Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596

MULTI-LEVEL COHERENCY MANAGEMENT FOR HIGH-PERFORMANCE SHARED VIRTUAL MEMORY MULTICOMPUTERS

Yuval Tamir
G. Janakiraman

August 1990
CSD-900024

Multi-Level Coherency Management for High-Performance Shared Virtual Memory Multicomputers †

Yuval Tamir and G. Janakiraman

Computer Science Department
4731 Boelter Hall
University of California
Los Angeles, California 90024-1596
U.S.A.
Phone: (213)825-4033 E-mail: tamir@cs.ucla.edu

Abstract

Systems composed of thousands of VLSI computing nodes interconnected by point-to-point links can achieve cost-effective supercomputing performance by exploiting parallelism. While the memory in such *multicomputers* is physically distributed, shared virtual memory is desirable for many applications. Hence, the underlying 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 overhead and poor performance.

If coherency is managed at the granularity of pages (thousands of bytes), network traffic that is orders of magnitude greater than necessary may result. On the other hand, single-level coherency management at the granularity of cache blocks (tens of bytes) results in unacceptably large mapping tables. We propose a solution to this problem using multi-level 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 preliminary indications of its performance and area overheads. The results of trace-driven simulations of the multi-level coherency scheme are reported. It is shown that the multi-level scheme makes it possible to achieve the performance advantage of uniform block-based schemes without their large storage overhead.

† This research is supported by the SDIO Innovative Science and Technology Office, managed by the Office of Naval Research, contracted to the Jet Propulsion Laboratory under task plan #80-2984; and by TRW Corporation and the State of California MICRO program.

August 1990

I. Introduction

Ensembles of a large number of tightly-coupled high-speed VLSI computing nodes have the potential of achieving high performance, at a relatively low cost, by exploiting parallelism. Many existing and proposed multiprocessors provide shared memory to all their nodes using a single shared bus [19], a hierarchy of buses [20], or multi-dimensional grids of buses [11]. These multiprocessors rely on wide, fast, buses. Such buses and the related drivers and arbiters, are expensive and bulky. The need to keep the buses short places tight constraints on the physical structure of the system. Furthermore, the failure of one of the buses or an interface to one of the buses can disable or severely degrade a significant portion of the system.

Multicomputers composed of a large number of independent computing elements (nodes), which communicate using messages via point-to-point links, are an alternative to shared-memory multiprocessors [2]. Each node is a complete computer containing local memory as well as a processor. One of the advantages of this type of system is that there is no single component, such as a common bus or shared memory, which can become a performance bottleneck and whose correct operation is critical for the operation of the entire system, or to the operation of a significant portion of the system. There are many alternative implementations of the point-to-point links, including byte-wide links [8], serial cables [7], or fiber-optics links. These alternatives provide lower cost, more compact packaging, and the potential for higher performance than in systems based on wide buses. In addition, there is great flexibility in inter-connecting nodes that are inches or dozens of yards away from each other using the same basic organization. Furthermore, the basic system components — the nodes — are capable of self-diagnosis and of active participation in reconfiguration in response to changes in the system (e.g. performance bottlenecks or node failures). Hence, systems of this type can scale up to many thousands of nodes and can use sophisticated fault tolerance schemes to recover from component failures.

There are cases where it is desirable to provide for the application a virtual machine on which all the processors can share a common address space even though memory is physically distributed [15]. 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). A shared virtual memory system may also be required if there is a large existing application, which was written based on the shared memory model, and there is a need to run the application on a multicomputer where memory is distributed. A potential advantage of the shared memory model is that process migration (e.g., for load balancing) becomes relatively cheap — in order to migrate a process only the contents of a few registers need to be migrated while the memory can remain in place.

The idea of implementing shared memory in a system with distributed memories is not new. For example, this was done in Cm* multiprocessor [10]. However, with the Cm* data from remote memories was not cached locally, resulting in poor performance for remote accesses. In a more recent system [15], when a remote page is accessed, it is brought to the local memory, where it is kept until another processor needs it or the page-replacement algorithm pages it out. Coherency of the resulting shared virtual memory is maintained using directory-based cache coherency algorithms [1, 5] with pages in each node's main memory being managed as "cache blocks."

With page-level shared virtual memory schemes, a page (several thousand bytes) is the unit of mapping, the unit of 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 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. A key difficulty with support for sharing at the granularity of blocks is that the total number of blocks in such a multicomputer 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. 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* [12] at both the page level and the block level in order to provide block-level sharing and also support dynamic page migration to reduce message traffic and the size of mapping tables.

The basic system organization and a review of Li's [15, 14] page-level scheme are presented in Section II. A straightforward block-level scheme is presented in Section III, where the sizes of the required tables are computed. The proposed efficient block-level scheme is described in Section IV, where a preliminary evaluation of the savings in mapping tables and the effectiveness of the scheme are presented. Section V is a summary of simulation results, based on address traces of parallel applications.

II. Page-Level Management of Shared Virtual Memory

The node of a message-passing multicomputer, designed for high-performance, 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 that handles most of the tasks involved with sending, receiving, and forwarding messages [2, 7]. 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 [15, 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.

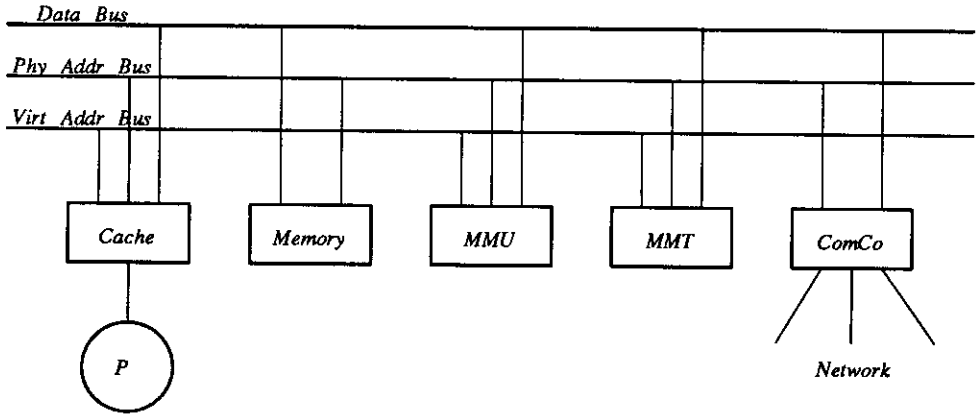

Figure 1: A multicomputer node.

