In particular, we studied a hardware component that is most likely to be affected by the multi-tasking environment: the translation look-aside buffer (TLB). The purpose of the TLB is to speed up the virtual-to-physical address translation for the virtual memory system by storing the most recently used translation information for a process. Because accessing information from this TLB usually takes one clock cycle, if this buffer can capture the majority of the translation information most of the time, the performance of the system will be greatly enhanced. However, since, as indicated above, the hardware designers configure their TLBs according to single-process experiment, the TLBs may not be able to function well as expected in a multi-tasking environment.

The architecture we are working on in this study is the Intel i486 microprocessor, which has an on-chip 32-entry TLB, organized in 4-way set associative [Intel92]. The i486 microprocessor is unique because its TLB is managed by hardware instead of software, as is the case with previous study [Uhlig 94a]. In addition, the Intel i486 based machine has shared a significant portion of personal computer market, where there are a large amount of software available, including multi-tasking programs, such as Linux and Window 95.

Therefore, this work explores the performance of the i486's TLB in supporting the address translation under a multi-tasking environment we built. To measure performance, we implemented a trap-driven TLB simulator in Mach 3.0, a microkernel operating system, on an i486-based personal computer. Moreover, to magnify the importance of address translation, we did not use simple test programs, such as SPEC92, which are usually used to evaluate system performance. Instead, we developed client-server style workloads that stress the TLB by switching between several tasks frequently. This report discusses the implementation of the TLB simulator and presents the results from these test workloads.

The organization of the remainder of this report is as follows: Section 2 outlines related work. Section 3 describes the design of the trap-driven simulator. Section 4 presents the preliminary results and analyses of our experiments. Section 5 presents some concluding remarks and proposes of our future work.

2 Related Work

Because this work is an evaluation of hardware performance under a multi-tasking environment, previous similar studies focusing on hardware performance under such environments deserve review. In addition, because the experimental method used in this work is critical to the success of this kind of evaluation, previous studies involving similar methods are also reviewed here.

2.1 Similar Multi-tasking Evaluations

Only in the last ten years have researchers begun to study hardware performance under multi-tasking environments or comparable software structures. In these recent studies, two kinds of software structures are commonly investigated: (1) a new-generation operating system, such as a microkernel OS and (2) a new application, such as X window and multi-media applications. In these studies, furthermore, the evaluation of hardware performance is primarily focused on memory system performance.

Studies of a microkernel OS, such as Mach 3.0, have consistently demonstrated that cache and TLB misses occur more frequently with the microkernel OS than with traditional OS structures [Chen93a, Nagle93, Nagle94]. A miss is, simply, an event which caches or TLBs do not contain the data requested by central process unit (CPU), and therefore CPU has to spend more clock cycles to get the data from main memory. The more misses are, the worse the cache and TLB performance is degraded because the purpose of the cache and TLB design is to satisfy most of the CPU's data requests to avoid expensive main memory experiment. Because Tapeworm is capable of capturing multi-tasking and OS kernel activities, we modified it into the monitoring tool for our work.

This work extends previous work using Tapeworm. In the work of Uhlig et al., Tapeworm was used to study instruction caches and software-managed TLBs in MIPS R3000-based systems. Implementing Tapeworm on i486-based machines, which employ hardware-managed TLBs, represents a new area of study that will also (1) demonstrate the portability of the Tapeworm and (2) allow us to compare the performance of Tapeworms on different underlying hardware platforms. In addition to that, as the most popular general purpose machine, i486-based machines support more intensively interactive workloads than do MIPS R3000-based machines. These workloads are requiring more operating system services because they do more input/output activities, but the performance of these workloads on the i486 machines are still unknown. An third contribution of this study, therefore, is to give initial performance evaluation for these interesting and frequently more popular workloads.

3 Experimental Method

We tested our trap-driven, Tapeworm-based TLB simulator on a Gateway 2000 i486-based personal computer with a Mach 3.0 operating system. Using this trap-driven TLB simulator, we can count the number of TLB misses and hence evaluate the TLB performance and design trade-offs in TLB structures under a multi-tasking environment.

