CCCiCC:
A Cross Core Cache-independent Covert Channel

Carl-Daniel Hailfinger[1,2], Kerstin Lemke-Rust[2], Christof Paar[3]

[1]Ruhr-University Bochum
[2]Bonn-Rhein-Sieg University of Applied Sciences
[3]Max Planck Institute for Cyber Security and Privacy

2019-11-12

Hochschule
Bonn-Rhein-Sieg

RUB RUHR-UNIVERSITÄT BOCHUM

Microarchitectural constraints and solutions

- Constraints
  - Resources are finite
  - Chip area is costly
  - Higher clock frequencies are costly
  - Present but unused resources are useless

Microarchitectural constraints and solutions

- Constraints
  - Resources are finite
  - Chip area is costly
  - Higher clock frequencies are costly
  - Present but unused resources are useless
- Optimization 'solutions'
  - Resource sharing
  - Parallelization
  - Reordering
  - Buffering/caching
  - Speculation
  - Postponing checks until after calculation
  - Powering down unused resources

Microarchitectural challenges

- Architectural expectations vs. microarchitectural implementation
- Operations leaking into the microarchitectural state (side effects)
- Programming manuals focusing on architectural properties

Measuring microarchitectural state

- Physically present hardware attack
  - Power consumption
  - Emissions
  - Physical debug interface (if available)

Measuring microarchitectural state

- Physically present hardware attack
  - Power consumption
  - Emissions
  - Physical debug interface (if available)
- Privileged internal software attack
  - Builtin performance measurement instructions
  - Builtin debug instructions

Measuring microarchitectural state

- Physically present hardware attack
  - Power consumption
  - Emissions
  - Physical debug interface (if available)
- Privileged internal software attack
  - Builtin performance measurement instructions
  - Builtin debug instructions
- Unprivileged internal software attack
  - Timing

Measuring microarchitectural state

- Physically present hardware attack
  - Power consumption
  - Emissions
  - Physical debug interface (if available)
- Privileged internal software attack
  - Builtin performance measurement instructions
  - Builtin debug instructions
- Unprivileged internal software attack
  - Timing
- Unprivileged attack over the network
  - Timing

Measuring microarchitectural state II

- Domain measurement
  - Direct measurement only within sharing domain
  - Indirect measurement requires gadget within sharing domain to transmit info outside domain

Measuring microarchitectural state II

- Domain measurement
  - Direct measurement only within sharing domain
  - Indirect measurement requires gadget within sharing domain to transmit info outside domain
- Resource measurement
  - Direct measurement only possible within same resource
  - Indirect measurement requires gadget with access/dependency in source resource and access/dependency/side effect in target resource

Microarchitectural resources

- Persistent resources
  - Cache (Spectre v1 etc.)
  - Memory ((Rowhammer))

Microarchitectural resources

- Persistent resources
  - Cache (Spectre v1 etc.)
  - Memory ((Rowhammer))
- Semi-persistent resources: high change rate
  - Buffers (MDS)
  - Registers (LazyFP)
  - Power states (NetSpectre AVX-256 power domain)

Microarchitectural resources

- Persistent resources
  - Cache (Spectre v1 etc.)
  - Memory ((Rowhammer))
- Semi-persistent resources: high change rate
  - Buffers (MDS)
  - Registers (LazyFP)
  - Power states (NetSpectre AVX-256 power domain)
- Non-persistent resources

Microarchitectural resources

- Persistent resources
  - Cache (Spectre v1 etc.)
  - Memory ((Rowhammer))
- Semi-persistent resources: high change rate
  - Buffers (MDS)
  - Registers (LazyFP)
  - Power states (NetSpectre AVX-256 power domain)
- Non-persistent resources
  - Execution engine
    - Execution port contention (PortSmash)
    - Instruction decoder contention (CCCiCCv1)

