PREPRINT: To appear in the *Proceedings of the Twenty-seventh Hawaii International Conference on System Sciences*, January 1994.

Locating Multiprocessor TLBs at Memory

Patricia J. Teller[1]
New Mexico State University
Department of Computer Science
Las Cruces, NM 88003-0001
Tele. 505-646-6243 / Fax 505-646-6281
Email: teller@nmsu.edu

Allan Gottlieb[2]
New York University
715 Broadway, 10th Floor
NYC, NY 10003
Tele. 212-998-3344 / Fax 212-995-4040
Email: gottlieb@nyu.edu

Abstract

This paper compares the performance, in shared-memory multiprocessors, of locating translation-lookaside buffers (TLBs) at processors with that of locating TLBs at memory. Our comparison is based on trace-driven simulations of multiprocessors with log N-stage networks interconnecting N processors and N memory modules.

For the systems and workloads studied, memory-based TLBs perform noticeably better than processor-based TLBs, provided that memory is organized as multiple paging arenas, i.e., multiple clusters of memory modules where the mapping of a page to a cluster is fixed. The cost of a processor-based TLB reload is at least log N because of network transit. In contrast, the cost of a memory-based TLB reload can be smaller, since network transits are not required. Furthermore, with multiple paging arenas, the number of reloads is smaller with memory-based TLBs.

1: Introduction

To implement virtual memory efficiently *translation-lookaside buffers* (*TLBs*) [15], special-purpose virtual-address caches, are used to provide processors with fast access to recently-referenced virtual-to-physical address translation information [20] that are stored in page tables. Conventionally, a TLB is located in a processor. However, in a shared-memory multiprocessor, TLBs can be placed anywhere on the path from the processors to the memory; in particular, as suggested by Teller, Kenner, and Snir [28] TLBs can be located at memory modules.

This paper presents a comparison of the performance, in scalable shared-memory multiprocessors, of consistent TLBs located at processors (*processor-based TLBs*) with that of consistent TLBs located at memory (*memory-based TLBs*). Whether TLBs are located at processors or memory, they must be kept consistent because they store shared read/write data. If they are not, a memory request can be mapped to an incorrect page and, thus, an erroneous memory access can ensue.

Our performance comparison is based on the results of trace-driven simulations of shared-memory multiprocessors with N processing elements (*PEs*) interconnected via a *log N*-stage network to N equidistant memory modules (*MMs*), for $N = 2^n$, $n = 1, ..., 6$. Our multiprocessor simulator is driven by augmented uniprocessor address traces of the execution of parallel programs, which were compressed to include full information w.r.t. shared read/write data and summary information w.r.t. private and shared read-only data. To our knowledge no comparison of this kind has been published.

The results of our study indicate that memory-based TLBs perform noticeably better than processor-based TLBs if the memory modules are partitioned into multiple equal-size paging arenas. A *paging arena* is a cluster of memory modules to which the mapping of a set of pages is fixed. If memory is organized as a set of paging arenas, then whenever a virtual page is memory resident, it is mapped to the same paging arena, but not necessarily to the same frame in the paging arena.

In a shared-memory multiprocessor with a *log N*-stage interconnection network, the service time for a memory request, for example a load, consists of the time it takes to transmit the request through the *log-N* network switches in the PE-to-MM direction, the memory access time, and the time it takes to transmit the requested data through the *log-N* switches in the MM-to-PE direction. Thus, in a system with such a network, the cost of managing TLBs grows as $R\,T_{reload}$, where R is the number of TLB reloads and T_{reload} is the average cost of a reload. Since the cost of a processor-based TLB reload is at least proportional to *log N* because of required network transits, the cost of managing processor-based TLBs grows at least as *R log N*. The cost

[1] Teller was supported in part by Sandia National Laboratories Sandia-University Research Program (SURP) grant no. AE-1680.

[2] Gottlieb was supported in part by U.S. Dept. of Energy grant no. DE FG02 88ER25052-07 and National Science Foundation grant no. NSF MIP-8915488.

of a memory-based TLB reload can be made smaller than that of a processor-based TLB reload because network transits are not required. In addition, our simulations indicate that for systems with memory-based TLBs and multiple paging arenas both R and T_{reload} decrease as the number of paging arenas increases.

This paper is organized into five additional sections: section 2 describes the base multiprocessor model simulated; section 3 describes additions to the model required to implement consistent processor- and memory-based TLBs; section 4 outlines our simulation methodology; section 5 presents results of our simulation study and compares the performance of processor- and memory-based TLBs; and section 6 summarizes our findings, draws conclusions, and discusses future research.

2: Multiprocessor model

Figure 1 depicts our base shared-memory multiprocessor model, which employs a packet-switched *log N*-stage Omega network composed of 2 x 2 switches that have a cycle time equal to that of a processor.

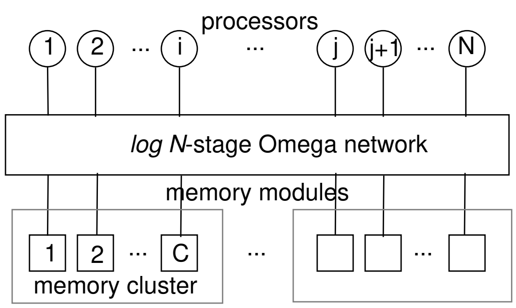

Figure 1: Base multiprocessor model

In the simulated multiprocessor systems:

- a processor with a Harvard architecture executes at 10 MIPS;
- memory modules are equidistant from each processor, (*i.e.*, a uniform-memory-access (UMA) architecture is assumed); the modules are partitioned into equal-size groups called memory clusters;
- main memory is interleaved across each cluster on a double-word (*i.e.*, 8 bytes) basis;
- the memory access delay is two switch cycles;
- memory requests and responses are each composed of two packets, except for double-word load responses and double-word store requests, which are composed of four packets;
- private data, shared read-only data, and instructions are cacheable, as are the address translation information for the associated pages; shared read/write data and the address translation information for the associated pages are not cacheable;
- a cache miss generates two double-word load requests (a cache line is 16 bytes);
- data references are double-word loads or stores (since the applications studied are double-precision floating-point intensive);
- the physical address space is divided into 4K-byte frames, each of which is fully contained in one memory cluster and is interleaved across all memory modules in the cluster; consecutive frames are interleaved across the memory clusters;
- the global virtual address space, in which concurrent processes never use the same virtual address to refer to different objects, is divided into 4K-byte pages; all pages are assumed to be memory resident;
- TLBs have four 32-entry partitions, two each for the user and supervisor address spaces, with one partition of each pair for private and shared read-only pages, and the other for shared read/write pages;
- TLBs are not flushed on context switches and are modeled as fully associative with strict LRU replacement; and
- a round-robin policy is used to schedule tasks to processors, the cost of a context switch being 200 cycles and the time slice (quantum) being 100 ms., *i.e.*, 1,000,000 cycles for a 10 MIPS processor.

The simulated multiprocessor systems differ with respect to:

- TLB placement and consistency methods (processor-based or memory-based);
- N, the number of processors and memory modules ($N = 2^n$, $n = 1, ..., 6$); and
- the number of memory clusters (one or two).

Processor-based TLB systems were simulated with one memory cluster, while memory-based TLB systems were simulated with one and two (equal size) memory clusters. For memory-based TLBs, each cluster is a paging arena.

We explicitly simulate only the TLB partition used for user shared read/write pages and probabilistically model the partition for user private and shared read-only pages. Our traces do not include operating system references, thus, we neither simulate nor model the operating system partitions. Merely modeling (and not simulating) user private and shared read-only pages is a reasonable approach since the number of references to private and shared read-only pages, although much larger than the number of references to shared read/write pages (approximately 10 times as many for two of our benchmarks, Simple and Weather, and over 30 times as many for our third benchmark, Peskin), constitute references to a relatively small number of pages, for which the TLB hit rate would be high.

3: TLB consistency

A page-table modification can result in TLBs storing stale (*i.e.*, no longer valid) translation information, the use of which can lead to erroneous memory accesses, the so called TLB consistency problem (see [27] for a description of the problem and a survey of solutions). Invalidating a virtual-to-physical memory mapping, increasing page protection (*e.g.*, from read-write to read-only), and changing the mapping of a page to a frame can result in instances of this problem. We refer to these three page-table modifications collectively as *unsafe changes*.

Whether TLBs are located at the processors or memory, TLB consistency must be ensured. TLB consistency is ensured in our simulated multiprocessors by the Validation and Memory-based TLBs algorithms, which are described briefly below and in more detail in [28].

3.1: Processor-based TLBs

The Validation algorithm is used to ensure the consistency of processor-based TLBs. Alternatively, we could have used a form of the TLB Shootdown algorithm [4] or a directory-based cache coherence scheme (*e.g.*, see [5]). Like the Validation algorithm, these software alternatives also have an *O(log N)* TLB reload cost.

Using this method, stale TLB entries remain resident in TLBs until they are used to access memory, at which time they are recognized as being stale and are updated. As shown in Figure 2, to implement Validation, each memory cluster has a validation table with an entry for each of its frames--this table is replicated at each MM in the cluster. To identify stale entries, two *generation counts* (*GCs*), which act like time stamps and are related to Smith's one-time identifiers [21], are associated with each frame: one is stored in the page-frame table and the other is stored in the validation table(s) of the frame's memory cluster. GCs (from the page-frame table) are cached in TLBs when the translation information for a page is loaded.

A memory request is accompanied by the GC of the TLB entry for the referenced page and is validated at the memory module by comparing it with the GC stored in the cluster's validation table. As described below, the comparison fails if the address translation information used to make the memory request is stale or if an unsafe change associated with the referenced frame is in progress. In this way a validation table GC acts as a gate, shutting off access to a frame when necessary. In such a case, the requesting processor is instructed to execute a TLB reload, after which it retries the memory request. If the comparison succeeds, then the requested memory access is satisfied.

When a processor desires to make an unsafe change to the translation information of a resident page, it multicasts a message to the memory modules of the frame's cluster causing the frame's GC in the cluster's (replicated) validation table to be updated (*e.g.*, incremented) so that it is different than the associated GC in the page-frame table and the GCs associated with the frame that are stored in TLBs. When the unsafe change is complete, then the GC in the page-frame table is updated as well (making it identical to the GC in the validation table) and, thus, the gate to the memory module is opened.

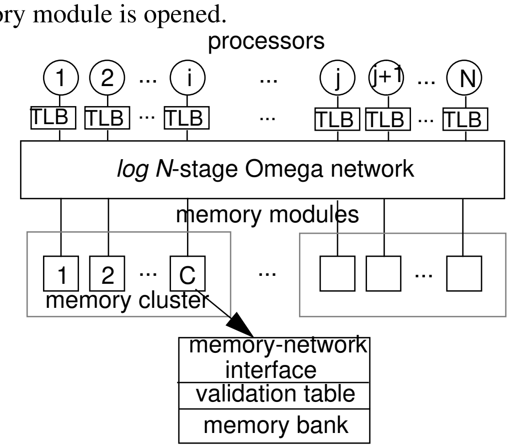

Figure 2: Multiprocessors with processor-based TLBs

To load the translation information for a page into a TLB entry, a trial frame number is obtained from the page table, the GC is obtained from the page-frame table, and then the frame number (along with other translation information) is reread from the page table. The process is repeated until this frame number equals the trial frame number (which ensures that the translation information is not being changed), at which point the GC and translation information are loaded into the TLB.

Employing Validation, the cost of managing processor-based TLBs is $S\,T_{stale} + R\,T_{reload},$ where S is the number of times stale TLB entries are used and T_{stale} is the time required for the invalid memory request, *i.e.*, one round trip through the network. R is the number of TLB reloads and T_{reload} is the average processor time expended to execute Validation's TLB-reload routine. In our simulations, T_{reload} for an AMD29000 [3] consists of 18 instructions including four or five single-word loads of noncacheable data [26]. The extra load occurs if the reload is triggered by the use of a stale entry. The cost assigned to network traversals (*e.g.*, those required for T_{stale} and T_{reload}) include estimates of network queueing delays (see section 4).

