Micro-Architectural Attacks and Defenses

11/15/2021

Heechul Yun

Associate Professor, EECS

University of Kansas

Micro-Architectural Attacks

- Software attacks on hardware
- Complex hardware → **many attack vectors**

Micro-Architectural Attacks

- Micro-architectural hardware components
  - E.g., cache, tlb, DRAM, OoO engine, …
- Can leak secret
  - E.g., Meltdown, Spectre
- Can alter the content of the stored data
  - E.g., RowHammer
- Can affect execution timing
  - E.g., DoS attack on real-time tasks
- **Logically correct software is also vulnerable**

Today's Talk

- A new contention-based covert channel
  - Jacob Fustos, Michael Garrett Bechtel, Heechul Yun. SpectreRewind: Leaking Secrets to Past Instructions. *Workshop on Attacks and Solutions in Hardware Security (ASHES)*, 2020.
- A new denial-of-service (DoS) attack
  - Michael Garrett Bechtel and Heechul Yun. Memory-Aware Denial-of-Service Attacks on Shared Cache in Multicore Real-Time Systems. *IEEE Transactions on Computers*, 2021.
- A hardware defense mechanism for DoS attacks
  - Farzad Farshchi, Qijing Huang, and Heechul Yun. BRU: Bandwidth Regulation Unit for Real-Time Multicore Processors. *IEEE Intl. Conference on Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2020.

SpectreRewind: Leaking Secrets to Past Instructions

Jacob Fustos, Michael Bechtel, Heechul Yun

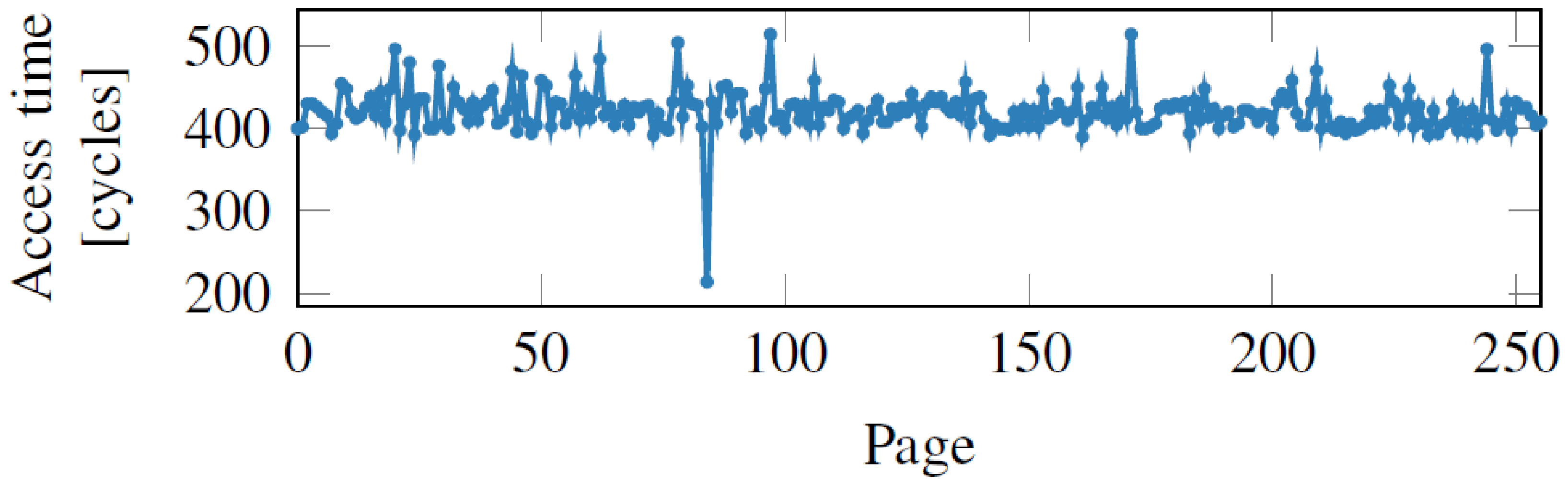
University of Kansas, USA

Workshop on Attacks and Solutions in Hardware Security (ASHES), 2020.

Speculative Execution Attacks

- Attacks exploiting microarchitectural side-effects left by speculative (transient) instructions
- Many variants: Spectre, Meltdown, Foreshadow, MDS, LVI, …
- Secrets are transferred over microarchitectural covert channels
- Most known attacks use **cache covert channels**

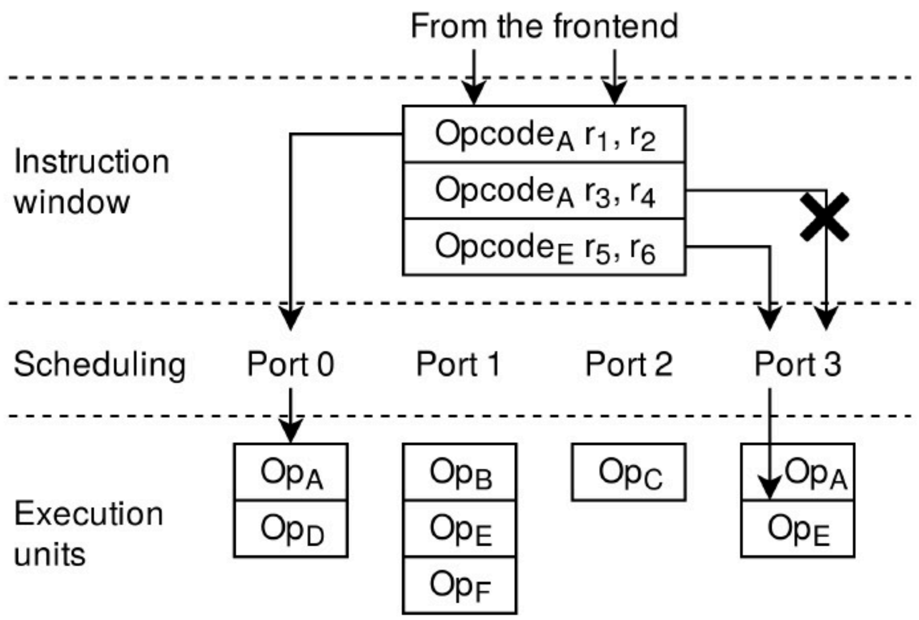
Cache Covert Channel



- By measuring access timing differences of a memory location, an attacker can determine whether the memory is cached or not.
- Secret is recovered ***after*** transient executions are squashed
- Many proposals exist to mitigate cache-based channels

Image source: M. Lipp et al., "Meltdown: Reading kernel memory from user space.," USENIX Security., 2018.

