March 1986

Report No. STAN-CS-86-1105
Also numbered CSL-86-294

Software-Controlled Caches in the VMP Multiprocessor

by

David R. Cheriton

Gert A. Slavenburg

Patrick D. Boyle

Department of Computer Science

Stanford University
Stanford, CA 94305

Software-Controlled Caches in the VMP Multiprocessor[†]

David R. Cheriton
Stanford University

Gert A. Slavenburg
Philips Research

Patrick D. Boyle
Stanford University

Abstract

VMP is an experimental multiprocessor that follows the familiar basic design of multiple processors, each with a cache, connected by a shared bus to global memory. Each processor has a synchronous, virtually addressed, single master connection to its cache, providing very high memory bandwidth. An unusually large cache page size and fast sequential memory copy hardware make it feasible for cache misses to be handled in software, analogously to the handling of virtual memory page faults. Hardware support for cache consistency is limited to a simple state machine that monitors the bus and interrupts the processor when a cache consistency action is required.

In this paper, we show how the VMP design pro vides the high memory bandwidth required by modern high-performance processors with a minimum of hardware complexity and cost. We also describe simple solutions to the consistency problems associated with virtually addressed caches. Simulation results indicate that the design achieves good performance providing data contention is not excessive.

This work was sponsored in part by the National Science Foundation Grant DCR-83-52048 and by Philips Research, Bell-Northern Research, ATT Information Systems and NCR.

1 Introduction

VMP is an experimental shared memory multiprocessor being built at Stanford University. It follows the familiar **model[4]** of multiple processors connected by a shared bus to global memory with per-processor caches to reduce bus traffic.

Our research focuses on the problem of connecting multiple high-performance processors to a shared memory without significant performance degradation, rather than connecting a large number of processors of more modest **capabilities[14]** or not providing shared **memory[17]**. By ***high-performance***, we mean the 20-30 MIPS microprocessors of modest cost expected **in** the near future.

This particular focus is motivated by three observations. First, it appears to be much easier to program parallel applications for shared memory machines than for networked processors because management of the shared program state is familiar and direct. Second, initial **experimentation[5,13]** with parallel applications indicates that few, fast processors are more effective than many slow processors, simply because most applications exhibit a low degree of parallelism. Finally, we are interested in medium to high performance workstations with uniprocessor or multiprocessor configurations. For these machines, the processor of choice is obviously the microprocessor of greatest performance within standard VLSI technology.

The performance of future processors will be limited primarily by the memory bandwidth provided. Current conventional processors, such as the Motorola 68020, run at about 75 to 80 percent memory bandwidth utilization. Some RISC processors achieve much higher utilization. Thus, the primary design problem for multiprocessor machines is providing sufficient memory bandwidth to a shared memory to accommodate multiple processors. This view argues for per-processor caches with very efficient **processor-**cache coupling.

In the VMP design, each processor has a synchronous, virtually addressed, single master connection to its cache, providing very high memory bandwidth except on cache miss. An unusually large cache page size and fast sequential memory copy hardware make it feasible for cache misses to be handled in software, analogously to the handling of virtual memory page faults. Hardware support for cache consistency

† **A version of this paper will be** presented at the 13th Symposium on Computer Architecture in June 1986.

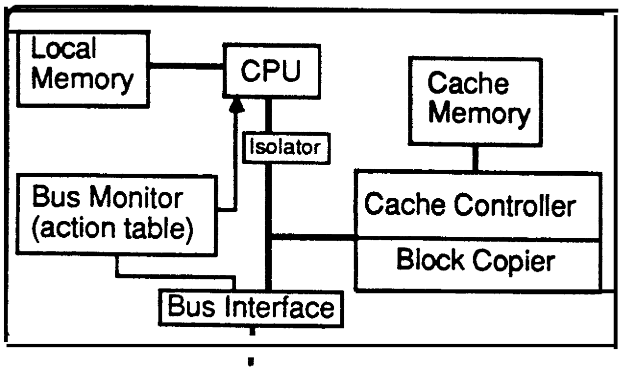

Figure 1: VMP Processor Board Organization

is limited to a simple state machine that monitors the bus and interrupts the processor when a cache consistency action is required.

We argue that these simple hardware resources, operated under software control, provide memory bandwidth for a very high-performance processor and bring the power of the processor and the flexibility of software management to bear on the cache management (and virtual memory) problem. Simulation results indicate that the design achieves good performance providing data contention is not excessive. We also describe simple solutions to the consistency problems associated with virtually addressed caches. The paper emphasizes the techniques rather than our specific hard ware design.

The next section describes the cache miss handling mechanism. Section 3 describes our approach to cache consistency, including consistency with respect to virtual address translation. Section 4 describes additional details of the VMP design. Section 5 provides some indication of expected performance for VMP and raises some software issues with the design. Section 6 compares this design to some other representative multiprocessor designs. We close with a summary of the key points plus an indication of future directions.

2 I Cache Access and Cache Miss Handling

The processor is directly connected to a ***virtually addressed*** cache, as depicted in Figure 1. That is, the cache contents are addressed by virtual address, rather than by physical addresses.[1] Thus, in the absence of a cache miss, the memory reference is satisfied at maximum speed because the processor is the single master of the cache and it executes synchronously with respect to the cache, i.e. no arbitration is required and there is no virtual-tophysical address translation as part of a cache reference.

[1] An address space identifier is included as part of the address presented to the cache so that the cache need not be flushed on context switch.

