Evaluating the Impact of Branch Predictor Design on Spectre Attacks

Michael Shen
Northeastern University
Boston, Massachusetts, USA
shen.mich@northeastern.edu

Derek Rodriguez
Northeastern University
Boston, Massachusetts, USA
rodriguez.der@northeastern.edu

David Kaeli
Northeastern University
Boston, Massachusetts, USA
kaeli@ece.neu.edu

Abstract—**Security research targeting today's high-performance CPU microarchitectures helps to insure that tomorrow's program execution will be secure and reliable. With the adoption of branch predictors and speculative execution to overcome data and control dependencies on nearly every microprocessor on the market today, timing side channel attacks have become a critical issue. In this paper we explore how different branch predictor designs, implemented on the SonicBOOM RISC-V architecture, can improve performance, but are also susceptible to side channels.**

I. Introduction

Given the need for performance improvements by microprocessor manufacturers (i.e., AMD, Intel, and ARM) to deliver the fastest design, we continue to see microarchitecture improvements. While this focus has produced highly efficient microprocessors, there has not been enough attention devoted to considering the vulnerabilities of these features.

Execution time on a microprocessor can vary when performing the same operation with different input data values, allowing malicious users to infer knowledge about what is being executed without permission to do so. The recent news of Spectre attacks [1] impacts a large number of commercial microprocessors. This attack targets branch prediction hardware to leak secret data to an attacker. Spectre is just one of many hardware attacks that has been reported to exploit various design features on CPU architectures. There have been a number of high-profile attacks reported over the past 5 years: TLBleed [2], Meltdown [3], and Foreshadow [4].

Even though researchers continue to make progress on mitigating these vulnerabilities in new microarchitectures (e.g., the Tage branch predictor possesses some inherent architectural resistance to specific variants of Spectre), new vulnerabilities are continually being reported. For instance, within the last year HertzBleed [5], a vulnerability that affects AMD/Intel processors, and PACMAN [6], an attack on Apple's M1 SoCs, have been reported.

The goal of this research is to further our understanding of the relationship between branch prediction design choices and their potential for exposing intrusion attack surfaces for side-channel attacks similar to Spectre. By quantitatively evaluating changes made during the microarchitectural design phase, computer architects can better characterize the security and performance of next generation architectures before they are committed to silicon. By tempering microarchitectural choices in terms of their potential to expose security holes, the number eventual vulnerabilities will be reduced, reducing the cost to remediate these insidious issues.

In this paper we present BranchBench, a tool for generating code sequences of branch instructions that are paired with timing information. We demonstrate how these benchmarks can be used to evaluate microarchitectural features such as branch predictors, in terms of side-channel resilience and performance.

II. Background

A. Spectre & Side Channel Attacks

Spectre [1] is a class of side channel attacks that exploits the execution features of many CPU architectures, including branch predictors and speculative execution. A simple fix would be to disable these features to mitigate the effects of Spectre, but both of these features provide major performance advantages for the CPU microarchitecture [7].

Branches can be taken or not taken, supporting the execution of high-level language syntax that implements if/then/else clauses and different forms of conditional loops. The microprocessor stalls whenever a branch is incorrectly predicted, since the pipeline needs to be flushed, and instructions on the correct path need to be fetched and decoded. Additionally, speculative execution and branch predictors make verifying a microarchitecture much more difficult. However, by predicting whether branches are taken or not taken ahead of time, selective instructions can be speculatively executed early, improving the overall instruction throughput within a compute pipeline.

To test a timing-based side channel, we can execute an instruction sequence which will cause the branch predictor to mispredict. The potential side effects of a branch misprediction (e.g., instructions that are fetched and speculatively executed, and the recovery of mispredicted instruction information) can then be exploited using the side channel.

B. Akita

Akita is an event-driven microarchitectural simulation framework. Akita's framework is reusable and allows for efficient development for modeling different CPU/GPU architectures [8]. Akita's engine includes features, such as components and ports, that allow users to easily change aspects of a simulator design (e.g., adding branch predictors).

C. SonicBOOM

In this work we utilize the SonicBOOM microarchitecture, an out of order core that targets the RISC-V instruction set. SonicBOOM was implemented using ChipYard, which generates Verilog from source code that is simulated in Verilator [9].

