VIRTUAL-ADDRESS CACHES

Michel Cekleov*, Michel Dubois
Jin-Chin Wang, Fayé A. Briggs*

USC Technical Report No. CENG 90-18

Department of Electrical Engineering – Systems
University of Southern California
Los Angeles, California 90089-0781
(213)743-8080

*Workstation and Server Systems Operation
Sun Mircosystems Inc.
2550 Garcia Avenue
Mountain View, California 94043
(415)960-1300

VIRTUAL-ADDRESS CACHES

ABSTRACT

Most general-purpose computers support virtual memory. Generally, the cache associated with each processor is accessed with a physical address obtained after translation of the virtual address in a Translation Lookaside Buffer (TLB). Since today's uniprocessors are very fast, it becomes increasingly difficult to include the TLB in the cache access path and still avoid wait states in the processor. The alternative is to access the cache with virtual addresses and to access the TLB on misses only. This configuration reduces the average memory access time, but it is a source of consistency problems that must be solved in hardware or software. The basic causes of these problems are the demapping and remapping of virtual addresses, the presence of synonyms, and the maintenance of protection and statistical bits.

All these problems are addressed in this paper. Solutions are compared based on their feasibility and their transparency to the software in both uniprocessor and multiprocessor systems. Overall, it appears that all problems can be solved efficiently at the cost of more complex hardware and/or non-transparency from the software.

TABLE OF CONTENTS

1. INTRODUCTION

In this survey, we consider one type of cache memories: virtual-address caches. It is assumed that the reader is familiar with the various aspects of physical-address cache design. One can refer to [Smith 82] for a general presentation of cache memories.

When the processor architecture supports virtual memory, the cache can be accessed either directly with virtual addresses (virtual-address cache) or with physical addresses obtained after translation (physical-address cache). Because of the consistency problems caused by virtual-address caches, almost all computer systems use a physical-address cache. Although the translation of virtual addresses to physical addresses is supported by a special-purpose cache (usually called a Translation Lookaside Buffer or TLB) virtual memory tends to increase the memory access latency. With the advent of RISC technology [Patterson 81] and the latest improvements in VLSI technology the cache access is becoming the critical path of most instruction pipelines. In physical-address caches, the TLB and cache accesses must be either pipelined or performed in parallel.

In some high-performance computer systems, cache access and virtual-to-physical address translation are pipelined. For example, in the microprocessor R2000 from MIPS Computer Systems [Mous-souris 1986] TLB access and cache access are performed in two distinct stages of the instruction pipeline. In most implementations, the current limitations of packaging and VLSI technology impose to have the TLB on the same chip as the CPU, which restricts the silicon area available for the other functional units and limits the TLB size.

In many mainframe systems, the TLB and the cache are accessed in parallel. Since the set selection in a set-associative cache is done with the low-order address bits, the bits specifying the displacement within the page can select the set in the cache while the TLB translation takes place, provided the size of the cache is not greater than the page size times the degree of associativity (number of elements in a set). In this case, the TLB translation time must be less than the set-selection time. Access times of static RAM chips are currently between 10 and 20 ns and therefore this design constraint becomes increasingly difficult to meet. For example, in the Motorola 88000 chip set, both the TLB and the cache are on the same chip and are accessed in parallel.

In most systems the page size is between 512 bytes and 8 K-bytes, the cache size varies between 1 and 256 K-bytes and the degree of associativity is at most 8. In a shared memory multiprocessor system, the private caches associated with each processor must have a very good hit ratio, and therefore a large size, in order to reduce effectively the memory access latency and to increase the bandwith of the memory system [Stone 87]. Another important parameter of a cache design for multiprocessors is the block size [Lee 87][Patil 87] (one can find a good discussion on cache blocks in [Goodman 87]).

Even if the cache size is increased the cache can still be accessed in parallel with the TLB by extending the degree of associativity or, by selecting more than one set at each access as it is done in the Amdahl 470V/6 system. However, in both cases, the complexity of the tag comparison logic increases which tends to lengthen the cache access time.

In other architectures, the cache is directly accessed with virtual addresses. Virtual-address caches have traditionally been avoided because they are not totally transparent to the software even in uniprocessors. Consistency problems occur within the same cache whenever a virtual-to-physical mapping is changed or when different virtual addresses are mapped to the same physical address. The problems are even more complex in multiprocessors because these inconsistencies can occur in more than one processor. Nevertheless, a virtual-address cache has many attractive features. First and foremost, most accesses to data and instructions are satisfied in one cycle of the cache. Because there is no restriction on the bits used for the set selection, the cache size can easily be very large without having to increase the degree of associativity.

Moreover, since virtual-to-physical address translations are primarily required on a cache miss, TLB access time is not critical. For low-cost systems, virtual-address caches can be used in conjunction with relatively slow, off-the-shelf MMUs (Memory Management Units) [Furht 87][Frink 88]. The TLB can be very large and therefore exhibit an excellent hit ratio.

In this paper, the problems related to virtual-address caches are exposed in the contexts of uniprocessor and multiprocessor systems. Some solutions are presented and discussed. To appreciate and understand these problems, we must first overview the relevant properties of virtual memory models.

2. VIRTUAL ADDRESSING

2.1 INTRODUCTION

All modern general-purpose computer systems implement virtual memory. Generally, the virtual memory is paged and managed through a demand paging algorithm, in which pages residing on a paging device are swapped into main memory as the executing processes need them. Swapping is triggered by a page-fault, which is usually treated as an exception in the processor. The success of paging algorithms is based on the principle of "locality of reference" exhibited by most useful programs.

A virtual memory system is implemented by a combination of hardware and software mechanisms. The software support is part of the operating system kernel and comprises various data structures as well as primitives to manipulate them; these data structures are needed to locate the information which running processes wish to access. Hardware support includes:

- Support for page-fault exceptions, i.e. the ability of the CPU to restart or continue the execution of an instruction interrupted by a memory fault [MacGregor 83].
- Support to reduce the mapping overhead, usually in the form of a TLB or MMU.

Besides the obvious advantages of transparent memory management, virtual memory provides basic support for *protection*. Protection in a computer system enforces access rights of processes to information. We see protection as having two dimensions. The first dimension is provided by the privilege level at which a process runs; the second one is the access control. While the access control restricts the accessible set of data and the type of accesses, the privilege levels define process states in which specific access controls are effective.

Most architectures only support two privilege levels: the kernel and user execution modes. A logical extension of this simple model has been adopted in the Intel i286/i386 microprocessors [Furht 87], in which several levels are nested in a ring structure. This protection system is called a *ring* protection system.

There can be multiple units of access control. Generally there are two; the first one is defined by the virtual space model and the second one is the same as the unit of mapping, generally the page. The access rights to a unit of mapping can be defined for each privilege level supported by the logical architecture of the system. The protection can be further refined if other units of protection are defined, such as segments.

2.2 VIRTUAL ADDRESSING MODELS

There are two different classes of virtual memory models:

- The private virtual space model.
- The global virtual space model.

2.2.1 Private Virtual Space Model

In this model, a distinct virtual space is allocated to each process. Therefore, virtual-to-physical mappings are characteristic of the running process and the kernel executes in the "context" of the process. Each process virtual space is divided in two main regions: the kernel or system space and the user space (Figure 1). The partition between the kernel and user spaces is fixed, and the user space is usually structured in three segments: the text (or code) segment, the data segment and the stack segment.

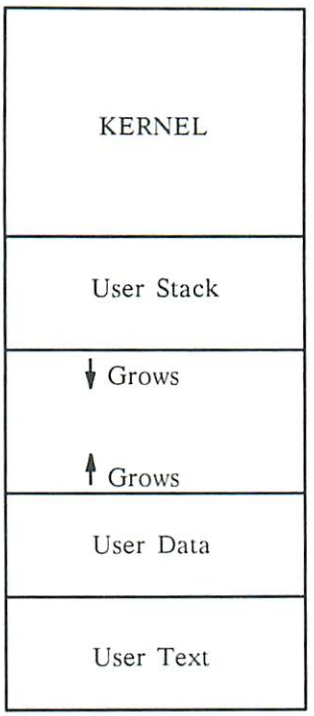


Figure 1: Virtual Address Layout

The virtual-to-physical address mapping of kernel pages is common to all processes while the mapping of user pages is different for each process. Thus, there is one translation table for each process and one for the kernel. Although a single level of table is logically enough to translate a virtual page number (noted VPN) into a physical page number (noted PPN), two or three levels are usually provided to support sparse addressing more efficiently. Each entry of the last table, which is called a page table entry (noted PTE), contains a physical page number and various bit fields used by the kernel to implement demand paging and protection.

This addressing model implicitly provides part of the second dimension of the protection by confining the references of a process to its own virtual space. However, there are many cases where it is necessary to share some information among processes. The most common case is when a process creates another one. Then, usually, the parent and the child processes share the same text segment; in some kernel implementations (e.g. UNIX bsd 4.2), distinct page table entries therefore point to the same physical page frame. When two or more virtual addresses map to the same physical address they are said to be *synonyms* or *aliases*.

In the private virtual space model a virtual address is usually extended by concatenating a process identifier (noted PID); distinct mappings of otherwise identical virtual page numbers can therefore be present in the TLB at the same time, and the TLB does not have to be flushed at each context switch. When the cache is a virtual-address cache the same benefit is gained by the PID extension.

The private virtual space addressing and protection model is the one supported by the most popular operating systems such as UNIX [Bach 86] and VMS [Levy 82].

2.2.2 Global Virtual Space Model

This second model of addressing defines a single virtual space common to all processes and is also called the *single* or *unique* virtual space model. This approach is viable only if the virtual space is huge so that separate address ranges can be assigned to distinct processes. Therefore, this very large virtual space must be structured in segments. The virtual address is built by the concatenation of a segment identifier/descriptor and a displacement inside the segment. In paged virtual memory systems, this address is then split in a virtual page number and a displacement inside the page (Figure 2).

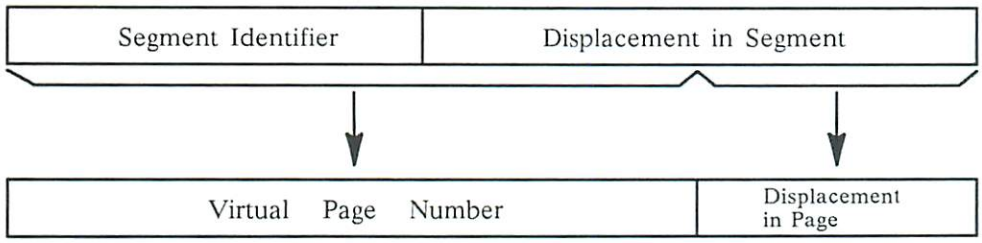


Figure 2: Virtual Address Layout

The notion of segment provides an efficient way to protect and share some information while the notion of page provides a convenient unit for I/O exchange with the mass storage devices and for main memory allocation [Bensoussan 72].

Because the virtual space is very large it is difficult for the kernel to translate a virtual page number into a physical page number through a cascade of page tables, as it is done in the private virtual space model: there simply would not be enough space in main memory to store all the page tables. Furthermore, the tables would have to be "stored" in the single virtual space and multiple address faults could occur during a translation. Hence, for performance reasons a different algorithm is usually implemented in the kernel to translate a virtual address in a single virtual space.

The translation is accomplished through two tables: an index table or hash table and a page-frame table (Figure 3).

The virtual page number is hashed to access the hash table entry where an index to an entry of the page-frame table is stored. Multiple virtual addresses map to the same value of the hashing function; entries of the page-frame table corresponding to such addresses must therefore be linked in a collision chain. Each entry of the page-frame table contains the virtual number of the page currently residing in the page frame. The translation software first selects an entry in the page-frame table through hashing; then it compares the virtual number in the table with the virtual address of the page. If they match, the physical page address is given by the position of the entry; otherwise the collision chain must be searched until a match is found or until the end of the chain is reached. In this latter case, the page is not in main memory i.e., a page fault is triggered. The purpose of the hash table is to minimize the average length of collision chains in the page-frame table.

There is no need to store physical page addresses in the entries of the page frame table: the physical page number is given by the position of the matching entry in the table. For this reason the page-frame table has also been called an *inverted page table* [Chang 88].

In a single virtual space there is no need for using aliases to share information between two processes or between the kernel and a process. It is clear that the one-to-one address mapping described

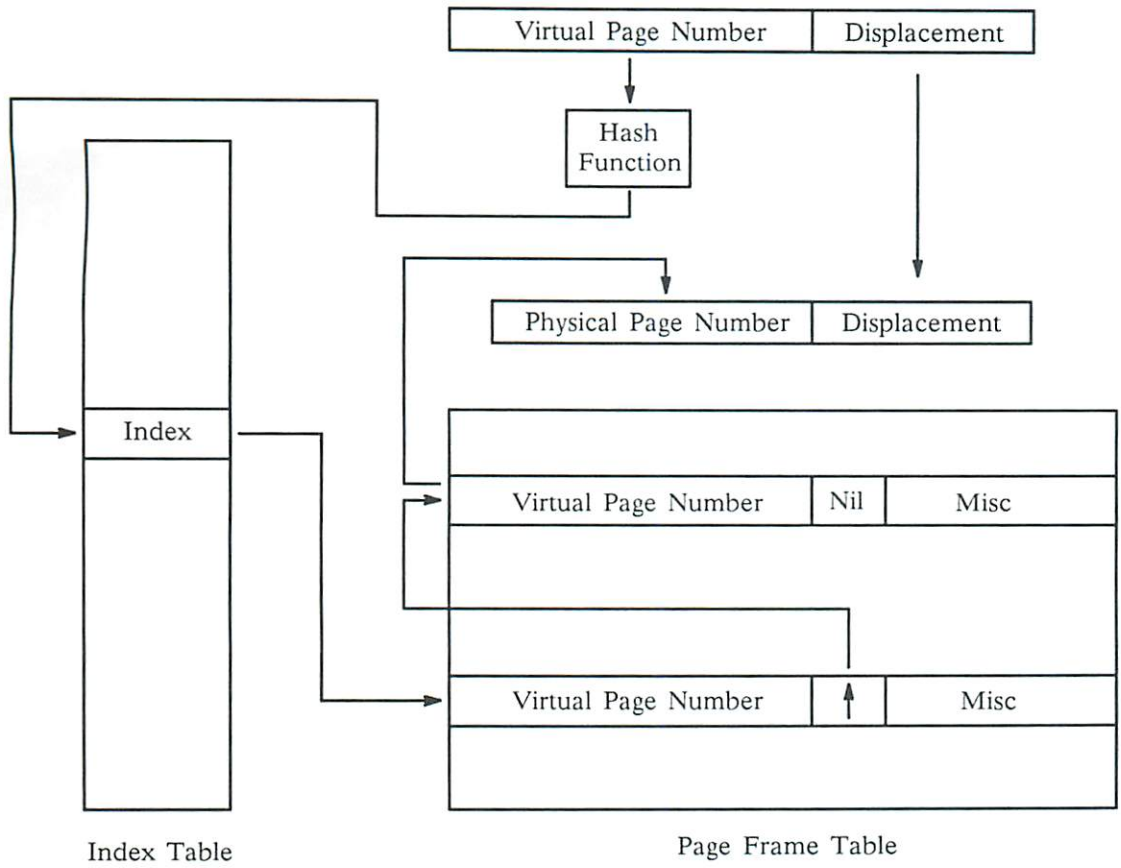


Figure 3: Single Virtual Space Page Address Translation

above cannot support aliasing. Thus, this addressing model is very attractive for systems with virtual-address caches because it removes the aliasing problem and greatly simplifies the hardware implementation. This is not the only advantage of the single virtual space. Another important benefit is the ability (offered by the hugeness of the virtual space) to map all files in the virtual space resulting in a *one-level store* [Chang 88]. Swapping and file storage spaces on disks are now in the same space.

