BadgerTrap: A Tool to Instrument x86-64 TLB Misses

[1]Jayneel Gandhi [2]Arkaprava Basu [1]Mark D. Hill [1]Michael M. Swift

[1]Department of Computer Sciences
University of Wisconsin-Madison
Madison, WI, USA

{jayneel,markhill,swift}@cs.wisc.edu

[2]AMD Research
Advanced Micro Devices, Inc.
Austin, TX, USA

arkaprava.basu@amd.com

http://research.cs.wisc.edu/multifacet/BadgerTrap

ABSTRACT

The overheads of memory management units (MMUs) have gained importance in today's systems. Detailed simulators may be too slow to gain insights into micro-architectural techniques that improve MMU efficiency. To address this issue, we propose a novel tool, BadgerTrap, which allows online instrumentation of TLB misses. It allows first-order analysis of new hardware techniques to improve MMU efficiency. The tool helps to create and analyze x86-64 TLB miss trace. We describe example studies to show various ways this tool can be applied to gain new research insights.

1. INTRODUCTION

Memory management units (MMUs) are a critical component of modern computers. They provide programmers with virtual memory abstraction, which helps improve performance, security, and programmer productivity. The overhead of TLB misses has recently gained importance because TLB sizes are not scaling with the growth of physical memory. Figure 1 shows the amount of physical memory that can be bought with $10,000 in the last 20 years. In contrast, the number of TLB entries per core has barely grown (see Table 1). This discrepancy has renewed interest in MMU research [1,3,4]. To quote from a recent ACM Turing Award Lecture:

"Virtual memory was invented in a time of scarcity.
Is it still a good idea?"
- Charles Thacker, 2010 ACM Turing Award Lecture

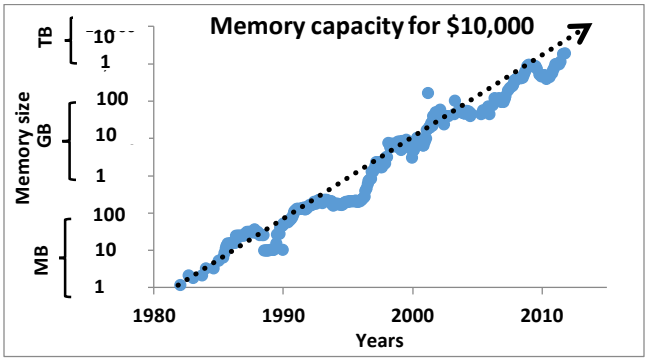

Figure 1: Memory capacity per system in $10,000 [2]

Table 1: TLB sizes in Intel processors over the years

Year	1999	2001	2008	2012
Processor	**Pent. III**	**Pent. 4**	**Nehalem**	**IvyBridge**
L1 DTLB entries	72	64	96	100
L1 ITLB entries	32	64	64	64
L2 TLB entries	NA	NA	512	512

MMU research is often performed with cycle-level simulators like gem5 [5]. While these simulators provide the flexibility of modeling any architectural innovation, they suffer from three drawbacks. First, TLB misses are rare events (a few per thousand instructions), which require long simulations to accurately measure their performance. Especially for big-memory workloads, a full-system simulation would take weeks, if not months, of simulation time to provide any insightful information. Second, the memory requirement for running big-memory applications in a cycle-level simulator is much higher, requiring long startup times to initialize memory. Also, a single simulation point in gem5 takes at least twice as much physical memory as the workload [1]. Third, OS-level issues like timer-interrupts are usually approximately modeled, which makes simulators not good for studying systems–level issues like TLB misses. Therefore, it is challenging to use such detailed simulators to gain insight into micro-architectural techniques for improving MMU efficiency.

We address this issue with the *BadgerTrap* tool, which allows online instrumentation of TLB misses. It enables a higher-level analysis of a large number of TLB misses to get a better understanding of a proposed architecture. Specifically, BadgerTrap enables analysis of hardware or software functions that affect x86-64 TLB misses, such as new page table layouts or address translation mechanisms. These are especially important to study in big-memory workloads, where TLB misses are more frequent [1].

