The Case of Virtually-Addressed Memory Hierarchies

Xiaogang Qiu and Michel Dubois

CENG 00-03

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California 90089-2562
(213-740-4475)
2000

The Case for Virtually-Addressed Memory Hierarchies

Abstract

Currently cache hierarchies are indexed in parallel with a TLB but their tags are part of the physical address so that the memory hierarchy is physically addressed. Recently, it has been argued that this design faces mounting problems as more concurrency is exploited in the processor core and as the memory demand of emerging applications is growing fast. The traditional TLB is fast becoming a bottleneck in the processor and its hit rate can be disastrous for data-intensive applications or scientific applications without much locality.

This paper introduces new ideas to enable the use of virtual addresses throughout the memory hierarchy, thus removing the TLB from the processor and associating it with memory where it scales much better. The major idea is the replacement of the TLB by a Synonym Lookaside Buffer (SLB), which can remain small because its size depends on the number of synonyms, not the size of the application or of the physical memory. We also show performance data on various applications, from scientific computing to database to Java virtual machines. These evaluations show that virtually addressed memory hierarchies overall have better performance behavior than physically-addressed memory hierarchies. Moreover, we also show how virtually addressed memory hierarchies enable natural, scalable multiprocessor extensions, as well as computing-in-memory in the context of general-purpose computers.

1. Introduction

Along with increasing clock rates, instruction level parallelism (ILP) continually boosts processor performance at a fast pace and is currently the major commercial approach to improve processor architecture. The memory hierarchy of ILP processors must satisfy multiple memory accesses in every cycle at a rate currently approaching 1GHz. It must sustain high speed and high bandwidth memory accesses in the face of the growing disparity between processor execution rate and main memory access time, and of the growing memory demand of emerging applications. One additional complication is that, in the process of accessing memory, the virtual address issued by the processor must at some point be translated into a physical address in main memory.

This dynamic translation between virtual and physical addresses is typically supported by a translation lookaside buffer (TLB). Typically the virtual address indexes the first level cache, and address translation and cache access are performed in parallel, so that the memory hierarchy is accessed with physical addresses throughout[41][37]. Such a cache has been called a "virtually indexed and physically tagged" cache (V/P cache for short) since the cache entry is tagged with bits from the physical address.

The TLB of a V/P cache 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. To satisfy the access constraints of a V/P cache, the latency and bandwidth requirements of the TLB must scale up with the clock rate and instruction level parallelism[3]. The TLB

inside the processor core does not scale with growing application sizes and physical memory sizes or with the number of processors. In a multiprocessors, some TLB entries are replicated, wasting TLB space but, more importantly, creating a consistency problem[36]. Maintaining TLB consistency is very expensive and does not scale well.

Translation misses in the TLB trigger expensive "walks" through levels of page tables located in main memory. It has been shown in the past[14] that the execution overhead due to TLB handling was about 5% - 10% of the total execution time. However, current technology trends and the growing working set demands of applications are putting more pressure on the address translation hardware. Some studies have shown that the TLB service time alone can consume up to 50% of the user execution time in some workloads[29][35].

In the past, the strong motivation to remove some of the TLB overhead drove several companies to adopt virtually addressed caches (i.e. caches indexed and tagged with virtual addresses a.k.a V/V caches) and inspired some very ambitious academic research projects both software and hardware such as the Opal operating system project [11] at the University of Washington and the Spur project [39] at Berkeley. When the cache is virtually indexed and tagged, most memory accesses are completed without TLB involvement and the actual address translation, performed only when it is needed, can be done in different locations in the memory hierarchy[38], and can be implemented in various ways, such as in-cache translation[39] or even by software[22]. Putting the TLB after the virtual address cache hierarchy can dramatically reduce the number of TLB misses because of the filtering effect of the caches and also because the TLB may be much larger. When the TLBs are shared at the main memory in multiprocessors TLB misses can become insignificant because of sharing and prefetching effects[27].

The well-known "curse" of virtually-addressed caches is the synonym problem: Because of sharing and other factors, multiple virtual addresses may map to the same physical address. This complicates cache management because of the lack of a system-wide unique identifier for memory locations. Unique identifiers are also required to disambiguate memory addresses in the load/store unit of ILP processors.

One approach to create unique identifiers is segmentation[8][22][25][39]. Segment registers managed by software translate user virtual addresses into global virtual addresses before accessing the cache. Segments have fixed size and very coarse granularity. The operating system must support the global virtual address and the segment structures and the porting effort is large, requiring ad-hoc patches for various situations involving synonyms[8].

This paper introduces new ideas to enable the use of virtual addresses throughout the memory hierarchy. The major idea is to replace the TLB in the processor with a Synonym Lookaside Buffer (SLB), which dynamically translates synonyms into unique identifiers. The SLB is much more flexible than segment registers and impacts the software very little. An effective SLB is very small and scales well. It can be accessed before or in parallel with the first level cache.

Using contemporary workloads running on a real operating system (SGI Irix 5.3), we evaluate the performance of virtually addressed caches. We show that the SLB idea eases dramatically the scalability problems associated with TLBs. We also show how virtually addressed memory hierarchies create opportunities for natural and scalable multiprocessor extensions, as well as for the integration of computing-in-memory in the context of general-purpose computers.

The rest of the paper is structured as follows. Section 2 is a brief summary of common usage of synonyms in current computer systems. In section 3, we discuss dynamic address translation and virtual address caches, and we introduce the synonym lookaside buffer to enable V/V caches. In section 4, we evaluate the impact on miss rate of virtually addressed caches. Further discussion on V/V cache hierarchies is provided in section 5, where we show some novel architectures enabled by V/V cache hierarchies. After a brief overview of some related work, we conclude the paper in section 7.

2. Synonyms

Multiple virtual addresses mapped to the same physical address are called *synonyms*. Synonyms are very convenient to kernel and user software in many situations.

Synonyms are often used to implement shared memory semantics across different user virtual address spaces. Read-only segments such as libraries and text segments are widely shared and the kernel is usually shared among all private virtual address spaces at a fixed location. User processes can also request shared memory segments to facilitate communication with other processes. Users may even define synonyms within the same process for convenience.

Synonyms are also critical for the efficiency of various memory operations. Copy-on-write avoids unnecessary memory copies and reduces the consumption of physical memory. In this case, different virtual addresses share the same physical memory location for as long as the content is not modified.

Message passing layers exploit synonyms to avoid physical memory copies. If process 1 sends a message page at virtual address V1 to process 2 at virtual address V2, the system can simply remap the physical page P from V1 to V2. The V1-P mapping may remain valid as copy-on-write or be destroyed depending on the message semantics. As a special case, the operating system kernel usually buffers I/O transfers in kernel space and remaps them to user address space when needed.

It is also very common that V1 and V2 share the *resource* of physical page P rather than its content. In demand-paging systems, V1 may be swapped out and the physical page P is freed and allocated to another virtual page V2. In this case, V1 and V2 do not have any logical connection.

