In Proceedings of the 51st International Symposium on Microarchitecture (MICRO'18)

Farewell My Shared LLC!
A Case for Private Die-Stacked DRAM Caches for Servers

Amna Shahab Mingcan Zhu Artemiy Margaritov Boris Grot

Institute for Computing Systems Architecture (ICSA), School of Informatics
University of Edinburgh

***Abstract*—The slowdown in technology scaling mandates rethinking of conventional CPU architectures in a quest for higher performance and new capabilities. This work takes a step in this direction by questioning the value of on-chip shared last-level caches (LLCs) in server processors and argues for a better alternative. Shared LLCs have a number of limitations, including on-chip area constraints that limit storage capacity, long planar interconnect spans that increase access latency, and contention for the shared cache capacity that hurts performance under workload colocation.**

To overcome these limitations, we propose a Die-Stacked Private LLC Organization (SILO), which combines conventional on-chip private L1 (and optionally, L2) caches with a per-core private LLC in die-stacked DRAM. By stacking LLC slices directly above each core, SILO avoids long planar wire spans. The use of private caches inherently avoids inter-core cache contention. Last but not the least, engineering the DRAM for latency affords low access delays while still providing over 100MB of capacity per core in today's technology. Evaluation results show that SILO outperforms state-of-the-art conventional cache architectures on a range of scale-out and traditional workloads while delivering strong performance isolation under colocation.

***Index Terms*—last-level cache, private cache, DRAM, die-stacking, server workloads**

I. Introduction

Datacenters underpin today's digital society by providing real-time storage, retrieval and processing capabilities for increasingly complex information-centric tasks. The servers inside today's datacenters use many-core high-performance server processors to maximize overall throughput and control tail latency in latency-critical services [1]. As the volume of data consumed and created by both human and machine actors continues to grow, it is essential to increase per-server performance to keep up with increasing demand.

Problematically, the looming end of traditional technology scaling presents a challenge for extracting further processor performance. Power constraints have largely flattened the improvement in single-thread performance over the past decade. Meanwhile, the slowdown in Moore's Law combined with skyrocketing manufacturing costs for leading-edge technology nodes [2] spell an approaching end to growth in core count. These trends motivate the need to look beyond traditional chip-multiprocessor (CMP) architectures to mine further performance and efficiency.

This work takes a step in this direction by rethinking the design of cache hierarchies for servers. Today's server processors tend to employ large on-die LLC capacities that attempt to capture the massive data and instruction working sets of server workloads. For instance, recent 18-core Intel [3] and 12-core IBM [4] processors feature 45MB and 96MB LLCs, respectively, which consume 30-35% of the die area (estimated using die micrograph measurements). The massive area footprint proportionately reduces the die area and power available for the cores. In an attempt to shift the balance, recent work has argued for smaller LLC capacities in server processors [5]; however, small-LLC designs are incompatible with workload and VM consolidation, which are a staple feature of today's datacenters. For instance, in 2013, Google was already running multiple workloads per machine, sometimes with dozens of tasks consolidated on a single server [6].

We observe that while large cache capacities are, indeed, useful for servers, existing configurations are not ideal. First, area and power constraints limit the LLC capacities that can be afforded within a yield-effective die size. Secondly, the larger the capacity (and hence, the larger the die size), the more time it takes to access an LLC slice due to slow wires and multi-hop on-chip network topologies. Last but not least, the shared LLC designs in use today create a significant challenge in isolating the co-running workloads, as evidenced by a large body of recent work exploring the issues around LLC contention in multi-core chips [7]–[9]. And while a shared LLC is effective in facilitating low-latency inter-thread data sharing, this capability is not useful for server workloads that are engineered for high scalability and thus have minimal inter-thread data sharing [10].

In this paper, we argue for *private* LLCs in die-stacked DRAM as a preferred alternative to traditional on-chip shared LLC architectures. Die stacking naturally overcomes the area limitations of planar silicon by offering multiple layers of densely-integrated memory cells [11], [12]. Unfortunately, existing die-stacked caches are designed around commodity DRAM technology that favors capacity over latency. As such, existing DRAM cache architectures have high access latencies that are on par with main memory [13], [14], which renders them unsuitable for replacing on-chip caches.

We observe that access latencies to existing DRAM caches are high for two reasons: (1) significant interconnect delays incurred both on the CPU die (when routing to and from the DRAM cache interface) and on the DRAM die when accessing the target bank; (2) use of capacity-oriented DRAM architectures, which favor area efficiency over access latency; Our insight is that neither of these problems is fundamental and can be readily addressed at the architecture level.

We address the wire delay problem through per-core private DRAM caches. In our design, the DRAM cache is partitioned into vertical slices, or *vaults*. Similar to the existing Hybrid Memory Cube (HMC) [15], each vault has its own memory controller and is completely independent of other vaults in data storage and access. Unlike the HMC, which is a discrete memory chip, the vaults in our proposed design are stacked over the CPU die such that each vault sits directly above a core. This organization naturally avoids long on-chip wire delays inherent in shared LLC architectures and provides a low-latency path from the core to its private DRAM cache slice. We address the DRAM latency problem by engineering the vaults for low-latency access, at the expense of capacity, by provisioning a large number of banks, divided into many subarrays and tiles.

The resulting *Die-Stacked Private LLC Organization (SILO)* combines conventional on-chip per-core private L1's (and optionally L2's), with private LLC slices in die-stacked DRAM. The DRAM is optimized for latency, at the expense of capacity, to further reduce the access time to the LLC vaults. The caches are kept coherent through a conventional directory-based protocol with in-DRAM metadata. The high hit-rate of large, private DRAM caches and the use of low-latency DRAM for storing metadata makes directory accesses not detrimental to performance. Meanwhile, usage of private caches naturally eliminates inter-core LLC contention, facilitating workload isolation in a many-core setting.

Using full-system cycle-accurate simulation, we make the following contributions:

- We corroborate prior work showing that while scale-out server workloads benefit from large LLC capacities, they are highly sensitive to LLC access latency. We also show that inter-core data sharing is minimal in these scale-out workloads.
- We introduce Die-Stacked Private LLC Organization (SILO), a chip architecture with all-private caches. SILO avoids high interconnect latencies and overcomes on-chip area constraints by using a die-stacked DRAM LLC, with a private slice directly above each core. The private caches are kept coherent through a conventional coherence protocol with directory metadata embedded in the die-stacked LLC.
- We demonstrate a latency-optimized DRAM cache architecture that lowers the latency by 45% over a capacity-optimized design in 22nm technology node. When used in SILO, the proposed design affords an 11.5ns access latency to a core's private in-DRAM LLC slice with 256MB of capacity.
- We show that on CloudSuite workloads, a 16-core SILO deployment improves performance by 5-54% over a state-of-the-art server baseline that combines a shared LLC with a conventional DRAM cache. We also show that SILO provides strong performance isolation by preserving applications' performance under colocation.

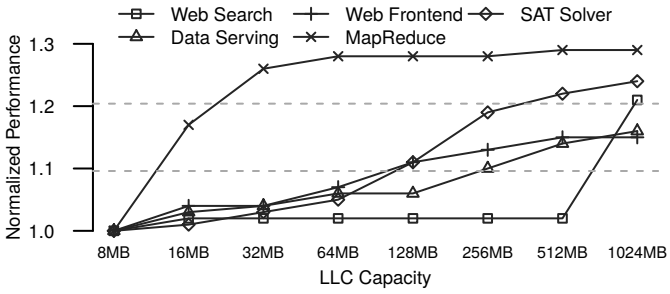


Fig. 1. Sensitivity to LLC capacity at fixed latency.

II. Motivation

We examine representative scale-out server workloads (details in Sec. VI-D to investigate their requirements from an LLC perspective. The goal is to characterize the performance sensitivity of these workloads to LLC capacity, access latency and inter-thread data sharing.

A. Sensitivity to Capacity

To understand the sensitivity of scale-out workloads to higher LLC capacities, we sweep the capacity range at a fixed access latency. We present the results for a 16-core setup, the details of which are available in Sec. VI. The baseline, with 8MB of LLC, is configuring per Scale-out Processors [5], a state-of-the-art specialized server processor architecture targeting scale-out workloads. For larger LLC capacities, the access latency is unchanged from the baseline design.

Fig. 1 plots workload performance with increasing LLC capacity. All data points are normalized to 8MB. We observe that for most workloads, there is marginal performance gain from 8MB to 64MB. This can be attributed to the fact that although the increased capacity can hold some part of the secondary working set, it is not large enough to capture it fully, thus limiting the performance benefit. This finding supports the Scale-out Processors design [5], which advocates a small LLC to minimize access latency and area footprint. Beyond 64MB, however, we observe greater performance benefits as the secondary working set starts fitting into the LLC. For Data Serving, Web Frontend and SAT Solver, the performance gain over the 8MB baseline is 10-20% at 256MB but only 2-6% at 64MB. Web Search differs somewhat, showing little benefit from increased capacity up to 512MB, but then gaining 20% in performance at 1024MB (1GB) as the secondary working set starts to fit.