Microarchitectural resources

- Persistent resources
  - Cache (Spectre v1 etc.)
  - Memory ((Rowhammer))
- Semi-persistent resources: high change rate
  - Buffers (MDS)
  - Registers (LazyFP)
  - Power states (NetSpectre AVX-256 power domain)
- Non-persistent resources
  - Execution engine
    - Execution port contention (PortSmash)
    - Instruction decoder contention (CCCiCCv1)
  - Uncore / Globally shared non-execution resources
    - (P)RNG contention (RDRAND covert channel)
    - Global synchronization (CCCiCCv2)

Multicore vs. Multithread

- Multicore design (AMD K10)
  - L3 shared
  - Uncore shared

Multicore vs. Multithread

- Multicore design (AMD K10)
  - L3 shared
  - Uncore shared
- Hybrid multi-threaded/-core design (AMD Family 15h)
  - The above plus
  - L1 instruction cache shared
  - L2 cache shared
  - Instruction fetching shared
  - Floating point / vector execution unit shared
  - (Some variants) Instruction decoder shared

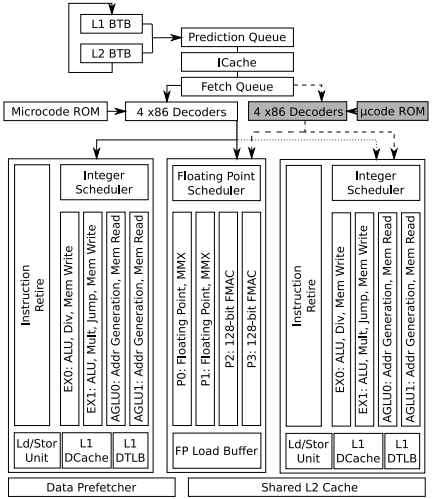
Multicore vs. Multithread

- Multicore design (AMD K10)
  - L3 shared
  - Uncore shared
- Hybrid multi-threaded/-core design (AMD Family 15h)
  - The above plus
  - L1 instruction cache shared
  - L2 cache shared
  - Instruction fetching shared
  - Floating point / vector execution unit shared
  - (Some variants) Instruction decoder shared
- Simultaneous multithreaded design (AMD Zen)
  - Everything shared

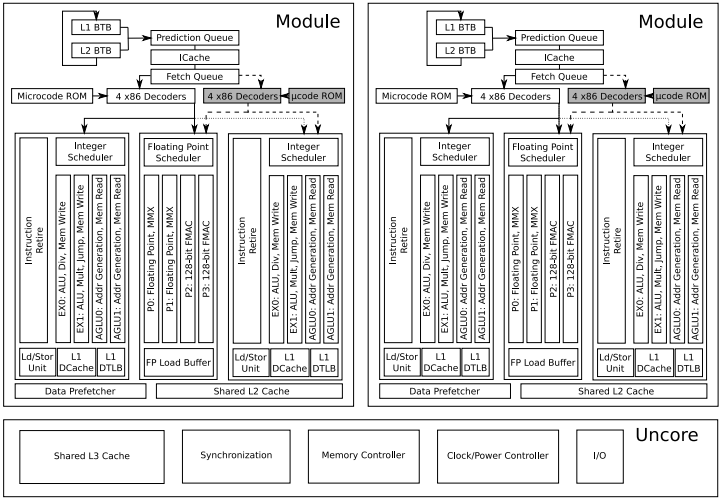
AMD Family 15h

- Bulldozer (2011)
  - Desktop: Zambezi
  - Server Opterons: Zurich, Valencia, Interlagos
- Piledriver (2012)
  - Desktop/Mobile: Trinity, Richland
  - Server Opterons: Delhi, Seoul, Abu Dhabi, Warsaw
- Steamroller (2014)
  - Desktop/Mobile: Kaveri, Godavari
- Excavator (2015/2017)
  - Desktop/Mobile: Carrizo, Bristol Ridge, Stoney Ridge
  - Server Opterons: Toronto