As one might think, the GCs are susceptible to wraparound. As discussed in [28], there are several ways of dealing with this problem, *e.g.* using the TLB-shootdown algorithm [4] to invalidate TLB entries associated with the related frame or associating multiple counters with each frame.

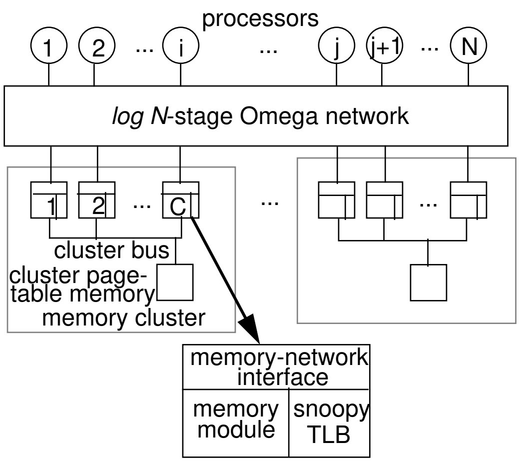

Figure 3: Multiprocessor with memory-based TLBs

3.2: Memory-based TLBs

In a memory-based TLB system, the virtual, rather than the physical address is transmitted to the targeted memory module, where the virtual-to-physical address translation occurs. Thus, the virtual address, in particular, the page number, must identify the targeted memory cluster. As shown in Figure 3, a TLB is located at each memory module and the memory modules comprising a memory cluster define a paging arena (to which the mapping of a page necessarily is fixed). To maintain consistency among a cluster's TLBs and its page table (which contains entries for all pages that can be mapped to the cluster), the TLBs and the memory storing the page table are interconnected by a bus on which each snoops (see *e.g.*, [11]).

In our simulated systems TLBs are lockup free [14], thus, a TLB miss does not block subsequent TLB hits and related memory accesses; only subsequent TLB misses are blocked. In this case, the cost of managing memory-based TLBs is $R_1C_{reload1}+R_2C_{reload2}+T_{busy\text{-}TLB}+T_{busy\text{-}bus}$, where: R_1 and $C_{reload1}$ are the number of TLB reloads satisfied by other TLBs in a cluster and the cost of each. R_2 and $C_{reload2}$ are the number of TLB reloads satisfied by a cluster's page-table memory and the cost of each. $C_{reload1}$ and $C_{reload2}$ are contention-free TLB reload costs, *i.e.*, they include the time that a TLB miss may wait for servicing due to the targeted TLB being busy servicing a previous TLB miss ($T_{busy\text{-}TLB}$) or the cluster bus being busy as a result of other intracluster TLB misses ($T_{busy\text{-}bus}$). $C_{reload1}$ and $C_{reload2}$ were computed by constructing a high-level hardware design of the memory module [26]: $C_{reload1}$ is five processor cycles and $C_{reload2}$ is seven processor cycles. $T_{busy\text{-}TLB}$ and $T_{busy\text{-}bus}$ are computed by our simulator, which is described briefly in the next section.

3.3: Processor- vs. memory-based TLBs

The performance differences between comparable multiprocessor systems with processor- and memory-based TLBs are attributable to the number of TLB reloads generated (R) and the average cost of a TLB reload (T_{reload}). The number of TLB reloads generated by processor- and memory-based TLB systems are quite different for three reasons.

1. In general, a processor-based TLB is accessed by only one process during a given time interval. (This is not the case when fast context switching among multiple processes is implemented.) In contrast, a memory-based TLB is accessed by multiple processes during a given time interval. As a result, memory-based TLBs are subject to *TLB-entry thrashing*, where a TLB entry required by one process replaces a TLB entry being used by another.
2. Data sharing during a given time interval among processes executing on P different processors results in P compulsory processor-based TLB reloads of the same page table entry. In contrast, such data sharing results in one compulsory memory-based TLB reload.
3. Multiple references to a page by one processor during a given time interval result in one compulsory processor-based TLB reload. In contrast, due to interleaving of each page across a cluster, these references can result in C compulsory memory-based TLB reloads, where C is the number of memory modules in a cluster.

4: Simulation methodology and benchmarks

We used trace-driven simulations to quantitatively compare the performance of processor- and memory-based TLBs and, like [1], we vary the system size. Address traces of three double-precision, floating-point intensive, scientific benchmark applications were used to drive the simulations. Each benchmark was used to run 18 simulations: six simulations of processor-based TLB systems for $N = 2^n$, $n = 1, ..., 6$; six simulations of memory-based TLB systems with one paging arena for $N = 2^n$, $n = 1, ..., 6$; and six simulations of memory-based TLB systems with two paging arenas for $N = 2^n$, $n = 1, ..., 6$. A simulation includes one iteration of the initialization, computation, and termination sections of a program. A production run would include many iterations of the computation section. Since even a single iteration of the computation section dominates a simulation, whether we simulate one or multiple iterations of the computation section, our results exhibit the same trends.

4.1: Trace generation and simulator

We simulate multiprocessors of different sizes and different architectures with just one augmented uniprocessor address trace [26]. In order to generate a trace, PSIMUL [24], which is based on an instruction-level simulator called SEMUL that was developed at IBM Research by A. Liles and G. Paul, was applied to the VM/EPEX parallel programming environment [9; 25] version of a parallel program. Instead of having PSIMUL produce a multiprocessor address trace, which would result in a fixed scheduling of parallel tasks to processors that is beyond our control, we modified PSIMUL to generate a uniprocessor trace augmented with markers identifying individual tasks (*i.e.*, basic scheduling units), precedence relationships among tasks, task groups (a set of tasks, not necessarily maximal, sharing the same precedence relationship), barriers, and user-specified program regions (*e.g.*, the initialization or termination section of the program). Our multiprocessor simulator, SIM, takes the augmented trace as input and uses the markers to control the scheduling of tasks to processors, consistent with their precedence relationships. Marker-based techniques have also been used by So, *et al.* [23] to implement SPAN, which analyzes a process flow graph and provides information to assist the user in attaining maximal speedup, and by Chaiken, *et al.* [5] to study the performance of shared-data caches in large-scale multiprocessors.

