

Revisiting Virtual L1 Caches

A Practical Design Using Dynamic Synonym Remapping

Hongil Yoon and Gurindar S. Sohi

Computer Sciences Department
University of Wisconsin-Madison
{ongal, sohi}@cs.wisc.edu

ABSTRACT

Virtual caches have potentially lower access latency and energy consumption than physical caches because they do not consult the TLB prior to 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, the paper proposes a practical design of an L1 virtual cache that (1) dynamically decides a unique virtual page number for all the synonymous virtual pages that map to the same physical page and (2) uses this unique page number to place and look up data in the virtual caches. Accesses to this unique page number proceed without any intervention. Accesses to other synonymous pages are dynamically detected, and remapped to the corresponding unique virtual page number to correctly access data in the cache. Such remapping operations are rare, due to the temporal properties of synonyms, allowing a Virtual Cache with Dynamic Synonym Remapping (VC-DSR) to achieve most of the benefits of virtual caches but without software involvement.

Experimental results based on real world applications show that VC-DSR can achieve about 92% of the dynamic energy savings for TLB lookups, and 99.4% of the latency benefits of ideal (but impractical) virtual caches for the configurations considered.

1. INTRODUCTION

Virtual memory and caches are two of the most common elements of computers today. A processing core generates virtual addresses, which are translated to physical addresses so that the appropriate memory accesses can be made. With Physically Indexed, Physically Tagged (PIPT) caches, a virtual to physical translation is performed, via a TLB, prior to cache access. This consumes significant power [1], and adds latency to the cache access. 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 (e.g., requiring a larger associativity) [2, 3, 4]. Over the years, a plethora of techniques have been proposed and deployed to reduce the latency and energy impacts of the (prior) TLB ac-

cess [5, 6, 7, 8, 9, 10, 11, 12, 13]. However, the basic problem still remains in physical caches—where a physical address is used for cache access—especially for L1 caches.

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 [14, 15], however, *virtual caches have not been considered to be practical*. The primary reason is the complications due to *synonyms*.

For a variety of reasons [2, 14, 16], multiple virtual pages can be mapped to the same physical page. The same data can be placed and looked up later with multiple 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 [17, 18], coherence [19], and violations of memory consistency and of sequential program semantics [16, 20] (Section 3.6). Several virtual cache designs have been proposed to obviate the overheads of physical caches [2, 15, 16, 19, 21, 22, 23, 24]. However, the plethora of problems complicate the cache design and impair the potential benefits of virtual caches (Section 4.4 and 5).

This paper describes and evaluates a practical L1 virtual cache design, called VC-DSR: *Virtual Cache with Dynamic Synonym Remapping*. 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 an 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 without intervention, while accesses made to the same data with a different address, V_j , are remapped to use V_i instead; dynamic remappings from a non-leading (virtual) address to a leading address are needed for *select accesses*, rather than performing virtual to physical translation, via a TLB, for every access.

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

In this paper:

- We present novel empirical observations for the temporal behavior of synonyms, based on real world server and mobile workloads (Section 2).
- We propose VC-DSR, a practical solution to a problem that has long vexed computer architects: achieving most of 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 (Section 3.6), we discuss synonym issues and solutions for VC-DSR.
- Presented experimental results show that VC-DSR saves about 92% of dynamic energy consumption for TLB lookups and also achieves most of the latency benefit (about 99.4%) of ideal (but impractical) virtual caches (Section 4).

The remainder of this paper is as follows. Section 2 presents an empirical evaluation of the temporal nature of synonyms. Motivated by this data, Section 3 presents the overall design of VC-DSR and details the hardware structures needed to ensure correct operation. Section 4 assesses the effectiveness of VC-DSR. Section 5 presents selected related work, and we conclude in Section 6.

2. TEMPORAL SYNONYM BEHAVIOR

A variety of common programming and system practices lead to synonyms [2, 14, 16], 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

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

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 $TC(X,T) \geq 2$ 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 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 $C(X) \geq 2$, 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 $TC(X,T) \geq 2$ 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 in-

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

Size (KB)	(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	
	32	64	128	256	32	64	128	256	32	64	128	256	32	64	128	256	32	64	32	64
TPC-H	6	20	57	133	2	5	8	12	2	3	8	32	2	2	2	2	63	57	73	74
SPECjbb2005	0	0	2	22	0	1	2	2	-	-	2	2	-	-	2	2	69	67	63	56
Memcached	36	87	158	282	52	93	160	284	2	3	12	28	2	2	2	2	4	0	5	5
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

creases 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 libraries and kernel interfaces. However:

Observation 2 (OB₂): *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 therefore 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 page for that EPS. We say that a change in the leading virtual address 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). Only pages in which active synonym accesses occur at least once 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 made with other virtual addresses. This results in fewer LVA changes.

Observation 3 (OB₃): *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

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

Table 2: Number of References (per 1000) to Pages with Active Synonyms in Caches

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 .

The first set of data suggests that the overall percentage of accesses to cache blocks in pages 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 in pages 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 en-

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

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

Observation 4 (OB₄): Typically only a small fraction of cache accesses are to cache lines from pages with active synonyms.

Observation 5 (OB₅): In most cases, a very small percentage of 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. If so, the address is used to look up the virtual cache. 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.

