Die-Stacked DRAM Caches for Servers

Hit Ratio, Latency, or Bandwidth? Have It All with Footprint Cache

Djordje Jevdjic Stavros Volos Babak Falsafi
EcoCloud, EPFL
{djordje.jevdjic, stavros.volos, babak.falsafi}@epfl.ch

ABSTRACT

Recent research advocates using large die-stacked DRAM caches to break the memory bandwidth wall. Existing DRAM cache designs fall into one of two categories — block-based and page-based. The former organize data in conventional blocks (e.g., 64B), ensuring low off-chip bandwidth utilization, but co-locate tags and data in the stacked DRAM, incurring high lookup latency. Furthermore, such designs suffer from low hit ratios due to poor temporal locality. In contrast, page-based caches, which manage data at larger granularity (e.g., 4KB pages), allow for reduced tag array overhead and fast lookup, and leverage high spatial locality at the cost of moving large amounts of data on and off the chip.

This paper introduces Footprint Cache, an efficient die-stacked DRAM cache design for server processors. Footprint Cache allocates data at the granularity of pages, but identifies and fetches only those blocks within a page that will be touched during the page's residency in the cache — i.e., the page's footprint. In doing so, Footprint Cache eliminates the excessive off-chip traffic associated with page-based designs, while preserving their high hit ratio, small tag array overhead, and low lookup latency. Cycle-accurate simulation results of a 16-core server with up to 512MB Footprint Cache indicate a 57% performance improvement over a baseline chip without a die-stacked cache. Compared to a state-of-the-art block-based design, our design improves performance by 13% while reducing dynamic energy of stacked DRAM by 24%.

Categories and Subject Descriptors

B.3.2 [**Memory Structures**]: Design Styles — *Cache memories*; B.3.1 [**Memory Structures**]: Semiconductor Memories — *Dynamic memory (DRAM)*

General Terms

Design, Measurement, Performance

1. INTRODUCTION

The slowdown in Dennard Scaling is shifting many-core server chips towards larger numbers of lean cores. Conventional (e.g, OLTP, DSS) and emerging scale-out (e.g., Data Serving, Web Search) server workloads exhibit an abundance of request-level parallelism and therefore benefit from many-core chips that enable high throughput. The growth in core count, however, ultimately drives designs into a memory bandwidth wall [10, 30] due to poor pin count scalability.

Emerging 3D die stacking technology interconnects a processor die with a stack of DRAM dies using high-density, low-latency through-silicon vias (TSVs). This technology virtually eliminates the memory bandwidth wall by providing orders of magnitude higher bandwidth, and exposes lower latency for stacked on-chip DRAM [10, 11, 14, 20, 21]. Technological constraints, however, limit the stacked DRAM capacity to levels that are far lower than what modern server workloads demand. As such, most proposals for die stacking advocate using the stacked DRAM as a cache [13, 22, 24].

There are two classes of die-stacked DRAM caches: *block-based* and *page-based*. Block-based designs store data at a fine granularity (e.g., 64B) [22, 24] in order to optimize for capacity and temporal locality. As such, they require a prohibitively large tag space infeasible to support in on-chip SRAM. Thus, they mandate co-location of tags with data in the DRAM cache [22, 24]. Such an implementation, however, increases hit latency and reduces DRAM locality. Furthermore, due to low temporal reuse in server workloads, block-based designs experience high miss rates, frequently exposing full off-chip latency to incoming requests. However, thanks to small fetch granularity, block-based designs result in low off-chip traffic. Because servers favor overall throughput over single-threaded performance, minimizing off-chip bandwidth to support more threads is a top design priority.

Page-based designs, whose allocation unit is in the order of a few kilobytes, rely on abundant spatial locality that benefits server workloads. As a result, page-based designs enjoy a much higher (up to 10x) hit ratio compared to block-based ones [11, 13]. Due to their large allocation granularity, page-based caches require smaller tag arrays that can be stored in SRAM. Moreover, page-based caches maximize the benefit of DRAM row-buffer locality. However, the large granularity magnifies off-chip traffic by up to an order of magnitude and results in poor utilization of available cache capacity, because much of the data in a page, although allocated and fetched, are never touched prior to the page's eviction.

To exploit the best aspects of both block- and page-based designs while avoiding their respective drawbacks, we propose *Footprint Cache*. Footprint Cache decouples the cache allocation unit from the fetch unit, allocating large pages but identifying and fetching only those 64-byte blocks that will be used during the residency of the page in the cache — i.e., the page's *footprint*. To identify pages' footprints, we leverage prior research on spatial correlation [34] and apply it in the context of page-based DRAM caches to design a simple and highly accurate footprint predictor. The predictor allows for exploiting the abundant spatial locality of scale-out workloads in a bandwidth-efficient manner. As a result, Footprint Cache achieves high hit ratios, comparable to those of page-based approaches, and low off-chip traffic as in block-based caches. Due to its large allocation unit, Footprint Cache allows for small and fast, SRAM-based tag arrays. To optimize for capacity, Footprint Cache further identifies pages that have few useful blocks (no spatial locality) and show no reuse (no temporal locality), which account for a significant fraction of the total pages. By not allocating such pages and fetching only the individual blocks on demand, our design improves the effective cache capacity.

In this paper we make the following contributions:

- We propose Footprint Cache, a novel die-stacked DRAM cache architecture that combines the benefits of block- and page-based designs through footprints. By predicting the footprint of each page and fetching only the predicted blocks, Footprint Cache overcomes the high off-chip bandwidth overhead of page-based designs, yet preserving their high hit ratio. We dem-

ISCA 2013, June 23-27, 2013, Tel-Aviv, Israel

onstrate that Footprint Cache reduces the off-chip traffic of the page-based design by 2.6x, while achieving 4.7x higher hit ratios compared to the block-based design, on average. Meanwhile, by managing capacity at the granularity of pages, Footprint Cache achieves small tag overhead and low access latency.
- We observe that a significant fraction of the pages in the cache exhibit neither temporal nor spatial locality, and thus do not contribute to any cache hits. By detecting and not caching these pages, Footprint Cache improves the effective cache capacity and reduces the miss rate by an additional 10%, on average.
- Using cycle-accurate, full-system simulation of a scale-out processor [26], we demonstrate that Footprint Cache achieves 57% performance improvement over a baseline system without a die-stacked cache, while outperforming state-of-the-art designs in terms of performance and energy. Footprint Cache reduces off-chip DRAM dynamic energy of the baseline system by 78%, also providing a 24% and 9% reduction in stacked DRAM dynamic energy compared to the state-of-the-art block-based design and the page-based design, respectively.

The rest of the paper is organized as follows. We examine existing DRAM cache designs in Section 2, and identify their key advantages and drawbacks. We describe the basic idea behind our proposal, Footprint Cache, in Section 3 and detail our design in Section 4. In Section 5 we describe our evaluation methodology. We present our experimental results in Section 6 and discuss interesting questions in Section 7. We revisit prominent prior work in Section 8 and conclude in Section 9.

2. BACKGROUND AND MOTIVATION

With the slowdown in Dennard Scaling server chips are resorting to larger numbers of lean cores (e.g., Tilera) to maintain a practical power envelope. Scale-out server workloads benefit from such many-core processor organizations, which enable high throughput thanks to the abundant parallelism in these workloads. The growth in core count, however, ultimately drives designs into a memory bandwidth wall [10, 30] due to poor pin count scalability. Emerging many-core chips with hundreds of cores/chip are already able to utilize and even exceed their bandwidth budgets [10, 14, 26], hitting the bandwidth wall before the power wall [26].

Recent research advocates using die-stacked DRAM to break the memory bandwidth wall [10, 11, 13, 14, 20, 21, 22, 24]. Figure 1 assesses the opportunity of the technology to boost performance of scale-out and multiprogrammed workloads from two aspects: bandwidth and latency (the details on experimental methodology are explained in Section 5). The first set of bars shows performance improvement for a many-core server system [26] with the main memory fully integrated on the chip using die stacking, providing 8x the bandwidth of the 2D baseline. The second set of bars shows performance improvement of the same high-bandwidth system, but with halved DRAM latency [24]. We see that both bandwidth and latency play a vital role in achieving high performance, which implies that future designs must exploit both opportunities given by the technology.

