行政院國家科學委員會專題研究計畫　成果報告

先進轉換搜尋緩衝器之研究與設計

計畫類別：個別型計畫
計畫編號：NSC92-2213-E-009-067-
執行期間：92 年 08 月 01 日至 93 年 07 月 31 日
執行單位：國立交通大學資訊工程學系(所)

計畫主持人：陳昌居

報告類型：精簡報告

處理方式：本計畫可公開查詢

中　華　民　國 93 年 12 月 16 日

行政院國家科學委員會補助專題研究計畫 V 成果報告
□期中進度報告

先進轉換搜尋緩衝器之研究與設計

計畫類別：V 個別型計畫　　□ 整合型計畫
計畫編號：NSC 92－2213－E009－067
執行期間：　92 年　8 月　1 日至　93 年　7 月 31 日

計畫主持人：陳昌居
共同主持人：
計畫參與人員：鄭緯民、陳仕偉、張繼文

成果報告類型(依經費核定清單規定繳交)：V 精簡報告　□完整報告

本成果報告包括以下應繳交之附件：
□赴國外出差或研習心得報告一份
□赴大陸地區出差或研習心得報告一份
□出席國際學術會議心得報告及發表之論文各一份
□國際合作研究計畫國外研究報告書一份

處理方式：除產學合作研究計畫、提升產業技術及人才培育研究計畫、列管計畫及下列情形者外，得立即公開查詢
□涉及專利或其他智慧財產權，□一年□二年後可公開查詢

執行單位：交通大學資訊工程系

中　華　民　國　93　年　12　月　15 日

The Study and Design of An Advanced Translation Lookaside Buffer

Principal investigator: Chang-Jiu Chen, Department of Computer Science and Information Engineering, National Chiao Tung University

Abstract

Address translations from virtual addresses to physical addresses are widely considered as one of the most important issue for memory system performance. In order to improve the performance, the *Translation Lookaside Buffer* (*TLB*) is used. If any misses occur, the performance of the processor will seriously degrade [7]. To reduce such misses, lots of methodologies are proposed. However, only a few works focus on reducing the miss rates in context switching under multiprogramming environment. This paper presents a novel mechanism for the TLB to reduce the miss rate in context switching with 32K page size. All simulations were done with modified SimpleScalar 3.0d tool suite and SPEC95 benchmarks. The results show that this design can be very useful for multiprogramming environment under specific conditions.

INTRODUCTION

In order to support larger memory requirements for modern applications, it's important for modern operating systems (OS) to provide the virtual memory mechanism. However, a lot of time is needed to fetch the translations from main memory. To reduce the cost of address translations, the translation lookaside buffers (TLBs) are widely implemented inside the processor [3].

In order to reduce the TLB miss rate, most processors increase the size (total entries) of TLBs with fully or set associative. For example, Intel® puts a 512-entry 4KB-page TLB inside the Intel® Pentium® !!! Processor [4]. Furthermore, some processors even try to provide multi-level TLBs, such as 2-level ITLB/DTLB design on the highest-end Intel® Itanium® 2 Processor [5]. In addition, some processors begin to provide larger page sizes to increase the TLB span, such as 2MB or even 4MB page size on all new Intel® x86 Processors after the Pentium® Pro Processor [6]. However, to provide several page sizes, most commercial designs put several TLBs inside the processor for each individual size. Recently, several interesting mechanisms are proposed to support multi-page size processor. Lee et al. propose a novel banked-promotion TLB structure to support two page sizes dynamically. Four 4KB pages can be promoted to a 16KB superpage dynamically. To support such mechanism, an interesting two-bank TLB is designed. The heuristic promotion algorithm can promote four consecutive entries from small-page TLB bank to large-page TLB bank [2,9]. In addition, Swanson et al. present a novel memory controller which can aggressively create superpages even from non-contiguous and unaligned regions of physical memory space [13]. Channon et al. presents the re-configurable partitioned TLBs to improve the TLB performance [1].

