TLB Performance in Multiprocessors

by

Patricia J. Teller[1]

Allan Gottlieb[2]

Ultracomputer Note #173

September, 1991

ABSTRACT

This paper compares the performance, in highly-parallel shared-memory multiprocessors, of locating translation-lookaside buffers (TLBs) at processors with that of locating TLBs at memory. Our performance comparison is based on results of 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 better than processor-based TLBs, provided that memory is organized as multiple *paging arenas*, where the mapping of pages to arenas is fixed. The cost of a processor-based TLB reload is at least $\log N$ because of network transit. The cost of a memory-based TLB reload can be made smaller than that of a processor-based TLB reload, since network transits are not required. Furthermore, with multiple paging arenas, the number of reloads is smaller with memory-based TLBs. For memory-based TLBs to continue to outperform processor-based TLBs for large N, it is likely that the number of paging arenas must grow with N.

---

[1]Supported by U.S. Department of Energy grant number DE-FG02-88ER25052, National Science Foundation grant number MDS-8301768, and the IBM Corporation.

[2]Supported in part by U.S. Department of Energy grant DE-FG02-88ER25052 and National Science Foundation grant number MIP-8915488.

1. Introduction

Computers supporting virtual memory normally use *translation-lookaside buffers (TLBs)* [Lee, 1969], special-purpose virtual-address caches, to store and provide processors with fast access to recently-referenced address translation information [Satyanarayanan and Bhandarkar, 1981]. Virtual-to-physical address translation information is cached in TLBs and stored in data structures, called *page tables*. A TLB conventionally is located in a processor, but could be present anywhere on the path from the processor to memory. In particular, as suggested by Teller, Kenner, and Snir [1988], in systems with virtual-address caches, TLBs can be located at memory modules.

We compare the performance of TLBs located at processors (processor-based TLBs) with that of TLBs located at memory (memory-based TLBs) in shared-memory multiprocessors with log$N$-stage networks interconnecting N processors and N memory modules. We maintain consistency among TLB and page-table entries using two methods proposed by Teller, Kenner, and Snir [1988], ‘‘Validation’’ for processor-based TLBs and ‘‘Memory-based TLBs’’ for, as the name implies, memory-based TLBs. Our performance comparison is based on the results of a trace-driven simulation study, which is described briefly herein and is explained in detail in [Teller 1991], where other consistency algorithms are studied as well.

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 [Teller 1990] for a description of the problem and a survey of solutions). We call a page-table modification that can result in TLB inconsistencies an *unsafe change*. An unsafe change results from invalidating a virtual-to-physical memory mapping, increasing page protection (*e.g.*, from read-write to read-only), or changing the mapping of a page to a frame. When memory sharing patterns are unrestricted, the TLB consistency problem is defined in its fullest generality. As noted below, memory-based TLBs with multiple paging arenas does not solve the fully general problem.

Methods for ensuring TLB consistency in highly-parallel multiprocessors have been proposed by Black, *et al.* [1989], Rosenburg [1989], and Teller, Kenner, and Snir [1988]. Black, *et al.* [1989] and

Rosenburg [1989] study the performance of their TLB consistency algorithms in real multiprocessor systems with processor-based TLBs. Black, *et al.* measure the performance of TLB Shootdown on a 16-processor NS32332 Encore Multimax that interconnects processors and memory via a shared bus. Rosenburg compares the performance of his modified TLB Shootdown with that of the original algorithm on IBM's 64-processor RP3 [Pfister, *et al.*, 1985], a NUMA (non-uniform memory access) multiprocessor that interconnects processors and remote memory modules via a $\log N$-stage network.

Two questions not answered by this related research, but answered in the present paper are: In a shared-memory multiprocessor with a $\log N$-stage network interconnecting N processors and N memory modules, such as the RP3, BBN Butterfly [BBN, 1985], CEDAR [Gajski, *et al.*, 1983], and the NYU Ultracomputer [Gottlieb, 1987],

(1) How does the total cost of managing processor-based TLBs (including reloads not related to consistency management) grow with N?

(2) How does the performance of processor-based TLBs compare with that of memory-based TLBs?

We use trace-driven simulations of multiprocessors having a range of values for various architectural parameters (*e.g.*, number of processors, TLB size, etc.) to answer these questions and to study the performance of TLB consistency solutions proposed by Teller, Kenner, and Snir [1988]. Due to space limitations, we do not present all the relevant data produced by these simulations, which can be found in [Teller, 1991]. We believe that, as with other caches, TLB performance can be studied well by trace-driven simulations. (See [Agarwal and Cherian, 1989], [Chaiken, *et al.*, 1990], [Eggers and Katz, 1988], and references therein for other multiprocessor cache studies.)

We follow Knuth [1976] in our use of asymptotic orders and write $f = \mathrm{O}(g)$ if there exist N and $C > 0$ such that, for all $n > N$, $f(n) < Cg(n)$. We write $f = \Omega(g)$ if $g = \mathrm{O}(f)$ and write $f = \Theta(g)$ if $f = \mathrm{O}(g)$ and $f = \Omega(g)$. Roughly speaking, $f = \mathrm{O}(g)$ (resp. $f = \Omega(g)$, $f = \Theta(g)$) can be read as: ''for large n, $f(n)$ is at most (resp. at least, approximately) proportional to $g(n)$''.

The cost of managing TLBs grows as $R \times T_{reload}$, where R is the number of TLB reloads and T_{reload} is the average cost of a reload. (A TLB reload, which is triggered by a TLB miss, loads translation information into a TLB.) Since the cost of a processor-based TLB reload is $\Omega(logN)$, the cost of managing processor-based TLBs grows at least as $R\log N$. With respect to the cost of managing memory-based TLBs, our simulations indicate the following:

(1) In systems with memory-based TLBs, both R and the cost of a TLB reload decrease with the number of paging arenas. The memory modules are partitioned into equal-size *paging arenas*, to which the mapping of pages is fixed, *i.e.*, all frames to which a single page can be mapped reside in one arena.

(2) For large N, memory-based TLBs with multiple paging arenas outperform processor-based TLBs, which in turn outperform memory-based TLBs with one paging arena.

This paper, which is based on Teller's dissertation [1991], is organized into five additional sections: Section 2 describes the generic multiprocessor model that was simulated. Section 3 describes additions to the model required for TLB consistency. Section 4 outlines our simulation methodology. Section 5 presents results of our simulation study and compares the performance of processor-based TLBs with that of memory-based TLBs. Section 6 summarizes our findings, draws conclusions, and discusses future research.