The processor is connected to some local memory in the same synchronous, single-master fashion. High-order bits of the address discriminate local memory references from cache references so no significant delay is introduced by having the two memories. Local memory is required for storing the code and data associated with cache miss handling, ensuring there can be no cache miss in the cache miss handling software.

On cache miss, the cache controller signals a processor exception interrupt (bus error) and generates a suggested cache ***slot***[2] to use for the missing cache **page**.

On exception interrupt, the processor saves its state on the supervisor stack in local memory and traps to the cache miss handler routine, also stored in local memory. The processor writes out the cache page if it has been modified. It then maps the virtual address that generated the miss to the physical address for the associated cache page. Assuming the virtual memory page is present in the main memory, the processor instructs the block copier to copy the required data **from** main memory into the cache, specifying the cache flags to be assigned to the cache slot if the copy succeeds. Concurrently with the copy operation the processor updates its data structures describing the current cache contents, returns from the original exception and continues execution as soon as the copy operation completes. If the copy operation fails (for instance because it is aborted by one of the bus monitors), the cache flags are left unchanged and the processor traps again in retrying the instruction, causing it to try again. If the required data is not in main memory, the operating system page fault handler is given control.

The virtual-to-physical mapping may be performed in a variety of **ways[9]**. A **two-level** page table is the scheme proposed for VMP. With page tables stored in virtual memory, a cache miss may result in additional cache misses as the processor references the page table. Each such miss results in the processor stacking another level of exception state on the **supervisor** stack contained in local memory. Some minimum amount of page table information is maintained in local memory (or non-cached global memory) so there is a small bounded depth to page table misses.

[2] A cache ***slot*** is the **cache** element holding **a** cache **page. The** term cache ***page*** is **used** the same **as *virtual page*** is **used** for **conventional virtual** memory **systems.** A ***cache page frame*** **is a** portion of main memory corresponding to one cache page.

After handling the cache misses (if any) involved with virtual address translation, the processor returns to handling the original cache miss.

A cache miss can also occur when the processor attempts to write data for which it has not secured write access. In this case, it negotiates write permission using the cache consistency protocol described in Section 3.

Cache miss handling by the processor is facilitated by the hardware providing fast data transfer. This hardware exploits three main techniques for performance:

- Sequential Memory Access: Main memory boards are optimized for fast sequential operation by using static column RAM chips (which provide 60 nanosecond access to successive locations). The first access to the memory board takes 300 ns but each subsequent sequential reference takes less than 100 ns.

- Sequential Bus Protocol: Bus protocols are optimized for sequential access by issuing a single address for a transfer and then simply strobing the data words across, relying on the source and destination modules to automatically increment the source and destination addresses. This is provided by the VMEbus block transfer mode in our prototype machine.

- Block Copier: A specialized block copy mechanism is embedded in the cache controller that allows us to take advantage of the sequential access on the VMEbus and memory board. It also eliminates the instruction fetching overhead which would arise if the processor did the copy.[3] The block copier can operate concurrently with the CPU executing out of local memory.

The block copier significantly reduces the bus occupancy for the transfers as well as the elapsed time. For example, the VMEbus-based VMP block copier should transfer data at 40 megabytes per second, achieving 100 percent VMEbus utilization during the transfer. In contrast, a simple copy loop using the processor can achieve less than 5 megabytes per second at best. The block copier allows some overlap of the copy time with the bookkeeping performed by the processor on cache miss.

Cache miss handling is more complicated with a virtually addressed cache than with a physically addressed cache. A virtually addressed cache requires virtual-to-physical address translation on cache miss and, if page tables are stored in virtual memory, has the possibility of incurring a real page fault as part of cache miss handling.

The software implementation of cache miss handling has the benefit of replacing rather complex cache-control hardware with relatively simple hardware: local memory that holds the cache management software. It also offers the flexibility to experiment with different techniques of virtual-to-physical address translation and cache loading and replacement policies without hardware modification.

The major concern with software controlled caches is performance. We claim that, by choosing an unconventionally large cache page size (and keeping the number of cache slots and degree of associativity large enough), one reduces the cache miss rate so that the overhead of software cache management is not a problem. The effect of cache page size on cache hit ratio is discussed in Section 5.

It remains to address the problem of maintaining cache consistency. Note that, with a virtually addressed cache, cache consistency is not strictly a multiprocessor issue. A single processor cache can be inconsistent with respect to itself if the same physical memory is mapped to two different virtual addresses and both virtual addresses are represented in the (single) cache.

3 Cache Consistency

There are two cache consistency problems to solve:

- ensuring that all copies of a cache page are consistent across all processors, and

- ensuring that the virtual-to-physical translation implicit in the per-processor caches is consistent with that specified by the system page tables.

We first describe the cache consistency protocol and then how this protocol is implemented with the aid of the bus monitor.

3.1 Cache Consistency Protocol

Cache consistency is maintained by a variant of the distributed ownership protocol described by Frank[11] and Goodman[12]. Main memory is viewed as a sequence of *cache* page *fames*.[4] For consistency, a cache page must be in one of two states:

[3]The elimination of instruction fetch is secondary in effect compared to the use of sequentid access, given that a copy loop fits in the processor's on-chip instruction buffer.

[4]Our prototype allows for experimentation with cache page sizes of 128, 256, and 512 bytes.

- shared - Main memory contains the most recently written value of the cache page. Several copies of the block may exist elsewhere, all of them being identical to that in main memory.

- private - Some cache *i* contains the only copy of the page. In this case, cache *i* is said to own this cache page.

