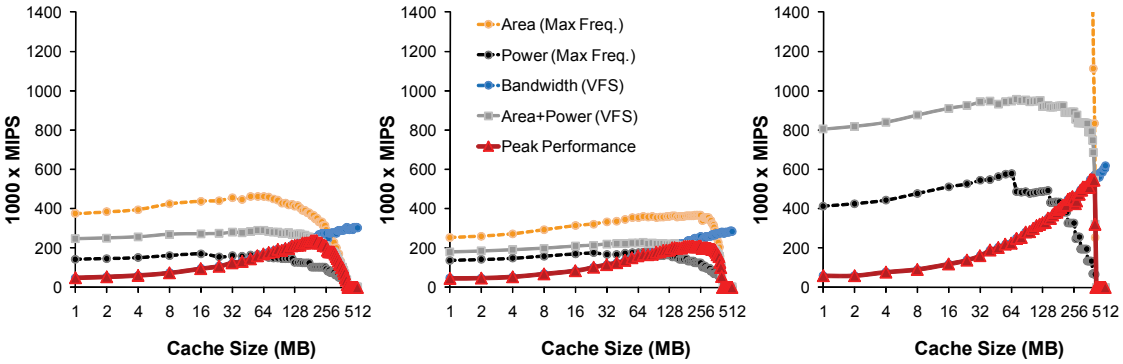
FIGURE 35: Core count for peak-performance HP, HP/LOP and LOP designs.

Implementing cores with LOP transistors can eliminate core leakage, at a potential hit in performance. However, to tame power consumption, the peak-performing designs employ on-chip clocks at less than 43% the maximum frequency supported by the technology. While LOP transistors are slower than HP ones, they still retain 54%-68% of the maximum switching speed and are well within the realm of the optimal clock rate. As such, LOP devices can be used to implement the (already slow) cores. Full-LOP CMPs attain similar performance as HP/LOP ones (Figure 34, right) and can achieve a 25% higher performance per watt.

In the interest of clarity, we focus the remainder of the discussion on LOP designs. While such designs offer high performance at lower power than their HP and HP/LOP counterparts, power-efficient cores like the ones commonly used in embedded systems hold the potential to push the power envelope even further.

4.3.1 Multicore Processors With milliWatt Cores

Lean cores deliver high performance when running commercial server workloads at reasonable power consumption (e.g., Sun UltraSPARC T1 consumes 2W per core [64]). However, the

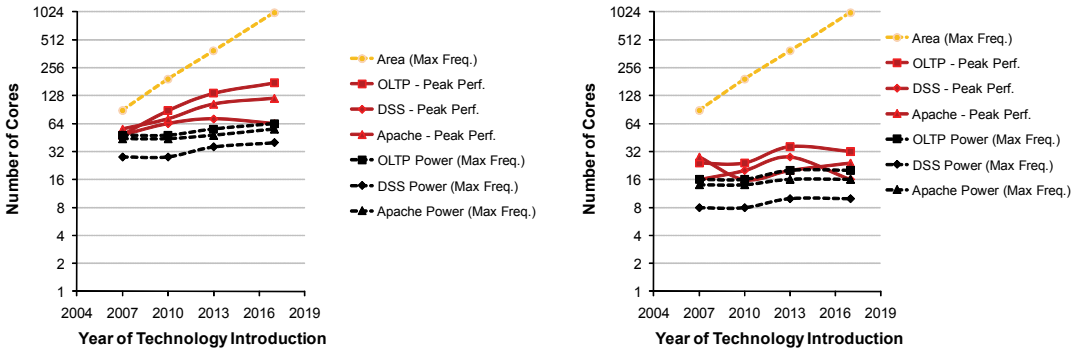


General Purpose (GPP) Embedded (EMB) Ideal (Ideal-P)

FIGURE 36: Performance of GPP, EMB, and Ideal-P 20nm CMPs running OLTP.

embedded systems have long been dominated by milliWatt cores that deliver reasonable performance at orders of magnitude lower power. For example, ARM1176JZ(F)-S consumes 279 mW while packing an 8-stage, scalar, out-of-order pipeline, with dynamic branch prediction, separate Ld/St and arithmetic pipelines, a SIMD unit, and a vector floating-point co-processor [5]. Prior research has indicated that simple in-order cores are the cores of choice for commercial server workloads [30,41]. These workloads typically exhibit tight data dependencies and adverse memory access and sharing patterns that hinder most microarchitectural optimizations. Thus, simple yet efficient embedded cores also present a viable alternative for a multicore building block.

We evaluate such systems using ARM11 MPCore (Chapter 4.1.2). We find that they generally exhibit trends similar to GPP-based multicores. The peak-performing designs are initially bandwidth-constrained, later become power-constrained, and the best design points lie at the intersection of the constraints (Figure 36). Both GPP and EMB designs require similar-sized caches to support multiple cores and remain within the bandwidth envelope.



Embedded Cores (EMB) Ideal Cores (Ideal-P)

FIGURE 37: Core count of CMPs with embedded and ideal cores.

However, we find that to reach peak performance, EMB multicores require almost twice the cores of their GPP counterparts (Figure 35 and Figure 37). While the additional cores deliver significantly higher performance in today's 65nm technology (Figure 38), at smaller technologies the number of cores is so high that the additional cores provide only diminishing returns due to Amdahl's Law. As a result, while the best 20nm EMB design packs 176 cores on chip rather than the 88 of the GPP design, it attains a mere 15% improvement in performance.

At the same time, the larger number of cores requires a large interconnect that dissipates almost half of the chip's power and 68% more power than the GPP-based interconnect (Figure 39). This leads to EMB designs yielding performance per watt similar to the GPP ones, as the power efficiency of the EMB cores is outweighed by the power consumption of the larger interconnect.

Due to the power requirements of large interconnects and the impact of Amdahl's Law on massive on-chip parallelism, EMB designs are expected to provide some benefit for a few technology nodes, but only marginal benefits at smaller ones. Novel interconnects (e.g., concentrated mesh topologies [9]) may lower the power requirements of large-scale EMB designs and make

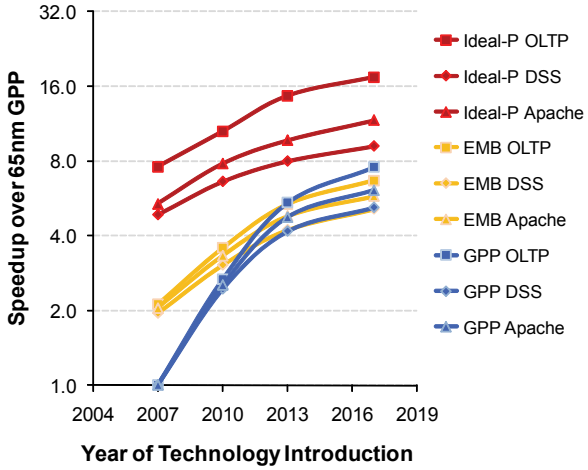


FIGURE 38: Speedup of CMPs with embedded and ideal cores.

them more viable, but further research and innovation is required in this field, in addition to rethinking the software stack to extend Amdahl's Law.