D. GShare

The Gshare [10] branch predictor is a microarchitecture structure that is used to predict branches in a microprocessor pipeline. Gshare uses a history register to record global history and notes the address of the branch instruction. When a branch instruction is encountered, the Gshare predictor indexes into a table of 2-bit counters using the location of the instruction (i.e., the program counter) XORed with the past history of all branches (the history is a binary vector of bits, indicating whether past branches were taken (1) or not-taken (0)). The history length can vary depending on the specific implementation. The 2-bit counters take on one of four values, (00) strongly not-taken, (01) weakly not-taken, (10) weakly taken and (11) strongly taken. The counter is incremented when the branch is taken, and decremented when not taken. The counter saturates at values of (00) and (11). The 2-bit counters are initialized to a weakly not-taken (01) state. Future branch instructions either utilize the existing counter entry for that branch or allocate a new entry to predict the outcome of future executions of this branch.

E. Tage

The Tage predictor uses several predictor tables of geometrically increasing size, each with its own accompanying global history register. Each entry of the predictor table consists of a prediction counter, usefulness counter, and a tag. The history of each table is hashed with the incoming instruction to produce the current branch's prediction. The table with the longest history is then selected for the actual prediction.

III. Methodology

A. BranchBench

Yori [11] is a RISC-V microarchitectural simulator built on Akita's event-driven simulation framework and SonicBOOM's [12] out of order execution microarchitecture. We are expanding the current capabilities of Yori to include simulation models of the GShare and Tage branch predictors. In order to validate the accuracy of our predictors, we have developed benchmarks using *BranchBench*.

The benchmarks generated by BranchBench contain sequences of assembly instructions, paired with timing data, so that we can better observe the simulated behavior of each branch (as seen in Listing 1). BranchBench is easily configurable so that we can vary the number of branch iterations and the architecture. These benchmarks are used to verify our work against the Chipyard standard using Verilator.

We execute the generated programs from BranchBench on Spike [13] and the reference implementation of SonicBOOM[12] via Verilator simulation. This generates two sets of instruction-level traces that contain information on branch instruction execution. By cross-referencing the branch instruction outputs and treating SonicBOOM's output as ground truth, we can verify the implementation of our branch predictors.

B. Expanding Yori's Feature Set

While Yori can faithfully simulate the RISC-V integer and CSR (Control Status Register) instruction set, easily understanding instruction execution is challenging to do with Yori's current implementation. Interpreting and tracking how instructions travel through the microarchitecture is difficult, even when utilizing techniques such as breakpoint debugging. Fetch packet information that

```c
#include <stdint.h>
#include <stdio.h>
#ifdef ARCH_GENERIC
    #include <time.h>
#endif
#ifdef ARCH_AMD64
    #include <x86intrin.h>
#endif

void function0() {
    int out, junk, addr;
    /* READ TIMER */
    uint64_t time1 = __rdtscp( & junk);
    printf("%ld\\n", time1);
}

void function1() {
    int out, junk, addr;
    /* READ TIMER */
    uint64_t time1 = __rdtscp( & junk);
    printf("%ld\\n", time1);
}

void function2() {
    int out, junk, addr;
    /* READ TIMER */
    uint64_t time1 = __rdtscp( & junk);
    printf("%ld\\n", time1);
}

int main() {
    for(size_t i = 0; i < 2; i++){
        function2();
        function1();
        function0();
    }
    return 0;
}
```

Listing 1: An example of a Benchmark (written in C) that is generated by BranchBench. The example calls 3 functions, and is run for 2 iterations targeting a generic architecture.

contains essential information (e.g., instruction, destination, program counter, time of execution, etc.) must be decoded several times before being readable. This process is arduous and makes Yori difficult to use. We are currently in the process of extending the features of Yori to improve upon this instruction-level observability.

After expanding the current capabilities of Yori, our plan is to utilize formal verification tools, such as CheckMate [14], to identify potential areas of vulnerabilities while assessing impacts on performance. Checkmate generates tests that the user can use to target different aspects of the hardware under varying conditions. We can use this enhanced version of Yori to detect and analyze potential avenues for Spectre attacks. We will additionally be able to map events precisely to a formal model of execution [11]

With Checkmate integrated into Yori, we can further explore novel approaches to protect the microarchitecture (e.g., implementing unique program execution checking (UPEC)) [15].

IV. Results & Discussion

Using BranchBench, we evaluate the performance of two different branch predictor designs: Gshare and a 2-bit branch predictor. Our implementation of Gshare maintains a global history register that is eight bits long and uses the last eight bits of the program counter. This means our Gshare implementation contains a matrix table of 256 unique 2-bit counter entries.

Number of Branch Instructions	70,998
Taken Branches	51,161
Not Taken Branches	19,837
Total Instruction Count	199,991

TABLE I: BranchBench benchmark RISC-V instruction level information.

