Journal of Parallel and Distributed Computing
Vol. 15(4) pp. 408-419 (August 1992).

Hierarchical Coherency Management for Shared Virtual Memory Multicomputers †

Yuval Tamir and G. Janakiraman
Computer Science Department
4731 Boelter Hall
University of California
Los Angeles, California 90024-1596
Phone: (310)825-4033 E-mail: tamir@cs.ucla.edu

Abstract

For many applications of scalable multicomputers with distributed memory it is desirable to provide transparent *shared virtual memory*. For such applications, the hardware and system software must maintain coherency among the local memories. Most existing coherency schemes for multicomputers manage memory uniformly at a single granularity of fixed size pages or cache blocks, leading to unnecessarily high storage overhead and poor performance. Coherency management at the granularity of pages (thousands of bytes), results in unnecessary network traffic. At the granularity of cache blocks (tens of bytes), for large systems unacceptably large mapping tables are needed.

We propose a solution to this problem using hierarchical management, where mapping and transfer at the block level are done only when necessary — during the time that the page is *actually* shared. When pages are not shared, they do not require more space for mapping tables than uniform page-level schemes. A detailed description of the scheme is presented, together with an evaluation of its performance and area overheads. The results of trace-driven simulations of the hierarchical coherency scheme are reported. It is shown that the hierarchical scheme makes it possible to achieve the performance advantage of uniform block-based schemes without their large storage overhead.

† This research is supported by Hughes Aircraft Company and the State of California MICRO program.

I. Introduction

Ensembles of tightly-coupled VLSI computing nodes have the potential of achieving high performance, at a relatively low cost, by exploiting parallelism. Such ensembles can be organized as *multicomputers*, where each node is a complete independent computer containing local memory as well as a processor, and where the nodes communicate using messages via point-to-point links [2]. Multicomputers can scale up to thousands of nodes since there is no single component, such as a common bus or shared memory, which can become a performance bottleneck or whose correct operation is critical for the operation of the entire system.

For some applications it is desirable to provide a virtual machine on which all the processors can share a common address space even though memory is physically distributed [12]. For example, this might be desirable for an application where the processes access a large common data structure (e.g. a memory-resident collection of images). Shared virtual memory may also be required to facilitate porting of existing applications which were developed for a shared memory system. A potential advantage of the shared memory model is that the instantaneous cost of process migration (e.g., for load balancing) is relatively low — in order to migrate a process only the contents of a few registers need to be migrated while the memory can remain in place.

Shared virtual memory (SVM) has been implemented [12], using directory-based coherency algorithms [1] to maintain coherent access to shared pages. With page-level schemes, a page (several thousand bytes) is the unit of mapping and transfer, and thus, the granularity of sharing. This large unit of transfer does not provide adequate support for fine-grain parallelism since it results in long delays for interprocessor communication even when the object being (logically) transferred is small. In addition, the large unit of mapping results in reduced system performance due to unnecessary data movement caused by ‘‘false sharing’’ [4]. The difficulties with page-level coherency management motivate the use of schemes which manage coherency at the granularity of cache blocks — the unit of transfer is a block and different blocks of a page can be simultaneously mapped to the memories of different nodes [5].

With the advent of 16 and 64 Mbit DRAM chips, a multicomputer with a thousand nodes is likely to include tens of giga-bytes of physical memory and an even larger virtual address space. These large address spaces will be supported by future microprocessors, such as the MIPS R4000, that will provide 2^{64} byte address space. A key difficulty with support for sharing at the granularity of blocks is that the total number of blocks in such a system will be very large, potentially leading to unacceptably high storage overhead for the required mapping tables. We propose a solution to this problem, based on the assumption, which is validated by trace-driven simulations, that at any one time only a small percentage

of the address space of every process is actually shared. Hence, the mapping tables are partitioned into tables for strictly local, non-shared pages and special tables for shared pages, where it is possible to specify different locations for different blocks in the page [17]. During normal execution, pages are dynamically moved between the different mapping tables, as the need arises. The proposed coherency algorithm introduces the use of *ownership* [10] at both the page level and the block level in order to support block-level sharing and also provide the foundation for dynamic page migration to reduce message traffic and the size of mapping tables.

The basic system organization and a brief review of Li's [12, 13] page-level scheme are presented in Section 2. A simple block-level scheme is presented in Section 3, where the sizes of the required tables are computed. The proposed efficient hierarchical scheme is described in Section 4, including a preliminary evaluation of the savings in mapping tables and the effectiveness of the scheme. Section 5 is a summary of simulation results, based on address traces of parallel applications. Most of the abbreviations and acronyms used in the paper are listed in the appendix.

II. Page-Level Management of SVM

A high-performance node of a message-passing multicomputer consists of: an application processor, local memory, cache memory to speed up access to the local memory, a memory management unit (MMU) that manages translation and protection for the local memory, and a communication coprocessor (ComCo) that handles most of the tasks involved with sending, receiving, and forwarding messages [2, 8]. Shared virtual memory, with management at the granularity of pages, can be provided on such hardware using the normal page fault mechanism of the MMU and relying on the application processor to perform all the necessary operation [12, 4]. For maximum system performance, the application processor should not be involved in the ''emulation'' of shared memory — special-purpose hardware should service remote requests for pages and convert local accesses to non-local pages into request messages to remote nodes. Just as the communication coprocessor handles the various tasks involved in message passing, a dedicated coprocessor, called a memory-to-messages / messages-to-memory transducer (MMT) will provide support for shared memory emulation. The resulting organization of the multicomputer node is shown in Figure 1.

The virtual cache satisfies most of the local processor's memory requests. The MMU performs the address translation in the event of a cache miss or a remote request. A translation lookaside buffer (TLB) in the MMU is used to speed up the translation. On receipt of a message from the network, the ComCo writes the message either into physical memory or into the MMT. Messages that are responses to requests sent out earlier into the network, are written into a physical memory location specified by the

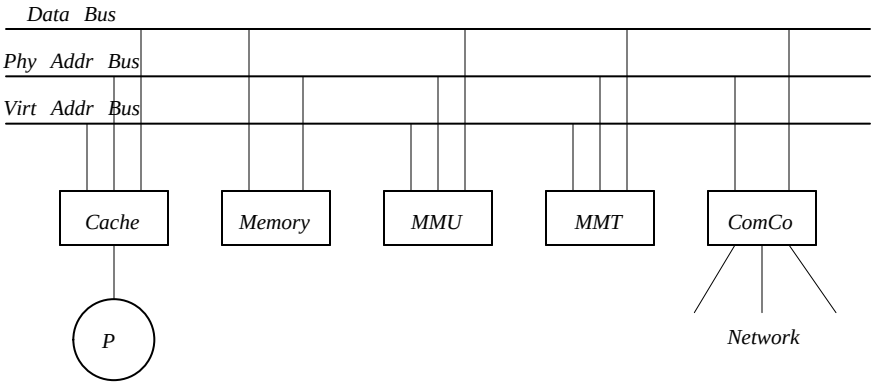

Figure 1: A multicomputer node.

MMU at the time of the request, using DMA. Unsolicited messages such as write-backs, invalidates, and remote requests are written to the MMT. The MMT coordinates with the MMU in servicing messages passed on by the ComCo, and writes/reads the physical memory using DMA. The virtually addressed cache snoops on the transactions between the MMU and MMT to maintain its data coherent with those in physical memory.