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 a 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 Active Synonym Detection Table 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), ensuring the correctness of overall cache operations.

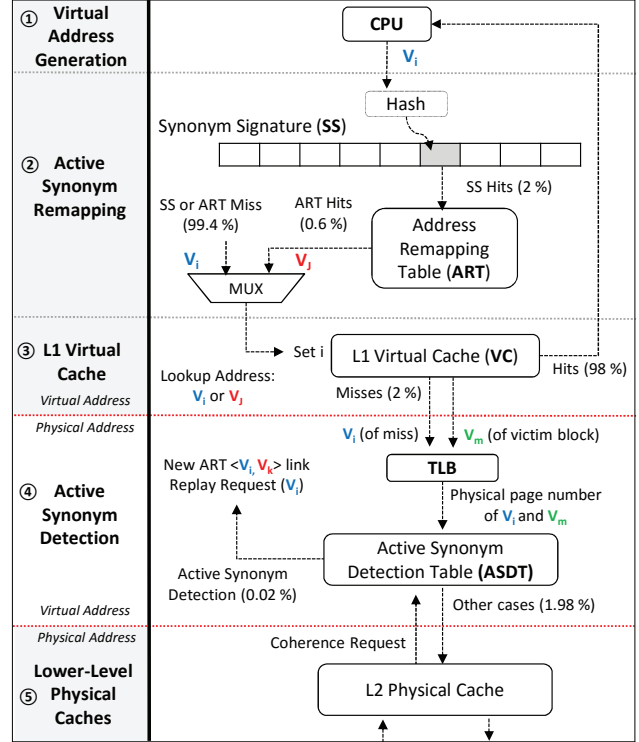


Figure 1: Schematic Overview of VC-DSR

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)⁴ generated by a CPU. 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 address, 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 will be used as the leading virtual page. If

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

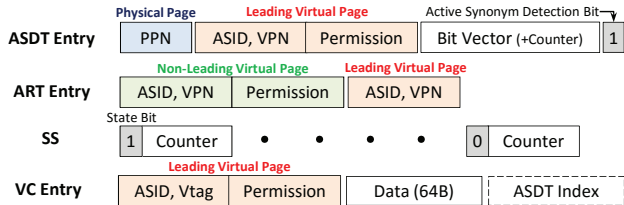


Figure 2: Overview of Entries for Structures Supporting VC-DSR

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 set in the SS. 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 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, along with its permission bits. For the former, a counter suffices for correctness. However, for more efficient evictions (Section 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, and an active synonym bit in the entry is set if there is one. In addition, the ASDT is also consulted to perform the physical to the (leading) virtual address translation for coherence requests from lower-level 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 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, along with the permission bits of the non-leading page, for an active synonym. On a match, the ART returns the leading virtual page and the permission bits for the requested (non-leading) virtual address. The permission check for accesses with a non-leading virtual address is performed at this point (details in Section 3.6) and 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 for that page, i.e., a virtual address 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 Section 3.4.5).

Accesses to the ART are unnecessary when the referenced virtual address is a leading virtual address, i.e., no match in the ART. A Synonym Signature (SS) is used to elide most unneeded ART lookups. The SS is a Bloom filter [25], which is accessed with the virtual address 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, and thus false positives are possible.

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 virtual address 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 is effectively the same as a traditional virtual cache, 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 with each entry.

3.4 Overall Operation

We describe the overall operation of VC-DSR. Figure 3 illustrates how virtual addresses are used 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 (if 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 (Section 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 bits.

The line is fetched from the lower-level(s).⁵ The corresponding bit in the bit vector of the ASDT entry is set and

