Short Paper

Reducing the TLB Context Switching Miss Ratio With Banked and Prefetching Mechanism

CHANG-JIU CHEN AND WEI-MIN CHENG
Department of Computer Science
National Chiao Tung University
Hsinchu, 300 Taiwan

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. Lots of different methodologies are proposed to reduce TLB misses. Most designs just simply try to increase the total size of their TLBs to reduce the capacity misses or just simply use the fully associativity to reduce the conflict misses. Furthermore, some designs even try to incorporate the operating system (OS) and TLBs with very complex methods. Only some studies consider influence of performance on the context switching issue. Most traditional designs just simply added some types of address space identifier within the TLB tags. Nevertheless, the worse case of all is the *x*86 architecture which flushes all its TLB entries on context switching. This paper proposes a banked TLB structure with prefetching mechanism to reduce the miss rate in context switching for 32K page size. All simulations were done with modified SimpleScalar 3.0d tool suite and SPEC95 benchmarks. The results show that the proposed mechanism can provide acceptable performance improvement than the worse case *x*86 style design. The miss rate may even be only 1/10 or less. Thus, the proposed architecture may be suitable to be implemented inside processors to reduce the context switching misses. Furthermore, we'll try to implement it inside our new asynchronous processor.

Keywords: TLB, operating system, context switching, virtual memory, address translation

1. 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. To provide virtual memory, the OS is responsible to provide mechanisms to map virtual address to physical address. However, all these virtual address to physical address translations are stored in main memory. To reduce the cost of address translations, the translation lookaside buffers (TLBs) are widely implemented inside the processor [1-5]. In fact, the virtual memory mechanism varies with different processor architecture and OS implementation. The page table organization dominates the page table traversal time that occupies most TLB miss handling time. Though some new architectures use some ad-

Received January 23, 2007; revised December 26, 2007 & March 20 & May 23, 2008; accepted May 29, 2008.
Communicated by Liang-Gee Chen.

vanced page table organizations to reduce the page table traversal time such as inverted page table structure, the forward-mapped hierarchical page table structure are still widely used, such as the latest AMD64 architecture [6]. It costs several main memory accesses to fetch the correct Page Table Entry (PTE) if any miss occurs. That impacts the overall system performance seriously. Thus it's important to reduce the TLB miss rates for systems with such page table structure.

In addition, frequently happened context switching may cause some extra TLB misses. Thus most processors except *x*86 implement some kinds of address space identifier (ASID) to distinguish each address space. However, the *x*86 simply flushes all its TLB entries when the context switch occurs [7]. In this paper, we treat it as the worse case performance. Though lots of different research about TLB has been done, only some notice the influence of context switching. 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 behavior first. In this paper, we tried to provide an alternative to address the context switching issue for TLB. To support the proposed mechanism, the OS should be modified a little. The mechanism can be easily implemented in future high performance systems.

All the simulations will be done by the modified SimpleScalar Version 3.0d tool suite [8] provided by the SimpleScalar LLC with SPEC95. Except the performance of traditional 1024-entry fully-associative TLB with *x*86-style assumption, we also compare the performance of 1024-entry fully-associative TLB with ASID and two different prefetching mechanisms incorporate with our proposed design. The results show that our banked design can work very well with sequential prefetching (SP, also called linear prefetching).

The work is trying to reduce the extra TLB miss rate caused by context switch. Though most processors reduce the miss rate caused by context switch with ASID, this paper provides an alternative to address this issue. This paper also discusses why sequential prefetching is more suitable for the proposed design. Moreover, we'll try to realize this design on the asynchronous processor which we currently work for. That would not be too hard to realize the proposed mechanism on an asynchronous processor with some extra handshaking signals on bundled delay or dual-rail design.

2. RELATED WORK

In order to reduce the TLB miss rate, most processors increase the size (total entries) of TLBs with fully or set associative. For example, recent AMD Opteron^{TM} processor has both 512-entry L2 instruction TLB (ITLB) and L2 data TLB (DTLB) [9] and the IBM POWER4 processor has a common 1024 entry TLB for each processor core [10]. Furthermore, some processors even try to provide multi-level TLBs, such as 2-level ITLB/ DTLB design on recent AMD Opteron^{TM} processor [9]. 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 *x*86 Processors after the Pentium$^{®}$ Pro Processor [11]. 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 [12]. Four 4KB pages can be promoted to a 16KB superpage. To support such mechanism, an interesting promotion TLB is designed. The heuristic promotion algorithm can promote four consecutive entries from small-page TLB bank to large-page TLB bank. Thus, the four 4KB TLB entries can be reused. Furthermore, in order to reduce the power consumption and TLB reference latency, they even divided the TLB for 4KB page into two banks [13]. Fig. 1 shows the structures of their promotion TLB and banked-promotion TLB. 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 [14]. Channon *et al*. presents the re-configurable partitioned TLBs to improve the TLB performance [15].