Usually, the processor would contain some segment registers, which are used as addressing registers. The contents of these registers are accessed either implicitly or explicitly during the execution of instructions to compute virtual addresses. The loading and storing of all or part of these registers are authorized only in the most privileged execution mode(s) (the supervisor mode if there are only two). The addressing domain of a process is now explicit and all its memory references are confined to the segments for which it has a descriptor.

The protection dimension offered by the virtual space segmentation can be augmented by the usual page-level protection mechanism. However, because the mapping between a virtual page and a physical page is one-to-one it is difficult to give different access rights to each of the processes sharing some information. Therefore, besides the page-frame table, other data structures are needed to define page access rights on a per-process basis. For example, the TLB could be loaded with different protection information for the same virtual address depending on the process running (the TLB must then be flushed at each context switch). Another solution is to use a *key/lock* mechanism as it is done in the HP Precision architecture [Mahon 86]. Each segment has an access identifier (15 bits in the HP Precision architecture) which constitutes the *lock* and some *keys* are associated with each process. During an access, the *keys*, which are

loaded in dedicated registers, are compared with the *lock* (i.e. the access ID of the segment) to determine if the information is accessible and which type of access (read, write, execute) is authorized.

The single virtual space model is not as common as the private virtual space model and can only be found in a few machines such as the IBM 38 [Soltis 81], the IBM 801 [Chang 88], the IBM PC/RT [Simpson 86] and the HP Precision Architecture [Mahon 86]. The HP Precision architecture demonstrates that an operating system assuming a private virtual space model (UNIX) can still be accommodated.

The previous discussion on virtual addressing models is not intended to be exhaustive and many variations are possible within each model. Furthermore, the classification of virtual memory systems in two categories is probably not the only one possible. Yet, this classification is sufficient for the scope of this paper because it identifies the cause of aliasing and the protection issues that may require direct support from a virtual-address cache.

2.3 TRANSLATION LOOKASIDE BUFFER

A TLB is a cache of translations which accepts a virtual page number and returns a physical page number or a signal indicating a missing translation. A TLB entry contains a virtual page number and its associated physical page number. Besides providing an efficient way to translate virtual addresses, a TLB usually includes:

- some hardware support for protection, and
- some hardware support for the management of the data structures used by the kernel to implement the virtual memory system.

For systems with physical-address caches, the TLB is a mandatory access path for most memory references. Thus, the TLB is the ideal place for checking access rights. Some protection bits are generally associated with the physical page number (Figure 4) and they are interpreted differently according to the current privilege level and the type of memory reference (Instruction fetch, Data write or read). If the virtual space is unique with a *key/lock* protection mechanism as described previously, the protection bits field is extended with the access ID of the segment containing the page [Mahon 86].

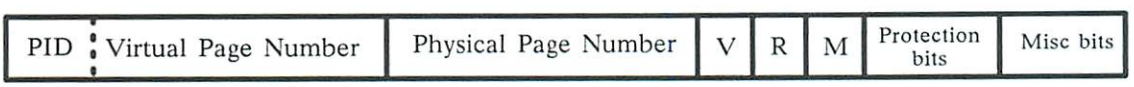


Figure 4: TLB Entry Structure

In the private virtual space model, the virtual page number can be extended with a process identifier (PID) to avoid purging (invalidating) all the contents of the TLB on every context switch. When all the possible PID values have been allocated a purge of the whole TLB is necessary if a selective purge according to the PID value is not supported by the hardware. Purging means that the entry must be invalidated. For this purpose each TLB entry is supplied with a valid bit (V).

A TLB entry usually contains two additional bits to support demand paging: the *reference* bit, R, and the *modify* bit, M (Figure 4). These are copies of the R and M bits contained in the corresponding entry of the page table or of the page-frame table (inverted page table) depending on the virtual space model.

The *reference* bit in the page table entry is used by the kernel to implement the page replacement algorithm. This bit is set whenever a process accesses the page and is reset by the page-stealer daemon. When an entry is loaded into the TLB a copy of the R bit is also loaded. When the page stealer resets the *reference* bits in the page table entries it must also reset the copies present in the TLB. When the page has not been referenced for a while, the R bit remains reset, and the page becomes eligible for swapping. This algorithm is an approximation to the working set policy for replacing pages in main memory.

An access to a TLB entry where the *reference* bit is not set either triggers the setting of the page table entry copy directly by the hardware (microcode) or raises a trap if there is no specific hardware support. The trap handler then updates the copy in main memory. Therefore the consistency between the copies of the *reference* bit is maintained in a kind of "write-through" manner. A write-back approach is also possible, if the hardware or the software handling a replacement in the TLB updates the *reference* bit in the page table entry corresponding to the entry or entries victimized by the replacement algorithm. When the management of the TLB is done by software the architecture must be extended for that purpose. This leads to the definition of specific instructions as in the R2000 architecture [DeMoney 86] and HP Precision architecture [Mahon 86] and/or the memory mapping of TLB structures [Kelly 85]. In this latter case, the TLB should be able to "lock" some of the entries (i.e., they cannot be victimized by the replacement algorithm) or an alternate mapping mechanism must be provided (e.g. a TLB bypass mode) so that "double misses" cannot occur.

The only purpose for the *reference* bit in each TLB entry is to avoid invalidating the translation held in an entry when the page stealer resets the corresponding *reference* bit in the main memory [Babaoglu 81]. Because this event is rare it has been argued that a *reference* bit is not really necessary in the TLB [DeMoney 86]. Other replacement policies for the allocation of the page frames in main memory, such as a pure FIFO algorithm [Levy 82], do not need the support of a *reference* bit.

The *modify* bit is used by the swapper process to decide if the page must effectively be copied back on disk when it is victimized by the page replacement algorithm. This bit must be set on the first modification of the page after it has been swapped in. As for the *reference* bit, a copy of the *modify* bit is loaded in the TLB entry with the virtual/physical page translation. Generally, when the processor attempts to modify the content of a page while the *modify* bit inside the TLB entry is not set, the hardware (microcode) automatically sets it as well as the copy in main memory (alternatively, a trap could be raised and a software handler would execute on the processor). A write-back approach is also possible as for the *reference* bit.

The functions of the *modify* bit can be "emulated" with the access rights bits. A page can be defined as non-writable in the TLB so that a trap to the kernel is triggered on the first attempt to modify the page. In the trap handler, the kernel can check whether the page is actually non-writable. If not, the *modify* bit in the PTE needs to be set, or some other actions such as making a copy of the page must be undertaken. For example, in most current implementations of UNIX the first write to a page must be detected to implement copy-on-write for "forked" processes or mapped files [Bach 86].

The management of a *modify* bit in the PTEs is really needed for performance reasons. In most systems only 40% of all pages need to be written back to the swap device(s).

One can often find other miscellaneous indicators in the TLB entries, such as a bit specifying whether a page is cacheable. All these other indicators are not relevant to the topic of this paper and therefore are not discussed.

3. VIRTUAL-ADDRESS CACHES IN UNIPROCESSORS

In this section we examine all the issues related to a virtual-address cache in a single processor environment. We analyze all the consistency problems and suggest possible solutions according to the cache organization.

We assume that I/O data residing in main memory are encapsulated in non-cacheable pages and/or I/O operations take place directly in the cache. Therefore, there is no hardware support to maintain data consistency with the cache during DMA accesses. In particular, there is no copy of the physical tag associated with each virtual tag, i.e. there is not a dual cache directory containing the physical addresses of the blocks in the cache. In the section dealing with multiprocessor systems we will consider the existence of hardware support for data consistency and its use for solving some of the problems described in this section.

3.1 VIRTUAL TO PHYSICAL ADDRESS DEMAPPING / REMAPPING

A virtual address VA is mapped to a physical address PA1 during a certain period of time (Figure 5). When the operating system decides to demap and then remap this virtual address VA to a new physical address PA2 an inconsistency can occur if some data associated with PA1 are kept in the cache. The CPU can access the data associated with PA1 instead of the data associated with PA2. This inconsistency happens if the access of the CPU is a read or a write and, in this sense, it is different from the inconsistency that can happen if two virtual addresses are synonyms as explained in the next section.

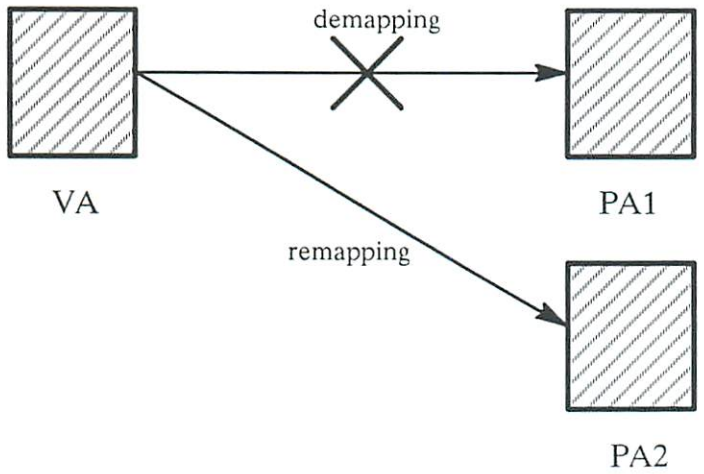


Figure 5: Virtual to Physical Mapping Change

In general, the demapping/remapping involves a subset of the virtual space. For instance the demapping/remapping of a virtual page to a new physical page occurs when a new page is swapped into main memory. In the private virtual space model, a context switch entails a demapping/remapping of all the user space if the virtual addresses are not extended with a PID (Figure 1). Otherwise, a demapping/remapping is mandatory when all PID values have been used. When the virtual space is unique, whole segments can also be remapped. These are not the only causes of mapping change; some mapping changes are peculiar to the implementation of the kernel of the operating system [Cheng 86].

3.1.1 Write-Through Caches

In the case of write-through caches, all the blocks belonging to the page(s) being demapped and then remapped must be purged, i.e. invalidated. For example, when a page is reclaimed by the page stealer and becomes candidate for swap-out the cache needs to be purged. Invalidations in the cache should take place before the invalidations of the corresponding translations in the TLB because the cache controller needs the physical address of a block on a processor write.

Because the kernel controls the mapping of virtual to physical addresses, it is responsible for initiating the purge actions. The minimum support which the hardware must provide is a way for the software to invalidate an entry. This can be done with specific instructions as in the HP Precision architecture [Mahon 86] or with memory mapped commands sent by the CPU to the cache controller.

In steady state when all the TLB entries are occupied, entries must be displaced on TLB misses. Hence, a cache hit can occur even if no translation is present in the TLB. The TLB needs to be accessed on cache misses and on all processor writes because the physical address is required to update the main memory. If the TLB is fast enough, it can also be accessed in parallel with the cache on every memory reference. In that latter case, TLB misses occurring on processor reads can be ignored if they do not cause any cache miss.

Some TLBs, called *table-walking* TLBs, handle a miss automatically, and therefore a TLB miss is always transparent to the CPU. In other architectures, TLB misses are handled in software through a regular trap and in this case the implementation of the cache controller may be more delicate.

In high-performance systems, the stores are buffered in a write-buffer and TLB misses can occur asynchronously to the CPU execution. Hence, accesses to the TLB must be carefully arbitrated between the CPU and the write-buffer; the write-buffer must be emptied before displacing any TLB entry. Another alternative, is to have the software be able to handle multiple nested TLB misses.

A more radical solution is to purge the cache whenever a translation is invalidated or displaced by the replacement algorithm in the TLB. This solution requires that the TLB hit ratio be very high to limit the resulting performance degradations.

A similar approach has been adopted in the Sun 3 [Sun 85][Sun 86] and Sun 4 architectures [Kelly 85] although, in both cases the cache is write-back. To avoid the displacement of translations, the MMU is a direct hardware implementation of two or three levels of page tables. All the page table entries of an active process can be present in the MMU. By extending virtual addresses with PIDs more than one context can reside in the MMU at any one time and the frequency of purges is reduced. With this peculiar "TLB" organization, replacements are made on a process basis instead of on a page basis. When a context or part thereof is removed from the MMU the corresponding blocks of the cache must still be purged.

The efficiency of the purge depends on the cache organization. In order to invalidate the blocks belonging to different units of virtual space (page, segment or context), the comparison logic must be able to accommodate different matching sizes. The standard use of this logic is to compare the virtual address sent out by the CPU and the tag(s) contained in the cache during a memory access. Nevertheless, to invalidate a block within a page, a segment or the entire process virtual space, the matching criterion must be respectively, the virtual page number, the segment identifier or the process identifier instead of the block virtual address. If the comparison logic does not support different matching criteria, the cache must be entirely purged whenever a virtual address is demapped, which is often unacceptable in a multiprogramming environment.

The degree of associativity of the cache also affects the efficiency of purges. If the cache is fully associative all the blocks within a demapped unit (page, segment or process space) can be invalidated in one single operation provided the CAM array is accessible with various address sizes.

In a direct-map cache all the entries where blocks within the demapped area might reside must be accessed one by one and invalidated whenever the matching criterion is satisfied. If the demapped area is larger than the size of the cache, which is generally the case for a segment purge or a process context purge, all the cache entries must be examined. If the size of the demapped area is less than the size of the cache only a subset of all the cache entries needs to be checked. The size of this subset is equal to the size in cache blocks of the demapped area. The number of entries to check is given by: (cache size) / (demapped area size). The purge of the cache can be done easily by the kernel with a simple loop issuing the invalidation commands.

With a set-associative cache all the blocks satisfying the matching criterion in the same set can be invalidated in one operation with some hardware support. As for a direct-map cache the software loop performing the invalidations must check all the sets where the blocks belonging to the demapped area may reside. The number of sets to examine is determined by the size of the cache and the size of the demapped area. This number is given by: (cache size) / (demapped area size * degree of associativity).

3.1.2 Write-Back Caches

In the case of a write-back cache the blocks of the demapped area must be flushed, i.e., entries with matching tags must be invalidated **and** main memory must also be updated if they have been modified. As

for a write-through cache, blocks should be flushed before the corresponding TLB entries are invalidated since physical addresses are necessary to write dirty blocks to main memory.

In high-performance systems dirty blocks are buffered and blocks are fetched first on a cache miss, in order to service the processor faster. If the translations held in the TLB can be removed by the replacement algorithm without flushing the cache, a TLB miss may occur for the memory access of the buffered victim block. Because this type of TLB misses is asynchronous to the activity of the processor, as already explained in the preceding section, some provisions must be made to prioritize TLB accesses or to handle multiple TLB misses. Again, a drastic solution to these problems is to flush the cache when a page translation is displaced from the TLB even if the virtual-to-physical address mapping is still valid.

Another solution, not evoked previously, is to keep the physical page number of the page together with the virtual address of the block in each cache entry. The cache controller can use this copy of the PPN to construct the physical address of a victimized block. The TLB is now accessed only from the CPU and the sequencing logic is simplified.

