Design, Implementation and Evaluation of the SVNAPOT Extension on a RISC-V Processor

Nikolaos-Charalampos Papadopoulos[1]*, Stratos Psomadakis[1], Vasileios Karakostas[2]
Nectarios Koziris[1] and Dionisios N. Pnevmatikatos[1]

[1]National Technical University of Athens
[2]National and Kapodistrian University of Athens

Abstract

The RISC-V SVNAPOT Extension aims to remedy the performance overhead of the Memory Management Unit (MMU), under heavy memory loads. The Privileged Specification defines additional Natural-Power-of-Two (NAPOT) multiples of the 4KB base page size, with 64KB as the default candidate. In this paper we extend the MMU of the Rocket Chip Generator, in order to manage the collocation of 64KB pages along with 4KB pages in the L2 TLB. We present the design challenges we had to overcome and the trade-offs of our design choices. We conduct a preliminary sensitivity analysis of the L2 TLB with different configurations/page sizes. Finally, we summarize on techniques which could further improve memory management performance on RISC-V systems.

Design

We extend the L2 TLB of Rocket[1] and add support for SVNAPOT[2], targeting the Page-based 39-bit Virtual Memory systems (*sv39*) scheme. We choose to collocate both 4KB and 64KB in the same structure, in order to be able to take advantage of every L2 TLB entry. We modify each entry, and add an N bit which if set, it indicates that the TLB entry holds a 64K page. Structures that collocate different page sizes exhibit an indexing problem: For each TLB lookup we do not know the page size and thus cannot determine the set (index) it may reside in. The common methods of resolving this problem are (1) multiple lookups for different page sizes, (2) Virtual-Page-Number (VPN) partitioning and (3) companion TLBs. We choose VPN partitioning in order to avoid wasting cycles in multiple lookups. We ignore the last 4 bits of the VPN, and select the next log2(#sets) bits (figure 1) as INDEX. We modify the following operations in order to be able to handle two page sizes:

Insert Set the N bit according to the PTE returned by the Page Table Walker (PTW).

Lookup If N=1 we found a 64KB page → return (PPN $\gg$ 4) + NAPOT_OFFSET. If N=0 return PPN.

Flush Locate INDEX and flush the whole set.

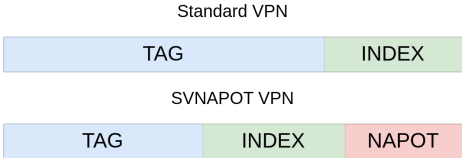

Figure 1: *VPN Partitioning*

Methodology

Software We use an unmodified v6.8-rc5 Linux kernel, which implements the SVNAPOT extension and supports allocations for 4KB and 64KB pages. Since 64KB SVNAPOT pages are not yet transparently supported, we opt to pre-allocate 8GB of 64KB pages. For 64KB allocations, the kernel includes the MAP_HUGE_64KB flag for the mmap() system call.

Hardware The system under test has a single Rocket core with an unmodified 32-entry DTLB and an 8-entry PTW Cache. We set up 4 different evaluation scenarios (Table 1) along 2 different L2 TLB configurations, a (1) 4-way set-associative and (2) 16-way set-associative organization. Both configurations have 1024 entries.

Simulation We employ FireSim[3], a fast FPGA-accelerated hardware simulator with a Xilinx Alveo U280 board that supports 16GB of DRAM.

Conf. No	Associativity	Page Size
①	4-way	4KB
②	16-way	4KB
③	4-way	64KB
④	16-way	64KB

Table 1: *Evaluated Configurations*

Evaluation

In order to evaluate our design, we compile a comprehensive sensitivity analysis using a microbenchmark that targets different memory requirements and access patterns. This TLB-stress microbenchmark spans a range of memory chunks from 4KB to 256MB, with a linear access pattern Ⓐ and a random access pattern Ⓑ. For each iteration, TLB-stress warms up the TLBs and then performs 1M memory accesses with 4KB step in order to stress the system. We also properly modify

*Corresponding author: ncpapad@cslab.ece.ntua.gr

and test the hashjoin benchmark, in order to be able to use 64KB page allocation.

In Figure 2, we exhibit the results we gathered with TLB-stress. In Ⓐ, TLB-stress performs a linear page stride for all configurations. We observe that in ① the L2 TLB gets thrashed after the 64KB mark. This happens due to the 4-bit shift in the TLB index that causes conflicts in every set; ignoring the last 4-bits means the index changes in a 16-page interval. Also, until the 128KB mark most lookups are serviced by the L1 DTLB. Early L2 thrashing is remedied in ②, in which the L2 TLB exhibits the expected 4MB reach. Configurations ③, ④ showcase a rapid decrease in misses (misses reached up to 70K). In Ⓑ, TLB-stress performs random accesses for ②, ④. We observe that both configurations present zero misses until the 4MB and 64MB mark for ② and ④ respectively, with ④ indicating 16× larger reach. Finally, in Ⓒ we demonstrate latency results in terms of cycles during a page stride for configurations ② and ④. Following the 16KB mark, ④ outperforms ② by 3-5%.

