Software TLB Management in OSF/1 and Mach 3.0

by

Richard Uhlig, David Nagle, Trevor Mudge and Stuart Sechrest

CSE-TR-156-93

Computer Science and Engineering Division
Room 3402 EECS Building

THE UNIVERSITY OF MICHIGAN

Department of Electrical Engineering and Computer Science
Ann Arbor, Michigan 48109-2122
USA

Software TLB Management in OSF/1 and Mach 3.0

Richard Uhlig, David Nagle, Trevor Mudge, Stuart Sechrest
Dept. of EECS, University of Michigan, Ann Arbor

Full Paper — Submitted to
3rd USENIX Mach Symposium

December 4, 1992

1.0 Abstract

Many new computer architectures provide reduced hardware support for virtual memory in the form of basic translation buffer (TLB) hardware, software traps for TLB miss handling, and a small set of instructions for probing and changing TLB state. This means that operating system writers are responsible for choosing page table structure and the policies governing the placement and replacement of page table entries in the TLB. If done carelessly, software TLB management can impose considerable penalties which are exacerbated by the structure of newer generation operating systems such as OSF/1 and Mach 3.0.

This work explores the current TLB management policies of OSF/1 and Mach 3.0-based systems and explains some of the reasons for their lower performance when compared against more traditional, monolithic-kernel designs such as Ultrix. We present a collection of improved TLB management techniques and measure their effectiveness with OSF/1 and Mach 3.0. Although our experiments are performed on a MIPS R2000-based machine, our techniques have been designed to exploit features of the newer MIPS R4000 TLB.

2.0 Introduction

Microkernel operating systems such as Mach 3.0 offer many advantages [Accetta et al. 1986], but have always suffered performance problems [Guillemont et al. 1992; Anderson et al. 1991]. In their 1991 ASPLOS paper, Anderson et. al suggested that a major reason for these performance problems is a divergence of operating system and computer architecture trends. They arrived at this conclusion by estimating the cost measuring the frequency of various primitive operating system functions such as context switching, system call invocation and translation buffer (TLB) miss handling. Figure 1 reproduces some of Anderson's results which compare the performance of the monolithic-kernel Mach 2.5 against the microkernel-based Mach 3.0. With the exception of instruction emulation, kernel TLB (KTLB) miss handling is by far the most frequently invoked operating system primitive considered in Anderson's study.[1] Note that although address space and thread context switches increase most dramatically in the Mach 3.0 system, their frequency is small when compared against KTLB misses.

The R3000-based DECstation 5000 used in Anderson's study is representative of a recent class of machines that require TLB misses to be serviced by OS software [Kane et al. 1992]. Other processor architectures with software-managed TLBs include the HP-PA and the DEC Alpha [Hewlett-Packard 1990; Digital 1992]. Such machines require operating system writers to choose both page table structure and the policies governing placement and replacement of page table entries (PTEs) in the TLB. Although the costs of software TLB management can be

1. KTLB misses typically represent only about 5% to 10% of all TLB miss types. When all TLB miss types are taken into account, they outnumber even the emulated instructions.

This work was supported by the Defense Advanced Research Projects Agency under DARPA/ARO Contract Number DAAL03-90-C-0028, by National Science Foundation Grant No. CDA-9121888 and by a National Science Foundation Graduate Fellowship.

Operating System	Time (seconds)	Address Space Switches	Thread Context Switches	System Calls	Emulated Instructions	KTLB Misses	Other Exceptions
Mach 2.5	307.2	10,740	18,225	90,605	2,650,879	439,067	173,937
Mach 3.0	416.5	215,811	248,274	278,771	5,233,938	3,857,602	390,385
Ratio	1.4 : 1	20.9 : 1	13.6 : 1	3.1 : 1	2.0 : 1	8.8 : 1	2.2 : 1

Table 1 Application Reliance on Operating System Primitives

These data are derived from Table 7 of [Anderson et al. 1991]. The counts are the sum of OS primitive invocations caused by running a workload consisting of spellcheck, latex, andrew (local and remote), linking a vmunix kernel and parthenon (1 and 10 threads). The time is the total required to run all of these programs once on an R3000-based DECstation 5000. The ratios show the increase in time (or counts) for the Mach 3.0 system when compared against Mach 2.5. Their Mach 3.0 system (which is similar to our Mach 3.0 + AFS system described in Table 2) supports UNIX and AFS functionality in two separate user-level servers.

substantial, there has been little investigation into this new OS responsibility.

In this work we examine the software TLB management problem in depth, considering all types of TLB misses. To perform this work, we extend the range of systems studied by Anderson et al. to include Ultrix, OSF/1 and two versions of Mach 3.0 (one with an AFS client cache manager and one without). We also construct workloads that emulate the activity of a system with many user-level servers and relate poor TLB performance to high degrees of system decomposition. To mitigate this cost, we propose and measure the effectiveness of various improved TLB management policies. Our experiments are performed on an R2000-based DECstation 3100, but our results extend to the newer MIPS R4000 processor [Kane et al. 1992].

This paper is organized as follows. Section 3.0 describes the tools, machines and operating systems used in our analysis. Section 4.0 is a case study of the current TLB management techniques used by Ultrix, OSF/1 and Mach 3.0 on a DECstation 3100. In Section 5.0 we suggest improvements to TLB management in the baseline systems and then measure their effectiveness. Section 6.0 summarizes the work.

3.0 Analysis Tools and Method

The performance of low-level operating system primitives is heavily dependent upon hardware parameters such as instruction and data cache size, TLB size and main memory bandwidth. To take these effects into account, we use a combination of hardware- and software-based analysis tools to perform our analysis of TLB management costs.

3.1 System Monitoring with Monster

The Monster monitoring system (see Figure 1) was developed to enable analyses of the interaction between operating systems and computer architectures. Monster consists of a DECstation 3100 whose motherboard has been modified so that a logic analyzer can be attached to the CPU pins. In this study, we use Monster to obtain accurate measurements of TLB miss handling costs. Monster's other capabilities are described more completely in a University of Michigan technical report [Nagle et al. 1992].

The logic analyzer component of Monster contains a programmable hardware state machine and a 128K-entry trace buffer. The state machine includes pattern recognition hardware that can sense the processor's address, data and control signals on every clock cycle. This state machine can be programmed to trigger on certain patterns appearing at the CPU bus and then dump these signals and a timestamp (with 1ns resolution) into the trace buffer.

