In Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2014)

Unison Cache: A Scalable and Effective Die-Stacked DRAM Cache

Djordje Jevdjic[†] Gabriel H. Loh[‡] Cansu Kaynak[†] Babak Falsafi[†]

[†]*EcoCloud, EPFL* [‡]*Advanced Micro Devices, Inc.*

***Abstract*—Recent research advocates large die-stacked DRAM caches in manycore servers to break the memory latency and bandwidth wall. To realize their full potential, die-stacked DRAM caches necessitate low lookup latencies, high hit rates and the efficient use of off-chip bandwidth. Today's stacked DRAM cache designs fall into two categories based on the granularity at which they manage data: block-based and page-based. The state-of-the-art block-based design, called Alloy Cache, colocates a tag with each data block (e.g., 64B) in the stacked DRAM to provide fast access to data in a single DRAM access. However, such a design suffers from low hit rates due to poor temporal locality in the DRAM cache. In contrast, the state-of-the-art page-based design, called Footprint Cache, organizes the DRAM cache at page granularity (e.g., 4KB), but fetches only the blocks that will likely be touched within a page. In doing so, the Footprint Cache achieves high hit rates with moderate on-chip tag storage and reasonable lookup latency. However, multi-gigabyte stacked DRAM caches will soon be practical and needed by server applications, thereby mandating tens of MBs of tag storage even for page-based DRAM caches.**

We introduce a novel stacked-DRAM cache design, Unison Cache. Similar to Alloy Cache's approach, Unison Cache incorporates the tag metadata directly into the stacked DRAM to enable scalability to arbitrary stacked-DRAM capacities. Then, leveraging the insights from the Footprint Cache design, Unison Cache employs large, page-sized cache allocation units to achieve high hit rates and reduction in tag overheads, while predicting and fetching only the useful blocks within each page to minimize the off-chip traffic. Our evaluation using server workloads and caches of up to 8GB reveals that Unison cache improves performance by 14% compared to Alloy Cache due to its high hit rate, while outperforming the state-of-the art page-based designs that require impractical SRAM-based tags of around 50MB.

***Keywords*-caches; DRAM; 3D die stacking; memory; servers**

I. Introduction

The steadily increasing processing capabilities of multi-core and many-core processors require a commensurate increase in memory bandwidth. However, memory speeds have not kept pace with CPU performance scaling, which has led to the so-called "Memory Wall" [30]. Modern data-centric server workloads further exacerbate the pressure on the memory system, as their vast working sets cannot be captured by today's SRAM caches [7], [8], [13], [21]. However, recent advances in die-stacking technologies (i.e., 3D integration) have made it possible to integrate a sizeable amount of DRAM in the same package as the processor. Such die-stacked DRAM can provide several gigabytes of storage [22] at a bandwidth of over 100GB/s. When combined with "2.5D" silicon interposer-based integration, multiple DRAM stacks may be placed in the same package [5], further increasing the in-package DRAM capacity.

Unfortunately, even several gigabytes of die-stacked DRAM is insufficient to satisfy a high-end server's memory capacity requirements (often exceeding one hundred gigabytes for data-intensive applications). As a result, researchers have been exploring various ways to use the die-stacked DRAM as a giant last-level cache [10], [11], [19], [20], [24], [33]. There are a number of fundamental challenges that these past works have tried to address:

- **Tag overhead:** If the cache uses a conventional block size (e.g., 64B), then the storage needed to record all of the tags for, say, a 1GB DRAM cache would be 96MB-128MB (assuming a tag size of 6-8 bytes per block). With larger granularities (e.g., a 4KB page), the tag overheads are reduced by a factor of 64 (i.e., 1.5MB-2MB). This may seem reasonable for the time being, but as the technology rapidly enables multi-gigabyte stacked DRAM capacities, even page-based tags quickly consume too much SRAM to be practical. To illustrate, 8GB of stacked DRAM would need 16MB of SRAM in the best case, which is larger than today's last-level caches. Moreover, this storage drastically increases if the cache uses sub-blocking to optimize for off-chip bandwidth.
- **Hit latency:** While the stacked DRAM provides a huge increase in bandwidth compared to conventional DDR channels, the *latency* of the die-stacked DRAM is not substantially better. If a DRAM cache architecture requires accessing the stacked-DRAM or a multi-megabyte SRAM table for tag lookups, then that could add several tens of cycles to the overall cache latency, offseting any latency advantage of stacked DRAM.
- **Hit ratio:** Little temporal locality exists at this level of the cache hierarchy as any repeated accesses to the same blocks would have likely hit in the higher cache levels (e.g., L1, L2). Block-based DRAM caches, which seek to exploit temporal locality [10], [11], provide relatively low cache hit rates, reducing the efficacy of the DRAM cache.

Recent DRAM-cache proposals have successfully addressed some of these challenges, but none (to the best of our knowledge) have overcome all of them at the same

time. The recent block-based Alloy Cache (AC) design [24] provides an architecture that completely avoids any large SRAM-based tag arrays, and overall provides low latencies on cache hits. The cache is organized as direct-mapped to avoid searching for the correct way throughout the DRAM-based tags. However, these advantages come at the cost of relatively low cache hit rates, which are further penalized by the cache's direct-mapped organization, and high miss penalty. To avoid DRAM cache lookups on cache misses, AC employs a miss predictor, sending cache requests to main memory if a miss is predicted.

Footprint Cache (FC) takes a different tack [10], and uses page-sized allocation units to reduce the SRAM tag arrays to a couple of MBs. FC uses a "footprint predictor" to fetch only the relevant subset of the 64B blocks from a page; this provides high cache hit rates by exploiting spatial locality within a page, while efficiently using the scarce off-chip bandwidth by not fetching blocks that won't be used. The downside of FC is that, as discussed above, the SRAM-based tag array will not gracefully scale to larger stacked DRAM sizes and the tag array imposes additional latency to service a request.

In this work, we present *Unison Cache* (UC). UC is carefully designed to combine the best traits of both AC and FC, while avoiding their shortcomings. Tags are directly embedded in the stacked DRAM, like AC, to avoid SRAM-based tag arrays. At the same time, Footprint Cache-like large allocation units are used to exploit spatial locality, with the added benefit of reducing the fraction of the stacked DRAM's capacity that must be set aside for the embedded tags. To effectively realize such a design we leverage the following insights:

- In order reduce hit latency Alloy Cache merges ("alloys") each data block and its tag into a single unit and streams both in a single access. However, the primary latency benefit comes from breaking the serialization between the tag and data accesses. Unison Cache instead uses a single tag per page, but overlaps the tag read with the data block read. In doing so, UC achieves the same hit latency, but also allows for an effective page-based organization with DRAM-based tags.
- By leveraging spatial locality, Unison Cache achieves high hit ratios (often 90% or better). With such a high hit ratio, the miss predictor used by Alloy Cache to reduce miss penalty is not necessary, as a static "always-hit" prediction achieves similar accuracy.
- Direct-mapped organization hurts page-based designs, causing many more conflicts compared to block-based designs. However, we find that direct-mapped organization is not necessary to achieve low hit latency. To reduce the number of conflict misses Unison Cache is organized as a set-associative cache. Instead of serializing tag and data accesses or fetching all the ways in parallel, Unison Cache relies on simple and highly accurate way prediction, increasing neither the cache hit latency nor the amount of transferred data.

	AC	FC	UC
No SRAM tag overhead	✓	✗	✓
Low hit latency	✓	✗	✓
High hit rate	✗	✓	✓
High effective capacity	✗	✓	✓
Scalability	✓	✗	✓

Table I. Comparison of Alloy Cache (AC), Footprint Cache (FC), and Unison Cache (UC).

The end result is that by carefully leveraging these insights, the proposed Unison Cache is able to outperform both Alloy Cache and Footprint Cache designs, approaching the performance of an ideal "latency-optimized" DRAM cache (100% hit rate, 0-cycle tag access). At the same time, Unison Cache does not require SRAM-based tag arrays, which allows Unison Cache to easily scale up to cache sizes of many gigabytes needed by server applications. A summary of the key features of Unison Cache, as well as the prior art, is listed in Table I.

