Ghost Loads: Side-Effect Free Speculative Memory Accesses

Christos Sakalis
Uppsala University
Uppsala, Sweden
christos.sakalis@it.uu.se

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

Modern, high-performance processors rely on speculative execution to fully utilize available hardware resources. While speculative execution is designed such that side-effects do not affect the correct execution of the applications running on the system, speculation is not completely side-effect free. Specifically, not only are the microarchitectural state of the processor and the memory system modified during speculative execution, but their state is also not reverted to their original state upon a mis-speculation. This can lead to a number of security problems that can be broadly described as speculative side-channel attacks, such as the infamous Spectre, Meltdown, and their variants [1, 2].

The purpose of this work is to hide all observable side-effects of speculation in the memory system. We do not focus on any specific known attacks, not only because there are specialized solutions for most of them, but more importantly, because further attacks are bound to appear in the future. Instead, we aim to eliminate the exploitation of speculation side-effects as a side-channel for attacks, without suffering a debilitating performance loss.

To achieve this, we propose a mechanism called "Ghost Loads", which are loads that do not update, to the degree possible, any of the state of the memory system. We only focus on loads since stores are kept hidden by default and any remaining types of memory accesses are rare and do not cause any significant performance loss if executed non-speculatively.

Ghost Loads have the following characteristics:

- They can hit on any level of the memory hierarchy including private caches, shared caches, and main memory, returning the data directly to the core.
- The replacement state of the caches is not updated, nor are the data installed in any of the caches in the hierarchy.
- Separate MSHRs are used for Ghost and regular accesses.
- The prefetcher is augmented to differentiate between Ghost and regular accesses, having any prefetches trained through Ghost Loads issued as ghost accesses.
- Ghost memory accesses do not leave any trace in the DRAM. If they find the row open, they leave the same row open. If on the other hand the row is closed, it will be opened and then closed again.
- Accesses to the TLB are still work in progress, but similar rules as for the data caches apply.

Given that the majority of the loads in the evaluated benchmarks are executed speculatively, issuing them simply as Ghost Loads would lead to immense performance degradation. For this reason, we introduce two mechanisms to regain the lost performance. The first mechanism, called Materialization (Mtz), is triggered when a load is no longer speculative. An Mtz packet is sent to the memory hierarchy, triggering all the side-effects that the memory request would have triggered, such as installing the data in the cache and updating the replacement information. The second mechanism, called the Ghost Buffer (GhB), is a small, read-only cache that is visible only to Ghost Loads. Each cache in the memory hierarchy has its own Ghost Buffer. When the cache receives the data for a Ghost access, it places the data in the Ghost Buffer instead of the cache itself. Other Ghost accesses can read from the Ghost Buffer, but not regular accesses. When an Mtz packet is received, the data are installed in the cache, making them available for all accesses. Figure 2 contains some example Ghost accesses and how they are handled.

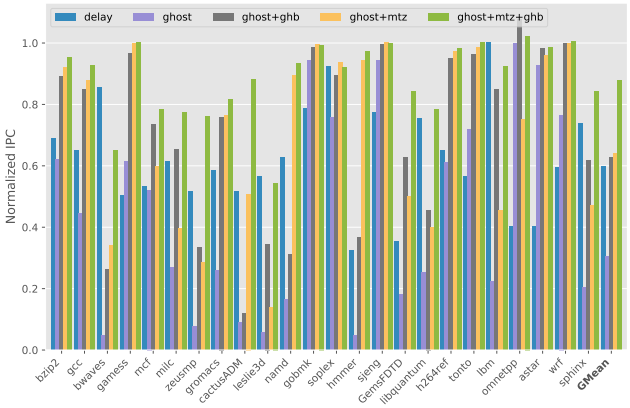
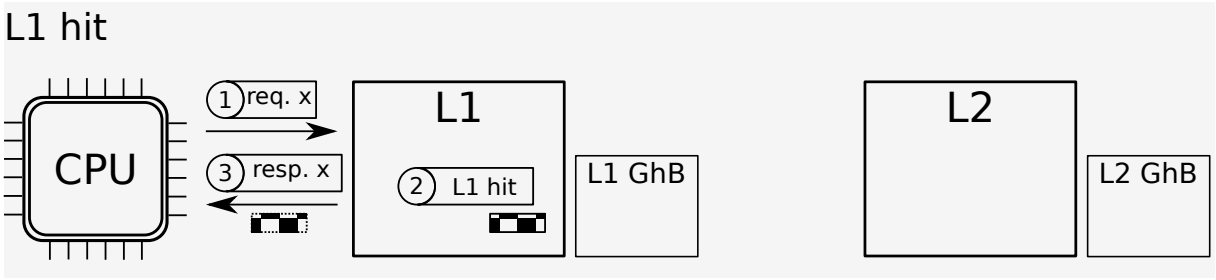


