May 1989 Report No. STAN-CS-89-1266

Multi-Level Shared Caching Techniques for Scalability in VMP-MC

by

D. R. Cheriton, H. A. Goosen, and P. D. Boyle

Department of Computer Science

Stanford University

Stanford, California 94305

unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a REPORT SECURITY CLASSIFICATION	1 b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) STAN-CS-89-1266	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Computer Science Dept. — 6b OFFICE SYMBOL *(If applicable)*	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, *and ZIP Code)* Stanford University Stanford, CA 94305	7b ADDRESS (City, State, *and ZIP Code)*
8a. NAME OF FUNDING / SPONSORING ORGANIZATION DARPA — 8b OFFICE SYMBOL *(If applicable)*	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00014-88-K-0619
8c. ADDRESS (City, State, **and** *ZIP Code)* Arlington, VA	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO | PROJECT NO | TASK NO | WORK UNIT ACCESSION NO

11 TITLE *(Include Security Classification)*
Multi-Level Shared **Caching** Techniques for Scalability in VMP-MC*

12 PERSONAL AUTHOR(S)
David R. Cheriton, Hendrik A. Goosen and Patrick D. Boyle

13a TYPE OF REPORT	13b TIME COVERED FROM ____ TO ____	14 DATE OF REPORT *(Year, Month, Day)*	15 PAGE COUNT

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (*Continue* on reverse *if necessary and identify by block number)*
FIELD	GROUP	SUB-GROUP	

19 ABSTRACT *(Continue on reverse if necessary and identify by block number)*

The problem of building a scalable shared memory multiprocessor can be reduced to that of **building a scalable** memory hierarchy, assuming interprocessor communication is handled by the memory system. **In this paper, we describe** the **VMP-MC** design, **a** distributed **parallel multi-computer based on** the **VMP** multiprocessor design, that is intended to provide a set of **building** blocks for **configuring** machines from one to several thousand processors. **VMP-MC** uses **a memory** hierarchy based on shared caches, **ranging** from on-chip caches to board-level **caches** connected by **busses** to, **at** the bottom, **a** high-speed fiber optic ring. In **addition** to **describing the building block components of** this architecture, we identify the key performance **issues associated with the design and provide performance evaluation** of these issues **using trace-**drive **simulation** and measurements from the **VMP.**

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION
22a NAME OF RESPONSIBLE INDIVIDUAL	22b TELEPHONE *(Include Area Code)* | 22c OFFICE SYMBOL

DD Form 1473, JUN 86 *Previous editions are obsolete.* SECURITY CLASSIFICATION OF THIS PAGE

S/N 0102-LF-014-6603

Multi-Level Shared Caching Techniques for Scalability in VMP-MC*

David R. Cheriton, Hendrik A. Goosen and Patrick D. Boyle
Computer Science Department
Stanford University

Abstract

The problem of building a scalable shared memory multiprocessor can be reduced to that of building a scalable memory hierarchy, assuming interprocessor communication is handled by the memory system. In this paper, we describe the VMP-MC design, a distributed parallel multi-computer based on the VMP multiprocessor design, that is intended to provide a set of building blocks for configuring machines from one to several thousand processors. VMP-MC uses a memory hierarchy based on shared caches, ranging from on-chip caches to board-level caches connected by busses to, at the bottom, a high-speed fiber optic ring. In addition to describing the building block components of this architecture, we identify the key performance issues associated with the design and provide performance evaluation of these issues using trace-drive simulation and measurements from the VMP.

1 Introduction

Our goal is to develop a *building block* technology from which components made from workstation-class hardware can be composed into a spectrum of machines, ranging from single-processor personal computers to supercomputer configurations with thousands of processors. All configurations should run the same software and be incrementally upgradeable from the smallest to the largest configurations. The availability of high-performance low-cost microprocessors makes this feasible from the standpoint of raw processing power. The problem lies in the interconnection. To address this, we propose a scalable shared memory multiprocessor based on characteristics of the VMP architecture [8, 7], extended by using multi-level, shared caches.

In this paper we present the overall design of VMP-MC, a distributed parallel multi-computer, focusing on the design of the building block components and the novel techniques which support scalability. We also identify the key performance issues with this design and investigate them using trace-driven simulation and experience from the original VMP design. We argue that VMP-MC provides a credible approach to a highly scalable architecture.

Novel aspects of the design include: (1) limited sharing of secondary caches to reduce miss rates and cost; (2) a hierarchically structured, directory-based consistency mechanism; and (3) locking and message exchange explicitly supported by the memory hierarchy.

The next section describes the function and interconnection of the VMP-MC components. Section 3 investigates and evaluates the critical performance issues. Section 4 describes the current status of the VMP-MC hardware and software. Section 5 compares our work to other relevant projects. We close with a summary of our results, identification of the significant open issues, and our plans for the future.

*This work was sponsored in part by the Defense Advanced Research Projects Agency under Contract X00014-88-K-0619.

2 VMP-MC Design

The basic VMP-MC design is shown in Figure 1.

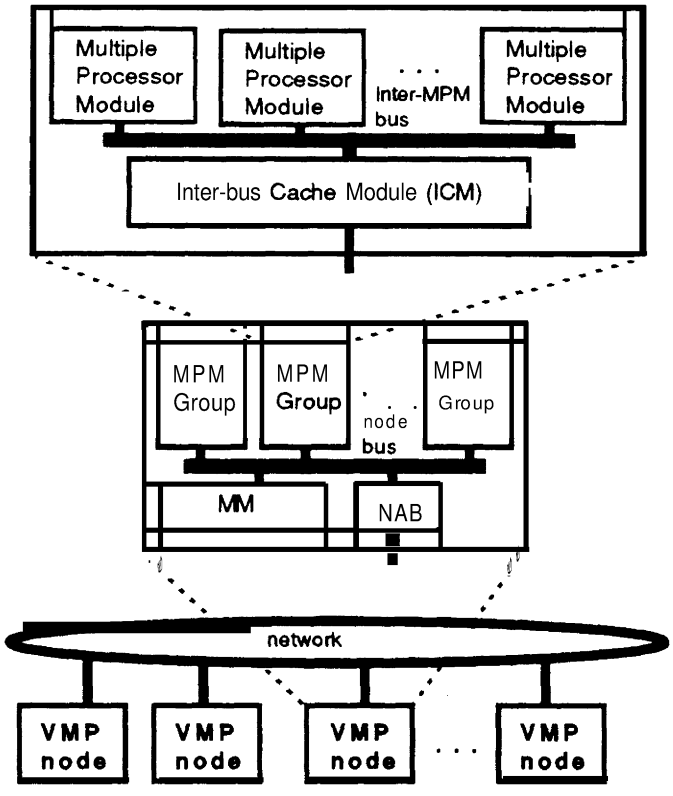

Figure 1: VMP-MC Overview

A VMP-MC configuration consists of one or more network nodes connected by a high-speed network. The V kernel and its virtual memory system manage the caching of data at each node and maintain consistency among nodes, relying on network file servers for non-volatile storage. The *Network Adapter Board* (NAB) provides high-performance communication between the Memory *Modules* (MMs) and the network. Consistency among *Multiple Processor Modules* (MPMs), *Inter-bus Caching Modules* (ICMs) and NABs on the node bus is ensured by the MM. An ICM connects a *Multiple Processor Module Group* (MPMG) to the node bus, providing caching and consistency within the MPMG. The MPM recursively provides the same caching and consistency for the multiple processors sharing the on-board cache.