It has been shown that the percentage of modified blocks in a write-back cache in usual programs lies between 20 and 80%, with a 50% average [Smith 85]. Since there is usually only one path between the cache and main memory, the processor must wait for the completion of each write to main memory before issuing the next flush command to the cache controller. Hence, a flush is significantly more expensive than a mere invalidation. Of course, augmenting the size of the write buffer would improve flushing efficiency. In general, flushing should be avoided when it is not necessary [Cheng 86]; sometimes, a mere purge is sufficient, for example when the information in a demapped area is not going to be re-used (case of the destruction of a segment in a single virtual space).

For a direct-map cache the flush of a portion of the virtual space is done in the same way as a purge, with the only difference that if a block has been modified the cache controller sends the block to the main memory before resetting the valid bit.

With a fully-associative cache it is not possible to flush multiple blocks in parallel because all modified blocks must be copied in main memory sequentially. All entries of a fully-associative cache must be checked whatever the size of the demapped area. The entries of the cache must be accessed randomly, and the cache controller must be equipped with an extra comparison logic to check the matching criterion. The random access path provided to load a new block must be made explicitly available to the CPU for flushing purposes.

With a set-associative cache, the flushing scheme is basically the same except that only a subset of all the sets must be accessed if the size of the demapped area is less than the size of the cache. Inside a set, all the entries could be flushed in parallel if there are as many copy-back buffers as the degree of associativity of the cache. This solution should be possible if the degree of associativity of the cache is low (2 or 4) and if the main memory bandwith is large enough. The rate at which dirty blocks are copied back to main memory must be larger than the rate at which the cache sets are flushed. The alternative is to access sequentially each entry in each selected set.

3.2 SYNONYMS

This section concerns only architectures with the private virtual space model. Synonyms are prohibited in the single virtual space model.

Virtual addresses are said to be synonyms or aliases when they all map to the same physical address (Figure 6). Synonyms introduce consistency problems in a virtual-address cache because multiple copies of the same information can be present at the same time in different cache entries. For read-only information, there is no consistency problem because all the copies are identical; the only drawback is a pollution effect in the cache if aliasing is used extensively. For modifiable information, multiple, inconsistent copies may coexist in the memory hierarchy of the system and the CPU can later access a stale copy.

In the example of Figure 6, X is writable and it is read successively with virtual address VA1 and with virtual address VA2. If the CPU changes X into X' (with virtual address VA2 in the example of Figure 6) two distinct copies are now present in the cache.

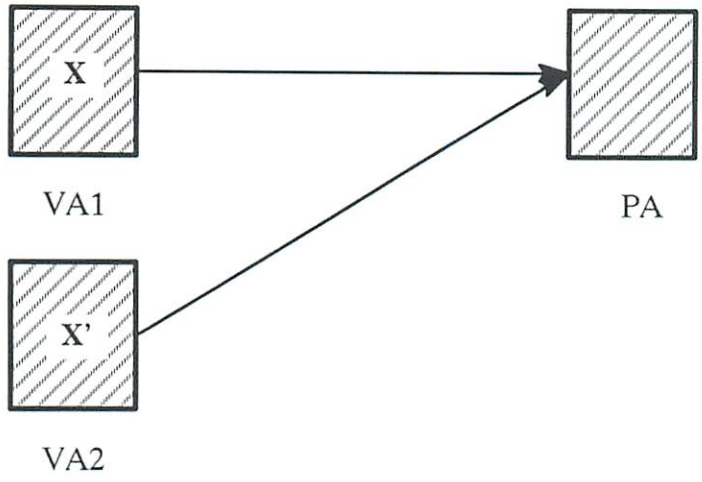


Figure 6: Synonyms

For a write-through-without-allocation policy, if the data X has never been loaded with address VA2 in the cache then, after the write, main memory contains the updated value X' while the cache keeps the stale copy X. Other more complex scenarios can be devised in the case of a copy-back cache; in these cases, the final copy in main memory may not even reflect the last modification of the processor; there may be an inconsistency even if the processor accesses the data with the same virtual address as the one of the last write.

There are different ways to solve the aliasing problem in uniprocessors.

The simplest solution is for the kernel to tag all pages known under several virtual addresses as non-cacheable. The only necessary hardware support is a cacheable/non-cacheable bit in each TLB entry. This bit determines whether a missing block should be loaded in the cache. This solution is viable only if the use of synonyms is very limited, because accesses to non-cached information are usually very slow.

Another solution is to flush entries in the cache to guarantee data consistency when the access pattern to the synonyms is totally predictable. Each time a mapping is discarded the cache can be purged or flushed. This is done in some implementations of the SunOs kernel for some I/O operations [Cheng 86]. This solution is acceptable for mapping changes that are infrequent because of the large overhead and is applicable only for the kernel.

As previously stated, synonyms of blocks belonging to read-only pages can be tolerated in a uniprocessor cache. However, if the access rights can change dynamically and one of the synonym pages become writable the kernel must flush or purge all the virtual pages that are synonyms in the cache. This is also true when a cacheable page becomes non-cacheable due to the synonym problem.

The previous solutions do not require additional hardware support for synonyms and essentially rely on software. We now consider the use of additional hardware to relieve the software of all or part of the task of maintaining consistency in the presence of synonyms.

If the hardware (the cache controller) systematically searches for synonyms of the missing block on each miss (including a write miss in a write-through-without-allocation cache) then, it can avoid the presence of multiple copies in the cache at any time. If there is no synonym of the missing block the miss handling can proceed normally, and the missing block is loaded from main memory. However, if a synonym is already present in the cache different possibilities exist depending on the write policy in the cache.

If two synonyms point to the same cache entry or set (which is always true in a fully-associative cache because there is only one set), the entry can simply be reallocated to reflect the new virtual address, i.e. only the cache tag must be changed. Otherwise, the copy must be invalidated and a new entry must be allocated by the replacement algorithm. In a write-through cache, the missing block can be copied either from main memory or from the cache. In a write-back cache the missing block must be copied from the pre-existing alias block if it is "dirty"; otherwise, the cache controller may fetch the missing block either from main memory or from the cache.

Alias detection is a difficult problem. In the following, we discuss possible approaches for different cache organizations.

3.2.1 Direct-Map Caches

The access to a block in a direct-map cache consists of two phases: a direct indexing of an entry followed by a comparison between the CPU address and the tag. If the cache size is less than the page size, all synonyms map into the same cache entry because synonyms belong to different pages. When a miss occurs, the presence of an alias in the cache entry is detected by translating the cache tag and by comparing the resulting physical address to the physical address of the missing block. In a write-back cache, an address translation is needed whenever the displaced block is dirty; because of synonyms this translation must be done whether or not the block is dirty.

The hardware support for this simple solution amounts to a comparator and a slightly more sophisticated cache controller.

In most modern systems, the cache size is larger than the page size; therefore, some of the bits used for indexing are not common to the virtual and to the physical addresses. The size of the cache in page units is equal to the total number of entries where synonyms can reside. The set composed of all these entries has been called the *superset* in [Goodman 87] (Figure 7). We will adopt the same terminology in this paper. To avoid exploring all the entries of the *superset*, a restriction can be imposed on the software to allow synonyms modulo the size of the cache only. In this way, all synonyms map to the same cache entry. This solution is adopted in the Sun 3/200 line of workstations [Van Loo 87] and the Apollo DN4000 workstation [Frink 88]. The consistency of pages for which the kernel cannot control the virtual-to-physical mapping (for example when the user requests a specific mapping) must be guaranteed by one of the "software" solutions exposed previously (non-cacheable pages, or systematic flushing).

When no restriction is placed on virtual addresses all the cache locations of the *superset* must be checked. If the size of the *superset* is too large it may not be possible to access and translate in the TLB all the virtual block addresses of the *superset* elements one by one while the missing block is being fetched from main memory.

One way to speed up alias detection in this case is to keep the physical address of the block along with its virtual address in the cache. The cache is still accessed with the virtual address but the physical addresses of the *superset* elements can be compared with the physical address of the missing block while waiting for main memory to respond. This solution is a way around a slow TLB, but it is costly in terms of cache space.

It can be quite effective for cache implementations in which all the elements of a *superset* can be accessed simultaneously and the physical address of the missing block can be compared with all the physical addresses of the *superset* entries in parallel.

The other solution is to have a dual cache directory accessible with physical addresses. This is different from the previous solutions in the sense that now a *reverse translation scheme* is applied: the *physical* address of the missing block is used to access the dual directory and detect the presence of an alias. In previous solution(s) alias detection was done with a *direct translation scheme,* i.e. the physical tags were

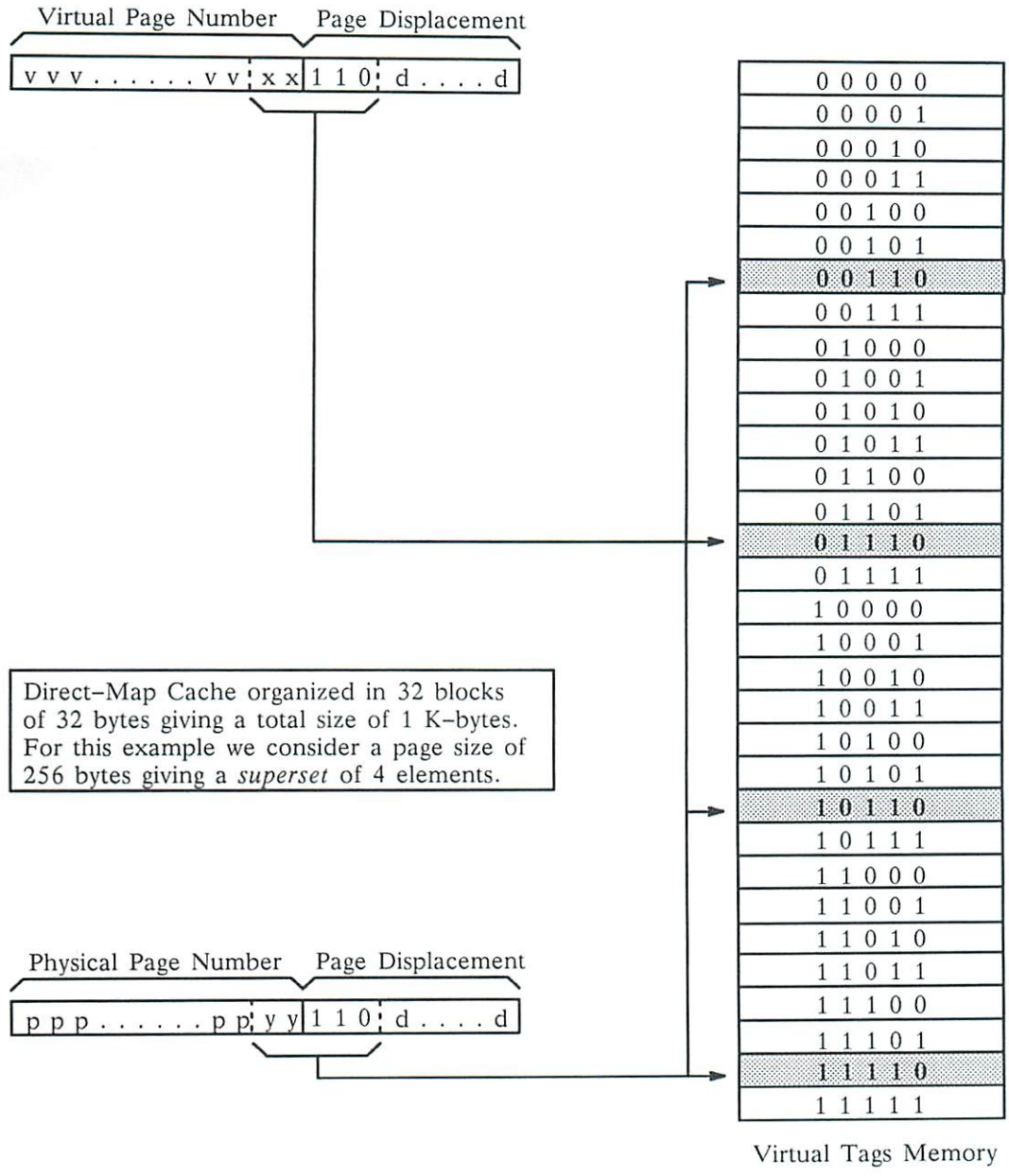


Figure 7: *Superset* Example

accessed with virtual addresses. We examine this solution in more details in the section dealing with multi-processors.

3.2.2 Set-Associative Caches

The solutions applicable to set-associative caches are fundamentally the same as those described in the previous paragraph because a direct-map cache is a set-associative cache with a single entry per set. However, for various cache sizes, the tradeoffs can be very different.

It is not possible to enforce the mapping of synonyms to the same cache entry because the search inside a set is associative. Nevertheless, the problem can be greatly simplified if all the synonyms map to the same cache set; the bits used to index the set may not be translated, i.e. they must all belong to the

page displacement. Therefore, the size of the cache is limited by the product of the degree of associativity by the page size. The degree of associativity is the number of elements in a set. This solution has been adopted for the cache of the Amdhal 470/V7 (32 K-bytes size, eight-way set associative, 32 bytes lines).

If the cache organization does not enforce the mapping of all synonyms to the same set it is still possible to rely on software. Virtual addresses that are synonyms could be allocated such that they select the same cache set, i.e. they are modulo the ratio of the cache size and of the degree of associativity. For a given cache size, the restriction on the mapping of virtual addresses is less severe in a set-associative cache than in a direct-map cache.

The presence of a synonym must still be checked within the selected set while the missing block is fetched from main memory. If a synonym is present there is no need to victimize one block, but the content of the tag must be changed. Alias detection can be made by translating the virtual address of each set entry sequentially in the TLB provided this whole process can be overlapped with the access to main memory. Otherwise, the addition of the block physical address in the tag or the use of a physical address dual directory as described previously must be envisioned.

When it is not possible to constrain synonyms to reside in the same set, all the cache sets forming the *superset* must be explored. The performance is affected unless all the sets in a *superset* are accessible in parallel; the cache must be split in as many modules as there are sets in a *superset* and all the modules must be accessed at the same time. On a cache miss a synonym detection must be made, probably with a *reverse translation scheme* for performance reason. This solution implies that many hardware resources, such as the comparison logic, must be duplicated. Thus, it is best suited for very high performance systems in which the size of the cache must be very large and for systems running existing software which uses synonyms without restrictions.

3.2.3 Fully-Associative Caches

In a fully-associative cache there is no indexing, and synonyms can reside in any cache entry. Hence, a *direct translation scheme* for alias detection is not feasible. The only solution is to add a dual cache directory containing all the physical addresses of the cache blocks. This solution is discussed in the section on multi-processor architectures.

3.3 SUPPORT FOR MEMORY MANAGEMENT AND PROTECTION

A page table entry and its corresponding TLB entry maintain bit fields for efficient memory management and for protection enforcement. Some of these fields must also be present in the virtual-address cache. In this section we examine how a virtual-address cache interacts with the Translation Lookaside Buffer. The implementation is greatly simplified when the TLB and the cache have the same access time.

3.3.1 "Fast" TLB

If the access time of the TLB is less than (or equal to) the cache access time the dynamic checking of access rights and the handling of statistical bits for memory management support need not to be moved to the virtual-address cache; the cache and the TLB are accessed in parallel on each memory reference and their contents are combined. Besides these savings in hardware complexity, overall performance of memory accesses is improved because of the reduction of the miss penalty. Also, in a write-through cache, bursts of memory writes can be executed at cache speed (if the TLB is slower than the cache but can be pipelined so that its cycle time is the same as the cache, then the advantages of this implementation can still be reaped.)