Though lots of these new mechanisms are proposed, just only a few studies focused on the TLB entries prefetching/preloading. Saulsbury *et al*. introduces an interesting mechanism, called the Recency-based TLB Preloading (RP), to prefetch the TLB entry according to the 'Recency' of the referenced pages [16]. 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 PTE 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)' [17]. 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.

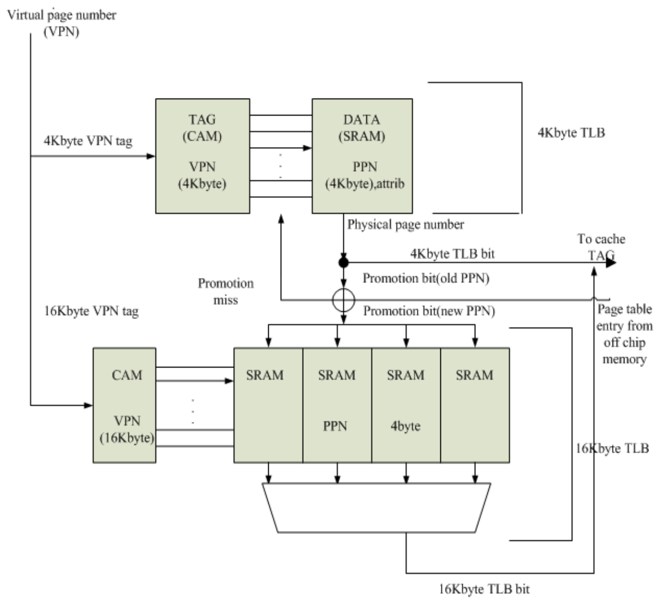

Fig. 1. (a) Promotion TLB structure and banked-promotion TLB structure.

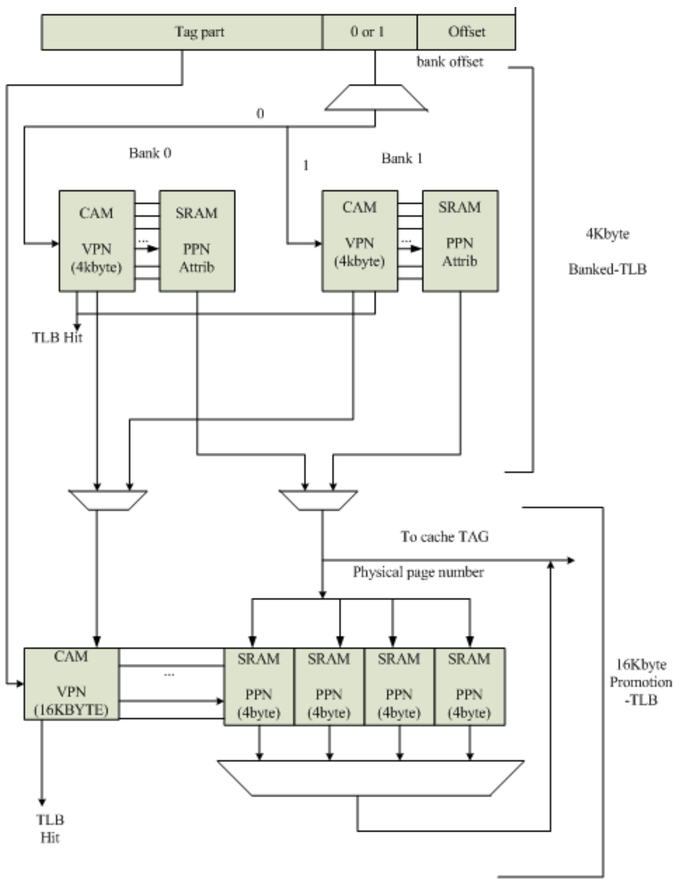

Fig. 1. (b) Promotion TLB structure and banked-promotion TLB structure.

