# Options for Dynamic Address Translation in COMAs

Xiaogang Qiu and Michel Dubois

CENG 98-08

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California 90089-2562
(213) 740-4475
March 1998

# Options for Dynamic Address Translation in COMAs

Xiaogang Qiu and Michel Dubois

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, CA 90089-2562
(213)740-4475
Fax: (213) 740-7290
{xiaogang,dubois}@paris.usc.edu

## Abstract

In modern processors, the dynamic translation of virtual addresses to support virtual memory is done before or in parallel with the first-level cache access. As processor technology improves at a rapid pace and the working sets of new applications grow insatiably the latency and bandwidth demands on the TLB (Translation Lookaside Buffer) are getting more and more difficult to meet. The situation is worse in multiprocessor systems, which run larger applications and are plagued by the TLB consistency problem.

We evaluate and compare five options for virtual address translation in the context of COMAs (Cache Only Memory Architectures). The dynamic address translation mechanism can be located after the cache access provided the cache is virtual. In a particular design, which we call V-COMA for Virtual COMA, the physical address concept and the traditional TLB are eliminated. While still supporting virtual memory, V-COMA reduces the address translation overhead to a minimum.

V-COMA scales well and works better in systems with large number of processors. As a machine running on virtual addresses, V-COMA provides a simple and consistent hardware model to the operating system and the compiler, in which further optimization opportunities are possible.

**Keywords**: dynamic address translation, TLB, virtual memory, COMA, virtual address cache.

## 1 Introduction

In modern processors, dynamic translation from virtual to physical addresses is supported by a TLB (Translation Lookaside Buffer). Its overhead depends on the TLB miss ratio and on the service time for TLB misses. The typical address translation overhead is about 5% - 10% of the execution time, and has changed little in the past ten years[9][21]. However, current technology trends and the growing working set demands of applications are putting more and more pressure on TLBs. Recent studies have shown that the TLB service time can consume up to 50% of the user execution time in some workloads[23][24][25].

The increasing pressure on the TLB comes from speed requirement and lack of scalability. Being in the critical path of every instruction and data access, the TLB latency and bandwidth requirements increase with the clock rate and instruction level parallelism[2]. Moreover, the memory system of modern processors such as dynamically scheduled[32] or VLIW processors[12] need to satisfy multiple memory and TLB accesses in each cycle. It is more and more difficult and costly to implement a large TLB meeting these requirements. At the same time, the working sets of applications keep growing and changing. The issue of scalability comes from the fact that the TLB is on chip and its size is fixed. Poor TLB scalability affects the efficiency of address translation when the processor is integrated in a system, where the TLB can not be scaled with the sizes of other components, especially main memory. In a multiprocessor, the *effective* amount of TLB does not increase as fast as the number of processors because entries are replicated; moreover, TLB consistency must be maintained[26].

Virtual address caches have been proposed to relieve the latency and bandwidth requirements of TLBs[5][6][11][15][31]. When the cache is virtually indexed and tagged, most memory accesses are completed without TLB involvement; in fact, address translation can be done in different locations in the memory hierarchy[28], and can be implemented in various ways, such as in-cache translation[31] or even by software[15].

In this paper we explore design options to reduce address translation overhead and make it more scalable, especially in multiprocessor systems. The basic idea is to move the address translation closer to memory, where the TLBs are shared, do not have coherence problems, and scale well with both the memory size and the number of processors. In CC-NUMAs (Cache-Coherent Non-Uniform Memory access Architectures), the sharing of TLBs is not efficient because of the lack of data migration and replication. However, because the main memory (or attraction memory) of COMAs (Cache Only Memory Architectures) acts as a cache, new and efficient virtual memory organizations are possible. In particular, we propose an architecture called V-COMA to eliminate the traditional TLB and even the concept of physical address while still supporting virtual memory. V-COMA is an extension of the virtual-address cache concept to the attraction memory. It indexes and tags both cache and attraction memory with virtual address. Instead of being in the critical path of every memory access within the processor, address translation is part of the cache coherence protocol.

Our simulations show that the address translation overhead in V-COMA is dramatically reduced and scales well. Besides the immediate performance benefits, we believe V-COMA has advantages for both the operating system and the programmer. The linear physical address space and its management are no longer needed. The whole memory hierarchy is indexed by virtual addresses, which gives a simple and consistent view to the compiler and programmer and offers new opportunities for virtual address layout optimizations.

The remainder of the paper is structured as follows. After some technical background in Section 2, the five design options for dynamic address translation are introduced in Section 3. The V-COMA architecture is described in Section 4. In Section 5, we show the results of our evaluations. We provide further discussions on the V-COMA architecture in Section 6, and an overview of related research in Section 7. Finally, section 8 presents our conclusions.

## 2 Background

### 2.1 Dynamic Address Translation

Address translation can be done at different levels of the memory hierarchy in a traditional CC-NUMA architecture, as shown in Figure 1. Most processors translate virtual addresses in a TLB before or in parallel with the first-level cache (L0-TLB). However, provided caches are virtually indexed and tagged, the TLB could be placed between the first- and second-level caches (L1-TLB) or after the second-level cache (L2-TLB). In these cases, the TLBs are private and their consistency must be maintained. Alternatively, the TLB could be associated with the home node (SHARED-TLB), an approach similar to the "In-memory TLB" proposed in [26]. In this latter case, the TLBs are shared, map the local memory only, and do not cause coherence problems. However, because the home node is selected with the virtual address the programmer has no control over page location and page migration is impossible. Because page placement cannot be optimized for locality, capacity misses are remote most of the time resulting in poor performance for applications whose significant working set does not fit in the second-level cache.

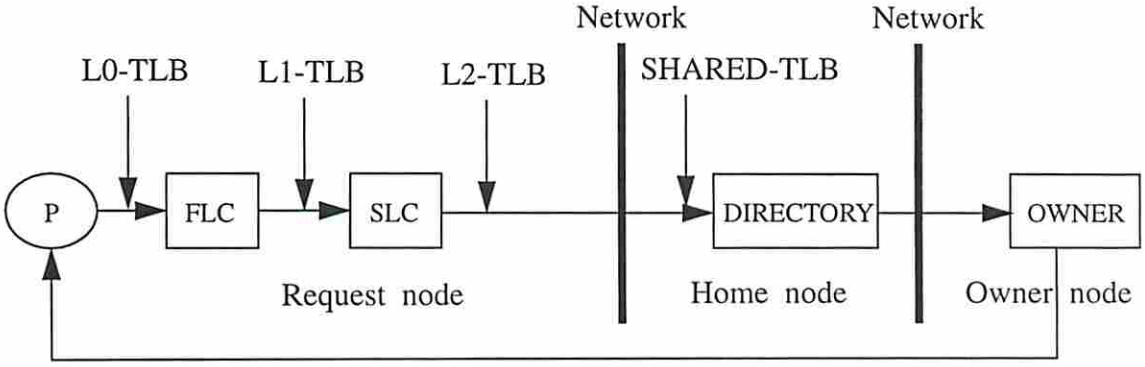


**Figure 1 Possible Locations for the TLB in CC-NUMA**

### 2.2 Issues Related to Virtual Address Cache

There are a series of issues related to any system with virtually indexed and virtually tagged caches, such as in Figure 1. We briefly review these issues here.

#### 2.2.1 Synonyms and Address Mapping Changes

Virtual-address caches suffer from synonym and address mapping change problems[5][6]. Synonyms happen when multiple virtual addresses map to the same physical address and may cause inconsistency in a virtual cache. The virtual-physical address mapping is subject to change due to deallocation and reallocation of pageframes. In general, mappings still remain in the virtual cache after a page has been demapped and must be flushed to avoid inconsistencies. These problems have been studied extensively in the literature and a number of solutions have been proposed.

A strategy to prevent synonyms is to have a global virtual address space that does not allow synonyms at all. Typically there are two possible approaches. In Single Address Space Operating Systems (SASOS)[7], all processes share one single address space. Synonyms are resolved by using the same virtual address. The second approach is segmentation[15][19][31]. Segment registers accessed before the TLB with some bits of the effective address contain base addresses, which are combined with the remain-

ing effective address bits to form a global virtual address. Sharing is implemented at the segment level. An example of a segmented architecture is the 32-bit PowerPC[19].

