Revisiting Virtual L1 Caches

A Practical Design Using Dynamic Synonym Remapping

Hongil Yoon and Gurindar S. Sohi
Department of Computer Sciences
University of Wisconsin-Madison
Madison, WI, USA
{ongal,sohi}@cs.wisc.edu

ABSTRACT

Virtual caches have potentially lower access latency and energy consumption than physical caches due not to consulting the TLB prior to every cache access. However, they have not been popular in commercial designs. The crux of the problem is the possibility of *synonyms*.

This paper makes several empirical observations about the temporal characteristics of synonyms, especially in caches of sizes that are typical of L1 caches. By leveraging these observations, we propose a practical design of an L1 virtual cache that (1) dynamically decides a unique virtual page for all the synonymous virtual pages that map to the same physical page and (2) uses this unique page to place and look up data in the virtual caches. Accesses with this unique page proceed without any intervention. Accesses to other synonymous pages are dynamically detected, and remapping to the corresponding unique virtual page is performed to correctly look up the data in the cache.

Such remapping operations are rare, due to the temporal properties of synonyms. This new Virtual Cache with Dynamic Synonym Remapping (VC-DSR) can handle them in an energy and latency efficient manner, which achieves most of the benefits of virtual caches without software involvement. Experimental results based on real world applications show over 96% dynamic energy savings for TLB lookups without significant impact on performance compared to the system with ideal virtual caches (less than 0.4% slowdown).

1. INTRODUCTION

Virtual memory and caches are two of the most common elements of computers today. A CPU generates virtual addresses, which are translated to physical addresses so that the appropriate memory accesses can be made. Modern processors use physically addressed (tagged) caches, performing the virtual to physical translation on every cache access, via a TLB. This consumes up to 13% of core power [9], and adds latency when the TLB is consulted prior to cache access, i.e., Physically Indexed, Physically Tagged (PIPT) caches. Our analysis shows that about 9% of execution time is wasted for the address translation. The use of Virtually Indexed, Physically Tagged (VIPT) caches can hide the latency overhead. However, it not only still consumes power/energy for TLB lookups but also could entail more power overhead for cache lookups due to constraints on the cache organization (i.e., a larger associativity) [11, 31, 44]. Over the years, a plethora of techniques have been proposed and deployed to reduce the latency and energy impacts of the (prior) TLB access [10, 21, 22, 27, 29, 32, 33, 34, 38, 40]. However, the basic problem still remains if physical addresses are used for cache operation, especially L1 cache operations.

Virtually Indexed, Virtually Tagged (VIVT) caches have potentially lower access latency and energy consumption than physical caches because a TLB is consulted only for L1 cache misses (e.g., 2% of L1 cache accesses). Despite the desirability [16, 24], however, *virtual caches have not been considered to be practical*. The primary reason is the complications due to *synonyms*.

For a variety of reasons [11, 16, 46, 49], multiple virtual pages can be mapped to the same physical page. The same data can be placed and looked up later with a variety of different virtual addresses, and thus the straightforward use of unmodified virtual addresses can lead to potentially erroneous data accesses. Moreover, depending upon the microarchitecture, additional challenges can arise, e.g., the x86 hardware page table walk [3, 28], coherence [35], and violations of memory consistency and of sequential program semantics [8, 46] (Sec 3.5). Several virtual cache designs have been proposed to obviate the overheads of physical caches [11, 24, 35, 36, 46, 50, 52, 53]. However, this plethora of problems complicate the cache design and impair the potential benefits of virtual caches (Sec 4.4 and 5).

This paper describes and evaluates a practical L1 virtual cache design, called a *Virtual Cache with Dynamic Synonym Remapping* (VC-DSR). The design leverages the following *temporal* properties of synonyms that are quantified in this paper:

- There are several physical pages with potential synonyms over the duration of the program. However, in a smaller window of time, e.g., the duration of a data item's residence in an L1 cache, the number of physical pages with potential synonyms is small (e.g., 4).
- The number of synonymous virtual pages mapped to such physical pages is small (e.g., 2) as well.
- For smaller caches such as L1 caches, only a small fraction (e.g., 1%) of cache accesses are to such physical pages.

The above empirical characteristics of synonyms suggest that it is practical to design a L1 virtual (VIVT) cache with physical (PIPT) caches for lower-levels. Furthermore data is cached and looked up with a *unique (leading) virtual page*, V_i, while data from the corresponding physical page resides in the L1 virtual cache[1]. Accesses to the leading virtual page V_i proceed as normal, while accesses made to the same data with a different address, V_j, are remapped to use V_i instead; dynamic remappings from non-leading (virtual) address to leading address are needed, rather than performing virtual to physical translation via a TLB.

Because synonyms are short-lived in smaller caches, such remappings are rare for the L1 virtual cache and a data structure used to link V_j to V_i is quite small as well. Thus VC-DSR can handle the needed remappings in an energy and latency efficient manner, which can achieve most of the benefits of canonical (but impractical) virtual caches *without* any modifications of software. In addition, the use of the unique (leading) virtual address for all the operations of the virtual caches prevents potential synonym issues, which greatly simplifies the design of our proposal, as well as the overall memory hierarchy.

In this paper, we make the following contributions:

- We present novel empirical observations for the temporal behavior of synonyms, based on real world workloads, e.g., server and mobile (Sec 2).
- We propose VC-DSR: a practical solution to a problem that has long vexed computer architects: achieving almost all the benefits of virtual caches but without *any* software involvement.
- Regarding specific microarchitectural details that are critical to a viable commercial implementation of any virtual cache (Sec 3.5), we discuss synonym issues and solutions for VC-DSR.
- VC-DSR saves over 96% of dynamic energy consumption for TLB lookups and also achieves most of the latency benefit (over 99.6%) of ideal (but impractical) virtual caches (Sec 4).

2. TEMPORAL SYNONYM BEHAVIOR

A variety of common programming and system practices lead to synonyms [11, 16, 46, 49] and it is well established that over the execution of an entire program, especially one that has a lot of OS activity, synonyms are not a rare occurrence. This has been the primary impediment to the adoption of virtual caches.

Let us call the set of one physical page (P_X) and possibly multiple virtual pages associated with P_X the *Equivalent Page Set* (EPS_X), and let C(X) be the *cardinality* of EPS_X, i.e., the number of virtual pages in EPS_X over the entire duration of a program's execution. Since synonyms in a virtual cache are an actual problem only if a block is cached and accessed with different addresses *during its residence in the cache*, the *temporal* characteristics of potential synonyms are what is important. Let TC(X,T) be the *temporal cardinality* of EPS_X, i.e., the number of virtual pages that are synonymous with physical page P_X in time interval T. An *active synonym* for P_X (or the associated EPS_X) occurs when TC(X,T) ≥ 2 in the time interval T; the time interval T that is of interest is where one or more lines from the page P_X are resident in the cache.

As we shall establish below, the temporal characteristics of active synonyms do indeed display properties that are amenable to designing a *software-transparent, practical, L1 virtual cache*. To gather the empirical data we use several real world applications running on a full-system simulator as we describe in Section 4.1. Several of the data items below are gathered using periodic sampling of the contents of a cache. Since in some cases the samples with different cache organizations may correspond to different points in a program's execution, comparing the absolute numbers of two arbitrary data points may not be very meaningful; the trends characterizing the temporal behavior are what we want to highlight. When absolute numbers are appropriate, they are presented without sampling.

2.1 Number of Pages with Active Synonyms

The first set of data (1) of Table 1 presents the average number of EPSs (for distinct physical pages) with active synonyms. The time interval T is the time in which data blocks from the corresponding P_X reside in the cache of the particular size and thus it varies with cache size. Both instruction and data caches of varying sizes are considered. For smaller cache sizes there are few pages with active synonyms; the average number of pages with the potential for synonymous access increases as the cache size increases.

This phenomenon occurs due to two complementary reasons: smaller caches not only contain data from fewer pages, but all the blocks from a given page are likely to be evicted from a smaller cache earlier than they would be from a larger cache. A large 256KB cache contains data from tens or hundreds of pages with the potential for synonyms; the number for all of memory (not shown) is even larger. The latter confirms that there are many EPSs with C(X) ≥ 2, i.e., the potential for synonyms is real. However, for small cache sizes, such as those one might expect to use for an L1 cache (e.g., 32-64KB), the number of pages residing in the cache with TC(X,T) ≥ 2 is small.

Observation 1 (OB_1): *The number of pages with active synonyms is quite small for small cache sizes that are typical of an L1 cache.*

2.2 Temporal Cardinality of an EPS

The next set of data (2) of Table 1 presents the average number of distinct virtual pages in an EPS for which an active synonym occurs, for different sized caches[2] This number increases with cache size, due to longer residence time of a block (page); it can become quite large for the larger instruction cache sizes due to shared

[1]The leading virtual page at a given time can change in different phases of program execution.