Timing software code paths also requires an instrumented OS kernel. For example, to measure the time spent in the software TLB miss handlers, the entry and exit points of the handlers are annotated with distinctive `nop` marker instructions. The logic analyzer's state machine is programmed to detect the marker instructions which are stored to the logic analyzer's buffer along with a timestamp while the system runs some workload of interest. When the workload completes, the trace buffer is post-processed to obtain a histogram of time spent in the different invocations of the TLB miss handlers.

This technique for timing code paths yields far more accurate timings for single invocations of a code path than can

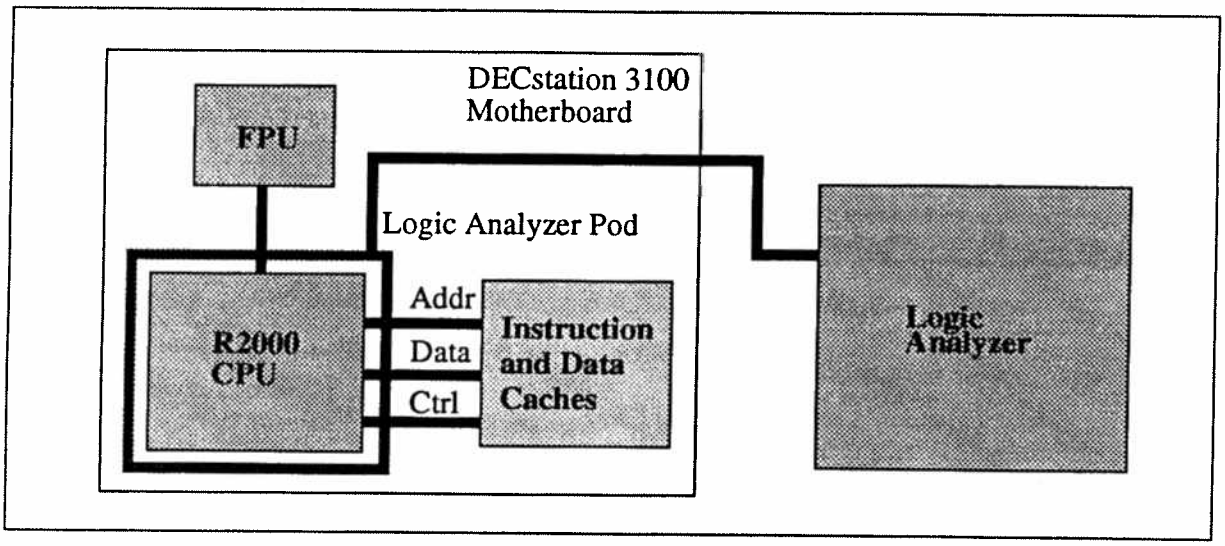

Figure 1 The Monster Monitoring System

Monster is a hardware monitoring system consisting of a Tektronix DAS 9200 Logic Analyzer and a DECstation 3100 running three operating systems: Ultrix, OSF/1 and Mach 3.0. The DECstation motherboard has been modified to provide access to the CPU pins, which lie between the processor and the cache. This allows the logic analyzer to monitor virtually all system activity. To enable the logic analyzer to trigger on certain operating systems events, such as the servicing of a TLB miss, each operating system has been instrumented with special marker NOP instructions that indicate the entry and exit points of various routines.

Operating System	Description
Ultrix	Version 3.1 of Ultrix, as shipped by Digital Equipment Corporation.
OSF/1	Version 1.0 from the Open Software Foundation (derived from Mach 2.5.)
Mach 3.0	Carnagie Mellon University's microkernel version mk77 and UNIX server version uk38.
Mach 3.0 + AFS	Same as Mach 3.0, but with an AFS cache manager running as a separate task outside of the UNIX server.

Table 2 Operating Systems Used in This Study

Benchmark	Description
gcc	The GNU cc compiler benchmark taken from the SPEC Benchmark Suite [SPEC 1991].
io	Creates an 8 Megabyte file.
Ousterhout	John Ousterhout's benchmark suite from [Ousterhout 1989].
spell	The UNIX spell utility applied to a 13K word latex document.

Table 3 Benchmarks Used in This Study

be obtained by using a system clock with much coarser resolution or, as is often done, by executing a code fragment in a loop N times and then dividing the total time spent by N [Clapp et al. 1986].

3.2 Method

All experiments were performed on an R2000-based DECstation 3100 (16.7MHz) running three different base operating systems (see Table 2): Ultrix, OSF/1 and Mach 3.0. Each of these systems includes a standard Berkeley fast file system [McKusick et al. 1984]. An additional Mach 3.0-based system was created by adding the Andrew File System (AFS) cache manager [Satyanarayanan 1990], running as a separate server task.[2] To obtain measurements, all of the operating systems were instrumented with counters and markers as outlined in the previous section.

[2] The current version of the UNIX server from CMU includes AFS functionality. For our experiments, we have (partially) moved this AFS code into a separate task.