In the remainder of this paper, we assume a PowerPC-like segmented memory system, where synonyms are not needed nor allowed in the virtual address space.

### 2.2.2 Write-backs and Inclusion

Besides misses, stores must also propagate up the hierarchy in the form of write-through or write-back accesses. Because write-backs may not be directly part of the current working set they have a higher probability of missing in the TLB.

Usually, inclusion is maintained between the caches. Thus, when a block is removed from an upper level cache, the caches down the hierarchy must be invalidated. A reverse translation mechanism, usually in the form of backpointers, is needed between a physical cache and the virtual cache under it[28].

Maintaining inclusion between the TLB and the virtual cache memory below it avoids TLB misses on write-backs from the cache. Although the TLB placed after a virtual cache can be very large and slow in uniprocessors, inclusion is expensive in multiprocessors. First, whereas a large cache cuts the number of capacity misses dramatically, coherence misses can not be filtered out and the longer address translation latency impacts coherence operations. Second, with large cache sizes, the size of the TLB to maintain inclusion also grows, leading to higher cost and longer latency. Third, it is not mandatory to maintain inclusion. Physical pointers stored in the virtual cache can avoid accesses to the TLB on a write back, just as pointers are used in the physical cache to access the virtual cache below it[28].

Because of the effect on coherence misses and the sheer cost of fast, large TLBs, we do not enforce inclusion between a TLB and the virtual caches below it.

### 2.2.3 Late Detection of Memory Faults

After a virtual address is successfully translated into a physical address, the processor assumes the memory access will succeed without fault because the TLB only contains mapping for valid pages in memory. The access will not interrupt the processor pipeline except for fatal errors. The traditional TLB coverage thus provides a "safe-access subset" such that any cache access that passes the TLB does not generate any exception in normal execution. However, this is a conservative strategy, in which every memory access outside the subset stalls the processor even though it may not lead to a memory fault.

Putting the address translation mechanism after the virtual cache postpones the decision time of the memory fault. The virtual cache and the following TLB expand the "safe-access subset", which means that a page fault as well as a miss of the TLB block the processor after a longer detection time as compared to a system with physical caches.

### 2.2.4 Access Rights

Each page table entry contains protection bits such as read, write, and execute for the page. These attributes migrate to the TLB on a TLB miss. Thus, in a system with virtual caches access right bits must be copied at least in the first level cache. This requires cache flushing when access rights are changed. Of course in a segmented system as the one we consider access rights can be easily checked at the segment granularity[15][19] by storing access right bits in the segment register.

## 3 Dynamic Address Translation in COMA

The design options for dynamic address translation and their trade-offs are different in COMA[16]. They are illustrated in Figure 2. Contrary to CC-NUMA, locating the TLB at the memory does not affect the latency of capacity misses much because of the automatic migration and replication of memory lines in a COMA. In this paper we will compare five schemes called L0-TLB, L1-TLB, L2-TLB, L3-TLB and V-COMA.

### 3.1 L0-TLB

**L0-TLB** is the traditional dynamic address translation design in most current processors[27][32] and the habitual scheme for a physical COMA. Every memory access issued by the processor is translated by the TLB. All caches and the attraction memory are physically addressed.

### 3.2 L1-TLB

The **L1-TLB** scheme puts the address translation mechanism after a virtual FLC (first-level cache), but before a physical SLC[28]. Backpointers in the physical SLC (second-level cache) are needed to maintain inclusion.

### 3.3 L2-TLB

In the **L2-TLB** scheme both FLC and SLC are virtual. Backpointers are needed in the attraction memory to maintain inclusion. The software-managed address translation scheme[15] can be seen as an L2-TLB scheme which has 0 entry and traps the processor on every SLC misses.

### 3.4 L3-TLB

In a COMA the attraction memory acts as a cache and may also be virtually indexed and virtually tagged. In this case, address translation is postponed until a miss at the local node, as shown in Figure 2. The coherence protocol is maintained with the physical address, which points to the home node.

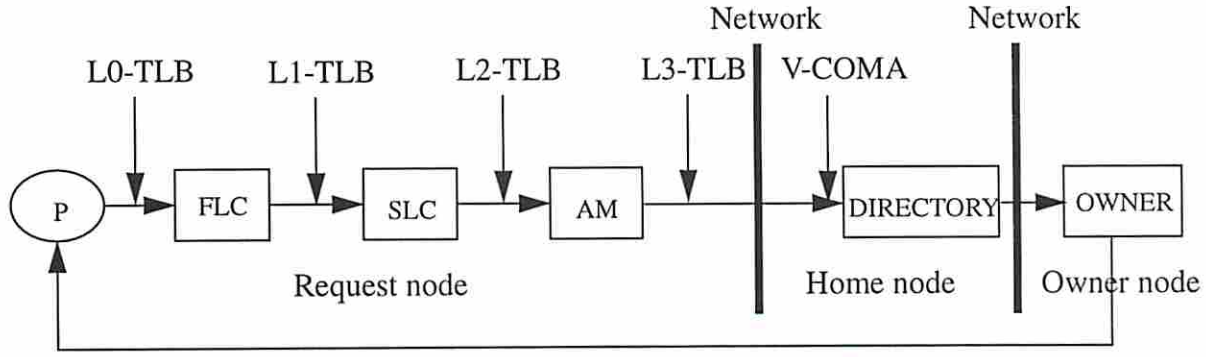


**Figure 2 Possible Locations of the TLB in COMA**

As in a typical COMA, the attraction memory in every node is divided into an equal number of sets. A *global set* is made of all the sets with the same number in all the attraction memories. This is illustrated in Figure 3 where each attraction memory is 4-way set associative. The size of each *global set*

increases linearly with both the number of processing nodes and the set size in each attraction memory.

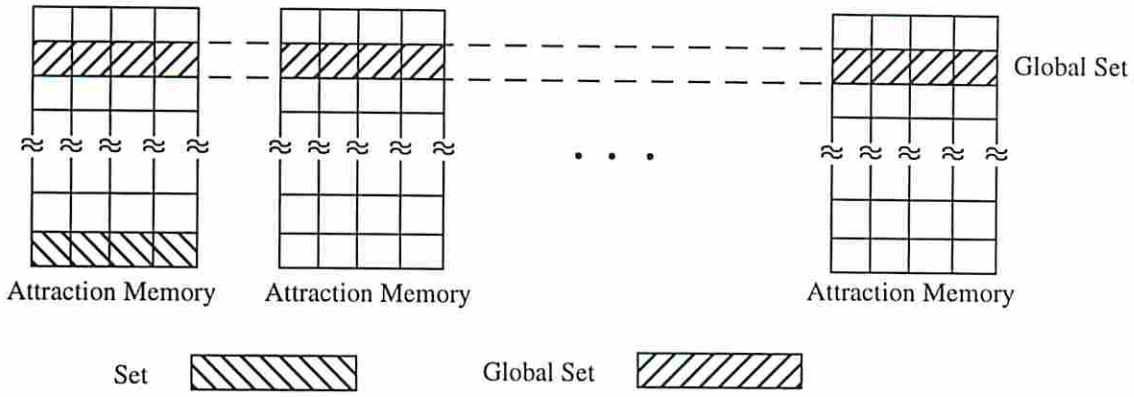


**Figure 3 Set and Global Set**

Unlike in a physically indexed attraction memory where a virtual address can map into different global sets depending on the virtual-to-physical address mapping, a block with a particular virtual address is restricted to reside in the global set indexed by its virtual address in the L3-TLB scheme. The number of *slots* for a given block is limited by the size of a global set and a page occupies the same slots in consecutive global sets, so that we can speak of the *slot of a page*. We can also speak of the *global page set*, which is made of all the contiguous global sets in which the blocks of the page can reside. The number of slots for a page is limited by *memory pressure*, which is given by the number of slots occupied in a global set divided by the size of the global set. When the pressure approaches 1, replication in the global set is inhibited. Therefore, if the pressure in the global sets to which a new page maps is too high, a page must be swapped out even though the pressure in other global sets is low. In a nutshell, the virtual-to-physical address mappings are set-associative instead of fully associative and the set size is equal to the size of a global set.

