VIRTUAL-ADDRESS CACHES

Part 1: Problems and Solutions in Uniprocessors

Michel Cekleov

Sun Microsystems

Michel Dubois

University of Southern California

To solve the virtual-to-physical address bottleneck, processors may access caches directly with virtual addresses. This survey introduces the problems and discusses solutions in the context of single-processor systems.

All modern general-purpose computer systems need a cache memory to reduce the average latency of memory accesses. This small, high-speed memory between the processor and the main memory keeps the information (instructions and data) currently in use.[1,2] In a system with virtual memory,[3] the processor issues virtual memory addresses, which are dynamically translated into physical addresses. Traditionally, processors have accessed the cache in a computer system with physical addresses. Although special-purpose hardware generally supports the virtual-to-physical address translation, it tends to increase the memory access time. So to remove this performance bottleneck, address translation and cache access must occur concurrently—which either complicates and slows the processor pipeline or limits the cache size.

The alternative is to access the cache directly with virtual addresses, before the virtual-to-physical address translation. We call caches accessed with virtual addresses virtual-address caches or simply virtual caches, as opposed to physical-address caches or physical caches. The tags in the directory of a virtual cache may be virtual or physical. In the past, designers have avoided virtual caches because they may not be totally transparent to the software even in uniprocessors.

Consistency problems occur within the same virtual cache whenever a virtual-to-physical mapping is changed or when different virtual addresses map to the same physical address. These problems seem even more intractable in multiprocessors because inconsistencies can occur in more than one cache.

Nevertheless, driven by technology trade-offs, more and more processor designers are taking advantage of the attractive features of virtual caches. First and foremost, accesses to data and instructions are satisfied in one cache cycle, and the cache size can be very large without undue complexity. Second, when the tags are virtual, virtual-to-physical address translations are needed only on a cache miss and are removed from the processor's critical path.

This survey exposes the problems related to virtual caches in the context of uniprocessor (Part 1, this issue) and multiprocessor (Part 2, *Micro*'s Nov.-Dec. issue) systems. We review proposed solutions that have been implemented or proposed in different contexts. The idea is to catalog all solutions, past and present, and to identify technology trends and attractive future approaches. We first overview the relevant properties of virtual memory and of physical caches. (All references appear in Part 2.)

Virtual memory

In a typical virtual memory system, each program is compiled in a virtual space, which is dynamically mapped onto the physical memory of the machine at runtime. Thus all processes have separate virtual spaces. A process context may host multiple execution threads, which can be executed concurrently on different processors. In this case, the threads share the resources of the context, including its virtual memory space.

Paging. At the lowest level, virtual memory is divided into equal chunks of consecutive memory locations called virtual pages (or simply, pages). Pages are dynamically mapped onto physical pages (or page frames) in main memory through a set of translation tables called page tables. Pages are brought into page frames on demand (demand-paging) as processes need them. An access to a page not resident in memory triggers a page fault, which the processor treats as an exception. A software page fault handler swaps the missing page in memory and validates the new virtual-to-physical

0272-1732/97/$10.00 © 1997 IEEE

Glossary

Cache indexing: First phase of the cache access in which the least significant bits of the block address are used to select and fetch the cache directory entries in the accessed set

Page: Unit of memory allocation in a virtual memory system

Page table: Dictionary of virtual-to-physical address translations accessed with the virtual page number and yielding the physical page number

Virtual address: Processor-computed address used to access the page tables

Physical address: Address in physical memory obtained after translation of the virtual address in the page tables

P/P cache: Physical cache indexed and tagged with bits from the physical address

V/P cache: Cache indexed with virtual-address bits but tagged with physical bits

P/V cache: Cache indexed with physical bits but tagged with virtual bits

V/V cache: Cache indexed and tagged with virtual bits

Physical cache: P/P cache

Virtual cache: V/P, P/V, or V/V cache

Superset bits: Bits used to index the cache and which are part of the page number

Page color: Number defined by the superset bits (either physical or virtual)

Victim block: Block selected for replacement on a cache miss

Cache coherence: Property of a shared-memory system that all cached copies of the same memory block contain the latest value of the block

Snooping: Monitoring the system bus in hardware to enforce cache coherence

Dual directory: Cache directory through which the cache can be accessed from the bus

TLB (translation look-aside buffer): Address cache storing the page table entry to speed up dynamic address translation

TLB consistency: Property of a shared-memory system that all copies of the same page table entry in different TLBs are identical

TLB shoot down: Algorithm to maintain TLB consistency

Synonyms: Several virtual addresses pointing to the same physical location

Homonyms: Several physical addresses having the same virtual address

Synonym alignment: Property of a system that all synonyms of the same page have the same superset bits or virtual color