B. Sensitivity to Latency

We analyze the performance sensitivity of scale-out workloads to the LLC access latency for a range of LLC sizes. Fig. 2 plots the results. To minimize clutter, we show only LLC capacities in the range of 64MB and beyond, as that was the region identified in the previous section as delivering the greatest benefit. For each capacity, we sweep the access latency from the baseline (an 8MB LLC) to twice the baseline latency.

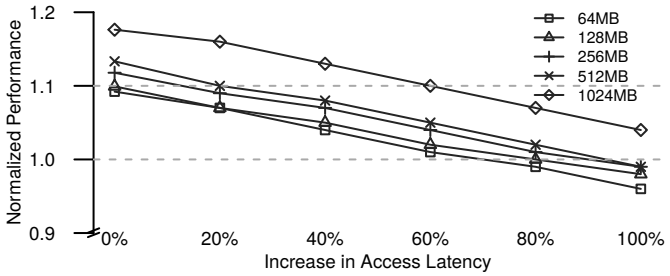


Fig. 2. Performance sensitivity to LLC latency at different capacities normalized to an 8MB baseline. The isocurves show geomean of scale-out workloads.

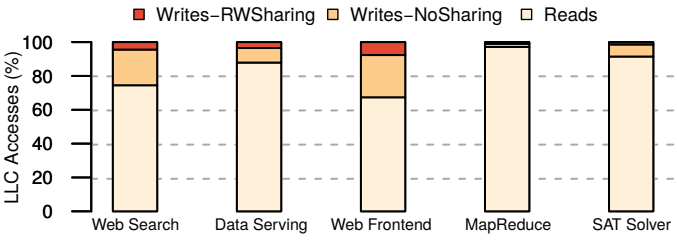


Fig. 3. Breakdown of accessed LLC blocks.

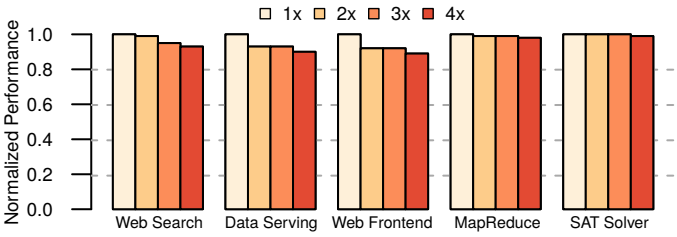


Fig. 4. Performance impact of increased latency to RW-shared blocks. The latency is increased as a multiple of the baseline.

Each line in the figure represents a geomean performance of the scale-out workloads normalized to the baseline for a given LLC capacity.

We observe that larger LLCs translate to higher performance only at lower latencies. The gains from higher capacity rapidly diminish with increased latency. For example, a 1024MB (1GB) LLC with latency 40% higher than the baseline performs only as well as a 64MB LLC at baseline latency and only 10% better than the 8MB LLC. In fact, as the latency approaches twice the baseline (i.e., the 100% point in the figure), most configurations approach or fall below the performance of the 8MB LLC. This result further corroborates Scale-out Processors [5], which showed larger and slower LLCs to be sub-optimal. Server workloads are highly sensitive to LLC access latency because of their low memory-level-parallelism [5], [10], which exposes the latency of an L1 miss to the issuing core. These results show that higher LLC access latencies are detrimental to scale-out workloads even if they are accompanied by a larger LLC capacity.

C. Sensitivity to Read/Write Sharing

Existing server processors deploy shared LLCs that naturally accommodate inter-thread read-write data sharing arising from producer-consumer data exchange or synchronization. Shared LLCs facilitate such sharing patterns by capturing dirty evictions from a writer's private cache and serving subsequent read requests from other cores without any indirection.

We study the access patterns of scale-out server workloads to characterize the extent of read-write sharing (RW-sharing) and the benefit delivered by a shared LLC in accommodating it. For this study, we use an 8MB shared LLC; full system parameters are described in Sec. VI. Results are shown in Fig. 3, which breaks down LLC accesses into three categories: (1) Reads; (2) Writes that see no reads by non-writing cores (Writes-NoSharing), and (3) Writes that see read(s) by at least one core that is not the writer (Writes-RWSharing).

Generally, we observe limited RW-sharing across the scale-out workloads, which matches the findings of Ferdman *et al.* [10]. MapReduce and SAT Solver have negligible RW-sharing. Web Search and Data Serving exhibit little RW-sharing (4% and 3%, respectively) due to the use of a parallel garbage collector that potentially runs a collector thread on a remote core [10]. This can appear as inter-thread communication between application threads.

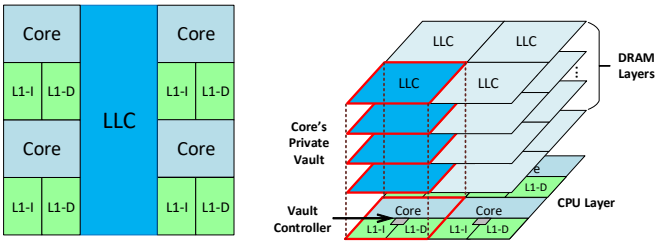
We further evaluate the impact of RW-shared data on system performance. In order to quantify the performance impact, we artificially increase the access latency of RW-shared blocks compared to other blocks by up to a factor of 4x. Fig. 4 presents the results of this experiment. We observe that increasing the access latency of the RW-shared blocks carries a small performance degradation. Doubling the latency of RW-shared blocks results in a performance drop of 0-8%. Even at 4x, the biggest drop observed is for Data Serving and Web Frontend, which lose 10% of performance.

Our conclusion is that the low incidence of true RW-sharing in scale-out workloads makes them largely insensitive to the LLC access latency for RW-shared data. This implies that the value delivered by a shared LLC in accommodating accesses to such data is low for the scale-out workload domain.

D. Summary

Overall, scale-out workloads benefit from large LLC capacities that help capture their vast working sets. However, the performance benefits diminish if larger capacities are accompanied by an increase in access latency. By design, scale-out workloads also have low degrees of inter-core data sharing making them insensitive to the latency of shared data.

Shared LLCs found in today's server processors fail to accommodate these workload characteristics. On-chip area constraints limit the LLC capacity that can be afforded on a die, while planar interconnect delays incur high access latencies to remote cache banks. The shared LLC organization does facilitate low-latency inter-thread data exchange; however, the impact of such sharing is low in scale-out workloads.



(a) Conventional LLC (b) Die-Stacked Private LLC

Fig. 5. Conventional (a) and proposed (b) cache architectures. A 4-core CPU is shown for simplicity.

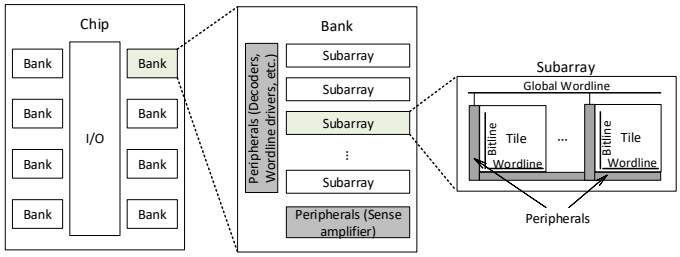


Fig. 6. DRAM internal design.

III. SILO Overview

SILO directly accommodates the needs and characteristics of server workloads through three optimizations that directly address the deficiencies of today's shared LLC designs.

First, SILO completely dispenses with a shared LLC in favor of an all-private cache hierarchy. Per-core private caches overcome the latency bottleneck of shared caches by limiting the length of interconnect that needs to be traversed on an access. Whereas a conventional shared LLC employing a non-uniform cache access (NUCA) organization may require a request to be routed over a large silicon plane to a remote cache bank, a private cache located near a core naturally minimizes the interconnect delay. Another advantage of private caches is that they are naturally immune to cache contention, which is a significant concern for shared LLCs in deployment scenarios involving multiple workloads.

Unfortunately, simply converting a conventional shared LLC into a private design would effectively constrain each core to a capacity of just a few MBs. For instance, the current generation of Intel's mainstream server processors (14nm Broadwell family) features a shared LLC with 2.5MB of capacity per core. In a shared configuration, the aggregate LLC capacity provided by a multi-core Broadwell CPU is sufficient to capture a meaningful portion of a server workload's instruction and data working set; however, in a private configuration, this capacity would be woefully inadequate.

To overcome the area limitations of planar silicon, SILO deploys the second optimization: die-stacked DRAM. To avoid long interconnect delays and maintain the latency benefits of a private cache, SILO organizes the DRAM into *vaults*, each of which sits above a processor core. A vault, introduced in the Hybrid Memory Cube [15], is a multi-die stack of DRAM banks with a dedicated vault controller in the logic layer at the base of the stack. As shown in Fig. 5, in the SILO design, the DRAM controller for a vault is located on the CPU die, next to the core directly beneath the vault.

The third optimization is aimed at reducing the access latency to a vault by engineering the DRAM stack for low access latency. As explained in the next section, this involves using many banks, shorter pages and shorter bitlines/wordlines. While these optimizations reduce the storage capacity per vault compared to traditional capacity-optimized DRAM, they afford ultra-low access latency while still providing over a hundred MBs of capacity per core in today's process technology.