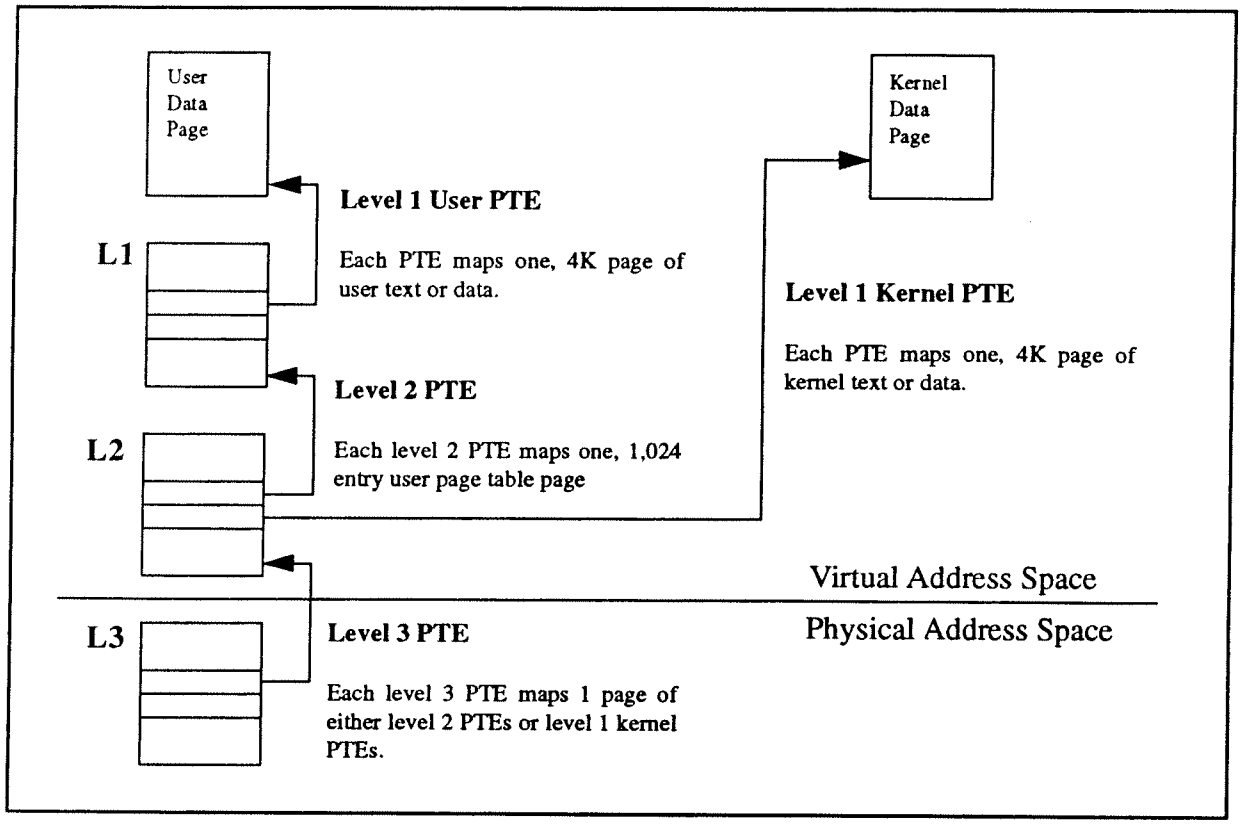

Figure 2 Page Table Structure in OSF/1 and Mach 3.0

The Mach page tables form a 3-level structure with the first two levels residing in virtual (mapped) space. The top of the page table structure holds the user pages which are mapped by level 1 user PTEs. These level 1 user PTEs are stored in the L1 page table with each task having its own set of L1 page tables.

Mapping the L1 page tables are the level 2 PTEs. They are stored in the L2 page tables which hold both level 2 PTEs and level 1 kernel PTEs. In turn, the L2 pages are mapped by the level 3 PTEs stored in the L3 page table. At boot time, the L3 page table is fixed in unmapped physical memory. This serves as an anchor to the page table hierarchy because references to the L3 page table do not go through the TLB.

The MIPS R2000 architecture has a fixed 4 Kilobyte page size. Further, each PTE requires 4 bytes of storage. Therefore, a single L1 page table page can map 1,024 level 1 user PTEs, or approximately 4 Megabytes of virtual address space. Likewise, the L2 page tables can directly map either 4 Megabytes of kernel data or indirectly map 4 Gigabytes of L1 user data.

Throughout the paper we use four benchmarks (see Table 3). The same binaries were used on all operating systems. To improve accuracy, each measurement cited in this paper is the average of three trials.

4.0 Case Study

In this section we describe the Ultrix, OSF/1 and Mach 3.0 page table structures and the hardware support for address translation provided by the R2000/R3000 and the newer R4000 microprocessors. We analyze the current TLB management policies for the different operating systems on an R2000-based DECstation 3100 and then present our measurements of TLB miss handling costs in these systems.

4.1 Page Tables and Translation Hardware

OSF/1 and Mach 3.0 on the DECstation both implement a linear[3] page table structure (see Figure 2). These page tables are used to store the mapping of pages from a large

[3] Rather than an inverted pages table [Wilkes et al. 1992].

TLB Miss Type	Ultrix	OSF/1	Mach 3.0
Level 1 User	20	20	20
Level 1 Kernel	294	294	294
Level 2	407	407	407
Level 3	----------	286	286
Other	405	405	405

Table 4 Costs for Different TLB Miss Types

This table shows the number of machine cycles (at 60 ns/cycle) required to service different types of TLB misses. To determine these costs, Monster was used to collect a 128K-entry histogram of timings for each type of miss. The histogram was then pruned by 25% to remove outliers caused by intervening interrupts, and the median value for each type of miss was found. We separate TLB miss types into the five categories described below. Note that Ultrix doesn't have level 3 PTE misses because it implements a 2-level page table.

Level 1 User TLB miss on a level 1 user PTE.

Level 1 Kernel TLB miss on a level 1 kernel PTE.

Level 2 TLB miss on level 2 PTE. This can only occur after a miss on level 1 user PTE.

Level 3 TLB miss on a level 3 PTE. Can occur after either a level 2 PTE miss or a level 1 kernel PTE miss.

Other All other exceptional conditions involving the TLB, including accesses to invalid pages (page faults) and page protection violations.

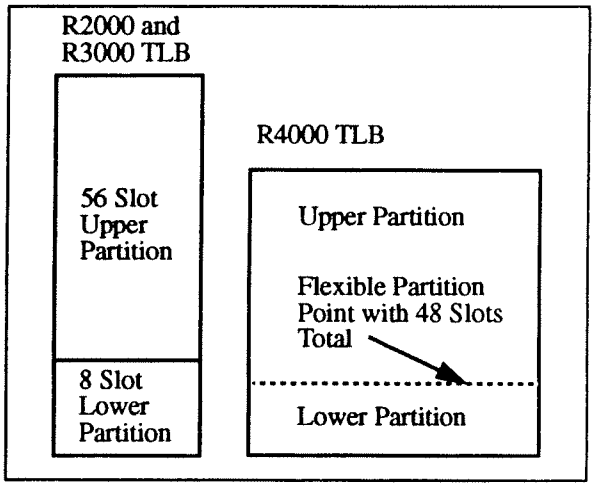

Figure 3 The MIPS TLBs

The MIPS R2000 and R3000 microprocessors both have an on-chip, 64-entry, fully-associative TLB that supports fixed lower and upper partitions of 8 and 56 slots, respectively. R2000/R3000 PTEs map 4 Kbyte pages.

The R4000 TLB is also on-chip and fully associative, but it has 48 slots which can hold pairs of consecutive PTEs. The partition between upper and lower slots is not fixed and can be set under software control. R4000 PTEs map pages that can vary from 4 Kbytes to 16 Mbytes in size.

virtual address space to a more limited, physical address space. Each task has its own L1 page table, which is maintained by machine-independent pmap code [Rashid et al. 1988]. Because the user page tables can amount to several megabytes, they are themselves paged. This is supported through L2 (or kernel) page tables, which also map other kernel data. Because kernel data is relatively large and sparse, the L2 page tables are also mapped. This gives rise to a 3-level page table hierarchy and a variety of different page table entry (PTE) types.