Each node may be time-shared between multiple processes. A set of processes that share their address space forms a *task*. The system may be executing multiple tasks simultaneously. Unique task identifiers partition the total virtual system space into disjoint task virtual spaces. Processes belonging to different tasks execute in disjoint address spaces and cannot share data.

Li [11, 12] has implemented coherent SVM on networked workstations and on a multicomputer [13]. Sharing and coherency are managed with the granularity of *pages*. The scheme is directory-based [1] where, with the most promising of the proposed alternatives, the directory is distributed among the nodes. Nodes must obtain exclusive ownership of a page before modifying it, invalidating all other copies of the page. Multiple read-only copies can be distributed throughout the system. With the ‘‘distributed fixed manager algorithm’’ each virtual page is initially managed by a *default manager* whose identity is determined by a simple function of the virtual page number.

III. Block-Level Management of Shared Virtual Memory

As discussed in Section 1, the use of pages as the unit of mapping and the unit of transfer can result in low performance due to the long latencies and the bandwidth requirements for transferring pages and due to false sharing. A simple solution to this problem is to use the same basic algorithms with much smaller pages, henceforth referred to as *blocks*, as the unit of mapping, transfer, and ownership [5]. Such a scheme requires address translation to map any virtual block number to a local block frame, a remote node, or secondary storage. Unfortunately, in a large multicomputer, the total number of blocks in all the

virtual address spaces of all the tasks in the system will be too large for address translation based on conventional page tables (block tables).

The above problem can be solved by using an *inverted table* [7] for translating virtual addresses to local physical addresses. The inverted block table on each node, called the *present block table* (PBT), contains one entry for each block frame in the local physical memory (Figure 2). Hence, the size of the PBT on each node is proportional to the size of the physical memory, rather than virtual memory, on the node. Hashing is used to provide fast associative lookups in the PBT [7]. If a local or remote memory access does not ''hit'' in the PBT, the default owner (manager) of the block is computed and a remote request for the block is sent to that processor. The default owner of a block may refer to a Migrated Block Table (MBT) to determine whether the block has migrated to another processor or if it is on disk.

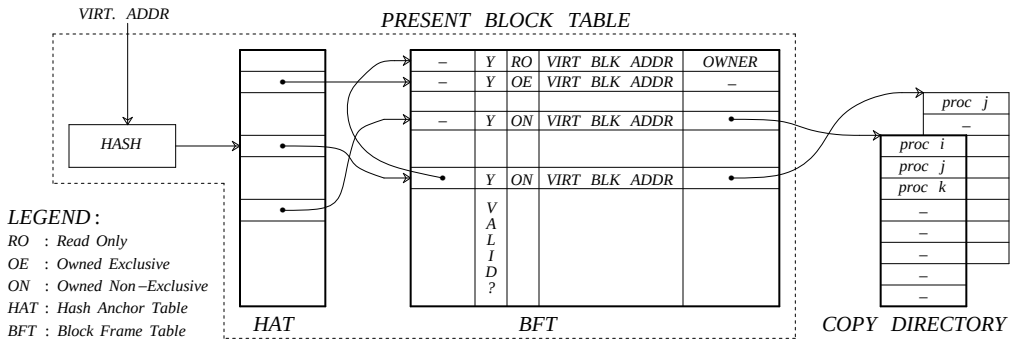

Figure 2: Address translation tables for the block-level (one-level) scheme.

If a block is in read-only state, its PBT entry specifies the block owner. If the block is owned non-exclusively, the same field in the PBT is used to point to the *copy directory*, that contains the identities of the processors that possess read-only copies of that block [12]. This information is used for invalidating all read copies prior to any write to the block. A very large copy directory may be needed to accommodate blocks that are read shared by many processors. There are several approaches to limiting the size of the copy directory. With broadcast protocols, if the number of read copies exceeds some fixed bound, a system wide broadcast is required to invalidate a block [1]. The disadvantage of this approach is that many multicomputer interconnection networks do not provide efficient support for broadcasts. An alternative to broadcast protocols is to invalidate one of the read-only copies of the block whenever a new read request is received by the block owner and the size of the copy directory has already reached its limit. With such an *eviction* policy [5], upon receipt of a read request that would ''overflow'' the copy directory, the block owner picks a victim from the copy directory, invalidates it, and uses the freed slot in the copy directory for the new request. The performance impact of limiting the size of the copy directory has been evaluated using the access-burst model [9]. The analysis [16] indicates that a copy directory

limited to eight entries has very little degradation with respect to the infinite copy directory, for write probability ranges observed in practical applications [3, 9].

As an example of the sizes of the mapping tables and directories, consider a multicomputer with 1024 nodes, each with 32 MBytes of physical memory. The task identifier is 16 bits wide and the maximum virtual address space per task is 1 TB (2^{40}). A page in this system is 4 KB and a block 128 bytes. The PBT on each node has 256K entries, each about 113 bits wide. Hence, the PBT requires 3.5 MB of the local memory. With 8 entries in the copy directory of each of the 256K blocks in the PBT, the total size of the copy directory in one processor would be approximately 2.5MB. The MBT does not have to be resident in local memory but its total size can become unmanageable.

IV. Hierarchical Management of Shared Virtual Memory

With uniform block-level sharing, all ownership and mapping information is at the granularity of blocks. For every block in physical memory, space is allocated to maintain its access rights and copy directory. The hierarchical scheme proposed in this section improves upon the one-level scheme by optimizations based on the following two observations. First, a large portion of physical memory is used exclusively by the local processor for long periods of time. Hence, the one-level scheme is wasteful in allocating storage space for access rights and copy directory for each block in the local memory. Second, private segments are likely to occur in contiguous chunks much larger than a block. A private page, that would require only one mapping entry in a page level scheme, would need as many entries as the number of blocks per page, in a block level scheme. Thus, the block level scheme also results in an unnecessarily large number of mapping and directory entries for private segments of the memory.

A. The Two-Level Management Scheme

The hierarchical SVM management scheme distinguishes between private and shared segments (pages) in physical memory and allocates space for coherency state information and mapping only for the shared pages. The private segments are managed at a page level granularity. Pages dynamically change state, on demand, between private and shared status, transparent to the application. To eliminate the overhead that is caused by false sharing, ownership of shared segments is maintained at both the page and block levels.

A private page located on its default owner (manager) does not require storage of any extra mapping information. On a remote request for a block of a privately owned page, the page changes to the shared state. For each shared page in the system, information is maintained regarding two levels of ownership: page level and block level. The processor owning the shared page maintains the block ownership

information for all blocks of that page. Ownership of a block of a shared page is the same as ownership in the one-level scheme. The processor owning a block can provide read-only copies, invalidate read-only copies, or transfer ownership of the block to another processor. If the block owner provides read-only copies of a block, it must maintain the copy directory for that block.

Initially, all pages are private. A remote request for a block of the page changes its state to shared. When a page enters the shared state, all of its blocks are owned by the page owner. Ownership of individual blocks may be transferred to other nodes that need to modify the block. Hence, the owner of a shared page does not necessarily possess copies of all blocks belonging to that page. As with sector mapping in cache memories [14], space for the entire page must be allocated in the local memory of every processor that has any blocks from that page.

