ShadowBinding: Realizing Effective Microarchitectures for In-Core Secure Speculation Schemes

Amund Bergland Kvalsvik
amund.kvalsvik@NTNU.no
Norwegian University of Science and Technology
Trondheim, Norway

Magnus Själander
magnus.sjalander@NTNU.no
Norwegian University of Science and Technology
Trondheim, Norway

Abstract

Secure speculation schemes have shown great promise in the war against speculative side-channel attacks and will be a key building block for developing secure, high-performance architectures moving forward. As the field matures, the need for rigorous microarchitectures, and corresponding performance and cost analysis, become critical for evaluating secure schemes and for enabling their future adoption.

In ShadowBinding, we present effective microarchitectures for two state-of-the-art secure schemes, uncovering and mitigating fundamental microarchitectural limitations within the analyzed schemes, and providing important design characteristics. We uncover that Speculative Taint Tracking's (STT's) rename-based taint computation must be completed in a single cycle, creating an expensive dependency chain that limits performance for wider processor cores. We also introduce a novel microarchitectural approach for STT, named STT-Issue, which, by delaying the taint computation to the issue stage, eliminates the dependency chain, achieving better instructions per cycle (IPC), timing, area, and performance results.

Through a comprehensive evaluation of our STT and Non-Speculative Data Access (NDA) microarchitectural designs on the RISC-V Berkeley Out-of-Order Machine, we find that the IPC impact of in-core secure schemes is higher than previously estimated, close to 20% for the highest performance core. With insights into timing from our RTL evaluation, the performance loss, created by the combined impact of IPC and timing, becomes even greater, at 35%, 27%, and 22% for STT-Rename, STT-Issue, and NDA, respectively. If these trends were to hold for leading processor core designs, the performance impact would be well over 30%, even for the best-performing scheme.

Through these findings, research sentiments that Spectre can be solved by in-core secure schemes at low-performance costs are challenged. ShadowBinding serves as a call to arms for a more in-depth evaluation of secure speculation schemes and further work on in-core methods and optimizations, which can help mitigate the high performance cost of current state-of-the-art schemes without requiring extensive and expensive modifications.

MICRO '25, October 18–22, 2025, Seoul, Republic of Korea

ACM ISBN 979-8-4007-1573-0/2025/10
https://doi.org/10.1145/3725843.3756074

CCS Concepts

• **Security and privacy → Hardware security implementation**; **Side-channel analysis and countermeasures**.

ACM Reference Format:
Amund Bergland Kvalsvik and Magnus Själander. 2025. ShadowBinding: Realizing Effective Microarchitectures for In-Core Secure Speculation Schemes. In *58th IEEE/ACM International Symposium on Microarchitecture (MICRO '25), October 18–22, 2025, Seoul, Republic of Korea.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3725843.3756074

1 Introduction

The Spectre attacks [27] acted as a wake-up call for the architectural community, placing security front-and-center for microarchitectural design. High-performance processors are dependent on a series of complex microarchitectural optimizations for high performance, including extensive speculation. Speculation has introduced a slew of speculative side-channel attacks, which have grown in prominence [42, 43], scope [9, 49, 53], and number [1, 6, 7, 15, 26, 32, 35, 48] since the release of Spectre [27] and Meltdown [33] in 2018. A multitude of software-based mitigation strategies have been developed, including improvements in compilation [54], libraries [18, 39], operating systems [25], and even microcode patches [19, 38]. However, as the attack vector inherently exists in the microarchitecture, microarchitectural mitigation strategies are critical for creating robust and low-cost defenses against these attacks [36], and many secure speculation schemes have been developed over the past few years [2–5, 10, 22, 29, 34, 40, 44–47, 55, 56, 58, 61].

Secure schemes come with trade-offs regarding the types of attacks they can block, i.e., how strict they are, and at what performance penalty. Schemes that are both strict and high-performance usually require substantial microarchitectural modifications. Some schemes even employ modifications beyond the core itself, such as the memory hierarchy [3, 56, 57] or software [28, 54], to improve performance. Particularly interesting are in-core secure schemes, which aim to block speculative side-channel attacks without modifying the memory hierarchy or requiring software changes, due to the cost and complexity of such changes.

ShadowBinding, named for trying to bind the impact of speculation shadows, presents realizable microarchitectural designs for two state-of-the-art in-core secure speculation schemes, namely Non-Speculative Data Access (NDA) [55] and Speculative Taint Tracking (STT) [58]. Our microarchitectural designs help uncover important limitations and benefits of the different secure speculation schemes and uncover novel design challenges that were not previously discovered during more abstract evaluation using architectural simulators.

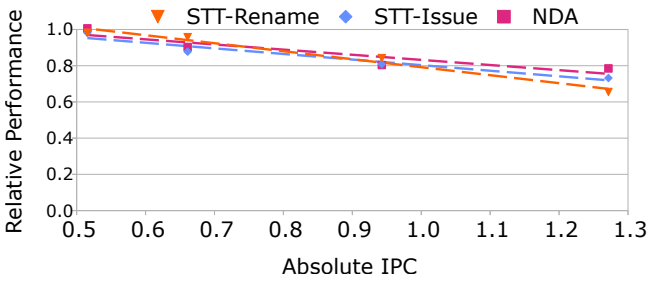


Figure 1: Normalized performance (IPC x Timing) of evaluated secure speculation schemes, with trend line. Data points are placed based on the achieved baseline IPC for the different configurations.

The original description of STT [58] suggests using the register renaming mechanism to track taints, but our microarchitectural design uncovers that taint tracking is fundamentally different from register renaming, as taint tracking and register renaming resolve same-cycle dependencies differently, see Section 4.1. We present a functional microarchitecture, named STT-Rename, to handle this difference, and highlight how rename tainting creates an expensive, unavoidable dependency chain. In addition, we propose an alternative microarchitecture that delays taint tracking until the issue stage, see Section 4.3. This new microarchitecture, named STT-Issue, achieves better results in instructions per cycle (IPC), timing, area, and performance.

For NDA, we show how its limited architectural complexity results in an effective microarchitecture, due to the ease of mapping its design to effective hardware. Though this microarchitecture performs worse in terms of IPC, it achieves superior results in timing, area, and performance when compared to STT due to its much lower design complexity.

Previous research has primarily evaluated secure schemes using architectural simulators, which, though useful for validating a core premise, often obfuscate microarchitectural design and provide limited insight into timing, area, and power. As the field matures and schemes are being considered for adoption by industry, rigorous microarchitectures and analysis of key design characteristics become critical.

We evaluate our microarchitectural designs through RTL implementations on the RISC-V Berkeley Out-of-Order Machine [60]. The IPC evaluation shows that previously estimated IPC reductions between 8.4% and 10.7%, as reported in the literature [2, 29, 55, 58], might be overly optimistic. We instead find that the IPC loss is 18.1% and 15.5% for STT, and 26.4% for NDA, with the relative IPC impact worsening as absolute IPC increases.

When considering performance as the combined effect of IPC and timing, the results are even worse. Our evaluation from synthesized RTL designs shows a performance slowdown of 34.5% and 26.8% for STT-Rename and STT-Issue, respectively, and 21.5% for NDA, when implemented on the highest performance core configuration for the BOOM. This configuration achieves an average absolute IPC of 1.27 on SPEC CPU2017, see Figure 1. This cost is already prohibitively high, but trends indicate that this cost will be even higher for higher-performing cores.

Based on a linear extrapolation of our results, a slowdown of more than 45% is predicted for an Intel Redwood Cove class of core with an IPC of 2.03 for SPEC2017. Growth is unlikely to be linear, but even with a less pessimistic estimate with only halved growth, the performance slowdown is still a staggering 49.5% and 39.8% for STT, and 35.3% for NDA. With this less pessimistic trend, NDA outperforms STT by 1.05-1.25×.

In total, our key contributions are as follows:

- We uncover that STT's rename-based taint computation must be completed in a single cycle, creating an expensive dependency chain, which scales poorly for wider processor cores. The design and performance implications of this are shown and evaluated for the first time.
- We introduce a new microarchitecture for STT, named STT-Issue, which delays tainting to the issue stage, eliminating the expensive dependency chain necessary for STT-Rename, improving IPC, timing, area, and performance results.
- We elucidate the relationship between secure speculation schemes and overall performance, by showing that designs with greater parallelism incur greater relative performance loss, and evaluate this performance impact for two leading in-core state-of-the-art secure speculation schemes.
- We show that NDA, perceived as less competitive than STT due to its lower IPC, performs better for our designs, and might be the better solution, as its simpler design translates into better timing results. We augment this insight with detailed performance, timing, area, and power evaluations of the secure speculation schemes.

ShadowBinding serves as a call to arms for renewed efforts into in-core techniques for secure speculation schemes, showing that current techniques are costly and that new in-core designs are needed to reduce the costs of secure speculation to an acceptable level. Such research needs to precisely consider the microarchitectural design of existing and new architectural schemes and evaluate the total costs for timing, IPC, and area. This work is also a first step in answering recent calls for more detailed evaluations, as the world is facing a sustainability crisis [14, 50], as we present results for increased costs in area and power.

2 Threat Model

Different secure speculation schemes employ different threat models, i.e., assumptions of adversary capabilities and what constitutes a successful attack. To discuss the different schemes, and their evaluation, we give a brief introduction to their threat model.

2.1 Speculative Shadows

To reason about speculative side-channel attacks, it is important to have a robust definition of speculation. We use the concept of speculative shadows introduced by Ghost Loads [45], which shows that speculation can occur as a result of control instructions, store-load forwarding, memory consistency, and exceptions, labeling them as C, D, M, and E-shadows, respectively. Shadows resolve in order and indicate that all following instructions are speculative. Our work focuses on C and D-shadows, as these are the most prominent shadows for speculative side-channel attacks. When an

instruction has no shadows, it is bound-to-commit and has reached the visibility point, as it is called by STT [58].

2.2 Speculative Taint Tracking (STT)

STT defends against an adversary that can monitor all covert channels within the system and can induce speculative execution to access speculative secret data [58]. Practically, this means that any instruction execution that causes an observable, data-dependent effect, is a successful attack if it uses speculatively acquired data.