Though lots of these new mechanisms are proposed, just only a few studies are focused on the TLB entries prefetching/preloading. Saulsbury et al. introduces a interesting mechanism, called the *Recency-based TLB Preloading* (RP), to prefetch the TLB entry according to the 'Recency' of the referenced pages [10]. The mechanism maintains the 'Recency Stack' via augmented translation table entry in memory and the TLB inside the processor according to the recently referenced pages. Thus the next possible referenced page number can be prefetched. However, the mechanism may increase the memory traffic and the TTE should do some changes to store the stack pointers for the link-list. To solve these possible problems, Kandiraju et al. propose a new prefetching technique, called the *Distance Prefetching* (DP), according to the recently referenced pages 'distance (stride)' [8]. The mechanism maintains a table to keep the track of differences between successive address references and do prefetching according to the predicted distance. The paper also compares other possible prefetching techniques borrowing ideas from the cache prefetching techniques, such as *Sequential Prefetching* (SP), *Arbitrary Stride Prefetching* (ASP) and the *Markov Prefetching* (MP). Because of the implementation costs, we'll focus on the studying of the SP and DP in this paper.

However, it is widely known that the frequent context switching under the multiprogramming environment also impact the overall performance very seriously. Though it may be one of the most serious 'performance killer' for the TLB performance, few studies really consider this issue very seriously. That may be because it's very hard to model and estimate the context switching activities caused by the OS and it's also hard to consider this issue without considering the OS issue first. In this paper, we propose a novel and easy implement TLB mechanism to reduce the miss rate caused by the context switching. To support this new novel mechanism, the OS part should be modified a little. The new mechanism can be easily implemented on future high performance systems.

All the simulations will be done by the modified SimpleScalar Version 3.0d tool suite [11] provided by the SimpleScalar LLC with SPEC95.

The rest of the paper is organized as follows. Section 2 demonstrates the architecture of the new novel TLB. Section 3 gives the simulation results. Finally, the conclusions and future work discussion are summarized in the Section 4.

ARCHITECTURE OF THE NOVEL TLB

This section describes in detail of the new novel TLB structure and mechanism we proposed for processors with 32KB-page size. The new novel design can be implemented not only in contemporary processors but future high performance processors comprised with billion of transistors. Furthermore, the mechanism is especially suitable to be implemented on processors with larger addressing space than current processors with just 32-bit addressing ability.

Overview

Figure 1 shows in detail our proposed novel TLB structure to reduce the miss rate in context switching with 32-KB page size. We select 32 KB as our default page size because we expect that processors tend

to provide larger page sizes with larger addressing space in the future. Furthermore, we'll have other new study for TLB to provide superpages with page promotion mechanism.

