The Synonym Lookaside Buffer: A Solution to the Synonym Problem in Virtual Caches

Xiaogang Qiu and Michel Dubois, *Fellow*, *IEEE*

Abstract—To support dynamic address translation in today's microprocessors, the first-level cache is accessed in parallel with a translation lookaside buffer (TLB). However, this current approach faces mounting problems as more concurrency is exploited in the processor core, as multiprocessors are becoming more and more prevalent, and as the memory demand of emerging applications is growing. This paper introduces new ideas to enable the use of virtual addresses in the cache hierarchy, thus removing the TLB from the critical path of the processor core. The major idea is the replacement of the on-chip TLB by a synonym lookaside buffer (SLB). The SLB translates synonyms into a primary virtual address, which is a unique identifier resolving all ambiguities due to synonyms in the memory system. We introduce various system configurations with SLBs and discuss all functional issues associated with them. An SLB is much more scalable than a regular TLB. It scales with memory data set sizes, physical memory sizes, and number of cores in a multiprocessor. Moreover, SLB entry flushes and shootdowns due to physical memory management are eliminated. We show performance data resulting from the simulation of several applications as diverse as scientific computing, database, and JAVA virtual machines. These evaluations target SLB miss rates and flushes as well as the impact of the SLB on cache miss rates. They show that small SLBs of 8-16 entries are sufficient to solve the synonym problem in virtual caches and that their performance overhead is negligible.

Index Terms—Microprocessor, virtual memory, cache, virtual address cache, synonyms, aliasing, translation lookaside buffer, multicore.

1 Introduction

The execution rate of modern microprocessors improves relentlessly due to faster clock rates, deeper pipelines, instruction-level parallelism (ILP), and thread-level parallelism (TLP). To match this computing power, the memory hierarchy must satisfy multiple memory accesses in every clock cycle in the face of a growing gap between processor execution rate and main memory access time, and of the growing memory demand of emerging applications. Virtual address translation both for instructions and data is an additional hurdle. In the process of accessing memory, the virtual (effective) address issued by the processor must be translated at some point into a physical address to access main (physical) memory.

This dynamic translation is currently supported by a translation lookaside buffer (TLB), a cache for virtual-to-physical address translations. Typically, the TLB and the first-level cache are accessed in parallel. This access is done in two steps. In the first step, the virtual page number is translated in the TLB and the tags and data of the first-level cache set are fetched (cache set indexing). Then, the physical page number obtained from the TLB is compared with the tags in the set to detect a hit. When the virtual address bits used to index the cache set are part of the page displacement and thus are physical address bits as well, the first-level cache is called "physical" and the memory hierarchy is accessed with physical addresses throughout. However, when the first-level cache size is too large, some bits from the virtual page number must be used to index the cache set, although the cache tags are still physical. These caches are called "virtual" to reflect the fact that bits from the virtual address are used to access them [4].

In these caches, the TLB is a hardware bottleneck because it must be accessed in parallel with time-critical accesses to the first-level cache integrated very closely with the processor core where the chip real-estate is very precious. This is the case for high-end out-of-order issue processors [14] as well as for multithreaded cores [18]. Because of this, the TLB size is usually small (64-512 entries) and translation misses trigger expensive "walks" through levels of page tables located in the main memory. On the other hand, technology trends and the growing working set demands of applications put more and more pressure on the address translation hardware [27], [29]. Moreover, the TLB, which is built inside the microprocessor chip, does not scale well with various main memory sizes and with the number of processing nodes in multiprocessor configurations.

Virtual caches may also be tagged with virtual address bits. When the first-level cache is virtually indexed and tagged, most memory accesses are completed without TLB involvement. The actual address translation is performed only when needed and can be done at various locations in the memory hierarchy [4], [15], [32], where its access time and bandwidth are no longer critical. Moving the TLB after the virtual cache hierarchy can dramatically reduce the number of TLB misses because most memory accesses do not reach the TLB. Moreover, when TLBs are shared among

- *X. Qiu is with NVIDIA Corporation, 2701 San Tomas Expressway, Santa Clara, CA 95050. E-mail: xqiu@nvidia.com.*
- *M. Dubois is with the Department of Electrical Engineering, University of Southern California, Los Angeles, CA 90089-2562. E-mail: dubois@paris.usc.edu.*

Manuscript received 15 Aug. 2006; revised 10 June 2008; accepted 24 June 2008; published online 16 July 2008.
Recommended for acceptance by A. Gonzalez.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0318-0806.
Digital Object Identifier no. 10.1109/TC.2008.108.

0018-9340/08/$25.00 © 2008 IEEE Published by the IEEE Computer Society

processors in a multiprocessor system such as a Chip Multiprocessor (CMP), TLB misses can become insignificant because of sharing and prefetching effects [26]. In [26], we showed that by moving the TLB down the hierarchy huge savings in TLB misses and large execution time reductions are achievable, even if the TLB is removed altogether and all translations are performed in page tables through processor exceptions or with table walking hardware.

Unfortunately, reaping the benefits of a virtual cache hierarchy is not easy because virtual caches are plagued by the synonym (or alias) problem: multiple virtual addresses may map to the same physical address. This problem complicates cache management and cache coherence because of the lack of a unique systemwide identifier for each memory location. Complex, anti-aliasing hardware that detects synonyms dynamically must be implemented. We propose a simple and efficient extension to a system with virtual address caches to expose synonyms dynamically throughout the memory hierarchy of a microprocessor.

This extension is called a synonym lookaside buffer (SLB). It is a translation buffer organized as a traditional TLB but whose role is to translate synonyms into a unique virtual address called the *primary virtual address*. The primary virtual address is a virtual address selected in each set of synonyms sharing the same page. It plays the role of a unique, systemwide identifier for the set of synonyms. The systemwide existence of such an identifier solves intra- and intercache coherence problems due to naming. Synonyms of primary virtual addresses are called *secondary virtual addresses*. An SLB is attached to each core and dynamically translates secondary virtual addresses into their primary virtual address.

We show in this paper that an SLB is much more scalable than a TLB (by orders of magnitude) so that very small SLBs of 16 entries or less suffice in general. The coverage of each entry in the SLB is much better than the coverage of entries in a regular TLB for two reasons. First, each SLB entry covers an entire segment containing a large number of pages and its coverage scales up with application data set sizes, physical memory sizes, and the number of processor nodes in a multiprocessor system. Second, the SLB only maps secondary virtual addresses. In general, the use of synonyms is not widespread, and therefore, the SLB only keeps translations in the rare cases where a virtual address has an alias. Because it is very small, the SLB can be accessed in parallel or serially with the first-level cache.

A major drawback of TLBs stems from the fact that the TLB must be flushed every time a translation between virtual and physical addresses is broken, e.g., on page swap-in or swap-out or on page migrations. In multiprocessors, from large-scale CC-NUMAs [19] to single-chip CMPs [18], each TLB flush must be global to prevent the existence of stale translations [5], [30]. This TLB consistency requirement is often maintained by a TLB shootdown, a complex software algorithm, which introduces significant overheads and does not scale well with the number of processors. The content of an SLB does not keep track of physical addresses and hence is not affected by most demapping and remapping of virtual to physical addresses. An SLB entry must only be flushed when synonyms are eliminated, e.g., when entire objects accessed with more than one virtual address are destroyed. Such events are much less frequent than changes of virtual-to-physical address mappings. We show through extensive simulations that flushes of SLB entries are extremely rare. Finally, we show that caches accessed with an SLB have a better hit rate than caches accessed with a TLB.

The SLB is part of the puzzle to support virtual address caches systemwide. To support an SLB, the operating system must identify and keep track of primary and secondary virtual addresses. Other classical issues with virtual address caches remain such as access right enforcement and interactions with I/O devices. Some of these issues have been solved in other contexts. Although we do not purport to present a complete systemwide solution to virtual address caches in this paper, we point out solutions to these issues exploiting the SLB throughout this paper.

The rest of this paper is structured as follows: Section 2 covers background material, including a brief summary of common usage of synonyms in current computer systems and an overview of cache/TLB architectures in modern processors. In Section 3, we propose the SLB to solve the synonym problem for virtually addressed caches and memories. A quantitative evaluation of SLBs is given in Section 4. After a brief overview of some related work in Section 5, we conclude this paper in Section 6.

2 Background

In this section, we first review the properties and usage of synonyms, and then, we review briefly the current memory architectures with TLBs and physical/virtual caches.

2.1 Synonyms

When multiple virtual addresses are mapped to the same physical address, these virtual addresses are called *synonyms*. Because virtual memory is managed in page granularity, synonyms are at least aligned on page boundaries. Two synonyms cannot exist within the same page and synonyms in two different pages must be located at the same address within each page.

Synonyms are very convenient to the kernel and user software in many situations. Synonyms are often used to implement shared memory semantics across different user virtual address spaces. Many read-only segments such as libraries and text segments are widely shared. In many operating systems, the kernel is globally shared at a fixed location in different virtual address spaces. Users may even define synonyms within the same thread for convenience.

Besides the true sharing semantics, synonyms are critical to optimize various memory operations. For example, *copy-on-write* avoids unnecessary memory copy operations and reduces the consumption of physical memory. In copy-on-write, two processes use the same page but do not really share it. They use different virtual addresses to access the same physical page until one of the processes modifies the page. At this time, a physical copy of the page is made and the new page is remapped in one of the process spaces. Copy-on-write is a compression optimization and is not part of shared memory semantics.

