

Efficient Footprint Caching for Tagless DRAM Caches

Hakbeom Jang^{†*} Yongjun Lee^{†*} Jongwon Kim[‡] Youngsok Kim[‡]
Jangwoo Kim[‡] Jinkyu Jeong[‡] Jae W. Lee[‡]

[†]Sungkyunkwan University, Suwon, Korea {hakbeom, yongjunlee, kimjongwon, jinkyu, jaewlee}@skku.edu
[‡]POSTECH, Pohang, Korea {elixir, jangwoo}@postech.ac.kr

ABSTRACT

Efficient cache tag management is a primary design objective for large, in-package DRAM caches. Recently, Tagless DRAM Caches (TDCs) have been proposed to completely eliminate tagging structures from both on-die SRAM and in-package DRAM, which are a major scalability bottleneck for future multi-gigabyte DRAM caches. However, TDC imposes a constraint on DRAM cache block size to be the same as OS page size (e.g., 4KB) as it takes a unified approach to address translation and cache tag management. Caching at a page granularity, or page-based caching, incurs significant off-package DRAM bandwidth waste by over-fetching blocks within a page that are not actually used. Footprint caching is an effective solution to this problem, which fetches only those blocks that will likely be touched during the page's lifetime in the DRAM cache, referred to as the page's *footprint*.

In this paper we demonstrate TDC opens up unique opportunities to realize efficient footprint caching with higher prediction accuracy and a lower hardware cost than the original footprint caching scheme. Since there are no cache tags in TDC, the footprints of cached pages are tracked at TLB, instead of cache tag array, to incur much lower on-die storage overhead than the original design. Besides, when a cached page is evicted, its footprint will be stored in the corresponding page table entry, instead of an auxiliary on-die structure (i.e., Footprint History Table), to prevent footprint thrashing among different pages, thus yielding higher accuracy in footprint prediction. The resulting design, called *Footprint-augmented Tagless DRAM Cache (F-TDC)*, significantly improves the bandwidth efficiency of TDC, and hence its performance and energy efficiency. Our evaluation with 3D Through-Silicon-Via-based in-package DRAM demonstrates an average reduction of off-package bandwidth by 32.0%, which, in turn, improves IPC and EDP by 17.7% and 25.4%, respectively, over the state-of-the-art TDC with no footprint caching.

1. INTRODUCTION

Die-stacked DRAM technologies have been widely embraced by industry as a means to overcome the long-standing “Memory Wall” problem [1]. A typical 3D form factor stacks 4–8 DRAM dies with an optional logic die at the bottom [2], whose capacity ranges from hundreds of megabytes to several gigabytes and will continue to scale up with technology scaling [3]. Major processor vendors

have announced their plans to integrate these technologies into their products, including Intel [4], AMD [5], IBM [6], Nvidia [7], and Xilinx [8].

There are proposals to architect this high-bandwidth in-package DRAM as large, software-transparent last-level caches [3, 9, 10, 11, 12, 13, 14, 15]. Compared to software-managed fast main memory they have an advantage in easy deployment without requiring any modification to the software stack. Since the cost of cache tags easily becomes prohibitive with large in-package DRAM, minimizing this overhead in terms of latency, storage, and energy consumption, is one of the main design challenges.

Recently, Tagless DRAM Caches (TDCs) [15] have been proposed to completely eliminate tagging structures from both on-die SRAM and in-package DRAM. TDC aligns the granularity of caching with OS page size (e.g., 4KB) and replaces the two-step address translation by TLB (for virtual-to-physical address translation) and cache tag array (for physical-to-cache address translation) into a single-step process. At TLB miss the TLB miss handler performs not only a page table walk but also cache block allocation, and directly stores the virtual-to-cache address mapping into the TLB. Since a TLB access immediately returns the exact location of the requested block in the cache, cache tags are no longer maintained. By eliminating the cache tags TDC achieves lowest hit latency and best scalability with ever-increasing DRAM cache size.

However, TDC imposes a constraint on the DRAM cache block size to be the same as OS page size and suffers an *over-fetching* problem. Caching at a page granularity incurs significant off-package bandwidth waste by unnecessarily fetching those blocks that are not actually used, which is a common problem of page-based caches.

Footprint caching [9, 10] is an effective solution to this over-fetching problem for page-based caches. Both Footprint Cache [10] and Unison Cache [9] track referenced blocks in a cached page since it is allocated. They use this information to fetch only the relevant subset of the 64-byte blocks, referred to as the page's *footprint*, when the page is reallocated into the cache. Thus, they maintain a high hit rate of page-based caching while mitigating the bandwidth waste by not fetching blocks that will not be used.

We identify several sources of inefficiency in the original footprint caching scheme [10], which can be effectively addressed on TDC. First, *footprint thrashing* can occur among different pages as the footprint of an evicted page is stored in a small Footprint History Table, which covers only a small fraction of the working set. Second,

*These authors contributed equally to this work.

the bit vectors for footprint tracking are stored in the cache tag array, whose storage overhead is proportional to the DRAM cache size. Finally, an in-package DRAM cache miss is known only after the request goes through multiple on-die cache misses to yield a relatively high miss penalty.

In this paper we propose *Footprint-augmented Tagless DRAM Cache (F-TDC)*, which synergistically combines footprint caching with TDC to improve its bandwidth efficiency. Exploiting unique opportunities offered by TDC, F-TDC outperforms the original footprint caching scheme [10] by alleviating its sources of inefficiency. Since there are no cache tags in TDC, the footprints of cached pages are tracked at TLB, instead of cache tag array, to reduce on-die storage overhead. Besides, when a page is evicted, its footprint will be stored in the corresponding page table entry (PTE), instead of Footprint History Table, to prevent footprint thrashing among different pages, thus yielding higher prediction accuracy. Finally, F-TDC reduces an off-package miss penalty by identifying an off-package block miss right after a TLB access. Our evaluation with 3D Through-Silicon Via (TSV)-based in-package DRAM demonstrates an average reduction of off-package bandwidth by 32.0%, which, in turn, improves the IPC and EDP by 17.7% and 25.4%, respectively, over the state-of-the-art TDC with no footprint caching.

The main contributions of this paper are summarized as follows:

- This paper is the first to identify unique opportunities in TDC, which enable efficient realization of footprint caching with higher prediction accuracy at a lower hardware cost than the original design [10].
- A Footprint-augmented Tagless DRAM Cache (F-TDC) is designed and implemented to capitalize on these opportunities.
- Detailed evaluation of F-TDC with SPEC CPU 2006 and CloudSuite is provided using a cycle-level simulator to demonstrate its effectiveness in reducing bandwidth waste and improving the performance and energy efficiency.

2. BACKGROUND AND MOTIVATION

2.1 Tagless DRAM Cache

Existing DRAM caches can be classified into two categories based on the granularity of caching: block-based [3, 12, 13, 14] and page-based [9, 10, 11, 15]. Block-based DRAM caches adopt conventional cache block size (e.g., 64B) for efficient utilization of DRAM cache space and off-package bandwidth. However, they require large storage for tags; assuming an 8-byte cache tag per 64-byte cache block, a 2GB DRAM cache requires 256MB storage just for tags. It is too expensive to place such a large SRAM array on the processor die. Therefore, block-based caches often put the tags, together with cached data, into in-package DRAM [3, 13], but only at the cost of an increase in hit latency, which negatively affects performance.

As an alternative, page-based DRAM caches alleviate this problem by caching at a page granularity, typically ranging from 1–8 KB [9, 10]. Smaller tags result in lower storage overhead and hit latency. Large block size often leads to higher cache hit rate by exploiting spatial locality. Besides, energy efficiency is improved by exploiting

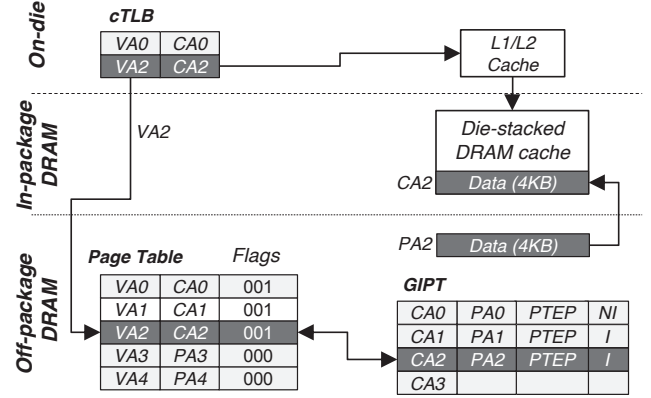


Figure 1: Organization and operations of Tagless DRAM Cache (TDC)

row buffer locality in DRAM devices. However, while much lower than block-based DRAM caches, the storage requirement for tags is still significant even for page-based caches with ever-increasing DRAM cache size. For example, assuming 4KB pages, an 8GB DRAM cache requires either tens of megabytes on-die SRAM [10] or hundreds of megabytes [9] in-package DRAM storage just for tags.