This page allocation strategy is equivalent to page coloring applied to the attraction memory. Page coloring allocates pages to pageframes sharing the same least significant bits so that the virtual address and the physical address index to the same set of a virtual cache[18]. As shown in Figure 4, if the virtual and the physical addresses have the same color, the set number for the data in the physically indexed attraction memory is determined and is the same as in the virtually indexed attraction memory. The most significant bits of the physical page number contain the slot number. The maximum number of slots of each *global set* is given by the number of processors multiplied by the associativity of each attraction memory.

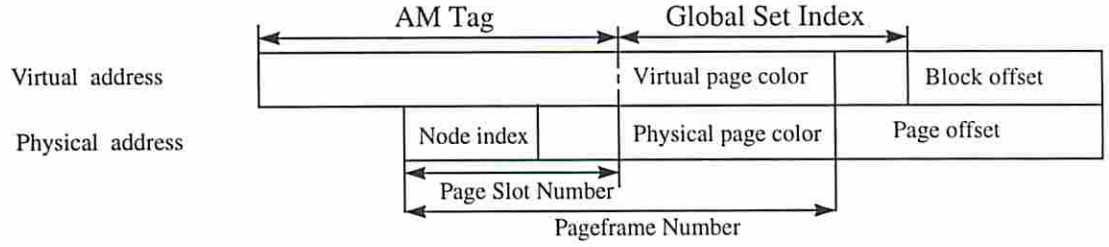


**Figure 4 Page Coloring in L3-TLB Attraction Memory**

## 3.5 V-COMA

In L3-TLB a TLB is still private to a processor node, which means replication of TLB entries across nodes and TLB consistency enforcement. We propose to move the address translation into the home node and

integrate it in the cache coherence protocol as shown in Figure 2. In this new design, which we call V-COMA, the support for address translation is located at the home node.

In V-COMA the attraction memory is accessed as in L3-TLB. However, the directory at the home node is accessed with virtual addresses instead of physical addresses. Thus, as in SHARED-TLB, the home node is fixed by the virtual address. Because its architecture is not conventional, V-COMA is described further in the next section.

# 4 V-COMA

## 4.1 V-COMA Processor Node Architecture

Figure 5 shows the architecture of one processing node in V-COMA. The processor P does not need TLB. The caches and the attraction memory are all virtually indexed and tagged. Some private memory in each processing node stores page tables for local pages and can also be used for private (non-replicated) data. The directory for local pages is stored in private memory or in a separate memory. The DLB (*Directory Lookaside Buffer*) is a cache to speed up translations from virtual addresses to directory addresses in the directory address space. The PE (Protocol Engine) executes the cache coherence protocol and refills DLB entries on DLB misses and is similar to the MAGIC chip in FLASH[22]. PE programs reside in local private memory. If the PE has enough processing ability and bandwidth, it could perform additional functions, such as periodically resetting the reference bits in page table entries.

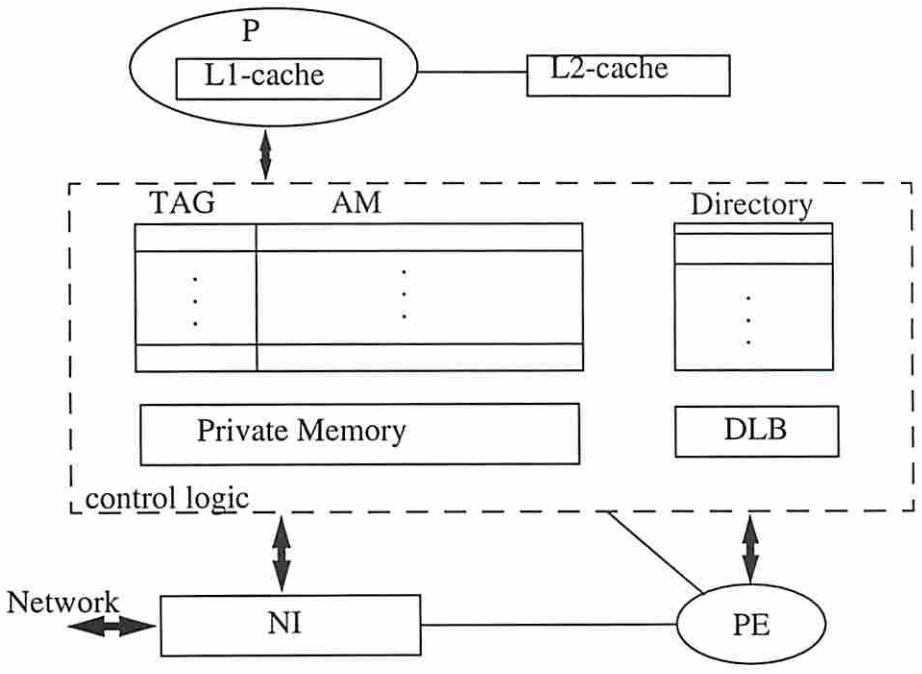


**Figure 5 Processor Node Architecture in V-COMA**

## 4.2 Directory Organization and Coherence Protocol

Virtual addresses are not suitable to address the directories. Compared to the physical address space, the virtual address space is huge and sparse. Every virtual address issued by the processor is not guaranteed to

reside in primary storage, or even exist. The size of the necessary directory memory is determined by the size of the main memory. Thus V-COMA moves the virtual address translation into the directory look up procedure by translating virtual address into *directory address* to find a directory entry.

The directory memory is organized in *directory pages*. Contiguous directory entries in a *directory page* correspond to contiguous memory blocks of a memory page. Therefore, a *directory page* has as many entries as there are blocks in a memory page. The directory memory is allocated and reclaimed in *directory page* unit by the virtual memory system. Due to the set-associative nature of the attraction memory, the mapping of virtual page to *directory page* in the page table is also set-associative, where the set is the *global set.*

Figure 6 shows how the virtual address is decomposed into fields to access the directory. Assume that the number of attraction memory sets per node is $S = 2^s$, the associativity is $K = 2^k$ blocks per set, the block size is $B = 2^b$ bytes, the number of processor nodes is $P = 2^p$, and the page size is $N = 2^n$ bytes. The p least significant bits of the page number point to the home node and the n-b most significant bits of the page displacement point to the entry within the directory page. s-p-n+b bits of the page number are used to index the *global set* in the page table (of size PxK) where the base address of the directory page can be found. The virtual address tag is then matched to the tags in the set. Since the global set is very large, we must use hashing or hierarchical translation based on the virtual tag to access the page table entry.

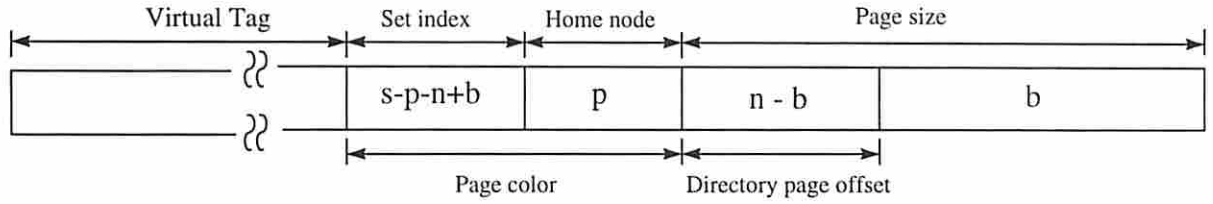


**Figure 6 Access to the Directory of V-COMA**

Because the latency to locate the directory for a memory block is in the critical path of the cache coherence protocol a DLB is put between the protocol engine and the directory memory in every processing node to accelerate the directory look up procedure. Accesses to the DLB are fully or set associative, as illustrated in Figure 7.

The protocol is write invalidate and is basically the same as in COMA-F[16]. Each block in attraction memory can be in one of four stable states: Shared, Master-shared, Exclusive and Invalid. Replacements of Exclusive or Master-shared copies send the block to the home node, which accepts the injection only if it has spare Invalid blocks in the same set. If not, the home node forwards the block copy to a random node. The selected node accepts the injection if it has an Invalid or Shared block available. If not, it forwards the request to the next node.