To summarize, SILO overcomes the area and delay constraints of shared LLCs through private die-stacked DRAM caches. An additional benefit of private caches is their immunity to cache contention, which plagues shared LLC designs. Finally, SILO reduces the access latency to the DRAM cache by engineering the DRAM for low latency at the expense of capacity. In the following two sections, we first describe our latency-optimized DRAM cache, followed by a detailed discussion of other aspects of the SILO architecture, including cache coherence.

IV. Architecting a Fast DRAM Cache

A. DRAM Technology Basics

Today's DRAM chips are comprised of DRAM cells and peripheral circuitry organized in a hierarchical structure as shown in Fig. 6. At the top-most level, a DRAM chip is divided into *banks* where cells share peripherals, including row and column decoders. The column width of a bank is referred to as a *page*. A bank is further divided into *subarrays* in which cells are connected through a common bitline and share sense amplifiers. In turn, a subarray consists of a number of *tiles* which have common global wordlines. Each tile has local wordlines and drivers. The thick grey segments in Fig. 6 represent the peripherals; vertical segments include wordline drivers and horizontal segments are sense amplifiers.

Tile dimensions determine the lengths of bitlines and local wordlines (subarrays and tiles have the same bitline length but different local wordline length as shown in Fig. 6). The electrical load observed on the lines is proportional to their lengths, resulting in higher transmission delays for longer lines. At the same time, longer bitlines and wordlines require less peripheral circuitry such as sense amplifiers and wordline drivers. Thus, while longer lines are good for area efficiency (defined as DRAM cell area divided by total chip area), they are bad for latency. Conversely, shorter lines reduce the electrical load but require more peripheral circuitry. The choice of line lengths is governed by design optimization targets.

B. Commodity DRAM Designs

Commodity DRAM products are designed to minimize cost-per-bit. This design target affects each level of the DRAM hierarchy. Firstly, DRAM manufacturers choose to limit I/O

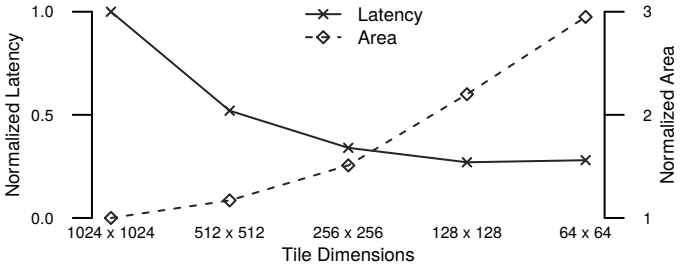


Fig. 7. Effect of DRAM tile dimensions on access latency and area.

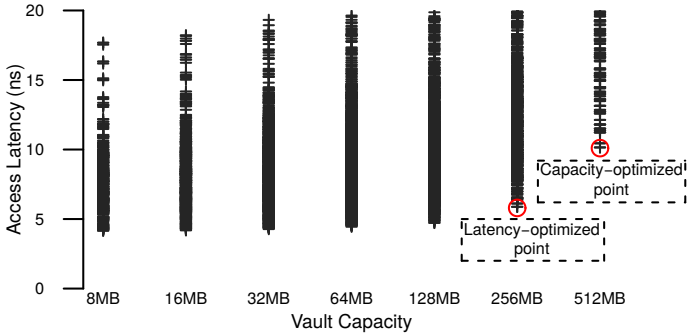


Fig. 8. A scatter plot of vault capacity vs access latency as a function of DRAM parameters.

and peripheral circuitry by sharing I/O between banks, thus allowing more chip area for DRAM cells. Secondly, in order to minimize the area of row and column decoders, a small number of banks is employed (e.g., 8 banks per chip in DDR3). Next, a bank is divided into only a small number of subarrays to minimize the area occupied by subarray-level sense amplifiers. Reducing the footprint of sense amplifiers is important for a density-optimized design because a sense amplifier can be 100 times larger than a DRAM cell [16]. Lastly, a subarray comprises of only a few tiles to reduce the number, and hence the area footprint, of the local wordline drivers. Fewer subarrays and tiles improve DRAM area efficiency but lead to longer bitlines and wordlines, which naturally increases latency as discussed above. In effect, tile dimensions determine the access latency of a DRAM core.

C. Effect of Tile Dimensions on DRAM Latency and Area

To quantify the effect of tile dimensions on DRAM area and latency we model a 1Gb DRAM die (details can be found in Sec. VI-B). In order to reduce the line lengths, we change tile dimensions by using a combination of DRAM parameters, namely: number of banks, page size, number of divisions per bitline (Ndbl) and number of divisions per wordline (Ndwl). Smaller tile dimensions correspond to shorter bitlines/wordlines.

Fig. 7 plots area and access latency as a function of tile dimensions. The values are normalized to a baseline design based on Micron DDR3 [17] having tile dimensions 1024x1024 DRAM cells. We observe that reducing the tile dimensions from the baseline 1024x1024 down to 256x256 decreases the access latency by 64% at a cost of a 49% increase in die area. Beyond that point, a mere 6% drop in access latency achieved with a tile size of 128x128 comes at a hefty 150% increase in area. Thus, we conclude that up to a certain point, reducing tile dimensions (and thus, increasing the number of tiles/subarrays for a fixed area) is effective in trading off capacity for latency, which makes for a valuable optimization space. However, beyond that, small latency gains come at exorbitant area cost and are not justified.

D. Optimizing Die-Stacked DRAM Cache Latency for SILO

As described in Sec. III, SILO uses the vault arrangement inspired by the HMC [18]. By treating each vault as a per-core private cache, SILO avoids long planar interconnect traversals that are detrimental to access latency. To further reduce latency within a vault, SILO sacrifices area efficiency by using a latency-optimized custom DRAM core. Following the discussion from Sec. IV-C, SILO introduces additional peripheral circuitry to effectively reduce line lengths in order to lower delays. This is achieved through the following optimizations:

- Large number of banks per vault to increase parallelism and minimize queuing at the memory interface.
- Shorter pages to reduce DRAM row size and hence global wordline length.
- Large number of subarrays per bank to reduce bitline length within a subarray.
- Many tiles per subarray to reduce wordline length.

Die Stacking and Thermal Feasibility. To maximize DRAM cache capacity, SILO employs die-stacking. In general, the height of a die-stacked cache, including SILO, is limited by thermal constraints. Up to 8 additional layers of DRAM have been shown to increase the temperature of the chip by only 6.5 degrees Celsius [19] and have a negligible effect on thermal distribution of the die [20]. Industrial specifications indicate feasibility of DRAM stacks with 8 layers and a logic die underneath [18], while 4-layered stacks are widely available in products [13].

Mapping the Design Space. To model the area and timing for a stacked DRAM vault in SILO, we perform technology analysis in CACTI (methodology details in Sec. VI-B). We conservatively model a 4-die DRAM stack and assume a 5mm^2 area per vault to match the core area beneath it. Using these constraints, we perform a DRAM parameter sweep to find all possible vault designs that fit in the area budget. The resulting designs are plotted as capacity-latency pairs in Fig. 8.

From the figure, we observe that lower capacity designs fit easily in the area budget while maintaining low access latency. Moving from 8MB to 128MB, the capacity increases by 16x while the latency increases by less than 10%. Going from 128MB to 256MB, the capacity doubles at the cost of a 15% latency increase. From there, another doubling in capacity to 512MB results in an 80% increase in access latency. Thus, for the set of parameters considered in this study, we find the

	Latency-optimized	Capacity-optimized
Area efficiency	1x	1.74x
Number of tiles	1x	0.25x
Access latency	1x	1.8x

TABLE I
COMPARISON OF LATENCY- VS CAPACITY-OPTIMIZED VAULT DESIGN POINTS. VALUES NORMALIZED TO THE LATENCY-OPTIMIZED POINT.

per-vault capacity of 256MB at a 5.5ns access latency to be the sweet spot for a latency-optimized design.

We further note that for a traditional DRAM cache, the higher-capacity (and higher latency) 512MB per-vault design point would be well justified, since the interconnect delays on the CPU side and in the chip-to-chip interface would add tens of nanoseconds to the DRAM access latency. Given that, an additional 4.5ns of latency, which is the difference between the lowest-latency 256MB and 512MB design points, would amount to a modest fraction of the overall delay, pointing to 512MB as a sweet spot for off-chip DRAM. Table I highlights the key differences in the two designs.

Technology Scaling. As both SRAM and DRAM memory technology are experiencing a gradual slowdown in their feature scalability, exploiting vertical stacking to overcome constraints of traditional cache hierarchies is an attractive option explored in this work. But how well is DRAM stacking projected to scale?

The number of layers, which ultimately determines the capacity of a die-stacked cache, is limited by two primary factors: thermals, which dictate the maximum height of the stack, and manufacturing technology (including testing and integration). Future improvements in waver-thinning and integration technology will allow more dies to be integrated in a fixed-height stack, thus providing a viable path to higher capacities. Indeed, ITRS 2.0 roadmap [21] projects die thickness to shrink to 5-15μm in the next 15 years, from the current 50-100μm, allowing tens of stacked layers.

V. SILO DESIGN DETAILS

This section details aspects of the SILO, including the organization of the DRAM cache (i.e., tag and data placement), cache coherence support and performance optimizations.