Piledriver microarchitecture single module



Piledriver microarchitecture two-module CPU



Attacking non-persistent resources

No speculation required

- Execution engine
  - Execution port contention (PortSmash)
  - **Instruction decoder contention (CCCiCC v1)**
- Uncore / Globally shared non-execution resources
  - (P)RNG contention (RDRAND covert channel)
  - **Global synchronization (CCCiCC v2)**

AMD Family 15h instruction decoder rules

- Shared instruction decoders serving one core per cycle in alternating order
- Microcoded instructions emitting ≥ 3 macro-ops
- ≤ 4 instructions decoded per cycle
- ≤ 4 macro-ops emitted per cycle
  - 4 instructions, 1-1-1-1 macro-ops
  - 3 instructions, 2-1-1 macro-ops
  - 2 instructions, 2-2 macro-ops
  - 1 instruction, 3+ macro-ops
  - 0 instructions if previous cycle generated > 4 macro-ops
- Leakage from other core (same module) for the 0 instruction case

Instruction decoder throughput as covert channel

- Blocking depends on macro-ops (Mops) emitted
- Instruction decoder is blocked for $\lfloor \frac{Mops-1}{4} \rfloor$ subsequent cycles
  - $\geq$ 5 Mops $\rightarrow$ 1 cycle blocked on other core
  - $\geq$ 9 Mops $\rightarrow$ 1 cycle blocked on other core, 1 cycle self blocked
  - $\geq$ 13 Mops $\rightarrow$ 2 cycles blocked on other core
- Blocking/non-blocking as 0/1 signal from sender core
- Instruction throughput measurement on receiving core

Sender instruction selection criteria

- Avoiding interaction with other parts of execution engine
  - No PortSmash by accident
  - Avoiding usage of any other shared resource
- Avoiding bottlenecks in earlier or later pipeline stages
- Avoiding other possible decoder side effects
- Short instruction
- 32-bit mode for compatibility reasons, 64-bit instructions work as well
- No trap to hypervisor
- No trap to OS
- No generated exception

Selected instructions

- NOP: 1 byte (0x90)
  - immediately resolved, no side effects, 1 Mop
- INC EAX: 1 byte (0x40)
  - sent to non-shared integer unit, no traps, no exceptions, 1 Mop
- RCL EAX, CL: 2 bytes (0xd3 0xd0)
  - sent to non-shared integer unit, no traps, no exceptions, 17 Mops
- CPUID: 2 bytes (0x0f 0xa2)
  - resolved without units, trap to hypervisor, no exceptions, 38-64 Mops
- RDTSCP: 3 bytes (0x0f 0x01 0xf9)
  - resolved without units, may trap to OS with exception, 44 Mops

Measurement code

```
measure_tight_empty_outerloop16:
        POP ESI
        PUSH ESI
        CMP ESI, EDI ; Are all measurements done
        JE out
        RDTSCP
        MOV ECX, EBX ; set counter, must do this after RDTSC because RDTSCP clobbers EC
ALIGN 32
; Run 16 NOP in a loop, ECX times
measure_tight_empty_innerloop16:
        NOP NOP NOP NOP NOP NOP NOP NOP NOP NOP NOP NOP NOP NOP NOP NOP
        LOOP measure_tight_empty_innerloop16
        MOV ESI, EAX ; back up low 32 bits of TSC
        RDTSCP
        SUB EAX, ESI ; diff low 32 bits of TSC
        MOV [EDI], EAX ; store low 32 bits of measurement in result array
        ADD EDI, 4
        JMP measure_tight_empty_outerloop16
out:
```