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

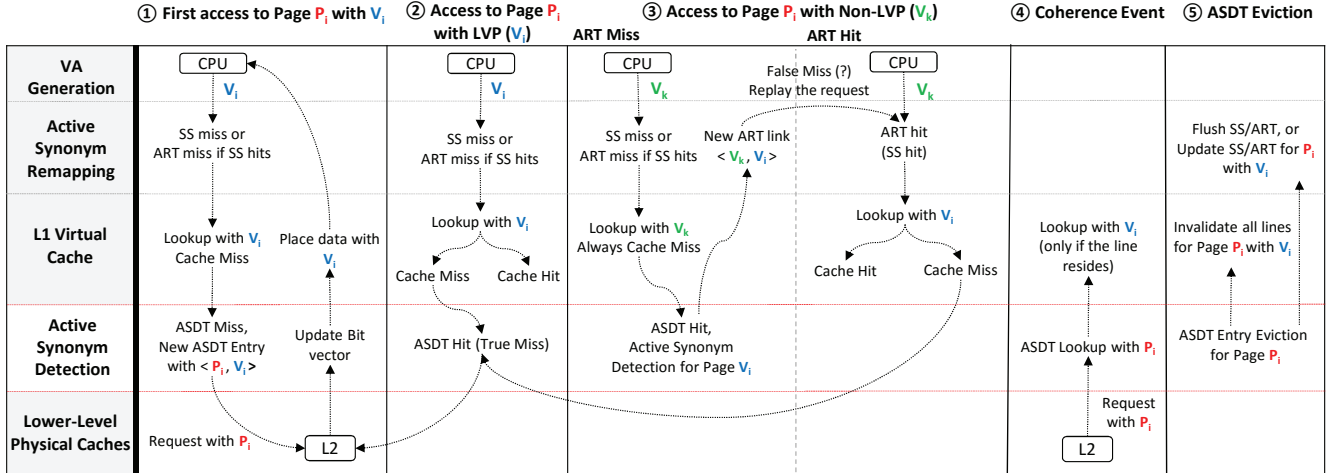


Figure 3: Overall Operations of VC-DSR

the counter is incremented. Then the fetched line (and the permission bits) is placed in the VC with the leading virtual address (V_i). 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 a 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). An active synonym has not yet been detected for this page. Thus, V_k is used to look up the virtual cache. Since V_k is not a leading virtual page (LVP), the access will result in a miss. The TLB is then accessed to translate V_k (to P_i), a matching entry for the common physical page (P_i) is found in the ASDT, a new active synonym is detected, and the ART is informed to create a new entry for the $\langle V_k, V_i \rangle$ tuple (and the SS is modified accordingly). Any further accesses to virtual page V_k will be remapped with V_i via the ART (right side of Case ③). Since the data may actually reside in the VC with the leading virtual address (i.e., it is a false miss), 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

A coherence event typically uses physical addresses. Thus, it is handled as normal for lower-level (PIPT) caches. For coherence activities between L2 and virtual L1 caches (Case ④), 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 [2]. 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).

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 Figure 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 changes 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).

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 Design Choices for SS and ART Lookups

In the previous sections, we describe the overall operations of our proposal by conceptually assuming that a synonym signature (SS) is consulted prior to the phase of looking up the L1 cache. This may lengthen the cache access path, although the SS is small; we can still expect most of the power/energy benefits of using virtual caches but perhaps not the potential latency benefits. However, the SS and ART (if needed) accesses could be performed in a manner that may not affect the timing of the actual L1 access, depending upon the microarchitecture.

Since memory operations typically require additional steps between an address generation and submission to the L1 cache (e.g., load/store queue, disambiguation), there are several choices for where/when the SS and ART are consulted. The SS and ART can be accessed in parallel with memory disambiguation (Option 1). In practice, the SS can be accessed *before* the address generation, based upon the contents of a base (or segment) register (Option 2). If the address generation is a multi-step process, the SS could be accessed after intermediate steps of the process (Option 3). By consulting an SS in advance (Options 2 and 3) or in parallel with other operations in the pipeline (Option 1), the ART could be selectively accessed before L1 cache lookups (or in the pipeline). Option 2 looks appealing since it can be applied to both I and D caches, and most of the energy and latency benefits can be achieved without an increase in design complexity. In addition, SS and ART accesses could be bypassed for store requests (e.g., 30% of data accesses) because write synonym accesses are rare [2].

3.6 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 (e.g., mapping and permissions) changes, all the entries corresponding to that page in all the components supporting VC-DSR need to be invalidated (or updated) to ensure consistency (Section 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, such events can be filtered out with a simple filter based on the ASDT information. When a matching entry is found, only the corresponding lines can be selectively evicted from the VC with a bit vector. Thus, this does not have a significant impact on the effectiveness of our proposal, although such events are not rare [26].

Permission check: Depending on whether an access is to a leading or non-leading virtual page, the location of checking (keeping) the page permission bits is different. For the former case, each line in the VC tracks the permission bits, while an entry in the ART maintains the permission bits for a non-leading virtual page. The permission check is done when a 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 store to read only page exception that includes the page information change discussed above.

If the active synonym remapping phase is completely by-

passed for stores (Section 3.5), the ART does not necessarily need to track permission bits for non-leading virtual pages. Stores with non-leading virtual addresses will simply miss, and the permission check will be performed by consulting a TLB as normal.

Non-cacheable data access: Cacheability of accessed data is recognized when a corresponding TLB entry is looked up. VC-DSR consults a TLB only when virtual cache (VC) misses occur. Thus an access to non-cacheable data triggers an L1 VC miss, increasing the access latency. The overhead (e.g., 1-2 ns) is a very small portion of the overall latency (e.g., 50-100 ns for memory reference). Hence, the actual timing overhead will not be significant even though such accesses are not rare.

Hardware page-table walk based on physical addresses: Some architectures (e.g., x86) support a hardware page-table walker [17, 18]. 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 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 addresses for a conventional write buffer (or store queue) can result in a violation of sequential semantics: a load may not identify the matching latest store due to synonyms, and vice versa. Thus, stale data could be returned from caches or from a matching older store. 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 that has been carried out (speculatively) for the corresponding data may need to be identified and replayed. In some commercial processors, these issues are handled by finding potentially offending loads by matching the physical addresses and replaying them. For virtual caches, however, synonyms can complicate their identification.

Most of the prior literature has not discussed these issues, and the proposed solution for the former issue may not be efficient [16]. However, VC-DSR can easily handle all of them, as follows: the key idea is to employ an LVA as an alternative to a physical address. Accesses with a non-leading virtual address always cause VC misses and are eventually replayed with the LVA via an ART (Case ③). Thus, once a load/store is (speculatively) executed, its LVA is identified and kept in the load/store queue. The unique LVA 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 to 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. A similar proposal has been made to tolerate late memory traps [27].