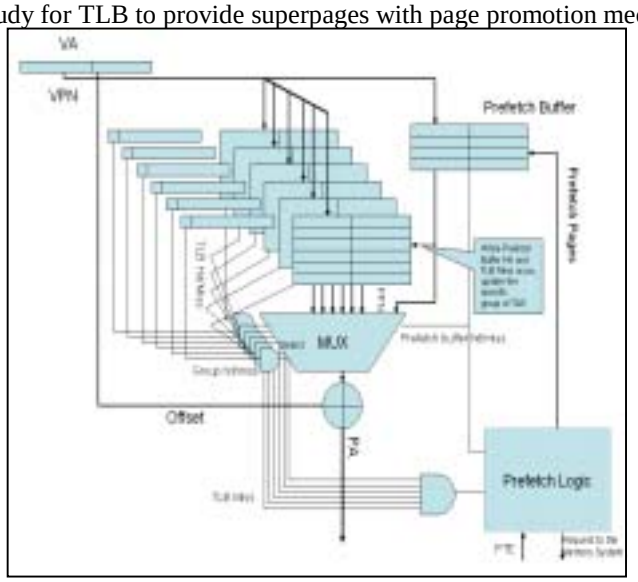


Figure 1. The Novel TLB Structure

The proposed structure consists of the following parts – 32 TLB banks with group tags to store the address translations, a multiplexer to select specific TLB banks, a prefetch buffer to store the prefetching entries, and the prefetch logic to activate the prefetching mechanism. Each TLB bank has 32 entries and it can be implemented with CAM (content addressable memory) which is commonly used in the traditional TLB. Furthermore, each TLB bank is implemented with fully associativity with the LRU entry replacement policy. That means each bank can be easily implemented the same as traditional design. Thus there are totally 1024 entries in this new design. However, we can easily find that other new processors also try to increase the total entries of their TLB (TLB size) to reduce the possibilities of the TLB misses, such as 512-entry TLB with 4KB page size support on Intel® Pentium® !!! Processor [4]. Furthermore, with larger page sizes, the cost is decreased. Except the 32 TLB banks, there are also 32 extra registers to store the bank tag for each bank as shown in Figure 1. The register contains task tag to identify each task, the current bit to identify the current task, the valid bit to validate a bank, and the LRU bits to replace the victim bank. It should be noted that the task tag can be PID (Process ID) or the PPN (Physical Page Number) of the executing instruction when the context switch occurs. The PID is selected as task tag on systems that the PID would be available. Otherwise, the PPN of the executing instruction when the context switching occurs from the PPN field (or last translation) is used. Considering more generic cases, the PPN is selected; however, the PID can be more easily implemented. The discussion will be ignored in this paper. However, we still have to point out that we treat ITLB and DTLB as a couple, and they share the same bank tag. That means they stores translations for the same task in the same related bank. Except previous discussed parts, the remainder parts are designed for the entry prefetching mechanism. The prefetched logic initiates only when the TLB misses occurs. When the lookup misses in the current TLB bank but hits in the prefetch buffer, the address translation is generated from that hit entry and it will be inserted into the current TLB bank that is the same as traditional TLB entry replacement. Then, the prefetch logic tries to prefetch other entries into the prefetch buffer. If the lookup are missed in both

current TLB bank and the prefetch buffer, the traditional address translation mechanism is initiated to generate the correct address translation and then the prefetch logic prefetches new entries into the prefetch buffer according to current address. The 'Prefetch Logic' can be SP or DP described in[8].

Modifications needed for OS

In order to implement the new mechanism, the OS is needed to do a little modification. Except the page size issue, the OS is required to send 'the clear TLB signal' to the processor only when page swapping with disks occurs or page frames release. Fortunately, it's very easy to realize. Most modern processors provide some ways to flush TLB entries, such as STA instruction with alternative addresses on Sparc architecture [12].

Mechanisms of the design

The proposed TLB structure is divided into 32 banks and once the virtual address is generated from the CPU, the virtual page number (VPN, from the most significant bit to the previous bit of the offset, for example [31:15] in 32-bit addressing environment or [35:15] in 36-bit addressing environment) is sent to the 32 banks and the prefetch buffer in parallel. Each bank and the prefetch buffer work as the conventional TLB, and the PPN of the hit entries of each bank and prefetch buffer are sent to a multiplexer. In addition, the select signals are obtained from '*AND*' of the current bit of group tags and hit signal of each TLB bank, and also the hit signal from the prefetch buffer, to select the correct translation. If it's a hit in current TLB bank, the current TLB bank works as conventional TLB. The physical address can be simply generated by combining the output PPN and the offset from the virtual address. If it's a miss in current TLB bank but a hit in prefetch buffer, the operations are the same as what mentioned in the previous section. However, except the simplest situation, all other conditions should be carefully handled by the MMU (Memory Management Unit). The Following describes them in details.