Secrets are defined as data that the processor would not be able to access during normal execution, i.e., they are the result of transient execution. As such, entering speculation and speculatively leaking data that resided in registers pre-speculation does not constitute a successful attack, as this data is not considered secret. STT blocks all data stemming from speculative loads from being used by transmitting, i.e., observable, data-dependent instructions, whether there is a direct or indirect data dependency.

2.3 Non-speculative Data Access (NDA)

NDA provides several threat models depending on the needs of the user [55]. Their two designs, NDA-Strict and NDA-Permissive, have different goals and defend against different threats. NDA-Strict defends against any leakage of data that would not occur during normal execution. This model is used for protecting against the leakage of transient data, but also data that resides non-speculatively in registers, and is a stricter scheme than STT. NDA-Strict prohibits any propagation of data across the point of speculation, effectively making speculation work as a barrier.

NDA-Permissive defends against the leakage of any data that is speculatively acquired, equivalent to STT. This means that the execution of any instruction with a data dependency, directly or indirectly, on a value acquired from speculative execution, is prevented. Of note, NDA-Permissive does not require an instruction to be observable for it to not receive speculative data. For this work, we focus on NDA-Permissive, and future references to NDA will be implicitly referencing NDA-Permissive as the evaluated scheme.

2.4 Combined Threat Model

Both NDA-Permissive and STT provide equivalent security guarantees for our purposes. For this work, we consider the two threat models to be equivalent and employ this combined threat model to evaluate security. As mentioned, we consider speculation stemming from C- and D-shadows only.

The architectural security offered by NDA and STT has been evaluated in the original works [55, 58]. ShadowBinding uses these models for applying security. We discuss the specific security considerations for ensuring secure microarchitectures in Section 4.5 and Section 5.2 for STT and NDA, respectively.

3 Background

Having established the threat models of the respective secure speculation schemes, we now briefly discuss how STT and NDA each mitigate speculative side-channel attacks.

3.1 Speculative Taint Tracking

As described in the original work [58], STT employs a form of dynamic information-flow tracking (DIFT) [52] to track speculative accesses to memory, and their dependency chains to transmitting instructions. When a speculative load is renamed, the destination register of the load is marked as tainted, with the load set as the taint root. Whenever an instruction depends on one or more tainted registers, two things happen:

- Firstly, the instruction compares all taint sources it depends on, and selects the youngest of them, setting this as its own youngest root of taint (YRoT).
- Secondly, the output register of the instruction is marked with the newly computed YRoT, unless the instruction is itself a load, in which case the register is marked with the load as described earlier.

Transmitting instructions that have a YRoT cannot execute until the YRoT source is no longer speculative. An instruction is defined as a transmitter if its execution has an observable effect on the system that varies depending on the data in its source operand(s), i.e., its execution is observable and data-dependent. Non-transmitting instructions can freely execute, regardless of their YRoT state, but they propagate their YRoT information to their output register as described earlier.

As no observable instructions can execute while they depend on speculative data, STT ensures that no secrets are leaked. Invisible operations, such as common integer operations, can execute normally, even while dependent on potentially secret speculative data. Loads that do not depend on tainted data can execute as normal, preserving some memory-level parallelism for long-latency loads.

3.2 Non-speculative Data Access — Permissive

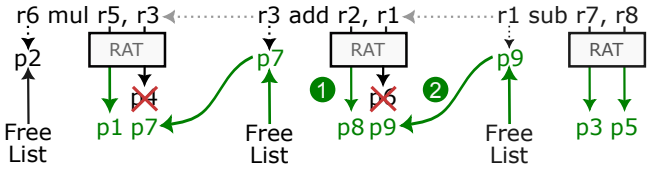
NDA-Permissive takes a direct approach to eliminating the propagation of secret data: Instructions that acquire potentially secret data, i.e., speculative loads, cannot pass this information to any other instructions until the load is non-speculative, and thereby the data is known to not be a secret. To track speculation, NDA proposes tracking the oldest point of speculation through the reorder buffer (ROB) by tracking which instruction types trigger speculation and under which conditions such speculation is resolved.

Whenever the oldest point of speculation is resolved, the new oldest head of speculation is computed, and all load instructions older than the new oldest head of speculation are declared safe to propagate. It is important to note that instructions that do not have a dependency chain on a potential secret are not limited in any way under NDA-Permissive, and can both read source registers, execute, and, unless they are a load, propagate their results as normal. Speculative loads will delay their writeback and broadcast until they are non-speculative.

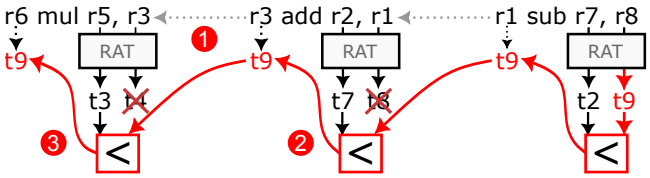
Loads that do not depend on speculative data can initiate their memory access as normal, preserving memory-level parallelism in those cases, and helping hide long-latency loads.

4 STT Design Considerations

In this section, we present two microarchitectures for STT, one that performs taint tracking in the rename stages (STT-Rename in Section 4.1) and one that delays taint tracking until the issue stage



(a) Register renaming showing that there is no dependency between an instruction's destination and source registers.



(b) Taint tracking with the critical path highlighted in red.

Figure 2: Illustration of register renaming and taint tracking for a three instruction-wide rename stage. Younger instructions are to the left.

(STT-Issue in Section 4.3). We also discuss the need for checkpointing when performing taint tracking in the rename stage, as well as some common design considerations and how the two microarchitectures scale with the core's width.

4.1 Taint Tracking During Register Renaming

The original STT work [58] does not present a microarchitectural solution for propagating taints, but describes taint tracking as similar and compatible with register renaming. However, as we will show, taint tracking is fundamentally different from register renaming, and when done during register renaming, it creates an expensive dependency chain that must be performed in a single cycle to ensure correct tainting.

Figure 2a shows how architectural registers are translated to physical registers during the rename stage. The physical destination register for a given instruction is assigned from the free list. This assignment is completely independent of its source registers. Source registers are first derived from the physical register stored in the register alias table (RAT), see ①. In cases where a source register depends on an older instruction being renamed in parallel, then a bypass is enabled that replaces the register from the RAT, see ②.

Taint tracking is fundamentally different. To calculate the youngest root-of-taint (YRoT) of an instruction, the youngest taint must be selected from the source registers of the instruction. This causes problems when a source register depends on an older instruction being tainted in parallel. As shown in Figure 2b, the dependence on `r3` ① requires the YRoT of the `add` instruction to be computed ② before the YRoT of the `mul` instruction can be computed ③. A similar dependency exists between the `add` and `sub` instructions, creating a chain of dependencies that, in the worst case, is as long as the number of renamed instructions.

The longest potential chain increases linearly with the number of instructions that are renamed in parallel. Figure 3 shows the worst-case critical path for a three-wide register-rename stage. This path is independent of the specific core microarchitecture in which taint tracking is to be implemented. For the scheme to function correctly,

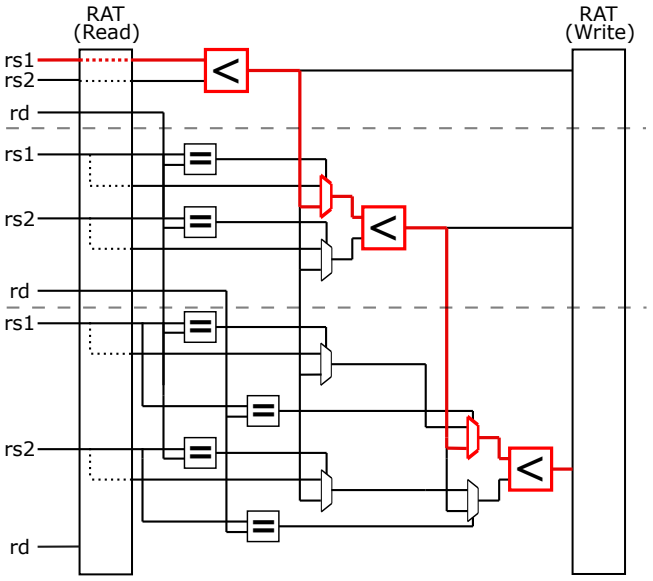


Figure 3: Microarchitecture of YRoT computation for a three instruction-wide rename stage. The critical path is highlighted in red. Dotted lines indicate a translation from index to YRoT. Stippled lines delineate the different instructions.

the existing YRoTs must be read and new YRoTs must be computed and written back in a single cycle to ensure that the next set of instructions receives up-to-date YRoT information. The single-cycle requirement introduces a notable timing limitation for wide cores, which negatively impacts overall performance, see Section 8.2.

4.2 Checkpointing During Register Renaming

Another challenge with implementing STT, as described in the initial work [58], is the overlooked cost for supporting branches. STT, which greatly improved shared understanding of implicit transmitters, clarified that branches are a form of transmitter and should only be resolved when all of their operands are untainted. In certain scenarios, this means that a younger branch might be resolved before an older branch, if the operands for the younger branch become available and untainted sooner than for the older branch. If the branch was incorrectly predicted, it is necessary to restore the architectural state to the point of the younger branch.

This is generally achieved by three mechanisms: A checkpoint of the RAT and free list is stored when a branch is detected in the rename stage; all in-flight instructions younger than the branch are squashed when its mispredict is detected; and the RAT and free list are reset to the stored checkpoint for the mispredicted branch.

As there may still exist live taints in the system, i.e., there may still exist an older source of speculation than the mispredicted branch, the YRoT information must also be restored. This requires that the YRoT information must also be checkpointed whenever a branch is detected in the rename stage.