[2]A blank entry indicates that the corresponding number is not meaningful since there are almost zero pages with active synonyms resident in the cache.

	(1) Avg. number of physical pages with active synonyms								(2) Avg. number of virtual pages in an EPS for active synonyms								(3) Frequency of changes in the LVA (%)			
	Inst. Cache				Data Cache				Inst. Cache				Data Cache				Inst. Cache		Data Cache	
Size (KB)	**32**	**64**	**128**	**256**	**32**	**64**	**128**	**256**	**32**	**64**	**128**	**256**	**32**	**64**	**128**	**256**	**32**	**64**	**32**	**64**
QEMU	0	0	0	1	0	2	3	24	-	-	-	-	-	2	2	2	79	78	56	94
Memcached	36	87	158	282	52	93	160	284	2	3	12	28	2	2	2	2	4	0	5	5
SPECjbb2005	0	0	2	22	0	1	2	2	-	-	2	2	-	-	2	2	69	67	63	56
TPC-H	6	20	57	133	2	5	8	12	2	3	8	32	2	2	2	2	63	57	73	74
bzip2	1	12	93	282	0	0	0	0	-	13	23	29	-	-	-	-	70	71	98	98
h264ref	0	0	5	115	0	0	0	1	-	-	25	24	-	-	-	-	79	75	82	82
Stream	0	1	23	106	1	1	2	4	-	-	4	4	-	-	2	2	76	70	70	70
Raytrace	0	0	0	0	7	15	29	57	-	-	-	-	2	2	2	2	100	100	26	26

Table 1: Analysis of EPSs with active synonyms in various sizes of caches

libraries and kernel interfaces. However:

Observation 2 (OB_2): *The average number of virtual pages mapped to the same physical page, for pages with an active synonym, in a small cache is quite small.*

2.3 Changes in Leading Virtual Address (LVA)

Suppose at some point in time address V_i was the first virtual page in an EPS_X and was being used as the leading virtual address (Page number) for the corresponding physical page P_X. Now suppose that after being cached with V_i, other references were made, and all the blocks from P_X were evicted from the cache. Next time P_X was referenced, virtual address V_j was the leading virtual for that EPS. We say that a change in the leading virtual page occurs if $V_j \neq V_i$.

Table 1, third set of data (3) presents the percentage of changes in the leading virtual address (LVA) by varying the size of the caches. Only pages for which multiple virtual addresses could be used as a LVA i.e., the C(X) ≥ 2, are considered. An entry indicates the percentage of time a change in the LVA occurs; 100 indicates that the LVC always changes and 0 means that it is always the same. The data indicates that it is quite common for different virtual addresses from the same EPS to be the leading virtual addresses at different times during the execution of the program. For *memcached*, the percentage of LVA changes is small. This is because, due to heavy synonym access activity, lines from synonym pages are frequently referenced. Thus they are not replaced, and continue to reside in the cache with the leading virtual address, even though the additional references may be with other virtual addresses. This results in fewer LVA changes.

Observation 3 (OB_3): *When multiple virtual addresses map to the same physical address, always using the same virtual address to cache the page can be unnecessarily constraining.*

2.4 Accesses to Pages with Active Synonyms

Table 2 presents the frequency of cache access to physical pages with active synonyms for different sized instruction and data caches. There are two sets of data for both instruction and data caches; each entry is the number of references per 1000 accesses.

The first set of data (1) is the number of references to cache blocks contained in pages with active synonyms, using any virtual address in an EPS. It is for these references that a virtual cache design may have to take additional steps to ensure correct operation. The second set (2) is the number of references made with a non-leading virtual address V_j, that is different from the current leading virtual address V_i. These are the references for which a virtual cache design, that we describe in section 3, will have to intervene to remap V_j to V_i and submit the access with V_i instead of V_j.

		(1)			(2)		
	Size (KB)	**32**	**64**	**128**	**32**	**64**	**128**
Inst. Cache Access	**TPC-H**	45	105	161	15	22	55
	SPECjbb2005	0	0.1	0.2	0	0.1	0.2
	Memcached	502	695	770	282	352	492
	QEMU	0	0.1	0.2	0	0.1	0.2
	bzip2	0.3	2.3	2.9	0.3	2.3	2.9
	h264ref	0	0.1	0.3	0	0.1	0.3
	Stream	0.3	3	28	0.3	0	26
	Raytrace	0	0	0	0.1	0	0
Data Cache Access	**TPC-H**	48	80	87	9	12	27
	SPECjbb2005	2.7	22.8	25.5	2.6	22.7	25.1
	Memcached	478	518	530	274	294	295
	QEMU	1.6	2	3.1	1.6	2	3.1
	bzip2	1	1.9	2.3	1	1.9	2.3
	h264ref	0.1	0.7	2	0.1	0.7	2
	Stream	20	24	25	20	24	25
	Raytrace	32	93	119	32	93	118

Table 2: Number of references (per 1000) to pages with active synonyms in caches

The first set of data suggests that overall percentage of accesses to cache blocks with active synonyms is quite small in most cases, especially for smaller cache sizes. However, in some cases it is quite large. For example, *memcached* has many references, both instruction and data, to lines within pages with active synonyms. This is due to heavy use of shared libraries (e.g., libc and libpthread) and kernel interfaces/structures (e.g., network communication (TCP/IP), memory operations, locks, I/O, etc.)[3]. The number of references to blocks with active synonyms increases as the cache size increases but is still somewhat small in most (although not all) cases. At first glance, we would expect the percentage of accesses to synonym pages to be the same regardless of the cache size. This is true. However, the percentage of accesses to *active synonyms* is smaller as there are fewer active synonyms in smaller caches.

Looking at the second set of data and comparing an entry with the equivalent entry in the first set, notice that the entries in the second set are smaller (in some cases by a large amount) than the entries in the first set. This suggests that, of the small number of references to

[3]For smaller caches, e.g., 32KB, most of the synonym accesses result from accessing kernel virtual address space. About 61% of memory references occur in the kernel space, and about 76% of such references are for active synonyms.

pages with an active synonym (1st set), even smaller number (2nd set) are made with a virtual address that is different from the leading virtual address.

Observation 4 (OB_4): *Typically only a small fraction of cache accesses are to cache lines from pages with active synonyms.*

Observation 5 (OB_5): *In most cases, a very small percentage of cache accesses (over total cache accesses) are to cache lines from pages with active synonyms that are cached with a different virtual address.*

3. PROPOSED VIRTUAL CACHE DESIGN

3.1 Rationale of Our Proposal

The above empirical observations suggest that it might be practical to design an L1 virtual cache in which data is cached with one virtual address, V_i, and synonymous accesses made to the same data with a different address, V_j, remapped to use V_i instead. In particular:

- OB_1 and OB_2 suggest that a structure tracking remapping links $<V_j, V_i>$ can be quite small.
- OB_3 suggests that this data structure would need to track the mappings dynamically since it is desirable that the leading virtual address V_i changes during a program's execution.
- OB_4 and OB_5 suggest that the (re)mapping data structure may be infrequently accessed.

Our proposal: Using these observations as a basis, we propose a practical virtual L1 cache design, called a *Virtual Cache with Dynamic Synonym Remapping* (VC-DSR). At a very high level, its operation is as follows. Data is cached and accessed with a *(dynamic) unique leading virtual address* for a given physical page. When a virtual address is generated, hardware structures are consulted to see if the address is a leading virtual address. Otherwise, the corresponding leading address is identified and used to access the virtual cache.

By employing a unique virtual address, the operation of virtual caches, as well as the overall memory hierarchy, is greatly simplified. Due to the temporal behavior of synonyms in smaller caches, VC-DSR can handle such dynamic remappings in an energy and latency efficient manner. Accordingly, most of the benefits of virtual caches can be achieved by bypassing TLB lookups.

3.2 Design Overview of VC-DSR

Figure 1 gives an overview of the overall microarchitecture: the processor generates virtual addresses (Phase 1), the L1 cache is virtually indexed and tagged (Phase 3), and the lower-level caches are traditional physical caches (Phase 5). Phase 4 is a boundary between a virtual and physical address, and thus the virtual to physical (or physical to virtual) address translation is performed via a traditional TLB for L1 virtual cache misses (or via an ASDT for coherence requests from lower-level caches). In addition, there are other microarchitectural structures for the active synonym remapping (Phase 2) and detection (Phase 4), which ensures the correctness of overall cache operations.