2. Multiprocessor Model

In this section we describe our generic multiprocessor model. As depicted in Figure 1, the multiprocessor interconnects N processors and a shared main memory, partitioned into N memory modules, via a packet-switched $\log N$-stage Omega network composed of 2×2 switches that have a cycle time equal to that of a processor. We use an idealized 10 MIPS processor with a Harvard architecture, which allows one instruction and one data reference to be generated at each cycle. Memory modules are equidistant from each processor and are partitioned into equal-size groups called *memory clusters*. Main memory is interleaved across each cluster on a double-word basis, and the memory access delay is two switch cycles. (Queueing delays resulting from memory temporarily receiving a higher request rate than it can service are not modeled. Since the heaviest offered load encountered was about .1 requests per cycle—most were much less—and our modeled memory can satisfy .5 requests per cycle, these queueing delays would be small.)

Figure 1. Generic Multiprocessor Model.

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. We assume private data, instructions, and address translation information for cacheable pages are cacheable, while shared data, including address translation information for shared pages, are not. A cache miss generates two double-word load requests (a cache line is 16 bytes). The cost of a TLB miss on referencing noncacheable data is discussed below. Since the applications studied were double-precision floating-point intensive, we modeled data references as double-word loads or stores.

The virtual address space is composed of global (system-wide) segments, each of which is divided into 4K-byte pages and is the smallest portion of the address space that can be shared among processes. With global segments, *i.e.*, a global address space, concurrent processes never use the same virtual address to refer to different objects. We define a global page number to be the composite of a global segment number and a segment page number, and refer to global page numbers simply as page numbers.

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 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,
TLB size (16, 32, or 64 entries),
physical memory size (33% or 50% of virtual memory size), and
number of memory clusters (one, two, or N).

N ranges from 1 to 128. Processor-based TLB systems were simulated with one memory cluster, while memory-based TLB systems were simulated with one memory cluster, with two of equal size, and with *N* of equal size. For memory-based TLBs, each cluster is a paging arena. The sizes of TLBs and physical memory and the consistency algorithms are discussed below.

The traces used in this study do not include operating system references and we have concentrated our attention on references to shared data. We assume that pages storing cacheable (*i.e.*, private or read-only) data and instructions are locked in memory, while pages storing noncacheable (*i.e.*, shared) data are managed using demand paging with a strict LRU replacement policy. We simulate TLBs with 16, 32, or 64 entries for noncacheable pages. TLBs are not flushed on context switches and are modeled as fully associative with strict LRU replacement.

Modifications of page-to-frame mappings caused by demand paging generate unsafe changes. We vary the frequency of unsafe changes by changing the size of that part of physical memory allocated to noncacheable pages (and, hence, the related paging rate). The number of frames in this part of physical memory is equal to either 33% or 50% of the number of pages allocated to noncacheable pages. Henceforth, these portions of physical and virtual memory are referred to simply as physical and virtual memory. Page tables are not paged.

In memory-based TLB systems, each memory cluster is defined to be a *paging arena, i.e.*, a region of main memory to which the paging algorithm is applied independently. The virtual address, in particular, the page number, must identify the targeted memory cluster. Consequently, the set of frames that may be used to store a page are restricted to the frames within a specific memory cluster.

When a process incurs a page fault for which no other process has begun the necessary page-in, the process becomes the pager for that page and performs the page-in. This may include evicting a resident page and writing a dirty page to disk prior to reading the faulted page into memory. After starting the final I/O, the pager inserts itself on the queue of processes waiting for the page. When a process incurs a page fault and the related page-in is in progress, then if the page will not be resident within twice the time it takes to context switch, the process inserts itself on the queue of processes waiting for the page. When a page becomes resident, its pager makes runnable all processes waiting for it. When a process becomes blocked waiting for I/O, the blocked processor context switches to another process.

To limit the impact of I/O, the number of I/O devices per paging arena is treated as infinite, *i.e.*, page I/O can commence at any time. We assume that a page can be located and transferred in 25 ms., a start-I/O costs 1000 cycles, and a context switch costs 100 cycles (which is reasonable for an efficient user-mode run-time scheduler).

3. Modeling TLB Consistency

The cost of maintaining TLB consistency for processor- and memory-based TLBs was determined by modeling the Validation and Memory-based TLBs algorithms, respectively. These algorithms are described below; see Teller, Kenner, and Snir [1988] for more details.

3.1. Processor-based TLBs

Validation allows stale entries to reside in processor-based TLBs but, employing a form of optimistic recovery [Strom and Yemini, 1985], detects an obsolete entry upon its use to access memory. To implement Validation, each memory cluster has a *validation table* with an entry for each of its frames. To identify stale entries, two generation counts (GCs) are associated with each frame: one stored in the page-frame table and the other stored in the validation table of the frame's memory cluster. In addition, GCs are cached in TLBs when the translation information for a page is loaded. These GCs are transmitted with main-memory requests (possibly requiring larger packets) and are validated atomically at the targeted memory module by comparing the transmitted GC with the one stored in the validation table (a cluster's validation table is replicated at each of its memory modules). If the GCs are equal, the requested memory access is performed; otherwise the TLB entry is stale and must be reloaded, after which the memory request is retried.

A processor desiring to make an unsafe change to the translation information of a resident page multicasts a message to the memory modules of the associated cluster to increment the GC in the cluster's (replicated) validation table. After the unsafe change has been completed, the GC stored in the page-frame table is updated. 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 assures 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

$$OH_{Val} = ST_{stale} + R_{Val}\,T_{reload},$$

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_{Val} is the number of TLB reloads and T_{reload} is the average processor time expended in an N-processor system to execute Validation's TLB-reload routine, which for the AMD29000 [AMD, 1987] consists of 18 instructions and four or five single-word loads of noncacheable data [Teller, 1991]. 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).

Read-locked TLBs, another method of ensuring TLB consistency proposed by Teller, Kenner, and Snir [1988], can be employed in systems with either processor- or memory-based TLBs. Its performance in processor-based TLB systems is similar to that of Validation [Teller, 1991].

3.2. Memory-based TLBs

In a memory-based TLB system, the virtual, rather than the physical, address is transmitted to memory (possibly requiring larger packets). To ensure TLB consistency, a memory-based TLB is designed similar to a snoopy cache (*e.g.*, as proposed in Goodman [1987]). The TLBs associated with each memory module of a memory cluster and the cluster's page table (with entries for all pages that can be mapped to the cluster) are interconnected by a bus that employs a snoopy-cache protocol to prevent stale TLB entries. Since a cluster's TLBs are interconnected by a bus, the number of memory modules in the cluster, C, is limited by the bandwidth of the bus. Hence, for scalability $\frac{N}{C}$, the number of clusters, should be proportional to N.