Each node in the system 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 [15, 13] has proposed the implementation of coherent shared virtual memory on loosely coupled networked workstations and has also implemented his scheme on a multicomputer [14]. In Li's scheme, sharing and coherency are managed with the granularity of *pages*. The proposed scheme is directory-based [1] where, with the most promising of the proposed alternatives, the directory is distributed among

the nodes. The scheme requires nodes to obtain exclusive ownership of a page before modifying it so that all other copies are invalidated. 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 I, using 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]. 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). However, address translation is needed to map any virtual block number to a local block frame, a remote node, or secondary storage.

The above problem is solved by using an *inverted table* [6] 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 PBT contains information only about those virtual blocks that are currently resident in physical memory. Hashing is used to provide fast associative lookups in the PBT [6]. The advantage of this scheme is that the size of the PBT on each node is proportional to the size of the physical memory, rather than virtual memory, on the node. 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. If the local processor is the default owner of the block, some mechanism is needed to determine whether the block has migrated to another processor or if it is on disk. The mechanism employed is a Migrated Block Table (MBT) which maintains the current owner of all default blocks that have been migrated away from the node.

Analysis

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. Although the PBT is large — 3.5 MB, it can fit in the physical memory of each node. The MBT does not have to be resident in local memory but its total size can become unmanageable.

The *copy set* of an owned block contains the identities of the processors that possess read-only copies of that block [15]. This information is used when one of the nodes needs to write to the block and

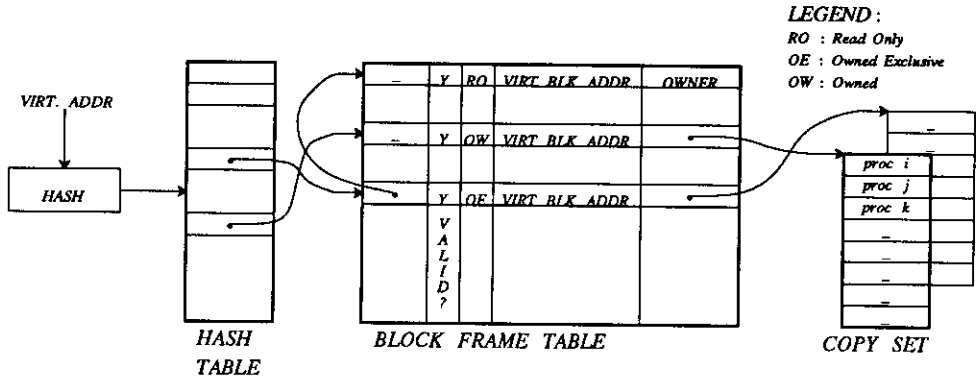

Figure 2: Address translation tables for the One-Level scheme.

all read copies must be invalidated. Very large copy sets may be needed to accommodate blocks that are read shared by many processors. There are several approaches to limiting the size of the copy set. 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 set has already reached its limit. With such an *eviction* policy [5], upon receipt of a read request that would "overflow" the copy set, the block owner picks a victim from the copy set, invalidates it, and uses the freed slot in the copy set for the new request.

The performance impact of limiting the size of the copy set has been evaluated based on the access-burst model [9]. Critical section accesses and the locality of references in shared blocks are modeled by assuming that accesses by one processor to a shared writable block are done in independent uninterrupted bursts called access bursts. An access burst can modify the block with a fixed write probability w. If a copy set overflow is associated with an invalidate, the performance degradation due to the limited number of entries in the copy set can be measured as the increase in the frequency of invalidates over an infinite copy set. The average number of access bursts before an invalidate, for an unlimited copy set is $(1 - w)/w$. For a limited copy set, the average number of access bursts before an invalidate B can be calculated from

$$B(N,w,v) = \sum_{i=1}^{i=v-1} i(1-w)^i w + \sum_{i=v}^{i=\infty} i(1-w)^i \left[\sum_{j=1}^{j=v} P(j,i,N)w + P(v,i,N)(1-w)\frac{N-v}{N} \right]$$

where N is the number of processors sharing the block, v is the maximum number of entries in the copy set, and $P(p,b,N)$ is the probability that exactly p out of the N processors have copies of the block after b read bursts. See the appendix for the derivation of this expression.

Figure 3 is a plot of the average number of access bursts before an invalidate for the infinite copy set and for a copy set limited to eight entries for various values of w. The figure indicates that a copy set with a maximum of eight entries has very little degradation with respect to the infinite copy set for write probability ranges observed in practical applications [9, 3]. With 8 entries in the copy set of each of the 256K blocks in the PBT, the total size of the copy sets in one processor would be about 2.5MB.