Because Amdahl's Law prohibits large core counts from delivering high aggregate performance (except for embarrassingly parallel applications), an alternative is to deliver higher performance with fewer cores. This can be achieved through heterogeneous computing, where a multicore chip may house hundreds of cores but every time power only the ones most useful to the application. The low number of cores lowers the impact of Amdahl's Law, while the match-making of the cores to the application's requirements holds the potential to provide high performance with high power efficiency in most cases.

4.3.2 CMPs with Ideal Cores

We explore the possible returns of heterogeneous computing by evaluating multicore chips built out of Ideal-P cores with ASIC-like properties: Ideal-P cores deliver 7x the performance of a single-threaded GPP core at 1/140th the power. While it is questionable whether cores may ever be

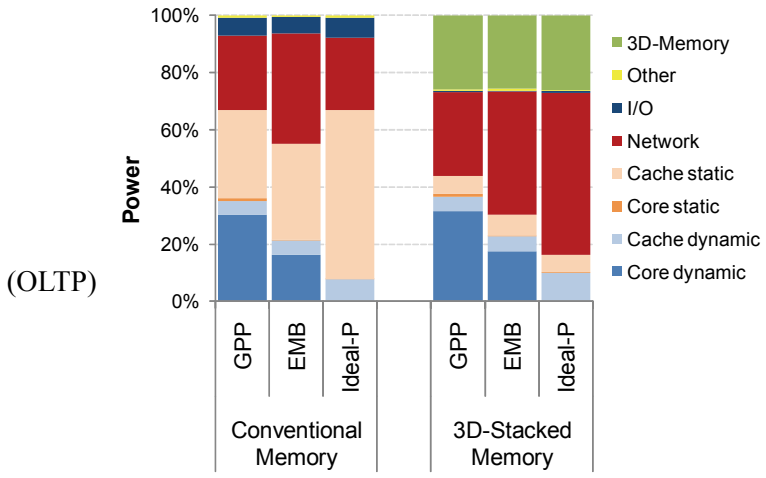
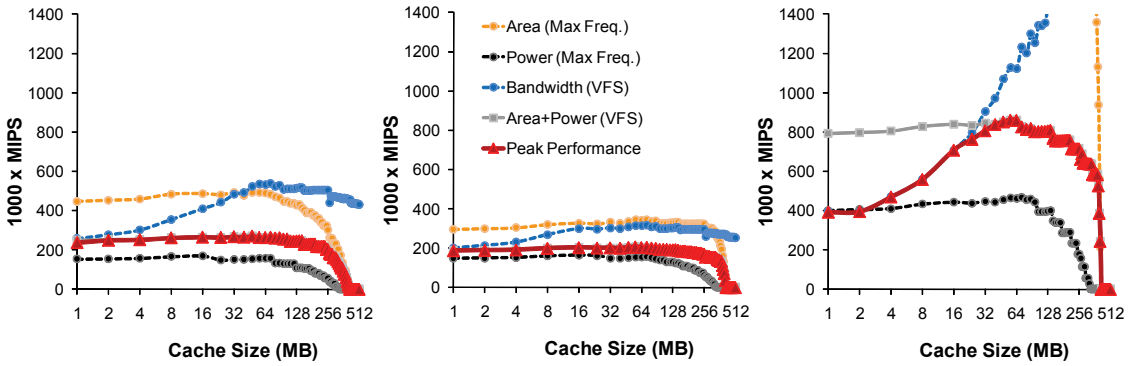


FIGURE 39: Power breakdown of conventional and 3D-memory CMPs at 20nm.

able to achieve such high returns, the inclusion of reconfigurable logic on chip and spatial computing [16] may approximate these assumptions in some cases. Thus, we consider our analysis the first step towards a feasibility study, rather than an accurate performance estimator.

We find that the superior power and performance characteristics of Ideal-P cores pushes the power envelope much further than possible with other core designs (Figure 36). As a result, Ideal-P multicores attain roughly a 2x speedup over the GPP and EMB designs (Figure 38). While previous designs are ultimately power-limited, Ideal-P designs are constrained mostly by off-chip bandwidth. The bandwidth limit pushes them towards designs with hundreds of megabytes of on-chip cache, inescapably leading to cache leakage dominating the power budget.

The superior single-core performance of Ideal-P, along with the limitations imposed by Amdahl's Law on massive parallelism, allows even small-scale CMPs to achieve higher performance than GPP or EMB-based designs with four times more cores. In fact, our results indicate that while almost a thousand cores can fit in a 20nm chip, the optimal (bandwidth-limited) designs are at 16 to 32 cores. The low core count may free a significant amount of die real estate to be used by heterogeneous CMPs for more cores that optimize for a wider spectrum of applications.



General Purpose (GPP) Embedded (EMB) Ideal (Ideal-P)

FIGURE 40: GPP, EMB and Ideal-P CMPs at 20nm with 3D-memory (OLTP).

While all designs we have looked so far are limited by bandwidth and power, the two constraints are not independent but are affecting each other. Bandwidth considerations result in designs with large caches, which consume power otherwise available to add more cores or allow for faster ones. There are recent research proposals, however, that promise to alleviate the bandwidth wall.

4.3.3 CMPs with 3D-Stacked Memory

Advances in fabrication technology have resulted in techniques that enable stacking multiple chip substrates on top of each other [69]. Communication between the substrates is performed through vertical buses which can deliver terabytes per second of bandwidth [69]. While stacking multiple processor chips may have prohibitive thermal implications, stacking memory on top of processing cores results only in a small increase in temperature (10 ^{o}C for 9 additional layers [69]) while it offers an unprecedented bandwidth to memory arrays. The resulting 3D-stacked memory can be used as a large “in-package” cache, that can ease the burden imposed by the cores on the off-chip pins.