SIM maintains the states of TLBs and virtual-to-physical address translation information so that TLB miss rates and TLB reload time can be measured. To calculate the execution time of tasks, input parameters to SIM define the service times for instruction execution, cache and TLB miss servicing, loads and stores of noncacheable (shared) data, and context switches. SIM does not simulate consistent general-purpose caches, therefore, as mentioned in section 2.0, we assume that references to shared read/write pages are not cacheable, while references to private and shared read-only pages are cacheable.

A memory access generated by a cache or TLB read miss or by a load or store of noncacheable data is assigned a service time that includes network transit and memory access time and delays the issuing processor until it completes. Cache write misses and write-throughs do not delay the processor and, thus, are not assigned service times. However, these network accesses are included when calculating the load offered to the network (and, hence, network delay).

The offered load (averaged over the entire run) is computed before SIM begins as follows (*cf.*, [19]). Given input parameters that describe the multiprocessor architecture, the estimated mix of instructions and data, the percentage of loads and stores, the hit rates for the instruction caches, data caches, and TLBs, and the average size of an event (which is defined as a noncacheable reference and the cacheable references that precede it), we estimate the number of network requests generated by an average-size event. This allows us to determine the total network load, *i.e.*, the total number of packets sent for all cache and TLB misses and noncacheable references. We then calculate the total run time assuming (temporarily) that there is no network queueing delay. The quotient gives an initial estimate for the offered load, from which a revised network queueing delay is calculated using formulas of [15] and [17]. This new estimate of the queueing delay gives a new estimate of the offered load and the process is repeated until convergence. The final queueing delay obtained is used during the simulation.

4.2: Benchmarks

Our simulations were driven by address traces of three parallel applications. These applications, Peskin [18], Simple [6], and Weather [12; 13], are double-precision, floating-point intensive, scientific applications, which employ iterative grid-based algorithms. *Peskin* is a simulation of incompressible flow of a fluid coupled to an elastic membrane, *i.e.*, simulation of a boundary behavior in a periodic square domain filled with fluid. *Simple* models the hydrodynamic and thermal behavior of fluids in two dimensions. *Weather i*s a weather forecasting code that given the state of the atmosphere, *i.e.*, the values of certain specified meteorological quantities, at time t, calculates the state at time $t + \Delta t$. (See [7] for more details concerning these applications.)

These programs were written using the SPMD (Single-Program Multiple-Data) computational model [8] and were ported to the VM/EPEX parallel programming environment. System limits restricted a program's shared data to 15 Mbytes, thus, the problem sizes used are smaller than desired: for Peskin there are 4.5 Mbytes of shared data (for a 256 x 256 grid with 512 boundary points), 4.5 Kbytes of private data per process, and 650 Kbytes of code; for Simple there are 10 Mbytes of shared data (for a 256 x 256 grid), 1 Kbyte of private data per process, and 100 Kbytes of code; and for Weather there are 12 Mbytes of shared data (for a 108 x 72 x 9 grid), 400 Kbytes of private data per process, and 128 Kbytes of code. The number of references to shared read/write pages are approximately 14.5 million for Peskin, 21 million for Simple, and 35 million for Weather. The number of references to private and shared read-only pages are much larger than the number of references to shared read/write pages (approximately 10 times as many for Simple and Weather, and over 30 times as many for Peskin).

4.3: Trace compression

The traces used in this study were compressed in such a way that noncacheable references (to shared read/write pages) are fully described, while only the number of cacheable references (to private and shared read-only pages) that occur between two noncacheable references is recorded. In accounting for the time expended to process cacheable references, we assume that 75% of cacheable data references are loads and 25% are stores (see, *e.g.*, [2]), and that the (write-through) data caches, instruction caches, and associated TLB references have hit rates of 97%, 99%, and 99%, respectively [22 and 10].

5: Simulation results

Our comparison of processor- and memory-based TLBs is based on the TLB miss rates, average TLB reload costs, and simulated execution times of different size multiprocessors reported by our simulations. Our simulation results are discussed w.r.t. TLB miss rates in section 5.1, w.r.t. TLB reload costs in section 5.2; and w.r.t. simulated execution times in section 5.3.

5.1: TLB miss rates

Figures 4, 5, and 6 illustrate the growth of the system-wide TLB miss rate with *N*. In all cases, *i.e.*, for the Peskin, Simple, and Weather simulations:

1. the processor-based TLB miss rate increases (albeit very slowly) with *N*,
2. for a fixed number of paging arenas, the memory-based TLB miss rate tends to increase with *N*,
3. the processor-based TLB miss rate is smaller than the TLB miss rate for memory-based TLB systems with one paging arena, and
4. increasing the number of paging arenas from one to two, decreases the memory-based TLB miss rates, producing miss rates smaller than processor-based TLB miss rates for systems with *N*<16.

