Doppelganger Loads

A Safe, Complexity-Effective Optimization for Secure Speculation Schemes

Amund Bergland Kvalsvik, Pavlos Aimoniotis[†], Stefanos Kaxiras[†], and Magnus Själander

Presentation by: Elias Nodland and Oskar Sveinsen

Norwegian University of Science and Technology, Trondheim, Norway
[†]Uppsala University, Uppsala, Sweden

09.Nov.2023

INTRODUCTION

Abstract

"Speculative side-channel attacks have forced computer architects to rethink speculative execution. Effectively preventing microarchitectural state from leaking sensitive information will be a key requirement in future processor design."
— Kvalsvik et al. [1]

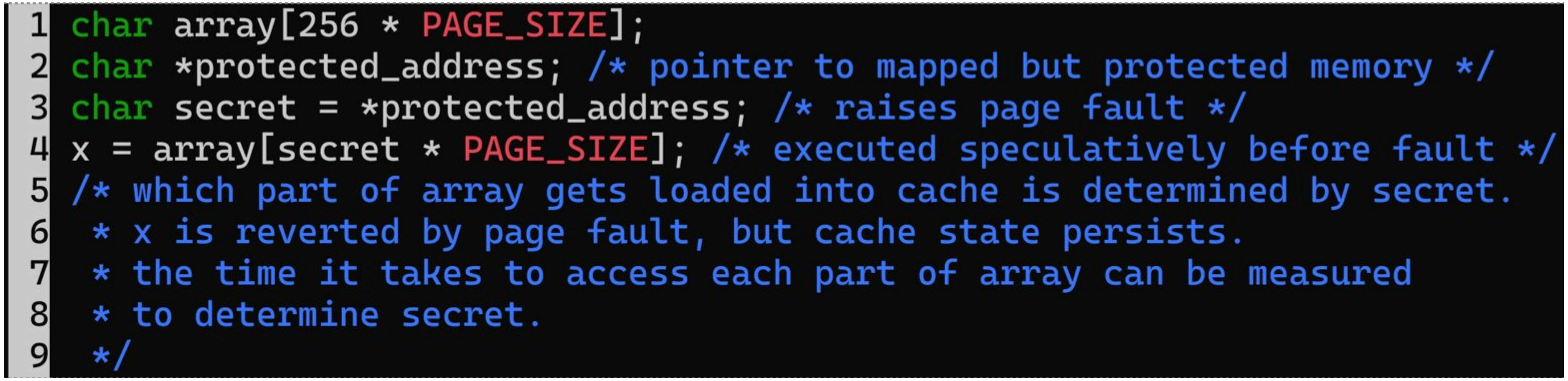
BACKGROUND AND MOTIVATION

Speculative side-channel attacks

- Access protected data in speculative context
- Use protected data to modify micro-architectural state
- Architectural state is reverted, micro-architectural state persists
- Measure micro-architectural state to receive data
- Cache is micro-architectural state
- Measure time to access
- Chosen-code vs control-steering attack

Meltdown

- Chosen-code attack
- Bypass hardware memory protection
  - e.g. access kernel memory
- Memory must already be mapped



```c
char array[256 * PAGE_SIZE];
char *protected_address; /* pointer to mapped but protected memory */
char secret = *protected_address; /* raises page fault */
x = array[secret * PAGE_SIZE]; /* executed speculatively before fault */
/* which part of array gets loaded into cache is determined by secret.
 * x is reverted by page fault, but cache state persists.
 * the time it takes to access each part of array can be measured
 * to determine secret.
 */
```

Spectre

- Control-steering attack
- Bypass software protections
- Branch predictor poisoning
- Same transmitter/receiver as Meltdown
- Originally 2 variants
  - Boundary check bypass (buffer over-read)
  - Indirect branch redirect (return-oriented programming)
- Many variants possible, different side-channels
- Violate JavaScript sandbox, VMs etc.

Software mitigations

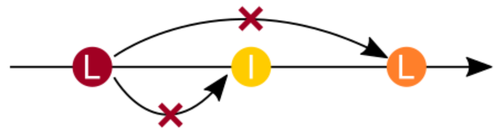
- Compiler modifications
- Large overhead
- Recompile all software
- Early solutions proved insufficient (e.g. retbleed)
- Only solutions for specific attacks

Secure Speculation Schemes 1/2

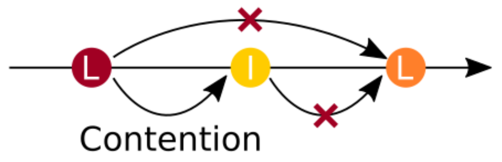
- L Speculative Load
- I General Dependent Instruction
- L Transmitting Load

