IChannels

Exploiting Current Management Mechanisms to Create Covert Channels in Modern Processors

Jawad Haj-Yahya

Jeremie S. Kim *A. Giray Yağlıkçı* *Ivan Puddu* *Lois Orosa*

Juan Gómez Luna *Mohammed Alser* *Onur Mutlu*

SAFARI ETH zürich

Executive Summary

Problem: Current management mechanisms throttle instruction execution and adjust voltage/frequency to accommodate power-hungry instructions (PHIs).
These mechanisms may compromise a system's confidentiality guarantees

Goal:
1. Understand the throttling side-effects of current management mechanisms
2. Build high-capacity covert channels between otherwise isolated execution contexts
3. Practically and effectively mitigate each covert channel

Characterization: Variable execution times and frequency changes due to running PHIs
We observe five different levels of throttling in real Intel systems

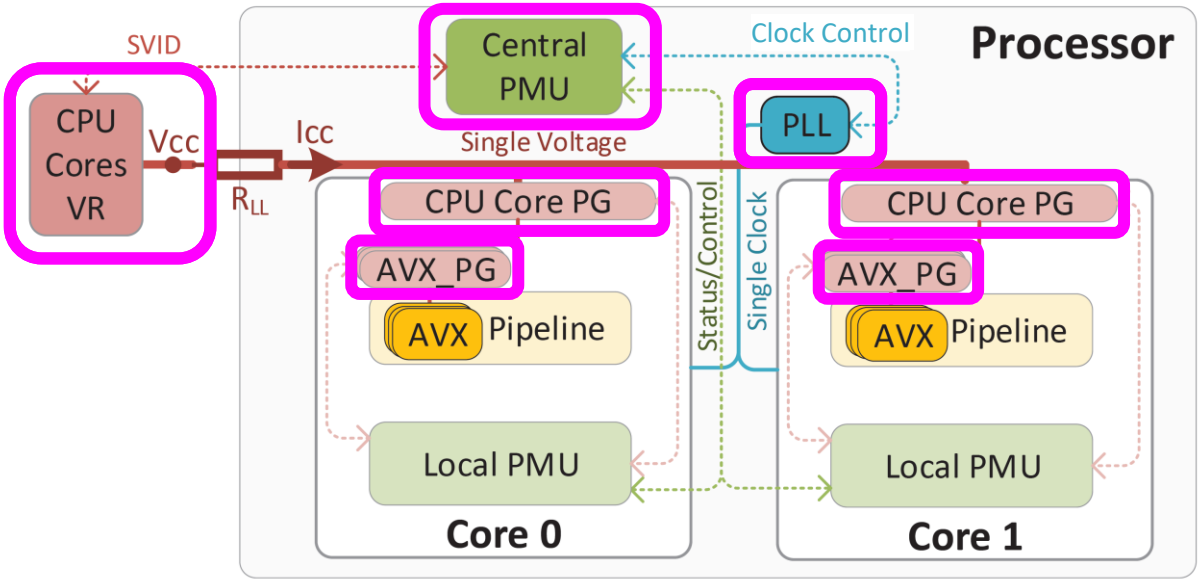
IChannels: New covert channels that exploit side-effects of current management mechanisms
- On the same hardware thread
- Across co-located Simultaneous Multi-Threading (SMT) threads
- Across different physical cores

Evaluation: On three generations of Intel processors, IChannels provides a channel capacity
- 2× that of PHIs' variable latency-based covert channels
- 24× that of power management-based covert channels

Presentation Outline

1. **Overview of Client Processor Architectures**
2. Motivation and Goal
3. Throttling Characterization
4. IChannels Covert Channels
   I. IccThreadCovert – on the same hardware thread
   II. IccSMTcovert – across co-located SMT threads
   III. IccCoresCovert – across different physical cores
5. Evaluation
6. Conclusion

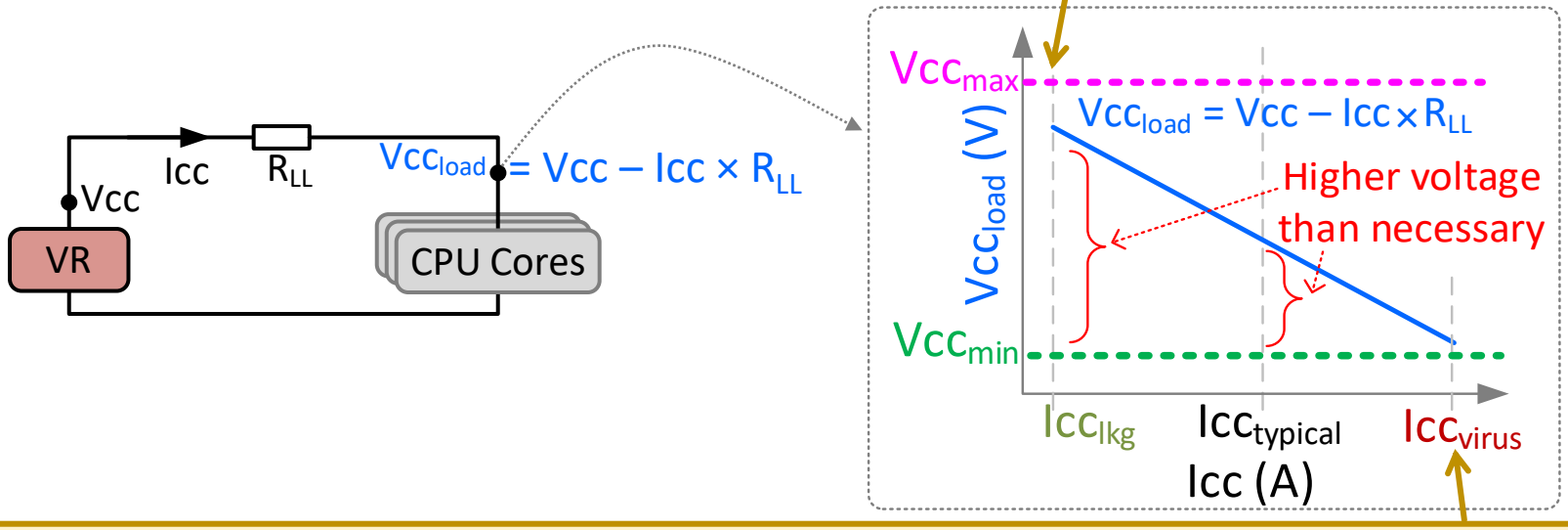
Overview of Client Processor Architectures



- In many recent processors (e.g., Intel Coffee Lake, Cannon Lake), CPU cores:
  - Share the same voltage regulator (VR) and clock domain
- A central power management unit (PMU) controls:
  - The VR using an off-chip serial voltage identification (SVID) interface
  - The clock phase-locked loop (PLL) using an on-chip interface
- Each CPU core has a power-gate (PG) for the entire core
  - Each SIMD unit (e.g., AVX-256, AVX-512) has a separate PG

Load Voltage and Voltage Guardband

Below the maximum operational voltage (Vcc_{max}) under the lightest load (leakage, Icc_{lkg})



Above the minimum functional voltage (Vcc_{min}) under the most intensive load (power-virus, Icc_{virus})

- The relationship between load voltage (Vcc_{load}), supply voltage (Vcc) and current (Icc) under a given system impedance (R_{LL}) is : $Vcc_{load} = Vcc - Icc \times R_{LL}$
- The PMU adds voltage guardband to Vcc to a level that keeps Vcc_{load} within limits
- For loads with current lower than Icc_{virus}, the voltage drop ($Icc \times R_{LL}$) is smaller than when running a power-virus
  - Results in a higher load voltage Vcc_{load} than necessary
  - Leading to a power loss that increases quadratically with the voltage level

Presentation Outline

1. Overview of Client Processor Architectures
2. **Motivation and Goal**
3. Throttling Characterization
4. IChannels Covert Channels
   I. IccThreadCovert – on the same hardware thread
   II. IccSMTcovert – across co-located SMT threads
   III. IccCoresCovert – across different physical cores
5. Evaluation
6. Conclusion