A. DRAM Cache Organization

In SILO, the die-stacked DRAM cache stores data, associated tags, and the directory metadata. Fig. 6 shows the layout of data and metadata in the cache. In this section, we focus on data and tag placement, while the next the section discusses cache coherence and directory organization.

To maximize available capacity, SILO uses a block-based cache organization. Based on the observations that separating the tag store and data store will lead to a significant latency increase due to the serialized accesses to both on a hit [22], SILO leverages a previously-proposed technique to integrate each data block with the corresponding tag into a single unified fetch unit called (TAD) [23]. Each access to the DRAM cache provides a single TAD, thus avoiding the delay of tag serialization.

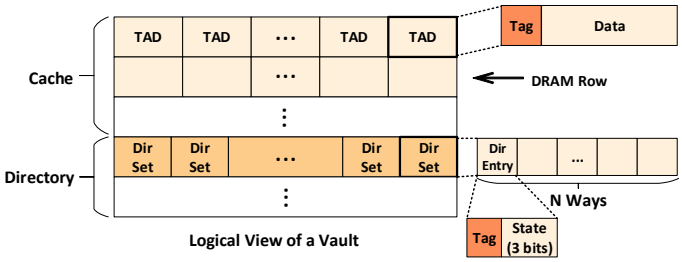


Fig. 9. DRAM organization.

SILO is inclusive of the on-chip private caches and is organized as a direct-mapped structure. The direct-mapped organization avoids the latency and energy overheads of a set-associative design, and is compensated for by the high capacity of the DRAM cache. Meanwhile, inclusion simplifies coherence and is easily afforded given the high capacity of the DRAM cache.

B. Directory-Based Cache Coherence

SILO uses a conventional directory-based MOESI protocol to maintain full coherence among its private caches. Misses in a core's private cache hierarchy (which consists of one or two levels of on-chip cache backed by a die-stacked DRAM vault) trigger a directory access. Logically, the directory sits below the DRAM LLC (i.e., logically closer to main memory). Physically, the directory is distributed in an address-interleaved fashion, with directory metadata stored in the DRAM caches as explained below.

Directory Organization: The fact that the LLC is private, inclusive and direct-mapped has two implications. First, because the LLC is private and inclusive, the size of the tag store is directly proportional to the total LLC capacity. Assuming every vault stores a unique set of blocks with respect to every other vault, the tag store must be able to accommodate the full set of tags across all vaults. Secondly, associativity at the directory is dictated by the core count, not by the higher-level caches. This is because the LLC is direct-mapped and is inclusive of higher-level caches.

Based on these observations, we use a duplicate-tag directory organization *without* a sharing vector. Fig. 9 shows the design. Logically, a directory is organized as an N-way associative tag store, where N is equal to the core count. Each directory entry stores a tag and the coherence state of the block. The way position of a given directory entry indicates the core caching the associated block. For instance, a tag in the directory way position 1 indicates that Core 1 is caching the block. Finding sharers requires reading the tags for all N logical ways in the directory. Most coherence state updates require modifying only one directory entry (tag and/or state bits); however, in the worst case, all N directory entries in a given set may need to be updated (e.g., when a block shared by all cores transitions to an exclusive state).

Coherence Protocol: In a processor with a conventional on-chip shared LLC, the LLC serves as the point of coherence. In such a system, a writeback from a core's L1 involves simply updating the LLC. However, in a system with only private caches, the point of coherence is main memory, so a writeback incurs the high latency, energy and bandwidth overhead of a main memory access. To avoid the need for such an expensive writeback on a dirty eviction from a core's L1, SILO uses the MOESI protocol for maintaining coherence. The O state indicates that the block is valid, dirty and Owned; that is, the cache that has the block in this state must respond to coherence requests for the block. Compared to the MESI protocol, the primary advantage of MOESI is that a modified block can be directly supplied to other cores that want to read it without first writing it back to memory.

C. Performance Optimizations

In SILO, a miss in the local DRAM cache vault requires an access to the directory node for the block. Because the directory metadata resides in the DRAM cache, fetching it requires a DRAM cache access. And if the requested block is found at another node, yet another DRAM cache access is incurred to get the block. In total, up to three DRAM cache lookups may be needed to access a block on chip.

In light of the high miss penalties on misses in the private DRAM cache, we consider two performance optimizations in the SILO architecture:

Local Vault Miss Predictor: The TAD organization in the DRAM cache means that a miss is not discovered until the DRAM access completes. A miss predictor, such as a MissMap [24], can avoid DRAM accesses if they are known to be misses, thus avoiding the associated latency cost.

Directory Cache: A miss in a core's private cache hierarchy triggers a DRAM access at the requested block's directory node to fetch the directory metadata. A directory cache [25] can eliminate the DRAM access for directory metadata by serving it from a fast on-chip SRAM.

These two optimizations can be applied separately or in concert. We consider all three options and show results in Sec. VII-B.

D. Discussion

SILO requires non-commodity DRAM to achieve low latency and maximize performance gains. Traditionally, the DRAM industry has resisted such designs; however, the booming datacenter market and the presence of a few hyper-scale players (e.g., Google, Amazon, Facebook) may tilt the dynamic toward DRAM customization to accommodate specific needs of datacenter customers. The trend of customizing processors [26] and deploying custom accelerators [27] for datacenters is already underway. This work shows the large gains that can be reaped by specializing the DRAM.

To help offset the cost of non-commodity DRAM, we note that on-chip shared LLCs occupy around a third of chip area in today's server processors [3], [4]. The associated die real-estate is expensive because server CPUs are generally built with leading-edge process technology. Because SILO completely avoids the need for an on-chip LLC, it vacates the associated die real-estate. In turn, this can afford either a reduction in die size, thus improving cost and yield, or addition of more cores within the same die area as a baseline processor with a shared LLC.

Another benefit of eliminating the on-chip LLC is that it reduces demands on the on-chip interconnect. High hit rates in the private die-stacked DRAM help reduce on-chip instruction and data traffic, while a smaller die (afforded by eliminating the on-chip LLC) helps reduce wire delays. Together, these features may lead to a less costly (area-wise) and/or faster NOCs. Our evaluation is conservative and does not take advantage of any such NOC optimizations afforded by SILO.

VI. Methodology

A. Evaluated Systems

We model a 16-core CMP with ARM-like 3-way OoO cores running at 2.0 GHz. Table II details the system parameters. We extract LLC and DRAM cache latencies from CACTI (details in Sec. VI-B). For DRAM (both cache and main memory), we assume a closed page policy, which has been shown to outperform open-page on server workloads [28]. We assume a fairly aggressive memory access latency of 50ns; the combination of modest core frequency and low memory access latency is disadvantageous for SILO since LLC misses are relatively "cheap". A faster core and/or slower memory would amplify the penalty of a miss in the LLC, providing a larger benefit to SILO, which has a lower LLC miss rate than conventional LLC organizations.

Our main evaluation focuses on 2-level cache hierarchies, which have been shown to be superior to 3-level designs for scale-out workloads [5]. Sec. VII-F evaluates cache hierarchies with three levels.

Baseline: The baseline processor is based on *Scale-out Processors* [5], which uses a fast but modestly sized shared LLC in a two-level cache hierarchy. We use an 8MB LLC split into 16 banks, with a 5-cycle bank access latency. The average round trip time for an LLC hit, including the NOC, is 23 cycles.

Baseline+DRAM$: Baseline augmented with an 8GB conventional DRAM cache. The DRAM cache is hardware managed and uses a page-based arrangement considered state-of-the-art for servers [29], [30]. Conventional DRAM caches use the same DRAM technology as main memory and as such have similar access latency. Indeed, the on-package DRAM cache in Intel's Xeon Phi Knight's Landing is slightly slower than main memory [13]. We optimistically assume that the access latency of the conventional DRAM cache is 20% faster than that of main memory. We further assume perfect miss prediction and infinite bandwidth.

SILO: A fully private two-level cache hierarchy with die-stacked DRAM vaults as the LLC. We use a custom, latency optimized vault design with 256MB of capacity per vault as discussed in Sec. IV-D. The vault access latency is 11 cycles. We use a 64-bit wide interface adding 8 serialization cycles

Processor	16-core, 2GHz, 3-way OoO, 128 ROB, ISA: UltraSPARC v9
L1-I/D	64KB, 8-way, 64B line, 3 cycles, private, stride data prefetcher
Interconnect	4x4 2D mesh, 3 cycles/hop
Baseline on-chip LLC	8MB shared NUCA, 5 cycles, 16-way, 64B line, non-inclusive MESI, LRU
SILO die-stacked DRAM LLC	Private, direct-mapped, 64B line, 512B page, inclusive MOESI SILO: 256MB vault/core, 23 cycles SILO-CO: 512MB vault/core, 32 cycles
Trad. DRAM cache	8GB, page-based, direct-mapped, 40ns
Main memory	Access latency 50ns

TABLE II
MICROARCHITECTURAL PARAMETERS OF THE SIMULATED SYSTEMS.