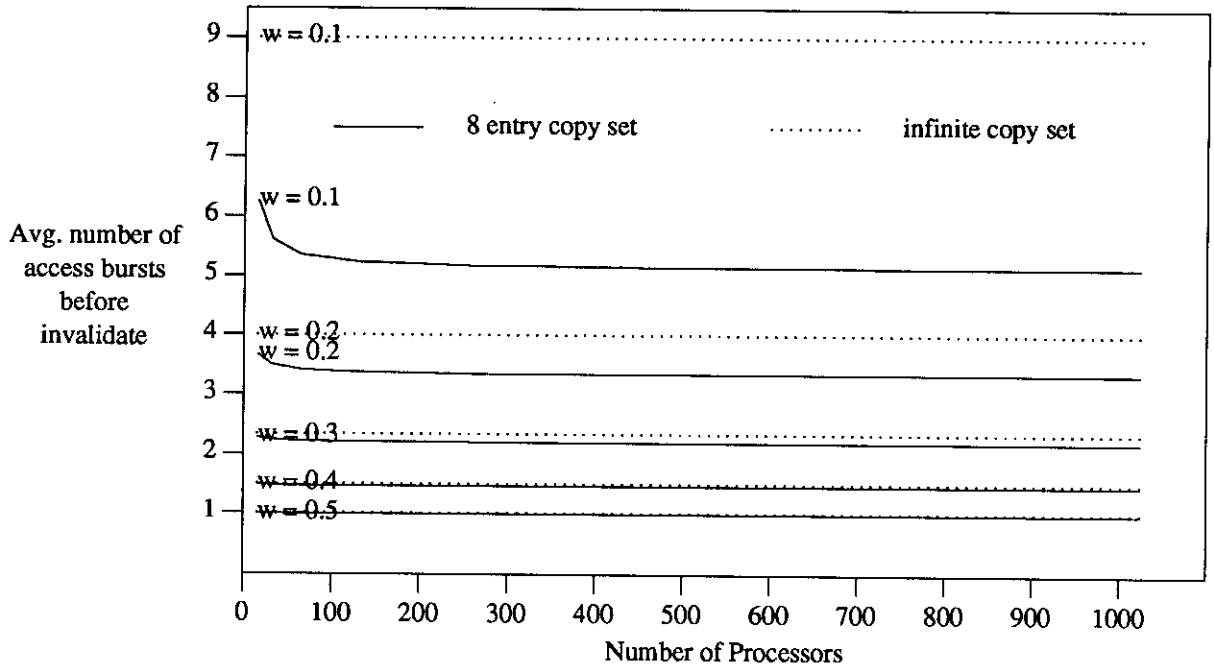

Figure 3: Number of Access Bursts prior to invalidate for the infinite and the finite copy sets.

IV. Multi-Level 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 set. Since a large portion of physical memory is used exclusively by the local processor for long periods of time, the one-level scheme is wasteful in allocating storage space for access rights and copy set for each block in the physical space. Furthermore, since private segments are likely to occur in contiguous chunks much larger than a block, a private page that would have 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. The two-level scheme proposed in this section improves upon the one-level scheme by optimizations based on these two observations. The 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. Since pages dynamically change state, on demand, between private and shared status, the scheme is transparent to the application. In order 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 in order to support the possibility that this page may have been previously shared and may be shared in the future. 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. A processor owning a page is responsible for maintaining 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. The block owner must maintain the copy set for that block.

When a page enters the shared state, all of its blocks are initially owned by the page owner. However, after this point, the page owner of a shared page does not necessarily possess copies of all blocks belonging to that page. As with sector mapping in cache memories [16], space is allocated for the whole page in the local memory. In every processor that has one or more blocks from a page, space must be allocated in the local physical memory for the entire page.

The address translation tables are organized in a two-level hierarchy. 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 mapping tables should indicate if the requested block is present with sufficient access permission. The second level table maintains this additional information about the blocks of each shared page. For every owned shared page, the second level table also maintains the owner identity of each block of that page and for every non-owned shared page, 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. A Migrated Page Table (MPT) akin to the MBT in the one-level scheme maintains the identity of the current owner of default pages that have been migrated. The organization of these tables is shown in Figure 4.

The first level table, the *present pages table* (PPT), is organized as an inverted table, consisting of a

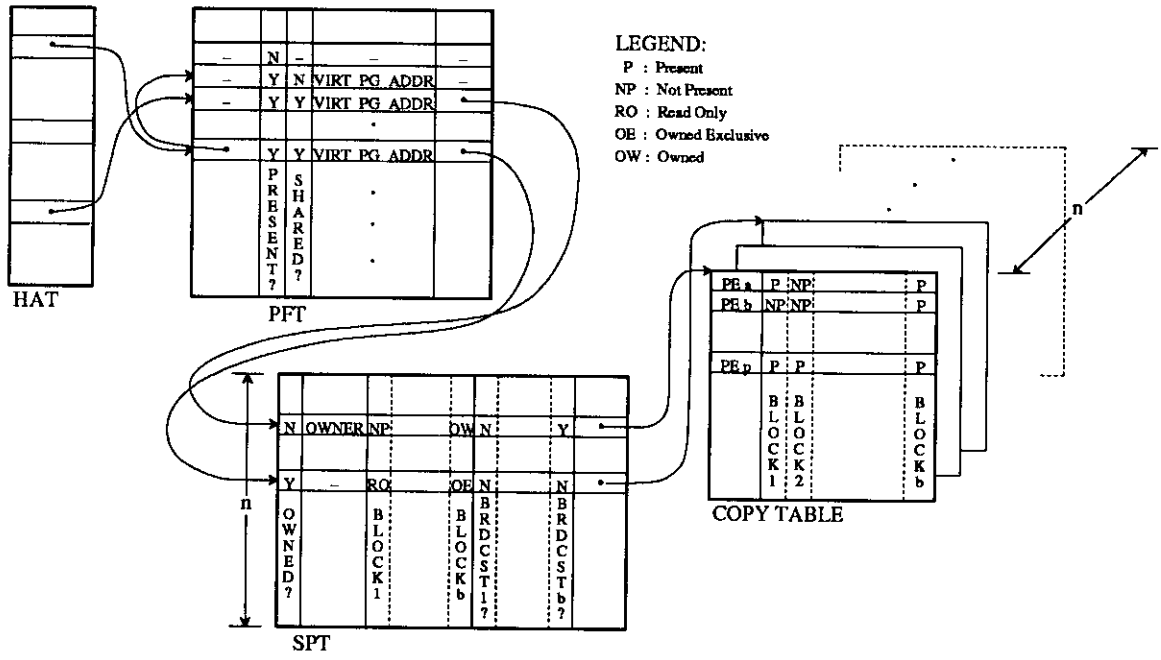

Figure 4: Address translation tables for mixed granularity sharing.

hash anchor table (HAT) and a page frame table (PFT) [6]. Since the PPT maps pages rather than blocks, it is smaller than the PBT in the one-level scheme. The Page Frame Table maintains for each page in the physical address space, the virtual address of the page currently mapped there, 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, the *shared page table* (SPT), maintains additional 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, and owned exclusive (write permission) [12]. Each SPT entry includes a field which may contain a pointer to the Copy Table (CT). The CT contains information similar to the copy set, previously discussed, but is organized differently. Specifically, if the page is owned, the CT must indicate the owners of each non-owned block. For every owned block, the CT indicates all the nodes that have read-only copies of the block. As in the one-level scheme, the CT size can be limited by following a policy of eviction or broadcast. If the broadcast policy is used, in each SPT entry there is a *broadcast bit* for each block in order to indicate that a broadcast is required to invalidate copies of that block (Figure 4).

Figure 5 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 include: Block, Page, Block Request, Page Request, Invalidate Block, Update Ownership, and Update Invalidate. The Block and Page messages transfer the block or page data from one node to the other. The rest of the messages are control messages for maintaining coherence. Of the control messages, the purpose of the first three is obvious from their names. The Update Ownership message is sent by a block owner to its page owner when ownership of a block is transferred. The Update Invalidate message is sent to a block owner by a node that has a copy of the block and decides to invalidate it.