BadgerTrap intercepts each hardware-assisted page walk on an x86-64 TLB miss and converts it into a page fault

handled specially inside the kernel with a new software-assisted TLB miss handler. The handler can be extended for online analysis while running applications. Workloads being analyzed by BadgerTrap slow down by about 2x to 40x based on their rate of TLB misses. However, BadgerTrap runs orders of magnitude faster than binary instrumentation tools like Pin [8]. The main contributions of the tool are:

- a novel tool to intercept both data and instruction TLB misses, converting hardware-assisted page walks to software-assisted page walks,
- use of software-assisted page walks to instrument and analyze x86-64 TLB misses on real hardware, which is orders of magnitude faster than full-system cycle-level simulators.

The rest of the paper is organized as follows: Section 2 describes the mechanism to intercept TLB misses for instrumentation. Section 3 gives some example use cases of the tool. This tool can be used to analyze TLB misses and generate real-system memory trace. Section 4 describes some of the limitations of the tool and ways to extend the tool to support more exotic features available in Linux.

2. DESIGN

This section describes the mechanism to convert a hardware-assisted page walk on an x86-64 TLB miss into a software-assisted page walk using an instrumented Linux kernel. The design has four main components:

1. **Intercepting TLB misses** by marking PTEs as invalid/poisoned
2. **Software-assisted TLB miss handler** to handle TLB misses being intercepted
3. **Attaching BadgerTrap** to processes whose TLB misses are to be instrumented
4. **Instrumenting TLB misses** to perform interesting studies on different programs.

Intercepting TLB misses: To intercept the hardware page walker, we *poison* the PTEs at the leaves of the page table to force the system to trap rather than load a PTE. In x86-64 systems, on a TLB miss, a hardware page-table walker walks the four-level page table to load a new TLB entry for virtual address that needs translation.

BadgerTrap poisons a PTE by setting a reserved bit (one of bits 48-51) in a PTE (see Figure 2). The poisoned PTEs can be at L4, L3 and L2 levels of the page table to support all page sizes (4KB, 2MB and 1GB, respectively). This causes the hardware page walker to raise a page fault exception with RSVD bit set in the page fault exception flags [6]. Using a reserved bit rather than the valid bit allows the page-fault handler to quickly determine from the RSVD flag whether the fault is real or caused by instrumentation, without accessing memory. We instrument the Linux kernel to handle this exceptional page fault with a special TLB miss handler. Note that these faults occur whether the page is referenced from user mode or kernel mode, allowing BadgerTrap to track kernel-induced TLB misses on user memory.

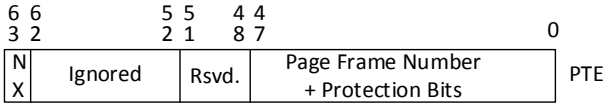

Figure 2: Format of a 64-bit Page Table Entry [6]