V-P alignment: Property of a system that virtual and physical pages must have the same color

Page-mapping change: Page demapping or remapping

Page demapping: Action taken by the kernel to remove a virtual-to-physical translation

Page remapping: Action taken by the kernel to allocate a physical page to a virtual page

Aliases: Synonyms due to page-mapping changes

Paired eviction: The removal of two blocks on a cache miss in a virtual cache with dual directory

Cache purge: Invalidation of part of a cache without updating memory

Cache flush: Invalidation of part of a cache with a memory update

address translation.[3]

Besides its primary role of transparent memory management, the virtual-address translation mechanism conveniently supports protection because it is in the required path of all memory accesses. Protection enforces access rights of processes to information, based on the processes' privilege level. Most architectures support only two privilege levels: supervisor and user modes.

Figure 1 illustrates the address translation process. The processor uses the virtual page number to fetch an entry in the page table, which is stored in memory. Note that there are many possible organizations for the page table. The representation in Figure 1 is conceptually closer to so-called hierarchical page tables,[4] in which each context has its own separate page table and which is accessed through a cascade of pointer tables.

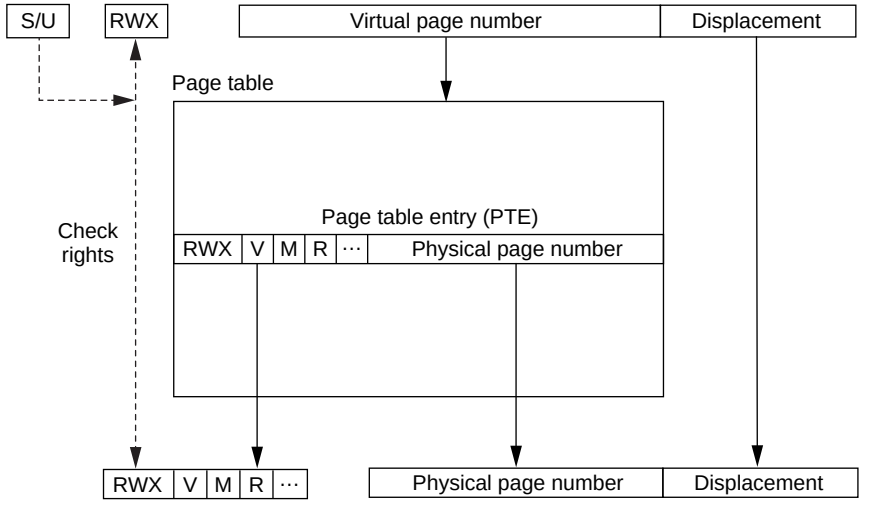


Figure 1. Virtual-to-physical address translation.

Another organization is the inverted page table,[5] which is common to all processes and is accessed by hashing the vir-

p_id	Virtual page number	Physical page number	V	R	M	RWX	Misc.

Figure 2. Entry of a direct-mapped TLB.

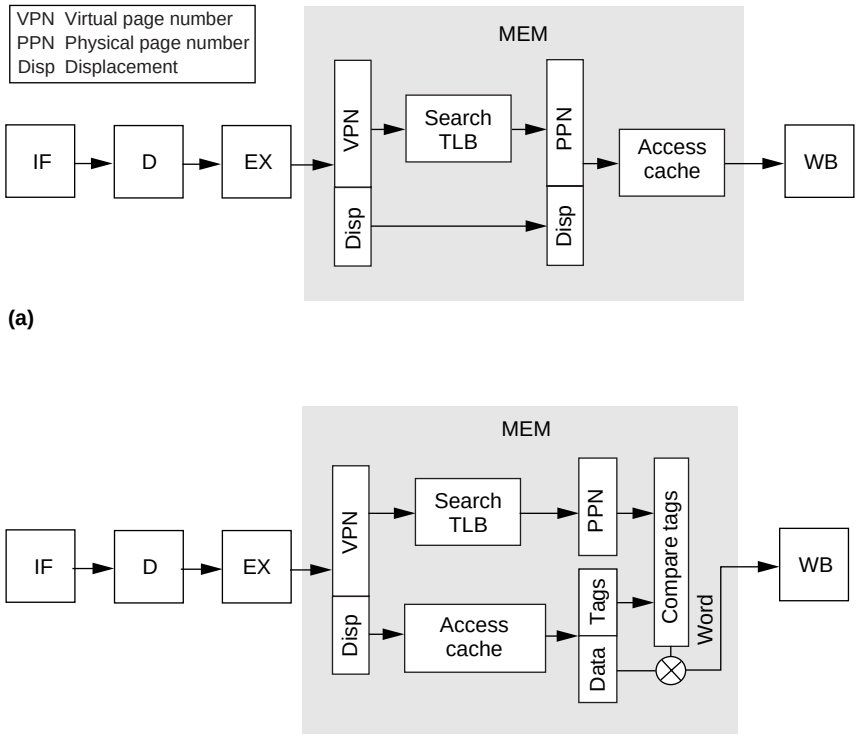


Figure 3. Concurrent access to cache and TLB (physical caches): pipelining TLB and cache accesses (a) and accessing TLB and cache in parallel (b).

tual address. In any case, the physical page number retrieved from the table plus the page displacement bits form the physical address. (Note that the page displacement bits are not translated.)