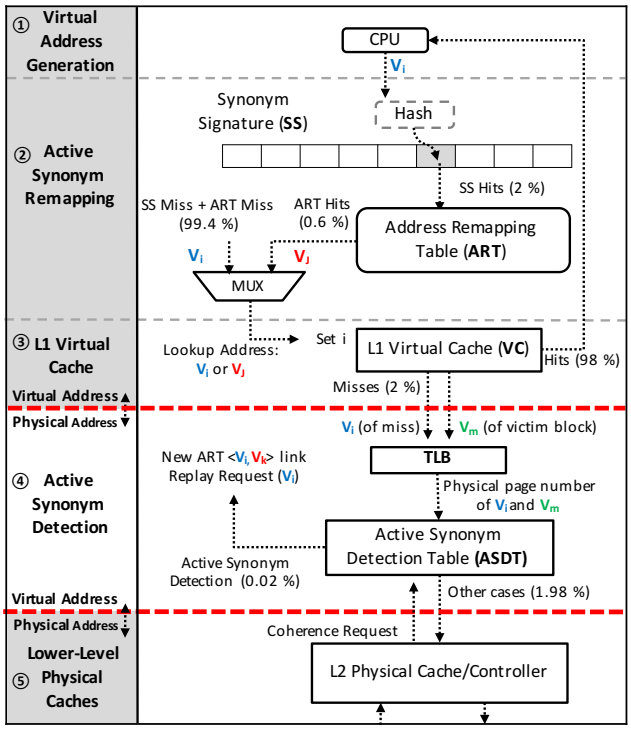

Figure 1: Schematic Overview of VC-DSR

Basic Operations: VC-DSR uses a unique leading virtual address for a given physical page not only to place data of the corresponding page but also to later access the data in an L1 virtual cache. Thus, conceptually, an *Address Remapping Table* (ART) is consulted on every cache access with a virtual address (V_i) generated by a CPU[4]. It identifies if V_i is a non-leading virtual address for a given physical page. If so, the corresponding leading virtual address, V_j, is provided, and V_i is remapped to V_j for that access. Otherwise, V_i is used to look up the virtual cache.

Since temporally there are expected to be few accesses to pages with active synonyms (OB_4), the chances for finding a matching entry in the ART are low, e.g., 0.6%, and thus most accesses to the ART are wasted. To reduce the number of such ART accesses, a *Synonym Signature* (SS) could be used. The SS is a hashed bit vector based on the virtual address (V_i) generated by the CPU and conservatively tracks the possibility of a match in the ART. If the corresponding bit in the SS indicates no possibility, e.g., 98% of the time, the ART is not consulted and V_i is used to look up the cache. Otherwise, e.g., 2% of the time, the ART is consulted to determine the correct lookup address (V_i or V_j).

On a cache miss, the virtual address (V_i) is used to access the TLB, and the corresponding physical page, P_i, is obtained. Next, an *Active Synonym Detection Table* (ASDT) is searched for an entry corresponding to P_i. A valid entry indicates that some data from that physical page resides in the virtual cache. If there is no match, one is created for the $<P_i,V_i>$ pair, and V_i

[4]All virtual addresses include an ASID to address the issue of homonyms.

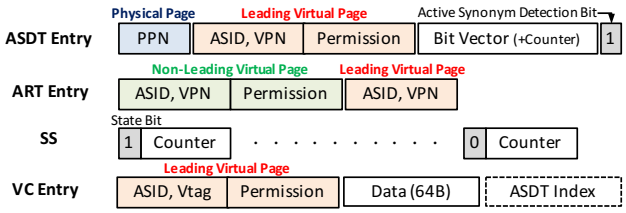

Figure 2: Overview of Entries for Structures supporting VC-DSR

will be used as the leading virtual page. If there is a match, the corresponding leading virtual page, V_k, is obtained. If $V_i \neq V_k$, an active synonym is detected, an entry is created in the ART for the $<V_i,V_k>$ tuple and the corresponding bit in the SS is set. V_k is then used as the lookup address for references made with V_i, while the $<V_i,V_k>$ tuple is valid in the ART.

3.3 Details of Structures supporting VC-DSR

More details of components supporting VC-DSR in each phase are described next. Figure 2 presents the basic organization of an entry for each component.

3.3.1 Active Synonym Detection

The Active Synonym Detection Table (ASDT) is a set-associative array indexed with a physical page number (PPN). A valid entry in the ASDT basically tracks (1) whether lines from a physical page are being cached in the virtual cache and (2) the leading virtual address being used to cache them as well as its permission bits. For the former, a counter suffices for correctness. However, for more efficient evictions (Sec 3.4.5), employing a bit-vector to identify the individual lines from the page in the cache may be a better option.

The ASDT is consulted on every L1 cache miss to see if an active synonym access occurs by comparing the current leading virtual address and the referenced virtual address; once it occurs, a single detection bit is set. In addition, the physical to the (leading) virtual address translation is performed via the ASDT for coherence requests from lower-lever caches.

3.3.2 Active Synonym Remapping

The design employs two other structures to efficiently perform a remapping between a non-leading and the corresponding leading virtual address, when needed.

The Address Remapping Table (ART) is a small set associative cache, indexed with a virtual address (VA) generated by a CPU, whose entries are created when the ASDT detects an active synonym. A valid entry in the ART tracks a <non-leading, leading virtual page> tuple with the permission of the non-leading page, indicating that an active synonym is present. On a match, the ART returns the leading virtual page and the permission for the requested VA (i.e., non-leading page). The permission check is performed at this point (details in Sec 3.5). The leading virtual address is used to look up the virtual cache for the request. The absence of a matching entry in the ART indicates the lack of an active synonym, i.e., a VA generated by the CPU is the correct lookup address. When an active synonym no longer persists, e.g., when all the cache lines from a physical page are evicted from the virtual cache, the corresponding entry in the ART has to be invalidated (details in Sec 3.4.5).

Accesses to the ART are unnecessary when the referenced virtual address (VA) is a leading virtual address, i.e., no match in the ART. A Synonym Signature (SS) is used to elide most unneeded ART lookups. This is built on a Bloom filter [14]. The SS is accessed with the VA of a request, and a *single bit* is read to determine if the ART should be accessed. When a new ART entry is populated, a bit in the SS is set based on the hash of the non-leading virtual address for the entry. Since multiple non-leading virtual addresses can be mapped to the same bit, each bit in the SS conservatively tracks the possibility of a match in an ART: false positives.

To prevent the SS from being overly conservative, a counter per bit tracks how many entries in the ART are associated with it; the counter increases/decreases when the corresponding ART entry with the VA is populated/invalidated. The size of the counter is proportional to the number of entries in an ART. It will be quite small (e.g., 4 bits) since an ART has few (e.g., 16) entries (see OB_1 and OB_2). As we shall see in Section 4.2, a small SS (e.g., 256 bits) with 5-bit counters is sufficient to filter out almost all unneeded ART lookups.

3.3.3 L1 Virtual Cache

The L1 virtual cache (VC) is effectively the same as a traditional VC, as depicted in Figure 2. ASIDs are employed to address *homonym* issues without flushing the cache on context switches. The leading virtual page's permission bits are stored per entry.

3.4 Overall Operation

We describe the overall operation of VC-DSR. Figure 3 illustrates accesses with virtual address synonyms over time and the needed operations in each phase.

3.4.1 Determining a leading virtual page (LVP):

An access is made with the virtual page number (V_i). Let us assume this is the first access to data from a physical page (P_i) (Case ①). A leading virtual page (LVP) has not been determined for P_i (i.e., no matching ASDT entry), and thus we have an SS miss or an ART miss although an SS hit occurs due to the false positive information. Either way, V_i is used to look up the virtual cache (VC), leading to a VC miss.

The TLB is accessed with V_i to obtain the physical page number (PPN), and then the ASDT is checked for a matching entry of the PPN. As this is the first access to P_i, no matching entry is found. An ASDT entry is chosen as a victim (e.g., an invalid entry, or a valid entry with the lowest counter value). For a valid entry victim, operations needed to invalidate the entry are carried out (Sec 3.4.5). The entry is allocated for the requested physical page P_i. The leading virtual page field is populated with the V_i and with its permission.