Motivation: Limitations of Prior Work

- **NetSpectre** [Schwarz+, ESORICS 2019] exploits the variable execution times of PHIs to create a covert channel. NetSpectre has three limitations:
  - Established only between two execution contexts on the same hardware thread
  - Uses only a single-level throttling side-effect (thread is throttled/unthrottled)
  - Does not identify the true source of throttling
    - Hypothesizes that the throttling is due power-gating of the PHI execution units

- **TurboCC** [Kalmbach+, arXiv 2020] exploits the core frequency throttling when executing PHIs to create cross-core covert channel. TurboCC has two limitations:
  - Focuses only on the slow (milliseconds) side-effect of frequency throttling that happens when executing PHIs at only Turbo frequencies
  - Does not uncover the real reason behind the vulnerability
    - Hypothesizes that the frequency throttling is due to thermal management

Motivation: Limitations of Prior Work

- **NetSpectre** [Schwarz+, ESORICS 2019] exploits the variable execution times of PHIs to create a covert channel. NetSpectre has three limitations:
  - Established only between two execution contexts on the same hardware thread
  - Uses only a single-level throttling side-effect (thread is throttled/unthrottled)
  - Does not identify the true source of throttling
    - Hypothesizes that the throttling is due power-gating of the PHI execution units

Recent works propose limited covert channels and use inaccurate observations

- ng
  - happens when executing PHIs at Turbo frequencies
  - Does not uncover the real reason behind the vulnerability
    - Hypothesizes that the frequency throttling is due to thermal management

Goal

Our goal in this work is to:

1. Experimentally understand the throttling side-effects of current management mechanisms in modern processors to gain several deep insights into how these mechanisms can be abused by attackers

2. Build high-capacity covert channels, IChannels, between otherwise isolated execution contexts located
   - On the same hardware thread
   - Across co-located Simultaneous Multi-Threading (SMT) threads
   - Across different physical cores

3. Practically and effectively mitigate covert channels caused by current management mechanisms

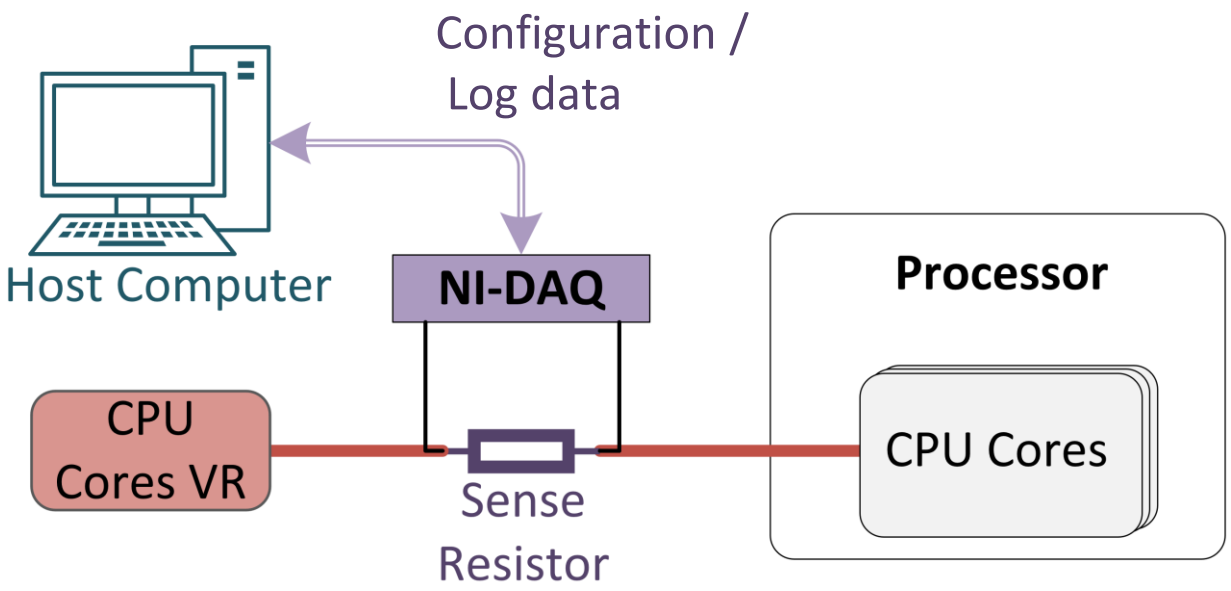
Presentation Outline

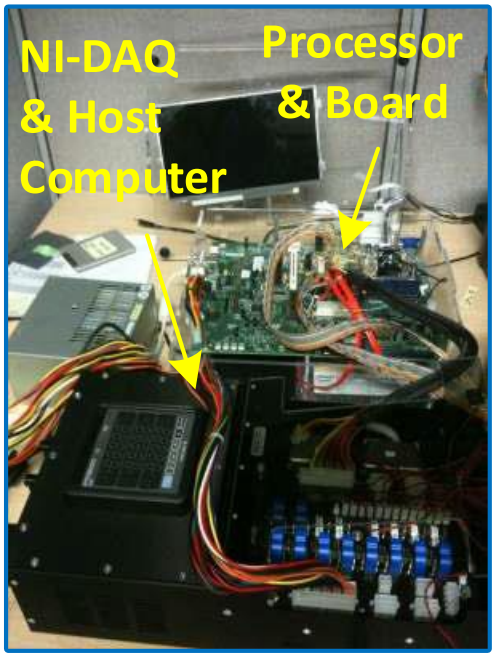
1. Overview of Client Processor Architectures
2. Motivation and Goal
3. **Throttling Characterization**
4. IChannels Covert Channels
   I. IccThreadCovert – on the same hardware thread
   II. IccSMTcovert – across co-located SMT threads
   III. IccCoresCovert – across different physical cores
5. Evaluation
6. Conclusion

Experimental Methodology

- We experimentally study three modern Intel processors
  - Haswell, Coffee Lake, and Cannon Lake

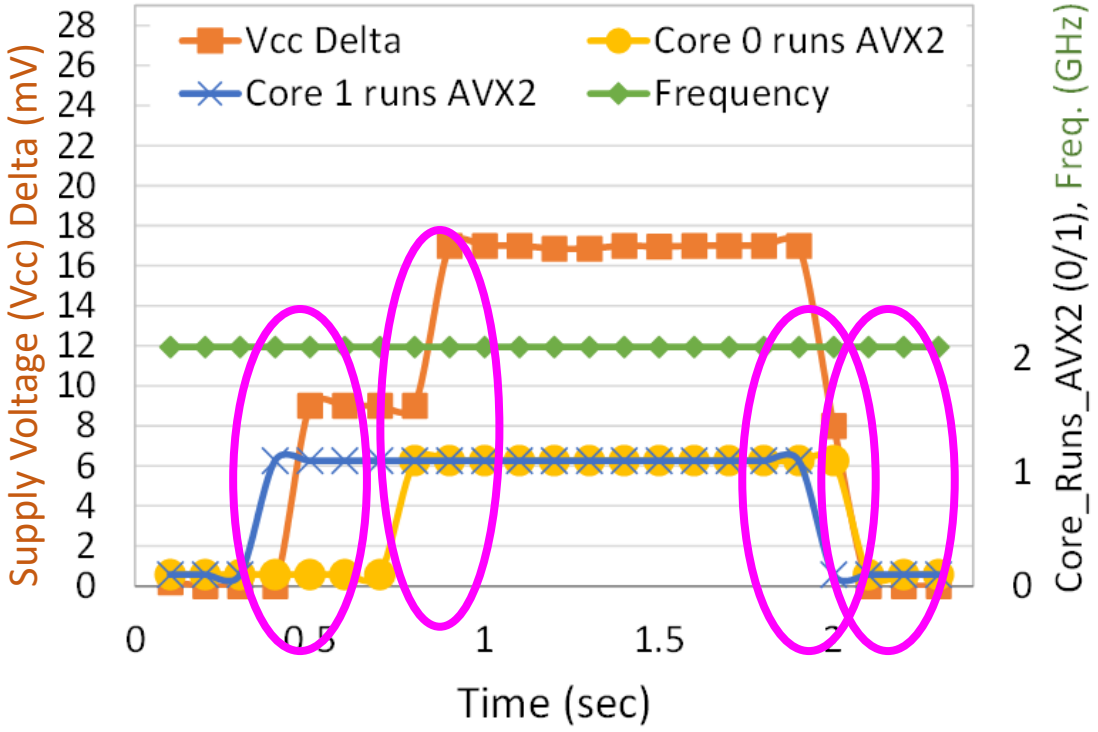
- We measure voltage and current using a Data Acquisition card (NI-DAQ)