Secure Speculation Schemes 1/2

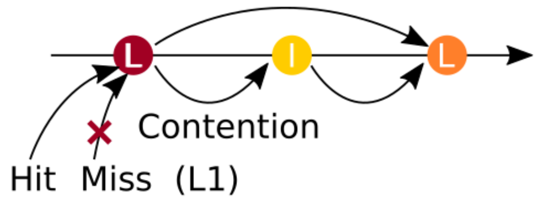
NDA: Non-Speculative Data Access
(-P = Permissive)



STT: Speculative Taint Tracking

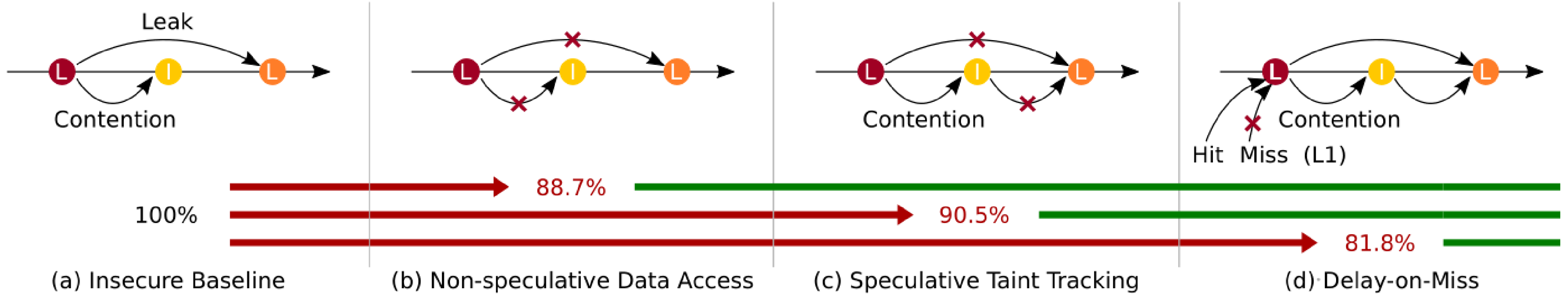


DoM: Delay-on-Miss



Secure Speculation Schemes 2/2

All of these schemes (NDA, STT, and DoM) limit the available memory level parallelism.



This is what we in the industry call “a bad thing”.

Aside: Register File Prefetching

(Not mentioned in the background, but the approach is very similar.)

What if we could predict which address a load was going to use and we prefetched into the physical register? We could save cycles spent waiting for cache access to complete.

This has been done in a paper from Intel. They observed a possible speed-up as high as 3.5% over the baseline [2].

Threat Models

- **NDA & STT:** Secrets exist in memory. Once they are loaded into registers, they are no longer secrets.
  - “The list of covert channels includes the memory hierarchy, timing differences in execution, port contention, and control flow.” [1]
- **DoM:** Secrets can be in registers and in memory.
  - “DoM’s threat model considers leakage through observable, secret-dependent execution of instructions that alter the memory hierarchy, but does not consider leakage through timing differences in execution, port contention, or control flow.” [1]

DOPPELGANGER LOADS

Doppelganger Loads [1]

“The **key insight from NDA-P and STT** is that a load is a potential transmitter when it receives as input an address that is dependent on speculatively loaded value(s).” [1]

- “The **key insight of our work** is that if we can safely predict the addresses for such dependent loads, we can then issue them without leaking speculatively loaded values.” [1]
- “A Doppelganger Load stands in for a delayed dependent load by:
  i) predicting the address of the dependent load,
  ii) preloading the value into the load’s destination register, and
  iii) propagating the preloaded value when the load becomes safe (according to the underlying secure speculation model), if the predicted address matches the load’s resolved address; otherwise, issuing the load with its resolved address when it is safe according to the underlying secure speculation model." [1]

Security of Doppelganger Loads for NDA-P and STT [1]

- "both prediction-based and resolution-based, as defined by Yu et al. [3]."
- "As a prediction-based implicit channel, it is eliminated by *preventing speculative data from affecting the state of the predictor.*"
- "As a resolution-based implicit channel, it is eliminated by delaying the effects of the implicit `imp_if` until the predicate `(ap!=r1)` becomes non-speculative"

```
// Conventional load
PC1 : load r2 , [r1]
// Load with its Doppelganger
PC1 : {
  ap = predict ( PC1 )
  // ap stored in LQ [ PC1 ]
  load r2 , [ap]
  // Doppelganger issues
  impl_if (ap != r1)
    load r2 , [r1] // Load re - issues
}
```

Listing 1: A doppelganger load as an implicit channel (Figure 2 in original paper)

Store-to-Load Forwarding and Memory Consistency

- "doppelganger loads issue independently of store-to-load forwarding." [1]
  - A doppelganger load getting its value forwarded from a store would reveal that the predicted address of the load is equal to the real address of the store.
  - Store-to-load forwarding is done by replacing the prefetched value with the stored value.
- Doppelganger loads can be invalidated because of memory ordering, but the loads themselves are not squashed. Instead, the invalidation is noted and takes effect when the load would normally propagate. Upon misprediction, the invalidation is ignored.
  - Behaviour remains the same w.r.t. invalidations.