3. Virtual Address Caches

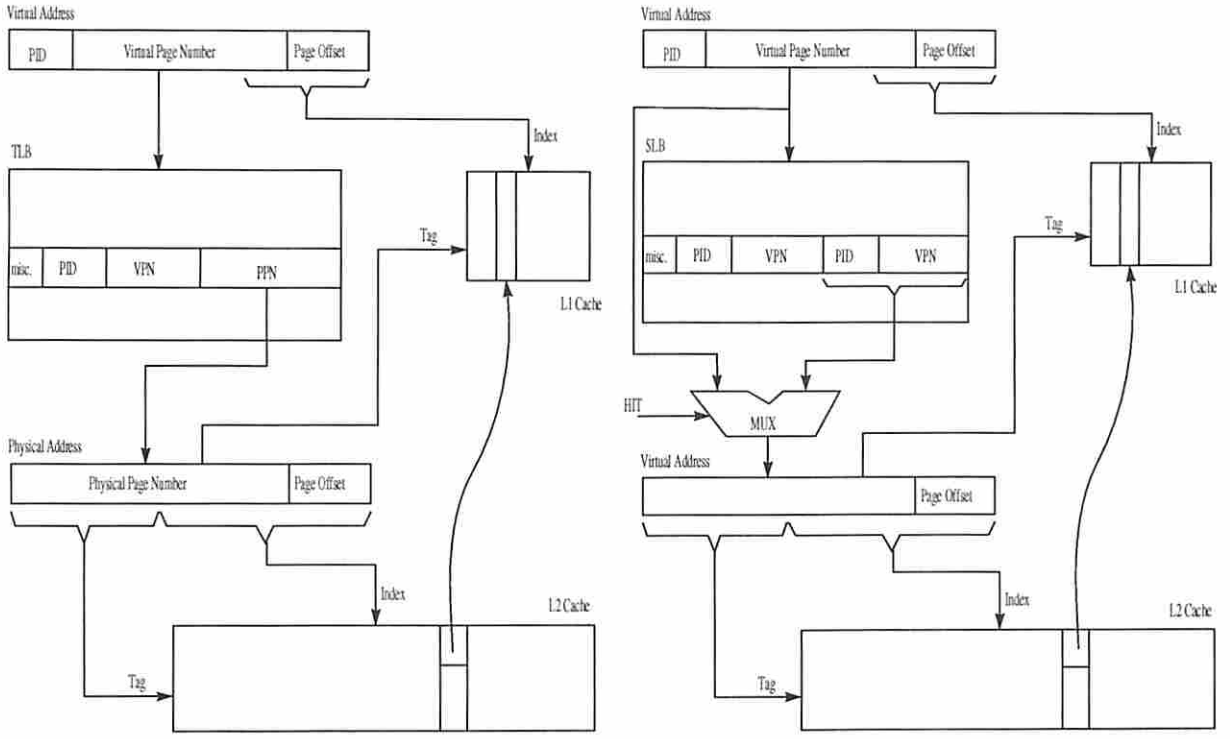
3.1. Translation Lookaside Buffer (TLB) and V/P Cache

Figure 1 (a) shows a typical architecture for a V/P cache and a TLB accessed in parallel. Cache blocks are named by their physical address, while their virtual address is used as a hint or hashing function to quickly find their location. After a cache set is indexed by the virtual address, all tags in that set are compared with the physical page number translated by the TLB. A miss in that set does not guarantee that the data block is not present in the cache. The block may reside in any set within the superset. The superset is made of all the cache sets whose set indexes share the same (p-b) least significant bits, where p is the page offset bits and b is the block offset bits.

To detect synonyms in the superset a backpointer[38] is stored in the L2 physical-address cache for every valid block in the L1 cache. Each pointer points to the frame in L1 where the block is valid. A synonym is detected whenever an L1 cache miss hits in the L2 cache and the backpointer is valid in L2. In this case, the block is moved to the new cache set in L1 and retagged, and the backpointer in L2 is updated. This is called a *short miss*.

When the TLB misses the processor is often trapped and the TLB is filled by the trap handler before resuming the access. The miss rate of the TLB is primarily determined by its reach or coverage, which is the aggregate virtual memory area mapped by all its entries. Superpages improve TLB coverage[35]. However, if one TLB entry maps a superpage, all the physical pages inside the superpage must be allocated in contiguous frames in physical memory. The size of a superpage is also hard to figure out at the time when the virtual address is first allocated. Page placement in multiprocessors and coloring of pages to

optimize cache behavior[4] may conflict with the use of superpages.



(a) **TLB and V/P Cache** (b) **SLB and V/V Cache**

Figure 1 Parallel Access to a V/P and a V/V cache

3.2. Synonym Lookaside Buffer and V/V cache

To solve the synonym problem, we must associate a unique identifier to each page. To do this, we split the entire virtual address space in two sets: the main address set and the synonym set. The synonym set contains all the virtual addresses that are synonyms to some virtual addresses in the main address set. The main address set contains the rest of the virtual addresses and include virtual addresses with no synonyms and one virtual address selected in each group of synonyms. The main addresses act as unique identifiers for all pages.

An SLB (synonym lookaside buffer) dynamically translates virtual addresses in the synonym set into their corresponding shared virtual addresses in the main address set. Just as a traditional TLB, the SLB can be accessed in parallel with the first level cache. Figure 1 (b) shows the parallel access to cache and SLB. Backpointers in the second-level cache take care of short misses as in the V/P cache. The main difference between Figure 1 (a) and Figure 1 (b) is that each SLB entry contains the main address instead of the physical address.

As we will see, a small SLB of 8 or 16 entries (similar to the number of segment registers in segmented architectures) is sufficient. It is therefore possible to access the SLB on the access path to the first-level cache. To be protected against programs with bad behavior, we can imagine adding a second level SLB accessed in parallel with the L1 cache. In case a synonym misses in the first-level SLB but hits in the second-level SLB, the cache access is aborted and reissued using the main address translated in the second-level SLB. When the SLB is accessed before the cache, L2 backpointers and short misses in L1 are eliminated.

A virtual address missing in the SLB may be either a synonym or a main address. Whether or not it misses in the SLB the memory access proceeds. If it is a synonym, it will miss in the cache hierarchy and will eventually be detected at the point where the physical address is needed, trapping the processor which then fills the SLB. The access is then retried using the main address.

Just as in the traditional TLB, the SLB entry must include access right bits corresponding to the synonym address. The cache hierarchy also contains protection bits associated with the main address and a synonym access is checked for protection twice, once in the SLB and one in the cache hierarchy. Thus the main address must have equal or less access restrictions than all of its synonyms.

Most current operating systems implement demand paging virtual memory. On top of the machine-dependent layer, there is usually a machine-independent layer organizing virtual memory in logical segments. Sharing of synonyms is decided in the logical layer independently of physical paging. Since sharing is already managed in the operating system in the logical layer, only some minor changes are needed in current operating systems to use an SLB.

3.3. Comparing TLB and SLB

Although the SLB has practically the same hardware organization as a traditional TLB, its scalability is much better. Some issues specific to SLBs must also be solved.

3.3.1. Coverage

The coverage of the SLB is much better than that of a TLB and scales with application and main memory sizes, for two major reasons.

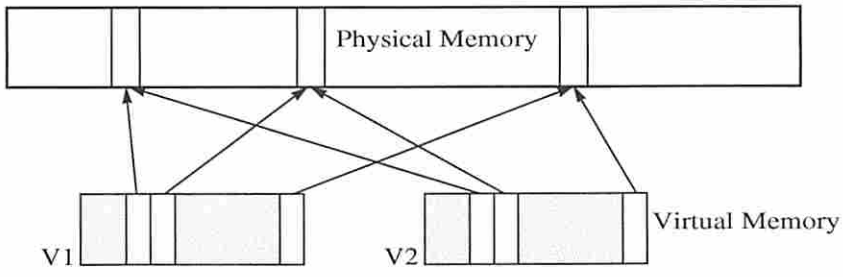


Figure 2 A Synonym Example

First, synonyms can be allocated in very coarse granularity and can cover a large number of pages. This has the same affect on the coverage as superpages in the TLB, but the major difference is that contiguous physical memory pages are not allocated. For example, in Figure 2, virtual address segments V1 and V2 are synonyms. The physical page mapping is not contiguous and most virtual pages in the synonym segments are not even mapped to physical space. Nevertheless the synonyms can be represented in one single SLB entry. For any given process, the consumption of SLB entries depends on the synonym usage instead of on the memory allocation. In general, increasing data set size does not create more synonyms -- the synonyms simply cover larger memory areas.

Second, unlike the traditional TLB which holds translations covering every single virtual page accessed by the processor, the SLB is only responsible for a subset of these pages, i.e. the synonyms. The main address set does not use the SLB and, with some cooperation from the operating system, we expect that it will cover the best part of the whole virtual address space.