Technological constraints, however, limit the stacked DRAM capacity to levels that are far lower than what modern server workloads demand [24]. While today's servers need tens to hundreds of gigabytes of DRAM each, the projections for die-stacked DRAM capacity vary between hundreds of megabytes to several gigabytes. Thus, most proposals for die stacking advocate using the stacked DRAM as a cache [13, 22, 24]. Unfortunately, the inherent limitations of DRAM cache designs prevent them from achieving the full potential of the technology, depicted in Figure 1. Firstly, DRAM caches, regardless of their organization, require significant tag storage due to their large capacity, whose lookup necessarily

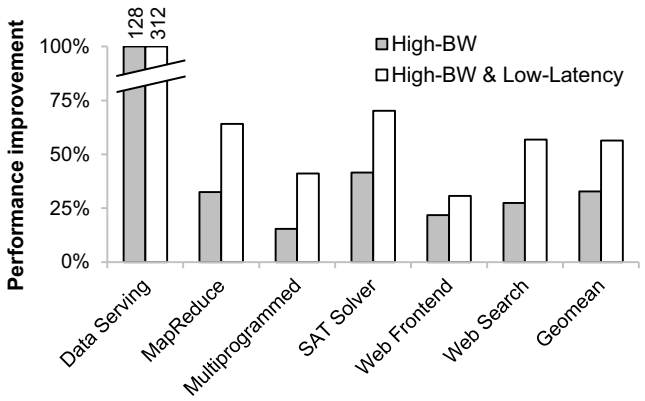

Figure 1. Opportunity for performance improvement with high-bandwidth and low-latency die-stacked DRAM.

adds extra latency to the critical path. Secondly, the limited capacity of the stacked DRAM limits the level of concurrency it can provide, despite the virtually unlimited TSV bandwidth. The stacked DRAM is orders of magnitude smaller than the off-chip DRAM, and, thus, experiences frequent bank conflicts and lower availability. In contrast to off-chip main memory systems with hundreds of gigabytes of DRAM that provide more bandwidth than the memory channels can sustain, the bandwidth to the DRAM cache is restricted by the parallelism in the stacked DRAM itself, and not by the interface. Therefore, stacked DRAM caches fall short of fully leveraging the abundant on-chip bandwidth enabled by dense TSV buses. Cache designs must be aware of this limitation, and optimize for the stacked DRAM locality to allow for higher concurrency and availability.

Furthermore, despite their capacity, DRAM caches exhibit high miss ratios, with each miss being satisfied from the off-chip memory at full off-chip latency. This behavior is caused by the low reuse of the data in lower-level caches [9], in contrast to L1 caches, where data are frequently reused and where most of the temporal locality is exploited. This phenomenon is further exacerbated by vast datasets of scale-out workloads, which do not form any well-defined working sets [6]. Besides their latency penalty, misses inherently result in DRAM cache evictions. We find that, for scale-out workloads, these are mostly dirty evictions, because data reside in the cache for long enough to become modified by dirty evictions from the upper-level caches. Dirty evictions consume additional off-chip and on-chip TSV bandwidth, affecting the stacked DRAM availability as well (the data have to be read from the stacked DRAM and written back to the off-chip DRAM). The same holds for cache fills that follow the misses.

2.1 Guidelines for Designing DRAM Caches

As existing DRAM cache designs fall short of leveraging the benefits that die stacking technology provides, we present a set of guidelines for designing effective DRAM caches:

- **Fast tag lookup**. Because tag lookups are on the critical path of all requests coming to the cache, tag lookup latency must be minimized. While this statement holds for all cache designs that serialize tag and data lookups, it gains more importance in the context of DRAM caches, due to their tag array size.
- **Small tag storage**. The total storage dedicated to tags or other metadata should be minimal, as it does not directly contribute to better system performance, but does incur high cost.
- **Low off-chip traffic**. While cache misses are responsible for most of the off-chip bandwidth overhead, various cache features can adversely impact off-chip bandwidth even further. Examples include the use of large cache blocks that saturate

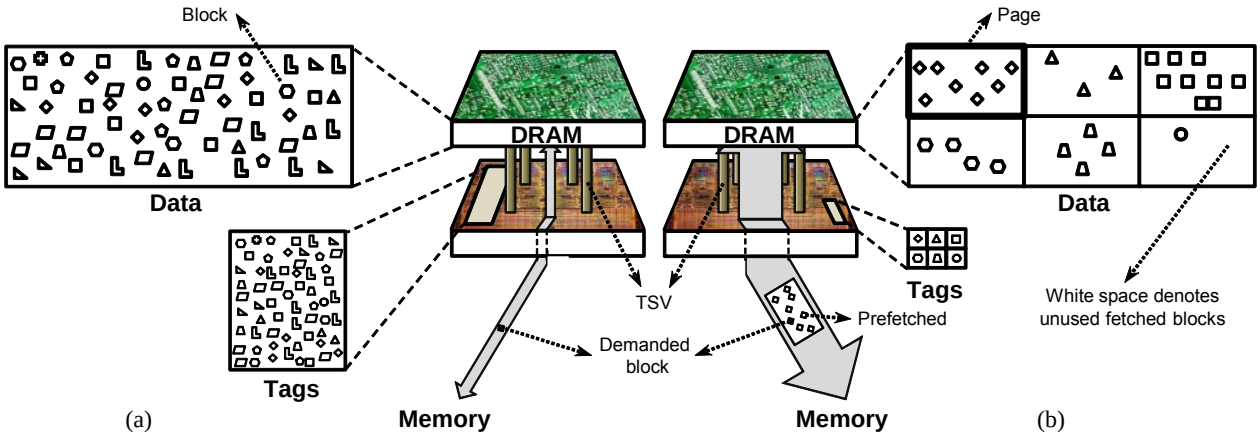

Figure 2. A DRAM die stacked on top of the logic die, used as (a) a block-based cache or as (b) a page-based cache. For the block-based design, one tag entry corresponds to one data block. For the page-based design, only the useful blocks (accessed by the cores) are shown in the figure, and one tag entry corresponds to one page.

off-chip bandwidth. Reduction in off-chip traffic is the main driver for 3D-stacked DRAM adoption, and as such should be among the top priority goals.

- **High hit ratio** is crucial to exploiting both the bandwidth and the latency benefits that the technology provides, demonstrated by Figure 1.
- **Low hit and miss latency**. To achieve the benefits depicted in Figure 1, DRAM caches must optimize for both hit and miss latency. Internal details of the cache organization should neither penalize hit latency nor postpone miss serving.
- **High DRAM access locality**. Accesses to DRAM structures experience unpredictable latency, highly dependent on the locality of references, availability, address-mapping schemes, row-buffer management policy, and access scheduling. To minimize the stacked DRAM and off-chip DRAM access latencies, cache designs must be take of all these parameters into account.
- **Efficient capacity management**. Allocation of space for data that are never used should be avoided. The problem is severe in page-based designs, which suffer from internal fragmentation.

Unfortunately, most of these requirements are mutually conflicting, which makes the design process more challenging. To better understand such challenges, we focus on two main DRAM cache designs that optimize for different constraints.

2.2 Block-Based Caches

On-chip caches have traditionally been designed to primarily exploit temporal locality, and to make the best use of their limited capacity. Trade-offs between the effective cache capacity, temporal and spatial locality resulted in 16- to 128-byte cache blocks, 64-byte being the most common block size employed today. Such a design is illustrated in Figure 2a. For large DRAM caches, 64-byte blocks would require huge tag storage, as illustrated in Figure 2a, which is infeasible to build in SRAM, thereby forcing the tags to be embedded in DRAM [13, 22, 24]. Embedding tags in DRAM, however, results in multiple DRAM accesses per cache request — and, consequently, in substantially higher hit and miss latencies. Intelligent co-location of data with the corresponding tags in the same DRAM row [22] accompanied with optimized access scheduling [24], obviates the need for multiple DRAM accesses per request. However, the optimization only partially reduces the high hit latency, because of the need for several operations to be performed within the DRAM row-buffer. Furthermore, the co-location of tags and data mandates particular data placement policies that diminish DRAM locality. It also requires a way to determine the presence of a block in the cache prior to accessing the tags, as well as additional multi-megabyte storage for that purpose (not shown in Figure 2a), whose access latency is on the critical path. Most importantly, block-based designs fall short of exploiting abundant spatial locality. Instead, they focus on limited temporal locality, experiencing high miss ratios, thus frequently exposing full off-chip latency to incoming requests. However, due to the small fetch unit and the efficient management of cache capacity, block-based designs minimize off-chip traffic, making them a favorable option for high-throughput servers.

2.3 Page-Based Caches

Increasing the block size allows for a proportionate reduction in tag storage. The use of larger allocation/fetch units (e.g., 1-8KB) makes the placement of tags in SRAM feasible at acceptable storage overhead [13]. We call such units *pages*.[1] The large fetch unit allows for maximum DRAM access efficiency, fully exploiting locality in both off-chip and stacked DRAM. For instance, a single DRAM row opening is needed per off-chip DRAM fetch, eviction, or stacked DRAM fill, for a whole page, assuming that the page size does not exceed the DRAM row size. While large DRAM caches exhibit limited temporal locality, they show significant spatial locality, which can be easily leveraged by large fetch units, as illustrated in Figure 2b, providing an order of magnitude more hits compared to a block-based cache of the same size [11, 13]. Cache hits are critical to exploiting the latency advantages of die-stacked DRAM and page-based caches provide them at lower latency. Unfortunately, many of the cached pages contain data that are not used prior to the page eviction, resulting in excessive data overfetch [13] and capacity waste. As a result, page-based caches tend to increase the off-chip traffic of the baseline system without a DRAM cache by up to an order of magnitude in the worst case, which negates a key benefit of die-stacked DRAM caches.