We initially assume that TLBs are blocking caches, *i.e.*, a TLB miss blocks the memory module until the miss is satisfied. Accordingly, the cost of managing memory-based TLBs is

$$OH_{Mem} = R_1 \times C_{reload_1} + R_2 \times C_{reload_2} + T_{busy-TLB} + T_{busy-bus} + T_{pgexcpt},$$

where: R_1 and C_{reload_1} are the number of TLB reloads satisfied by other TLBs in a cluster and the cost of each. R_2 and C_{reload_2} are the number of TLB reloads satisfied by a cluster's page-table memory and the cost of each. The total number of reloads is referred to as $R_{Mem} = R_1 + R_2$. C_{reload_1} and C_{reload_2} are *contention-free* TLB reload costs *i.e.*, they include neither the time that a memory access may wait for access to a TLB or cluster bus nor the extra access due to a stale TLB entry. These times are accounted for by $T_{busy-TLB}$, $T_{busy-bus}$, and $T_{pgexcpt}$, which are defined below. C_{reload_1} and C_{reload_2} were computed by constructing a high-level hardware design of the memory module [Teller, 1991]: C_{reload_1} is five processor cycles and C_{reload_2} is seven processor cycles.

$T_{busy-TLB}$ is the time that memory references are delayed while waiting for TLBs that are busy servicing TLB misses. $T_{busy-bus}$ is the time that TLB-miss servicing is delayed as a result of a busy cluster bus. $T_{pgexcpt}$ is the network transit time (including queueing delays) required to inform processors about access exceptions, *e.g.*, trying to write a read-only page. (Note that for processor-based TLBs there is no network transit required for exceptions.) $T_{busy-TLB}$, $T_{busy-bus}$, and $T_{pgexcpt}$ are computed by our simulator, *SIM*, which is described briefly in the next section. SIM simulates access to each TLB, the activity on each cluster bus, and paging activity; consequently, TLB accesses are delayed while a TLB is busy, TLB reloads are delayed while the cluster bus is busy, and memory accesses are not satisfied if an exception occurs.

4. Simulation Methodology

We simulate multiprocessors having a range of values of various architectural parameters with just one augmented uniprocessor address trace for each parallel program. We studied two double-precision, floating-point intensive, scientific applications, called Peskin and Simple, which employ iterative grid-based algorithms. The computation intensive portion of each program consists of a large section of code executed repeatedly until convergence is achieved. The Peskin program is based on an algorithm in Peskin [1977] for studying blood flow through heart valves. It is a 2D simulation of an incompressible fluid coupled to an elastic membrane. The program iteratively recomputes the position of the membrane and the velocity of the fluid for successive discrete times. The Simple program (*cf.*, [Crowley, Hendrickson, and Rudy, 1978]) is a benchmark physics code that models the hydrodynamic and thermal behavior of fluids in two dimensions. The equations of inviscid compressible hydrodynamics and simple heat conduction are solved by finite difference methods.

Lubachevsky [1981] produced parallel versions of each program using the WASHCLOTH multiprocessor simulator [Gottlieb, 1980]. We ported the programs to the VM/EPEX parallel programming environment [Darema-Rogers *et al.*, 1985; Stone *et al.*, 1985]. Both WASHCLOTH and EPEX support the SPMD (Single Program Multiple Data) computational model. Since EPEX limits shared data to 15 Mbytes, the problem sizes of Peskin and Simple are smaller than desired: for Peskin there are 4.5 Mbytes of shared data (for a 256×256 grid with 512 boundary points), 4.5 Kbytes of private data per process, and 650 Kbytes of code; for Simple there are 11 Mbytes of shared data (for a 256×256 grid), 1 Kbyte of private data per process, and 100 Kbytes of code.

Applying PSIMUL ([So, et al. 1986]; see also [Liles and Paul]) to the EPEX versions of Simple and Peskin can produce a multiprocessor address trace. This technique would result in a fixed scheduling of parallel tasks to processors that is beyond our control. Instead, we instructed PSIMUL to simulate a uniprocessor and modified PSIMUL to generate traces 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 (for example, the initialization or termination section of the program). The augmented trace is then supplied to our multiprocessor simulator, *SIM*. The markers enable SIM to control the scheduling of tasks to processors, consistent with their precedence relationships. Accordingly, by simulating memory management, SIM detects and models page faults, including blocking the faulting tasks and switching the faulting processors to another task.

SIM maintains the states of TLBs, virtual-to-physical address translation information, and the paging subsystem associated with shared data so that TLB miss rates, TLB management time, and paging behavior can be measured. To calculate the execution time of tasks, service times are assigned to instruction execution, cache and TLB miss servicing, loads and stores of noncacheable (shared) data, context switches, page faults, and page-ins. These service times are defined by input parameters to the simulator.

Marker-based techniques have also been used by So, *et al.* [1987] 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.* [1990] to study the performance of shared-data caches in large-scale multiprocessors.

The traces used in this study were compressed in such a way that noncacheable references are fully described, while cacheable references are enumerated only, *i.e.*, only the number of cacheable references 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, and that the (write-through) data caches, instruction caches, and associated TLB references have hit rates of 97%, 99%, and 99%, respectively. 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 simulation begins as follows (cf., [Norton and Pfister, 1985]). We first 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 the offered load from which a network queueing delay is calculated using formulas of Kruskal, Snir, and Weiss [1986] and Liu [1990]. This 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.

5. Simulation Results

In this section we present results of experiments that measure the performance of processor- and memory-based TLBs, using two program traces: Simple and Peskin. A group of experiments driven by, *e.g.*, the Peskin trace with physical memory equal to 33% of virtual memory is referred to as Peskin/33%. If the experiments are, in addition, only for a TLB size of 64 entries, then the group is referred to as Peskin/33%/64.

A *simulation* encompasses the execution of one iteration of the initialization, computation, and termination sections of a program. (A production run would include many computation iterates.) Unless otherwise noted, our figures are produced from data collected during entire simulations. The exceptions present data collected during the computation section only.

5.1. Processor-based TLBs

In this section we present the results of our simulations of processor-based TLB systems with Validation. Our simulations indicate the following:

(1) For the Simple experiments, R_{Val}, the number of reloads, grows linearly with N.

(2) Both the Simple and Peskin programs contain programming constructs that can cause R_{Val} to grow linearly with N. However, the Peskin simulations do not illustrate this growth for the range of N studied.