The processors use an extended form of read and write bus transactions that specify if ownership is being requested or released. It is up to each processor to observe and respond to bus transactions so as to ensure each page of memory is in one of the two legal states.

There are six types of bus transactions associated with bus monitor operation (plus the normal ones which are not, those used by DMA devices and CPUs to access device registers). A processor issues one of these six types of bus transactions, depending on the reason for the bus transaction (the first five are *consistency-related* bus transactions):

- read-shared - to acquire a non-exclusive or shared copy of a cache page.

- read-private - to acquire an exclusive copy of a cache page. The processor issues this bus transaction when it incurs a cache miss on a write to an address within that cache page but has no copy of that cache page.

- assert-ownership - to gain exclusive ownership of a cache page without reading it from main memory. It presumably acquired a shared copy of the cache page earlier using a read-shared operation.

- write-back - to write the cache page back to main memory, releasing ownership of the page.

- notify - to send notification to a processor (described in 5.4)

- write action table - to write an entry in the action table (described below).

To allow the processor to execute concurrently with bus transactions, we provide a simple state machine called a *bus monitor* that monitors the bus and interrupts the processor when either consistency actions are required or notification is signalled.

3.2 Per-Processor Bus Monitor

The bus monitor[5] performs one of four actions on each bus transaction depending on the type of bus transaction, the physical address of the bus transaction and the contents of the bus monitor's action table. The bus monitor's action table contains a two-bit entry per physical cache page frame[6] (of main memory) indicating:

- 00 - do nothing

- 01 - interrupt local processor on read-private, assert-ownership (ignore read-shared or notify)

- 10 - abort bus transaction and interrupt local processor on any consistency-related bus transactions (including read-shared)

- 11 - interrupt processor on a notification transaction.

The main function of the bus monitor is to enforce cache consistency, however the action table code 10 can be used to "protect" a page (prevent its modification or a change in its state), and entry 11 can be used for notification (see 5.4).

The action table of the bus monitor associated with a particular CPU is normally updated as a side effect of (and concurrently with) a consistency-related bus transaction issued by that CPU. Thus, in the common cases, checking and updating the action table over the bus does not entail additional bus occupancy. The action table can also be updated by the CPU using the *write action table* bus transaction. Update as part of a consistency-related bus transaction only takes place if the bus transaction is not aborted. The consistency check interval and action table update interval, each of 150 nanoseconds, are overlapped with the block transfer, as shown in Figure 2. On abort, the bus transaction is terminated at the end of the current memory reference. The assertownership bus transaction is a degenerate form of this behavior since it does not involve block transfer. Updating the action table as part of bus transactions minimizes bus overhead for action table management and avoids the cost of a dual-ported action table, the other solution. Note that completion of a few transfers during the consistency check does not compromise the correctness of main memory because write-back is the only

[5]The main difference between our bus monitor and a snoop is that the bus monitor is not connected to the cache (does not share the cache tag matching hardware, the cache flags, or even have a copy of the flags) and thus does not reduce the cpu/cache bandwidth. It can operate at the leisurely pace of our relatively long bus transactions rather than at the memory reference speeds required when using small cache page sizes.

[6]Allowing a maximum of 8 megabytes of physical memory for the prototype with 128 (256, 512) byte pages, each bus monitor has 16 (8, 4) kilobytes of memory for its action table. A larger physical memory would require additional memory for the action table.

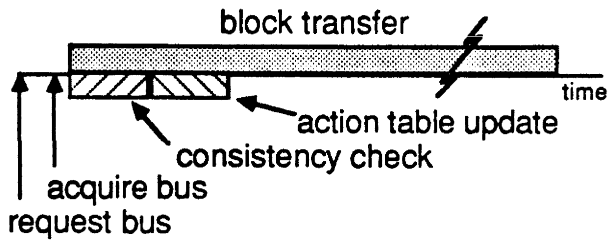

Figure 2: Action Table Update in a Bus Transaction

bus transaction that modifies main memory. Write-backs are only issued if a cache is releasing a privately held page and so are never aborted (unless there has been a consistency protocol violation).

The bus monitor is connected to the processor by a non-ma&able interrupt and a FIFO queue of interrupt requests. Each time a bus transaction occurs that should interrupt the processor, a word is queued in the FIFO for the processor. The word specifies the type of bus transaction and the physical address associated with the bus transaction. The FIFO provides a maximum of 128 entries, minimizing the likelihood of an interrupt word being lost. However, the FIFO also sets a flag for the processor when an interrupt word is dropped because the FIFO is full.

The bus monitor is a fairly general-purpose hardware resource available to each processor. We plan to explore its use in a variety of settings. However, its primary use is for ensuring cache consistency, as described in the next two subsections.

3.3 Cache Page Consistency

Each processor sets the action table of its bus monitor according to the cache pages its cache holds and acts on bus monitor interrupts so as to enforce this 2-state consistency of cache pages. There are three cases to consider for each cache page frame *k* in physical memory, corresponding to there being no copy, a shared copy or a private copy of the page in the processor's cache.

No Copy: The action table entry for cache page *k* is 00, indicating that the bus monitor can ignore all bus transactions on this cache page.

Shared Copy: The k-th action table entry is set to 01 causing the bus monitor to ignore read-shared transactions, and interrupt on read-private or assert-ownership bus transactions. Write-back operations are protocol violations and result in an abort and interrupt. Note that, due to virtual memory aliasing, the cache may contain (shared) multiple copies of this cache page in different cache slots. On interrupt from a read-private or assert-ownership bus transaction, the processor invalidates the cache slots holding this cache page and sets the k-th action table entry to 00. Consequently, when a cache page becomes private, all other cached copies of the page are discarded in parallel.