Summary. Table 1 provides a comparison between the two designs with respect to the most important features. The block-based and page-based designs show complementary, yet mutually exclusive, characteristics. To take the best of the both designs, we propose *Footprint Cache*.

3. FOOTPRINT CACHE

While the block-based and page-based designs show complementary properties, never achieving the same goals, together they could meet all of the requirements of an ideal die-stacked cache as

1. Not to be confused with a DRAM row, sometimes called DRAM page.

Table 1. Comparison: block- & page-based designs.

	Block-based	Page-based
Small and fast tag storage	×	✓
Low off-chip traffic	✓	×
High hit ratio	×	✓
Low hit latency	×	✓
High DRAM locality	×	✓
Efficient capacity management	✓	×

summarized in Section 2.1. The page-based design demonstrates superior properties overall, but due to its excessive off-chip traffic overheads, it is not a feasible option. Ideally, we would like to achieve the properties of the page-based design, but without the unnecessary traffic and with better capacity management. Our proposal, Footprint Cache, uses a page-based organization, but identifies the blocks that will be demanded by the cores during a page's on-chip residency. It then fetches only those blocks at the page allocation time, eliminating the unnecessary off-chip and on-chip traffic. Footprint Cache further identifies pages that have the fewest useful blocks and show no reuse, and neither allocates entries in the cache for such pages nor fetches them. Footprint Cache instead fetches 64-byte blocks from such pages, one by one and only on demand, and forwards them to the requestor, bypassing the cache. Such pages account for a significant fraction of all pages that are fetched and are the biggest contributor to the capacity waste. Thus, Footprint Cache mitigates both the bandwidth and capacity problems of page-based designs and manages to get the best of the both designs.

Footprint Cache decouples the cache allocation unit from the fetch unit, similar to sub-blocked (or sectored) caches, allocating large pages while fetching 64-byte blocks. The set of useful blocks accessed during page's on-chip residency constitute the page's *footprint*. Upon a miss in the cache, a new page is allocated and the whole page's footprint is fetched at once from the main memory. To detect the useful blocks, we leverage prior work on spatial correlation [34] and design a simple and highly accurate predictor that identifies the page's footprint. As a result, Footprint Cache achieves high hit ratios, comparable to those of page-based approaches, and low off-chip traffic as in conventional block-based caches. Because the whole footprint is fetched and evicted at once, Footprint Cache enforces high DRAM access locality both for the off-chip and stacked DRAM, thus allowing for lower DRAM access latency. Last, but not least, Footprint Cache allows for a small and fast, SRAM-based tag array due to its large allocation unit. In summary, Footprint Cache meets all of the requirements of an ideal die-stacked cache as summarized in Section 2.1.

3.1 Footprint Prediction

To achieve the desired properties, Footprint Cache relies on the footprint predictor. The accuracy of the footprint predictor is crucial for both performance and energy efficiency. The predictor must have high coverage, ideally predicting all the blocks that will be later demanded. Every unpredicted block that is demanded later would result in a cache miss and consequently, in performance and energy loss. We call such an event *underprediction*. While it is important to correctly predict as many blocks as possible, it is essential that the predictor has minimal overprediction rate. *Overpredictions* represent blocks that are fetched but not used prior to eviction, and their transfer to and from main memory merely wastes off-chip and on-chip bandwidth and energy.

An example of a system with no overpredictions is a sub-blocked cache, which allocates pages but fetches every block on demand. However, such a system would have the maximum number of underpredictions, as it would experience a miss for each demanded block in a page. A system with no underpredictions would be a page-based cache, that fetches all the data from the demanded page at once. Fetching all the blocks is, however, an even worse solution, as it produces the maximum number of overpredictions, saturating the off-chip bandwidth.

Minimizing both the underprediction and overprediction rates at the same time is a challenging task for a predictor. To achieve the two conflicting goals, Footprint Cache relies on the observation that there is a high correlation between data and the code that accesses those data. For instance, server software exposes a well-defined interface with only several functions for accessing their structured datasets — e.g., get and set methods of a class or various data structure iterators. Traversals of data structures require repetitive calls to these functions, resulting in recurring memory access patterns. The access pattern observed from the first call to such a function can be used to predict the memory access patterns of its subsequent calls. This fundamental property has been exploited in various contexts [18], mostly for data prefetching [2, 16, 34] and speculating on data granularity [15, 17, 39].

Footprint Cache achieves high prediction accuracy by monitoring the code that accesses data residing in the cache. The first instruction that accesses a page provides valuable information about the data that the page contains and is a good indicator of future accesses within that page [34], due to regular and repetitive layouts of data structures. By observing which blocks the code further accesses and by remembering that information, we can later predict, with high accuracy, which blocks will be demanded when another page, possibly previously unvisited, is accessed by the same piece of code [34].

Prediction based on the first instruction that accesses a page is highly accurate provided that data structures always have the same layout within a page. However, this is not necessarily the case for all the workloads and page sizes. To account for various possible data structure alignments across different pages, we base our prediction mechanism not only on the instruction that caused the page miss, but also on the distance between the requested block and the beginning of the page, which we call *offset*. The combination of PC and offset (noted as PC & offset) provides near-perfect prediction accuracy at low overhead. Previous work has a detailed study of other related prediction mechanisms and their trade-offs in the context of data prefetching [34].

3.2 Capacity Optimization

Our analysis shows that a significant fraction of pages (more than a quarter, on average) contain only a single useful block. Such pages often account for the largest share of the pages in workloads with low spatial locality. Moreover, we find that, on average, 95% of these pages show no reuse in the DRAM cache, and therefore waste capacity. We call such pages *singleton* pages. Footprint Cache is able to identify such pages with almost perfect accuracy, thanks to the fact that these pages are accessed by a single instruction, and obviously, with a single offset. Footprint Cache does not allocate entries for such pages. The requested block is directly forwarded to the higher cache level and its subsequent eviction is not tracked. This mechanism increases the effective cache capacity reusing the existing footprint prediction mechanisms. It avoids eviction of useful pages, thus allowing for even higher hit ratios. The optimization plays an important role at smaller cache sizes, for which efficient use of cache capacity matters the most.

4. FOOTPRINT CACHE DESIGN

Footprint Cache tightly couples the footprint prediction mechanism with the tag array. The footprint predictor uses the information from the tag array to learn pages' footprints, storing

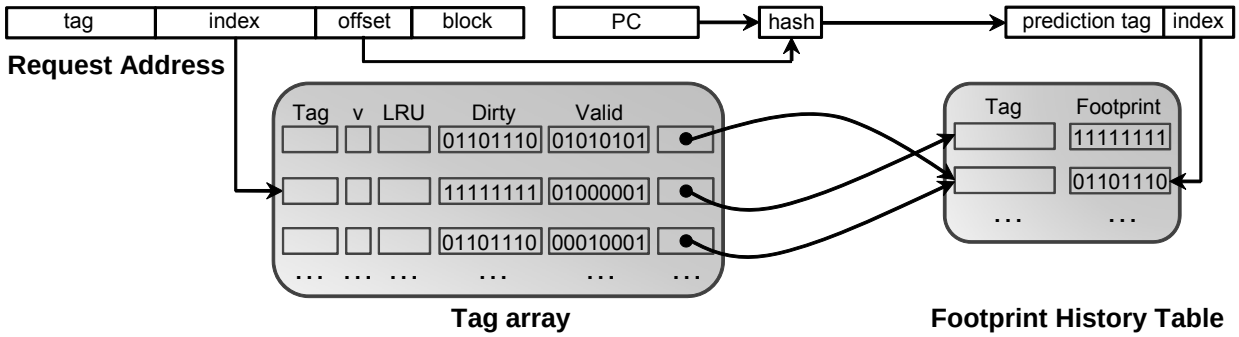

Figure 3. Footprint Cache tag array and Footprint History Table (FHT).

the footprints into a history table upon page evictions and using the footprint information upon page misses to fetch useful blocks.We next detail the predictor design and its integration with the tag array, and we further explain the prediction history management.

4.1 Footprint Cache Tag Array

Similar to page-based and sub-blocked caches, Footprint Cache requires almost two orders of magnitude smaller tag arrays, which can be kept within reasonably small SRAM storage. The tag array is organized as a set-associative structure; set and way pairs directly determine physical addresses of pages cached in DRAM. The size of a page is selected to match commonly used DRAM row sizes (e.g., 1-4KB),[2] keeping in mind the impact on the prediction accuracy and tag overhead. Similarly to sub-blocked caches, Footprint Cache keeps two bit vectors to track valid and dirty blocks, and a page-level valid bit (Figure 3). In the multicore system we evaluate, the tag array is distributed into four tiles organized in a page-interleaved fashion, each tile being responsible for a partition of the DRAM cache. We use page-interleaved placement in the Footprint Cache tag array for efficiency.[3] Each tag tile is attached to an on-chip memory controller that controls a 128-bit TSV channel associated with the corresponding partition of the DRAM cache. The only additional overhead introduced in the tag array is a pointer that links pages to the prediction history described in Section 4.2.