(3) Our simulations indicate that OH_{Val}, the cost of all the reloads, grows as $\log N$, even for simulations that report R_{Val} growing linearly with N, for which we expect OH_{Val} to grow as $N \log N$. This

apparent inconsistency is due to the constants in the equation that describes R_{Val}'s growth, which cause the $N \log N$ growth of OH_{Val} to be seen only beyond our experiments' range of N. (We use the term asymptotic loosely since a program's maximum available parallelism limits the value of N.)

(4) As expected, R_{Val} and OH_{Val} decrease as the TLB size increases. This is supported by data presented in Table 1.

The next three subsections present data supporting claims 1-3.

5.1.1. TLB Reloads

Using linear regression we found that a linear growth of R_{Val} with N is exhibited by the Simple experiments, but not by the Peskin experiments. Table 1 presents the related regression equations, which are of the form $R_{Val} = aN + b$. For the Peskin experiments, Table 1 gives, in parentheses, the number of the figure illustrating the related experimental data and regression lines (the latter are labeled *Fit* in the figures). Figures 2 and 3 are for Peskin/33% and Peskin/50%, respectively. Note that in our figures *TLBEs* stands for TLB entries. Since the regression equations for the Simple experiments fit so well, we do not include corresponding figures.

Experiment Set (Figure No.)	**Number of TLB Entries**	**Equation for $\mathbf{R}_{Val}$**	$\mathbf{r^2}$
Simple/33%	16	$6130N + 905758$	.9996
	32	$6109N + 904831$	.9996
	64	$6094N + 899240$	.9996
Simple/50%	16	$5014N + 1180031$	.9995
	32	$5304N + 1102218$	.9996
	64	$5270N + 1102032$	.9996
Peskin/33% (2)	16	$1630N + 838583$	.8841
	32	$2308N + 814470$	.8856
	64	$1936N + 775704$	.8961
Peskin/50% (3)	16	$1387N + 843623$	.6828
	32	$1152N + 825394$	.5095
	64	$517N + 793134$	.1294

Table 1. Regression Equations for R_{Val}

For all workloads, the growth of R_{Val} with N is traceable to the computation section of the associated program. We illustrate this in Figures 2 and 3 for systems with 64-entry TLBs.

5.1.1.1. Simple Experiments

The definite linear relationship between R_{Val} and N for the Simple experiments, *i.e.*, the good fit of the regression equations, is indicated in Table 1 by the values of r^2, the coefficient of determination, being greater than .99.

A linear growth of R_{Val} with N in a processor-based TLB system can be caused by a parallel task group (*i.e.*, a group of tasks that can be executed concurrently) or a replicated task (*i.e.*, a task executed by all processors cooperating in the execution of a program) all sharing a page (or pages). The translation information for the shared page(s) is loaded into N TLBs in an N-processor system.

For Simple the majority of page-ins (in this context, a page-in refers to a particular main-memory residency of a noncacheable page) are referenced by one processor and the majority of the remaining page-ins are referenced by all N processors. (Recall that our programming model is SPMD.) These latter *fully-productive page-ins*, which generate a minimum of N processor-based TLB reloads, may well be caused by the parallel or replicated tasks mentioned above. Simple/33%/64 data is presented in Table 2, which indicates the percentage of page-ins referenced during an entire simulation by each number of processors in systems with $N = 2^i$, $i = 0, \ldots, 6$, processors. In parentheses below each of these percentages, we indicate the percentage relative to *multiprocessor page-ins, i.e.*, those referenced by more than one processor. Very similar percentages apply if we examine the results gathered during only the computation section, because

Number of Referencing Processors	**Percentage of Page-Ins Referenced**						
	N = 1	**N = 2**	**N = 4**	**N = 8**	**N = 16**	**N = 32**	**N = 64**
1	100%	98%	98%	98%	98%	98%	98%
2 through $N - 1$ for $N > 2$			.4% (15%)	.4% (20%)	.4% (20%)	.5% (21%)	.5% (20%)
N for $N > 1$		2% (100%)	1.6% (85%)	1.6% (80%)	1.6% (80%)	1.5% (79%)	1.5% (80%)

Table 2. Analysis of Page-Ins Referenced by Processors—Simple/33%/64

the majority of page-ins occur in that section.

For the Simple experiments, the fully-productive page-ins explain the linear growth of R_{Val} with N: There is a high correlation between the number of additional TLB reloads generated during the computation section of the Simple simulations in an N-processor system ($N > 1$), as compared to a one-processor system, to the minimum number of additional reloads due to fully-productive page-ins. Data from Simple/33%/64 and Simple/50%/64 are presented in Tables 3 and 4, respectively. In these tables, the column labeled $R_N - R_1$ indicates the additional number of TLB reloads in an N-processor system. The column labeled f gives the number of fully-productive page-ins, and the column labeled $(N-1)f$ indicates the minimum number of additional TLB reloads generated by the fully-productive page-ins. The correlation coefficient between $R_N - R_1$ and $(N-1)f$ exceeds .99 for both Simple/33%/64 and Simple/50%/64.

N	R	$\mathbf{R_N - R_1}$	f	$\mathbf{(N-1)f}$
1	421961		386528	
2	432238	10277	6925	6925
4	447730	25769	6374	19122
8	472610	50649	6026	42182
16	521073	99112	6006	90090
32	617672	195711	5985	185535
64	810206	388245	5977	376551

Table 3. Effect of Fully-productive Page-ins on R_{Val}—Simple/33%/64, Computation Section Only

N	R	$\mathbf{R_N - R_1}$	f	$\mathbf{(N-1)f}$
1	624374		382199	
2	633616	9242	4651	4651
4	647481	23107	3866	11598
8	668355	43981	3751	26257
16	710174	85800	3751	56265
32	793864	169490	3742	116002
64	960549	336175	3733	235179

Table 4. Effect of Fully-productive Page-ins on R_{Val}—Simple/50%/64, Computation Section Only

5.1.1.2. Peskin Experiments

The Peskin program also contains parallel programming constructs that can cause the number of processor-based TLB reloads to grow linearly with N. But, as mentioned above and shown in Table 1 and Figures 2 and 3, the results of the Peskin experiments do not illustrate a $\Theta(N)$ growth of R_{Val}. This growth may be delayed beyond our experiments' range of N or TLB sizes.

As illustrated in Table 5, for Peskin, unlike Simple, fully-productive page-ins constitute only a small portion of the multiprocessor page-ins. Hence, even though $(N-1)f$ grows linearly with N, at least starting with $N=8$, the presence of a large number of multiprocessor page-ins referenced by only a few processors prevents R_{Val} from exhibiting such growth. Data for the computation sections of the Peskin/33%/64 and Peskin/50%/64 simulations are presented in Tables 6 and 7. For these experiments the correlation coefficients between $R_N - R_1$ and $(N-1)f$, are: .78 and .18, respectively.