The following sections describe these modules and their interaction in greater detail.

2.1 Memory Module (MM)

The memory module (MM) provides the bulk memory for the system, and is a physically-addressed slave module on the node bus. It includes a directory, the Memory *Module Directory* (MMD), that records the consistency state of each *cache block* (an aligned 128 byte unit of memory) that it stores. Rapid data exchanges with the MPMs are achieved by block transfers using a sequential access bus protocol and interleaved fast-page mode DRAMs.

For each 128 byte block of memory in the MM, the MMD has a 16-bit entry indicating the block's state:

cc	L	P_{12}	P_{11}	...	P_0

where CC is a two bit code, and L is the LOCK bit used for locking and message exchange (described below). Each P_i corresponds to one MPM or ICM, allowing up to 13 MPMs and ICMs to share

this memory board[1]. The meaning of the CC and P fields is summarized in Figure 2.

CC	Meaning if P_i set
00	undefined
01	MPMs/ICMs with a shared copy of block
10	MPM/ICM with private copy of block
11	MPMs/ICMs requesting notification

Figure 2: CC Bit Interpretation

If the P_i are all clear, then the block is neither cached nor in use for message exchange. Directory entries can be written and read directly, but they are normally modified as a side effect of bus operations. The MMD is designed to support the implementation of consistent cached shared memory, memory-based locking and a memory-based multicast message facility, as described below.

2.1.1 Consistent Shared Memory Mode

The consistency protocol follows the same *invalidation* protocol used in VMP, ensuring either a single writable *(private)* copy or multiple read-only *(shared)* copies of a block.

If the block is uncached, the P field of its MMD entry will contain zeros. A read-shared or read-private bus operation by module i on an uncached block returns the block of data. As a side-effect, P_i is set, and the CC bits are set to 01 (shared) or 10 (private). A read-shared operation on a shared block returns the data and sets P;. A read-private or assert-ownership operation by module i on a shared block changes the CC to 10 (private), interrupts all modules j for which P_j is set, clears all P_j, and sets P,. When a block is private, the MM aborts read-shared and read-private operations and interrupts the owner. A writeback operation by the owner i sets the CC to 01 (shared). Depending on the type of writeback, P_i is either reset or left unchanged.

Using this MMD entry format, the MM requesting a block of memory knows exactly which modules to interrupt, if any, to allow it to acquire a copy of the block in the desired mode. This attribute of the design is important to its scalability.

2.1.2 Memory-Based Locking

The unit of locking in VMP-MC is the cache block (128 bytes). A *lock bus* operation by module i on an unlocked block (the L bit in the MMD entry is clear) succeeds and sets the L bit and P;. Otherwise, the bus operation fails and P_i is set. (Variants of the read-shared and read-private bus operations include the locking action, and fail if the lock is already set.)

An *unlock bus* operation by module i clears the MMD entry's lock bit, and all modules j for which P_j is set, where $j \neq i$, are signalled that the lock has been released. This mechanism allows different processes to set and clear the lock, as is required in some applications. Variants of the write-back bus operation include the unlock action.

Read-shared and read-private operations without the lock action succeed independently of the lock setting and do not change the lock setting. This behavior allows the application process that sets a lock to migrate between processors.

The expected use of this facility is for the application to first attempt to lock a block corresponding to some shared data. Once the block is locked, the application updates the logically locked data structures and then releases the lock. Other waiting caches are notified of unlocking, relying on the P field for notification.

The provision of locking as part of the consistency mechanism provides several optimizations over a conventional lock mechanism using test-and-set operations and memory consistency. In our

[1] An MPM and an ICM appear identical to the MM on the node bus. We use MPM in the exposition for brevity.

scheme, a processor needing to acquire a lock is forced to wait until it is unlocked, rather than steal the block containing the lock away from the lock holder, as would occur in the original VMP architecture. Thus, the locking mechanism serves as *contention control* on data structures. Used in combination with the read operations that specify locking, this facility allows one to acquire both the lock and the data in one bus operation, but not until the lock is free. In contrast, the conventional approach may induce a high level of contention when, for example, processors spin on locks while the lock holder is updating data in the same cache block.

2.1.3 Memory-Based Message Exchange Protocol

The message exchange protocol uses blocks of shared memory as message buffers. A separate protocol is needed since the semantics of message exchange differs from that of consistent shared memory. A receiving processor wants to be notified after a block (message buffer) has been written, and not before it is read, as in consistent shared memory mode. A sending processor wants to be able to write a block without having read it.

A *Notify* bus operation (i.e., notify me when the block is written) by module i on a given block places the block in message exchange mode by setting the CC field in the corresponding MMD to 11, and setting P_i. A subsequent writeback to that block causes every module specified in the P field to be interrupted and the L bit to be set. The L bit indicates that the block has been written, but not yet read. A read-shared operation then causes the L bit to be cleared and returns the data.

One use of this facility is for interprocessor messages, as part of the operating system kernel implementation. A kernel operation on one processor that affects a process on another processor sends a message to that processor. Each processor has one or more message buffers for which it requests notification when they are written. One communicates with a processor by simply writing to one of its message buffers. For synchronization, the write is aborted if the L bit of the block is set (i.e., the block has been written and not subsequently read).

Another use is notification of memory mapping changes. A memory block is associated with each portion of the kernel memory mapping information (e.g., one MM cache block per address space). If an MPM is caching data from some virtual memory space, it requests notification of writes to the corresponding message block. When a kernel memory management operation modifies the virtual memory mapping, the changes are written to the associated message blocks. The affected modules are notified and update their caches and memory mapping information. Gap-free sequence numbers on the updates are used so a processor can detect that it missed an update (i.e., it failed to read the message block before the block was overwritten), without requiring the hardware to provide this level of synchronization. When a processor does miss an update, it invalidates all of the cache data associated with that portion of virtual memory.

This scheme builds upon the memory coherency mechanism to provide interprocessor interrupts and message data transfer, eliminating the need for a separate facility. It requires only two extra bits in each directory entry and one additional type of bus operation.