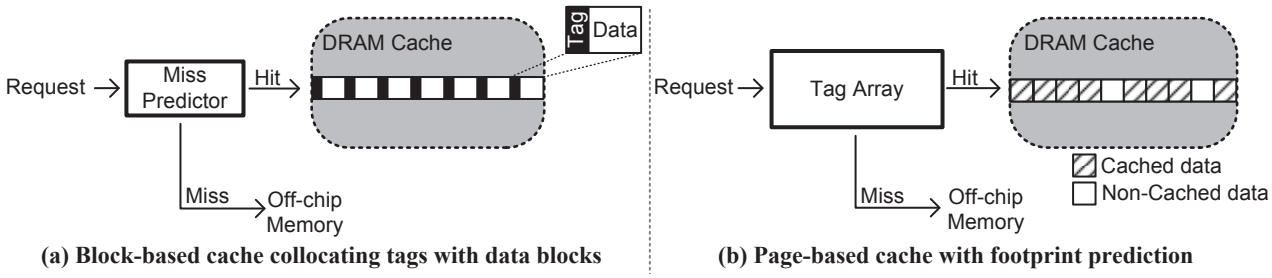
II. Background and Motivation

Die-stacked DRAM has been advocated as a promising technology to break the memory bandwidth and latency wall. It delivers several times more bandwidth compared to off-chip memory due to dense on-chip TSV buses, as well as lower access latency. Unfortunately, the feasible die-stacked DRAM capacities lag far behind the working set sizes of data-intensive emerging server applications [10], [20]. This capacity constraint precludes the use of die-stacked DRAM as main memory. Hence, most proposals advocate employing die-stacked DRAM as a cache to filter out accesses to off-chip main memory [10], [11], [20], [24].

While existing die-stacked DRAM cache proposals significantly reduce off-chip memory traffic, they fall short of achieving one of the other two major design goals, namely low hit latency and high hit rate. In the rest of this section, we divide the existing designs into two classes and explain why each class fails to achieve either low hit latency or high hit rate.

A. Block-Based Caches

Similar to conventional on-chip caches, block-based DRAM caches seek to exploit temporal locality and maximize storage efficiency by only storing requested cache blocks. The use of larger cache lines could result in significant data overfetch [10], [11], penalizing the scarce off-chip bandwidth. Unfortunately, at this level of the memory hierarchy, server workloads do not exhibit as much temporal locality as they do at higher levels (i.e., most temporal locality has already been filtered out by the L1 and L2 caches) [7]. Furthermore, server workloads typically do not have small, well-defined working sets due to their enormous



(a) Block-based cache collocating tags with data blocks **(b) Page-based cache with footprint prediction**

Figure 1. Overview of the state-of-the-art (a) block-based and (b) page-based DRAM cache designs.

memory footprints and complex access patterns [7]. As a result, block-based DRAM cache designs can exhibit excessive miss rates (e.g., as high as 70% [10]) on server workloads.

Because the data is managed at a small granularity, block-based DRAM caches require several tens or hundreds of MBs for tags. Such a large volume of tag metadata rules out a conventional on-chip, SRAM-based tag array, and forces the tags to be placed in the stacked DRAM along with the data blocks [19], [20], [24]. However, storing tags directly in the DRAM cache can potentially require two DRAM accesses per cache lookup (one for the tag and another for data), thereby doubling the effective DRAM cache access latency in the worst case.

To improve the effective DRAM cache access latency, Loh and Hill proposed organizing each DRAM row as a cache set and colocating all the ways of a set and their corresponding tags in the same DRAM row [20]. On a DRAM cache request, first, the tags in the beginning of a row are accessed for tag comparison. Upon a tag match, the request for the corresponding data block is issued separately, causing serialization of the tag lookup and data access. However, the accesses to tags and data are scheduled in a way that ensures a row buffer hit for the data block after the tag access.

Even though this scheduling optimization reduces the DRAM cache hit latency by exploiting row buffer locality, cache hits suffer from tag lookup and data fetch serialization, while cache misses suffer from high miss latencies due to the tag lookup in the DRAM cache prior to issuing the request to the off-chip main memory. To reduce the DRAM cache miss latency, Loh and Hill propose employing an on-chip SRAM "MissMap" to maintain cache block presence information. This way, DRAM cache misses can bypass the high-latency lookups and an off-chip memory request can be issued directly. Unfortunately, this comes at the cost of further increasing the DRAM cache hit latency by adding the MissMap access to the cache lookup path, and the multi-MB MissMap itself will not scale up to support multi-GB DRAM caches.

The state-of-the-art block-based approach, Alloy Cache (AC) [24], organizes the DRAM cache as direct-mapped, further reducing the already low hit rate, but compensating for this by greatly improving the cache access latency. AC merges (or "alloys") each single data block with the corresponding tag in unified tag-and-data units (TAD), as shown in Figure 1(a). The direct-mapped organization eliminates the need to search for the correct way in the DRAM, allowing AC to stream out a TAD in a single read, thereby breaking the tag-then-data serialization on cache hits and thus significantly reducing the lookup latency compared to Loh and Hill's design.

To minimize the DRAM cache miss latency, AC employs a simple low-latency miss predictor, moving the DRAM cache tag lookup off the critical path when the predictor correctly predicts misses. However, when a cache hit is predicted to be a miss, AC creates extra off-chip traffic by sending an unnecessary fetch request for a block that is already in the cache. When a cache miss is predicted to be a hit, the actual off-chip memory request is delayed by the tag lookup in the cache.

In summary, the best existing block-based DRAM cache designs are able to effectively mitigate tag-lookup latencies. However, they fail to provide sufficiently high hit rates for server workloads.

B. Page-Based Caches

Page-based caches allocate and fetch data at a coarse granularity (e.g., 1-8KB pages) to maximize hit rates by exploiting spatial locality. While server workloads do not exhibit much temporal locality at lower levels [1] of the memory hierarchy, they still exhibit significant spatial locality. Spatial locality is abundant at lower levels of the hierarchy due to longer residency of data. For example, a 2KB page would typically stay in a 1GB cache for hundreds of milliseconds, leaving much more time for different data pieces to be accessed within the page compared to, say, an 8MB cache. The CPU cores see this phenomenon as high spatial locality [10]. As a result, page-based caches can greatly increase cache hit rates when compared to block-based designs [9], [10], [11]. Unfortunately, many pages contain data that are not accessed prior to the page's eviction from the cache, causing

[1] We use terms higher and lower levels of the memory hierarchy to refer to the levels closer to and further away from the core, respectively.

significant waste of precious off-chip memory bandwidth. In extreme cases, page-based designs may increase the off-chip traffic by an order of magnitude compared to a baseline design without any DRAM cache [10], [11].

To prevent wasting off-chip memory bandwidth in page-based caches, the state-of-the-art page-based DRAM cache design, Footprint Cache (FC) [10], organizes the DRAM cache in pages, but fetches only data that are going to be touched during a page's residency in the DRAM cache (the page's footprint), as Figure 1(b) depicts. FC relies on a simple, but highly-accurate spatial correlation predictor [27] to identify the footprint of a page (i.e., the set of blocks within a page that are demanded by the processor during the page's residency in the cache). The footprint is predicted at page allocation time based on the instruction that triggers the first access to the missing page and the relative position of that access within the page (i.e., block offset). When a page is allocated in the cache, the triggering (PC, offset) pair is stored in the tag array. Upon the page's eviction, its actual footprint is associated with the triggering (PC, offset) pair and stored in an SRAM-based history table for later prediction of footprints of other pages that are traversed by the same code. By fetching only the useful data in each DRAM cache page, FC eliminates the off-chip bandwidth waste stemming from fetching untouched data, while preserving the high hit rates attributed to the high spatial locality within a page.