Additionally, unlike a conventional checkpoint, the YRoT state in a checkpoint may be outdated. Although which YRoT a given instruction depends on will not have changed, the state of the YRoT might have, i.e., the load that is the source of the taint may no

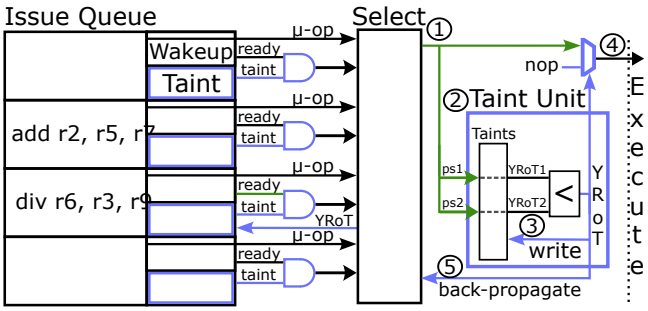


Figure 4: Microarchitecture for STT-Issue. Added structures to support tainting are highlighted in blue. Note that Wakeup and Select are not affected by STT-Issue. Critical path and YRoT depend on a single instruction, unlike for STT-Rename.

longer be speculative. As such, in addition to restoring from a YRoT checkpoint, it is necessary to also invalidate any entries that are no longer valid due to progress in resolving speculation. This can be performed by checking the YRoT of each taint and seeing if it is within the life span for possible YRoT values, i.e., between the youngest load and the youngest non-speculative load.

4.3 Taint Tracking During Instruction Issue

It is possible to delay tainting and YRoT computation [21], as tainting only matters once all source operands are ready and the instruction is to be issued. Such an approach has merit: (1) source operands are more likely to be non-speculative, causing fewer tainted destination registers, (2) the total amount of broadcasted YRoT wakeups is reduced, and (3) there are no same-cycle dependency chains as dependent instructions are not issued together. We show the microarchitecture for such an implementation in Figure 4.

We describe the STT-Issue microarchitecture in generic terms that apply to most scheduling approaches. The STT-Issue tainting mechanism is divided into three main steps:

(1) Initially, the YRoT of an instruction is not known, and as such, the instruction will assume it is safe to issue. The wakeup logic operates as normal and emits a ready signal when all operands have become available.
(2) Once the instruction is selected to issue ①, the YRoT for the instruction is computed by a taint unit ②, which selects the youngest YRoT of all its operands. If any of the operands are tainted, then the entry corresponding to the physical destination register of the instruction is marked as tainted by the calculated YRoT ③. If the instruction is tainted and is a transmitter, then it is barred from executing, and a no-operation (nop) is issued, which wastes an issue slot for this cycle ④.
(3) Tainted transmitters need to be replayed and woken up once they become non-speculative. This is achieved by back-propagating the YRoT to the entry in the issue queue ⑤. A valid YRoT masks the ready signal, preventing the instruction from being selected for issue until the YRoT is broadcast to be non-speculative, see Section 4.4.

High-performance architectures already support speculative issue and replay to improve performance for load-dependent instructions [24]. As such, much of the required logic already exists.

A disadvantage of this approach is the increased cost of using physical registers instead of architectural registers. After the rename stage, all dependencies are tracked through physical registers. In high-performance processors, the number of physical registers is often an order of magnitude higher than the number of architectural registers [30]. This means that storing taint information and using comparators for YRoT computation become notably larger. The total cost of this remains limited, as we show later in Table 4.

Only using physical registers means that there is no need for YRoT checkpoints, see Section 4.2. This is because YRoT values are stored with physical register indexes, which are always live: If a register is no longer assigned after a misprediction, it would need to be reassigned, thereby overwriting its previous (stale) YRoT value before it can be used in a YRoT computation. Similarly, if a register and taint chain is still valid after a misprediction, so too would its values, thereby making YRoT computations correct.

4.4 Shared Design Considerations and Scaling

For both microarchitectures, information about the state of taints must be stored and accessed for tainting. For STT-Rename, this information can be stored together with the RAT, while for STT-Issue, it makes more sense to store it in a separate taint unit.

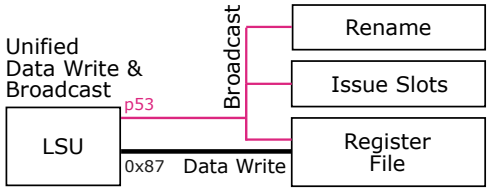
For both STT-Rename and STT-Issue, it is necessary to broadcast whenever a load becomes non-speculative, to inform instructions about the state of their YRoTs. Whenever the YRoT of an instruction becomes non-speculative, it is safe to execute that instruction, even if it is a transmitter. This YRoT broadcast functions as an extension of existing broadcast mechanisms, but occurs whenever a load becomes non-speculative. This broadcast network is expensive, as it requires broadcasting every load that becomes non-speculative to every issue slot, as well as the rename stage or taint unit for STT-Rename and STT-Issue, respectively.

Though they share some microarchitectural features, STT-Issue has several properties that enable it to scale better than STT-Rename. Key among these is the lack of dependencies between issued instructions since dependent instructions cannot be issued in the same cycle, which is not the case for register renaming. The same-cycle dependencies during register renaming create a dependency chain for STT-Rename, which is not present for STT-Issue.

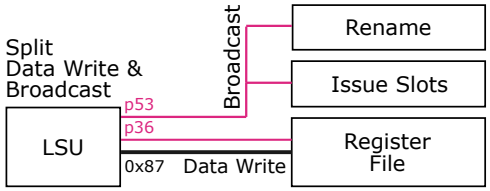
As we highlight in Section 8.2, STT-Issue pays a higher flat cost in terms of timing, achieving worse timing results than STT-Rename for smaller designs, but shows better scaling than STT-Rename due to the lack of same-cycle dependencies. For smaller designs, STT-Rename may offer higher performance, as we show in Section 8.3.

4.5 STT Security Considerations

STT's security premise consists of two points: (1) speculatively acquired data must be marked as tainted, and (2) transmitters must be blocked from executing if they depend on tainted data. Transmitters are defined as all instructions that create a data-dependent visible effect, i.e., there is some observable effect that correlates to the value of the data.



(a) Baseline data write & broadcast



(b) NDA modified data write & broadcast.

Figure 5: Impact of NDA on broadcast and writeback. Note that NDA requires data write be able to select a different physical register than the broadcast.

Correctly identifying all transmitters is complex and requires a detailed understanding of the given microarchitecture. Variable latency execution, control flow changes, and memory dependence handling can all be transmitters, and transmitters can also stem from optimizations such as value predictor training.

Importantly, data leakage occurs only when a data-dependent effect occurs. A tainted transmitter being blocked, i.e., its execution being delayed, reveals only that it depends on tainted data, but does not leak any information about the data. This is why STT-Issue is secure, as even though tainted transmitters might get scheduled for execution, the data is never acted upon.

As tainted data propagate through the microarchitecture under STT, there are large attack surfaces, and any unidentified transmitter risks leaking data. As complexity increases, comprehensively identifying transmitters becomes more difficult. Incorrectly identifying non-transmitters as transmitters is secure, but lowers IPC, as more instructions will be blocked from executing.

5 NDA Design Considerations

In this section, we present the microarchitecture for NDA. NDA's simple design translates well into a realizable microarchitecture, only requiring limited modifications to broadcast and otherwise extending mechanisms that already exist within high-performance processor core designs. NDA's simplicity results in an insignificant impact on timing, as detailed in Section 8.2.

5.1 Delayed Broadcast

NDA requires that the data from a load is not propagated until the load is non-speculative. A mechanism is therefore needed that delays broadcasts until speculation is resolved. Conventionally, when a load request completes, the data is written to the register file, and the register is broadcast as ready. As loads are often on the critical path, such broadcasts happen immediately upon load completion. On load completion, both the broadcast and data writeback refer to the same register, enabling the use of a shared bus, as shown in pink in Figure 5a.

For NDA, the data writeback is decoupled from the broadcast. If the load is speculative at the time of completion, then the data is written to the register file, but the broadcast must be delayed until the load becomes non-speculative. To avoid halving the effective throughput of load operations, the LSU must handle writebacks and broadcasts independently, as shown in Figure 5b. This decoupling limits the possibility of sharing a bus for the physical register index and requires a slightly more complex LSU that can perform a writeback in parallel with a broadcast for two separate loads.

We describe mechanisms for tracking speculation for both STT and NDA in Section 6. Whenever the visibility point increments past a load, a broadcast should be initiated, and several such loads can become non-speculative in the same cycle. However, the number of parallel broadcasts is limited to the core memory width.

For this work, NDA does not support speculative scheduling of dependent instructions by predicting an L1 hit. As we show in Section 8.2, removing this logic for NDA, which is unlikely to be able to benefit from it regardless, improves the timing of NDA.

We note that the delayed broadcast mechanism of NDA hinges on whether an instruction is speculative, and does not in any way depend on the data that the instruction loads. This means that the delayed broadcast mechanism does not introduce any new leakage, as there is no data-dependent behavior for speculative data.

5.2 NDA Security Considerations

NDA's security premise blocks potentially secret data from propagating. We describe a mechanism for loads (Section 5.1), but highlight that all speculatively acquired data must be similarly restricted from being used. NDA's potential security complications are limited, as long as data are restricted at the source, and do not propagate throughout the microarchitecture until non-speculative. This limits the attack surface and simplifies ensuring a secure design.

6 Tracking Speculation

Both NDA and STT require a way of detecting whether a given load instruction is speculative. STT needs to track speculation to know when to taint and untaint, while NDA needs to track speculation to delay the propagation of speculative loads and initiate broadcasts.

For our work, we only focus on speculation stemming from store-to-load forwarding prediction and control speculation, i.e., D and C-shadows [45, 46], as discussed in Section 2.1.

High-performance processors already have methods to track C-shadows, which enable them to quickly squash any in-flight instructions that are mispredicted. Essentially, such a mechanism tracks whether an instruction is dependent on a given branch. Regardless of the specifics of the chosen mechanism, we can use such mechanisms to track whether a load is speculative from a branch, by checking whether it could potentially be squashed by the branch tracking mechanism.

Similarly, because of the risk of address aliasing, load-store units (LSUs) need a method to track D-shadows for memory instructions that might alias. If aliasing is not detected and checked correctly, stale data could be read in the case of store-to-load forwarding. The exact implementation to check for forwarding and forwarding errors will vary based on the microarchitecture, but in all systems,

loads that read stale data must flush all following instructions and restart execution with the correct data.