In contrast, interprocessor communication implemented purely in terms of message buffers in conventional shared memory would result in considerable extra cache and bus traffic for locking and coherency, imposing unnecessary overhead on key system resources, and limiting scalability.

2.2 Multiple Processor Module (MPM)

The Multiple Processor Module (MPM) occupies a single printed circuit board, and is shown in Figure 3. Multiple CPUs (microprocessors) are attached by an on-board bus to a large virtually addressed cache and a small amount of local memory. The cache lines are large, and the cache is managed under software control, as in VMP [8]. The local memory contains cache management code and data structures used by a processor incurring an on-board cache miss. A FIFO buffer queues requests from the node-bus for actions required to maintain cache consistency, and to support the

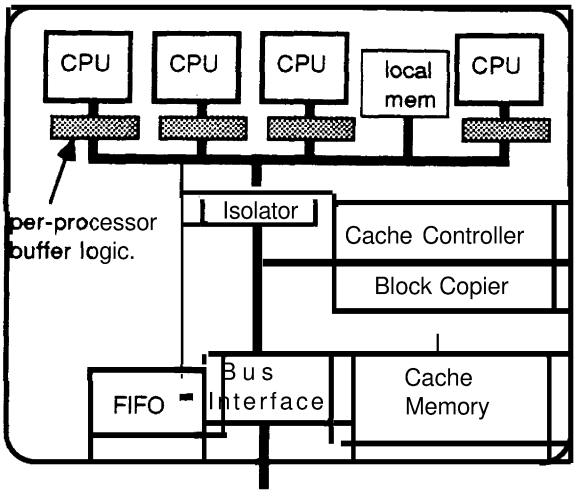

Figure 3: MPM Board Layout

locking and message exchange protocols. One of the processors is interrupted to handle each such request as it arrives.

Each CPU is a high-speed RISC processor with a large (16K or more) virtually addressed on-chip cache with a moderate cache line size (32 bytes). Interference between processors is reduced by transferring data (in 2 cycles) from the on-board cache to a wide per-processor holding register [2], which then transfers the line to the on-chip cache in burst-mode. With each on-chip cache line, in addition to the *usual flags* such *as valid, modified* and ***writable***, we require *locked, held* and ***requested***[3]. Encodings of the extra flags are summarized in Figure 4.

LHR	meaning
000	on-chip cache does not hold the lock
001	on-chip cache has requested lock from on-board cache
110	on-chip cache holds the lock and it is locked
010	on-chip cache holds the lock but it is unlocked
111	on-chip cache holds the lock, it is locked, and the on-board cache has requested the lock

Figure 4: LHR Flags Encoding

The processor has a *lock* and an *unlock* instruction. The lock instruction specifies an address aligned to a cache block. If the lock is held and not requested (LHR=110 or 010), lock and unlock instructions execute locally (i.e., lock acquisition is done entirely in the cache, and locking has low latency if the lock is held and unlocked). *If'* the *requested* flag is set for a held lock, the lock instruction returns a failure indication. If the lock is not held, the lock instruction causes the request of the lock from the on-board cache (like a cache miss), which either returns the lock (110), indicates the lock should be marked as requested (001) or causes the processor to handle an on-board cache miss, as described below. The unlock instruction simply clears the lock flag unless the requested flag is set, in which case it releases the lock to the on-board cache and clears the held flag[4]. Finally, the on-board cache can signal the processor to writeback and invalidate a specific

[2] This is an aggressive requirement. Slower transfers would degrade the MPM performance through increased interference between processors, and further study is required to evaluate the cost/performance tradeoff.

[3] We also require a *privileged* tag bit so that kernel and user data can reside in the cache together. This eliminates the need to flush the cache on return from a kernel call.

[4] A cache line may be removed from the cache even if the lock flag has been set. An unlock instruction then incurs a cache miss, which causes the lock bit to be cleared at the memory module level, or some cache level in between.

cache line, that a lock on a cache line has been granted, or that a particular lock has been requested.

The on-board cache implements the same consistency, locking and message exchange protocols as the MM. The cache flag entry per cache line is the same as that of the MM except that it includes 4 additional control bits (replacing 4 P bits). An *exclusively* held bit indicates whether or not the cache holds exclusive ownership of the block. This allows a block to be shared by processors within the MPM, while it is exclusively owned by the MPM relative to the rest of the system. A dirty bit indicates whether the entry has been modified since last being written to its MM. Finally, there are the *requested* and *held* bits associated with the locking. The *held* bit allows the cache to hold the lock even if no processor in the MPM has the lock set. The *requested* bit indicates that the lock should be released to the lower level when it is released, rather than just held within the on-board cache (in anticipation of a processor in the MPM requesting the lock).

Upon on-board cache miss, the faulting processor behaves like a VMP processor. It traps to a software miss-handling routine, determines the physical address of the missing data and a cache slot to use (writing out the data if modified), initiates a block transfer of the data into the cache slot by the cache controller, and resumes execution when the block transfer completes. The cache software is synchronized to allow multiple processors to incur cache misses at the same time. Cache access from other processors may also proceed concurrently with miss handling except for when actual bus transfers are taking place.

The block transfer can fail if the block is not available immediately, either because it is not up-to-date in memory, it is not cached in the local ICM, or it is locked and a lock bus operation was invoked. In the first two cases, the cache management software retries the transfer (perhaps after a short delay to allow writebacks and the ICM to acquire the data) until it succeeds, up to some maximum number of retries. The memory system takes the necessary actions to make the requested block available. In the lock case, the processor marks the block as requested in the on-chip cache, signals to the lock instruction that the instruction failed to acquire the lock and resumes execution. If the processor spins on the lock, the instruction is handled entirely by the on-chip cache until the on-board cache notifies the processor that the lock has been released.

The design of the MPM has several significant advantages. First, it recognizes and exploits the trend of the increasing sizes of on-chip caches on microprocessors. The large line size of the on-board cache is compatible with increasing on-chip line sizes. The inclusion of the locked, requested and held cache flag bits in both the on-chip and on-board caches effectively improves the cache and bus behavior by reducing latency, coherence interference, and contention. The bits impose a modest space overhead which decreases with increasing cache line size. The virtually addressed on-board cache eliminates the need for memory management on chip, thereby freeing chip area for a larger cache. Absence of mapping on chip also simplifies the invalidation of on-chip cache lines. The value of large cache blocks has been demonstrated by the VMP design.

Sharing the on-board cache has three major advantages. First, it results in a higher on-board cache hit ratio due to the sharing of code and data in the on-board cache and by localizing access to some shared data to the on-board cache. Compared to per-processor on-board caches, the sharing reduces the total bus traffic imposed by the processors. The reduction in bus traffic contributes to scalability, and hence performance[5]. Second, sharing the on-board cache reduces the total hardware cost for supporting N processors, since only N/K MPM boards (and on-board caches) are required if K processors share each on-board cache[6]. Finally, the increased hit ratio of the on-board cache reduces the average memory access time of the processor, resulting in a higher instruction execution rate. However, this effect is relatively small since the on-chip cache will typically have a high hit ratio, limiting the possible improvement in the memory access time.