Page-based designs result in lower tag storage overheads compared to block-based designs, which potentially allows for accommodating the tags in on-chip SRAM tables for faster tag lookups. For example, a 512MB Footprint Cache requires around 3MB of tags, which, while not trivially small, is still feasible to be implemented on-chip. However, the improvements in die-stacked DRAM technology have already pushed the feasible DRAM capacities up to several GBs (e.g., Micron's 4GB Hybrid Memory Cube [22]). Device scaling and the continuing increase in the number of layers that can be stacked promise for larger die-stacked DRAM cache capacities. The introduction of 2.5D [4] or silicon-interposer-based integration [26] could enable the incorporation of multiple DRAM stacks, further increasing the potential sizes of future DRAM caches. Unfortunately, the expansion of DRAM cache capacities causes the tag storage size and the associated tag lookup latency to be a problem for page-based caches, as the tag storage size quickly reaches several tens of MBs, exceeding what can be economically built using conventional on-chip SRAM (and even if die area were not a constraint, the latency of such a large SRAM would be significant).

In conclusion, the best existing page-based DRAM caches outperform block-based designs due to their high hit rates without increasing the off-chip traffic. However, continued increases in die-stacked DRAM capacity forces a redesign of the tag architecture for page-based DRAM caches.

III. Getting the Best of Block- and Page-Based Caches

In this section we examine the approaches to getting the best properties of block-based and page-based designs, which include: scalable DRAM-based solution for the tag array, high hit rates, low off-chip traffic, and low cache-hit and cache-miss latencies. We present our design, called Unison Cache, and also discuss alternative ideas in Section III-B.

Unison Cache employs a page-based DRAM cache organization, but leverages a footprint predictor to only fetch the useful blocks within each page [10]. The page tags in Unison Cache are embedded in the stacked DRAM, and each tag maintains the state of its blocks using bit vectors as well as the footprint prediction metadata to facilitate learning the footprints of pages during their residency in the cache.

A. Unison Cache

The first key insight that leads to an effective design is that while Alloy Cache's tag-and-data (TAD) colocation provides the ability to stream both in a single DRAM access, the primary latency benefit of such an approach comes from breaking the serialization between tag and data accesses rather than from the tag-and-data colocation itself. Unison Cache physically separates tags and data blocks within the DRAM row and uses a single tag per page, as shown in Figure 2, but the read operations for both the tag and the individual data block can be overlapped as they are not dependent on each other. While this may require two separate back-to-back read commands to the same row, the reads are not serialized and therefore the latency ends up being the same as for reading a TAD. Maintaining a single tag per page also allows footprint tracking to be easily implemented and reduces tag storage. A data block and the corresponding page tag are always read in parallel (i.e., the tags and data work "in unison"). The second observation is that by leveraging spatial locality, Unison Cache (like Footprint Cache) can achieve very high hit rates (often 90% or better). At this point, we can dispense with Alloy Cache's hit predictor, as a static "always-hit" prediction would achieve accuracy similar to a dynamic hit prediction. Finally, to avoid the price of direct-mapped organization, which is particularly high for page-based designs, Unison Cache is organized as set-associative, colocating all the pages of a set in the same DRAM row. However, instead of serializing tag and data accesses or fetching all the ways at the same time, Unison Cache relies on highly accurate way prediction, increasing neither the cache hit latency nor the amount of transferred data.

In the rest of this section, we describe the Unison Cache design and its operation in detail.

1) Footprint Prediction: Unison Cache learns and fetches page footprints to avoid off-chip bandwidth waste. The footprint of a page comprises all the blocks that are touched between the first access to the page, which happens

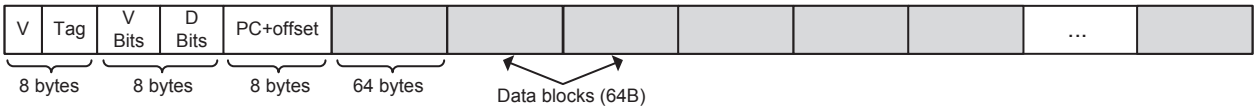


Figure 2. DRAM row content in Unison Cache (not drawn to scale).

upon an access to a page that is not in the cache, and the eviction of the page.

Our design leverages FC's footprint predictor [10]. The predictor relies on the correlation between the code and page footprints. This correlation stems from repeated calls to a limited set of functions to access large amounts of data, especially in well-structured object-oriented server software. Repetitive calls to these functions result in repetitive data access patterns (i.e., page footprints) that can be exploited to predict future data accesses upon subsequent calls to the same function. The correlation between code and data access patterns has been heavily exploited for data prefetching [2], [15], [27] and filtering of unused data [10], [14], [16], [32].

The instruction that accesses the first data block in a page has been shown to accurately predict footprints of pages that are later accessed by the same instruction [10], [27]. To account for different alignments of data structure instances in different memory pages, there is also a need to combine the instruction information (i.e., *PC*) with the distance of the first accessed block from the beginning of the page (i.e., *offset*) [10], [27]. Hence, the footprint predictor predicts page footprints based on the (PC, offset) pair that initiates the first access to a page, the *trigger access*. Each footprint prediction table entry consists of a (PC, offset) pair and a bit vector to indicate the page footprint correlated with that pair.

2) Learning Footprints: To facilitate footprint learning, each page in Unison Cache is augmented with a (PC, offset) pair that corresponds to the first access to the page (triggering miss). This information is inserted into a DRAM row along with the data when the page is allocated (Figure 2). During the page's residency in the cache, each access to a block within a page updates the corresponding valid/dirty bits in the bit vector that belongs to the page's tag to indicate that the block had been demanded. To determine the footprint of a page it is necessary to make a distinction between fetched blocks that are actually demanded by the CPU at some point and those that are not (overfetched blocks). To enable such a distinction without extra storage, we modify the semantics of the existing valid and dirty bits and use a different block state encoding scheme, as was done in the Footprint Cache study [10]. Upon eviction, the triggering (PC, offset) pair and the footprint bit vector (constructed based on valid and dirty bits) of the evicted page are used to update the footprint prediction table, which associates a footprint to each (PC, offset) pair.

3) Fetching Footprints: When the requested page is not found in the cache, the footprint prediction table is queried for the (PC, offset) pair that triggered the cache miss. If a match is found, the corresponding footprint is used to determine what blocks will be fetched. In the case of a miss to a block whose page is already allocated in the cache (i.e., footprint *underprediction*), there is no need to initiate footprint prediction and new page fetch. Instead, only a single fetch request for the missing block is sent to memory. However, when the page is evicted, the footprint of the page will indicate that the block was touched during the page's residency and the footprint prediction table is updated accordingly to avoid future underpredictions for the same (PC, offset) pair. Likewise, the footprint prediction might fetch blocks that are not touched during a page's residency in the DRAM cache (i.e., *overpredictions*). Similar to underpredictions, overpredictions are also propagated to the footprint prediction table when a page is evicted to avoid future overpredictions.

4) Singleton Prediction: Prior work showed that a significant fraction of page footprints consists of only a single block [10], [27]. Such pages are called *singletons*. Singleton pages reduce the effective DRAM cache capacity because they allocating space for an entire page, but accommodate only a single block. Hence, Unison Cache does not allocate a page in the cache if the footprint prediction table predicts the page to be a singleton. The missing block is fetched from memory and simply forwarded to the requestor. However, as singleton pages are not allocated in the cache, it is not possible to correct footprint mispredictions (corrections happen upon page evictions). To track the singleton pages that might become non-singleton later, Unison Cache employs a small singleton table as in Footprint Cache [10].

5) Associativity: Alloy Cache uses direct-mapped organization to quickly locate the requested block in the cache if it is present, without searching through the DRAM tags to find the correct way. Unison Cache inherits the same mechanism to quickly locate the requested page. However, UC is page-based and direct-mapped page-based caches are highly vulnerable to cache conflicts. While zero associativity does not severely affect the hit ratio of block-based DRAM cache designs due to the large number of sets [24], it has a huge impact on page-based designs. According to our analytical model, which we omit for space reasons, for a 1GB cache and 2KB pages, the probability of conflicts increases by a factor of $\sim$500 in the worst case compared to a block-based direct-mapped cache of the same size.