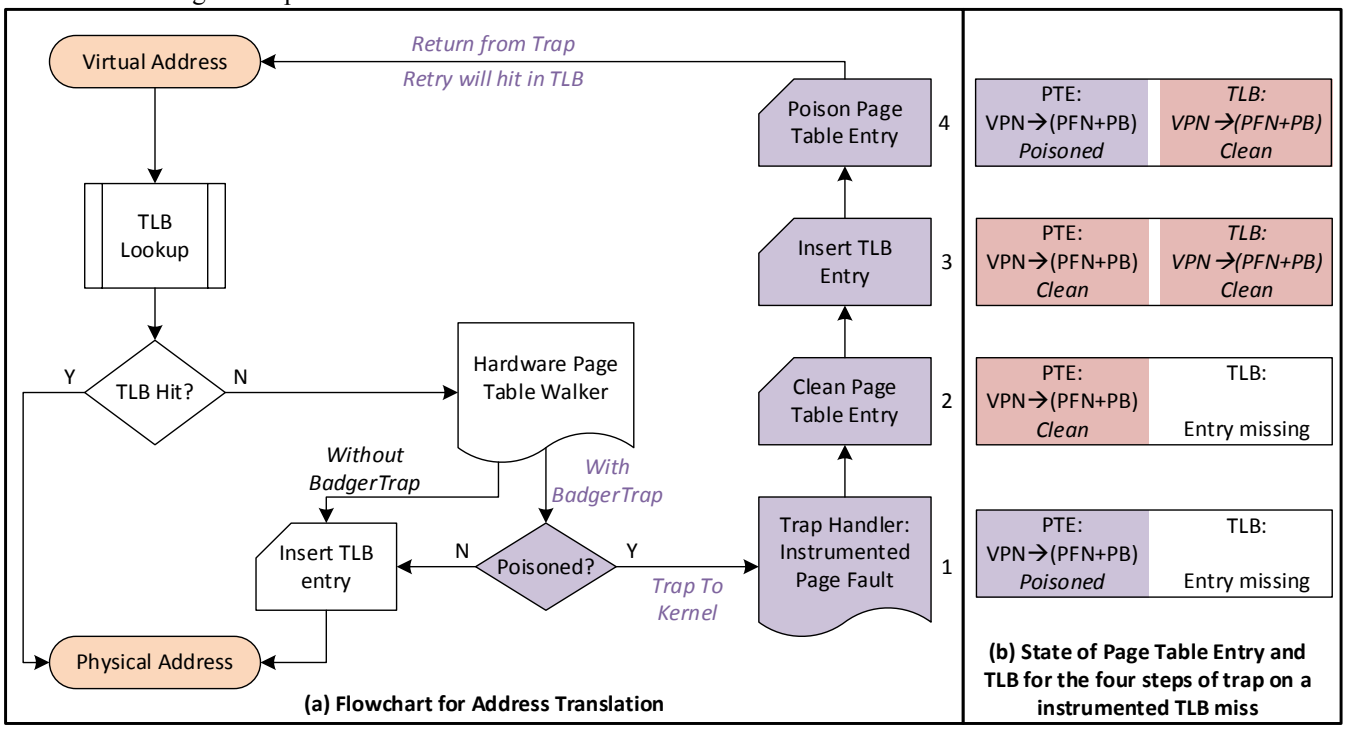

Figure 3: Flowchart for each translation with and without BadgerTrap along with state of PTE and TLB during an instrumented TLB miss

Software-assisted TLB miss handler: To make forward progress, BadgerTrap handles these exceptional TLB misses by *inserting* a translation into the TLB. Figure 3 (a) shows the steps (marked 1-4) involved in handling this exceptional TLB miss. While handling the TLB miss in kernel mode, we un-poison the PTE for which the exception was raised (clear the reserved bit). We introduce the correct PTE into the TLB by referencing the page, and then poison the PTE again. This approach works since the x86-64 architecture allows the page tables and TLBs to be incoherent. Thus, the TLB can cache translations until the OS explicitly invalidates the translation. Figure 3 (b) shows how incoherence can exist between PTE and TLB entry with each step in the TLB miss handler.

The main trick involved in the TLB miss handler is explicitly loading both user data and instruction TLB entries for a process while in kernel-mode. To load a DTLB entry from kernel mode, BadgerTrap reads from the virtual address causing the fault. This causes the page table walker to load the (now valid) PTE into the DTLB.

To load an ITLB entry is more involved. To introduce an ITLB entry while in kernel mode, an instruction needs to be executed on the page containing the faulting address. The kernel cannot start executing user code, because it would not regain control. Instead, we use a technique similar to Rosenblum's *context sensitive mappings* [9]. We *dynamically overwrite* the first 13 bytes of the faulting user code page by saving the original instructions and adding a jump instruction to regain control. After executing this code, we replace the original instructions in the user code page before restarting execution in user-mode.

Every core has its own MMU unit to support multithreaded and multi-programmed workloads. Our tool works naturally with multiprogrammed workloads. But it works with multithreaded programs only for DTLB misses. This limitation on ITLB misses comes from the usage of the above technique which exposes our code patch to other threads in user-mode. We are not aware of an effective mechanism for each core to have exclusive access to a code page while it dynamically writes to it.