[5] For example, if the bus traffic is decreased by 50%, the number of processors on the bus may be doubled. For an application with linear speedup, this will result in a doubling of performance.

[6] Because sharing on-board cache significantly reduces the parts count and the number of connectors and thus presumably improves the reliability, the sharing also contributes to scaling through improved reliability.

The on-board cache exploits a number of ideas of the original VMP processor cache. First. the cache is virtually addressed so there is a direct connection between the on-chip cache and the on-board cache, i.e., no MMU. Thus, miss handling is fast and the complexity of virtual-to-physical mapping is placed (in software) between the MPM and the inter-MPM bus, simplifying both the processor chip and the on-board logic, and reducing the translation frequency. For example, with an on-chip TLB one expects 0.004 TLB faults per memory reference [15] whereas we have measured 0.00004 translation misses [7] using the VMP cache, an improvement of a factor of 100. Also, the cache miss software uses compact data structures to replace conventional page tables, thereby reducing the memory space overhead of virtual memory implementations.

Second, the on-board cache minimizes replacements and flushing by using set-associative mapping and an address space identifier as part of the virtually addressed cache mechanism. Thus, the cache can hold data from multiple address spaces and need not be flushed on context switch. The on-board cache provides one address space identifier register per processor. Each off-chip reference by a processor (cache miss) is presented to the on-board cache prepended with the address space identifier. Thus, the on-board cache knows about separate address spaces but the processor chip need not.

Third, the large cache block size makes it feasible for the on-board cache to be quite large (i.e., .5 megabytes or more), reducing the replacement interference and thereby permitting multiple processors to share the on-board cache even when running programs in separate address spaces.

With 8 processors per MPM, it is possible to configure up to 104 processors on a single bus as 13 MPMs and one or more MMs. To scale larger, we introduce extra levels of caching and busses using the ICM and the NAB.

2.3 Inter-bus Cache Module (ICM)

The inter-bus cache module (ICM) is a cache, shared by the MPMs on an inter-MPM bus (an MPM group or MPMG), which connects such an MPMG to a next level bus. It appears as an MPM on the node bus and an MM on the inter-MPM bus. It caches memory blocks from the MMs, implementing the same consistency, locking and message exchange protocols as the MPMs. These blocks are cached in response to requests from MPMs on its inter-MPM bus. The MMD entry per block in the ICM is the same as that of the MPM, limiting the P field to 9 bits [7].

When an ICM receives a read transfer request for a block[8], it determines whether it has the block cached. If so, it responds in the same manner as an MM to the request. However, if the operation is a read-private request, it may have to gain exclusive ownership of the block on the node bus before responding. If the ICM does not contain the referenced block, it aborts the transfer and then attempts to acquire the block from the MM on the node bus, in the same way an MPM would.

To accommodate device access and uncached references, the ICM also provides direct uncached references to the node bus. In particular, an MPM can write a block directly through to the node bus, allowing it, for example, to transfer data to the NAB control register.

The ICM supports the message exchange facility by implementing the same states for its cached entries as the MPM cache. In addition, the exclusive flag is used to indicate when the message receivers are entirely local to the MPMG, automatically allowing the message activity to be localized to the group when appropriate.

Several merits of the ICM are of note. First, as a shared cache, the ICM makes commonly shared blocks, such as operating system code and application code available to an MPMG without repeated access across the node bus. This contrasts with the *cluster* controller approach described by Wilson [18][9], where repeated reads by MPM's in one group would result in repeated read requests to another

[7] The restriction of the entry to 16 bits is primarily to minimize the chip count for the board.

[8] The ICM has switches to indicate the range of physical addresses it should cover.

[9] Wilson also mentions the caching approach as used by the ICM.

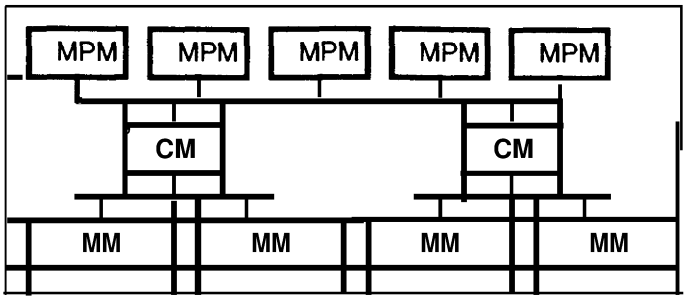

Figure 5: Partitioned Memory Hierarchy

group if the block is not in a memory module local to the requesting group. The ICM shared cache is important for scalability, for the same reasons identified for the MPM on-board cache. Second, the ICM supports the hierarchical directory-based consistency, providing a complete record of cache page residency, thereby minimizing consistency bus traffic and interprocessor interrupt overhead. Finally, because the ICM appears the same as an MPM, one can mix MPM's and ICM's on the node bus without change to the MMs.

A maximal configuration of 8-processor MPMs, ICMs and MMs would produce a 936-processor machine. Even larger configurations can be achieved using multiple levels of ICMs and busses. The address range switches on the ICM allow the memory load to be split below the MPM bus level between two or more separate ICMs and separate node-level busses and MMs, as illustrated in Figure 5. However, we see a more common configuration being a group of more modestly configured machines, connected by a high-speed network using the NAB.

2.4 Network Adapter Board (NAB)

The NAB [11] provides reliable transport-level communication between network nodes connected by a high-performance network. Thus, the NAB performs all packetizing, checksumming and encryption required as part of transport-level transmission, and the reverse on reception. Several aspects of the NAB are specifically relevant to multiprocessors. First, on-board processing and "intelligent" DMA provided by the NAB imposes the minimal load on the node bus and MPMs by performing a single block transfer to memory on reception and from memory on transmission. Data is delivered page-aligned with headers removed, allowing the data to be mapped directly to application memory without copying. Second, because the NAB performs the protocol processing, the MPM caches are not polluted by packetizing and checksumming data to be transmitted or received. It also reduces the network-related interrupt activity at the MPMs because the NAB handles multi-packet segments on transmission and reception. Finally, the NAB transfers to and from physical memory using the same bus operations used by the MPMs and ICMs so these block transfers can be aborted by an MM to cause an ICM or MPM to writeback exclusively-owned blocks. This approach avoids the cost of brute-force techniques to ensure that none of the data being read or overwritten is cached, as would otherwise be required.

A NAB-style network interface is also required for pure performance reasons, now that networks are available in the gigabit range. The serial, pipelined nature of protocol processing is not well-suited to the multi-level cache architecture supporting the general-purpose processors in VMP-MC.