4.2 Prediction History

The prediction history, shared by all the tag tiles, is kept as a separate tiled structure, called *Footprint History Table* (FHT). The FHT is a set-associative structure indexed by PC & offset pairs, storing predicted footprints for PCs & offsets that trigger page misses. Each entry keeps a tag identifying the PC & offset key, while the corresponding prediction information is kept as a bit vector, determining the footprint associated with the key [34]. The FHT size is independent of the workload's dataset, as it holds only a small fraction of the workload's instruction footprint, measured in kilobytes. The FHT is updated upon every page eviction with the most recent footprint generated during the page's on-chip residency.

The FHT is accessed only in case the page containing a requested address is not found in the cache. Upon a page miss, which we call a *triggering miss*, the table is queried by the PC of the instruction that caused the miss and the offset bits of the request address, returning the predicted footprint. The triggering miss is served by the off-chip memory and a page eviction takes place. If the PC & offset pair exists in the FHT, which is the common case, the rest of the blocks, encoded in the predicted footprint, are fetched from memory, and a pointer to the FHT entry is stored in the tag entry. If the FHT does not contain the PC & offset pair, which mostly happens at the beginning of the program execution, a new FHT entry is allocated, and a pointer to the new FHT entry is stored in the tag entry.

Table 2. Block state encoding.

Dirty-Valid Bits	Block State
00	the block is not in the cache
01	the block is valid, clean, not demanded yet
10	the block is valid, clean, was demanded
11	the block is valid, dirty, was demanded

Upon a cache eviction, a bit vector containing the blocks that were indeed demanded by the cores, is sent to the FHT, using the pointer created during the allocation of the page. This bit vector gives feedback to the prediction mechanism, correcting mispredictions if any, and keeping the FHT in harmony with the workload's execution phase. As we do not store the PC in the tag entry, but only the pointer to the FHT entry, it is possible, although unlikely, that the pointer becomes stale, as a result of an FHT eviction. While this may affect prediction accuracy, we could not see any observable impact. The reason is, unlike the cached data, the content of the FHT is stable, therefore, such situations almost never happen.

For practical reasons, similar to the tag array, the FHT is designed as a tiled structure, and its entries are distributed based on PC & offset keys of incoming requests, not necessarily matching the distribution criterion for the tag tiles (physical addresses). The mismatch can result in frequent accesses to neighboring FHT tiles. However, this does not have any timing impact due to the FHT's negligible access latency, which is not on the critical path of memory accesses.

4.3 Footprint Generation

Blocks that are placed in the cache must be set to the valid state regardless of whether they are demanded by a core or predicted by the predictor. On the contrary, providing correct feedback to the FHT requires a distinction between the blocks that are demanded and the ones that are in the cache but were not demanded during the page's on-chip residency (overpredictions). Upon a core's request, whether a hit or a miss, the corresponding block should be marked as demanded. To make this distinction possible without additional storage, we reuse the existing valid and dirty bits to create the block state encoding listed in Table 2.

We are able to achieve this encoding, because a block cannot be in a dirty state if it has not been used by a core. The high order bits for all block states together represent the demanded bit vector (i.e., the page's footprint), used to update the FHT upon a page eviction.

2. The exact matching of the page size and DRAM row size is not crucial for the proposed technique.

3. The placement policy in the upper-level caches, however, is not affected by this design decision and can use either block- or page-interleaving, with no observable performance difference for our workloads. In this work we assume block-interleaving.

Table 3. Archtiectural parameters.

Technology	20nm, 0.85V, 3GHz
CMP features	Scale-Out Processor with six 16-core pods
Core	ARM Cortex-A15-like: 3-way OoO
L1-I/D caches	64KB, split, 64B blocks 2-cycle load-to-use
L2 cache per pod	Unified, 4MB, 16-way, 64B blocks, 4 banks, 13-cycle hit latency
Interconnect per pod	16x4 crossbar
Off-chip DRAM	16-32 GB per pod, 6 DDR3-1600 channels, one per pod Maximum bandwidth of 76.8GB/s per chip, 8 banks per rank, 2KB row-buffer
Stacked DRAM	DDR3-3200 (1.6GHz bus frequency) 4 channels per pod, 8 banks per rank, 2KB 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

4.4 Capacity Optimization

Footprint Cache increases the effective cache capacity by avoiding allocation of singleton pages — i.e., pages with a single useful block. In our design, if the footprint bit vector corresponding to a missing PC & offset pair in the FHT has a single bit set, the corresponding cache entry is allocated neither in the cache nor in the tag array. Not allocating entries for singleton pages in the tag array implies that the FHT would never receive feedback regarding its single block predictions. Once a page is classified as singleton, it would remain singleton until its FHT entry is evicted, regardless of any mispredictions or changes in the application's behavior.

To avoid this scenario, we add a small table, which is further partitioned and co-located with the tag array tiles, called *Singleton Table* (ST). In case the FHT predicts a singleton block, the page is not allocated in the cache, but an ST entry is allocated, containing the PC, offset, and the page tag. The ST is indexed by a page tag, and only upon a page miss. Finding the tag in the ST, but with a different offset, implies that there is a second access to a page that was originally predicted to be a singleton page (underprediction). In this case, the new tag and FHT entry is allocated with the PC & offset information found in the ST, and the corresponding ST entry is invalidated. An entry stays in the ST until a second access to the page, or until its eviction. The ST has negligible overhead (3KB, 512 entries), but allows for more accurate and adaptive prediction of singleton pages. This is important for smaller caches, where pages could be misclassified as singleton due to their short on-chip residency and capacity conflicts.

5. METHODOLOGY

We evaluate Footprint Cache in terms of performance, energy efficiency, and hardware overhead in the context of high-throughput, bandwidth-demanding scale-out processors, which can benefit from the die stacking technology [26]. We compare Footprint Cache to a state-of-the-art implementation of conventional block-based DRAM caches as well as to page-based DRAM caches.

5.1 Baseline System

We evaluate Footprint Cache using scale-out processors. The scale-out processor architecture splits the available chip resources into multiple stand-alone servers, called pods [26], which are multicore configurations designed to match the needs of scale-out workloads and deliver the highest throughput for given silicon real-estate. A pod is a complete server that tightly couples a number of cores to a modestly-sized last-level cache using a fast interconnect. Replicating the pod to fill the die area yields processors that have optimal performance density. Moreover, as each pod is a stand-alone server, scale-out processors avoid the expense of global (i.e., inter-pod) interconnect and coherence. These features synergistically maximize throughput, lower design complexity, and improve technology scalability. We model a chip in 20nm technology with on-chip supply voltage of 0.85V, assuming area and power budgets of 250mm^2 and 105W, respectively. The chip features six 16-core pods and six single-channel DDR3-1600 interfaces. Area and power estimates are measured by scaling down the published data at 40/45nm process technology [12, 26], following ITRS projections. Table 3 summarizes the parameters for the highest-performance baseline chip that can be designed under the area, power, and bandwidth constraints described above [26].

Table 4. Cache parameters.

Cache capacity (MB)	**64**	**128**	**256**	**512**
Footprint Cache				
Tag storage (MB)	0.40	0.8	1.58	3.12
Tag latency (cycles)	4	6	9	11
Block-based cache				
MissMap entries	192K	192K	192K	288K
MissMap storage (MB)	1.95	1.95	1.95	2.92
MissMap associativity	24	24	24	36
MissMap latency (cycles)	9	9	9	11
Page-based cache				
Tag storage (MB)	0.22	0.44	0.86	1.69
Tag latency (cycles)	4	5	6	9

5.2 DRAM Cache Organizations

Footprint Cache parameters are listed in Table 4. We use an open-page policy both for the on-chip and off-chip DRAM, as our design exhibits near-optimal data locality for off-chip DRAM, on-chip DRAM fills and evictions, while data locality for on-chip read/write requests is workload-dependent. We use 2KB pages and 2KB address-interleaving for on-chip memory channels. The FHT has 16K entries (144KB) while the ST has 512 entries (3KB).

Block-based caches are modeled after a state-of-the-art proposal that provides an elegant solution to tag handling, by co-locating tags from one cache set with all the blocks in that set in the same DRAM row [24]. For 2KB DRAM rows, it is possible to fit 29 64-byte data blocks in one row, using the three remaining blocks for the corresponding tags. A crucial optimization to avoid two DRAM accesses per cache access includes intelligent memory controller scheduling [24]. An access to a cache block involves a single row activation, and one column activation signal (CAS) to read the tags, a one-cycle tag lookup to determine the location of the data block, another CAS to retrieve/write to the data block, and a third CAS to write back the updated tags. The last CAS is required as the tags must be updated, however, our evaluation does not account for that latency, as we assume the algorithm can be re-engineered to take the tag updates off the critical path.

The exact location of the requested block is stored in DRAM tags and determined after the row activation. However, the presence of the block in the cache is tracked by a compact structure called *MissMap*, which keeps track of the cached data at 4KB granularity, storing bit vectors that determine only the presence of blocks within a page. If the requested block is found in MissMap, the cache is accessed, otherwise the request is serviced by memory. This optimization avoids unnecessary tag lookups in DRAM in case of cache misses [24].