Enabling BadgerTrap: BadgerTrap can be attached to a running process by providing its process ID to the newly created system call. It can also be attached to a new process at startup by passing the program's filename to the tool. Every time a binary with that name is launched, BadgerTrap automatically attaches to it. We provide a user-mode utility that works as a wrapper for these details and provide an easy to use interface for the user.

At process startup or when attached dynamically, BadgerTrap walks the page tables and marks each leaf PTE as poisoned. In addition, as a program makes progress, whenever a physical page is allocated to the process by having a page fault, we intercept these page faults and poison the newly created PTEs. This helps in keeping all leaf PTEs in the dynamically changing page table poisoned while the process is running.

Instrumenting TLB misses: From the function in which we handle these exceptional TLB misses, we can instrument the misses to perform various studies. We have access to various process-level structures and registers like the task structure, program counter, faulting virtual address and the physical page address, when we are in the software-assisted page fault handler. We discuss examples of different studies using this mechanism in the next section.

3. USING BADGERTRAP

In this section, we discuss a few ways to use the tool to perform interesting architectural studies. We will cover two studies in general which have been published using the same mechanism.

3.1 Study 1: Direct Segments

Direct segments use a form of segmentation along with paging to largely eliminate virtual memory overhead for big-memory workloads on native hardware [1]. A direct segment maps a portion of a process's linear address space with a segment rather than paging. Thus, a large chunk of a contiguous virtual address space can be mapped to contiguous physical addresses with only three registers per hardware context: BASE, LIMIT and OFFSET. For compatibility, the rest of the linear address space is mapped using conventional paging. On a memory reference, the processor consults the segment registers and L1 TLB in parallel, with at most one match.

Basu et al. [1] used an earlier version of the tool that later evolved into BadgerTrap. The TLB misses were instrumented and analyzed to split into two categories: the TLB misses that would be eliminated using the new hardware and the TLB misses that would be serviced by conventional paging. Each TLB miss was put in the respective bin based on the virtual address of the TLB miss. The authors developed a linear model to estimate the reduction in TLB miss handling cost using performance counters and the above information. The authors used this simple model to estimate performance improvement using such a hardware technique.

3.2 Study 2: Coalesced and Shared MMU

A coalesced MMU uses the spatial contiguity available in PTEs to coalesce them into a single entry, thereby increasing the reach of the MMU cache. In addition, a shared MMU which is shared between cores allows multiple cores to share MMU cache entries [4], thereby improving capacity of the MMU cache. Bhattacharjee recently proposed a hardware-software co-design with these two optimizations to reduce MMU overheads [4].

To evaluate such an MMU optimization, the author created real-system memory trace tool using an independently developed mechanism similar to BadgerTrap. The tool dumps system memory trace on allocation of a new DTLB entry. This trace has a list of distinct page references to the

DTLB. The trace basically can be thought of as a trace generated by a one-entry DTLB having DTLB misses. To create a DTLB miss trace for a one-entry DTLB, DTLBs are flushed between every DTLB miss detected. They used such a memory trace in order to get an estimate of improvement in hit-rate of their new MMU design.

3.3 Performance

Since we are converting a TLB miss to a software-assisted TLB miss handler, BadgerTrap does slow down the application being analyzed. BadgerTrap, in general, slow down workloads by around 2x to 40x based on the rate on TLB misses. Binary instrumentation tools like Pin [8], which instrument all instructions, usually slows down applications by a magnitude higher than BadgerTrap.

3.4 Discussion

BadgerTrap, with its limited-support for ITLB, can be used to dump real-system memory trace similar to the tools like Pin [8]. BadgerTrap helps to induce a DTLB miss for every memory access by flushing both TLBs while servicing any TLB miss. This support improves upon the memory-system trace used by Bhattacharjee for his analysis (Section 3.2) by having a DTLB miss trace for a zero-entry DTLB instead of a one-entry DTLB using ITLB support in BadgerTrap.