3. RELATIONSHIPS BETWEEN THE MISS RATES AND TLB SIZES

It is widely known that the two most important issues for cache system performance are lower miss rate and the miss penalty. It's almost the same for the TLB performance. In fact, because the miss rate has the greatest impact on TLB performance, most studies focus on it. In this section, we consider the relationships among miss rates, page sizes and TLB sizes.

Let's consider the relationship between the miss rates and TLB sizes with 4KB page size. Fig. 2 shows the relationship between TLB sizes and miss rates of running *gcc*. The two results show that the miss rates would be lower if the TLB sizes can be increased. We can also find that in order to obtain better performance for 4KB page the size should be at least 64 entries. However, that's not always true for all applications. Let's observe the result of *ijpeg* showing in Fig. 3. It's very clear that a 16-entry TLB is enough. It's useless to increase the number of TLB entries. In fact, it's almost the same for some other benchmark programs, such as *vortex* and *li*. Thus the results vary from application to application.

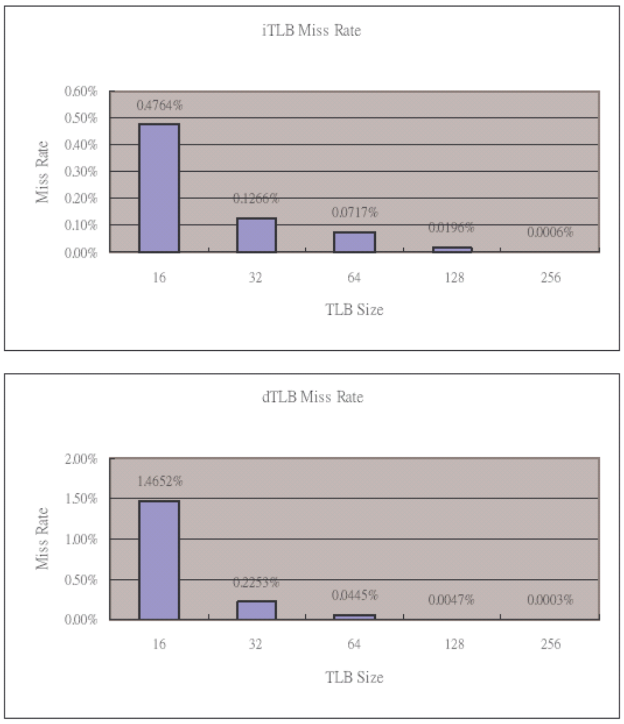

Fig. 2. ITLB/DTLB miss rate for *gcc* with 4KB page.

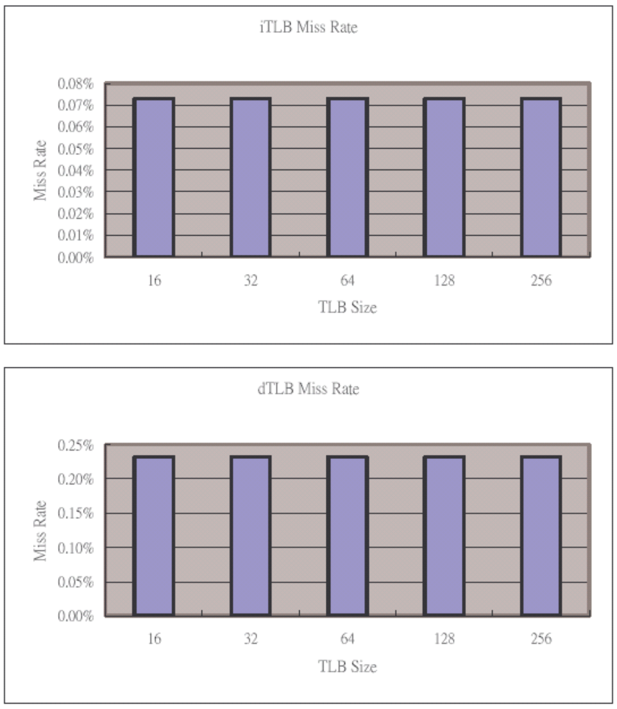

Fig. 3. ITLB/DTLB miss rate for *ijpeg* with 4KB page.