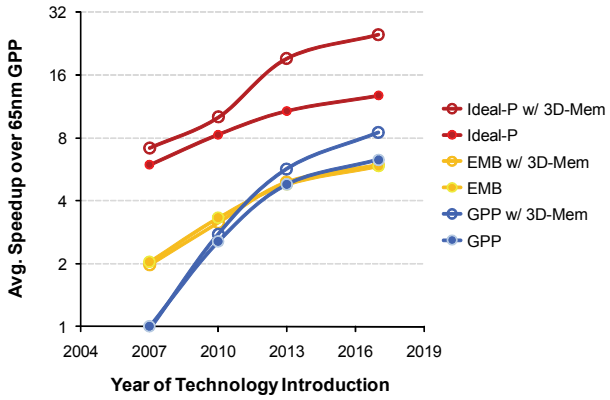
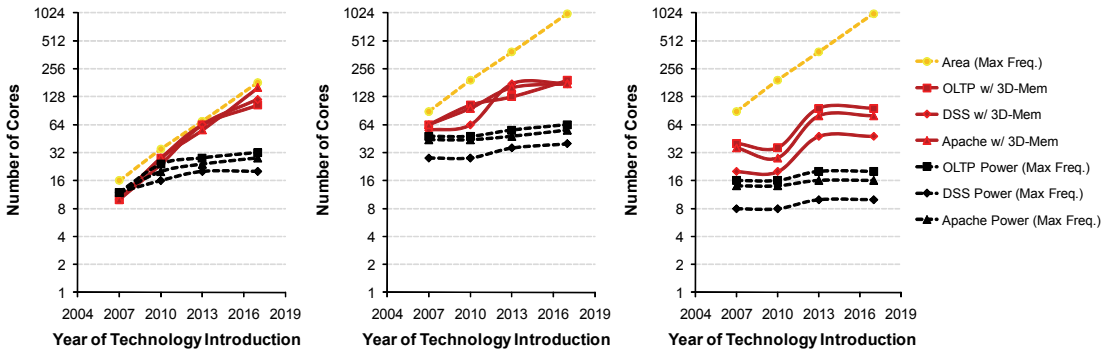


FIGURE 41: Speedup of CMPs with conventional and 3D-stacked memory.

We evaluate 3D-stacked CMP designs across technologies, assuming the stacked memory is used as a cache. We find that 3D-stacked memory pushes the bandwidth constraint beyond the power constraint in most cases (Figure 40). This leads to peak-performance designs that are only power-constrained and achieve higher performance than their conventional-memory counterparts (Figure 41 shows the speedup of each design averaged over all our workloads).

While 3D-memory delivers a relatively small performance improvement in GPP or EMB multicore processors (less than 35%), it results in almost 2x speedup when employed on a CMP with Ideal-P cores. By pushing the bandwidth constraint far out and lowering the memory latency by 32.5%, caches do not need to be so large anymore. In fact, if we assume perfectly scalable applications, caches are kept at less than 16MB for 20nm designs, while most of the chip is populated by cores. However, Amdahl's Law results in diminishing returns from high core counts, and as a result our peak-performance designs employ fewer cores than allowed by linear scaling, and use the remaining area and power for cache. Caches in multicore designs with 3D-stacked memory are typically between 15% to 23% the size of the corresponding designs with conventional memory at 20nm, which still amounts to a respectable 48-56MB for peak-performance designs.



General Purpose (GPP) Embedded (EMB) Ideal (Ideal-P)

FIGURE 42: Core counts for CMPs with 3D-stacked memory.

As the addition of 3D-stacked memory alleviates the bandwidth wall for most memory accesses, it allows for more cores on chip (about two to three times more for our designs at 20nm—Figure 42). The high core counts along with smaller caches lead to designs where the network subsystem dominates the power of the chip (Figure 39) and becomes the new bottleneck.

4.4 Summary

Overall, we find that as technology moves forward, the desire for performance leads to multicore designs with a relatively large amount of cores. Yet, Amdahl's Law and physical constraints restrict the number of cores that can be practically employed. Utilizing faster and more efficient ones (e.g., Ideal-P) results in the ability to provide high performance with a smaller core count, rescuing the chip from Amdahl's Law and delivering superior performance at lower power. At the same time, a large area of the chip is left unused (almost half the chip at 20nm with 3D-stacked memory) which could be used to implement customized cores and power them only when necessary. As such, moving towards heterogeneous computing where only parts of the chip operate at any time holds promise in delivering high performance at low power.

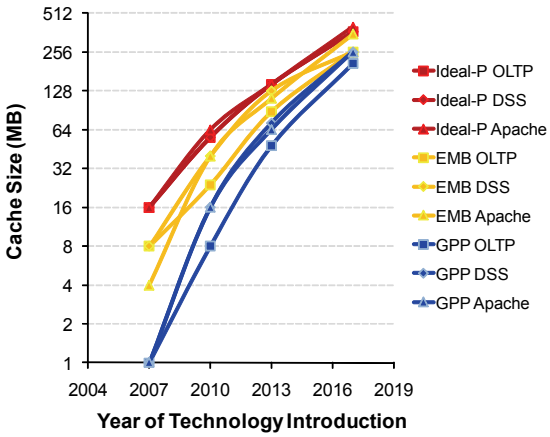


FIGURE 43: On-chip cache sizes for CMPs with conventional memory.

Realizing this goal requires techniques that push the power and bandwidth walls further out. Implementing cores with low-operational-power (LOP) devices can mitigate core leakage, which can be significant as core counts increase. To tame dynamic power, the cores will have to run at low speeds which are well within the capabilities of LOP devices, thereby allowing designers to use them for time-critical components contrary to conventional wisdom and without performance loss. The bandwidth wall can be mitigated through techniques like 3D-die stacking, allowing tera-bytes-per-second access to a multi-gigabyte memory array.

The bandwidth limitations, coupled with the exponential increase in the applications' datas-ets, results in caches growing at an exponential rate (Figure 43). Thus, new multicore chips will require techniques to provide fast access and manage data placement on enormous physically-dis-tributed on-chip caches (e.g., R-NUCA). Finally, as the core counts and caches grow, and the per-formance of individual cores is likely to increase, the power dissipated by the on-chip interconnect and the cache dominates. Thus, further innovation is required in providing feather-weight on-chip interconnects and storage.

Chapter 5

Related Work

Barroso *et al*. [11] conduct a performance study of OLTP and DSS workloads on a distributed shared memory multiprocessor and conclude that instruction and data locality on OLTP can be captured effectively by large caches. Our study shows that data stalls dominate execution on chip multiprocessors even with very large caches, because the dominant stall component then shifts to L2 hits. Barroso *et al*. [10] compare the performance of the Piranha chip multiprocessor against a single out-of-order processor with similar resources. However, they do not consider FC cores for the Piranha chip, and lack the key feature of fine-grain multi-threading to mask memory latency inside the processor core, which is critical to achieve high aggregate CMP performance.

Ranganathan *et al*. [79] examine the performance of database workloads on shared-memory multiprocessors and identify simple optimizations that improve performance when employed by aggressive out-of-order processors. However this study does not consider lean cores, which outperform their aggressive out-of-order counterparts on saturated workloads despite their low single-thread performance. Lo *et al*. [68] study the performance of database workloads on simultaneous multithreaded (SMT) processors and show that aggressive wide-issue out-of-order SMT processors can outperform their single-threaded counterparts. However, wide-issue out-of-order processors with many hardware contexts are complex designs, and their area and power overhead render them unsuitable for CMPs that target database workloads. In our study we show that even simple

in-order multithreaded processors can outperform aggressive out-of-order ones when the workload exhibits significant thread-level parallelism.