Cache coherence is maintained using virtual addresses. If a processor access is satisfied at the local node, no address translation is needed. If the access misses in the local node, a request is sent to the home node specified in the virtual address. The home node translates the virtual address to a directory address to find the state and copy set of the accessed block. The rest of the protocol transaction is handled with virtual addresses.

## 4.3 Impact on Virtual Memory Management

The impact of V-COMA's memory organization on memory management is no greater than for traditional COMAs. From the point of view of virtual memory management, the *directory page* corresponds to the

pageframe in a classical system.

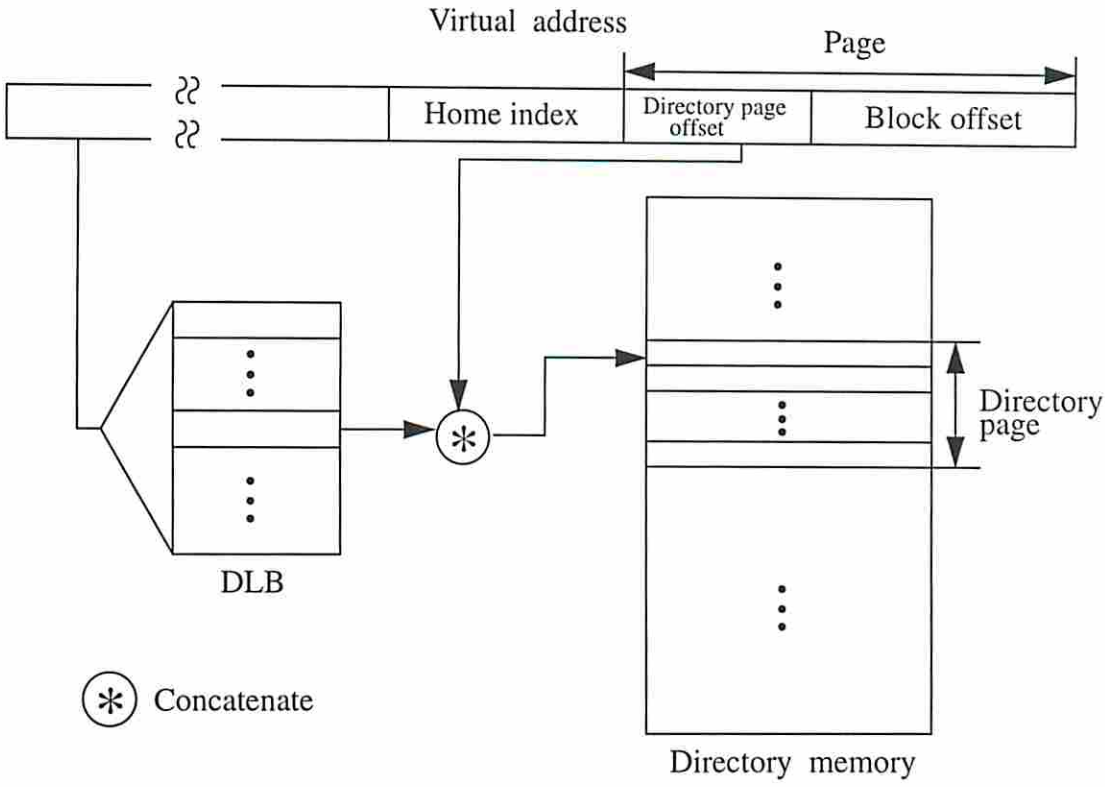


**Figure 7 V-COMA Address Translation with DLB**

On a page fault, a *directory page* and a page table entry are requested from the page's home node. A resident page may have to be swapped out by the page daemon if the memory pressure of the page's *global set* is higher than a threshold. A page table entry is filled with the necessary virtual address bits as well as other information, and is inserted in the page table. This action allocates a *directory page* to the new page.

Page access information such as the reference and modify bits must be maintained. Since the DLB sees the virtual address stream after the attraction memory the reference bit in the DLB of a page with many coherence misses is updated much more often than the reference bit of shared read-only or non-shared pages. The impact on the accuracy of the replacement algorithm is hard to predict. The Modify bit in the DLB is implemented as follows. When a writable page is first created or loaded from disk, the Modify bit is set immediately if the page fault is caused by a write access. Otherwise, the state of the page's memory blocks is set to Master-shared. Then, if any node tries to get exclusive ownership for any attraction memory block of the page, the request is sent to the home node where the modification bits are set in both the DLB and the page table.

Besides segment-level protection, page-level protection can also be implemented in V-COMA. The protection bits must be copied in every level of the memory hierarchy. If a processor wants to change the protection bits of a page, it sends a message to the home node which hosts the page. The PE at the home node changes the bits in the page table and in the DLB. Then, according to the directory entries, it sends update messages to the nodes holding the blocks of that page. Another solution is to dedicate access right checking hardware within the processor, such as the PLB (Protection Lookaside Buffer) proposed for Single Address Space Operating Systems (SASOS)[7][17].

## 5 Experimental Evaluation

### 5.1 Methodology

We have run execution-driven simulations to compare the five options for dynamic address translation in COMAs. Our simulated baseline architecture has 32 nodes, each of which has a 200 Mhz Sparc processor. Because we can only simulate applications in which the data set sizes are much smaller than those expected for the target machine, we have to scale down the sizes of attraction memories, caches, and TLBs.

Each node contains 4 MB of attraction memory, a 16 KB first-level cache (FLC), and a 64 KB second-level cache (SLC). The FLC is direct-mapped and write through with a block size of 32 bytes. The SLC is 4-way set associative and write-back with a block size of 64 bytes. The attraction memory is also 4-way set associative, and its block size is 128 bytes. The page size is 4 KB for all simulations. The timing and network model are the same as in [20]. An FLC hit has no latency charge and an SLC hit takes 6 cycles. A hit in attraction memory takes 74 cycles. The network is an 8-bit wide crossbar clocked at 100Mhz. An 8-byte request takes 16 cycles and a message containing a block takes 272 cycles.

We only simulate shared data accesses. Random replacement is used for the fully associative TLB/DLB. The parameters of the six Splash2 benchmarks[30] are shown in Table 1. The important working sets always fit in our simulated attraction memory, but sometimes do not fit in caches. The data sets fit in main memory and are preloaded so that no paging activity is simulated.

| Benchmark | Parameters | Shared Memory (MB) |
|---|---|---|
| RADIX | -n524288 -r2048 -m1048576 | 6.12 |
| FFT | -m20 -t | 51.29 |
| FMM | 16384 particles | 29.23 |
| OCEAN | 258*258 | 15.52 |
| RAYTRACE | car | 34.86 |
| BARNES | 16384 particles | 3.94 |

**Table 1: Benchmarks**

### 5.2 Address Translation Misses

Figure 8 shows the number of address translation misses per node as a function of the TLB/DLB size. The solid line for L2-TLB shows the number of misses in the case where SLC writebacks access the TLB. The L2-TLB/no_wback curve in dash line shows the number of L2-TLB misses without the writeback impact. Recall that we do not enforce inclusion between a TLB and the caches underneath it.

One obvious observation is that the number of address translation misses consistently decreases with the level of the TLB provided SLC writebacks do not access the TLB in L2-TLB. This is due to a *filtering effect* by the caches. Namely, the number of misses in a TLB cannot be larger than the number of misses in the cache underneath it. This effect is especially large when the TLB reach is less than the working set. SLC writebacks affect the number of L2-TLB misses significantly, especially for FFT and OCEAN. With the writebacks, L2-TLB is sometimes even worse than L0-TLB. As explained before this effect is due to the poor locality of writebacks. Thus, it may be preferable to keep physical pointers in a virtual SLC so that writebacks can bypass the TLB.