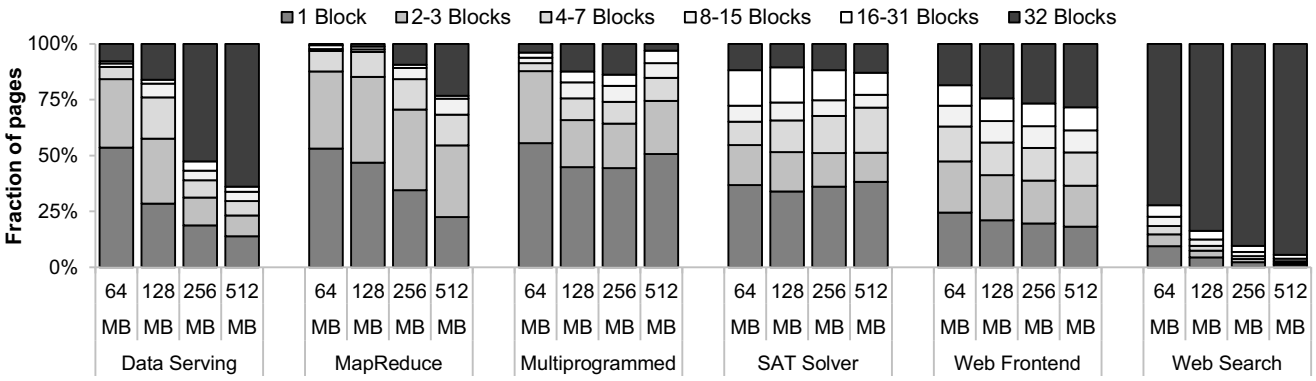

Figure 4. Page access density as a function of cache capacity. Note the increase in page access density with the cache capacity.

Table 4 lists the parameters we used for evaluation of the proposed block-based DRAM cache. By dropping coherence bits from the tags, we were able to achieve higher storage density with 30 data blocks per DRAM row, and only two tag blocks, assuming ARM's extended 40-bit physical addressing. This also increased the cache associativity from 29 to 30, and reduced the latency of tag retrieval, which is on the critical path. Within proposed 2MB of SRAM dedicated to MissMap [24], we were also able to fit more MissMap entries. However, with larger cache capacity, we observed performance degradation due to a high number of MissMap evictions, which in return generate many dirty cache evictions. Although this operation is not on the critical path, we found that MissMap evictions interfere with regular read/write cache requests as well as with cache fills of other blocks, contending for the same DRAM bank. The reason is that MissMap keeps and evicts spatially consecutive blocks, which are all in different cache sets, and therefore, in different DRAM rows, causing excessive row activations. To avoid this situation, we increased the size of the MissMap structure by 50% to evaluate 512MB caches.

We use close-page policy both for off-chip and on-chip DRAM, as we found that it performs better due to absence of data locality, and 64-byte address interleaving between memory channels to increase DRAM-level parallelism.

Page-based cache parameters are listed in Table 4. We use open-page policy both for stacked and off-chip DRAM, as page-based caches exhibit optimal data locality for off-chip DRAM, optimal data locality in stacked DRAM for fills and evictions, while data-locality for on-chip read/write requests is workload-dependent. We use 2KB pages and 2KB address-interleaving for on-chip memory channels.

Row-buffer management policies and address-mapping schemes are chosen for each evaluated system separately to allow for optimal performance and DRAM-level parallelism.

5.3 Workloads

Our scale-out workloads, which include Data Serving, MapReduce, SAT Solver, Web Frontend, and Web Search, are taken from CloudSuite 1.0 [3, 6]. Their memory footprints exceed the available memory, which is 16-32GB. As a reference, we also simulate a multiprogrammed desktop workload that consists of SPEC INT2006 applications using the reference input set. We run a mix of different applications and inputs to utilize all available cores.

These workloads, when running on the baseline chip we consider, have off-chip bandwidth of 0.6-1.6GB/s per core, or 60-150GB/s for the whole chip. While today's dominant server processors, which integrate a handful of fat cores, are not able to utilize the available bandwidth when running these workloads [6], our design utilizes and even exceeds the bandwidth budget [26].

5.4 Simulation Infrastructure

We evaluate Footprint Cache using a combination of trace-driven and cycle-accurate full-system simulations of a scale-out pod using *Flexus* [37]. Flexus extends the *Simics* functional simulator with timing models of out-of-order cores, caches, on-chip protocol controllers, interconnect, and DRAM. Flexus models the SPARC v9 ISA and is able to run unmodified operating systems and applications. The details of the simulated architecture are listed in Table 3.

Our trace-based analyses use memory access traces collected from Flexus with in-order execution, no memory system stalls, and a fixed IPC of 1.0. For each workload, we collect a trace of 20-40 billion instructions per core and use half of the trace for warm-up prior to collecting experimental results. For cycle-accurate simulations, we use the SMARTS sampling methodology [38]. Our samples are drawn over an interval of 10 seconds of simulated time, with 400-800 equidistant measurements. For each measurement, we launch simulations from checkpoints with warmed caches and branch predictors, and run 300K cycles (2M cycles for Data Serving) to achieve a steady state of detailed cycle-accurate simulation prior to collecting measurements for the subsequent 150K cycles (400K for Data Serving). To measure the performance of scale-out workloads, we use the ratio of the aggregate number of application instructions committed (i.e., summed over the 16 cores) to the total number of cycles (including the cycles spent executing operating system code); this metric has been shown to accurately reflect overall system throughput [37]. For the multiprogrammed workload, we calculate the IPC for each core independently and report the geometric mean. Performance measurements are computed at a 95% confidence level and an average error below 3%. To model on-chip and off-chip DRAM performance and power, we use two separately adapted and configured instances of DRAMSim2 [31], parametrized with data borrowed from commercial DDR3 device specifications. We report all results for one 16-core pod.

6. RESULTS

6.1 Spatial Characterization

To understand the behavior of scale-out workloads in the context of large caches, first, we examine their spatial characteristics. Figure 4 shows the page density for each workload for a page size of 2KB while varying the cache capacity. We define page density as the number of demanded 64-byte blocks within the page.

Not only do scale-out workloads exhibit high page density, but they also show increase in page density as the cache capacity increases. This can be contributed to the longer on-chip residency of pages, which at larger cache sizes reaches hundreds of milliseconds, leaving more time for data to be accessed within a page. As the cache grows in capacity, the number of high-density pages increases while the number of low-density pages decreases, result-

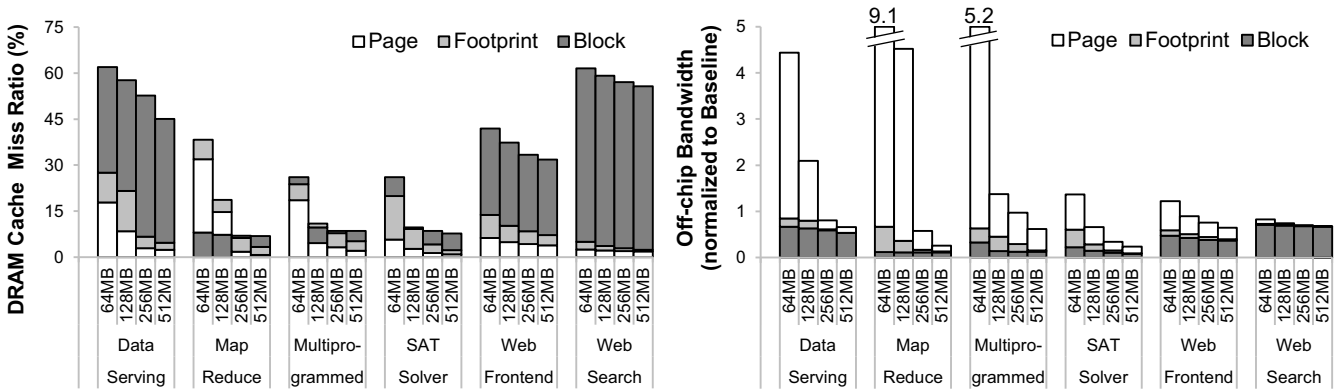

Figure 5. (a) Miss ratio and (b) off-chip bandwidth requirements (normalized to a baseline system without a DRAM cache) of block-based, Footprint, and page-based cache organizations.

ing in a bimodal distribution for some of the workloads. The multiprogrammed desktop workload, on the contrary, does not show a regular trend. Our analysis shows that a 512MB cache captures its working set, most of the dense pages being cache-resident, while the pages that are constantly fetched and evicted exhibit lower density.

The increase in page density with the cache capacity has an interesting implication on the footprint predictor effectiveness. Compared to the prior work [34] that uses a similar predictor to prefetch into block-based L1 caches, our predictor is more effective due to (1) the higher prediction opportunity at this cache level (more blocks to predict per each obligatory page miss), and (2) a larger fraction of fully-accessed pages that are easier to identify due to their simple access patterns (sequential accesses).