The address translation tables are organized in a two-level hierarchy (Figure 3). The first level table indicates which virtual page is mapped in each page frame of the physical memory and whether the page is shared. On a memory request, the first level table identifies if the page containing the block being requested is present in local memory. If the page is private, the block is guaranteed to be present. If the page is shared, the second level table indicates whether the requested block is present with sufficient access permission. For every owned shared page, the second level table maintains the identity of each block owner. For every shared page that is not owned, the second level table maintains the identity of the page owner. If the page is present and shared, but the block is absent, the request is forwarded to the page owner, or to the block owner if this processor is the page owner. As in the one-level scheme, on a page miss the request is forwarded to the default owner.

The first level table, the *present pages table* (PPT), is organized as an inverted table, consisting of a hash anchor table (HAT) and a page frame table (PFT) [7]. Since the PPT maps pages rather than blocks, it is smaller than the PBT in the one-level scheme. For each page in the physical address space, the PFT maintains the virtual page number, a bit to indicate if the page is shared or private and, if the page is shared, a pointer to a second level table entry. The address translation for a private page need not access the second level table.

The second level table, *shared page table* (SPT) maintains information about shared pages. The SPT entry indicates whether the page is owned, and includes the access rights of each block in that page. If the page-owned bit is not set, the page owner field holds the identity of the processor owning the page. Two state bits for every block in a shared page indicate the access rights to the block: invalid, read only, owned non-exclusive, and owned exclusive (write permission) [10]. Each SPT entry includes a field which may contain a pointer to the copy directory. If the page is owned, the copy directory must indicate

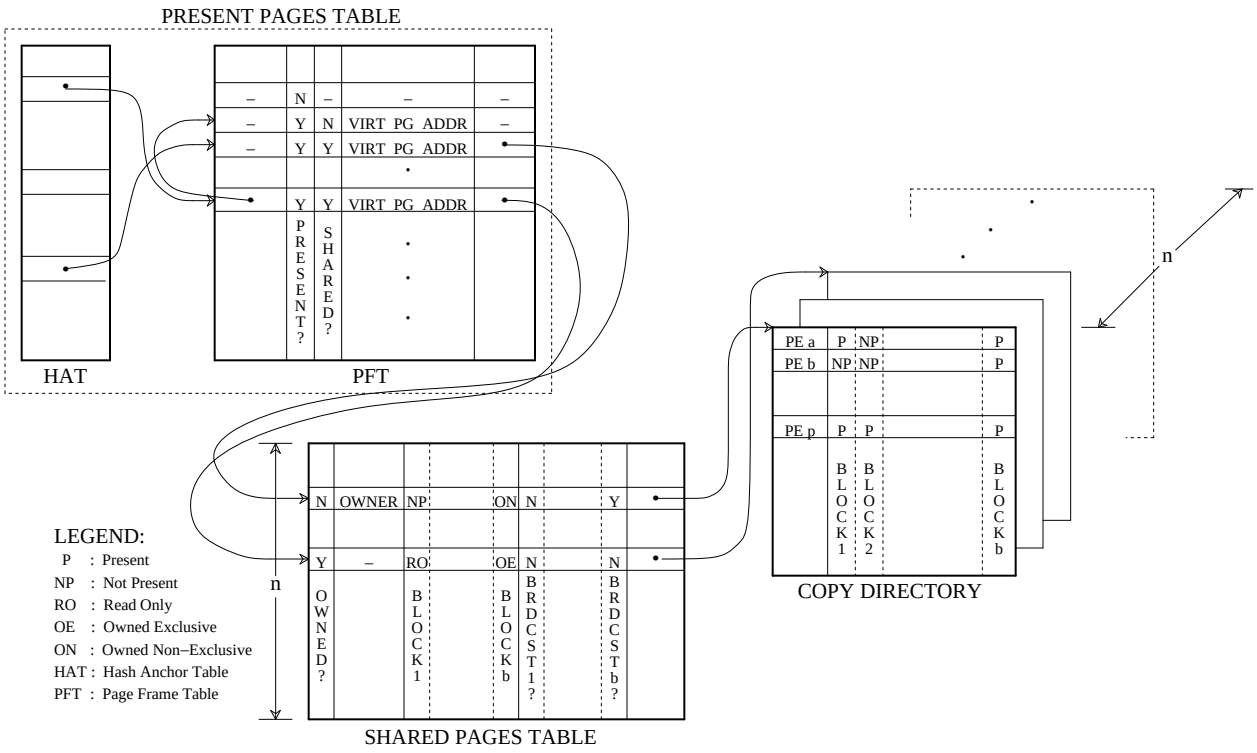

Figure 3: Address translation tables for mixed granularity sharing.

the owners of each block which is not owned locally. For every block owned non-exclusively, the copy directory indicates all the nodes that have read-only copies of the block. If neither the page nor the block are owned by the local node, messages destined to the block owner are sent to the the *page* owner, specified in the SPT. The page owner may then forward such messages to the block owner. As in the one-level scheme, the copy directory size can be limited by following a policy of eviction or broadcast. The organization of the copy directory is discussed in Subsection 4.2. If the broadcast policy is used, each SPT entry includes one *broadcast bit* per block (Figure 3).

A key to the efficiency of the two-level scheme is that the required number of entries in the SPT is significantly smaller than the number of entries in the PPT. In many parallel applications the proportion of references to shared storage is less than 15% [3]. Furthermore, only part of the shared space is actually shared during each ‘‘phase’’ of the program. Hence, an SPT with significantly fewer entries than the PPT should be sufficient to accommodate the ‘‘shared working set’’ of most applications. When the SPT ‘‘overflows,’’ an entry can be freed up by flushing a page that is not owned by the local node back to its owner or by ‘‘recalling’’ all the blocks of an owned page and making the page private. If page migration is supported, an SPT entry may be freed by migrating an owned page to another node which already owns some of its blocks. We expect an SPT which can accommodate about 10% of the working set of the process, roughly 10% of the entries of the PPT, to be sufficient (see Section 5).

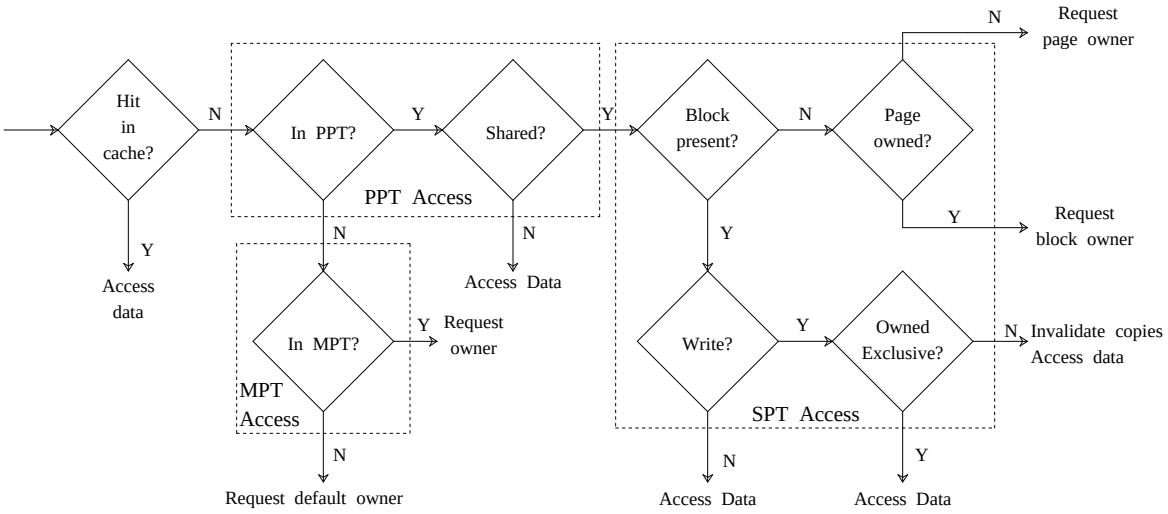

Figure 4: Address translation for local requests.

