Int. Computer Symposium, Dec. 15-17, 2004, Taipei, Taiwan.

A Novel TLB Architecture to Reduce the Miss Rate in Context Switching

Chang-Jiu Chen,Wei-Min Cheng,Chi-Wen Chang, and Wen-Chiuan Liao*
Department of Computer Science and Information Engineering
National Chiao Tung University, Taiwan,ROC
{cjchen,wmcheng,chiwen}@csie.nctu.edu.tw
**CCL,ITRI, Taiwan,ROC*
liao@itri.org.tw

Abstract*-It is widely known that the Translation Lookaside Buffer (TLB) plays an important role in the address translation mechanism from virtual addresses to physical addresses. If any misses occur, the performance of the processor will seriously degrade. In order to reduce such misses, some methodologies are proposed. Some designs try to improve the associativity or sizes to reduce the conflict or capacity misses, while others try to use superpages to cover more memory spaces. Furthermore, some papers even propose methods to dynamically merging several smaller pages into superpages during the processor execution. These methodologies, especially superpages, can effectively reduce lots of misses for most applications. However, to support the multiprogramming characteristic in all modern OS, the context switching mechanism is needed and the context switching will cause the flush operations for all TLB entries. It will impact on the performance very seriously, especially on today's high performance processors. However, it's hard to find an easy implement solution to reduce the misses in context switching. This paper shows what would happen if the page sizes are increased from 4KB to 1MB to explain why larger page size are selected. The paper also presents a novel and easy implemented TLB architecture to reduce the misses in context switching. All simulations were done with modified SimpleScalar 3.0d tool suite and SPEC95 benchmarks. The results show that our methodology can be very useful for multiprogramming environment under specific conditions.*

Keywords: TLB, Multiprogramming, Context Switch, SimpleScalar.

1. Introduction

To support large memory requirements for modern applications, all new advanced general-purpose processors will support the virtual memory. In order to support virtual memory, the address translation mechanism is needed. It is well known that all the address translations are stored in main memory and maintained by the operating system; to reduce the cost of address translation, the translation lookaside buffers (TLBs) [4] are implemented inside the processor. If there's any TLB miss occurring, at least two or three memory accesses are needed to fetch the translation from main memory by the memory management unit (MMU). With the VLSI technology improving rapidly, the new microprocessors become much faster than ever before and it causes the gap between memories and the processor core is larger and larger. We can easily find that the TLB is in the critical path of memory accesses. It's an important issue to reduce the miss rate of TLB [8].

To reduce the miss rate, most processors try to improve the sizes (total entries) of TLBs with fully or set associativity, such as 512 entries 4K page TLB on Intel® Pentium !!!® Processor [5]. In addition, some processors even implement multi-level TLBs, such as the highest-end Intel® Itanium2® Processor with 2-level ITLB and DTLB [6]. However, some processors begin to provide superpage to increase the TLB span. Assuming a processor with only 64-entry TLB and 4 KB pages are used (such as MIPS R3000[3]), only 256KB memory space can be mapped (64×4KB) but total 4GB in 32-bit addressing or even more main memory space for future processors. For example, the new Intel® Processors from Pentium Pro® begin to provide larger page with sizes of 2MB and 4MB [7]. In order to provide different page sizes, several TLBs are needed. Some studies focus on dynamically supporting pages with different sizes. Lee *et al.* proposes a novel banked-promotion TLB structure to support two page sizes dynamically. Four 4KB pages can be promoted to a 16KB page dynamically and to support such mechanism a interesting two-bank TLB are proposed. The heuristic promotion algorithm can promote four consecutive entries from small page TLB bank to large TLB bank [9,2]. Swanson et al. presents a novel memory controller which can aggressively create superpages even from non-contiguous and unaligned regions of physical memory [12]. Channon *et al.* introduces re-configurable partitioned TLBs to improve the TLB performance [1].