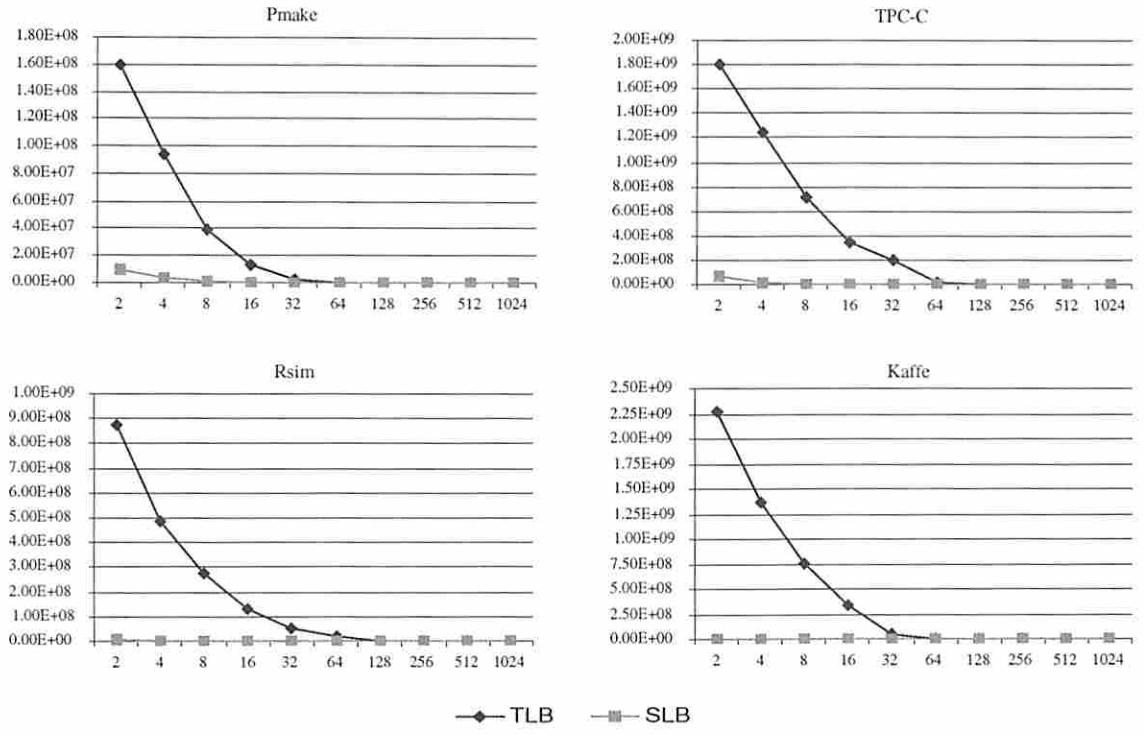


Figure 3 Total Number of Misses as a Function of the Number of Entries in a TLB and in an SLB for Four Benchmarks

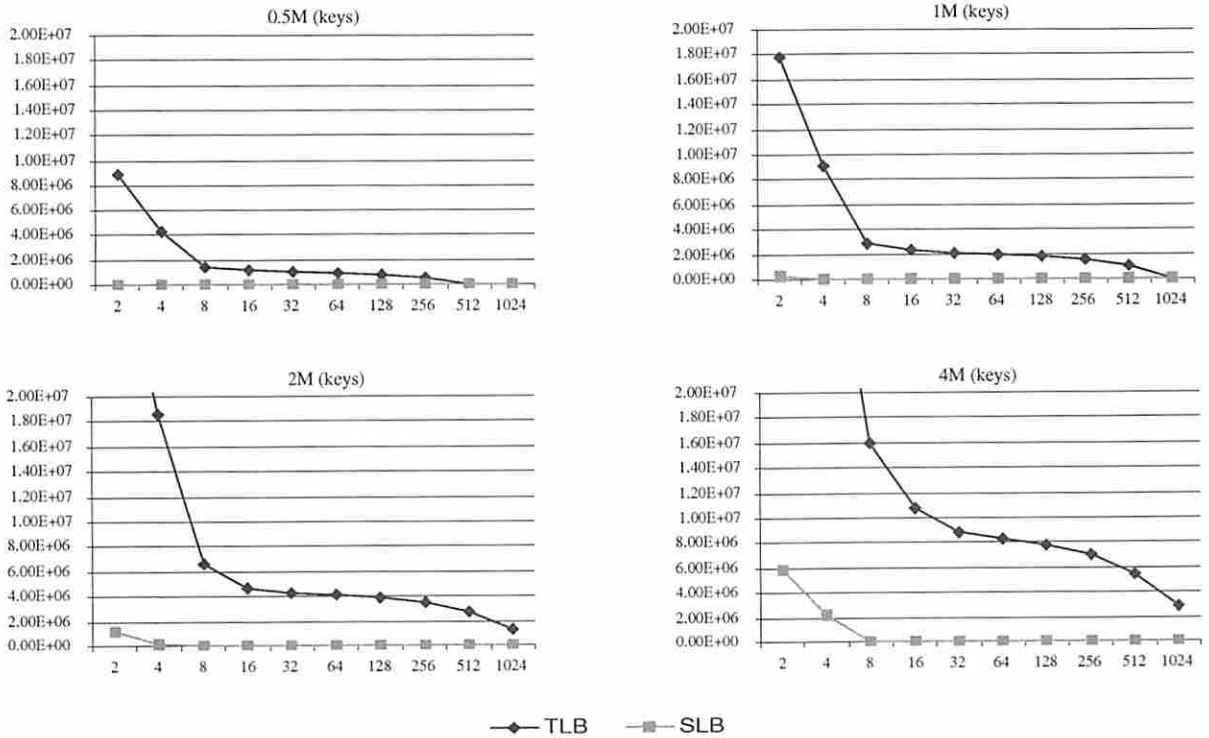


Figure 4 Total Number of Misses as a Function of the Number of Entries in a TLB and in an SLB for Radix with Different Data Set Size

To demonstrate the better coverage of an SLB as compared to a TLB, we have run some simulation experiments. (The five benchmarks and the experimental set-up are described in section 4.) Figure 3 compares the number of TLB and SLB misses in four benchmarks as a function of the number of entries. At the low end of these graphs, the number of TLB misses overwhelms the number of SLB misses by orders of magnitudes.

Figure 4 shows the miss rate comparison for a Splash benchmark, Radix, with 4 different data set sizes. We see from these graphs that the miss curve for the TLB takes off as the data set size increases. An SLB of size 8 or bigger has negligible amount of misses,

3.3.2. Page Mapping Changes and TLB/SLB Shootdowns

Another important issue for V/P and V/V caches is the demapping and remapping of pages. In a V/P cache, every time a virtual address is remapped to a different physical address the TLB must be updated, which triggers TLB shootdowns in multiprocessors. In V/V caches, the cache hierarchy must sometimes be flushed.

In contrast with a TLB, not every virtual-to-physical address mapping changes need to update the SLB. Paging activities (swap-in and swap-out) as well as page migrations do not affect the SLB nor the V/V caches. The only mapping changes that affect the SLB or the V/V caches is when the content of a *virtual* page is actually changed through the re-mapping. For example when the process image is overlapped through a EXEC system call, or when a process is terminated and its virtual space is reclaimed by another process, the SLB must be updated for all addresses in the synonym set and the V/V cache must be flushed for all addresses in the main set.

Table 1 shows the activity associated with virtual-physical page remaps in our five benchmarks. In our simulation, we always flush either the SLB entry (synonym access) or the cache hierarchy (main address access) on a remap, even if this is not always needed. Our counts do not include the activity associated with reclaiming virtual address pages at the end of the execution.

Benchmarks	Remaps		SLB flush		L2-cache Flush (Blocks)	
	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

Table 1: Remaps and Flushes

In the table, "remaps" is the number of virtual-physical page remaps in the execution; "SLB flush" is the number of SLB flushes. The rest of the remaps flush the cache hierarchy but not the SLB. "L2-cache flush (blocks)" is the total number of L2 cache blocks flushed by all these remaps.

These numbers are very small and practically negligible. The majority of the remaps are for synonyms in kernel space, which explains the very low number of L2 cache blocks that are flushed. User address remap is very rare even for Pmake, which is a multiprogramming workload.

3.3.3. Some Issues Specific to SLB

Memory Traps. The SLB cannot help verify/resolve memory trap conditions for virtual addresses as the traditional TLB does. Memory traps such as page faults may be triggered deep in the memory hier-

archy. The performance impact of such late memory traps was addressed in [28], where it was shown that the latency of late memory traps can be largely tolerated by ILP processors.