Voltage Emergency Avoidance Mechanism

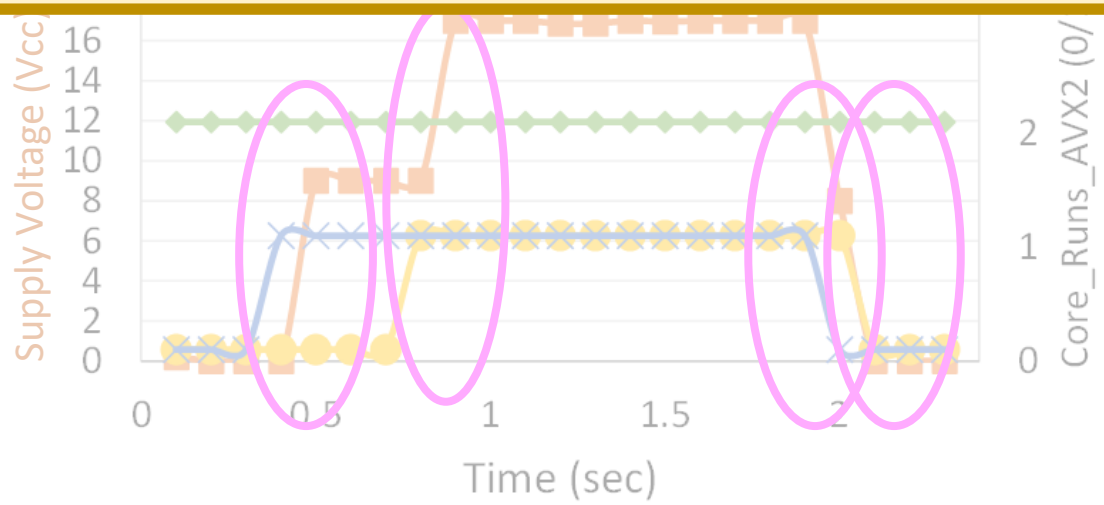
- We study the impact of Power-Hungry Instructions (PHIs) on the CPU core supply voltage (Vcc)
- We track the Vcc change during an experiment on a two-core Coffee Lake system executing code that includes PHI (AVX2) phases
- Vcc increases once a CPU core begins executing AVX2 instructions
  - The more cores executing AVX2 instructions, the higher the Vcc



Voltage Emergency Avoidance Mechanism

- We study the impact of Power-Hungry Instructions (PHIs) on the CPU core supply voltage (Vcc)
- We track the Vcc change during an experiment on a two-core Coffee Lake system executing code that includes PHI (AVX2) phases
- Vcc increases once a CPU core begins executing AVX2 instructions

Voltage emergency avoidance mechanism prevents the core voltage from dropping below the minimum operational voltage limit when executing PHIs



Icc_{max} and Vcc_{max} Limit Protection Mechanisms

- Systems:
  - A single-core Coffee Lake desktop CPU operating at Turbo frequencies (4.9 GHz and 4.8 GHz)
  - A two-core Cannon Lake mobile CPU operating at Turbo frequencies (3.1 GHz and 2.2 GHz)
- Workloads (Non-AVX and AVX2) while measuring current and voltage

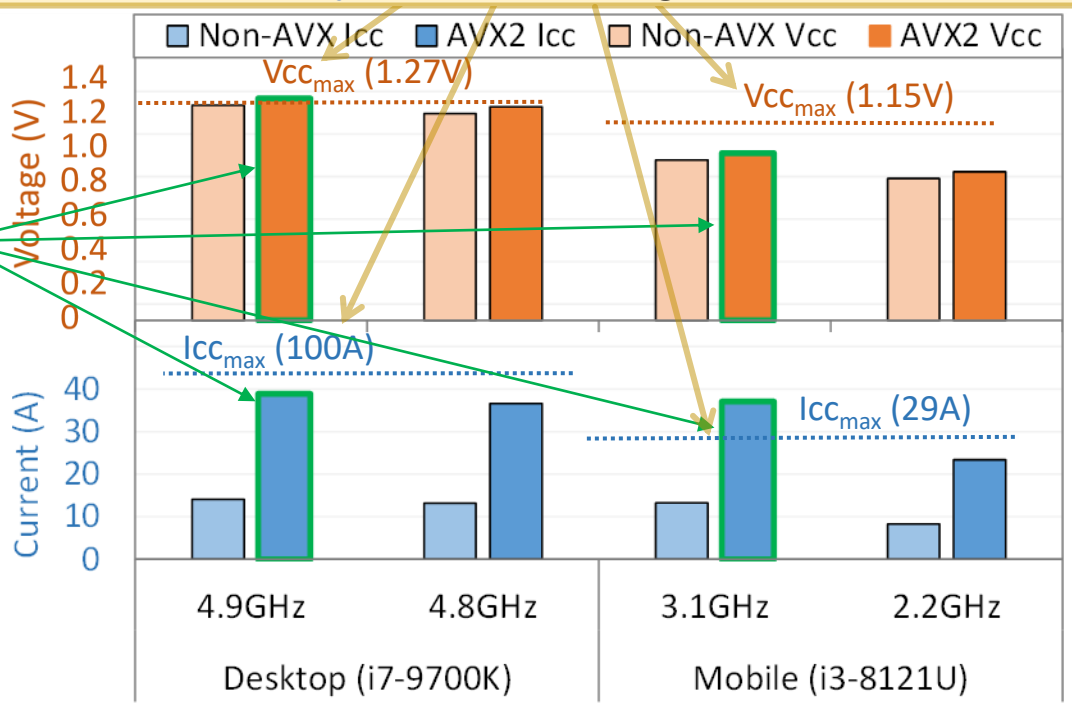
For both **desktop** frequencies, current (Icc) is below the system limit (Icc_{max})

Vcc will exceed the voltage limit (Vcc_{max}) when executing AVX2 code at a frequency of 4.9 GHz

For both **mobile** frequencies, the voltage (Vcc) is below the system limit (Vcc_{max})

Icc will exceed the current limit (Icc_{max}) when executing AVX2 code at a frequency of 3.1 GHz

The bars with green borders are **projected**



Icc_{max} and Vcc_{max} Limit Protection Mechanisms

- Systems:
  - A single-core Coffee Lake desktop CPU operating at Turbo frequencies (4.9GHz and 4.8GHz)