Though lots of research is done, few consider the context switching problem under multiprogramming environment. However, all context switchings will cause the flush operations for whole of the TLB entries. It affects the performance very seriously. In this paper, we propose a novel and easy implement TLB structure to reduce the miss rate caused by the context switching under multiprogramming environment. The structure is so easy to implement that it's also suitable to be implemented inside the processor core of SoC.

The simulation will be done by the modified SimpleScalar version 3.0d tool suite with SPEC 95 benchmark. We modified the original SimpleScalar version 3.0d tool suite to accommodate our requirements.

The rest of the paper is organized as follows. In Section 2 we discuss the relationship between the miss rates, page size and TLB sizes. The new novel TLB architecture to reduce miss rate in context switching is presented in Section 3. The expected performance is demonstrated in Section 4. Finally, in Section 5 we summarize the conclusion and describe the possible future work.

2. Relationships between the Miss Rates, Page Sizes and TLB Sizes

It is well-known that the most important two issues for cache system performance are to reduce the miss rate and the miss penalty. It is almost the same for the TLB performance. However, the most important issue is the miss rate. We focused on the miss rate in our research. In order to select the suitable page size, we did some study on the relationships between the miss rates, page sizes and TLB sizes.

First, we consider the relationship between the miss rate and TLB sizes with traditional 4KB page sizes. Figure 1 below illustrates the relationship between TLB sizes and miss rate of running *gcc* of *SPEC95*. The two results show that the miss rates would be lower with the TLB sizes increasing. The performance of both ITLB and DTLB are the best with sizes over 256 entries.

However, that's not always true for most applications. Let's observe the result of *ijpeg* of SPEC95 showing in figure 2. It's very clear that only 16 entries are enough. It's useless to increase the number of TLB entries. The result would be roughly the same on some other SPEC95 benchmarks such as *vortex* and *li*.