As mentioned above in section 3.3, the processor-based TLB miss rate growing with *N* may be attributable to PEs accessing the same shared pages. For example, consider comparable systems with one and *P* processors, and a parallel task group consisting of *P* tasks that operates on a shared array, each task in the group operating on a different row of the array. If the parallel task group is executed by the one-processor system, it is conceivable that a page-table entry for a page that stores elements of several rows of the shared array will be loaded into the processor's TLB and then will be used by multiple tasks in the group. If the same parallel task group is executed by the *P*-processor system, multiple reloads of the page's page-table entry will be required, the number being equal to the number of rows stored in the page. Pages containing synchronization variables and shared data structures would cause such a

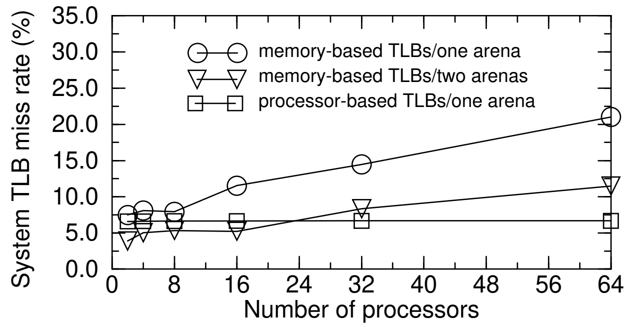

Figure 4: Peskin: system-wide TLB miss rates

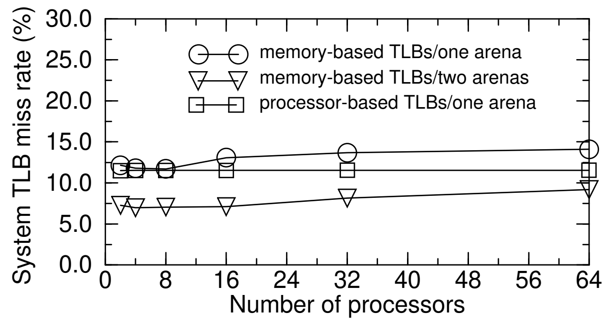

Figure 5: Simple: system-wide TLB miss rates

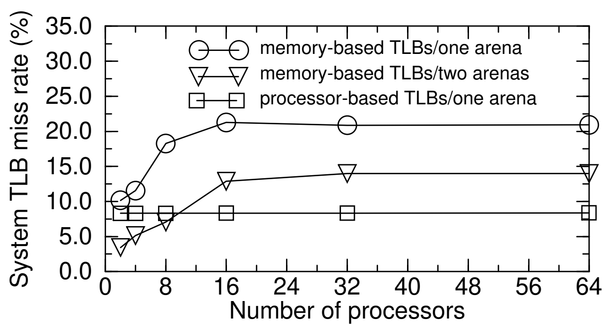

Figure 6: Weather: system-wide TLB miss rates

situation to arise.

The memory-based TLB miss rate generally increases with *N* for two reasons.

1. *C*, the number of MMs in a cluster, increases with *N*. As *C* increases, the number of TLBs at which a page's translation information can be replicated, and thus, due to spatial locality of reference, the number of TLB misses increases.
2. TLB-entry thrashing, which occurs in memory-based TLBs due to multiple processes accessing shared TLBs for different address translation information, is likely to increase with *N*.

An advantage of processor-based TLBs w.r.t. the number of generated misses (especially for larger *N*) is that the large majority of pages are referenced by only one processor during a given time interval and a page referenced by one processor produces one compulsory processor-based TLB reload, whereas it produces as many as *C* compulsory memory-based TLB reloads. In addition, in a system with processor-based TLBs only one process is accessing a TLB, whereas in a system with memory-based TLBs multiple processes are accessing a TLB and, thus, it is more likely that an TLB entry will be evicted before its use is complete.

The TLB miss rate for two paging arenas is less than that for one paging arena for two reasons.

1. As the number of paging arenas increases, the number of TLBs per cluster and, hence, the replication of translation information and the number of compulsory TLB misses decreases. For example, with one paging arena a page is interleaved across *N* memory modules and, thus, if the references to the page span across the *N* memory modules, *N* related compulsory TLB misses occur. With two paging arenas a page is interleaved across half as many memory modules, thus, *N*/2 compulsory TLB misses occur.

2. As the number of paging arenas increases, the number of pages that map to an arena decreases. Accordingly, a TLB is used to cache translation information for a decreasing number of pages. This reduces competition for memory-based TLB entries and, thus, leads to a reduction in TLB-entry thrashing and TLB misses.

The reduction in memory-based TLB miss rate due to increasing the number of paging arenas from one to two is about 32-55% for the Peskin simulations, 34-46% for the Simple simulations, and 33-66% for the Weather simulations. But, comparing TLB miss rates alone does not indicate the relative performance of processor- and memory-based TLBs.

5.2: TLB management overhead

Memory-based TLB miss rates being greater than processor-based TLB miss rates does not necessarily mean that the cost of memory-based TLB management is also greater. This is because the cost of a processor-based TLB reload is proportional to *log N* due to network transit, while the cost of a memory-based TLB reload is relatively small if TLB servicing is not delayed significantly by busy TLBs and cluster buses. As depicted in Figures 7, 8, and 9, the memory-based TLB (weighted) average reload overhead for systems with one paging arena for N < 32 and systems with two paging arenas for *N* < 64 is consistently less than for processor-based TLB reload overhead for comparable systems, even though in many of these cases the processor-based TLB miss rate is less than the memory-based TLB miss rate.