It is also very frequent that a physical page P can be remapped from one virtual page V1 to another virtual page V2. In this case, we consider that V1 and V2 are synonyms although the V1-P and V2-P mappings may not coexist at the same time. Traditional message passing semantics can be optimized using this remapping operation

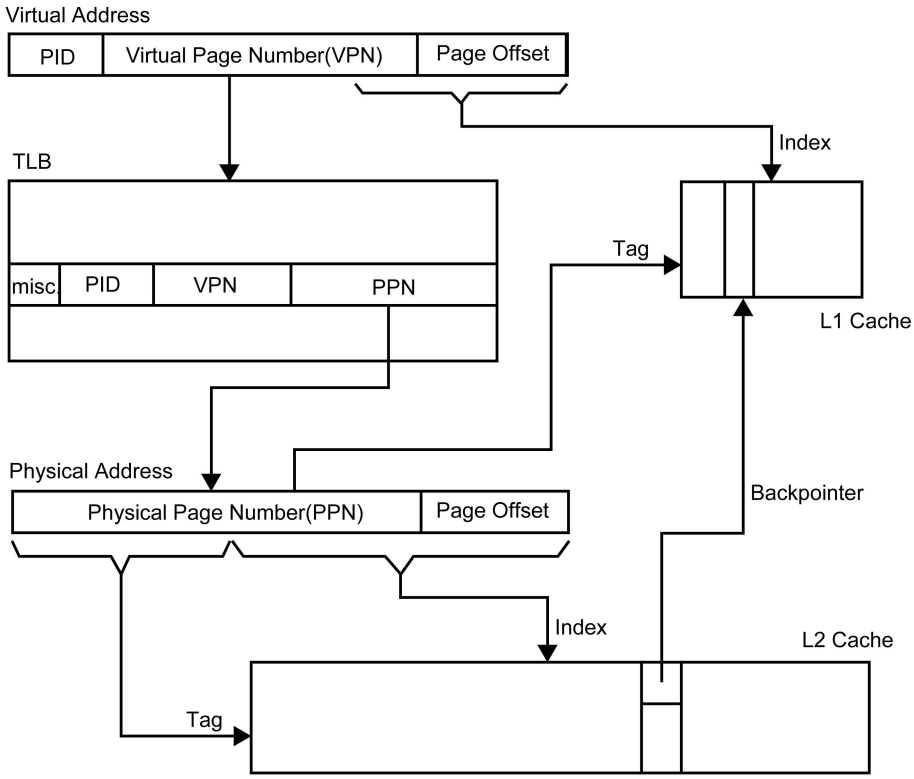


Fig. 1. Access to a classical TLB and a virtual first-level cache tagged with physical addresses. The second-level cache is physical with inclusion and a backpointer to the first-level cache helps locate synonyms in the first-level cache.

to avoid physical memory copies. As an example, if process 1 wants to send a message at virtual address V1 to process 2 at virtual address V2, the system can simply remap the physical page P containing the message from V1 to V2. The old V1-P mapping can either remain valid as copy-on-write or may be destroyed depending on the message passing semantics. As a special case, the operating system kernel usually buffers an I/O transfer in physical memory through kernel addresses and remaps them to user space when needed.

Finally, it is also very common that V1 and V2 share the *resource* of the physical page instead of its content. For example, in demand-paging systems, V1 may be swapped out and physical page P is freed and reallocated to another virtual page V2. V1 and V2 do not have any logical connection. Physical page P must be overwritten with new content before it is allocated to V2.

2.2 Caches with Translation Lookaside Buffers

Physical caches are accessed only with bits of the physical address, which is a unique systemwide identifier for each memory location. Accesses to TLB and L1 cache are done in parallel using physical address bits only. Physical caches are often preferred because they do not face the complications due to synonyms. However, keeping the L1 cache physical limits the possible L1 cache sizes to the size of a page times the degree of associativity [4]. Because of this, some microprocessors [14] adopt first-level caches with wide associativity, which is not necessarily the best choice from a design standpoint.

Several microprocessors employ virtual caches tagged with physical addresses. In virtual caches, there is no restriction on cache sizes, but the same memory block may be accessible with different synonyms and the ensuing coherence problems must be solved. Fig. 1 shows a typical architecture for a virtual L1 cache and a TLB accessed in parallel. L1 cache blocks are tagged with their physical address, but virtual addresses are used to index the cache. After a cache set is indexed with the virtual address, all tags in this set are compared with the physical page number translated in parallel by the TLB. A miss in the set does not guarantee that the data block is not in cache. The block may still reside in another set previously indexed with a synonym. If a memory access misses in the indexed cache set but is found in another set, a *short miss* moves the cache block to the indexed set.

As shown in Fig. 1, a common approach to detect the presence of a synonym in a virtual L1 cache is a reverse mapping implemented with backpointers [32] stored in the L2 cache, which is indexed and tagged with physical addresses. Inclusion is required: If a block is in the L1 cache, then it must have a copy in the L2 cache. If an L1 cache miss hits in the L2 cache and the backpointer points to an L1 cache block, the block is moved to the new cache set, and the backpointer in the L2 cache is updated to point to the new location in the L1 cache (short miss). The penalty of a short miss is similar to the penalty of a cache miss serviced by the second-level cache.

To satisfy the access constraints, the latency and bandwidth requirements of the TLB must scale up with the clock rate, ILP, and TLP, which is getting more difficult and costly. A recent example is the Niagara core. In Niagara, a 64-entry TLB is shared by the four threads running on the same core, and the TLB bandwidth and latency requirements are pipeline bottlenecks [18].

Notwithstanding the problems of access time and bandwidth, the TLB size is fixed within the microprocessor chip and does not adapt well to growing application sizes and various system memory sizes. In a CMP, the effective amount of TLB does not increase as fast as the aggregate hardware dedicated to TLBs in all processor cores because some entries are replicated. In turn, this replication creates the problem of TLB consistency [30]. Maintaining TLB consistency is very expensive and does not scale well. It is done by TLB shootdown, which must be invoked every time the mapping between virtual and physical addresses must be broken.

For a given application, the miss rate of the TLB is primarily determined by the TLB reach or coverage, which is the aggregate memory area mapped by all the TLB entries. One way to improve the performance and scalability of a traditional TLB is to use *superpages* [29]. The size of a superpage is a power-of-two multiple of the size of the base page. Each entry of the TLB can be configured to map superpages of various sizes, effectively increasing the coverage of each TLB entry. One major drawback of superpages is that physical pages inside a superpage must be contiguous in physical memory. The operating system must allocate a contiguous area of physical memory to accommodate the superpage size and must swap in the entire superpage. In order to allocate large pieces of contiguous physical memory, it is necessary to dynamically reallocate physical pages, which requires the remapping of virtual-to-physical translations, a very expensive operation in multicore systems due to the TLB consistency overhead. In general, the size of a superpage cannot be determined easily at the time when the virtual address is first allocated, which leads to suboptimal allocations.

Fig. 2 shows a cache hierarchy in which L1 and L2 caches are indexed and tagged with virtual addresses. The TLB is accessed on L2 misses only. This configuration has several advantages. First of all, the TLB is off the critical path and thus may be very large without slowing down the processor. Second, the TLB benefits from a *filtering* effect by the caches above it, because the TLB does not need to cover all blocks stored in the L1 and L2 caches. A new entry needs to be brought into the TLB on L2 cache misses only. Additionally, the TLB can be moved further down the hierarchy. In a multicore system such as a CMP, the TLB could be moved to shared memory or to a shared cache [18], in which case the TLB is shared by all the cores. In this case, TLB shootdown is avoided within the CMP and the TLB benefits from *sharing* and *prefetching* effects. The sharing of TLB entries increases the effective TLB size by avoiding replications. A thread running on a different core may load an address translation in the shared TLB before a thread accesses it, thus in effect prefetching or preloading the entry.

To conclude this section, in a virtual cache with physical tags, synonyms are exposed by comparing their physical tags to the TLB entry accessed in parallel with the cache. When the cache tags are virtual, the virtual-to-physical address translation is postponed until an access reaches some point in the memory hierarchy (e.g., right after the L2 cache in Fig. 2). There are many advantages to this, but, since the cache tags are virtual, synonyms cannot be exposed by their tags, even if they index the same cache set. Synonyms can be eliminated by software or by organizing the virtual space into a single virtual address space shared by all processes, as is done in the PowerPC architecture [16], [21] in which synonyms are supported at the segment level. Otherwise, synonyms must be dynamically detected to avoid ambiguities in the memory hierarchy. The hardware to do this is often referred to as *anti-aliasing hardware*. One form of anti-aliasing is to keep L2 physical and have a backpointer to L1, as shown in Fig. 1. In Fig. 2, on a miss in L2, all the sets in L2 in which the block could reside must be searched line by line with the help of the TLB. To simplify this complex procedure and enable the use of virtual addresses throughout the memory hierarchy, we propose a new form of anti-aliasing hardware: the addition of a small SLB to each core.

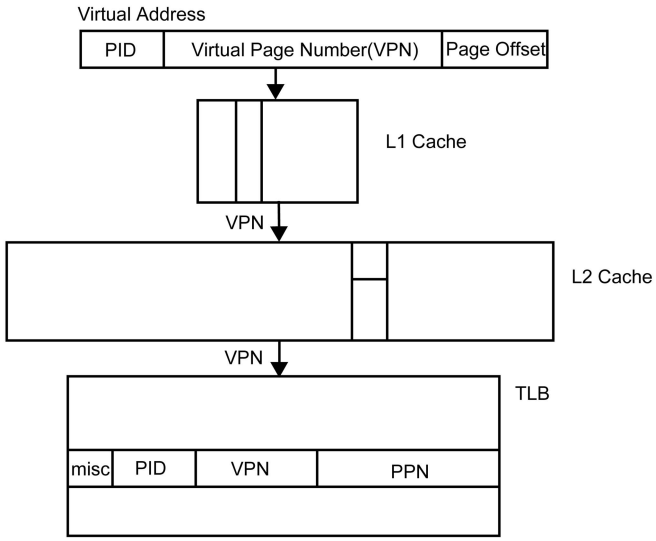


Fig. 2. Virtual L1 and L2 caches. L1 and L2 are indexed and tagged with virtual addresses. The TLB is accessed on an L2 miss only.