Typically, the page table entry contains several bits to control accesses to the physical page:

- The RWX (Read, Write, Execute) bits check access rights by matching them against the type of request (RWX) submitted by the processor at the privilege level indicated by the S/U (supervisor/user) bit.
- V (Valid) bit indicates whether the page is valid in main memory (page fault detection).
- M (Modify) bit indicates whether the page has been modified since it was swapped in. It is set at the first modification to the page.
- The kernel uses the R (Reference) bit to implement a page replacement algorithm. The kernel periodically clears this bit. The bit is set at the first reference to the page.

Translation look-aside buffer. To speed up virtual address translation, the system stores current address translations in the translation look-aside buffer (TLB), a separate cache. Figure 2 shows an entry of a direct-mapped TLB. In a typical access, the hardware fetches the TLB entry at the address given by the least significant bits of the virtual page number, compares the process identifier (p_id)/virtual page number field to the current p_id and to the processor's virtual address, and checks the V bit. On a hit, the physical page number is returned, and the other bit fields of the TLB entry are checked. The TLB may also be either set associative or fully associative on the p_id/virtual page number.

Since most accesses bypass the page table through the TLB, access-right bits must be copied in the TLB entry. Additionally, a TLB entry may contain copies of the R and M bits obtained from the page table entry at the TLB miss. In this case, an access to a page with the R bit reset in the TLB or a modification of a page with the M bit reset must trigger an update of the bit in the page table in main memory. Copies of the R and M bits are not necessary in the TLB because their function can be emulated by trapping on other bits such as the access-right bits.[6]

Often, software manages the TLB by trapping the processor. In some cases, special microcode and hardware exist in the TLB to reload entries from the page tables or to update the state bits. We call these units either table-walking TLBs or MMUs (memory management units).[7]

Synonyms. Although the virtual spaces of different contexts are disjoint, processes need to share information. The most common case is when a process creates another one; then, usually, the parent and the child processes share the same text segment. Another case is the sharing of data memory. To enable sharing, addresses in different contexts must be used to access the same page. When different virtual addresses map to the same physical address, we say that they are synonyms. A programmer may also define synonyms in the same context, for convenience. As shown later, synonyms are at the root of many problems in virtual-address caches.

Physical caches in uniprocessors

In practice, all caches are either direct-mapped or set-associative[1] and have two parts: directory and data memory. The cache directory contains the physical addresses (tags) and the state bits. The cache data memory contains the data blocks. The least significant bits of the block address provide the set number. On each processor access, a set's directory and data memory entries are first fetched in parallel (cache

indexing). Then, the tags of the blocks in the set are compared in parallel with the tag field of the processor address. A cache access therefore proceeds in two phases: cache indexing followed by tag comparison.

By definition, a physical cache is only accessed with bits of the physical address. The simplest solution is to access the TLB first and then the cache. Then each data access and possibly each instruction access takes one additional cycle. To avoid this overhead, the cache and TLB must be accessed concurrently either by pipelining the accesses or by performing the two accesses in parallel. Figure 3 illustrates these solutions for a simple RISC pipeline with five stages: instruction fetch (IF), instruction decode (D), instruction execute (EX), memory access (MEM), and write back (WB).

When the TLB and cache are accessed in two consecutive instruction pipeline stages, the additional stage increases the operation latency of loads[8] and elongates the pipeline. This produces a higher penalty for mispredicted branches. For these reasons, designers often favor parallel access over pipelined access.