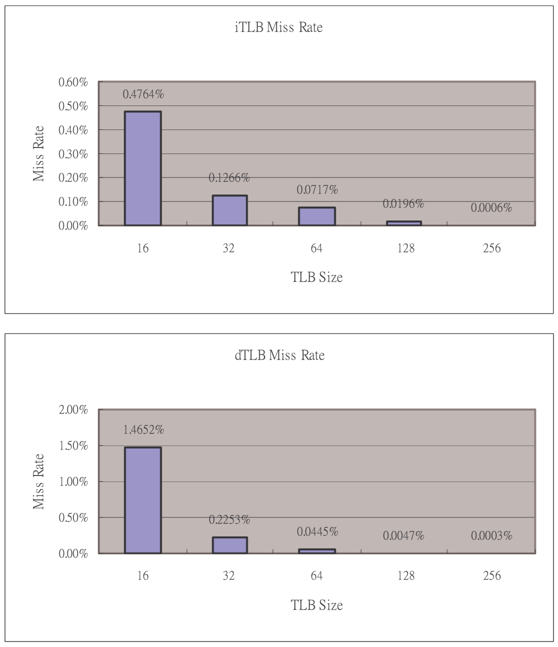


Figure 1 ITLB/DTLB miss rate for *gcc* of SPEC95

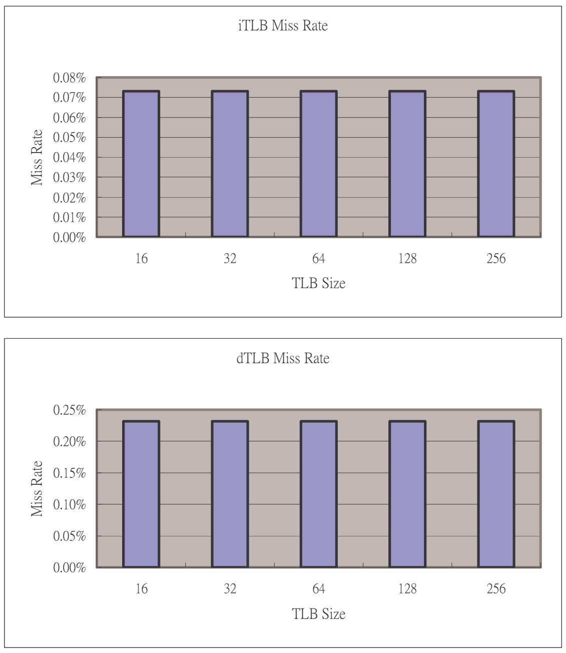


Figure 2 ITLB/DTLB miss rate for *ijpeg* of SPEC95

Another solution to improve the performance of TLB is to extend the page size into larger one. It's easy to find that some modern processors begin to provide multiple page sizes such as 4KB, 2MB and 4MB sizes on all new Intel® advanced x86 processors after the Pentium Pro® processor[7]. The advantages of larger page sizes are not only obtaining better performance but saving the implementation cost with less tags (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 space. Certainly, it's suitable to implement on the processor core of SoC or embedded systems.

What would happen if we extend the page size to 1MB. Figure 3 shows the miss rate for *gcc* of SPEC95 of 4KB, 16KB, 64KB, 256KB and 1MB page sizes with different TLB sizes. Observing the result, we can find that the ITLB performance of 1MB page with 8 entries can easily outperform that of 4KB page or even 256KB page with 256 entries. In addition, the DTLB performance of 1MB page with 8 entries can greatly outperform that of 4KB page with 256 entries. Furthermore, it is roughly the same with that of 256KB page with 256 entries. In addition, the results are almost the same on all the SPEC95 benchmarks. Thus, we selected the 1MB page size in our design.

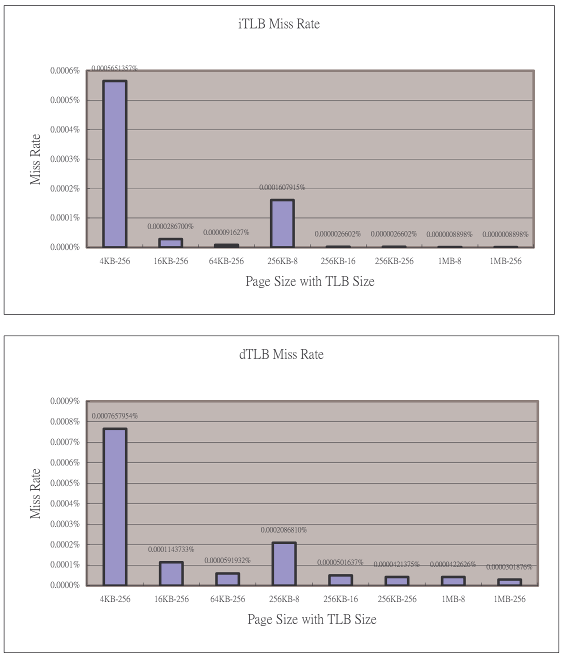


Figure 3 ITLB/DTLB miss rate for *gcc* of SPEC95 of 4KB, 16KB, 64KB, 256KB and 1MB page sizes with different TLB sizes

3. Structure of the Novel TLB

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

3.1. Overview

Figure 4 shows in detail our proposed novel TLB structure to reduce the miss rate in context switches. To reduce the miss rate, most designs just try to increase the TLB size to reduce the capacity misses; however, we have showed in previous section that it is more helpful if the page can be enlarged. Furthermore, with larger page size, the size of tags and translations needed to be stored can be much smaller. Thus, we used 1MB page in our design.

The proposed structure consists of the following parts—the TLB banks with group tags, and a multiplexer to select a specific TLB bank. Each TLB bank contains eight entries, and the tag can be implemented with CAM (content addressable memory) which is the same as that being implemented on conventional TLB. Furthermore, each TLB bank is implemented with fully associative with LRU replacement policy. There are total 32 or more TLB banks. Though there are 32 banks, compared with 256-entry conventional TLB the total cost is not increased very much. In fact, there are also total 256 (32*8) entries in our proposed structure. In addition, because of larger page size, the cost of each entry is decreased. Thus the increased cost can be ignored. Except the 32 TLB banks, there are also 32 extra registers to store the bank tag as shown in Figure 4. 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[1] to replace the victim bank. We have to point out 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 will be sent into the processor; otherwise, the PPN of the executing instruction when the context switch occurs from the PPN field (or last translation) is selected. Considering the general cases, the PPN is selected in the latter discussions; however, the PID can be more easily selected and implemented under the previous situation. 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 tags. That means they stores translations for the same task in the same related bank.