To mitigate the access latency of large on-chip caches, Kim proposed Non-Uniform Cache Architectures (NUCA) [60], showing that a network of cache banks can be used to reduce average access latency. Chishti proposed to decouple physical placement from logical organization [22] to add flexibility to the NUCA design. R-NUCA exploits both proposals. The work by Chishti *et al.* on CMP-NuRAPID [23] is the closest to our design. The authors advocate migration of private blocks, replication for shared read-only blocks and in-situ communication for shared read-write blocks, but they rely on complicated coherence protocol modifications and hardware indirection structures to guarantee coherence and perform block placement. In contrast, R-NUCA obviates the need for hardware coherence mechanisms at the last-level cache, thereby eliminating indirection upon cache lookup, while it collaborates with the operating system to make placement decisions.

Beckmann evaluated NUCA architectures in the context of CMPs [13], concluding that dynamic migration of blocks within a NUCA can benefit performance but requires smart lookup algorithms and may cause contention in the physical center of the cache. Kandemir proposed migration algorithms for the placement of each cache block [58], and Ricci proposed smart lookup mechanisms using Bloom filters [80]. In contrast to these works, R-NUCA avoids block migration in favor of intelligent block placement, avoiding the central contention problem and eliminating the need for complex lookup algorithms.

Zhang observed that different classes of accesses benefit from either a private or shared system organization [108] in multi-chip multi-processors. Falsafi proposed to apply either a private or shared organization by dynamically adapting the system on a page granularity [33]. R-NUCA sim-

ilarly applies either a private or shared organization at page granularity, however, we leverage the OS to properly classify the pages, avoiding reliance on heuristics.

Huh extended the NUCA work to CMPs [51], investigating the effect of sharing policies. Yeh [105] and Merino [76] proposed coarse-grain approaches of splitting the cache into private and shared slices. Guz [39] advocated building separate but exclusive shared and private regions of cache. R-NUCA similarly treats data blocks as private until accesses from multiple cores are detected. Finer-grained dynamic partitioning approaches have also been investigated. Dybdahl proposed a dynamic algorithm to partition the cache into private and shared regions [32], while Zhao proposed partitioning by dedicating some cache ways to private operation [109]. R-NUCA enables dynamic and simultaneous shared and private organizations, however, unlike prior proposals, without modification of the underlying cache architecture and without enforcing strict constraints on either the private or shared capacity.

Chang proposed a private organization which steals capacity from neighboring private slices, relying on a centralized structure to keep track of sharing. Liu used bits from the requesting-core ID to select the set of L2 slices to probe first [67], using a table-based mechanism to perform a mapping between the core ID and cache slices. R-NUCA applies a mapping based on the requesting-core ID, however this mapping is performed through boolean operations on the ID without an indirection mechanism. Additionally, prior approaches generally advocate performing lookup through multiple serial or parallel probes or indirection through a directory structure; R-NUCA is able to perform exactly one probe to one cache slice to look up any block or to detect a cache miss.

Zhang advocated the use of a tiled architecture, coupling cache slices to processing cores [107]. Starting with a shared substrate, [107] creates local replicas to reduce access latency, requir-

ing a directory structure to keep track of the replicas. As proposed, [107] wastes capacity because locally allocated private blocks are duplicated at the home node, and offers minimal benefit to workloads with a large shared read-write working set which do not benefit from replication. R-NUCA assumes a tiled architecture with a shared cache substrate, but avoids the need for a directory mechanism by only replicating blocks known to be read-only. Zhang improves on the design of [107] by migrating private blocks to avoid wasting capacity at the home node [106], however this design still can not benefit shared data blocks.

Beckmann proposed an adaptive design that dynamically adjusts the probability by which read-only shared blocks are allocated at the local slice [12]. Unlike [12], R-NUCA is not limited to replicating blocks to a single cache slice, allowing for clusters of nearby slices to share capacity for replication. Furthermore, the heuristics employed in [12] require fine-tuning and adjustment, being highly sensitive to the underlying architecture and workloads, whereas R-NUCA offers a direct ability to smoothly trade off replicated capacity for access latency. Marty studied the benefits of partitioning a cache for multiple simultaneously-executing workloads [74] and proposed a hierarchical structure to simplify handling of coherence between the workloads. The R-NUCA design can be similarly applied to achieve run-time partitioning of the cache while still preserving the R-NUCA access latency benefits within each partition.

OS-driven cache placement has been studied in a number of contexts. Sherwood proposed to guide cache placement in software [89], suggesting the use of the TLB to map addresses to cache regions. Tam used similar techniques to reduce destructive interference for multi-programmed workloads [96]. Jin advocated the use of the OS to control cache placement in a shared NUCA cache, suggesting that limited replication is possible through this approach [54]. Cho used the same placement mechanism to partition the cache slices into groups [24]. R-NUCA leverages

the work of [54] and [24], using the OS-driven approach to guide placement in the cache. Unlike prior proposals, R-NUCA enables the dynamic creation of overlapping clusters of slices without additional hardware, and enables the use of these clusters for dedicated private, replicated private, and shared operation. Fensch advocates the use of OS-driven placement to avoid the cache-coherence mechanism [35]. R-NUCA similarly uses OS-driven placement to avoid cache-coherence at the L2, however, R-NUCA does so without placing strict capacity limitations on the replication of read-only blocks or requiring new hardware structures and TLB ports.

Hartstein *et al.* [45] evaluate the nature of cache misses for a variety of workloads and validate the square-root rule-of-thumb for cache misses. Rogers *et al* [82] extend this work to CMPs and conclude that miss rates follow a simple power law. In this work, through robust fitting of hundreds of candidate functions, we wind that x-shifted power laws describe accurately the cache miss behavior of commercial server workloads, while simpler power laws may generate relative errors in excess of 50%. Hill and Marty [46] explore analytically how different levels of software parallelism and core asymmetry affect the performance of multicore processors, while our model focuses on the trade-offs between physical constraints and performance.

Rogers *et al* [82] model how die area allocation to cores and caches affect the on-chip memory traffic in current as well as future technology generations, and conclude that bandwidth is the main performance constraint. However, the authors don't consider the power implications on performance, and base their models on the assumption that modern multicores are already bandwidth constrained which contradicts prior research [11]. While we agree with their observation that 3D-stacked memory does not alleviate the bandwidth wall when a single layer is considered, we find that the addition of multiple layers of dense DRAM arrays [69] effectively mitigates the bandwidth wall for all designs across technologies.

A recent study that explored the design space of CMPs [30] focuses on throughput as the primary performance metric to compare server workload performance across chip multiprocessors with varying processor granularity, but stops short of a detailed performance characterization and breakdown of where time is spent during execution, and fails to address all the tunable parameters. Similarly, Huh *et al.* [50] explore the design space of CMPs to determine the best configuration and extrapolate SPEC results to server workloads, but do not consider the power implications of CMP designs, and their studies focus on smaller systems where bandwidth and power and less critical. Moreover, the performance model in [50] employs private L2 caches per core, which greatly increase the data sharing overhead and off-chip miss rate.