Recently, Tagless DRAM Caches (TDCs) [15] have been proposed to completely eliminate tagging structures from both on-die SRAM and in-package DRAM. Figure 1 shows the organization and operations of TDC. At TLB miss the TLB miss handler performs a page table walk to fetch the page table entry (PTE). If the PTE indicates that the page is currently not cached, the TLB miss handler allocates the page into the DRAM cache. Once the cache fill request is completed, the global inverted page table (GIPT) is updated, which maintains cache-to-physical address mappings (along with a pointer to the corresponding PTE (PTEP)) for all cached pages. Finally, the TLB miss handler updates both the TLB and the PTE with the newly created virtual-to-cache address mapping, and returns. Thus, an access to the memory region within the TLB reach always hits in the cache with low hit latency as a TLB access immediately returns the exact location of the requested block in the cache, hence obviating the needs to maintain cache tags. By consolidating two-step address translation in the cache access path into a single-step process, TDC achieves lowest hit latency by saving a tag-checking operation.

By completely eliminating cache tag array TDC is the most scalable caching solution known to date. However, TDC imposes a constraint on the cache block size to be aligned with OS page size (e.g., 4KB) and suffers an over-fetching problem, which is common to page-based caches.

2.2 Over-fetching Problem in Tagless DRAM Caches

Page-based caches, including Tagless DRAM Caches (TDCs), commonly have an over-fetching problem, where unused blocks are moved on and off the package to cause bandwidth and capacity wastes. Figure 2 compares the bandwidth consumption of the conventional page-based DRAM cache running a multi-programmed workload, with an oracle cache, which fetches only those blocks that will be used. The results demonstrate the existence of sig-

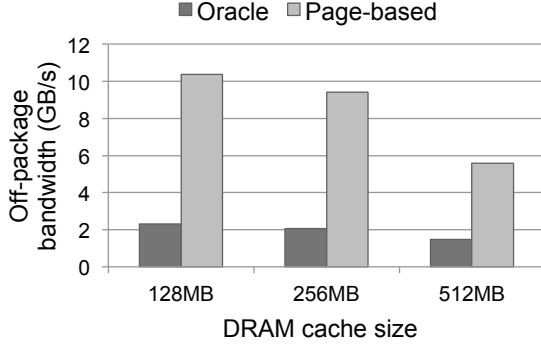


Figure 2: Off-package bandwidth usage for oracle and page-based caches while running a multi-programmed workload (MIX 1 in Table 4)

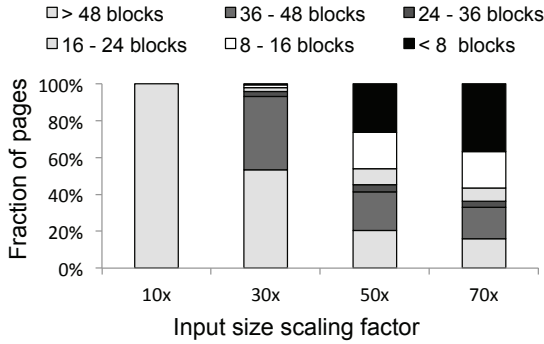


Figure 3: Page access density for 2GB DRAM cache with varying input size

nificant bandwidth bloat in the page-based DRAM cache, which is more pronounced for smaller caches due to higher cache miss rate—by up to $5\times$ over the oracle cache.

The over-fetching problem can manifest even for a large DRAM cache if the working set outgrows the cache size. For example, Figure 3 shows the page access density of Memcached in CloudSuite [16] with a 2GB DRAM cache. (Refer to Section 4 for the details of simulation methodology.) The access density of a cached page is defined as the number of referenced blocks during the page’s lifetime in the cache from allocation to eviction [17]. When the input is relatively small (with a $10\times$ input scaling factor), the working set fits in the cache, and ample spatial locality is observed in cached pages. However, as the input size scales up, the page access density quickly drops to make a majority of blocks not used in the cache. With a $70\times$ input scaling factor, about 40% of cached pages are sparse with eight or fewer blocks referenced (out of 64 blocks) since frequent capacity misses shorten the lifetime of a cached page, often shorter than the page’s reuse distance. These sparse pages cause significant bandwidth and capacity wastes in page-based DRAM caches. In short, this over-fetching problem can still exist even for multi-gigabyte DRAM caches if applications have huge memory footprints (e.g., in-memory database [18], genome assemblies [19]).

To address this problem, Jevdjic et al. have recently proposed Footprint Cache [9, 10] to improve the bandwidth efficiency of page-based DRAM caches. Footprint Cache uses page-sized allocation units, but identifies and fetches

only those cache blocks within a page that will likely be used during the page’s lifetime. Footprint Cache effectively reduces bandwidth bloat and improves the performance and energy efficiency of page-based DRAM caches.

However, the idea of footprint caching is not readily applicable to TDC since Footprint Cache extends cache tags to track the footprint of a cached page, while TDC eliminates all tagging structures. Besides, there are several sources of inefficiency in the original caching scheme. For example, an 8GB Footprint Cache requires either a 48MB SRAM tag array [10] or up to 512MB DRAM storage in package [9], along with several auxiliary on-die structures, such as footprint history table, singleton table, and way predictor. This paper proposes Footprint-augmented Tagless DRAM Cache (F-TDC), which integrates a low-cost footprint caching scheme into TDC, to yield higher prediction accuracy and scalability than the original design.

3. FOOTPRINT-AUGMENTED TAGLESS DRAM CACHE (F-TDC)

3.1 Overview

F-TDC builds on Tagless DRAM Cache (TDC) [15] to employ cache-map TLB (cTLB), which replaces the conventional TLB to store virtual-to-cache, instead of virtual-to-physical, address mappings. At a TLB miss, the TLB miss handler performs not only the page table walk but also cache block allocation into the DRAM cache, and updates both cTLB and page table with the virtual-to-cache mapping.

For footprint tracking and prediction, both cTLB and page table are augmented with the following bit vectors:

- **Valid bits:** Valid bits indicate which blocks in the page are currently valid in the DRAM cache. When a new block is fetched, the corresponding valid bit is set to one. If the page is evicted from the cache, all of its valid bits are reset to zero.
- **Reference bits:** Reference bits indicate which blocks in the page have been accessed since the page is allocated into the DRAM cache. When the page is reallocated into the cache, these bits are used as fetching hints as they represent the page’s footprint. Upon the completion of cache fill they are reset to zero.

F-TDC decouples cache fill unit from cache allocation unit. This decoupling introduces a new type of cache miss, called *block miss*, which occurs when the page is already allocated but the requested block is not filled yet (i.e., its valid bit is zero). Note that there is only one type of cache miss in the original TDC, which we call *page miss*; TDC guarantees that, once a page is allocated, all of its blocks are valid.

Figure 4 shows the flow chart of the memory access path of F-TDC, with the modified parts from the original TDC shaded in gray. The operations of the memory access path in F-TDC are sketched as follows:

- Upon a TLB miss, the TLB miss handler performs a page table walk to fetch the page table entry (PTE). If the page being accessed is already allocated in the cache, the TLB miss handler simply returns the PTE