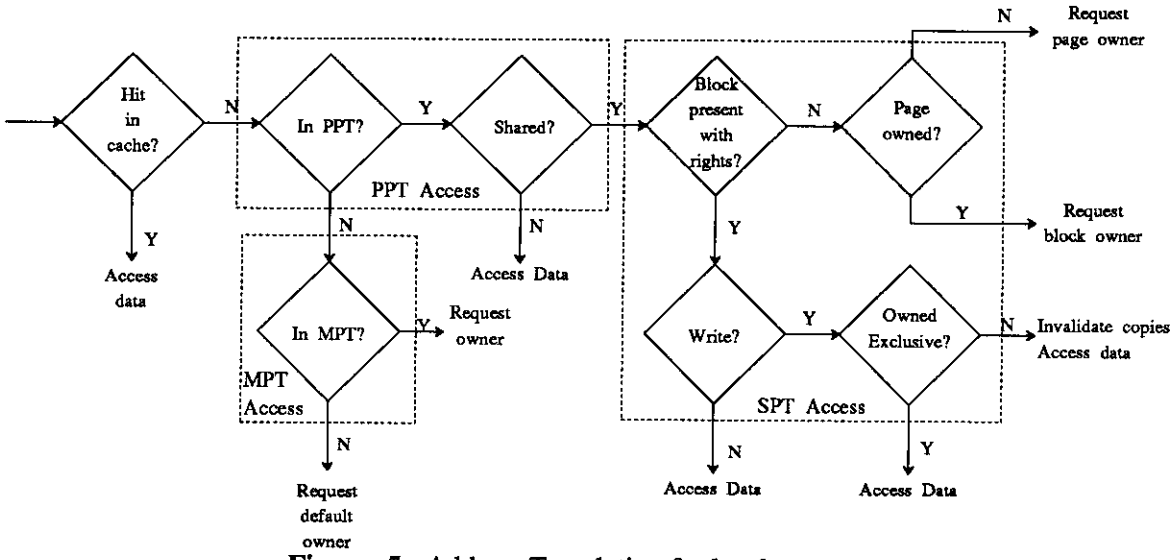

Figure 5: Address Translation for local requests.

Two of the key ideas in our proposed scheme have also been proposed by O'Krafka and Newton [18]: 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. However, the first idea is not fully exploited in [18] since the scheme requires invalidating all the sub-blocks of a block when only one sub-block needs to be invalidated. This limitation results in poor performance [18]. While in [18] exploiting the second idea above requires separate associative tag tables, we have shown that such a table can be eliminated by integrating the required functionality with a standard virtual memory management scheme. There is no discussion in [18] 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.

Analysis of the Proposed Coherency Scheme

With the system parameters discussed in Section III, the PPT has just 8K entries as opposed to 256K entries in the PBT. As a result, the total size of the PPT is only 88 KB. Since the SPT entries are large (on the order of 132 bits) and often require even larger Copy Tables, it is desirable to bound the

number of SPT entries and the size of the Copy Table.

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 an un-owned page back to its owner, "recalling" an owned page and making it private, or 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. With the system parameters discussed in Section III, the SPT will have approximately 1K 132 bit entries, taking up approximately 17 KB.

As discussed in Section III, we expect that the maximum number of entries in the copy set for each shared block can be bounded with little performance degradation. With the two level scheme, there may be a need to maintain copy set entries for every block in every shared page. In the example system, where the number of copy set entries per block may be bounded by eight, we may need to maintain 256 copy set entries per shared page. The Copy Table, shown in Figure 4, is a different organization of the copy set information. This organization attempts to minimize the required storage based on the assumption that, on average, two nodes that share one block of a particular page are likely to share other blocks from that page. In each entry there is a bit vector with one bit per block indicating whether the particular remote node has a copy of the corresponding block. With this organization, the maximum number of entries in the CT per shared page is the maximum number of nodes that may share a page. Since every entry in the CT keeps track of all the blocks of the page, the total number of entries needed is expected to be significantly smaller than in an organization based on one entry per block. For the example system, it is expected that 32 entries will be sufficient. Investigation of other possible organizations of the copy set information is one of the topics of our ongoing research.

With the proposed organization of the Copy Table, given a fixed memory size and page size, the total storage needed for the copy set is sensitive to the size of the block. For the example system, Figure 6 shows the size of the Copy Table per node versus the block size for different number of entries in the CT per shared page. Larger blocks lead to smaller CT entries (fewer blocks per page), and thus reduce the memory used for CT storage. However, small blocks are needed to minimize false sharing. Based on the graph, a block size of about 128 bytes offers a good compromise. At this block size, the total CT size per node is about 192 KB, when the number of entries per CT is 32. For the example system, it should be noted that, in the one-level scheme the total size of memory resident tables — PBT