Li *et al*. [66] present a comprehensive study of the CMP design space subject to physical constraints and jointly optimize across a large number of design parameters. However, they investigate only SPEC benchmarks and a single technology node (65nm), while we focus on commercial server workloads across technologies. Moreover, [66] assumes that cache latency remains constant when scaling the cache size, which does not allow the accurate exploration of a wide range of cache sizes.

Alameldeen [3] studies how compression improves processor performance, and develops an analytic model to balance cores, caches and communication. In contrast, we explore how physical constraints determine the configuration of CMPs across technologies and evaluate different devices, core technologies and 3d-stacked memory to extend the constraints. Kumar *et al*. [62] present a performance evaluation of a heterogeneous CMP, but focuses on a CPU-intensive diverse workload instead of homogeneous commercial server workloads that are the target of our study.

Chapter 6

Future Work

As cache sizes grow to accommodate more cores and mitigate the bandwidth wall, on-chip accesses dominate the execution time. While our work on R-NUCA is a step forward in mitigating the increasing on-chip block access latency, our design targets commercial server workloads. To be effective for all types of applications, R-NUCA needs to optimize access patterns prevalent in other application domains as well. Multi-programmed workloads can be further optimized by applying Rotational Interleaving on the private data, thereby easing the pressure to the private cache by distributing blocks to neighbors. Ideally, the operating system employs a mechanism that estimates the demands of each application and spreads them on the multicore such that applications with high storage demands are far apart. The same mechanism can guide the size of the Rotational Interleaving clusters, expanding them or contracting them as necessary in response to changing demands.

Similarly, non-universal sharing patterns, like producer-consumer or single-writer-multiple-readers may benefit from a different placement of the data. While the current incarnation of R-NUCA interleaves shared data across the entire chip, producer-consumer patterns may benefit from declaring these data private to the consumer. The producer's write latency can potentially be hidden by the store buffer, thus it is less critical than the reads issued by the consumer, which may cause the processor to stall waiting for the data to become available. At the same time, the abstrac-

tion of Rotational Interleaving offers the ability to share data between any subset of cores in the system, by properly defining the corresponding functions. For example if four neighboring cores want to share data that are private to their core group, these data can be treated similar to instructions. If these cores are not neighbors but spread across the entire chip, R-NUCA can still operate on a virtual cluster, as if the cores are neighbors, and in the last step map the virtual destination core to the appropriate physical one.

Mechanisms to make R-NUCA adaptive should also be considered. While for our workloads the behavior of the blocks at steady state does not change over time, this is generally not the case in other applications that go through phases with different computational and storage demands. Aging mechanisms or user-level library calls to "forget" classifications and force R-NUCA to re-learn the class of a page may prove useful in this context.

As we move into the deep nanometer regime, however, other problems may become equally pressing. Amdahl's Law prevents most applications from utilizing an abundance of cores, simply because the applications become serial for even a very small fraction of their execution. Heterogeneous multicores may provide significant relief, by optimizing not only the parallel execution of the application but also speeding up the serial one. Our design space exploration has only scratched the surface of this area, and further detailed analysis is required to evaluate its benefits.

Another way to fight back is to rethink the software stack, and restructure applications with both parallelism and locality in mind. This has been an on-going theme in our group, where we develop our ideas in the context of a database management server as a proof-of-concept. Our prototype staged database engine [43,44] splits conventional single-threaded requests into many smaller tasks that can execute in parallel and utilize otherwise idle hardware resources. These

smaller tasks form producer-consumer pairs and exploit pipelining and operator-level parallelism [44] to reuse data quickly and reduce overall execution time.

To mitigate the rising cache latency, the application's data can be partitioned logically across the cores, while incoming request are scheduled on the cores based on what data they access. Such a software system can transform data that were previously shared across requests and slow to access, into core-private data with fast local access times. Thereby, this software architecture complements R-NUCA by optimizing for private data that can be efficiently mapped close to each core. At the same time, it renders data sharing patterns within a parallel request predictable, because the communication between tasks is explicit and the access and sharing patterns are known in advance and can be exposed to the execution system. Thus, no complicated hardware prediction mechanisms are required to re-discover the patterns [21,78,102], overcoming a significant obstacle of conventional server software [94] and enabling simple hardware streaming engines to hide data access latencies.

Finally, more work is needed on techniques to alleviate the bandwidth and power walls. While 3D-stacked memory promises to deliver high bandwidth to a portion of the memory, on-chip interconnects and cache leakage consume the majority of the chip's power and prevent it from reaching its full potential. New techniques are required to either ease the pressure on these two structures, or reduce their power requirements.

Chapter 7

Conclusions

As Moore's Law continues for at least another decade, the number of cores on chip and the cache size will continue to grow at an exponential rate. While in the last decade off-chip accesses were the primary determinant of performance for commercial server workloads, the increasing latency of the on-chip cache results in cache accesses dominating the execution time. To tame the on-chip cache latency, caches become distributed and accesses to cache blocks become a function of the block's physical location. Now, the block's placement on-chip determines performance.

We observe that cache accesses in server workloads can be classified at run time into classes that exhibit distinct characteristics, leading to different on-chip cache block placement policies. Based on this observation, we propose Reactive NUCA, a low-overhead, low-latency mechanism for block placement in distributed caches. R-NUCA improves performance by allocating cache blocks close to the cores that access them, replicating or migrating them as necessary without the overhead of a hardware coherence mechanism.

Alleviating the cache access bottleneck allows multicore processors to continue realizing improvements in performance. However, increasing core counts does not directly translate into performance improvements, as chips are physically constrained in area, power and bandwidth. We explore the design space of physically-constrained CMPs across technologies in search of peak-performance designs, and find that heterogeneous multicores is a viable alternative that holds great

promise in optimizing the multicore architecture. Contrary to conventional wisdom, we find that low-operational-power devices can be used for time-critical components without loss in performance (but with significant reduction in power), while 3D-stacked memory shows promise in alleviating the bandwidth wall.

Bibliography

[1] A. Ailamaki, D. J. DeWitt, M. D. Hill, and M. Skounakis. Weaving relations for cache performance. In *Proceedings of VLDB*, 2001.

[2] A. Ailamaki, D. J. DeWitt, M. D. Hill, and D. A. Wood. DBMSs on a modern processor: Where does time go? In *The VLDB Journal*, pages 266–277, September 1999.

[3] A. R. Alameldeen. *Using compression to improve chip multiprocessor performance*. PhD thesis, University of Wisconsin at Madison, Madison, WI, USA, 2006.

[4] ARM. ARM MPCore. http://www.arm.com/products/CPUs/ARM11MPCoreMultiprocessor.html.

[5] ARM. ARM1176JZ(F)-S: enhanced security and lower energy consumption for consumer and wireless applications. http://www.arm.com/products/CPUs/ARM1176.html.

[6] M. Azimi, N. Cherukuri, D. N. Jayasimha, A. Kumar, P. Kundu, S. Park, I. Schoinas, and A. S. Vaidya. Integration challenges and tradeoffs for tera-scale architectures. *Intel Technology Journal*, August 2007.