A simple solution is to check all younger loads for potential address matches whenever a store is committed, and in case of a match, check whether store-to-load forwarding occurred. This check can be performed at the earliest when the address of a store is generated. For a load, when all stores older than the load have performed such a check, it is known whether a store-to-load forwarding error has occurred, and as such, the load is either no longer speculative or marked as having a forwarding error.

C and D-shadows are most prolific in attacks, and using them as the basis for a secure speculation scheme provides defenses against Speculative Store Bypass [20] and Spectre, but does not protect against the full Futuristic model, as defined in InvisiSpec [56]. Protecting against the Futuristic model would require tracking more speculation points, which can be accomplished by adding M and E-shadows to the speculation tracking system. An efficient method for tracking shadows is described by Sakalis et al. [46].

7 Methodology

To evaluate the overall cost of secure speculation schemes, we implement STT and NDA on the RISC-V BOOM core [60], and evaluate STT with rename tainting, STT with issue slot tainting, and NDA under equal conditions. We synthesize STT, NDA, and an unsecure baseline for four different BOOM configurations, using AMD Vitis v2022.2 targeting a U250 Alveo FPGA to evaluate timing, area, and power. Using FireMarshal [41], we create Linux images (Ubuntu 18.04) with the SPEC2017 CPU benchmark suite [11] compiled for RISC-V with GCC 10.1.0. Using Firesim [23], we boot Linux on the synthesized designs and run the full SPEC2017 CPU benchmark suite, as a representative selection of single-threaded workloads. Firesim enables timing-accurate modeling of external I/O through a rate-limiting token scheme, and models external DRAM with a DDR3 interface [8, 23]. Area and power analysis were performed using `report_utilization` and `report_power` through AMD Vitis targeting the BOOM tile, using the Mega core configuration.

We collect results by executing each benchmark from the SPEC2017 suite for 100 billion cycles, examining a large instruction window by taking advantage of the high execution speed of FPGAs compared to simulators, giving us a low margin of error.

Analyzing hardware performance is normally limited to built-in hardware performance counters, but by using TraceDoctor [17] we extract key performance indicators such as committed instructions, latencies, stalls, and their causes. With TraceDoctor, we uncover behaviors such as those discussed in Section 9.2.

We also evaluate NDA and STT-Rename on gem5, using the provided configurations in the original works as a guideline. We run SPEC2017 in full system simulation mode, using SimPoints gathered from the first 100 billion instructions of each benchmark, with up to five SimPoints for each benchmark, a warmup period of 50 million, and an execution period of 50 million instructions. Though we use the provided configuration to the best of our ability, information about the choice of prefetcher and branch predictor is not provided in the original papers. Table 2 shows our configuration for those parameters, which are the same for our NDA and STT evaluation. We were not able to evaluate the following benchmarks from SPEC2017 using gem5: `namd, parest, povray`, so for comparisons between BOOM and gem5, we are not including any results for those benchmarks.

Table 1: The four BOOM configurations, with their key characteristics, and a comparison to the latest Intel processor Redwood Cove, including average absolute IPC for SPEC2017.
[1] **Redwood Cove has a unified Int+FP pipeline.**
[2] **Redwood Cove supports 3 loads + 2 stores in a cycle.**

Parameter	Small	Medium	Large	Mega	Intel
Fetch Width	4	4	8	8	6
Core Width	1	2	3	4	6
Phys Int Regs	52	80	100	128	280
Phys Fp Regs	48	64	96	128	332
ROB Entries	32	64	96	128	512
Issue Slots (Int+FP)	8+8	20+16	32+24	40+32	97[1]
Func Units (Int+FP)	1+1	2+1	3+1	4+2	5[1]
LDQ Entries	8	16	24	32	192
Memory Ports	1	1	1	2	3+2[2]
L1D Size	16KiB	16KiB	32KiB	32KiB	48KiB
L1D MSHRs	2	2	4	8	16
SPEC2017 IPC	0.52	0.66	0.94	1.27	2.03 [31]

Table 2: Additional configuration details for gem5.

Parameter	Value
Branch Predictor	Multiperspective Perceptron TAGE 64KiB
L1D Prefetcher	Stride Prefetcher
L2 Prefetcher	Stride Prefetcher

We evaluate the following schemes:

- **Baseline:** The unsafe baseline is the unmodified BOOM core, which is not protected against Spectre attacks.
- **STT–Rename:** Speculative Taint Tracking, with taint computation and propagation occurring during the register rename stage, see Section 4.1.
- **STT–Issue:** Speculative Taint tracking, with taint computation and propagation occurring during the issue stage, see Section 4.3.
- **NDA:** Non-speculative Data Access, with a split broadcast and data bus, that broadcasts non-speculative loads, and holds speculative data in registers until they become non-speculative, see Section 5.

We evaluate the secure schemes on four BOOM configurations. Table 1 shows their key characteristics and the absolute IPC that the unsafe baseline achieves on SPEC2017. The configurations are created to achieve a balanced design for a given core width, which defines the number of parallel instructions in the decode, rename, dispatch, and commit stages. The full set of configuration parameters for each BOOM configuration can be found on GitHub [59], under `config-mixings`. We also include the characteristics of an Intel Redwood Cove as a representative example of a recent high-performance core, sourced from Chips and Cheese [30]. By default, we present results for the Mega BOOM configuration, unless another configuration is explicitly mentioned, as it has the highest performance. In some sections, we explicitly refer to the configurations from Table 1, and we refer to higher performance configurations with more parallelism as wider.

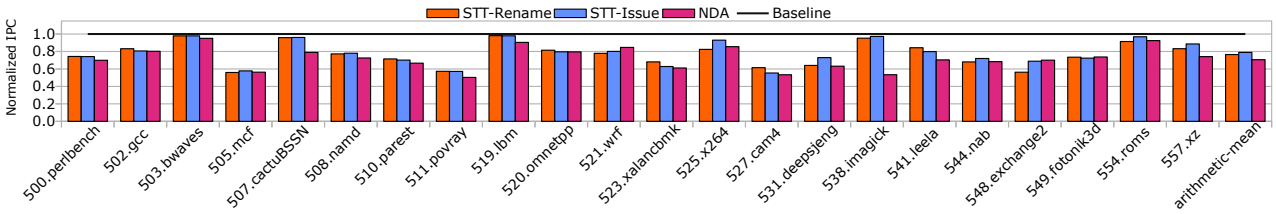


Figure 6: IPC normalized to baseline for different secure speculation schemes for mega size config.

We apply the security analysis described in Section 4.5, identifying data-dependent effects from variable latency, control flow, and similar, and marking such instructions as transmitters. Based on findings in DOLMA [34], we take a restrictive approach to generating memory addresses, delaying these until the data source is untainted. Based on NDA's considerations in Section 5.2, we ensure that speculative data is only stored in registers and that such registers are not broadcast as ready until non-speculative. In addition to these methods, we test against a practical example through the use of the BOOM-attacks [16], verifying that the applied schemes mitigate the Spectre v1 attack.

8 Results

Our results highlight key characteristics of the evaluated secure speculation schemes, such as IPC for four different BOOM configurations, and their impact on timing, area, power, and total runtime. We also show comparable results between the BOOM implementations and equivalent implementations on gem5.

8.1 Overall Instruction Per Cycle (IPC) Loss

Figure 6 shows the IPC for each of the implemented schemes, normalized to the IPC of the unsafe baseline. The results clearly show that the IPC loss of the secure speculation schemes varies greatly depending on the workload. Some workloads, such as `503.bwaves`, have insignificant IPC loss, regardless of the chosen scheme. Other workloads, such as `538.imagick`, show that versions of STT are close to baseline performance, while NDA suffers a massive slowdown, with nearly half the IPC of baseline.

If a benchmark is compute-bound instead of memory-bound, the IPC loss for STT is limited, as most computations are not transmitters, and can execute irrespective of the taint status of their data dependencies. For NDA, the IPC loss can still be significant, as the delayed load broadcast means that no dependent computations can be completed, even if they are invisible to an attacker. Benchmarks such as `507.cactuBSSN` and `538.imagick` highlight this.

`548.exchange2` displays an unexpected behavior, in that NDA achieves a higher IPC than both versions of STT, NDA achieving an absolute IPC of 1.77 compared to STT-Rename's 1.44. Intuitively, one would expect STT to always have equal or better IPC than NDA, due to being able to execute all the same loads and more load-dependent non-transmitter instructions. During the execution of the benchmark, STT-Rename had 1 350 as many store-to-load forwarding errors as NDA, losing many cycles handling these errors. This is due to STT-Rename preventing store addresses from becoming visible in the LSU, despite it being safe to do so. Further details are provided in Section 9.2.

We highlight notable characteristics that are significantly affected by the secure speculation schemes. The average branch latency goes from 20 cycles for the baseline to 34, 38, and 45 cycles for STT-Rename, STT-Issue, and NDA, respectively. Similarly, the average of filled integer issue slots goes from 7.1 to 8.3, 9.5, and 13.1. This indicates that μ-ops remain in the core much longer under secure speculation schemes, especially for NDA, which, unlike STT, is unable to issue μ-ops that depend on speculative data. The branch misprediction rate remains unchanged.

To calculate the average IPC for SPEC2017, we calculate the arithmetic mean of cycles and instructions separately, and calculate the IPC from these averages [12]. The mean shows that STT-Rename, STT-Issue, and NDA achieve, on average, 81.9%, 84.5%, and 73.6% of baseline IPC, respectively, which is a significant IPC impact. The impact is considerably lower for both STT implementations than NDA, and there is a noticeable difference between STT-Rename and STT-Issue. Potentially surprising, STT-Issue generally outperforms STT-Rename, due to being able to take advantage of more aggressive scheduling and avoiding limitations with partial issues. We explore this in more depth in Section 9.1 and Section 9.2.

We also evaluate the IPC loss for each of the four BOOM configurations to analyze how the secure speculation schemes scale with core width. Figure 7 shows the normalized IPC for each of the schemes, with Figure 8 showing a linear trend for the average IPC. We include data for all benchmarks to show that the average normalized IPC gets worse with wider cores, and this trend is consistent across all benchmarks, except for benchmarks that are not meaningfully affected, such as `bwaves` and `roms`. NDA scales worse in general, which is expected as NDA delays more instructions.