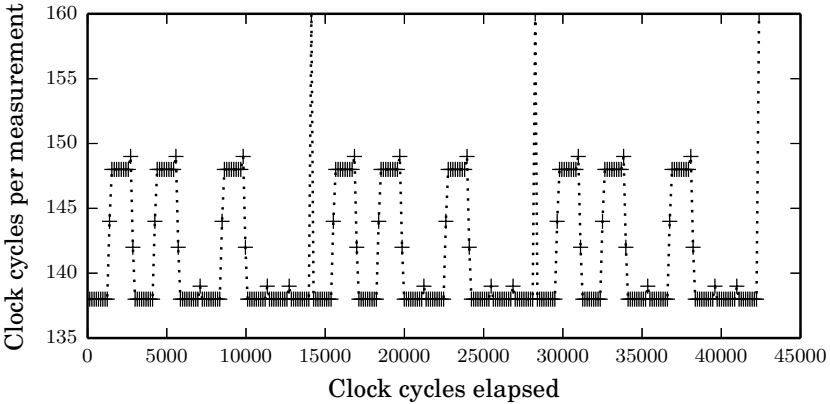
Reality check: Piledriver CPUID/NOP

Table: Clock cycles per measurement, Piledriver A6-4400M

Loops on core 1	Clock cycles on core 1 (CPUID on core 0)	Clock cycles on core 1 (NOP/INC on core 0)	difference
1	88 (99.8%)	89 (51.2%), 90 (47.1%)	1, 2
10	129 (99.7%)	129 (49.9%), 132 (49.9%)	0, 3
11	134 (99.8%)	137 (99.6%)	3
20	174 (99.6%)	220 (99.9%)	46
100	627 (47.0%), 635 (47.0%)	946 (99.9%)	319, 311
1000	5793 (90.8%)	9046 (99.6%)	3253
10000	57327 (49.4%), 58235 (46.9%)	90046 (98.8%)	32719, 31811

Proportional difference

Reality check: Piledriver CPUID/NOP



1 Mbit/s throughput-based signal, pattern 0101001000, 12 loops per measurement

Reality check: Piledriver RCL/NOP

Table: Clock cycles per measurement, Piledriver A6-4400M

Loops on core 1	Clock cycles on core 1 (RCL on core 0)	Clock cycles on core 1 (NOP/INC on core 0)	difference
1	89 (88.3%)	90 (35.2%), 91 (24%), 88 (24%)	-1, 1, 2
10	130 (86.5%), 129 (10.2%)	129 (95.2%)	-1, 0
11	135 (90.4%)	140 (95.0%)	5
20	213 (80.6%), 214 (9.1%)	220 (94.6%)	6, 7
100	949 (86.1%), 955 (10.4%)	963 (97.6%)	8, 14
1000	9049 (74.5%), 9055 (24.5%)	9063 (99.2%)	8, 14
10000	90049 (73.6%), 90055 (23.9%)	90063 (98.1%)	8, 14

Constant difference

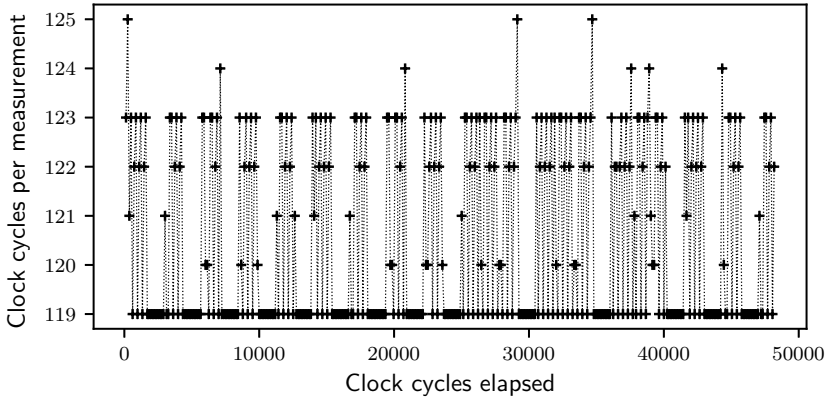
Reality check: Steamroller CPUID/NOP cross-core same-module