The VMP-MC building blocks described above allow a large parallel machine to be configured. However, the feasible scale of configuration depends significantly on the actual speeds of the busses and memories and the program performance characteristics. The following section provides an initial evaluation of the design with a focus on identifying realistic parameters for this design using hardware technology we see available in the foreseeable future.

3 Design Evaluation

This section describes the results of several studies undertaken to provide a preliminary evaluation of key aspects of the design and aid in choosing certain design parameters. An important assumption is that on-chip processor caches can reduce on-board cache misses to the extent that the performance benefits of sharing the on-board cache, in terms of reduced bus traffic and reduced memory access time, overwhelms the interference cost of multiple processors sharing the cache. We evaluated this approach using trace-driven simulation.

3.1 Primary/Secondary Cache Parameters

In the simulations, each processor chip is assumed to have a virtually-addressed 16 kilobyte unified cache with a cache block size of 32 bytes[10]. Caches of this size will be feasible on microprocessors in the near future.

The on-board cache is a 4-way set associative virtually addressed cache of .5 megabytes using a 128 byte cache block size, the same as previous VMP on-board caches [7]. Upon a hit to the on-board cache, the data is transferred to a 16 byte wide by 2 deep per-processor FIFO in 2 processor cycles[11] . The data is then transferred to the processor in 8 cycles. Similarly, a FIFO (16 bytes wide, 8 deep) is used to reduce the cache busy time on a read and writeback on the inter-MPM bus. An on-board cache block (128 bytes) is moved over the 64-bit wide inter-MPM bus into this FIFO in 16 cycles (250 Mbytes/sec if the cycle time is 30 ns). Using this approach, we can write 16 bytes in parallel from the FIFO into the on-board cache, and fill the cache in 8 cycles.

On a miss in the on-board cache, the cache is busy for 1 cycle signalling the miss and then another 8 cycles transferring data from the latch. During the software cache miss handling by the faulting processor, the cache is busy only during the bus transfer, not during the entire processing of the miss. The cache is also made busy by invalidations and writebacks that occur as part of consistency interrupts. A slot invalidation makes the cache busy for 2 cycles (invalidation time plus arbitration time). A writeback makes the cache busy for 8 cycles. The on-board cache signals the affected processors to write-back or invalidate blocks as required by the ownership and locking protocol that we use.

3.2 Cache Behavior

In this section we examine the tradeoff between the benefits of sharing the on-board cache (decreased traffic on the inter-MPM bus), and the interference introduced by having more than one processor share the on-board cache. We will refer to an on-chip cache as an *L1 cache,* and to an on-board cache as an *L2 cache.*

Simulations were run using several multiprocessor traces. The traces were collected using a combined hardware/software method, using the VAX T-bit mechanism to single-step the processor through each process in round-robin fashion. The traces do not include operating system references, and all the traces are of 16-processor parallel executions. The characteristics of the following traces are summarized in Table 1 [17]:

Locusroute: This is a global router for VLSI standard cells. Each processor removes a wire from the task queue and selects the best route for that wire. No locks are used in the cost data structure.

Mp3d: This is a three-dimensional particle simulator for rarefied flow. During each time step, the particles are moved one at a time. One lock protects an index into the global particle array.

[10] The actual processor chip will probably have split instruction and data caches to increase the available bandwidth.

[11] In this discussion, time is expressed in terms of processor cycles, which will be 20-30 ns for the processors we consider.

Distributed Csim: This is a distributed logic simulator which does not rely on a global time during simulation. The trace does not include references to locks.

Name	references in trace ($\times 10^6$)	i-fetches (%)	reads (%)	writes (%)
mp3d	7.05	61	33	6
dcsim	7.09	50	39	11
locusroute	7.70	50	38	12

Table 1: Trace characteristics

The 16-processor traces were run against different MPM configurations, obtained by varying the number of processors sharing the L2 cache. The L1 and L2 cache sizes were the same for all the simulations. We compensate for start-up effects by keeping track of the blocks that a cache has accessed, and ignoring the first access to a block when calculating miss ratios and bus traffic. This approximates the stationary behavior of a cache.

Table 2 shows the L1 miss ratio for different numbers of processors sharing each L2 cache. The miss ratios for *Zocusroute* are comparable to those reported in [16] for a similar size cache, considering that we compensate for start-up effects. The higher miss ratios for the other applications reflect a higher degree of coherence activity. Significantly, the L1 miss ratios stay almost constant as we increase the degree of sharing. This means that we can optimize the degree of sharing without impacting the L1 cache performance.

Name	Processors per MPM				
	1	2	4	6	8
mp3d	7.6	7.6	7.6	7.6	7.6
dcsim	2.5	2.5	2.5	2.6	2.5
locusroute	.80	.80	.79	.82	.79

Table 2: L1 miss ratio (% of references)

Table 3 shows the decrease in the L2 miss ratio as we increase the number of processors sharing an L2 cache from 1 to 8. The improvements are 55% for *dcsim*, 57% for ***mp3d***, and 61% for *locusroute*. The lower miss ratios imply a reduction in the average memory access time. For the system we described, this improvement in L2 miss ratio will double the instruction execution sate *for* ***mp3d*** and *dcsim*, but result only in a 3% increase for *locusroute*. This is because the high L1 hit ratio measured for *locusroute* makes it difficult to tither decrease the average memory access time.

Name	Processors per MPM				
	1	21	41	6	8
mp3d	77	67	54	43	33
dcsim	20	17	14	11	9.3
locusroute	3.3	3.1	2.2	1.7	1.3

Table 3: L2 miss ratio (% of L2 references)

Table 4 shows how the number of L2 cache coherence actions per processor decreases as we increase the amount of sharing. The coherence actions consist of block invalidations, changes in ownership mode from private to shared, and writeback transactions if an invalidated or downgraded block was dirty.

The simulations show a decrease in the number of coherence actions of 50% for ***mp3d***, 65% for *dcsim*, and 67% for *Zocusroute* as we move from no sharing to 8 processors sharing an L2 cache.

This supports our claim that L2 cache sharing reduces coherence activity. The shared L2 cache allows fewer invalidations to propagate beyond the MPM.

Name	Processors per MPM				
	1	2	4	6	8
mp3d	7.8	7.1	5.9	4.9	3.9
dcsim	.88	.72	.54	.43	.31
locusroute	.06	.05	.04	.03	.02

Table 4: Number of coherence actions (% of processor references)

The decrease in the L2 miss ratio (shown in Table 3) should directly result in sharply lower traffic on the inter-MPM bus. This is supported by Table 5, which shows how the number of block move transactions (read and writeback) on the inter-MPM bus change as we increase the sharing. The seduction in block move traffic is 44% for *mp3d*, 46% for *dcsim,* and 59% for *Zocusroute.* The block move transactions constitute more than 90% of the traffic on the inter-MPM bus. This reduction in traffic on the inter-MPM bus means that we can put roughly twice as many processors on the inter-MPM busses when sharing the L2 caches by 8 processors, compared to the case where we do not share the L2 caches. This enables us to double the performance of an MPM group, while seducing the cost of the system at the same time.