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).

Supporting H/W Prefetchers: H/W prefetchers employed by modern processors can be seamlessly integrated with our proposal. For example, a stride prefetcher and stream buffer [28, 29] can use the physical address from the translation caused by a cache miss (i.e., no additional TLB lookups). Multiple prefetches initiated in a (large) page can also hide the potential overhead of multiple TLB lookups (or ASDT lookups if prefetched data is placed in the L1 VC).

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 [30], 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.

⁶Half 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 accuracy.

	1	2
CPU type	AtomicSimple	3GHz, 8-way Out-of-Order
Num. CPU	Single-Core	
L1 Virtual \$	Classic \$ Model Ruby 3-level hierarchy	
	Separate I and D\$, 32 KB, 8-way 64B Line, 1ns access latency	
L2 Physical \$	4096KB 16-way	256KB, 8-way, 3ns latency
L3 Physical \$	None	4096KB, 16-way, 10ns latency
Main Memory	3GB Simple Memory Model, 50ns access latency	

Table 3: System Configurations

TPC-H [31]	1-21 Queries, 1GB DB on MonetDB[32]
SPECjbb2005 [33]	2 Warehouses
Memcached [34, 35]	Throughput Test, 3GB Twitter Data-set
bzip2, h264ref [36]	reference input size
Raytrace [37]	mobile input size
Stream (Copy, Add, Scale, Triad) [37]	desktop input size, sequential execution of four different Stream workloads

Table 4: Workloads

4. EVALUATION

We now evaluate the effectiveness of VC-DSR. First, Section 4.1 describes the evaluation methodology and workloads. Then, we evaluate how much dynamic energy consumption can be saved in Section 4.2, briefly evaluate the latency benefits in Section 4.3 and compare VC-DSR with three other virtual cache proposals in Section 4.4.

4.1 Evaluation Methodology

We attached modules to simulate our proposal in the Gem5 x86 simulator [38]. 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 [38, 39] supporting a full-system environment are too slow, and instrumentation tools [40], while relatively fast, provide inadequate full-system support.

For most of the evaluation, we use a functional CPU model (AtomicSimple⁷) running Linux and simulate a maximum of 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 models. The configurations are presented in the right column of Table 3 and for this we simulate up to 1B instructions.

We model hardware structures supporting VC-DSR at a 32 nm process technology with CACTI 6.5 [41]. For benchmarks, we use several real world applications, e.g., DB, server,

⁷More sophisticated processors may show different instruction cache access patterns, to some extent, by using diverse techniques to improve the efficiency of the fetch stage, e.g., fetch/loop buffers, branch predictors, prefetchers, etc.

	Instruction Cache Access								Data Cache Access							
	W/O Op		Kernel Op		LLVA Op		Kernel+LLVA		W/O Op		Kernel Op		LLVA Op		Kernel+LLVA	
	Acc.	Hits	Acc.	Hits	Acc.	Hits	Acc.	Hits	Acc.	Hits	Acc.	Hits	Acc.	Hits	Acc.	Hits
Memcached	56.9	28.1	3.41	2.13	1.61	0.9	0.08	0.06	39.3	26.8	0.97	0.63	13.9	10.1	0.05	0.03
TPC-H	3.86	1.5	1.88	0.26	0.12	0.1	0.02	0.01	3.95	0.92	0.21	0.15	0.52	0.3	0.04	0.03
SPECjbb	0	0	0	0	0	0	0	0	0.35	0.27	0.33	0.27	0.04	0.02	0.04	0.02
bzip2	0.03	0.03	0	0	0	0	0	0	0.12	0.1	0.06	0.05	0	0	0	0
h264ref	0	0	0	0	0	0	0	0	0.01	0.01	0	0	0	0	0	0
Stream	0.04	0.03	0	0	0	0	0	0	3.02	2.02	2.86	1.91	0.8	0.5	0.79	0.44
Raytrace	0.01	0.01	0	0	0	0	0	0	5.54	3.19	5.53	3.18	0.2	0.1	0.17	0.06

Table 5: Analysis of ART Accesses

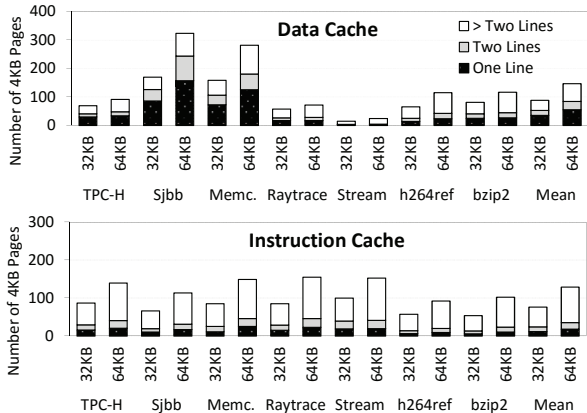


Figure 4: Diversity in Pages whose Data Resides in Caches

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 also 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 the cache miss ratio, degrading performance. Hence, an ASDT needs to be adequately provisioned to prevent this case.