However, not all the workloads show high page density. In fact, Figure 4 demonstrates wide variations in workloads' spatial locality. Thus, no single fetch unit size can simultaneously exploit the available spatial locality while using bandwidth and storage efficiently [34]. Among the workloads with lower page density, the highest fraction of the pages are singleton pages, which only have a single block accessed. While possibly reused in L1 and L2 caches, these blocks are rarely reused in the DRAM cache (less than 5% of the time), resulting in high bandwidth and capacity losses for page-based caches. Footprint Cache successfully detects such pages and avoids their allocation in the cache.

The high degree of spatial locality observed in most of the scale-out workloads implies that page-based caches achieve higher hit ratios compared to the block-based ones; due to their large fetch unit, they always experience a single miss per page. Unfortunately, fetching whole pages is a brute-force approach to achieving high hit ratios and it comes at an unacceptable bandwidth cost, which is the most severe for low-density pages.

6.2 Coverage and Off-Chip Bandwidth

Footprint Cache learns and predicts the footprint of each page and hence it is able to achieve high hit ratios and eliminate the fetch of blocks that will not be accessed during the residency of the page in the cache. Figure 5 compares Footprint Cache with block-based and page-based caches in terms of the miss ratio (Figure 5a) and bandwidth demands (Figure 5b). The bars are plotted in a stacked fashion. For instance, the bar showing the miss ratio at 64MB for Data Serving (Figure 5a) indicates that the miss ratio for the page-based cache, Footprint Cache and block-based cache is 18% (white part), 27% (white and light gray parts together) and 62% (white, light gray and dark gray parts together), respectively. The same representation is used in Figure 5b to represent the off-chip bandwidth demands of the three cache organizations.

For all workloads, the page-based cache achieves up to an order of magnitude lower miss ratio, as expected due to the high page access density. The exception is MapReduce, which at 64 and 128MB shows very low page access density, giving the block-based cache considerable capacity advantages. As expected, Footprint Cache always achieves a miss ratio close to the page-based cache. Only SAT Solver, at smaller caches sizes, shows significantly larger miss ratios compared to the page-based design, but still performing better than the block-based design. The reason is that SAT Solver performs symbolic execution as part of software testing [3] and as such does not have a static, well-structured dataset. On the contrary, it creates its dataset on-the-fly, throughout the whole program execution, which interferes with the prediction mechanism.

On the bandwidth side the situation is the reverse, as Figure 5b demonstrates. The block-based cache achieves the lowest off-chip traffic, while the page-based increases the off-chip traffic by up to an order of magnitude compared to the baseline. Footprint Cache, on the contrary, demands almost the same bandwidth as the block-based design by eliminating most of the unnecessary traffic.

6.3 Performance

Figure 6 compares performance of the three cache designs at various cache sizes for all workloads except Data Serving. We plot the results for Data Serving in Figure 7 due to the large difference in the scale, caused by the excessive bandwidth requirements of this workload. The block-based design provides the greatest initial performance boost at 64MB, which is mostly contributed to the significant cut in off-chip traffic. However, the design fails to deliver considerable further improvements, due to its high and steady miss ratio, as shown in Figure 5a. The page-based design initially suffers from a considerable performance loss due the excessive off-chip traffic it causes. As the cache capacity increases, it quickly recovers due to fewer misses and decreased pressure on off-chip bandwidth. On the contrary, Footprint Cache shows steady performance improvement across all cache sizes, outperforming the other two designs, consistently matching our findings from Figure 5. For some of the workloads, we observe a slight advantage of the block-based design over Footprint Cache at smaller cache sizes, due to its superior capacity management.

As the stacked DRAM cache requires on-chip SRAM storage for the tags (under 2MB for the 512MB stacked cache), we also consider a baseline system with additional L2 capacity to compensate for the difference in total on-chip storage. This enhanced baseline provides negligible benefit on scale-out workloads, as expected based on earlier research results [6, 26].

Across all workloads, Footprint Cache is able to deliver 82% of the

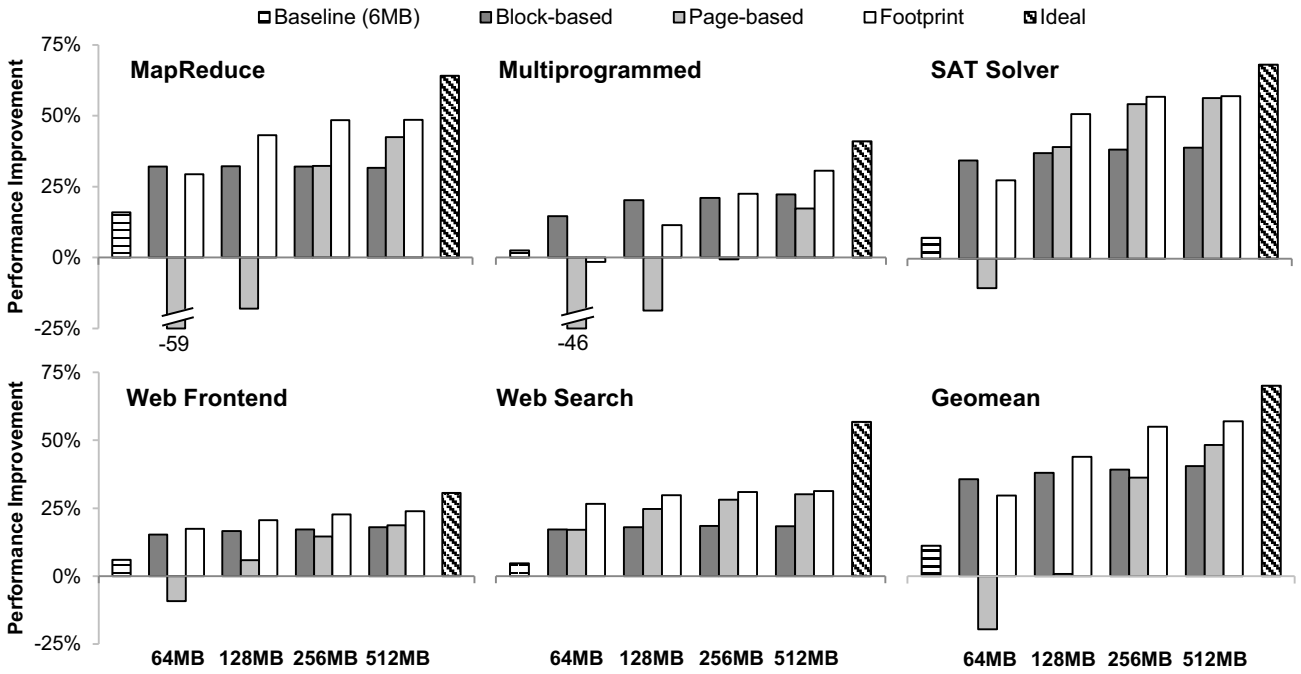

Figure 6. Performance improvement of various designs over the baseline system.

system performance of an Ideal cache — i.e., a cache that never misses and has no tag overheads (die-stacked main memory).

6.4 Sensitivity to Page Size and History Size

Figure 8 compares the predictor accuracy assuming various page sizes, showing a fraction of the blocks that are successfully predicted, the blocks that are not predicted (underpredictions), and the blocks that are overpredicted. While for most of the workloads 1KB and 2KB pages are the best options, larger pages might be desirable as they provide further tag storage reduction. Larger pages, however, require larger footprint history, due to an increase in number of PC & offset combinations per instruction. In this work, we find 2KB to be the sweet spot, considering the trade-off between the accuracy and storage overheads.

Because the Footprint Cache prediction mechanism relies on the missing instruction, its history storage requirements are independent of the dataset size. The prediction history, captured by the FHT, contains only the fraction of the application's instruction working set that causes page misses in the DRAM cache. Thus, its size is small and its content is stable. Figure 9 illustrates the Footprint Cache hit ratio sensitivity to the number of history entries. In this work we assume 16K FHT entries, which require 144KB of SRAM storage, but other trade-offs are possible with, as Figure 9 shows, minimal performance impacts.

6.5 Impact of Capacity Optimization

As we can see from Figure 4, singleton pages account for a quarter of the pages in the cache, on average. Their elimination allows for a proportionate increase in the effective cache capacity, ultimately resulting in a 10% reduction in the miss rate, on average. The miss rate reduction is in accordance with our observation that miss rates for scale-out workloads follow a power low [10] (the miss rate-capacity relationship can be also estimated from Figure 5).