For all schemes, as the core width increases, the absolute IPC improves (Table 1), and the impact on the IPC by the secure schemes gets worse, meaning that more relative IPC is lost the higher the overall IPC of the core.

8.2 Timing Impact

In this section, we discuss the impact the secure speculation schemes have on timing after synthesis, i.e., the highest achievable frequency for a given design with and without a secure speculation scheme integrated. This provides insight into the complexity and performance impact of a given secure speculation scheme.

Figure 9 shows the highest achieved frequency for each secure speculation scheme for the four BOOM configurations. The results indicate that the timing impact of STT increases for wider core designs, while NDA consistently achieves the same, or even higher, frequency as the unsafe baseline. This is expected as NDA does not introduce much additional logic, and due to not supporting

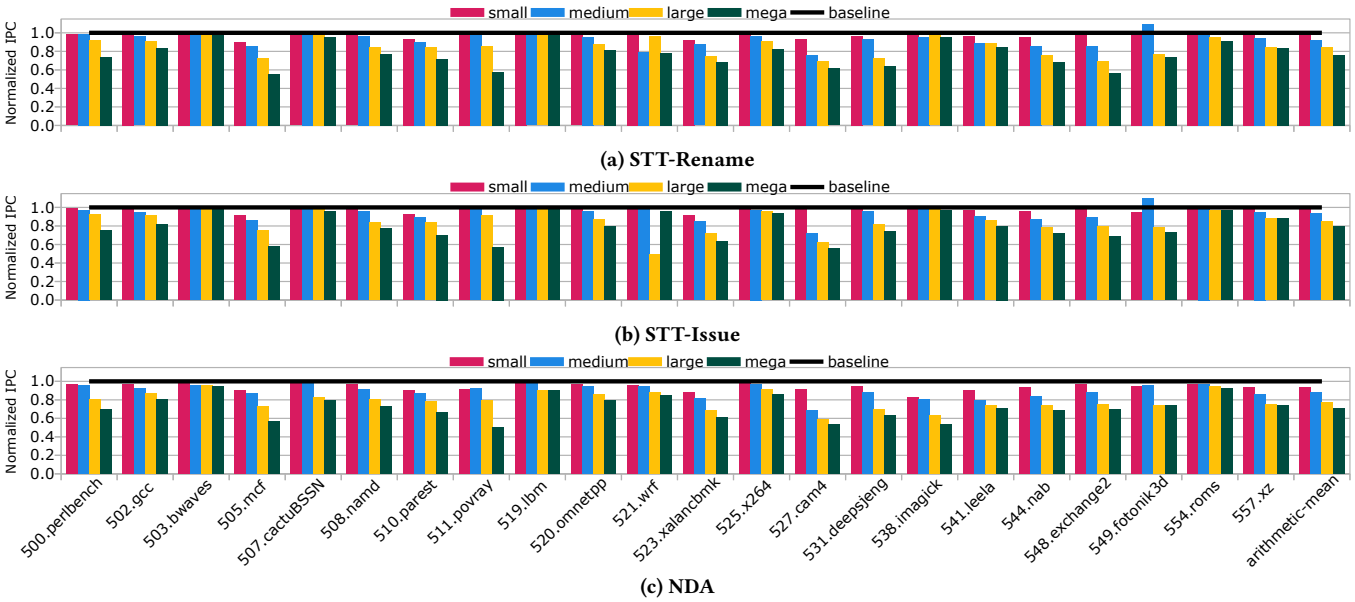


Figure 7: Normalized IPC for the four core configurations for (a) STT-Rename, (b) STT-Issue, and (c) NDA.

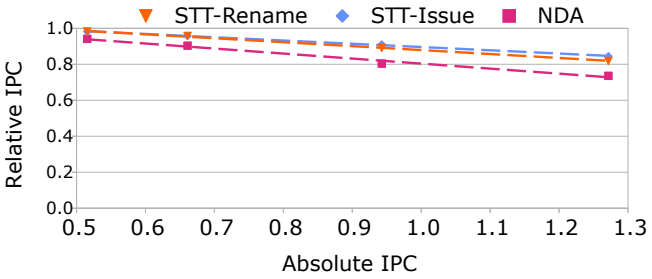


Figure 8: Relative IPC across wider designs, normalized to baseline, with trend line.

speculative L1 hit broadcasts, it simplifies the core design, achieving better timing for some configurations.

We analyze the critical path for each evaluated microarchitecture for the Mega BOOM. For STT-Rename, we observe that the critical path goes through the new taint tracking mechanism shown in Figure 3. For STT-Issue, the new critical path goes through the modified issue logic detailed in Figure 4. NDA shows a critical path through the register file, which imposes fewer timing limitations than the baseline's critical path from the L1 cache to the core.

STT-Rename and STT-Issue both significantly limit timing for the wider configurations. As explained in Section 4.1, STT-Rename scales poorly due to the YRoT computation relying on a chain of dependencies that must be resolved in a single cycle (Figure 3), resulting in greater timing impact the more instructions are renamed in parallel. For the four-wide Mega BOOM core, STT-Rename achieves only 80% of the baseline frequency. STT-Issue does not have such a dependency chain, as its YRoT computations are independent. As tainting is performed at a timing-sensitive stage of the core, there is still a notable cost in timing.

Figure 10 shows the trend in relative timing compared to the baseline. The relative timing of NDA does not change with core width,

Table 3: Tabled data of Figure 1, normalized performance for configurations.

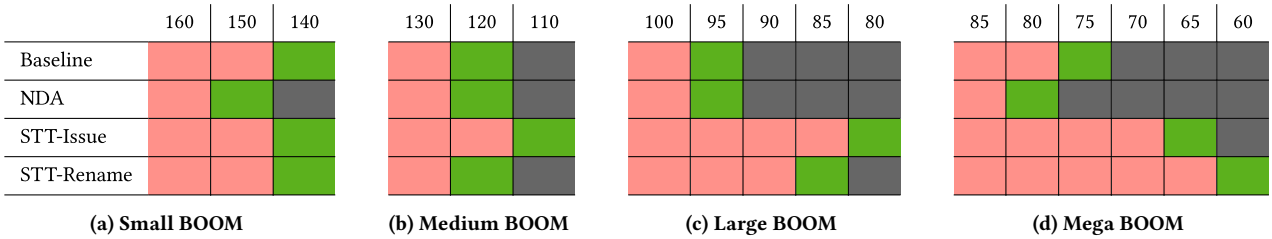
Scheme	Small	Medium	Large	Mega
STT-Rename	0.98	0.96	0.84	0.66
STT-Issue	0.98	0.88	0.81	0.73
NDA	1.01	0.90	0.80	0.78

while STT-Issue sees a notable impact for the Medium configuration, but with only slight increases for wider designs. STT-Rename, on the other hand, has a small timing impact for smaller configurations, but the impact grows for wider cores. These trends might not hold for all wider designs, as design constraints vary greatly, but generally, wider configurations and designs should experience greater timing limitations for STT-Rename and STT-Issue, with NDA likely to achieve similar timings as the baseline.

8.3 Performance = IPC × Timing

Performance is the combination of IPC and timing, and Table 3 shows the performance for each secure scheme, normalized against the baseline. The trends across core widths are shown in Figure 1, which highlights that the wider a core gets, the higher impact the secure speculation schemes have on overall performance. This is due to their increasing IPC impact (for all schemes) and timing limitations (for STT), as the core width increases.

STT has been viewed by the research community as outperforming NDA due to reporting a lower reduction in IPC. However, we show that for our RTL-based design, NDA scales better than STT due to its simpler design, which incurs lower penalties in terms of timing for wider cores. NDA has an overall higher performance for the Mega BOOM configuration, with trends indicating a growing gap. This gap is notable for STT-Rename, but less significant for STT-Issue, due to STT-Issue having better timing and IPC results



(a) Small BOOM (b) Medium BOOM (c) Large BOOM (d) Mega BOOM

Figure 9: Achieved timings (in MHz) during synthesis for four different BOOM configurations, see Table 1. Green: successfully met timing, red: did not meet timing, and gray: better result available.

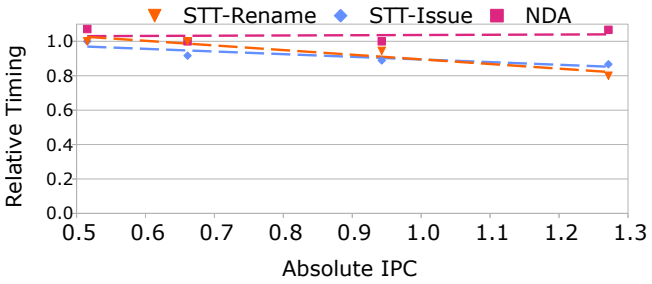


Figure 10: Best timing results for different core configurations, normalized to baseline, with trend line.

Table 4: Area (lookup tables (LUTs) and flip-flops (FFs)), and power results normalized to the baseline for all schemes synthesized at 50 MHz.

Scheme	LUTs	FFs	Power
STT-Rename	1.060	1.094	1.008
STT-Issue	1.059	1.039	1.026
NDA	0.980	1.027	0.936

than STT-Rename. Even if NDA could achieve only baseline timing, it would still have higher performance than both versions of STT for the Mega BOOM (0.74). This challenges previous analyses that only use IPC. We delve further into this in Section 9.4.

8.4 Area and Power

Table 4 shows the area and power increase for each of the secure speculation schemes when synthesized at 50 MHz. While both STT-Rename and STT-Issue have a similar increase in lookup tables (LUTs), STT-Rename has a considerably higher increase in flip-flops (FFs), due to its need for checkpoints, see Section 4.2. Both of these increases for STT are significant, while NDA sees a reduction in LUTs, due to simpler logic, but has a slight increase in FFs.

The power remains unchanged for STT-Rename, increasing slightly for STT-Issue, and decreasing notably for NDA. This means that in terms of sustainability [14], NDA has a significant edge in all categories compared to STT-Rename and STT-Issue.