Synonym Coherence. In the rare instance when the same data is accessed with different addresses at very fine granularity coherence may be violated. A store to an address in the synonym set may miss in the SLB and in the virtual cache hierarchy even though its main address is allocated in the cache hierarchy. In this case, a following load accessing the same data with the main address hits in the V/V cache before the trap for the store is detected, thus bypassing the previous synonym access. Coherence is violated.

This is not an issue for processor architecture implementing sequential consistency such as the MIPS or PA-RISC architectures because memory operations are retired one after another, after they are completed.

However for weaker memory models[15] the coherence issue is real because stores are retired as soon as they reach the local store buffer. We now propose some solutions in the context of weak memory models.

Since, the main address has, in general, less restrictions than its synonyms, the main address for a writable synonym is also writable. This indicates that, in the coherence violation described above, there are at least two writable virtual addresses shared within the same virtual address space. The use of synonyms within the same address space is purely optional and can be avoided by simply using the same virtual address. Most synonyms are generated and managed by the operating system kernel without the involvement of user programs. The OS kernel should not generate two writable sharings for user programs.

A more flexible solution is to allocate a quota of SLB mappings (for example 8) for each virtual address space to map writable sharings within the process. As long as the number of user writable sharings does not exceed the quota, coherence violation is prevented because suspicious stores always hit in SLB.

A general and totally flexible solution is consists in checking all stores before retiring the store instruction as is done in TLBs. If a store misses in the SLB it must traps the processor to fill the SLB. Load misses in SLB are still issued to the cache hierarchy as usual. Although the SLB must now map writable main addresses, its coverage is still scalable --at least as scalable as segment registers[25]. The coverage of each SLB entry still scales well because it does not depend on the application or physical memory size.

Choosing the Main Address. Among all the virtual addresses sharing the same page, one single virtual address must be selected as the main address. The operating system should select the address with the longest lifetime as the main address, because the overhead associated with remapping a main address is much higher than remapping addresses in the synonym set. Allocating main addresses by first touch is both simple and effective because of the way the operating system fork processes. We have used first touch in our simulation experiments except for one special case, now described.

Lock Bit Optimization. The remap of physical addresses is widely used by the OS kernel to communicate with user processes and to optimize message passing in general. One common scenario is one in which 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. The current SLB scheme may generate excessive cache flushing for this very common case.

We propose an optimization for remap-based message-passing in general, especially for kernel buffer handling. Suppose V2 is a receiver user buffer expecting a message from V1 which is mapped to a physical page P. The current procedure is to demap V1-P and remap V2-P to pass the page. In this case V1 is the main address and its demapping is very costly.

To reduce the overhead, V2 is deemed the main address and V1 a synonym. In order to synchronize, V2 accesses must be locked out while 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 V/V caches. While the lock is set any access via V1 proceeds as usual. An access to the page via V2 is locked out by the page table.

4. Impact on the Miss Rate

Beside the overhead caused by SLB misses, SLB flushes and cache flushes one major concern associated with virtual memory hierarchies is the effect of cache addressing on miss rates.

Because the cache index function is different, conflict misses are affected. In a V/P cache accessed in parallel with a TLB or in a V/V cache accessed in parallel with an SLB the index is the virtual address. In a physical (P/P) cache the index is the physical address. Finally in a V/V cache accessed **after** an SLB, the index is the main virtual address. **From now on we will refer to these three systems as V/P cache, P/P cache and V/V cache respectively, knowing that V/P caches and V/V caches with parallel access to SLB have the same behavior.**

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[26] 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

Table 2: Benchmarks

The second issue is short misses in V/P caches. Short misses are just as expensive as regular misses hitting in the L2 cache. Their number grows with the cache size, contrary to conflict misses whose number tends to decrease with the cache size.

4.1. Methodology

Table 2 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 in our traces. These benchmarks represent different categories of applications and include a JAVA virtual machine, an OLTP commercial workload, a computer architecture simulation, a multiprogramming workload, and a compute intensive splash benchmark

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

With the trace, we simulate a single-processor system with a 2-level cache hierarchy. The L2 cache is fixed with a size of 1MB and is 2-way set associative. L1 cache size varies from 8KB to 1MB, and its organization is direct mapped or 4 way set associative. For V/P caches, the L2 cache is physically indexed with backpointers to L1. For V/V caches, both L1 and L2 are virtually indexed and tagged, and a 16-entry SLB is accessed *before* the L1 cache. The cache block size is 64 bytes for all caches. The LRU policy is used whenever replacement is needed. To be conservative, we always flush either the V/V cache (main address access) or the SLB (synonym access) on every page remapping, even if some of these

flushes could be avoided.

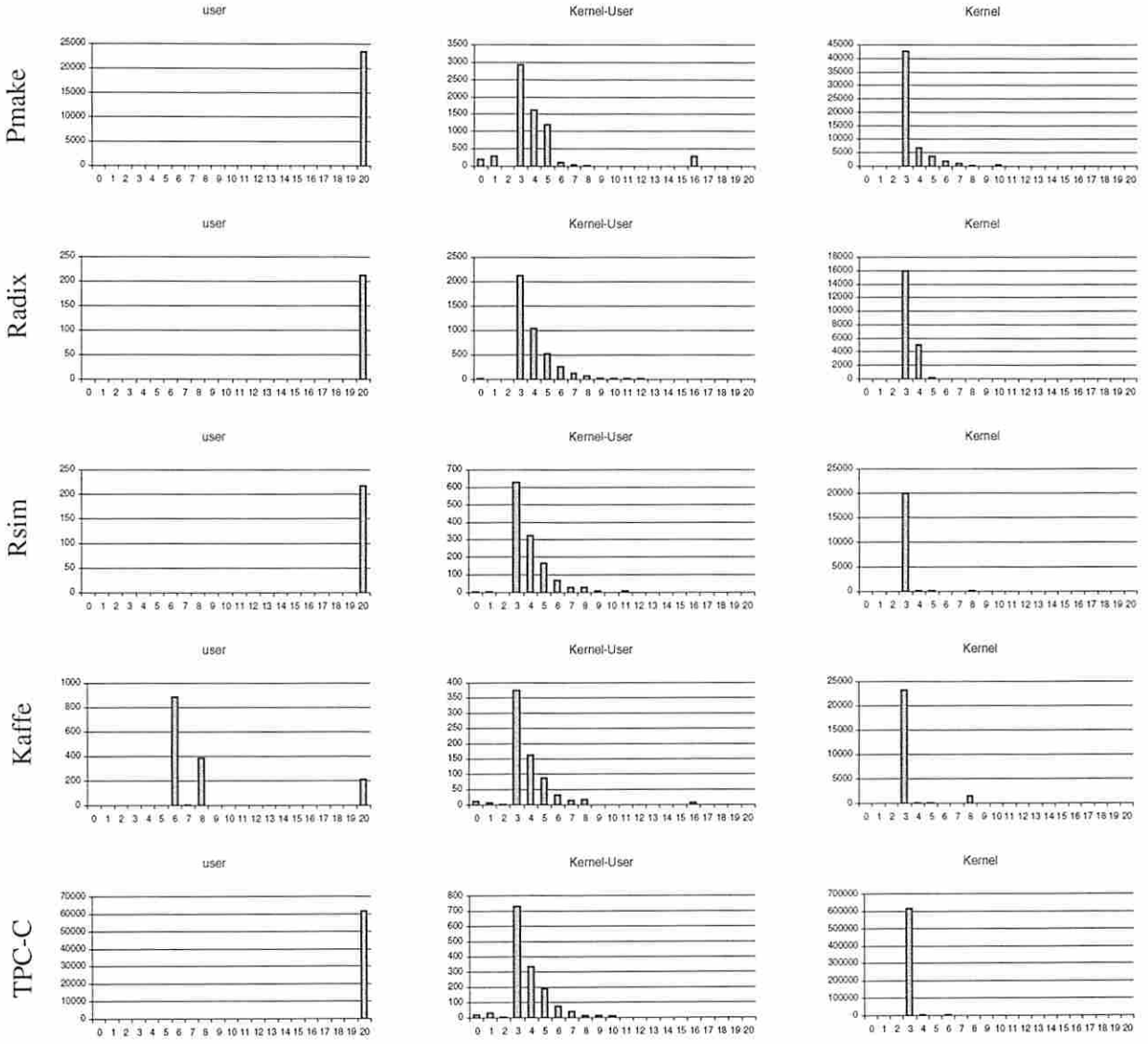


Figure 5 Histograms of Synonym Alternations

In order to support the SLB in the simulation, 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. To apply the lock-bit optimization in the SLB scheme, we identify kernel pages that should have a main address in user space in a preliminary run of the trace.