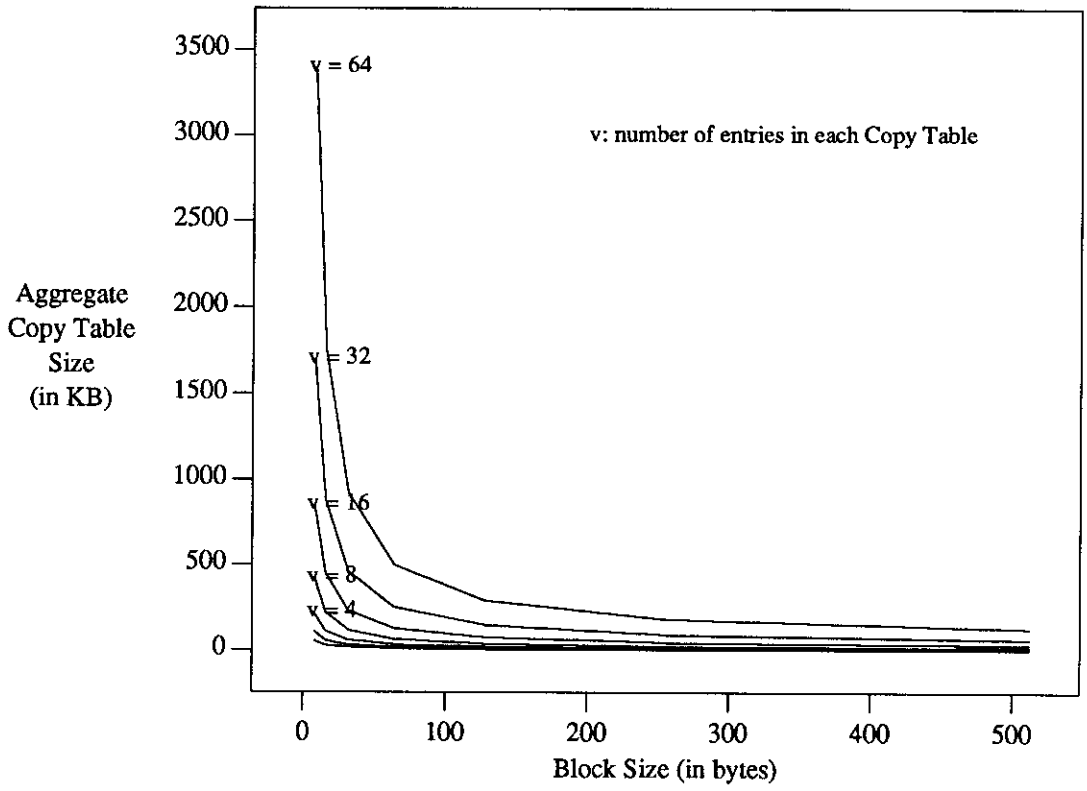

Figure 6: Total size of all the Copy Tables on a node versus the granularity of sharing. 1K entries in the SPT. 4 KB pages. 32 MB local memory. The parameter v is the number of Copy Table entries per shared page. The total memory required for the Copy Tables increases rapidly for block sizes below 64 bytes.

and the copy set — 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) for the same proportion of shared blocks (10%) as in the two-level scheme. The total size of memory resident tables — PPT, SPT and the CT — in the two-level scheme is about 300 KB, consuming less than 1% of the local memory.

V. Evaluation Based on Trace-Driven Simulations

To evaluate the performance of the multi-level coherency scheme, its behavior has been simulated using address traces of parallel programs. This methodology provides the flexibility of studying the performance of the multi-level coherency scheme on realistic applications over a range of parameters. The simulation results corroborate our assumptions and the results of the simple analysis regarding the SPT and Copy Table sizes.

Our simulations were based on address traces from 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

Application	Trace length (10^6 refs)	Number of Instruction refs (10^6 refs)	Number of Data refs (10^6 refs)	Number of Shared data refs (10^6 refs)	Max data foot print (KB) per processor	Shared foot print (KB)	
						Average per processor	of system
Weather	31.76	13.64	18.12	5.02	5346	5192	5330
Speech	11.77	10.78	11.77	11.77	510	434	596
FFT	7.44	3.11	4.33	1.05	164	130	130

Table 1: 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.

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 1. More details about the applications and the traces can be found in [5].

The simulator faithfully implements the multi-level coherency scheme described in Section IV. LRU replacement in used with the PPT. For this study, the PPT size is chosen to accommodate most of the process footprint in order 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. Pseudo-random replacement is used to pick a victim Copy Table entry when eviction is necessary. When an SPT entry has to be freed, if the victim selected is an unowned page, it is flushed back to its owner; if it is an owned page, all of its shared blocks are ''recalled'' and the page is made private. The simulator determines the state of the PPT, SPT, and the CT after each memory reference and builds statistics on miss ratios, network traffic, and the number of SPT and PPT replacements. The size of the PPT was chosen to be sufficient to accommodate all, or a major portion, of the applications' address space, since our primary objective was to study the impact of sharing on performance. Simulations were performed for both 1 KB and 4 KB pages for a range of block sizes, SPT sizes and Copy Table sizes. The system miss ratio on accesses to the shared address space and the total network traffic are good indicators of the potential system performance, and will be used to present the results of the simulation. The network traffic is calculated 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.

Figures 7 and 8 show the data access miss ratios, for all three applications with various SPT and block sizes. The corresponding network traffic is shown in Figures 9 and 10. There is a negligible

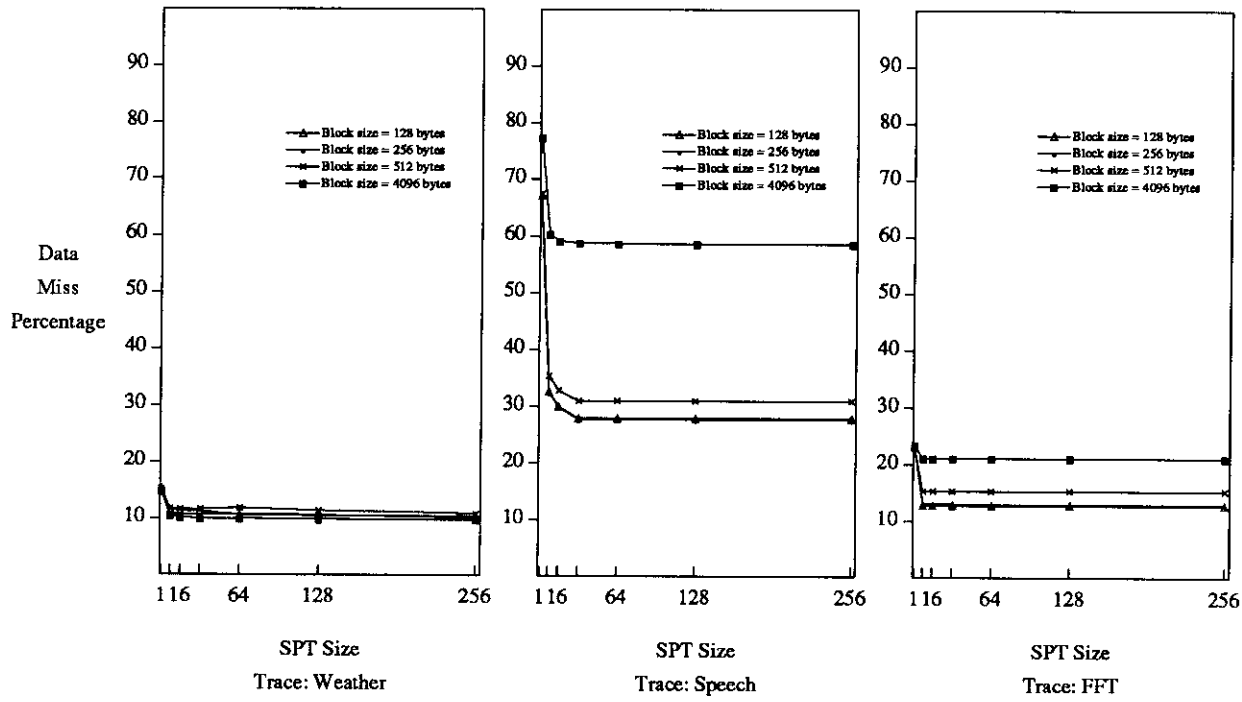

Figure 7: Data miss percentage versus SPT size; 4KB pages; Copy Table size 32.

