Speculative Cache Hierarchy For Mitigating Transient Side-Channel Attacks

Aman Dhammani
18D180002
IIT Bombay
amandhammani@iitb.ac.in

Devansh Jain
190100044
IIT Bombay
devanshdvj@cse.iitb.ac.in

Sameer Ahmad
22m0789
IIT Bombay
sameerahmad@cse.iitb.ac.in

***Abstract*—Spectre class of attacks break isolation between security domains by observing the effect of transient instructions on micro-architecture. Caches are the most preferred vector for performing spectre attacks. To hide the effect of transient instructions on normal caches, a separate speculative cache can be used. To achieve the goal of mitigating said attacks with minimal performance overhead, we implemented speculative cache with strictness ordering in all levels of the cache hierarchy. The average performance overhead observed is 1% for SPEC17 benchmark and ClientServer Workload benchmark.**

1. Introduction

Context isolation is an important security feature that ensures that private information of one context cannot be read by another context. Out-of-order execution in modern processors executes transient instructions without knowing the result of the dependent condition. These instructions when retired, can leave permanent state change in the cache hierarchy, which can be recorded by an attacker. This breaks the isolation boundary between the two contexts. In the project, we implement a special class of cache called Speculative Cache, which store data for speculative instructions until they are committed, so that any permanent state in normal caches is prevented if the instruction is squashed.

2. Motivation

Speculative cache has been implemented in GhostMinion [1] for which the author claimed little performance overhead of 2.5% for SPEC CPU2006 benchmark. In GhostMinion, the author used a single speculative cache alongside the normal L1 cache. The main criticism for the observation is that he used ROB with 64 entries for the evaluation model, which is quite small compared to ROBs implemented in modern processors. With larger ROBs, the number of in-flight speculative instructions will increase, and hence the performance overhead of GhostMinion may also significantly increase.

Our goal is to implement a secure speculative cache in all levels of the cache hierarchy to minimize the performance overhead for processors with larger ROBs. We also implemented Speculative Caches for TLBs to mitigate Spectre attacks targeting page translation.

3. Contribution

In this paper, the authors contribute the following:

- A novel cache hierarchy having speculative and non-speculative caches to prevent transient channel-based cache attacks on modern Out-Of-Order processors.
- Extension of the idea of strictness ordering principle [1] to ensure cache security with a small performance overhead of 1%

4. Idea - 10K feet view

The idea is to extend the principle of strictness-ordering implemented in [1] for all layers of the cache hierarchy. In order to do so, we add a speculative cache at each level of the cache hierarchy, i.e., from Translation Lookaside Buffers (TLB) to last-level caches (LLC). The speculative cache blocks are strictness ordered and hence, do not allow vulnerabilities based on transient execution, including backward-in-time attacks.

The block diagram for the modified cache at each level is shown in Fig 2. Each level consists of two cache blocks - one corresponding to the normal cache hierarchy and the other corresponding to the speculative hierarchy. The flow of data consists of three flows:

4.1. Access request from the processor (blue)

When the processor sends a request for data read/writeback, the index is scanned through in parallel in each of the two cache blocks, thereby keeping the access latency low. If there is a miss in both caches at a particular level, the request is forwarded down to the next cache level.

4.2. Data flow (red)

In the event of a cache hit in the normal hierarchy, the data is sent back to the upper levels of the cache/processor. Otherwise, we check for a hit in the speculative hierarchy,

which involves an additional step of verifying that the timestamp of the block accessed is less than the timestamp of the instruction, thereby maintaining temporal ordering amongst the instructions. If that fails, the access is deemed to be a miss and handled appropriately.

During the upward movement of data (from memory to processor), a small arbitration module is provided to identify the appropriate cache block to be filled based on the instruction's nature (speculative, non-speculative). For data brought in by a non-speculative instruction, a standard replacement policy like LRU is employed in the cache.

On the other hand, if the data is brought in by a speculative instruction, then it can replace the data in the speculative cache only if the timestamp of the instruction is less than the timestamp of the instruction that brought in the block to be evicted. If this is not satisfied, then the data is discarded and loaded into the cache when a slot eventually becomes free. It should be noted that there will not be any deadlock because the earlier instruction will eventually get committed, thereby freeing up the resource for the newer instruction.

4.3. Retire Requests (green)

To reduce the load on the limited speculative resources and prevent timing attacks, it is imperative that they are freed as soon as the instruction is retired. To ensure this, the processor sends the information about the commits and squashes it to the cache through the commit queue. In the case of an instruction commit, the data can be transferred to the normal hierarchy and hence, is forwarded in the same cycle. The entry is invalidated in the speculative cache to free up the cache line.