Another solution to improve the performance of TLB is to extend the page size into larger one. In fact, most modern processors provide multiple page sizes, such as 4KB, 2MB, and 4MB on all new Intel® *x*86 series processors [11]. The advantages of larger page size are not only obtaining better performance but saving the implementation cost with shorter tags of virtual page number (VPN) and translations (physical page number, PPN) needed to be stored. It is also a good method to reduce the cost on TLB implementation of processors with larger addressing space, such as processors with 64-bit addressing capability. Certainly, larger page size is suitable to be implemented for processor core of SoC or embedded systems. Fig. 4 shows the miss rate of *compress* for 4KB, 16KB, 32KB, 64KB, and 1MB page sizes with different TLB sizes. Observing the results, we can easily find that the performance of 1MB page size of TLB with only 8 entries can even outperform 4KB page size of TLB with 256 entries. In fact, with the larger page size the larger working set can be covered. In addition, we can also find that the performance of 32KB page size TLB with 32 entries is good enough for *compress*. With prefetching mechanism, the performance would be even better. However, according to the previous discussion, even with 4KB page size, the total TLB entries needed may still vary from application to application. Sometimes, even 16-entry TLB is good enough for 4KB page. But, for reliable reason, we selected the 32KB page size. That's why the page size we select is 32KB, and each TLB bank has 32 entries. In fact, the new proposed model can be implemented to support different page size and the TLB size of each bank is also configurable depending upon the system needs. It's an implementation tradeoff!

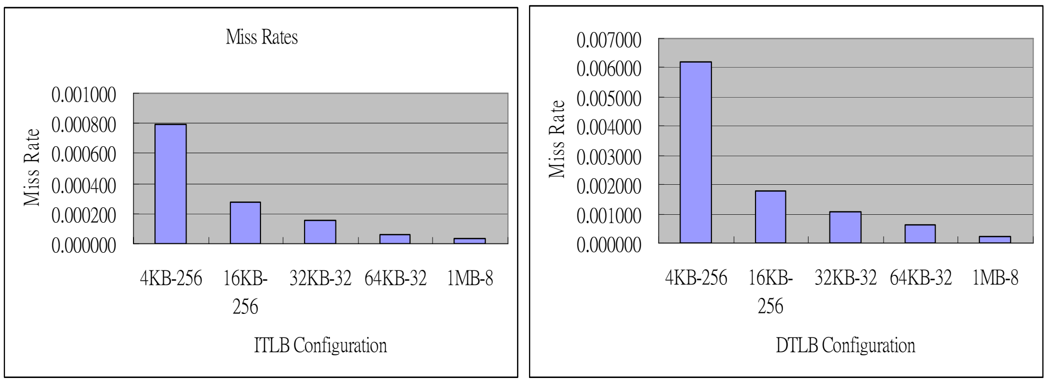

Fig. 4. ITLB/DTLB miss rate for *compress* with different page sizes and TLB sizes.

4. ARCHITECTURE OF THE PROPOSED TLB

This section describes in detail of the new 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.

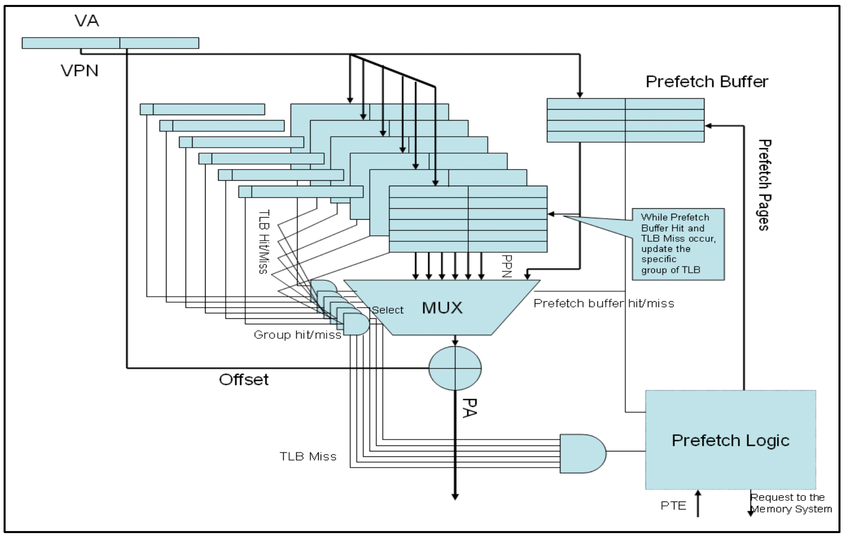

Fig. 5. The proposed TLB architecture.