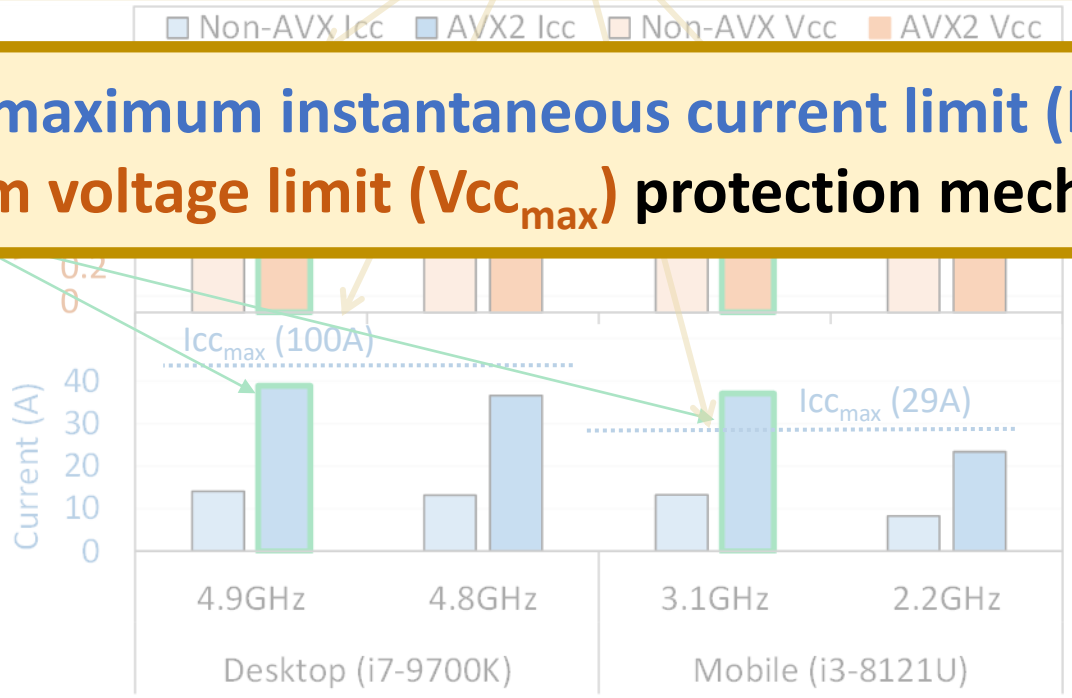
Contrary to the state-of-the-art work's hypothesis:

The core frequency reduction that directly follows the execution of PHIs at the Turbo frequency is not due to thermal management

Icc will exceed the current limit (Icc_{max}) when executing AVX2 code at a frequency of 3.1 GHz

It is due to maximum instantaneous current limit (Icc_{max}) and maximum voltage limit (Vcc_{max}) protection mechanisms

are projected



AVX Throttling is Not Due to Power Gating

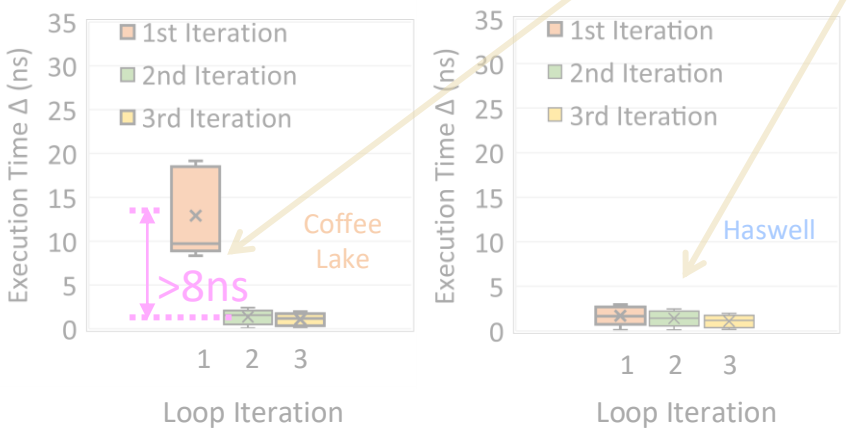
- We study the time it takes to open the AVX power-gate of Coffee Lake
  - By comparing it to Haswell system, which doesn't have an AVX power-gate
- When running AVX2 instructions in a loop
  - Consisting of 300 AVX (VMULPD) instructions that use registers

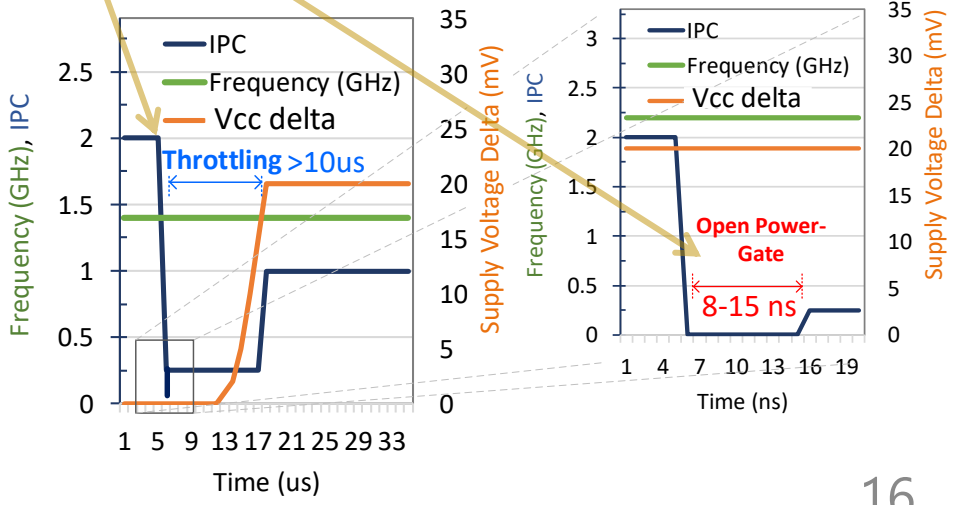
The first iteration of the loop running on Coffee Lake is > 8ns longer than the other two iterations

For the Haswell processor all iterations have nearly the same latency

AVX power-gating feature has approximately 8–15 ns of wake-up latency

About 1% of the total throttling time when executing PHIs (>10us)





AVX Throttling is Not Due to Power Gating

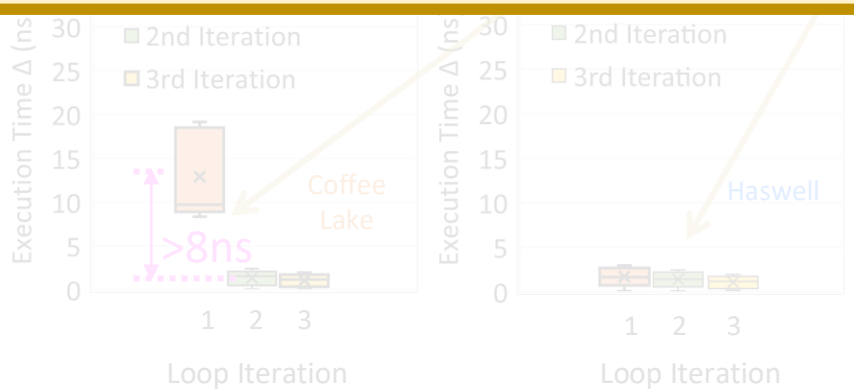
- We study the time it takes to open the AVX power-gate of Coffee Lake
  - By comparing it to Haswell system, which doesn't have an AVX power-gate
- When running AVX2 instructions in a loop

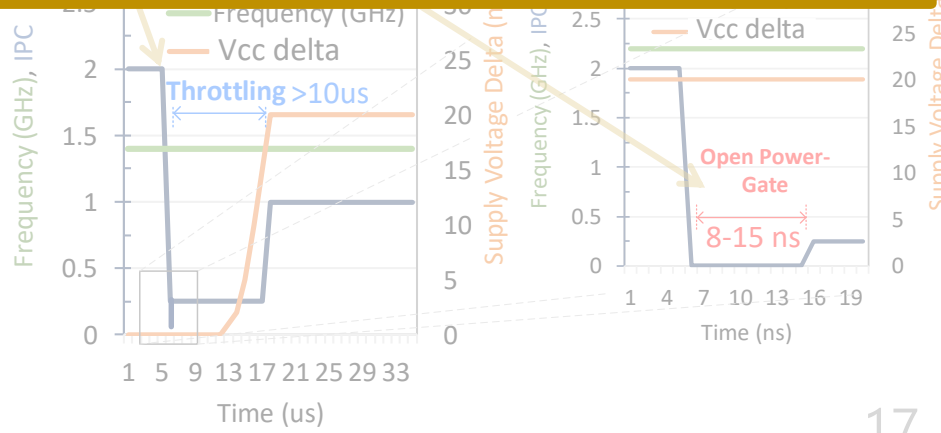
Contrary to the state-of-the-art work's hypothesis:

Power-gating AVX execution units accounts for only ~0.1% of the total throttling time observed when executing PHIs

About 1% of the total throttling time when executing PHIs (> 10us)