Because of demappings and remappings of virtual addresses the management of the TLB and the cache cannot be totally decoupled. In order to keep the cache and TLB controllers simple, the cache should preferably be flushed/purged whenever a translation is displaced or invalidated in the TLB. There-

fore a copy of the *reference* bit should be included in each TLB entry to reduce the number of invalidations in the TLB. Nevertheless, if the resetting of the *reference* bits is done globally for all the entries of all the page tables (i.e. the page replacement algorithm is global and not local to each process) it could be acceptable to purge all the non locked TLB entries and flush/purge the whole cache due to the very infrequent nature of this event.

The possible implementations of the *modify* bit as well as of the access rights bits are the same as for a physical address cache and all the usual solutions are applicable.

3.3.2 "Slow" TLB

The most attractive feature of a virtual-address cache is that it works well with a slow TLB. Therefore the TLB does not need to be integrated on the same chip as the CPU and can be implemented as an external unit with off-the-shelf components. In this case, the TLB can be huge because there is no silicon area restriction.

However, when the access to the TLB is slower than the access to the virtual-address cache, some functions that are usually handled by the TLB, such as access-right checking, must be taken care of in the cache. Now, the cache is the only mandatory access path for most memory references; thus, besides the virtual address of the block, the cache directory must also hold a copy of the access right fields found in the TLB entries to support protection. The cache controller must take into account the current priority level of the processor and the type of access, matching them with the access rights contained in the tag in order to validate the access. This can be done in parallel with the tag(s) comparison and usually does not slow down the cache access.

In some architectures, the protection scheme is very sophisticated; for example there may be several protection levels (ring) or a *key/lock* mechanism (as described in section 2.2.2) to enforce. In these cases, the tag sizes must be greatly extended to hold access rights for each privilege level and/or the *lock* to the segment. Because the protection at the segment and at the page levels must be implemented at the cache block level, the hardware cost to support the protection is much higher than for a physical-address cache. The protection fields in the cache tags cannot replace those in the TLB entries; they are only copies. Consistency between these copies must be enforced by the usual flushing mechanism if the protection of a page and/or a segment changes dynamically.

Statistical bits needed to optimize memory management are the *modify* bit and the *reference* bit. The *modify* bit does not need to be copied in cache. In a write-through cache the TLB is accessed on each processor write and the *modify* bit in the TLB is managed as in a physical-address cache. The write-access protection bit and the *modify* bit cannot be merged as it is done in some architectures such as the R2000 from MIPS Computer Systems [DeMoney 86] as it would incur a severe performance degradation. To enforce the protection a merged write-access/modify bit must reside inside each virtual cache tag which would cause each first write to blocks loaded in cache before the page is marked as modified to raise a trap.

In a write-back cache each cache tag holds a *dirty* bit that is set on the first modification of a part of the block by the processor. The value of this bit indicates whether the block must be copied back to main memory when it is victimized. One can decide to set the *modify* bit of a TLB entry whenever the processor modifies a block of the page for the first time, as indicated by the *dirty* bit. In this scheme, there will be many redundant settings of the *modify* bits in the TLB entries. However, this may be acceptable because the writes are only a small fraction of all memory references and the average access rate to the cache is usually much less than its total bandwith; except for bursts of writes to different blocks, conflicts at the TLB due to the updating of *modify* bits should be very infrequent.

Another possible design is to maintain a copy of the *modify* bit in the cache tags. This solution reduces the overhead of updating the copy in the TLB but does not eliminate it. All the blocks that have

been loaded before the first modification of the page trigger a redundant setting operation when they are later modified by the CPU.

In all cases, the updating of the copy kept in a page table entry can be done in a write-through or write-back manner as already explained in the section introducing TLB issues.

With a virtual-address write-back cache, the *reference* bits in the TLB entries are updated on a cache miss only. The *reference* bit of the page to which the missing block belongs is checked during the miss handling and set if necessary. In a write-through cache the checking and possible setting of the *reference* bit in the TLB can also be done on each processor write. Therefore, with this implementation, the exact usage of the pages is not reflected in the *reference* bit. However, this approximation does not affect the overall performance of the virtual memory system noticeably.

Other alternatives exposed in Section 2.3 can also be applied but they are not really cost/effective. Hence, they should not be considered as a way of supporting the page replacement algorithm.

4. VIRTUAL-ADDRESS CACHES IN MULTIPROCESSORS

4.1 INTRODUCTION

We consider shared-memory multiprocessors where the interconnection between the processors and the main memory is a single bus (Figure 8). A private cache associated with each processor can significantly increase memory bandwidth and reduce memory access time. The main issue in this type of architecture is to guarantee the coherency of the information stored in the *shared-memory image*. Recent studies have shown that many different solutions are possible [Archibald 85][Papamarcos 84][Frank 84][Katz 85][McCreight

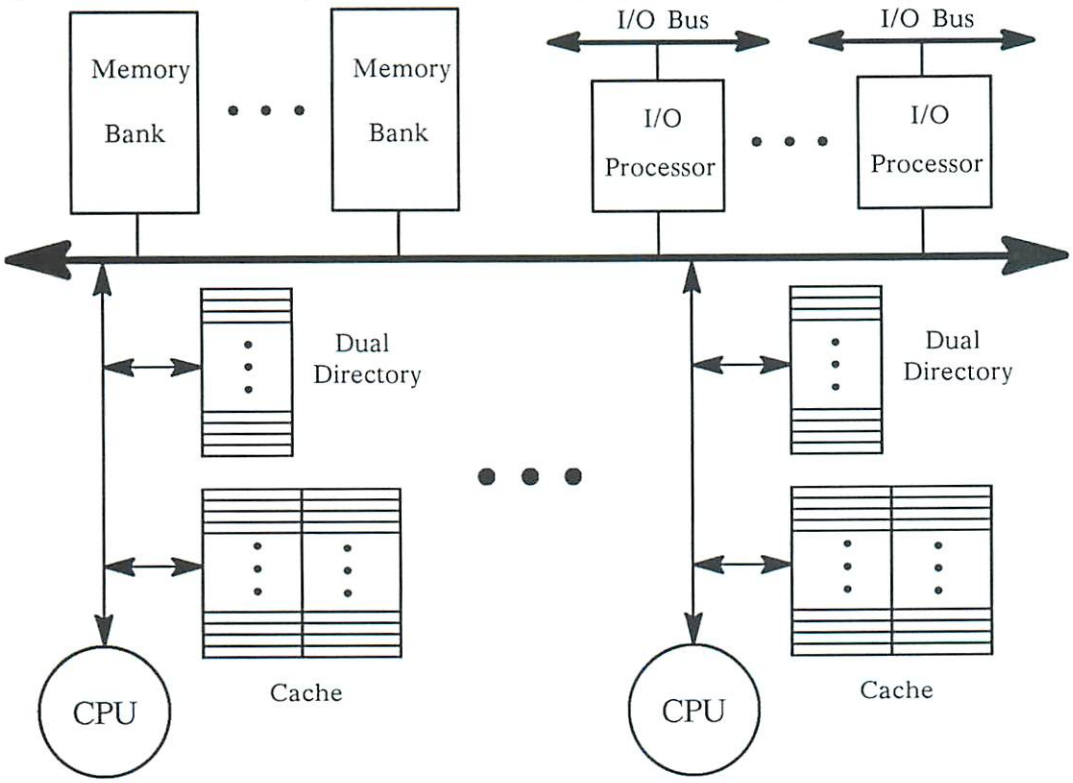


Figure 8: Single-Bus Shared-Memory Architecture

84]. Readers who are not familiar with cache consistency protocols can consult [Sweazey 86] for a good synthetic presentation of most of the published cache consistency protocols.

All the cache consistency protocols for single-bus multiprocessors suppose that every bus transaction is "watched" by all processors to check if their cache has a copy of the referenced information. For this reason, they are often designated informally as *snooping* protocols. To monitor bus transactions, it is necessary to duplicate the cache directory, at least in high performance systems. The copy of the cache tags is called the dual directory (Figure 8). It is used by the bus interface to "filter" the bus transactions without perturbing the activity of the local processor, except when the local cache has to be updated. Although a dual-ported cache directory is sufficient to support a snooping cache consistency protocol, we consider that for high-performance system (where virtual-address caches do make sense) the dual directory is absolutely required.

For performance reasons, we also assume that the *snooping* consistency protocol is implemented entirely in hardware and is totally transparent to the software. However, the Stanford VMP machine adopted an approach where misses in the virtual-address caches and the cache consistency protocol are handled in software [Cheriton 86][Cheriton 88]. We do not consider this approach in this paper.

4.2 MAIN MEMORY UPDATING POLICY

The main memory updating policy of the cache has a dramatic effect on the traffic generated by each processor on the bus. With a write-through cache all the processor stores access the bus and the main memory. With a write-back cache only the dirty blocks are copied to main memory when they are victimized. Most instruction mixes show a write frequency of 10 to 30% of all memory references and trace-driven simulations of copy-back caches show a fraction of dirty blocks between 20 and 50% [Smith 85]. Thus, the traffic generated by a copy-back cache is larger only if the hit ratio is low.

It is obvious that to support a large number of fast processors on a single bus a write-back cache is the best approach, and this has been demonstrated with the Balance 8000 and Symmetry systems from Sequent Computer Systems [Mayberry 84][Lovett 88]. Hence, high-performance multiprocessor systems are most likely to adopt a write-back updating policy for their caches. However the issues discussed in this section and the proposed solutions concern both write-through and write-back caches. We suppose that in all cases, there is a dual directory in the bus interface to help maintain consistency, although other solutions are possible, especially for write-through caches [Dubois 82].

4.3 VIRTUAL OR PHYSICAL ADDRESS BUS

In a system with the private virtual space model the bus must carry physical addresses, because of the synonym problem. When two processes running on distinct processors share information, they access it with different virtual addresses in general. Thus, it is not possible to snoop on the virtual addresses. Before accessing the bus the processor must translate virtual addresses in the TLB.

However, with a single virtual space the synonym problem does not exist and it is possible to envision a *virtual address bus*, i.e. a bus carrying virtual addresses. In this case, the translation of virtual addresses to physical addresses should be done at the main memory. In the logical representation of Figure 8, each memory bank contains a TLB to translate virtual addresses into physical addresses. The complexity of this design depends on the memory interleaving.

If memory banks are selected only by fields of the VPN, then all cache blocks of a given page reside in the same memory bank and each TLB maps the part of the physical space in its associated memory bank. In this architecture, there would only be one copy of each VPN/PPN couple in the system and the TLB consistency problems existing in multiprocessor systems is eliminated.

If the TLB is fully-associative and large enough to contain all the translations for its memory bank, there is no more need for the *inverted page table* and a TLB miss becomes identical to a page fault. However, such a large, fully-associative TLB is probably not feasible. With a set-associative TLB a page of the virtual space can logically be allocated to any page frame but translation tables are still necessary. Otherwise, a page would have to be displaced when the corresponding translation is displaced in the TLB. Trashing effects in the TLB would cause unnecessary page replacements. If the TLB degree of associativity is sufficient, it might be possible to unify a page fault and a TLB miss.

It should be noted that in the single virtual space model, main memory usually behaves as a true fully-associative cache relative to the secondary storage (the files are all mapped in the virtual space). Each page can be allocated to any page frame in the main memory. But, with the preceding organization, main memory becomes a set-associative cache, a set being a memory bank. In this case, the size of a set should be large enough to prevent any thrashing effect due to this particular organization.

If the memory bank is selected with bits of the page displacement, then blocks in the same page are scattered across different memory banks. In this case, all the TLB units must contain the same translations and coherence must be maintained.

A virtual bus is implemented in the HP 9000 multiprocessor architecture [Beyers 85][Lob 85] but this system is cacheless. A virtual-address bus for cache-based single-bus multiprocessors is a very attractive design but more studies on the paging behavior of the resulting memory hierarchies are needed. In the rest of this paper we assume that the bus carries physical addresses. Thus, the dual directory used for snooping contains the physical addresses of the cache blocks and for this reason is also called the physical directory in the following.

4.4 VIRTUAL-TO-PHYSICAL DIRECTORY BINDING

4.4.1 Introduction

In the usual operating mode, the cache is accessed by the processor with virtual addresses while the dual directory is accessed from the bus with physical addresses. However, some accesses to the dual directory must also reach the cache, and vice versa. Therefore, a binding must be defined between the entries in the cache directory and in the dual directory pointing to the same cache block. This binding is particularly tedious when the directories are set-associative or direct-mapped. The following solution is described in [Goodman 87].

4.4.2 Set-Associative and Direct-Map Directories

Let suppose first that both directories have the same organization. When the size of the set-associative (or direct-map) cache is larger than the product of the page size by the degree of associativity, the indexed set (or entry) of each directory can be different, because some of the bits used for indexing are translated. However, we know that both selected sets in the case of a set-associative cache or entries in the case of a direct-map cache belong to the same *superset*.

To access the entry holding the physical address of a block in the dual directory it is necessary to keep a pointer in the cache tags. This pointer is made of the bits of the physical address indexing the set inside the *superset*. Some bits locating the block within the set must also be kept. Conversely, in each entry of the dual directory a pointer made of the bits of the virtual address indexing the set inside the *superset* and some bits locating the block within the set must be kept. With a direct-map cache only the index of the set inside the *superset* is necessary. With these pointers, the virtual directory can now be accessed through the physical directory and vice versa, but this requires one associative access followed by a "random" access. With a set-associative organization, if the pointers in the virtual and physical directories are respectively the entire physical and virtual address of the block, the second access can be associative.

An example for a direct-map organization is depicted on Figure 9.

