Spectre on Xiangshan RISC-V

ACA Project

POLITECNICO
MILANO 1863

Paraula Rey - 10781357
rey.paraula@mail.polimi.it

Petenzi Roberto - 10797415
Roberto.petenzi@mail.polimi.it

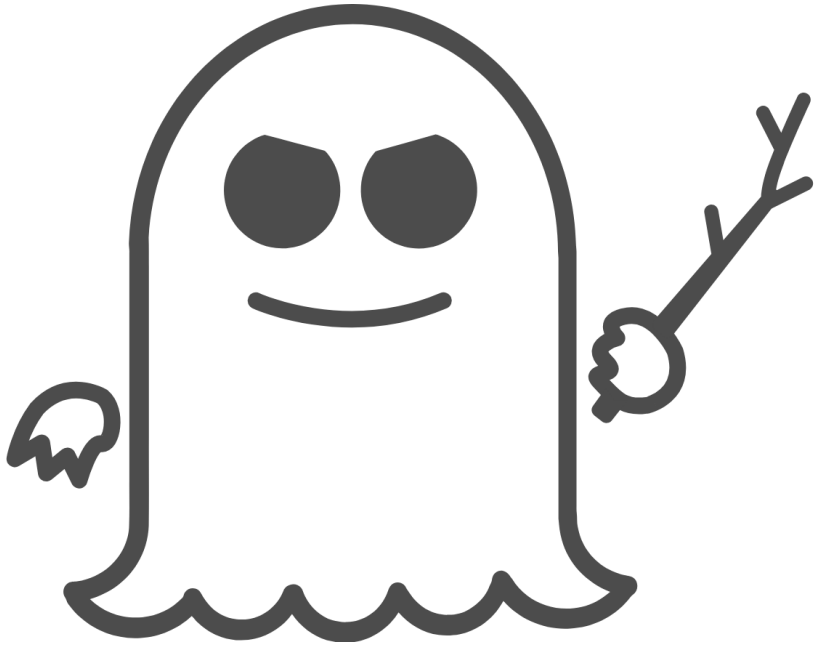
Hardware vulnerabilities are expensive

- Not always fixable via sofware updates
- If possible, loss of performances/functionalities
- Can circumvent software security measures
- Chip design is slooooow

Goals

1. Assess RISC-V processor vulnerability to the Spectre attack
2. Case study: the Xiangshan processor
3. Impact of mitigations

What is Spectre?



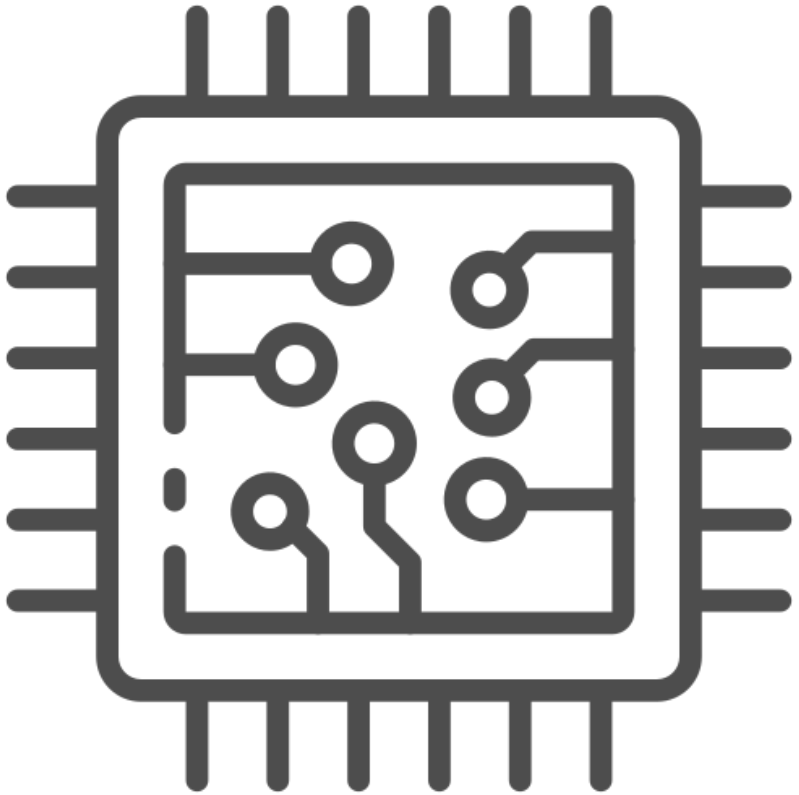
"Spectre [1] *exploits wrong assumptions (previously deemed as "safe") about reverting the results of speculative execution and exfiltrates data through cache microarchitectural side-channels."*

2 main ingredients:

- Speculative execution
- Cache side-channels

[1] *Spectre Attacks: Exploiting Speculative Execution*

When is a CPU vulnerable to Spectre?