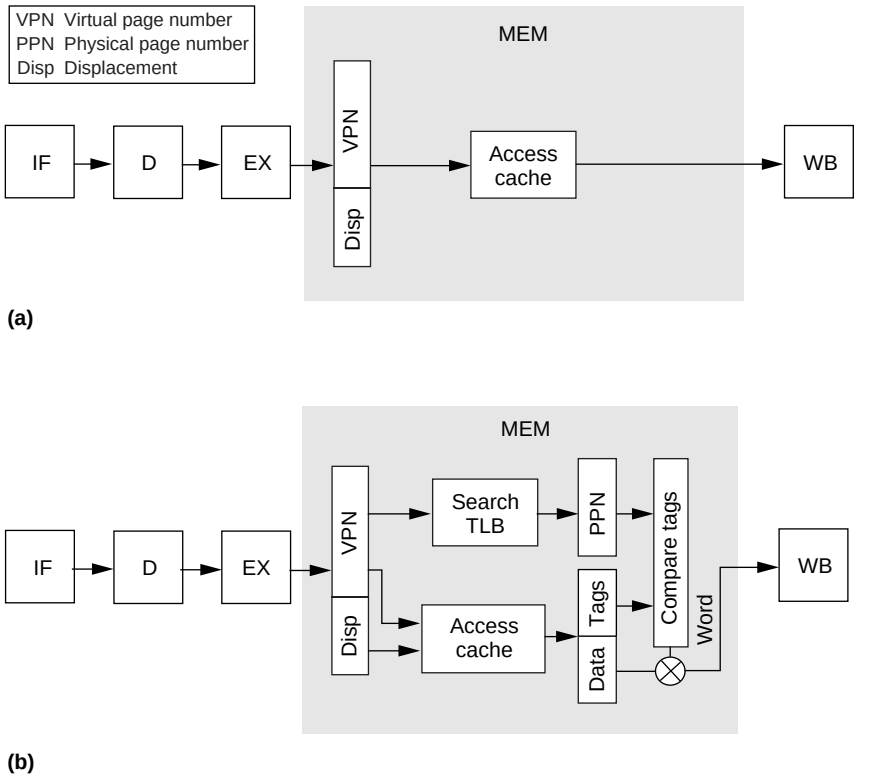


Figure 4. Virtual-address cache configurations: pipelining the TLB and cache accesses (a); accessing the TLB and cache in parallel (b).

The parallel access to the TLB and cache is possible provided the bits selecting the set are within the page displacement field. At the end of the first (cache indexing) phase, the physical address from the TLB and the cache tags are available for comparison. A miss occurs if no tag in the set matches or if the TLB misses. A rapid calculation shows that the restriction on the field selecting the set limits the possible cache sizes to one page per way of set associativity. For example, the maximum size for a direct-mapped cache is the size of one page.

Virtual caches in uniprocessors

Because of the cache size limitations imposed by physical caches, the first-level cache of some high-performance processors is indexed with virtual addresses. The tags in the cache directory may be virtual or physical. If the tags are virtual, the TLB is only accessed on a cache miss. If the tags are physical, the TLB and the cache are accessed in parallel. We call the first organization a virtually indexed, virtually tagged (V/V) cache, and the second one a virtually indexed, physically tagged (V/P) cache. Thus we also call a P/P cache a physical cache. P/V caches are indexed with physical addresses and are virtually tagged.

Processor accesses to a V/V cache are very simple since no translation is required as long as the cache hits (see Figure 4a). The V/P cache organization of Figure 4b looks very similar to the physical cache of Figure 3b. The only difference is that the most significant bits of the field indexing the V/P cache are part of the virtual page number. These bits, called the superset bits, generally differ from the corresponding bits in the physical address. By generating all the possible values for the superset bits, we obtain the superset containing the accessed set. Thus, if we need s superset bits to index the cache, the superset contains 2^s cache sets. All synonyms as well as the physical address always map into the superset. Note that when the number of superset bits is zero, a V/P cache is indistinguishable from a P/P cache.

The only example we know of a P/V cache is the cache of the Mips R6000.[9] In this design a small memory called the TLB slice contains the physical superset bits of recently accessed pages. The TLB slice is accessed with the least significant bits of the virtual page number before indexing the cache. P/V caches have properties similar to those of V/V caches.

In general, consistency problems in virtual-address caches are due to synonyms and to address-mapping changes.

Synonym problem. Unless synonyms are checked, multiple copies of the same block could end up in the virtual cache. In the case of read-only blocks, all the copies are identical and the multiple copies pollute the cache. In the case of read/write blocks, the processor may access a stale copy.

For example, say variable *X* at address p_3 is writable and is read successively with virtual address v_2 and then with virtual address v_3. Then two incoherent copies are present in the cache after the processor modifies the copy with vir-