The majority of the throttling time is due to voltage transitions

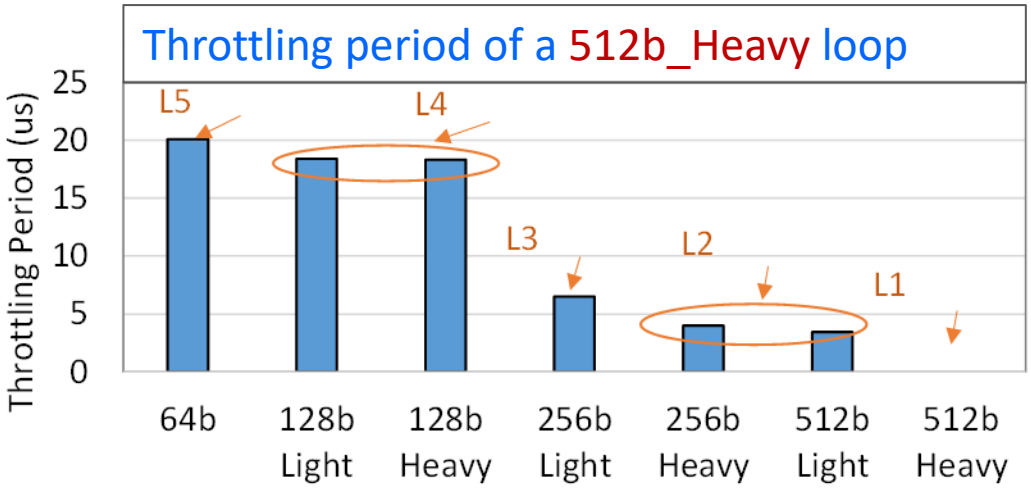




Multi-level Throttling

- We execute one of 7 instruction types in a loop followed by a 512b_Heavy loop
  - Inst0: 64b, 128b_Light, 128b_Heavy, 256b_Light, 256b_Heavy, 512b_Light, and 512b_Heavy
  - Heavy instructions: require the floating-point unit or any multiplication
- The throttling period of the 512b_Heavy loop increases when
  - The computational intensity of the instructions executed in the preceding loop decreases
- The lower the instructions' computational intensity in the preceding loop, the lower the applied voltage guardband to this instruction
  - Hence, the 512b_Heavy loop requires more time to increase the voltage to the required level
- We observe at least five throttling levels (L1–L5) corresponding to the computational intensity of instruction types

Inst0 loop
...
T0:
512b-Heavy loop

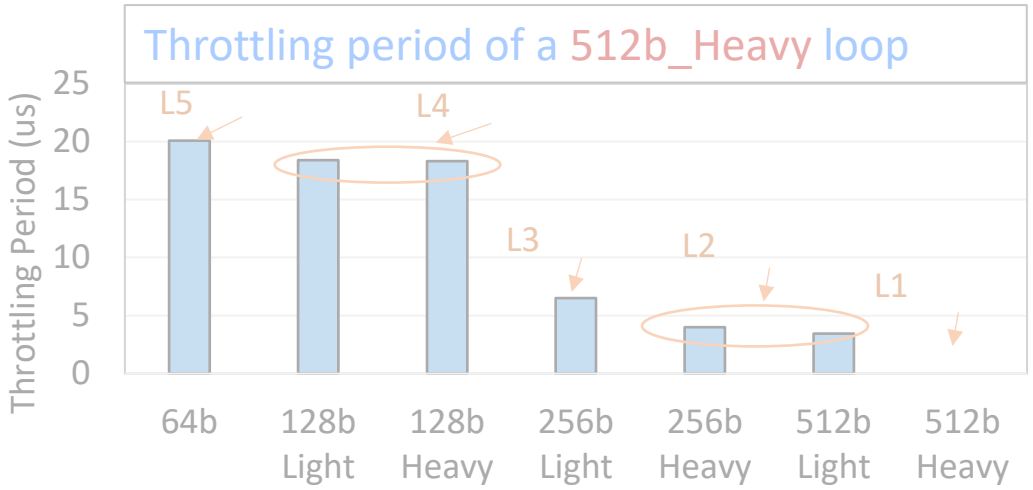


Multi-level Throttling

- We execute one of 7 instruction types in a loop followed by a 512b_Heavy loop
  - Inst0: 64b, 128b_Light, 128b_Heavy, 256b_Light, 256b_Heavy, 512b_Light, and 512b_Heavy
  - Heavy instructions: require the floating-point unit or any multiplication
- The throttling period of the 512b_Heavy loop increases when
  - The computational intensity of the instructions executed in the preceding loop decreases

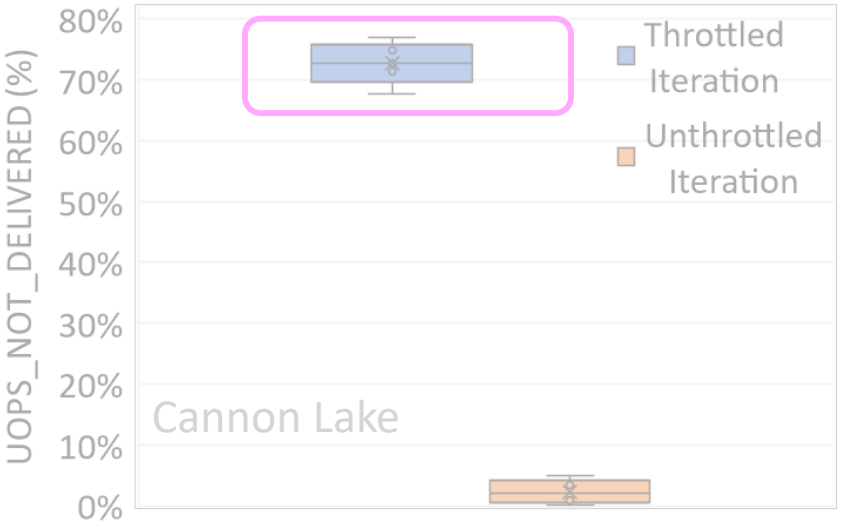
Current management mechanisms result in a multi-level throttling period depending on the computational intensity of the PHIs

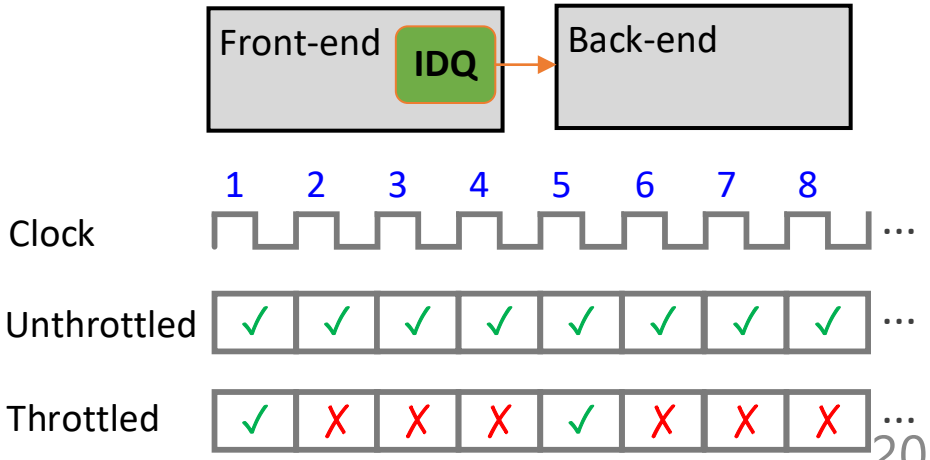
Inst0 loop
...
T0:
512b-Heavy loop



Throttling Affects SMT Threads

- We study the source of throttling and its microarchitectural impact
- We track the number of micro-operations (uops) that the core pipeline delivers from the front-end to the back-end during throttled and non-throttled AVX2 loops
- The front-end does not deliver any uop in approximately three-quarters (~75%) of the core cycles even though the back-end is not stalled
- The core uses a throttling mechanism that limits the number of uops delivered from the front-end to the back-end during a certain time window
- We found that this throttling mechanism affects both threads in Simultaneous Multi-Threading (SMT)