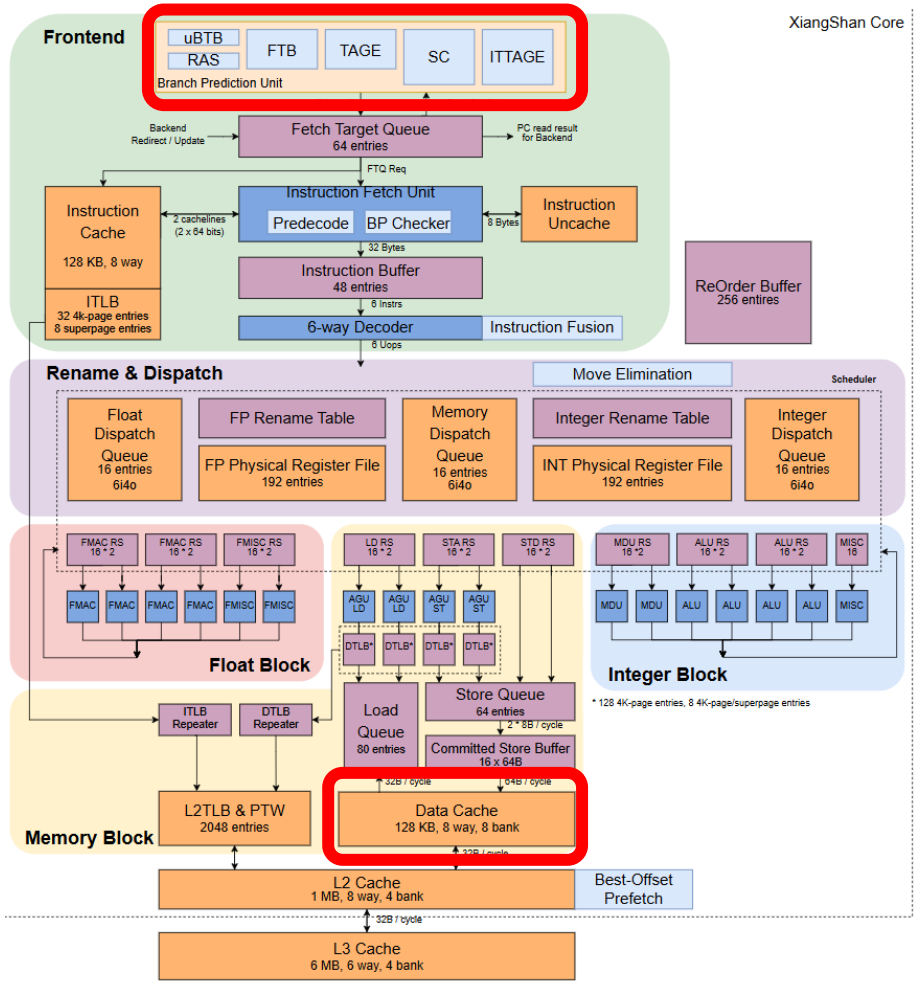
- Speculative execution
- Out Of Order execution is not needed
- Data prefetching is detrimental for the attack
- Timers and/or counters

What is Xiangshan?[3]

- Superscalar high-performance RISC-V processor
- Open-source
- Meets industrial-grade standards[4]
- Developed using an agile workflow: very fast development cycle (tape out in 2 years)

[3] *Towards Developing High Performance RISC-V Processors Using Agile Methodology*
[4] XiangShan: An Open-Source Project for High-Performance RISC-V Processors Meeting Industrial-Grade Standards

Xiangshan versions



- V1: Yanqihu
- V2: Nanhu (stable)
  - No CMO extension yet :(
- V3: Kunminghu (still under development)
  - CMO extension available
  - No relevant architectural changes