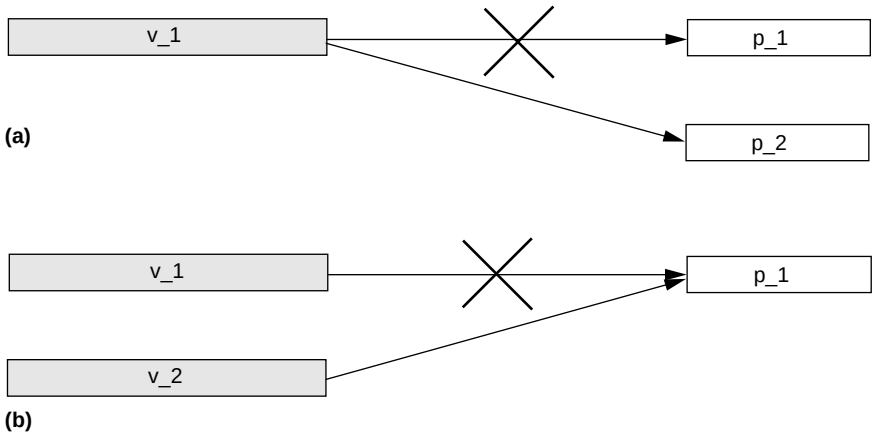


Figure 5. Virtual-to-physical mapping changes. Physical address change: v_1/p_1 is broken, and v_1 is remapped to p_2 (a); Virtual address change: v_1/p_1 is broken, and v_2 is remapped to p_1 (b).

tual address v_3. It is impossible to keep track of the latest copy, unless the copies are time stamped; even then, searching for the latest copy in the whole superset on each load is out of the question. Therefore multiple, modifiable copies of a block must never coexist in the cache under different addresses. To enforce this condition at all times is the synonym problem.

Virtual-to-physical address-mapping changes. At times, the processor may need to break the mapping between a page and its page frame and then remap the page to a new page frame as well as the page frame to a new page. In a P/P cache, invalidating the TLB entry efficiently handles page demapping by preventing the processor from accessing any cache block with that translation. When the page frame is remapped to a new page, the blocks in the page frame become accessible again with the new virtual address. Remapping takes place after it is safe to do so and after a new TLB entry has been allocated for that page frame. Thus demapping and remapping areas of virtual memory are very simple in a physical cache. This is not the case for many virtual caches.

Consider the scenario illustrated in Figure 5. Virtual address v_1 is first mapped to physical address p_1. Then, the operating system decides to demap v_1 from p_1 and later remaps v_1 to p_2 (see Figure 5a). Since the same virtual address points to two different physical addresses, we call these translations homonyms. New virtual address v_2 may then be mapped to p_1 (see Figure 5b). In the following, we refer to synonyms due to mapping changes—such as v_1 and v_2—as aliases. Because of homonyms and aliases, invalidating the TLB on a page demap is not sufficient.

Mapping changes occur constantly in a system where different programs run concurrently. The most obvious cause of mapping changes is the deallocation and reallocation of page frames; in this case, the memory management software dynamically creates aliases. Mapping changes are also an efficient mechanism for communicating safely among processes without copying.

Aligning virtual and physical addresses. Two synonyms cannot exist within the same page because displacement bits are not translated. Thus two synonyms must share the same least significant d bits, where 2^d is the page size. We can further restrict synonyms so that synonyms are not permitted within a certain range of virtual addresses. Let s be the number of superset bits for a given cache organization and size. If all synonyms share the same least significant $s+d$ bits, they all map to the same cache set and are said to be aligned.

A stricter address-mapping constraint is V-P alignment.[10] Pages are V-P aligned if the physical address and the virtual addresses of the same page index the same set of the cache. This means that their superset bits are identical. When the physical and virtual addresses are V-P aligned, a V/P cache behaves like a physical cache.[11]

The superset bits in a virtual or a physical address define the page's virtual or physical color. With V-P alignment, all synonyms are automatically aligned—that is, they have the same virtual color. Additionally, the page's physical color is the same as the virtual color, a memory allocation strategy called page coloring.[12]

Page coloring imposes constraints on memory allocation. When a new physical page is allocated on a page fault, the memory management algorithm must pick a page with the same color as the virtual color from the free list. Because systems allocate virtual space systematically, the pages of different programs tend to have the same colors, and thus some physical colors may be more frequent than others. Thus page coloring may impact the page fault rate. Moreover, the predominance of some physical colors may create mapping conflicts between programs in a second-level cache accessed with physical addresses.

Solutions to the synonym problem

To solve the synonym problem, we can consider two approaches: having the software prevent or avoid synonyms—in which case no hardware support is needed for synonyms—or tolerating synonyms and having the hardware detect them dynamically.

Synonym prevention. By eliminating the need for synonyms, we prevent the problems they cause. The simplest way to do this is to implement sharing at the segment level.[13] Typically, a processor address contains several bits pointing to a segment register, which contains a segment's identifier. The virtual address is then built by concatenating the segment identifier with the rest of the processor address for use in accessing the page table. Software manages the segment registers, and processors can share a segment by simply loading the segment identifier in one of their segment registers. Thus a page has only one systemwide virtual number, and all protection problems can be resolved at the segment level.