Private Copy: The k-th action table entry is set to 10 causing the bus monitor to abort the bus transaction and interrupt the processor on all consistency-related bus transactions on this page (including write-back operations which are protocol violations). On interrupt, the processor writes out the cache page (if dirty). If the bus transaction was read-private (or assert-ownership), it invalidates the cache page and sets the action table entry to 00. If not, it "downgrades" the cache page to read-only and changes the action table entry to 01 (shared). The processor issuing the bus transaction detects that the bus transaction was aborted and retries the bus transaction.

This scheme also solves the *alias* consistency problem that arises with a physical cache page mapped to two or more different virtual addresses. Each processor observes the consistency protocol "competing against itself". Thus, for instance, should a processor issue a read-shared for a cache page its cache already owns (referenced by a different virtual address), its own bus monitor will abort the bus transaction and interrupt that CPU. In response to the interrupt, the CPU flushes (or writes back) the owned page. The read-shared bus transaction is then retried.

Using this protocol, a request for a shared copy of a shared cache page is satisfied immediately. A request for a shared copy of a private cache page fails but causes the owner to relinquish ownership, allowing the requestor to succeed on retry. A request for a private copy of a shared cache page succeeds immediately but causes all cache copies of the cache page to be discarded. A request for a private copy of a private cache page fails but causes the owner to relinquish ownership.

Each processor is trusted to set its bus monitor action table appropriately for the cache pages it holds and to act on interrupts from the bus monitor according to this protocol. Information about the state of each cache page and the mapping from physical address to cache page is maintained by the processor in the local memory.

The consistency scheme is deadlock-free because ownership of cache pages can be preempted (no bloc king) and a processor is guaranteed to make at least one successful reference to a newly acquired page before that page is flushed from the cache (non-zero progress). One worst case example is that of two processors simultaneously attempting to acquire a pri-

vate copy of a cache page. In this case, the first processor to acquire the bus gets the page, then the second issues the read-private resulting in **an** interrupt to the first processor by the first's bus monitor leading to subsequent flushing of the page from the first processor's cache, and so on. However, interrupts are only serviced between instructions and the CPU blocks on the cache controller mid-instruction while awaiting the completion of the block transfer. Thus, the first processor makes at least one successful reference so the contention results in performance degradation but not deadlock.

Correctness of consistency maintenance is rendered independent of the processor's ability to keep up with bus monitor interrupts as follows. The interrupt FIFO includes a flag that indicates that an interrupt word was dropped (which only occurs if the processor is unable to keep up with the bus monitor interrupt rate). When this flag is set, the processor recovers by invalidating (or rereading) shared cache entries from main memory and updating its bus monitor action table. Note that **loss** of the interrupt word for a bus transaction requesting ownership of cache page owned by this processor is not a problem since the bus transaction is aborted by the bus monitor and then retried by the requesting processor until successful.

Dropping an interrupt word in the bus monitor FIFO is extremely unlikely for several reasons. First, the FIFO queue provides considerable buffer space giving the processor time to handle bursts of consistency actions. Second, the only operations that leave the processor unresponsive to these interrupts for a significant time are its block transfers. During the transfer the bus is fully consumed so other bus transactions cannot occur, limiting the rate of accumulation of interrupt words. Finally, the rate of interrupt word generation is no worse than the rate of cache misses, which is assumed to be reasonably low. (Of course, there is no problem with the bus monitor keeping up with the rate of bus transactions.)

The flexibility of the bus monitor allows **VME**-standard DMA devices to be used in the system. To set up a DMA into a particular area of memory, the operating system code acquires a high-level lock on that area of memory so that it is not accessed by other processors. The cache management software then does an assert-ownership bus transaction on this area of memory, forcing every other processor to **dis**card any cached copies of this memory or write back the private copy, if any. It then sets the bus monitor to abort any consistency-related bus transactions addressing this area (which should not occur in any case). Since DMA operations have no associated consistency operation the DMA completes without abort by a bus monitor. Once the DMA transfer completes, the processor can release its lock on this area of memory at the operating system level and clear the corresponding entries in the bus monitor's action table.

3.4 Virtual Address **Translation** Consistency

A virtually addressed cache implicitly stores a portion of the virtual-to-physical address mapping specified in the operating system page tables. To ensure consistency, this implicit mapping must be updated when the page tables change. This problem of virtual address translation consistency is handled in a straight-forward fashion in our design, as described below.

The operating system and cache management software ensure that every valid cache slot corresponds to some portion of a virtual memory page currently in main memory. To change the mapping of virtual page up which currently maps to physical page pp, the processor first issues a read-private for the cache page ***pt*** corresponding to the page table entry for up. If the page table is in virtual memory, obtaining exclusive ownership of ***pt*** may entail page faults as well. The processor then issues an assert-ownership on page pp, causing all cached copies to be flushed or written back, depending on whether the copy is shared or private. This flushes the implicit mappings for this virtual page in all other processor caches. The processor then updates the page table entry and relinquishes ownership of the two cache pages. Note that cache page pp need not be read into the cache of the processor performing this mapping operation.

Deletion of an address space can be handled similarly with an assert-ownership on every resident page in the address space.

A similar technique can be used to keep page table reference information consistent with cache page references in the cache. The page-out daemon can periodically use assert-ownership to flush cache pages chosen as candidates for reclamation out of all caches. The processors then update the page table reference information if they subsequently refer to these cache **pages**.