In this section, we describe our Tapeworm-based experimental method in detail. First, we describe the software environment of our experiments, which is composed of several user-level programs and the underlying operating system, Mach 3.0. Because Mach 3.0 is the very program which makes the multi-tasking environment in our experiment sophisticated, we focus on explaining its structure in the section below. In particular, we describe its module and data structure that we used for our experiments, namely PMAP and *pv_list*. Second, we discuss the hardware environment of our experiments. We focus primarily on the memory management unit (MMU) of the i486 microprocessor. In particular, we discuss the i486's two-level page table structure and hardware-managed TLB.

After having described our software and hardware experimental environments, we explain the Tapeworm algorithm in detail. Lastly, we mention some problems we encountered when we were implementing Tapeworm on our i486 machine and propose some solutions.

3.1 Mach 3.0 Microkernel

Mach 3.0 represents a new generation of OS, which is called "microkernel," as opposed to the traditional monolithic OS, such as UNIX BSD. In this section, we first describe the structure of a microkernel OS by using Mach 3.0 as an example. Then we describe those module and data structure in Mach 3.0 which we used for our experiment, the PMAP module and the *pv_list* data structure.

3.1.1 Monolithic vs. Microkernel

In the traditional operating system design, all the OS related codes are implemented in a single address space. This way of implementation is rather straight-forward and allows programmers to easily begin writing the codes. However, as the OS is required to provide more and more functions and services, the OS code grows huger and may contain several times the amount of code it initially may have had. This growth makes mechanism the i486 provides for virtual memory system. which is called "two-level page table." Also. we describe an additional microprocessor component the i486's MMU uses to speed up this mechanism. which is the on-chip, translation-look-aside buffer (TLB).

3.2.1 Two-level page table

The i486 provides a paging mechanism to support virtual memory multi-tasking operating systems. The i486 uses a page table to translate virtual addresses to physical addresses. This page table is hierarchically composed of three components: the *page directory*, the *page tables*, and the *page frames*. Every virtual address requires two-level translation to get its corresponding physical address, i.e. from the *page directory* to the *page subdirectories* and from the *page tables* to the *page frames*. Therefore, this page table is considered two-level page table. (Figure 3.1) All memory-resident elements of this page table are the same size, 4k bytes.

3.2.2 Hardware-managed TLB

To speed up the virtual-to-physical address translation, the i486's MMU employs a hardware-managed TLB to cache the address translation for the most recently accessed pages. This TLB has 32 entries that are organized as 8 sets with 4-way set associativity. The i486's TLB does not distinguish code and data address translations; it does not reserve special room for kernel address space; and it does not use process identifiers (PID) to distinguish address spaces. If a TLB miss occurs, the central process unit (CPU) will search for the corresponding mapping entry in the page table and place it directly in the TLB without informing the OS.

The OS, however, may modify a page table associated with an active process as result of either swapping pages, writing pages, or changing page protections. If the OS modifies a page table, it has to flush the TLB to maintain consistency between the TLB and the page table. To flush the TLB, the OS needs to reload a process' page directory base address into a control register in the CPU (Fig 3.1) to invalidate all TLB entries. Because the OS may modify the page table frequently, this flushing TLB can occur many times during a program execution in the i486.

3.3 The Trap-Driven TLB Simulator—Tapeworm

Since the i486's TLB is managed by hardware, all TLB misses are viewed as hardware events that are transparent to software. In spite of this, some operations, such as a page fault, can help expose TLB misses to the OS. The basic idea with Tapeworm is to force a page fault happen on each TLB miss so that all TLB misses are visible to the OS kernel. To do this, all pages are first marked invalid except those pages whose PTEs are held in a specific kernel-controlled data structure. We use this specific data structure to simulate the TLB, and can vary the size, set associativity, etc., of this emulated TLB structure for different study purposes.

The i486's page table entry (PTE) is a one-word item (32 bits) and is organized as shown on Fig. 3.2. The lowest bit of the PTE is the Present (P) bit, which indicates whether this PTE is valid or not. When a TLB miss occurs, hardware automatically searches the page table for the PTE corresponding to this missing virtual address. To recall, a miss is an event which the TLB does not hold the data requested by CPU. If the P bit of this PTE is 1, this PTE is valid, the hardware simply stores this PTE into the TLB and resumes normal execution. All of these actions are invisible to the OS. If the P bit is 0, the PTE is not valid, which indicates the page pointed by this PTE is not residing in the physical memory and a page fault exception needs to be generated. In a page fault exception, the OS is responsible for bringing in the faulting page from