Security of Doppelganger Loads for DoM

- Doppelganger used only on L1 miss
- Doppelganger loads allowed to miss L1
- Delayed load on misprediction as in plain DoM
- DoM provides register protection
- Explicit branches as an implicit channel
  - prevented by resolving branches in order
- Implicit branches solved same way as for NDA/STT

```
if ( mispredict ) {
  load r1, [ secret ] // secret loaded speculatively
  if (r1) {
    load r2 , [ X ] // AP and Miss
  } else {
    load r3 , [ Y ] // AP and Miss
  }
}
--
load r1, [ secret ] // secret loaded non-speculatively
if ( mispredict ) {
  if (r1) {
    load r2 , [ X ] // AP and Miss
  } else {
    load r3 , [ Y ] // AP and Miss
  }
}
```

Listing 2: Examples of doppelganger loads in implicit channels (Figure 4 in original paper)

IMPLEMENTATION

Implementation for NDA-P

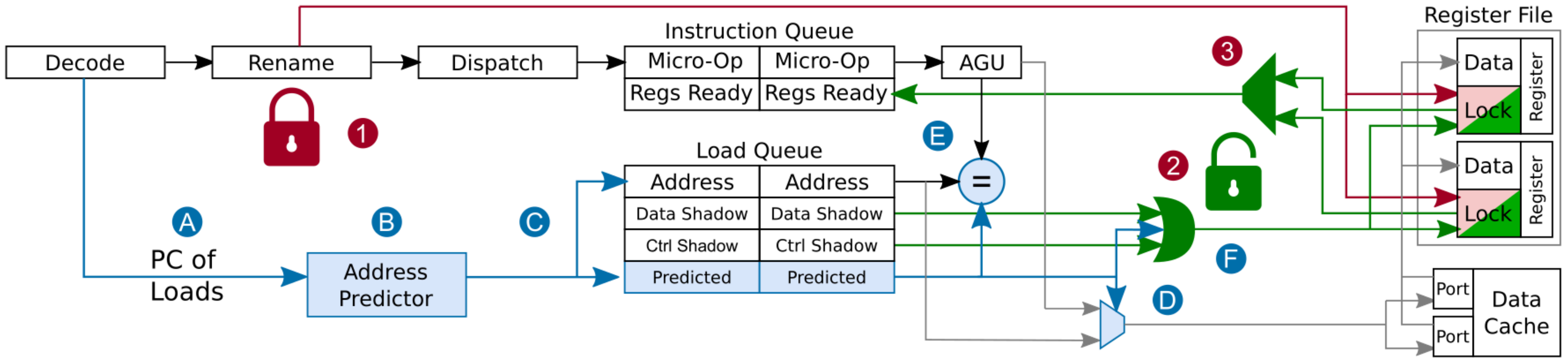


Figure 1: Doppelganger Loads for NDA-P: New structures required are highlighted in blue(Figure 5 in originial paper)

Show that doppelganger loads can feasibly be implemented on top of secure speculation schemes. (Implementation is slightly different for STT and DoM.)

Implementation Cost

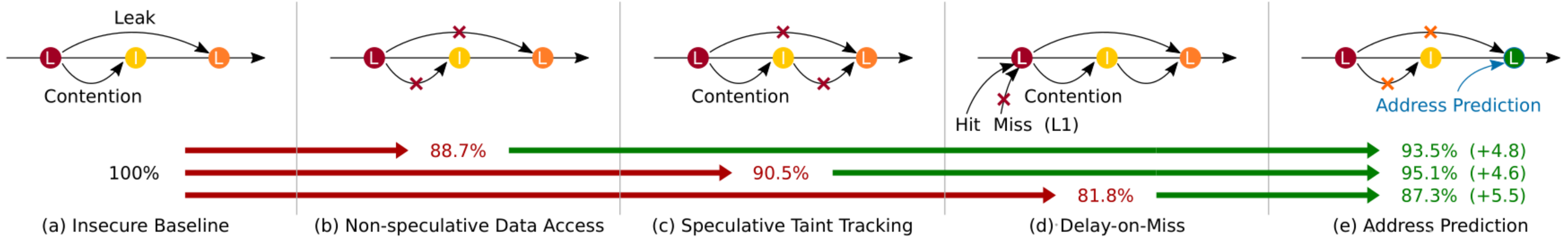
- Load and doppelganger share resources:
  - Load queue entry
  - Physical destination register
- Load and doppelganger resource needs don’t overlap in time
- Upon incorrect prediction, an extra address translation and memory request is used.
  - Don’t need to squash dependent instructions.
- Address predictor can share resources with an L1 prefetcher except: predict current access instead of future access
- No modification to memory hierarchy