4.1 Overview

Fig. 5 shows in detail the proposed 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. However, it can be easily changed to adequate for different page sizes with little configuration changes. Furthermore, we'll have other new study for TLB to provide superpages with page promotion mechanism.

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 & control 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 1024-entry common TLB for each processor core of IBM POWER4 processor [10]. 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 Fig. 5. 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 any address space identifier (ASID) which the processor itself provides or the PPN (Physical Page Number) of the executing instruction when the context switch occurs on processors without any ASID support (*x*86 based processors). On processors without ASID support, the PPN of the executing instruction when the context switching occurs from the PPN field (or last translation) is used. Considering the worse *x*86-style case, the PPN is selected; however, the implementation with ASID provided by the processor itself can be more easily. 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 prefetch & control logic initiates 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 & control 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 & control logic prefetches new entries into the prefetch buffer depending upon the current address. The 'Prefetch Logic' can be SP or DP described in [17].

4.2 OS Modification

In order to implement the 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. If the signal is received by the control logic, the control logic should flush all the TLB banks and the prefetch buffer for the worse case example (the only example is *x*86) or the corresponding TLB bank and the prefetch buffer for the general cases. Fortunately, it's not hard to realize. In fact, almost all modern processors, except *x*86 processors, provide some ways to flush TLB entries, such as STA instruction with alternative addresses on SPARC architecture [18].

4.3 Mechanism 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) 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 prefetch & control logic. 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 address space identifier (ASID) supported by the architecture 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. On the contrary, if it's a miss, the prefetch & control logic 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 ASID provided by the processor that is stored into the task tag field of the current bank tag. Finally, the prefetch logic & control 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 prefetch & control logic can clear the valid bit of all bank tags on system without architecture supported ASID (*x*86) or and flush the prefetch buffer.

5. SIMULATION RESULTS

All of the simulations were done with modified SimpleScalar Version 3.0d tool suite. The SPEC95 benchmark programs were simulated to estimate the performance. We assume that the context switching would happen after executing one million instructions, and we also assume that the compared 1024-entry TLB is the worse case *x*86-style example. In addition, we compared the miss rates of worse case style 1024-entry fully-associative TLB with the proposed TLB structure of 32 entries each bank with SP and DP prefetching mechanism after correctly keeping the entries and 1024-entry full-associative TLB with ASID of the same workload assumption with proposed TLB structure. We assume that the SP can prefetch entries with VPN of + 9 and – 8. That means total 17 entries are prefetched. Moreover, we also assume that 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 is required in the DP methodology. Figs. 6 (a) and (b) give the simulation results of SPEC95 benchmark.