The software implementation of address translation in combination with the bus monitor and local memory allows considerable latitude in handling virtual address translation consistency. We have sketched a basic scheme permitting the storage of page tables in either physical memory or virtual memory.

4 Details of VMP

This section provides some details of VMP, the multiprocessor machine we are building to investigate the performance of the cache design described in the previous sec tions.

The system consists of the following major components:

- A shared central bus (VMEbus) that is used for all communication between processing nodes, memory and I/O devices.
- A central memory connected to the bus. The memory is optimized to do the transfer of cache pages at 40 MBytes/Second.
- I/O units which adhere to the standard VME protocol and can be obtained from external suppliers. Expected I/O units include an Ethernet interface and a framebuffer.
- Several VMP processor boards.

Each VMP processor board consists of a 68020 CPU running at maximum speed (currently 60 nanosecond cycle, 180 nanosecond memory cycle) coupled to a 68881 FPU (Floating Point Coprocessor), local RAM (32 **KBytes**), a 4-way set associative 256 **KByte** cache that responds to virtual addresses, a bus monitor (with associated action table), and local devices (UART, timer). The CPU, FPU, local RAM, local devices and bus monitor are connected to a private **onboard** bus which may be connected to the VME interface through the ***bus isolator***. The bus isolator permits concurrent execution of the CPU accessing local memory with transfers between the cache and VME memory. Note the absence of components found in other systems: memory management unit, translation lookaside buffer and reverse translation buffer.

The basic VMP processor board organization is shown in Figure 1. The memory space seen by the CPU is divided into 5 regions. The lowest addressed region (2^{27} bytes or 128 MBytes) maps straight-through to VME address space and is used to access device registers and execute boot ROM code. The next region (128 MB) is set aside for local accesses (local memory, **ASID** register, bus monitor FIFO, and other local devices). The third region (128 MB) addresses cache control. The fourth region (128 MB) addresses kernel virtual address space. The last region (3.5 Gigabytes) is the user virtual address space, which is extended by an 8 bit Address Space Identifier *ASID*.[7] Accesses to regions other than the user virtual memory require supervisor privilege.[8]

[7]This is similar in function to the context register in the SUN workstation architecture.

Virtual addresses are mapped into the **4-way** set associative cache. The cache page replacement strategy is LRU, with the replacement slot "suggested" by the hardware based on references. For each cache slot, flags are maintained that indicate: valid, modified, exclusive-ownership, supervisor writable, user readable and user writable. Because the cache matches on <ASID, VirtAddress>, the operating system simply changes the *ASID* to specify the new address space on each context switch.

The cache in the prototype is configurable for a choice of 128,256 or 512 byte cache pages to allow us to experiment with a variety of cache page sizes. The number of sets is variable from 1 to 4, and number of pages per set is variable from 16 to 256. In addition to experimenting with **different** hardware configurations, we are interested in investigating the benefit of software techniques that improve the utilization of large cache blocks.

5 Expected Performance

We are building a prototype of the VMP design that is highly instrumented in order to measure performance and investigate the effects of different cache page sizes, cache sizes, associativity, modifications to the cache management software, and various software techniques for improving locality and reducing contention. This machine is an initial prototype for the VMP design since the choices of processor (68020 over a RISC-style processor), bus (VMEbus over a much higher-speed bus) and memory boards (commercial sequential-access VME memory over high performance boards) make a significant concession to budget and fast construction over ideal performance. The prototype **will** allow us to evaluate the expected performance of this design since, as pointed out by **Clark[8]**, trace-driven simulation is frequently a poor indication of real performance. However, since our prototype is not yet operational, we provide some expected performance figures based on: simulation, instruction counts for the key software cache management routines, and timings for hardware components.

In the VMP design, the performance of a processor is degraded by three factors:

- Cache Misses - some proportion of the resulting bus transactions are [illegible] retried when an ownership conflict arises on the data,

[8] This organization allows the kernel space to be part of each user virtual space.

Cache Page *Size* ***(bytes)***	*Replaced* Page State	Elapsed *The* (psecs)	*Bus* *The* (psecs)
128	not modified	17	3.5
256	not modified	20	6.6
512	not modified	26	13.0
128	modified	17	7.0
256	modified	23	13.2
512	modified	36	26.0

Table 1: Elapsed Time and Bus Time per Cache Miss

- Consistency Interrupts - both for cache data as well **as** page table updates,
- Bus Load - which affects the time for the above two operations.

In this discussion, we **first** estimate processor performance **as** a function of the cache miss ratio for different cache page sizes, assuming no consistency interrupts and no bus contention. We then use simulation results (cache miss ratios) to determine the ranges for the cache parameters which will give the desired processor performance. Finally, we calculate bus **uti**lization per processor as a function of the miss ratio to estimate the number of processors that one can **feasibly** configure without significant bus contention. Consistency interrupts introduce the same overheads as cache misses (and possibly increase the miss ratio by flushing cache entries). Thus, consistency overhead can be incorporated in these performance estimates by hypothesizing a higher miss ratio than that suggested by the simulations.

5.1 Cache Miss Time

The elapsed times for a cache miss (assuming no bus contention and no bus transaction abort) are given in Table 1. These times assume a 16 MHz 68020 running with 0 wait state access to cache memory plus a block copier and memory that perform block transfers in 300 nanoseconds for the first long word **(32-bits)** and 100 nanoseconds for each subsequent long word. These times were calculated by summing instruction execution times for the cache miss handler and the time to update the cache (one block transfer if the page to be replaced was not modified, two block transfers if it was modified). Block transfer time is overlapped with the CPU processing where possible.