3 Synonym Lookaside Buffer (SLB)

The coherence of any memory hierarchy is greatly simplified (both in single core and multicore systems) if a unique identifier exists for each page. In traditional memory hierarchies, the unique identifier is given by the physical page number. In a virtual address memory hierarchy, this identifier must remain virtual. The SLB relies on operating system support to identify and maintain a unique virtual address identifier for each page.

3.1 Primary Virtual Address as Unique Virtual Address Identifier

The unique identifier for each page is the *primary virtual address*. When a page is accessed with a single virtual address (no synonyms), then its virtual address is also its primary virtual address. When a page is accessible with multiple synonyms, then one of the virtual addresses among all the synonyms is selected as the primary virtual address. Any virtual address that is not a primary virtual address is called a *secondary virtual address*. Table 1 illustrates the concept of primary and secondary virtual addresses. In this example, there are eight virtual addresses mapping to four physical pages. Pages P1 and P3 are each accessed with a single virtual address, while P2 and P4 can be accessed with four and two virtual addresses, respectively. W and Y are primary virtual

TABLE 1
Primary and Secondary Virtual Addresses

PRIMARY VIRTUAL ADDRESSES	SECONDARY VIRTUAL ADDRESSES	PHYSICAL ADDRESSES
W	--	P1
X	X1, X2, X3	P2
Y	--	P3
Z	Z1	P4

addresses and have no synonyms. For P2 and P4, virtual addresses X and Z have been selected as primary virtual addresses. X1, X2, and X3 are synonyms of X, and Z1 is a synonym of Z. X1, X2, X3, and Z1 are secondary virtual addresses.

The operating system must be involved in identifying and maintaining the primary and secondary virtual addresses. The operating system keeps track of page sharing information in the virtual memory system. The virtual memory system is made of several layers. The bottom layer is responsible for physical activities such as demand paging. The information on page sharing is mostly kept in the logical segment layer. A few modifications should be made to the operating system in the logical segment layer to take advantage of an SLB. The major modification is to integrate into the internal page management tables the information related to primary and secondary addresses for each page.

The primary virtual address must be selected among all the virtual addresses sharing the same page. The operating system should select the address with the longest lifetime as the primary address, because the overhead to remap a primary virtual address is much higher than the overhead to remap a secondary virtual address. Allocating primary virtual addresses by first touch is both simple and effective because of the way the operating system forks processes.

We have mostly used first touch to assign primary virtual addresses in our simulation experiments. There are situations when the first touch policy is not the best. For example, the remap of physical addresses is widely used by the kernel to communicate with user processes and to optimize message passing. In one very common scenario, the kernel prepares a page on behalf of a user process in kernel space and then remaps the page to user space. The kernel address is then freed and reused by other pages. Since this is a primary virtual address change, the current first touch policy may generate excessive cache flushing for this very common case. Thus, we propose an optimization for remap-based message passing, especially for kernel buffer handling. We call this optimization the *lock-bit optimization*. Suppose V2 is a user receiver buffer expecting a message from V1, which is mapped to a physical page P. The operating system demaps V1-P and remaps V2-P to pass the page. Under first touch, V1 is the primary virtual address and its frequent demapping is very costly. To reduce the overhead, V2 is deemed the primary virtual address and V1 a secondary virtual address. In order to synchronize, V2 accesses must be locked out before the page is actually transferred. A lock bit is added in the page table and in the TLB for the V2-P translation. This lock bit is part of the access right bits and is also copied in the virtual caches. While the lock bit is set, any access via V1 proceeds as usual and an access to the page via V2 is locked out by the page table. This lock-bit optimization was implemented in all our simulation experiments.

3.2 Basic Design: Parallel SLB and L1 Cache Accesses

Virtual caches can use the primary virtual addresses for global naming purposes, a role commonly fulfilled by the physical address. Secondary virtual addresses must be translated dynamically into their corresponding primary virtual addresses. This is the function of the SLB. The SLB can be accessed in parallel with the L1 cache, as shown in Fig. 3. This organization is similar to that of a system with a traditional TLB shown in Fig. 1. Like a TLB, the SLB is accessed with the virtual address and contains the same state bits as a TLB. Backpointers in the L2 cache expose synonyms in the L1 cache. The major differences between the schemes in Figs. 1 and 3 are that

1. the SLB translates a virtual address into a primary virtual address instead of into a physical address,
2. the SLB only translates secondary virtual addresses,
3. the L1 cache is indexed with virtual addresses and tagged with primary virtual addresses,
4. the L2 cache and all virtual caches below it are indexed and tagged with primary virtual addresses, and
5. the SLB does not always trap the processor on a miss.

If a virtual address hits in the SLB, then it must be a secondary virtual address. Its primary virtual address is retrieved from the SLB and is used to check the L1 cache tag. If the L1 cache hits, the value is returned. If it misses, the primary virtual address is used to access the L2 cache, resulting in a short miss (L2 hit with valid backpointer) or in a reload of the block from the L2 cache or any lower level memory.

When a virtual address misses in the SLB, the SLB is bypassed and the virtual address is used to check the L1 cache tag. The address could be either a primary or a secondary virtual address. If the address hits in the L1 cache, then it is a primary virtual address and the value is returned. If the address misses in the L1 cache, the block could be present in a different set. Consequently, the virtual address is used to access the L2 cache. A hit in the L2 cache

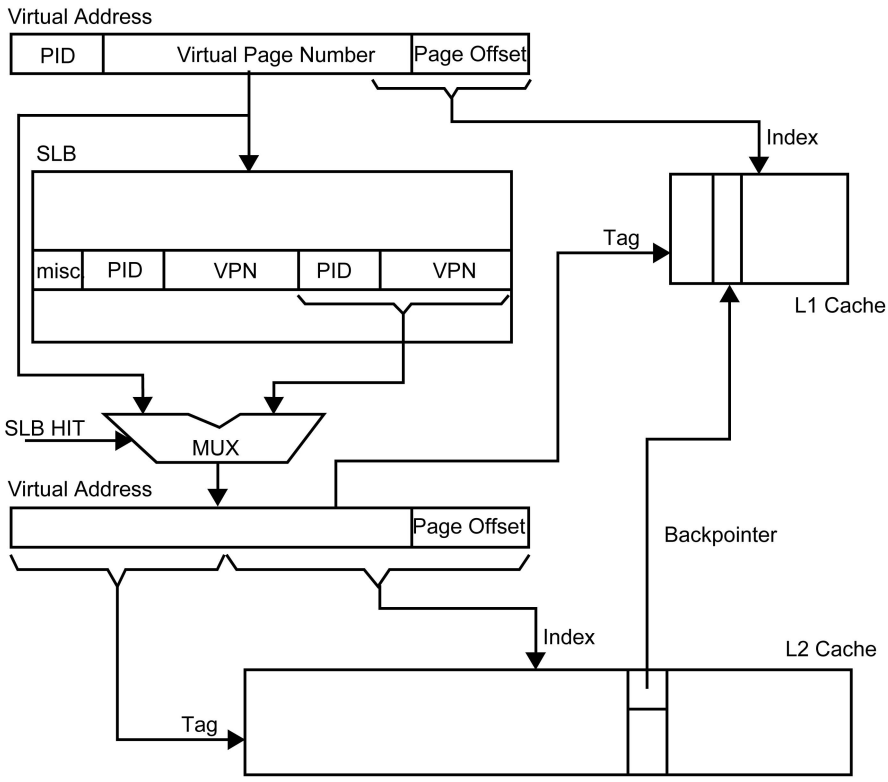


Fig. 3. Access to a virtual cache in parallel with an SLB.

TABLE 2
Action Taken on All Possibilities in Parallel SLB/L1 Cache Accesses

SLB Hit	L1 Hit	TLB Address	Action
Yes	Yes	Primary	Not possible
Yes	Yes	Secondary	L1 Hit. Value is returned from L1 after translation in SLB
Yes	No	Primary	Not possible
Yes	No	Secondary	L1 Miss. Value is returned from lower level caches/memories
No	Yes	Primary	L1 Hit. Value is returned from L1
No	Yes	Secondary	Not possible
No	No	Primary	L1 Miss. Value is returned from lower level cache/memories
No	No	Secondary	Trap processor. Refill SLB. Retry.

with a valid backpointer indicates that the address is a primary virtual address and triggers a short miss in the L1 cache. Otherwise, the data is returned by the L2 cache or by any memory below L2. If the address hits in any memory above the TLB, then it was a primary virtual address and no translation is loaded in the SLB. Synonyms are eventually exposed at the TLB and the TLB indicates whether the address is a primary or a secondary virtual address. If the address is a secondary virtual address, a NACK is sent back to the processor, a software trap handler fills the SLB with the secondary/primary virtual address translation, and the access is then retried.

These possibilities are illustrated in Table 2. The "TLB Address" column indicates the type of virtual memory address (primary or secondary) recorded in the TLB. We do not show the case of a TLB miss to simplify. On a TLB miss, the TLB must be reloaded from the page tables, possibly resulting in a page fault.

It is important to note that an SLB reload only happens whenever 1) the SLB misses and 2) the TLB detects that the address is a secondary virtual address.

3.3 Serial SLB and L1 Cache Accesses

In fact, the SLB is so scalable that our results show that an SLB of 8 to 16 entries is largely sufficient. Hence, it is possible to envision the architecture in Fig. 4, in which the first-level virtual cache and the SLB are accessed serially.