8.5 gem5

For completeness's sake, we also implement NDA and STT-Rename on gem5, and run the SPEC2017 suite, comparing benchmarks and overall results with the most comparable BOOM implementations.

Table 5: IPC for the Medium, Large, and Mega BOOM, and gem5, for STT-Rename, STT-Issue, and NDA.
[3] using STT gem5 configuration [58]. [4] using NDA gem5 configuration [55].

Configuration	Baseline Abs. IPC	STT-Rename IPC Loss	STT-Issue IPC Loss	NDA IPC Loss
BOOM Medium	0.59	4.5%	4.8%	9.0%
BOOM Large	0.83	11.3%	10.0%	18.6%
BOOM Mega	1.09	17.6%	15.8%	22.4%
gem5	1.12[3]/0.79[4]	17.2%[3]	N/A	13.0%[4]

As seen in Table 5, NDA achieves a relatively low absolute IPC using the given configuration, with an IPC between the Medium and Large configurations. The overall IPC impact is lower than expected on the gem5 evaluation than the BOOM evaluation, compared to what is expected for the given baseline absolute IPC.

STT achieves a relatively high baseline absolute IPC with its configuration, similar to the absolute IPC achieved by the Mega configuration. The IPC impact is nearly identical for the BOOM Mega and the gem5 implementation, for STT-Rename.

We discuss the implications of these results in Section 9.4.

9 Discussion

In this section, we discuss the feasibility implications for secure schemes, performance phenomenons, the scaling design costs of the schemes, and why our results differ from earlier evaluations, emphasizing our contributions contra those using gem5.

9.1 STT-Rename vs. STT-Issue

For most of the SPEC2017 suite, STT-Issue outperforms STT-Rename. This might seem counterintuitive, as STT-Issue has the potential of issuing instructions that will be killed in-flight (Section 4.3), potentially wasting resources, something that STT-Rename avoids by tainting instructions and determining their YRoT during rename. However, the difference comes down to at what point an instruction gets scheduled to be issued and when the YRoT gets calculated.

An instruction is conventionally scheduled for issue as soon as all its operands are ready. Using STT-Issue, an instruction has not yet determined if it is tainted, and will be scheduled as normal. Using STT-Rename, on the other hand, tainting has already occurred, and an instruction might therefore be blocked due to being tainted. Thus, a potentially tainted instruction can be selected to issue under STT-Issue, and if its YRoT is declared safe in the following cycle, the instruction will execute and complete, as the YRoT will be safe

at the time of determining taints. This enables STT-Issue to, in certain cases, issue instructions one cycle earlier than STT-Rename. In benchmarks for which STT-Issue outperforms STT-Rename, this and the advantages described in Section 9.2 are the main reasons.

Overall, as STT-Issue outperforms STT-Rename in all categories for our evaluation, STT-Issue is generally preferred instead of STT-Rename, unless there are other factors to consider, such as the issue stage being more expensive to modify.

9.2 Partial Issues and Store-to-Load Forwarding

Store instructions consist of an address and a data element. Depending on the specifics of the implementation of a given ISA, a store instruction might be split into an address and a data micro-op, might be a unified micro-op for both that can partially issue whenever one operand is ready, or might be a single micro-op for both that can only issue whenever both operands are ready. Issuing the address generation early is beneficial, as it makes the address of a store available in the store queue, and thereby enables store-to-load forwarding checks to proceed earlier, lowering the chances of store-to-load forwarding errors.

In the BOOM core, stores are a single micro-op that can partially issue whenever either the data or address element is ready, with the other half issuing whenever its other operand becomes ready. However, with STT, this now becomes more complicated. When the YRoT is computed, it uses both operands to find the YRoT, meaning that partial issues for address calculation might be blocked, not because the address operand is not ready, but because the instruction is tainted, and its YRoT is not safe.

In the case of `exchange2`, a sudoku solver [37], which uses a lot of memory operations that span very small memory spaces, this results in the STT implementations having a considerably higher amount of store-to-load forwarding errors, because the store addresses are delayed. STT-Issue suffers less from this problem, as it does not calculate its YRoT for the entire store if only part of its operands are ready, and as such, can issue partial, untainted address generation in most cases. An optimization could enable STT-Rename to work the same way, by having two taints, one for each operand in the specific case of stores, or by fully splitting the store operation into two micro-ops, one for address and one for data. This type of minute detail can have significant performance and design implications for certain workloads.

9.3 Microarchitectural Complexity

An important consideration for selecting the right secure speculation scheme for an architecture is the design cost such a scheme will create. Design cost can be measured through physical results, such as effects on timing and area overhead. We document such measurements in Figure 9 and Table 4, which indicate that STT-Rename incurs higher costs than STT-Issue.

In addition to direct physical design costs, we can evaluate microarchitectural complexity. As mentioned earlier, NDA incurs limited changes, only modifying aspects of broadcast for loads specifically. This is in contrast to STT, which modifies every instruction that passes through rename/issue, to ensure that its DIFT conventions are followed. Similarly, every instruction that can be delayed by a taint, i.e., all transmitters, needs a broadcast mechanism for when its YRoT is safe. Regardless of the specifics of the underlying core, such modifications will be more complex, more invasive, demand a higher amount of area, and have a graver impact on timing compared to NDA. However, if timing costs are not incurred, STT-Issue might outperform NDA due to higher IPC.

9.4 Comparing to gem5

Previously reported performance results for gem5 vary greatly between different implementations and evaluations [2, 4, 29, 34, 55, 58]. Many works use SPEC CPU2006, which is outdated, and results in different IPC trends than with the updated 2017 version. Even accounting for this, there is a notable gap between previously reported results and ours. The reasons for this are multifold:

Firstly, gem5 allows users to freely modify key core parameters, such as memory latency. Changing these parameters can obscure key details or lead to incorrect conclusions if the chosen configuration is not representative. For example, earlier works have evaluated STT with a single cycle latency for the L1 data cache, which is 3–4 cycles faster than the latest Intel processors [51]. If memory is a notable limitation of a given secure speculation scheme, incorrect memory behavior might give misleading results.

Secondly, simulators such as gem5 are slower than FPGAs. This makes evaluating long benchmark suites such as SPEC2017 difficult, as running the entire benchmark is not feasible. Many evaluations run only small portions of a benchmark, which might not capture all or even a representative selection of benchmark phases, giving inaccurate results Ideally, SimPoints based on all or a large part of the benchmark should be used to provide more reliable numbers.

Finally, due to first implementing the schemes using RTL, we were able to create an equivalent microarchitecture in gem5, which achieves a similar IPC to our BOOM implementation. Hardware prototyping provides better guidance for microarchitecture, as the given constraints emulate a real design, and it is therefore harder to make infeasible configurations and design choices.

Previously, STT [58] reported a performance overhead of 8.5%, while NDA [55] reported a performance overhead of 22.3%. The reported results for NDA correlate well with our BOOM Mega configuration results, but our results for STT are more pessimistic, showing equivalent performance impact to the smaller BOOM Large configuration. We emphasize the need for hardware prototyping as a tool for indicative results and design analysis when performing such evaluations, as relying purely on architectural simulators can give an imprecise understanding of performance.

Critically, conclusions such as STT outperforming NDA due to lower IPC reduction are incorrect when other factors, such as timing, are considered, which cannot be adequately assessed by current architectural simulators.

9.5 Trends for Wider Designs

A key contribution of this work is the investigation into the performance cost of secure speculation schemes by using hardware prototypes and determining the scalability of the evaluated schemes. The presented trends on IPC (Figure 8), timing (Figure 10), and performance (Figure 1) indicate that the greater absolute IPC the baseline processor achieves, the greater the relative loss incurred by the secure speculation schemes.

Our results for IPC are replicated across our evaluation platforms, indicating a consistent phenomenon with an impact of 15%-26% IPC loss for our widest core. Based on these trends, as seen in Figure 8, though imperfect as designs are not continuous, the total IPC loss might be upward of 20% or greater, for leading-edge processors.

Exact timing limitations will vary depending on the specific microarchitecture of a given core and the target process technology, but our results and trends indicate notable impacts, as seen in Figure 10. The results show that the critical path goes through the taint mechanism for STT (Section 8.2) and timing gets noticeably worse for STT-Rename as the core width increases (Figure 9).

Many aspects of the evaluated secure speculation schemes seemingly scale poorly with wider core designs, an issue potentially exacerbated even further for leading-edge processors, which commonly boast a core width of six or more instructions. Even at a halved linear estimate using trends from IPC and timing for a Redwood Cove-class processor, the total performance loss would be 49%, 40%, and 35% for STT-Rename, STT-Issue, and NDA, respectively.

From our analysis, NDA shows better overall performance than both versions of STT, and considerably better than STT-Rename, despite lower IPC, due to a much lower impact on timing. Similarly, NDA has the lowest impact on area overhead of the three schemes and is the only scheme with reduced power consumption compared to the baseline. When evaluating sustainability, NDA is the preferred choice, regardless of whether the embodied or operational equivalent greenhouse gas footprint is dominating [13, 14].

STT-Rename shows the worst results in all categories (except IPC compared to NDA), clearly showing that even if STT's higher IPC is desired, the originally proposed microarchitecture is not optimal. This work highlights the need for future work to find more efficient in-core solutions to speculative side-channel attacks.

9.6 Limitations and Future Work

The BOOM is less optimized than leading commercial cores. For example, the BOOM has a naïve memory-retry policy and simple issue queues. While the BOOM offers good performance, the greater complexity required for achieving the performance of leading-edge cores might affect results for those designs. Future work evaluating the IPC and timing impact on high-performance cores is encouraged to further explore the nuances of these takeaways.

As discussed in Section 9.5, our IPC results replicate across different evaluation platforms and indicate notable impacts. Through our timing analysis, we have shown how STT can impact timing, and how alternative applications of the STT strategy, such as STT-Issue, make it possible to limit this impact. While our results highlight real limitations, there are notable timing differences for leading-edge technologies compared to FPGA synthesis, and exact results are likely to differ for those designs. Similarly, area and power also indicate potential impact scope and direction, but the exact results are specific to our methodology.