Clearly, the software time associated with miss handling (about 15 **μsecs**) means that there is limited benefit using in a smaller cache page size. If we

Cache Page Size ***(bytes)***	***Elapsed Time*** ***(μsecs)***	***Bus Time*** ***(μsecs)***
128	17	4.4
256 512	21 29	8.3 16.

Table 2: Average Cache Miss Cost

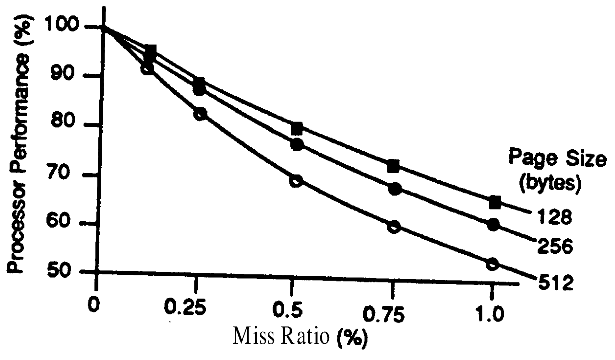

Figure 3: Processor Performance to Cache Miss Ratio

assume a mix of different cache miss scenarios with 75 percent of the replaced pages being unmodified then the average cache miss cost is given in Table 2.

Figure 3 plots the processor performance as a **function** of the miss ratio, assuming the average cache miss cost is incurred on each miss, with data for cache page sizes of 128, 256 and 512 bytes. The processor performance is normalized so that processor performance with no cache misses is **1**.[9]

Note that the miss ratio is a function of the cache page size so it is inappropriate to use this graph to compare the benefits of different cache page sizes.

Next we determine the characteristics of the cache that are required to achieve a sufficiently low miss rate, given the large cache page sizes, to realize reasonable processor performance.

5.2 Cache Miss Ratio and Processor Performance

The ranges of the variable hardware parameters of the VMP prototype (cache size from 64K to 256K bytes, page size of 128, 256 or 512 bytes) were established using four VAX 8200 traces obtained by the ATUM **technique[2]**. These traces include VMS op-

[9] *per jofmance* = (1 + ***MissRatio*** * ***AverageMissCost*** • ***Re fsPerInstr*** * ***InstrExecutionRate***)$^{-1}$ **From MacGregor[16]: RefsPerInstr=1.2** and ***InstrExecutionRate*** = ***(7clocks/instn*** * ***60nsecs/clock)***$^{-1}$ = 2.4 MIPS

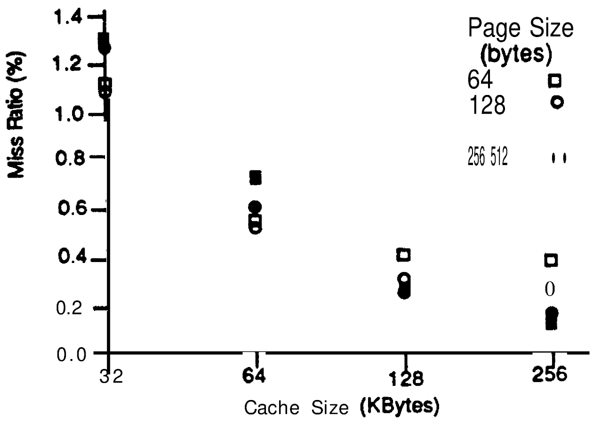

Figure 4: Cache Miss Ratio and Cache Size

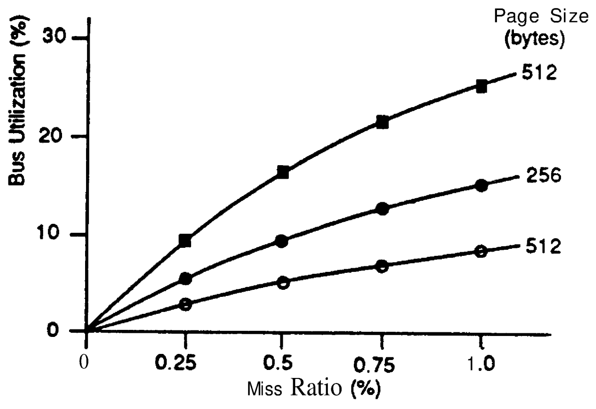

Figure 5: Bus Utilization to Cache Miss Ratio

erating system references and a small degree of multiprogramming. The trace lengths vary **from** 358,000 to 540,000 four-byte references.

The cold-start simulation results of a **4-way** set associative cache for various cache sizes and cache page sizes are summarized in Figure 4. These low miss ratios contrast with **most** cache measurements published to date. However, with the parameters of our cache, it is better compared to a 4-way set **associa**tive translation lookaside buffer with 512,256 or 128 sets of 4 entries (with 128, 256 or 512 byte cache page sizes) except that the cache also has the ass+**ciated** data. **Smith[19]** indicates that **.4%** miss ratio has been observed in **TLB's** with 128 sets of 2.

In these traces operating system references account for approximately 25% of the references and 50% of the misses, and the application programs employ no special locality enhancing techniques. We anticipate that application of appropriate software techniques could lead to even lower miss ratios.

Applying these results, for example, using a 256 byte cache page size and 128 kilobyte total cache size, one would expect a miss ratio of 0.24 giving processor performance of 87% according to Figure 3.