Contention Covert Channels



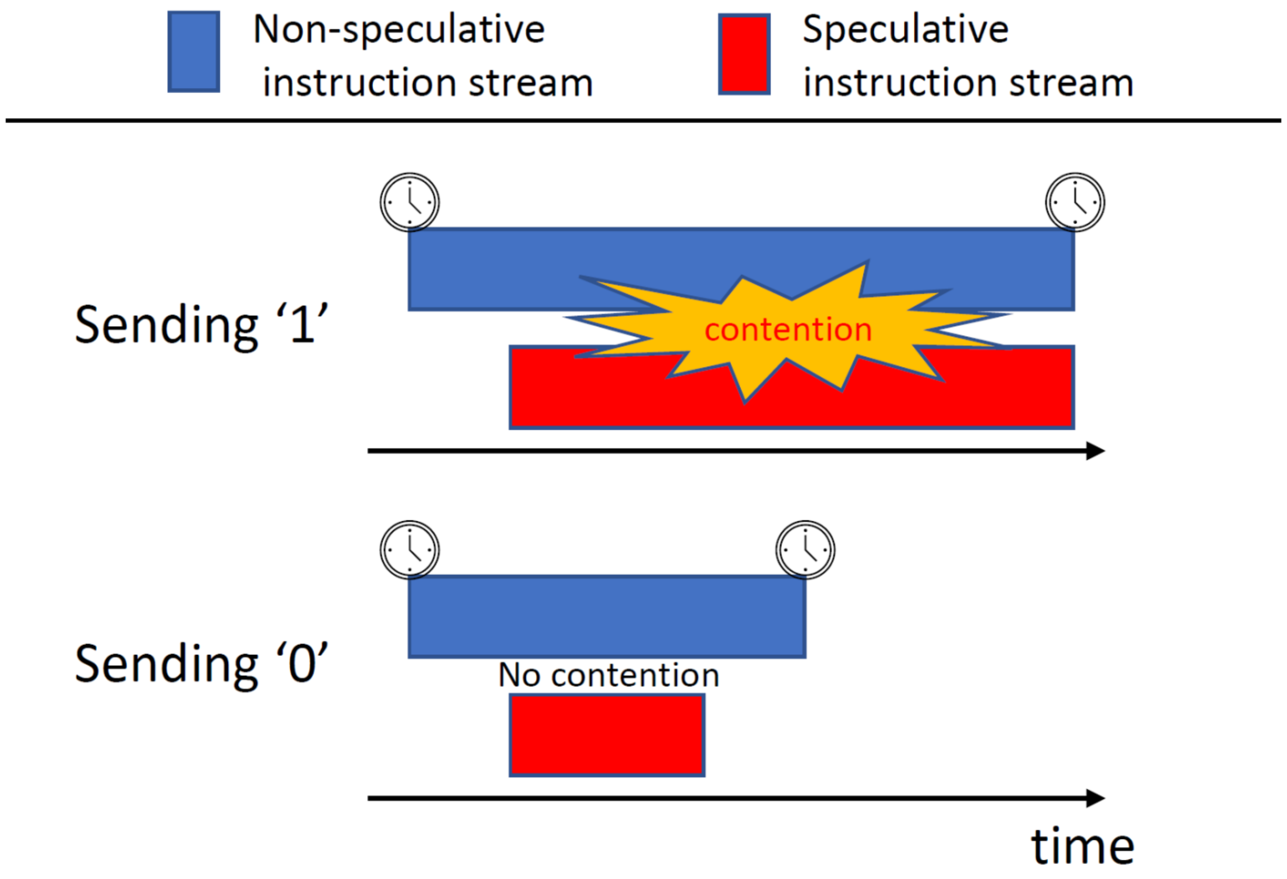
- Exploit that contention on shared functional units/ports between Simultaneous multithreading (SMT) threads
- Secret is transmitted ***during*** the speculative execution
- Mitigation solutions including disabling SMT

Image source: Bhattacharyya et al, "SmotherSpectre," ACM CCS, 2019

SpectreRewind

- A novel **contention-based covert channel**
- Transmits secret from speculative instructions to (non-speculative) **past instructions**
- Through **non-pipelined functional units** on a single hardware thread (no SMT)
- Bypasses all existing defenses against cache or SMT based covert channels

SpectreRewind

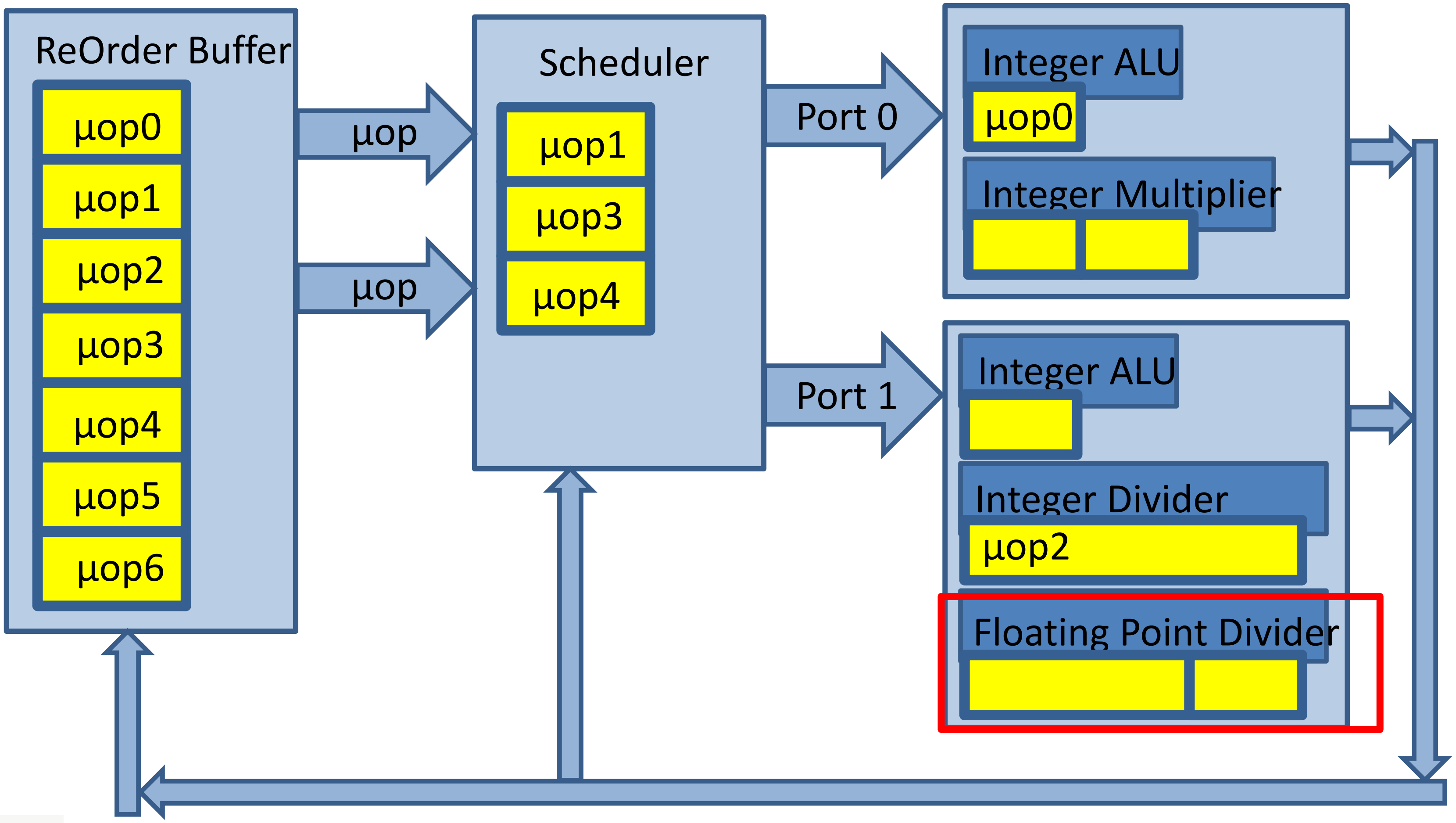


🟦 Non-speculative instruction stream
🟥 Speculative instruction stream
Sending '1'
contention
Sending '0'
No contention
time