Table: Clock cycles per measurement, Steamroller A10-7800, same module

Loops on core 1	Clock cycles on core 1 (CPUID on core 0)	Clock cycles on core 1 (NOP/INC/RCL on core 0)	difference
1	119 (33.3%), 121 (33.0%), 122 (33.0%)	119 (99.9%)	0, 2, 3
10	161 (62.4%), 170 (29.2%)	160 (99.9%)	1, 10
100	592 (40.1%), 593 (39.8%), 599 (19.7%)	592 (99.9%)	0, 1, 7
1000	5092 (20.4%), 5093 (59.4%), 5099 (19.7%)	5092 (97.8%)	0, 1, 7
10000	50092 (41.0%), 50093 (37.7%), 50097 (18.7%)	50092 (98.7%)	0, 1, 5

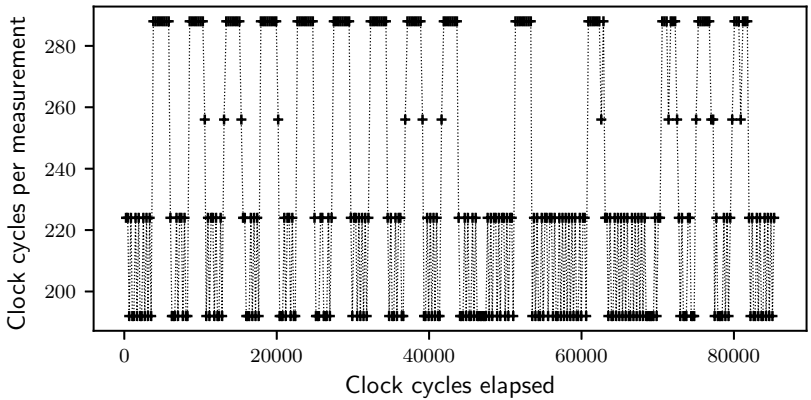
Oops. CPUID should have no effect here

Reality check: Steamroller CPUID/NOP cross-core same-module



1 Mbit/s noise-based signal, pattern 0101010101010101000100010001010101, 1 loop per measurement

Reality check: Ryzen CPUID/NOP cross-thread same-core



1 Mbit/s throughput-based signal, pattern 0101010101010101000100010001010101010, 40 loops per measurement

Why is CPUID so special

- Serializing instruction according to Intel
  - Recommended as a barrier for RDTSC precision

Why is CPUID so special

- Serializing instruction according to Intel
  - Recommended as a barrier for RDTSC precision
- Access to uncore properties and fuses according to informal AMD review
  - Fuses are per-CPU, not per-module nor per-core

Why is CPUID so special

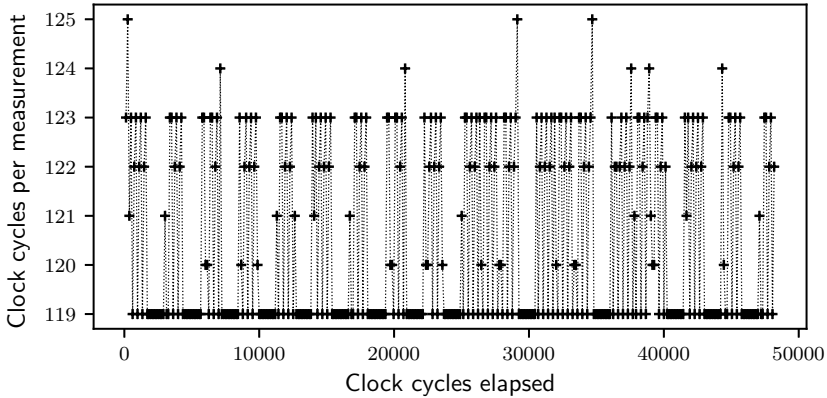
- Serializing instruction according to Intel
  - Recommended as a barrier for RDTSC precision
- Access to uncore properties and fuses according to informal AMD review
  - Fuses are per-CPU, not per-module nor per-core
- If the effect happens outside the instruction decoder, maybe it works cross-module?