Throttling Affects SMT Threads

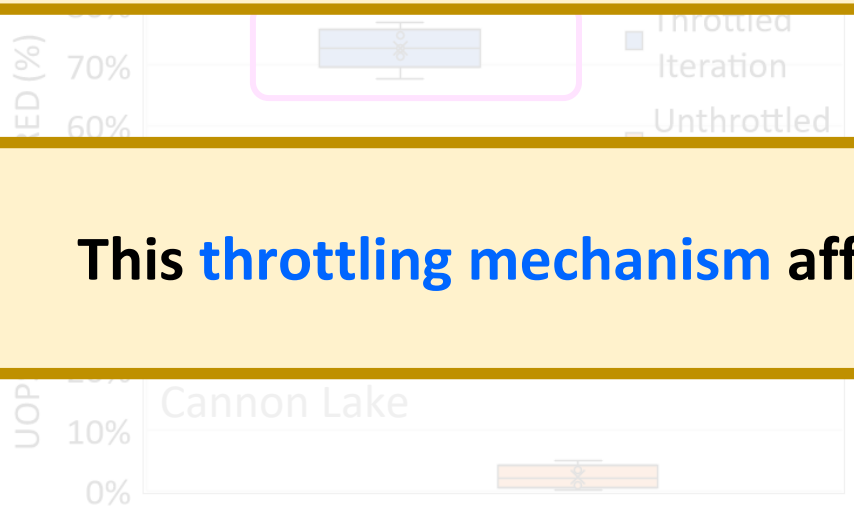
- We study the source of throttling and its microarchitectural impact

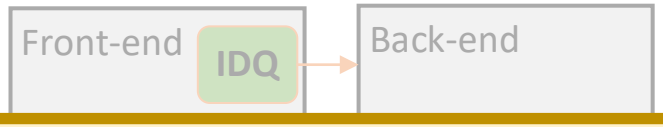
Contrary to the state-of-the-art work's hypothesis:

The 4× core IPC reduction that directly follows the execution of PHIs is not due to reduced core clock frequency of 4×

- The core uses a throttling mechanism that limits the number of uops delivered from the front-end to the back-end during a certain time window

It is rather because the core blocks the front-end to back-end uop delivery during 75% of the time





This throttling mechanism affects both threads in an SMT Core

Presentation Outline

1. Overview of Client Processor Architectures

2. Motivation and Goal

3. Throttling Characterization

4. **IChannels Covert Channels**
   - I. IccThreadCovert – on the same hardware thread
   - II. IccSMTcovert – across co-located SMT threads
   - III. IccCoresCovert – across different physical cores

5. Evaluation

6. Conclusion

IChannels Covert Channels

- Threat model consists of two malicious user-level attacker applications, sender and receiver, which cannot communicate through overt channels
- We build three high-throughput covert channels between sender and receiver that exploit throttling side-effects of current management mechanisms
  - On the same hardware thread
  - Across SMT threads, and
  - Across cores
- Each covert channel sends 2 bits from Sender to Receiver in every transaction
  - Each covert channel should wait for reset-time (~650us) before starting a new transaction
  - We demonstrate the covert channels on real Intel Coffee Lake and Cannon Lake system

Sender

```
case (send_bits[i+1:i])
  00: 128b_Heavy loop()   //L4
  01: 256b_Light_loop()   //L3
  10: 256b_Heavy_loop()   //L2
  11: 512b_Heavy_loop()   //L1
```

Receiver

```
start = rdtsc
if (same-thread) 512b_Heavy_loop()
if (across-SMT)   64b_loop()
if (across-cores) 128b_Heavy_loop()
TP = rdtsc - start
case ( TP )
  L4_range: received_bits[1:0] = 00
  L3_range: received_bits[1:0] = 01
  L2_range: received_bits[1:0] = 10
  L1_range: received_bits[1:0] = 11
```

Covert Channel 1: IccThreadCovert (1/2)

- **IccThreadCovert** covert channel exploits the side effect of **Multi-Throttling-Thread**
- **Multi-Throttling-Thread:** Executing an instruction with high computational intensity results in a throttling period proportional to the difference in voltage requirements of
  - The currently and previously executing instructions

When 512b_Heavy loop is executed, it is first throttled (IPC = 1/4) while ramping the Vcc to accommodate 512b_Heavy

Inst0 loop is throttled (IPC=1/4) while ramping the voltage (Vcc)

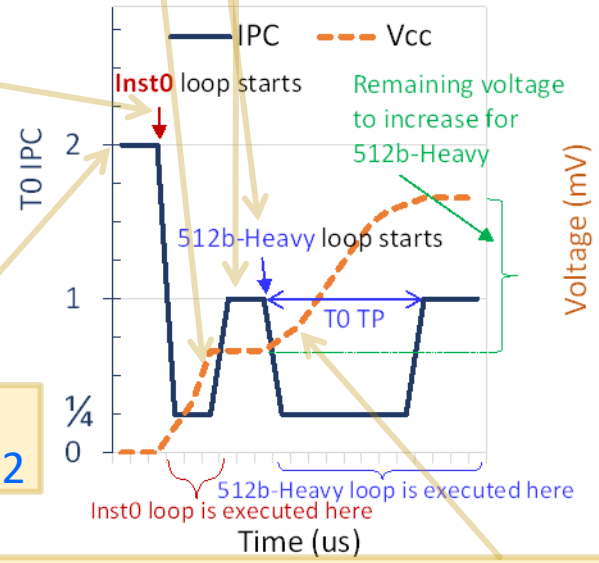
Once the target Vcc is reached, the throttling is stopped (IPC=1)

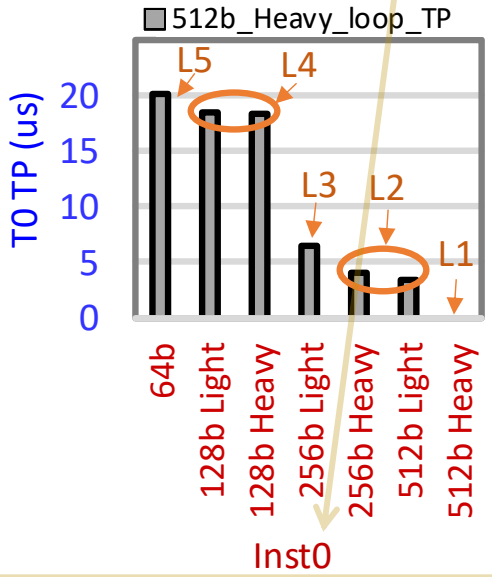
T0 throttling period (TP) dependent on the computational intensity of Inst0 loop

Inst0 loop starts executing with IPC=1

...
T0:

Executes scalar instruction with IPC=2





The remaining voltage required to execute a 512b_Heavy instruction depends on the previous Vcc level that was reached when Inst0 loop was executed

Covert Channel 1: IccThreadCovert (2/2)

Sender

```
case (send_bits[i+1:i])
  00: 128b_Heavy loop()   //L4
  01: 256b_Light_loop()   //L3
  10: 256b_Heavy_loop()   //L2
  11: 512b_Heavy_loop()   //L1
```

Receiver

```
start = rdtsc
if (same-thread) 512b_Heavy_loop()
if (across-SMT)  64b_loop()
if (across-cores) 128b_Heavy_loop()
TP = rdtsc - start
case ( TP )
  L4_range: received_bits[1:0] = 00
  L3_range: received_bits[1:0] = 01
  L2_range: received_bits[1:0] = 10
  L1_range: received_bits[1:0] = 11
```

- **IccThreadCovert** exploits the **Multi-Throttling-Thread** side-effect to build a covert channel between Sender and Receiver:
- The Sender executes a PHI loop with a computational intensity level (L1–L4) depending on the values of two secret bits it wants to send
- The Receiver can infer the two bits sent by the Sender based on the measured TP of the 512b_Heavy loop
  - The higher the power required by the PHI loop executed by the Sender, the shorter the TP experienced by the Receiver will be