[7] J. Balfour and W. J. Dally. Design tradeoffs for tiled CMP on-chip networks. In *ICS'06: Proceedings of the 20th Annual International Conference on Supercomputing*, 2006.

[8] L. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese. Piranha: A scalable architecture base on single-chip multiprocessing. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, June 2000.

[9] L. A. Barroso, K. Gharachorloo, and E. Bugnion. Memory system characterization of commercial workloads. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, pages 3–14, June 1998.

[10] B. M. Beckmann, M. R. Marty, and D. A. Wood. ASR: Adaptive selective replication for CMP caches. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 39)*, pages 443–454, 2006.

[11] B. M. Beckmann and D. A. Wood. Managing wire delay in large chip-multiprocessor caches. In *Proceedings of the 37th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 37)*, pages 319–330, 2004.

[12] S. Borkar. Microarchitecture and design challenges for gigascale integration. In *MICRO 37: Proceedings of the 37th Annual IEEE/ACM International Symposium on Microarchitecture*, 2004.

[13] K. Brill. The invisible crisis in the data center: The economic meltdown of moore's law. *white paper, Uptime Institute*, 2007.

[14] M. Budiu, G. Venkataramani, T. Chelcea, and S. C. Goldstein. Spatial computation. In *ASPLOS-XI: Proceedings of the 11th international conference on Architectural support for programming languages and operating systems*, pages 14–26, 2004.

[15] T. Burd, T. Pering, A. Stratakos, and R. Brodersen. A dynamic voltage scaled microprocessor system. In *2000 IEEE International Solid-State Circuits Conference, 2000. Digest of Technical Papers. ISSCC*, pages 294–295, 2000.

[16] L. Ceze, J. Tuck, P. Montesinos, and J. Torrellas. Bulk enforcement of sequential consistency. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, 2007.

[17] J. Chang and G. S. Sohi. Cooperative caching for chip multiprocessors. In *Proceedings of the 33rd Annual International Symposium on Computer Architecture*, pages 264–276, 2006.

[18] G. Chen, H. Chen, M. Haurylau, N. Nelson, P. M. Fauchet, E. G. Friedman, and D. H. Albonesi. Electrical and optical on-chip interconnects in scaled microprocessors. In *IEEE International Symposium on Circuits and Systems*, pages 2514–2517, 2005.

[19] T. M. Chilimbi and M. Hirzel. Dynamic hot data stream prefetching for general-purpose programs. In *Proceedings of the SIGPLAN'02 Conference on Programming Language Design and Implementation (PLDI)*, June 2002.

[20] Z. Chishti, M. D. Powell, and T. N. Vijaykumar. Distance associativity for high-performance energy-efficient non-uniform cache architectures. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 36)*, 2003.

[21] Z. Chishti, M. D. Powell, and T. N. Vijaykumar. Optimizing replication, communication, and capacity allocation in CMPs. In *Proceedings of the 32nd Annual International Symposium on Computer Architecture*, pages 357–368, 2005.

[22] S. Cho and L. Jin. Managing distributed, shared L2 caches through OS-level page allocation. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 39)*, pages 455–468, 2006.

[23] Y. Chou, L. Spracklen, and S. G. Abraham. Store memory-level parallelism optimizations for commercial applications. In *Proceedings of the 38th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 38)*, pages 183–196, 2005.

[24] Concord Idea Corp. Elpida SyncMAX press conference. http://www.syncmax.com/doc/Press_Conference_Sept15_2005_ver0_95H.pdf, September 2005.

[25] W. J. Dally and C. L. Seitz. The torus routing chip. *Distributed Computing*, 1(4):187–196, 1986.

[26] J. D. Davis, J. Laudon, and K. Olukotun. Maximizing CMP throughput with mediocre cores. In *Proceedings of the Thirteenth International Conference on Parallel Architectures and Compilation Techniques*, pages 51–62, 2005.

[27] J. Deng, K. Kim, C.-T. Chuang, and H.-S. P. Wong. Device footprint scaling for ultra thin body fully depleted SOI. In *ISQED '07: Proceedings of the 8th International Symposium on Quality Electronic Design*, pages 145–152, 2007.

[28] H. Dybdahl and P. Stenstrom. An adaptive shared/private NUCA cache partitioning scheme for chip multiprocessors. In *Proceedings of the Thirteenth IEEE Symposium on High-Performance Computer Architecture*, pages 2–12, 2007.

[29] B. Falsafi and D. A. Wood. Reactive NUMA: A design for unifying S-COMA and CC-NUMA. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, pages 229–240, June 1997.

[30] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *ISCA '07: Proceedings of the 34th annual international symposium on Computer architecture*, pages 13–23, 2007.

[31] C. Fensch and M. Cintra. An OS-based alternative to full hardware coherence on tiled CMPs. In *Proceedings of the 14th IEEE Symposium on High-Performance Computer Architecture*, 2008.

[32] Gartner, Inc. Gartner says worldwide relational database market increased 8 percent in 2005, press release. http://www.gartner.com/press_releases/asset_152619_11.html, May 2006.

[33] Gartner, Inc. Gartner says worldwide server shipments grew 9 percent, while industry revenue grew 4 percent in third quarter of 2006, press release. http://gartner.com/it/page.jsp?id=498468, November 2006.

[34] K. Gharachorloo, A. Gupta, and J. Hennessy. Two techniques to enhance the performance of memory consistency models. In *Proceedings of the 1991 International Conference on Parallel Processing (Vol. I Architecture)*, pages I–355–364, August 1991.

[35] Z. Guz, I. Keidar, A. Kolodny, and U. C. Weiser. Utilizing shared data in chip multiprocessors with the Nahalal architecture. In *Proceedings of the 20th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 1–10, 2008.

[36] N. Hardavellas, M. Ferdman, B. Falsafi, and A. 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, 2009.

[37] N. Hardavellas, I. Pandis, R. Johnson, N. Mancheril, A. Ailamaki, and B. Falsafi. Database servers on chip multiprocessors: Limitations and opportunities. In *3rd Biennial Conference on Innovative Data Systems Research*, pages 79–87, 2007.

[38] N. Hardavellas, S. Somogyi, T. F. Wenisch, R. E. Wunderlich, S. Chen, J. Kim, B. Falsafi, J. C. Hoe, and A. G. Nowatzyk. SimFlex: a fast, accurate, flexible full-system simulation framework for performance evaluation of server architecture. *SIGMETRICS Performance Evaluation Review*, 31(4):31–35, April 2004.

[39] S. Harizopoulos and A. Ailamaki. A case for staged database systems. In *Proceedings of the 1st Biennial Conference on Innovative Data Systems Research (CIDR '03)*, January 2003.