Figure 4 shows the sequence of actions of the address translation mechanism for local requests. Remote requests are handled similarly but without the initial lookup in the local cache. The messages exchanged between the nodes to maintain coherency are presented in Table 1. The Block Request message requests transfer of a block and the Block message transfers the block. While coherency is maintained at the granularity of blocks, it is sometimes beneficial to perform coherency transactions at the level of a page. The Page and Page Request messages perform such page-level coherency operations. When a shared page is chosen to be made private to free up an entry in the SPT, it is more economical to invalidate all the blocks of the page on remote nodes using a single message rather than to invalidate the blocks of the page individually. To perform the page invalidate, the invalidating node requests the page from all nodes having copies of blocks of the page using a Page Request message. Nodes that owned blocks of the invalidated page write-back their owned blocks through Page messages. All the remote nodes with copies of blocks of the page invalidate those copies and free up the corresponding SPT and PPT entries. The Copy Invalidated message is sent to a block owner by a node that has a copy of the block and decides to invalidate it. The Ownership Transferred message is sent by a block owner to its page owner when ownership of a block is transferred.

Two of the key ideas in our scheme have been pursued independently by O'Krafka and Newton [15]: 1) managing coherency at two levels of granularity, block and sub-block, and 2) exploiting the fact that not all the blocks are actually shared at each point in time. The scheme proposed in [15] is for a system with a limited number of processors where the main memory is shared and there is a central directory. Our scheme is suitable for scalable, distributed memory multicomputers without a central directory. Furthermore, the first idea is not fully exploited in [15] since, even when only one sub-block of

Message	Function	Type
Block	Transfers a block	Data
Page	Transfers a page	Data
Block Request	Requests transfer of a block	Control
Page Request	Requests transfer of a page	Control
Invalidate Block	Invalidates a remote node's copy of a block	Control
Copy Invalidated	Informs the block owner of the invalidation of a copy of the block	Control
Ownership Transferred	Informs the page owner of change of block ownership	Control

Table 1: Messages and their function.

a block needs to be invalidated, invalidations are sent to any node holding any sub-blocks of that block. This results in poor performance [15] due to excessive network traffic. In [15] exploiting the second idea above requires separate associative tag tables. Through the use of simple pointers in the PPT, we accomplish the same functionality with simpler hardware that should result in faster operation. Since the pages are large, the number of entries in the PPT is relatively small and the storage overhead for the additional pointers is insignificant (see Subsection 4.3). There is no discussion in [15] of how the two ideas for reducing the storage requirements for mapping tables can be combined in one efficient scheme, as described earlier in this section. In particular, as discussed in the following subsection, the two-level management scheme allows a new efficient copy directory organization to be used.

B. The Copy Directory

As discussed in Section 3, in most cases the maximum number of entries in the copy directory for each shared block can be bounded with little performance degradation. With the two level scheme, there may be a need to maintain copy directory entries for every block in every shared page. Using a conventional organization of a copy directory [1], a limited number of processor identifiers is maintained for each block of the page (Section 3). We call this organization the *Processor List per Block* (PLpB) organization.

For the hierarchical coherency management scheme, we propose a new organization for the copy directory, called the *Block Map per Processor* (BMpP) organization (Figure 3). This organization minimizes the required storage for the copy directories based on the assumption that two nodes that share one block of a page are likely to share other blocks from that page. For each shared *page* there is a limited number of BMpP entries. Each entry identifies one remote processor that shares one or more blocks of that page. In each entry there is a bit vector (map) with one bit per block indicating whether the particular remote node has a copy of the corresponding block. The maximum number of entries in the copy directory per shared page is the maximum number of nodes that may share a page at the same time.

With the PLpB organization, if a processor shares several blocks of a page, the processor identifier is stored multiple times for that one page. Since every entry in the BMpP copy directory keeps track of all the blocks of the page, the processor identifier is stored once so the total storage needed for good performance is expected to be significantly smaller than with the PLpB copy directory (see Section 5). As with the limited PLpB organization, when a BMpP copy directory is full and a new entry is needed, an existing entry can be evicted by recalling all the blocks in the associated remote node. If necessary, it is possible to use a broadcast bit per block (Figure 3) with the limited BMpP directories to deal with blocks which are shared by a large number of processors.

With both the organizations discussed above, the copy directory is placed at the owner. Since local storage for the copy directory is limited, performance may degrade with such organizations in the presence of heavy sharing in large systems. One way to solve this problem is to implement the copy directory of a block as a linked list of entries (a *chained directory*) distributed among the processors sharing the block [5]. This and other [6] extensions may improve performance under heavy sharing, but will not be discussed further here.

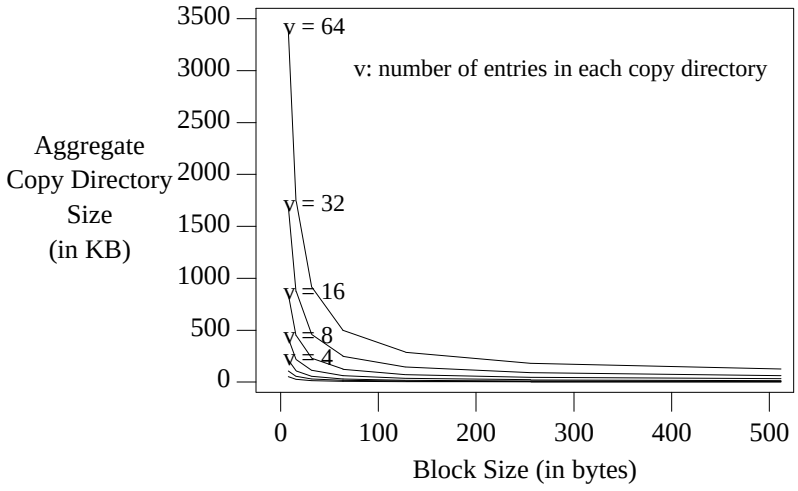

Figure 5: Total size of all the copy directories with the BMpP organization on a node versus the granularity of sharing. 1K entries in the SPT. 4 KB pages. 32 MB local memory. The parameter *v* is the number of BMpP copy directory entries per shared page. The total memory required for the copy directories increases rapidly for block sizes below 64 bytes.