[5] *OpenXiangShan GitHub Repo*

Branch (mis)prediction

- Attack divided into rounds for each byte:
  - Mistrain the branch predictor with FP ops
  - Guess the value with cache side-channel

```
if (idx < array1_sz){
  dummy = array2[array1[idx] * L1_BLOCK_SZ_BYTES];
}
```

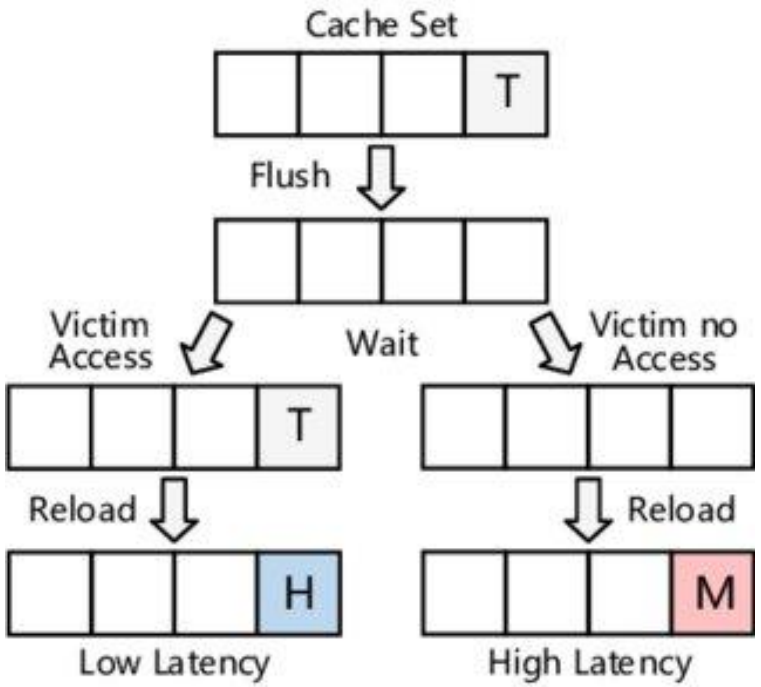
Primer on cache side channels

- Leak sensitive information by measuring cache access times[6]

- Types[7]:
  - **Flush+Reload**
  - Prime+Probe
  - Flush+Flush
  - Flush+Fault

[6] *Low-Level Software Security for Compiler Developers*
[7] *A Systematic Evaluation of Novel and Existing Cache Side Channels*

Flush + Reload




```
start = rdmcycle();
dummy &= array2[i * L1_BLOCK_SZ_BYTES];
diff = (rdmcycle() - start);
if ( diff < CACHE_HIT_THRESHOLD ){
    results[i] += 1;
}
```

1. Flush attacker controlled `array2`
2. Data cached through speculation
3. Test each `array2[i *L1_BLOCK_SZ_BYTES]` cell to see which was cached.
4. The cached `i` is the out of bounds secret that we wanted to read.

XiangShan Simulators

Verilator: Cycle accurate

```
Using simulated 8192MB RAM
The image is build/cachetest-riscv64-xs.bin
data cache test
iteration=0, read from ram, cycles = 116
iteration=0, read from data cache, cycles = 33
iteration=1, read from ram, cycles = 44
iteration=1, read from data cache, cycles = 33
iteration=2, read from ram, cycles = 96
iteration=2, read from data cache, cycles = 33
iteration=3, read from ram, cycles = 115
iteration=3, read from data cache, cycles = 33
iteration=4, read from ram, cycles = 90
iteration=4, read from data cache, cycles = 33
```