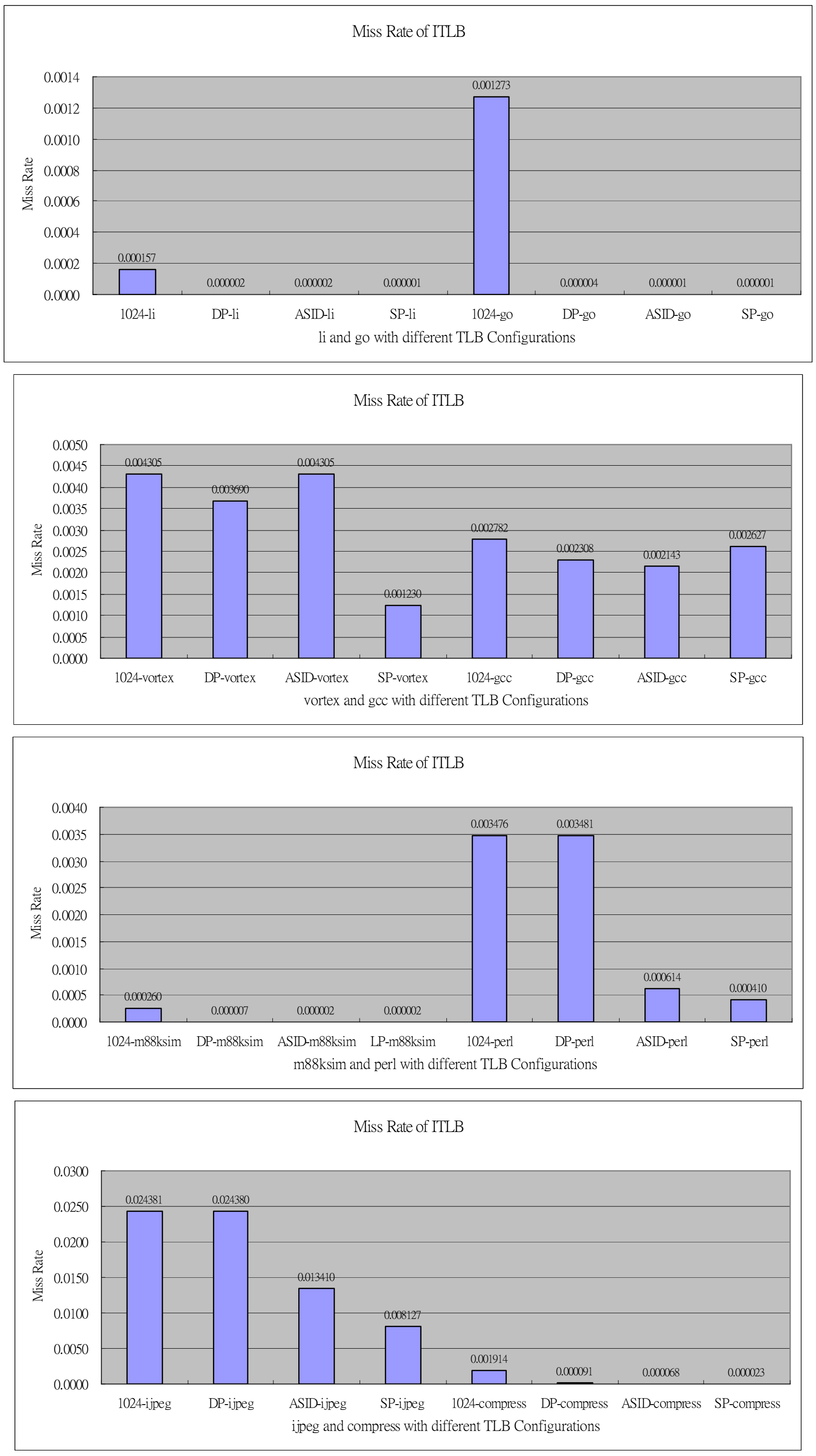

Fig. 6. (a) ITLB miss rates for SPEC95 benchmarks.