C. The Storage Overhead of the Hierarchical Coherency Scheme

With the system parameters discussed in Section 3, the PPT has just 8K entries as opposed to 256K entries in the PBT discussed in Section 3. As a result, the total size of the PPT is only 88 KB. As discussed in Subsection 4.1, the number of entries in the SPT, and thus the number of BMpP copy

directories can be significantly smaller than the number of PPT entries. With the broadcast bit shown in Figure 3, each SPT entry requires 132 bits of storage. Hence, approximately 17 KB will be required for 1K entries.

With the proposed BMpP organization of the copy directory, given a fixed memory size and page size, the total storage needed for the copy directory is sensitive to the size of the block. For the example system, Figure 5 shows the size of the copy directory per node versus the block size for different number of entries in the copy directory per shared page. Larger blocks lead to smaller entries (fewer blocks per page), and thus reduce the memory used for copy directory storage. However, small blocks are needed to minimize false sharing. For the example system, with a block size of 128 bytes and 32 entries per copy directory, the total copy directory size per node is about 192 KB. It should be noted that, in the one-level scheme, the total size of memory resident tables — PBT and the PLpB copy directory — is approximately 6 MB (18% of local memory) if all blocks are allowed to be shared, and is approximately 3.8 MB (11% of local memory) if 10% of the blocks may be shared. With the two-level scheme, the total size of memory resident tables — PPT, SPT, and the BMpP directory — is about 300 KB, consuming less than 1% of the local memory.

D. Implementation of the Hierarchical Coherency Scheme Tables

Local cache misses and remote requests require the Memory Management Unit (MMU) to refer to mapping and coherency information. To exploit locality, the MMU caches this information in a Translation Lookaside Buffer (TLB). As in conventional systems, the TLB provides fast translation of virtual to physical page numbers. For shared pages, it also provides the full SPT entry. Since the total size of the mapping tables is on the order of a few hundred thousand bytes, it may be possible to store them on chip with the MMU in order to speed up accesses. Alternatively, assuming that the TLB is effective, it may only be useful to store the BMpP copy directories on the MMU chip. The physical organization of the PPT and SPT in normal external memory is straightforward [7] and similar to the logical structure shown earlier (Figure 3).

For ease of access, the BMpP copy directory needs to be physically organized differently than its logical structure (Figure 3). Copy directory accesses fall into two categories: (1) The block is owned locally and a remote processor requests a copy of the block or sends a notification that it has deleted its copy. (2) The block is locally owned non-exclusive and the MMU needs to find the locations of all the copies, or the block is not owned locally and the MMU needs to find the block owner. With the first type of copy directory access, the directory must be searched sequentially based on the processor identifier. Once a match is found (or a new entry is allocated), the corresponding bit vector is updated. For the

second type of access, the directory is accessed directly using the block offset within the page. In this case the desired result is a column bit vector from the table shown in Figure 3, i.e., from each directory entry, the one bit that corresponds to the specified block. Once the bit vector is read, it is used to obtain the processor identifiers corresponding to ‘‘1’’ entries.

The physical organization of the copy directory is geared to the above access requirements. The copy directory is organized as two separate banks. One maintains the processor identifier field of all copy directory entries; the other maintains the bit vectors. The bit vector bank is transposed with respect to the logical structure of the copy directory (Figure 3). Each word in this bank (or multiple words, depending on the word length of the bank and the number of entries in the copy directory) maintains the set of bits associated with a *specific* block of a page (i.e., a column in Figure 3) It should be noted that a memory reference that involves a copy directory access must also involve network communication. Hence, the copy directory access latency is likely to be overshadowed by the network communication latency. Thus, additional hardware support to speed up copy directory access may not be justified.

V. Evaluation Based on Trace-Driven Simulations

We have implemented a simulator that faithfully implements the hierarchical coherency scheme described in Section 4. Our simulations were driven by address traces from a possible execution on a 64 node multiprocessor of three parallel applications: Weather, Speech, and FFT. The Weather application partitions the atmosphere into a three dimensional grid and uses finite difference methods to solve a set of partial differential equations describing the state of the system. The Speech application comprises the lexical decoding stage of a phonetically-based spoken language understanding system. The application uses a modified Viterbi search algorithm to find the best match between paths through a directed graph representing a dictionary and another directed graph representing the input. FFT is a radix-2 fast Fourier transform. The basic characteristics of the traces are summarized in Table 2. In this summary a *footprint* is the total number of distinct 1 KB pages accessed during the execution of the application. The shared footprint is the number of distinct 1 KB pages accessed by more than one processor. More details about the applications and the traces can be found in [5].

In our simulations the PPT size is chosen to accommodate most of the process footprint to ensure that the number of misses due to PPT overflows will be negligible. Specifically, the PPT size for Weather, Speech, and FFT, was 2048, 1024, and 256 pages, respectively. LRU replacement is used with the PPT. Pseudo-random replacement is used to pick a victim copy directory entry when eviction is necessary. The simulator determines the state of the PPT, SPT, and the copy directory after each memory reference and accumulates the information needed for calculating the miss ratios for various types of

Application	Number of Instruction refs (10^6 refs)	Number of Data refs (10^6 refs)	Number of Shared data refs (10^6 refs)	Max data footprint (KB) per processor	Shared footprint (KB)	
					Average per processor	System
Weather	13.64	18.12	5.02	5346	5192	5330
Speech	10.78	11.77	11.77	510	434	596
FFT	3.11	4.33	1.05	164	130	130

Table 2: Trace characteristics. The system shared footprint is the number of 1 KB pages accessed by more than one processor during the execution of the application. The per-processor shared footprint is the average number of pages, out of the system shared footprint, that were accessed by individual processors.

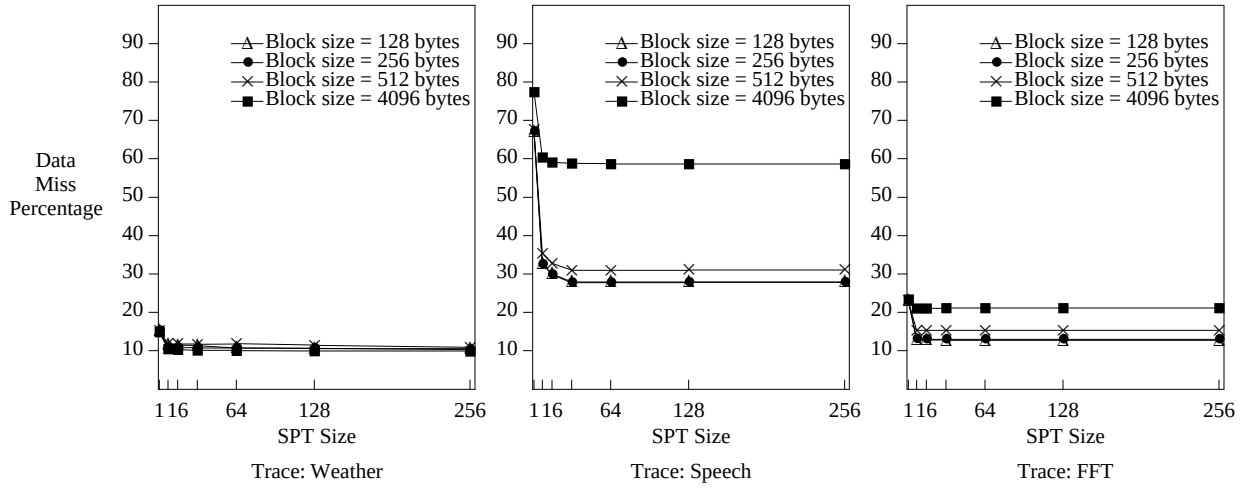

Figure 6: Data miss percentage versus SPT size. 4KB pages. 32 BMpP copy directory entries.