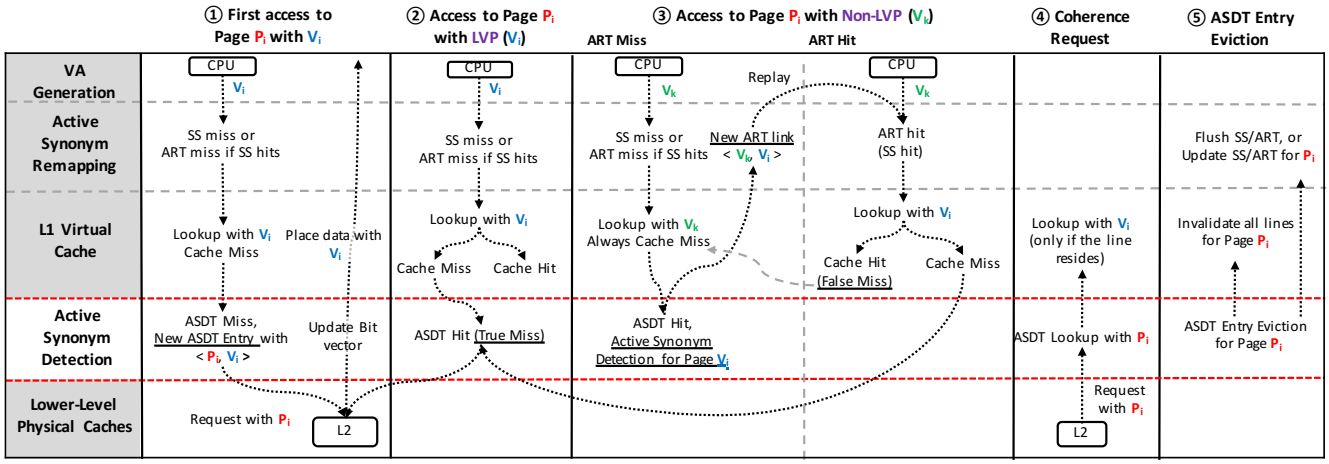

Figure 3: Overall Operations of VC-DSR

The line is fetched from the lower-level(s)[5]. The corresponding bit in the bit vector of the ASDT entry is set and the counter is incremented. Then the fetched line is placed in the VC with the leading virtual address (V_i) including the permission. A victim line in the VC may need to be evicted to make place for the new line.

3.4.2 *Accesses with a leading virtual page:*

Further accesses to the physical page (P_i) with the leading virtual page (V_i) (Case ②) proceed without any intervention. A VC hit proceeds as normal. A VC miss indicates that the line does not reside in the VC (i.e., true miss) since V_i is the leading virtual page. Thus, the line is fetched from the lower-level caches, and then the request is satisfied as discussed above.

3.4.3 *Accesses with non-leading virtual page:*

We now consider the first synonymous access (left side of case ③). Data of the common physical page (P_i) is accessed with a virtual address synonym (V_k). Any active synonyms have not been detected yet in this page. Thus, V_k is used to look up the virtual cache. Since V_k is not a leading virtual page (LVP), we always have VC misses for any accesses with V_k.

A matching entry for the common physical page (P_i) is found in the ASDT, and a new active synonym is detected by comparing the referenced virtual page number (V_k) with the current LVP (V_i). Thus, the ART is informed to create a new entry for the $<V_k, V_i>$ tuple, which also results in the SS being modified. Any further accesses to virtual page V_k will be remapped with V_i via an ART (right side of case ③).

On a VC miss with V_k, the data may actually reside in the VC with the leading virtual address (i.e., false miss), and thus the request is resubmitted (replayed) to the VC with V_i. In this case, the VC hit/miss is handled like accesses with a leading virtual page.

3.4.4 *Coherence Request (Case ④):*

A coherence event typically uses physical addresses. Thus, it is handled as normal for lower-level (PIPT) caches. Only for coherence activities between L2 and virtual L1 caches, the ASDT has to be consulted to translate a physical address to the corresponding leading virtual address. This may add additional latency and energy overhead. However, such coherence events between an L2 and a (relatively small) L1 are rare [11]. The ASDT can also be used to shield the VC from unnecessary VC lookups if an ASDT entry includes information identifying individual lines from the page present in the cache (the bit vector). Thus, the reverse translation will be considered as overhead only when the corresponding line resides in the small L1 VC.

3.4.5 *Entry Eviction of VC-DSR Components*

Cache line eviction: On evicting a VC entry, the corresponding ASDT needs to be updated. A matching entry is always found in the ASDT since at least one line (i.e., the victim line) from the page was still resident in the VC. The corresponding bit in the bit vector is unset, and the counter decremented. If the counter becomes zero, the ASDT entry can be invalidated.

To find the matching ASDT entry, the PPN is needed, via a TLB lookup. Thus, on a VC miss, with the straightforward approach the ASDT and TLB are consulted twice, once each for the evicted line and the newly fetched line. An alternative is to keep the index of the relevant ASDT entry with each line in the VC (e.g., with 8 bits) (see Fig. 2) so that the corresponding ASDT entry for the victim line can be directly accessed without a TLB (and ASDT) lookup.

ASDT entry invalidation/eviction: A valid ASDT entry is normally evicted when there is no available entry for a new physical page. Page information change or cache flushing also triggers invalidations of corresponding ASDT entries. To evict an ASDT entry, it first has to be invalidated (Case ⑤). All lines from the corresponding (victim) page that are still in the VC have to be evicted. Furthermore, if active synonyms have been observed (a detection bit was set), the mappings in the ART are now stale. Thus the relevant ART/SS entries have to be invalidated/updated (or simply flush all).

[5]MSHR entries keep the index of the relevant ASDT entry. Hence no additional ASDT lookups are needed when the response arrives.

ART entry invalidation/eviction: If space is needed in the ART, a victim entry can be chosen and evicted without correctness consequences as accesses made with a non-leading virtual address will simply miss (and result in recreation of the ART entry, Case ③). However, the corresponding counter/bit in the SS should be updated so that it can retain its effectiveness. Page information change for non-leading virtual pages also triggers the invalidation of the related ART entries.

3.5 Other Design Issues

We now discuss several other issues that need to be addressed for a practical virtual cache design. Most of these issues also arise in other VC designs.

Page information changes: When page information changes (e.g., mapping and permissions), all the entries corresponding to that page in all the components supporting VC-DSR need to be invalidated (or updated) to ensure consistency (see 3.4.5). The event is triggered by TLB invalidations, which potentially requires multiple ASDT lookups to search for an ASDT entry that has the target virtual page as a leading virtual page. In practice, the events rarely occur (less than once per 0.5M instruction) and can be filtered out with a simple filter. Thus, this does not have a significant impact on the effectiveness of our proposal.

Permission check: Depending on whether an access is to a leading or non-leading virtual page, the location of checking (keeping) the page permission is different. For the former case, each line in the VC tracks the permission, while an entry in the ART maintains permission of a non-leading virtual page. The permission check is done when the matching entry is found. Once a page permission mismatch occurs, the request is handled like a cache miss after carrying out all the actions for the page information change discussed above.

Hardware page-table walk based on physical addresses: Some architectures (e.g., x86) support a hardware page-table walker [3, 28]. The PTE may be accessed using a virtual address by the OS, and thus may end up in the cache, whereas it is only accessed using physical addresses by the page table walker. The key to handling this case is the observation is that if a line (from a physical page) resides in the L1 cache, there will be a corresponding entry in the ASDT. An access made with a physical address will consult the ASDT to obtain the leading virtual address (LVA) with which the block is being cached, and access the cache (if the block is there) with that LVA.

Supporting a virtually addressed Load-Store Unit: Using virtual address for a conventional write buffer (or store queue) (e.g., x86 TSO) can result in a violation of sequential semantics: a later load may not identify a matching earlier store due to synonyms, and vice versa. Thus stale data could be returned from caches. In the similar vein, using virtual addresses for a load queue could potentially violate memory consistency models; when a coherence event, e.g., an invalidation or eviction, occurs in L1 caches, a load has been carried out (speculatively) for the corresponding data

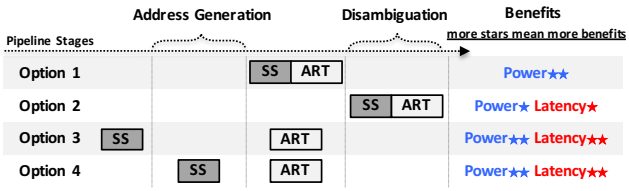

Figure 4: SS and ART Placement

may need to be identified and replayed. In commercial processors, the issues are handled by finding potentially offending loads by matching the physical addresses and replaying them. For virtual caches, however, synonyms can complicate the identification of them.

Most of the prior literature has not discussed these issues, and the proposed solution for the former issue is not efficient [46]. However, VC-DSR can easily handle all of them as follows: the key idea is to employ a leading virtual address as an alternative to a physical address. Accesses with a non-leading virtual address always cause VC misses and are eventually replayed with the leading address via an ART (Case ③). Thus, once a load/store is (speculatively) executed, its leading address is identified and kept in the load/store queue. The unique leading virtual address of each load/store is used to find potential violations.

Now we briefly discuss how TLB misses are handled for stores in the write buffer. One option is hold the younger stores in a separate buffer until the TLB miss is resolved. An alternative would be to restart the program from the offending store, e.g., by delaying the release of the state in the ROB. Regarding this problem, [45, 46] also proposed to tolerate late memory traps.