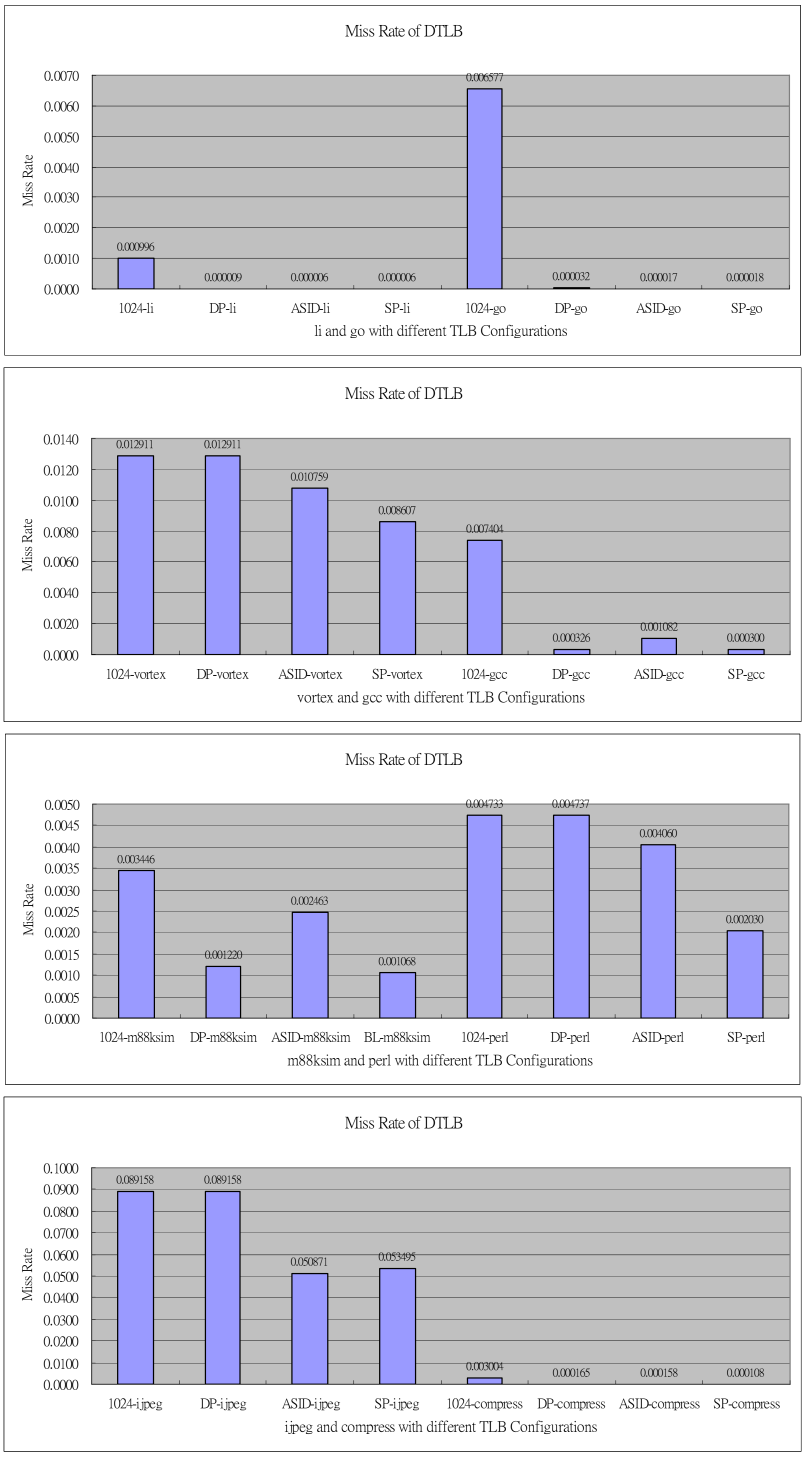

Fig. 6. (b) DTLB miss rates for SPEC95 benchmarks.

Figs. 6 (a) and (b) show the simulation results for ITLB and DTLB with 1024-entry conventional TLB, new TLB structures with DP and SP prefetching mechanism, and 1024-entry conventional TLB with ASID 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 proposed banked TLB with SP prefetching mechanism can deliver better performance than DP prefetching mechanism and conventional TLB with ASID under multiprogramming environment. Through observing the simulation results, we can also find that prefetching mechanism may be sometimes more important than just increasing more entries. For example, both of the DTLB performances of new TLB structures with SP and DP mechanism are better than conventional TLB with ASID for *gcc*. However, in most cases, the performance of new TLB structure with DP prefetching mechanism is still worse than conventional TLB with ASID. 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.

Even so, we still have to point out several important issues. The first, it's not really very fair to assume the conventional fully associative TLB works as the worse case *x*86-style. The only architecture which flushes all its TLB entries in context switching is *x*86. Most processors incorporate their own address space identifiers with TLB tags. These designs, including our methodology, incorporated tags with ASID may have almost the same performance. However, our structure can save some tag bits because of our banking method. Fig. 7 shows the TLB entries with ASID tag. It's very clear that it needs more tag bits than our design. We provide an alternative method to store the ASID. Second, it's not a very nice model to assume that context switch occurs after executing each one million instructions. In fact, it may differ from different environments. Most OS defines its own time slice with several milliseconds, and with different processors, the total instructions executed may have enormous differences. In addition, the real situation depends upon real OS running situation. In fact, we seriously consider developing a new generic simulator incorporated with Linux OS to more accurate modeling the real environment. Third, though the page size we assume in this paper is 32KB, it's not very hard to change it to other sizes with some configurations change. Finally, though only a few studies about TLB entry prefetching, it still possible to provide more heuristic prefetching mechanism for TLB entry prefetching. Furthermore, it may be also possible to incorporate other prefetching mechanism with our structure.

ASID	VPN	PPN
A	0x8000	0x100
B	0x8000	0x400
B	0x8200	0x500
A	0x8020	0x120
	……	
A	0x8100	0x200

Fig. 7. TLB entry with ASID.