4.2. Synonym Alternation

We first look at the characteristic of synonyms in our benchmarks. We define and detect a synonym alternation as an instance of using a different virtual address in two consecutive accesses to the same physical page. Synonym alternation characterizes the dynamic interleaving of synonym accesses, which

affects cache miss rate.

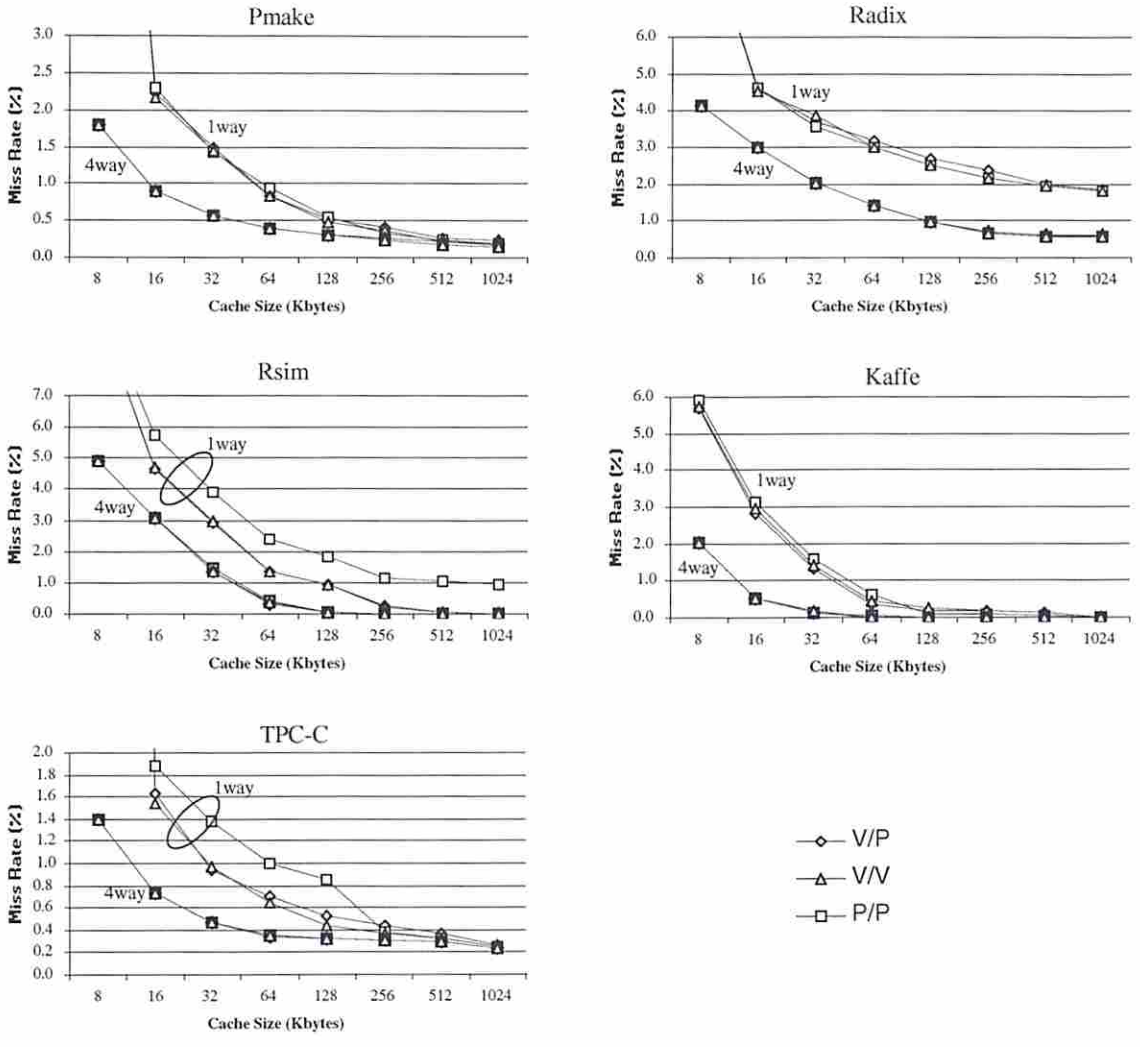


Figure 6 Absolute Miss Rate of L1 caches

For every synonym alternation, we define the degree of proximity between the two synonyms as the number of identical least significant bits in the two virtual page numbers. In the simulation, we collect a histogram of the degree of proximity between the two synonyms in each alternation.

To interpret these results keep in mind that the least significant bits of the virtual page number are used to index a V/P cache; so, if the V/P cache index uses n bits of the virtual page address, then all synonym alternations with a degree of proximity less than n create short misses.

For each benchmarks, we separate the synonym alternations into three categories. "User" and "kernel" count the cases that the two synonyms are both within user or kernel domain respectively. "Kernel-user" is for the case when the two synonyms are in different address spaces. Figure 5 shows the resulting histograms. The Y-axis is the number of synonym alternations, and the X-axis is the degree of

proximity between the two synonyms.