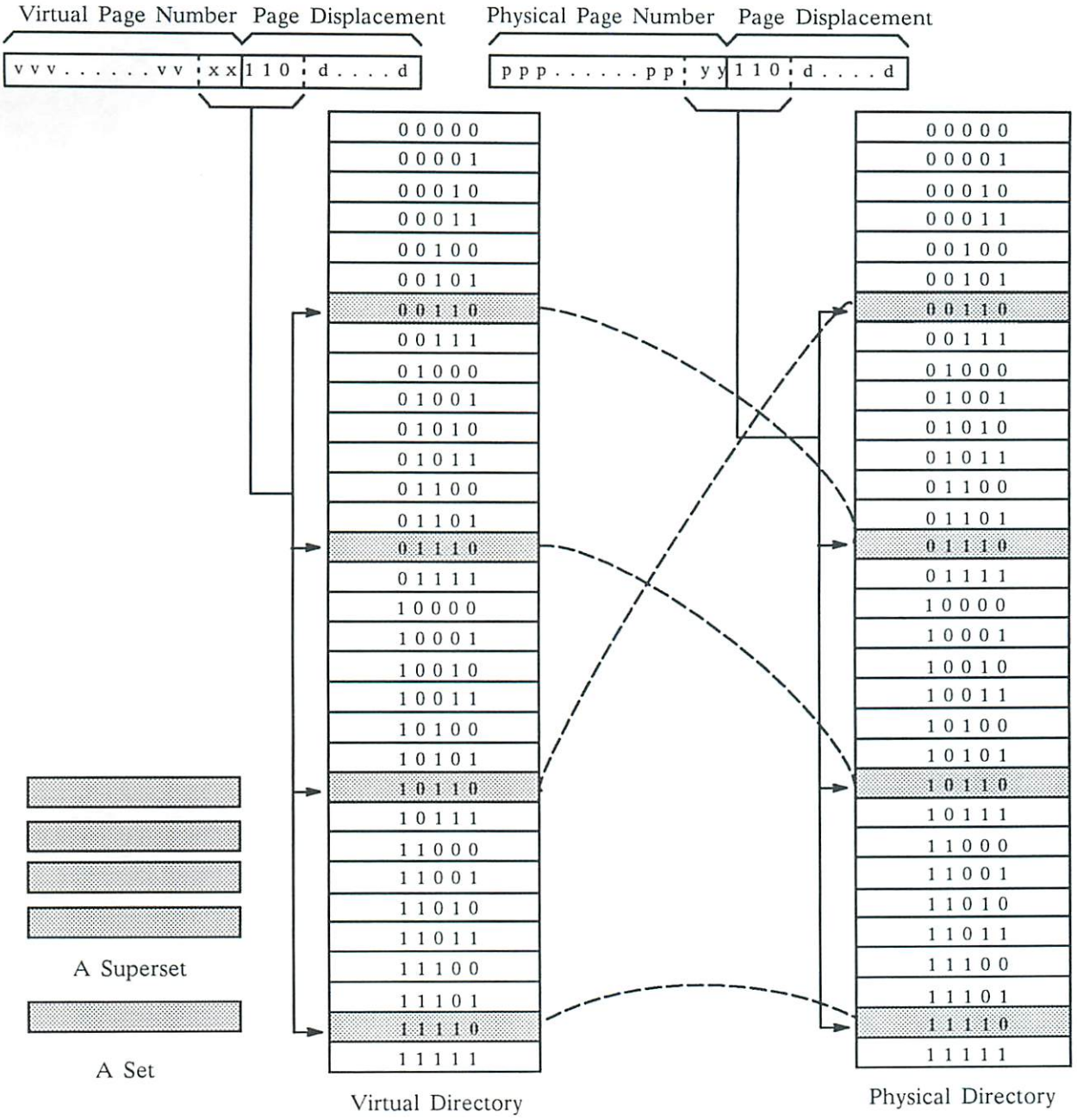


Figure 9: Virtual-to-Physical Directory Binding

When there is a miss in the cache, one of the blocks of the selected set must be victimized. In the dual directory a distinct entry must also be victimized if the physical address of the displaced block and the physical address of the missing block do not index in the same set. Therefore in some cases two blocks of

the cache must be allocated to a missing block. To choose a victim in the selected set of the dual directory a simple, purely random selection algorithm can be applied. Another block may have to be displaced in the cache due to this second allocation. If the main memory is updated with a write-back policy it can happen that the two victim blocks have to be copied back. To reduce the average miss penalty the write-back buffer should be able to accommodate two blocks instead of one.

It therefore appears that the logic of the cache controller and the bus interface are more complex with a virtual-address cache than with a physical-address cache in a multiprocessor system.

4.4.3 Cache Occupancy

With the above organization, the occupancy ratio of the cache in steady state is less than 100% and this under-utilization affects the hit ratio. The occupancy ratio must be estimated in order to validate this organization.

We consider in first approximation that there is no correlation between the bit fields selecting the set inside the *superset* in the VPN and in the PPN. This is a reasonable assumption because the page replacement algorithm is oblivious to page frame addresses. Similarly we assume that the replacement algorithm in the cache and the dual directory is random. Then, the probability that the missing block and the victimized block have physical addresses indexing in the same set of the dual directory is equal to: q=1/N, where N is the number of sets in a *superset*. If C is the cache size in blocks and k is the number of useful blocks in the cache, we have the following relation: $k^*q + k^*(1-q)^*2 = C.$

Thus, the occupancy ratio of the cache is given by: $k/C = 1/(2-q) = N/(2N-1).$

A formal proof of this relation is given in appendix of this paper. It should be noted the previous relation is true whatever the degree of associativity, provided the replacement is random in both directories.

The chart of Figure 10 represents the occupancy ratio as a function of the *superset* size. It appears clearly that the cache is under-utilized. Even with only two sets per *superset*, one third of the capacity of the cache is wasted but rapidly nearly half of the cache is wasted.

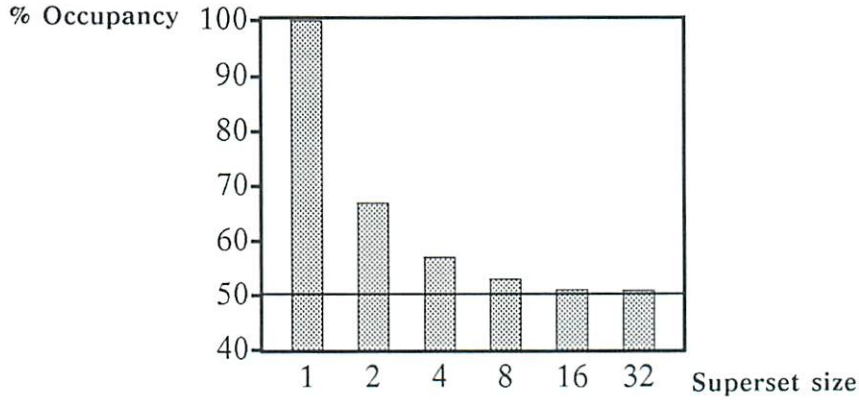


Figure 10: Cache occupancy ratio vs *superset* size / Random Replacement

4.4.4 Critical Associativity

The deleterious effect on the occupancy ratio can be eliminated provided the degree of associativity of at least one of the two directories is equal to or greater than the *critical associativity*.

For a given cache size the *critical associativity* is defined as: [Cache Size] / [Page Size]. In this case, the set selection is done only with bits belonging to the page displacement and all blocks that are

synonym map in the same set. Since timing constraints are usually less stringent at the bus interface, the best solution is to increase the associativity of the physical directory. A pointer must still be kept inside each entry of both directories. The size (in bits) of these pointers depends on the particular organization of the directories.

This solution can be extended up to the point where both directories are fully–associative. In this case, a full pointer to address "randomly" the cache blocks must also be held in each entry of the dual directory. If both CAM arrays implementing the directories and the RAM array implementing the data memory, are integrated on the same chip these pointers are not necessary. The match lines of the CAM can directly feed the RAM array decoders. Then, the binding of the two directories is guaranteed by the hardware implementation.

4.4.5 Replacement Algorithm

When the replacement algorithm preferably allocates an invalid entry in one or both directories to a missing block, the occupancy ratio improves significantly. Figure 11 presents results of simulations done for various organizations of the cache and the dual–directory. The various graphs display the cache occupancy ratio as a function of the superset size. The superset size is expressed in number of sets.

In all cases, the replacement algorithm takes the following steps to allocate an entry when a missing block is loaded into the cache:

- If invalid entries are found in the selected set of both directories they are allocated and linked.
- If an invalid entry is found in only one of the directories, this entry is allocated and an entry is picked at random in the other directory.
- If no invalid entry is found in the selected sets of both directories, the replacement algorithm tries to find a pair of entries which are linked together.
- Finally, when there is no other alternative, the replacement algorithm picks an entry at random in both directories.

This replacement algorithm is optimal relative to the cache occupancy. It displaces two blocks only when there is no other alternative. However, it is not optimal for the cache hit ratio as valid entries are picked at random. LRU (Least Recently Used) and FIFO (First–In First–Out) algorithms are known to give better results.

The graphs of Figure 11–a depict the occupancy ratio for an architecture where both directories have the same degree of associativity. It is important to note the occupancy ratio remains 100 % until the superset size is strictly larger than the degree of associativity. This effect can be explained simply. After a transient period where the cache is filled up, all entries of any set are linked with entries of all sets of the superset. At this point, the replacement algorithm always find a pair of linked entries to displace in favor of the missing block.

Figure 11–b corresponds to architecture where one of the directories is direct–map while the other is set–associative. The results obtained through simulations prove the organization of the cache directory and the dual directory can be interchanged.

Finally, Figure 11–c displays the occupancy ratios of architectures where both directories are set–associative but have different degree of associativity. As before there is a symmetry. The same results are obtained when the organization of both directories are interchanged. The occupancy ratio departs from 100 % when the superset size is strictly larger than the lowest of the two degrees of associativity.

Because the implementation of such a sophisticated replacement algorithm is complex the results presented here should be considered very carefully by cache designers. Although this algorithm is

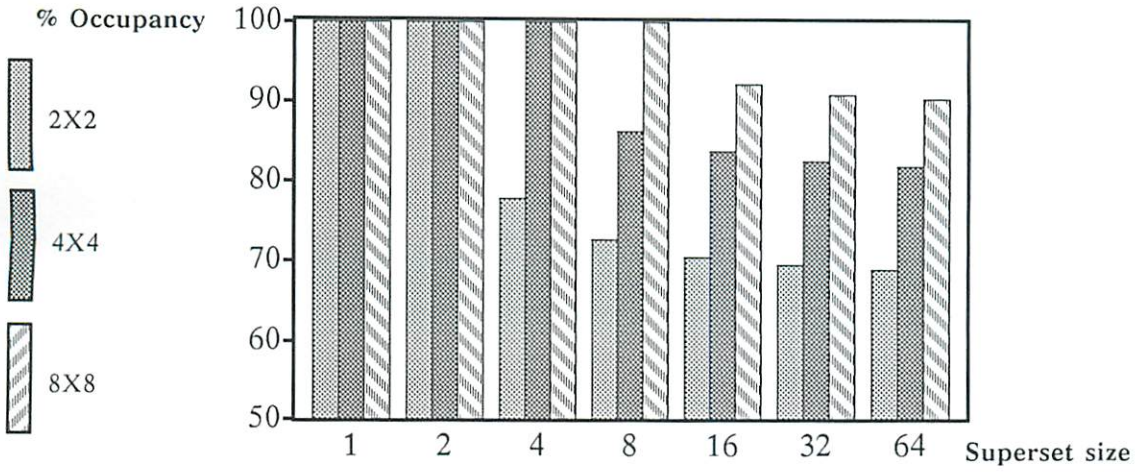


Figure 11-a

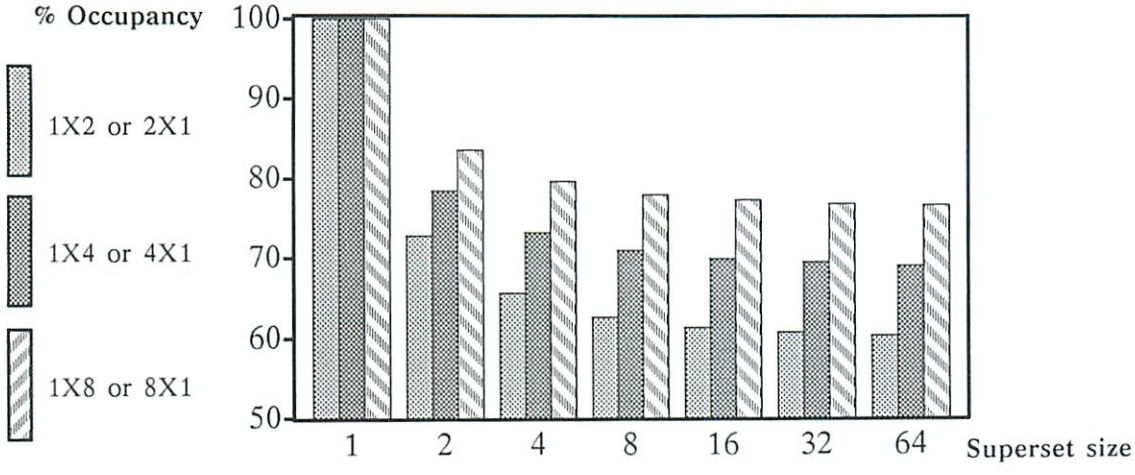


Figure 11-b

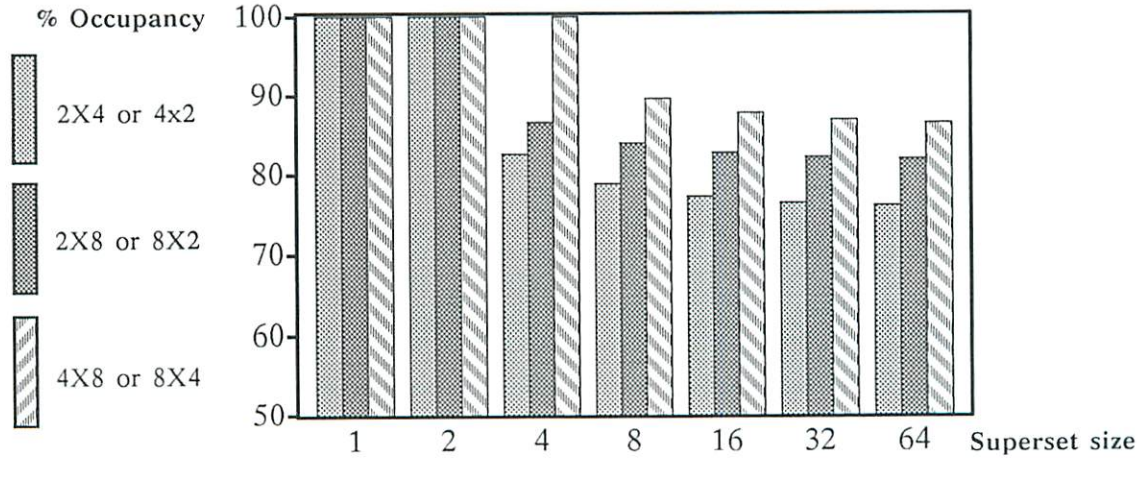


Figure 11-c

Figure 11: Cache occupancy

optimal for the cache occupancy, the effect it may have on the hit ratio is not clear. A simple LRU algorithm might give better overall performance. In the following section and in section 4.4.8, we examine other solutions to the cache occupancy problem which do not require a specific replacement policy.

4.4.6 Virtual Indexing of the Dual Directory

Another way to solve the binding problem in the case of a single virtual space is to index both directories with virtual addresses. Therefore, all bus transactions must pass with the physical address the bits of the virtual address selecting the set inside the *superset*. The dual directory still contains the physical addresses but the set is selected with the same bits as those used to index in the cache. With this scheme, the virtual and physical addresses of the cache blocks are always in the same set and there is no more need to victimize two blocks on a replacement. Moreover, now the binding of the two directories is becoming very simple because only the bits selecting the block inside the set are necessary.

It is very important to note that this organization is viable only because there are no synonyms in a single virtual space. This approach can also be applied to a multiprocessor with a private virtual space model *if synonyms are restricted*. As already evoked in Section 3.2, it is possible to impose a restriction on page frame allocation such that all synonyms are modulo the ratio of the cache size and of the degree of associativity. In this case, they all map to the same cache set and the snooping cache consistency protocol can always detect information sharing.

With this solution, all DMA I/O bus transactions must also pass the virtual address bits selecting the set inside the superset. This means that the I/O subsystem must be equipped with a mapping support device (i.e. TLB or MMU) and abides by the restriction on synonyms. When this restriction cannot be guaranteed or if the I/O subsystem uses only physical addresses, all pages where a DMA I/O operation takes place must be flushed from the caches before the beginning of the I/O transfer. These pages must be also be marked as non-cacheable for the duration of the I/O transfer to avoid any data inconsistency.

This approach is very appealing because it simplifies the hardware while providing a 100% cache occupancy. However, the restriction imposed on synonyms must be manageable by the software. The next section examines the implications of the restriction on synonyms in a multiprocessor system using a private virtual space model.