3.2. Implementation of the novel TLB

In order to realize the new proposed mechanism, the OS is needed to do a little modification. Except larger page size[2], the OS needs to send 'the clear TLB signal' to the processor only when swapping pages with disks occurs or page frames release.

[1] Pseudo-LRU can be implemented.
[2] We are currently working on developing new structure to support 4KB page size and superpages with TLB prefetching mechanism based on the proposed novel TLB structure.

Fortunately, it's very easy to realize. Most modern processors provide some ways to flush TLB entries, such as using *STA* instruction with alternative addresses on Sparc processors [11].

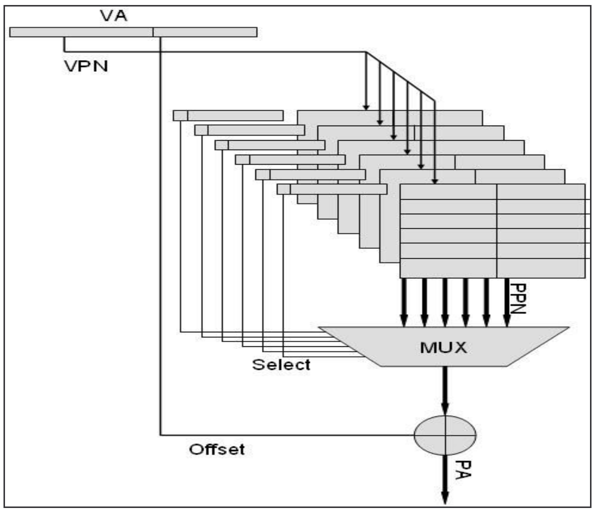


Figure 4 A Novel Low Context Switching Miss Rate TLB Architecture

3.2.1. Mechanisms of the novel TLB

The proposed TLB structure is divided into 32 banks. 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 [31:20] in 32-bit addressing space environment or [35:20] in 36-bit addressing space environment) is sent to the 32 banks in parallel. Each bank works the same as conventional TLB. The PPNs of the hit entries of each bank are sent to a multiplexer. In addition, the select signals are obtained from the current bit of all group tags in order to select the right bank. If it's a hit, the current TLB bank works as conventional TLB and the physical address can simply be generated by adding the output PPN and the offset from virtual address. However, except the simplest situation, all other situations should be carefully handled by the MMU (memory management unit). The following describes these in details.

1) ***No current bit set in all bank tag:*** The situation could be happened only when the first instruction fetching after a context switching for ITLB, the system initialization, or the swapping pages with disks occurring. Under this situation, no valid physical address can be provided via TLB translation. The address should be generated in general way by the MMU and OS. After the physical address (or PID if it is available) is generated, it is compared with the task tag field of bank tags. If any of it is hit in 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 eight 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. Finally, the correct translation is stored into the current ITLB bank entry and the task tag of current bank tag should be set. Moreover, it is the generated PPN (or PID under the situation which PID is available) that is stored into the task tag field of current bank tag.

2) ***One current bit found but no valid translation:*** If one current bit is found but no valid translation can be generated, it means that the TLB (DTLB or ITLB) reference of the current task is available before but the missed page has not referenced yet. The action of the current TLB bank just simply acts as a conventional TLB, and no bank tag modification is needed.

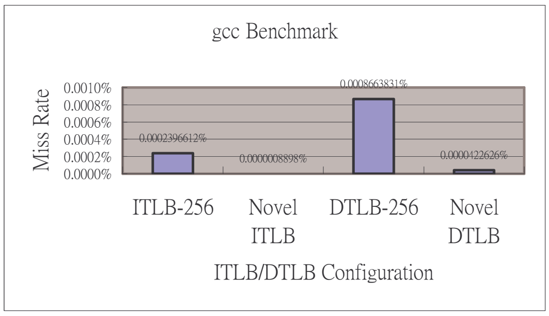
3) ***Context switching:*** Once the context switching happens, the MMU just needs to clear the current bit of the bank tag and no more other actions.