1) ***No current bit set in all banks:*** The situation could be happened only when the first instruction fetching after a context switching for ITLB, the system initialization, or swapping pages with disks occurs. In this situation, no valid physical address can be provided via TLB translation. The address should be generated in conventional way by the OS and MMU. After the physical address (or PID if it is available) is generated, it is compared with the task field of bank tags. If any of it is hit with a valid bank tag, the current bit of that bank tag is set. Otherwise, the MMU should try to select a victim bank with invalid bit and LRU bits from the bank tag and flush all its 32 entries (both related ITLB and DTLB). Then the current bit of this bank should be set and the LRU bits of all bank tags should be updated. Then the correct translation is stored into the current ITLB bank entry, and the task tag of the current bank tag should be set. Moreover, it is the generated PPN (or PID in the environment if it is available) that is stored into the task tag field of the current bank tag. Finally, the prefetch logic initiates the prefetching mechanism that is the same as what mentioned in previous section.

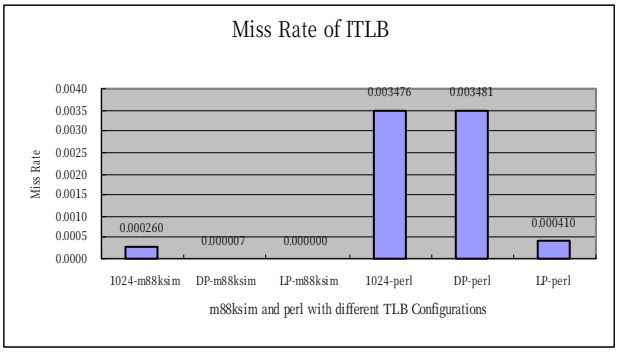
2) ***One current bit found but no valid translation in both current bank and prefetch buffer:*** If one current bit is found but no valid translation can be generated, that means the TLB (ITLB or DTLB) reference of the current task is available before but the missed page has not referenced yet. The operation of the current TLB bank just simply acts as a conventional TLB, and no bank tag modification is needed. Then the prefetch mechanism is worked as what mentioned in previous section.

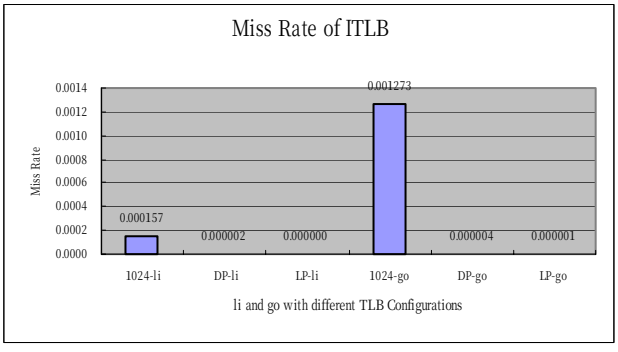
3) ***Context switching:*** Once the context switching occurs, the MMU just needs to clear the current bit of the bank tags and flush the prefetch buffer. No more other actions are needed.

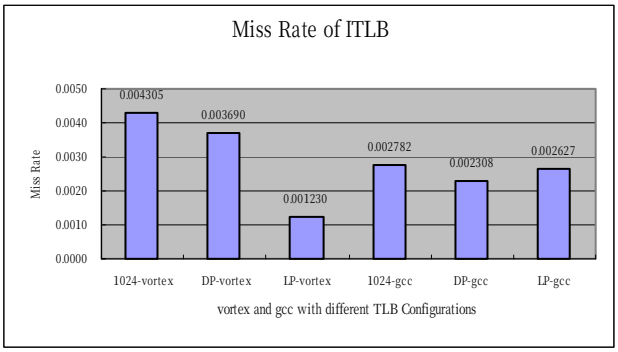
4) ***Page swapping with disk occurring or page frame releasing:*** If the page swapping with disks or page frame releasing occurs, the modified OS that we already discussed sends the 'clear TLB signal' to the MMU. Hence, the MMU can clear the valid bit of all bank tags and flush the prefetch buffer.

SIMULATION RESULTS

All of the simulations were done with the modified SimpleScalar Version 3.0d tool suite. We simulated all the SPEC95 benchmark to present the expected performance. We assumed that the context switching would happen after executing one million instructions. We compared the miss rates of 1024-entry TLB after context switching with the proposed TLB structure of 32 entries each bank with SP and DP prefetching mechanism after correctly keeping entries. We assume the SP can prefetch entries with VPN of +9 and -8. That means total 17 entries are prefetched. Moreover, we also assumed the DP can prefetch total 16 entries with 64-row distance table and each row has 2 predicted distance slots. Though we assume the DP with only 16-entry prefetch buffer, the costs of DP is still higher than SP. That's because the extra distance table are required in DP methodology. Figure 2 gives the simulation results of all SPEC95 benchmark.