4.4.7 Restriction on Synonyms

When the kernel cannot control the virtual-to-physical mapping of shareable pages, the page must be made non-cacheable after a flush (write-back caches) or a purge (write-through caches) of the cache of the processors which were accessing its content. Thus, with the organization described above, a flush mechanism is still necessary although we will see in Section 4.6 that flushing can be generally avoided in a multiprocessor system when a virtual-to-physical mapping is modified.

It is not possible to rely on systematic flushing to guarantee the consistency of information in the presence of synonyms, as in a uniprocessor system (Section 3.2). In a true symmetric multiprocessor system the kernel can run on any processor. Hence, it becomes very difficult to control the access pattern to the synonyms.

Sharing in a multiprocessor system can occur in the kernel because there is only one set of data structures and because the kernel can run on any processor. However, the kernel space is common to all processes, i.e. all virtual spaces (Figure 1), thus the shared information is accessed with the same addresses making all the kernel data structures cacheable. Sharing in the user process space is necessary when the execution of a single job is executed as a set of cooperating processes. This partitioning is generally called *multithreading*.

In most existing operating systems, the overhead of scheduling a process is very high and the private space associated with each process does not facilitate the sharing of information. Hence, the classical

concept of process, as it is defined in UNIX for example [Bach 86], is not a good support for parallel programming. New operating systems, such as Mach and SunOs 4.0 supports the notion of *threads* [Rashid 87] or *lightweight processes* [Kepecs 85]. A thread or lightweight process is the most basic unit of CPU utilization. Threads or lightweight processes can share the same virtual space and communicate via shared memory. Therefore, a multithreaded application running in parallel on different processors does not need to use synonyms to access shared information.

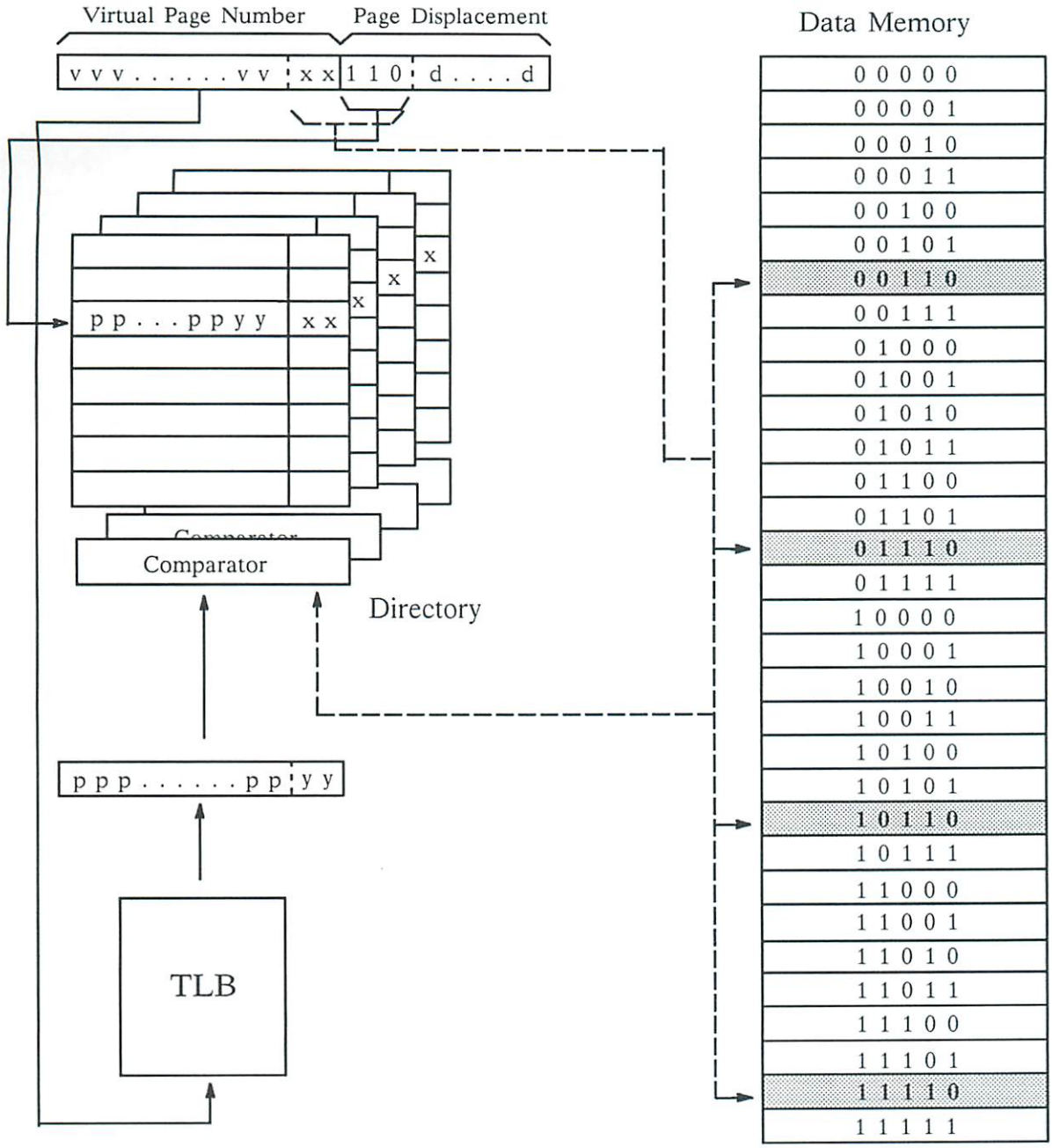
To conclude this section, the restriction imposed on the synonyms does not appear to be constraining in a multiprocessor system using a private virtual space model.

4.4.8 Hybrid Virtual/Physical Address Cache

With the new generation of RISC processors, the cache is expected to deliver the requested data to the processor every clock cycle. With a direct-map organization it is possible to anticipate the result of the tag comparison and deliver immediately the indexed block to the processor on reads. If there is a cache miss, the processor is notified during the following cycle. At this time, generally no internal state of the processor has changed, and the processor can be stalled by the cache controller while the miss is handled. This scheme is used in two implementations of the Sun Microsystems' Scalable Processor Architecture (SPARC): the MB86900 from Fujitsu Semiconductor [Namjoo 88] and the CY7C600 from Cypress Semiconductor [Cypress 88].

This scheme can be extended to a set-associative organization as it has been proposed in [Chang 87]. In this solution, the Most Recently Used (MRU) block of the set is delivered to the processor at the end of the first clock cycle before the tags are compared with the address sent out by the processor. If the data fetched by the processor is in the selected set but is not in the MRU block, the processor is notified and the right data is delivered one or two cycles later.

All these schemes anticipate the result of the tag comparison by "making a guess" on the indexed block (case of direct-map cache) or one of the blocks of the selected set (MRU approach). In the previous section, we examined a solution where the dual directory is virtually indexed while the tags are physical. It is possible to combine this virtual indexing scheme with a guessing scheme to obtain an *hybrid Virtual/Physical Address Cache*. In this organization, the data memory is direct-mapped and virtually indexed (Figure 12). The indexed block is delivered immediately to the processor and the address translation and the tag matching are performed in parallel [Tucker 86]. The directory has a different organization from the data memory. The degree of associativity of the tag memory must be at least equal to the critical associativity. In the example of Figure 12, this memory is 4-way set-associative.The next paragraph describes the principles of operation of such a cache.



Page = 256 bytes
Cache size : 1 K-bytes
Block size : 32 bytes

Figure 12: Hybrid Virtual/Physical Cache

The data memory is virtually indexed and the selected block is delivered to the processor at the end of the current cycle. The directory is accessed with physical addresses which requires a preliminary address translation through the TLB. Along with the block physical address, the bits of the virtual address

selecting the set inside the superset are kept in each directory entry. When the tag comparison is performed these bits are matched against the same bit field of the virtual address issued by the processor. If there is match on both the physical address and the virtual bit field, the cache access is a hit. Then, the virtually indexed block already delivered to the processor was a good guess and no further actions are necessary. If there is no match on the physical address the cache access is a miss. This means the virtually indexed block was a wrong guess and the missing block must be fetched from main memory.

If there is a match on the physical address but not on the virtual bit field, it means that a synonym of the requested data has been delivered to the processor. Because all synonyms of a given address are in the same superset and because the degree of associativity of the directory is at least equal to the critical associativity, the synonym detection can be done on the fly. The information requested by the processor is inside the set but is not in the right entry of the data memory. This is a *short-miss* because no access to the main memory is necessary to handle it. The data memory must be re-accessed with the right index built by concatenating the bits of the page displacement that are part of the data memory index with the virtual bits associated to the matching tag. It is very likely that the processor is going to re-access the same block in the next future with the same virtual address thus, the block must be moved in the data memory and the tag updated with the new virtual bits. The block displaced by this move can be invalidated or it can be exchanged with the accessed block. Future accesses to the "displaced" block will provoke a *short-miss*.

In a multiprocessor system, the dual directory must have the same organization as the cache directory. No binding between both directories is needed. Moreover, no restriction on synonyms is required. The hybrid virtual/physical address cache behaves exactly as a physical-address cache.

This hybrid virtual/physical organization is very attractive because it provides the speed of a virtual address cache and the transparency of a physical address cache. It is very well suited for a multiprocessor environment because short-misses do not induce any traffic on the bus.

4.4.9 State Bits

Some state bits for the support of the cache consistency protocol must be present in both directories. The state bits maintained in the physical directory entries must at least specify whether the copy held in the cache is valid and whether it is different from the one in main memory (case of a write-back cache) so that the bus interface can intervene on a request from the bus. In each cache directory entry a bit must specify if the block is shared or not so that the cache controller can decide if a store sent out by the processor can take place immediately or must be preceded by a request for ownership of the block. All the other indicators (Valid bit, Modify bit, etc...), discussed previously in the section dealing with a single processor environment, must still be present.

It is important to understand that the valid bit in the entry of the dual directory and the valid bit in the entry of the cache directory may have different values. When the block is discarded, for example by the consistency protocol, they both must be cleared. However, when the virtual-to-physical mapping of the block is changed it is possible to clear only the valid bit in the cache directory. The cache consistency protocol ensures that the block still belongs to the *shared-memory image*. If the block is dirty, the cache consistency protocol ensures that requests for a copy coming from other processors or I/O devices will be satisfied by this cache. When a dirty block is displaced in a write-back cache, the value of the valid bit in the dual directory specifies if it actually must be recopied in main memory.

In the following we use the term *invalidated* to specify that only the valid bit in the cache directory is cleared and the term *discarded* to specify that both valid bits are cleared.

We do not discuss the detailed implementation of a cache consistency protocol because it is "orthogonal" to the issues discussed here.

4.5 SYNONYMS

In the private virtual space model, a check for the presence of a synonym in the cache must be made whenever a missing block is fetched from the shared-memory image.

In a multiprocessor, a *reverse translation* through the physical directory is the right approach to detect the presence of a synonym when a miss occurs in the cache. The cache controller must access the dual directory with the virtual address derived from the PPN obtained from the TLB and the displacement in the page.

As already explained in Section 3.2 different courses of action are possible after the detection of a synonym. With a write-through cache the missing block can be either loaded from main memory or recopied from the existing copy in the cache. If both synonyms index in the same set, which is always the case when there is a restriction on synonyms to avoid cache under-utilization (Section 4.4.3), the tag can be changed to the new virtual address. In this case no block needs to be displaced.

With a write-back cache if the synonym block present in the cache is not dirty the same approach is applicable. However, if the displaced block is dirty it must be recopied to the new entry when both synonyms do not index in the same set (again, only the tag must be changed when they do).

When the block is moved from one set to the other, only one block must be displaced because the entry in the dual directory does not need to be moved. Only the pointer to the cache entry must be changed.

In a single-bus multiprocessor system limiting the traffic on the bus is critical to performance. In all cases, when a synonym is detected, it is better to use it as the source for the missing block rather than to send a read block request on the bus.

Because, the physical directory is accessed associatively to check for the presence of a synonym it is very delicate to allow several copies of the same information in the cache at the same time even if they belong to read-only pages. If multiple synonyms are tolerated in a cache, some cases of multi-match could occur if the cache is set-associative or fully-associative. This type of condition is difficult to handle in hardware.

Multi-matches can occur during the bus snooping activity. However, for cache consistency protocols in which the main memory keeps the ownership of a block as long as it is not modified in the cache of a processor, there is no need to search among all the entries satisfying the matching criterion. A missing block is always fetched from the main memory. In this case, provided the caches are purged/flushed whenever a read-only page becomes modifiable multiple synonyms of read-only data can be tolerated in the caches.

4.6 VIRTUAL TO PHYSICAL ADDRESS DEMAPPING / REMAPPING

As in a single processor system, when a virtual-to-physical address translation is invalidated some actions must be taken to maintain the consistency in the virtual-address cache of each processor.

4.6.1 Flushing Avoidance

When a portion of the virtual address space (page, context, segment) is demapped, it is still necessary to purge the caches (i.e. to invalidate but not to discard the entries) and TLBs of the system which were holding blocks and page translations contained in the demapped area. However, it is no more necessary in the case of a write-back cache to recopy the modified blocks to main memory (i.e. to flush) because the consistency protocol ensures that dirty blocks are always part of the *shared-memory image*. The valid bit in the entry of the dual directory can remain set to indicate that the data contained in the block are still valid while the valid bit in the corresponding entry of the cache directory can be cleared to indicate that the address mapping has been modified.

When a translation is displaced in the TLB by the replacement algorithm the cache controller can always find the physical address of the block in the dual directory. In particular main memory updates on cache misses can proceed even if the TLB entry for the displaced dirty block is missing. Hence, it is possible to decouple totally the management of the TLB and the virtual-address cache.

In [Goodman 87] there is a long discussion on the distinction between cache misses where there is a tag match and the valid bit is not set from cache misses where there is no match at all. The first type of miss is called an *invalid miss* and the second one a *no-match miss*.

On an *invalid miss* the data contained in the block may still be valid but the virtual-to-physical mapping is or was temporarily invalid. The handling of an *invalid miss* supposes the dual directory entry is accessed through the pointer held in the entry of the cache directory to check if the virtual-to-physical mapping is still valid. If the valid bit is still set in the entry of the dual directory, a check on the physical page number must be made. If the PPN delivered by the TLB is identical to the one formed by the concatenation of the block physical address and the bits of the cache tag pointer used to index the set inside the *superset*, the data contained in the block are still valid. In this case, the cache controller only has to set the valid bit in the cache directory entry.

It is important to note that actually there is **no reason** to distinguish *invalid misses* from *no-match misses*. When the cache controller handles a miss it must first check if a synonym is already present in the cache. This detection is done by accessing the dual directory with the physical address of the missing block. Therefore, an *invalid miss* corresponds to a positive synonym detection where the synonym indexes in the same entry as the missing block. As explained previously in section 4.5, only the cache tag needs to be updated in this case. This is exactly how *invalid misses* should be handled according to [Goodman 87].

If a page of the virtual space is demapped and then remapped to the same page frame (which contains now different data), it is important to note that the conjunction of the synonym detection mechanism and the cache consistency protocol ensures there is no coherence problem. When the new mapping is built, the page is initialized with information swapped-in from the disks or is explicitly filled with null data by the kernel. In both cases the cache consistency protocol ensures that the copies in the cache(s) are either discarded or updated.