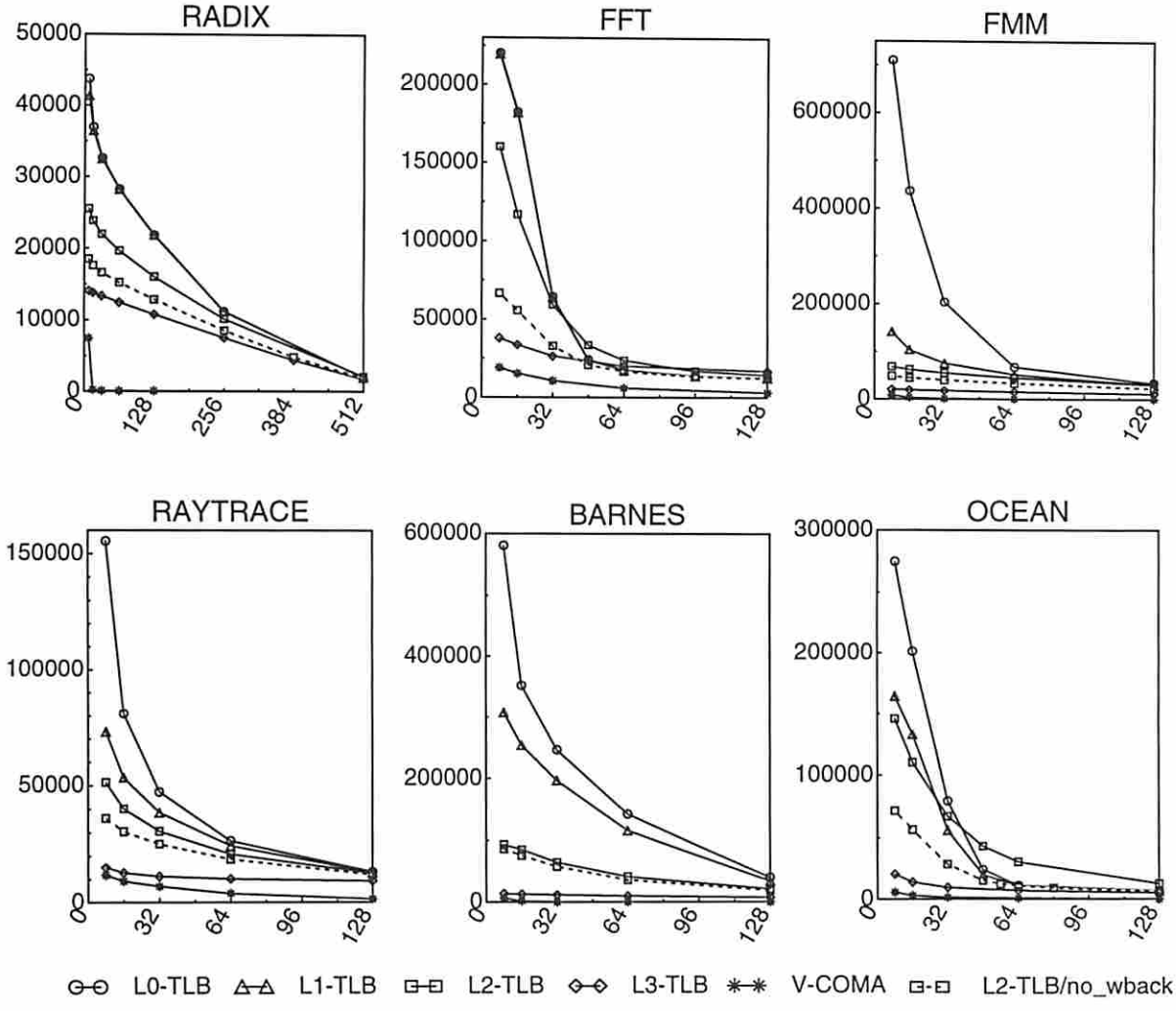


**Figure 8 Number of Address Translation Misses vs. TLB/DLB size**

The case of RADIX stands out. (Note the different scale in the graph.) The curves show no clear significant working set for any TLB organization or size, until the size reaches 512 entries. RADIX has a disproportionately large number of writes and these write accesses cause coherence transactions and are not filtered by the caches or the attraction memory. Except for RADIX, the TLB-miss curve for L3-TLB is much flatter than for L2-TLB.

As we expected, the number of DLB misses is negligible for all benchmarks, even for very small DLB sizes. This is due to a *sharing effect*. DLB entries are shared and are not replicated. Thus the effective amount of DLB entries increases proportionally with the number of processors. This is not true for systems with TLBs. This effect can be huge, as in RADIX. In each pass of RADIX, a key is written into a large output array shared and distributed among all nodes. The number of DLB misses in RADIX is consistently less than the number of TLB misses in an L3-TLB system with 32 times more TLB. (Recall that we simulate 32 processors.) All other benchmarks show similar trends, albeit not as pronounced because their access pattern is more complex.

Since, in the case of RADIX, the DLB miss rate is lower than the miss rate of a TLB 32 times larger another effect is at play besides the sharing effect. For example, a 16-entry DLB has even less misses than a 512-entry TLB in L3-TLB. This comes from a *prefetching effect*, a consequence of the shar-

ing of DLB entries. Although the nodes do not interfere with the TLBs of other nodes in L3-TLB, they can not benefit from each other either. For example, if processor 1 writes a shared data which is then read by processor 2, a TLB miss at processor 1 does not help prevent a TLB miss at processor 2. The impact of this prefetching effect is significant for cold misses when the whole working set fits in TLB or DLB. In this case, every page table entry is loaded only once in the whole system in V-COMA instead of once per node in L3-TLB.

| TLB/DLB size | 8 | | | | | 32 | | | | | 128 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SYSTEM | L0 | L1 | L2 | L3 | V-COMA | L0 | L1 | L2 | L3 | V-COMA | L0 | L1 | L2 | L3 | V-COMA |
| RADIX | 10.8 | 10.2 | 6.31 | 3.48 | 1.84 | 8.06 | 8.03 | 5.43 | 3.30 | 0.02 | 5.39 | 5.39 | 3.96 | 2.67 | 0.01 |
| FFT | 2.02 | 2.01 | 1.47 | 0.35 | 0.17 | 0.59 | 0.59 | 0.54 | 0.24 | 0.10 | 0.11 | 0.11 | 0.13 | 0.15 | 0.03 |
| FMM | 8.44 | 1.68 | 0.80 | 0.24 | 0.11 | 2.43 | 0.89 | 0.65 | 0.21 | 0.01 | 0.40 | 0.36 | 0.35 | 0.13 | .004 |
| RAYTRACE | 2.23 | 1.05 | 0.74 | 0.22 | 0.17 | 0.68 | 0.55 | 0.44 | 0.16 | 0.10 | 0.19 | 0.19 | 0.18 | 0.13 | 0.02 |
| BARNES | 2.68 | 1.42 | 0.43 | 0.06 | 0.03 | 1.13 | 0.91 | 0.30 | 0.05 | 0.0001 | 0.18 | 0.16 | 0.10 | 0.03 | 0.0001 |
| OCEAN | 6.45 | 3.86 | 3.42 | 0.48 | 0.14 | 1.87 | 1.32 | 1.58 | 0.23 | 0.04 | 0.16 | 0.16 | 0.30 | 0.12 | 0.003 |

**Table 2: TLB/DLB Miss rates Per Processor Reference (%)**

Table 2 shows the miss rates (misses/processor reference) for DLB and TLBs. In L0-TLB the miss rates are comparable to SLC miss rates when the TLB has 8 or 32 entries. Thus the TLB effects cannot be ignored. The situation improves somewhat in L2- and L3-TLB. V-COMA is the only case where we could neglect address translation misses as compared to cache misses.

Another way to compare the five options is to ask which TLB size it would take to have the same performance as the DLB in V-COMA. The answer to this question is shown in Table 3. We see that in many cases it would take TLBs with several hundred entries to have the same miss rate as an 8-entry DLB in V-COMA. Because of the writebacks, some equivalent L2-TLBs are larger than L1-TLBs, or even L0-TLBs.

| | L0-TLB | L1-TLB | L2-TLB | L3-TLB |
|---|---|---|---|---|
| RADIX | 360 | 360 | 344 | 256 |
| FFT | 60 | 60 | 86 | 86 |
| FMM | 335 | 321 | 347 | 187 |
| RAYTRACE | 157 | 152 | 144 | 27 |
| BARNES | 327 | 318 | 298 | 160 |
| OCEAN | 175 | 174 | 251 | 113 |

**Table 3: TLB Size Equivalent to an 8-entry DLB**

So far we have considered fully associative TLBs. However large, fully associative memories are slow and expensive. Figure 9 compares the number of misses for direct-mapped and fully associative TLBs and DLBs. The systems with fully associative TLB/DLBs are shown in dashed lines. We add /DM to the notation to indicate a direct mapped TLB/DLB. The huge gap between L0-TLB/DM and L0-TLB makes L0-TLB/DM an impractical design, and actually no existing processor has adopted it. However, the gap between direct-mapped and fully associative TLBs becomes quite small in L2-TLB, L3-TLB and even more so in V-COMA. The *filtering effect* of the direct mapped write through FLC is not as good as that of the larger set associative write back SLC. The *sharing effect* in the DLB also helps reduce the gap because the DLB coverage grows very fast with the DLB size in each node. The large coverage makes the organization of the DLB less important.