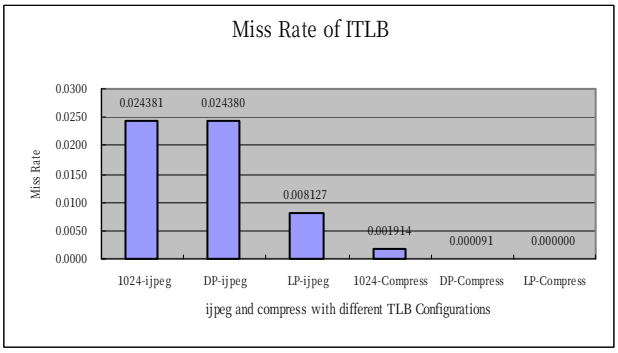
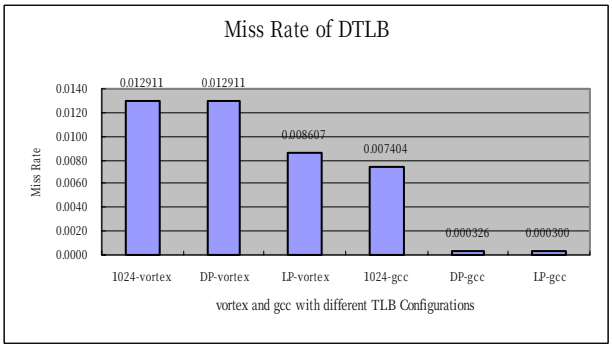
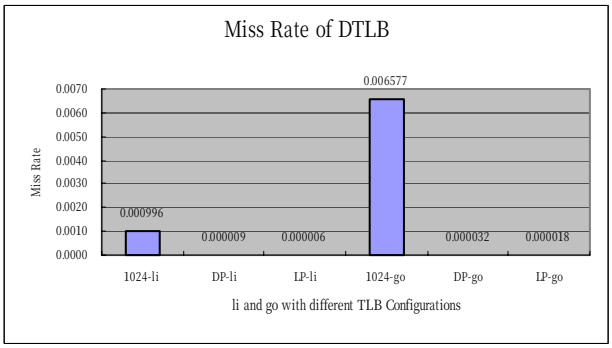