Baseline on-chip SRAM LLC	30mW per bank static power, 0.25nJ/access dynamic energy
SILO die-stacked DRAM LLC	120mW per vault static power, 0.4nJ/access dynamic energy
Main memory	4W static power, 20nJ/access dynamic energy

TABLE III
MEMORY SUBSYSTEM ENERGY/POWER PARAMETERS.

for a TAD block. We add 4 cycles of vault controller delay bringing the total cache access latency to 23 cycles.

SILO-CO: SILO with capacity-optimized vaults of 512MB at an access latency of 20 cycles. The total cache access latency, including vault controller and serialization delay, is 32 cycles.

Vaults-Sh: Die-stacked *shared* LLC organization with latency-optimized vaults. This design point evaluates the effect of DRAM latency optimization without the private organization. The latency-optimized vaults are stacked directly on top of cores (just like in SILO) but the aggregate vault capacity of 4GB is shared by all cores in a NUCA address-interleaved manner. The average round trip time for a hit, including a vault access and NOC traversal, is 41 cycles.

B. DRAM and SRAM Technology Modeling

We use CACTI-3DD to model DRAM and SRAM access latencies. We model DRAM and SRAM technologies at 22nm. For the SRAM LLC, we account for advanced latency reduction techniques [31] and use the low-standby-power cell type. Area and/or capacity constraints imposed by individual studies are highlighted in the text where appropriate.

To measure the energy and power consumed in the memory subsystem, including (as appropriate) the SRAM LLC, DRAM cache and main memory, we use a hybrid energy modelling framework that makes use of technology-specific parameters and cycle-accurate simulation statistics. We use CACTI-3DD to extract energy and power parameters for SRAM and stacked DRAM technology [31], [32]. We estimate main memory DRAM parameters using commercial DDR3 device specifications [33]. Table III summarizes the energy and power values obtained from these tools and used in the evaluation.

Scale-out	
Web Search	*Apache Nutch 1.2 / Lucene 3.0.1,* 92 clients, 1.4 GB index, 15 GB data segment
Data Serving	*Apache Cassandra 0.7.3,* 150 clients, 8000 operations per second
Web Frontend	*Apache HTTP Server v2.0,* 16K connections fastCGI, worker threading model
MapReduce	*Hadoop MapReduce, Apache Mahout 0.6,* Bayesian classification algorithm
SAT Solver	*Cloud9 parallel symbolic execution engine,* Klee SAT Solver
Enterprise	
TPCC	*IBM DB2 v8 ESE Database Server,* , 64 clients 100 warehouses (10GB), 2GB buffer pool
Oracle	*Oracle 10g Enterprise Database Server,* 100 warehouses (10GB), 1.4GB SGA
Zeus	*Zeus Web Server,* 16K connections, fastCGI

TABLE IV
SERVER WORKLOADS USED FOR EVALUATION.

C. Simulation Infrastructure

We use Flexus [34], a full system multiprocessor simulator, based on Simics. Flexus models the SPARC v9 ISA and extends Simics with out-of-order (OoO) cores, memory hierarchy, and on-chip interconnect (NOC). To reduce simulation time, Flexus integrates the SMARTS [35] methodology for sampled execution. For each sample, we first warm-up architectural and microarchitectural state, then run cycle-accurate simulation and measure performance. In order to evaluate performance, we measure the number of application instructions executed per cycle (including time spent executing operating system code); this metric has been shown to reflect system throughput [34].

D. Workloads

We evaluate the various cache architectures using a range of workloads. Our scale-out workloads include Web Search, Data Serving, MapReduce and SAT Solver workloads, which are taken from CloudSuite [36], and a Web Frontend workload from SPECweb2009. The latter replaces the Cloudstone Web Frontend workload from CloudSuite, which exhibits poor scalability at high core counts [5]. For the same reason, we do not use the Media Streaming workload from CloudSuite, as it does not scale beyond 2-4 threads [37]. We further investigate the utility of SILO for traditional enterprise applications. Details of these workloads are listed in Table IV. We further investigate the utility of SILO for traditional enterprise applications, listed in Table IV

For simulation, samples are drawn over 80 billion instructions (5 billion per core) for each workload. For each sample, we run cycle-accurate simulations from checkpoints that include full architectural and partial microarchitectural state, which includes caches and branch prediction structures. We run for 100K cycles to achieve steady state and measure over the following 200K cycles per sample.

We also consider multi-programmed batch workload deployments, representative of public cloud use cases. We

Name	Description
mix1	sjeng-calculix-mcf-omnetpp
mix2	lbm-gamess-namd-gromacs
mix3	mcf-zeusmp-calculix-lbm
mix4	tonto-gamess-bzip2-namd
mix5	mcf-povray-gcc-cactusADM
mix6	gobmk-perlbench-milc-astar
mix7	xalancbmk-sjeng-cactusADM-bwaves
mix8	calculix-leslie3d-astar-gcc
mix9	gromacs-gobmk-gamess-astar
mix10	omnetpp-zeusmp-soplex-povray

TABLE V
SPEC'06 MIXES USED FOR EVALUATION.

generate 10 randomly-drawn mixes, each consisting of four workloads from SPEC'06 [38]. For each mix, workloads are drawn without replacement. The mixes are listed in Table V. We draw samples over 20 billion instructions (5 billion per core) for the 4-core setup. Cycle-accurate simulation is run for 300K cycles with measurement over the last 200K cycles.

VII. Evaluation and Discussion

We first evaluate SILO against traditional cache architectures, on scale-out workloads, which are the primary target of this work. Next, we assess the designs on enterprise and batch applications. Following that, we examine performance isolation and three-level cache hierarchies.

A. Performance on Scale-out Workloads

Fig. 10 plots the performance of the evaluated systems on scale-out workloads, with results normalized to the baseline system (see Sec. VI for a description of evaluated systems). We observe that both SILO designs consistently provide better performance than the baseline designs. This is an expected result as SILO provides a higher LLC capacity with the same hit latency as the baseline. SILO improves performance by 5-54%, with a geomean performance improvement of 28%, across the scale-out workloads. The highest performance gains are observed for MapReduce and SAT Solver at 54% and 37%, respectively. On Web Search, SILO achieves a speedup of 29%. Sec. II-A identified that aggregate LLC capacities greater than 512MB are beneficial for the performance of Web Search. Thus SILO, which has an aggregate LLC capacity of 4GB (256MB per vault), delivers higher performance on this workload than the baseline.

The capacity-optimized SILO design (SILO-CO) delivers a geomean performance improvement of 25%, slightly below the 28% speedup provided by the latency-optimized SILO. Despite twice the per-vault capacity, the SILO-CO design has higher vault access latency, as shown in Sec. IV-D. Consistent with our sensitivity studies in Sec. II, higher capacity is only beneficial if not accompanied by a higher access latency. Similarly, the shared vaults design (Vaults-Sh), delivers a geomean performance improvement of only 6%, despite employing latency-optimized vaults. NOC traversal adds to the overall access latency of Vaults-Sh, diminishing the performance benefits of the high capacity vaults.

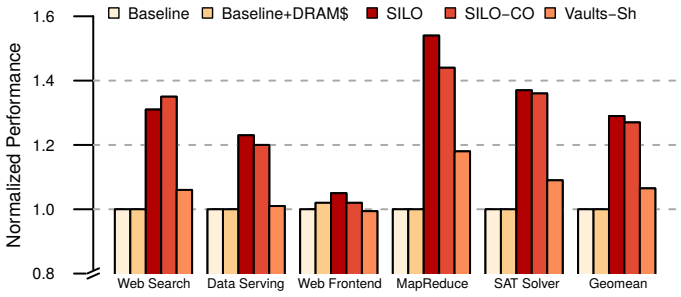


Fig. 10. Performance on scale-out workloads.

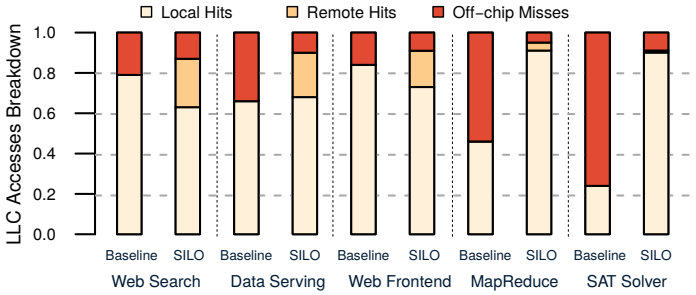


Fig. 11. Normalized LLC hits and misses for SILO vs baseline. Note that all hits in the baseline shared NUCA LLC are shown as 'local'.

Finally, we observe that baseline and baseline+DRAM$ designs provide similar performance. We identify the high access latency to the conventional DRAM cache as the main reason for the limited benefit offered by the baseline+DRAM$ design. As noted in Sec. II-B, benefits of large cache capacities disappear at high access latencies. Conventional DRAM caches *are* beneficial in alleviating the bandwidth bottleneck [29]; however, today's server CPUs are not bandwidth limited on scale-out workloads as shown in recent work [10] and corroborated by Google [39] and our own results. As such, conventional DRAM caches do not benefit these designs.

B. Analysis

In this section, we characterize LLC effectiveness in SILO as compared to the baseline, and explore the usefulness of SILO design optimizations.