[40] S. Harizopoulos, V. Shkapenyuk, and A. Ailamaki. Qpipe: A simultaneously pipelined relational query engine. In *Proceedings of the 24th Annual ACM International Conference on Management of Data (SIGMOD '05)*, June 2005.

[41] A. Hartstein, V. Srinivasan, T. R. Puzak, and P. G. Emma. Cache miss behavior: is it sqrt(2)? In *CF '06: Proceedings of the 3rd conference on Computing frontiers*, pages 313–320, 2006.

[42] M. D. Hill and M. R. Marty. Amdahl's law in the multicore era. *Computer*, 41(7):33–38, 2008.

[43] K. Hirata and J. Goodacre. ARM MPCore: The streamlined and scalable arm11 processor core. In *ASP-DAC '07: Proceedings of the 2007 Asia and South Pacific Design Automation Conference*, pages 747–748, 2007.

[44] Z. Hu, M. Martonosi, and S. Kaxiras. Tcp: Tag correlating prefetchers. In *Proceedings of the Ninth IEEE Symposium on High-Performance Computer Architecture*, 2003.

[45] H. Hua, C. Mineo, K. Schoienfliess, A. Sule, S. Melamed, and W. Davis. Performance trend in three-dimensional integrated circuits. In *Interconnect Technology Conference, 2006 International*, pages 45–47, 2006.

[46] J. Huh, D. Burger, and S. W. Keckler. Exploring the design space of future CMPs. In *Proceedings of the Ninth International Conference on Parallel Architectures and Compilation Techniques*, pages 199–210, 2001.

[47] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, and S. W. Keckler. A NUCA substrate for flexible CMP cache sharing. In *Proceedings of the 19th Annual International Conference on Supercomputing*, pages 31–40, 2005.

[48] IBM Corp. Pmcount for linux on power architecture. http://www.alphaworks.ibm.com/tech/pmcount, 2006.

[49] Intel. Intel core duo processor and intel core solo processor on 65nm process datasheet, 2006.

[50] Intel. Intel fact sheet: Intel previews intel xeon nehalem-EX processor, press release. http://www.intel.com/pressroom/archive/releases/20090526comp.htm?iid=pr1_rel%easepri_20090526m, May 2009.

[51] L. Jin, H. Lee, and S. Cho. A flexible data to L2 cache mapping approach for future multicore processors. In *Proceedings of the 2006 Workshop on Memory System Performance and Correctness (MSPC'06)*, pages 92–101, 2006.

[52] D. Joseph and D. Grunwald. Prefetching using Markov Predictors. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, pages 252–263, June 1997.

[53] N. P. Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 364–373, May 1990.

[54] R. Kalla, B. Sinharoy, and J. M. Tendler. IBM power5 chip: A dual-core multithreaded processor. *IEEE Micro*, 24(2):40–47, 2004.

[55] M. Kandemir, F. Li, M. J. Irwin, and S. W. Son. A novel migration-based NUCA design for chip multiprocessors. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–12, 2008.

[56] S. Kaxiras and J. R. Goodman. Improving cc-numa performance using instruction-based prediction. In *Proceedings of the Fifth IEEE Symposium on High-Performance Computer Architecture*, pages 161–170, February 1999.

[57] C. Kim, D. Burger, and S. W. Keckler. An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches. *ACM SIGPLAN Not.*, 37(10):211–222, 2002.

[58] P. Kongetira, K. Aingaran, and K. Olukotun. Niagara: A 32-way multithreaded SPARC processor. *IEEE Micro*, 25(2):21–29, March-April 2005.

[59] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas. Single-ISA heterogeneous multi-core architectures for multithreaded workload performance. In *ISCA '04: Proceedings of the 31st annual international symposium on Computer architecture*, 2004.

[60] J. Larus. Spending moore's dividend. *Communications of the ACM*, 52(5):62–69, 2009.

[61] A. S. Leon, K. W. Tam, J. L. Shin, D. Weisner, and F. Schumacher. A power-efficient high-throughput 32-thread SPARC processor. *IEEE Journal of Solid-state circuits*, 42(1):7–16, 2007.

[62] B. Li, L.-S. Peh, and P. Patra. Impact of process and temperature variations on network-on-chip design exploration. In *NOCS '08: Proceedings of the Second ACM/IEEE International Symposium on Networks-on-Chip*, pages 117–126, 2008.

[63] Y. Li, B. Lee, D. Brooks, Z. Hu, and K. Skadron. CMP design space exploration subject to physical constraints. In *The 12th International Symposium on High-Performance Computer Architecture*, pages 17–28, 2006.

[64] C. Liu, A. Sivasubramaniam, and M. Kandemir. Organizing the last line of defense before hitting the memory wall for CMPs. In *Proceedings of the Tenth IEEE Symposium on High-Performance Computer Architecture*, page 176, 2004.

[65] J. L. Lo, L. A. Barroso, S. J. Eggers, K. Gharachorloo, H. M. Levy, and S. S. Parekh. An analysis of database workload performance on simultaneous multithreaded processors. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, pages 39–50, 1998.

[66] G. H. Loh. 3D-stacked memory architectures for multi-core processors. In *ISCA '08: Proceedings of the 35th International Symposium on Computer Architecture*, 2008.

[67] C.-K. Luk and T. C. Mowry. Compiler based prefetching for recursive data structures. In *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS VII)*, pages 222–233, October 1996.

[68] C.-K. Luk and T. C. Mowry. Automatic compiler-inserted prefetching for pointer-based applications. *IEEE Transactions on Computers*, 48(2), February 1999.

[69] C.-K. Luk and T. C. Mowry. Memory forwarding: Enabling aggressive layout optimizations by guaranteeing the safety of data relocation. In *Proceedings of the 26th Annual International Symposium on Computer Architecture*, 1999.

[70] M. K. Martin, M. D. Hill, and D. A. Wood. Token coherence: Decoupling performance and correctness. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, June 2003.

[71] M. R. Marty and M. D. Hill. Virtual hierarchies to support server consolidation. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, 2007.

[72] A. M. G. Maynard, C. M. Donnelly, and B. R. Olszewski. Contrasting characteristics and cache performance of technical and multi-user commercial workloads. In *ASPLOS-VI: Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 145–156, 1994.

[73] J. Merino, V. Puente, P. Prieto, and J. 'Angel Gregorio. SP-NUCA: a cost effective dynamic non-uniform cache architecture. *ACM SIGARCH Computer Architecture News*, 36(2):64–71, 2008.

[74] N. Muralimanohar, R. Balasubramonian, and N. Jouppi. Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0. In *MICRO '07: Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 3–14, 2007.

[75] K. J. Nesbit and J. E. Smith. Data cache prefetching using a global history buffer. In *Proceedings of the Tenth IEEE Symposium on High-Performance Computer Architecture*, February 2004.

[76] P. Ranganathan, K. Gharachorloo, S. V. Adve, and L. A. Barroso. Performance of database workloads on shared-memory systems with out-of-order processors. In *Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS VIII)*, pages 307–318, October 1998.