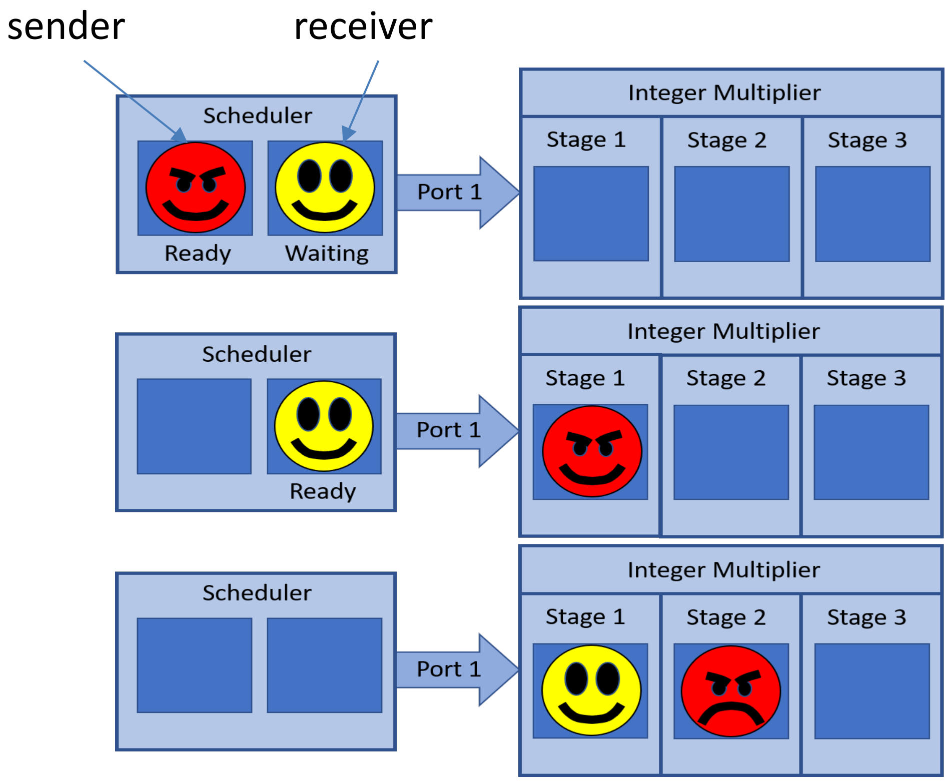
SpectreRewind

- Receiver
  - – Non-speculative (bound-to-retire) instructions
- Sender
  - – Secret depend speculative instructions
- Covert Channel
  - – Shared **non-pipelined functional units**
  - – Other possibilities: prefetcher, MSHRs, etc…

Modern Out-of-Order Architecture



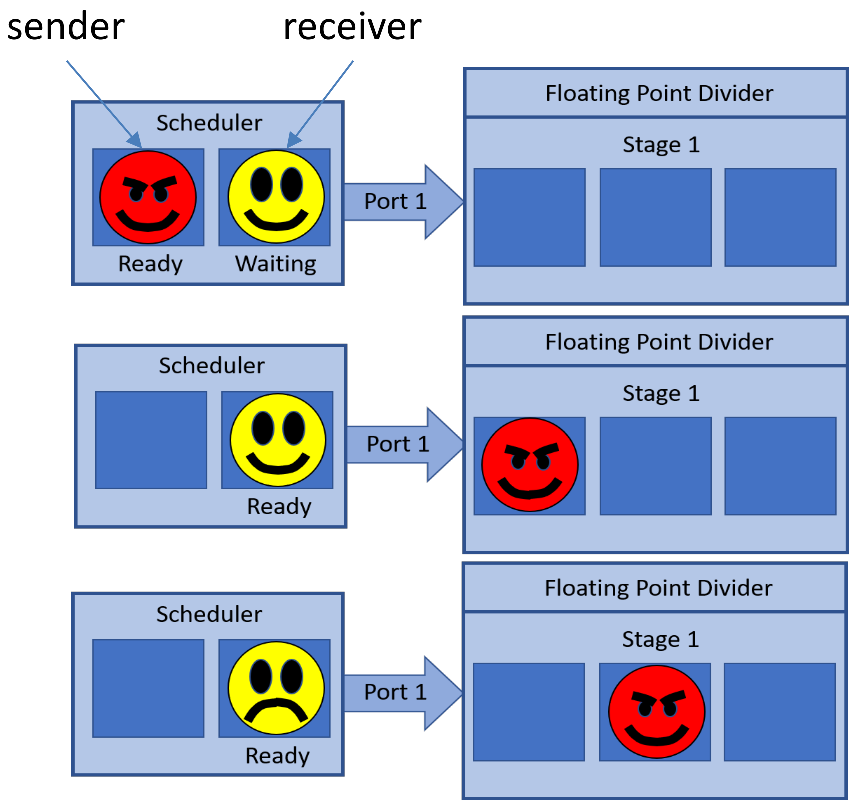
Pipelined Functional Unit



(a) Fully pipelined functional unit

- Sender (young) cannot delay Receiver (old)

Non-pipelined Function Units

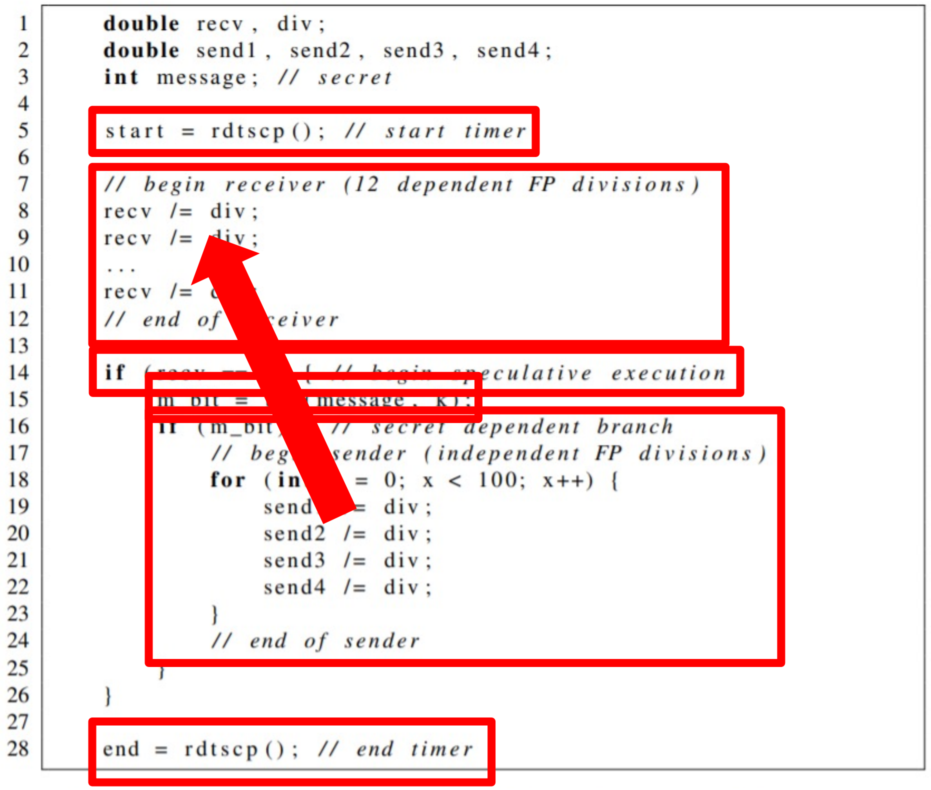


(b) Non-pipelined functional unit

- Sender (young) delays Receiver (old)

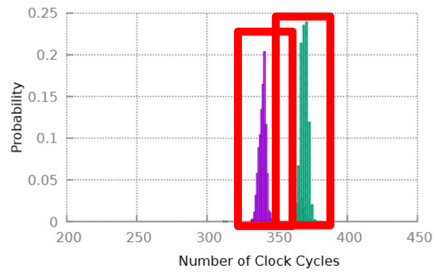
Floating Point Division Covert Channel

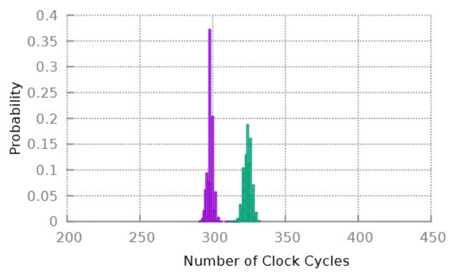
- Start a timer
- Perform multiple divisions
- Cause a mis-speculation
- Calculate a bit to transmit
- If bit is ‘1’ do more division
- Cause contention with receiver
- Time entire attack