Name	Processors per MPM				
	1	2	4	6	8
mp3d	7.2	6.9	5.9	5.3	4.0
dcsim	.81	.75	.60	.61	.44
locusroute	.005	.005	.004	.004	.002

Table 5: Block move transactions (% of references in trace)

3.3 Loading of shared resources

The on-board cache and the on-board bus are the two bottlenecks in the MPM. The utilization of these resources limit the number of processors that can share the L2 cache: if they are too busy, a processor may have to wait when handling an L1 cache miss. In the following evaluation, the utilization is approximated by counting the total number of processor cycles that the resource is occupied, and dividing that by the number of cycles that the processors will take to execute the trace.

Table 6 shows the utilization of the on-board bus. We see that the utilization starts out low and increases linearly as the number of processors is increased. *Mp3d* shows a slight superlinearity due to the increased on-board bus traffic caused by the coherence traffic confined to the L2 cache. For all three traces, the on-board bus utilization is fairly low up to 8 processors sharing the L2 cache. This suggests that the on-board bus will probably not be a bottleneck in the system.

Name	Processors per MPM				
	1	21	41	6	8
mp3d	2.8	5.6	13	22	34
dcsim	2.6	5.5	11	18	25
locusroute	1.1	2.2	4.3	6.8	8.5

Table 6: On-board bus utilization (% of available cycles)

Next we look at the on-board cache utilization, shown in Table 7. The cache is occupied by the following: cache hits (2 cycles), reads from the inter-MPM bus (8 cycles), writebacks (8 cycles), and invalidations (2 cycles), as explained earlier. Requests to the cache are handled on a FCFS basis. The cache management software is not a bottleneck since it is executed in parallel by the on-board processors. We assume that contention for the cache data structures can be minimized by fine-grain locking. The cache utilization is reasonably low for all the traces up to four processors sharing the L2 cache. After that, the cache is very busy for both *dcsim* and ***mp3d***.

Name	Processors per MPM				
	1	2	4	6	8
mp3d	9.3	19	40	59	78
dcsim	6.9	14	27	41	51
locusroute	2.2	4.3	8.3	12	16

Table 7: On-board cache utilization (% of available cycles)

A first order estimate of the average length of the request queues at the cache can be obtained by approximating the cache as an M/M/1 queueing system [12]. For the organization outlined above, and using the measurements of utilization given in Table 7, this yields average queue lengths of *0.2* for *Zocusroute* (with 8 processors per MPM). For the other two applications it seems that 4 processors per MPM would be more appropriate. This organization yields queue lengths of 0.4 for *dcsim,* and 0.7 for ***mp3d***.

From these results we make the following conclusions:

1. The **traffic** on the inter-MPM bus is sharply seduced when the L2 cache is shared by 8 processors, each with its own **L1** cache. We observe a 50% seduction in inter-MPM bus **traffic** when we share an L2 cache among 8 processors. We speculate that it may be possible to reduce the **traffic** even further by software techniques which attempt to localize interprocess communication to an MPM.

2. The hardware cost of the system decreases significantly while increasing the scalability, and therefore also the performance of the system.

3. The instruction execution sate of a single processor increases because of the decrease in the L2 cache miss ratio. This effect is more pronounced when the **L1** cache hit ratio is low.

4. The figures show that, for ***locusroute***, 8 is seasonable number of processors to share an on-board cache, given the constraints on board seal estate and the interference level introduced by higher degrees of sharing. Programs with poorer cache behavior *(dcsim* and ***mp3d***) *will* not perform well if more than 4 processors share an L2 cache.

The traces deal only with running one single address space parallel program. If the processor runs different applications in separate address spaces, replacement interference is not a problem because the on-board cache is large and set associative. We conjecture that separate applications will run with a higher miss ratio primarily because of lack of miss sharing, rather than replacement int erfesence.

3.4 Inter-MPM Bus Loading

On the inter-MPM bus, each MPM used approximately 3% of the available bus bandwidth with our preferred configuration of 8 processors per board, executing *Zocusroute.* Thus, it may well be feasible to configure up to 16 or more **MPMs** per bus, yielding an **128-processor configuration**. However, it is optimistic to extrapolate our results to larger processor configurations. Further evaluation

requires either traces for larger-scale parallel applications, or the realization of VMP-MC on that scale.

The use of an ICM and another level of bus allows an even larger configuration, potentially up to 1000 processors or more. Given our lack of data on this scale of system, we limit ourselves to a few comments. First, the ICM allows one to (largely) isolate a computation node as part of an extended workstation. It will share the MM, network adapter, and possibly local disks with the workstation, but with only slightly greater loading than a single additional processor. For example, an engineer might add such an expansion cabinet to his multiprocessor workstation, allowing him to run compute-intensive simulations on the ICM-connected module while running normal CAD software on the rest of the machine.

Second, if one can partition the application sufficiently well, these very large configurations of VMP-MC would work well. This partitioning problems seems easier than that imposed by distributed memory systems, such as the Cosmic Cube [13], since it is only an optimization. *Most* of the references should be to data that is locally cached, although this is not required for correctness.

3.5 Hierarchical Latency

We estimate that it will cost the MPM 20 cycles to access a 128-byte block from the ICM. The extra delay for accessing a block MPM-to-MM in this design (going through an ICM) is estimated as another 20 cycles. This is assuming a copy into the ICM cache while passing it through to the inter-MPM bus, with no consistency OS bus contention at the MM OS inter-MPM bus level. Using measured cache miss ratios of less than 0.05 percent *(Zocusroute),* the extra delay is about 1% of the cycle time per memory reference on average. Thus, the extra delay is not significant in the absence of contention.

With consistency contention, the faulting MPM must force a write-back in another MPMG. This cost is estimated as an extra 65 cycles. Again, with the low expected frequency of these events, the incremental cost on the average memory reference time is not significant.

The limited size of the ICM memory (compared to the total number of MMs) makes it feasible to provide faster memory in the ICM than in the MMs. Thus, with a good ICM hit ratio, the lower delay for ICM hits should compensate for the higher cost of the ICM misses. (This point was also made by Wilson [18].)

Latency for page faults and contention with other networks nodes is significantly higher than for MPMs within a single network node. For example, with a 100 Mb network and NAB, we expect roughly 1.1 milliseconds for a 1 kilobyte page fault from a file serves without contention. With file server contention, we expect the page fault time to be approximately 2.2 milliseconds in the absence of packet loss.