1) LLC Hit Rate: Fig. 11 plots normalized LLC hits and misses for the baseline and SILO designs. In general, SILO consistently reduces off-chip misses compared to the baseline across all workloads. Miss rate reductions range from 8% to 67%, with the largest reductions on SAT Solver (67%) and MapReduce (49%). Not surprisingly, these two workloads observed the greatest performance improvement as noted in Sec. VII-A. Web Search, Data Serving and Web Frontend observe comparatively lower miss reductions consistent with their performance improvements.

As the figure shows, the majority of hits in SILO come from the local vault (63-91% of all hits). This is important for performance, because local hits are faster than remote hits, which incur a directory lookup and a multi-hop NOC traversal.

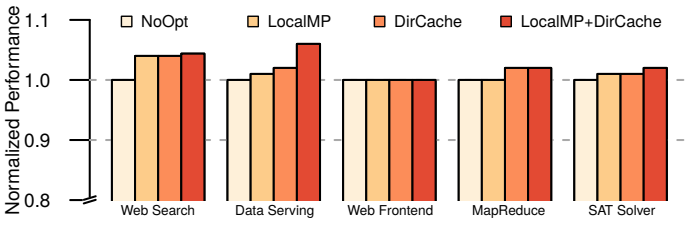


Fig. 12. Effect of SILO design optimizations on CloudSuite. Study assumes ideal vault miss predictor and ideal directory cache.

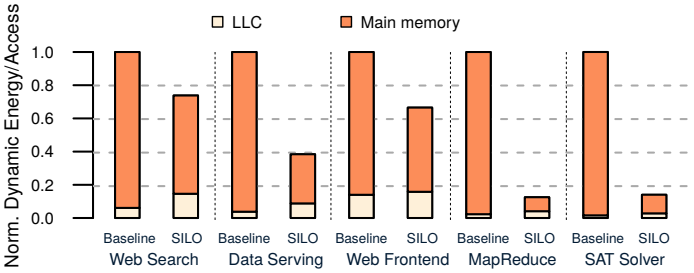


Fig. 13. Dynamic energy of the memory subsystem.

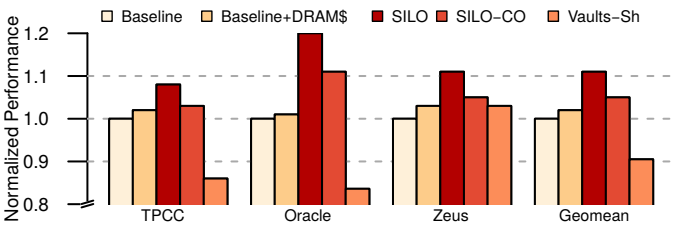


Fig. 14. Performance results for Enterprise workloads.

Nonetheless, remote hits in SILO are also beneficial as they are faster than main memory accesses.

2) SILO Performance Optimizations: Sec. V identified two possible optimizations to reduce the latency incurred by DRAM accesses to (i) the local vault, and (ii) the in-DRAM directory in the case of a local vault miss. We now evaluate the usefulness of these optimizations *in the limit*. We consider the following configurations:

- *NoOpt*: SILO with no optimizations.
- *LocalMP*: SILO with a Miss Predictor for local vault accesses. The predictor is perfect, requiring 0 time and having 100% accuracy.
- *DirCache*: SILO with a directory cache. The directory cache is perfect, requiring 0 time and having 100% accuracy.
- *LocalMP+DirCache*: SILO with both local vault Miss Predictor and an ideal directory cache.

Fig. 12 plots the performance of the four designs. We observe marginal performance improvements in the optimized designs. Data Serving observes the largest benefit at 6% speedup with both local vault miss predictor and a directory cache. This result is consistent with the RW-sharing characterization study of Sec. II-C, which shows Data Serving to have a high sensitivity to the LLC access latency for RW-shared data. Additionally, Data Serving shows a significant amount of remote vault hits (as shown in Fig. 11). LocalMP and DirCache optimizations reduce the latency of remote vault hits, thus improving performance. Web Frontend exhibits similar sensitivity to the latency of RW-shared data and fraction of remote vault hits as Data Serving. However, the overall speedup achieved by SILO is modest on Web Frontend (Fig. 10), and hence the benefits of performance optimizations are small. We conclude that the benefits of the considered design optimizations do not outweigh their cost and extra design complexity.

C. Energy Efficiency

We examine the effects of SILO architecture on the memory subsystem energy dissipation using the parameters in Sec. VI-B. Fig. 13 illustrates memory subsystem dynamic energy in SILO normalized to that in the baseline system. Compared to the baseline, SILO reduces dynamic energy by 26-87% across the evaluated scale-out workloads. The high hit rate in SILO significantly reduces off-chip traffic, thereby reducing dynamic energy in main memory and I/O, which explains SILO's energy-efficiency advantage.

While not shown in the figure, we note that SILO expends more power in the LLC than the baseline due to a combination of (i) higher static power of the large number of DRAM banks, (ii) higher dynamic energy per LLC access, and (iii) more accesses per unit time due to higher IPC. The total LLC power consumption in SILO does not exceed 2.5W across all evaluated workloads, which is a small fraction of the total power budget of a 16-core server processor.

D. Performance on Other Workloads

We compare SILO against alternatives on enterprise and batch workloads.

1) Enterprise workloads: Fig. 14 plots the performance of enterprise workloads with results normalized to the baseline system (see Sec. VI for a description of evaluated systems). Compared to the baseline, SILO provides a geomean performance improvement of 11% while for SILO-CO the gain is 5%. The Vaults-Sh design, with shared vaults, delivers a 9% slowdown in performance. This slowdown can be attributed to the long access latency to the LLC due to a combination of DRAM access and NOC traversal.

Unlike on the scale-out workloads, the baseline+DRAM$ design shows performance gains across all enterprise workloads, achieving up to a 3% speed-up over the baseline. These workloads operate on smaller datasets compared to the scale-out applications from CloudSuite. The high capacity DRAM cache in the baseline+DRAM$ design captures the smaller datasets, resulting in frequent cache hits. Because the conventional DRAM cache has an access latency 20% lower than that of main memory, the DRAM cache hits contribute to performance improvement.

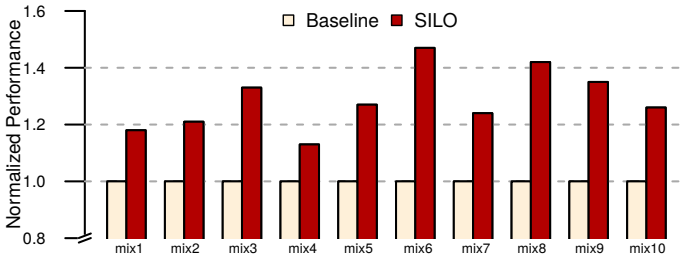


Fig. 15. Results for 4-core SPEC2006 mixes.

2) Multi-programmed Batch Workloads: Fig. 15 plots the performance of 4-core SPEC'06 mixes for baseline and SILO designs. Overall, SILO delivers a significant performance gain of up to 47% (28% on average) due to its massive core-private cache capacity. The capacity advantage comes at a similar latency as the shared LLC and in a contention-free manner – an issue further explored in the next section. While SILO performs better on all mixes, we observe higher performance gains on certain mixes, e.g. mix3, mix6, mix8, and mix9. These mixes include memory-intensive applications such as `mcf`, `lbm`, `milc` and `astar`, which benefit the most from the larger capacity, and therefore, exhibit higher performance improvement.

E. Performance Isolation

Heterogeneous applications colocated on the same physical server contend for the available shared LLC. This inter-core contention can compromise performance, which is a particular concern for applications with strict latency targets. An all-private cache hierarchy, provided by SILO, offers the premise of removing LLC contention and guaranteeing strong performance isolation.

In order to evaluate the degree of performance isolation SILO provides, we measure the performance of Web Search running on a processor (i) alone and (ii) together with `mcf`, a memory-intensive SPEC'06 benchmark. Web Search runs on 8 cores while `mcf`, when present, runs on the other 8 cores of the 16-core setup. We use two LLC configurations: a traditional shared LLC and SILO.

Table VI shows the results of the experiment where the performance of Web Search is normalized to stand-alone Web Search setup with a shared LLC. We observe two trends. First, SILO improves performance of Web Search by 20% when running alone. Secondly, the performance in a system with SILO is unaffected by colocation with `mcf`. In contrast, Web Search suffers a 10% performance degradation when running on a shared LLC system under colocation. We conclude that SILO not only delivers a significant performance improvement compared to a shared LLC baseline, but also provides performance isolation under colocation.

F. 3-Level Cache Hierarchy

So far, our evaluation has focused on a latency-optimized two-level cache hierarchy based on Scale-out Processors [5]. In this section, we evaluate a more conventional three-level cache hierarchy, adding a 512KB private second-level SRAM cache to all configurations. We consider two baseline designs: (1) an Intel-like design featuring a 32MB SRAM-based NUCA LLC, referred to as *3level-SRAM*, and (2) a 128MB eDRAM-based NUCA LLC similar to the POWER 9 [40], referred to *3level-eDRAM*. Using CACTI, we find the bank access latency in the SRAM design to be 7 cycles, and optimistically assume the same access latency for the larger eDRAM banks. To keep the study focused, we only consider the latency-optimized SILO variant titled *3level-SILO*.