The R2000 processor contains a 64-slot, fully-associative TLB, which is used to cache recently-used PTEs. When the R2000 translates a virtual address to a physical address, the relevant PTE must be held by the TLB. If it is absent, the hardware invokes a trap to a software TLB miss handling routine that finds and inserts the missing PTE into the TLB by using various TLB control instructions. The R2000 supports two different types of TLB miss vectors. A special vector is used to trap on missing translations for level 1 user pages. This special vector is justified by the fact that level 1 user PTE misses are assumed to be the most frequent of all PTE types [DeMoney et al. 1986]. All other TLB miss types (such as those caused by references to kernel pages, invalid pages or write-protected pages) are vectored to a generic exception vector.[4] This broad collection of misses is the same as the KTLB misses described in Anderson's study [Anderson et al. 1991].

For the purposes of this study, we refine the definition of TLB miss types (see Table 4) to correspond to the page table structure implemented by OSF/1 and Mach 3.0. In addition to level 1 user TLB misses, we define four subcategories of KTLB misses (level 1 kernel, level 2, level 3 and other). Table 4 also shows our measurements of the time required to handle the different types of TLB misses. The wide differential in costs is primarily due to the two

4. In addition to these TLB related traps, the generic vector is shared among all other interrupts, exceptions and traps such as clock and device interrupts, arithmetic exceptions and system calls.

different miss vectors and the way that the OS uses them. Level 1 user PTEs can be retrieved within 20 cycles because they can be serviced by a highly-tuned handler inserted at the special vector. However, all other miss types require from about 300 to over 400 cycles because they are serviced by a generic handler that is inserted at the generic exception vector.

The R2000 TLB hardware supports partitioning of the TLB into two sets of slots. The lower partition is intended for infrequently referenced PTEs with high retrieval costs, while the upper partition is intended to hold more frequently used PTEs that can be re-fetched more quickly. The TLB hardware also supports random replacement of PTEs in the upper partition through a hardware index register that returns random numbers in the range 8 to 63. This effectively fixes the TLB partition at 8, so that the lower partition consists of slots 0 through 7, while the upper partition consists of slots 8 through 63.

The TLB of the MIPS R4000 differs from the R2000/R3000 TLB design in four significant ways (see Figure 3) [Kane et al. 1992]. First, the slot partition point is adjustable because the range of numbers generated by the random index register can be changed under software control. Second, level 1 kernel PTE misses are eligible to be serviced by the same special miss handler that fixes level 1 user PTE misses. This reduces their retrieval time by an order of magnitude. Third, page size in the R4000 is variable from 4 Kbytes to 16 Mbytes. Finally, the R4000 TLB has only 48 slots, but each slot can hold a pair of PTEs if they are contiguous in the virtual address space. In this study, we are primarily interested in the first two differences.

4.2 The TLB Management Problem

The challenge of TLB management is to minimize overall TLB miss handling time through the allocation of a fixed number of TLB slots to the different types of PTEs. As we shall see, optimal TLB management is dependent upon the structure of the operating system and its use of virtual memory. The management problem is also complicated by the varying costs to retrieve the different PTE types and by the different patterns with which they are accessed. For example, level 1 user PTEs are used on every memory reference by a user task, but level 2 PTEs are only needed when servicing level 1 PTE misses. Thus, level 2 PTEs are used less frequently but are much more costly to retrieve if they are missing from the TLB. Level 3 PTEs are only used when servicing level 2 or level 1 kernel PTE misses.

PTE Type	Ultrix	OSF/1	Mach 3.0	Mach 3.0 + AFS
Level 1 User	1,776,190	2,111,741	2,771,397	7,417,212
Level 1 Kernel	27,112	541,323	42,615	172,649
Level 2	420	1,449	23,448	206,257
Level 3	----------	56,365	45,493	135,764
Other	21,291	47,841	54,140	77,733
Total	1,825,013	2,758,719	2,937,093	7,855,140

Table 5 TLB Miss Counts in the Base Systems

These are the total occurrences of TLB misses for each of the different PTE types. The counts are the total obtained by running each of the workloads from Table 3.

So, level 3 PTE use is infrequent unless the level 2 and level 1 kernel PTE miss rates are high.

The varying retrieval costs and interrelationships between different PTE types are the basis for TLB management policies such as determining into which TLB partition a given PTE type should be placed, and what the replacement policy for evicting PTEs from the TLB should be. In a more flexible TLB design, such as in the R4000, an additional TLB management concern is the setting of the partition point.

4.3 TLB Management in the Baseline Systems

Each of the systems that we studied use similar TLB management policies. All three systems use the 8-slot lower

PTE Miss Type	Ultrix	OSF/1	Mach 3.0	Mach 3.0 + AFS
Level 1 User	2.13	2.53	3.33	8.90
Level 1 Kernel	0.48	9.55	0.73	2.96
Level 2	0.01	0.04	0.57	5.04
Level 3	----------	0.97	0.80	2.39
Other	0.52	1.16	1.32	1.89
Total	3.14	14.25	6.75	21.18

Table 6 TLB Miss Costs in the Base Systems

This table shows the total cost (in seconds) to service each of the different TLB miss types. The costs were obtained by combining the counts from Table 5 with the code path timings of Table 4.

TLB partition to hold level 2 PTEs and the 56-slot upper partition to hold all other PTE types. All systems use a first-in-first-out (FIFO) replacement policy in the lower partition and random replacement in the upper partition.

Although management policies between the systems are mostly identical, there are some differences. First, because Ultrix implements a 2-level page table structure, it has no level 3 PTEs. Second, the version of OSF/1 shipped by OSF actually doesn't use the lower partition at all; OSF/1 mixes all PTE types in the same 56 slots of the upper partition, throwing away 12.5% of the available TLB resource. We considered this a code bug and modified the OSF/1 miss handlers to utilize the lower partition according to the same policies used by Ultrix and Mach 3.0.

We measured TLB performance under these management policies for each of the four systems. Table 5 shows the TLB miss counts, while Table 6 shows the cost (in seconds) to service the different types of misses. Although all of these operating systems use similar TLB management policies and ran the same workload with identical binaries, their TLB performance varies widely.

The policies described above work well for Ultrix, which spends only 3.14 seconds (out of 281 seconds total) handling TLB misses. By separating level 2 PTEs from the other PTE types, the miss counts for this expensive PTE type are kept very low. Because of its monolithic-kernel structure, Ultrix is also able to store all of its program text and most of its data structures in unmapped space. This is evident by the low rates of level 1 kernel PTE misses. Note that Ultrix, with its 2-level page table structure, also completely avoids the miss handling costs associated with the level 3 PTEs of the OSF/1 and Mach 3.0-based systems.

OSF/1 spends a much greater amount of time (14.25 seconds) handling TLB misses. This is primarily due to a high component of level 1 kernel misses. Although OSF/1 is a monolithic-style kernel like Ultrix, it maps a greater portion of its data structures because of its dynamic kernel memory allocator that relies on virtual memory support.