[77] R. Ricci, S. Barrus, D. Gebhardt, and R. Balasubramonian. Leveraging bloom filters for smart search within NUCA caches. In *Proceedings of WCED*, June 2006.

[78] S. Rodriguez and B. Jacob. Energy/power breakdown of pipelined nanometer caches (90nm/65nm/45nm/32nm). In *ISLPED '06: Proceedings of the 2006 international symposium on Low power electronics and design*, pages 25–30, 2006.

[79] B. M. Rogers, A. Krishna, G. B. Bell, K. Vu, X. Jiang, and Y. Solihin. Scaling the bandwidth wall: challenges in and avenues for CMP scaling. In *ISCA '09: Proceedings of the 36th annual international symposium on Computer architecture*, pages 371–382, 2009.

[80] A. Roth, A. Moshovos, and G. S. Sohi. Dependence based prefetching for linked data structures. In *Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS VIII)*, October 1998.

[81] A. Roth and G. S. Sohi. Effective jump pointer prefetching for linked data structures. In *Proceedings of the 26th Annual International Symposium on Computer Architecture*, May 1999.

[82] S. Rusu, S. Tam, H. Muljono, D. Ayers, and J. Chang. A dual-core multi-threaded xeon processor with 16MB L3 cache. In *IEEE International Solid-State Circuits Conference, 2006. ISSCC 2006. Digest of Technical Papers*, pages 315–324, 2006.

[83] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, R. Espasa, E. Grochowski, T. Juan, and P. Hanrahan. Larrabee: a many-core x86 architecture for visual computing. *ACM Trans. Graph.*, 27(3):1–15, 2008.

[84] Semiconductor Industry Association. The international technology roadmap for semiconductors (ITRS), process integration, devices, and structures. http://www.itrs.net/, 2002 Update.

[85] Semiconductor Industry Association. The international technology roadmap for semiconductors (ITRS). http://www.itrs.net/, 2007 Update.

[86] Semiconductor Industry Association. The international technology roadmap for semiconductors (ITRS). http://www.itrs.net/, 2008 Edition.

[87] M. Shao, A. Ailamaki, and B. Falsafi. DBmbench: Fast and accurate database workload representation on modern microarchitecture. In *Proceedings of the 15th IBM Center for Advanced Studies Conference*, October 2005.

[88] T. Sherwood, B. Calder, and J. Emer. Reducing cache misses using hardware and software page placement. In *Proceedings of the 13th Annual International Conference on Supercomputing*, pages 155–164, 1999.

[89] T. Sherwood, S. Sair, and B. Calder. Predictor-directed stream buffers. In *Proceedings of the 33rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 33)*, pages 42–53, December 2000.

[90] T. Skotnicki. Materials and device structures for sub-32 nm CMOS nodes. *Microelectronics Engineering*, 84(9-10):1845–1852, 2007.

[91] Y. Solihin, J. Lee, and J. Torrellas. Using a user-level memory thread for correlation prefetching. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, May 2002.

[92] S. Somogyi, T. F. Wenisch, A. Ailamaki, B. Falsafi, and A. Moshovos. Spatial memory streaming. In *Proceedings of the 33rd Annual International Symposium on Computer Architecture*, June 2006.

[93] S. Somogyi, T. F. Wenisch, N. Hardavellas, J. Kim, A. Ailamaki, and B. Falsafi. Memory coherence activity prediction in commercial workloads. In *Proceedings of the Third Workshop on Memory Performance Issues (WMPI-2004)*, June 2004.

[94] Sun Microsystems. UltraSPARC T2 supplement to the UltraSPARC architecture 2007, draft D1.4.3. http://opensparc-t2.sunsource.net/specs/UST2-UASuppl-current-draft-HP-EXT.pdf

September 2007.

[95] D. Tam, R. Azimi, L. Soares, and M. Stumm. Managing shared L2 caches on multicore systems in software. In *Proceedings of the Workshop on the Interaction between Operating Systems and Computer Architecture*, 2007.

[96] S. Thoziyoor, N. Muralimanohar, and N. P. Jouppi. CACTI 5.1. *PHL Technical Report HPL-2008-20*, 2008.

[97] B. Towles and W. J. Dally. Route packets, net wires: On-chip interconnection networks. *Design Automation Conference*, 0:684–689, 2001.

[98] Transaction Processing Performance Council. The TPC Benchmark. *Transaction Processing Performance Council*, 2001.

[99] Z. Wang, D. Burger, K. S. McKinley, S. K. Reinhardt, and C. C. Weems. Guided region prefetching: a cooperative hardware/software approach. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, pages 388–398, June 2003.

[100] T. F. Wenisch, A. Ailamaki, B. Falsafi, and A. Moshovos. Mechanisms for store-wait-free multiprocessors. *ACM SIGARCH Computer Architecture News*, 35(2):266–277, 2007.

[101] T. F. Wenisch, S. Somogyi, N. Hardavellas, J. Kim, A. Ailamaki, and B. Falsafi. Temporal streaming of shared memory. In *Proceedings of the 32nd Annual International Symposium on Computer Architecture*, June 2005.

[102] T. F. Wenisch, S. Somogyi, N. Hardavellas, J. Kim, C. Gniady, A. Ailamaki, and B. Falsafi. Store-ordered streaming of shared memory. In *PACT '05: Proceedings of the 14th International Conference on Parallel Architectures and Compilation Techniques*, pages 75–86, 2005.

[103] T. F. Wenisch, R. E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J. C. Hoe. SimFlex: statistical sampling of computer system simulation. *IEEE Micro*, 26(4):18–31, July-August 2006.

[104] J. Wuu, D. Weiss, C. Morganti, and M. Dreesen. The asynchronous 24MB on-chip level-3 cache for a dual-core itanium-family processor. In *IEEE International Solid-State Circuits Conference Digest of Technical Papers*, 2005.

[105] T. Y. Yeh and G. Reinman. Fast and fair: data-stream quality of service. In *Proceedings of the 2005 International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES '05)*, pages 237–248, 2005.

[106] M. Zhang and K. Asanovic. Victim migration: Dynamically adapting between private and shared CMP caches. Technical report, MIT, 2005.

[107] M. Zhang and K. Asanovic. Victim replication: Maximizing capacity while hiding wire delay in tiled chip multiprocessors. In *Proceedings of the 32nd Annual International Symposium on Computer Architecture*, pages 336–345, 2005.

[108] Z. Zhang and J. Torrellas. Reducing remote conflict misses: Numa with remote cache versus coma. In *Proceedings of the Third IEEE Symposium on High-Performance Computer Architecture*, page 272, 1997.

[109] L. Zhao, R. Iyer, M. Upton, and D. Newell. Towards hybrid last-level caches for chip-multiprocessors. *SIGARCH Computer Architecture News*, 36(2):56–63, 2008.

[110] ZunZun.com. ZunZun.com online curve fitting and surface fitting web site. http://zunzun.com/.