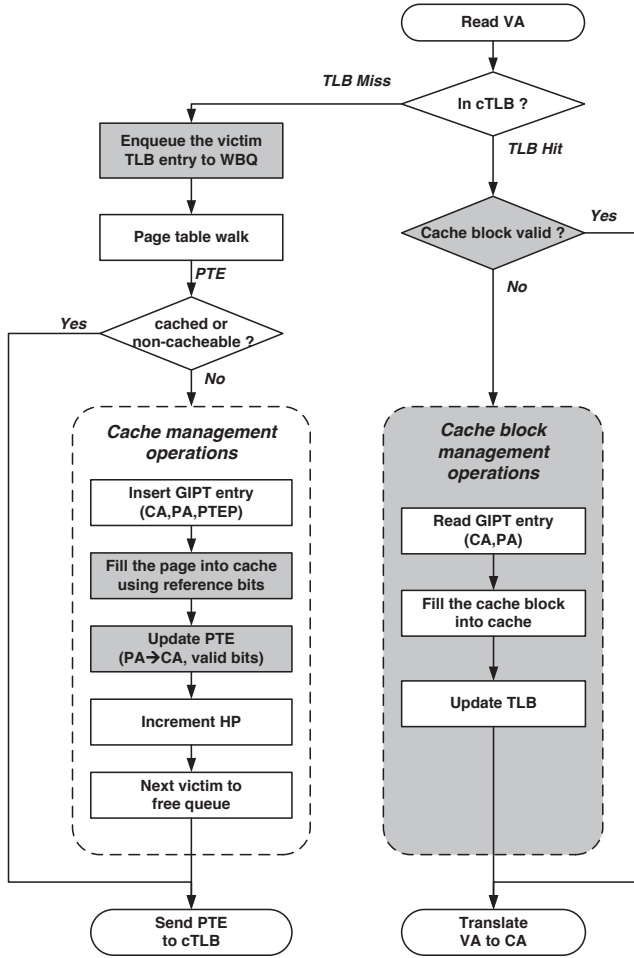


Figure 4: Flow chart of F-TDC operations

to update the cTLB. If not cached, it allocates a free page in the cache, generates a cache fill request to copy the requested page into the cache. Note that, only those blocks whose reference bits are set to one (i.e., the page’s predicted footprint) are copied over to reduce off-package bandwidth consumption. The global inverted page table (GIPT), which maintains cache-to-physical mappings for all cached pages, is also updated with the new entry. Once completed, the TLB miss handler replaces the target PTE with the new virtual-to-cache address mapping and returns the PTE. The reference bits and valid bits of the victim TLB entry are written back to its corresponding PTE asynchronously through write-back queue (WBQ).

- If the requested page misses in the cTLB but hits in the DRAM cache (i.e., victim hit), the TLB miss handler updates the cTLB with the virtual-to-cache address mapping obtained from the corresponding PTE, along with its valid and reference bit vectors.
- At a TLB hit, the valid bit of the requested block is checked. If the block is valid, the memory access percolates through the memory hierarchy (e.g., L1, L2 and DRAM caches) and is serviced in package. The reference bit of the requested block is set to track the page’s footprint.

- If the valid bit is not set (i.e., block miss), the physical address of the missed block is obtained by looking up the GIPT. Then, the requested block is fetched from off-package DRAM, and both valid bit and reference bit of the requested block are updated.
- Cache block eviction is performed asynchronously in the same way as the original TDC [15]. To take the write-back overhead off from the cache access path, F-TDC has a small number of free blocks always available. The PTE of an evicted cache block is updated to restore the original virtual-to-physical address mapping by consulting the GIPT. The valid bits of the corresponding PTE are also cleared.

3.2 Overall Structure

Figure 5 shows the overall structure of the proposed F-TDC and data flows among components. There are three major hardware components of F-TDC: cTLB (with WBQ), page table, and GIPT. Among them, GIPT is not modified from TDC, and the only modification for the page table is inclusion of the two bit vectors (valid and reference bits) as explained before. Hence, we focus on discussion of cTLB and WBQ in this section.

In addition to maintaining virtual-to-cache address mappings, cTLB also tracks a page’s footprint. For this a TLB entry is augmented with valid bits and reference bits. For every memory access, the corresponding valid bit is tested and reference bit is updated. A block miss can be detected by a simple combinational logic as shown in Figure 6.

When a TLB entry is evicted, its reference and valid bits should be spilled into the corresponding PTE, which introduces a new writeback path from cTLB to page table. To take the writeback path off from the critical path of TLB miss handling, we introduce a simple TLB WBQ, which buffers writeback commands. The entries in WBQ are written back asynchronously. By using WBQ, the writeback path adds negligible overhead to the TLB miss handling latency.

The storage cost of the two bit vectors is very low. Assuming 32 entries for L1 TLBs (both instruction and data TLBs) and 512 entries for unified L2 TLBs, the storage overhead is just 1.125KB per core with 8-bit reference/valid vectors (our default). Refer to Section 3.4 for the selection of bit vector width and Section 5.1 for evaluation results. Note that this is the only extra space overhead introduced by F-TDC, which is highly scalable with ever-increasing DRAM size.

Finally, an update to the same reference and valid bits may occur in multiple cores. This may cause a coherence problem of those bits. Actually, the inconsistency of *reference bits* across cores causes no problem as the reference bit vectors can be merged when they are written back to the page table. However, the inconsistency of *valid bits* is problematic for a block being fetched by multiple cores redundantly. If such a block is modified by one or more cores, its value might get corrupted. To address this problem, F-TDC adopts a recent proposal to exploit a cache coherence network for keeping TLBs coherent via broadcasting [20]. Note that a TLB update command is issued only at a block miss, which relatively rarely occurs, hence incurring negligible bandwidth pollution in the cache coherence network.

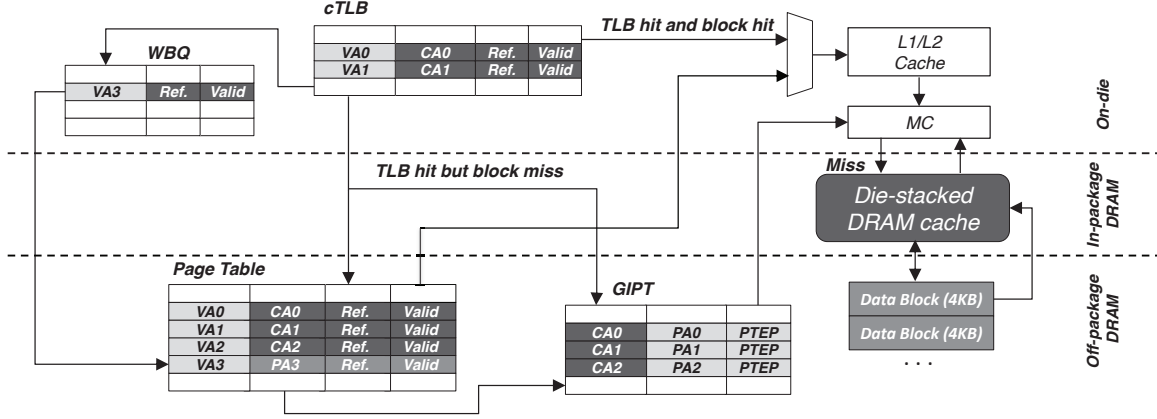


Figure 5: The overall structure of F-TDC

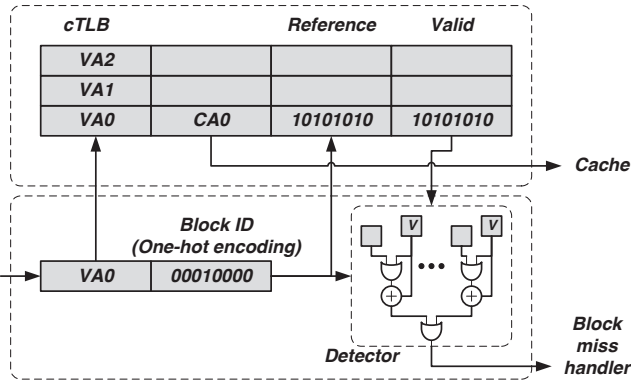


Figure 6: Modified cTLB

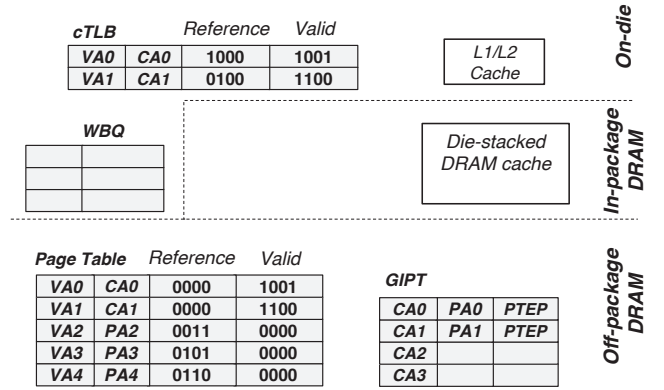


Figure 7: Initial state of F-TDC

3.3 F-TDC Operations

To demonstrate the cache operations of F-TDC, we walk through all cache access scenarios using a running example. Figure 7 depicts a snapshot of the initial state for the running example, and Figure 8 shows the new state after each operation completes. The dark-shaded blocks indicate those components which are just updated. We use a history-based algorithm for footprint prediction. The reference bit vector of each page records touched cache blocks during the lifetime of the page in the DRAM cache. When a TLB entry is evicted, its bit vector is spilled into the corresponding PTE and used as prefetching hints at reallocation of the page as it represents the page's footprint.