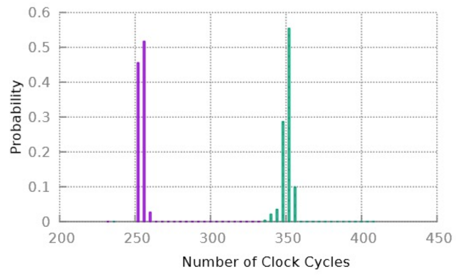


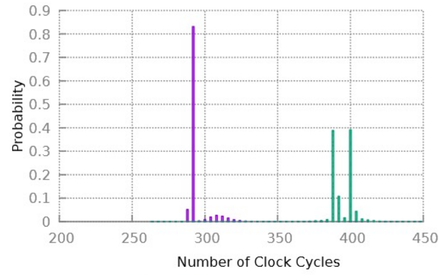
```
double recv, div;
double send1, send2, send3, send4;
int message; // secret

start = rdtscp(); // start timer

// begin receiver (12 dependent FP divisions)
recv /= div;
recv /= div;
...
recv /= [illegible]
// end of [illegible]eiver

if ([illegible]) { // begin speculative execution
    m_bit = [illegible](message, k);
    if (m_bit) { // secret dependent branch
        // beg[illegible]sender (independent FP divisions)
        for (in[illegible] = 0; x < 100; x++) {
            send[illegible]= div;
            send2 /= div;
            send3 /= div;
            send4 /= div;
        }
        // end of sender
    }
}

end = rdtscp(); // end timer
```

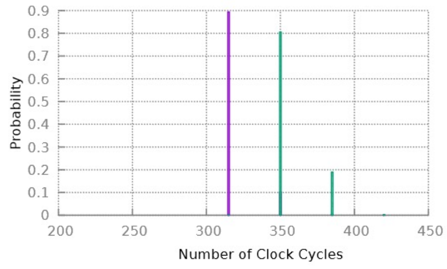
Channel Properties

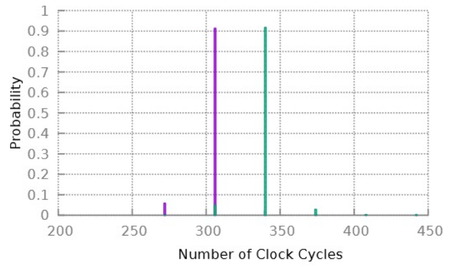

(a) Kabylake R (i5-8250U)

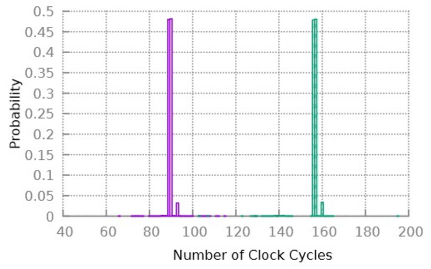

(b) Skylake (i5-6500)

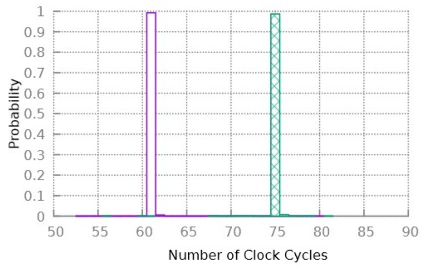

(c) Haswell (E5-2658v3)


(d) Ivybridge (i5-3340M)


(e) Zen (Ryzen3 2200G)

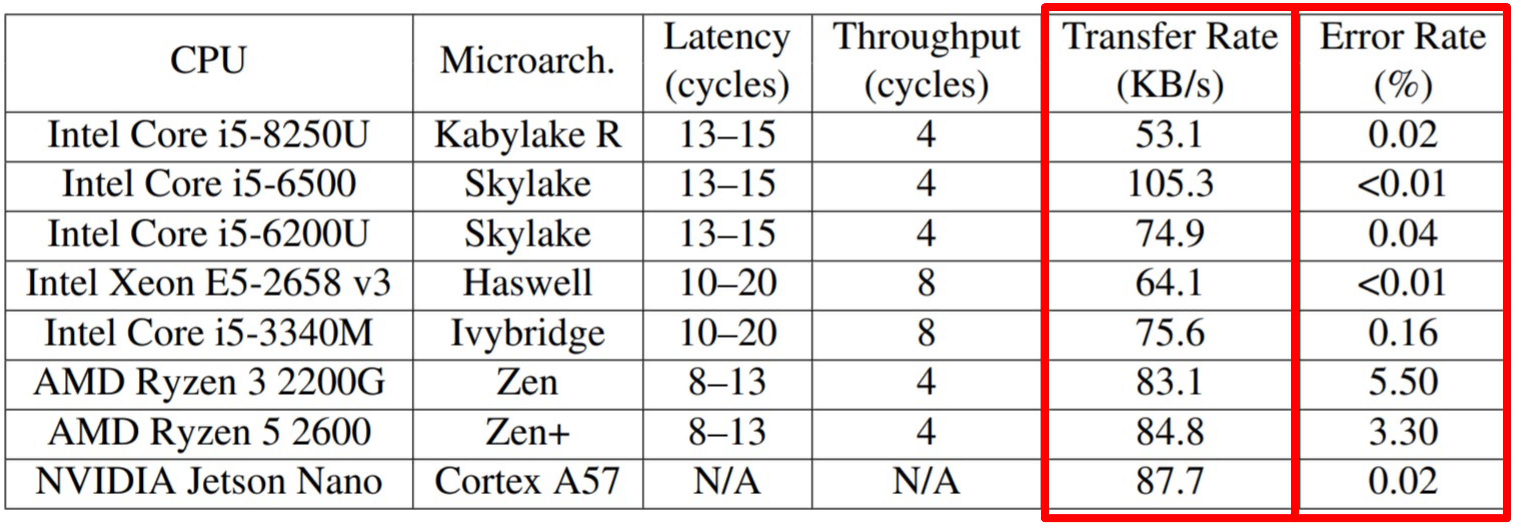

(f) Zen+ (Ryzen5 2600)