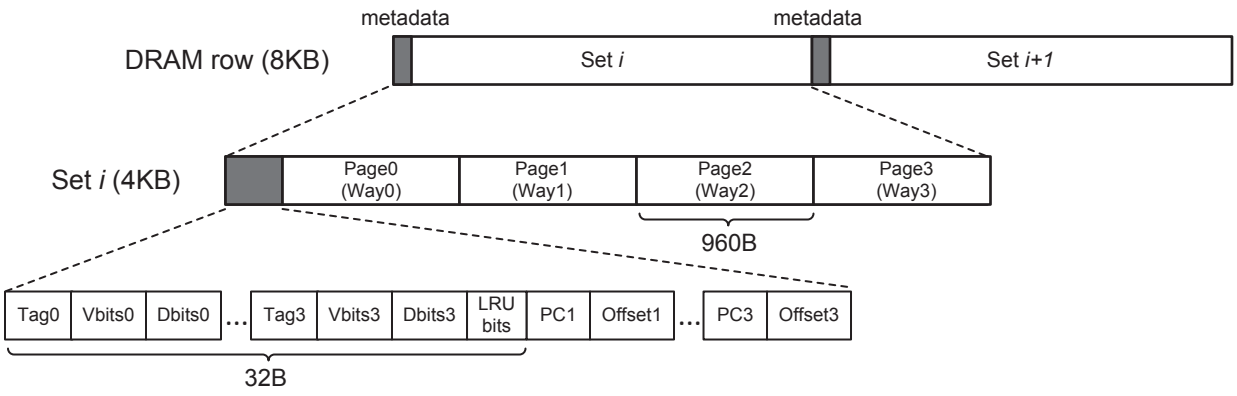


Figure 3. DRAM row organization in the Unison Cache design.

The reason lies in false conflicts introduced by the page-organization. Namely, in block-based designs will be in conflict if and only if they belong to the same set and if they are both requested at the same time. On the contrary, in page-based designs two blocks belonging to the same set will be in conflict not only if the two blocks themselves are needed at the same time, but also if *any* two blocks from the pages they belong to are needed at the same time.[2] The probability of conflicts thus grows quadratically with the page size and creates a severe problem despite the large cache size.

To reduce page conflicts and achieve higher hit rates, we organize Unison Cache as a set-associative page-based cache. We do not, however, go back to tags-then-data serialization, as it would be highly inefficient; nor do we fetch several ways in parallel, as it would create vast data overfetch and eventually lead to serialization of the fetched ways on the bus, significantly increasing the latency [24]. Instead, we use a simple way predictor that yields an accuracy of over 95% and use this information to fetch the correct way from a DRAM row. We describe the details below.

6) DRAM Row Organization and Operations: So far we assumed, for simplicity, that the size of a cache page equals to the DRAM row size. In reality, DRAM rows are typically larger than the desirable page size. For the sake of generality, let's assume that the cache is four-way associative, the page size is 1KB, and the DRAM row size is 8KB. In this example, each set is 4KB, and one DRAM row accommodates two whole sets, as shown in Figure 3. One of the two sets (half of a DRAM row) is shown in more detail with its four pages. The metadata of each page (valid bit, page tag, valid and dirty bit vectors, replacement policy bits, and (PC, offset) is maintained in the beginning of the row, such that the metadata required to determine the presence of a block is stored first (page tags and bit vectors), whereas (PC, offset) pairs and other metadata for all pages are stored after all the tag information. This placement is chosen for efficiency reasons, so that all the tags from a set can be read together in a single access. For this particular configuration, the total size of the tag metadata for the four pages is 32B, which can be transferred in two bursts over a 128-bit TSV bus, corresponding to one bus cycle or two CPU cycles in the system we evaluate.[3] The metadata read command is immediately followed by the read command for the data block whose position in the DRAM row is determined by the page offset and by the predicted way; the two read operations are overlapped.

The two cycles that represent an overhead to read the tags leave enough room for way prediction, which is done by the DRAM controller and is not on the critical path. We use a simple way predictor, which is a 2-bit array directly indexed by the 12-bit XOR hash of the page address (16-bit XOR for caches above 4GB). Prior work on way prediction has found that address-based way predictors are the most accurate way predictors for L1 caches [1], [23]. However, such predictors are not an option for L1 caches because the actual address is not known at the time when the prediction has to be made for L1 blocks. We do not have such a constraint here. While the accuracy of address-based way predictors is found to be around 85% for individual blocks [1], [23], our way predictor achieves much higher accuracy (∼95%), because it operates at the page level. The abundant spatial locality leads to repeated accesses to the same page; subsequent accesses to the same page result in correct predictions. The predictors page-based operation also reduces its storage overhead to 1KB (16KB for caches above 4GB). Because all the ways of a set reside in the same DRAM row, way mispredictions, apart from being rare, are also relatively cheap. Due to the DRAM row organization shown in Figure 3, the correct way in case of mispredictions is likely to be found in the row buffer, thus the uncommon case is not severely penalized.

The (PC, offset) information is stored in the DRAM row

[2]This is analogous to the false sharing problem.

[3]For systems with more than 1TB of memory (more than 40 physical address bits), three bursts would be needed to transfer ∼48B of tags.

upon the page's allocation and it is read only upon its evictions. This information is then used to update an SRAM-based footprint prediction table with the actual footprint of the evicted page, constructed from the page's bit vectors.

In case of cache misses, it is easy to distinguish between triggering misses (the requested page is not in the cache) or regular misses resulting from incorrect footprint prediction (i.e., underprediction), because the page tags for all the ways and the block presence bit vectors are stored in one place. The (PC, offset) information is also stored in the DRAM row upon page allocation and it is read only upon its evictions. This information is then used to update an SRAM-based footprint prediction table along with the actual footprint of the evicted page.

7) Address mapping: Integrating any kind of metadata into DRAM causes alignment problems, because a fraction of each DRAM row must be reserved for the metadata. In the case of Unison Cache, embedding the tag array into DRAM results in the page size being a non-power-of-two number (e.g., the pages sizes are 960B or 1984B, containing 15 or 31 64-byte blocks, respectively). Such page sizes require specialized logic for address manipulation instead of simply relying on address bits. Designing a general-purpose modulo-computing unit for such address manipulation would incur high area and latency overheads. However, here we compute modulo with respect to a constant in a specific form (2^n-1), which can be computed with several adders using residue arithmetic [24]. We estimate the calculation to take two cycles and only a few hundred gates, as in AC and it can be overlapped with last-level SRAM cache accesses.

B. Alternative Approaches

In this section we discuss alternative approaches to getting the best of block- and page-based designs. Looking at the two ends of the spectrum, there are two seemingly obvious ways to combine the two designs.

1) Block-based cache with footprint prediction: One naïve way of combining the two state-of-the-art block- and page-based designs is to start with Alloy Cache's direct-mapped, block-based organization with the tags colocated with data blocks, and then apply footprint prediction as a prefetcher in attempt to exploit spatial locality. Since the footprint prediction mechanism learns and predicts the blocks within pages, such a design would require grouping a number of neighboring blocks into a logical page and fetching and evicting them at the same time. Unlike existing page-based DRAM cache proposals, such a design could theoretically allow multiple pages to co-exist in the same DRAM row as depicted in Figure 4(a). Unfortunately, multiple pages (shown as different shades of gray in the figure) could only co-exist in the same row if their footprints are completely disjoint; an overlap would cause a conflict and require the other page (i.e., its current footprint) to be prematurely evicted, as allocations and evictions happen at page granularity.

Such a design would introduce major problems due to the mismatch between the cache organization and the footprint prediction mechanism. First, there is no fast lookup mechanism to indicate the presence of a page in the cache. In case of a miss, it is not possible to easily determine whether other blocks of the same page are cached or not. Thus, to identify if a cache miss is a trigger miss (the first miss to a page that initiates footprint prediction and fetching the page's footprint from off-chip memory), the entire DRAM row of the missing cache block needs to be scanned to determine if any block from the same logical page is present in the cache, because the block presence information is spread out over the entire DRAM row. Not finding any block within the page would indicate that the current miss is a trigger access. Such a scan is also needed to identify the footprint of the page that will be evicted as a result of the miss, and update the footprint predictor state accordingly. Unfortunately, scanning all tags in a DRAM row upon each cache miss and block eviction would significantly reduce DRAM cache availability, waste energy, and increase miss latency. Also note that for each page in the cache, we must keep its (PC, offset) pair that caused the initial miss, which are used to update the footprint predictor state upon eviction as in FC [10]. It is not straightforward to augment each DRAM row with the metadata corresponding to each of the variable number of logical pages it contains.