5.3 Bus Utilization and Number of Processors

Each cache miss results in bus traffic. Table 2 **pro**vides the bus cost for the "average" cache miss. Figure 5 shows bus utilization as a function of the cache miss ratio for the three cache page sizes, using this average bus cost per **miss.**[10] For example, for a 256 byte cache page size, with a miss ratio under 0.6%, the bus utilization by a single processor is under 10%. Using a simple single-server (the bus) multiple-client (several processors, ignoring DMA I/O devices) queuing model, and observing the request service times, we estimate that one can accommodate up to 5 processors on a **single** bus. Additional processors can be expected to degrade individual processor performance by increasing bus contention as well as possibly increasing the miss ratio because of consistency contention.

5.4 Consistency Overhead and System Software

The effect of consistency interrupts can be **incorpo**rated into the above figures by assuming a higher miss ratio. The rate of consistency interrupts and its effect on the cache miss ratio are unknown and highly dependent on the programming of the system and the cache page size. For instance, the straightforward use of test-and-set locks on the same cache pages as the data being modified could result in enormous consistency overhead. Thus, this design requires "good behavior" from the software it is executing to realize its performance, just as the performance of virtual memory systems is highly dependent on program behavior. We are developing software support at the operating system and programming system level that is tuned for the VMP design. In this vein, we are interested in exploring how far the software support can go to ensure good behavior, **as** opposed to how

[10] bus utilization is the bus **use** time **during** the execution of N instructions divided by the execution time (including miss handling) of JV instructions. $Util =$ $(MissRatio * BusTimePerMiss)/((InstrExecutionRate * RefsPerInstr)^{-1} + MissRatio * MissServiceElapsedTime)$

well the hardware can deal with bad behavior.

For operating system support, we are porting the V kernel[6] to VMP and adding kernel-supported locking and queuing primitives. These primitives can be implemented either using the notification facility offered by the bus monitor, or by using non-cached, globally-addressable physical memory. A process requesting the lock accesses the lock as part of a kernel operation and suspends for a timeout period if the lock is taken. As an additional optimization, the pro cessor can set the action table entry associated with this memory to 11 (notify) so it can wake up the process to retry when the lock is cleared. As another operating system support mechanism, we are planning to allow the application to specify whether an area of virtual memory is going to be shared or not. If not, a read cache miss to this area is handled by a read-private bus transaction, eliminating the need to later do an assert-ownership on the first write operation. Since the data is not shared, this should not conflict with other processors and may in fact serve to flush this data from the cache of another processor that was previously running this process. It is interesting to note that the bus monitor can also be used to implement interprocessor messages: the bus monitor would interrupt the processor when a message is written to the cache page corresponding to its mailbox. Other specialized uses are also possible, an . example being notification locks.

To realize the maximum performance offered by VMP, programming systems need to recognize the importance of clustering related data on cache pages and compiling code and data for high cache page utilization. These demands on software technology are significant but are also a common theme in previous efforts to redefine some of the hardware/software boundaries. We have been exploring a parallel programming paradigm which we call *work-form processing*[7] that draws analogy from the pro - cessing structure of the (human) office. Determining the quantitative effects of these programming techniques in the VMP prototype is a focus of future research.

6 Related Work

A central focus of our work has been to better understand the proper trade-off between hardware and software. Our design proposes operating system control of the caches with suitable hardware support to make this efficient. An alternative software control scheme proposed for the MIPS-X project[1] is to have the compiler generate cache control instructions to ensure consistency. This relies on using a language that includes explicit constructs for accessing shared data, such as the monitor construct[15], and all data sharing being properly controlled by these constructs. Except for instructions which selectively flush cache entries, this scheme requires no hardware support for consistency. However, the MIPS-X scheme must flush all shared data in anticipation of shared access whereas the VMP scheme only flushes on demand. It remains to be seen which is most expensive and how application-sensitive the behavior is.

The performance of cache memories for single processor machines has been studied extensively[12,19,18]. Much of this work studies much smaller cache page sizes, so the results have limited application. However, as mentioned previously, our expected performance is consistent with that expected and observed with TLB's of comparable size.

There has also been interest in cache consistency protocols for multiprocessor machines[11,12,18,10,3]. The cache consistency algorithm we describe is basically the ownership protocol used in the Synapse multiprocessor[11]. The alternative to an ownership protocol is to use a write-broadcast protocol, as used with the snoopy cache schemes.[10] With a write-broadcast protocol, the system bus acts as a sequencer, imposing a total ordering on memory up dates consistent with that observed by each processor. However, a write-broadcast scheme requires a data path from the bus to the cache that can up date the cache as required at near memory-reference speed. (Replicating or dual-porting the cache flags can reduce the contention at some cost in hardware.) It also requires a write-broadcast on every update of (potentially) shared memory at the level of the unit of indivisible memory update, typically a memory word or byte. This precludes the use of the large cache page sizes required for very low cache miss rates. Finally, it requires the cache either be physically addressed with a virtual-tophysical translation between the processor and cache or a physical-to-virtual address translation for use by the bus spy. (Note that the latter translation may be one-to-many unless virtual address aliasing is ruled out.) Thus, a write-broadcast approach requires a multi-master cache together with physical-tovirtual address translation and complex bus spy hardware, all operating at near memory-reference speed.

Most researchers have focused on the performance of different cache consistency protocols, looking only at the bus traffic levels. However, the consistency schemes providing the lowest bus traffic also tend to be the most complex and present a potential bottleneck between processor and cache memory, especially as processor and memory speeds increase. In contrast, we are interested in cache consistency schemes that are simple enough so there is minimal complexity in the processor-to-cache path and so a significant portion of the cache management can be performed in software.