Figure 4 shows the average number of distinct 4KB pages from which blocks reside in 32KB (64KB) L1 physical caches with a 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

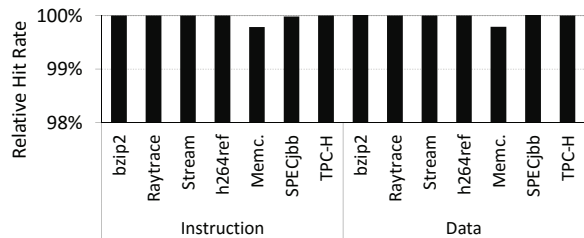


Figure 5: Analysis of VC-DSR L1 VC Hits

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 an ASDT entry eviction occurs rarely (less than once per 100 L1 misses) 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. We assume that the SS is consulted before the address generation, with bits 19-12 of the base register with a 4KB fixed offset (Option 2 in Section 3.5). The SS decision not to consult the ART is considered valid if these bits do not change as a result of address generation.⁸ For these results, we consider 32KB L1 virtual caches and the ART has 32 entries (8 sets and 4 ways). The data presented is: 1) percentage of all cache accesses 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 any optimizations (W/O Op), with an LLVA register (LLVA Op), and with the optimization for accesses to the kernel virtual address space (Kernel Op).

For most cases, the small SS can filter out a significant number of ART lookups. For example, for *tpc-h*, the SS filters out about 96% 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 (i.e., SS hits) in the base case. The use of the optimization for kernel virtual address space significantly reduces the number of ART accesses. This is because most of the active synonym accesses for *memcached*

⁸ 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.

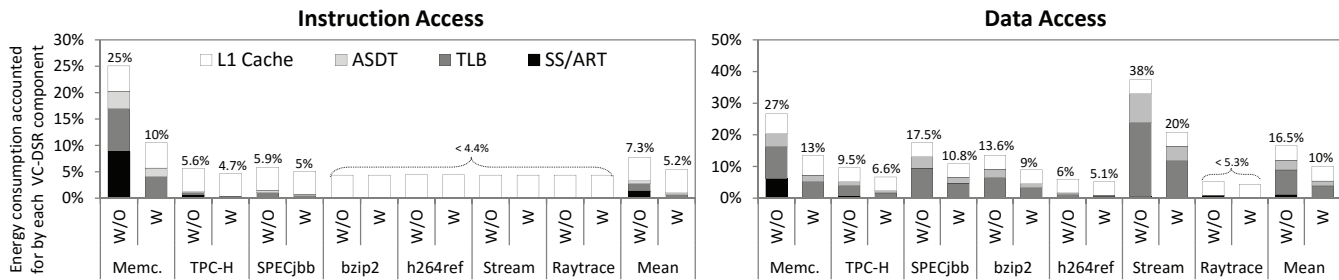


Figure 6: Breakdown of Dynamic Energy Consumption for VC-DSR (baseline 100%, lower is better)

occur in the kernel space. The optimization completely prevents such accesses from being considered as active synonym accesses. Thus this makes the SS less conservative, considerably reducing the number of SS hits. Employing an LLVA register is also effective. For *memcached*, notice a significant reduction 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 (about 14%) with an LLVA register. This can be reduced to almost zero with the optimization for kernel virtual address 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 5 shows the hit rate of a 32KB L1 VC-DSR, relative to a PIPT cache; the false misses due to synonymous accesses (Case ③ in Figure 3) are treated as a miss. Notice almost the same results across all of the workloads regardless of instruction and data accesses.

Energy saving: Putting it together, we now consider how much TLB lookup energy can be saved with VC-DSR. Figure 6 presents the breakdown of dynamic energy consumption accounted for by each component of VC-DSR. A 32 entry, fully associative 3-ported TLB is used as a baseline. 100% indicates the TLB lookup energy of the baseline. 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. For virtual cache lookups (bars with L1 Cache label), we consider the extra overhead due to reading additional bits (e.g., ASID and extra virtual tag bits) on every access as well as the false misses.

A few points before discussing the results. First, TLBs with more than 32 entries are common in real designs, e.g., ARM Cortex [42] and AMD Opteron [43]. 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 VC-DSR, although VC-DSR would permit us to have a larger, slower, lower-associativity, fewer ports design, which has lower energy consumption (and miss rate). Third, our proposal removes constraints on the organization of virtually indexed caches (i.e., a larger associativity). Thus we can also expect lower power/energy consumption for L1 cache lookups [2, 3]. We do not consider these benefits from

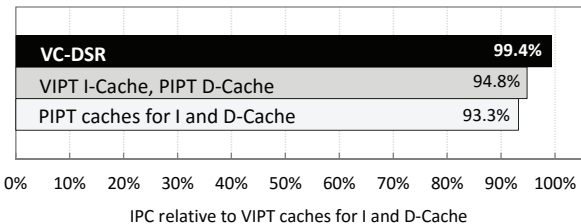


Figure 7: Performance Analysis of VC-DSR

the flexible design choices of VC-DSR in our comparison, thereby disadvantaging VC-DSR.