The ELF binary of the benchmark shown in Listing 1 is used to generate the instruction counts reported in Table I after being run through Spike. By inspecting the program counters associated with branch instructions, we are able to determine the number of taken versus not taken branches. This information is included in Table I.

	Gshare	2-Bit Predictor
Branch Hits	70,877	59,544
Branch Misses	121	11,454
Hit Rate	99.83%	83.87%

TABLE II: Branch predictor hit rates for BranchBench-generated benchmarks for the Gshare, and 2-bit predictors.

	Gshare	2-Bit Predictor
Taken Branch Hits	51,040	42,676
Taken Branch Misses	121	8485
Not Taken Branch Hits	19,837	16,868
Not Taken Branch Misses	0	2969

TABLE III: Detailed branch predictor statistics.

The branch instruction execution data produced by Spike with our implementations for the Gshare and 2-bit predictors is shown in Table II and Table III. Our

BranchBench benchmark in conjunction with our branch predictor designs show that Gshare is 19% more accurate compared to the 2-Bit predictor and can speculatively handle not taken branches more consistently. We hope to extend this comparison to include the Tage predictor, which is currently in the process of being implemented. Given Tage's resistance to some Spectre variants and added layers of complexity, BranchBench benchmark's predictor performance data will be useful for examining the performance trade-off.

By using Branchbench benchmarks, we can quickly generate an abundance of branch instructions in RISC-V ELF binaries. In the future, timing register accesses will be used when simulating Spectre attacks. This gives the user additional observability on how branch predictors perform at an instruction level with Spectre attacks.

V. Related Work

Microarchitectural simulators are an essential tool for both monitoring and analyzing performance metrics, supporting efforts made by researchers to make hardware more secure. FPGA-based simulators such as Firesim are capable of multi-core simulation and can reliably reproduce Spectre attacks using BOOM's microarchitecture [16]. Zsim is another event-driven microarchitectural simulator that has been used to test defenses against cache-based timing related side-channel attacks [17]. MI6 [18] is a speculative out of order processor capable of FPGA simulation based on RiscyOO [19] and [20] Sanctum. MI6 explores the mitigation and performance impact associated with Secure Enclaves, instruction-level code that isolates and encrypts data.

Coppelia and Speculator, like Checkmate, can be used to identify exploits in the hardware and potential security threats [21], [22]. Coppelia is a tool that generates exploits in order for microarchitectures to contextualize security threats, while Speculator acts as a debugger that provides metrics relevant for identifying vulnerabilities similar to Spectre (i.e., program counters, branching instructions).

Previous research initiatives have made progress developing simulators targeting specific architectures and mitigating some variants of Spectre. Yori aims to improve and build upon this with our usage of the Akita framework. Akita's framework gives us the ability to swap out components, allowing for the potential to test different microarchitectures and attacks within a single simulator.

VI. Conclusion

The goal of this project is to develop a methodology that academia and industry can use to evaluate how microarchitectural changes affect performance, as well as expose potential vulnerabilities, associated with a microarchitecture. In this paper we described how our BranchBench tool can be used to evaluate various aspects of a microarchitecture's performance. In the future, we plan to incorporate BranchBench with a verification framework, and integrate this as part of our RISC-V BOOM simulator, Yori. We expect that Yori will be used to study various microarchitectures and provide reliable feedback on performance, as well as security. This will aid researchers working on hardware security to develop defenses against side channel attacks such as Spectre.

VII. Acknowledgements

The authors of this paper would like to thank: Northeastern's Undergraduate Research and Fellowship office for their support through a PEAK Summit Grant; The Center for Hardware and Embedded Systems Security and Trust (CHEST) for their advice and support with this project; Derek Rodriguez for his mentorship and guidance throughout the course of this project; and Dr. David Kaeli for his mentorship and support throughout this project and my continued research endeavors.

References

[1] N. Abu-Ghazaleh, D. Ponomarev, and D. Evtyushkin, "How the Spectre and Meltdown hacks really worked," *IEEE Spectrum*, vol. 56, no. 3, pp. 42–49, 2019.

[2] B. Gras, K. Razavi, H. Bos, and C. Giuffrida, "Translation Leak-aside Buffer: Defeating Cache Side-channel Protections with TLB Attacks," in *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD: USENIX Association, Aug. 2018, pp. 955–972.

[3] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, M. Hamburg, and R. Strackx, "Meltdown: Reading Kernel Memory from User Space," *Commun. ACM*, vol. 63, no. 6, p. 46–56, may 2020. [Online]. Available: https://doi.org/10.1145/3357033

[4] O. Weisse, J. Van Bulck, M. Minkin, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, R. Strackx, T. F. Wenisch, and Y. Yarom, "Foreshadow-NG: Breaking the virtual memory abstraction with transient out-of-order execution," *Technical report*, 2018.