2) Page-based cache with tagged blocks: Another naïve way of combining the two designs is to start with FC and preserve its page organization, but augment each block in DRAM with its tag in order to stream tag and data blocks together in a single DRAM access, as in Alloy Cache. A DRAM row in such an organization is shown in Figure 4(b). As each DRAM row now accommodates a single page, upon a DRAM cache miss it is possible to determine whether or not the miss is the first access to the page that initiates the missing page's footprint fetch. However, this requires writing the correct page tag and resetting the valid bit even for blocks that are not fetched upon page insertions, which means an extra DRAM write for each block that does not belong to the footprint of a newly fetched page. Furthermore, upon page evictions following a miss, there is no simple lookup mechanism to identify the footprint of the evicted page; the entire DRAM row would need to be scanned to determine the valid blocks within the page. In contrast to the previous design point, the (PC, offset) pair that triggered a page access could be stored at a predetermined position in the corresponding DRAM row and later used to update the footprint prediction table with the correct footprint.

In both naïve design points each data block is colocated with its corresponding tag to minimize latency, leading to a vast amount of replication. The tag replication wastes around 1/8 of the total cache capacity and further reduces

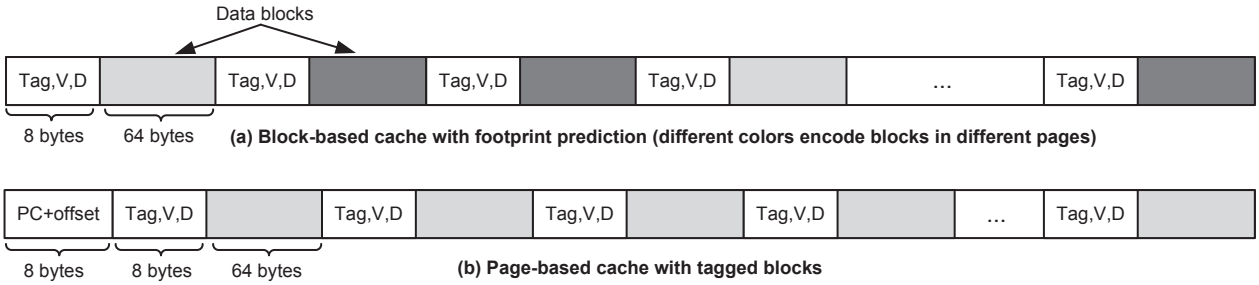


Figure 4. DRAM row organizations for (a) block-based cache with footprint prediction, and (b) page-based cache with tagged blocks.

	Alloy Cache	Footprint Cache	Unison Cache
Cache Miss Rate	Medium-High	**Low**	**Low**
Hit Latency	**Predictor + DRAM TAD Read**	SRAM Tag + DRAM Data Read	**Overlapped DRAM Tag + Data Reads**
Miss Latency	**Predictor Lookup**	SRAM Tag Lookup	DRAM Tag Lookup
Associativity	Direct-mapped	**32-way**	4-way (two pages)
64B Blocks per 8KB Row	112	**128**	120-124
SRAM Tag Array @ 8GB	—	∼48MB	—
In-DRAM Tag Size @ 8GB	1GB (12.5% of DRAM)	—	256-512MB (3.1-6.2% of DRAM)
Miss-Predictor Size	96B per core, 1.5KB total	—	—
Way Predictor	—	—	1-16KB
Footprint History Table	—	144KB	144KB
Singleton Table	—	3KB	3KB

Table II. Comparison of key characteristics of different DRAM cache schemes.

the hit ratio. Furthermore, the footprint predictor is partially integrated into DRAM-based tags, which contain various metadata needed for prediction, most importantly the block presence information. Spreading this information throughout a DRAM row causes, as discussed, a variety of problems related to footprint tracking, detecting triggering misses, page evictions, and unnecessary DRAM row scans and writes. Unison Cache avoids these problems by centralizing the tag information for all data blocks within a page and accessing this information in parallel with data blocks to avoid any latency penalty.

C. Summary and Comparisons

Unison Cache leverages insights and ideas from both the Alloy Cache and the Footprint Cache, but synthesizes and extends them in unique ways to "get the best of both worlds" while side-stepping their pitfalls. Given the many interacting and inter-dependent components, Table II provides a summary of the key characteristics of the different DRAM cache design approaches to more easily distinguish the contributions and strengths of Unison Cache.

Unison Cache maintains the low miss rate of Footprint Cache (FC), the low hit latency of Alloy Cache (AC), avoids the impractically large SRAM tag arrays of FC, has lower embedded DRAM tag overheads than AC, and has no miss predictor like AC. Assuming an 8GB die-stacked DRAM and 2KB pages, FC would require about 50MB for its SRAM tag array.

On a cache miss, AC has the best latency (assuming the hit-predictor was correct), but in practice both FC and Unison Cache have sufficiently high hit rates that the additional tag-lookup latency for misses has a much smaller impact. FC and Unison Cache often have hit rates in excess of 90%, which is functionally equivalent to having a static hit-predictor with a 90% accuracy.

FC has by far the highest associativity. However, the additional associativity beyond four ways provides rapidly diminishing returns, as discussed in Section V. This is why Unison Cache's comparatively lower 4-way set associativity is not a significant constraint.

Like FC, Unison Cache requires some on-chip SRAM resources to implement the footprint predictor structures, but these are fixed sizes and *do not* grow with increasing stacked DRAM capacities.

IV. Methodology

A. Simulation Infrastructure

We evaluate Unison Cache through a combination of trace-driven and cycle-level simulation of a 16-core CMP running server workloads. We use the Flexus [29] full-system multiprocessor simulator, which extends the Virtutech Simics functional simulator with OoO cores, on-chip network, and memory hierarchy and models the SPARC v9 ISA. We use DRAMSim2 [25] integrated into Flexus to model both the die-stacked DRAM and the off-chip DRAM, with the parameters listed in Table III.

The trace-driven experiments are based on the memory traces that consist of 30 billion instructions per core, two

CMP Organization	16-core Scale-Out Processor pod
Core	ARM Cortex-A15-like, 3-way OoO @3GHz
L1-I/D caches	64KB, split, 64B blocks 2-cycle load-to-use latency
L2 cache per pod	4MB, unified, 16-way, 64B blocks, 4 banks, 13-cycle hit latency
Interconnect	16x4 crossbar
Off-chip DRAM	16-32GB, one DDR3-1600 (800MHz) channel 8 banks per rank, 8KB row buffer
Stacked DRAM	DDR-like interface (1.6GHz) 4 channels, 8 banks/rank, 8KB row buffer, 128-bit bus width
t_{CAS}-t_{RCD}-t_{RP}-t_{RAS} t_{RC}-t_{WR}-t_{WTR}-t_{RTP} t_{RRD}-t_{FAW}	11-11-11-28 39-12-6-6 5-24

Table III. Architectural system parameters.

Cache size (B)	128M	256M	512M	1G	2G	4G	8G
Tags (MB)	0.8	1.58	3.12	6.2	12.5	25	50
Latency (cycles)	6	9	11	16	25	36	48

Table IV. Footprint Cache parameters.