We will first consider the results without optimization (bars with W/O label). Notice about 93% and 84% (average) energy saving for an instruction and for data TLB, respectively. We observe that a small portion of the energy overhead is accounted for by reading the additional bits on every L1 VC lookup, regardless of instruction and data accesses. The consumption is dominated by the TLB and ASDT lookups on cache misses (especially for data accesses). The energy overhead except the overhead for the L1 VC lookups can be reduced further by leveraging the proposed optimization (Section 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 use of an LLVA register can reduce ART lookups. We observe that all the optimizations reduce the energy consumption by half (bars with W label), resulting in about 95% and 90% energy saving for an instruction and for data TLB, respectively. In comparison, an ideal virtual cache would achieve about a 99% and 96% reduction for an instruction and data TLB, respectively, in this situation. *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

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 three cycles for active synonym detection (TLB and ASDT lookups). The optimizations discussed above and the virtually addressed Load-Store Unit discussed in Section 3.6 are employed.

Figure 7 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 (0.6%) 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, e.g., TLB organization lowering misses [44] and virtual cache organization fully employing spatial locality of memory access [45]. *The results suggest that VC-DSR also achieves most of the latency benefits of employing virtual caches.*

In addition, some commercial designs [42, 46] use a PIPT data cache to simply avoid problems with synonyms. VC-DSR has a significant timing benefit over such designs.

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) [2], which is a recent proposal, Two-level Virtual-real Cache hierarchy (TVC) [22], which is an early proposal that has several pluses for a practical implementation, and Synonym Look-aside Buffer (SLB) [16], which takes a somewhat similar approach for handling synonyms (see detailed differences in Section 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. For TVC, we assume that an ASID is employed to address the issue of homonyms, which excludes the impact of flushing virtual caches on a context switch [47].

Figure 8 presents the L1 cache hit rate of two approaches, OVC and TVC, relative to VC-DSR for 32KB and 64KB caches. Notice that VC-DSR achieves a slightly higher hit rate due to more efficient handling of active synonym accesses. 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 with kernel same-page merging dynamically allowing multiple processes to share the same physical page (e.g., virtualized environments) In addition, this approach could impose restrictions on designing a viable practical implementation because the issues of virtually addressed Load-Store Units (Section 3.6) may not be efficiently handled.

TVC does not keep duplicate copies of data in virtual caches like our proposal, but accesses with a synonymous address that is different from a virtual address used to cache data result in cache misses. This requires additional operations to cache a synonym copy with a most recently referenced virtual address. It is possible for the synonym copy to be relocated to a different set, and thus TVC also has the overhead of data replication in virtual caches. The cost of handling synonymous accesses could be more noticeable depending on the multiprogramming environment and microarchitecture design (e.g., SMT). Furthermore, TVC works only with an inclusive cache hierarchy.

For SLB, performance degradation can result due to SLB traps that occur when the SLB cannot provide a correspond-

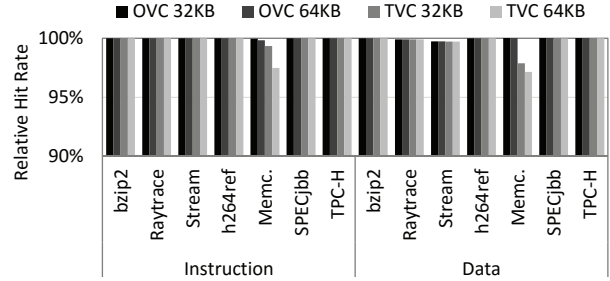


Figure 8: Comparison with Other Approaches

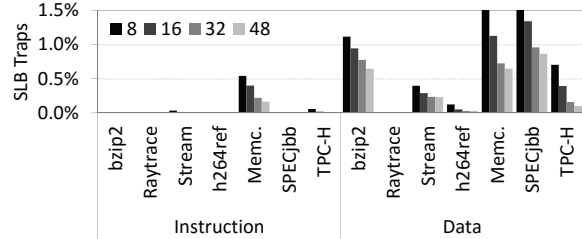


Figure 9: Frequency of SLB Miss Traps

ing 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 that result in an SLB trap for different SLB sizes in Figure 9. In this analysis, we take a checkpoint after the initial boot processes (including setting up a server or loading data) is over, and start the SLB simulation from this point. This initial phase has significant OS activity and results in many synonyms and thus a significant source of the total number of potential synonyms is left out in our analysis, 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 [14, 48, 49] summarized the problems and some proposed solutions. We discuss some of the most relevant work below.

Qui *et al.* [16] 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: Figure 9), and all the mappings are shared across different caches in the system. Extra operations are needed to guarantee the consistency among them (like TLB shutdown) although it rarely occurs.

Several others [15, 21, 22] 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 [15] proposed one of the earliest hardware solutions by using dual tags in virtual caches as the reverse maps. Wang *et al.* [22] augmented a back-pointer per line in a physical L2 cache to point to 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 [23, 50, 51, 52] employ a single global virtual address space OS that can eliminate the occurrence of synonyms itself. Zhang *et al.* [24] proposed Enigma using an additional indirection to efficiently avoid synonym issues. It uses a unique intermediate address (IA) space across the entire system for cases where data sharing is not expected. Recently, Basu *et al.* [2] proposed 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 [19] 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.* [53, 54] propose a (virtual) cache design by keeping a way index information in the TLB, obtaining substantial energy saving for cache lookups. Woo *et al.* [55] employ a Bloom filter to reduce serial cache lookups searching for possible locations of synonyms in virtually indexed caches.