Case 1: TLB Miss & Cache Miss (Page Miss)

Figure 8(a) depicts how F-TDC operates when a memory access misses at both the cTLB and the cache. Assuming the initial state as shown in Figure 7, a TLB access for VA2 causes a TLB miss as an entry for VA2 does not exist in the cTLB ❶. To serve the TLB miss, the TLB miss handler allocates a TLB entry for VA2. In case that cTLB is full, it evicts an existing TLB entry into WBQ (e.g., the TLB entry for VA1) ❷. WBQ later writes back the bit vectors of the evicted TLB entry into the page table in an asynchronous manner. Meanwhile, the TLB page miss handler performs a conventional page table walk to fetch the PTE for VA2 ❸. The fetched PTE indicates that the corresponding page is not currently cached (with a VA2-to-PA2 map-

ping); then the TLB miss handler generates a cache fill request to selectively fetch the page's footprint from off-package DRAM into the in-package DRAM cache using the *reference bits* ❹. Once the cache fill request is completed, the TLB miss handler updates the PTE for VA2 to store the VA2-to-CA2 mapping and backs up the original VA2-to-PA2 mapping at GIPT ❺. Finally, the reference bits of the page are copied over into the *valid* bit vector of the TLB entry along with the VA2-to-CA2 mapping. The reference bit vector is cleared to track the page's footprint during the page's sojourn time in the cache ❻.

Case 2: TLB Hit & Cache Hit

When a memory access hits in both cTLB and the cache as shown in Figure 8(b), F-TDC achieves the lowest access latency by requiring neither a page table walk nor a TLB miss handling. As the first step, F-TDC retrieves the TLB entry for the requested memory address from cTLB ❶. Then, it marks the corresponding reference bit to one in cTLB and checks the corresponding valid bit ❷. The reference bit vector serves as the page's footprint to be used when the page is reallocated. Note that both read and write accesses update the reference bit as reference bits track which blocks of a page have been accessed while the page resides in the cache. Once the validity of the requested block is confirmed, the cache address (CA2 in this example) is used to access the block ❸.

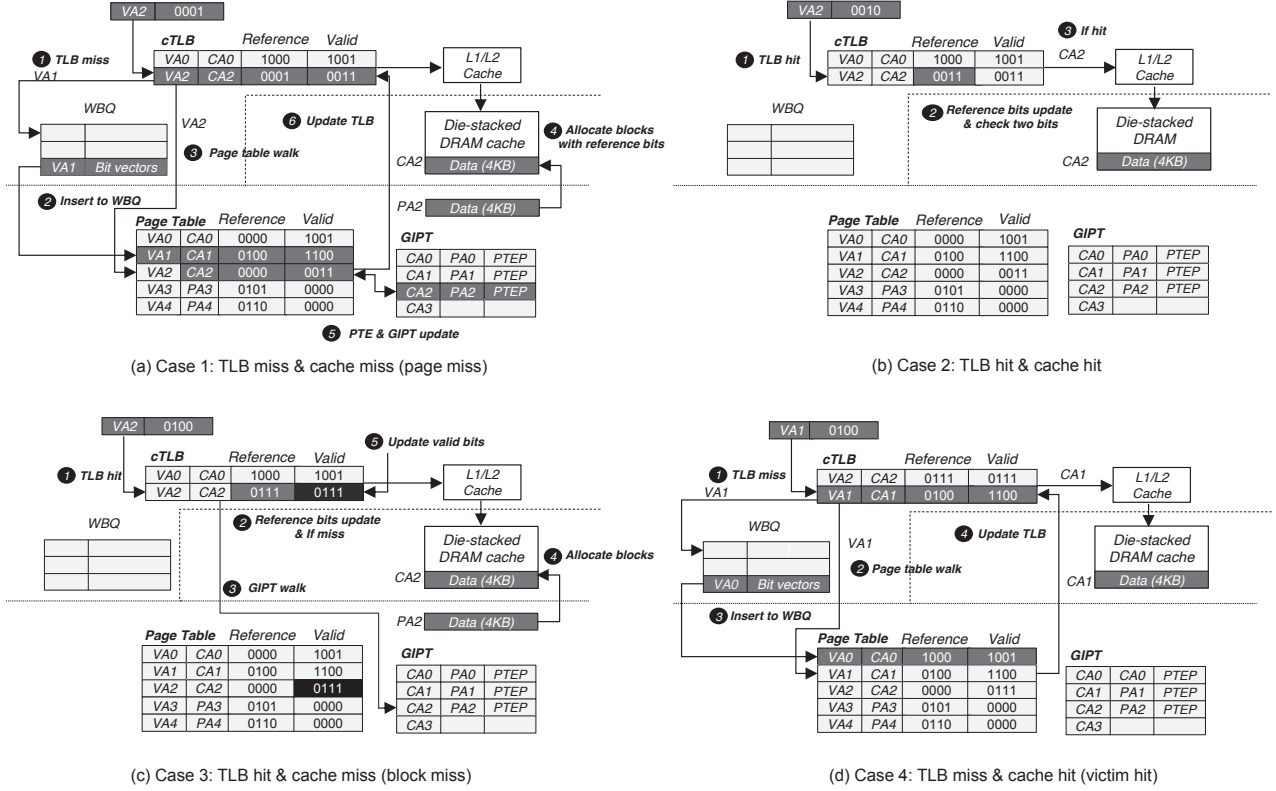


Figure 8: Four operations of F-TDC

Case 3: TLB Hit & Cache Miss (Block Miss)

The third case, in which a memory request hits in the cTLB but misses in the cache, can occur due to a footprint misprediction. Figure 8(c) shows how F-TDC handles such a case. Once a memory access to VA2 hits in the cTLB ①, the corresponding reference bit in the TLB entry is updated and the valid bit is checked ②. Since the requested block is not in the cache (i.e., valid bit is zero), the TLB miss handler looks up the GIPT to retrieve PA2, the physical address of the requested page ③. The cache fill unit uses this address to copy the requested block into the DRAM cache ④ and updates the corresponding valid bit to one ⑤. Then the memory request is retried to hit in the cache.

Case 4: TLB Miss & Cache Hit (Victim Hit)

A *victim hit* occurs when a memory access misses in the cTLB but hits in the cache. This case happens as F-TDC uses the in-package DRAM region beyond the TLB reach as victim cache. Figure 8(d) illustrates the operations of F-TDC for such memory accesses. When a TLB miss occurs for VA1 ①, the TLB miss handler performs a page table walk to fetch the requested PTE ② along with an optional eviction of a TLB entry to WBQ ③. Once the VA0-to-CA0 mapping is restored in the cTLB ④, the request is retried to hit in the cache since the requested block already resides in the DRAM cache. Note that this process does not involve any latency penalty except for TLB miss penalty, which must be paid to handle the TLB miss, anyway.

In summary, Table 1 lists the four memory access operations in F-TDC with their latency implications. F-TDC significantly reduces hit latency at the cost of a slight increase in miss penalty for both page and block misses.

cTLB	DRAM cache	Descriptions
Hit	Hit	Cache hit. Zero latency penalty.
Hit	Miss	Block miss. Costs GIPT access time and off-package block fill time.
Miss	Hit	In-package victim hit. Zero latency penalty (except for TLB miss penalty).
Miss	Miss	Off-package cache miss. Costs cache fill and GIPT update latency.

Table 1: Four possible cases for a memory access

3.4 Selection of Bit Vector Width

Careful selection of the width of *reference* and *valid* bits is important because it affects the system behavior in various ways. The width determines the granularity of data movement. In F-TDC, the cache allocation unit is aligned to the OS page size (e.g., 4KB page). Hence, if the bit width is 8, each bit in the reference/valid bits represent whether 8 adjacent cache blocks (or 512 bytes of data) are referenced/valid in F-TDC. If more bits are used, data will be moved at a finer granularity to mitigate the over-fetching problem at the cost of a lower hit rate. If fewer bits are used, data will be moved at a coarser granularity to waste bandwidth, but the cache hit rate will be improved due to the spatial locality in block accesses.