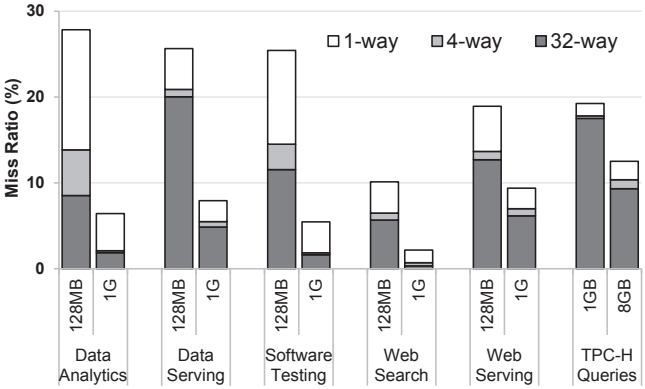


Figure 5. Unison Cache's miss ratio as a function of associativity.

thirds of which are used for cache warm-up. We evaluate performance through a set of cycle-level experiments, leveraging the SimFlex [29], [31] multiprocessor sampling methodology for server workloads. Our samples are collected over 15 seconds of workload execution. For each measurement point, the cycle-level simulation starts from checkpoints with warmed up architectural state (i.e., caches and branch predictors) and runs for 800K cycles (2M for Data Serving) to warm up the queues and the interconnect state. Then, we collect measurements for the subsequent 400K cycles of the cycle-level simulation. To measure performance, we use the ratio of the number of user instructions to the total number of cycles (including the cycles spent executing the operating system code), as this metric has been shown to accurately reflect overall server throughput [29]. Performance measurements are computed with an average error of less than 2% at a 95% confidence level.

B. Baseline System Configuration

Our baseline processor is a 16-core CMP design based on the Scale-Out Processor design methodology [21], which seeks to maximize throughput per die area. The chip features a modestly sized last-level cache to capture the instruction working set and shared OS data, which are independent of the core count, and dedicates the rest of the die-area to the cores to maximize throughput. The architectural features are listed in Table III.

C. DRAM Cache Organizations

1) Unison Cache: The evaluated design is organized as a four-way set associative cache. Each DRAM row accommodates two sets, each of which contains four pages. Each page contains 15 blocks (960B), and the whole DRAM row accommodates 120 data blocks. We also evaluate a direct-mapped organization of Unison Cache as well as organizations with 1984B pages. The parameters for footprint prediction are taken from the original Footprint Cache design [10].

2) Footprint Cache: We evaluate the original design with 2KB pages, which is found to be the sweet spot between the accuracy and tag storage overhead [10]. The 8KB DRAM row can accommodate four pages with 128 data blocks. While 1KB pages are a better match for Unison Cache, Footprint Cache cannot afford that page size as the already high SRAM-based tag storage would double. The aggregate size of the tag storage for various cache sizes is listed in Table IV along with the conservatively estimated latencies. Note that for larger cache sizes Footprint Cache's tag array grows up to ∼50MB, which cannot even fit alone in the area of today's chips, but we evaluate these hypothetical designs as reference points.

3) Alloy Cache: The 8KB row buffer is able to accommodate 112 data blocks. Alloy Cache also employs a miss predictor with a one-cycle latency to bypass the DRAM cache lookup in case of a DRAM cache miss.

D. Workloads

As a representative set of emerging scale-out server applications that are highly data-intensive and exhibit abundant request-level parallelism, we use the CloudSuite [3] workloads, including Data Analytics, Data Serving, Software Testing, Web Search, and Web Serving [7]. To evaluate multi-gigabyte cache designs, we use a set of analytic queries from the industrial TPC-H benchmark (referred to as TPC-H), running on a modern column-store database engine, MonetDB [12]. While the datasets of other workloads are scaled from hundreds of gigabytes down to 5-20GB (depending on the workload) to allow for practical full-system simulation, the TPC-H dataset is unchanged and exceeds 100GB.

		Data Analytics	Data Serving	Software Testing	Web Search	Web Serving	TPC-H Queries	Average Value
Alloy Cache	MP Accuracy (%)	96.4	90.0	93.2	97.2	91.8	89.0	92.3
	MP Overfetch (%)	7.3	6.4	16.2	13.5	7.9	1.9	8.7
Footprint Cache	FP Accuracy (%)	92.4	97.7	81.5	98.6	92.3	93.8	92.7
	FP Overfetch (%)	9.2	4.0	24.7	1.6	9.0	6.18	9.1
Unison Cache - 960B	FP Accuracy (%)	93.1	97.1	84.2	95.5	89.8	84.0	90.6
	FP Overfetch (%)	9.0	3.7	20.6	3.2	12.8	10.7	10
	WP Accuracy (%)	89.6	90.6	92.4	96.6	94.6	95.9	93.3
Unison Cache - 1984B	FP Accuracy (%)	90.2	95.7	78.2	94.4	83.4	79.9	87.0
	FP Overfetch (%)	11.5	5.4	26.8	4.4	18.9	15.4	13.0
	WP Accuracy (%)	91.1	93.9	96.2	98.1	96.9	96.8	95.5

Table V. Accuracy of various predictors: Miss Predictor (MP) in Alloy Cache, and Footprint Predictor (FP) in Footprint Cache and Unison Cache, and Way Predictor (WP) in Unison Cache for a 1GB cache (8GB for TPC-H queries).

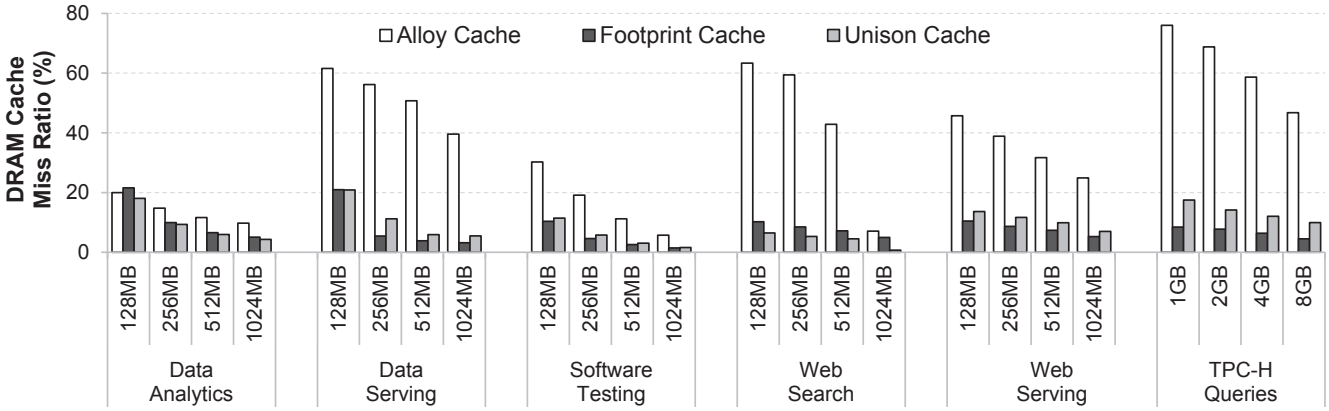


Figure 6. Miss ratio comparison of Alloy Cache, Footprint Cache, and Unison Cache.

V. Evaluation

A. Predictor Accuracy

The three designs we evaluate in this paper rely on various predictors to predict if an access is a hit or miss, to predict page footprints, or to predict the correct way in a set-associative cache. Table V summarizes the effectiveness of these predictors as well as the extra off-chip traffic generated by some of the predictors due to mispredictions, assuming a 1GB cache (8GB for TPC-H queries). We observed similar trends for other cache sizes, so we do not show the results for them due to space constraints. For Unison Cache (UC), we show two design points: with 960B and 1984B pages, both 4-way associative. For Alloy Cache (AC), we show the accuracy of the miss predictor—the fraction of misses correctly identified as such. Misses that are wrongly predicted as hits increase miss latency. AC's miss predictor is highly effective achieving over 90% accuracy on our server workloads. The hits that are wrongly identified as misses and thus cause unnecessary off-chip traffic are also shown and are not significant.

For Footprint Cache (FC) and UC, we show the footprint predictor's accuracy—the fraction of a page's footprint that is correctly predicted. We note that this metric is not comparable to AC's accuracy metric. The difference in accuracy for FC and UC stems from the differences in associativity and page size. For most of the workloads, UC's accuracy match the accuracy of FC. We also note that the UC organization with 960B pages on average provides better prediction accuracy compared to the 1984B organization, which is what the FC study also concluded [10]. While FC cannot afford this granularity because of its SRAM-based tag array, UC keeps tags in DRAM and is not restricted to large page sizes.