Figure 2(a). ITLB Miss Rates for all SPEC95 benchmark





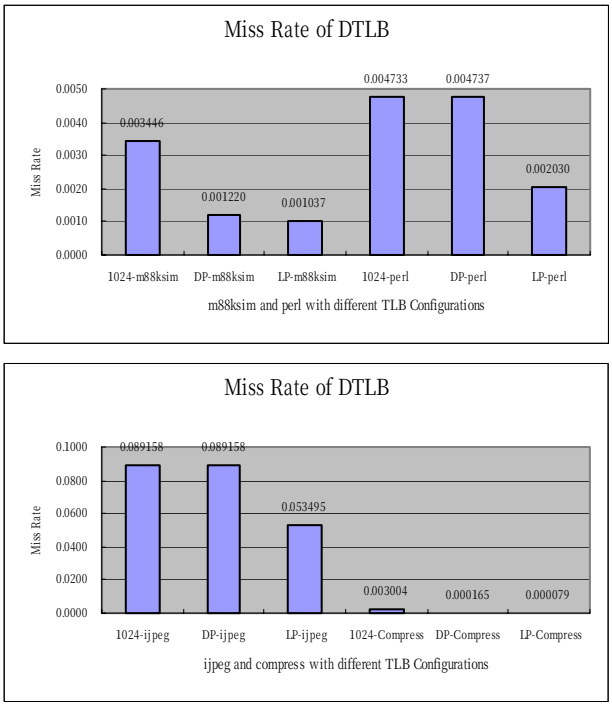


Figure 2(b). DTLB Miss Rates for all SPEC95 benchmark

Figure 2(a) and Figure 2(b) show the simulation results for ITLB and DTLB with 1024-entry conventional TLB and new TLB structures with DP and SP prefetching mechanism respectively. Observing the simulation results, we can find that our design can deliver better performance than conventional TLB structure if correct TLB entries can be kept. Furthermore, we can also find that the SP can provide better performance than DP prefetching mechanism under multiprogramming environment. That's because after the context switching occurring the DP prefetching mechanism needs the learning time to fill in the distance table. According to the simulation results, we strongly suggest to use the simplest SP prefetching mechanism in our design.

CONCLUSIONS AND FUTURE WORK

The TLB misses cause serious performance degradation on modern processors. In addition, the context switching under the multiprogramming OS may cause this problem even more seriously. However, few studies focus on the context switching issue. In this paper, we presented a new novel TLB mechanism for 32-KB page size environment to reduce the miss rate in context switching. We also discuss how OS should be modified to support this mechanism. Furthermore, we also discussed how to implement TLB entry prefetching mechanism in this structure. Finally, according to the simulation results, we suggested just simply to use the sequential prefetching (SP) mechanism in this design. Except the proposed mechanism, we have already begun to find solution to integrate the proposed structure to support multi-page size with bank promotion methodology. We believe that still lots of work should be done in this field.

REFERENCES

[1] David Channon and David Koch, "Performance Analysis of Re-configurable Partitioned TLBs," in *Proceedings of the 30th Hawaii Int'l Conf. on System Sciences*, Vol. 5, pp.168-177, 1995.

[2] Zhen Fang, Lixin Zhang, John B. Carter, Wilson C. Hsieh, and Sally A. Mckee, "Reevaluating Online Superpage Promotion with Hardware Support," in *Proceedings of the 7th Int'l Symp. on High-Performance Computer Architecture*, pp.63-72,2001.

[3] Michael J. Flynn, Boston: Computer Architecture – Pipelined and Parallel Processor Design. Jones and Bartlett Publishers, ch. 5.16, pp.323-325.

[4] Intel Corp., *IA-32 Intel® Architecture – Software Developer's Manual Vol. 3 – System Programming Guide*, 2004.

[5] Intel Corp., *Intel® Itanium® 2 Processor Reference Manual – For Software Development and Optimization*, May 2004.

[6] Intel Corp., *Pentium® Pro Family Developer's Manual Vol. 3 – Operating System Writer's Guide*, Dec. 1995.

[7] B.L. Jacob et al., "Virtual Memory: Issues of Implementation," *IEEE Computer*, Vol. 31, NO. 6, pp.33-43, June 1998.

[8] Gokul B. Kandiraju and Anand Sivasubramaniam, "Going Distance for TLB Prefetching: An Application-driven Study," in *Proceedings of the 29th Annual Int'l Symp. on Computer Architecture*, 2002.

[9] Jung-Hoon Lee, Jang-Soo Lee, She-Woong Jeong, and Shin-Dug Kim, "A Banked-Promotion TLB For High Performance and Low Power," in *Proceedings of the 2001 Int'l Conf. on Computer Design*, pp.118-123, 2001.

[10] Ashley Saulsbury, Fredrik Dahlgren, Per Stenström, "Recency-Based TLB Preloading," in Proceedings of the 27th Int'l Symp. on Computer Architecture, pp.117-127, 2000.

[11] SimpleScalar LLC, http://www.simplescalar.com/

[12] SPARC International Inc. The SPARC Architecture Manual Version 8, pp.241-260, 1992.

[13] Mark Swanson, Leigh Stoller, and John Carter, "Increasing TLB Reach Using Superpages Backed by Shadow Memory," in Proceedings of the 25th Annual Int'l Symp. On Computer Architecture, pp.204-213, 1998.