4.6.2 Cache Purge Avoidance

If the TLB is fast enough it is possible to avoid purging the cache when a portion of the virtual space is demapped. Only the translations in the TLB must be invalidated. Besides the virtual address of the block, the cache tags must contain the PPN or even the entire block physical address. On each cache access, the TLB is accessed in parallel with the cache and the tag match is validated by comparing the PPN contained in the cache entry with the one delivered by the TLB. If they also match the access hits. A valid bit must still be present in the cache tag for the cases where blocks are discarded; the valid bits in the cache directory and the dual directory must always be identical.

The matching criterion for a cache hit includes the fact that the PPN contained in the cache entry must be the same as the one obtained from the TLB.

As before, when a virtual page number is remapped to the same page frame which may have been allocated to a different page in the meantime, the consistency protocol guarantees that the cache blocks are either updated or invalidated.

4.6.3 Implementation of Purge/Flush

The problems and solutions for a purge/flush of the cache, dual directory or TLB are similar to those exposed in the section on uniprocessors. However, in a shared-memory multiprocessor more than one cache can hold copies of blocks within the demapped memory area and more than one TLB can hold the

virtual-to-physical translation(s) of any page located in the demapped area. Thus, the purge/flush operation requires the intervention of all processors that were previously accessing data in the demapped area.In systems with physical-address caches, the consistency of TLBs is usually maintained through TLB shootdown [Black 89][Teller 90]. When caches are accessed with virtual addresses, the TLB shootdown protocol must be extended to include cache flushes or purges.

The matching criterion for the purge/flush operations for the cache can be (part of) the physical address or (part of) the virtual address. However, the invalidation of the page address translations in the TLBs must still be made through the virtual addresses unless a dedicated comparison logic for the physical page numbers is added. In general, it is preferable to use the virtual address as matching criteria for purge/flush operations but there is an implication on the PIDs. To be able to perform the cache and/or TLB purge/flush on virtual addresses, the PIDs must be system wide identifiers. When a process migrates it must keep the same PID. This has also the advantage of simplifying the implementation of the process dispatching algorithm whose main goal is to balance the workload across the multiple processors.

One can find an advantage of doing the purge/flush on the physical address with a private virtual space model if the cause is a reallocation of the page frame. A purge/flush "command" must be sent on the bus for each synonym (if there are any) of the page being demapped if the matching criterion is based on the virtual address. If the purge/flush operations are based on the physical address, it is not necessary for the kernel to keep a special data structure linking all the virtual page numbers that are synonym.

The binding between the cache and the dual directory may limit the efficiency of the purge/flush operation. Even if the dual directory is fully associative and can be purged in a single operation, the organization of the cache and/or the recopying of the modified blocks in the case of a write-back cache limit the efficiency of purge/flush operations.

The decision to demap a portion of the virtual space is taken by the operating system kernel. Hence, flush or purge operations are under the control of software but some hardware support is required. To inform the other processors, the processor executing the kernel must be able to send/receive interrupt signals to/from other processors. The interrupt is physically issued by the bus interface when it receives a specific command sent out by its attached CPU. The hardware can provide the ability to send out an interrupt to a particular processor, a group of processors and/or to broadcast an interrupt to all the processors connected to the bus. When the kernel maintains data structures to record the PIDs of all the processes which were accessing the range of addresses being demapped as well as the processors on which these processes were scheduled it is more efficient to interrupt these processors only. If this information is not available, an interrupt must be sent one at a time or be broadcast simultaneously to all the processors. There is no need to be able to exchange complex messages directly between processors on the bus, because they can exchange information easily in the shared-memory image. Thus, when receiving an interrupt, the attached handler can retrieve the data specifying the range of addresses to demap at a conventional memory location and the CPU can issue the purge/flush commands to the cache, dual directory and TLB.

In the multiprocessor architectures considered in this paper, the activity of each processor is asynchronous (independent) from the activity of the others. This type of architecture is commonly designated as MIMD (Multiple-Instruction stream, Multiple-Data stream) [Flynn 66]. Therefore, there must be an explicit **synchronization** so that the kernel knows when all the purge/flush operations are completed. This synchronization can be done in software with a regular synchronization primitive. For example, a common counter could be decremented by each processor when it has completed all the purge/flush operations in its cache, dual directory and TLB. The kernel knows it can safely do the remapping when the counter has reached the minimal value.

The scenario described above for the demapping and remapping of a portion of a virtual space in a multiprocessor assumed the presence of a very minimal hardware support; namely the ability for a processor to interrupt another one plus the ability to send out purge/flush commands to its own cache, dual

directory and TLB. However, the purge/flush operations in the caches and TLBs of the system can be made more or less transparent to the software with more sophisticated hardware.

For example the cache, dual directory and TLB controllers may have the capability to purge/flush autonomously all the entries in the cache, dual directory and TLB respectively, which satisfy a certain matching criterion (on the virtual or physical address); in this case only one command must be issued by the attached CPU. Moreover, if these controllers can also receive flush commands through the bus interface (which must always monitor and somewhat decode all the bus transactions for maintaining data consistency) there is no need to use the interrupt mechanism. To demap a part of the virtual space, the kernel must just issue a specific request on the bus with the help of a specific instruction or of a memory-mapped command. The execution of this instruction or command provokes the purge/flush of the cache, dual directory and TLB attached to the CPU and the emission of a "purge/flush request" transaction on the system bus.

There is no need for an explicit synchronization if there is an upper bound for the flushing/purging time in the whole system; after issuing the flush request on the bus the kernel simply has to wait "for a while" before remapping. Another solution is to have an hardware synchronization which enforces the atomicity of a demap in a transparent manner. When the processors are heavily pipelined it can be difficult to enforce the atomicity of a demap in hardware. A demap is atomic if the following three conditions are verified when this one completes:

- no processor can access any information located in the demapped area,
- all memory references to the demapped area have been performed,
- all references to the PTEs associated to the demapped area are completed.

We have described the two extreme hardware supports which can be envisioned for maintaining the consistency of the shared-memory image when a part of the virtual space is demapped. Of course, many variations are possible and the level of sophistication of the hardware support must be driven exclusively by the frequency of demapping and remapping operations in the system. This frequency depends on the organization of the kernel and the virtual addressing scheme. It is likely to be more frequent with a private virtual space model. More performance studies are needed to clarify these design tradeoffs.

4.7 SUPPORT FOR MEMORY MANAGEMENT

In this section we only examine the problems specific to multiprocessor architectures. Possible configurations for virtual-address caches and their associated TLB have been exposed in Section 3.3.

In the single virtual space model, all the processes share the entries of the *inverted page table*. Hence, the information contained in an entry of this table can be cached in more than one TLB and maybe also in the cache tags (with the access rights). Therefore, multiple copies of the *reference* and *modify* bits can be present in different TLBs of the system.

Although each process has its own page tables for virtual-to-physical address translation in the private virtual space model, there are many cases where a page table entry (noted PTE) can be shared by different processors. For example, in UNIX System V, when a parent process *"forks"* a child process, the resulting processes share access to the page table for the shared text region [Bach 86]. If the child process is scheduled on a separate processor, multiple copies of the information contained in a PTE are cached in distinct TLBs and possibly cache tags.

When the kernel supports the notion of threads or lightweight processes, distinct processors can also dynamically share page table entries. Because, the threads or lightweight processes share the same virtual space and can be scheduled on separate processors multiple copies of the information held in PTEs can be present in the system.

These multiple copies of the same information lead to a classical coherence problem when one of them is modified. For example, when the page stealer clears the *reference* bit of a page table entry in the

shared memory image, all copies in the TLBs of the system should also be cleared. In a uniprocessor, the kernel runs on the same processor as user processes, thus all *reference* bits in the TLB are accessible and can be cleared easily. In a symmetric multiprocessor system, the kernel can run on any processor. Without any special hardware support the page stealer cannot clear the other copies of the *reference* bit held in the TLBs of other processors. We have already examined a very similar problem in Section 4.6.3 for the implementation of the purge/flush operations.

In fact, it is possible not to support a *reference* bit in the TLB entries and only rely on an invalidation mechanism. As explained in Section 2.3, the TLB miss handler checks the value of the *reference* bit in the PTE and sets it if necessary. This is made possible because in a multiprocessor system, whatever the speed of the TLB, the management of the TLB and the virtual-address cache can be decoupled (see Section 4.6).

Hardware support for clearing *reference* bits can lie between the use of an interrupt mechanism and a dedicated bus transaction which clears the *reference* bit in TLB entries where a given virtual-to-physical translation is contained. This bus transaction could be interpreted and executed by the bus interface of each processor transparently to the processor (i.e. to the software). A good tradeoff is to use a TLB purge transaction because these invalidations are infrequent. In this case, only the TLB is invalidated but not the caches.

Although distinct processes can share a page table in the private virtual space model, they can use distinct entries inside the same TLB to store the virtual-to-physical translation. This is due to the fact that a different PID is allocated to each process in order to be able to share the TLB. In this case, virtual addresses are "synonyms" although they come from the same page table entry. If the organization of the TLBs is direct-map or set-associative, an access to clear the *reference* bit must be done for the PID values of all processes sharing the page table entry. With a fully-associative organization one access could be enough if the comparison can be done only on the virtual page number excluding the PID bits.

In a multiprocessor system the *reference* bit stored in the PTE should be updated in a write-through manner either by software with a trap handler or by the hardware. The updating of this bit stored in the shared memory image cannot be done with a write-back approach as for a uniprocessor because the page stealer cannot examine the contents of all TLBs and PTEs in a *critical* section.

Hence, if multiple processes running on distinct processors are sharing a page table, some redundant updates of *reference* bits can occur. However, the traffic on the bus due to these extra updates (if the PTEs are not cached) or to the consistency protocol (if the PTEs are cached) is expected to be very small.

In the private virtual space model, processes can share information with synonyms coming from distinct page table entries. Thus, the kernel must keep track of the number of processes which reference a page. For example, in UNIX System V, this is done with a reference counter associated with each physical page frame in the *page frame data table* (noted *pfdata*) [Bach 86]. However, the page stealer takes into account only the value of the *reference* bit in the PTEs to determine if a page is eligible to be swapped out. Thus, a page shared and modified by multiple processes is recopied onto the swap device each time it is victimized by the page stealer. The physical page frame is never reallocated until the reference counter is null. Thus, the page remains in the shared memory image as long as the page stealer has not victimized all the synonyms.

With this implementation of the page replacement, there is no need to maintain the consistency of the *reference* bits at the physical page frame level in the private virtual space model.

As for the *reference* bit the updating of the *modify* bit in the page table entries must be done in a write-through fashion because the page stealer cannot access the TLBs of all processors in a critical section. Hence, there will be some redundant attempts to set the copy of the *modify* bit in the PTEs corresponding to pages shared and modified by processes running on distinct processors.

Because the copy of the *modify* bit in each PTE is updated only on the first modification done by each process referencing the page, the induced traffic on the bus is very small and does not affect the overall performance.

The management of the *reference* and *modify* bits and the invalidation of page translations (see Section 4.6) are part of the TLB coherence problem. This problem has rarely been addressed in its entirety but an original solution has been proposed in [Wood 86] with an *In-Cache-TLB*. In this design, there is no TLB per se, but the page tables are mapped at conventional locations of the single virtual space in order to be able to store the page table entries in the virtual-address caches. Then, all the TLB coherence problems are handled by the cache consistency protocol. This solution is very elegant, but this particular organization for the memory management hardware support has not yet been proven. There is no real data on the performance of this particular caching technique of virtual-to-physical translations. When the Berkeley SPUR prototype [Hill 86] will be assembled some data will become available.

5. CONCLUSIONS

In this paper, we have shown that problems related to virtual-address caches could be solved at acceptable hardware cost and/or with acceptable restrictions on the software. Software transparency is highly desirable for complex programs. However, hardware cost and overall performance are the basic factors affecting the cost effectiveness of a design.

To maintain the coherence within and among virtual-address caches in both uniprocessor and multiprocessor systems the hardware is much simplified and the machine is more efficient when synonyms are restricted to map into the same cache entry (case of modulo synonyms). If synonyms are not restricted, then the only solutions are to search through the *superset* on each miss (uniprocessor) and to bind the entries in the cache and the dual directories (multiprocessor). This later solution either under-utilizes the cache or requires a very high degree of associativity.

When a virtual to physical mapping is changed, this change must be reflected in the cache and even in the cache of other processors in a multiprocessor. A mechanism to flush and/or to purge the cache(s) must be included in the design of virtual address caches. With some hardware support, the flush and the purge can, in some cases, be avoided in multiprocessors; these solutions have been explained in the paper.

The consistency of the *reference* bit and the *modify* bit for each page table entry must also be maintained. The handling of these bits can be greatly simplified with some cooperation from the software.

6. APPENDIX

Derivation of the Cache Occupancy

Let assume first that both directories are direct-map, and let N be the superset size. When a miss occurs, the new block occupies a given block frame in the superset with the probability $\frac{1}{N}$. There is no correlation between the positions of the bounded entries in the virtual and physical directories. If we assume that k entries are valid for the superset in both directories ($1 \leq k \leq N$), four different cases can occur:

1. The new block displaces a valid block in the virtual directory (probability $\frac{k}{N}$), and maps to the same entry as the displaced block in the physical directory or to an invalid entry (probability $\frac{(N-k+1)}{N}$). The overall probability of this case is $\frac{k.(N-k+1)}{N^2}$. The number of valid blocks k is unchanged after the miss.

2. The new block displaces a valid block in the virtual directory (probability $\frac{k}{N}$), and maps to a valid entry in the physical directory which is not the entry occupied by the displaced block (probability $\frac{k-1}{N}$). One valid block must be invalidated and therefore the number of valid blocks in the cache after the miss is $(k-1)$. The overall probability of this case is $\frac{k.(k-1)}{N^2}$.

3. The new block maps to an invalid entry in the virtual directory (probability $\frac{(N-k)}{N}$), and displaces a valid entry in the physical directory (probability $\frac{k}{N}$). The overall probability of this case is $\frac{k.(N-k)}{N^2}$. The number of valid blocks k is unchanged after the miss has been handled.

4. The new block maps to an invalid entry in both directories. The overall probability of this case is $\frac{(N-k)^2}{N^2}$. The number of valid blocks in the cache after the miss is $(k + 1)$.

All these cases are also applicable to a set-associative organization, provided the replacement in both directories is done randomly. Therefore the result derived in this appendix is also valid for set-associative directories with random replacement. This result is not applicable if the replacement policy selects invalid entries first (See section 4.4.5).

The state of a superset, and therefore the state of the whole cache, defined by the number of valid blocks is a discrete Markov process. The state transition occur at miss times. The Markov chain is depicted on Figure 13. The transition probabilities are derived directly from the four cases described above.