We also show the overfetch ratios of the two predictors to determine the extra off-chip traffic they generate. AC's miss predictor causes overfetch when it incorrectly predicts a DRAM cache hit to be a miss. Footprint predictor causes overfetch when it fetches blocks that are not accessed prior to a page's eviction. It is important to note that all three designs are highly bandwidth-efficient with small overfetch rates ($\sim$10% on average), which are offset by the benefits their predictors provide.

B. Miss Ratio

As explained in Section III, UC increases the associativity to four by adding only two CPU cycles to the hit latency, which is negligible compared to the $\sim$60 cycles it takes to

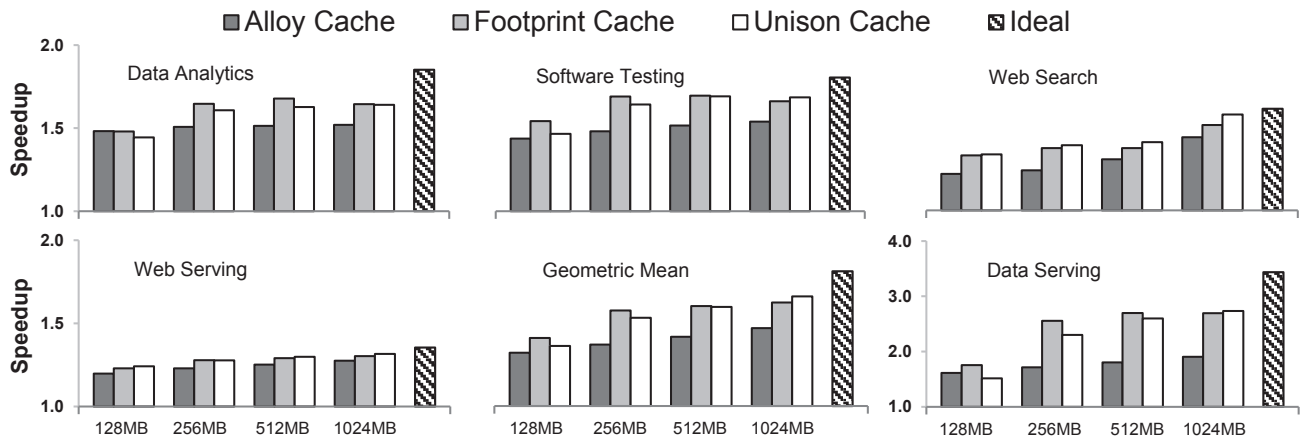


Figure 7. Performance comparison of Alloy, Footprint, and Unison Caches. Note the difference in scale for Data Serving.

access DRAM, and without causing data overfetch. Figure 5 shows the miss ratio for the UC organization with 960B pages while varying the cache associativity, for both large and small cache sizes. The miss ratios are plotted in a stacked fashion. For example, the dark gray bars show the miss ratios for a 32-way cache, while the sum of dark and light grey bars shows the miss ratios for the 4-way organization. The total height corresponds to the direct-mapped organization. We see that the four-way organization provides a sizable reduction in miss ratio, sometimes by a factor larger than two compared to the direct-mapped organization (the reduction is captured by the white bar). We note that beyond four ways, there is no significant reduction in the hit ratio to compensate for the increased tag lookup latency and reduced accuracy of the way predictor.

Way prediction and associativity have orthogonal effect. While reasonably small associativity halves the miss ratio (Figure 5), way prediction enables an effective implementation of associativity by eliminating the latency and bandwidth overheads. In our case, for a 4-way associative cache, way prediction reduces the latency by 12 cycles (needed to transfer extra ways, 20% of hit latency) and reduces the hit traffic by 4x, as all the ways would otherwise have to be fetched in parallel.

We further compare the three designs with respect to their miss ratios in Figure 6 for a range of DRAM cache sizes. As expected, AC has by far the highest miss ratio due to low temporal locality. The exception is Data Analytics, a Map-Reduce workload that exhibits the lowest spatial locality due to its pointer-intensive nature caused by frequent hash table lookups. For this workload, the differences in miss ratio between the designs are less pronounced.

FC and UC, on the other hand, significantly reduce the cache miss ratio by exploiting spatial locality and fetching whole page footprints. The small differences between the miss ratios of FC and UC stem from different page sizes used in the two designs (2KB and 1KB, respectively), the difference in associativity, and a slight difference in the effective cache capacity. Because of the larger page size, FC provides slightly better miss ratios for applications with extremely high spatial locality, such as Web Search. In the case of Data Analytics, UC achieves a better miss ratio due to the higher footprint prediction accuracy and low spatial locality of this workload, which prefers smaller page sizes.

Because AC is a block-based design, all the cache hits come solely from the temporal reuse. In other words, the hit ratio directly corresponds to the bandwidth savings provided by the cache. It is interesting to note that AC's miss ratio for TPC-H is consistently high, dropping down only for very large cache sizes; caches smaller than 2-4GB hardly provide any hits. This is in line with our intuition that multi-gigabyte caches are indeed required to provide a noticable reduction in the off-chip traffic for realistic server setups.

C. Performance

Figure 7 compares the performance of the three designs for a range of DRAM cache sizes for all workloads except TPC-H. We also compare the three designs against an ideal DRAM cache that never misses and has no tag overheads, an equivalent to die-stacked main memory.

For small cache sizes, FC performs the best. Compared to AC, it enjoys a much higher hit ratio. The exception is Data Analytics (Map-Reduce), which for the smallest cache size prefers block-based designs due to the lack of spatial locality. As we increase the cache size, the pages stay longer in the cache and their footprints become denser [10], increasing the spatial locality. However, FC's tag array access latency increases with the cache size, increasing both the hit and miss latency and ultimately resulting in diminishing performance returns despite higher hit ratios. In

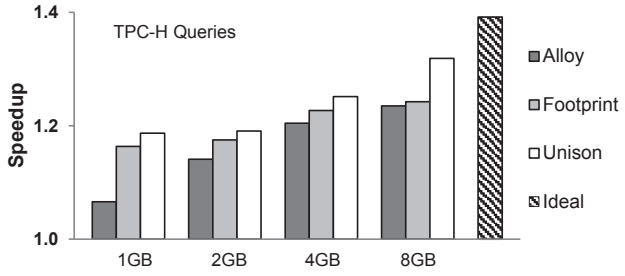


Figure 8. Performance comparison for TPC-H queries.

contrast, the cache size affects neither the hit nor the miss latency in case of UC and AC, which is why UC outperforms FC for larger cache sizes.

A more realistic scenario is shown in Figure 8, which compares the performance of the three designs for TPC-H queries, for 1-8GB caches. In this case Unison Cache constantly outperforms the hypothetical Footprint Cache design due to its low and constant access latency, whereas the tag array access latency precludes performance improvements for Footprint Cache. Alloy Cache sees steady performance improvements, which are however limited by its low hit ratio.

Overall, Unison Cache provides a 14% performance improvement over Alloy Cache and 2% over the hypothetical Footprint Cache design for a 1GB cache (7% and 6% in case of an 8GB cache for TPC-H queries). We note once again that beyond 256-512MB, Footprint Cache is not a feasible option due to its SRAM-based tag array, which requires up to 50MB for an 8GB design.

D. Energy Considerations

All designs reduce the off-chip main memory energy by reducing the number of accesses to it. However, both UC and FC provide a significant further reduction in energy by reducing the number of DRAM row activations, the most energy-demanding operations, by an order of magnitude [10], [28]. Namely, while cache misses in the case of AC result in random memory accesses, both UC and FC perform off-chip data transfers at the granularity of footprints, which fit in a DRAM row. In case of AC, for almost every block transferred between the cache and memory, a DRAM row needs to be activated both in off-chip DRAM and in the cache, whereas for UC a row activation happens once for the whole footprint (i.e., once per $\sim$10 blocks). Similarly, the DRAM cache energy is reduced due to the cache evictions and fills that happen at the footprint granularity. Data transfers between the die-stacked and off-chip DRAM are, thus, much more energy-efficient in the case of UC and FC. The FC study already quantifies these benefits [10], which are around 20-25% of dynamic DRAM energy and to the first order are the same for FC and UC. Because a similar analysis has been done before [10], we omit it for space reasons.