Investigation is required to understand the trade-offs between the "height" and "width" of the memory hierarchy. In particular, placing more MPMs on the same bus seduces the latency of interaction between these MPMs as compared to placing them on separate busses and possibly separate VMP-MC nodes. However, placing them on a common MPM bus imposes more load on this bus. In essence, this says that sharing MPMs should be on the same MPM bus or at least the same node, whereas non-sharing ones should be separated at the highest levels of the hierarchy.

3.6 Locking Performance Effects

To directly evaluate the benefits of the VMP-MC locking mechanism would require designing applications specifically for this architecture. While we plan to do this eventually, we approximate the behavior by identifying memory locations used for locking, and ignoring these references in the simulation.

Previously, we reported a 40% reduction in bus traffic when locks were ignored in a trace [7]. For the traces used here, *only* one (*mp3d*) contains access to locks. Although only 3.4% of the

accesses in *mp3d* is to the lock, we observe substantial reductions in bus traffic when lock access is ignored. There is a reduction of 20% in cycles on the inter-MPM bus, a reduction of 21% in cycles on the on-board bus, and an increase of 18% in the L1 cache hit ratio. This substantial seduction in the traffic supports the notion of a specialized locking mechanism that will reduce memory contention for locks.

3.7 Message Exchange and Mapping Performance

A message send takes roughly 50 cycles, including the cost of a *Notify* and a *Writeback*. Message reading time varies depending on the processor activity at the time of the message write. However, if these is no miss or consistency handling active at the time the message is sent, the processor receives the message in the time required to interrupt, transfer the block and continue, roughly 100 cycles.

If the action occurs between processors in the same MPM, no bus action is generated. If the action is local to an MPMG, the ICM ensures that it does not result in traffic on the node bus.

The primary use at present for the inter-processor communication is to allow efficient notification of processors when aspects of the memory mapping is changed, affecting the implicit mapping represented in the caches. We draw on measurements done of Accent [9] to argue the acceptability of this mechanism.

Measurements of Accent indicate a rather low level of mapping changes. Although no memory reference counts were given, we estimate these to be roughly 2600 million references in the measurements (assuming an average instruction time of 3 microseconds for the Pesq). Using these measurements as a sough guide, these was approximately 1 memory mapping change per 3 million memory references. Thus, remapping imposes an overhead of .003 percent on each processor, assuming one processor performs the remapping and the rest are interrupted. This figure does not incorporate the cost of additional misses resulting from the remapping. Note that VMP-MC normally remaps the memory when copy-on-write is performed, rather than simply invalidating the cache entry. This technique reduces the number of cache misses resulting from mapping changes.

4 Status

The VMP-MC represents (and requires) the cumulation and focus of several projects with the V software and VMP hardware. It would be very costly (in terms of time and money) to build a full-scale VMP-MC configuration, so we are progressing incrementally in the development, evaluation and construction of hardware.

The MM design and layout is complete. The transfer speed in the prototype (using the VME bus) is approximately 40 megabytes per second. (Our board utilizes a two-edge handshake protocol, not the VME standard block transfer protocol.) We expect to have working boards in mid-1989. We plan to use existing VMP processor boards initially, since it will require only minor modifications to work with the MM. The MPM is still in design as we evaluate the possible choices of microprocessor. The ICM, combining the logic of the MM and MPM, is still at the initial design stage.

A NAB prototype (wire-wrap) board has been completed and we are now doing a PC board version for FDDI. To get a prototype VMP-MC working quickly and build on our prior work, we are using the VMEbus as the bus. However, future wide bus standards with more support for block transfers will clearly be a better long-term choice.

The V distributed system has been posted and runs on the VMP. It is planned to be the operating system for VMP-MC. V supports light-weight processes, symmetric multiprocessing, distributed shared memory and high-performance interprocess communication. We are currently reworking the V kernel to provide cleaner and faster parallel execution within the kernel. In related work on distributed operating systems, we have been investigating a distributed virtual memory

system [6] that provides memory consistency of virtual memory segments shared across a cluster of workstations.

5 Related Work

Most work on scalable architectures to date has resulted in machines that do not support shared memory OS that require a high initial investment, machines with limited general computation flexibility, and machines with large numbers of relatively slow or limited processors. For example, the Connection Machine [10] provides a large number of processors of limited power and is unable to sun a conventional operating system. Similarly, the Cosmic Cube [13] does not run a general-purpose operating system and thus is not usable as a workstation OS general-purpose computing node. From our experience, we view the shared memory multiprocessor as the most desireable form of general-purpose machine.

The extension of the VMP design to a hierarchically structured memory system is similar to the design described by Wilson [18] with the ICM corresponding to *his cluster cache.* However, we have provided a detailed design for handling coherency and caching that was lacking in his description. Also, we focus on using a cache module to interconnect busses rather than a simple bus interconnect *(routing switch* in his terminology). All the VMP-MC memory is attached to the lowest level bus, the node bus, rather than distributed across the clusters, OS bus groups. We believe that the ICM caching eliminates the extra bus traffic one might otherwise expect from locating all the memory on the node bus and in fact leads to a lower level of traffic on non-local MPM busses.

In general, we believe that the caching approach to bus interconnect is superior to using *routing switches* and distributing the physical memory among the MPMGs (as suggested by Wilson). First, the caching approach avoids the need to optimize the allocation of physical memory relative to processors on a bus. Memory effectively migrates to an MPMG based on demand. Thus, the system must concern itself only with locating interacting processes within the same MPMG. Allocating physical memory for these processes from within their MPMG is not required. Second, it avoids multiple transfers by the MPMG to move a given data block into several MPMs. Third, the ICM knowledge of data blocks in its MPMG allows it to selectively filter out irrelevant invalidation operations from the bus.

We argue that sharing the on-board cache is necessary given the low hit ratio, and the resulting low hardware utilization, also predicted by other studies [14].

Merits of software control and additional performance evaluation for VMP have been described elsewhere [7]. In summary, the three major changes to the MPM from the original VMP design axe:

- Multiple processors share the on-board cache, rather than a single processor, assuming sizeable on-chip caches.
- The bus monitor and action table of VMP have been replaced by the MM directories (and the equivalent on the ICMs). The elimination of the action table makes the MPM configuration independent of the amount of physical memory in the system.
- Support for locking and message exchange has been added.

These changes do not detract from the relative simplicity of the VMP design.

The memory-directory based consistency scheme has been described and studied in various forms by a number of researchers [4, 1, 2]. The use of large cache line size in VMP-MC makes it feasible to store a processor bitmask per directory entry while keeping the space overhead around 2 percent. (This corresponds roughly to the *DirallB* scheme of Agarwal et al. [1].) The hierarchical distribution of the cache directory information minimizes space cost while avoiding unnecessary broadcasting of coherency-induced traffic. Our approach contrasts with that of Archibald and Baer

[2], who use 4 bits per directory entry to keep the space overhead reasonable, using 32-bit cache line sizes. Their scheme leads to a node-wide broadcast whenever a cache page frame appears to be shared with another processor node.