6. CONCLUSIONS

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, only some studies focus on the context switching issue. In this paper, we presented an alternative 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 discuss 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 superpaging with bank promotion methodology. To obtain more accurate performance evaluation in real environment, the new simulation model and simulator are under developing. In addition, we'll try to realize this mechanism in our current new RISC asynchronous processor project. We believe that still lots of work should be done in this field.

REFERENCES

1. B. Jacob and T. Mudge, "Virtual memory: issues of implementation," *IEEE Computer*, Vol. 31, 1998, pp. 33-43.
2. R. Case and A. Padegs, *Architecture of the IBM System/370*, McGraw-Hill Book Company, New York, 1982.
3. B. Jacob and T. Mudge, "Virtual memory in contemporary microprocessors," *IEEE MICRO*, Vol. 18, 1998, pp. 60-75.
4. D. W. Clark and J. S. Emer, "Performance of the VAX-11/780 translation buffer: simulation and measurement," *ACM Transactions on Computer Systems*, Vol. 3, 1985, pp. 31-62.
5. M. J. Flynn, *Computer Architecture: Pipelined and Parallel Processor Design*, Jones and Bartlett Publishers, Boston, 1995.
6. Advanced Micro Devices, Inc., *AMD64 Technology – AMD64 Architecture Programmer's Manual*, *Volume 2: System Programming*, Publication No. 24593, 2006.
7. Intel Corp., *Intel®64 and IA-32 Architecture Software Developer's Manual*, *Vol. 3A: System Programming Guide*, *Part 1*, Order No. 253668, 2006.
8. T. Austin, *SimpleScalar LLC*, http://www.simplescalar.com/.
9. Advanced Micro Devices, Inc., *Software Optimization Guide for AMD Athlon™ 64 and AMD Opteron™ Processors*, Publication No. 25112, 2003.
10. J. M. Tendler, J. S. Dodson, J. S. Fields Jr., H. Le, and B. Sinharoy, "POWER4 system microarchitecture," *IBM Journal of Research and Development*, Vol. 46, 2002, pp. 5-25.
11. Intel Corp., *Pentium® Pro Family Developer's Manual*, *Vol. 3: Operating System Writer's Guide*, Intel Corp., 1995.
12. J. H. Lee, J. S. Lee, and S. D. Kim, "A dynamic TLB management structure to support different page sizes," in *Proceedings of the 2nd IEEE Asia-Pacific Conference on ASICs*, 2000, pp. 299-302.

13. J. H. Lee, J. S. Lee, S. W. Jeong, and S. D. Kim, "A banked-promotion TLB for high performance and low power," in *Proceedings of the International Conference on Computer Design*, 2001, pp. 118-123.
14. M. Swanson, L. Stoller, and J. Carter, "Increasing TLB reach using superpages backed by shadow memory," in *Proceedings of the 25th Annual International Symposium on Computer Architecture*, 1998, pp. 204-213.
15. D. Channon and D. Koch, "Performance analysis of re-configurable partitioned TLBs," in *Proceedings of the 30th Hawaii International Conference on System Sciences*, Vol. 5, 1995, pp. 168-177.
16. A. Saulsbury, F. Dahlgren, and P. Stenstrom, "Recency-based TLB preloading," in *Proceedings of the 27th International Symposium on Computer Architecture*, 2000, pp. 117-127.
17. G. B. Kandiraju and A. Sivasubramaniam, "Going the distance for TLB prefetching: an application-driven study," in *Proceedings of the 29th Annual International Symposium on Computer Architecture*, 2002, pp. 195-206.
18. SPARC International Inc., *The SPARC Architecture Manual*, *Version 8*, Prentice-Hall, Inc., 1992.

Chang-Jiu Chen (陳昌居) received his Ph.D. degree in Computer Science from University of Oklahoma, in 1993, and his B.S. and M.S. degrees in Computer Engineering from National Chiao Tung University, Hsinchu, Taiwan, in 1980 and 1986, respectively. He has been an Associate Processor of Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan, since 1994. His research interests include asynchronous circuit design, computer architecture, digital system, microprocessor, and parallel processing.

Wei-Min Cheng (鄭緯民) received the B.S. and M.S. degrees in Computer Science and Engineering from Tatung Institute of Technology, Taipei, Taiwan, in 1995 and 1997, respectively. He is current a Ph.D. student in Computer Science in National Chiao Tung University, Hsinchu, Taiwan. His research interests include computer architecture, asynchronous system design, microprocessor and SoC.