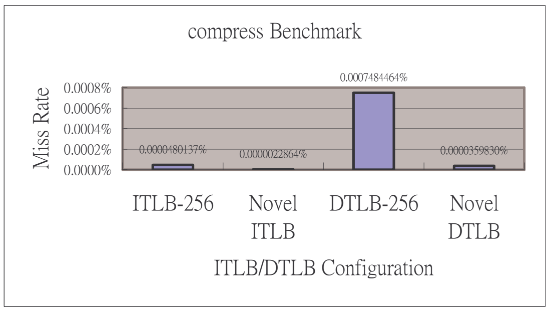
4) ***Page swapping with disk occurring or page frame releasing:*** If the page swapping with disk occurs or page frames release, the modified OS sends the 'clear TLB signal' to the MMU. Hence, the MMU can clear the valid bit of all bank tags.

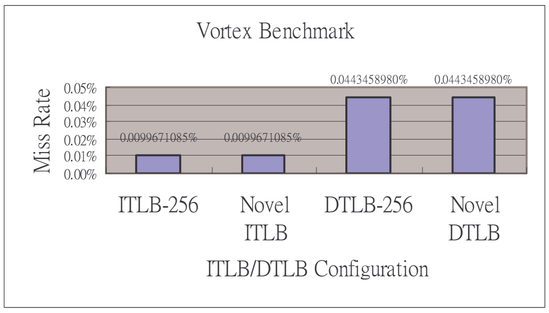
4. Simulation Results

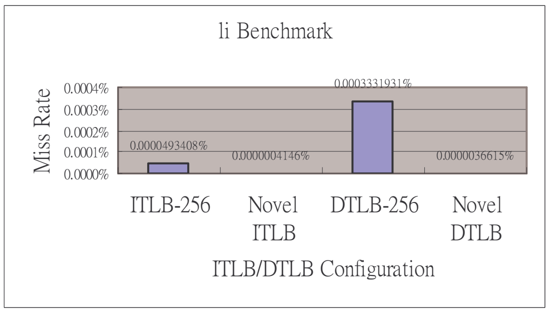
All of our simulations were done with SimpleScalar 3.0d tool suite. We simulated all the SPEC95 benchmark to demonstrate the expected performance. We assumed that the context switching would happen after executing one million instructions. We compared the miss rates of conventional 256-entry TLB with flushing all entries after context switching and our novel TLB structure with 8-entries each bank after correctly keeping entries. Figure 5 shows the simulation results of all SPEC95 benchmarks.

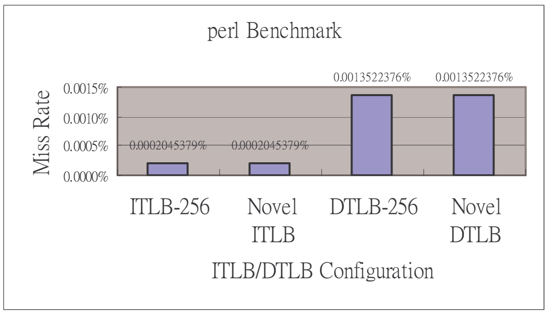
As can be seen in Figure 5, we can find that our design can deliver at least the same performance or better performance than the conventional structure. However, these results show two major differences among all of them. Some results show that the new novel design can greatly outperform conventional structure if the keeping entries are correct, such as *gcc*, *compress*, *li*, and *go*. That's because these applications need more computation time to finish and more context switching occurs during their execution. Others show the same result between conventional design and ours, such as *Vortex*, *ijpeg*, and *perl*. These applications need only shorter computation time, and the computation can be finished before the first context switching. Thus, both of the two structures can give the same performance. However, both of the two structures have the same total entries. That means that with the same total TLB size our new structure can perform better than conventional TLB structure. Our design has better TLB-entry utilization.