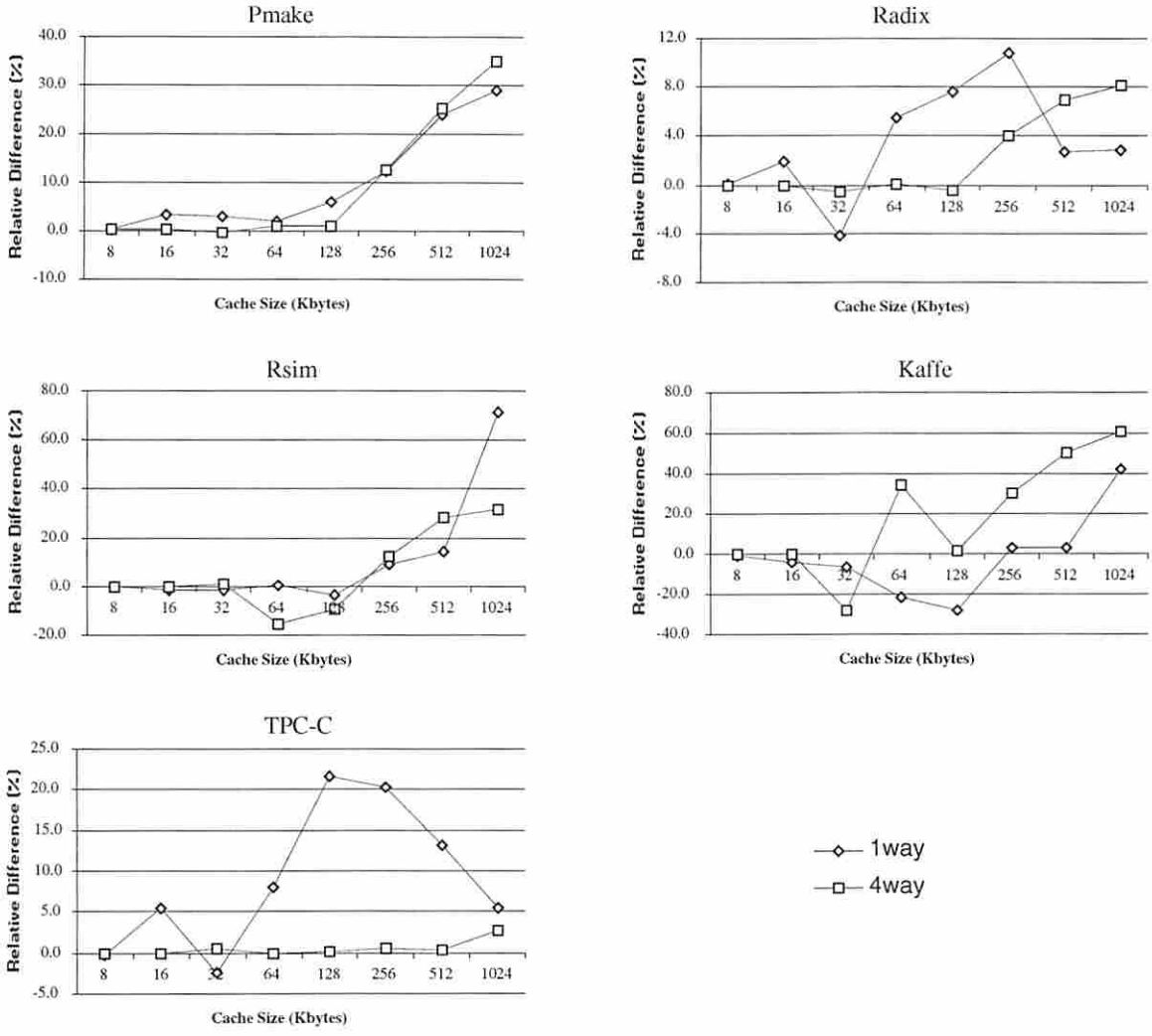


Figure 7 Relative Miss Rate Difference Between V/P and V/V L1 Caches

For 32-bit virtual addresses (without counting the PID), and a page size of 4 KB, the degree of proximity ranges between 0 and 20. The figure illustrates both the total number and the distribution of synonym alternations. Except for Kaffe, user synonyms all have the same virtual addresses and index into the same cache set in any V/P cache. Unlike the other benchmarks, Kaffe contains system calls making use of the memory mapping facilities provided by the kernel.

Unlike user synonyms kernel-user and kernel synonyms may generate a large number of short misses in V/P caches. Most kernel-user synonyms pass information across the kernel user boundary. If every block in the page is touched by both kernel and user, one synonym alternation may cause one short miss per block in the page. Within the kernel however, the reason for an alternation is not to transmit the content of a page but to share it. Each synonym usually addresses different data at each alternation and synonyms are interleaved in fine granularity. Thus kernel alternation creates fewer short misses as compared to kernel-user alternations.

4.3. Overall Miss Rates

Figure 6 shows the miss rates, including kernel and user, for direct-mapped, and 4-way set associative L1 caches. All curves show the same trends. The index to the cache only affects conflict misses, which are significant in a direct mapped cache.

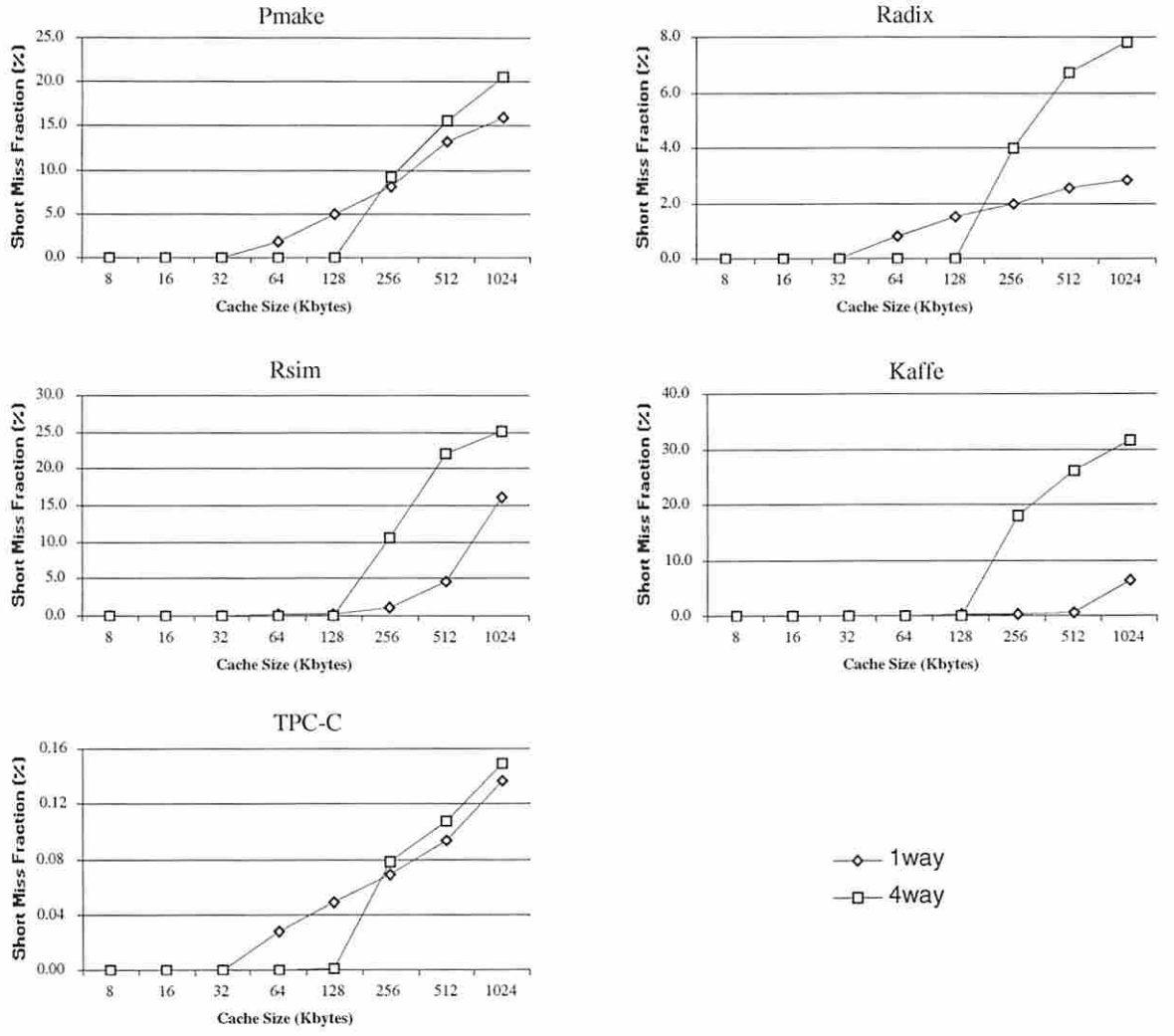


Figure 8 Fraction of Short Misses in V/P L1 Caches

A very big gap exists between P/P caches on one hand and V/P or V/V caches on the other in the case of direct mapped caches for RSIM and TPC-C benchmark. 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 1MB boundary. This suggests that for low associativity, V/P or V/V caches are more robust than physical address 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 locality naturally exhibited by most programs[24].

The relative differences between the miss rates of V/V and V/P caches are displayed in Figure 7. They are computed as the number of V/P misses minus the number of V/V misses divided by the number of V/V misses. As the cache size increases, the impact of cache indexing becomes relatively more significant. The miss rate of the V/P cache becomes much worse than the V/V cache (up to 70% worse in RSIM). A significant part of this difference is due to short misses.

4.4. Short Misses

V/P caches are affected by short misses. Figure 8 shows the fraction of short misses in all the benchmarks, i.e. the number of short misses divided by the total number of misses for V/P caches. Again we show results for direct-mapped and four-way set-associative caches. When the cache size increases, more synonyms are indexed into different sets (see Figure 5) and tend to stay longer in the cache leading to an increase in short misses; at the same time, the number of capacity misses decreases sharply. We observe a sharp increase of the fraction of short misses for all benchmarks as soon as the cache size reaches 128KB. Actually if we compare the short miss rates with the relative miss rate difference in Figure 7, we can see that the short misses dominate the relative difference for 4 way set associative caches, especially for large cache sizes.