6. CONCLUSION

This paper proposes Virtual Cache with Dynamic Synonym Remapping (VC-DSR) for use as an L1 cache. 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 submits such requests with a different (leading) virtual address that was used to place the corresponding data in the virtual cache. 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 reliance on software.

Acknowledgment

This work was supported in part by funding from the William F. Vilas Trust Estate (Vilas Research Professor), the University of Wisconsin Foundation (John P. Morgridge Professor), and the Wisconsin Alumni Research Foundation. The authors thank the anonymous reviewers for their insightful feedback, and Srinath Sridharan, Gagan Gupta, Mark D. Hill, David A. Wood, Karu Sankaralingam, Lena E. Olson, Nilay Vaish, Jayneel Ghandi, Clint Lestourgeon, and Aditya Venkataraman for their helpful comments and discussions.

7. REFERENCES

- [1] A. Sodani, "Race to Exascale: Opportunities and Challenges," MICRO 2011 Keynote talk.
- [2] A. Basu, M. D. Hill, and M. M. Swift, "Reducing Memory Reference Energy with Opportunistic Virtual Caching," in *Proceedings of the 39th Annual International Symposium on Computer Architecture (ISCA)*, pp. 297–308, 2012.
- [3] B. Jacob, *The Memory System: You Can'T Avoid It, You Can'T Ignore It, You Can'T Fake It*. Morgan and Claypool Publishers, 2009.
- [4] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufmann Publishers Inc., 3rd ed., 2007.
- [5] M. Ekman, P. Stenström, and F. Dahlgren, "TLB and Snooper Energy-reduction Using Virtual Caches in Low-power Chip-multiprocessors," in *Proceedings of the 2002 International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 243–246, 2002.
- [6] D. Fan, Z. Tang, H. Huang, and G. R. Gao, "An Energy Efficient TLB Design Methodology," in *Proceedings of the 2005 International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 351–356, 2005.
- [7] J.-H. Choi, J.-H. Lee, S.-W. Jeong, S.-D. Kim, and C. Weems, "A Low Power TLB Structure for Embedded Systems," *IEEE Comput. Archit. Lett.*, vol. 1, pp. 3–3, Jan. 2002.
- [8] B. Jacob and T. Mudge, "Uniprocessor Virtual Memory Without TLBs," *IEEE Trans. Comput.*, vol. 50, pp. 482–499, May 2001.
- [9] T. Juan, T. Lang, and J. J. Navarro, "Reducing TLB Power Requirements," in *Proceedings of the 1997 International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 196–201, 1997.
- [10] I. Kadayif, P. Nath, M. Kandemir, and A. Sivasubramaniam, "Reducing Data TLB Power via Compiler-Directed Address Generation," *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, vol. 26, pp. 312–324, Feb. 2007.
- [11] I. Kadayif, A. Sivasubramaniam, M. Kandemir, G. Kandiraju, and G. Chen, "Generating Physical Addresses Directly for Saving Instruction TLB Energy," in *Proceedings of the 35th Annual ACM/IEEE International Symposium on Microarchitecture (MICRO)*, pp. 185–196, 2002.