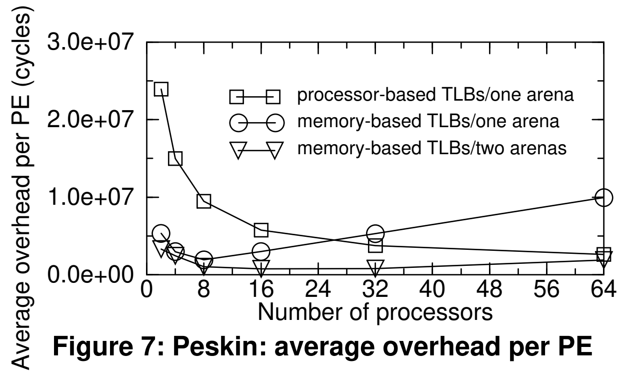

Figure 7: Peskin: average overhead per PE

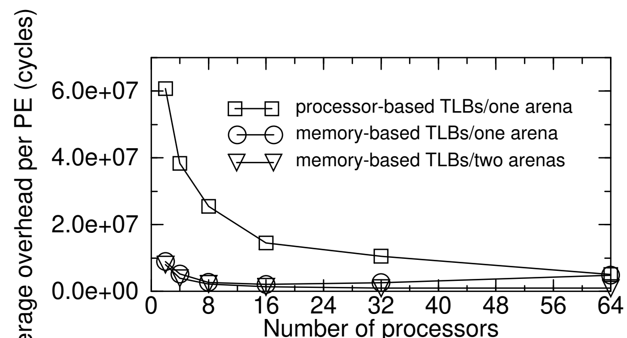

Figure 8: Simple: average overhead per PE

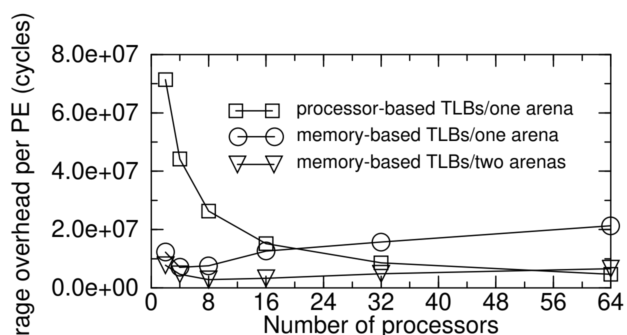

Figure 9: Weather: average overhead per PE

As indicated in Figures 7, 8, and 9, the performance benefit due to memory-based TLBs depends on the number of paging arenas and *N*. In the previous section we observed that for a fixed *N* as the number of paging arenas increases from one to two, the TLB miss rate decreases. As a result, as exemplified in Figure 10, contention for TLBs (TLB Busy) and cluster buses (Bus Busy) decrease and, therefore, as depicted in Figures 7, 8, and 9, the TLB reload overhead decreases.

TLB reload cost (TLB Miss in Figure 10) is small and tends to remain relatively constant or decrease as *N* increases. For a fixed number of clusters, increasing *N* increases the number of TLBs in a cluster, which increases the probability of a TLB miss being serviced by a intracluster TLB (which is less expensive than being serviced by the page-table memory). In contrast, as depicted in Figure 10 the other components of the cost of memory-based TLB reloads, *i.e.*, $T_{busy\text{-}TLB}$ and $T_{busy\text{-}bus}$, increase with N. This is expected since as N increases, so does the TLB miss rate and, as a result, TLB and cluster bus contention increase as well.

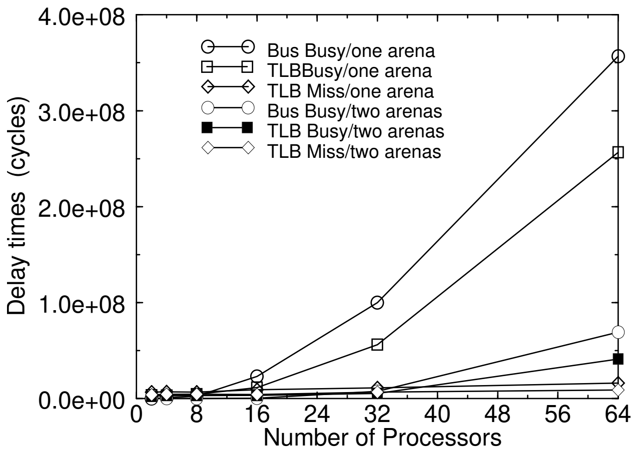

Figure 10: Peskin: components of memory-based TLB overhead

As N increases, the memory-based TLB reload overhead approaches the processor-based TLB reload overhead from below, suggesting that for best performance the number of arenas in memory-based TLB systems should grow with N (experiments we intend to perform). It is not yet clear how the number of arenas should grow with N since as the number of arenas increases, network and memory contention may also rise because a large data structure that is frequently accessed will be distributed among a smaller number of memory modules. Multiple paging arenas in a processor-based TLB system would not change the TLB miss rate and should not significantly change the cost of TLB management.

5.3: Simulated execution time

Optimally the TLB reload overhead would remain constant as N increases and would be evenly distributed among the N processors in the system, affording perfect parallelism and speedup. As depicted in Figures 7-9, for the studied benchmarks the processor- and memory-based TLB average reload overhead first decreases as N increases (for small N); but for larger N, it increases with N and, thus, does not afford perfect parallelism, if the application also allows it. To understand the effect of the TLB reload overhead on system performance we examine the reload overhead w.r.t. simulated execution time.

Average TLB reload overhead as a percentage of simulated execution time varies. As an example, for