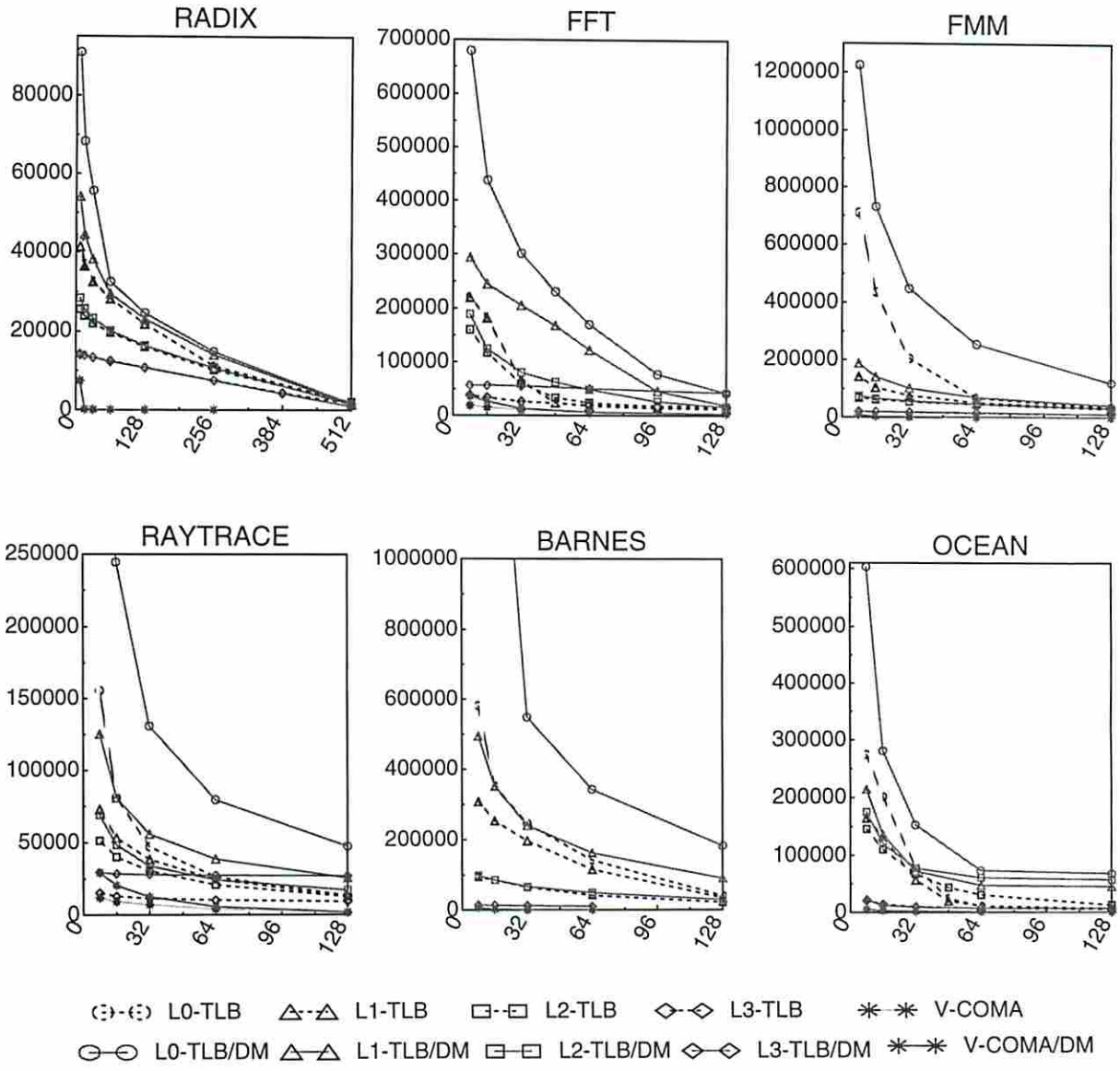


**Figure 9 Direct Mapped TLB/DLB**

## 5.3 Execution Time

We compare the execution times of V-COMA with those of the L0-TLB scheme, a traditional physical COMA. Physical addresses are assigned round robin. Based on the timing of our simulation model and the average TLB service times in current systems[21][24], we charge 40 cycles for each L0-TLB miss. Each DLB miss also takes 40 cycles. The memory consistency model is sequential consistency.

Table 4 shows the average TLB/DLB overhead divided by the average processor stall time on local and remote memory accesses for DLB/TLBs of sizes 8 and 16. Note that these small TLB sizes have been selected to offset the effects of the small data set sizes in our benchmarks. The data show that address translation is a significant part of the memory latency in the traditional L0-TLB system and that its effect is

at least comparable to the effect of memory consistency models [10]. Fortunately, the memory access penalties due to translation can be drastically cut to a point where they are negligible by translating addresses at the home node.

| | RADIX | FFT | FMM | RAYTRACE | BARNES | OCEAN |
|---|---|---|---|---|---|---|
| L0-TLB/8 | 10.61 | 15.24 | 96.54 | 30.95 | 38.14 | 21.53 |
| DLB/8 | 1.25 | 0.88 | 1.15 | 1.04 | 0.45 | 0.45 |
| L0-TLB/16 | 8.93 | 12.56 | 59.54 | 17.46 | 22.12 | 15.95 |
| DLB/16 | 0.04 | 0.76 | 0.38 | 0.82 | 0.01 | 0.23 |