Another factor affected by the bit width selection is whether the two bit vectors can be packed in a PTE. Since our scheme does not maintain either SRAM tags or footprint history table, both bit vectors should be spilled into the PTE when a TLB entry is evicted. Wide bit vectors

Component	Parameters
CPU	Out-of-order, 4 cores, 3GHz
L1 TLB	32I/32D entries per core
L2 TLB	512 entries per core
L1 cache	4-way, 32KB I-cache, 32KB D-cache, 64B line, 2 cycles
L2 cache	16-way, 2MB shared cache per core, 64B line, 6 cycles
SRAM-tag Array	16-way, 256K entries
In-package DRAM (128~512MB)	
Bus frequency	1.6GHz (DDR 3.2GHz)
Channel and Rank	1 channel, 2 ranks
Bank	16 banks per rank
Bus width	128 bits per channel
Off-package DRAM (8GB)	
Bus frequency	800MHz (DDR 1.6GHz)
Channel and Rank	1 channel, 2 ranks
Bank	64 banks per rank
Bus width	64 bits per channel

Table 2: Architectural parameters

Parameter	In-package DRAM	Off-package DRAM
I/O energy	2.4pJ/b	20pJ/b
RD or WR energy without I/O	4pJ/b	13pJ/b
ACT+PRE energy (4KB page)	15nJ	15nJ
Activate to read delay (tRCD)	8ns	14ns
Read to first data delay (tAA)	10ns	14ns
Activate to precharge delay (tRAS)	22ns	35ns
Precharge command period (tRP)	14ns	14ns

Table 3: Parameters for 3D in-package DRAM and off-package DRAM devices (adapted from [22])

cannot be packed in a PTE and require additional memory space.

By default F-TDC uses 8 bits for each bit vector for two reasons. First, the bit width sensitivity test reveals that it is a choice that balances prediction accuracy and bandwidth waste (see details in Section 5.1). Second, 16 bits of both vectors can be packed in a PTE practically. In many 64-bit CPU architectures, the width of each PTE is 64 bits but part of the bits are used for virtual to physical address translation. For example, a recent Intel 64-bit processor [4] supports 48 bits virtual to 46 bits physical address translation but each PTE is of size 64 bits. Accordingly, 18 most significant bits in each PTE are unused, and 16 bits for the valid and reference bit vectors can be packed in a PTE without requiring extra memory space.

4. EXPERIMENTAL SETUP

We use McSimA+ simulator [21] to evaluate the performance and energy efficiency of F-TDC. Our system model consists of four out-of-order CPU cores, an in-package DRAM cache, and an 8GB off-package DRAM-based main memory. The in-package DRAM cache serves as the L3 cache, which is directly connected to the CPU die with TSV channels. Three different cache sizes are considered—128MB, 256MB, and 512MB. We configure the in-package DRAM to have $4\times$ larger bandwidth than the off-package DDR3 DRAM. Table 2 summarizes the architectural parameters used for evaluation.

We extract the timing and power of the CPU cores and SRAM-based caches using McPAT [23] and those of

Name	Composition
MIX1	milc, leslie3d, omnetpp, sphinx3
MIX2	milc, leslie3d, soplex, omnetpp
MIX3	milc, soplex, GemsFDTD, omnetpp
MIX4	soplex, GemsFDTD, lbm, omnetpp
MIX5	mcf, leslie3d, lbm, sphinx3
MIX6	mcf, leslie3d, GemsFDTD, omnetpp

Table 4: Composition of multi-programmed workloads derived from SPEC CPU 2006

Application	# Memory Accesses	Total # Instructions	Memory Footprint
Data analytics (MapReduce)	2 billion	29 billion	0.68GB
Data caching (Memcached)	2 billion	38 billion	2.15GB
Graph analytics (Tunkrank)	2 billion	23 billion	1.3GB
Media Streaming	2 billion	28 billion	3.71GB

Table 5: Details of the multi-threaded workloads collected from CloudSuite [16]

the DRAM devices using a modified version of CACTI-3DD [24]. We adapt the parameters from recent work [22]. We recompute the I/O energy for accessing memory channels to be 2.4pJ/b (instead of 4pJ/b in [22]) as we replace silicon interposers with TSV bumps. Table 3 lists the detailed timing and power parameters used for both in-package and off-package DRAM.

We use six multi-programmed workloads and four multi-threaded workloads for evaluation. To construct the multi-programmed workloads, we first collect the 11 most memory-intensive benchmarks from SPEC CPU 2006 by measuring misses per kilo instructions (MPKI). Then, for each memory-bound benchmark, we identify the four most representative program slices using Simpoint [25], each of which has 100 million instructions. Multi-programmed workloads are created by randomly composing four out of those 11 memory-bound benchmarks. Table 4 summarizes the six multi-programmed workloads.

For the multi-threaded workloads, we take four CloudSuite benchmarks with large memory footprints [16] that run on our simulation framework without an error. We use memory access traces [10] collected from a modified version of QEMU (Version 2.4.0) [26]. For each workload, we extract a trace of 2 billion memory reference instructions after initialization to capture realistic behaviors of the workload. The details of the four benchmarks are summarized in Table 5.

We compare the following three DRAM cache designs:

- **Baseline Tagless DRAM Cache (TDC)** models a page-based DRAM cache, which uses 4KB pages for the unit of both allocation and cache fill. Due to data movement at a coarse granularity, the baseline TDC suffers significant bandwidth waste.
- **Block-level Fetching without Footprint (Block-Fetching)** is a simplified version of F-TDC for which a DRAM cache block is allocated at an OS page granularity (i.e., 4KB) but filled at a L1/L2 cache block granularity (i.e., 64 bytes). Footprint caching is disabled. This design minimizes off-package bandwidth waste but shows a poor DRAM cache hit rate for not exploiting spatial locality within a page.

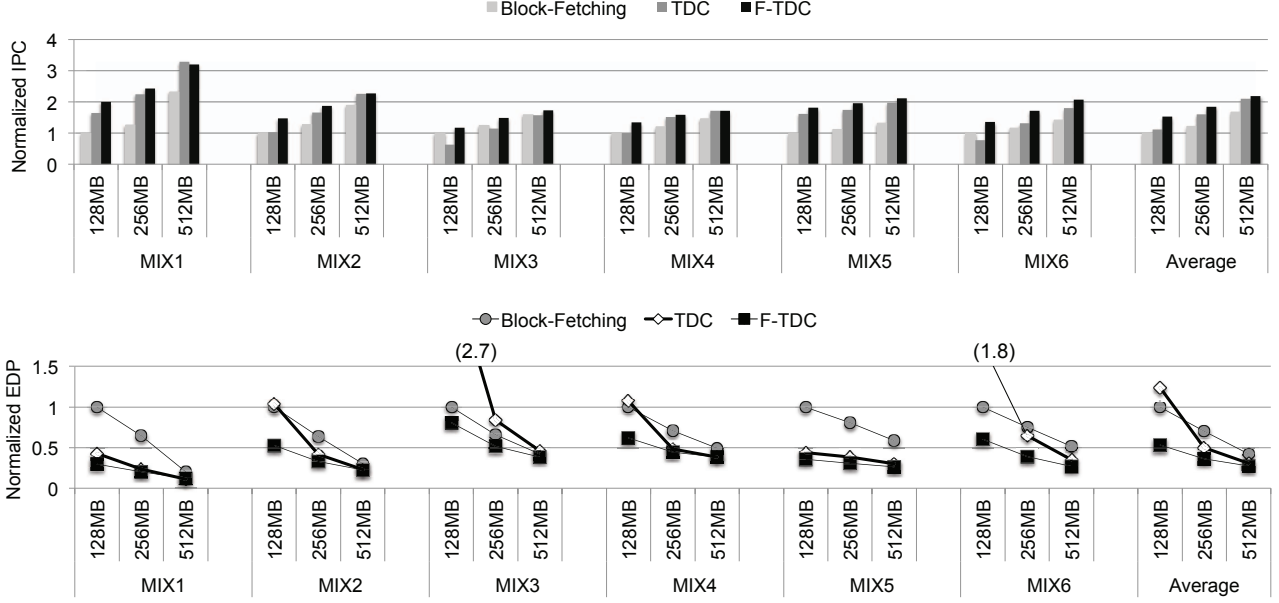


Figure 9: IPC and EDP of the three cache designs. The numbers are normalized to the block-granularity caching scheme (Block-Caching) with 128MB in-package DRAM.

- **Footprint-Augmented Tagless DRAM Cache (F-TDC)** is our proposed DRAM cache design. Initially, F-TDC fetches all blocks for a first-touched page. The allocation unit (4KB) and cache fill unit (8 64-byte blocks by default) are decoupled, and the footprint prediction and prefetching are applied. This cache minimizes bandwidth waste while maintaining a high cache hit rate.

5. EVALUATION

5.1 Multi-programmed Workloads

We first evaluate F-TDC using multi-programmed workloads. Due to cache contentions among four concurrent processes, DRAM cache pressure is significantly amplified compared to single-programmed workloads. Since F-TDC aims to improve system performance especially for workloads with large memory footprints (i.e., much larger than DRAM cache size), we use multi-programmed workloads not only for IPC and EDP measurement but also for comparison against Footprint Cache [10] as well as sensitivity analysis.