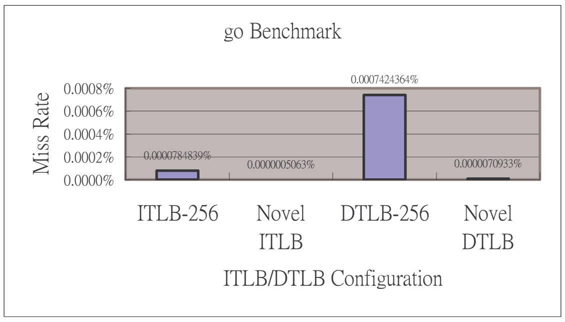












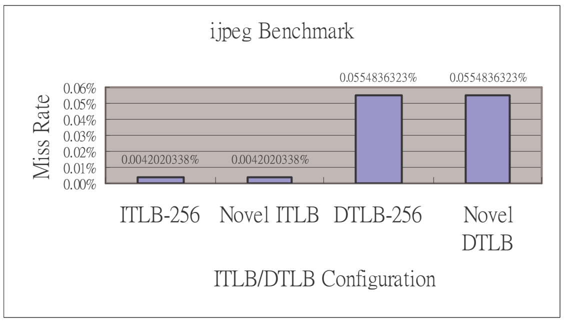


Figure 5 Miss rates of all SPEC95 benchmarks

5. Conclusions and Future Work

The TLB misses have great impact on the overall performance of modern processors. However, there is only a little research on it for the past decade. Furthermore, few studies focus on the context switching issue which is considered the most important issue of all. In this paper, we presented a new novel TLB mechanism to reduce the miss rate in context switching.

We showed that with 1MB page size only eight entries TLB can easily cover near the whole working set of general applications. With 1MB page size, it's useless to increase the TLB size from 8 entries to 256 or even more entries. According to the result, we tried to divide the 256 entries into 32 banks in order to store translations of different tasks. We proposed a mechanism to implement the new TLB structure and how to modify OS to support it. With this methodology, the utilization of the TLB entries is improved. Furthermore, we also showed that the miss rate in context switches can be decreased if the TLB entries are correctly kept.

However, though lots of modern processors have already provided multiple page sizes especially large page sizes, most modern OS only support 4KB page for their paging mechanism. Fortunately, some research begins to focus on supporting larger page size, or multiple page sizes. For example, Naohiko Shimizu and Ken Takatori proposed a Linux superpage kernel for Alpha, Sparc64 and IA32 [10].

Though we proposed a new novel TLB structure in this paper, it is only for 1MB or larger page size.

However, it's also important to provide such solution for conventional 4KB page environment. We have already begun to find a solution to integrate the proposed structure and TLB entry prefetching mechanism for 4KB page environment. Furthermore, we also begin trying to find solutions to support multiple page sizes with lowest cost and even provide hardware superpage mechanism with 4KB page based. We believe that still lots of works 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, pp168-177,1995

[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*, pp63-72, 2001

[3] Erin Farquhar and Philip Bunce, *The MIPS Programmer's Handbook*, Morgan Kaufmann, San Francisco, CA, 1994

[4] Michael J. Flynn, *Computer Architecture-Pipelined and Parallel Processor Design*, Jones and Bartlett Publishers, Boston, 1995

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

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

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

[8] B.L. Jacob, et al., "Virtual Memory: Issues of Implementation," IEEE Computer, Vol. 31, No6, pp33-43, June 1998

[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*, pp118-123, 2001

[10] Naohiko Shimizu, and Ken Takatori, "A Transparent Linux Super Page Kernel for Alpha, Sparc64 and IA32 – Reducing TLB Misses of Applications," *ACM SIGARCH Computer Architecture News*, Vol. 31 Issue 1, March 2003

[11] SPARC International Inc. *The SPARC Architecture Manual Version 8*, 1992

[12] 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*, pp204-213, 1998.