The prospect of single-address-space operating systems (SASOSs) offers another opportunity to eliminate the need for synonyms.[14] In these operating systems, all processes share a single, global virtual address space, taking advantage of the advent of 64-bit microprocessors. Because the virtual space is shared, different processes sharing information use the same virtual address, and synonyms are useless.

In both cases, a single virtual space common to all processes is created and is accessed through a shared page table. Because the virtual space is so large, the page tables are often inverted and accessed by hashing the virtual address.[5]

Synonym avoidance. When the software (kernel or user) exploits synonyms, we can still avoid the synonym problem without hardware support. In this case, the software invalidates all or part of the cache when needed so that synonyms are never present in the cache at the same time. These cache invalidations are either purges or flushes. A cache purge is a simple invalidation of cache blocks with no memory update. A cache flush is an invalidation of cache blocks with write-back of dirty blocks to memory and is often required in virtual write-back caches. A cache flush is a costly operation, which typically takes place one block at a time. To simplify the wording, we say "to purge or flush the cache" instead of "to purge or flush a block/page/segment in the cache," provided the unit to flush or purge is obvious from the context.

There are two systematic synonym avoidance strategies in uniprocessors. One relies on the page table and page-faulting mechanism, and the other relies on context switching. The page allocation software may enforce a rule that only one virtual page can map to a physical page at a time in the page tables. When a different virtual name is used for the same page, a false page fault is triggered, the previous translation demapped, and the new translation loaded in the page table. Unfortunately, demapping and remapping of translations entail costly cache purges or flushes, as we will see. In some machines, such as the Intel i860,[15] the entire content of the V/V cache is flushed on every context switch; additionally, synonyms are disallowed within the same context. This solution is practical only if the cache is small and the frequency of context switches is low.

Other synonym avoidance solutions are ad hoc approaches and can only alleviate the problem without really solving it. For example, the kernel could tag all pages known under several virtual addresses as noncacheable, unless they are read only. This solution is viable only if the use of synonyms is very limited, because accesses to noncacheable pages are usually very slow. If access rights change dynamically and a read-only page becomes writable or if a writable page becomes accessible under a different synonym, the virtual cache must be purged or flushed because the page must become noncacheable. Another ad hoc solution is to flush the cache to guarantee data consistency when the access pattern to the synonyms is totally predictable. Whenever a different synonym may be used, the cache is purged or flushed, as is done in some implementations of the SunOS kernel for some I/O operations.[16] This solution has a large overhead and applies only to the kernel.

Dynamic synonym detection—basic approach. Hardware support can relieve the software of all or most of the task of maintaining consistency in the presence of synonyms. These solutions enforce the rule that only one copy of a block is present in a cache at any one time, even if the block is read only. To achieve this, the hardware removes synonyms from the cache on a miss. The basic approach does not rely on any special-purpose hardware.

At the time of a cache miss, the cache controller must search for a synonym in every set in the superset (including the indexed set). The controller visits the sets one by one until it finds a synonym or until it has visited all the sets. In a V/P cache this search is efficient. It exploits the hardware available for normal cache accesses: all the physical tags in one set are matched against the TLB entry in one cache cycle. In contrast, in a V/V cache every virtual tag in each set of the superset must be translated one by one in the TLB, and the tag's physical address must be compared to the physical address of the processor. If, during this search, the controller finds the block in the cache under a synonym, we say that the cache miss is a short miss. The controller simply retags and/or moves the block within the cache, and aborts the memory fetch.

The operating system kernel should strive to align as many synonyms as possible to lessen the miss penalty. In fact, an access to a block present in a V/P cache under a different (but aligned) synonym always hits, and the presence of a synonym is transparent to the hardware. In a V/V cache, when the controller finds an aligned synonym in the indexed set, it simply retags the block. In the ideal case, when all synonyms are constrained to be aligned, the miss penalty is further cut, as the search for synonyms is limited to the indexed set. In particular, in a V/P cache, the hardware can simply ignore synonyms altogether when they are all aligned.

Dynamic synonym detection—reverse maps. The basic approach to detect synonyms may be very slow, especially when some synonyms are not aligned and the superset and the cache are large. In practical terms, the size of the cache is still limited because the search through the superset must be completed before the memory block is retrieved. Thus, researchers have proposed many ways to find synonyms fast, independently of the cache size. They all rely on a reverse map indexed with physical addresses.

One solution is based on a copy of the main cache directory called the dual directory and accessed with physical addresses. Ideally, we can imagine a fully associative dual directory with as many entries as there are block frames in the virtual cache. Each entry of the dual directory contains a backpointer to the block frame in the cache where the block is stored. On a miss in the cache, the dual directory is accessed with the physical address obtained from the TLB. A valid backpointer to the main cache points to the location of a synonym. Overall this solution is very expensive in uniprocessor systems, but may make sense in bus-based multiprocessors with snooping protocols,[17] where a dual directory is present to maintain coherence. We will therefore examine this solution more closely in the context of multiprocessors.