In the event of the instruction being squashed, the system is expected to roll back to a state consistent with non-speculative execution. This is done by simply invalidating the cache lines that have been brought in by the transient instruction, thereby mitigating transient channel attacks.

5. Results

5.1. Benchmarks

The following benchmarks are used for evaluation:

- SPEC17 benchmark
- ClientServer Workload benchmark

5.2. Simulation Parameters

For each load, we keep the size of the normal hierarchy the same as the baseline and vary the overhead incurred by adding the speculative cache blocks over the normal hierarchy. The parameters describing the system are listed in Table 1.

All the simulations are done on Champsim, cycle-accurate trace-based microarchitecture simulator [2]. The number of warm-up and simulation instructions were both set to 1 million.

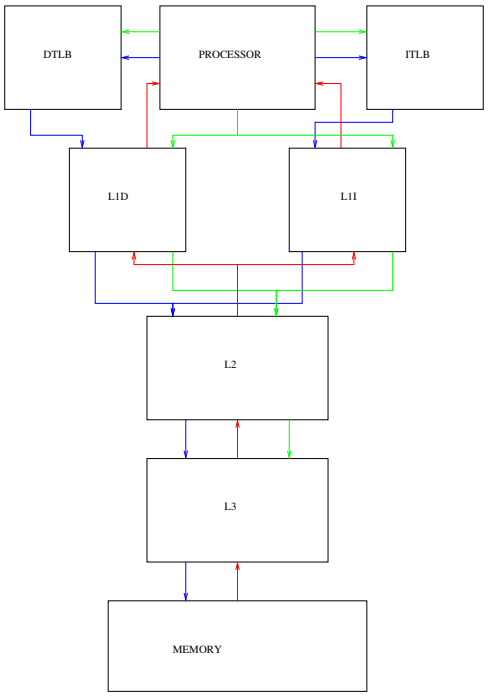

Figure 1. Structure Of The Cache Hierarchy, with three data levels and one TLB level. The individual

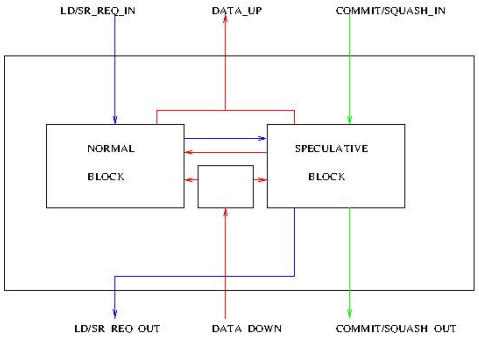

Figure 2. Block Diagram For A Single Cache Level. Blue arrows represent the path for load-store requests, red arrows represent the path for data movement, and green arrows represent the path for retire requests

TABLE 1. SIMULATION PARAMETERS

Parameter	Value
Number Of CPUs	1
ROB Size	352
L1I Size	32KB
L1D Size	48KB
L2C Size	512KB
LLC Size	2MB
ITLB Size	4KB
DTLB Size	4KB
STLB Size	96KB

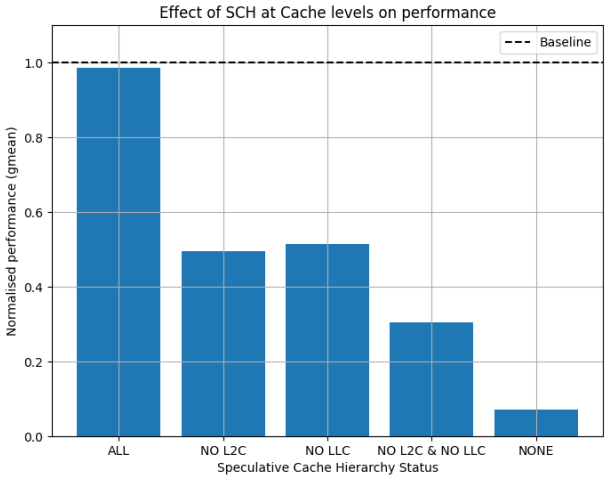

Figure 3. Bar graph showing the gmean of normalized performance with respect to baseline for various architectures of speculative cache hierarchy (Data source - [3])