In this memory architecture, the first-level cache can only hit on primary virtual addresses. If the SLB hits, then the primary virtual address is used to access the L1 cache. If the SLB misses, the address may be a primary or secondary virtual address. If the address hits in any one of the virtual caches, then it is a primary virtual address and the value is

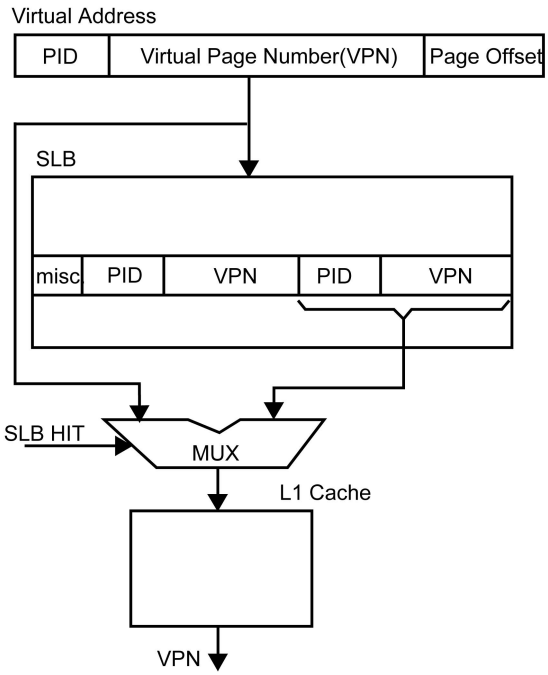


Fig. 4. Serial accesses to an SLB and a virtual cache.

returned. At the point where the physical address is needed, the TLB decides whether this is a primary or a secondary virtual address. If it is a secondary virtual address, an SLB miss trap is triggered, the SLB is reloaded with the secondary/primary virtual address translation, and the access is retried.

The L1 cache behavior and complexity are improved as compared to the parallel access. The drawback is the increased latency of L1 cache accesses. Note, however, that the small SLB simply introduces an additional indirection similar to a small segment register file in segmented architectures. We will evaluate this architecture as well later.

A particular implementation may combine serial and parallel accesses to the SLB. Serial and parallel accesses to SLB are not exclusive. For example, a "micro-SLB" to map primary addresses before L1 cache indexing, plus an SLB accessed in parallel with L1 cache might be a good idea. The addition of the micro-SLB reduces the number of short misses without really affecting L1 access time.

3.4 SLB/TLB/Cache Flushes

In a virtual cache with physical tags and a TLB, every time a virtual address is remapped to a different physical address the TLB entry must be flushed. By contrast, in a system equipped with an SLB, most virtual-to-physical address mapping changes such as changes resulting from paging activities (swap-in and swap-out) or from page migrations do not flush the SLB.

The only mapping changes that affect the SLB or the virtual caches are those where the content of a *virtual* page is changed through the remapping. For example, when the process image is overlapped through an exec() system call, or when a process is terminated and its virtual space is reclaimed by another process, the virtual caches must be flushed for all demapped primary virtual addresses and the SLB must be flushed for all demapped secondary virtual addresses.

The TLB is not eliminated in a system with SLBs because, at one point, possibly deep into the hierarchy, the virtual address must be translated to a physical address to access the main memory. Unless the TLB is shared at the memory (which is the most aggressive approach), TLB shootdown in multiprocessors is still needed when a virtual-to-physical mapping is changed. However, the TLB shootdown (for virtual-to-physical mapping changes) does not require cache flushing (because all caches above the TLB use virtual addresses).

3.5 Comparison with Segment Registers

Most current operating systems implement demand paging virtual memory. On top of the machine-dependent layer, which manages virtual-to-physical address translations, a machine-independent layer structures virtual memory in logical segments. The sharing of synonyms is decided in this logical layer independently of physical paging. Shared synonym segments are identified in the machine-independent layer of the virtual memory system and are allocated in coarse granularity. Segmented architectures such as the PowerPC architecture [16], [21] take advantage of this to manage segment registers used to translate logical addresses into virtual addresses.

This scheme is depicted in Fig. 5 and requires the translation of a logical address into a virtual address through a set of segment registers. Segment registers play a similar role as the SLB and are supported by the same O/S layer as the SLB. The major difference between an SLB and segment registers is that the content of the SLB is dynamically managed in hardware while segment registers are loaded and reclaimed in software, which puts an additional burden on software and is less dynamic or flexible. Furthermore, the SLB contains translations for secondary virtual addresses only. Since synonyms are the exception rather than the norm, we expect that SLBs managed in hardware will be more effective than segment registers managed in software. Because segment sharing is already managed in the operating system, some minor changes are only needed for current operating systems to identify synonym segments and to support an SLB to resolve ambiguities between virtual addresses at the granularity of segments.

3.6 SLB Scalability

The major advantage of an SLB over a TLB is its much better scalability, as the performance evaluations will show. Here, we explain why the SLB is so scalable.

Synonyms are allocated in very coarse granularity and cover a large number of pages. Each SLB entry can cover an entire segment. This has the same effect on the coverage of SLBs as superpages have on the coverage of TLBs, but the major difference is that all the pages of a superpage must be collocated in physical memory. By contrast, pages in segments accessed by synonyms do not have to be contiguous or even allocated in physical memory. In the example shown in Fig. 6, segments V1 and V2 are synonyms. The physical address mapping is not contiguous and the virtual pages in the synonym segments do not all map to a physical page.

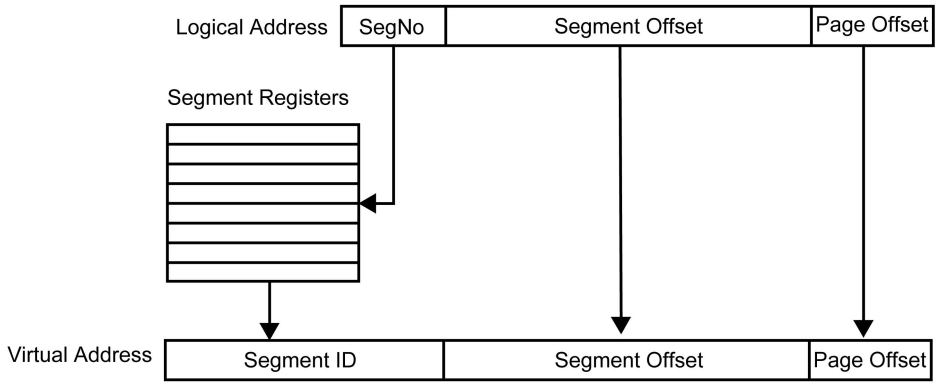


Fig. 5. Dynamic segment translation in a segmented system.

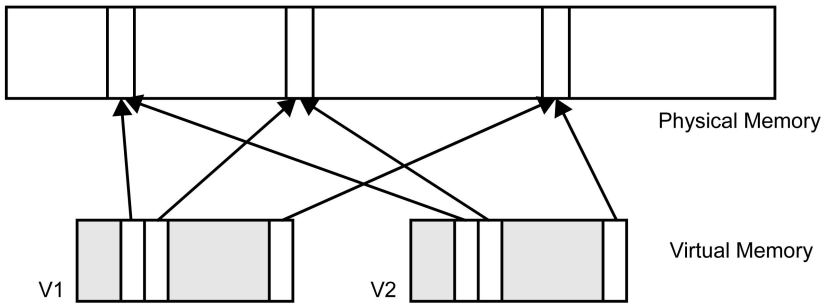


Fig. 6. A synonym example.

Nevertheless, the synonym can be represented in one single SLB entry if it is appropriately aligned. The consumption of SLB entries by each process depends on its synonym usage instead of on its memory allocation. Increasing data set sizes does not create more synonyms—the synonyms simply cover larger memory areas. In fact, the size of the SLB is mostly dictated by the operating system and the programming style and not by application sizes, number of processors, or size of physical memory.

Another advantage of the SLB boosting its scalability is that it only maps secondary virtual addresses, whereas a traditional TLB must translate *all* virtual addresses. In general, the use of synonyms is not widespread, and therefore, SLB entries are occupied by secondary/primary virtual address translations in the rare cases where a virtual address has an alias. This is an advantage over segment registers as well (since segment registers must map all segments).

3.7 Impact on Memory Exceptions

In a traditional TLB-based system, when a Load or a Store is ready to issue to cache, it accesses the TLB and a TLB hit guarantees that the access will be exception-free. A Store can retire when it is placed in the Store buffer, since it is then guaranteed to complete without exceptions. Stores are then propagated to memory according to the memory consistency model [9].

Unfortunately, the SLB cannot verify/resolve memory trap conditions for virtual addresses as the traditional TLB does. Accesses missing in the SLB are nonblocking and may trigger memory traps inside the memory hierarchy (as was explained in Section 3.2). Thus, the processor must be able to execute late memory traps efficiently. This problem was thoroughly discussed elsewhere [25]. The major issue is the use of a Store buffer. Loads are not a problem because they cannot retire before the value is returned, at which time the exception condition is known. However, Stores could be retired as soon as they enter the Store buffer if they are exception-free. When memory exceptions can be triggered deep in the memory hierarchy, the processor cannot retire instructions until the exception condition on a previous Store is returned from the memory hierarchy, rendering the Store buffer all but ineffective.

This is not a problem for sequentially consistent systems since the Store buffer is mostly ineffective in such systems. However, it can greatly impact systems with a relaxed memory model such as Total Store Order (TSO) or Release Consistency (RC). To be able to take advantage of the Store buffer in such systems, a *tagged Store buffer* that implements deferred memory exceptions was proposed [25]. The tagged Store buffer makes sure that Stores in the Store buffer will complete even if they trigger memory exceptions later in the memory hierarchy.

3.8 Synonym Coherence

In the rare instances when the same data are accessed with different synonyms in close proximity (i.e., within the same process) in process order, coherence may be violated. A Store to a secondary virtual address may miss in the SLB and in the virtual address cache hierarchy even though its primary virtual address is present in the cache hierarchy. In this case, a following Load accessing the same data with the primary virtual address may hit in a virtual cache before the trap for the Store is detected deep in the memory hierarchy, thus bypassing the previous synonym access. Coherence is violated.