A second organization for the reverse map is an unaligned-synonym cache. We have seen that the miss penalty is reduced or even eliminated when all synonyms are aligned. However, it may not be possible nor desirable to align all

synonyms. A cost-effective solution is to keep a pointer to every unaligned synonym residing in the main cache in a small unaligned-synonym cache. The unaligned-synonym cache is accessed with the physical address obtained from the TLB on a miss in the main cache, and its entries contain a backpointer to the main cache.

The major problem is detecting when to load a new entry in the unaligned-synonym cache. One solution is to include in the TLB entry one state bit indicating whether unaligned synonyms exist for the page. Another solution[10] consists of V-P aligning most pages (by page coloring) and loading an entry in the unaligned-synonym cache (called the U cache) for every cached block that is not V-P aligned. (We can check this by comparing the colors of the virtual and physical pages.) Simulations have shown that an eight-entry U cache may be as good as a full dual directory, provided more than 90% of all pages are V-P aligned. The U cache is a very simple and efficient addition to a direct-mapped V-V cache or to a set-associative V/P cache when all synonyms cannot be aligned and when page coloring is imperfect.

Finally, in systems with second-level caches, the reverse map can be implemented in the directory of the second-level cache. In this case, the first-level (on-chip) cache may be virtual (for speed), whereas the second-level cache is physical.[18] The inclusion property must be maintained. If a block is present and valid in the first-level cache, it is also present and valid in the second-level cache.[19] On a miss in the first-level cache, the physical address is obtained from the TLB. Each entry of the second-level cache may contain a valid backpointer to an entry in the virtual cache. Mips processors[20] support this approach. To support unaligned synonyms in the first-level V/P cache, the second-level cache is physical. In addition, each tag contains the virtual color of the block in the first-level cache, which acts as the backpointer. The virtual coherency exception is raised whenever a first-level cache miss hits in the second-level cache and the virtual color in the second-level cache does not match the color of the virtual address. In response, the processor removes the block in the first-level cache using the virtual color from the second-level cache and updates the virtual color in the second-level cache.

Page-mapping changes

Since these changes create homonyms and aliases in virtual caches, we need to clarify the consistency issues and the possible solutions.

V/P caches. In a V/P cache, homonyms created by mapping changes are not a problem since the TLB is checked on every access. When pages are not V-P aligned, the hardware must take care of unaligned aliases by dynamic synonym detection. Consider the example of Figure 5b. If v_1 and v_2 index in different sets, stale copies of some blocks of p_1 may exist. These stale copies may overwrite page frame p_1 on replacement when they are written back later. Moreover, if p_1 is eventually remapped to a page indexing in the same set as v_1, accesses to p_1 could hit on a stale copy.

Say that no hardware support exists for unaligned synonyms and pages are not V-P aligned. Then the software must flush (write-back cache) or purge (write-through cache) the V/P cache when addresses are demapped. In some cases, the cache flush or purge may be delayed until the page remapping, at which time they may become unnecessary.[21,22]

Clearly, synonym prevention per se does not eliminate purges or flushes due to mapping changes in V/P caches, unless pages are also V-P aligned.

V/V caches. In a V/V cache, aliases can be treated as other synonyms, as well: either the hardware detects unaligned synonyms or it detects aligned synonyms only and pages are V-P aligned. Synonym prevention per se does not eliminate purges or flushes due to mapping changes in this cache either.

Homonyms create a unique problem in V/V caches. In Figure 5a, the old memory block at physical address p_1 may remain accessible in the cache under virtual address v_1 after the remapping of v_1 to p_2. Synonym detection hardware or V-P alignment cannot alleviate this consistency problem. All blocks in the cache known under the old mapping must be purged (write-through cache) or flushed (write-back cache) at the demapping or at the remapping of v_1.

Problems specific to V/V caches

The most attractive feature of a V/V cache is that it works well with a slow TLB, allocated outside the processor chip.[18] Such a TLB can be huge and have a very high hit rate without slowing down the processor. However, this advantage is the source of additional problems, which are specific to V/V caches. First, access-right bits must migrate to the cache because the TLB is bypassed most of the time. Furthermore the cache controller must access the TLB to check for synonyms, to update the M bit, or to propagate buffered writes or write-backs.

Access-right bits and page state bits. In a V/V cache, the cache directory must hold a copy of the access-right field found in the TLB entry to support protection. Consistency between these copies must be enforced by flushing the cache if the access right to a page by a process is dynamically restricted.