NEMU: Cache not simulated :(

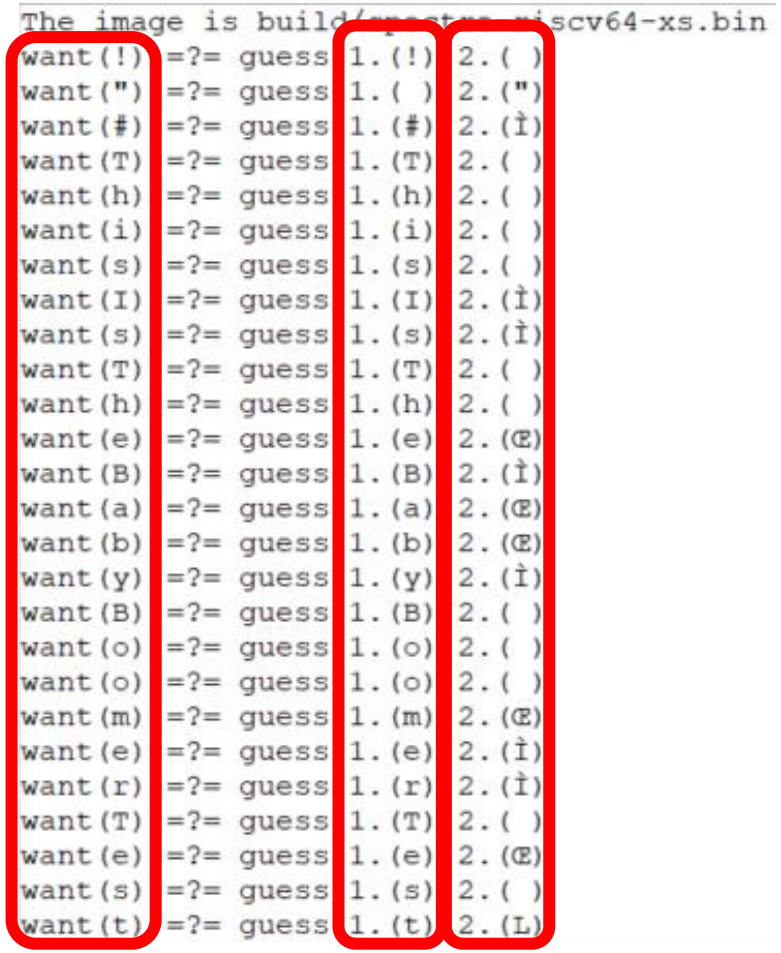
```
Welcome to riscv64-NEMU!
For help, type "help"
data cache test
iteration=0, read from ram, cycles = 0
iteration=0, read from data cache, cycles = 0
iteration=1, read from ram, cycles = 0
iteration=1, read from data cache, cycles = 0
iteration=2, read from ram, cycles = 0
iteration=2, read from data cache, cycles = 0
iteration=3, read from ram, cycles = 0
iteration=3, read from data cache, cycles = 0
iteration=4, read from ram, cycles = 0
iteration=4, read from data cache, cycles = 0
```

Running the attack – V2

- Attack based on previous findings on BOOM[8]

- We modified
  - Cache HIT threshold
  - Cache parameters to match XiangShan ones
  - Used counter register