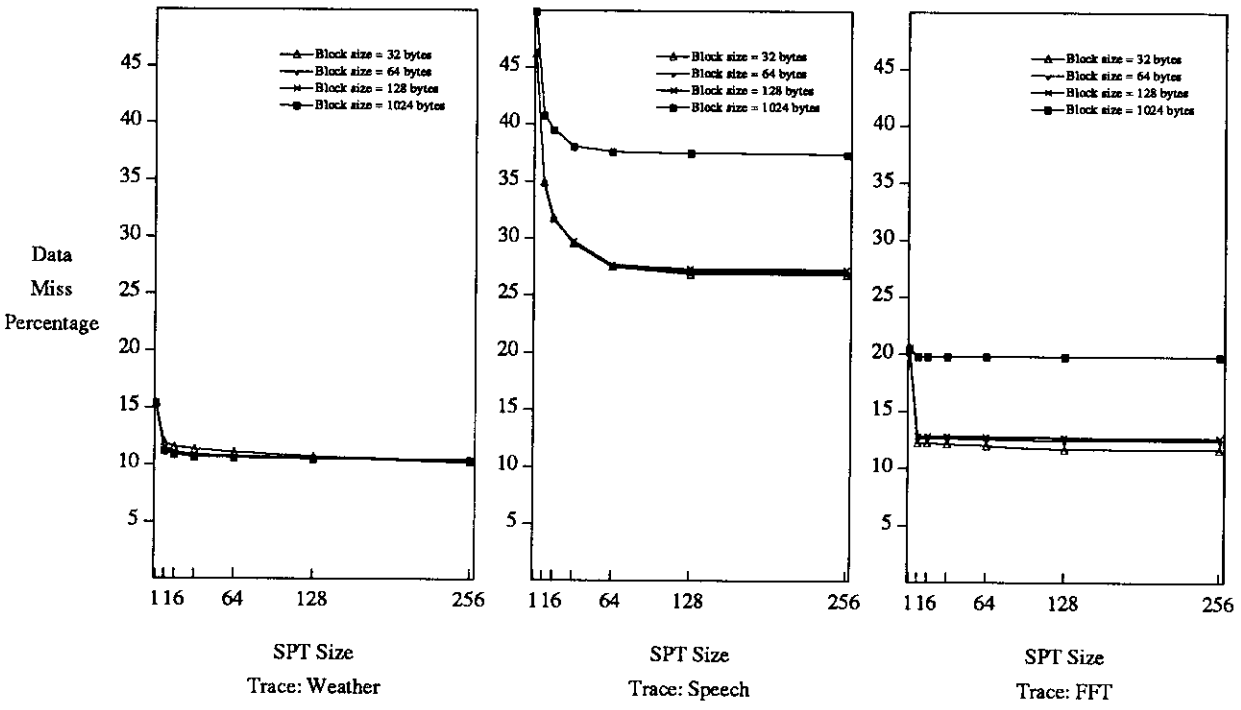

Figure 8: Data miss percentage versus SPT size; 1KB pages; Copy Table size 32.

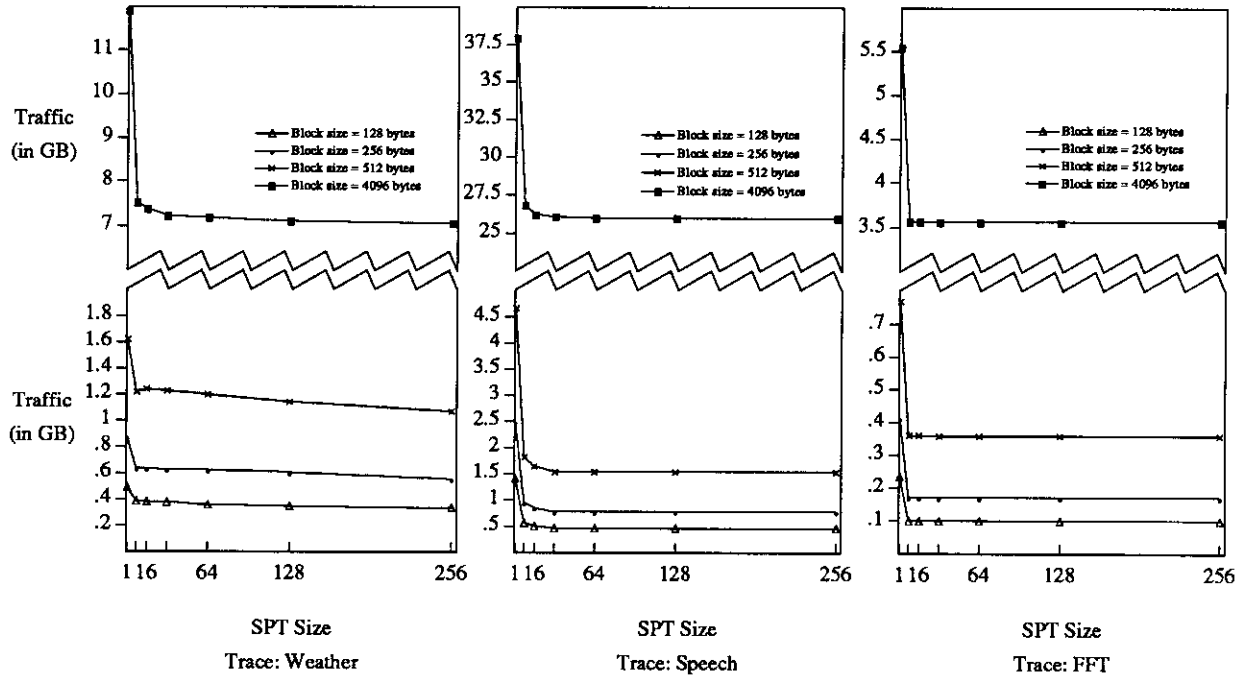

Figure 9: Network traffic versus SPT size; 4KB pages; Copy Table size 32.