IPC and EDP. Figure 9 shows the normalized IPC and EDP of the three cache designs explained in Section 4: Block-Fetching, TDC, and F-TDC. Both TDC and F-TDC generally achieve higher performance than Block-Fetching by better exploiting spatial locality within a page. When the cache size is small (e.g., 128MB), F-TDC significantly outperforms TDC due to higher off-package bandwidth efficiency. That is, F-TDC achieves average IPC improvements by 36.4%, 14.7%, and 4.2%, over TDC, for 128MB, 256MB, and 512MB DRAM caches, respectively. As for energy-delay product (EDP), F-TDC lowers the EDP by 56.9%, 27.0%, and 10.6%, respectively, over TDC. Since a large number of unnecessary off-package memory requests are filtered, fewer blocks are fetched at a cache miss, and a memory request experiences shorter queueing delay at the memory controller. This, in turn, leads to sig-

Memory type	Footprint Cache	F-TDC
On-die SRAM	1.58MB + 144KB	4.5KB
Off-package DRAM	0	0.64MB

Table 6: Storage overhead of Footprint Cache [10] and F-TDC for a 256MB DRAM cache

nificantly lower L3 access latency and energy consumption. We expect the performance gap between F-TDC and TDC to widen as the working set size increases as the over-fetching problem becomes more pronounced. In summary, F-TDC significantly outperforms the other two designs.

Off-Package DRAM Bandwidth. Figure 10 quantifies the amount of off-package DRAM bandwidth reduction by F-TDC, compared to TDC, for three different cache sizes. The results show that F-TDC reduces the bandwidth consumption of TDC by 32.0% on average by efficient footprint caching. F-TDC consistently consumes less off-package DRAM bandwidth than TDC regardless of the cache size. Again, the bandwidth gap widens for smaller cache size, larger footprint, or both. Note that TDC and F-TDC consume a comparable amount of off-package DRAM bandwidth when running MIX1 with a 512MB cache because the working set of MIX1 fits in the cache.

Average L3 Latency. Figure 11 shows the average L3 cache latency of both TDC and F-TDC with varying cache sizes. The latencies are normalized to that of F-TDC with 512MB in-package DRAM, which yields the lowest number. As the in-package DRAM size decreases (towards 128MB), we observe a rapid increase in TDC’s average L3 latency due to frequent page replacement, which causes a lot of bandwidth pollution. In contrast, F-TDC experiences much more gradual increases in latency as it transfers only the useful blocks of a page between in-package and off-package DRAM. The reduced off-package DRAM traffic results in about 40% reduction of the average L3 access latency for F-TDC, compared to TDC, for a 128MB DRAM cache.

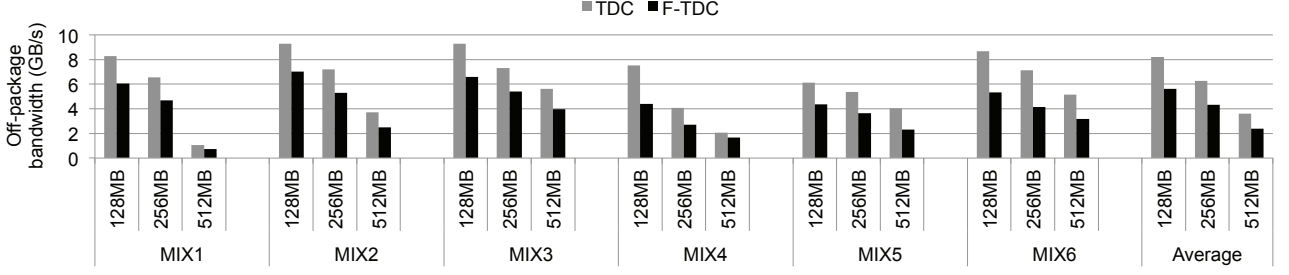


Figure 10: Off-package DRAM bandwidth consumption of TDC and F-TDC.

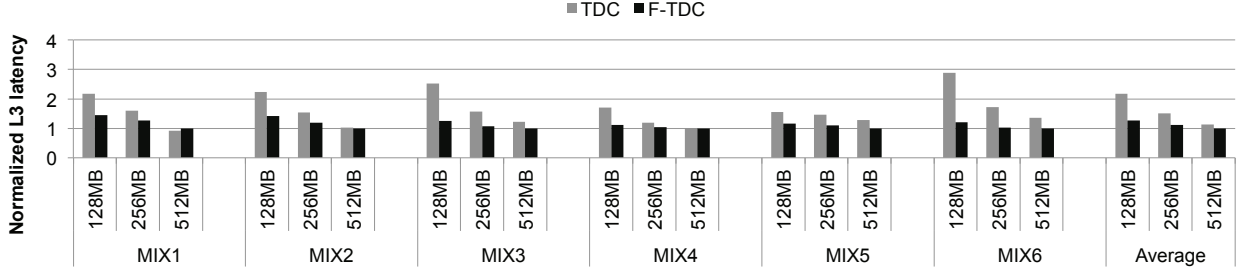


Figure 11: Average L3 latency of TDC and F-TDC, normalized to that of F-TDC with 512MB in-package DRAM.

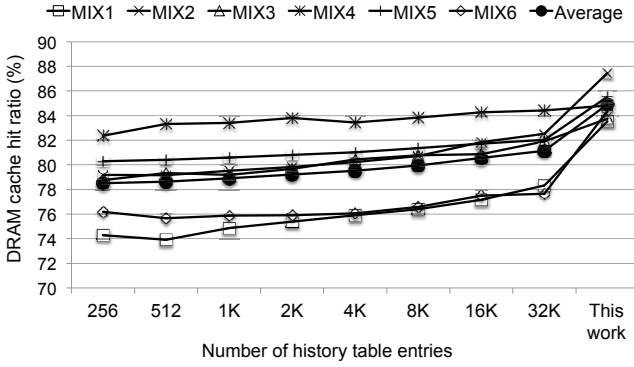


Figure 12: Cache hit rate of the original Footprint Cache [10] and F-TDC

Comparison against Footprint Cache [10]. We compare F-TDC against the original Footprint Cache for storage overhead and cache hit rate. Table 6 summarizes the storage overhead of F-TDC and Footprint Cache [10] for a 256MB DRAM cache. Our scheme requires only 4.5KB of additional on-die SRAM space, which is used to augment cTLB with reference and valid bit vectors. In contrast, Footprint Cache requires 1.58MB storage for cache tags and 144KB for a 16K-entry Footprint History Table (FHT). The hardware cost of F-TDC is much lower (i.e., less than 0.3% of that of Footprint Cache) as F-TDC eliminates cache tags with tagless caching and exploits PTEs to maintain footprints. F-TDC also requires 0.64MB of off-package DRAM storage for GIPT, which is much cheaper than both on-die SRAM and in-package DRAM. In summary, F-TDC is not only more scalable to the size of DRAM cache but also more energy-efficient than Footprint Cache with a minimal increase in chip size.

Figure 12 shows the benefit of tracking all pages' footprints in PTEs (F-TDC) compared to the FHT of limited size (Footprint Cache). To emulate FHT, we limit the max-

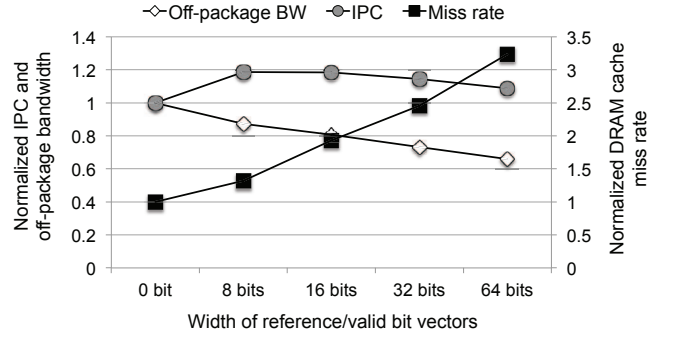


Figure 13: Off-package DRAM bandwidth usage, IPC, and cache miss rate of F-TDC with varying bit vector width. All numbers are normalized to TDC without footprint caching (0-bit vectors).

imum number of tracked footprints to a finite number, ranging from 256 to 32K, on F-TDC. Without such a limitation F-TDC achieves significantly higher cache hit rate by maintaining more accurate per-page footprints. By lifting this limitation (labeled “This work” in Figure 12) F-TDC achieves about 4% higher cache hit rate than the one tracking the footprints of up to 32K pages.