- Speed: 0.98 B/Mcycle



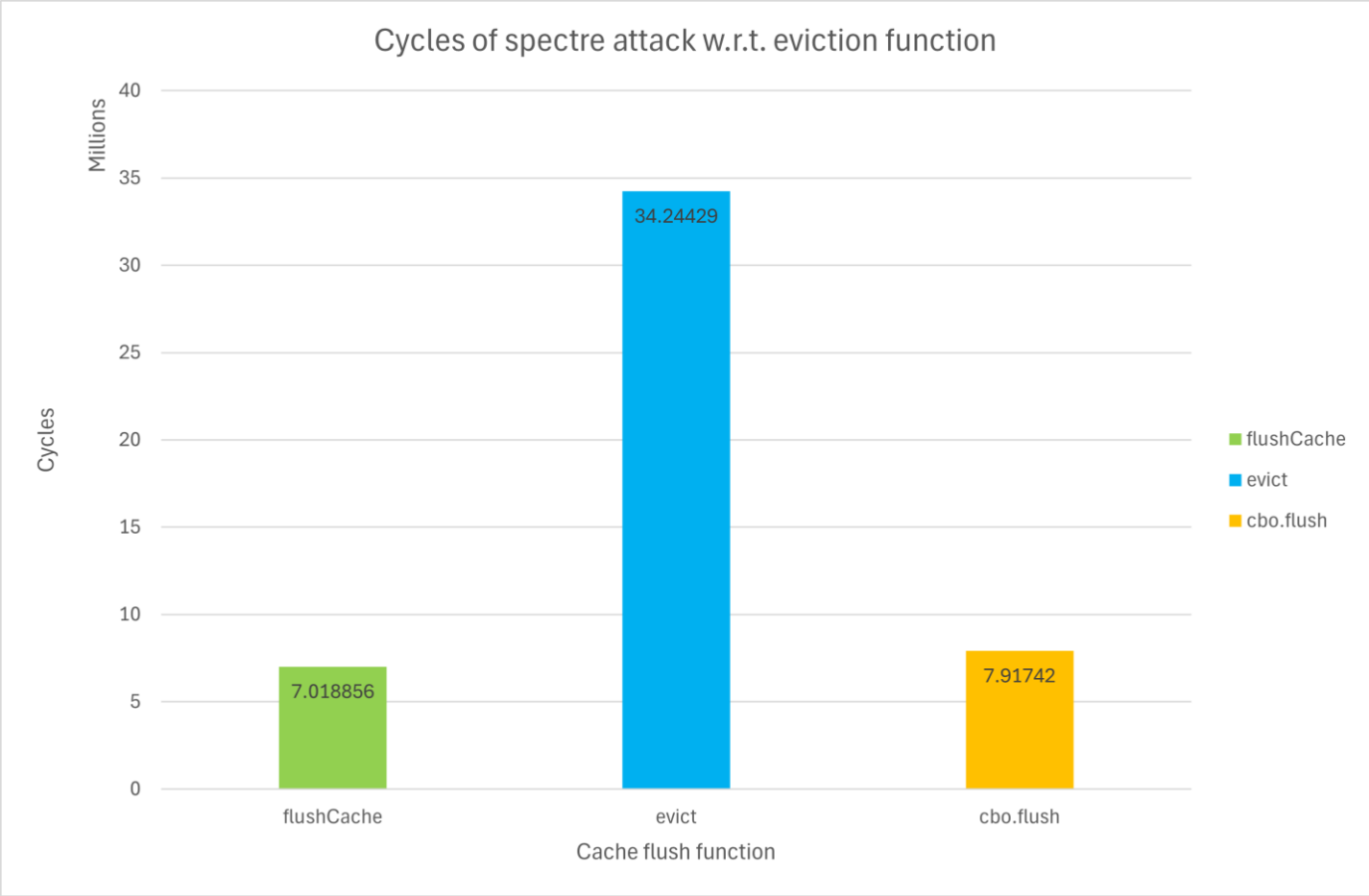
[8] BOOM Speculative Attack

Running the attack – V3

- Does not work as is
- Out of Order read of CSR (counter)
- Fences to the rescue
- Using CMOs
- Speed: 3.70 B/Mcycles

```
The image is build/spectre-riscv64-xs.bin
want(!) =?= guess 1.(!) 2.(?)
want(") =?= guess 1.(") 2.(|)
want(#) =?= guess 1.(#) 2.(?)
want(T) =?= guess 1.(T) 2.(?)
want(h) =?= guess 1.(h) 2.(?)
want(i) =?= guess 1.(i) 2.(|)
want(s) =?= guess 1.(s) 2.(|)
want(I) =?= guess 1.(I) 2.(?)
want(s) =?= guess 1.(s) 2.(|)
want(T) =?= guess 1.(T) 2.(|)
want(h) =?= guess 1.(h) 2.(?)
want(e) =?= guess 1.(e) 2.(?)
want(B) =?= guess 1.(B) 2.(|)
want(a) =?= guess 1.(a) 2.(|)
want(b) =?= guess 1.(b) 2.(|)
want(y) =?= guess 1.(y) 2.(?)
want(B) =?= guess 1.(B) 2.(|)
want(o) =?= guess 1.(o) 2.(?)
want(o) =?= guess 1.(o) 2.(|)
want(m) =?= guess 1.(m) 2.(|)
want(e) =?= guess 1.(e) 2.(|)
want(r) =?= guess 1.(r) 2.(?)
want(T) =?= guess 1.(T) 2.(|)
want(e) =?= guess 1.(e) 2.(|)
want(s) =?= guess 1.(s) 2.(?)
want(t) =?= guess 1.(t) 2.(|)
```