Large pages: For large pages, individually identifying lines from the page resident in the L1 virtual cache (VC) is not practical since the bit vector would be extremely large. However, keep in mind that a bit vector is simply an optimization to invalidate lines in the VC before evicting the corresponding ASDT entry. Without precise information about individual lines, the lines from a page could be invalidated by walking through the lines in the VC to search for lines from the page, and using the associated counter to track when the necessary operation is completed (counter is zero). We can minimize the likelihood of this potentially expensive operation by not evicting an ASDT entry for a large page unless absolutely necessary (e.g., all candidate entries are for large pages–an extremely unlikely event).

3.6 Design choices for SS and ART placement

Since memory operations typically require additional steps between an address generation and submission to the L1 cache (e.g., load/store queue, memory disambiguation), there are several choices for where in the pipeline the SS and the ART are placed. Figure 4 shows four design choices and the pros and cons of them.

The SS can be consulted immediately after address calculation, and the ART looked up only if needed (Op1). This achieves most of power/energy benefit of using virtual caches but perhaps not the potential latency bene-

fits. To obtain latency benefit as well, the SS and ART can be accessed in parallel with memory disambiguation (Op2). However, the disambiguation needs to be done again with the correct leading virtual address (on ART hits) although they are rare. In practice, the SS can be accessed *before* the address generation, based on base (or segment) registers (Op3), and the ART consulted before the disambiguation. If address generation is a multi-step process, the SS could be accessed after intermediate steps of the process (Op4). Op3 looks appealing since it can be applied to both I and D caches, and most of energy and latency benefits can be achieved without the increase in design complexity.

3.7 Optimizations

Last LVA Register: Though few, ART accesses can be reduced even further by exploiting the fact that often successive references are to the same page (especially true for instructions). Similar to the VIBA/PIBA register used for *pre-translation* for instruction TLB accesses in the VAX-11/780 [20], we can employ a *Last LVA* (LLVA) register, which maintains the leading virtual address (LVA) for the last page that was accessed, close to (or within) address generation. Thus consecutive access to the same page need not consult the ART.

Kernel Address Space Access: All the distinct processes globally share kernel code and data structures, and ASIDs are used to solve the homonym problem. Done naively, this could lead to multiple entries in the ART. To avoid the associated overhead, we can use on the fly remapping of an ASID only for accesses to the kernel space; a predefined (or unique) ASID value is used for such accesses. For the current design (x86-64 Linux) supporting 48-bit virtual address, the OS takes the upper half of the address space and thus the remapping process can easily be carried out by reading the 48th bit of a virtual address.

3.8 Storage Requirements

The storage requirements for the proposal are quite modest. An over-provisioned ASDT (e.g., 128 entries) requires 2.4KB [6]. A 32-entry ART (large given the number of pages with active synonyms (OB_1 and OB_2)), along with 256-bit SS (with 5-bit counters), needs approximately 550B. A 256-entry ASDT would also imply 8 extra bits with each VC line if eliminating the TLB access for cache line eviction was desired.

4. EVALUATION

4.1 Evaluation Methodology

We attached modules to simulate our proposal in the Gem5 simulator [13]. To carry out a meaningful study of virtual caches, especially for synonyms, two essential features have to be considered for the experimental methodology. First, the infrastructure needs to support a full-system environment so that all kinds of memory access, e.g., access to dynamically linked libraries and access to user/kernel address space, can be considered. Second, the experiments need to be run long enough to reflect all the operations of a workload and interaction among multiple processes running on the system. Detailed CPU models in architecture simulators [13, 43] supporting a full-system environment are too slow, and instrumentation tools [39], while relatively fast, provide inadequate full-system support.

[6]Half of the storage is taken by a 64-bit bit vector per entry for 4KB page with 64B lines. To further reduce the overhead, the information can be maintained in a coarse grained manner, although it could lose the accuracy.

	1	2
CPU type	AtomicSimple	Out-of-Order 8-issue
Num. CPU	Single-Core	
L1 Virtual $	Classic $ Model	Ruby 3-level hierarchy
	Separate I and D$, 32 KB, 8-way, 64B Line	
L2 Physical $	4096KB 16-way	256KB, 8-way
L3 Physical $	None	4096KB, 16-way
Main Memory	3GB Simple Memory Model	

Table 3: System Configurations

TPC-H [7]	1-21 Queries, 1GB DB on MonetDB[15]
specjbb2005 [6]	2 Warehouses
memcached [4, 23]	Throughput Test, 200 TPC/IP connections, 3GB Twitter Data-set
QEMU [12]	Linux OS in a virtual machine
bzip2, h264ref [26]	reference input size
Raytrace, Stream [2]	mobile input size

Table 4: Workloads

For most of the evaluation we use a functional CPU model (AtomicSimple) running Linux and simulate 100B instructions. The default system configurations are presented in the left column of Table 3. Although this does not provide accurate timing information, it provides information regarding all memory accesses and is fast enough to test real world workloads for a long period. We also evaluate potential performance impact conservatively by using more detailed CPU (Out-of-Order) and cache model. The configurations are presented in the right column of Table 3 and for this we simulate 1B instructions at most.

We model hardware structures supporting VC-DSR at a 32 nm process technology with CACTI 6.5 [42]. For benchmarks, we use several real world applications, e.g., DB, server, and mobile workloads, described in Table 4.

4.2 Dynamic Energy Saving

We first consider how much dynamic energy consumption for TLB lookups can be saved with VC-DSR. Ideally, the benefit will be proportional to the cache hit ratio for L1 virtual caches since a TLB is consulted only when cache misses occur. For VC-DSR, in practice, the ART is selectively looked up to obtain a leading virtual address for synonym access, and the ASDT is also referenced for several cases such as cache misses, coherence, TLB invalidations, etc. The organization of the needed structures (e.g., ASDT size) could affect cache misses. These aspects have all to be accounted for when evaluating the overall benefits.

ASDT size: An ASDT with fewer entries tracks fewer pages, which could evict pages whose lines are not dead yet from the cache in order to track newly referenced data. This could increase miss ratio, degrading performance. Hence, an ASDT needs to be adequately

	Instruction Cache Access						Data Cache Access					
	W/O Op		LLVA		LLVA+Kernel		W/O Op		LLVA		LLVA+Kernel	
	Acc.	Hits	Acc.	Hits	Acc.	Hits	Acc.	Hits	Acc.	Hits	Acc.	Hits
QEMU	0	0	0	0	0	0	0.26	0.16	0	0	0	0
Memcached	57	28.1	1.6	0.9	0.1	0.1	39.3	26.8	14	10.2	0.2	0.1
TPC-H	3.9	1.5	0.1	0.1	0.1	0	4	0.9	0.5	0.3	0	0
SPECjbb	0	0	0	0	0	0	0.31	0.26	0	0	0	0
bzip2	0.03	0.03	0	0	0	0	0.12	0.1	0	0	0	0
h264ref	0	0	0	0	0	0	0.01	0.01	0	0	0	0
Stream	0.04	0.03	0	0	0	0	3.02	2.02	0.8	0.5	0.8	0.5
Raytrace	0.01	0.01	0	0	0	0	5.54	3.19	0.2	0.1	0.2	0.1

Table 5: Analysis of ART Accesses

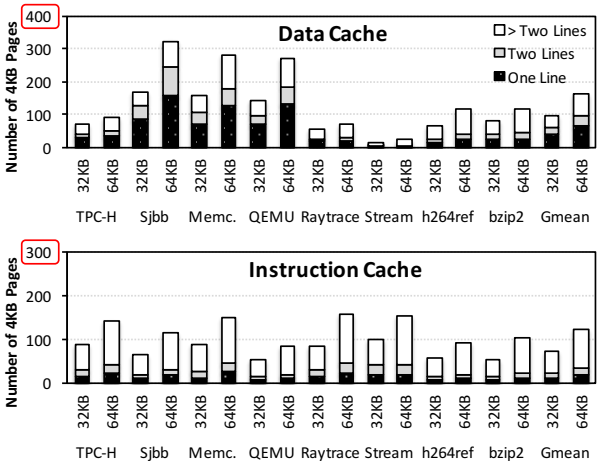

Figure 5: Diversity in pages from which data resides in caches

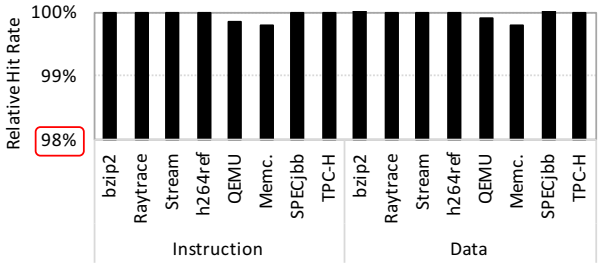

Figure 6: Analysis of VC-DSR L1 VC Hits

provisioned to prevent this case.