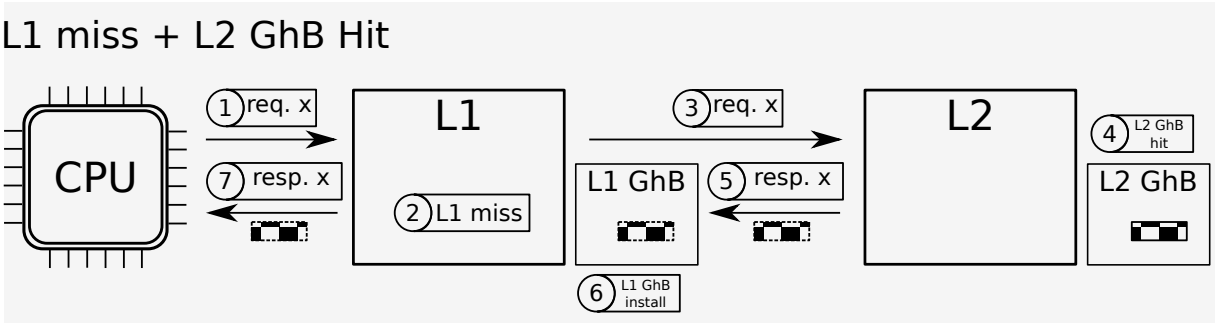
Figure 1: Normalized IPC for the different versions.

Specifically, for this work, we evaluate the following alternatives, as seen in Figure 1:

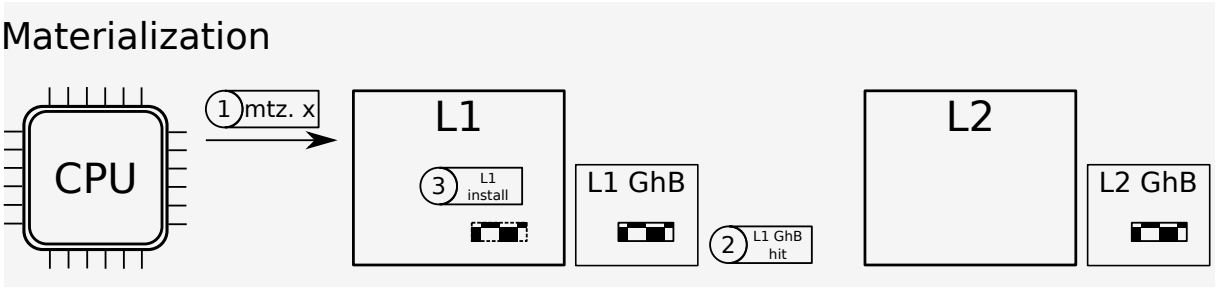
- Non-speculative Loads (Delay): Loads are stalled until they can be executed non-speculatively. This is used to measure the cost of disabling speculation for loads, which, as expected, is impractically high.
- Ghost Loads: Speculative loads are executed as Ghosts. No measures are taken to improve the performance of the Ghost Loads.
- Ghost Loads & Ghost Buffer: Each cache is augmented with a small cache that is only accessible by ghosts.
- Ghost Loads & Mtz: As soon as a Ghost Load is no longer speculative, an Mtz packet is sent to the memory system, triggering the side-effects of the memory access.
- Ghost Loads & Ghost Buffer & Mtz: Mtz packets check if the data exists in the Ghost Buffer of the cache, in order to avoid traversing the rest of the memory hierarchy.



(a) L1 hit: The Ghost access (1) hits in the L1 (2) and the data are returned to the CPU (3) without making any changes to the cache (e.g. LRU).



(b) L1 miss & L2 GhB hit: The Ghost access (1) misses in the L1 cache and Ghost Buffer (2) and the request is forwarded to the L2 (3). The data are found in the L2 GhB (4) and are sent to the L1 (5). Since the access is a Ghost access, the data are not installed in the L1 cache but instead in the L1 GhB (6). The data are then returned to the CPU (7).



(c) Materialization: An Mtz. packet is sent to the L1 (1). The data are found in the L1 GhB (2) so they are installed in the L1 cache (3). The Mtz. packet is dropped.

Figure 2: Example Ghost and Materialization requests. The data are indicated by the box with the checkered pattern. A solid outline indicates that the data already existed before the request, while a dashed outline indicates that the data were installed because of the request.

By combining the Ghost Loads with the Mtz and the Ghost Buffer mechanisms, we can achieve our goal of effectively hiding the side-effects of speculative execution, thus ensuring security at the cost of only 12% performance degradation. We are still investigating where the remainder of the performance is lost and how it can be regained, with the end goal of achieving security without any performance penalty.

ACKNOWLEDGMENTS

I would like to thank my co-supervisors Alexandra Jimborean and Stefanos Kaxiras. I would also like to thank Mehdi Alipour and Alberto Ros for their involvement in this project.

REFERENCES

[1] Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2018. Spectre Attacks: Exploiting Speculative Execution. *ArXiv e-prints* (Jan. 2018). arXiv:1801.01203

[2] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. 2018. Meltdown. *ArXiv e-prints* (Jan. 2018). arXiv:1801.01207