Statistics – Different eviction function

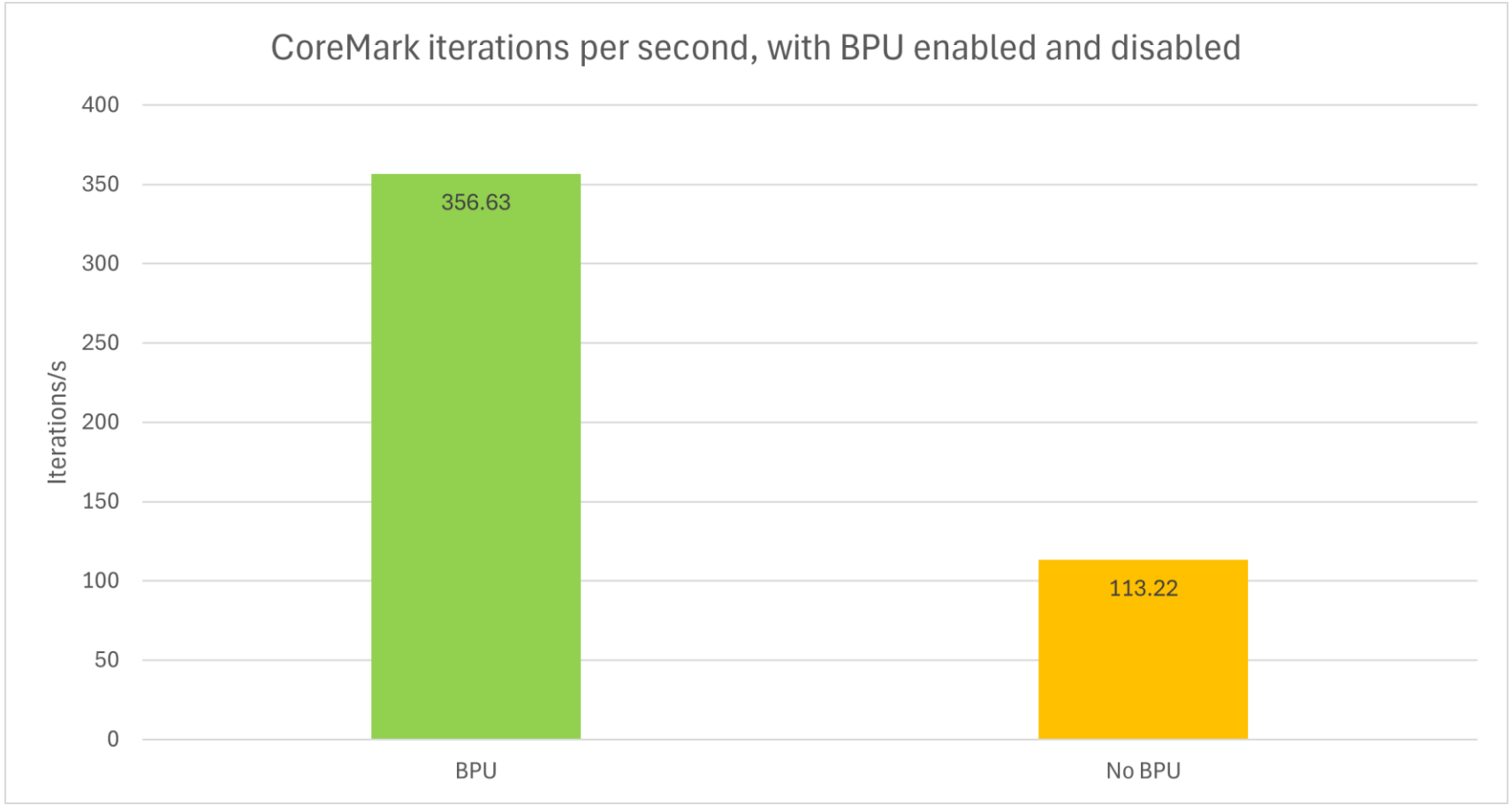


- `flushCache`: same as BOOM attacks, with tuned parameters
- `evict`: used in a side-channel attack on Nanhu[9]
- `cbo.flush`: “native” RISC-V eviction function

[9] L1 side channel on Nanhu

Statistics – Disabling BPU mitigation

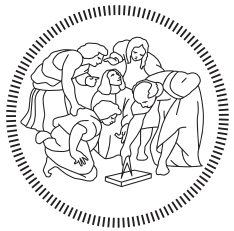
- Worst case, yet effective mitigation



CoreMark: 2 iterations, 2K performance run parameters

Conclusions

- Xiangshan’s architecture is currently vulnerable to Spectre
- Mitigations greatly reduce the processor’s performance



POLITECNICO
MILANO 1863

Thank you for your attention!

Paraula Rey
rey.paraula@mail.polimi.it

Petenzi Roberto
roberto.petenzi@mail.polimi.it