Regarding hashjoin, we tested ② and ④. Using 64KB pages, hashjoin exhibited 4.2% speedup in terms of cycles while reducing L2 TLB misses by 14.2%.

Latency The cost of one 64KB page PTW (TLB refill) is equivalent to (and saves up to) 16×4KB PTWs. L2 TLB Lookup latency is not affected and is identical for both page sizes (3 cycles to report hit/miss).

Area Negligible, 1.1% overhead in the L2 TLB structure (1024 N bits), plus simple logic in order to distinguish between 4KB and 64K pages. We do not add any other persistent elements such as registers/SRAMs.

Power A PTW is a costly operation since multiple memory reads are needed in order to locate the PPN. This can get worse for schemes that support extra page table levels, such as *sv48* and *sv57*, and much worse for 2D PTWs needed for virtualised systems. Our design may be more power efficient than a standard L2 TLB, since it can support up to 16× larger reach and thus may reduce costly PTWs.

Future Work

In this section we summarize further techniques with which SVNAPOT could improve system performance.

TLB Organization We chose to evaluate VPN partitioning as a first approach to SVNAPOT due to the simplicity of the changes we had to make in the L2 TLB. Different approaches/organizations such as multiple TLB lookups or companion TLBs would yield different results[4] and may benefit architectures that support larger page sizes (128K, 256K etc.).

Contiguity This is of critical importance, as without ample contiguity we cannot allocate large physical memory areas backed up by larger pages [5]. Increasing contiguity requires OS support.

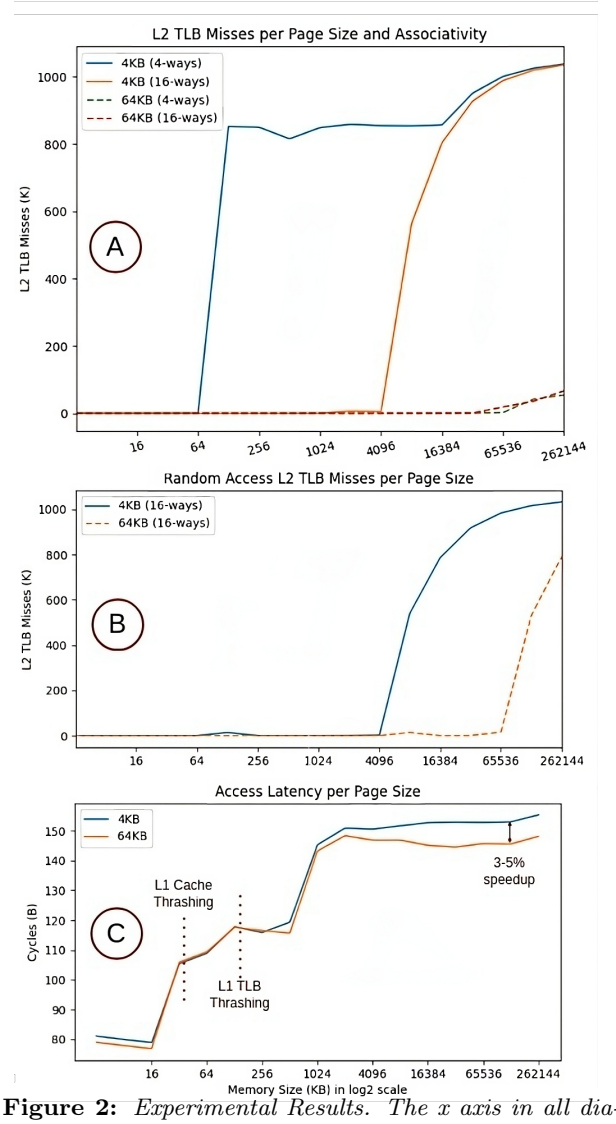

Figure 2: *Experimental Results. The x axis in all diagrams reports Memory Size (KB) in log2 scale.*

Transparency Larger allocations that can be backed up by contiguous regions could automatically be serviced by the kernel. Linux release v6.9 will support multi-size Transparent Huge Pages (mTHP) for ARM; RISC-V is not yet supported.

References

[1] Krste Asanović et al. *The Rocket Chip Generator.*

[2] Andrew Waterman et al. *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Document Version 20211203.*

[3] Sagar Karandikar et al. "FireSim: FPGA-Accelerated Cycle-Exact Scale-Out System Simulation in the Public Cloud". In: *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA).*

[4] Nikolaos Charalampos Papadopoulos et al. "A Configurable TLB Hierarchy for the RISC-V Architecture". In: *2020 30th International Conference on Field-Programmable Logic and Applications (FPL).*

[5] Chloe Alverti et al. "Enhancing and Exploiting Contiguity for Fast Memory Virtualization". In: *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA).*