Figure 5 shows the average number of distinct 4KB pages from which blocks reside in 32KB (64KB) L1 physical caches with 64B line size. Each bar has three sub-bars. They are classified according to the number of cached lines from each page. Even though 512 (1024) different pages are possible, typically there are an average of less than 150 distinct pages at a given time in most cases. There are fewer distinct pages in instruction caches (e.g., 80) than in data caches (e.g., 100). Moreover, most instruction pages have more than 2 lines cached, whereas many data pages have only 1 or 2 lines cached. The data suggest that a middle-of-the-road sized ASDT is enough and that a smaller ASDT can be used for instruction caches: 128 and 256 entries for 32KB L1 I-cache and for D-cache respectively.

Based on the configuration, the experimental results show that the ASDT entry eviction occurs rarely (less than once per 100 L1 miss) and that this results in at most 1 or 2 VC line evictions. The results suggest that the overhead resulting from the ASDT entry eviction is not significant and that identifying VC-resident lines individually with a bit vector is likely to be useful.

ART access: Table 5 shows how efficiently ART lookups can be managed with a 256-bit SS. The SS is consulted before the address generation, with bits 19-12 of the base register (Op3 in Sec 3.6), and the SS decision not to consult the ART is considered valid if these bits do not change as a result of address generation[7]. For these results, the ART has 32 entries (8 sets and 4 ways). The data presented is: 1) percentage of all cache access that consult the ART after the SS lookup (Acc.) and 2) percentage of all cache accesses that find a matching entry in the ART (Hits), without and with an LLVA register, and the optimization for accesses to the kernel virtual address space.

For most of cases, the small SS can filter out a significant number of ART lookups. For example, for *tpc-h*, the SS filters out about 96.1% of instruction accesses, and only 1.5% of accesses hit in the ART without optimizations. For *memcached* showing most frequent active synonym accesses, notice frequent ART lookups in the base case. Employing an LLVA register can further reduce the number of ART accesses. For *memcached*, notice 55.4% of reductions in ART lookups for instruction accesses. However, due to relatively low temporal and spatial locality of data accesses, we can still see noticeable ART accesses (14%) with an LLVA register. This can be reduced to almost zero with the optimization for kernel virtual address space because most of active synonym accesses occurs in the kernel space.

Since very few ART accesses end up finding a matching entry, a larger sized SS does an even better job of filtering out unnecessary ART accesses (data is not presented). To be conservative in presenting our results, we use a 256-bit SS for the rest of the evaluation.

L1 VC Hits: Figure 6 shows the 32KB L1 virtual cache hit rate of VC-DSR, relative to a PIPT cache. The false misses due to synonymous accesses (Case ③) could increase the miss rate. Notice almost same results across all of the workloads regardless of instruction and

[7]Accessing the SS with a more complex hash (e.g., using more address bits or the sum of the ASID and the address bits) further reduces the number of ART accesses but we present the base results for a conservative configuration.

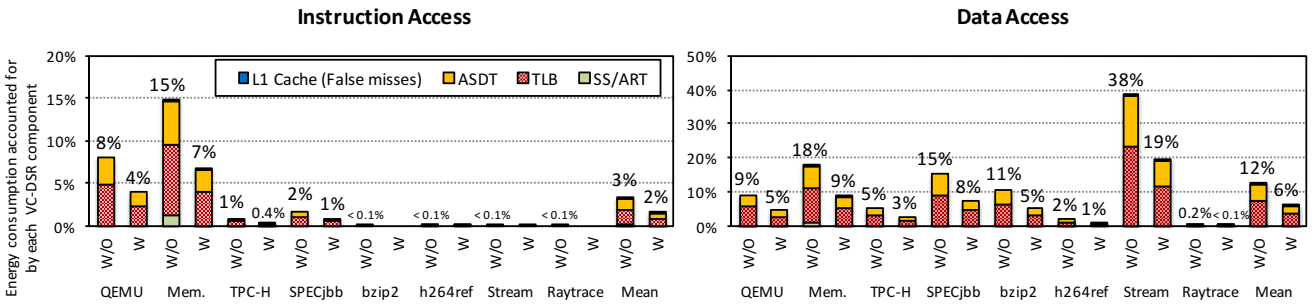

Figure 7: Breakdown of Dynamic Energy Consumption for VC-DSR (baseline 100%, lower is better)

data accesses. This is because the dynamic synonym remapping is performed in an efficient manner.

Energy Saving: Putting it together, we now consider how much TLB lookup energy can be saved with VC-DSR. Figure 7 presents the breakdown of dynamic energy consumption accounted for by each component of VC-DSR; 100% indicates the TLB lookup energy of the baseline. The normal cache access energy is the same in either case, thus we only consider the overhead of additional virtual cache references due to the false misses (Case ③). The without optimization (W/O) bars correspond to the baseline design that we described in Section 3.3 and the with optimization (W) bars include the optimization mentioned in section 3.4.5 and 3.7. A 32-way TLB is used as a baseline.

A few points before discussing the results. First, TLBs with more than 32 entries are common in real designs: ARM Cortex [1] and AMD Opteron [25]. They consume significantly higher energy than the 32-way baseline, and thus we can expect more energy benefits when they are used as a baseline. Second, we use the same TLB organization as the baseline with a VC-DSR, although a VC-DSR would permit us to have a larger, slower, lower-associativity design, which has lower energy consumption (and miss rate). That is, we disadvantage VC-DSR in our comparison.

We will first consider the results without optimization (bars with W/O label). Notice about 97% and 88% (average) energy saving for an instruction and for data cache, respectively. The consumption is dominated by the TLB and ASDT lookups on cache misses. The energy overhead discussed can be reduced further by leveraging the proposed optimization (Sec 3.4.5 and 3.7). We can save the TLB and ASDT lookup overhead for handling an evicted line on every miss by keeping the ASDT index with a cache line. The usage of the LLVA register can reduce ART lookups. We observe that all the features reduce the overall energy consumption by half (bars with W label). Notice about 98% and 94% energy saving for an instruction and for data cache, respectively. This is almost similar to the maximum TLB energy saving from ideal virtual caches (i.e., the baseline L1 cache hit ratio: 99.1% and 96.3% hits for an instruction and for data cache, respectively). *These results suggest that VC-DSR achieves most of the energy benefits of employing pure, but impractical virtual caches.*

4.3 Latency and Timing Benefits

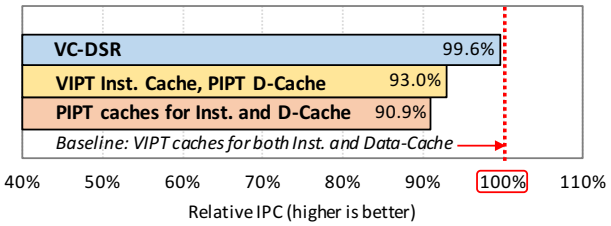

Figure 8: Performance Analysis of VC-DSR

While improved cache access latency—the original motivation for virtual caches—and the consequent performance impact, was not our primary driver, we do mention the quantitative timing benefits we obtained in our experiments (the right side of Table 3). We assume that one cycle is consumed by consulting a TLB, one cycle for consulting an ART and two cycles for active synonym detection (TLB and ASDT lookups). The optimizations discussed above are employed.

Figure 8 shows relative performance of various L1 cache configurations. The baseline uses L1 VIPT caches for both instruction and data accesses, which can completely hide the one cycle latency overhead of TLB lookups. We saw a trivial (0.4%) timing overhead for VC-DSR. We can expect such overhead could be hidden by taking advantage of flexible design choices supported by the usage of virtual caches although we do not consider the impacts in this paper, e.g., TLB organization lowering misses [35] and virtual cache organization fully employing spatial locality of memory access [41]. *The results suggest that VC-DSR also achieves most of the latency benefits of employing virtual caches.*

In addition, some of modern commercial designs [1, 5] use a PIPT data cache to simply avoid problems with synonyms, and thus we can expect latency benefits to some extent (yellow bar).