	Shared LLC	SILO
Web Search alone	-	+20%
Web Search + `mcf`	-10%	+20%

TABLE VI
PERFORMANCE OF WEB SEARCH UNDER DIFFERENT SETUPS.

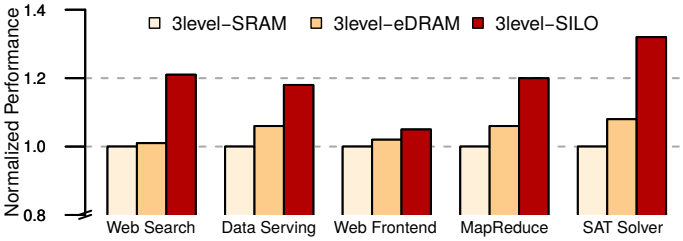


Fig. 16. Performance on scale-out workloads with a 3-level hierarchy.

Fig. 16 shows the results of the study, which follow the same trend as the results for a two-level hierarchy. Compared to the 3-level SRAM LLC, the 3-level SILO design improves performance on all workloads by 14%, on average, and a max of 32%. Consistent with earlier results and studies in the motivation section, the largest gains are registered on MapReduce and SAT Solver workloads (20% and 32%, respectively), while the smallest improvement of 5% is on Web Frontend, which generally has the lowest sensitivity to cache optimizations among the evaluated workloads.

The 3-level eDRAM design provides a modest performance improvement of up to 8% over its SRAM counter-part, but is always inferior to SILO. The overall trend is consistent with the study in Sec. II-A, which showed that higher LLC capacities at low latencies are beneficial to scale-out workloads.

VIII. Related Work

Maximizing performance for planar LLCs: To reduce the average access time for large, distributed LLCs, prior work has proposed Non-Uniform Cache Architecture (NUCA) [22]. Static NUCA (S-NUCA) designs use address interleaving to spread data across cache banks distributed on chip. While simple to implement, such designs require multi-hop NOC traversals in the common case, which result in high access latencies to remote cache banks. Dynamic NUCA (D-NUCA) designs use adaptive data placement to reduce the average access latency through a combination of data placement, replication and migration to make data available in the cache banks nearest to the requesting core [41]–[45]. Fundamentally, such schemes are limited by the small capacity of nearby banks

on a planar die. SILO circumvents capacity-latency trade-off of planar caches by providing core-private die-stacked DRAM vaults with hundreds of MBs of capacity.

Stacked DRAM Caches: Die-stacked DRAM technology has been identified as a suitable means to provide gigascale caches [23], [29], [30]. The technology extends high density commodity DRAM with higher bandwidth and better power efficiency. The target applications are bandwidth-intensive, such as those running on GPUs and many-core HPC processors. Indeed, the latest Nvidia and AMD GPUs and Intel's Knight's Landing feature die-stacked DRAM [13], [46], [47]. Due to the significant delays involved in routing the request from the requesting core to the desired DRAM bank, access latencies are comparable to main memory [48]. In fact, Intel's Knight's Landing has a higher access latency to the DRAM cache than to main memory [13].

To improve the high access latency of a serialized tag and data lookup, research proposals have argued for tag placement in SRAM [24], [29], [49], [50] and for using direct-mapped in-DRAM tag designs [13], [23]. These policies reduce the tag lookup cost, but leave the underlying DRAM technology and processor organization unchanged. Jenga [51] introduced a reconfigurable cache hierarchy composed of SRAM and die-stacked DRAM tiles. While improving access locality is part of Jenga, cores of a given application share the full set of cache banks allocated to that application; as such, Jenga's caches are fundamentally shared across cores. SILO differs from these works in its use of an all-private cache hierarchy and custom DRAM technology.

DRAM Latency Optimization: Various custom DRAM technologies have been introduced in commercial products to provide lower latency but at higher cost-per-bit than commodity DRAM [52]–[55]. Technical details are generally scarce for these products, but they tend to advertise more banks and subarrays than commodity DRAM, making them similar to the vaults in SILO. However, due to the fact that the latency-optimized dies are packaged into discrete DRAM chips, the actual end-to-end latency savings are small due to the CPU-side and chip-to-chip interconnect delays. In contrast, SILO minimizes interconnect delays by using DRAM stacked directly on top of the processor die, and treating each DRAM vault as a core-private cache.

On the research side, prior works have looked at mitigating the overhead of additional peripheral circuitry for latency reduction by using segmented bitlines in DRAM [56], providing additional circuitry for selective banks [57] and partitioning the DRAM die into independent units [58], [59]. These techniques can be applied to the custom DRAM technology in SILO to increase vault capacities without compromising the access latency. Other techniques target reducing DRAM latency by overlapping accesses to different subarrays [60] and improving row-buffer locality by exploiting access patterns [61]–[64]. While these techniques allow overlapping access latencies of different requests, they do not reduce the actual access latency.

IX. Conclusion

As traditional technology scaling nears its end, processor architectures must embrace emerging technologies and new architectural paradigms in the quest for higher performance. This work takes a step in this direction by showing that traditional shared LLCs offer limited room for improving performance in future server processors as they are unable to satisfy the requirement of large cache capacity and low access latency demanded by scale-out workloads. In response, we introduce SILO – a Die-Stacked Private LLC Organization which combines on-chip private caches with per-core LLC slices in die-stacked DRAM. SILO resolves the latency/capacity conundrum through the use of a private LLC organization and latency-optimized die-stacked DRAM. Our evaluation of SILO shows that it improves performance by 5-54% over a state-of-the-art server processor design on a set of scale-out workloads while also providing a high degree of performance isolation.

Acknowledgment

We would like to thank Priyank Faldu, Nikos Nikoleris, Vijay Nagarajan, Daniel Sorin and the anonymous reviewers for their valuable feedback. This work was supported by the Engineering and Physical Sciences Research Council (grant EP/L01503X/1), EPSRC Centre for Doctoral Training in Pervasive Parallelism at the University of Edinburgh, School of Informatics and ARM PhD Scholarship Program.

References

[1] L. A. Barroso, J. Clidaras, and U. Hölzle, "The datacenter as a computer: An introduction to the design of warehouse-scale machines," *Synthesis Lectures on Computer Architecture*, vol. 8, 2013.

[2] N. R. Council *et al.*, *Productivity and Cyclicality in Semiconductors: Trends, Implications, and Questions: Report of a Symposium*. National Academies Press, 2004.

[3] *Intel Xeon Processor E7-8890 v3*, https://www.intel.co.uk/content/www/uk/en/products/processors/xeon/e7-processors/e7-8890-v3.html.

[4] *IBM Power8*, https://www.ibm.com/power/hardware.

[5] P. Lotfi-Kamran, B. Grot, M. Ferdman, S. Volos, Y. O. Koçberber, J. Picorel, A. Adileh, D. Jevdjic, S. Idgunji, E. Özer, and B. Falsafi, "Scale-out processors," in *39th International Symposium on Computer Architecture*, 2012.

[6] X. Zhang, E. Tune, R. Hagmann, R. Jnagal, V. Gokhale, and J. Wilkes, "CPI2: CPU performance isolation for shared compute clusters," in *8th Eurosys Conference*, 2013.

[7] H. Yang, A. Breslow, J. Mars, and L. Tang, "Bubble-flux: Precise online qos management for increased utilization in warehouse scale computers," in *40th International Symposium on Computer Architecture*, 2013.

[8] H. Kasture and D. Sanchez, "Ubik: efficient cache sharing with strict qos for latency-critical workloads," in *19th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2014.

[9] D. Sanchez and C. Kozyrakis, "Vantage: scalable and efficient fine-grain cache partitioning," in *38th International Symposium on Computer Architecture*, 2011.

[10] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, "Clearing the clouds: a study of emerging scale-out workloads on modern hardware," in *17th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2012.

[11] D. H. Woo, N. H. Seong, D. L. Lewis, and H.-H. S. Lee, "An optimized 3D-stacked memory architecture by exploiting excessive, high-density tsv bandwidth," in *16th International Conference on High-Performance Computer Architecture*, 2010.

[12] T. Kgil, S. D'Souza, A. Saidi, N. Binkert, R. Dreslinski, T. Mudge, S. Reinhardt, and K. Flautner, "Picoserver: using 3D stacking technology to enable a compact energy efficient chip multiprocessor," in *12th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2006.

[13] A. Sodani, "Knights landing (KNL): 2nd generation Intel® Xeon Phi processor," in *Hot Chips 27 Symposium*, 2015.

[14] J. Kim and Y. Kim, "HBM: Memory solution for bandwidth-hungry processors," in *Hot Chips 26 Symposium*, 2014.

[15] J. T. Pawlowski, "Hybrid memory cube (HMC)," in *Hot Chips 23 Symposium*, 2011.

[16] D. Lee, S. Ghose, G. Pekhimenko, S. Khan, and O. Mutlu, "Simultaneous multi-layer access: Improving 3D-stacked memory bandwidth at low cost," *ACM Transactions on Architecture and Code Optimization*, vol. 12, 2016.