Reality check: Steamroller CPUID/NOP cross-core same-module

Table: Clock cycles per measurement, Steamroller A10-7800, same module

Loops on core 1	Clock cycles on core 1 (CPUID on core 0)	Clock cycles on core 1 (NOP/INC/ RCL on core 0)	difference
1	119 (33.3%), 121 (33.0%), 122 (33.0%)	119 (99.9%)	0, 2, 3
10	161 (62.4%), 170 (29.2%)	160 (99.9%)	1, 10
100	592 (40.1%), 593 (39.8%), 599 (19.7%)	592 (99.9%)	0, 1, 7
1000	5092 (20.4%), 5093 (59.4%), 5099 (19.7%)	5092 (97.8%)	0, 1, 7
10000	50092 (41.0%), 50093 (37.7%), 50097 (18.7%)	50092 (98.7%)	0, 1, 5

Reality check: Steamroller CPUID/NOP cross-core same-module



1 Mbit/s noise-based signal, pattern 0101010101010101000100010001010101, 1 loop per measurement

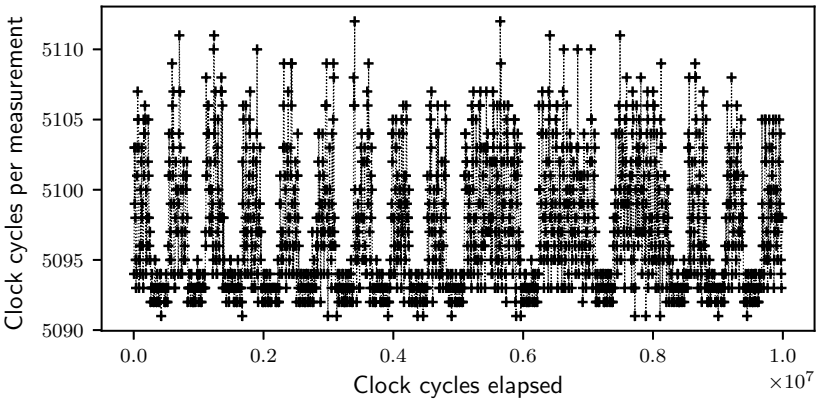
Reality check: Steamroller CPUID/NOP cross-core cross-module

Table: Clock cycles per measurement, Steamroller A10-7800, different module

Loops on core 3	Clock cycles on core 3 (CPUID on core 0)	Clock cycles on core 3 (NOP/INC/ RCL on core 0)	difference
1	119 (58.3%), 120 (30.4%), 121 (6.3%)	119 (60.2%), 120 (35.1%)	0, 1
10	160 (54.5%), 161 (30.3%), 159 (7.2%)	160 (61.2%), 161 (30.5%)	-1, 0, 1
100	592 (44.1%), 593 (25.2%), 591 (19.3%)	592 (59.1%), 591 (26.1%), 593 (13.1%)	0
1000	5093 (17.5%), 5092 (17.2%), 5094 (15.1%)	5092 (57.8%), 5091 (22.1%), 5093 (15.2%)	1
10000	50110 (4.1%), 50112 (4.1%), 50111 (4.1%)	50092 (53.8%), 50091 (21.0%), 50093 (18.5%)	inconclusive

There is an effect, but average doesn't work

Reality check: Steamroller CPUID/NOP cross-core cross-module



10 kbit/s noise-based signal, pattern 0101010101010101000100010001010101, 1000 loops per measurement

Current status

- CCCiCC v1
  - Same-module: 10 Mbit/s

Current status

- CCCiCC v1
  - Same-module: 10 Mbit/s
- CCCiCC v2
  - Same-module: 8 Mbit/s
  - Cross-module: 10 kbit/s

Questions?