4.4 Comparison with Other Proposals

There has been a plethora of proposals for virtual caches, each with their own pluses and minuses. We compare VC-DSR with three of them: Opportunistic Virtual Cache (OVC) [11], which is a recent proposal, Two-level Virtual-real Cache hierarchy (TVC) [50], which is an early proposal that has several pluses for a practical implementation, and Synonym Look-aside Buffer (SLB) [46], which takes a somewhat similar approach for

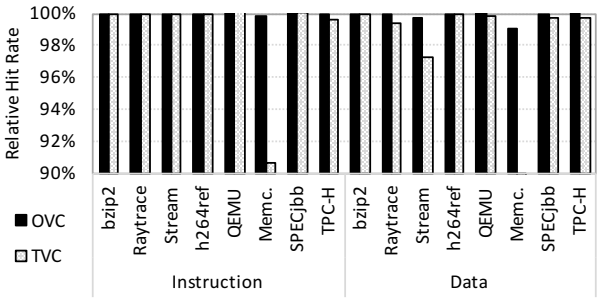

Figure 9: Comparison with Other Approaches

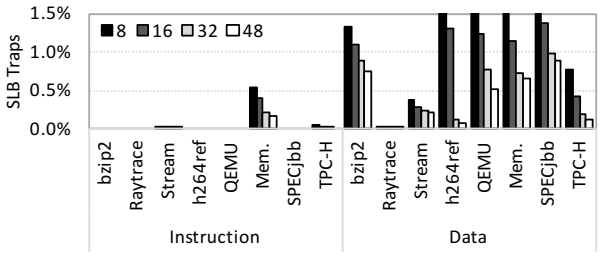

Figure 10: Frequency of SLB miss Traps

handling synonyms (see detailed differences in Sec 5). Note that OVC and SLB require assistance from software, whereas VC-DSR does not. For OVC, we assume an optimistic case where all lines will be cached with virtual addresses, saving the energy consumption for TLB lookups as much as possible.

Figure 9 presents the 32KB L1 cache hit rate of two approaches, OVC and TVC, relative to VC-DSR. Notice that all three proposals show almost same results across most of workloads for instruction accesses, though VC-DSR achieves a slightly higher rate for data accesses showing relatively frequent active synonym accesses. This is because OVC allows duplicate copies of data to be cached with synonyms in a virtual cache, resulting in a reduction in cache capacity. This overhead could be noticeable for caches with smaller associativity or in systems (or hypervisors) with kernel same-page merging dynamically allowing multiple processes (or virtual guests) to share the same physical page (e.g., large disk images). TVC does not have the overhead of data replication, but accesses with a non-leading virtual address increase the number of misses.

Regarding SLB, the performance degradation can result due to SLB traps that occur when the SLB cannot provide a corresponding leading virtual address (LVA) for synonym access. This leads to misses in all the caches in the hierarchy. Further it requires not only TLB lookups but also OS involvement to search for the corresponding LVA (not latency efficient). Accordingly, we analyze the percentage of cache accesses (over total cache accesses) that result in an SLB trap for different SLB sizes in Figure 10. We take a checkpoint after about 10 billion instructions, after the initial boot processes (including setting up a server or loading data) is over (which has significant OS activity and results in many synonyms), and start the SLB simulation from that point. Thus, a significant source of synonyms is left out (e.g., some accesses may never be considered synonyms after the checkpoint, even though they are indeed synonyms and would be identified as such by the OS), thereby advantaging the SLB. Regardless, we see noticeable SLB traps for data cache access although a larger SLB is employed for some cases, e.g., 0.6% for *memcached* with 48 entries. This suggests that using a smaller SLB instead of a TLB may not be a viable solution. Such traps could potentially be further reduced by profiling and dynamically changing the LVA for virtual pages that are being frequently accessed at a given time during the program's execution. However, this entails even more OS involvement.

5. RELATED WORK

A variety of designs have been proposed to achieve the benefits of virtual caches for over four decades; prior literature [16, 17, 30] summarized the problems and some proposed solutions. We discuss some of the most relevant work below.

Qui, *et al.* [46], proposed a *synonym look-aside buffer* (SLB) for virtual caches. The high level approach of VC-DSR is similar to that of the SLB for enforcing a unique (primary) virtual address among synonyms. The SLB is focused on efficiently resolving the scalability issue of the address translation via a TLB by using a relatively smaller SLB structure. On the other hand, VC-DSR aims for a software-transparent latency and energy efficient L1 virtual cache access by exploiting the temporal behavior of *active synonyms*. Significant differences in how SLB and VC-DSR achieve their goals, and their impact, are discussed below.

First, VC-DSR is a software-transparent virtual cache. For the SLB, considerable OS involvement is required to manage synonyms. It requires additional page-table like software structures to track a primary virtual address and other synonyms. Further, all synonyms need to be identified while the related data resides in a large main memory even though many such pages do not actually face synonym issues during their lifetime in smaller caches. In addition, the primary virtual address can change frequently in many cases (OB_3), thus restricting caching to a single, static (OS-determined) primary address can be unnecessarily constraining. These aspects further complicate the OS memory management.

Second, VC-DSR seamlessly provides a synonym remapping only for accesses with active synonyms. The SLB has to be consulted on every cache access to decide the leading virtual address (not power/energy efficient). Although it could be smaller and more scalable than a normal TLB, the misses for synonyms, i.e., SLB miss traps, are expensive, akin to a page table walk (not latency efficient:Fig 10), and all the mappings are shared across different caches in the system. Extra operations are needed to guarantee the consistency among them (like TLB shootdown) although it rarely occurs.

Last, L1 caches are indexed with virtual addresses generated by a CPU and tagged with a primary virtual

address. This results in replication of data with synonyms in the caches, similar to proposal such as OVC, whereas VC-DSR does not have any replication.

Several others [24, 36, 50] have proposed solutions that employ a variant of the (physical to virtual) reverse map to identify and track the data with synonyms in virtual caches. Goodman [24] proposed one of the earliest hardware solutions by using dual tags in virtual caches as the reverse maps. Wang, *et al.* [50], augmented a back-pointer per line in a physical L2 cache to point out the matching data in L1 virtual caches.

Other proposals are supported by the OS to attain the benefits of virtual caches. Some software-based approaches [18, 19, 37, 52] employ a single global virtual address space OS that can eliminate the occurrence of synonyms itself. Zhang, *et al.* [53], proposed Enigma using an additional indirection to efficiently avoid synonym issues. It uses a unique intermediate address (IA) space across the entire system only for cases where data sharing is not expected. Recently, Basu, *et al.* [11], propose Opportunistic Virtual Caching (OVC) based on minor OS modifications. OVC caches a block either with a virtual address or with a physical address depending on the access pattern in a page. A virtual address will be used when caching data with it is safe (i.e., no read-write synonyms occur) or efficient (i.e., few permission changes occur), which could result in multiple duplicates in virtual caches.

Recent work [35], takes a different approach to simplify virtual cache coherence. It eliminates the reverse translation by employing a simple request-response protocol (e.g., self-invalidation/downgrade). Sembrant, *et al.* [47, 48] propose a (virtual) cache design by keeping a way index information in the TLB, obtaining substantial energy saving for cache lookups. Woo, *et al.* [51], employs a bloom filter to reduce serial cache lookups searching for possible locations of synonyms in virtually indexed caches.

6. CONCLUSION

This paper proposes the L1 Virtual Cache with Dynamic Synonym Remapping (VC-DSR). VC-DSR is a purely hardware solution that functions correctly *without any software assist*. By leveraging the temporal behavior of synonyms, VC-DSR dynamically detects active synonyms, and submit such requests with a different (leading) virtual address that was used to place the corresponding data in the virtual cache in an energy and latency efficient manner. Empirical results show that VC-DSR can achieve most of the energy and latency benefits of ideal (but impractical) virtual caches.

We believe that VC-DSR is a practical solution to a problem that has long vexed computer architects: achieving the benefits of virtual caches in a practical manner, i.e., without any reliance on software.

7. REFERENCES

[1] Arm cortex-a72 mpcore processor technical reference manual.

[2] Geekbench 3. http://www.primatelabs.com/geekbench/.

[3] Intel 64 and ia-32 architectures software developer's manual, volume 3a, part1, chapter 2. 2009.

[4] memcached. http://memcached.org.

[5] Migrating a software application from armv5 to armv7-a/r application note 425.

[6] specjbb2005. http://www.spec.org/jbb2005.

[7] tpc-h. http://www.tpc.org/tpch/default.asp.

[8] Sarita V. Adve and Kourosh Gharachorloo. Shared memory consistency models: A tutorial. *IEEE Computer*, 29:66–76, 1995.

[9] Sodani Avinash. Race to exascale: Opportunities and challenges.

[10] Thomas W. Barr, Alan L. Cox, and Scott Rixner. Spectlb: a mechanism for speculative address translation. In *Proc. of the 38th Annual Intnl. Symp. on Computer Architecture*.

[11] Arkaprava Basu, Mark D. Hill, and Michael M. Swift. Reducing memory reference energy with opportunistic virtual caching. In *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ISCA '12, pages 297–308, Washington, DC, USA, 2012. IEEE Computer Society.