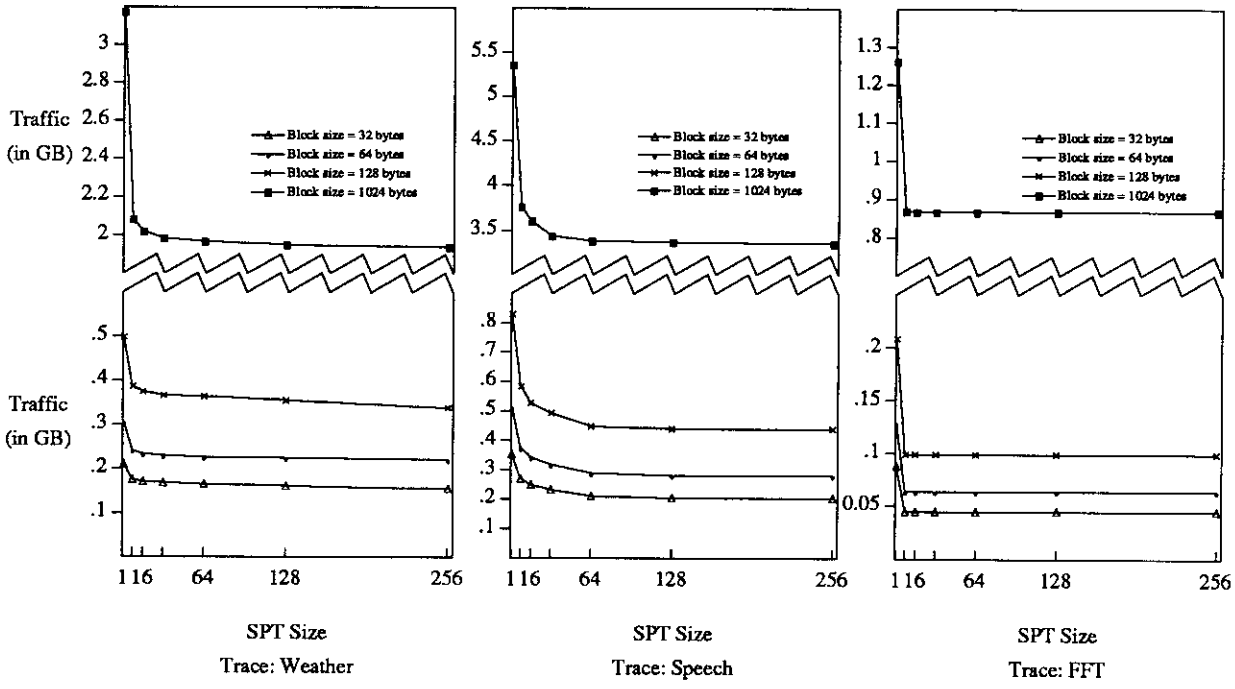

Figure 10: Network traffic versus SPT size; 1KB pages; Copy Table size 32.

contribution to the miss ratio due to private accesses, i.e., due to accesses to unshared data. Hence, the miss ratio for accesses to the shared address space can be easily derived from the information shown, as the product of the total data miss ratio and the ratio of the number of data accesses to the number of accesses to shared space. The key advantage of a small block size is demonstrated by the decrease in network traffic with decreasing block size. Furthermore, with the Speech and FFT applications, decreasing block size results in a decrease in false sharing, leading to a decrease in miss ratio.

An important result of the simulations is that, for all three applications, 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.

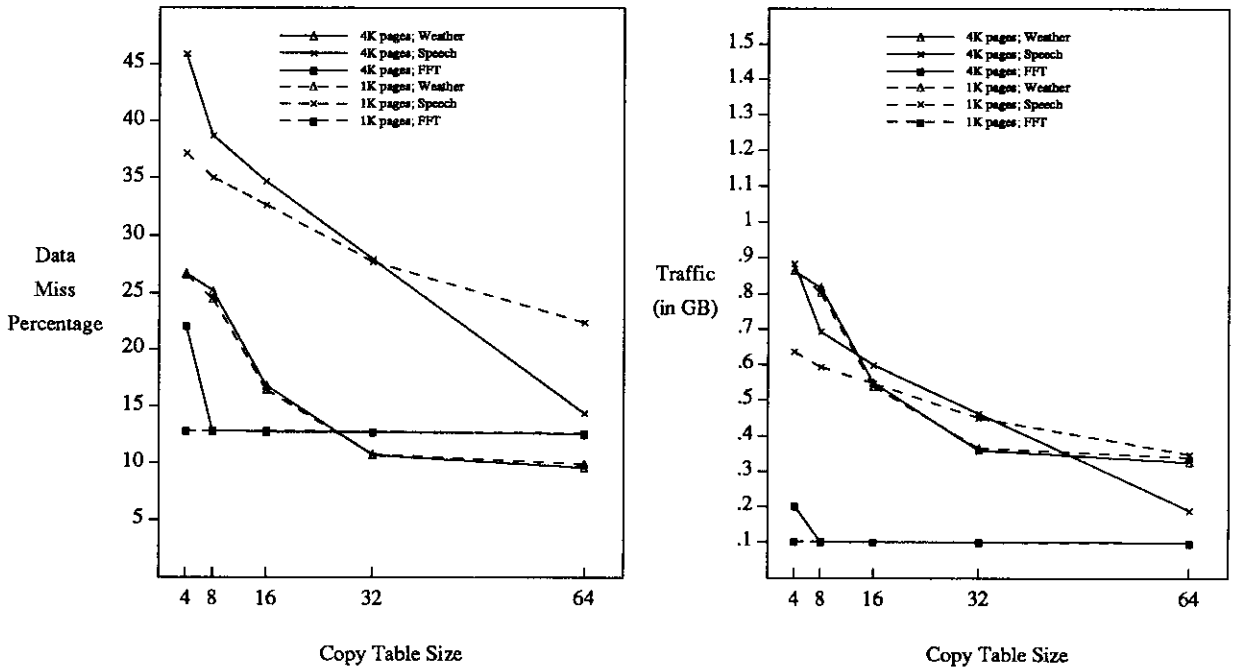

Figure 11: Shared miss percentage and network traffic versus Copy Table size for page sizes 1KB and 4KB; SPT size is 64. Block size is 128 bytes.