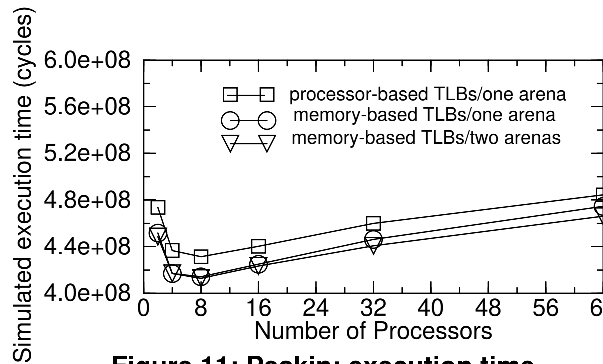

Figure 11: Peskin: execution time

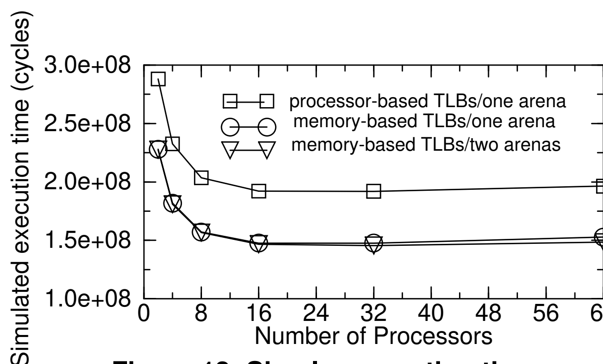

Figure 12: Simple: execution time

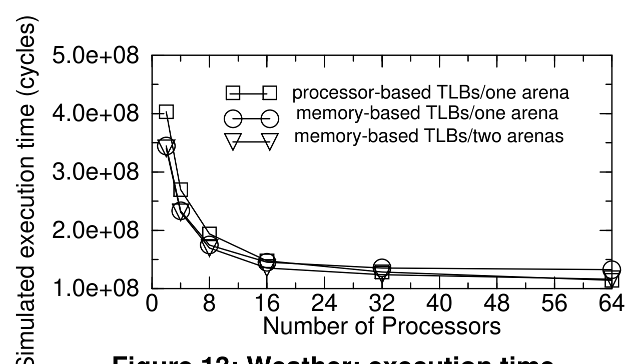

Figure 13: Weather: execution time

Weather, it varies from 1.7% to 16%. Specifically, for memory-based TLB systems with one paging arena it varies from 3.6% for N=2, down to 3.0% for N=4, and up to 16% for N=64. For memory-based TLB systems with two paging arenas it varies from 2.3% for N=2, down to 1.7% for N=16, and up to 5.7% for N=64. For processor-based TLB systems it varies from 17.7% for N=2 down to 4.1% for N=64. Optimally the average overhead would decrease as 1/N. In our study, processor-based TLB systems approach this behavior but still do not outperform memory-based TLB systems. This is due primarily to the fact that the cost of a memory-based TLB reload can be made smaller than the cost of a processor-based TLB reload by increasing the number of paging arenas.

Figures 11, 12, and 13 plot simulated execution time vs. the number of processors in the system. Comparing processor-based TLB systems with memory-based TLB systems that have two paging arenas, we observe that for memory-based TLBs in the Peskin simulations, the execution time is reduced by about 5%. For memory-based TLB systems that have one paging arena the gain is 2-5%. The Simple simulations show memory-based TLB system execution times being approximately 20% smaller than processor-based TLB system execution times, regardless of the number of paging arenas for systems with as few as 8 or 16 PEs. The Weather simulations show that for $N<64$ execution times of memory-based TLB systems with two paging arenas are from 3.7-16% smaller than processor-based TLB system execution times; for N<32 execution times of memory-based TLB systems with one paging arena are from 1.7-15% smaller.

Figure 11 also indicates that for the Peskin workload speedup is no longer achievable beyond $N=8$, in fact, a slowdown occurs. Although parallelism is exhausted after $N=8$, execution time continues to increase. This is due to the TLB reload cost increasing with N and an increase in TLB misses due to changes in task scheduling.

6: Conclusions and future research

We measured the performance of consistency-ensuring processor- and memory-based TLBs in a range of multiprocessor architectures. Although the location of TLBs at memory is unconventional, the results of our trace-driven simulation study suggest that using memory-based TLBs, rather than processor-based TLBs, can result in significant performance gains. Our simulations show that multiprocessors with memory-based TLBs having one paging arena perform better than multiprocessors with processor-based TLBs for $N < 32$, and that multiprocessors with memory-based TLBs having two paging arenas perform better than multiprocessors with processor-based TLBs for $N < 64$. These results suggest that memory-based TLBs will outperform processor-based TLBs for larger N, provided that memory is organized as multiple paging arenas, the number of which increases with N. Recall that the cluster size is bounded by the cluster-bus bandwidth and hence the number of arenas needs to be at least proportional to N asymptotically. We have begun experiments with three and more paging arenas to quantify the relative performance gain as the number of paging arenas increases and the effect that the number of paging arenas has on network and memory contention and I/O. It is not clear how the number of paging arenas should increase with N. For example, for systems with N paging arenas, even though there is no cluster bus contention, spatial locality of reference suggests that network and memory contention will increase with N because words of a page are not interleaved among multiple memory modules. Since our method for determining network queueing delays assumes a uniform distribution of memory module accesses, our current simulator does not detect this increased contention. We are currently working on modifying our simulator to allow us to model a dynamic load and, thus, dynamic network contention.

The size of memory-based TLBs are not constrained by the physical size of the processor chip, as are processor-based TLBs. Thus, realistically we could assume that memory-based TLBs are much larger than processor-based TLBs. Undoubtedly, this would decrease the memory-based TLB miss rate and the memory-based TLB management overhead, providing an increased performance advantage over processor-based TLBs. Shortly we will be simulating multiprocessors with larger memory-based TLBs.

We have run simulations of NUMA architectures with processor- and memory-based TLBs, and have observed results similar to those reported in this paper. A NUMA architecture decreases the average cost of a processor-based TLB reload, which becomes more significant as N increases. The execution time of both processor- and memory-based TLB systems is decreased if a NUMA, rather than a UMA, architecture is assumed since the cost of an average of 1/N of memory references is reduced as a result of not having to transmit the reference through the network.

We have also run simulations of demand-paged UMA architectures in order to validate the TLB consistency algorithms that we implemented. Again, we observed results similar to those reported in this paper.

As of yet, we have neither simulated multiprocessors with faster processors nor multiprocessors with consistent general-purpose caches. It is our belief that such simulations will produce results more favorable to memory-based TLBs. If faster processors are assumed, both processor- and memory-based systems will experience faster execution times and the cost of memory references will represent a larger overhead. Since processor-based TLB reloads require multiple memory references, this translates into a larger overhead for processor-based TLBs and a larger performance advantage for memory-based TLBs. The inclusion of consistent general-purpose caches in either processor- or memory-based TLB systems is likely to decrease the number of memory accesses. The number of processor-based TLB misses should not change since a processor-based TLB is accessed whether or not the referenced data is cache-resident. However, the number of memory-based TLB misses is likely to change since the number of memory-based TLB accesses would be reduced. Again the performance advantage would be in the favor of memory-based TLBs.

The address traces used in our simulation study were generated by the EPEX system, which limited the size of a program's shared data and, thus, limited the problem sizes. In turn, the problem sizes limited the maximum available parallelism and, thus, the number of processors that could cooperate in program execution. We are moving to another simulation environment which will permit the study of larger problem sizes and higher levels of parallelism.