- [12] H.-H. S. Lee and C. 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 (ISLPED)*, pp. 306–311, 2003.
- [13] K. C. Yeager, "The MIPS R10000 Superscalar Microprocessor," *IEEE Micro*, vol. 16, pp. 28–40, Apr. 1996.
- [14] M. Cekleov and M. Dubois, "Virtual-Address Caches Part 1: Problems and Solutions in Uniprocessors," *IEEE Micro*, vol. 17, pp. 64–71, Sept. 1997.
- [15] J. R. Goodman, "Coherency for Multiprocessor Virtual Address Caches," in *Proceedings of the Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 72–81, 1987.
- [16] X. Qiu and M. Dubois, "The Synonym Lookaside Buffer: A Solution to the Synonym Problem in Virtual Caches," *IEEE Trans. Comput.*, vol. 57, pp. 1585–1599, Dec. 2008.
- [17] Intel, "Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3A, Part 1, Chapter 2." http://www.intel.com/Assets/en_US/PDF/manual/253668.pdf.
- [18] B. Jacob and T. Mudge, "Virtual Memory in Contemporary Microprocessors," *IEEE Micro*, vol. 18, pp. 60–75, July 1998.
- [19] S. Kaxiras and A. Ros, "A New Perspective for Efficient Virtual-cache Coherence," in *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA)*, pp. 535–546, 2013.
- [20] S. V. Adve and K. Gharachorloo, "Shared Memory Consistency Models: A Tutorial," *Computer*, vol. 29, pp. 66–76, Dec. 1996.
- [21] J. Kim, S. L. Min, S. Jeon, B. Ahn, D.-K. Jeong, and C. S. Kim, "U-cache: A Cost-effective Solution to Synonym Problem," in *Proceedings of the 1st IEEE Symposium on High-Performance Computer Architecture (HPCA)*, pp. 243–252, 1995.
- [22] 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)*, pp. 140–148, 1989.
- [23] 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)*, pp. 358–365, 1986.
- [24] L. Zhang, E. Speight, R. Rajamony, and J. 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)*, pp. 159–168, 2010.
- [25] B. H. Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors," *Commun. ACM*, vol. 13, pp. 422–426, July 1970.
- [26] C. Villavieja, V. Karakostas, L. Vilanova, Y. Etsion, A. Ramirez, A. Mendelson, N. Navarro, A. Cristal, and O. S. Unsal, "DiDi: Mitigating the Performance Impact of TLB Shootdowns Using a Shared TLB Directory," in *Proceedings of the 2011 International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pp. 340–349, 2011.
- [27] X. Qiu and M. Dubois, "Tolerating Late Memory Traps in ILP Processors," in *Proceedings of the 26th Annual International Symposium on Computer Architecture (ISCA)*, pp. 76–87, 1999.
- [28] J.-L. Baer and T.-F. Chen, "An Effective On-chip Preloading Scheme to Reduce Data Access Penalty," in *Proceedings of the 1991 ACM/IEEE Conference on Supercomputing*, pp. 176–186, 1991.
- [29] S. Palacharla and R. E. Kessler, "Evaluating Stream Buffers As a Secondary Cache Replacement," in *Proceedings of the 21st Annual International Symposium on Computer Architecture (ISCA)*, pp. 24–33, 1994.
- [30] D. W. Clark and J. S. Emer, "Performance of the VAX-11/780 Translation Buffer: Simulation and Measurement," *ACM Trans. Comput. Syst.*, vol. 3, pp. 31–62, Feb. 1985.
- [31] TPC-H. <http://www.tpc.org/tpch/default.asp>.
- [32] P. A. Boncz, M. L. Kersten, and S. Manegold, "Breaking the Memory Wall in MonetDB," *Commun. ACM*, vol. 51, pp. 77–85, Dec. 2008.
- [33] SPECjbb2005. <http://www.spec.org/jbb2005>.
- [34] Memcached. <http://memcached.org>.
- [35] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafae, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, "Clearing the Clouds: A Study of Emerging Scale-out Workloads on Modern Hardware," in *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 37–48, 2012.
- [36] J. L. Henning, "SPEC CPU2006 Benchmark Descriptions," *SIGARCH Comput. Archit. News*, vol. 34, pp. 1–17, Sept. 2006.
- [37] Geekbench. <http://www.primatelabs.com/geekbench/>.
- [38] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The Gem5 Simulator," *SIGARCH Comput. Archit. News*, vol. 39, pp. 1–7, Aug. 2011.
- [39] A. Patel, F. Afram, S. Chen, and K. Ghose, "MARSS: A Full System Simulator for Multicore x86 CPUs," in *Proceedings of the 48th Design Automation Conference (DAC)*, pp. 1050–1055, 2011.
- [40] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation," in *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pp. 190–200, 2005.
- [41] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "CACTI 6.0: A tool to model large caches," *HP Laboratories*, 2009.
- [42] ARM, "ARM Cortex-A72 MPCore Processor Technical Reference Manual." http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.100095_0001_02_en/index.html.
- [43] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Fourth Edition: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., 2006.
- [44] A. Bhattacharjee, D. Lustig, and M. Martonosi, "Shared Last-level TLBs for Chip Multiprocessors," in *Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture (HPCA)*, pp. 62–63, 2011.
- [45] W. L. Lynch, "The Interaction of Virtual Memory and Cache Memory," Tech. Rep. CSL-TR-93-587, Stanford University, 1993.
- [46] ARM, "Migrating a Software Application from ARMv5 to ARMv7-A/R Application Note 425." <http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dai0425/index.html>.
- [47] A. Agarwal, J. Hennessy, and M. Horowitz, "Cache Performance of Operating System and Multiprogramming Workloads," *ACM Trans. Comput. Syst.*, vol. 6, pp. 393–431, Nov. 1988.
- [48] M. Cekleov and M. Dubois, "Virtual-Address Caches, Part 2: Multiprocessor Issues," *IEEE Micro*, vol. 17, pp. 69–74, Nov. 1997.
- [49] B. Jacob, S. Ng, and D. Wang, *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann Publishers Inc., 2007.
- [50] J. S. Chase, H. M. Levy, M. J. Feeley, and E. D. Lazowska, "Sharing and Protection in a Single-address-space Operating System," *ACM Trans. Comput. Syst.*, vol. 12, pp. 271–307, Nov. 1994.
- [51] J. S. Chase, H. M. Levy, E. D. Lazowska, and M. Baker-Harvey, "Lightweight Shared Objects in a 64-bit Operating System," in *Conference Proceedings on Object-oriented Programming Systems, Languages, and Applications (OOPSLA)*, pp. 397–413, 1992.
- [52] E. J. Koldinger, J. S. Chase, and S. 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)*, pp. 175–186, 1992.
- [53] A. Sembrant, E. Hagersten, and D. 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 Architecture (ISCA)*, pp. 133–144, 2014.
- [54] A. Sembrant, E. Hagersten, and D. Black-Schaffer, "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)*, pp. 49–61, 2013.
- [55] D. H. Woo, M. Ghosh, E. Özer, S. Biles, and H.-H. 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)*, pp. 179–189, 2006.