Covert Channel 2: IccSMTcovert (1/2)

- **IccSMTcovert** covert channel exploits the side effect of **Multi-Throttling-SMT**
- **Multi-Throttling-SMT:** when a thread is throttled due to executing PHIs, the co-located SMT thread is also throttled
  - We discover that co-located hardware threads are throttled together because the throttling mechanism in the core pipeline blocks the front-end to back-end interface during three-quarters of the TP for the entire core

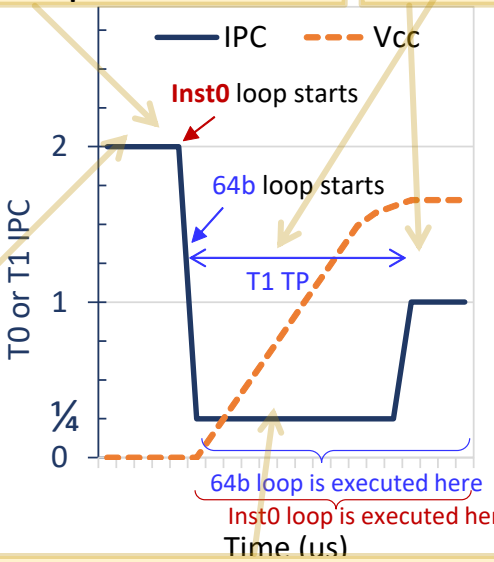
T1 throttling period (TP) depends on the computational intensity of Inst0 (executed by T0), which determines Vcc level to which the processor needs to increase the supply voltage

T0 starts executing Inst0 loop with IPC=1
T1 starts executing 64b loop with IPC=1

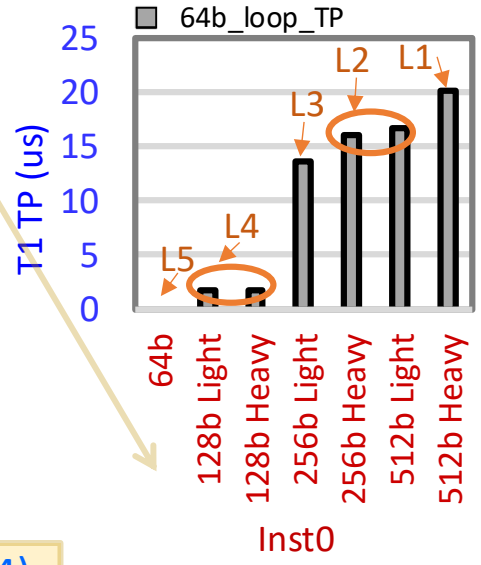
Once the target Vcc is reached, the throttling is stopped (IPC = 1)

T0 (Core0):
...

T1 (Core0):
...
...



Voltage (mV)



Threads Execute scalar instructions with IPC=2

T0 and T1 loops are throttled (IPC=1/4) while ramping the voltage (Vcc)

Covert Channel 2: IccSMTcovert (2/2)

Sender

```
case (send_bits[i+1:i])
  00: 128b_Heavy loop()  //L4
  01: 256b_Light_loop()  //L3
  10: 256b_Heavy_loop()  //L2
  11: 512b_Heavy_loop()  //L1
```

Receiver

```
start = rdtsc
if (same-thread) 512b_Heavy_loop()
if (across-SMT)  64b_loop()
if (across-cores) 128b_Heavy_loop()
TP = rdtsc - start
case ( TP )
  L4_range: received_bits[1:0] = 00
  L3_range: received_bits[1:0] = 01
  L2_range: received_bits[1:0] = 10
  L1_range: received_bits[1:0] = 11
```

- **IccSMTcovert** exploits the **Multi-Throttling-SMT** side-effect to build a covert channel between Sender and Receiver:
- The Sender executes a **PHI loop** with a computational intensity level (L1–L4) depending on the values of two secret bits it wants to send
- The Receiver can infer the two bits sent by the Sender based on the measured TP of the 64b loop
  - The higher the power required by the PHI loop executed by the Sender, the higher the TP experienced by the Receiver will be

Covert Channel 3: IccCoresCovert (1/2)

- **IccCoresCovert** covert channel exploits the side effect of **Multi-Throttling-Cores**
- **Multi-Throttling-Cores:** when two cores execute PHIs at similar times, the throttling periods (TP) are exacerbated proportionally to the computational intensity of each PHI executed in each core
  - This increase in the TP is because the power management unit (PMU) waits until the voltage transition for core A to complete before starting the voltage transition for core B

T1 TP depends on the computational intensity of Inst0, which determines Vcc level to which the PMU needs to increase the supply voltage before handling T1 voltage transition

T0 and T1 loops are throttled (IPC=1/4)

T1 continues to be throttled since the PMU will not handle T1 voltage transition until T0 voltage target is reached

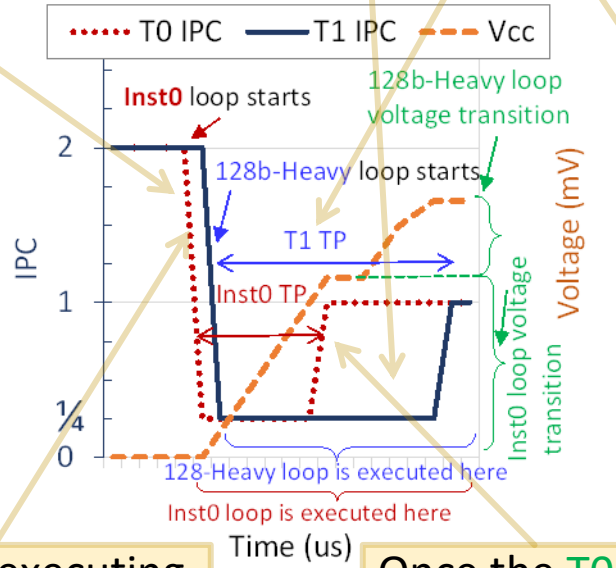
T0 (Core0):

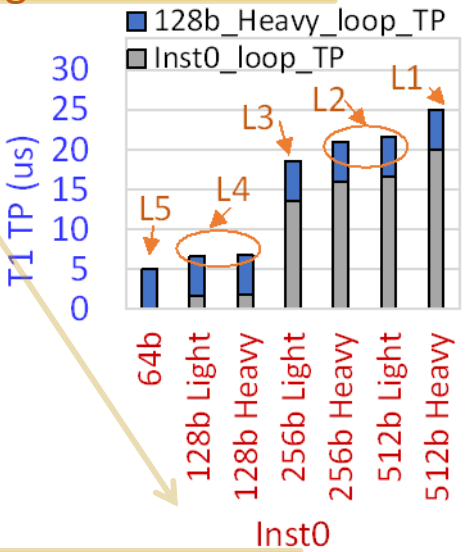
...

T1 (Core1):

...

...