VI. Related Work

Prior work has shown that die-stacked DRAM is a promising technology to bridge the latency gap between processor and memory and to break the memory bandwidth wall. A large body of prior work considered die-stacked DRAM either as main memory [9], [13], [17], [18], or as a large hardware-managed cache [10], [11], [19], [20], [24], [33] due to the limited stacked DRAM capacity. Stacked DRAM has also been considered as a software-managed level in the memory hierarchy [6].

The prior work that considered die-stacked DRAM as a cache has targeted maximizing the hit rates [10], [11] and minimizing the wasted off-chip bandwidth consumption [10], [11], [20], [24] leaning toward page-based organizations that fetch data within pages selectively [10]. Because the technology allowed for only relatively small stacked DRAM cache sizes, tag storage and latency overheads did not impose any considerable challenge to page-based designs, while block-based caches required storing the tags in the cache and minimizing the associated tag latency [20], [24]. Unison Cache eliminates the tag overhead for a page-based DRAM cache, given the rapidly increasing stacked DRAM capacities, relying on the insights into the tag storage optimizations for block-based caches.

VII. Conclusion

This paper introduces Unison Cache, a practical and scalable stacked DRAM cache design, which brings together the best traits of the state-of-the-art block- and page-based designs. Unison Cache achieves high hit rates and low DRAM cache access latency, while eliminating impractically large on-chip tag arrays by embedding the tags in the DRAM cache. Cycle-level simulations of scale-out server platforms using Unison Cache show a 14% performance improvement over the state-of-the-art block-based DRAM cache design, stemming from the high hit rates achieved by Unison Cache. Unlike prior page-based designs, Unison Cache requires no dedicated SRAM-based tag storage, enabling scalability to multi-gigabyte stacked DRAM cache sizes.

Acknowledgment

The authors would like to thank Alexandros Daglis, Onur Kocberber, Nooshin Mirzadeh, Javier Picorel, Georgios Psaropoulos, Stavros Volos, and the anonymous reviewers for their insightful feedback on earlier drafts of this paper. This work has been partially supported by the Intel Doctoral Student Honor Programme, the Google Anita Borg Memorial Scholarship, and the IBM Ph.D. Fellowship Award.

REFERENCES

[1] B. Calder, D. Grunwald, and J. Emer, "Predictive sequential associative cache," in *Proceedings of the 2nd International Symposium on High-Performance Computer Architecture*, Feb. 1996.

[2] C. F. Chen, S.-H. Yang, B. Falsafi, and A. Moshovos, "Accurate and complexity-effective spatial pattern prediction," in *Proceedings of the 10th International Symposium on High Performance Computer Architecture*, feb 2004.

[3] CloudSuite benchmarks, http://parsa.epfl.ch/cloudsuite.

[4] Y. Deng and W. P. Maly, "Interconnect characteristics of 2.5-d system integration scheme," in *Proceedings of the 2001 International Symposium on Physical Design*, ser. ISPD '01. New York, NY, USA: ACM, 2001, pp. 171–175.

[5] Y. Deng and W. P. Maly, *3-Dimensional VLSI: A 2.5-Dimensional Integration Scheme*, 1st ed. Springer Berlin Heidelberg, 2010.

[6] X. Dong, Y. Xie, N. Muralimanohar, and N. P. Jouppi, "Simple but effective heterogeneous main memory with on-chip memory controller support," in *Proceedings of the 2010 International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 2010.

[7] 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 *Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems*, Mar. 2012.

[8] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, "Toward dark silicon in servers," *IEEE Micro*, vol. 31, no. 4, pp. 6–15, July-August 2011.

[9] D. Hyuk 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 *Proceedings of the 16th International Symposium on High Performance Computer Architecture*, Jan. 2010.

[10] 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 *Proceedings of the 40th Annual International Symposium on Computer Architecture*, Jul. 2013.

[11] X. Jiang, N. Madan, L. Zhao, M. Upton, R. Iyer, S. Makineni, D. Newell, Y. Solihin, and R. Balasubramonian, "Chop: Adaptive filter-based dram caching for cmp server platforms," in *Proceedings of the 16th International Symposium on High Performance Computer Architecture*, Jan. 2010.

[12] M. L. Kersten, S. Idreos, S. Manegold, and E. Liarou, "The Researcher's Guide To The Data Deluge: Querying A Scientific Database In Just A Few Seconds," in *Proceedings of International Conference on Very Large Data Bases (VLDB)*, 2011.

[13] 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 *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 2006.

[14] H. Kim, P. Ghoshal, B. Grot, P. V. Gratz, and D. A. Jimenez, "Reducing network-on-chip energy consumption through spatial locality speculation," in *Proceedings of the 5th International Symposium on Networks-on-Chip*, May 2011.

[15] S. Kumar and C. Wilkerson, "Exploiting spatial locality in data caches using spatial footprints," in *Proceedings of the 25th International Symposium on Computer Architecture*, Jun. 1998.

[16] S. Kumar, H. Zhao, A. Shriraman, E. Matthews, S. Dwarkadas, and L. Shannon, "Amoeba-cache: Adaptive blocks for eliminating waste in the memory hierarchy," in *Proceedings of the 45th International Symposium on Microarchitecture*, 2012.

[17] C. Liu, I. Ganusov, and M. Burtscher, "Bridging the processor-memory performance gap with 3D IC technology," *IEEE Design & Test of Computers*, Nov-Dec 2005.

[18] G. H. Loh, "3d-stacked memory architectures for multi-core processors," in *Proceedings of the 35th International Symposium on Computer Architecture*, Jun. 2008.

[19] G. H. Loh, "Extending the effectiveness of 3d-stacked dram caches with an adaptive multi-queue policy," in *Proceedings of the 42nd International Symposium on Microarchitecture*, Dec. 2009.

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

[21] P. Lotfi-Kamran, B. Grot, M. Ferdman, S. Volos, O. Kocberber, J. Picorel, A. Adileh, D. Jevdjic, S. Idgunji, E. Ozer, and B. Falsafi, "Scale-out processors," in *Proceedings of the 39th International Symposium on Computer Architecture*, Jun. 2012.

[22] Micron's Hybrid Memory Cube Earns High Praise in Next-Generation Supercomputer. Available: http://investors.micron.com/releasedetail.cfm?ReleaseID=805283

[23] M. Powell, A. Agarwal, T. Vijaykumar, B. Falsafi, and K. Roy, "Reducing set-associative cache energy via way-prediction and selective direct-mapping," in *Proceedings of the 34th International Symposium on Microarchitecture*, Dec. 2001.

[24] M. Qureshi and G. H. Loh, "Fundamental latency trade-offs in architecting DRAM caches," in *Proceedings of the 45th International Symposium on Microarchitecture*, Dec. 2012.

[25] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "Dramsim2: A cycle accurate memory system simulator," *Computer Architecture Letters*, vol. 10, no. 1, pp. 16 –19, jan.-jun 2011.

[26] M. Santarini, "Stacked & Loaded: Xilinx SSI, 28-Gbps I/O Yield Amazing FPGAs," Xilinx Xcell Journal, Tech. Rep., 1st Quarter 2011.

[27] S. Somogyi, T. F. Wenisch, A. Ailamaki, B. Falsafi, and A. Moshovos, "Spatial memory streaming," in *Proceedings of the 33rd International Symposium on Computer Architecture*, Jun. 2006.

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

[29] 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, pp. 18–31, Jul. 2006.

[30] W. A. Wulf and S. A. McKee, "Hitting the memory wall: Implications of the obvious," *SIGARCH Comput. Archit. News*, vol. 23, no. 1, pp. 20–24, Mar. 1995.

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

[32] D. H. Yoon, M. K. Jeong, M. Sullivan, and M. Erez, "The dynamic granularity memory system," in *Proceedings of the 39th International Symposium on Computer Architecture*, Jun. 2012.

[33] L. Zhao, R. Iyer, R. Illikkal, and D. Newell, "Exploring dram cache architectures for cmp server platforms," in *Proceedings of the 25th International Conference on Computer Design*, Oct. 2007.