Generally, access-right bits can emulate the functions of the R and M bits in the TLB and in the cache, thus simplifying the hardware. Even when these bits are copied in the TLB, they need not be present in the cache directory. The R bit in the TLB can be set at the time of a miss for a block in the page, with some loss in precision. In a write-through cache, the TLB is accessed on each processor write, and the M bit in the TLB is managed as in a physical cache. In a write-back cache, each cache entry has a dirty bit. Whenever the processor modifies a block in the page for the first time, as indicated by the dirty bit, the M bit of the TLB entry must be set.

Special accesses to the TLB. Virtual-tag translation in the TLB is a basic mechanism to detect synonyms. Moreover, in systems with write-through V/V caches, the stores are often buffered in a write buffer, and their virtual addresses must eventually be translated. In systems with write-back V/V caches, a dirty block selected for replacement is buffered locally while the missing block is fetched first on a cache miss. Later on, the physical address of the block to write back must be found in the TLB.

These special accesses to the TLB may create TLB misses if cache entries are allowed to exist without valid translation in

Table 1. Hardware support for synonyms and mapping changes.

Software restrictions	Synonyms Cache types P/P	V/P	P/V	V/V	Mapping change Cache types P/P	V/P	P/V	V/V
No synonym + V-P alignment	Ignore	Ignore	Ignore	Ignore	Ignore	Ignore	Flush	Flush
No synonym		Ignore	Ignore	Ignore	Ignore	Flush or detect	Flush	Flush
V-P alignment	Ignore	Ignore	Flush or detect	Flush or detect	Ignore	Ignore	Flush	Flush
Aligned synonyms	Ignore	Ignore	Flush or detect	Flush or detect	Ignore	Flush or detect	Flush	Flush
Unaligned synonyms	Ignore	Flush or detect	Flush or detect	Flush or detect	Ignore	Flush or detect	Flush	Flush

the TLB. Because this type of TLB miss is asynchronous to the processor's activity, the hardware accessing the TLB must be designed carefully and may have to handle multiple TLB misses concurrently. Alternatively, inclusion may be maintained between the TLB and the cache. Whenever a translation is invalidated or displaced by the replacement algorithm in the TLB, the cache is purged or flushed. The TLB hit ratio must be very high to limit the performance degradations. The Sun 3 and the first-generation Sun 4 architectures[4,23] with write-back caches adopted this solution.

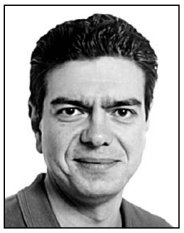
Hardware support

Table 1 summarizes the hardware support needed in the cache for synonyms under several software restrictions on synonyms and page mappings. The table also shows the support needed for mapping changes. "Ignore" means that no hardware support is needed, and "detect" that the hardware must dynamically detect synonyms (or aliases). "Flush" means that hardware support is needed to flush (write-back) or purge (write-through) the cache.

P/P caches and V/P caches with V-P page alignment can ignore synonyms and mapping changes. P/V and V/V caches need the same level of hardware support because V-P alignment per se does not help V/V caches. Even when synonyms are prevented, caches with virtual tags must be flushed at times to avoid problems with homonyms and aliases. A V/P cache may need extra flushes as well to process aliases when pages are not V-P aligned and the hardware does not detect unaligned synonyms. (In some cases such flushes may be avoided in software.) Neither synonym alignment nor V-P alignment of pages completely solves the synonym problem for caches with virtual tags.

WE RESTRICTED THIS FIRST PART of our two-part survey on virtual caches to uniprocessor systems. In part 2 in the next issue of *IEEE Micro*, we will address multiprocessor issues.

In multiprocessors, cache coherence and TLBs must be maintained. Whereas this requirement adds to system complexity, virtual caches can take advantage of the mechanisms needed to enforce coherence. Therefore some solutions will make more sense than others in the context of multiprocessors.



Michel Cekleov is a system architect in the Sun Microsystems Desktop Systems Group, where he works on the next-generation high-performance workstations. Earlier, he was one of the architects of the company's Sparccenter 2000 and Sparcserver 1000 multiprocessor servers.

Cekleov received his engineering degree from Ecole Supérieure d'Ingénieurs de Marseille and his doctorate from Ecole Nationale Supérieure des Télécommunications.

Michel Dubois is a professor in the Department of Electrical Engineering at the University of Southern California. His main interests are computer architecture and parallel processing.

Dubois holds a PhD from Purdue University, an MS from the University of Minnesota, and an engineering degree from the Faculte Polytechnique de Mons in Belgium, all in electrical engineering. He is a member of the ACM and a senior member of the IEEE Computer Society.

Direct questions regarding this article to Michel Dubois, University of Southern California, School of Engineering, EE Systems, Los Angeles, CA 90089-2562; dubois@paris.usc.edu.