[5] Y. Wang, R. Paccagnella, E. He, H. Shacham, C. W. Fletcher, and D. Kohlbrenner, "Hertzbleed: Turning power side-channel attacks into remote timing attacks on x86," in *Proceedings of the USENIX Security Symposium (USENIX)*, 2022.

[6] J. Ravichandran, W. T. Na, J. Lang, and M. Yan, "PACMAN: attacking arm pointer authentication with speculative execution," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ser. ISCA '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 685–698. [Online]. Available: https://doi.org/10.1145/3470496.3527429

[7] D. Kaeli and P.-C. Yew, *Speculative Execution In High Performance Computer Architectures (Chapman and Hall/Crc Computer and Information Science Series)*. USA: CRC Press, Inc., 2005.

[8] Y. Sun, T. Baruah, S. A. Mojumder, S. Dong, X. Gong, S. Treadway, Y. Bao, S. Hance, C. McCardwell, V. Zhao *et al.*, "MGPUSim: enabling multi-GPU performance modeling and optimization," in *Proceedings of ISCA-46*, 2019, pp. 197–209.

[9] V. Karakostas, K. Nikas, N. Koziris, D. N. Pnevmatikatos, and N. C. Papadopoulos, "Enabling Virtual Memory Research on RISC-V with a Configurable TLB Hierarchy for the Rocket Chip Generator," in *CARRV 2020*, 2020, pp. 1–7.

[10] S. Mittal, "A Survey of Techniques for Dynamic Branch Prediction," *Concurrency and Computation Practice and Experience*, vol. 31, 04 2018.

[11] G. Knipe, D. Rodriguez, F. Yunsi, and D. Kaeli, "RISC-V Microarchitecture Simulation State Enumeration," in *Fifth Workshop on Computer Architecture Research with RISC-V (CARRV 2021)*, 2021, pp. 1–6.

[12] J. Zhao, B. Korpan, A. Gonzalez, and K. Asanovic, "Sonicboom: The 3rd generation berkeley out-of-order machine," in *Fourth Workshop on Computer Architecture Research with RISC-V (CARRV 2020)*, 2020, pp. 1–7.

[13] A. Waterman, "Risc-v spike," 2022. [Online]. Available: https://github.com/riscv/riscv-tools

[14] C. Trippel, D. Lustig, and M. Martonosi, "Security Verification via Automatic Hardware-Aware Exploit Synthesis: The CheckMate Approach," *IEEE Micro*, vol. 39, no. 3, pp. 84–93, 2019.

[15] M. R. Fadiheh, J. Müller, R. Brinkmann, S. Mitra, D. Stoffel, and W. Kunz, "A formal approach for detecting vulnerabilities to transient execution attacks in out-of-order processors," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[16] S. Karandikar, H. Mao, D. Kim, D. Biancolin, A. Amid, D. Lee, N. Pemberton, E. Amaro, C. Schmidt, A. Chopra, Q. Huang, K. Kovacs, B. Nikolic, R. Katz, J. Bachrach, and K. Asanovic, "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)*, 2018, pp. 29–42.

[17] V. Kiriansky, I. Lebedev, S. Amarasinghe, S. Devadas, and J. Emer, "DAWG: a defense against cache timing attacks in speculative execution processors," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018, pp. 974–987.

[18] T. Bourgeat, I. Lebedev, A. Wright, S. Zhang, Arvind, and S. Devadas, "Mi6: Secure enclaves in a speculative out-of-order processor," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52. New York, NY, USA: Association for Computing Machinery, 2019, p. 42–56. [Online]. Available: https://doi.org/10.1145/3352460.3358310

[19] S. Zhang, A. Wright, T. Bourgeat, and Arvind, "Composable building blocks to open up processor design," in *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-51. IEEE Press, 2018, p. 68–81. [Online]. Available: https://doi.org/10.1109/MICRO.2018.00015

[20] V. Costan, I. Lebedev, and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Aug. 2016, pp. 857–874. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/costan

[21] R. Zhang, C. Deutschbein, P. Huang, and C. Sturton, "End-to-end automated exploit generation for validating the security of processor designs," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018, pp. 815–827.

[22] A. Mambretti, M. Neugschwandtner, A. Sorniotti, E. Kirda, W. Robertson, and A. Kurmus, "Speculator: A tool to analyze speculative execution attacks and mitigations," in *Proceedings of the 35th Annual Computer Security Applications Conference (San Juan, Puerto Rico) (ACSAC '19)*, 2019, pp. 747–761.