The Mach 3.0-based systems also spend a greater deal of time handling TLB misses, either about 6 or 20 seconds depending on whether or not there is an additional user-level AFS server in the system. The large jump in level 1 user misses is due to the migration of OS services from the Mach 3.0 microkernel into the user-level UNIX server where they must be mapped. The Mach 3.0+AFS system also shows a large jump in level 2 misses due to the addition of the AFS server. We will explore the relationship between level 2 misses and the number of user-level OS servers more completely in subsequent sections.

The data in Table 5 and Table 6 demonstrate a breakdown of the old, Ultrix-style TLB management policies when carried over to the OSF/1 and Mach 3.0-based systems. In the next section, we will explore a collection of TLB management techniques that are more effective for the OSF/1 and Mach 3.0-based systems.

5.0 Better TLB Management

When we set out to formulate new TLB management techniques, we asked ourselves several questions about the existing policies and how they might be changed. These questions are listed below and the answers we arrived at can be found in the following subsections.

- *What should the PTE placement policy be? That is, into which partition (upper or lower) should the 4 various PTE types be placed?*
- *What policies should guide adjustment of the partition point? What factors change the position of the optimal partition point?*
- *Which replacement policies are most effective for the two different partitions? That is, when a new PTE is inserted into the TLB, should the PTE that is evicted be selected by a FIFO or a random policy?*
- *What are the benefits of reducing miss costs for PTE types other than the most frequently used level 1 user PTEs?*

5.1 PTE Placement and Partitioning

All of the baseline systems place level 2 PTEs into the lower partition and all other PTEs in the upper partition. Level 1 user PTEs, which have a low retrieval cost, are separated from level 2 PTEs because they have a much higher retrieval cost. However, it is unclear why the baseline systems mix the costly-to-retrieve level 1 kernel and level 3 PTEs with level 1 user PTEs. Furthermore, if mixing costly-to-retrieve mappings with level 1 user PTEs is acceptable, it is unclear if PTE partitioning is required at all.

To determine the effects of other PTE placement choices, we modified the miss handlers of the baseline systems to implement other meaningful policies. The results are shown in Table 7. Policy A is identical to that implemented by the baseline systems. The importance of some

sort of partitioning is shown by Policy D, where all PTEs are mixed together, and which demonstrates very poor performance. At first glance, the baseline policy A appears to be the most desirable. However, note that with policies B and C, the lower partition was not permitted to grow beyond 8 slots to accommodate the additional PTE types allocated to this partition.

To see if the performance of policies B and C would improve with a larger lower partition, we modified the baseline miss handlers to vary the partition point from its fixed location at 8. Although the R2000 TLB only *efficiently* supports a partition point of 8, it is possible to overcome this limitation by generating the random replacement index in software. Although the resulting handlers are slowed down to the extent that this technique doesn't make sense for usual system operation, it does enable us to emulate partition point movement for an architecture, like the R4000, that better supports this operation in hardware. Figure 4 shows the effect of changing the partition point for a Mach 3.0 system that implements policy B. The graph shows the varying cost of PTE misses from the upper and lower TLB partitions as the partition point is changed. The sum of these two competing costs is plotted as a third curve that shows an optimal region in which the partition point would best be set.

Figure 5 shows the results for this same experiment performed for PTE placement policies A, B and C. Only the total curves are shown. The constant line that represents policy D (no partition) is off the scale of this plot, and is therefore not shown. Note that each of the policies have

Policy	PTE Placement	Cost
A	Level 2 PTEs in lower partition. All other PTEs in upper partition.	6.46
B	Level 2 and 3 PTEs in lower partition. Level 1 user and kernel PTEs in upper partition.	9.62
C	All PTEs in lower partition, except for level 1 user PTEs. which are placed in upper partition.	8.06
D	No partitioning at all.	55.15

Table 7 Alternate PTE Placement Policies

This table shows the affect of alternate PTE placement policies on TLB management cost. The cost is the total time spent (in seconds) to handle TLB misses for the Mach 3.0 system running all of the benchmarks listed in Table 3. The partition point is fixed at 8 slots for the lower partition and 56 slot for the upper partition.

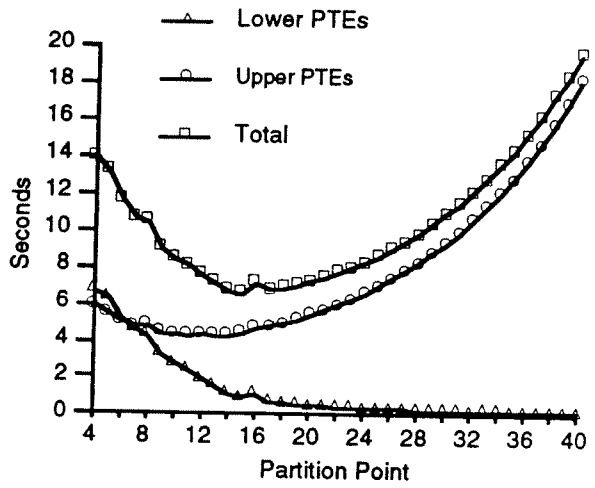

Figure 4 Changing the TLB Partition Point

This experiment was performed on the Mach 3.0 system implementing PTE placement policy B (see Table 7). Each data point represents a complete run of the benchmark suite with the partition statically fixed at given point for the duration of the run. The x-axis (partition point) shows the number of slots allocated to the lower partition. The upper partition always consists of (64 - partition point) slots.

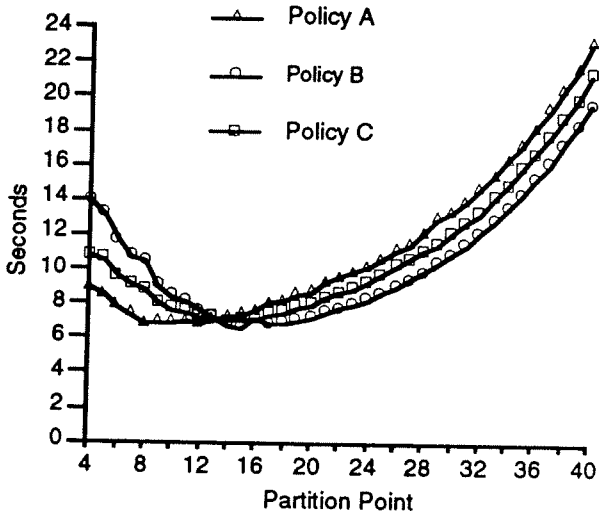

Figure 5 TLB Partitioning for Different PTE Placement Policies

This is the same experiment as that of Figure 4, except with different PTE placement policies (see Table 7). Only the total TLB management costs are shown. The total cost for policy D (no partition) is off the scale of the plot at 55.15 seconds.