5.1.2. The Cost of Processor-based TLB Reloads

Data presented in Table 8 and illustrated in Figure 4 for Simple/33% indicate that OH_{Val} grows as $\log N$. The regression equations presented in Table 8 are of the form $OH_{Val} = a\log N + b$ and the logarithmic growth of OH_{Val} is verified by all the values of r^2 being greater than .97.

Number of Referencing Processors	**Percentage of Page-Ins Referenced**						
	N = 1	**N = 2**	**N = 4**	**N = 8**	**N = 16**	**N = 32**	**N = 64**
1	100%	71%	69%	67%	66%	66%	65%
2		29% (100%)	22% (72%)	22% (66%)	22% (64%)	22% (65%)	22% (65%)
3			6% (19%)	7% (21%)	7% (20%)	7% (21%)	7% (20%)
4			3% (9%)	2% (6%)	2% (7%)	2% (7%)	3% (7%)
5 through $N-1$ for $N>4$					5% (1%)	1% (3%)	1% (4%)
N for $N>4$				2% (6%)	2% (6%)	1% (3%)	1% (4%)

Table 5. Analysis of Page-Ins Referenced by Processors—Peskin/33%/64

N	R	$R_N - R_1$	f	$(N-1)f$
1	690382		71261	
2	699862	9480	18951	18951
4	729170	38788	2308	6924
8	728796	38414	1568	10976
16	738096	47714	1442	21630
32	791739	101357	1385	42935
64	819050	128688	1342	84546

Table 6. Effect of Fully-productive Page-ins on R_{Val}—Peskin/33%/64, Computation Section Only

N	R	$R_N - R_1$	f	$(N-1)f$
1	676077		2589	
2	704050	27973	2087	2087
4	746420	70343	1584	4752
8	748817	72740	1497	10479
16	759099	83022	1416	21240
32	769455	93378	1383	42873
64	734937	58860	1333	83979

Table 7. Effect of Fully-productive Page-ins on R_{Val}—Peskin/50%/64, Computation Section Only

Experiment Set (Figure No.)	Number of TLB Entries	Equation for OH_{Val}	r^2
Simple/33% (4)	16	$55706414 \log N + 23481226$	.9753
	32	$55599574 \log N + 23475710$	.9753
	64	$55325349 \log N + 23305418$	.9751
Simple/50%	16	$63168834 \log N + 34028252$	.9889
	32	$60962068 \log N + 31069553$	.9862
	64	$60845289 \log N + 31103111$	.9863
Peskin/33%	16	$40177991 \log N + 26022986$	.9917
	32	$41913901 \log N + 24122378$	.9952
	64	$39109473 \log N + 23364348$	.9947
Peskin/50%	16	$39952860 \log N + 26434542$	.9922
	32	$38503194 \log N + 26095587$	.9903
	64	$35374829 \log N + 25596236$	.9830

Table 8. Regression Equations for OH_{Val}

Recall that $OH_{Val} = S \times T_{stale} + R_{Val} \times T_{reload}$. Our simulations indicate that S (the number of times a stale TLB entry is used) is extremely small for Peskin and zero for Simple and, thus, $S \times T_{stale}$ contributes little to OH_{Val}. Since the offered load to the network was small in all our experiments, the resulting network queueing delays were also small. Hence, one expects that the contention-free cost of a noncacheable single-word load ($2\log_2 N + 3$) would be a good approximation to the actual cost. Consequently, a reload, which consists of 18 instructions with 5 such loads would be approximated well by the contention-free reload cost

$$C_{reload} = 18 + 5(2\log_2 N + 3) = 33 + 10\log_2 N \ .$$

Thus, the overhead would be approximated well by

$$OH_{Val} = R_{Val} \times C_{reload} \ .$$

For the Simple experiments, we have regression equations of the form $R_{Val} = aN + b$. When these are substituted into the above equation for OH_{Val}, we get good agreement with our simulation results. In Figure 5 (for Simple/33%/64), this equation is plotted and labeled *Validation-No Stale*. The curve labeled *Simulation Trend* plots the regression equation $OH_{Val} = a\log N + b$ determined by the seven indicated experimental data points; the good fit of the regression equation to our data is verified by $r^2 = .98$ (see Table 8).

Although the data points agree with *Validation-No Stale*, they agree even better with *Validation-All Stale*, which corresponds to the equation

$$OH_{Val} = R_{Val} \times (C_{reload} + T_{stale}) \ ,$$

i.e., all reloads result from using stale entries. When we conducted our study, we did not realize stale TLB entries would be used rarely and instead assumed that each reload would be caused by using a stale entry, which increased the cost of each reload by one network round trip.

In contrast, note the sharp disagreement for large N between *Simulation Trend* and the other curves. This disagreement is expected since *Simulation Trend* = $\Theta(\log N)$ and the overheads are $\Theta(N\log N)$. The reason the data points indicate a logarithmic growth, instead of an $N\log N$ growth, is that $R_{Val} = aN + b$, where $b >> a$. As a result, the linear term is not apparent for the modest values of N simulated. To illustrate this, we consider the Simple/33%/64 data and show why the asymptotic $N\log N$ growth rate of OH_{Val} is not seen until $N = 128$.

Referring to Table 1, we see that for Simple/33%/64, $R_{Val} \approx 6094N + 899240$ and, hence,

$$OH_{Val} \approx R_{Val} \times C_{reload} \approx (aN+b)(c+d\log_2 N) = caN + cb + daN\log_2 N + db\log_2 N \ ,$$

where $a = 6094$, $b = 899240$, $c = 33$, and $d = 10$. As shown in Table 9, the $\log_2 N$ term dominates until $N = 128$. At $N = 256$, the $N\log_2 N$ term begins to dominate.

Note that speculative information is included in Figure 5 for $N \geq 256$. Our Simple and Peskin experiments are driven by traces of programs that solve fixed-size problems. Therefore, a maximum available parallelism is defined for each program. In this figure, we extrapolate beyond the maximum available parallelism for Simple, which is 256. For comparison purposes, Figure 5 also includes *Generic*, which is the contention-free overhead for R_{Val} invocations of a TLB reload routine that does not ensure consistency. Such a TLB reload routine is assumed in Section 5.3.3, where we compare the performance of memory-based TLBs to that of processor-based TLBs with Validation and with TLB Shootdown.

5.2. Memory-based TLBs

In this section we present simulation results for memory-based TLB systems. We begin with a summary and, then, in the two subsections that follow, present supporting data generated by our experiments.