[17] K. H. Kyung, C. W. Kim, J. Y. Lee, J. H. Kook, S. M. Seo, J. H. Kim, J. Sunwoo, H. C. Lee, C. S. Kim, B. H. Jeong *et al.*, "A 800mb/s/pin 2Gb DDR2 SDRAM using an 80nm triple metal technology," in *International Digest of Technical Papers. Solid-State Circuits Conference*, 2005.

[18] *Hybrid Memory Cube Specification 2.1*, http://hybridmemorycube.org/.

[19] G. H. Loh, "3D-stacked memory architectures for multi-core processors," in *35th International Symposium on Computer Architecture*, 2008.

[20] B. Black, M. Annavaram, N. Brekelbaum, J. DeVale, L. Jiang, G. H. Loh, D. McCaule, P. Morrow, D. W. Nelson, D. Pantuso, P. Reed, J. Rupley, S. Shankar, J. P. Shen, and C. Webb, "Die stacking (3D) microarchitecture," in *39th International Symposium on Microarchitecture*, 2006.

[21] *International Technology Roadmap for Semiconductors 2.0*, http://www.itrs2.net/.

[22] C. Kim, D. Burger, and S. W. Keckler, "An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches," in *10th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2002.

[23] M. K. Qureshi and G. H. Loh, "Fundamental latency trade-off in architecting dram caches: Outperforming impractical sram-tags with a simple and practical design," in *45th International Symposium on Microarchitecture*, 2012.

[24] G. H. Loh and M. D. Hill, "Efficiently enabling conventional block sizes for very large die-stacked DRAM caches," in *44th International Symposium on Microarchitecture*, 2011.

[25] A. Gupta, W.-D. Weber, and T. Mowry, "Reducing memory and traffic requirements for scalable directory-based cache coherence schemes," in *Scalable shared memory multiprocessors*. Springer, 1992, pp. 167–192.

[26] L. Gwennap, "ThunderX rattles server market," *Microprocessor Report*, vol. 29, 2014.

[27] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *44th International Symposium on Computer Architecture*, 2017.

[28] S. Volos, J. Picorel, B. Falsafi, and B. Grot, "BuMP: Bulk memory access prediction and streaming," in *47th Annual International Symposium on Microarchitecture*, 2014.

[29] D. Jevdjic, S. Volos, and B. Falsafi, "Die-stacked dram caches for servers: hit ratio, latency, or bandwidth? have it all with footprint cache," in *40th International Symposium on Computer Architecture*, 2013.

[30] D. Jevdjic, G. H. Loh, C. Kaynak, and B. Falsafi, "Unison cache: A scalable and effective die-stacked dram cache," in *47th International Symposium on Microarchitecture*, 2014.

[31] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing nuca organizations and wiring alternatives for large caches with cacti 6.0," in *40th International Symposium on Microarchitecture*, 2007.

[32] K. Chen, S. Li, N. Muralimanohar, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "CACTI-3DD: Architecture-level modeling for 3D die-stacked dram main memory," in *Design, Automation & Test in Europe Conference & Exhibition*, 2012.

[33] *Micron. 1.35V DDR3L power calculator (4Gb x16 chips)*, 2013.

[34] 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*, vol. 26, 2006.

[35] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe, "SMARTS: accelerating microarchitecture simulation via rigorous statistical sampling," in *30th International Symposium on Computer Architecture*, 2003.

[36] *CloudSuite: The Benchmark Suite of Cloud Services*, http://cloudsuite.ch/.

[37] K. Biswas, *Darwin Streaming Server 6.0.3 - Performance and Load tests*, https://www.codeproject.com/Articles/41874/Darwin-Streaming-Server-setup-customization.

[38] J. L. Henning, "SPEC CPU2006 benchmark descriptions," *SIGARCH Computer Architecture News*, vol. 34, 2006.

[39] S. Kanev, J. P. Darago, K. Hazelwood, P. Ranganathan, T. Moseley, G.-Y. Wei, and D. Brooks, "Profiling a warehouse-scale computer," in *42nd International Symposium on Computer Architecture*, 2015.

[40] S. K. Sadasivam, B. W. Thompto, R. Kalla, and W. J. Starke, "IBM Power9 processor architecture," *IEEE Micro*, vol. 37, 2017.

[41] M. Awasthi, K. Sudan, R. Balasubramonian, and J. Carter, "Dynamic hardware-assisted software-controlled page placement to manage capacity allocation and sharing within large caches," in *15th International Conference on High-Performance Computer Architecture*, 2009.

[42] B. M. Beckmann, M. R. Marty, and D. A. Wood, "ASR: adaptive selective replication for CMP caches," in *39th International Symposium on Microarchitecture*, 2006.

[43] M. Zhang and K. Asanovic, "Victim replication: maximizing capacity while hiding wire delay in tiled chip multiprocessors," in *32nd International Symposium on Computer Architecture*, 2005.

[44] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, and S. W. Keckler, "A NUCA substrate for flexible CMP cache sharing," *IEEE Transactions on Parallel Distributed Systems*, vol. 18, 2007.

[45] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, "Reactive NUCA: near-optimal block placement and replication in distributed caches," in *36th International Symposium on Computer Architecture*, 2009.

[46] *NVIDIA TESLA V100*, https://www.nvidia.com/en-us/data-center/tesla-v100/.

[47] *High Bandwidth Memory*, https://www.amd.com/en/technologies/hbm.

[48] D. W. Chang, G. Byun, H. Kim, M. Ahn, S. Ryu, N. S. Kim, and M. Schulte, "Reevaluating the latency claims of 3D stacked memories," in *18th Asia and South Pacific Design Automation Conference*, 2013.

[49] N. Madan, L. Zhao, N. Muralimanohar, A. Udipi, R. Balasubramonian, R. Iyer, S. Makineni, and D. Newell, "Optimizing communication and capacity in a 3D stacked reconfigurable cache hierarchy," in *15th International Conference on High-Performance Computer Architecture*, 2009.

[50] S. Franey and M. Lipasti, "Tag tables," in *21st International Symposium on High Performance Computer Architecture*, 2015.

[51] P. Tsai, N. Beckmann, and D. Sánchez, "Jenga: Software-defined cache hierarchies," in *44th International Symposium on Computer Architecture*, 2017.

[52] *Micron. RLDRAM 2 and 3 Specifications*, https://www.micron.com/products/dram/rldram-memory.

[53] Y. Sato, T. Suzuki, T. Aikawa, S. Fujioka, W. Fujieda, H. Kobayashi, H. Ikeda, T. Nagasawa, A. Funyu, Y. Fuji, K. Kawasaki, M. Yamazaki, and M. Taguchi, "Fast cycle RAM (FCRAM); a 20-ns random row access, pipe-lined operating DRAM," in *Symposium on VLSI Circuits. Digest of Technical Papers*, 1998.

[54] W. Leung, F.-C. Hsu, and M. E. Jones, "The ideal soc memory: 1T-SRAM™," in *13th International ASIC/SOC Conference*, 2000.

[55] *Tezzaron DiRAM4 3D Memory*, https://tezzaron.com/applications/diram4-3d-memory/.

[56] D. Lee, Y. Kim, V. Seshadri, J. Liu, L. Subramanian, and O. Mutlu, "Tiered-latency DRAM: A low latency and low cost DRAM architecture," in *19th International Symposium on High Performance Computer Architecture*, 2013.

[57] Y. H. Son, O. Seongil, Y. Ro, J. W. Lee, and J. H. Ahn, "Reducing memory access latency with asymmetric DRAM bank organizations," in *40th International Symposium on Computer Architecture*, 2013.

[58] M. O'Connor, N. Chatterjee, D. Lee, J. Wilson, A. Agrawal, S. W. Keckler, and W. J. Dally, "Fine-grained DRAM: energy-efficient DRAM for extreme bandwidth systems," in *50th International Symposium on Microarchitecture*, 2017.

[59] T. Zhang, K. Chen, C. Xu, G. Sun, T. Wang, and Y. Xie, "Half-DRAM: A high-bandwidth and low-power DRAM architecture from the rethinking of fine-grained activation," in *41st International Symposium on Computer Architecture*, 2014.

[60] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu, "A case for exploiting subarray-level parallelism (SALP) in DRAM," in *39th International Symposium on Computer Architecture*, 2012.

[61] K. Sudan, N. Chatterjee, D. W. Nellans, M. Awasthi, R. Balasubramonian, and A. Davis, "Micro-pages: increasing DRAM efficiency with locality-aware data placement," in *15th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2010.

[62] R. Ausavarungnirun, K. K.-W. Chang, L. Subramanian, G. H. Loh, and O. Mutlu, "Staged memory scheduling: Achieving high performance and scalability in heterogeneous systems," in *39th International Symposium on Computer Architecture*, 2012.

[63] O. Mutlu and T. Moscibroda, "Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared DRAM systems," in *35th International Symposium on Computer Architecture*, 2008.

[64] H. Hassan, G. Pekhimenko, N. Vijaykumar, V. Seshadri, D. Lee, O. Ergin, and O. Mutlu, "Chargecache: Reducing DRAM latency by exploiting row access locality," in *22nd International Symposium on High Performance Computer Architecture*, 2016.