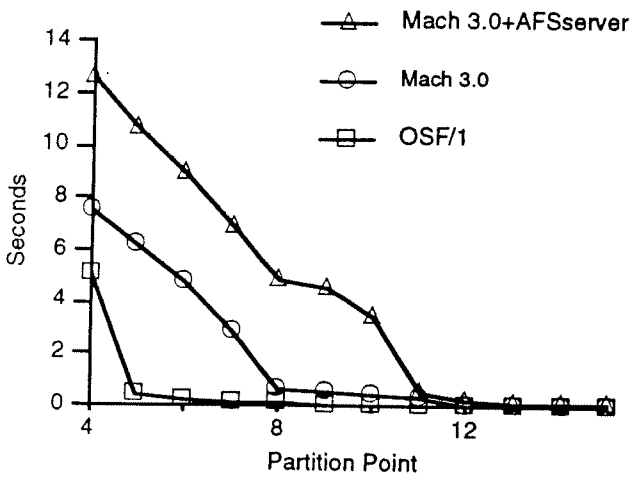

Figure 6 Optimal TLB Partitioning for Different Operating Systems

This graph shows the cost to handle level 2 PTE misses for the three systems using PTE placement policy A. The partition point is varied as described in Figure 4.

different optimal points, but at these optimal points, the performance is roughly the same for each system. From this we can conclude that the most important PTE placement policy is the separation of level 1 user and level 2 PTEs (to avoid the very poor performance of policy D). However, the differences between the other PTE placement policies A, B and C are negligible, provided that the partition point is tuned to the optimal region.

Our studies show that the optimal partitioning point is also dependent upon OS structure. To illustrate this point, we ran our benchmark suite under the OSF/1, Mach 3.0 and Mach 3.0+AFS systems and measured the time to service level 2 PTE misses. The results are shown in Figure 6. These curves illustrate that as operating system services are migrated from kernel space (in OSF/1) to user space (in the Mach 3.0-based systems), the reliance upon level 2 PTEs increases. This is because each user-level task requires a minimum of 3 level 2 PTEs to cover its text, data and stack segments. The OSF/1 system requires only 5 TLB slots[5] for level 2 PTEs in order to avoid TLB thrashing. However, the Mach 3.0+AFS system requires 6 additional TLB slots to hold level 2 PTEs for its user-level UNIX and AFS servers, bringing the total to 11.

5. 2 PTEs are used to map kernel data structures and 3 additional PTEs are required for the UNIX process running in an OSF/1 task.

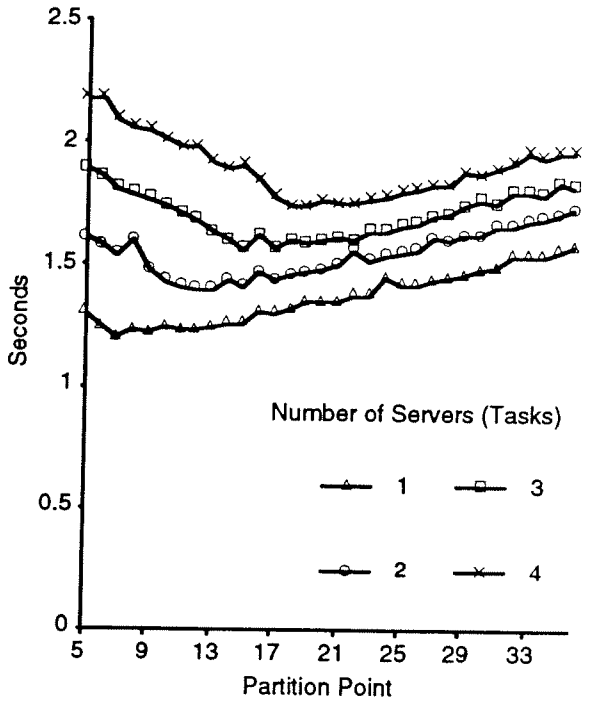

Figure 7 TLB Partitioning under Multi-server Operating Systems

This graph shows total TLB management costs for Mach 3.0 running a workload that emulates a multi-server system by passing a token among different numbers of user-level tasks.

A key characteristic of microkernel system structuring is that logically independent services reside in separate user-level tasks that communicate through message passing. The degree of service decomposition can be limited for performance reasons. However, careful software management of the TLB can extend the degree to which separate services can coexist in a system before performance degrades to an unacceptable level. To illustrate this more clearly, we constructed a workload that emulates the interaction between servers in a multi-server microkernel OS. In this workload, a collection of user-level tasks mimic the behavior of communicating OS servers by passing a token between each other. The number of servers and the number of pages that each server touches before passing the token along can be varied. Figure 7 shows the results of running this multi-server emulator on the Mach 3.0 kernel. With each additional server, the optimal partitioning point moves farther to the right. A system that leaves the partition point fixed at 8 will quickly encounter a performance bottleneck due to the addition of servers. However, if the TLB partition point is adjusted to account for the number

Workload	Optimal Partition Point	TLB Management Cost
Ousterhout	18	1.27
spell	25	0.11
gcc	14	4.39
io	32	0.43

Table 8 Optimal TLB Partitioning for Different Workloads

These data were obtained by running the same experiment as described in Figure 4 (Mach 3.0 + policy B). Only the management cost (in seconds) for the *optimal* partition point of each benchmark is shown.

Workload	Fixed Partitioning	Static Partitioning	Dynamic Partitioning
Ousterhout	3.92	1.27	1.11
spell	0.28	0.11	0.11
gcc	4.94	4.39	4.28
io	1.30	0.43	0.43

Table 9 Different TLB Partitioning Schemes

This table compares the total TLB management costs (in seconds) when fixing the partition at 8, when setting it to the static optimal point, and when using dynamic partitioning. The PTE placement policy is B.

of interacting servers in the system, a much greater number of servers can be accommodated. Nevertheless, note that as more servers are added, the optimal point still tends to shift upwards, limiting the number of tightly-coupled servers that can coexist in the system. This bottleneck is best dealt with through additional hardware support in the form of larger TLBs or miss vectors dedicated to level 2 PTE misses.

In addition to OS structure, the optimal TLB partition point is also dependent upon workload. This is shown in Table 8 where we determined the optimal partition for each of the four different programs in our benchmark suite. Note that the optimal point is often very far away from the point fixed by the R2000 hardware (at 8).