TABLE 3
Benchmarks

Benchmarks	Instructions	Loads/Stores	Description
Pmake	1427M	570M	4 way parallel Makes for Modified Andrew Benchmark, many small, short-lived processes that make heavy use of OS services
Radix	1212M	269M	Splash benchmark, with parameters -n2097152 -r2048 -m4194304
Rsim	4771M	2434M	Instruction driven simulator rsim[22] runs radix, sequential consistency with speculation, simulate uniprocessor for 200k cycles
TPC-C	34422M	15695M	OLTP benchmark TPC-C runs on Postgres95 database system, 4 warehouses, scale 100 times, 46 transactions mixed
Kaffe	13698M	5368M	JAVA interpreter Kaffe runs a raytracer written in java

This is *not* an issue for processor architectures implementing sequential consistency because memory operations are retired one after another, after they are completed. Loads are not retired until all previous Stores are retired and globally performed, at which point the synonym coherence problem has been resolved. However, for relaxed memory models, the synonym coherence problem is real because the Stores can be retired as soon as they reach the local Store buffer. To solve the synonym coherence problem in the context of relaxed memory models, one may restrict synonym usage or alter the management of the SLB to include some primary virtual addresses as well [24]. However, the synonym coherence problem can be solved simply by taking advantage of the memory disambiguation and memory forwarding hardware in the microarchitecture and the fact that synonyms are aligned on page boundaries.

Typically, Loads and Stores are dispatched to a Load/Store queue, waiting to be issued to the data cache [23]. Memory disambiguation and forwarding hardware is needed to detect for each Load whether a previous Store with the same address is currently pending so that the data can be forwarded from the Store to the Load. Since the Load/Store queue uses effective (virtual) addresses, synonyms must be resolved in the Load/Store queue. An easy way to do this is to check the page displacements and the virtual page numbers separately. If the page displacements and the virtual page numbers match, then the data should be forwarded from the Store to the Load. If the page displacements do not match, then the addresses point to different data and no forwarding should occur (no synonym). If the page displacements match but the virtual page numbers do not match, then the two addresses may be synonymous and the Load should not issue until the Store is retired. A similar approach can be used for Stores in the Store buffer.

3.9 Access Right Support

Just as in the traditional TLB, an SLB entry includes protection bits associated with a secondary virtual address. All virtual cache entries tagged with primary virtual addresses must also contain protection bits associated with the primary virtual address. Accesses to a secondary virtual address are checked for protection twice, once in the SLB and once in the cache. Therefore, access restrictions to a primary virtual address should be less or equal to access restrictions to all its synonyms, to avoid useless access right violation traps.

4 Performance Evaluations

4.1 Methodology

Table 3 shows the benchmarks used in the simulations. The column titled "Instructions" shows the total number of instructions simulated, and the column titled "Loads/Stores" indicates the total number of memory accesses. These benchmarks represent applications from widely different domains including a JAVA virtual machine, an OLTP commercial workload, an architecture simulation, a multiprogramming workload, and a compute intensive Splash benchmark.

We use trace-driven simulations. Traces are collected by running SimOS, a complete machine simulator developed at Stanford University [13]. The simulated SGI workstation has 64 Mbytes of main memory, 64 Kbytes of instruction cache, 64 Kbytes of data cache, and a 1-Mbyte L2 cache, and it runs the Irix 5.3 operating system. The page size is 4 Kbytes. All user and kernel memory accesses are recorded in the trace.

With the trace, we then simulate a single-processor system with a two-level cache hierarchy. The L2 cache is fixed with a size of 1 Mbyte and is two-way set associative. L1 cache sizes vary from 8 Kbytes to 1 Mbyte, and its organization is either direct-mapped or four-way set associative. The cache block size is 64 bytes for all caches. The LRU policy is used whenever replacement is needed.

In order to support the SLB in the simulations, we dynamically detect and construct sharings among virtual addresses. We maintain a page table, a reverse page table, and a segment table in the simulation. Every memory access is first checked with these tables in an efficient way before it is sent to the trace-driven modules. Any new mapping or change of mapping between virtual and physical addresses triggers an update of the table structures, where we aggressively construct and merge virtual address segments and sharings among the segments. We are very conservative with demaps. We always flush either the virtual caches (demap of a primary virtual address) or the SLB (demap of a secondary virtual address) on every page demapping, even if some or most of these flushes could be avoided.

4.2 TLB/SLB Misses

Table 4 compares the number of TLB and SLB misses in our experiments. The TLB column shows the number of TLB misses for a 32-entry TLB, the SLB column shows the number of SLB misses for a 16-entry SLB, and the last

TABLE 4
Number of TLB/SLB Misses

Benchmarks	TLB(32entries)	SLB(16entries)	Ratio
Pmake	2419455	241011	10
Radix	4170193	5246	795
Rsim	45285839	2738	16540
TPC-C	178191107	58404	3051
Kaffe	33613434	4665	7205

TABLE 5
SLB Hit Rate Distribution (in Percent)

Benchmarks	hit(16)	hit(>16)	miss
Pmake	99.5627	0.4097	0.0275
Radix	99.9697	0.0036	0.0267
Rsim	99.9956	0.0006	0.0038
TPC-C	99.9873	0.0053	0.0074
Kaffe	99.9831	0.0034	0.0136

column gives the ratio between the numbers of TLB and SLB misses.

Even though the TLB has double the size of the SLB, the number of TLB misses surpasses the number of SLB misses by orders of magnitude. The only benchmark for which the TLB could compare with the SLB is Pmake. Pmake consists of many small and short-lived processes with small data sets. It does not have big enough data sets to pressure the TLB, while it creates a lot of cold misses because of remaps and process termination/creation. Still, the ratio between the miss rates is one order of magnitude.

Table 5 shows the distribution of SLB accesses in an SLB of infinite size, excluding the accesses in kernel idle and synchronization. The column titled "hit(16)" gives the percentage of secondary virtual address accesses that hit within the first 16 entries. This is the hit rate for a 16-entry SLB with LRU replacement policy. (If kernel idle and synchronization were included, this hit rate would be much higher because their synonym accesses always hit.) The column titled "hit(> 16)" gives the percentage of secondary virtual address accesses that hit on SLB entries other than the first 16 entries. The "miss" column is the percentage of secondary virtual address accesses that miss in an SLB of infinite size. It is clear that a very small SLB with 16 entries can cover the overwhelming majority of synonym accesses and that the remaining misses are largely due to cold misses and remappings, which cannot be improved by increasing the size of the SLB.

Fig. 7 compares the number of TLB and SLB misses in our five benchmarks as a function of the number of entries. At the low end of these graphs, we observe that the number of TLB misses overwhelms the number of SLB misses by orders of magnitude.

Fig. 8 shows the number of TLB and SLB misses as a function of the TLB/SLB size for two SPLASH benchmarks, namely RADIX and OCEAN, and for four different data set sizes. These are the only benchmark we have for which we can easily scale the data set size. From these graphs, the miss curve for the TLB rapidly deteriorates as the data set size increases, especially for RADIX. By contrast, an SLB of size 8 or bigger has a negligible amount of misses in all cases.

4.3 SLB Flushes

Table 6 shows the activity associated with page remaps in our five benchmarks. In our simulations, we always flush *either* the SLB entry (secondary virtual address access) *or* the cache hierarchy (primary virtual address access) on a remap, even if this is not always needed. Our counts do not include the activity associated with reclaiming pages at the end of the benchmark execution because they are not in the application's critical path. In the table, "Remaps" is the number of virtual-physical page remaps in the execution. Some of these remaps flush the SLB only, and the rest only flush the caches. "SLB entry flushes" is the number of SLB flushes. "L2-cache block flushes" is the total number of L2 cache blocks flushed by all these remaps. L2 is accessed with primary virtual addresses only.

These numbers are practically negligible, as compared to the number of Loads and Stores in these benchmarks. The majority of the remaps are for synonyms in kernel space, which explains the tiny number of L2 cache blocks that are flushed. User address remap is very rare even for Pmake, which is a multiprogramming workload.

4.4 Cache Miss Rates

Because the cache index function is different in different types of L1 caches, conflict misses are affected. Short misses are also an issue in some caches. Their number grows with the cache size (because indexes are spread over more sets), contrary to conflict misses whose number tends to decrease with the cache size.

We consider three different address indexings of an L1 cache referred to as *PHYSICAL*, *PRIMARY*, and *VIRTUAL*. In *PHYSICAL*, the cache is indexed and tagged with physical addresses (case of Fig. 1 when the index is part of the page displacement). *PHYSICAL* is not affected by