6.6 Energy Implications

Figure 10 compares various designs in terms of dynamic off-chip DRAM energy. All the cache designs use 256MB of DRAM cache. Because the systems differ in performance, and therefore, in the

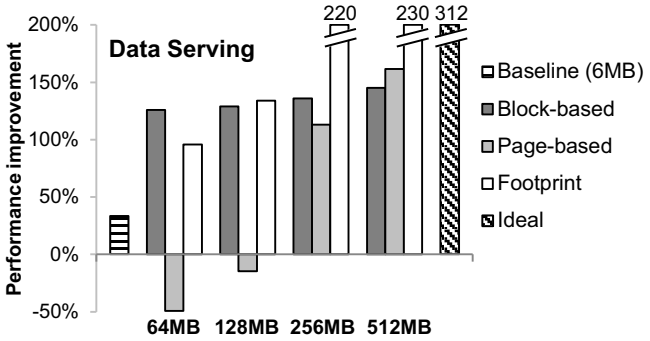

Figure 7. Performance improvement of various designs over the baseline system for Data Serving.

rate at which they access off-chip memory, we present the energy per instruction normalized to the baseline system without a cache. The dynamic energy is broken down into activate/precharge energy, burnt for DRAM row manipulations, and burst energy, spent on reads and writes.

All designs achieve significant energy reduction compared to the baseline system. As expected, the page-based design burns the most burst energy due to its high off-chip traffic per instruction. However, the page-based design exhibits the best DRAM access locality and the highest row-buffer hit ratio, significantly reducing the activate/precharge energy. On the contrary, the block-based design consumes the lowest burst energy due to its low off-chip traffic. However, as almost every read results in a row opening, they exhibit very high activate/precharge energy, which dominates the total dynamic energy.

Footprint Cache delivers the lowest off-chip DRAM energy per instruction. In particular, Footprint Cache is able to reduce both activate/precharge and burst energy thanks to its page organization and its high prediction accuracy, which allows for reducing off-chip traffic significantly. Across all workloads, Footprint Cache reduces the total dynamic off-chip DRAM energy of the baseline by 78%, whereas the block-based and page-based designs reduce the energy by 71% and 69%, respectively.

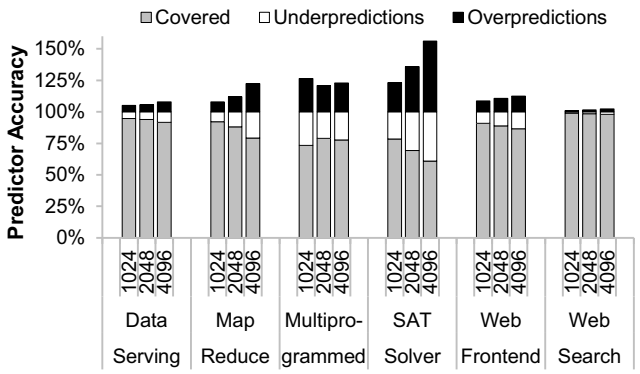

Figure 8. Predictor accuracy sensitivity to the page size, for a 256MB cache with 16K FHT entries.

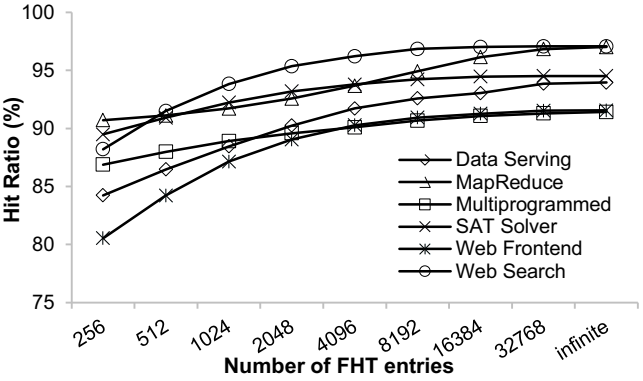

Figure 9. Hit ratio sensitivity to the history size. The DRAM cache capacity is 256MB and the page size is 2KB.

We observe similar trends in stacked DRAM energy consumption. Figure 11 plots stacked DRAM energy consumption for various die-stacked designs normalized to the block-based design. Not surprisingly, the savings in activate/precharge energy for the page-based and Footprint Cache are not as low as in off-chip DRAM, despite the excellent row-buffer locality of cache fills and evictions. The reason is that regular read/write requests show much fewer row-buffer hits for majority of the workloads. Overall, Footprint Cache reduces the total dynamic DRAM energy by 24% compared to the block-based design, whereas the page-based achieves only a 17% reduction.

6.7 Other Page-Based Proposals

We evaluated a recently proposed page-based cache system [13] that tracks the topmost accessed pages, called *hot* pages, that contribute to 80% of the total accesses. Only pages predicted to be hot are allocated in the cache and fetched at page granularity. The prediction is based on the previous history of each page's behavior. The idea behind this approach is that only a small fraction of pages contribute to the majority of cache accesses. However, we could not make the same observation with scale-out workloads due to their vast data set, most of which is randomly distributed across memory, without forming a particular working set. Previous work also noted the same problem [24]. Figure 12 plots the amount of cache needed to capture a desired fraction of total accesses, assuming a perfect predictor and an ideal cache replacement policy and 4KB pages (4KB was found to be the optimal page size [13]). As we can see, even in the idealized case, to capture 80% of the pages we need caches over 1GB. While the proposed mechanism does not work well for our workloads, we find this work important and believe the idea of page-level filtering has a lot of potential for many applications, if equipped with a predictor that is dataset-independent, such as instruction-based ones. In fact, Footprint Cache uses a similar approach to eliminate singleton pages.

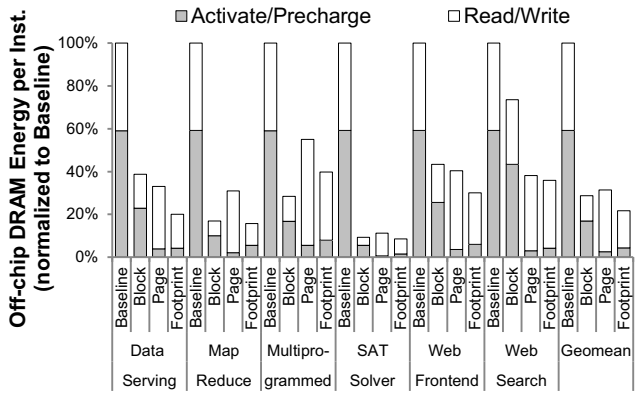

Figure 10. Off-chip DRAM dynamic energy per instruction normalized to the baseline system.

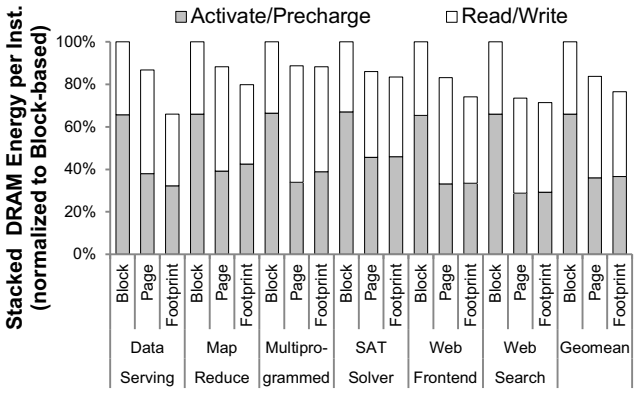

Figure 11. Stacked DRAM dynamic energy per instruction normalized to the block-based design.

7. DISCUSSION

Footprint Cache and coherence. To facilitate the shared memory programming model, contemporary server processors provide hardware-enforced coherence at the chip level. Existing designs enforce coherence at the last-level of the on-chip SRAM cache. Given this organization, the addition of the Footprint Cache does not entail any modifications to the underlying coherence protocol and implementation, as it sits below the level at which coherence is enforced.

In systems with multiple sockets, Footprint Cache can easily provide page-level coherence tracking [1] by extending tag entries with per-page coherence bits. Tracking coherence at fine granularity across sockets is not necessary for server workloads as they share little or no data [6, 9].

Transfer of PC. Footprint Cache relies on the knowledge of instructions that cause page misses. Such information is typically not available in the last-level cache. Therefore, our design must transfer the PC information along with read/write requests through the on-chip network [36]. Because Footprint Cache does not track evictions from the higher-level cache, it does not need to store PC information at any cache level. The transfer of PC information via the on-chip network has no performance implications due to the underutilization of the network [35]. Such transfers, however, do have energy implications. We find that PC transfers increase the on-chip network power by 30mW per pod in the worst case, which is a negligible overhead.

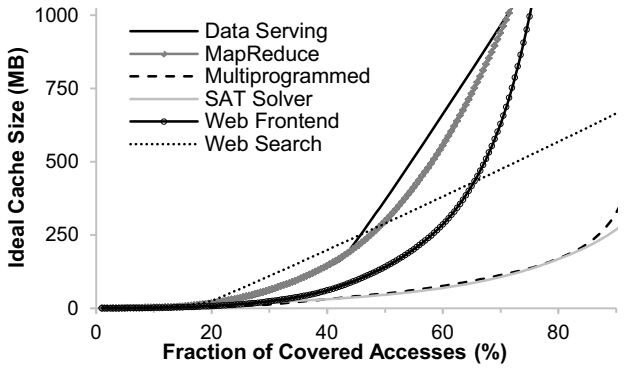

Figure 12. Minimum size of an ideal cache needed to cover a given fraction of cache accesses.