5.2 Dynamic Partitioning

We have shown that the best place to set the TLB partition point varies depending upon the PTE placement policy, operating system structure, and workload. Given knowledge of these factors ahead of time, it is possible to determine the optimal partition point and then statically fix it for the duration of some processing run. However, although an operating system can control PTE placement policy and have knowledge of its own structure, it can do little to predict the nature of future workloads that it must service. Although system administrators might have knowledge of the sort of workloads that are typically run at a given installation, parameters that must be tuned manually are often left untouched or are set incorrectly.

To address these problems, we have designed and implemented a simple, adaptive algorithm that dynamically self-tunes the TLB partition to the optimal point. The algorithm is invoked after some fixed number of TLB misses at which time it decides to move the partition point either up, down, or not at all. It is based on a hill-climbing approach where the objective function is computed from the two most recent settings of the partition point. At each invocation, the algorithm tests to see if the most recent partition change resulted in a significant increase or decrease in TLB miss handling costs when compared against the previous setting. If so, the partition point is adjusted as appropriate. This algorithm tends to hone-in on the optimal partition point and tracks this point as it changes with time.

We tested this algorithm on each of the benchmarks in our suite and compared the resultant miss handling costs

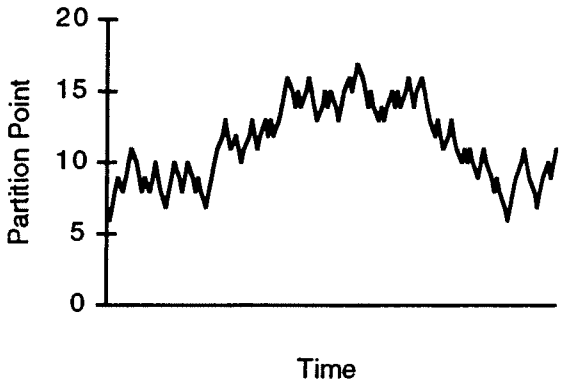

Figure 8 Dynamic TLB Partitioning during gcc

This graph shows the movement of the TLB partition with time while running gcc on a system that implements the dynamic, adaptive partitioning algorithm.

against runs that fix the partition at 8 and at the static optimal point for the given benchmark. The results are shown in Table 9. Note that dynamic partitioning yields results that are at times slightly better than the static optimal. To explain this effect, we performed another experiment that records the movement of the TLB partition point during the run of the gcc benchmark. The results (see Figure 8) show that the optimal partition point changes with time as a benchmark moves among its working sets. Because the dynamic partitioning algorithm tracks the optimal point with time, it has the potential to give slightly better results than the static optimal which remains fixed at some "good average point" for the duration of a run.

The invocation period for the dynamic partitioning algorithm can be set so that its overhead is minimal. It should be noted, however, that there is an additional cost for maintaining the TLB miss counts that are required to compute the objective function. Although this cost is negligible for the already costly level 2 and level 3 PTE misses, it is more substantial for the highly-tuned level 1 PTE miss handler.[6] Hardware support in the form of a register that counts level 1 misses could help to reduce this cost.

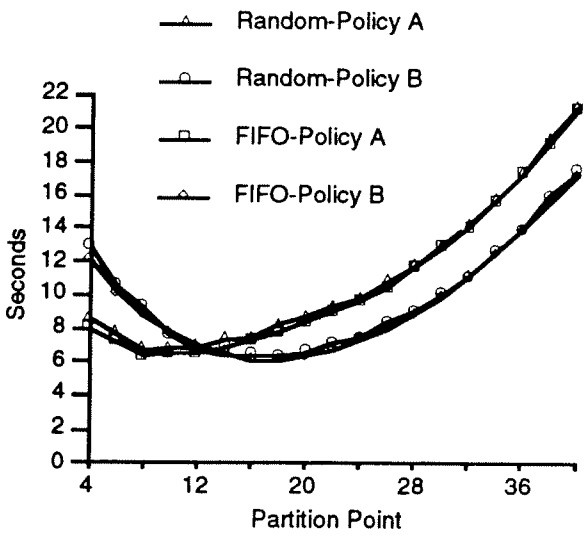

Figure 9 Replacement Policies

This graph shows the performance of different replacement policies (random or FIFO) for the Mach 3.0 system implementing PTE placement policies A or B.

5.3 Replacement Policies

The baseline systems use a FIFO replacement policy for the lower partition and a random replacement policy for the upper partition when selecting a PTE to evict from the TLB after a miss. To explore the effects of using other replacement policies in these two partitions, we modified the miss handlers to try other combinations of FIFO or random replacement in the upper and lower partitions. The results of this experiment are shown in Figure 9. For these workloads, differences between the replacement policies are negligible over the full range of TLB partition points, indicating that the choice of replacement policy is not very important.

5.4 Reducing Miss Costs

The design of the R2000 TLB was influenced by previous research results that showed level 1 user PTE misses to be by far the most frequent type of TLB miss, occurring greater than 95% of the time [Clark et al. 1985]. This was the justification for providing special vectoring support only for this miss type in the R2000 TLB [DeMoney et al. 1986]. Although our measurements for the older-style Ultrix OS support this assumption (recall Table 5), it

OSF/1	Level 1 User	Level 1 Kernel	Level 2	Level 3
Counts	2,111,741	541,323	1,449	56,365
% of Total Misses	76.55%	19.62%	0.05%	2.04%
Previous Cost Per Miss (cycles)	20	294	407	286
New Cost Per Miss (cycles)	20	20	50	100
Previous Total Cost (seconds)	2.53	9.55	0.04	0.97
New Total Cost (seconds)	2.53	0.65	0.00	0.34
Total Time Saved	0.00	8.90	0.04	0.63

Table 10 Reducing PTE Miss Costs for OSF/1

These computations are based on the counts of Table 5 for OSF/1. We estimated the minimal achievable miss costs for each of the PTE types and recomputed the resultant total handling times. The category of "other misses" is too complex (and infrequently occurring) to justify any attempt at reducing its handling cost, and is therefore not shown here.

6. Maintaining a memory-resident counter in the level 1 miss handler requires a load-increment-store sequence. On the R2000, this can require anywhere from 4 to 10 cycles, depending on whether the memory reference hits the data cache. This is a 20% to 50% increase over the 20-cycle average currently required by the level 1 miss handler.