In considering memory-based TLBs, the reader should keep in mind that multiple paging arenas restrict the mapping of a page to the set of frames in a particular memory cluster. Thus, in a system with a global virtual memory, where different pages can be mapped to the same frame, those pages must map to the same memory cluster.

Acknowledgments

We wish to thank colleagues at NYU's Ultracomputer Research Laboratory, IBM, and New Mexico State University for their help. In particular, we thank Jan Edler, Richard Kenner, Bryan Rosenburg, and Harold Stone for constructive criticism and good advice; Frederica Darema, Evelyn Melton, Gregory Pfister, Kimming So, and Jan Stone for their help with collecting address traces; and Qidong Xu for modifying our simulator and plotting the results.

References

[1] Agarwal, A., and M. Cherian, "Adaptive Backoff Synchronization Techniques," *Proceedings of the 16th International Symposium on Computer Architecture*, pp. 396-406, May/June 1989.

[2] Agarwal, A., and A. Gupta, "Memory-reference characteristics of multiprocessor applications under MACH," *Proceedings of the 1988 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pp. 215-255, May 1988.

{3] AMD, *Am29000 Streamlined Instruction Processor User's Manual*, 1987.

[4] Black, D., R. Rashid, D. Golub, C. Hill, and R. Baron, "Translation lookaside buffer consistency: a software approach," *Proceedings of the Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 113-122, Apr. 1989.

[5] Chaiken, D., C. Fields, K. Kurihara, and A. Agarwal, "Directory-based cache coherence in large-scale multiprocessors," *Computer*, **23**, 6, pp. 49-58, June 1990.

[6] Crowley, W., C. Hendrickson, and T. Rudy, "The Simple code," Lawrence Livermore Laboratory, Livermore, CA, Feb. 1978.

[7] Darema, F., A. Karp, P. Teller, "Applications Survey Reports - I," IBM Research Report No. RC12743, May 1987.

[8] Darema-Rogers, F., D. George, V. Norton, and G. Pfister, "A single-program-multiple-data computational model for EPEX/FORTRAN," IBM Research Report No. RC11552, Nov. 1986.

[9] Darema-Rogers, F., D. George, V. Norton, and G. Pfister, "VM/EPEX--a VM environment for parallel execution," IBM Research Report No. RC11225, Jan. 1985.

[10] Emer, J., and D. Clark, "A characterization of processor performance in the VAX-11/780," *Proceedings of the11th Annual International Symposium on Computer Architecture*, pp. 301-309, June 1984.

[11] Goodman, J., "Coherency for multiprocessor virtual address caches," *Proceedings of the Second International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 72-81, Oct. 1987.

[12] Kalnay-Rivas, E., and D. Hoitsma, "Documentation of the fourth order band model," NASA Modeling and Simulation Facility Laboratory for Atmospheric Sciences, NASA/Goddard Space Flight Center, Greenbelt, MD, Nov. 1979.

[13] Korn, D., and N. Rushfield, "Washcloth simulation of three-dimensional weather forecasting codes," Ultracomputer Note No. 55, N.Y.U. Courant Institute, Jan. 1981.

[14] Kroft, D., "Lockup-free instruction fetch/prefetch cache organization," *Proceedings of the 8th Annual International Symposium on Computer Architecture*, pp. 81-88, June 1981.

[15] Kruskal, C., M. Snir, and A. Weiss, "The distribution of waiting times in clocked multistage interconnection networks," *Proceedings of the 1986 International Conference on Parallel Processing*, pp. 12-19, Aug. 1986, and *IEEE Transactions on Computers*, **37**, 11, pp. 1337-1352, Nov. 1988.

[16] Lee, F., "Study of 'look aside' memory," *IEEE Transactions on Computers*, **C-18**, 11, pp. 1062-1064, Nov. 1969.

[17] Liu, Y-S., *Architecture and Performance of Processor-Memory Interconnection Networks for MIMD Shared Memory Parallel Processing Systems*, Doctoral Dissertation, N.Y.U., Courant Institute, Sept. 1990.

[18] Peskin, C., "Numerical analysis of blood flow in the heart," *Journal of Computation Physics*, **25,** pp. 220-252, 1977.

[19] Pfister, G., and V. Norton, "'Hot spot' contention and combining in multistage interconnection networks," *IEEE Transactions on Computers*, **C-34**, 10, pp. 943-948, Oct. 1985.

[20] Satyanarayanan, M., and D. Bhandarkar, "Design trade-offs in VAX-11 translation buffer organization," *Computer*, **14**, 12, pp. 103-111, Dec. 1981.

[21] Smith, A., "CPU cache consistency with software support and using 'one time identifiers'," *Proceedings of the Pacific Computer Communications Symposium*, pp. 142-150, Oct. 1985.

[22] Smith, A., "Cache evaluation and the impact of workload choice," *Proceedings of the 12th Annual International Symposium on Computer Architecture,* pp. 64-73, June 1985.

[23] So, K., A. Bolmarcich, F. Darema, and V. Norton, "A speedup analyzer for parallel programs," Proceedings of the 1987 International Conference on Parallel Processing, pp. 653-662, Aug. 1987.

[24] So, K., F. Darema-Rogers, D. George, V. Norton, and G. Pfister, "PSIMUL--a system for parallel simulation of the execution of parallel programs," IBM Research Report No. RC11674, Jan. 1986.

[25] Stone, J., F. Darema-Rogers, V. Norton, and G. Pfister, "The VM/EPEX FORTRAN preprocessor reference," IBM Research Report No. RC11408, Sept. 1985.

[26] Teller, P., *Translation-Lookaside Buffer Consistency in Highly-Parallel Shared-Memory Multiprocessors*, Doctoral Dissertation, N.Y.U., Courant Institute, May 1991.

[27] Teller, P., "Translation-lookaside buffer consistency," *Computer*, **23**, 6, pp. 26-36, June 1990.

[28] Teller, P., R. Kenner, and M. Snir, "TLB consistency on highly-parallel shared-memory multiprocessors," *Proceedings of the 21st Hawaii International Conference on System Sciences*, pp. 184-193, Jan. 1988.