The results show that, for very small SPTs, significant reduction in miss ratio and network traffic is 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. These results confirm our expectation that a small SPT (about 10% of the total working set) would be capable of accommodating the "shared working set" and provide good performance. It

should be noted that 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 11 shows the miss ratio and network traffic for various sizes of the Copy Table. As expected, performance is reduced if the CT is too small. For Weather and FFT, the reduction in miss ratio and network traffic with increasing CT size levels off for a CT with 32 entries or more. This agrees with the expectation, based on the access burst model (Section III), that there is little performance advantage in maintaining full copyset tables 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 broadcast policy [1] or a policy that allows small, dynamically determined, set of pages to have larger copy tables [18].

VI. Summary and Conclusions

For many important applications, it is useful to allow processes executing on different processors of a multicomputer to share their address spaces. Previously proposed techniques for providing shared memory on multicomputers, perform the mapping, data transfers, and ownership management at the granularity of pages (thousands of bytes). Hence, when sharing small objects, 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 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 in a large multicomputer results in unacceptably high storage overhead for the various mapping tables. We propose the use of inverted tables and multiple levels of mapping and coherency management 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 read-only copies are not allowed to grow beyond a static bound. The performance impact of this static bound has been analyzed using the access burst model and shown to be negligible. 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. It is shown that some applications can benefit from special support for a large number of copies of strictly read-only data. In general, the results indicate that significant performance improvement over conventional page-level schemes are 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 — The Average Number of Access Bursts Before an Invalidate

Based on the access-burst model [9], the average number of access bursts before an invalidate is:

$$B(N,w,v) = \sum_{i=1}^{i=v-1} i(1-w)^i w + \sum_{i=v}^{i=\infty} i(1-w)^i \left[\sum_{j=1}^{j=v} P(j,i,N)w + P(v,i,N)(1-w)\frac{N-v}{N} \right]$$

where N is the total number of processors sharing the block, w is the probability that an access burst modifies the block, v is the maximum number of entries in the copy set and $P(p,b,N)$ is the probability that exactly p out of the N processors have copies of the block after b read bursts.

In the above expression for B, the first summation corresponds to the case when the invalidate happens within v access bursts. The invalidate, in this case, happens on the first write burst. When the number of access bursts is greater than v, an invalidate can occur in two ways: by a write burst or by a copy set overflow. The probability of it occurring due to a write burst is the product of the probability that there are not more than v processors sharing the block and the probability that the current burst is the first write burst. The first term within the square brackets expresses this. The second term gives the probability that the invalidate is due to a copy set overflow, i.e, exactly v processors shared the block prior to this burst and a new processor began this burst.

$P(p,b,N)$ can be derived, given that each processor is independent and has equal probability of beginning an access burst. We apply the Principle of Inclusion and Exclusion in Combinatorial Theory (see page 101, reference [17]), to obtain:

$$P(p,b,N) = \frac{\left[N \atop p \right]}{N^b} \left[\sum_{i=0}^{i=p} (-1)^i \left[p \atop i \right] (p-i)^b \right]$$

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. We also acknowledge the help of Eugene Brooks, of Lawrence Livermore National Laboratory, for providing access to computing resources used for some of our simulations.

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. A. Agarwal, R. Simoni, J. Hennessy, and M. Horowitz, "An Evaluation of Directory Schemes for Cache Coherence," *15th Annual International Symposium on Computer Architecture*, Honolulu, Hawaii, pp. 280-289 (May 1988).
2. W. C. Athas and C. L. Seitz, "Multicomputers: Message-Passing Concurrent Computers," *Computer* **21**(8), pp. 9-24 (August 1988).
3. S. J. Baylor and B. D. Rathi, "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. W. J. Bolosky, R. P. Fitzgerald, and M. L. Scott, "Simple But Effective Techniques for NUMA Memory Management," *Twelfth ACM Symposium on Operating Systems Principles*, Litchfield Park, AZ, pp. 19-31 (December 1989).
5. D. Chaiken, C. Fields, K. Kurihara, and A. Agarwal, "Directory-Based Cache Coherence in Large-Scale Multiprocessors," *IEEE Computer* **23**(6), pp. 49-58 (June 1990).
6. A. Chang and M. F. Mergen, "801 Storage: Architecture and Programming," *ACM Transactions on Computer Systems* **6**(1), pp. 28-50 (February 1988).
7. E. Chow, H. Madan, J. Peterson, D. Grunwald, and D. Reed, "Hyperswitch Network for the Hypercube Computer," *15th Annual International Symposium on Computer Architecture*, Honolulu, Hawaii, pp. 90-99 (May 1988).
8. W. J. Dally and C. L. Seitz, "The Torus Routing Chip," *Distributed Computing* **1**(4), pp. 187-196 (October 1986).
9. M. Dubois and J.-C. Wang, "Shared Data Contention in a Cache Coherence Protocol," *International Conference on Parallel Processing*, St. Charles, IL **1**, pp. 146-155 (August 1988).
10. S. H. Fuller, J. K. Ousterhout, L. Raskin, P. I. Rubinfeld, P. J. Sindhu, and R. J. Swan, "Multi-Microprocessors: An Overview and Working Example," *Proceedings of the IEEE* **66**(2) (February 1978).
11. J. R. Goodman and P. J. Woest, "The Wisconsin Multicube," *15th Annual International Symposium on Computer Architecture*, Honolulu, Hawaii, pp. 422-431 (May 1988).

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

13. K. Li, "Shared Virtual Memory on Loosely Coupled Multiprocessors," Ph.D. Dissertation, YALEU/DCS/RR-492, Yale University (September 1986).

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

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

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

17. C. L. Liu, *Introduction to Combinatorial Mathematics*, McGraw-Hill (1968).

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

19. S. Thakkar, P. Gifford, and G. Fielland, "The Balance Multiprocessor System," *IEEE Micro* 8(1), pp. 57-69 (February 1988).

20. A. W. Wilson, "Hierarchical Cache/Bus Architecture for Shared Memory Multiprocessors," *14th Annual Symposium on Computer Architecture*, Pittsburgh, PA, pp. 244-252 (June 1987).