7 Concluding Remarks

We have described the design of VMP, a shared-memory multiprocessor machine that uses software-controlled virtually addressed caches. We have argued that the basic approach of a virtually addressed cache with the processor being its single master provides the high memory bandwidth connection that will be required by processors of the future. Using this high-speed processor in combination with the local memory for cache management software and high-speed block data transfer hardware makes cache miss handling in software efficient. The software implementation provides a high degree of flexibility as well.

There are two major novel aspects of the design. First, an unusually large cache page size is used in combination with a large total cache size and a high-speed block data bus transfer facility, reducing the cache miss ratio so that software control of the caches is feasible. This eliminates the need for a considerable amount of specialized hardware, including memory management unit and cache miss handler. Instead, we simply provide per-processor local memory for the cache management code and data.

Second, the simple bus monitor in combination with software control solves the consistency problems associated with a virtually addressed cache. In particular, the scheme handles virtual address aliases or synonyms with no restrictions and virtual address translation consistency. It also allows DMA devices to be accommodated with no special consistency support.

The bus monitor state machine is a hardware resource provided to the processor for cache consistency. However, the generality of the mechanism suggests there may be other uses.

The challenge of the VMP design is in the software. Clearly, the cache management software itself must be highly optimized as well as correct. Moreover, VMP operating system software must provide means of synchronization between processes that does not induce the thrashing that one would expect with conventional test-and-set busy-wait loops on top of the VMP design. Finally, programming systems for the VMP design need to recognize the importance of clustering related data on cache pages and compiling code and data for high cache page utilization. These demands on software technology are significant but also a common theme in previous efforts to redefine some of the hardware/software boundaries.

Finally, there appears to be some issues in designing processors for a VMP-like design. First, the ideal VMP processor is as fast as memory technology allows. Faster processors reduce the speed advantage of implementing complex control logic in hardware. Second, the processor has minimal overhead for taking a bus error (or suitable cache miss signal) trap and returning from the trap, including making some registers available for the trap handler. Fortunately, many of the RISC-style processors appear to being going in these directions.

8 Acknowledgements

The Stanford Center for Integrated Systems and Philips Research made the collaboration among the authors possible. We are grateful to Tim Mann, Michael Stumm, Ross Finlayson and Helen Davis for their critique of the design and early versions of the paper, and to Naguine Navab for her work on the cac he cont rol software. We thank Anant Agarwal for providing the cache simulation results and Digital Equipment Corporation for supplying the instruction traces.

References

[1] A. Agarwal and M. Horowitz.
MIPS-X Internal and External Caches.
Technical Report, Computer Systems Laboratory, Stanford University, 1985.

[2] A. Agarwal, R.L. Sites, and M. Horowitz.
ATUM: A New Technique for Capturing Address Traces Using Microcode.
In *Proc. 13th Int. Symp. of Computer Architecture,* June 1986.

[3] J. Archibald and J.-L. Baer.
An Evaluation of Cache Coherence Solutions in Shared-Bus Multiprocessors.
Technical Report 85-10-05, Computer Science, U. of Washington, October 1985.

[4] C.G. Bell.
Multis: a new class of multiprocessor computers.
Science, 228:462-467, April 1985.

[5] David R. Cheriton and Michael Stumm.
The Multi-Satellite Star: Structuring Parallel Computations for a Workstation Cluster.
To appear in Distributed Computing.

[6] D.R. Cheriton.
The V kernel: A Software Base for Distributed Systems.
IEEE Software, 1(2), April 1984.

[7] D.R. Cheriton.
Workform Processing: a model and language for parallel computation.
Stanford University, Computer Science Technical Report, to appear 1986.

[8] D. Clark.
Cache Performance in the VAX-11/780.
ACM Trans. on Computer Systems, 1(1), Feb. 1983.

[9] H.M. Deitel.
Introduction to Operating Systems.
Addison-Wesley, 1983.

[10] R. Katz et al.
Implementing a Cache Consistency Protocol.
In *Proc. 12th Int. Symp. on Computer Architecture,* pages 276-283, ACM SIGARCH, June 1985.
also SIGARCH Newsletter, Volume 13, Issue 3, 1985.

[11] S. Frank.
Tightly-coupled Multiprocessor System Speeds Memory Access Times.
Electronics, 57(1), January 1984.

[12] J.R. Goodman.
Using Cache Memory to Reduce Processor-Memory Traffic.
In *Proc. Tenth International Symposium on Computer Architecture,* pages 124-131, June 1983.

[13] A. Gupta, C. Forgy, and R. Wedig.
Parallel Algorithms and Architectures for Rule-Based Sytems.
In *Proc. 13th Int. Symp. of Computer Architecture,* June 1986.

[14] W.D. Hillis.
The Connection Machine.
MIT Press, 1985.

[15] C.A.R. Hoare.
Monitors: An Operating System Structuring Concept.
CACM, 17(10):549-557, October 1974.

[16] D. MacGregor and J. Robinstein.
A Performance Analysis of MC68020-based Systems.
IEEE Micro, 5(6):50-70, December 1985.

[17] C.L. Seitz.
The Cosmic Cube.
CACM, 28(1):22-33, January 1985.

[18] A.J. Smith.
Cache Evaluation and the Impact of Workload Choice.
In *Proc. 12th Int. Symp. on Computer Architecture,* pages 64-73, ACM SIGARCH, June 1985..
also SIGARCH Newsletter, Volume 13, Issue 3, 1985.

[19] A.J. Smith.
Cache Memories.
Computing Surveys, 14(3), September 1982.