T0/T1 in core0/1 starts executing Inst0/128b-Heavy loop with IPC=1

Once the T0 target Vcc is reached, T0 throttling is stopped (IPC = 1)

Covert Channel 3: IccCoresCovert (2/2)

Sender

```
case (send_bits[i+1:i])
  00: 128b_Heavy loop()   //L4
  01: 256b_Light_loop()   //L3
  10: 256b_Heavy_loop()   //L2
  11: 512b_Heavy_loop()   //L1
```

Receiver

```
start = rdtsc
if (same-thread) 512b_Heavy_loop()
if (across-SMT)  64b_loop()
if (across-cores) 128b_Heavy_loop()
TP = rdtsc - start
case ( TP )
  L4_range: received_bits[1:0] = 00
  L3_range: received_bits[1:0] = 01
  L2_range: received_bits[1:0] = 10
  L1_range: received_bits[1:0] = 11
```

- **IccCoresCovert** exploits the **Multi-Throttling-Cores** side-effect to build a covert channel between Sender and Receiver:
- The Sender executes a PHI loop with a computational intensity level (L1–L4) depending on the values of two secret bits it wants to send
- The Receiver can infer the two bits sent by the Sender based on the measured TP of the 128b_Heavy loop
  - The higher the power required by the PHI loop executed by the Sender, the higher the TP experienced by the Receiver will be

Presentation Outline

1. Overview of Client Processor Architectures
2. Motivation and Goal
3. Throttling Characterization
4. IChannels Covert Channels
   I. IccThreadCovert – on the same hardware thread
   II. IccSMTcovert – across co-located SMT threads
   III. IccCoresCovert – across different physical cores
5. **Evaluation**
6. Conclusion

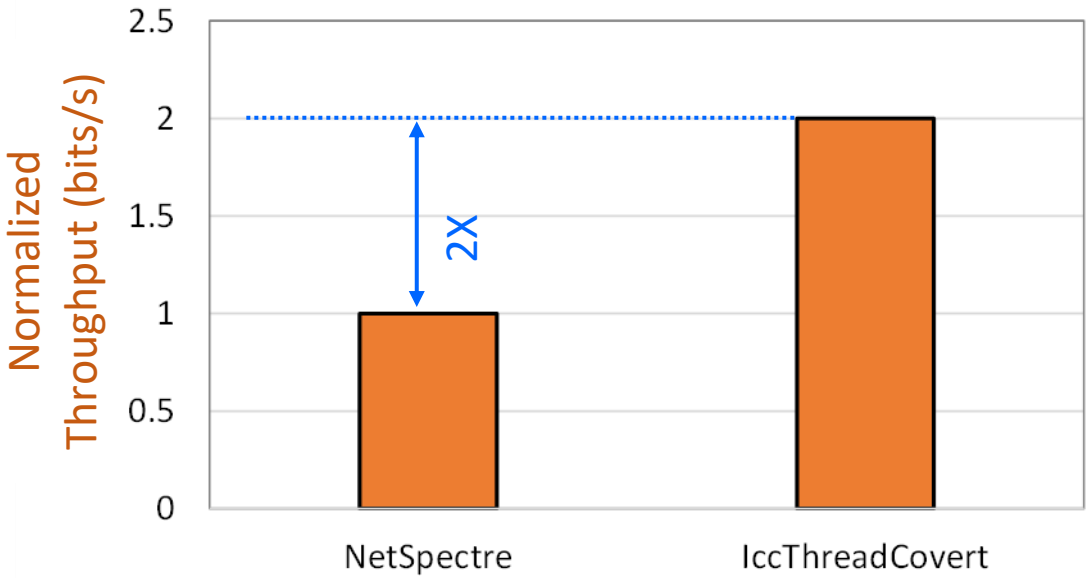
Methodology

- **Framework:** We evaluate IChannels on Coffee Lake and Cannon Lake
  - We test IccThreadCovert and IccCoresCovert on both processors, but we test IccSMTcovert only on Cannon Lake as Coffee Lake does not support SMT

- **Workloads**: Proof-of-concept codes of each of the three IChannels covert channels

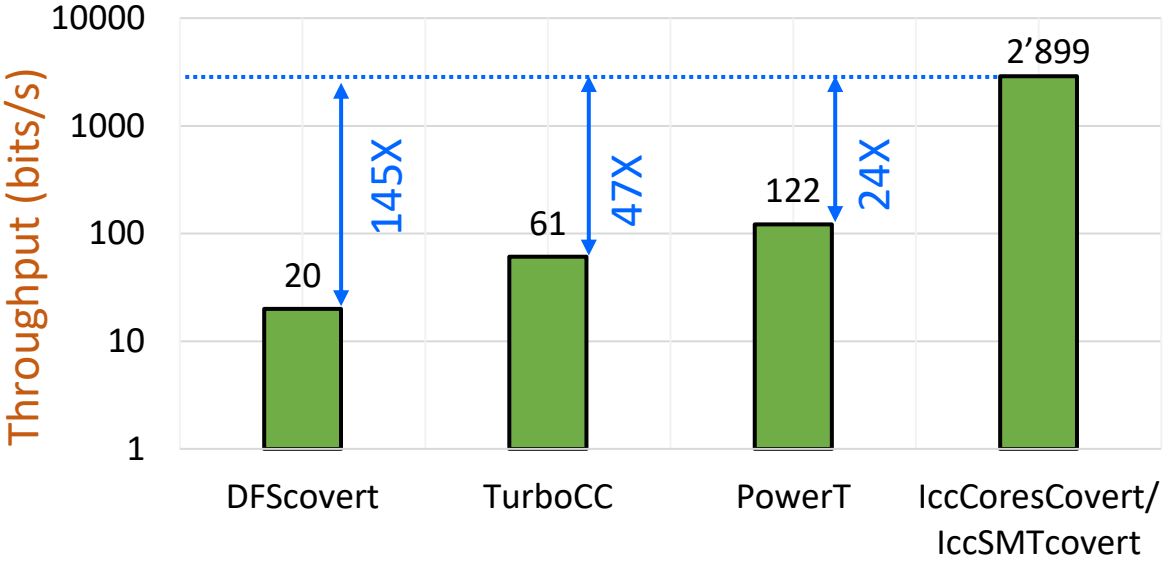
- **Comparison Points**: We compare IChannels to four recent works
  - That exploit different power management mechanisms of modern processors to build covert channels

Results – IccThreadCovert



- We compare IccThreadCovert against NetSpectre
  - The state-of-the-art work that exploits the variable latency of PHIs to create a covert channel between two execution contexts running on the same hardware thread
- The NetSpectre covert channel can send one bit per transaction,
  - IccThreadCovert covert channel can send two bits per transaction

Results – IccSMTcovert & IccCoresCovert



- We compare IccSMTcovert and IccCoresCovert against DFScovert, TurboCC and PowerT
  - The state-of-the-art works that exploit different power management mechanisms of modern processors to build covert channels across cores and SMT threads
- IccSMTcovert/IccCoresCovert throughput is 145×, 47×, and 24×
  - The throughput of DFScovert, TurboCC, and PowerT, respectively
- The three works exploit slow mechanisms (e.g., frequency/thermal changes)
  - Compared to the current management side-effects that our IChannels exploits

Other Results in the Paper

- We propose practical hardware or software techniques for the mitigation of IChannels covert channels:
  - Fast Per-core Voltage Regulators
  - Improved Core Throttling
  - New Secure Mode of Operation

Mitigation	IccThreadCovert	IccSMTcovert	IccCoresCovert	Overhead
Per-core VR	Partially	Partially	✓	11%-13% more area
Improved Throttling	✗	✓	✗	Some design effort
Secure-Mode	✓	✓	✓	4%-11% additional power

Presentation Outline

1. Overview of Client Processor Architectures
2. Motivation and Goal
3. Throttling Characterization
4. IChannels Covert Channels
   I. IccThreadCovert – on the same hardware thread
   II. IccSMTcovert – across co-located SMT threads
   III. IccCoresCovert – across different physical cores
5. Evaluation
6. **Conclusion**

Conclusion

Problem: Current management mechanisms throttle instruction execution and adjust voltage/frequency to accommodate power-hungry instructions (PHIs).
These mechanisms may compromise a system's confidentiality guarantees

Goal:
1. Understand the throttling side-effects of current management mechanisms
2. Build high-capacity covert channels between otherwise isolated execution contexts
3. Practically and effectively mitigate each covert channel

Characterization: Variable execution times and frequency changes due to running PHIs
We observe five different levels of throttling in real Intel systems

IChannels: New covert channels that exploit side-effects of current management mechanisms
- On the same hardware thread
- Across co-located Simultaneous Multi-Threading (SMT) threads
- Across different physical cores

Evaluation: On three generations of Intel processors, IChannels provides a channel capacity
- 2× that of PHIs' variable latency-based covert channels
- 24× that of power management-based covert channels

IChannels

Exploiting Current Management Mechanisms to Create Covert Channels in Modern Processors

Jawad Haj-Yahya

Jeremie S. Kim *A. Giray Yağlıkçı* *Ivan Puddu* *Lois Orosa*

Juan Gómez Luna *Mohammed Alser* *Onur Mutlu*

SAFARI ETH zürich