Other processor architectures. We evaluated Footprint Cache in the context of scale-out processors [25, 26]. However, our design is not limited to such an organization. In fact, any many-core chip design that stresses off-chip bandwidth (e.g., Tilera TILE100) would yield similar results. In contrast, processor designs with a handful of large cores (e.g., Intel Westmere) would see less benefit from die-stacked caches as they cannot utilize the available memory bandwidth due to their low degree of on-chip parallelism [6].

Cache capacity. In this work we covered die-stacked DRAM caches ranging from 64-512MB per pod (up to 3GB per chip). However, the datasets of these workloads are scaled down from hundreds of gigabytes to tens of gigabytes to allow for practical full-system simulation. Because miss rates for server workloads follow a power law [10], which we verified for these workloads, the observed miss rate curve will shift to the right for larger datasets. This means that the simulated cache sizes correspond to an order of magnitude larger caches in an industrial-strength setup.

SRAM area overhead. All the designs we discussed, including Footprint Cache, impose a multimegabyte SRAM overhead for tags and other metadata. We assume this area overhead will be compensated by the reduction in the number of off-chip memory channels, for all designs except the page-based, which exacerbates bandwidth demands.

Footprint Cache in non-3D systems. We evaluated Footprint Cache in the context of die-stacked DRAM. However, nothing in this work is 3D-specific, and our design and conclusions remain valid for other forms of high-bandwidth low-latency on-chip DRAM, such as eDRAM [32] or systems integrated via silicon interposer [4].

8. RELATED WORK

Die stacking has been recognized as a powerful technology, thus many researches have tried to exploit the advantages it provides and address the challenges it imposes [8, 27], assuming die-stacked DRAM in form of a main memory [11, 14, 20, 21], DRAM cache [13, 22, 23, 24, 29, 40], or assuming a heterogeneous, or a software-managed extension to off-chip main memory [5]. Other researchers also looked at off-chip DRAM caches for systems with non-volatile main memory [5, 28].

Woo et al. explore spatial locality of desktop applications, concluding that large cache blocks in L2 caches boost performance better than conventional prefetchers, if supported with a high-density TSV bus [11]. Jiang et al. arrived at a similar conclusion for DRAM caches [13]. While most of the researchers agree that large cache blocks are beneficial for overall performance for systems that are not bandwidth-constrained [7, 11, 13], many of them proposed filtering unused data. Lin et al. proposed filtering of unused data coming from aggressive prefetchers [19].

Instruction-based predictors are used extensively in data prefetching [2, 34], dead-block prediction [18], last-write prediction [36], traffic reduction in networks-on-chip [15], and on-chip and off-chip fetch granularity speculation [17, 39]. Our work is conceptually similar to the work of Kumar and Wilkerson [16], who used a similar predictor based on spatial footprints to predict and fetch only useful words within an L1 cache block, and store such words in a decoupled sectored cache [33]. Their predictor, though, relies on the missing instruction and the full missing address, requiring larger history storage and covering only previously accessed data.

9. CONCLUSION

In this paper we presented Footprint Cache, a cache architecture that combines the best aspects of current die-stacked DRAM cache designs, which fall short of achieving the potential of the die-stacking technology. Footprint Cache fully exploits abundant spatial locality of scale-out applications observed in large DRAM caches, without introducing unnecessary off-chip and on-chip traffic. Footprint Cache is able to achieve the hit ratio of page-based designs and stay within the bandwidth requirements of the block-based ones, while fully preserving on-chip and off-chip DRAM locality. Furthermore, the small tag array overhead makes Footprint Cache practical for implementation.

Using full-system, cycle-accurate simulation of scale-out server platforms, we demonstrated that Footprint Cache delivers 57% performance improvement on average, outperforming existing designs, while reducing off-chip DRAM dynamic energy by 78% compared to the baseline system and reducing stacked DRAM dynamic energy by 24% compared to state-of-the-art.

10. ACKNOWLEDGMENTS

The authors would like to thank Boris Grot, Cansu Kaynak, members of the PARSA lab at EPFL and the anonymous reviewers for their insightful feedback on earlier drafts of this paper. This work was partially supported by EuroCloud, Project No 247779 of the European Commission 7th RTD Framework Programme — Information and Communication Technologies: Computing Systems. The authors thank the EuroCloud project partners for their support.

11. REFERENCES

[1] J. F. Cantin, M. H. Lipasti, and J. E. Smith. Improving multiprocessors performance with coarse-grain coherence tracking. In *Proceedings of the 32nd International Symposium on Computer Architecture*, May 2005.

[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 1.0. http://parsa.epfl.ch/cloudsuite.

[4] Y. Deng and W. Maly. Interconnect characteristics of 2.5-D system integration scheme. In *Proceedings of the International Symposium on Physical Design*, Apr. 2001.

[5] 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.

[6] 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.

[7] P. A. Franaszek, L. A. Lastras-Montan, S. R. Kunkel, and A. C. Sawdey. Victim management in a cache hierarchy. *IBM Journal of Research and Development*, 50(4/5):507–523, Jul-Sep. 2006.

[8] M. Ghosh and H.-H. S. Lee. Smart refresh: An enhanced memory controller design for reducing energy in conventional and 3D die-stacked DRAMs. In *Proceedings of the 40th International Symposium on Microarchitecture*, Dec. 2007.

[9] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki. Reactive NUCA: Near-optimal block placement and replication in distributed caches. In *Proceedings of the 36th International Symposium on Computer Architecture*, Jun. 2009.

[10] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki. Toward dark silicon in servers. *IEEE Micro*, 31(4):6–15, Jul-Aug. 2011.

[11] 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.

[12] ITRS. http://www.itrs.net/Links/2011ITRS/Home2011.htm.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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*, Dec. 2012.

[18] A.-C. Lai, C. Fide, and B. Falsafi. Dead-block prediction & dead-block correlating prefetchers. In *Proceedings of the 28th International Symposium on Computer Architecture*, Jun. 2001.

[19] W.-F. Lin, S. K. Reinhardt, D. Burger, and T. R. Puzak. Filtering superfluous prefetches using density vectors. In *Proceedings of the 19th International Conference on Computer Design*, Sep. 2001.

[20] 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.

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

[22] 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.

[23] 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.

[24] G. H. Loh and M. D. Hill. Supporting very large DRAM caches with compound access scheduling and MissMaps. *IEEE Micro*, 32(3):70 –78, May-Jun. 2012.

[25] P. Lotfi-Kamran, B. Grot, and B. Falsafi. NOC-Out: Microarchitecting a scale-out processor. In *Proceedings of the 45th International Symposium on Microarchitecture*, Dec. 2012.

[26] 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.

[27] N. Madan, L. Zhao, N. Muralimanohar, A. Udipiu, R. Balasubramonian, R. Iyer, S. Makineni, and D. Newell. Optimizing communication and capacity in a 3D stacked reconfigurable cache hierarchy. In *Proceedings of the 15th International Symposium on High Performance Computer Architecture*, Feb. 2009.

[28] J. Meza, J. Chang, H. Yoon, O. Mutlu, and P. Ranganathan. Enabling efficient and scalable hybrid memories using fine-granularity DRAM cache management. In *Computer Architecture Letters*, Feb. 2012.

[29] 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.

[30] B. M. Rogers, A. Krishna, G. B. Bell, K. Vu, X. Jiang, and Y. Solihin. Scaling the bandwidth wall: challenges in and avenues for CMP scaling. In *Proceedings of the 36th International Symposium on Computer Architecture*, Jun. 2009.

[31] P. Rosenfeld, E. Cooper-Balis, and B. Jacob. DRAMSim2: A cycle accurate memory system simulator. *Computer Architecture Letters*, 10(1):16 –19, Jan-Jun. 2011.

[32] V. Salapura, J. Brunheroto, F. Redigolo, and A. Gara. Exploiting eDRAM bandwidth with data prefetching. In *Proceedings of the International Conference on Computer Design*, Oct. 2007.

[33] A. Seznec. Decoupled sectored caches: Conciliating low tag implementation cost and low miss ratio. In *Proceedings of the 21st International Symposium on Computer Architecture*, Apr. 1994.

[34] 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.

[35] S. Volos, C. Seiculescu, B. Grot, N. Khosro Pour, B. Falsafi, and G. De Micheli. CCNoC: Specializing on-chip interconnects for energy efficiency in cache-coherent servers. In *Proceedings of the 6th International Symposium on Networks-on-Chip*, May 2012.

[36] Z. Wang, S. M. Khan, and D. A. Jimenez. Improving write-back efficiency with decoupled last write prediction. In *Proceedings of the 39th International Symposium on Computer Architecture*, Jun. 2012.

[37] T. F. Wenisch, R. E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J. C. Hoe. Simflex: Statistical sampling of computer system simulation. *IEEE Micro*, 26:18–31, Jul-Aug. 2006.

[38] 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.

[39] 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.

[40] 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.