From the Markov diagram, we can write down the balance equation for each state as follow (P_k is the stationary probability of state k):

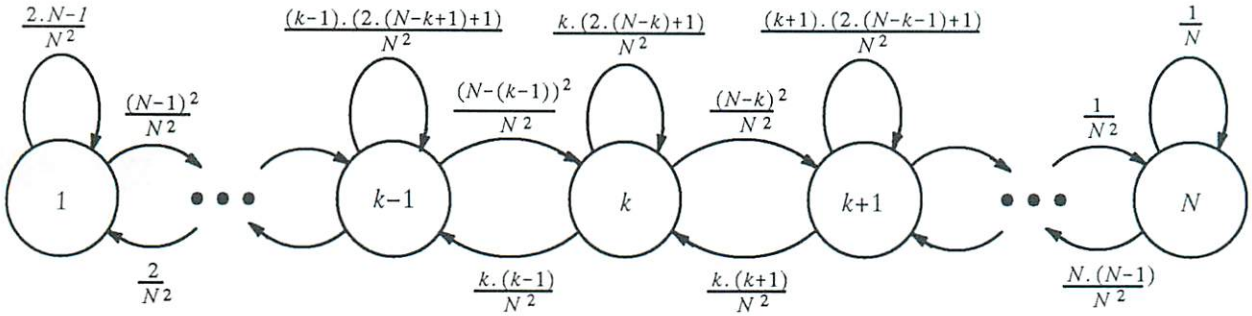


Figure 13: Markov Chain for the superset state

$$P_2 = \frac{(N-1)^2}{2 \cdot 1} \cdot P_1$$

$$P_3 = \frac{(N-2)^2}{3 \cdot 2} \cdot P_2$$

$$\vdots$$

$$P_k = \frac{(N-(k-1))^2}{k \cdot (k-1)} \cdot P_{k-1}$$

$$\vdots$$

$$P_N = \frac{1^2}{N \cdot (N-1)} \cdot P_{N-1}$$

Consequently:

$$P_k = \frac{(N-(k-1))^2 \cdot (N-(k-2))^2 \ldots (N-1)^2}{k \cdot (k-1) \cdot (k-2) \cdot (k-2) \ldots 3 \cdot 2 \cdot 2 \cdot 1} \cdot P_1$$

$$P_k = \frac{\left(\frac{(N-1)!}{(N-k)!}\right)^2}{k! \cdot (k-1)!} \cdot P_1$$

$$P_k = \frac{(N-1)!}{k! \cdot (N-k)!} \cdot \frac{(N-1)!}{(k-1)! \cdot (N-k)!} \cdot P_1$$

$$P_k = \frac{1}{k} \cdot \frac{(N-1)!}{(k-1)! \cdot (N-k)!} \cdot \frac{(N-1)!}{(k-1)! \cdot (N-k)!} \cdot P_1$$

$$P_k = \frac{1}{k} \cdot \binom{N-1}{k-1}^2 \cdot P_1 \qquad (1)$$

In particular, we have:

$$P_N = \frac{1}{N} \cdot P_1 \qquad (2)$$

Equation (1) divided by equation (2) yields:

$$\frac{P_k}{P_N} = \frac{N}{k} \cdot \binom{N-1}{k-1} \cdot \binom{N-1}{k-1}$$

$$\frac{P_k}{P_N} = \binom{N}{k} \cdot \binom{N-1}{k-1}$$

Hence:

$$P_k = \binom{N}{k} \cdot \binom{N-1}{k-1} \cdot P_N$$

Then summing over all probabilities:

$$\sum_{k=1}^{N} P_k = 1 = \sum_{k=1}^{N} \binom{N}{k} \cdot \binom{N-1}{k-1} \cdot P_N$$

Solving the previous equation for P_N:

$$P_N = \left\{ \sum_{k=1}^{N} \binom{N}{k} \cdot \binom{N-1}{k-1} \right\}^{-1}$$

$$P_N = \left\{ \sum_{k=1}^{N} \left[\binom{N-1}{k} + \binom{N-1}{k-1} \right] \cdot \binom{N-1}{k-1} \right\}^{-1}$$

$$P_N = \left\{ \sum_{k=1}^{N} \binom{N-1}{k} \cdot \binom{N-1}{k-1} + \sum_{k=1}^{N} \binom{N-1}{k-1} \cdot \binom{N-1}{k-1} \right\}^{-1}$$

$$P_N = \left\{ \sum_{k=1}^{N} \binom{N-1}{k} \cdot \binom{N-1}{N-k} + \sum_{k=1}^{N} \binom{N-1}{k-1} \cdot \binom{N-1}{N-k} \right\}^{-1}$$

$$P_N = \left\{ \binom{2N-2}{N} \cdot \binom{2N-2}{N-1} \right\}^{-1}$$

$$P_N = \left\{ \binom{2N-1}{N} \right\}^{-1}$$

$$P_N = \frac{N! \,.\, (N-1)!}{(2N-1)!} \qquad (3)$$

From equations (2) and (3), we have:

$$P_1 = N \cdot P_N = \frac{(N!)^2}{(2N-1)!} \tag{4}$$

From equations (1) and (4), we have:

$$P_k = \frac{1}{k} \cdot \binom{N-1}{k-1}^2 \cdot P_1$$

$$k \cdot P_k = \binom{N-1}{k-1}^2 \cdot P_1$$

By summing for all k, we have:

$$\sum_{k=1}^{N} k \cdot P_k = \sum_{k=1}^{N} \binom{N-1}{k-1}^2 \cdot P_1$$

$$\sum_{k=1}^{N} k \cdot P_k = \sum_{k=1}^{N} \binom{N-1}{k-1} \cdot \binom{N-1}{k-1} \cdot P_1$$

$$\sum_{k=1}^{N} k \cdot P_k = \sum_{k=1}^{N} \binom{N-1}{k-1} \cdot \binom{N-1}{N-k} \cdot P_1$$

$$\sum_{k=1}^{N} k \cdot P_k = \binom{2N-2}{N-1} \cdot P_1 \tag{5}$$

Form equations (4) and (5):

$$\sum_{k=1}^{N} k \cdot P_k = \frac{(2N-2)!}{(N-1)! \cdot (N-1)!} \cdot \frac{(N!)^2}{(2N-1)!}$$

$$\sum_{k=1}^{N} k \cdot P_k = \frac{N^2}{2N-1}$$

Finally the occupancy ratio is given by:

$$\frac{\sum_{k=1}^{N} k \cdot P_k}{N} = \frac{N}{2N-1}$$

7. REFERENCES

[Archibald 85] James Archibald, and Jean Loup Baer, "An Economical Solution to the Cache Coherency Problem", Proceedings of the 11th Annual Symposium on Computer Architecture, pp. 355–362, June 1984.

[Babaoglu 81] O. Babaoglu, and W. Joy, "Converting a Swap–Based System to do Paging in an Architecture Lacking Page–Referenced Bits," ACM Operating System Review, **15**, 5, pp. 78–86, December 1981.

[Bach 86] Maurice J. Bach, "The Design of the Unix Operating System," Prentice–Hall, 1986.

[Bensoussan 72] A. Bensoussan, C. I. Clingen, and R.C. Daley, "The Multics virtual memory: concepts and design, " Communications of the ACM, **15**, 5, May 1972.

[Beyers 85] Joseph W. Beyers, Eugene R. Zeller, and Dana Seccombe, "VLSI Technology Packs 32–bit Computer System into a Small Package," Hewlett–Packard Journal, pp. 3–6, August 1983.

[Black 89] D. Black, R. Rashid, D. Golub, C. Hill, and R. Baron, "Translation Lookaside Buffer Consistency: A Software Approach," Proceedings of the Third International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 113–122, April 1989 .

[Chang 88] Albert Chang, and Mark F. Mergen, "801 Storage: Architecture and Programming, " ACM Transactions on Computer Systems, **6**, 1, pp. 28–50, February 1988.

[Chang 87] J.H Chang, H. Chao, and K. So, "Cache Design of a Sub–Micron CMOS System 370," Proceedings of the 14th Annual Symposium on Computer Architecture, pp. 208–213, June 1987.

[Cheng 86] Ray Cheng, "Virtual Address Cache in Unix," Proceedings 1987 Summer USENIX Conference, pp. 217–224, 1987.

[Cheriton 86] David R. Cheriton, Gert A. Slavenburg, and Patrick D. Boyle, "Software–Controlled Caches in the VMP Multiprocessor," Proceedings of the 13th Annual Symposium on Computer Architecture, pp. 366–374, June 1986.

[Cheriton 88] David R. Cheriton, Anoop Gupta, Patrick D. Boyle, and Hendrik A. Goosen, "The VMP Multiprocessor: Initial Experience, Refinements and Performance Evaluation," Proceedings of the 15th Annual Symposium on Computer Architecture, pp. 410–421, June 1988.

[Clark 85] Douglas W. Clark, and Joel Emer, "Performance of the VAX 11/780 Translation Buffer: Simulation and Measurement," ACM Transactions on Computer Systems, **3**, 1, pp. 31–62, February 1985.

[Cypress 88] "CY7C600 Risc Family Users Guide," Cypress Semiconductor, June 1988.

[DeMoney 86] M. DeMoney, J. Moore, and J. Mashey, "Operating System Support on a RISC," Proceedings 1986 COMPCON, IEEE, March 1986.

[Dubois 82] Michel Dubois, and Faye' A. Briggs, "Effects of Cache Coherency in Multiprocessors," IEEE Transactions on Computers, **31**, 11, November 1982.

[Fabry 73] R. S. Fabry, "The Case for Capability–Based Computers," Proceeding 4th Symposium on Operating Systems Principles, October 1973.

[Flynn 66] M. J. Flynn, "Very High–Speed Computers," Proceedings of the IEEE, **54**, pp. 1901–1909, December 1966.

[Frank 84] Steven J. Frank, "Tightly Coupled Multiprocessor System Speeds Memory Access Time," Electronics, pp. 164–169, January 12, 1984.

[Frink 88] Craig R. Frink, and Paul J. Roy, "The Cache Architecture of the Apollo DN4000," Proceedings 1988 COMPCON, IEEE, pp. 300–302, 1988.

[Furht 87] Borivoje Furht, and Veljko Milutinovic, "A Survey of Microprocessor Architectures for Memory Management," Computer, pp. 48–67, March 1987.

[Goodman 87] James R. Goodman, "Coherency for Multiprocessor Virtual Address Caches," Proceedings 2nd International Conference on Architecture Support For Programming Languages and Operating Systems, ACM, 1987.

[Hill 86] M. Hill, S. Eggers, J. Larus, G. Taylor, G. Adams, B. K. Bose, G. Gibson, P. Hansen, J. Keller, S. Kong, C. Lee, D. Lee, J. Pendleton, S. Ritchie, D. Wood, B. Zorn, P. Hilfinger, D. Hodges, R. Katz, J. Ousterhout, and Dave Patterson, "Design Decision in SPUR," Computer, pp. 8–22, November 1986.

[Katz 85] R. H. Katz, S. J. Eggers, D. A. Wood, C. L. Perkins, and R. G. Sheldon, "Implementing a Cache Consistency Protocol," Proceedings 12th Annual International Symposium on Computer Architecture, pp. 158–166, June 1985.

[Kelly 85] Ed Kelly, "The SUN 4 Architecture," Sun Microsystems Internal Report, October 1985.

[Kepecs 85] Jonathan Kepecs, "Lightweight Processes for UNIX Implementation and Applications," Proceedings 1985 Summer USENIX Conference, pp. 299–308, 1985.

[Lee 87] Roland Lee, Pen-Chung Yew , and Duncan Lawrie, "Multiprocessor Cache Design Considerations," Proceedings 14th International Symposium on Computer Architecture, pp. 253–262, June 1987.

[Levy 82] Henry M. Levy, and Peter H. Lipman, "Virtual Memory Management in the VAX/VMS Operating System," Computer, pp. 35–41, March 1982.

[Lob 83] Clifford G. Lob, Mark J. Reed, Joseph P. Fucetola, and Mark A. Ludwig, "High-Performance VLSI Memory System," Hewlett-Packard Journal, pp. 14–20, August 1983.

[Lovett 88] Tom Lovett and, Shreekant Thakkar, "The Symmetry Multiprocessor System," Proceedings of the 1988 International Conference on Parallel Processing, August 1988.

[McCreight 84] E. M. McCreight, "The Dragon Computer System: An Early Overview," NATO Adavanced Study Institute on Microarchitecture of VLSI Computers, Urbino, Italy, July 1984.

[Mac Gregor 83] Douglas Mac Gregor, and David S. Mothersole, "Virtual Memory and the MC 68010," IEEE Micro, pp. 24–38, June 1983.

[Mahon 86] Michael J. Mahon, Ruby Bei-Loh Lee, Terrence C. Miller, Jerome C. Huck, and William R. Bryg, "Hewlett-Packard Precision Architecture: The Processor," Hewlett-Packard Journal, pp. 4–21, August 1986.

[Mayberry 84] Walter Mayberry and, Gregory Efland, "Cache Boosts Multiprocessor Performance," Computer Design, pp. 133–138, November 1984.

[Moussouris 86] John Moussouris, Les Crudele, Dan Freitas, Craig Hansen, Ed Hudson, Steve Przybylski, Tom Riordan, and Chris Rowen, "A CMOS RISC Processor with Integrated System Functions," Proceedings 1986 COMPCON, IEEE, March 1986.

[Namjoo 88] M. Namjoo, A. Agrawal, D. C. Jackson and L. Quach, "CMOS Gate Array Implementation of the SPARC Architecture," Proceedings of the 1988 COMPCON, IEEE, March 1988.

[Patil 87] Indira Patil, "Evaluation of SPARC Multiprocessor Architectures," Sun Microsystems Internal Report, September 1987.

[Papamarcos 84] Mark Papamarcos, and Janak Patel, "A Low-Overhead Coherence Solution for Multiprocessors with Private Cache Memories," Proceedings 11th International Symposium on Computer Architecture, pp. 348-354, June 1984.

[Patterson 85] David A. Patterson, "Reduced Instruction Set Computer," Communications of the ACM, 28, 1, pp. 8-21, January 1985.

[Rashid 87] Richard Rashid, Avadis Tevanian, Michael Young, David Golub, Robert Baron, David Black, William Bolosky, and Jonathan Chew, "Machine-Independent Virtual Memory Management for Paged Uniprocessor and Multiprocessor Architectures," Proceedings 2nd International Conference on Architecture Support For Programming Languages and Operating Systems, ACM, 1987.

[Simpson 86] R. O. Simpson, "The IBM RT Personal Computer," Byte 11, 11, pp. 43-78, 1986.

[Smith 82] Alan J. Smith, "Cache Memories," ACM Computing Surveys, 14, 3, pp. 473-530, September 1982.

[Smith 85] Alan J. Smith, "Cache Evaluation and the Impact of Workload Choice," Proceedings of the 12th Annual Symposium on Computer Architecture, pp. 64-73, June 1985.

[Soltis 81] F. G. Soltis, "Design of a Small Business Data Processing System," Computer, pp. 77-81, September 1981.

[Stone 87] H. S. Stone, "High-Performance Computer Architecture", Addison-Wesley Publishing Company, 1987.

[Sun 85] "The Unix System: A Sun Technical Report," Sun Microsystems, 1985.

[Sun 86] "Sun-3 Architecture: A Sun Technical Report," Sun Microsystems, 1986.

[Sweazey 86] Paul Sweazey, and Alan J. Smith, "A Class of Compatible Cache Consistency Protocols and their Support by the IEEE Futurebus," Proceedings of the 13th Annual International Symposium on Computer Architecture, pp. 414-423, June 1986.

[Teller 90] P. J. Teller, "Translation-Lookaside Buffer Consistency," IEEE Computer, June 1990.

[Tucker 86] Stuart G. Tucker, "The IBM 3090 System: An Overview," IBM Systems Journal, 25, 1, pp. 4-19, 1986.

[Van Loo 87] William Van Loo, "Maximize Performance by Choosing Best Memory," Computer Design, pp. 89-94, August 1987.

[Wood 86] David A. Wood, Susan J. Eggers, Garth Gibson, Mark D. Hill, Joan Pendleton, Scott A. Ritchie, George S. Taylor, Randy H. Katz, and David A. Patterson, "An In-Cache Address Translation Mechanism," Proceedings of the 13th Annual International Symposium on Computer Architecture, pp. 358-365, June 1986.