Mach 3.0 + AFS	Level 1 User	Level 1 Kernel	Level 2	Level 3
Counts	7,417,212	172,649	206,257	135,764
% of Total Misses	92.60%	2.16%	2.58%	1.70%
Previous Cost Per Miss (cycles)	20	294	407	286
New Cost Per Miss (cycles)	20	20	50	100
Previous Total Cost (seconds)	8.90	2.96	5.04	2.39
New Total Cost (seconds)	8.90	0.21	0.62	0.81
Total Time Saved	0.00	2.75	4.42	1.58

Table 11 Reduced Miss Costs in Mach 3.0 +AFS

These computations are the same as described in the caption for Table 10, but they apply to the Mach 3.0 + AFS system.

begins to break down in the newer OSF/1 and Mach 3.0 systems.

We wanted to quantify the benefit of reducing miss handling costs for other PTE types to determine if the additional hardware or software needed to accomplish this is justified. Our computations are shown in Table 10 (for the OSF/1 system) and Table 11 (for the Mach 3.0 + AFS system). The OSF/1 system benefits substantially from a reduced level 1 kernel miss cost due to its heavy use of mapped kernel data structures. The Mach 3.0 + AFS system benefits mostly from a reduced level 2 miss cost due to its increased degree of service decomposition. Although both systems benefit from a reduced level 3 miss cost, the improvements are slight and probably do not justify the effort required to reduce the handling time for this miss type.

Miss costs can be reduced either through better hardware support or through more careful coding of the software miss handlers. The R4000 lowers the cost of level 1 kernel misses by allowing them to be serviced by the special TLB miss vector. So, we assumed that the R2000 could handle level 1 kernel misses in the same way the R4000 does and estimated the cost to be the same as the level 1 user miss cost (about 20 cycles). For level 2 and level 3 misses, we believe that a software approach could be used to reduce these costs to about 50 and 100 cycles, respectively. This would be implemented by testing for level 2 and level 3 PTE misses at the *beginning* of the generic exception vector, before invocation of the code that saves register state and allocates a kernel stack. Although this state saving code is required to handle more complex interrupts, exceptions and kernel traps, it is not needed for TLB miss handlers which simply index into the appropriate page table for a PTE and insert it into the TLB. We are currently implementing this in the OSF/1 and Mach 3.0-based systems to test the approach.

6.0 Summary

TLBs that require software management are becoming more commonplace, implying that operating system writers are charged with the new duty of writing TLB miss handlers. This task is complicated by the large number of management policy choices and the special requirements of newer generation operating systems like OSF/1 and Mach 3.0.

Our studies show that the most important management policies regard the partitioning of TLB slots to separate level 1 user PTEs from level 2 PTEs. Other policies governing PTE placement or replacement are mostly irrelevant, provided that the partition point is tuned to the optimal region for the choices selected. The optimal partition point varies not only with management policy, but also with operating system structure, workload and time. We have shown that partition point tuning can be performed automatically through a simple, adaptive algorithm that extends the number of user-level servers that a microkernel OS can efficiently support.

Reducing the cost of handling individual PTE misses also proves to be an effective TLB management technique. Although older operating systems only justify special treatment of the level 1 user PTE misses, the newer OSF/1 and Mach 3.0 systems also show high levels of level 1 kernel and level 2 PTE misses. Special hardware support or more careful coding of the miss handlers for these PTE types is worthwhile.

Although our studies were performed on an R2000-based machine, our results and techniques extend to the MIPS R4000 and to other multi-server operating systems such as Windows NT [Custer 1993].

7.0 Acknowledgments

We thank Mary Thompson for her patience and assistance as we assembled our OSF/1 and Mach 3.0 systems. We

also extend thanks to Richard Draves and Brian Bershad for helping us to interpret our results.

8.0 References

[Accetta et al. 1986] M. Accetta, R. Baron, et al. *Mach: A new kernel foundation for UNIX development*, In Summer 1986 USENIX Conference, USENIX, 1986.

[Anderson et al. 1991] T.E. Anderson, H.M. Levy, B.N. Bershad and E.D. Lazowska. *The interaction of architecture and operating system design*, In Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, Santa Clara, California, ACM, 108-119, 1991.

[Clapp et al. 1986] R.M. Clapp, L.J. Duchesneau, R.A. Volz, T.N. Mudge, and T. Schultze, "Toward Real-time Performance Benchmarks for Ada," *Communications of the ACM*, vol. 29, no. 8, Aug. 1986, pp. 760-778

[Clark et al. 1985] D.W. Clark and J.S. Emer. "Performance of the VAX-11/780 translation buffer: Simulation and measurement." *ACM Transactions on Computer Systems* **3** (1): 31-62, 1985.

[Custer 1993] H. Custer. *Inside Windows NT*. Redmond, Washington, Microsoft Press, 1993.

[DeMoney et al. 1986] M. DeMoney, J. Moore and J. Mashey. *Operating system support on a RISC*, In COMPCON, 138-143, 1986.

[Digital 1992] Digital. *Alpha Architecture Handbook*. USA, Digital Equipment Corporation, 1992.

[Guillemont et al. 91] M. Guillemont, J. Lipkis, D. Orr, and Marc Rozier. *A second-generation micro-kernel based UNIX; Lessons in performance and compatibility*. In USENIX Winter 1991 Proceedings, Dallas, Texas, USENIX.

[Hewlett-Packard 1990] Hewlett-Packard. *PA-RISC 1.1 Architecture and Instruction Set Reference Manual*. Hewlett-Packard, Inc., 1990.

[Kane et al. 1992] G. Kane and J. Heinrich. *MIPS RISC Architecture*. Prentice-Hall, Inc., 1992.

[McKusick et al. 1984] M.K. McKusick, W.N. Joy, S.J. Leffler and R.S. Fabry. "A fast file system for UNIX." *ACM Transactions on Computer Systems* **2** (3): 181-197, 1984.

[Nagle et al. 1992] D. Nagle, R. Uhlig and T. Mudge. Monster: A Tool for Analyzing the Interaction Between Operating Systems and Computer Architectures. The University of Michigan. 1992.

[Ousterhout 1989] J. Ousterhout. "Why aren't operating systems getting faster as fast as hardware." *WRL Technical Note* (TN-11): 1989.

[Rashid et al. 1988] R. Rashid, A. Tevanian, et al. "Machine-Independent Virtual Memory Management for Paged Uniprocessor and Multiprocessor Architectures." *IEEE Transactions on Computers* **37** (8): 896-908, 1988.

[Satyanarayanan 1990] M. Satyanarayanan. "Scalable, secure, and highly available distributed file access." *IEEE Computer* **23** (5): 9-21, 1990.

[SPEC 1991] SPEC. *The SPEC Benchmark Suite*. 1991.

[Wilkes et al. 1992] J. Wilkes and B. Sears. A comparison of protection lookaside buffers and the PA-RISC protection architecture. HP Laboratories. 1992.