(g) Cortex-A57 (Jetson Nano)*


(h) Cortex-A72 (Raspberry Pi 4)*

- Clearly distinguishable patterns on all tested Intel, AMD, ARM processors

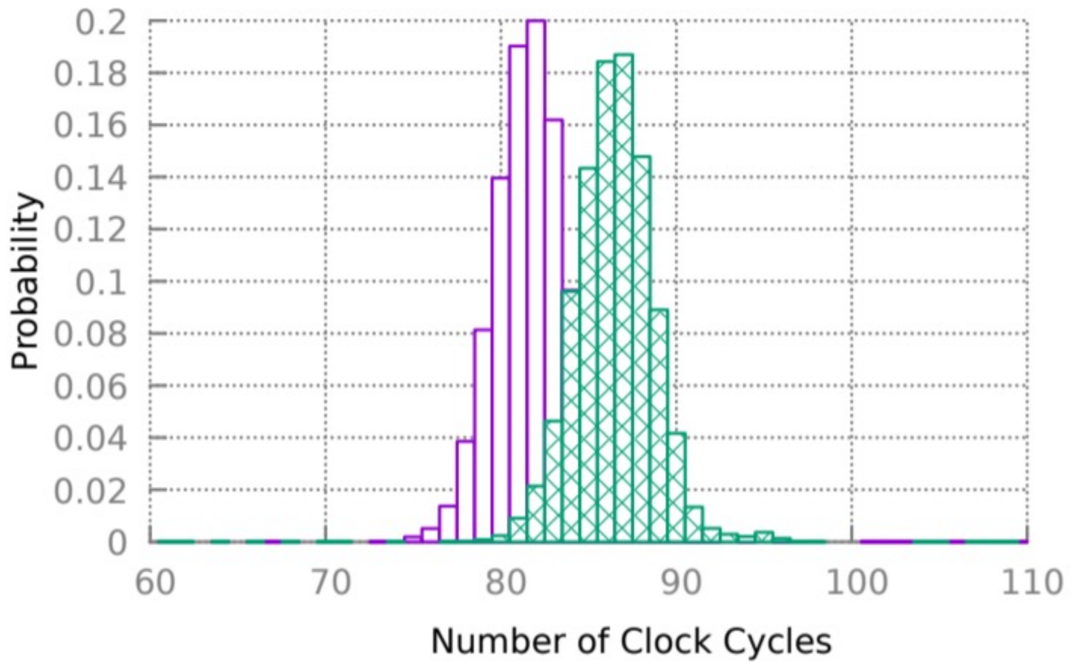
Performance Analysis



CPU	Microarch.	Latency (cycles)	Throughput (cycles)	Transfer Rate (KB/s)	Error Rate (%)
Intel Core i5-8250U	Kabylake R	13–15	4	53.1	0.02
Intel Core i5-6500	Skylake	13–15	4	105.3	<0.01
Intel Core i5-6200U	Skylake	13–15	4	74.9	0.04
Intel Xeon E5-2658 v3	Haswell	10–20	8	64.1	<0.01
Intel Core i5-3340M	Ivybridge	10–20	8	75.6	0.16
AMD Ryzen 3 2200G	Zen	8–13	4	83.1	5.50
AMD Ryzen 5 2600	Zen+	8–13	4	84.8	3.30
NVIDIA Jetson Nano	Cortex A57	N/A	N/A	87.7	0.02

- High transfer rates and low error rates

Google Chrome Sandbox



- Implemented a SpectreRewind PoC in JavaScript on Chrome
- Noisier but still distinguishable timing differences

Discussion

- Benefits
  - Does not require SMT hardware (single thread)
  - Defeats all known hardware solutions for stateful (cache) covert channels
  - Alternative to cache-based covert channels like Flush+Reload
- Limitations (*)
  - Limited to same address space attacks
  - Finding division-based gadgets may be difficult
  - Attacker controls both receiver and sender

(*) M. Behnia et al., "Speculative Interference Attacks: Breaking Invisible Speculation Schemes," ASPLOS, 2021.

Summary

- A novel **contention-based covert channel**
  - Transmits secret from speculative instructions to (non-speculative) **past instructions**
  - Through **non-pipelined functional units** on a single hardware thread (no SMT)
  - Bypasses all existing defenses against cache or SMT based covert channels
  - Achieves **high throughput and low error rates**
  - Works on all tested Intel, AMD, and ARM processors

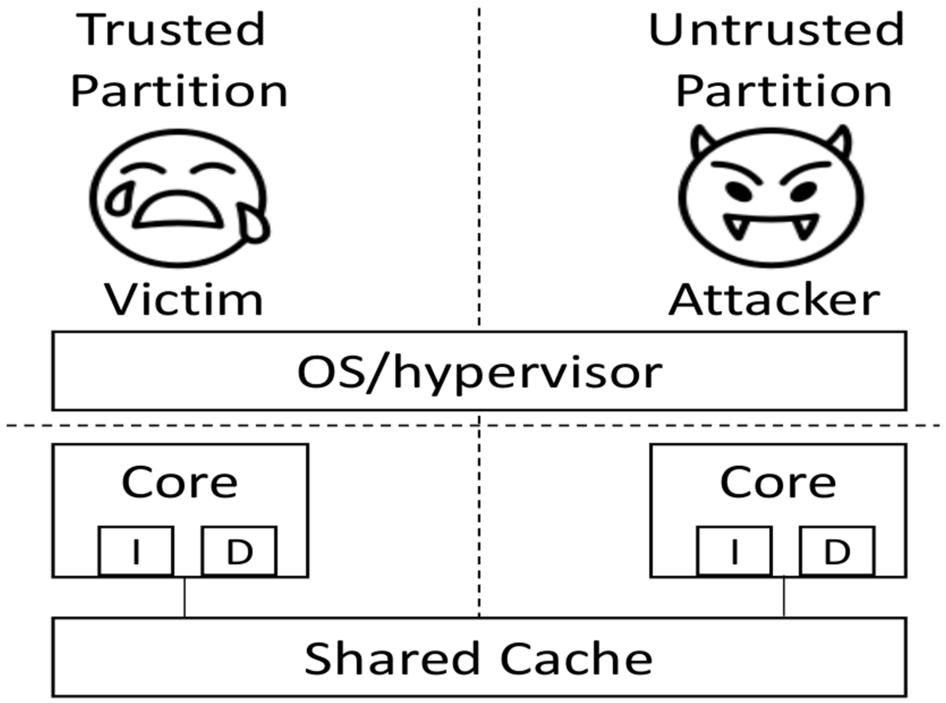
Memory-Aware Denial-of-Service Attacks on Shared Cache in Multicore Real-Time Systems

Michael Bechtel, Heechul Yun

University of Kansas, USA

IEEE Transactions on Computers, 2021.

Denial-of-Service Attacks



- Attacker's goal: increase the victim's task **execution time**
- The attacker is on different core/memory/cache partition
- The attacker can only execute non-privileged code.

Cache DoS Attacks

```
for (i = 0; i < mem_size; i += LINE_SIZE)
{
        sum += ptr[i];
}
```

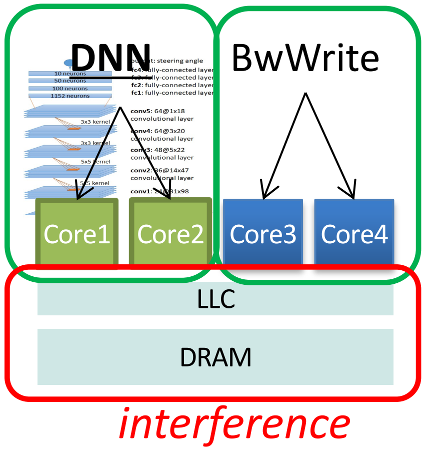
BwRead
(target MSHRs)

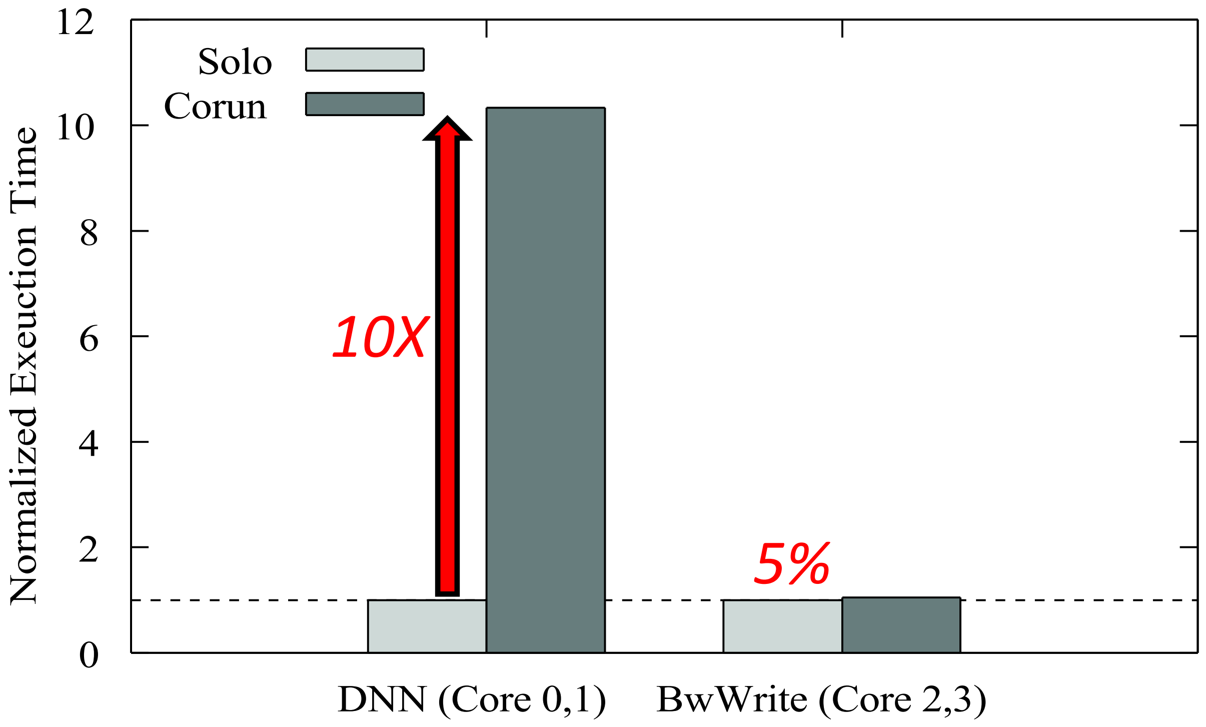
```
for (i = 0; i < mem_size; i += LINE_SIZE)
{
        ptr[i] = 0xff;
}
```