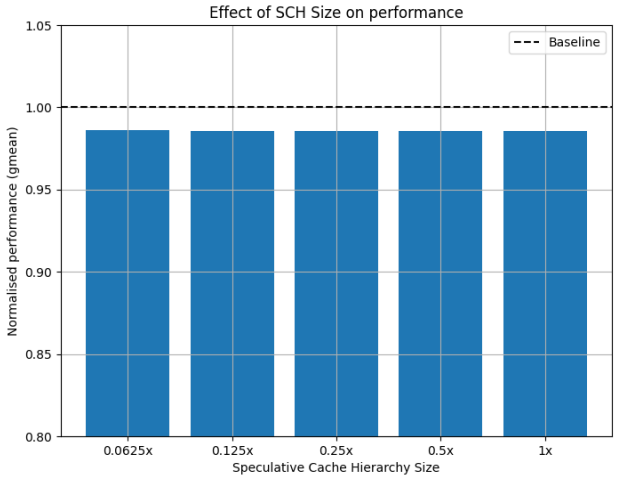

Figure 4. Plot showing the sensitivity of performance towards the size of the speculative caches. The size of the normal hierarchy remains constant (Data source - [3])

5.3. Evaluation

We perform two tests to evaluate the performance of the proposed hierarchy:

5.3.1. Varying the architecture. In the first evaluation, we implement following architectures for the cache hierarchy:

- ALL: This represents the case where the speculative cache is enabled across all levels. This ensures robust security against all attacks, at the expense of higher area
- NO LLC: In this case, speculative caches are implemented at all levels except LLC
- No L2C: The speculative cache is implemented at all levels except L2C. In this case, a load corresponding to a speculative instruction will bypass the L2C, and directly be passed to L1.
- NO L2C-LLC: No speculative hierarchy is implemented beyond L1
- NONE: In this case, none of the caches or TLBs have speculative region. Any speculative load will be directly sent to the processor. We still retain security with no power overhead at the expense of the performance

5.3.2. Varying the size. In this case, the hierarchy was fixed to 'ALL', and the size of the speculative cache as a fraction of the normal hierarchy was varied from $\frac{1}{16}$ to 1 in steps of 2.

5.4. Observations And Inferences

5.4.1. Varing the architecture. Fig 3 shows the geometric mean of the normalized performance for various cache hierarchies. We see that the hierarchy ALL performs best, with a small performance overhead of 1%. This is due to the ability to exploit the cache size at all levels, thereby maintaining performance. The small performance overhead is also linked with the use of a different replacement policy and the fact that some hits are not served to maintain strictness ordering.

Additionally, using "NO LLC" performs better than "NO L2C". This is because skipping L2C leads to speculative misses at the L2 level, translating an L1 miss to an L2 miss, thereby degrading the performance.

When we use none of the speculative caches, security is ensured by no commit to the cache until the instruction is committed. This destroys the locality of the cache by not having any data stored into the caches, thereby drastically degrading the performance.

5.4.2. Varying the size. Fig 4 shows the bar graph for sensitivity analysis towards different overheads. We observe that changing the size of the Speculative Cache has no visible effect on performance. The number of ways for the speculative hierarchy is the same as that of the normal hierarchy. Since the instruction window is small (roughly 400 instructions), all the sets of the speculative hierarchy are not required, and hence, we get no performance drop by reducing the size of the speculative hierarchy.

Also, since the number of ways was kept constant, reducing the size doesn't increase the latency. Thus, we demonstrate that the performance is not affected by using a very small overhead for the speculative cache. Furthermore, this analysis reduced the number of sets in TLBs to 1, thereby giving a fully associative cache, making a further reduction of sets impossible while keeping the ratio of normal cache to speculative cache constant. This can be taken as future work.

6. Conclusion

- Implementing a transient-cache-based hierarchy at all cache levels (including TLBs) can give us security against side-channel attacks with a negligible performance overhead and small area overheads (less than 7% of cache area)
- Most of the data is shifted to a non-speculative cache, keeping speculative resources free for speculative execution, thereby providing robustness to transient attacks.

Division Of Work

Member	Work Done
Aman Dhammani	Implemented Speculative Cache Read Implemented Free-Slotting and TimeGuarding
Devansh Jain	Developed Testing & Verification Framework Debugging & Fixing Bugs, Data Analysis
Sameer Ahmad	Implemented Commit Information Flow Ported Speculative Instruction Modelling

Acknowledgments

The authors would like to thank Prof. Biswabandan Panda, Dept. of CSE, IIT Bombay, for providing them with the opportunity to work on the project for the requirements of the course CS773 - Computer Architecture For Performance And Security. They are also grateful to Sumon Nath, M.S. at CSE, IIT Bombay for providing his invaluable assistance in initiating the project.

References

[1] S. Ainsworth, "Ghostminion: A strictness-ordered cache system for spectre mitigation," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 592–606. [Online]. Available: https://doi.org/10.1145/3466752.3480074

[2] "Champsim." [Online]. Available: https://github.com/ChampSim/ChampSim

[3] "Data source." [Online]. Available: https://docs.google.com/spreadsheets/d/1FTAI6n2S5PncKy_ky9HtW7kDRofKTGrvTM61a7VmJno/edit?usp=sharing