As described above, the contention-free cost of a memory-based TLB reload, C_{reload}, is very small (five or seven cycles) and remains constant as N increases because, unlike for a processor-based TLB reload, network transit is not required. However, contention for TLBs and cluster buses strongly deter-

N	caN	daN$\log_2$N	db$\log_2$N
1	2010862	0	0
2	402171	121870	8992401
4	804343	487481	17984802
8	1608686	1462441	26977203
16	3217371	3899844	35969604
32	6434742	9749610	44962005
64	12869485	23399063	53954406
128	25738969	54597814	62946807
256	51477939	1.2 E+08	71939208
512	1.0 E+08	2.8 E+08	80931609
1024	2.1 E+08	6.2 E+08	89924010

Table 9. Analysis of OH_{Val}—Simple/33%/64

mines the average cost of a memory-based TLB reload.

For the studied workloads and within our experiments' range of N, our results show that:

(1) The performance of memory-based TLBs depends on the number of paging arenas. For a fixed N and fixed physical memory and TLB sizes, as the number of paging arenas increases from one to two, the number of reloads, R_{Mem}, decreases. As a result, contention for TLBs and cluster buses decrease and, therefore, the average cost of a TLB reload decreases. We believe this trend will continue for larger numbers of arenas and have preliminary data (not shown) supporting this belief. However, more data is needed.

(2) Memory-based TLBs with two paging arenas outperform processor-based TLBs, which in turn outperform memory-based TLBs with one paging arena.

5.2.1. TLB Reloads

Our experimental data show that:

(1) For systems with one or two paging arenas and $N > 2$, R_{Mem} grows sublinearly with N.

(2) R_{Mem} decreases as the TLB size or the number of paging arenas increases.

Figures 6 and 7 illustrate the growth of R_{Mem} as N increases, and compare R_{Mem} to R_{Val}. Each figure includes: the experimental data for memory-based TLB systems with one paging arena and 16-, 32-, and 64-entry TLBs and for memory-based TLB systems with two paging arenas and 64-entry TLBs. Figure 6 also plots the regression equation (presented in Table 1) that characterizes the growth of R_{Val} for the Simple/50%/64 experiments. Figure 7 also plots the Peskin/50%/16 and Peskin/50%/64 data points for R_{Val}; since we did not get a good linear fit to these data, we cannot use regression lines.

5.2.1.1. Monotonic Increase of R with N

For our memory-based TLB experiments, the growth of R_{Mem} with N likely is due to two factors:

(1) C, the number of memory modules in a cluster, increases with N and, thus, a page's translation information can be replicated at as many as C TLBs.

(2) TLB-entry thrashing, which occurs in memory-based TLBs due to multiple processes accessing shared TLBs for different address translation information, increases with *N*.

For the range of *N* simulated, R_{Mem} clearly does not grow linearly. But, as illustrated in Figures 6 and 7, regression lines fitted to the data points for the larger simulated systems indicate that for larger *N* linearity may be the trend. The data provided by our experiments are not sufficient to settle this question.

5.2.1.2. R Decreases as TLB Size or Number of Paging Arenas Increases

As with any cache, the number of misses decreases with TLB size, which lowers R_{Mem}. The effect of the number of arenas likely is due to two factors. First, assuming spatial locality of reference, many words of a page are referenced during a short time interval. Since these words are interleaved across the cluster, many TLBs will reload the page's address translation information. As the number of paging arenas increases, the number of TLBs per cluster and, hence, the replication of translation information across the TLBs decrease.

Second, 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 and, thus, TLB-entry thrashing is likely to decrease. At the extreme, preliminary simulations for $N = 2^i$, $i = 0, \ldots, 4$, indicate indicate that R_{Mem} for systems with *N* paging arenas (*i.e.*, non-interleaved memory) actually *decreases* with *N*. In this case the memory-based TLBs constitute one, large, shared TLB with no replication of translation information. Although R_{Mem} decreases with *N*, we see in the next section that network and memory contention increases with *N*.

In addition to the fact that R_{Mem} decreases as the number of paging arenas increases, for 17 out of 24 simulations the number of page-ins generated with two paging arenas is less than the number generated with one.

5.2.2. The Cost of Memory-based TLB Reloads

As expected, memory-based TLBs' reload overhead (OH_{Mem}), like R_{Mem}, decreases as the TLB size or the number of paging arenas increases. This is illustrated in Figures 8, 9, and 12. But, unlike R_{Mem}, OH_{Mem}, within our experiments' range of *N*, grows more than linearly with *N*, even though it initially appears to grow as $\log N$. The additional growth is caused by the cost per reload increasing with N. Unlike our processor-based TLB experiments, for which we have a model for TLB-reload overhead, we are not able to characterize the growth of OH_{Mem} with N.

Memory-based TLBs' contention-free TLB reload costs are small and remain constant as N increases, but the other components of the cost of memory-based TLB reloads, *i.e.*, $T_{busy-TLB}$, $T_{busy-bus}$, and $T_{pgexcpt}$, increase with N and depend on the number of paging arenas. Our simulator records each component of OH_{Mem} and our results indicate that:

(1) $T_{pgexcpt}$ grows logarithmically with N. This is because one network round trip is required per exception and, in the majority of our experiments, the number of exceptions, which equals the number of page-ins, remains fairly constant as N increases. As illustrated in Figures 8 and 9, $T_{pgexcpt}$ is comparable in systems with one and two paging arenas.

(2) As illustrated in Figures 8, 9, 10, and 11, $T_{busy-TLB}$ and $T_{busy-bus}$ increase with N and decrease with the number of paging arenas. These trends are expected: As N increases, so does R_{Mem} (which means that TLBs and cluster buses are more busy). As the number of paging arenas increases, R_{Mem} decreases.

In contrast, as N (and, hence, R_{Mem}) increases, the average contention-free cost of a single reload decreases, which lessens the impact that N has on OH_{mem}. This decrease occurs because (for a fixed number of clusters), increasing N increases the number of TLBs in a cluster, which increases the percentage of TLB misses serviced by other TLBs. (Recall that we assume a TLB miss serviced by another TLB is less expensive than one serviced by page-table memory.) The decrease in the number of reloads serviced by the page-table memory is at first dramatic, but at a certain N levels off, which we suspect occurs because at different points in a program the context changes and, as a result, no matter how many TLBs are in a cluster, the translation information for some pages will not be TLB resident. Even if the TLBs are large enough to store the translation information for all pages mapped to their clusters, each page's translation information must be loaded once.