BwWrite
(target WBBuffer)

- Denial-of-Service (DoS) attacks targeting internal hardware structures of a shared cache.
  - Block the cache → delay the victim's execution time

Effects of Denial-of-Service Attacks





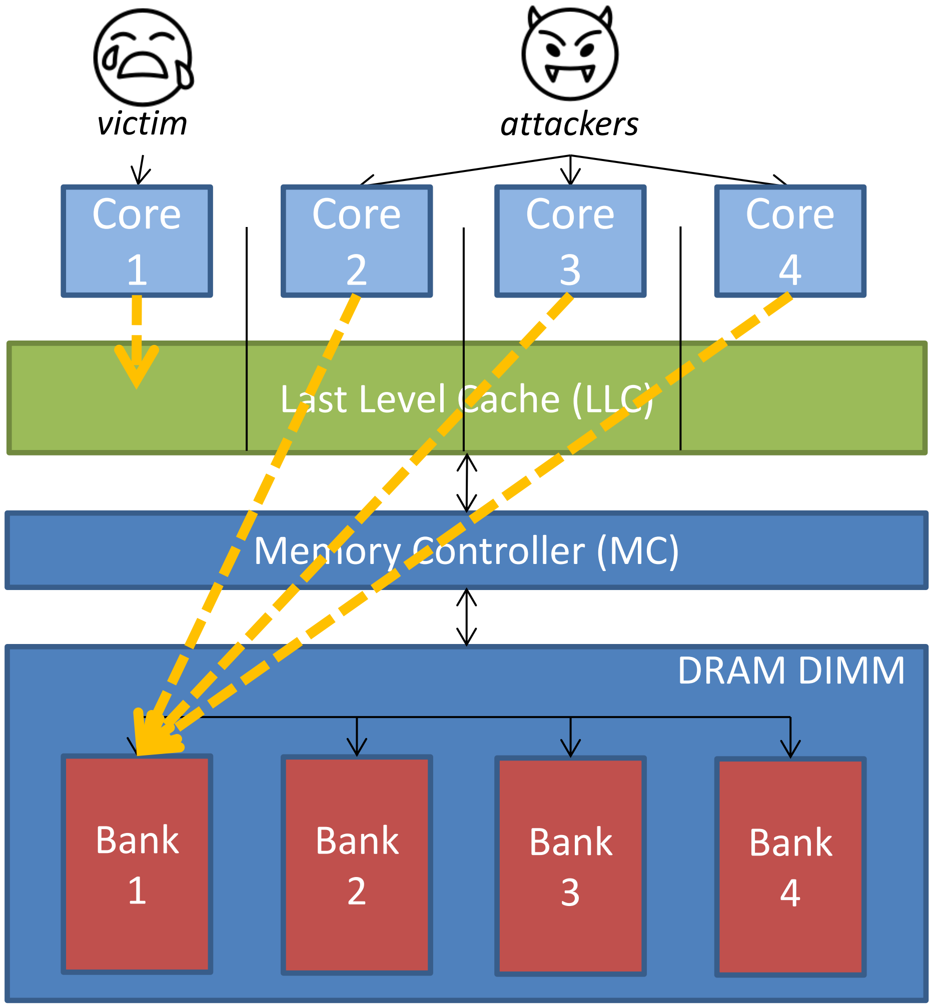
- **Delay execution time** of time sensitive code
  - Observed up to **10X** increase$^{(**)}$
    - Of a realistic DNN-based real-time control program

(**) Michael Garrett Bechtel, Elise McEllhiney, Minje Kim, Heechul Yun. "DeepPicar: A Low-cost Deep Neural Network-based Autonomous Car." In *RTCSA*, IEEE, 2018

Hypothesis

- Effective cache DoS attacks require many concurrent in-flight memory requests to DRAM to induce cache blocking
- Cache blocking will last longer if the DRAM memory requests are processed **slowly**
- Sequential memory requests in prior cache-DoS attacks are processed efficiently, leveraging DRAM bank-level parallelism
- Intentionally **inefficient memory requests** can make more **effective cache DoS attacks**

Memory-Aware Cache DoS Attack



- Attacker intentionally generate DRAM bank conflicts
- Induce longer cache blocking
- Victim's execution time increases

Memory-Aware Cache DoS Attack

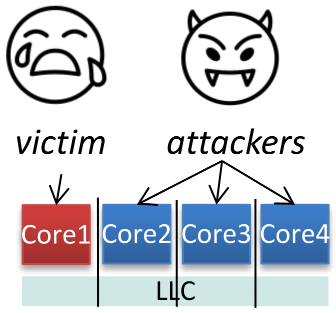
```
static int* list[MAX_MLP];
static int next[MAX_MLP];

for (int64_t i = 0; i < iter; i++) {
      switch (mlp) {
      case MAX_MLP:
      .
      .
      case 2:
            next[1] =
                  list[1][next[1]];
            /* fall-through */
      case 1:
            next[0] =
                  list[0][next[0]];
      }
}
```

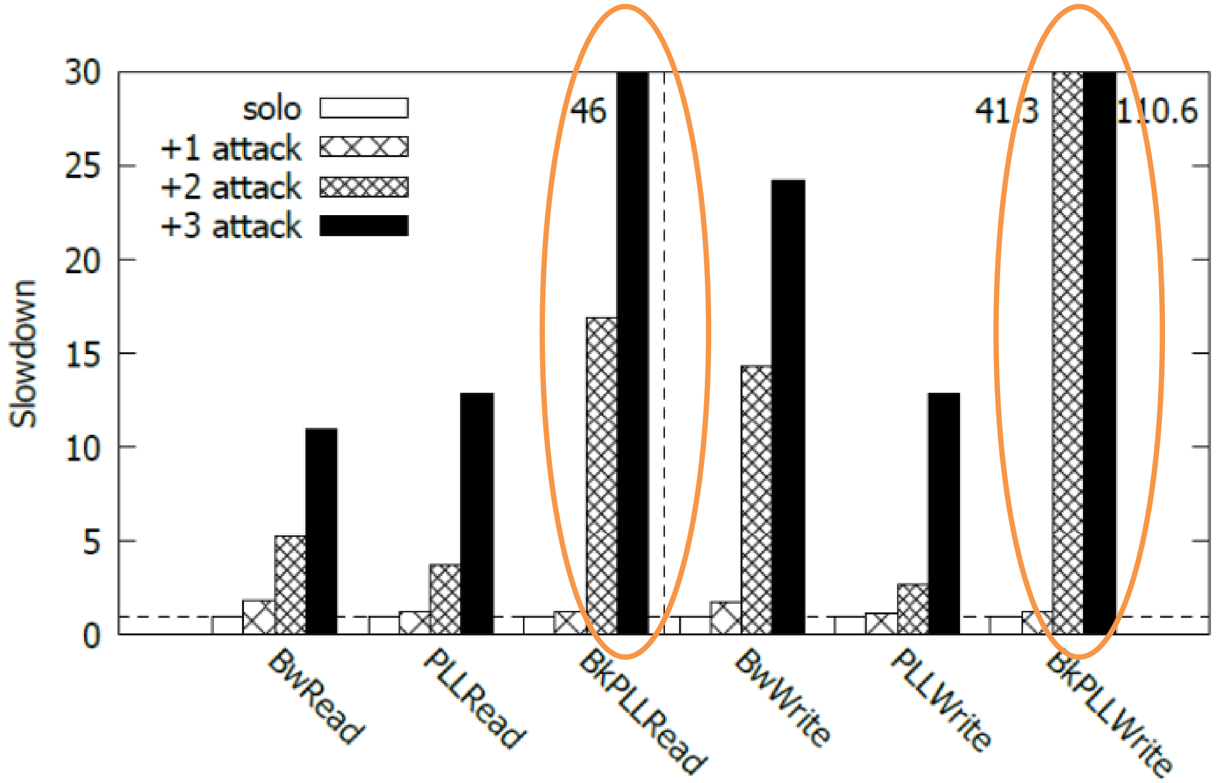
PLLRead

```
static int* list[MAX_MLP];
static int next[MAX_MLP];

for (int64_t i = 0; i < iter; i++) {
      switch (mlp) {
      case MAX_MLP:
      .
      .
      case 2:
            list[1][next[1]+1] =
                  0xff;
            next[1] =
                  list[1][next[1]];
            /* fall-through */
      case 1:
            list[0][next[0]+1] =
                  0xff;
            next[0] =
                  list[0][next[0]];
      }
}
```