METHODOLOGY

Methodology

- Implement the schemes in gem5
- Using SPEC CPU2006 and SPEC CPU2017 for representative single-threaded programs.
- Detailed simulations using the out-of-order (o3) CPU, configured to resemble IceLake
- Evaluate many schemes:
  - Unsafe Baseline
  - NDA-P
  - STT
  - DoM
  - Unsafe Baseline + Address Prediction
  - NDA-P + Address Prediction
  - STT + Address Prediction
  - DoM + Address Prediction

EVALUATION


Leak
Contention
Contention
Contention
Hit
Miss (L1)
Address Prediction
100%
88.7%
93.5% (+4.8)
90.5%
95.1% (+4.6)
81.8%
87.3% (+5.5)
(a) Insecure Baseline
(b) Non-speculative Data Access
(c) Speculative Taint Tracking
(d) Delay-on-Miss
(e) Address Prediction

Evaluation: IPC

- Unsafe baseline + Address Prediction only achieved a geometric mean speedup of 0.5%.

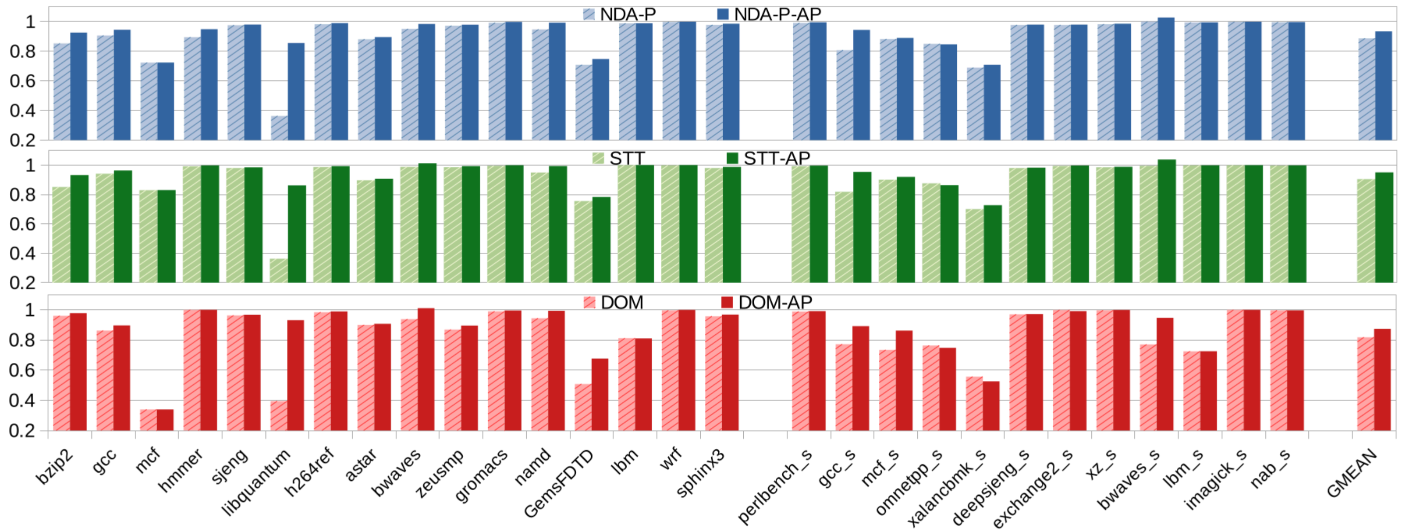


Figure 2: Normalized IPC to baseline of the schemes, enhanced with doppelganger loads. (Figure 6 in the original paper)

Evaluation: Coverage and Accuracy

“looking at the predictor performance and the overall performance of address prediction, there is no clear correlation”