Our experiments for systems with one, two, and N paging arenas indicate that memory-based TLBs perform better as the number of paging arenas increases. 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 simulation does not detect this increased contention.

5.3. Processor-based vs. Memory-based TLBs

In this section we compare the performance of processor-based TLB systems with that of memory-based TLB systems, with respect to the number of reloads generated, R, and the cost of these reloads, *i.e.*, the overhead, OH.

5.3.1. A Comparison of R

As illustrated in Graphs 5 and 6, generally R_{Mem} for systems with two paging arenas is smaller than R_{Val}, which in turn is smaller than R_{Mem} for systems with one paging arena.

R_{Mem} with one paging arena generally exceeding R_{Val} is due to replication of translation information among multiple TLBs and TLB-entry thrashing due to multiple processes accessing the same shared TLBs. Since an increase of the number of paging arenas leads to a decrease in the replication of translation information and TLB-entry thrashing, it is not surprising that R_{Mem} for systems with two paging arenas is smaller than R_{Mem} for systems with one paging arena.

As N increases, R_{Mem} for systems with two paging arenas approaches R_{Val} from below, suggesting that for best performance the number of arenas should grow with N (experiments we intend to perform). Unlike memory-based TLBs, there is no reason to believe that multiple paging arenas would decrease the number of processor-based TLB reloads.

5.3.2. A Comparison of Reload Costs

Memory-based TLBs can tolerate a higher TLB miss-rate than processor-based TLBs and still have a lower TLB-reload overhead. In other words, even if $R_{Mem} > R_{Val}$, it is not necessarily the case that $OH_{Mem} > OH_{Val}$. For example, consider Figures 6 and 12 for $N \leq 16$. This effect is more dramatic if we improve the design of the memory-based TLBs to be lockup-free [Kroft, 1981]. To estimate the improvement possible from using lockup-free cache techniques, we subtract $T_{busy-TLB}$ from OH_{Mem}. This revised estimate of OH_{Mem} is somewhat optimistic because some of $T_{busy-TLB}$ should have been in $T_{busy-bus}$ instead. As shown in Figure 12, lockup-free TLBs can significantly decrease OH_{Mem} and, thus, significantly increase the the performance advantage of memory-based TLBs.

As mentioned above, although our results indicate that memory-based TLBs perform better than processor-based TLBs, provided that the number of paging arenas grows with *N*, as the number of paging arenas increases, network and memory contention may increase.

5.3.3. A Comparison with TLB Shootdown

Although we did not simulate systems employing TLB Shootdown, we give evidence that, for the SPMD workloads studied in which *R* grows linearly with *N*, Shootdown would have a significantly higher overhead than our solutions for large *N*. This comparison is based on estimates of Shootdown's overhead, a generic TLB-reload routine, and simulation data that defines the number of invocations of Shootdown.

We use Rosenburg's modified TLB Shootdown algorithm [1989], which he showed to have better performance than the original, more generally applicable algorithm of Black, *et al.* [1989]. TLB Shootdown is invoked at each unsafe change and the algorithm interrupts the execution of each executing process with access to the related page table.

In comparing the performance of Shootdown, Validation, and Memory-based TLBs, we include the cost of TLB reloads in each case. We define Shootdown's overhead as

$$OH_{Shoot} = R \times C_{reload} + \beta \times T_{Shoot},$$

where *R* is the number of TLB reloads, C_{reload} is the contention-free cost of a generic TLB-reload routine (which does not ensure TLB consistency), β represents the number of invocations of the TLB Shootdown algorithm, and T_{Shoot} is our estimated cost of an invocation of the algorithm.

Our 29000 implementation of a generic reload routine for shared translation information contains 9 instructions with two single-word loads; hence, as defined earlier,

$$C_{reload} = 9 + 2(2\log_2 N + 3) \ .$$

Our simulation data permits us to determine R and β:

We assume a 64-entry TLB and define $R = R_{Val}$, since Shootdown, like Validation, uses processor-based TLBs. We use the regression equation for R_{Val} computed from the Simple/33%/64 data points to extrapolate for N beyond the range of our experiments. This is reasonable because the regressions equations fit the Simple data very well.

Shootdown is invoked on unsafe page table changes, which in our simulations are caused only by page evictions. Hence β equals the number of evictions, which we approximate as the number of page-ins minus the number of frames. These values are reported by the simulator.

Rosenberg [1991] estimates that responding to a shootdown request requires 100 instructions and 3 single word loads of shared data. In an SPMD environment all N processors need to participate in the shootdown since the page-table entry undergoing the unsafe change may be in any of the TLBs. Hence we estimate

$$T_{shoot} = N[100 + 3(2\log_2 N + 3)]$$

As illustrated in Figure 13, our comparison indicates that our solutions with one or two paging arenas outperform TLB Shootdown. Note that, as with the data presented in Figure 5, speculative information is included in Figure 13 because we extrapolate beyond the maximum available parallelism for Simple, *i.e.*, beyond $N \geq 256$. OH_{Shoot} clearly depends on β, which in our study depends on the number of page evictions. The programs that we use in this study have not proven to be amenable to a demand paging environment. Therefore, we also estimate TLB Shootdown's overhead for smaller β. Again using the Simple/33%/64 experiment, dividing β by 10 produces similar results. However, when β is divided by 20, TLB Shootdown outperforms memory-based TLB systems with one paging arena, for $N = 128$.

Since $R_{Val} \times C_{reload}$ and our estimate of T_{Shoot} asymptotically grow as $N \log N$ and β does not grow with N (see data in [Teller, 1991]), it is not surprising that OH_{Shoot}, like OH_{Val}, asymptotically grows as $N \log N$.

6. Conclusions and Future Research

To understand the performance impact of algorithms in a highly-parallel multiprocessor, it is not sufficient to study performance in a single multiprocessor system, where the cost of network transit is relatively constant and the sizes of system components are fixed. In order to characterize performance in a highly-parallel multiprocessor, performance must be studied for a range of system sizes with the cost of network transit increasing with system size and with system components of variable size.

Using such a technique, 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 architects seriously should consider this alternative to processor-based TLBs.

Our simulations show that multiprocessors with memory-based TLBs and two or N paging arenas have better performance than multiprocessors with processor-based TLBs. 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 should be proportional to N. 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.

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. In contrast, Validation and TLB Shootdown are fully general solutions to the TLB consistency problem.

Although TLB Shootdown is an attractive solution to the TLB consistency problem because it is fully general and requires essentially no hardware support, it may degrade performance in SPMD systems with large numbers of processors. Thus, architects should consider alternative solutions, such as Validation and Memory-based TLBs [Teller, Kenner, and Snir, 1988].