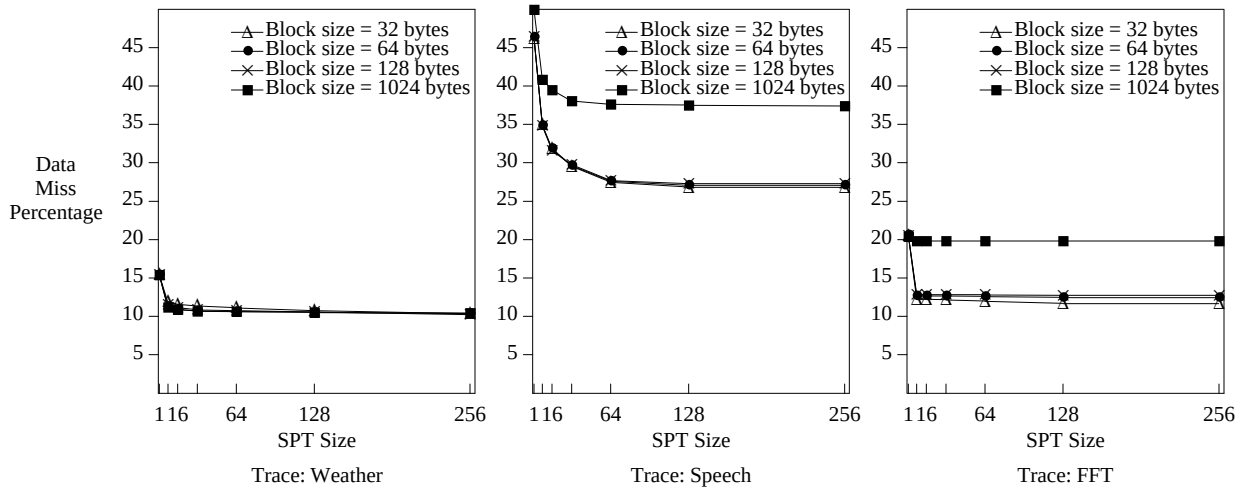

Figure 7: Data miss percentage versus SPT size. 1KB pages. 32 BMpP copy directory entries.

accesses and the network traffic. The miss ratio on accesses to the data address space and the total network traffic are good indicators of the potential system performance. Hence, these measures are used to present the results of the simulation. The network traffic is the total number of bytes transmitted between nodes. It is calculated by summing up the sizes of all the messages exchanged, assuming that each control message is 16 bytes long, while the size of data messages (Block or Page) is 16 bytes plus the data being transmitted.

Simulations of our hierarchical coherency scheme were performed with both 1 KB and 4 KB pages for a range of block sizes, SPT sizes and copy directory sizes. As shown in the rest of this section, the results from these simulations largely validate the basis of our hierarchical coherency scheme. 1) The performance of the coherency scheme is shown to be sensitive to the granularity of management — network load is higher with larger block sizes. 2) Since the hierarchical coherency scheme provides good performance even with small SPTs, it appears that the shared working set of a process is often much smaller than its total shared address space. 3) Given the support for sharing at small block sizes, there is no performance degradation due to the management of private data at large page sizes.

Figures 6 and 7 show the data access miss ratios, for all three applications with various SPT and block sizes. The corresponding network traffic is shown in Figures 8 and 9. The advantage of a small block size is demonstrated by the decrease in network traffic with decreasing block size. For Speech and FFT, decreasing block size also results in a decrease in false sharing, leading to a decrease in miss ratio.

An important result of the simulations is that performance appears to be dependent on block size rather than page size. Specifically, for a page-level scheme (block size equal page size), 4 KB pages lead to significantly higher miss ratios and network traffic than 1 KB pages. However, for equal block size (128 bytes), performance of the system with 4 KB pages is almost identical to the system with 1 KB pages. Since larger pages result in smaller tables, this is a clear demonstration of the advantage of our multi-level coherency scheme.

The results shown in Figures 6-9 also confirm that at any one time, only a small percentage of the address space of every process is actually shared. For very small SPTs, significant reductions in miss ratio and network traffic are obtained by increasing the size of the SPT. However, once the knee of the curves is reached (in this case, at around 32 entries), further increases in the SPT size do not provide significant performance improvement. A small SPT would be capable of accommodating the ‘‘shared working set’’ and provide good performance. The knee occurs at a higher SPT size for 1 KB pages than for 4 KB pages. This indicates that, for these applications, in our two-level scheme, for a fixed block size, a smaller page size will actually *reduce* performance unless a larger SPT is used.

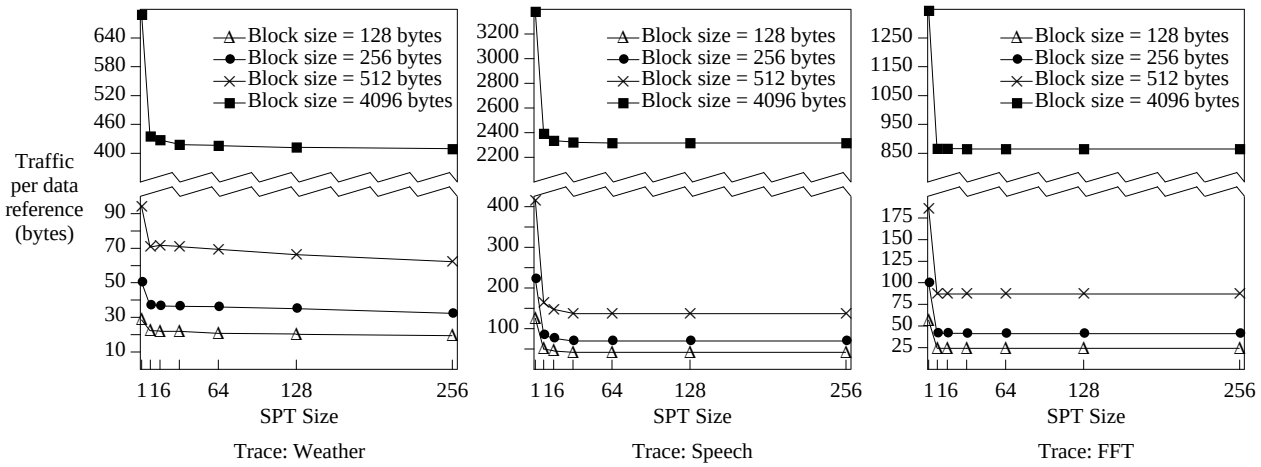

Figure 8: Network traffic versus SPT size. 4KB pages. 32 BMpP copy directory entries.