The locking scheme bears some similarity to that proposed by Bitar and Despain [3]. Although our scheme also uses cache flags, we are free to discard locked cache blocks from the cache, relying on the memory module to record locking. Since they do not use a directory scheme, they require a separate lock bit that has to be written to memory. We view our scheme as more consistent with uses of locks at the application level, especially when processes may migrate between different processors.

VMP-MC is designed to work well with the virtual memory and transaction management system that we are developing for the V distributed system. VMP-MC appears well suited to support the Mach virtual memory system [19] as well as the 801 transaction software [5], both of which reflect current directions in operating system design.

6 Concluding Remarks

The VMP-MC design is a simple but powerful extension of the basic VMP design we have been investigating for a number of years. We propose it as a building block technology for configuring workstations and parallel machines with 1 to several thousand powerful (50 OS more MIPS) processors.

Several aspects of the VMP-MC design are of particular interest. First, secondary-level cache sharing is exploited to reduce the miss ratios of these caches, the hardware costs of these caches, and the contention between caches. Our simulation results indicate that the seduced miss and contention activity from cache sharing allows more than twice as many modules loading the next level bus. This sharing also significantly seduces the amount of hardware required to support a large-scale configuration, particularly at the MPM level. The reduction in cost and reliability problems makes the architecture practically scalable. The sharing also reduces the average memory reference cost.

Second, the hierarchical directory-based consistency scheme allows coherency, locking and message traffic to be selectively broadcast, if not unicast, to just the affected processor(s). In contrast to the original VMP design, the memory directory-based consistency scheme eliminates the per-processor action table from each processor module, making this module independent of the physical memory size. The large cache line size of VMP allows this scheme to be implemented with less than 2 percent space overhead. The hierarchical extension of VMP is transparent to the software except for various scheduling controls and heuristics that we are introducing to improve the inter-MPM cluster behavior.

Finally, VMP-MC provides explicit support for locking and message exchange, reducing the cost of these operations, particularly for large-scale configurations. The locking facility essentially provides a contention *control* mechanism, allowing the software to synchronize with little contention. The message facility allows the operating system to avoid contention as part of implementing int esprocess and memory management operations.

Our work to date has developed the design and implemented several of the components of VMP-MC as well as provided an initial performance evaluation of the design based on trace-driven simulation. Further work is required to fully evaluate the feasibility of this design, including construction of the multiple processor board. This is the next focus of our hardware effort. Considerable software effort will be required along the way to properly exploit this architecture.

Overall, we see the VMP-MC as providing a credible approach to building a scalable multiprocessor without using costly technology or giving up the availability of shared memory, an important facility for many parallel applications. As a building block technology, it provides a means of configuring a wide range of general-purpose parallel machines, ranging from a moderate

scale multiprocessor to a teraop multi-computer configuration. This approach offers a lower entry cost, greater generality and easier extensibility than the approaches to large-scale parallel machines proposed by many other research projects. We hope to further substantiate these conclusions by the construction and experimentation evaluation of a VMP-MC configuration following the design described in this paper.

7 Acknowledgements

We are grateful to Anoop Gupta and Wolf Weber for making the trace data used in this paper available to us. This paper has benefited from comments and criticisms of members of the Distributed Systems Group at Stanford.

References

[1] A. Agarwal, R. Simoni, J. Hennessy, and M. Horowitz. Scalable directory schemes for cache coherence. In *Proc. 15th Int. Symp. on Computer Architecture,* pages 280–289. ACM SIGARCH, IEEE Computer Society, June 1988.

[2] J. Archibald and J.L. Baer. An economical solution to the cache coherence problem. In *Proc. 12th Int. Symp. on Computer Architecture,* pages 355-362. ACM SIGARCH, June 1985. Also SIGARCH Newsletter, Volume 13, Issue 3, 1985.

[3] P. Bitar and A.M. Despain. Multiprocessor cache synchronization issues, innovations, evolution. *In 13th Int. Symp. on Computer Architecture,* pages 424-433. ACM SIGARCH, IEEE Computer Society, June 1986.

[4] M. Censier and P. Feautier. A new solution to coherence problems in multicache systems. *IEEE TC,* C-27(12):1112-1118, December 1978.

[5] A. Chang and M. Mergen. 801 Storage: Architecture and Programming. In *11th Symp. on Operating Systems Principles.* ACM, November 1987.

[6] D.R. Cheriton. Unified management of memory and file caching using the V virtual memory system. Submitted for publication, 1989.

[7] D.R. Cheriton, A. Gupta, P. Boyle, and H.A. Goosen. The VMP multiprocessor: Initial experience, refinements and performance evaluation. *In Proc. 15th Int. Symp. on Computer Architecture,* pages 410-421. ACM SIGARCH, IEEE Computer Society, June 1988.

[8] D.R. Cheriton, G. Slavenburg, and P. Boyle. Software-controlled caches in the VMP multiprocessor. In *13th Int. Conf. on Computer Architectures.* ACM SIGARCH, IEEE Computer Society, June 1986.

[9] R. Fitzgerald and R.F. Rashid. The integration of virtual memory management and interprocess communication in accent. *ACM Transaction on Computer Systems,* 4(2):147–177, May 1986.

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

[11] H. Kanakia and D.R. Cheriton. The VMP network adapter board (NAB): High-performance network communication for multiprocessors. In *SIGCOMM* '88 Symposium, pages 175–187. ACM SIGCOM, IEEE Computer Society, August 1988.

[12] Leonard Kleinrock. *Queueing Systems, Volume 1:* Theory. Wiley Interscience, 1975.

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

[14] R.T. Short and H.M. Levy. A simulation study of two-level caches. *In Proc. 15th Int. Symp. on Computer Architecture,* pages 81-88. ACM SIGARCH, IEEE Computer Societv. June 1988.

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

[16] A.J. Smith. Line (block) size choice for cpu cache memories. *IEEE Transactions on Computers,* C-36(9):1063-1075, September 1987.

[17] W. Weber and A. Gupta. Analysis of cache invalidation patterns in multiprocessors. To appear, ASPLOS 1989.

[18] A.W. Wilson, Jr. Hierarchical cache/ bus architecture for shared memory multiprocessors. In *14th Int. Conf. on Computer Architectures,* pages 244-253. ACM SIGARCH, IEEE Computer Society, June 1987.

[19] M. Young et al. The duality of memory and communication in the implementation of a multi-processor operating system. *In 11th Symp. on Operating Systems Principles.* ACM, November 1987.