[12] Fabrice Bellard. Qemu, a fast and portable dynamic translator. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '05, pages 41–41, Berkeley, CA, USA, 2005. USENIX Association.

[13] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7, 2011.

[14] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, July 1970.

[15] Peter A. Boncz, Martin L. Kersten, and Stefan Manegold. Breaking the memory wall in monetdb. *Commun. ACM*, 51(12):77–85, December 2008.

[16] Michel Cekleov and Michel Dubois. Virtual-address caches part 1: Problems and solutions in uniprocessors. *IEEE Micro*, 17(5):64–71, September 1997.

[17] Michel Cekleov and Michel Dubois. Virtual-address caches, part 2: Multiprocessor issues. *IEEE Micro*, 17(6):69–74, November 1997.

[18] Jeffrey S. Chase, Henry M. Levy, Michael J. Feeley, and Edward D. Lazowska. Sharing and protection in a single-address-space operating system. *ACM Trans. Comput. Syst.*, 12(4):271–307, November 1994.

[19] Jeffrey S. Chase, Henry M. Levy, Edward D. Lazowska, and Miche Baker-Harvey. Lightweight shared objects in a 64-bit operating system. *SIGPLAN Not.*, 27(10):397–413, October 1992.

[20] Douglas W. Clark and Joel S. Emer. Performance of the vax-11/780 translation buffer: Simulation and measurement. *ACM Trans. Comput. Syst.*, 3(1):31–62, February 1985.

[21] Magnus Ekman, Per Stenström, and Fredrik Dahlgren. Tlb and snoop energy-reduction using virtual caches in low-power chip-multiprocessors. In *Proceedings of the 2002 international symposium on Low power electronics and design*, pages 243–246. ACM, 2002.

[22] Dongrui Fan, Zhimin Tang, Hailin Huang, and Guang R. Gao. An energy efficient tlb design methodology. In *Proceedings of the 2005 International Symposium on Low Power Electronics and Design*, ISLPED '05, pages 351–356, New York, NY, USA, 2005. ACM.

[23] Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. Clearing the clouds: a study of emerging scale-out workloads on modern hardware. In *Proceedings of the seventeenth international conference on*

Architectural Support for Programming Languages and Operating Systems, ASPLOS '12, pages 37–48, New York, NY, USA, 2012. ACM.

[24] James R. Goodman. Coherency for multiprocessor virtual address caches. In *Proceedings of the Second International Conference on Architectual Support for Programming Languages and Operating Systems*, ASPLOS II, pages 72–81, Los Alamitos, CA, USA, 1987. IEEE Computer Society Press.

[25] John L. Hennessy and David A. Patterson. *Computer Architecture, Fourth Edition: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

[26] John L. Henning. Spec cpu2006 benchmark descriptions. *SIGARCH Comput. Archit. News*, 34(4):1–17, September 2006.

[27] Jin hyuck Choi, Jung hoon Leek, Seh woong Jeong, Shin dug Kim, and Charles Weems. A low power tlb structure for embedded systems. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA*, 1:2002, 2002.

[28] Bruce Jacob and Trevor Mudge. Virtual memory in contemporary microprocessors. *IEEE Micro*, 18(4):60–75, July 1998.

[29] Bruce Jacob and Trevor Mudge. Uniprocessor virtual memory without tlbs. *Computers, IEEE Transactions on*, 50(5):482–499, 2001.

[30] Bruce Jacob, Spencer Ng, and David Wang. *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.

[31] Bruce L. Jacob. *The Memory System: You Can't Avoid It, You Can't Ignore It, You Can't Fake It*. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2009.

[32] Toni Juan, Tomas Lang, and Juan J Navarro. Reducing tlb power requirements. In *Low Power Electronics and Design, 1997. Proceedings., 1997 International Symposium on*, pages 196–201. IEEE, 1997.

[33] Ismail Kadayif, Partho Nath, Mahmut Kandemir, and Anand Sivasubramaniam. Reducing data tlb power via compiler-directed address generation. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 26(2):312–324, 2007.

[34] Ismail Kadayif, Anand Sivasubramaniam, Mahmut Kandemir, Gokul Kandiraju, and Gu Chen. Generating physical addresses directly for saving instruction tlb energy. In *Microarchitecture, 2002.(MICRO-35). Proceedings. 35th Annual IEEE/ACM International Symposium on*, pages 185–196. IEEE, 2002.

[35] Stefanos Kaxiras and Alberto Ros. A new perspective for efficient virtual-cache coherence. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, pages 535–546, New York, NY, USA, 2013. ACM.

[36] Jesung Kim, Sang Lyul Min, Sanghoon Jeon, Byoungchu Ahn, Deog-Kyoon Jeong, and Chong Sang Kim. U-cache: A cost-effective solution to synonym problem. In *Proceedings of the 1st IEEE Symposium on High-Performance Computer Architecture*, HPCA '95, pages 243–, Washington, DC, USA, 1995. IEEE Computer Society.

[37] Eric J. Koldinger, Jeffrey S. Chase, and Susan J. Eggers. Architecture support for single address space operating systems. In *Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS V, pages 175–186, New York, NY, USA, 1992. ACM.

[38] Hsien-Hsin S Lee and Chinnakrishnan S Ballapuram. Energy efficient d-tlb and data cache using semantic-aware multilateral partitioning. In *Proceedings of the 2003 international symposium on Low power electronics and design*, pages 306–311. ACM, 2003.

[39] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. Pin: building customized program analysis tools with dynamic instrumentation. *ACM Sigplan Notices*, 40(6):190–200, 2005.

[40] Daniel Lustig, Abhishek Bhattacharjee, and Margaret Martonosi. Tlb improvements for chip multiprocessors: Inter-core cooperative prefetchers and shared last-level tlbs. *ACM Trans. Archit. Code Optim.*, 10(1):2:1–2:38, April 2013.

[41] William L. Lynch. The interaction of virtual memory and cache memory. Technical Report CSL-TR-93-587, Stanford University, 1993.

[42] Naveen Muralimanohar and Rajeev Balasubramonian. Cacti 6.0: A tool to model large caches.

[43] Avadh Patel, Furat Afram, Shunfei Chen, and Kanad Ghose. Marss: a full system simulator for multicore x86 cpus. In *Proceedings of the 48th Design Automation Conference*, pages 1050–1055. ACM, 2011.

[44] David A. Patterson and John L. Hennessy. *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2007.

[45] Xiaogang Qiu and Michel Dubois. Tolerating late memory traps in ilp processors. In *Proceedings of the 26th Annual International Symposium on Computer Architecture*, ISCA '99, pages 76–87, Washington, DC, USA, 1999. IEEE Computer Society.

[46] Xiaogang Qiu and Michel Dubois. The synonym lookaside buffer: A solution to the synonym problem in virtual caches. *IEEE Trans. Comput.*, 57(12):1585–1599, December 2008.

[47] Andreas Sembrant, Erik Hagersten, and David Black-Schaffer. The direct-to-data (d2d) cache: Navigating the cache hierarchy with a single lookup. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ISCA '14, pages 133–144, Piscataway, NJ, USA, 2014. IEEE Press.

[48] Andreas Sembrant, Erik Hagersten, and David Black-Shaffer. Tlc: A tag-less cache for reducing dynamic first level cache energy. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-46, pages 49–61, New York, NY, USA, 2013. ACM.

[49] Alan Jay Smith. Cache memories. *ACM Comput. Surv.*, 14(3):473–530, September 1982.

[50] W. H. Wang, J.-L. Baer, and H. M. Levy. Organization and performance of a two-level virtual-real cache hierarchy. In *Proceedings of the 16th Annual International Symposium on Computer Architecture*, ISCA '89, pages 140–148, New York, NY, USA, 1989. ACM.

[51] Dong Hyuk Woo, Mrinmoy Ghosh, Emre Özer, Stuart Biles, and Hsien-Hsin S. Lee. Reducing energy of virtual cache synonym lookup using bloom filters. In *Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, CASES '06, pages 179–189, New York, NY, USA, 2006. ACM.

[52] D. A. Wood, S. J. Eggers, G. Gibson, M. D. Hill, and J. M. Pendleton. An in-cache address translation mechanism. In *Proceedings of the 13th Annual International Symposium on Computer Architecture*, ISCA '86, pages 358–365, Los Alamitos, CA, USA, 1986. IEEE Computer Society Press.

[53] Lixin Zhang, Evan Speight, Ram Rajamony, and Jiang Lin. Enigma: Architectural and operating system support for reducing the impact of address translation. In *Proceedings of the 24th ACM International Conference on Supercomputing*, ICS '10, pages 159–168, New York, NY, USA, 2010. ACM.