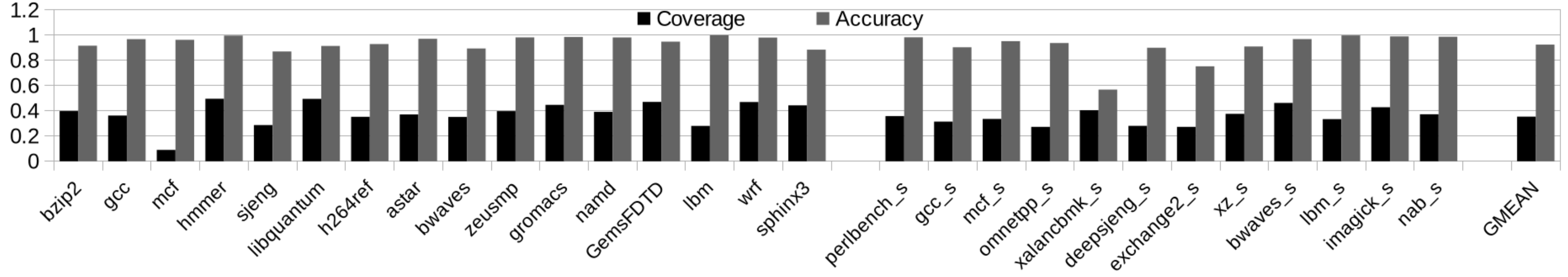


Figure 3: Coverage and Accuracy for address prediction under DoM (similar results observed for NDA-P, STT) (Figure 7 in original paper)

Evaluation: L1 and L2 accesses

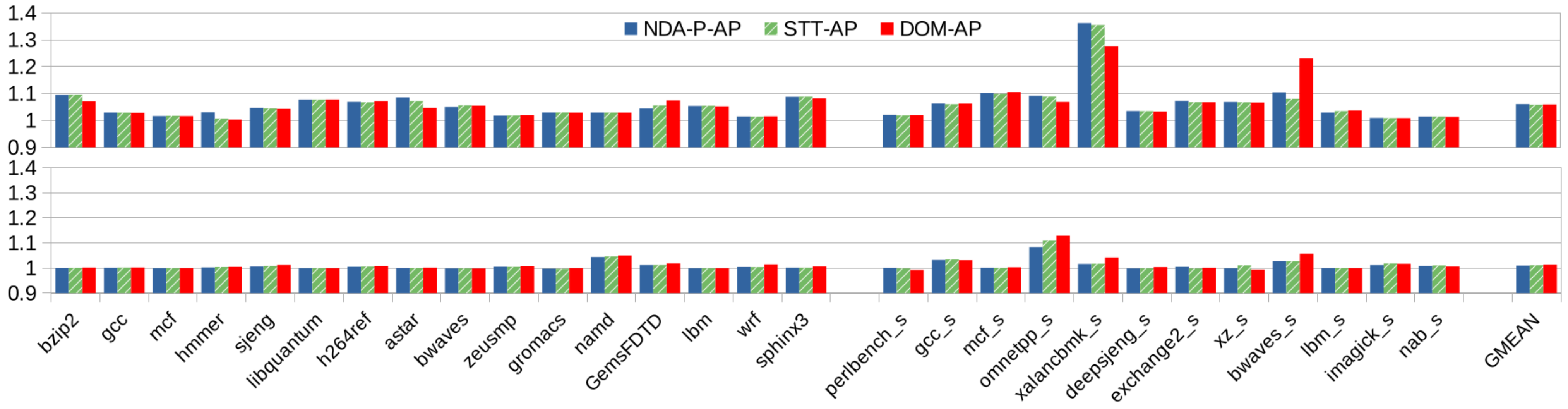


Figure 4: Normalized L1 and L2 accesses (Figure 8 in original paper)

Bibliography

[1] A. B. Kvalsvik, P. Aimoniotis, S. Kaxiras, and M. Sjalander, “Doppelganger Loads: A Safe, Complexity-Effective Optimization for Secure Speculation Schemes”, Association for Computing Machinery, Jun. 2023. Available: https://doi.org/10.1145/3579371.3589088

[2] S. Shukla, S. Bandishte, J. Gaur, and S. Subramoney, “Register file prefetching”, Association for Computing Machinery, Jun. 2022, pp. 410–423. Available: https://doi.org/10.1145/3470496

[3] Jiyong Yu, Mengjia Yan, Artem Khyzha, Adam Morrison, Josep Torrellas, and Christopher W. Fletcher, “Speculative Taint Tracking (STT): A comp”, Association for Computing Machinery, Oct. 2019, pp. 954–968. Available: https://doi.org/10.1145/3352460.3358274

QUESTIONS

Questions

- Why are predictor performance and the performance gain from address prediction not clearly correlated?
- Why is in-order branch resolution required?