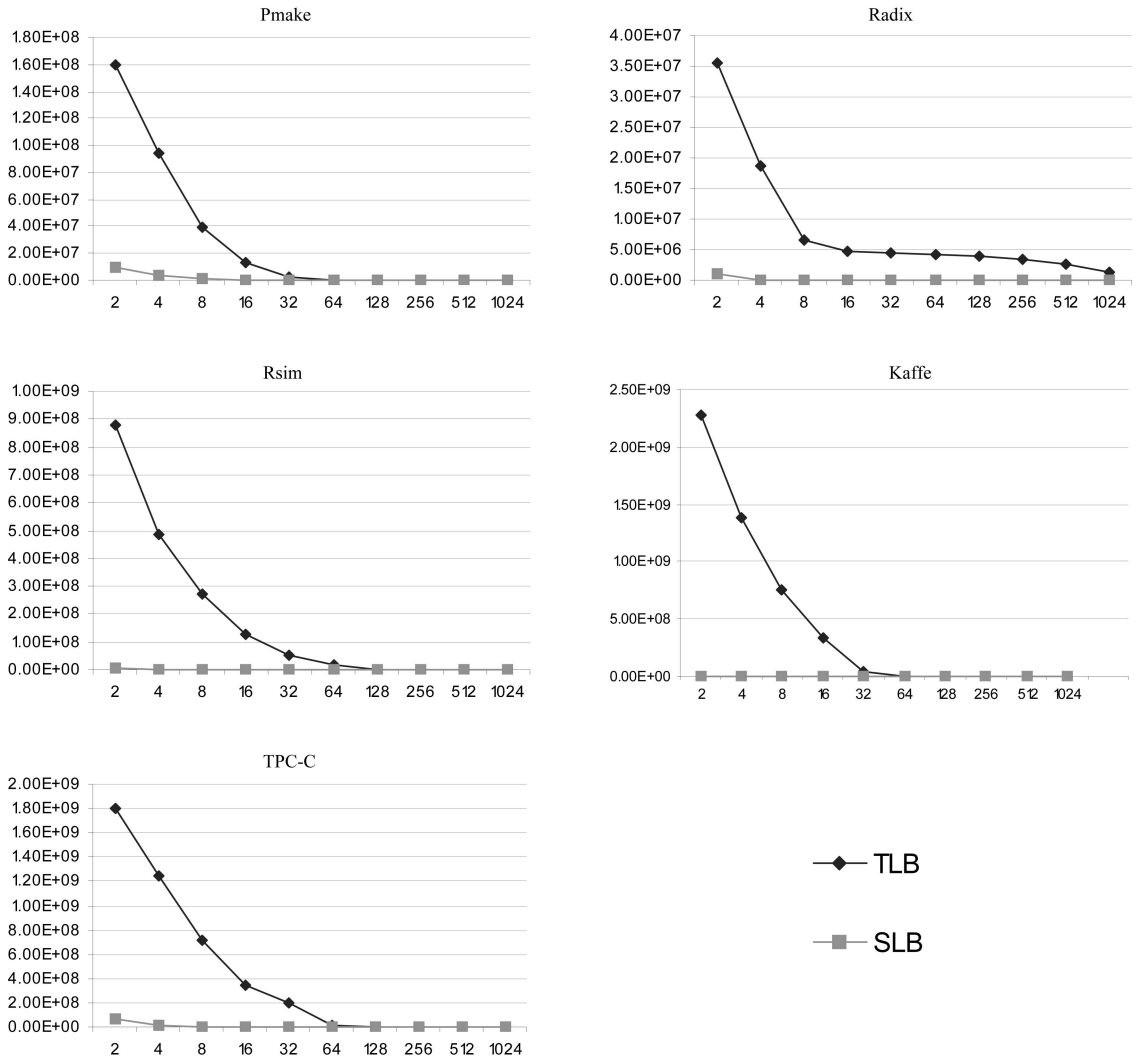


Fig. 7. Number of TLB and SLB misses as a function of the number of entries.

synonyms and has no short misses. *PRIMARY* is the case where a cache is accessed with primary virtual addresses only (case of Fig. 4). *PRIMARY* is not affected by synonyms and has no short misses. Finally, in *VIRTUAL*, the cache may be indexed with synonyms. This is the case where a virtual cache is accessed in parallel with a TLB (see Fig. 1) or where a virtual cache is accessed in parallel with an SLB (see Fig. 3). *VIRTUAL* must deal with short misses.

4.4.1 Total Miss Rates

Fig. 9 shows the total miss rates, including kernel and user, for direct-mapped and four-way set associative L1 caches. All curves show the same trends. The index to the cache only affects conflict misses, which are significant in direct-mapped caches.

A very big gap exists between *PHYSICAL* on one hand and *PRIMARY* or *VIRTUAL* on the other in the case of direct-mapped caches for RSIM and TPC-C benchmarks. This gap is clearly due to conflict misses. A closer look into the RSIM trace exposes that one physical page for the stack was aligned with one data page at a 1-Mbyte boundary. This suggests that for low associativity, virtual caches are more robust than physical caches. Although the operating system can select the virtual-to-physical page mapping to minimize cache conflicts, its effectiveness at doing so is questionable, whereas virtual address indexing takes full advantage of the spatial and temporal localities naturally exhibited by most programs [20]. In four-way caches, all methods of indexing the L1 cache exhibit similar miss rates.

4.4.2 Short Misses

The slight difference between the miss rates of *VIRTUAL* and *PRIMARY* is due to short misses. Fig. 10 shows the fraction of short misses in all the benchmarks, i.e., the number of short misses divided by the total number of misses in *VIRTUAL*. Again, we show the results for direct-mapped and four-way set-associative caches.

When the cache size increases, more synonyms are indexed into different sets (because there are more sets) and tend to stay longer in the cache (because the miss rate drops), leading to an increase in short misses. At the same time, the number of capacity misses decreases sharply. In our benchmarks, we observe a sharp increase of the fraction of short misses for all benchmarks as soon as the cache size reaches 128 Kbytes.

These results suggest that for small virtual caches there is little difference between the hit rates of virtual and primary

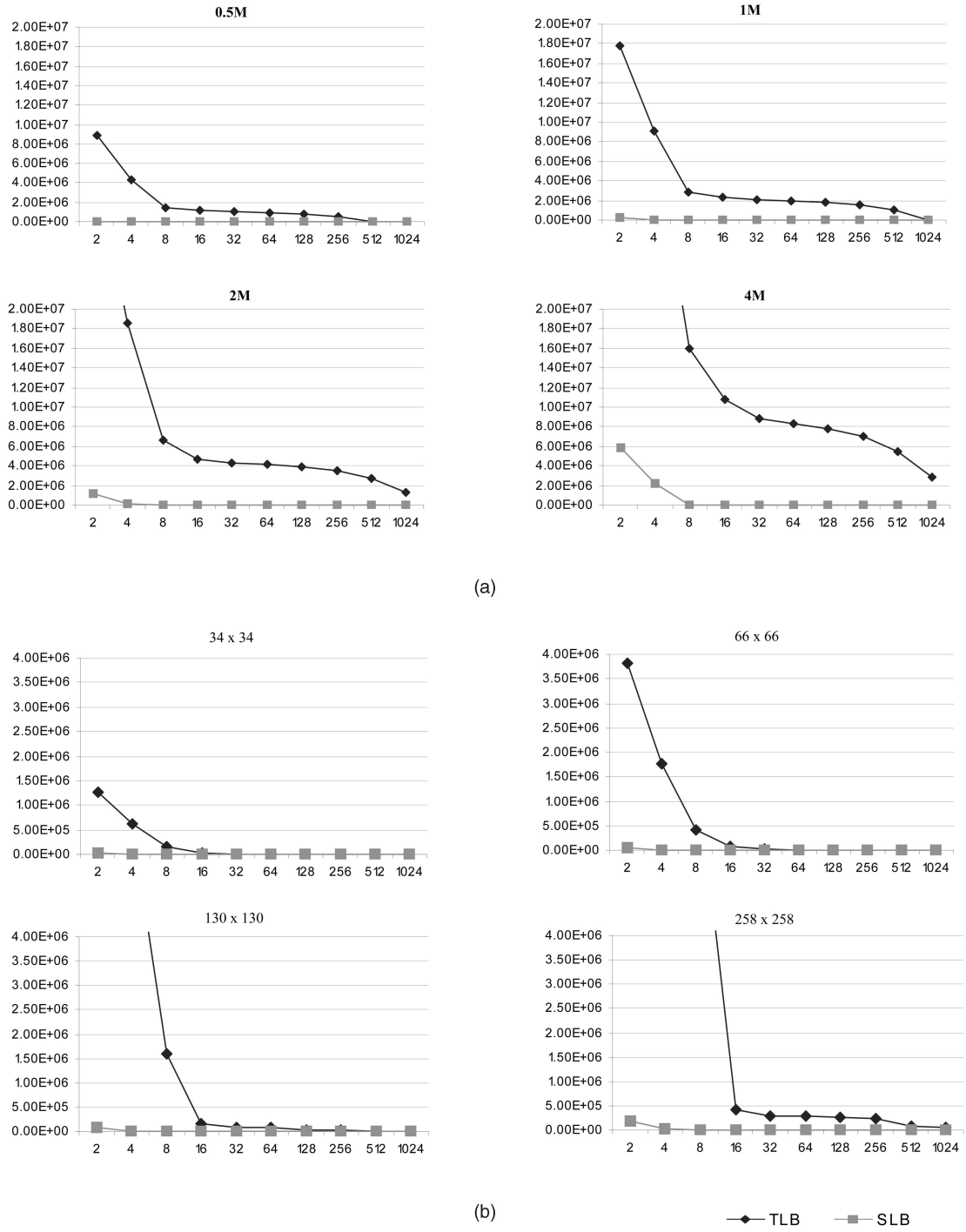


Fig. 8. Number of TLB and SLB misses versus number of entries in RADIX and OCEAN with various data set sizes. (a) RADIX. (b) OCEAN.