Other insights, such as the difference between taint tracking and register renaming, the cascading dependency chain in STT-Rename, and how to employ STT later in the pipeline, are not contingent on the specifics of the methodology.

10 Related Work

There have been limited investigations into the practical cost of secure speculation schemes using hardware designs. Other work has previously implemented STT on the BOOM core, but none have, to our knowledge, focused specifically on performance evaluations.

Tobias Jauch et al., in their Secure-by-Construction work [21], implement STT as part of their framework for verifying secure properties. Their work focused on finding examples of leakage and can analyze all microarchitectural information leakage. Their implementation of STT is based on being comprehensive, and generic, tainting post register-read instead of post-issue, and communicating execution status broadly, which incurs extra area cost.

Other mitigation strategies have employed modifications outside the core to mitigate Spectre, such as InvisiSpec [56], MuonTrap [4], GhostMinion [3], and SDO [57]. As we focus on less complex in-core techniques, these fall outside the scope of this work, but such strategies show promising performance results, though their current evaluation is limited to simulator-based approaches. Modifications to the cache hierarchy are generally more invasive than in-core schemes, though they may be necessary if the performance penalties found with in-core schemes are unavoidable.

11 Conclusion

In this work, we have presented several novel insights into the microarchitectural designs and their implications for state-of-the-art secure speculation schemes. We have uncovered how STT-Rename's tainting necessitates a chain of dependencies, which must be resolved in a single cycle to keep the taint information correct. We present an effective microarchitecture for STT-Rename, which addresses the original limitations of tainting during the rename stage, as well as an alternative design, STT-Issue, which eliminates the chain of dependencies by delaying tainting until the issue stage, achieving better performance, timing, area, and power results. We have shown how NDA's limited complexity easily translates into an effective microarchitecture, with only limited design impact.

We have provided rigorous microarchitectures and corresponding design characteristics for both NDA and STT. Importantly, our results challenge established performance evaluation for secure schemes, highlighting that the real cost of Spectre mitigation might be much higher than previously estimated. As the field matures, such evaluations become critical for scheme evaluation and industry adoption. We are, to the best of our knowledge, the first work to present timing, area, and power analysis for secure speculation schemes based on the synthesis of microarchitectural RTL designs.

In total, our evaluation puts a spotlight on the importance of detailed evaluation for secure speculation schemes, and invites future research into better solutions for in-core schemes, as current designs incur higher overheads than previously reported. Particularly, this motivates further investigation into optimization strategies such as InvarSpec [61], Doppelganger Loads [29], and ReCon [2], to help improve the limited IPC of underlying schemes.

References

[1] Ayush Agarwal, Sioli O'Connell, Jason Kim, Shaked Yehezkel, Daniel Genkin, Eyal Ronen, and Yuval Yarom. 2022. Spook.js: Attacking Chrome Strict Site Isolation via Speculative Execution. In *Proceedings of the IEEE Symposium on Security and Privacy*. 699–715. https://doi.org/10.1109/SP46214.2022.9833711

[2] Pavlos Aimoniotis, Amund Bergland Kvalsvik, Xiaoyue Chen, Magnus Själander, and Stefanos Kaxiras. 2023. ReCon: Efficient Detection, Management, and Use of Non-Speculative Information Leakage. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*. 828–842. https://doi.org/10.1145/3613424.3623770

[3] Sam Ainsworth. 2021. GhostMinion: A Strictness-Ordered Cache System for Spectre Mitigation. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*. 592–606. https://doi.org/10.1145/3466752.3480074

[4] Sam Ainsworth and Timothy M. Jones. 2020. MuonTrap: Preventing Cross-Domain Spectre-Like Attacks by Capturing Speculative State. In *Proceedings of the International Symposium on Computer Architecture*. 132–144. https://doi.org/10.1109/ISCA45697.2020.00022

[5] Kristin Barber, Anys Bacha, Li Zhou, Yinqian Zhang, and Radu Teodorescu. 2019. SpecShield: Shielding Speculative Data from Microarchitectural Covert Channels. In *Proceedings of the International Conference on Parallel Architectural and Compilation Techniques*. 151–164. https://doi.org/10.1109/PACT.2019.00020

[6] Mohammad Behnia, Prateek Sahu, Riccardo Paccagnella, Jiyong Yu, Zirui Neil Zhao, Xiang Zou, Thomas Unterluggauer, Josep Torrellas, Carlos Rozas, Adam Morrison, Frank Mckeen, Fangfei Liu, Ron Gabor, Christopher W. Fletcher, Abhishek Basak, and Alaa Alameldeen. 2021. Speculative interference attacks: breaking invisible speculation schemes. In *Proceedings of the Architectural Support for Programming Languages and Operating Systems*. 1046–1060. https://doi.org/10.1145/3445814.3446708

[7] Atri Bhattacharyya, Alexandra Sandulescu, Matthias Neugschwandtner, Alessandro Sorniotti, Babak Falsafi, Mathias Payer, and Anil Kurmus. 2019. SMoTherSpectre: Exploiting Speculative Execution through Port Contention. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. 785–800. https://doi.org/10.1145/3319535.3363194

[8] David Biancolin, Sagar Karandikar, Donggyu Kim, Jack Koenig, Andrew Waterman, Jonathan Bachrach, and Krste Asanovic. 2019. FASED: FPGA-Accelerated Simulation and Evaluation of DRAM. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 330–339. https://doi.org/10.1145/3289602.3293894

[9] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H. Lai. 2019. SgxPectre: Stealing Intel Secrets from SGX Enclaves Via Speculative Execution. In *Proceedings of the IEEE European Symposium on Security and Privacy*. 142–157. https://doi.org/10.1109/EuroSP.2019.00020

[10] Rutvik Choudhary, Jiyong Yu, Christopher Fletcher, and Adam Morrison. 2021. Speculative Privacy Tracking (SPT): Leaking Information From Speculative Execution Without Compromising Privacy. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*. 607–622. https://doi.org/10.1145/3466752.3480068

[11] Standard Performance Evaluation Corporation. 2017. SPEC CPU2017 Benchmark Suite. http://www.specbench.org/cpu2017/

[12] Lieven Eeckhout. 2010. *Computer Architecture Performance Evaluation Methods*. Morgan & Claypool Publishers. https://doi.org/10.2200/S00273ED1V01Y201006CAC010

[13] Lieven Eeckhout. 2022. A First-Order Model to Assess Computer Architecture Sustainability. *IEEE Computer Architecture Letters* 21 (July 2022), 137–140. Issue 2. https://doi.org/10.1109/LCA.2022.3217366

[14] Lieven Eeckhout. 2024. FOCAL: A First-Order Carbon Model to Assess Processor Sustainability. In *Proceedings of the Architectural Support for Programming Languages and Operating Systems*. 401–415. https://doi.org/10.1145/3620665.3640415

[15] Jacob Fustos, Michael Bechtel, and Heechul Yun. 2020. SpectreRewind: Leaking Secrets to Past Instructions. In *Proceedings of the ACM Workshop on Attacks and Solutions in Hardware Security*. 117–126. https://doi.org/10.1145/3411504.3421216

[16] Abraham Gonzalez, Ed Younis, Ben Korpan, and Jerry Zhao. 2019. BOOM Speculative Attacks. https://github.com/riscv-boom/boom-attacks

[17] Björn Gottschall, Silvio Campelo de Santana, and Magnus Jahre. 2023. Balancing Accuracy and Evaluation Overhead in Simulation Point Selection. In *Proceedings of the IEEE International Symposium on Workload Characterization*. 43–53. https://doi.org/10.1109/IISWC59245.2023.00019

[18] Intel. 2018. Retpoline: A Branch Target Injection Mitigation. https://www.intel.com/content/dam/develop/external/us/en/documents/retpoline-a-branch-target-injection-mitigation.pdf

[19] Intel. 2018. Speculative Execution Side Channel Mitigations. https://www.intel.com/content/dam/develop/external/us/en/documents/336996-speculative-execution-side-channel-mitigations.pdf

[20] Intel. 2018. Speculative Store Bypass / CVE-2018-3639 / INTEL-SA-00115. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/advisory-guidance/speculative-store-bypass.html

[21] Tobias Jauch, Alex Wezel, Mohammad R. Fadiheh, Philipp Schmitz, Sayak Ray, Jason M. Fung, Christopher W. Fletcher, Dominik Stoffel, and Wolfgang Kunz. 2023. Secure-by-Construction Design Methodology for CPUs: Implementing Secure Speculation on the RTL. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. 1–9. https://doi.org/10.1109/ICCAD57390.2023.10323843

[22] Hai Jin, Zhuo He, and Weizhong Qiang. 2022. SpecTerminator: Blocking Speculative Side Channels Based on Instruction Classes on RISC-V. *ACM Transactions on Architecture and Code Optimization* 20 (Nov. 2022), 3566053. Issue 1. https://doi.org/10.1145/3566053

[23] Sagar Karandikar, Howard Mao, Donggyu Kim, David Biancolin, Alon Amid, Dayeol Lee, Nathan Pemberton, Emmanuel Amaro, Colin Schmidt, Aditya Chopra, Qijing Huang, Kyle Kovacs, Borivoje Nikolic, Randy Katz, Jonathan Bachrach, and Krste Asanovic. 2018. FireSim: FPGA-Accelerated Cycle-Exact Scale-Out System Simulation in the Public Cloud. In *Proceedings of the International Symposium on Computer Architecture*. 29–42. https://doi.org/10.1109/ISCA.2018.00014

[24] I. Kim and M.H. Lipasti. 2004. Understanding scheduling replay schemes. In *Proceedings of the International Symposium High-Performance Computer Architecture*. 198–209. https://doi.org/10.1109/HPCA.2004.10011

[25] Vladimir Kiriansky, Ilia Lebedev, Saman Amarasinghe, Srinivas Devadas, and Joel Emer. 2018. DAWG: A Defense Against Cache Timing Attacks in Speculative Execution Processors. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*. 974–987. https://doi.org/10.1109/MICRO.2018.00083

[26] Vladimir Kiriansky and Carl Waldspurger. 2018. Speculative Buffer Overflows: Attacks and Defenses. arXiv:1807.03757 [cs] http://arxiv.org/abs/1807.03757