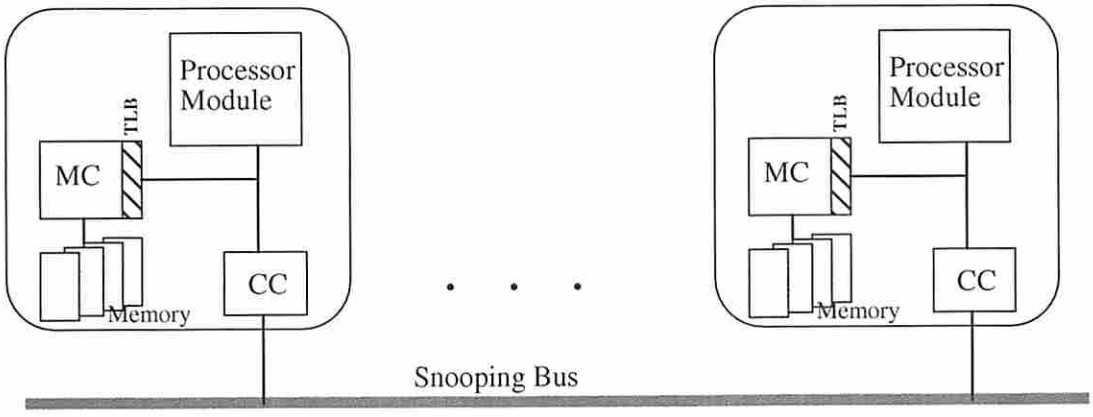
5. Discussion

Regardless of the scheme used to solve the synonym problem virtually-addressed cache hierarchies have many attractive features. In this section, we show how virtually-addressed memory hierarchies enable natural, scalable multiprocessor extensions, as well as computing-in-memory in the context of general-purpose computers

5.1. Multiprocessor Architecture Extensions

Virtually-addressed memory hierarchies enables new designs for multiprocessors. Qiu and Dubois[27] showed some new design options for COMA architectures. Here we propose an architecture based on SMP clusters.

Figure 9 shows the basic architecture of an SMP cluster similar to the Sun Starfire enterprise server[10]. Memory is distributed among system nodes. Cache coherence is maintained by one or more snooping buses. All caches are virtually addressed and SLBs translate synonyms inside processors. Every memory controller has a TLB containing translations for the physical memory it controls..



CC: Coherence Controller
MC: Memory Controller

Figure 9 SMP cluster

Caches are accessed with addresses in the main set. On a miss, the local cache controller(CC) broadcasts the virtual address on the snooping bus. If the snooping is successful, a cache-to-cache transfer satisfies the memory access. Otherwise the TLBs in all memory controllers are accessed (snooped) in parallel. If the virtual address hits in one TLB the block is delivered by the main memory otherwise the processor is trapped. A miss in all TLBs may means that the address belongs to the synonym set. The trap

handler searches the page table and then either fills the SLB, fills the TLB fill or triggers a page fault. Eventually the access will be retried.

To construct large-scale systems, we take advantage of the global naming provided by the main virtual address. If a page is not allocated in the local cluster, the page table entry and TLB entry at the memory contain a cluster ID pointing to the home cluster of the page. In the snooping phase, if a virtual address hits on a TLB and the entry contains a cluster ID, the CC intervenes and sends the request to that remote cluster. When the request reaches the remote cluster, the virtual address is snooped on the bus to locate the data. Each cluster has its own page table. The first access to a virtual address in the cluster triggers a page fault even if that virtual address reside in main memory in a remote cluster. The page fault handler must setup the page table entry pointing to that cluster ID so that following accesses are correctly directed

This multiprocessor architecture has some very attractive features.

1) The dynamic address translation overhead is expected to be very small since the TLB is located after the virtual cache hierarchy. It benefits from the filtering, sharing and prefetching effects[27].

2) Page migration overhead is dramatically reduced, because of two reasons. The first one is the elimination of the TLB consistency problem. The second one is that the cache hierarchy is not affected by page migration. When a page is migrated, the physical address in the page table entry at the home node is simply replaced by the cluster ID of the new home. Any incoming remote accesses on that page is forwarded to the new home and a message reply to the requester redirects the page table entry to the new home.

3) Simple COMA[30] can be implemented very efficiently to replicate data among clusters. In the original simple COMA proposal, an additional level of address translations is required to translate physical addresses into global logical address. This additional level of address translation is unnecessary when virtual addresses can be used for naming.

5.2. The Impact of Integration

With technology improvements, computer systems are integrated into less and less chips. We observe two trends in this integration.

The Alpha 21364[18] integrates the L2 cache, memory controller, and a high-speed inter-processor connection into the processor chip. Integrating the memory controller into the processor helps relieve the memory wall problem by increasing the bandwidth and speed of the memory system. Other processors integrated with a memory controller include the Sun Ultrasparc-3 and Cyrix MediaGX. In general, the multiprocessors shown in Figure 9 could be more efficient with better technology and higher level of integration such as in the alpha 21364.

In addition, it is possible to support more complex and flexible memory system in software. Instead of building complex memory controller hardware to search the page table and others, the processor can be utilized to implement complex functions. Assuming future processors have the ability to run multiple threads simultaneously, a nanothread[31] or microthread[9] can be dedicated to support memory controller functions with very little impact on CPU performance.

The other trend is to integrate processing logic into memory chips. Targeting the same memory wall problem, the Processor-In-Memory (PIM) approach uses very simple processors exploiting the huge internal DRAM bandwidth for memory accesses. There have been some research efforts to use PIMs to build memory systems for high-performance processors[21][32], where data intensive computing can be moved into PIMs to boost system performance. In the context of general-purpose computing, it is difficult to compute in memory on behalf of the kernel or of a user if the main memory "sees" physical addresses.

By migrating virtual memory functions to memory, a thread running in memory is encapsulated in the same addressing space as the threads cooperating with it in the CPU and benefit from the same protection, debugging facilities, and access to system-wide resources in a general-purpose environment.

6. Related Work

Over the years many researchers in both the software and hardware community have advocated and proposed ideas supporting virtually addressed memory hierarchies.

On the software side, Opal[11] 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 is clearly not ready for such radical change.

Talluri et al.[35] and Romer et al.[29] looked at using superpages to increase TLB coverage without enlarging the TLB size. They demonstrated that superpages dramatically cut the TLB overhead by mapping physical memories in big chunk. In [29] superpages are constructed dynamically by promoting small pages to a large page. The promotion itself requires copying physical pages and updating kernel data structures and TLB shootdowns. Not only the 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 cooperate with other memory allocation issues, especially NUMA-oriented memory system and the coloring considerations for cache friendly optimizations[4].

Impulse[34] 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 non-contiguous 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[27] 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.

In the simple COMA architecture proposed by Saulsbury et. al[30], the virtual memory system is also involved in the cache coherence handling to allocate pages for replication. WildFire[20] connects several large SMP to build large scale shared memory multiprocessors using an R-NUMA-like[16] protocol. We have discussed in this paper how this type of multiprocessors can be implemented with higher performance on a V/V cache hierarchy.

Teller[36] argued the scalability 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 address cache have been the topic of many research papers[1][5][6][8][12][17][19][22][27][33][36][39][40], there are very few quantitative analysis of virtual cache performance. Agarwal[1] analyzed virtual address cache performance using traces from VAX. Wu et. al[40] evaluated different cache types including V/P and V/V caches using a trace from IBM 370. Although some of their observations are similar to the ones in this paper, the quantitative results are hard to compare because of the different schemes adopted for virtual address caches and of the different workloads and operating systems used in the experiments. Lynch[24] observed that physical cache performance varies for each run depending on the allocation of pages in the operating system, while on the other hand the performance of virtual caches is not sensitive to these implementation decisions. We observe the same

trend that the miss rate of virtual caches is more robust than that of physical caches, especially for caches with low associativity.

7. Conclusions

In this paper we have taken a new look at virtually-addressed memory hierarchies. Moving the virtual-to-physical translation down the hierarchy presents some technical challenges, but we argue that the problems have been mostly solved. We also show a new, simple, and effective solution to the synonym problem, which is one of the major technical problems. In this solution, one main virtual address is used to name each virtual page uniquely in the processor and in the memory hierarchy and a Synonym Lookaside Buffer (SLB) translates synonyms into main 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 the purging and flushing of virtual-address caches is very limited in practice and has insignificant impact on performance. The problem associated with late memory traps also has many solutions, elaborated upon in other papers.

We have presented extensive performance results comparing the miss rate behavior of physical and virtual address caches. It appears that virtual-address caches have better miss rates than physical caches and the solution using a small SLB in front of the caches avoids short misses in larger caches, while safeguarding the benefits of temporal and spatial locality in the virtual space.