TABLE 6
Page Remaps and SLB/Cache Flushes

Benchmarks	Remaps		SLB entry flushes		L2-cache block flushes	
	User	Kernel	User	Kernel	User	Kernel
Pmake	799	11311	490	10211	11008	24587
Radix	10	4471	4	4460	345	245
Rsim	6	2301	3	2275	112	738
TPC-C	18	33072	9	33028	317	943
Kaffe	923	3002	326	2925	2710	833

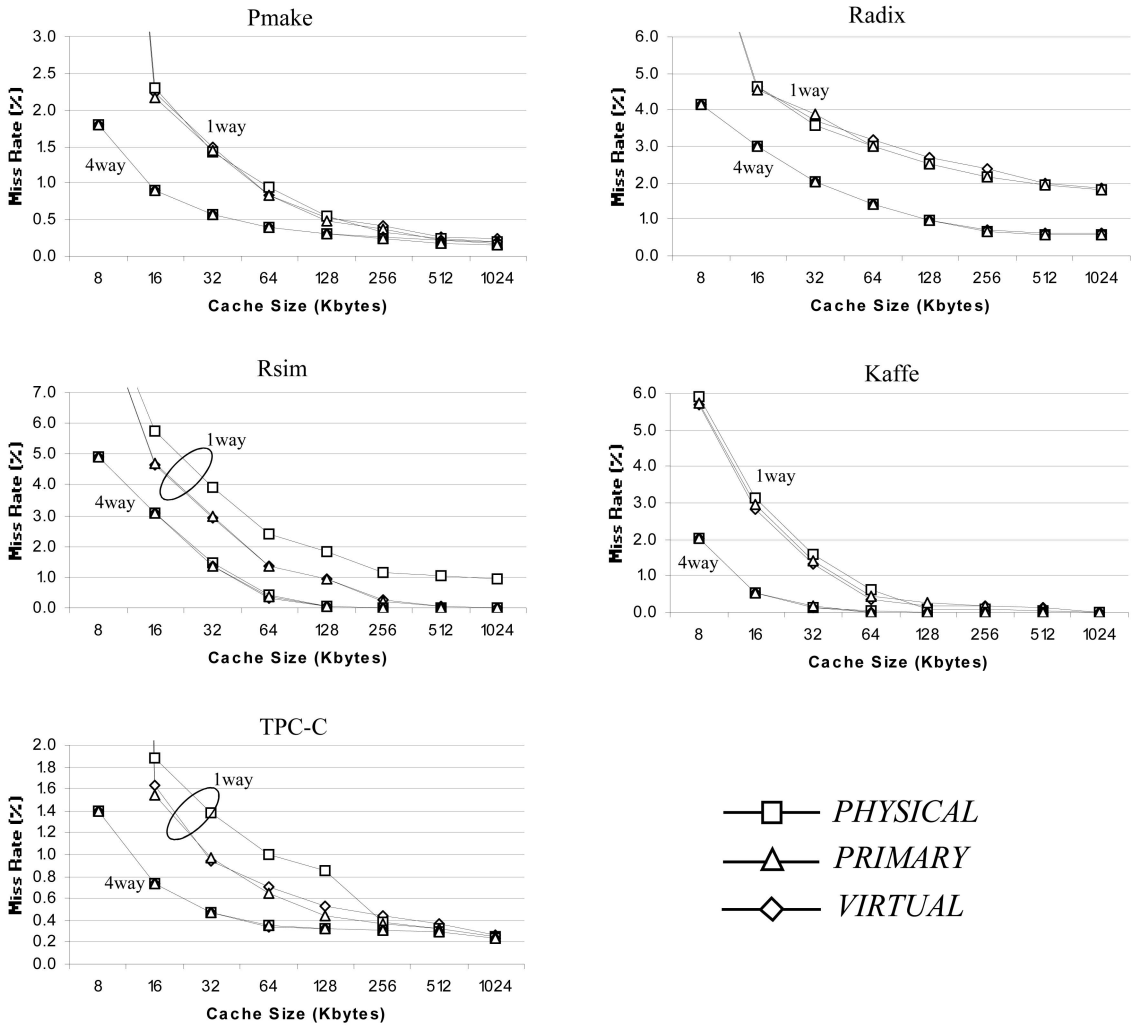


Fig. 9. Total miss rate of L1 caches.

indexings. Thus, the organization in which the SLB and the L1 cache are accessed in parallel may be preferable. However, for large L1 caches, it may become advantageous to access the SLB and the L1 cache serially, to avoid the effect of short misses while the access time to SLB becomes a smaller fraction of L1 cache access time. None of the systems we have evaluated have short misses outside the L1 cache.

5 Related Work

Over the years, many researchers in both software and hardware communities have advocated and proposed ideas supporting virtual caches. On the software side, Opal [7] was a novel approach to operating system (SASOS) running on a single global virtual address space shared by all procedures and all data. Synonyms do not exist in an SASOS. However, the engineering community was clearly not ready for such radical change.

Talluri et al. [29] and Romer et al. [27] looked at using superpages to increase TLB coverage without enlarging the TLB. They demonstrated that superpages dramatically cut the TLB overhead by mapping physical memories in big chunk. In [27], superpages are constructed dynamically by promoting small pages to a large page. The promotion itself requires copying physical pages, updating kernel data structures, and TLB shootdowns. This operation is very costly. The decision of when and how to promote superpages requires significant hardware and software efforts and usually increases the data set size of applications. It is unclear how these superpage schemes interact with other memory allocation issues, especially in NUMA-oriented memory system and in page coloring schemes for cache friendly optimizations [2].

Impulse [28] attempts to increase TLB coverage by backing up superpages with shadow physical memory. A superpage can be constructed by mapping to contiguous shadow physical pages, which will be translated into noncontiguous real physical pages by the memory controller. Although this can boost performance for particular applications, it is not a general solution for superpages because shadow memory is limited and has to be managed like physical memory.

Qiu and Dubois [26] looked at the locations in the memory hierarchy where virtual-to-physical address translations can be done. In their V-COMA architecture, virtual memory management is combined with the cache coherence protocol and distributed among processing nodes. They advocated the virtualization of the memory hierarchy

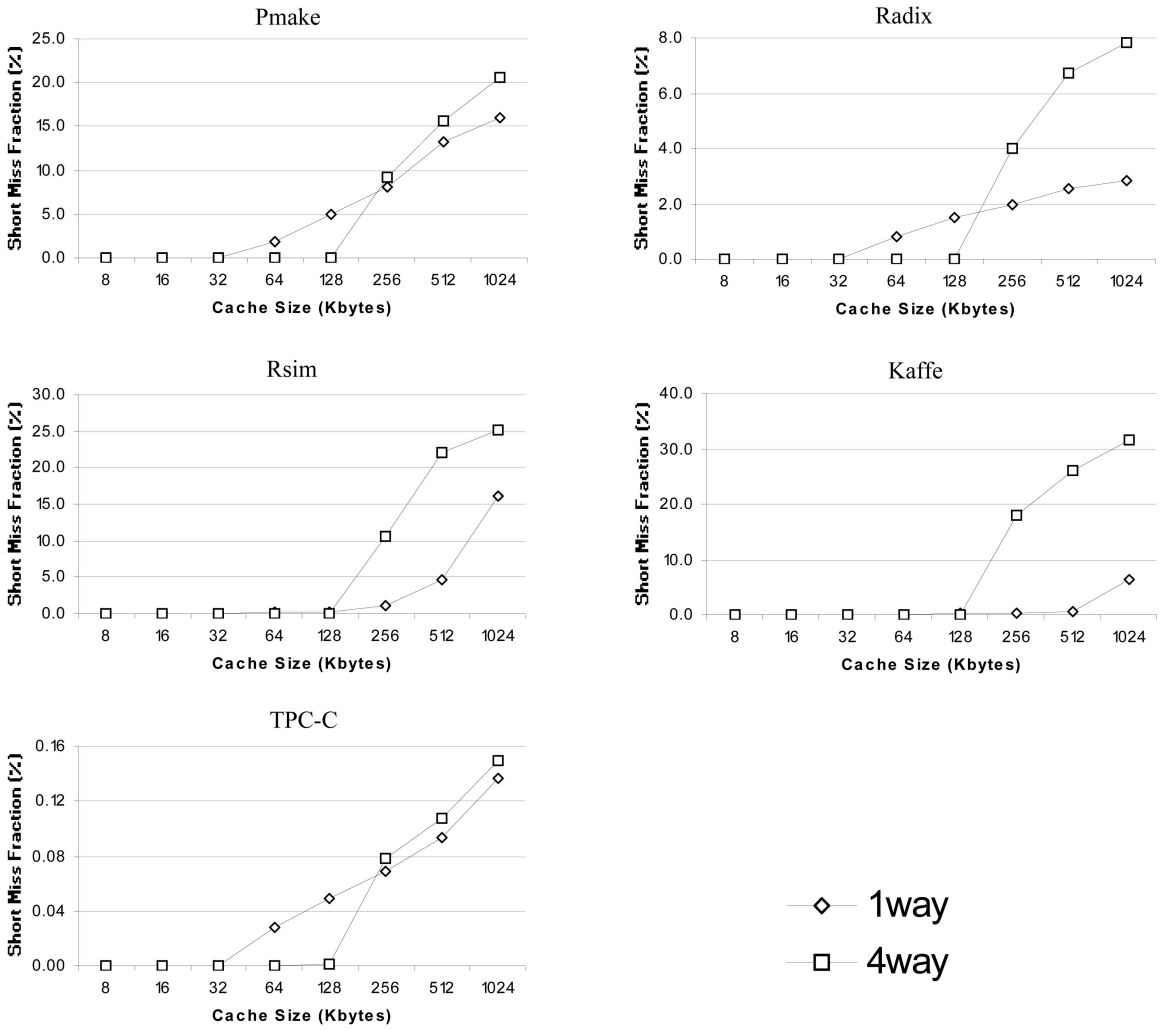


Fig. 10. Fraction of short misses in *VIRTUAL*.

in order to simplify memory systems and move functions to the memory side [24]. The SLB is a viable approach to achieve these goals.

Teller [30] addressed the scalability issue of maintaining TLB consistency in large-scale multiprocessors. In particular, she proposed a memory-based TLB scheme in the context of UMA architecture. She demonstrated that TLB consistency scales poorly in large-scale multiprocessors and moving the TLB to memory can radically solve the problem.

Although virtual caches have been the topic of many research papers, there are very few quantitative analysis of virtual cache performance. Agarwal [1] analyzed virtual address cache performance using traces from VAX. Wu et al. [34] evaluated different virtual cache types using an IBM System/370 trace. Although some of these observations are similar to the ones in this paper, the quantitative results are hard to compare because of the different schemes adopted for virtual caches and of the different workloads and operating systems used in the experiments. Lynch [20] observed that physical cache performance varies for each run depending on the allocation of pages by the operating system, while, on the other hand, the performance of virtual caches is not sensitive to these implementation decisions. We also observe that the miss rate of virtual caches is more robust than that of physical caches, especially for caches with low associativity.