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. Without being able to simulate larger systems, we were not able to determine if the number of processor-based TLB reloads for Peskin grows linearly with N or if, for large N, the number of memory-based TLB reloads grows linearly with N. We plan to study these and other programs in another simulation environment so that we can study larger problem sizes and higher levels of parallelism.

The programs used in our study proved to be poor candidates for demand paging. Even when the number of frames allocated permitted us to store half of the shared read-write pages, many page faults occurred and performance was significantly degraded. We are persuing how prevalent this phenomenon is for ‘‘dusty deck’’ scientific programs.

We intend to continue our study of TLB reload and other overheads since a large reload overhead that grows strongly with N will have a deleterious effect on speedups attained for programs executed on highly-parallel multiprocessors.

Acknowledgements

We wish to thank colleagues at NYU's Ultracomputer Research Laboratory and IBM for their help. In particular, we thank Jan Edler, Richard Kenner, Bryan Rosenburg, and Harold Stone for constructive criticism and good advice, and Frederica Darema, Evelyn Melton, Gregory Pfister, Kimming So, and Jan Stone for their help with collecting address traces.

References

Agarwal, A., and M. Cherian, ‘‘Adaptive Backoff Synchronization Techniques,’’ *Proceedings of the 16th International Symposium on Computer Architecture*, IEEE Catalog No. 89CH2705-2, pp. 396-406, May/June 1989.

AMD, *Am29000 Streamlined Instruction Processor User’s Manual*, 1987.

BBN, *Butterfly Parallel Processor Overview*, Cambridge, MA, June, 1985.

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*, IEEE Catalog No. 89CH2710-2, pp. 113-122, April 1989.

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.

Crowley, W., C. Hendrickson, and T. Rudy, ‘‘The Simple code,’’ Lawrence Livermore Laboratory, Livermore, CA, February 1978.

Darema-Rogers, F., D. George, V. Norton, G. Pfister, ‘‘A single-program-multiple-data computational model for EPEX/FORTRAN,’’ IBM Research Report No. RC11552, November 1986.

Darema-Rogers, F., D. George, V. Norton, and G. Pfister, ‘‘VM/EPEX—a VM environment for parallel execution,’’ IBM Research Report No. RC11225, January 1985.

Eggers, S., and R. Katz, ‘‘A Characterization of sharing in parallel programs and its’’ application to coherency protocol evaluation," *Proceedings of the 15th Annual International Symposium on Computer Architecture*, IEEE Catalog No. 88CH2545-2, pp. 373-382, June 1988.

Gajski, D., D. Kuck, D. Lawrie, and A. Sameh, ‘‘CEDAR — a large scale multiprocessor,’’ *Proceedings of the 1983 International Conference on Parallel Processing*, IEEE Catalog No. 83CH1922-4, pp. 524-529, August 1983.

Goodman, J. ‘‘Coherency for multiprocessor virtual address caches,’’ *Proceedings of the Second International Conference on Architectural Support for Programming Languages and Operating Systems*, IEEE Catalog No. 87CH2440-6, pp. 72-81, October 1987.

Gottlieb, A., ‘‘An overview of the NYU Ultracomputer project,’’ in J. Dongarra, editor, *Experimental Parallel Computing Architectures*, Amsterdam, The Netherlands: North-Holland, pp. 25-95, 1987. (formerly Ultracomputer Note No. 100, N.Y.U., Courant Institute, July 1986.)

Gottlieb, A., ‘‘WASHCLOTH—the logical successor to Soapsuds,’’ Ultracomputer Note No. 12, N.Y.U., Courant Institute, December 1980.

Knuth, D., ‘‘Big Omicron, big omega, and big theta,’’ SIGACT News (ACM), **8**, No. 2, pp. 18-24, 1976.

Kroft, D., ‘‘Lockup-free instruction fetch/prefetch cache organization,’’ *Proceedings of the 8th Annual International Symposium on Computer Architecture*, IEEE Catalog No. 81CH1593-3, pp. 81-88, June 1981.

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*, IEEE Catalog No. 86CH2355-6, pp. 12-19, August 1986, and *IEEE Transactions on Computers*, **37**, 11, pp. 1337-1352, November 1988.

Lee, F., ‘‘Study of ‘look aside’ memory,’’ *IEEE Transactions on Computers*, **C-18**, 11, pp. 1062-1064, November 1969.

Liles, A., and G. Paul, SEMUL was developed at IBM Research to trace execution of scientific programs.

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, September 1990.

Lubachevsky, B., ‘‘Parallelizing an algorithm of Charles S. Peskin for describing incompressible flow of a fluid coupled to an elastic membrane,’’ Ultracomputer Note No. 42, N.Y.U., Courant Institute, January 1981.

Peskin, C., ‘‘Numerical analysis of blood flow in the heart,’’ *Journal of Computation Physics*, 25, pp. 220-252, 1977.

Pfister, G., and V. Norton, ‘‘‘Hot spot’ contention and combining in multistage interconnection networks,’’ *IEEE Transactions on Computers*, **C-34**, 10, pp. 943-948, October 1985.

Pfister, G., W. Brantley, D. George, S. Harvey, W. Kleinfelder, K. McAuliffe, E. Melton, V. Norton, and J. Weiss, ‘‘The IBM research parallel processor prototype (RP3): introduction and architecture,’’ *Proceedings of the 1985 International Conference on Parallel Processing*, IEEE Catalog No. 85CH2140-2, pp. 764-771, August 1985.

Rosenburg, B., ‘‘Low-synchronization translation lookaside buffer consistency’’ in large-scale shared-memory multiprocessors," *Proceedings of the 13th Symposium on Operation Systems Principles*, pp. 137-146, September 1989.

Satyanarayanan, M., and D. Bhandarkar, ‘‘Design trade-offs in VAX-11 translation buffer organization,’’ *Computer*, **14**, 12, pp. 103-111, December 1981.

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, August 1987.

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, January 1986.

Stone, J., F. Darema-Rogers, V. Norton, and G. Pfister, ‘‘The VM/EPEX FORTRAN preprocessor reference,’’ IBM Research Report No. RC11408, September 1985.

Strom, R., and S. Yemini, ‘‘Optimistic recovery in distributed systems,’’ *ACM Transactions on Computer Systems*, **3**, 3, pp. 204-226, August 1985.

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

Teller, P., ‘‘Translation-lookaside buffer consistency,’’ *Computer*, **23**, 6, pp. 26-36, June 1990.

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*, IEEE Catalog No. 85TH0209-7, pp. 184-193, January 1988.