PLLWrite

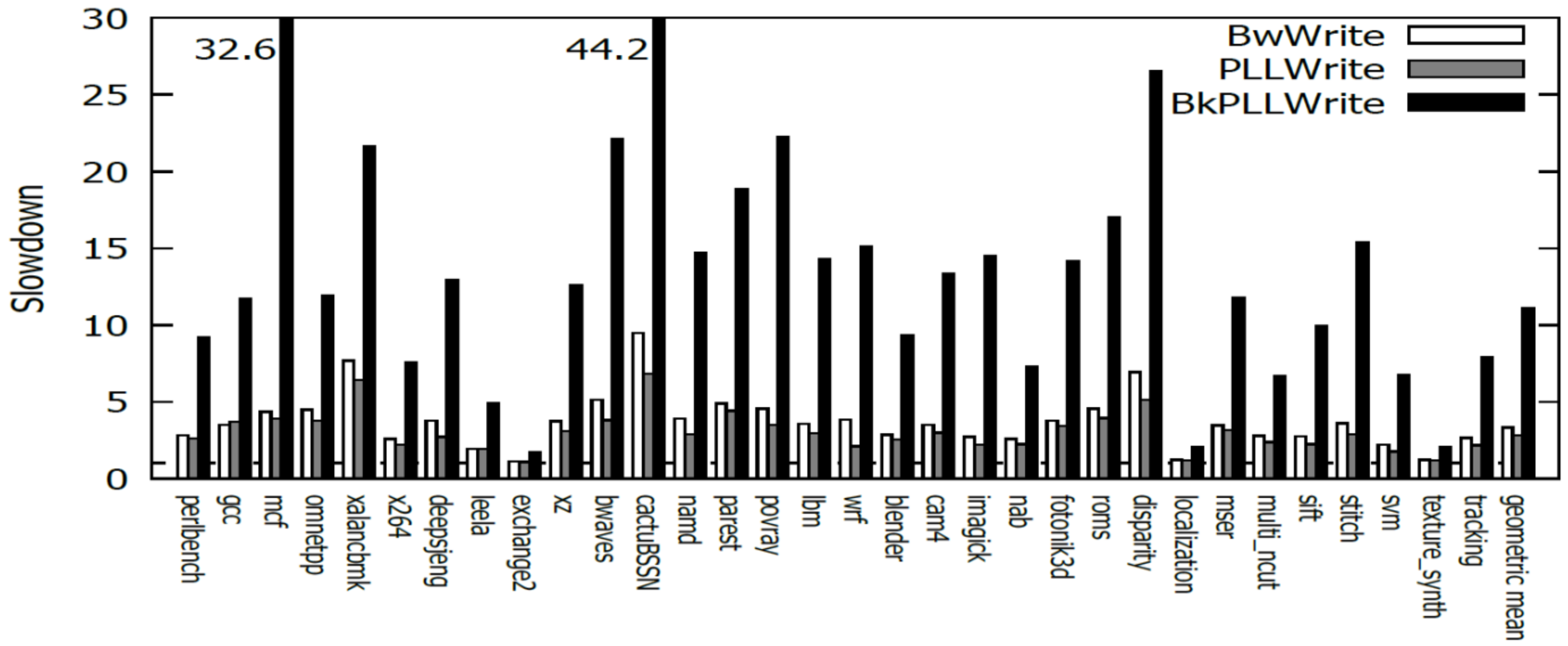
Evaluation Results (Synthetic)





- Memory-aware attacks (BkPLLRead/BkPLLWrite) are much more effective than baselines (BwRead/BwWrite)

Evaluation Results (SPEC2017)



- Memory-aware attacks outperforms baselines

Summary

- DoS attacks are more effective when attacker’s memory requests are processed slowly
- We developed memory-aware DoS attacks that target a subset of DRAM banks
- Evaluation results show significantly improved attack efficiency (more victim slowdown) on the tested embedded computing platforms

BRU: Bandwidth Regulation Unit for Real-Time Multicore Processors

Farzad Farshchi[§], Qijing Huang[¶], Heechul Yun[§]

[§]University of Kansas, [¶]University of California, Berkeley

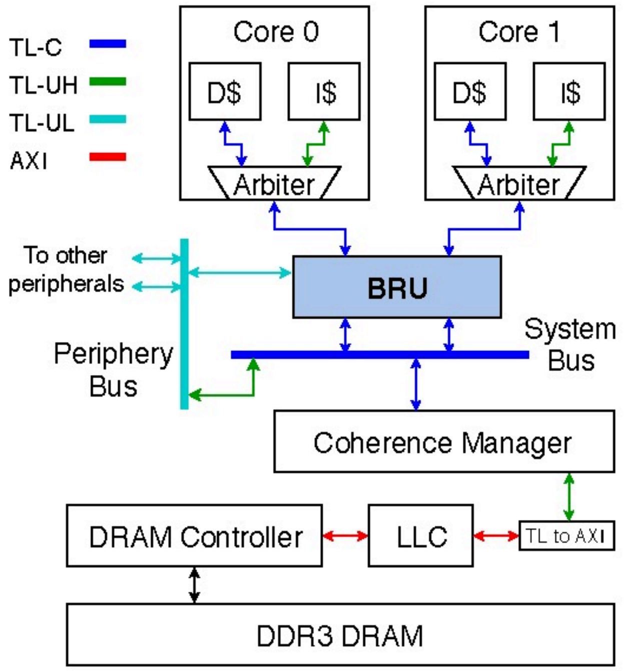
IEEE Intl. Conference on Real-Time and Embedded Technology and Applications Symposium (RTAS), April 2020

Motivation

- DoS attacks are possible due to unregulated access to the shared resources
- Software regulation mechanisms exist, but suffer high overhead [Yun+,2013]
- We need simple, low overhead mechanism to regulate access to shared resources

* Yun et al., “MemGuard: Memory Bandwidth Reservation System for Efficient Performance Isolation in Multi-core Pl atforms.” In *RTAS*, 2013