6 Conclusions

In this paper, we have taken a new look at virtual caches. Moving the virtual-to-physical translation down the cache hierarchy presents some technical challenges. We show a simple and effective solution to the synonym problem, which is one of the major technical problems of virtual caches. In this solution, one primary virtual address is used to name each virtual page uniquely in the processor and in the memory hierarchy and an SLB translates synonyms into primary virtual addresses dynamically. We have given reasons why a very small SLB is effective and we have shown performance data to back this claim up. Using actual applications from different domains, we have shown that virtual cache flushing is very limited in practice and has insignificant impact on performance. We have presented extensive performance results comparing the miss rate behavior of physical and virtual caches. Virtual caches have better miss rates than physical caches and the solution using a small first-level SLB in front of the caches avoids short misses in larger caches, while safeguarding the benefits of temporal and spatial localities in the virtual space.

REFERENCES

[1] A. Agarwal, *Analysis of Cache Performance for Operating System and Multiprogramming.* Kluwer Academic, 1989.

[2] E. Bugnion et al., "Compiler-Directed Page Coloring for Multiprocessors," *Proc. Seventh Conf. Architecture Support for Programming Languages and Operating Systems (APLOS '96),* Oct. 1996.

[3] M.J. Bach, *The Design of the UNIX Operating System.* Prentice Hall, 1986.

[4] M. Cekleov and M. Dubois, "Virtual-Address Caches, Part 1: Problems and Solutions in Uniprocessors," *IEEE Micro,* pp. 64-71, Sept./Oct. 1997.

[5] M. Cekleov and M. Dubois, "Virtual-Address Caches, Part 2: Multiprocessor Issues," *IEEE Micro,* pp. 69-74, Nov./Dec. 1997.

[6] C. Chao, M. Machey, and B. Sears, "Mach on a Virtually Addressed Cache Architecture," *Proc. First Mach USENIX Workshop,* pp. 31-51, Oct. 1991.

[7] J. Chase, H. Levy, and M. Feeley, "Sharing and Protection in a Single-Address-Space Operating System," *ACM Trans. Computer Systems,* pp. 271-307, Nov. 1994.

[8] D. Cheriton, G. Slavenburg, and P. Boyle, "Software-Controlled Caches in the VMP Multiprocessor," *Proc. 13th Ann. Int'l Symp. Computer Architecture (ISCA '86),* pp. 366-375, 1986.

[9] Y. Chou, L. Spracklen, and S.G. Abraham, "Store Memory-Level Parallelism Optimizations for Commercial Applications," *Proc. 38th Ann. IEEE/ACM Int'l Symp. Microarchitecture (MICRO),* 2005.

[10] D.W. Clark and J.S. Emer, "Performance of the VAX-11/780 Translation Buffer: Simulation and Measurement," *ACM Trans. Computer Systems,* vol. 3, no. 1, Feb. 1985.

[11] J.R. Goodman, "Coherency for Multiprocessor Virtual Address Caches," *Proc. Second Conf. Architecture Support for Programming Languages and Operating Systems (ASPLOS),* 1987.

[12] L. Gwennap, "Alpha 21364 to Ease Memory Bottleneck," microprocessor report, Oct. 1998.

[13] S.A. Herrod, "Using Complete Machine Simulation to Understand Computer System Behavior," PhD thesis, Stanford Univ., Feb. 1998.

[14] G. Hinton et al., "The Microarchitecture of the Pentium 4 Processor," *Intel Technology J.,* pp. 1-12, Q1, 2001.

[15] B. Jacob and T. Mudge, "Software-Managed Address Translation," *Proc. Third Int'l Symp. High Performance Computer Architecture (HPCA '97),* Feb. 1997.

[16] R. Kalla, B. Sinharoy, and J. Tendler, "IBM Power5 Chip: A Dual-Core Multithreaded Processor," *IEEE Micro,* pp. 41-47, Mar./Apr. 2004.

[17] E.J. Koldinger, J.S. Chase, and S.J. Eggers, "Architecture Support for Single Address Space Operating System," *Proc. Fifth Conf. Architecture Support for Programming Languages and Operating Systems (ASPLOS '92),* pp. 175-186, Oct. 1992.

[18] P. Kongetira, K. Aingaran, and K. Olukotun, "Niagara: A 32-Way Multithreaded Sparc Processor," *IEEE Micro,* pp. 21-29, Mar./Apr. 2005.

[19] J.P. Laudon and D. Lenoski, "The SGI Origin: A CC-NUMA Highly Scalable Server," *Proc. 24th Ann. Int'l Symp. Computer Architecture (ISCA),* 1997.

[20] W. Lynch, "The Interaction of Virtual Memory and Cache Memory," PhD thesis, Technical Report CSL-TR-93-587, Stanford Univ., 1993.

[21] *The PowerPC Architecture: A Specification for a New Family of RISC Processors,* C. May, E. Silha, R. Simpson, and H. Warren, eds. Morgan Kaufmann, 1994.

[22] V. Pai, P. Ranganathan, and S. Adve, "RSIM Reference Manual," Technical Report 9705, Dept. of Electrical and Computer Eng., Rice Univ., Aug. 1997.

[23] I. Park et al., "Reducing Design Complexity of the Load/Store Queue," *Proc. 36th Ann. Int'l Symp. Microarchitectures (MICRO-36 '03),* pp. 411-422, 2003.

[24] X. Qiu and M. Dubois, "Towards Virtually-Addressed Memory Hierarchies," *Proc. Seventh Int'l Symp. High Performance Computer Architecture (HPCA '01),* pp. 51-62, Jan. 2001.

[25] X. Qiu and M. Dubois, "Tolerating Late Memory Traps in Dynamically-Scheduled Processors," *IEEE Trans. Computers,* vol. 53, no. 6, pp. 732-743, June 2004.

[26] X. Qiu and M. Dubois, "Moving Address Translation Closer to Memory in Distributed Shared Memory Multiprocessors," *IEEE Trans. Parallel and Distributed Systems,* vol. 16, no. 7, pp. 612-623, July 2005.

[27] T.H. Romer, W.H. Ohlrich, and A.R. Karlin, "Reducing TLB and Memory Overhead Using Online Promotion," *Proc. 22nd Ann. Int'l Symp. Computer Architecture (ISCA '95),* pp. 176-187, 1995.

[28] M. Swanson, L. Stoller, and J. Carter, "Increasing TLB Reach Using Superpages Backed by Shadow Memory," *Proc. 25th Ann. Int'l Symp. Computer Architecture (ISCA '98),* pp. 204-213, 1998.

[29] M. Talluri, S. Kong, M.D. Hill, and D.A. Patterson, "Tradeoffs in Supporting Two Page Sizes," *Proc. 19th Ann. Int'l Symp. Computer Architecture (ISCA '92),* pp. 415-424, May 1992.

[30] P. Teller, "Translation Lookaside Buffer Consistency," *Computer,* vol. 23, no. 6, pp. 26-36, June 1990.

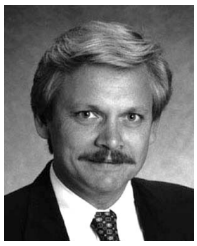
[31] M. Tremblay and J.M. O'Connor, "Ultrasparc I: A Four-Issue Processor Supporting Multimedia," *IEEE Micro,* pp. 42-50, Apr. 1996.

[32] W.H. Wang, J.-L. Baer, and H.M. Levy, "Organization and Performance of a Two-Level Virtual-Real Cache Hierarchy," *Proc. 16th Ann. Int'l Symp. Computer Architecture (ISCA '89),* pp. 140-148, June 1989.

[33] D. Wood, S. Eggers, G. Gibson, M. Hill, and J. Pendleton, "An In-Cache Address Translation Mechanism," *Proc. 13th Ann. Int'l Symp. Computer Architecture (ISCA '86),* pp. 358-365, Jan. 1986.

[34] C.E. Wu, Y. Hsu, and Y.-H. Liu, "A Quantitative Evaluation of Cache Types for High-Performance Computer Systems," *IEEE Trans. Computers,* vol. 42, no. 10, pp. 1154-1162, Oct. 1993.

Xiaogang Qiu received the PhD degree in computer engineering from the University of Southern California. He is currently a hardware engineer at NVIDIA. Before joining NVIDIA in 2006, he was a staff engineer at Sun Microsystems. His research interests include computer architecture, design and verification of microprocessors, and parallel systems.



Michel Dubois received the degree in electrical engineering from the Faculte Polytechnique de Mons, Mons, Belgium, the MS degree in electrical engineering from the University of Minnesota, and the PhD degree in electrical engineering from Purdue University. He is a professor of computer engineering in the Department of Electrical Engineering-Systems, University of Southern California (USC). Before joining USC in 1984, he was a research engineer at the Central Research Laboratory, Thomson-CSF, Orsay, France. His main research interests are in computer architecture and parallel processing. He has published more than 150 technical papers on computer architectures and algorithms. He is well known for his early work on cache coherence and memory consistency models. From 1993 to 2001, he led the RPM Project, a project funded by the US National Science Foundation. RPM stands for "Rapid Prototyping engine for Multiprocessors" and is a hardware platform used to implement multiprocessor systems with widely different multiprocessor architectures. In this project, a multiprocessor machine was built with off-the-shelf components and FPGAs. Multiprocessor prototypes could be developed by programming the FPGAs. His current research interests are in CMPs and in the impact of technological trends on such microarchitectures. He has edited two books, one on multiprocessor caches and one on scalable shared memory multiprocessors. He has been on numerous technical committees of leading conferences, as well as program chair and general chair of several conferences. He is currently an area editor of the *Journal of Parallel and Distributed Processing*. He is a fellow of the ACM and the IEEE and a member of the IEEE Computer Society.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.