Previous work has shown that TLBs do not scale well inside the CPU, for various reasons. On the other hand their effectiveness improves as they are moved down the hierarchy. At one extreme the TLB may be associated with the physical main memory it covers. This approach has many performance advantages, but we also argue in this paper that it opens the door to new solutions for scaling up the size of multiprocessor systems and that it facilitates the integration of processing in memory for both user and kernel in the context of general-purpose computing.

8. Reference

[1] Anant Agarwal, "Analysis of cache performance for operating system and multiprogramming," Kluwer Academic Publishers, Boston, 1989.

[2] Andrew Appel and Kai Li, "Virtual Memory Primitives for User Programs", In *Proceedings of the 4th Conf. on Architecture Support for Programming Languages and Operating Systems(ASPLOS)*, pages 96-107, 1991.

[3] Todd Austin and Gurindar Sohi. "High-Bandwidth Address Translation for Multiple-Issue Processors," In *Proceedings of the 22nd Annual International Symposium on Computer Architecture(ISCA)*, pages 158-167, 1996.

[4] E. Bugnion, J. M. Anderson, T. C. Mowry, M. Rosenblum, and M. S. Lam, "Compiler-Directed Page Coloring for Multiprocessor," In *Proceedings of the 7th Conf. on Architecture Support for Programming Languages and Operating Systems(ASPLOS)*, Oct. 1996.

[5] Michel Cekleov and Michel Dubois. "Virtual-Address Caches, Part 1: Problems and Solutions in Uniprocessors", IEEE *Micro*, pages 64-71, Sep/Oct. 1997.

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

[7] Albert Chang and Mark Mergen, "801 Storage: Architecture and Programming", *ACM Transaction on Computer Systems*, Vol 6, No. 1, February 1988, Pages 28-50.

[8] Chia Chao, Milon Machey, Bart Sears, "Mach on a Virtually Addressed Cache Architecture",

Proceedings of the 1st Mach USENIX Workshop, pages 31-51, Oct. 1991.

[9] R Chappel, J Stark, S. Kim, and Y. Patt, "Simultaneous Subordinate Microthreading (SSMT)", In *Proceedings of the 26th Annual International Symposium on Computer Architecture(ISCA)*, May 1999

[10] Alan Charlesworth, "STARFIRE: Extending the SMP Envelop", *IEEE Micro*, pages 39-49 January/ February 1998.

[11] Jeffrey Chase, Henry Levy, and Michael Feeley, "Sharing and Protection in a Single-Address-Space Operating System," In *ACM transaction on computer systems*, pages 271-307, Nov., 1994.

[12] David Cheriton, Gert Slavenburg, and Patrick Boyle, "Software-Controlled Caches in the VMP Multiprocessor", In *Proceedings of the 13rd Annual International Symposium on Computer Architecture(ISCA),* pages 366-375, 1986.

[13] Hsiao-keng Jerry Chu, "Zero-Copy TCP in Solaris", *1996 USENIX Technical Conference*, pages 253-264, January 22-26, San Diego, CA

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

[15] Michel Dubois, Christoph Scheurich, and Faye Briggs, "Memory Access Buffering in Multiprocessors", In *Proceedings of the 13rd Annual International Symposium on Computer Architecture(ISCA),* pages 320-328, 1986.

[16] Babak Falsafi, and David Wood, "Reactive NUMA: A Design for Unifying S-COMA with CC-NUMA," In *Proceedings of the 24th Annual International Symposium on Computer Architecture(ISCA)*, 1997.

[17] J. R., Goodman, "Coherency for Multiprocessor Virtual Address Caches," In *Proceedings of the 2nd Conf. on Architecture Support for Programming Languages and Operating Systems(ASPLOS)*, 1987.

[18] Linley Gwennap, "Alpha 21364 to Ease Memory Bottleneck," Microprocessor Report, Oct. 26, 1998

[19] ANONYMOUS (removed for submission)

[20] Erik Hagersten and Michael Koster, "WildFire: A Scalable Path for SMPs", In *Proceedings of the 5th International Symposium on High Performance Computer Architecture(HPCA)*, January, 1999.

[21] Mary Hall, Peter Kogge, Jeff Koller, et al, "Mapping Irregular Application to DIVA, a PIM-based Data-Intensive Architecture", in *proceedings of SC99*, Nov., 1999.

[22] Bruce Jacob and Trevor Mudge. "Software-Managed Address Translation," In *Proceedings of the 3rd International Symposium on High Performance Computer Architecture(HPCA)*, Feb. 1997.

[23] Eric J. Koldinger, Jeffrey S. Chase, and Susan J. Eggers. "Architecture Support for Single Address Space Operating System," In *Proceedings of the 5th Conf. on Architecture Support for Programming Languages and Operating Systems(ASPLOS)*, pages 175-186, Oct. 1992.

[24] William Lynch. "The Interaction of Virtual Memory and Cache Memory," Ph.D. Thesis, Technical Report CSL-TR-93-587, Stanford University, 1993.

[25] C. May, E. Silha, R. Simpson, and H. Warren, Eds. "The PowerPC Architecture: A Specification for a New Family of RISC Processors," Morgan Kaufmann Publishers, San Francisco CA, 1994.

[26] Vijay Pai, Parthasarathy Ranganathan, Sarita Adve, "RSIM Reference Manual", Technical Report 9705, Department of Electrical and Computer Engineering, Rice University, Aug, 1997

[27] Xiaogang Qiu and Michel Dubois, "Options for Dynamic Address Translation for COMAs", In *Proceedings of the 25th Annual International Symposium on Computer Architecture(ISCA),* pages 214-225, 1998.

[28] Xiaogang Qiu and Michel Dubois, "Tolerating Late Memory Traps for ILP Processors", to be appeared

in *Proceedings of the 26th Annual International Symposium on Computer Architecture(ISCA)*, 1999.

[29] Theodore H. Romer, Wayne H. Ohlrich, and Anna R. Karlin. "Reducing TLB and Memory Overhead using Online Promotion," In *Proceedings of the 22nd Annual International Symposium on Computer Architecture(ISCA)*, page 176-187, 1995.

[30] Ashley Saulsbury, Tim Wilkinson, John Carter, and Anders Landin, "An Argument for Simple COMA", In *Proceedings of the 1st International Symposium on High Performance Computer Architecture(HPCA)*, January, 1995.

[31] Yong Ho Song and Michel Dubois,"Assisted Execution", Technical Report #CENG 98-25, Department of EE-Systems, University of Southern California, October 1998.

[32] Thomas Sterling, Larry Bergman, "A Design Analysis of a Hybrid Technology Multithreaded Architecture for Petaflops Scale Computation", In *proceedings of the International Conference on Supercomputing(ICS'99)*, Rhodes, Greece, June, 1999.

[33] ANONYMOUS (removed for submission)

[34] Mark Swanson, Leigh Stoller, and John Carter, "Increasing TLB reach Using Superpages Backed by Shadow Memory", In *Proceedings of the 25th Annual International Symposium on Computer Architecture(ISCA)*, pages 204-213, 1998.

[35] M. Talluri, S. Kong, M. D. Hill, and D. A. Patterson. "Tradeoffs in Supporting Two Page Sizes," In *Proceedings of the 19th Annual International Symposium on Computer Architecture(ISCA)*, pages 415-424, May 1992.

[36] Patricia Teller, "Translation Lookaside Buffer Consistency," Ph.D thesis.

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

[38] W, H. Wang, J-L, Baer, and H. M. Levy, "Organization and performance of a two-level Virtual-Real cache hierarchy," In *Proceedings of the 16th Annual International Symposium on Computer Architecture(ISCA)*, pages 140-148, June 1989.

[39] David Wood, Susan Eggers, Garth Gibson, Mark Hill, and Joan Pendleton. "An In-Cache Address Translation Mechanism," In *Proceedings of the 13th Annual International Symposium on Computer Architecture(ISCA)*, pages 358-365, Jan. 1986.

[40] C. Eric Wu, Yarsun Hsu, and Yew-Huey Liu, " A Quantitative Evaluation of Cache Types for High-Performance Computer Systems", *IEEE Transactions on Computers*, pages 1154-1162, Vol 42, No. 10, Oct. 1993.

[41] K. C. Yeager, "The MIPS R10000 Superscalar Microprocessor," IEEE *Micro*, pages 28-40, April 1996.