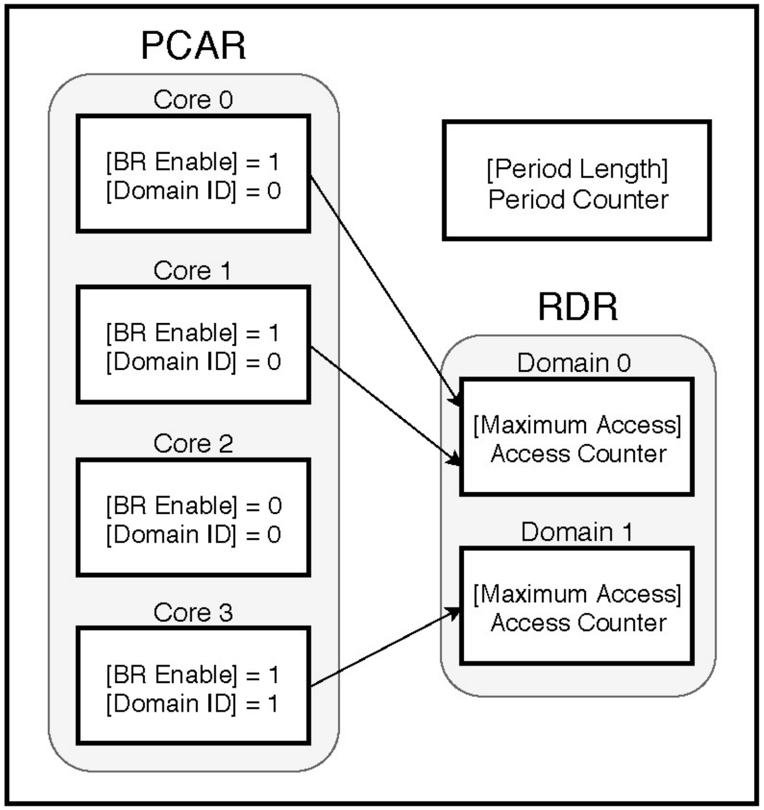
Bandwidth Regulation Unit (BRU)



- Regulate per-core/group memory bandwidth
- Drop-in addition to existing processor design

Bandwidth Regulation Unit (BRU)

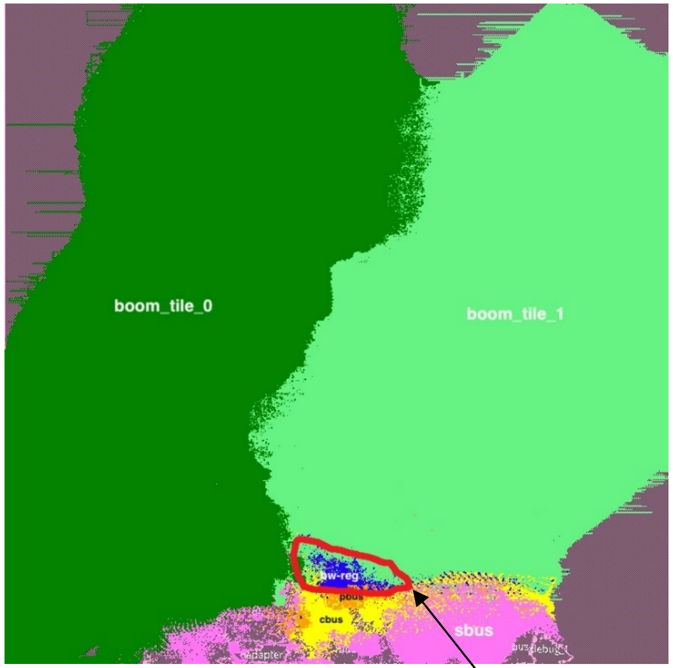
- Access regulation
  - Regulate cache misses
- Writeback regulation
  - Regulate cache write-back
- Group regulation
  - Multiple cores can be regulated as a group



Bandwidth Regulation Unit (BRU)

Dual-core BOOM with BRU

- BOOM: high-performance out-of-order RISC-V core
- Cadence synthesis result at 7nm node
- Less than 2% impact on max. frequency
- Less than 0.2% space overhead



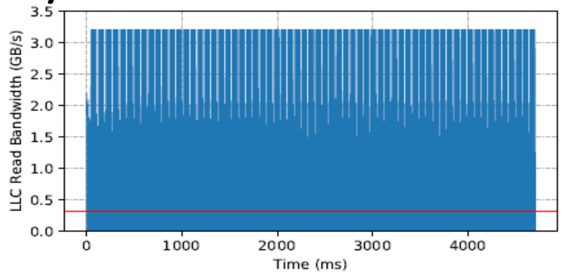
DUAL-CORE BOOM CHIP AREA BREAKDOWN

Modules	Area (μm^2)	Ratio
BRU	4,669	0.19%
Boom Core $\times$ 2	2,309,681	92.41%
Others (System Bus, Manager, etc.)	184,950	7.40%
Total	2,499,300	100%

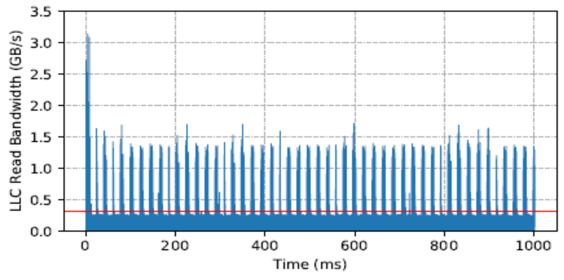
KU THE UNIVERSITY OF KANSAS

Effects of BRU

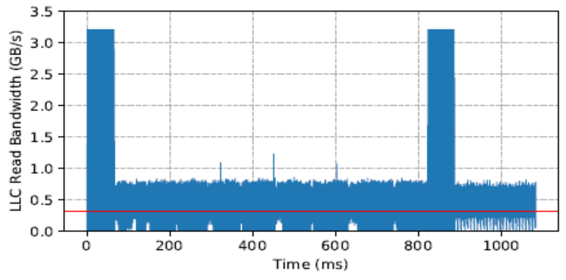
w/o BRU



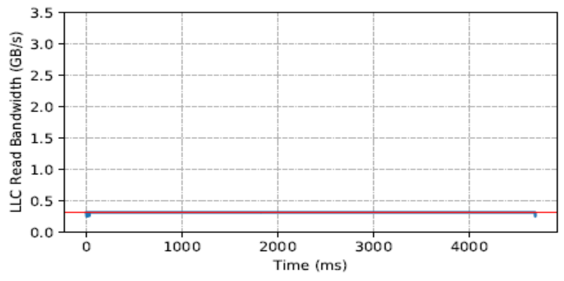
(a) disparity

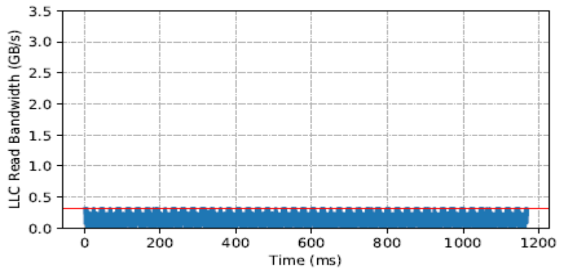


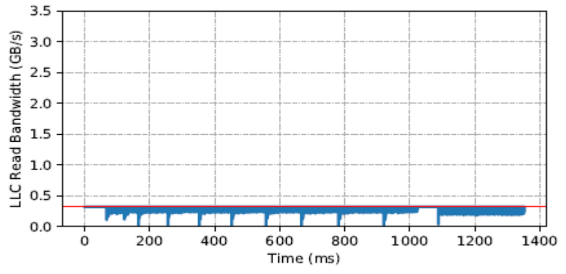
(b) localization



(c) svm







w/ BRU regulation (@320MB/s budget, 100ns period)

- BRU = MemGuard in hardware + alpha

Summary

- BRU
  - A synthesizable hardware IP that regulates memory traffic at the source (cores)
  - Demonstrates the feasibility of fast AND predictable processors
- Future work
  - Accelerator regulation support
  - More software/hardware co-design

Conclusion

- Micro-architectural attacks are serious threats on modern computing platforms
  - Can leak secret (confidentiality)
  - Can alter data (integrity)
  - Can affect real-time performance (correctness)
- We have developed new attacks and effective defense mechanisms
- Fast and secure computing is possible with cross-layer collaborative approaches

Thank You!

Questions?