**Table 4: Address Translation Time / Total stall time (%)**

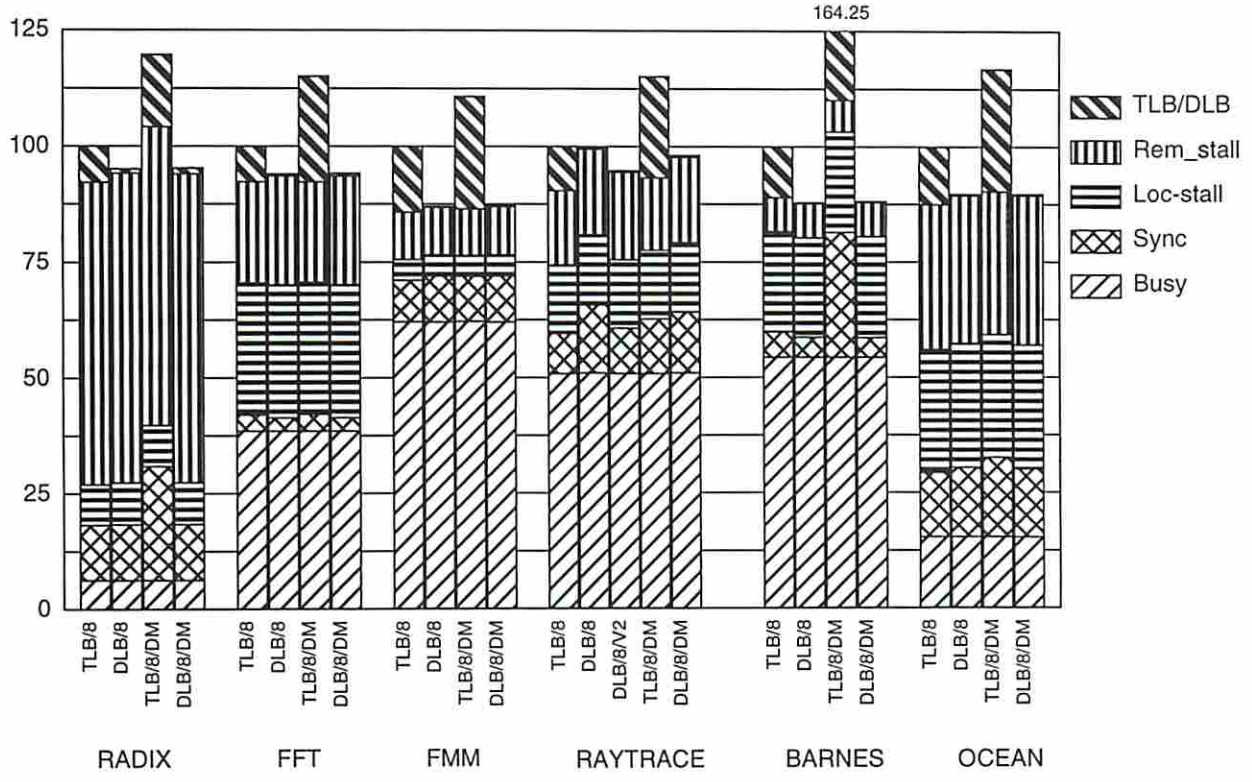


**Figure 10 Execution Time**

The effects on the execution time are shown in Figure 10, where Busy indicates the time spent executing the instructions in each processor, sync is the synchronization time, loc-stall counts time spent on local cache misses, and rem-stall refers to the service time for attraction memory misses. TLB/8 refers to L0-TLB with an 8-entry fully associative TLB, DLB/8 refers to V-COMA with an 8-entry fully associative DLB, TLB/8/DM and DLB/8/DM correspond to the same systems but with direct mapped TLB/DLBs.

Address translation overhead is negligible in V-COMA, as we expected. The total execution time improvement over the physical COMA depends on the original address translation overhead. Virtual address indexing of attraction memory does not have a significant impact on overall execution times (excluding the TLB overhead), although in general it is no better than for the physically indexed attraction memory except for BARNES. In RAYTRACE however we observe that the execution time of V-COMA (excluding the TLB overhead) is much larger than in COMA due to the increased synchronization time.

This effect is due to the fact that the virtual address layout is not optimized, while the round robin page-frame assignment is a good strategy for the COMA. RAYTRACE shows an extreme case of this effect since in the definition of *raystruct*, which is the private stack for the ray tree of each node, padding is used to avoid false sharing. The padding is aligned on multiple of 32KB in the virtual address space, which causes uneven conflicts in V-COMA leading to the increase of the synchronization time. By simply aligning the padding to one page size (4 KB), the synchronization time is reduced significantly, as shown as DLB/8/V2 in Figure 10. This example indicates that there is plenty of room for virtual address layout optimization for the V-COMA architecture.

## 6 Discussion

Although the V-COMA architecture eliminates address translation overheads, the restrictions on memory mappings may be of concern. If the attraction memory at each node can contain N pages and is K-way set associative, the number of global page sets is N/K. Each global page set contains up to P*K page slots, where P is the number of processing nodes. The memory management unit in a traditional physical COMA can control the *global set* index of a new page through the virtual-to-physical page mapping. On the other hand, V-COMA has no control on the *global set* index for the virtual address. The danger is that the conflicts in the virtual address space will affect the page fault rate and thus overall system performance.

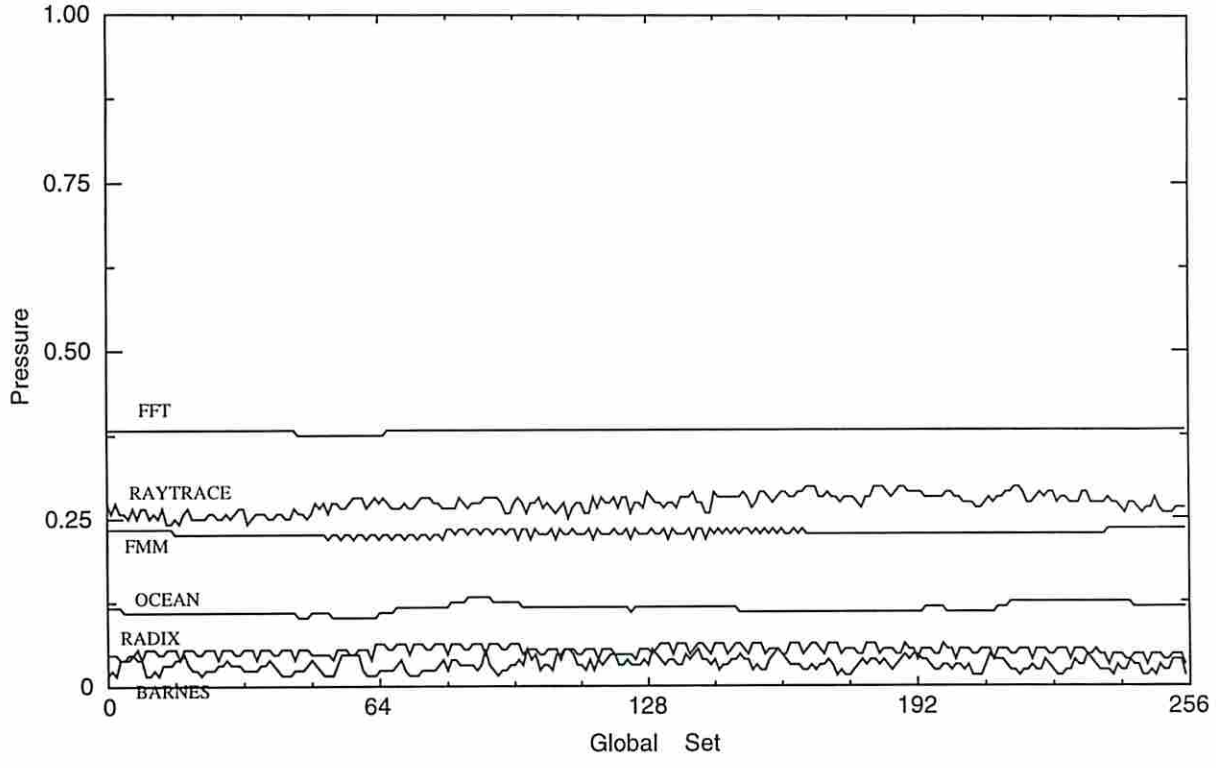


**Figure 11 Pressure Profile for the Benchmarks**

Although this danger is real, it can be overcome. First, the virtual address has good locality within each process[18]. Although virtual address layout tends to be uniform across processes and may cause high conflicts in some *global sets* and under-utilization of others, this situation can be improved by several methods. First a base address could be added to each virtual address; this base address can be different for every context. Second the code and data segments of different contexts could be located in different areas of their virtual space by the compiler[1]. Third, the operating system controls dynamic memory allocation,

and can optimize the scheduling strategy to spread paging activity evenly across the *global sets*. Fourth, the whole problem is a non-issue in large scale systems, considering that the capacity of each *global set* scales up linearly with the number of processing nodes. For example, if each node has a 32MB 8-way set associative attraction memory, a 1024 processor system has up to 1024*8 = 8192 page slots per global page set, which is equal to the total number of pageframes on each node!

Figure 11 shows the pressure profile on each global page set in our evaluation. It shows, that without even trying we observe a very uniform pressure on every global set. This is because program locality is maximized in the virtual space. In the physical space, locality only exists within a page. Our experience with allocating memory in physical COMA has taught us that the intervention of the memory management unit can result in disastrous --albeit well-intended-- physical page allocations for COMAs.

In fact, the advantages of this memory structure are very attractive. As a machine running on virtual addresses, V-COMA provides a simple and consistent hardware model to the operating system and the compiler where further optimization opportunities are possible. Virtual addresses are divided into S sets each of which has P*K page slots. The linear contiguous physical memory space no longer exists, and the management of the pages in the main memory is much simplified.

While the operating system guarantees correct execution, the performance of V-COMA is enhanced by the compiler and the programmer. Unlike in the traditional flat COMA where the programmer and compiler have no chance of affecting memory pressure and conflicts, the virtual address layout directly controls the occupation of global sets in the attraction memory. Static virtual address layout optimization can generate good programs which get more performance benefit from the V-COMA architecture.