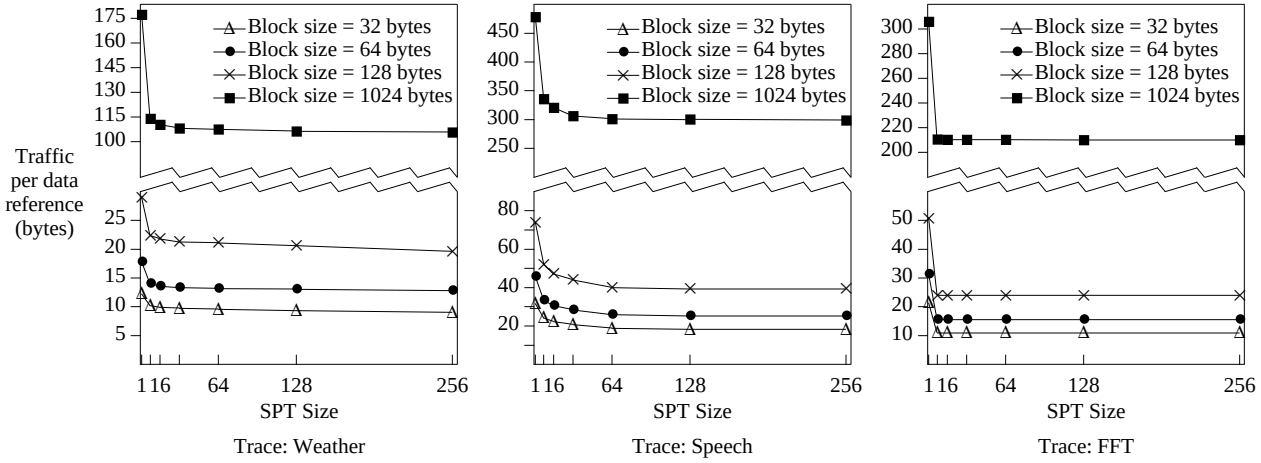

Figure 9: Network traffic versus SPT size. 1KB pages. 32 BMpP copy directory entries.

To further evaluate the impact of limiting the size of the copy directory, we simulated the hierarchical coherency scheme for various sizes of the BMpP copy directory. Figure 10 shows the miss ratio and network traffic. As expected, performance is reduced if the directory is too small. For Weather and FFT, the reduction in miss ratio and network traffic with increasing directory size levels off for a BMpP copy directory with 32 entries or more. This agrees with the expectation that there is little performance advantage in maintaining full copy directories for shared blocks. The different behavior for Speech is due to the high read-only sharing of a dictionary. A policy of eviction, such as we have employed, will fair poorly in the presence of significant read sharing by a large number of processors. Better support for read-shared pages can be provided with a broadcast policy [1] or a policy that allows a small, dynamically determined, set of pages to have larger copy tables [15].

Figure 11 presents the results comparing the performance of the two copy directory organizations:

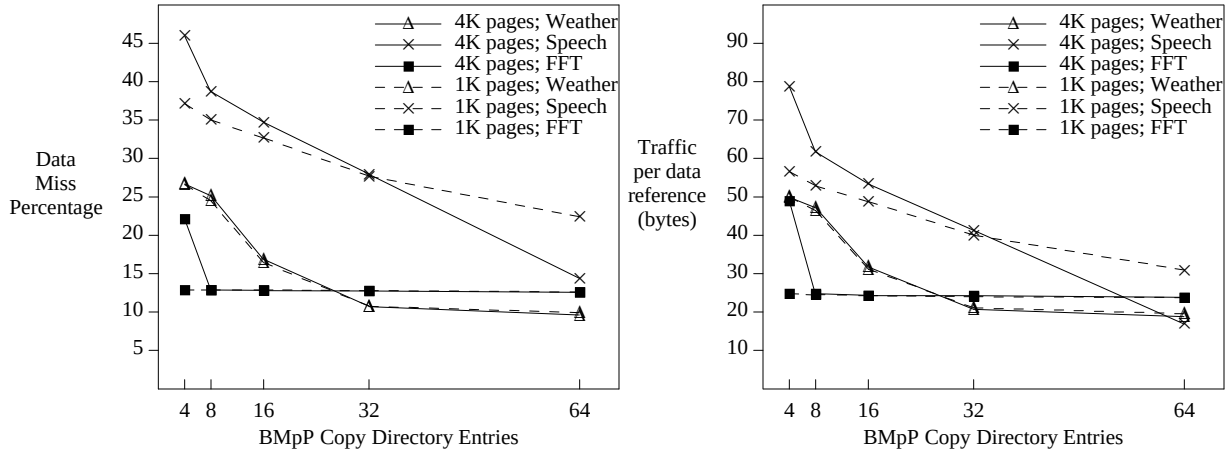

Figure 10: Data miss percentage and network traffic versus number of entries in BMpP copy directory for page sizes 1KB and 4KB. SPT size is 64. Block size is 128 bytes.

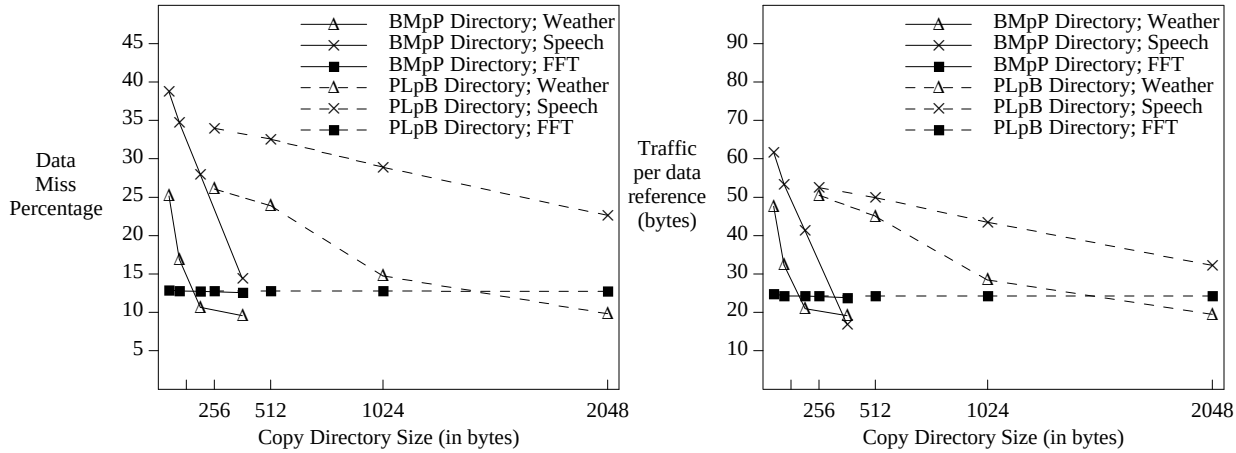

Figure 11: Data miss percentage and network traffic versus copy directory size per page for BMpP and PLpB organizations. Page size is 4KB. SPT size is 64. Block size is 128 bytes.

the BMpP and the PLpB. The graphs show the data miss percentage and the network traffic as a function of the size of the copy directory per shared page, for both organizations. The data points for the BMpP organization correspond to 8, 16, 32 and 64 entries per page, and the data points for the PLpB organization correspond to 4, 8, 16 and 32 entries per block. As mentioned earlier, the storage overhead of the BMpP organization is smaller than the overhead of the PLpB organization. Assuming that table entries are byte aligned in memory, a BMpP copy directory with 32 entries requires 192 bytes per page. With the PLpB directory organization, 256 bytes per page support directories with four entries per block. In our simulations, for comparable performance, the copy directory in the PLpB organization had to be several times larger than in the BMpP organization.