Sensitivity Analysis with Footprint Granularity. An important design choice of F-TDC is the footprint bit vector width (i.e., the number of bits in each reference/valid bit vector). Using too wide bit vectors reduces the cache hit rate of F-TDC for not exploiting spatial locality. In contrary, too narrow bit vectors make footprint caching less effective as F-TDC behaves like page-based DRAM caches which suffer from the over-fetching problem. Thus, we perform a sensitivity analysis to quantify the tradeoffs in selecting an optimal bit vector width.

Figure 13 shows how the off-package DRAM bandwidth usage, IPC, and cache miss rate change with varying bit vector width. As we increase the number of refer-

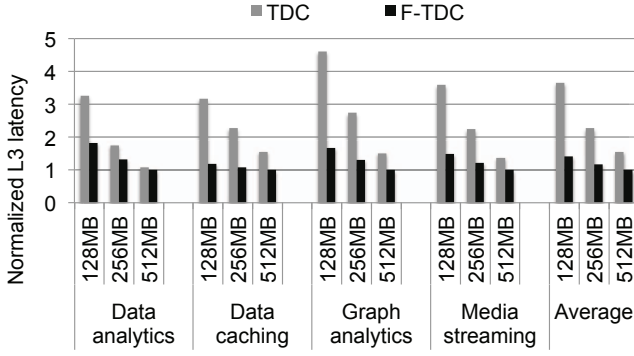


Figure 14: Average L3 latency of TDC and F-TDC with multi-threaded workloads, normalized to that of F-TDC with 512MB in-package DRAM.

ence and valid bits, we observe an increase in the cache miss rate and a decrease in the off-package DRAM bandwidth usage. This means that we can control the bit vector width to find a sweet spot between block-based caches (with wide bit vectors) and page-based caches (with narrow bit vectors) to take the best of both. Our evaluation demonstrates that the highest IPC improvement of 18.7% is achieved with 8-bit vectors for both reference and valid bits. Therefore, we choose the width of 8 bits as default.

5.2 Multi-threaded Workloads

To evaluate how effective F-TDC is with real-world data-intensive workloads, we now compare TDC and F-TDC using four multi-threaded workloads taken from CloudSuite [16]. Unlike multi-programmed workloads, we use a trace of memory reference instructions for these workloads instead of all instructions. It is because it takes multiple tens of billions of instructions to capture a working set whose size is larger than the DRAM cache size as shown in Table 5. It is infeasible to perform detailed microarchitectural simulation with so many instructions for every measurement. Thus, we use a memory access trace with 2 billion load/store instructions for each application.

Figure 14 compares the average L3 latency of TDC and F-TDC. F-TDC significantly reduces TDC’s average L3 latency by 62%, 49%, and 36% on average, for 128MB, 256MB, and 512MB caches, respectively. Furthermore, the L3 access latency increases more rapidly for TDC than F-TDC as the cache size decreases, because the over-fetching problem is more pronounced for the former. This result hints that F-TDC can bring significant performance benefits to emerging data-intensive big data workloads, which feature huge working sets.

6. DISCUSSION

Shared page support. When a page is shared by multiple page tables (i.e., in multiple process scenarios), a page can be cached at different locations, thereby causing a page aliasing problem [15]. The original TDC suggests several solutions to the problem: (1) updating all PTEs to keep them mapped to the same cache address; (2) disabling caching of a shared page; (3) tracking PA-to-CA mappings in an auxiliary structure to prevent a page from being located in multiple locations. In F-TDC, the three solutions can also be applied. However, the valid bit vector coher-

ence (explained in Section 3.2) should be extended to make valid bit vectors in different TLB entries coherent. The recent work on maintaining TLB coherence [20] identifies the same TLB entries on different cores by using PID and virtual address as the key. However, to support a shared page in F-TDC, the cache address should also be used to correctly identify multiple TLB entries for a shared page.

Superpage support in F-TDC Unlike tagged DRAM caches, which decouple OS page size and DRAM cache block size, TDC can exacerbate the over-fetching problem when a superpage is used [15]. Footprint caching can, however, alleviate this problem. Using footprints, only a necessary subset of blocks within a superpage can be cached. Nevertheless, if there is a small set of hot subpages within a superpage, a large amount of cache space can be wasted. One workaround is to split a superpage into smaller pages. The hierarchical page table structure facilitates this breakdown [15].

GIPT cache for reducing cache block miss penalty. In F-TDC, upon a block miss, the TLB miss handler should look up the GIPT to obtain the physical address of the requested block. This requires a long-latency DRAM access to increase block miss penalty. One workaround to alleviate this overhead is to use a GIPT cache, which is an on-die SRAM cache specialized to cache recently referenced GIPT entries. Especially for workloads with a relatively small set of hot pages the GIPT cache can be an effective solution to mitigate block miss penalty.

7. RELATED WORK

The overhead of cache tags in terms of latency, storage, and energy consumption, is a primary scalability bottleneck for future multi-gigabyte DRAM caches. To address this problem, caching at a page granularity, or *page-based caching*, has been actively investigated. CHOP [11] improves the efficiency of a page-based cache by caching hot pages only. Lee et al. [15] propose Tagless DRAM Caches (TDCs), which effectively eliminate tagging structures from both on-die SRAM and in-package DRAM. The key idea is to align the granularity of caching with OS page size and consolidate the two-step address translation by TLB and cache tag array into a single-step process by cache-map TLB (cTLB). Thus, TDCs achieves the lowest hit latency and best scalability known to date.

Unfortunately, page-based DRAM caches suffer an over-fetching problem, which wastes off-package DRAM bandwidth by fetching unused data. To reduce such waste, Footprint Cache [10] and Unison Cache [9] track which blocks in a page have been accessed while the page resides in the cache, referred to as the page’s *footprint*, and associate the footprint with the corresponding program counter. At a cache miss, the footprint is looked up to selectively fetch only those blocks that are likely to be used in the future. Bi-Modal Cache [27] uses two different caching granularities (e.g., 512B and 64B) at the same time. It prevents fetching unused data from off-package DRAM by caching at the finer granularity for data with low spatial locality. These mechanisms effectively reduce off-package bandwidth waste; however, they maintain cache tags, which can incur significant cost. In contrast, F-TDC exploits existing data structures in TDCs to reduce bandwidth waste effectively with a minimal hardware cost, while preserving the scalability benefit of TDCs.

Recently, extended TLB (eTLB) has been proposed for TagLess Cache (TLC) [28] and Direct-to-Data (D2D) [29] cache, which keep way index for cached blocks in a TLB. Similar to F-TDC a TLB lookup returns the exact location of the requested block in the cache. However, both target conventional on-die L1/L2 caches and do not address unique design challenges or leverage opportunities with die-stacked DRAM caches.

Alternatives to page-based DRAM caches include block-based DRAM caches [12, 14]. Although free of the over-fetching problem, block-based caches have much greater overhead with cache tags due to their fine-grained caching, and reducing this overhead is the primary design objective for them. Loh-Hill Cache [3] stores both tags and cache blocks in the same DRAM row, making a single row buffer open sufficient to serve both a tag access and the subsequent block access. Sim et al. [30] reduce the implementation cost of Loh-Hill Cache by introducing a multi-level cache hit/miss predictor. Like Loh-Hill Cache, Alloy Cache [13] also stores tags and their blocks in the same DRAM row, but collocates a tag and the corresponding block to further reduce the cost of a tag-and-block access to be a single-burst latency. ATCache [31] aims to accelerate tag accesses by employing an SRAM-based tag cache, which caches recently-accessed tags. Unfortunately, the inherent tagging overhead of block-based DRAM caches easily becomes a scalability bottleneck for future multi-gigabyte DRAM caches.

Chou et al. find that even block-based DRAM caches suffer from significant DRAM bandwidth pollution from secondary cache operations (other than tag and data accesses, or the primary operations) [32]. To dedicate more bandwidth to these primary operations, they propose Bandwidth Efficient ARchitecture (BEAR), which reduces the bandwidth consumption of cache fills, write-back probes, and miss probes. While BEAR focuses on achieving higher in-package DRAM bandwidth efficiency, F-TDC aims to minimize off-package DRAM bandwidth waste via efficient footprint caching.