[27] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2019. Spectre Attacks: Exploiting Speculative Execution. In *Proceedings of the IEEE Symposium on Security and Privacy*. 1–19. https://doi.org/10.1109/SP.2019.00002

[28] Esmaeil Mohammadian Koruyeh, Shirin Haji Amin Shirazi, Khaled N. Khasawneh, Chengyu Song, and Nael Abu-Ghazaleh. 2020. SpecCFI: Mitigating Spectre Attacks using CFI Informed Speculation. In *Proceedings of the IEEE Symposium on Security and Privacy*. 39–53. https://doi.org/10.1109/SP40000.2020.00033

[29] Amund Bergland Kvalsvik, Pavlos Aimoniotis, Stefanos Kaxiras, and Magnus Själander. 2023. Doppelganger Loads: A Safe, Complexity-Effective Optimization for Secure Speculation Schemes. In *Proceedings of the International Symposium on Computer Architecture*. 1–13. https://doi.org/10.1145/3579371.3589088

[30] Chester Lam. 2024. Intel's Redwood Cove: Baby Steps are Still Steps. https://chipsandcheese.com/p/intels-redwood-cove-baby-steps-are-still-steps

[31] Chester Lam. 2024. Running SPEC CPU2017 at Chips and Cheese? https://chipsandcheese.com/p/running-spec-cpu2017-at-chips-and-cheese

[32] Mengming Li, Chenlu Miao, Yilong Yang, and Kai Bu. 2022. unXpec: Breaking Undo-based Safe Speculation. In *Proceedings of the International Symposium High-Performance Computer Architecture*. 98–112. https://doi.org/10.1109/HPCA53966.2022.00016

[33] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. 2018. Meltdown: Reading Kernel Memory from User Space. In *Proceedings of the USENIX Security Symposium*. 973–990. https://www.usenix.org/conference/usenixsecurity18/presentation/lipp

[34] Kevin Loughlin, Ian Neal, Jiacheng Ma, Elisa Tsai, Ofir Weisse, Satish Narayanasamy, and Baris Kasikci. 2021. DOLMA: Securing Speculation with the Principle of Transient Non-Observability. In *Proceedings of the USENIX Security Symposium*. 1397–1414. https://www.usenix.org/conference/usenixsecurity21/presentation/loughlin

[35] Giorgi Maisuradze and Christian Rossow. 2018. ret2spec: Speculative Execution Using Return Stack Buffers. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. 2109–2122. https://doi.org/10.1145/3243734.3243761

[36] Ross Mcilroy, Jaroslav Sevcik, Tobias Tebbi, Ben L. Titzer, and Toon Verwaest. 2019. Spectre is here to stay: An analysis of side-channels and speculative execution. arXiv:1902.05178 [cs] http://arxiv.org/abs/1902.05178

[37] Michael Metcalf. 2020. 548.exchange2_r. https://www.spec.org/cpu2017/Docs/benchmarks/548.exchange2_r.html

[38] Shravan Narayan, Craig Disselkoen, Daniel Moghimi, Sunjay Cauligi, Evan Johnson, Zhao Gang, Anjo Vahldiek-Oberwagner, Ravi Sahita, Hovav Shacham, Dean Tullsen, and Deian Stefan. 2021. Swivel: Hardening WebAssembly against Spectre. In *Proceedings of the USENIX Security Symposium*. 1433–1450. https://www.usenix.org/conference/usenixsecurity21/presentation/narayan

[39] Andrew Pardoe. 2018. Spectre mitigations in MSVC. https://devblogs.microsoft.com/cppblog/spectre-mitigations-in-msvc/

[40] Arash Pashrashid, Ali Hajiabadi, and Trevor E. Carlson. 2023. HidFix: Efficient Mitigation of Cache-Based Spectre Attacks Through Hidden Rollbacks. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. 1–9. https://doi.org/10.1109/ICCAD57390.2023.10323979

[41] Nathan Pemberton and Alon Amid. 2021. FireMarshal: Making HW/SW Co-Design Reproducible and Reliable. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software*. 299–309. https://doi.org/10.1109/ISPASS51385.2021.00052

[42] Joseph Ravichandran, Weon Taek Na, Jay Lang, and Mengjia Yan. 2022. PACMAN: attacking ARM pointer authentication with speculative execution. In *Proceedings of the International Symposium on Computer Architecture*. 685–698. https://doi.org/10.1145/3470496.3527429

[43] Xida Ren, Logan Moody, Mohammadkazem Taram, Matthew Jordan, Dean M Tullsen, and Ashish Venkat. 2021. I See Dead μops: Leaking Secrets via Intel/AMD Micro-Op Caches. In *Proceedings of the International Symposium on Computer Architecture*. 14. https://doi.org/10.1109/ISCA52012.2021.00036

[44] Gururaj Saileshwar and Moinuddin K. Qureshi. 2019. CleanupSpec: An "Undo" Approach to Safe Speculation. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*. 73–86. https://doi.org/10.1145/3352460.3358314

[45] Christos Sakalis, Mehdi Alipour, Alberto Ros, Alexandra Jimborean, Stefanos Kaxiras, and Magnus Själander. 2019. Ghost loads: What is the cost of invisible speculation?. In *Proceedings of the ACM International Conference on Computing Frontiers*. 153–163. https://doi.org/10.1145/3310273.3321558

[46] Christos Sakalis, Stefanos Kaxiras, Alberto Ros, Alexandra Jimborean, and Magnus Själander. 2019. Efficient Invisible Speculative Execution through Selective Delay and Value Prediction. In *Proceedings of the International Symposium on Computer Architecture*. 723–735. https://doi.org/10.1145/3307650.3322216

[47] Christos Sakalis, Stefanos Kaxiras, Alberto Ros, Alexandra Jimborean, and Magnus Själander. 2020. Understanding Selective Delay as a Method for Efficient Secure Speculative Execution. *IEEE Trans. Comput.* 69 (Nov. 2020), 1584–1595. Issue 11. https://doi.org/10.1109/TC.2020.3014456

[48] Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, and Daniel Gruss. 2019. ZombieLoad: Cross-Privilege-Boundary Data Sampling. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. 753–768. https://doi.org/10.1145/3319535.3354252

[49] Michael Schwarz, Martin Schwarzl, Moritz Lipp, Jon Masters, and Daniel Gruss. 2019. NetSpectre: Read Arbitrary Memory over Network. In *Proceedings of the European Symposium on Research in Computer Security*. 279–299. https://doi.org/10.1007/978-3-030-29959-0_14

[50] Saeideh Sheikhpour, David Metz, Erling Jellum, Magnus Själander, and Lieven Eeckhout. 2024. Sustainable High-Performance Instruction Selection for Superscalar Processors. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. 89:1–89:9. https://doi.org/10.1145/3676536.3676757

[51] Sudhanshu Shukla, Sumeet Bandishte, Jayesh Gaur, and Sreenivas Subramoney. 2022. Register file prefetching. In *Proceedings of the International Symposium on Computer Architecture*. 410–423. https://doi.org/10.1145/3470496.3527398

[52] G. Edward Suh, Jae W. Lee, David Zhang, and Srinivas Devadas. 2004. Secure program execution via dynamic information flow tracking. In *Proceedings of the Architectural Support for Programming Languages and Operating Systems*. 85–96. https://doi.org/10.1145/1024393.1024404

[53] Youssef Tobah, Andrew Kwong, Ingab Kang, Daniel Genkin, and Kang G. Shin. 2022. SpecHammer: Combining Spectre and Rowhammer for New Speculative Attacks. In *Proceedings of the IEEE Symposium on Security and Privacy*. 681–698. https://doi.org/10.1109/SP46214.2022.9833802

[54] Kim-Anh Tran, Christos Sakalis, Magnus Själander, Alberto Ros, Stefanos Kaxiras, and Alexandra Jimborean. 2020. Clearing the Shadows: Recovering Lost Performance for Invisible Speculative Execution through HW/SW Co-Design. In *Proceedings of the International Conference on Parallel Architectural and Compilation Techniques*. 241–254. https://doi.org/10.1145/3410463.3414640

[55] Ofir Weisse, Ian Neal, Kevin Loughlin, Thomas F. Wenisch, and Baris Kasikci. 2019. NDA: Preventing Speculative Execution Attacks at Their Source. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*. 572–586. https://doi.org/10.1145/3352460.3358306

[56] Mengjia Yan, Jiho Choi, Dimitrios Skarlatos, Adam Morrison, Christopher Fletcher, and Josep Torrellas. 2018. InvisiSpec: Making Speculative Execution Invisible in the Cache Hierarchy. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*. 428–441. https://doi.org/10.1109/MICRO.2018.00042

[57] Jiyong Yu, Namrata Mantri, Josep Torrellas, Adam Morrison, and Christopher W. Fletcher. 2020. Speculative Data-Oblivious Execution: Mobilizing Safe Prediction For Safe and Efficient Speculative Execution. In *Proceedings of the International Symposium on Computer Architecture*. 707–720. https://doi.org/10.1109/ISCA45697.2020.00064

[58] Jiyong Yu, Mengjia Yan, Artem Khyzha, Adam Morrison, Josep Torrellas, and Christopher W. Fletcher. 2019. Speculative Taint Tracking (STT): A Comprehensive Protection for Speculatively Accessed Data. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*. 954–968. https://doi.org/10.1145/3352460.3358274

[59] Jerry Zhao. 2024. The Berkeley Out-of-Order RISC-V Processor. https://github.com/riscv-boom/riscv-boom

[60] Jerry Zhao, Ben Korpan, Abraham Gonzalez, and Krste Asanovic. 2020. SonicBOOM: The 3rd Generation Berkeley Out-of-Order Machine. In *Proceedings of the Workshop on Computer Architecture Research with RISC-V*. 7.

[61] Zirui Neil Zhao, Houxiang Ji, Mengjia Yan, Jiyong Yu, Christopher W. Fletcher, Adam Morrison, Darko Marinov, and Josep Torrellas. 2020. Speculation Invariance (InvarSpec): Faster Safe Execution Through Program Analysis. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*. 1138–1152. https://doi.org/10.1109/MICRO50266.2020.00094