VI. Summary and Conclusions

For many applications, it is useful to allow processes executing on different processors of a multicomputer to share their address spaces. Most existing techniques for providing shared virtual memory on multicomputers perform the mapping, data transfers, and ownership management at a single granularity of blocks (or pages). If large block sizes are used, these techniques suffer from unnecessary long latencies for remote requests, they require higher network bandwidth than is inherently needed by the application, and they lead to unnecessary transfers due to false sharing. We have introduced a practical coherency scheme for multicomputers that manages storage at a granularity close to a cache block. For parallel applications, the smaller block size will lead to increased block ownership time [3], a decrease in miss ratio [9], and is expected to reduce remote access latencies and the required network bandwidth.

A straightforward implementation of coherency management at the block level results in unacceptably high storage overhead for the various mapping tables. We propose the use of multiple levels of mapping and coherency management, a new organization for the copy directory, and inverted tables as a solution to this problem. The proposed scheme dynamically distinguishes between shared and private pages in order to manage each class efficiently. For shared pages, ownership is maintained at both the block and page level, thus distributing the coherency management load according to the characteristics of the application. In order to limit the storage overhead for coherency management, tables that keep track of block copies are not allowed to grow beyond a static bound. In a realistic example of a large multicomputer, less than 1% of the total system storage will be dedicated to tables needed for coherency management.

The performance of the proposed coherency scheme has been analyzed using trace-driven simulations of three parallel applications. In general, the results indicate that significant performance improvement over conventional page-level schemes is possible with our two-level scheme, without incurring the storage overhead that makes uniform block-level schemes impractical in large systems. Using multi-level coherency management it is thus possible to realize the potential of multicomputers for high performance with fine grain parallel applications, even when providing global shared virtual memory.

Appendix — Glossary of Abbreviations and Acronyms

BFT	Block Frame Table
BMpP	Block Map per Processor organization of the copy directory
ComCo	Communication Coprocessor
HAT	Hash Anchor Table
MBT	Migrated Block Table
MMT	Memory-to-Messages / Messages-to-Memory Transducer
MMU	Memory Management Unit
MPT	Migrated Page Table
OE	Owned Exclusively state of a block
ON	Owned Non-exclusively state of a block
PBT	Present Block Table
PFT	Page Frame Table
PLpB	Processor List per Block organization of the copy directory
PPT	Present Pages Table
RO	Read Only state of a block
SPT	Shared Pages Table
SVM	Shared Virtual Memory

Acknowledgements

We are grateful to Anant Agarwal and David Chaiken, of MIT's Laboratory for Computer Science, for providing us with the parallel address traces. They also provided a multi-cache simulator, which served as the basis for the simulator we constructed for our coherency scheme.

Some of the ideas presented in this paper were originally explored in course term projects at UCLA. We acknowledge contributions by Jaime Moreno, in 1986, and by Tiffany Frazier, in 1987.

References

1. Agarwal, A., Simoni, R., Hennessy, J., and Horowitz, M., ''An Evaluation of Directory Schemes for Cache Coherence,'' *15th Annual International Symposium on Computer Architecture*, Honolulu, Hawaii, pp. 280-289 (May 1988).
2. Athas, W. C. and Seitz, C. L., ''Multicomputers: Message-Passing Concurrent Computers,'' *Computer* **21**(8), pp. 9-24 (August 1988).

3. Baylor, S. J. and Rathi, B. D., ‘‘A study of the Memory Reference Behavior of Engineering/Scientific Applications in Parallel Processors,’’ *International Conference on Parallel Processing*, St. Charles, IL, pp. I/78-I/82 (August 1989).

4. Bolosky, W. J., Fitzgerald, R. P., and Scott, M. L., ‘‘Simple But Effective Techniques for NUMA Memory Management,’’ *Twelfth ACM Symposium on Operating Systems Principles*, Litchfield Park, AZ, pp. 19-31 (December 1989).

5. Chaiken, D., Fields, C., Kurihara, K., and Agarwal, A., ‘‘Directory-Based Cache Coherence in Large-Scale Multiprocessors,’’ *Computer* **23**(6), pp. 49-58 (June 1990).

6. Chaiken, D., Kubiatowicz, J., and Agarwal, A., ‘‘LimitLESS Directories: A Scalable Cache Coherence Scheme,’’ *Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, Santa Clara, CA, pp. 224-234 (April 1991).

7. Chang, A. and Mergen, M. F., ‘‘801 Storage: Architecture and Programming,’’ *ACM Transactions on Computer Systems* **6**(1), pp. 28-50 (February 1988).

8. Chow, E., Madan, H., Peterson, J., Grunwald, D., and Reed, D., ‘‘Hyperswitch Network for the Hypercube Computer,’’ *15th Annual International Symposium on Computer Architecture*, Honolulu, Hawaii, pp. 90-99 (May 1988).

9. Dubois, M. and Wang, J.-C., ‘‘Shared Data Contention in a Cache Coherence Protocol,’’ *International Conference on Parallel Processing*, St. Charles, IL **1**, pp. 146-155 (August 1988).

10. Katz, R. H., Eggers, S. J., Wood, D. A., Perkins, C. L., and Sheldon, R. G., ‘‘Implementing a Cache Consistency Protocol,’’ *12th Annual Symposium on Computer Architecture*, Boston, MA, pp. 276-283 (June 1985).

11. Li, K. and Hudak, P., ‘‘Memory Coherence in Shared Virtual Memory Systems,’’ *Fifth Annual ACM Symposium on Principles of Distributed Computing*, Alberta, Canada, pp. 229-239 (August 1986).

12. Li, K. and Hudak, P., ‘‘Memory Coherence in Shared Virtual Memory Systems,’’ *ACM Transactions on Computer Systems* **7**(4), pp. 321-359 (November 1989).

13. Li, K. and Schaefer, R., ‘‘A Hypercube Shared Virtual Memory System,’’ *International Conference on Parallel Processing*, St. Charles, IL, pp. I/125-I/132 (August 1989).

14. Liptay, J. S., ‘‘Structural Aspects of the System/360 Model 85, Part II: The Cache,’’ *IBM Systems Journal* **7**(1), pp. 15-21 (1968).

15. O’Krafka, B. W. and Newton, A. R., ‘‘An Empirical Evaluation of Two Memory-Efficient Directory Methods,’’ *17th Annual International Symposium on Computer Architecture*, Seattle, WA, pp. 138-147 (May 1990).

16. Tamir, Y. and Janakiraman, G., ‘‘Multi-Level Coherency Management for High-Performance Shared Virtual Memory Multicomputers,’’ Computer Science Department Technical Report CSD-900024, University of California, Los Angeles, CA (August 1990).

17. Tamir, Y. and Janakiraman, G., ‘‘Multi-Level Coherency Management for High-Performance Shared Virtual Memory Multicomputers,’’ *IEEE International Phoenix Conference on Computers and Communications*, Scottsdale, AZ, pp. 174-181 (March 1991).