There are proposals to use in-package DRAM not as caches but as part-of-memory to expand the main memory size. In such cases, it is important to efficiently migrate data between fast-but-small in-package DRAM and large-but-slow off-package DRAM as the placement of data can have significant impact on performance. Sim et al. [33] propose a hardware-only approach, which introduces another level of address translation from physical addresses into DRAM addresses. This approach does not require OS modifications as the hardware swaps OS pages between the fast and slow DRAM under the hood. In contrast, Meswani et al. [34] propose a hardware/software cooperative approach, which migrates hot OS pages. They extend TLB to measure a page's hotness and make OS not only determine the page's placement using this information but also perform page migrations if necessary. Chou et al. [35] propose to use in-package DRAM as both a DRAM cache and part-of-memory by migrating data between in-package and off-package DRAMs upon a cache miss.

8. CONCLUSION

This paper introduces Footprint-augmented Tagless DRAM Cache (F-TDC), which synergistically integrates footprint caching into the Tagless DRAM Cache

(TDC) [15]. Exploiting unique opportunities offered by TDC, F-TDC realizes efficient footprint caching with higher prediction accuracy at a lower hardware cost than the state-of-the-art footprint caches [9, 10]. Our evaluation with 3D TSV-based in-package DRAM demonstrates that F-TDC effectively reduces the off-package DRAM bandwidth, and thus significantly improves performance and energy efficiency, while retaining the scalability benefit of the original TDC.

9. ACKNOWLEDGMENTS

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (NRF-2014R1A1A1005894 and NRF-2014R1A1A1003746) and the Ministry of Education (NRF-2014R1A1A2054658).

10. REFERENCES

- [1] W. A. Wulf and S. A. McKee, "Hitting the Memory Wall: Implications of the Obvious," *ACM SIGARCH Computer Architecture News*, vol. 23, no. 1, 1995.
- [2] J. T. Pawlowski, "Hybrid Memory Cube (HMC)," in *Hot Chips 23*, 2011.
- [3] G. H. Loh and M. D. Hill, "Efficiently Enabling Conventional Block Sizes for Very Large Die-stacked DRAM Caches," in *Proc. 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2011.
- [4] S. Anthony, "Intel unveils 72-core x86 Knights Landing CPU for exascale supercomputing," <http://www.extremetech.com/extreme/171678-intel-unveils-72-core-x86-knights-landing>.
- [5] H. Mujtaba, "AMD Working With Hynix For Development of High-Bandwidth 3D Stacked Memory," <http://wccfttech.com/amd-working-hynix-development-highbandwidth-3d-stacked-memory/>.
- [6] S. S. Iyer, "The Evolution of Dense Embedded Memory in High Performance Logic Technologies," in *Proc. 2012 IEEE International Electron Devices Meeting (IEDM)*, 2012.
- [7] "Nvidia to stack up DRAM on future 'Volta' GPUs," http://www.theregister.co.uk/2013/03/19/nvidia_gpu_roadmap_computing_update/.
- [8] S. Lakka, "Xilinx SSI Technology: Concept to Silicon Development Overview," http://www.hotchips.org/wp-content/uploads/hc_archives/hc24/Hc24-Tut2-Die-Stack/Hc24.27.240-Overview-Lakka-Xilinx.pdf.
- [9] D. Jevdjic, G. H. Loh, C. Kaynak, and B. Falsafi, "Unison Cache: A Scalable and Effective Die-Stacked DRAM Cache," in *Proc. 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2014.
- [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 *Proc. 40th Annual International Symposium on Computer Architecture (ISCA)*, 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 *Proc. 16th International Symposium on High Performance Computer Architecture (HPCA)*, 2010.
- [12] G. H. Loh, "Extending the Effectiveness of 3D-Stacked DRAM Caches with an Adaptive Multi-Queue Policy," in *Proc. 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2009.
- [13] M. K. Qureshi and G. H. Loh, "Fundamental Latency Trade-offs in Architecting DRAM Caches: Outperforming Impractical SRAM-Tags with a Simple and Practical Design," in *Proc. 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2012.
- [14] L. Zhao, R. Iyer, R. Illikkal, and D. Newell, "Exploring DRAM Cache Architectures for CMP Server Platforms," in *Proc. 25th International Conference on Computer Design (ICCD)*, 2007.

- [15] Y. Lee, J. Kim, H. Jang, H. Yang, J. Kim, J. Jeong, and J. W. Lee, "A Fully Associative, Tagless DRAM Cache," in *Proc. 42nd Annual International Symposium on Computer Architecture (ISCA)*, 2015.
- [16] "CloudSuite benchmark 2.0." <http://parsa.epfl.ch/cloudsuite>.
- [17] S. Volos, J. Picorel, B. Falsafi, and B. Grot, "Bump: Bulk memory access prediction and streaming," in *Proc. 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2014.
- [18] "SAP HANA In-Memory Platform." <http://go.sap.com/solution/in-memory-platform.html>.
- [19] S. L. Salzberg, A. M. Phillippy, A. Zimin, D. Puiu, T. Magoc, S. Koren, T. J. Treangen, M. C. Schatz, A. L. Delcher, M. Roberts, G. Marçais, M. Pop, and J. A. Yorke, "GAGE: A critical evaluation of genome assemblies and assembly algorithms," *Genome Research*, vol. 22, no. 6, 2012.
- [20] V. Seshadri, G. Pekhimenko, O. Ruwase, O. Mutlu, P. B. Gibbons, M. A. Kozuch, T. C. Mowry, and T. Chilimbi, "Page Overlays: An Enhanced Virtual Memory Framework to Enable Fine-grained Memory Management," in *Proc. 42nd Annual International Symposium on Computer Architecture (ISCA)*, 2015.
- [21] J. H. Ahn, S. Li, S. O., and N. P. Jouppi, "McSimA+: A Manycore Simulator with Application-level+ Simulation and Detailed Microarchitecture Modeling," in *Proc. IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2013.
- [22] Y. H. Son, S. O., H. Yang, D. Jung, J. H. Ahn, J. Kim, J. Kim, and J. W. Lee, "Microbank: Architecting Through-Silicon Interposer-Based Main Memory Systems," in *Proc. International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2014.
- [23] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures," in *Proc. 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2009.
- [24] K. Chen, S. Li, N. Muralimanohar, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "CACTI-3DD: Architecture-level Modeling for 3D Die-stacked DRAM Main Memory," in *Proc. Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2012.
- [25] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically Characterizing Large Scale Program Behavior," in *Proc. 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2002.
- [26] F. Bellard, "Qemu open source processor emulator," URL: <http://www.qemu.org>, 2007.
- [27] N. Gulur, M. Mehendale, R. Manikantan, and R. Govindarajan, "Bi-Modal DRAM Cache: Improving Hit Rate, Hit Latency and Bandwidth," in *Proc. 47th International Symposium on Microarchitecture (MICRO)*, 2014.
- [28] A. Sembrant, E. Hagersten, and D. Black-Shaffer, "Tlc: A tag-less cache for reducing dynamic first level cache energy," in *Proc. 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2013.
- [29] A. Sembrant, E. Hagersten, and D. Black-Schaffer, "The direct-to-data (d2d) cache: Navigating the cache hierarchy with a single lookup," in *Proc. 41st Annual International Symposium on Computer Architecture*, 2014.
- [30] J. Sim, G. H. Loh, H. Kim, M. O'Connor, and M. Thottethodi, "A Mostly-Clean DRAM Cache for Effective Hit Speculation and Self-Balancing Dispatch," in *Proc. 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2012.
- [31] C.-C. Huang and V. Nagarajan, "ATCache: Reducing DRAM Cache Latency via a Small SRAM Tag Cache," in *Proc. 23rd International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2014.
- [32] C. Chou, A. Jaleel, and M. K. Qureshi, "BEAR: Techniques for Mitigating Bandwidth Bloat in Gigascale DRAM Caches," in *Proc. 42nd International Symposium on Computer Architecture (ISCA)*, 2015.
- [33] J. Sim, A. R. Alameldeen, Z. Chishti, C. Wilkerson, and H. Kim, "Transparent Hardware Management of Stacked DRAM as Part of Memory," in *Proc. 47th International Symposium on Microarchitecture (MICRO)*, 2014.
- [34] M. R. Meswani, S. Blagodurov, D. Roberts, J. Slice, M. Ignatowski, and G. H. Loh, "Heterogeneous Memory Architectures: A HW/SW Approach for Mixing Die-stacked and Off-package Memories," in *Proc. 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015.
- [35] C. Chou, A. Jaleel, and M. K. Qureshi, "CAMEO: A Two-Level Memory Organization with Capacity of Main Memory and Flexibility of Hardware-Managed Cache," in *Proc. 47th International Symposium on Microarchitecture (MICRO)*, 2014.