Because virtual address is usually longer than physical address, V-COMA tends to increase the tag memory. For example, 32-bit PowerPC implements 52-bit virtual address and 32-bit physical address; 64-bit PowerPC implements 80-bit virtual address and 64-bit physical address. Including the access right bits, the virtual tag may 2 to 3 bytes longer than physical tag. This will increase the tag memory by 1.5% ~ 2.5% of the attraction memory (assuming 128 byte block size), and 3% ~ 4.5% for 64 bytes, and 6% ~ 9% for 32 bytes cache block size. If this is a concern in the particular design, further tag size reduction mechanism can be considered[29].

## 7 Related Work

DDM[13] and KSR-1[4] provided the seminal idea and the first implementation concepts for COMA. Because the hierarchical directory in their proposal increases the remote access latency a "flat" COMA (COMA-F)[16] was later proposed. The key idea was to decouple access methods to data and directory. We have extended the protocol of COMA-F to V-COMA and have used COMA-F in our comparisons.

Virtual address caches have been the topic of many papers[1][11][18][28][31]. A survey of the issues in uniprocessors and multiprocessors has been recently published[5][6]. Lynch[18] has evaluated page coloring issues. His simulations indicate that the page fault rate does not noticeably increase with the number of colors. He concludes that the use of coloring has no deleterious performance effects on paging activity. He also indicates that physical cache performance varies for each run, depending on the allocation of pages from the free list in the operating system. On the other hand, the performance of virtual caches are not sensitive to these implementation decisions.

Jacob et al.[15] proposed a software-managed address translation scheme where the hardware TLB is eliminated. A big virtually-indexed virtually-tagged SLC drastically cuts the frequency of address translations. This scheme can be considered as a 0-entry L2-TLB. Wood et al.[31] proposed an in-cache translation scheme in SPUR. Although it had a single level cache, we can categorize it as an L2-TLB

scheme because there is no physically indexed cache after the address translation mechanism.

Wang et al.[28] proposed the idea of a two-level virtual-real cache hierarchy where the TLB is after the FLC. We have called this system L1-TLB. They proposed the pointers stored in the two caches to solve the synonym and the writeback problems and to enforce inclusion.

Austin and Sohi[2] showed the bandwidth requirement on TLB in L0-TLB for multiple issue processors. Instead of brute force multi-ported TLBs, they evaluated several method to expand TLB bandwidth, such as interleaved TLB, multi-level TLB, piggyback ports which send the translation to simultaneous arriving requests, and pretranslation that allows a single translation request to be used for multiple memory accesses.

Talluri et al.[24][25] and Romer et al.[23] tried to use superpages to increase the TLB coverage without enlarging the TLB. In [24] two page sizes, 4KB and 64 KB, are supported, and page reservation restricts the allocation of physical memory and subblock TLB, analogously to a subblock cache. Romer et al.[23] used the scheme of on-line promotion. TLB misses are counted and when the miss count reaches a threshold, a superpage is constructed by copying and reconstructing the physical memory layout.

Teller[26] proposed an in-memory TLB scheme for UMA(Uniform Memory Access) architectures to solve the TLB consistency problem. This scheme is similar to SHARED-TLB in CC-NUMA and inspired the design of V-COMA.

## 8 Conclusions

In this paper, we have evaluated five options for virtual address translation in COMAs. In the context of virtually indexed and virtually tagged caches, the dynamic address translation mechanism can be located after the first-level cache access. Various issues for each of the five design options are discussed and the simulation results comparing the five designs are presented. A novel multiprocessor architecture called V-COMA is proposed. While still supporting a conventional virtual memory system, V-COMA reduces the address translation overhead to a minimum. Three effects contribute to this feat: the filtering effect, the sharing effect and the prefetching effect. Among the five designs evaluated, V-COMA is the only one to capitalize on all three effects.

Although the virtual memory has a set-associative organization, V-COMA scales well and works better in larger-scale systems. As a machine running on virtual address, V-COMA provides a simple and consistent hardware model to the operating system and the compiler in which further optimization opportunities are possible.

## 9 Acknowledgments

We would like to thank Adrian Moga for helping us build the simulation environment, and Kang-woo Lee for the discussions and suggestions on the benchmarks.

## 10 References

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

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

[3] E. Bugnion, J. M. Anderson, T. C. Mowry, M. Rosenblum, and M. S. Lam, "Compiler-Directed Page

Coloring for Multiprocessor," In *Proceedings of the 7th Conf. on Architecture Support for Programming Languages and Operating Systems(ASPLOS)*, Oct. 1996.

[4] H. Burkhardt III et al. "Overview of the KSR-1 Computer System," Technical Report KSR-TR-9202001, Kendall Square Research, Feb. 1992.

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

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

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

[8] J. Bradley Chen and Anita Borg. "A Simulation Based Study of TLB Performance," In *Proceedings of the 19th Annual International Symposium on Computer Architecture(ISCA)*, pages 114-123, May 1992.

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

[10] K. Gharachorloo, A. Gupta, and J. Hennessy. "Performance Evaluation of Memory Consistency Models for Shared-Memory Multiprocessors," In *Proceedings of the 4th Conf. on Architecture Support for Programming Languages and Operating Systems(ASPLOS)*, pages 245-257, 1991.

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

[12] L. Gwennap, "Design Concepts for Merced, Forecasting the Inner Workings of the Decade's Most Anticipated Processor," pages 9-11, *Microprocessor Report*, vol. 11, no. 3, March 10, 1997.

[13] E. Hagersten, A. Landin, and S. Haridi. "DDM-A Cache-Only Memory Architecture," IEEE *Computer*, vol. 25, no. 9, pages 44-54, Sep. 1992.

[14] Jerry Huck, and Jim Hays. "Architecture Support for Translation Table Management in Large Address Space Machines," In *Proceedings of the 20th Annual International Symposium on Computer Architecture(ISCA)*, pages 39-50, 1993.

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

[16] T. Joe. "COMA-F: A Non-Hierarchical Cache Only Memory Architecture," PhD. Thesis, Stanford, 1995.

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

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

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

[20] Adrian Moga, Alain Gefflaut, and Michel Dubois, "Hardware vs. Software Implementation of COMA", In *Proceedings of the 1997 Int'l Conference on Parallel Processing*, pages 248-256, August 1997.

[21] David Nagle, Richard Uhlig, Tim Stanley, Stuart Sechrest, Trevor Mudge, and Richard Brown. "Design Tradeoffs for Software-Managed TLBs," In *Proceedings of the 20th Annual International Symposium on Computer Architecture(ISCA)*, pages 27-38, 1993.

[22] J. Kuskin, D. Ofelt, M. Heinrich, J. Heinlein, R. Simoni, K. Gharachorloo, J. Chapin, D. Nakahira, J. Baxter, M. Horowitz, A. Gupta, M. Rosenblum, and J. Hennessy. "The Stanford FLASH Multiprocessor,"

In *Proceedings of the 21st Annual International Symposium on Computer Architecture(ISCA),* pages 302-313, 1994.

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

[24] M. Talluri and M. D. Hill. "Surpassing the TLB Performance of Superpages with Less Operating System Support," In *Proceedings of the 6th Conf. on Architecture Support for Programming Languages and Operating Systems(ASPLOS),* 1994.

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

[26] Patricia Teller and Allan Gottlieb. "Locating Multiprocessor TLBs at Memory," In *Proceedings of the 27th Annual Hawaii International Conference on System Science*, pages 554-563, 1994.

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

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

[29] Hong, Wang, Tong Sun, and Qing Yang, "CAT -- Caching Address Tags, A Technique for Reducing Area Cost of On-chip Caches", In *Proceedings of the 22nd Annual International Symposium on Computer Architecture(ISCA),* page 381-390, 1995.

[30] S. C. Woo, M. Ohara, and E. Torrie. "The SPLASH-2 Programs: Characterization and Methodological Considerations," In *Proceedings of the 22nd Annual International Symposium on Computer Architecture(ISCA),* pages 24-36, 1995.

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

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