BadgerTrap can also be applied to analyze TLB misses in a virtual machine by attaching BadgerTrap to a process running inside of a guest OS. It works with Linux running on both VMware- and KVM-based virtual machines.

In the current form as released, BadgerTrap prints only the count of DTLB misses, but the TLB miss handler can be instrumented to dump real-memory system trace or instrument to perform other interesting studies. The user of the tool will have to instrument the Linux kernel to perform such studies. But since the tool has streamlined the function for instrumentation, we expect the instrumentation step will be fairly easy to write.

Some benefits of using BadgerTrap are:

1. The tool can help capture traces with real-systems effects along with physical addresses which other tools like Pin [8] do not.
2. The trace generation is faster than tools like Pin [8] since we only instrument memory references and not all instructions.
3. The tool helps capture more detailed information in the presence of additional levels of abstractions (e.g. virtual machines).

4. LIMITATIONS AND FUTURE WORK

This software has only been tested with Linux Kernel v3.12.13. This tool may need to be tweaked to port it to older or newer kernels. The steps provided above are generic but are only tested on an Ubuntu- and Fedora-based operating systems. We suggest that users gain some experience with Kernel Development before using this tool.

The mechanism may not work for a 32-bit system since their page-table entries do not have reserved bits. This mechanism has been tested on various Intel x86-64 processors. As per our knowledge, AMD processors may support reserved faults and thus the tool may work on AMD processors. The reserved bits in the PTEs may be different from that of Intel processors.

BadgerTrap can also be attached to multithreaded programs with the exception of ITLB (see Section 2). Even with multithreaded programs, the slowdown is not much larger than single-threaded programs since we use the distributed locking already inbuilt into the page table structure.

This tool currently does not support NUMA memory, Kernel Samepage Merging (KSM) [7] and Kernel TLB misses in Linux and many more exotic features available in the Linux kernel. But the tool can be easily be used to support these different memory management optimizations.

ACKNOWLEDGEMENTS

We thank Abhishek Bhattacharjee for his insightful comments and feedback on the paper. We thank Chris Feilbach, Sujith Surendran, Vasilis Karakostas and Somayeh Sardashti for their feedback on the tool. This work is supported in part by the National Science Foundation (CNS-0720565,CNS-0834473, CNS-0916725, CNS-1117280, CCF-1218323, and CNS-1302260), Google, and the University of Wisconsin (Kellett award and Named professorship to Hill). Arkaprava Basu's contribution to the tool occurred while at University of Wisconsin-Madison.

REFERENCES

1. Arkaprava Basu, Jayneel Gandhi, Jichuan Chang, Mark D. Hill, and Michael M. Swift. 2013. Efficient virtual memory for big memory servers. *In Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA '13)*. ACM, New York, NY, USA, 237-248.
2. Arkaprava Basu. Revisiting Virtual Memory. Ph.D. Thesis. University of Wisconsin-Madison, 2013.
3. Abhishek Bhattacharjee, Daniel Lustig, and Margaret Martonosi. 2011. Shared last-level TLBs for chip multiprocessors. *In Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture (HPCA '11)*. IEEE Computer Society, Washington, DC, USA, 62-63.
4. Abhishek Bhattacharjee. 2013. Large-reach memory management unit caches. *In Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-46)*. ACM, New York, NY, USA, 383-394.
5. Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The gem5 simulator. *SIGARCH Comput. Archit. News* 39, 2 (August 2011), 1-7.
6. *Intel® 64 and IA-32 Architectures Software Developer's Manual Combined Volumes: 1, 2A, 2B, 2C, 3A, 3B, and 3C*. .
7. KSM - KVM. http://www.linux-kvm.org/page/KSM.
8. Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. 2005. Pin: building customized program analysis tools with dynamic instrumentation. I*n Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation (PLDI '05)*. ACM, New York, NY, USA, 190-200.
9. Nathan E. Rosenblum, Gregory Cooksey, and Barton P. Miller. 2008. Virtual machine-provided context sensitive page mappings. *In Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments (VEE '08)*. ACM, New York, NY, USA, 81-90.