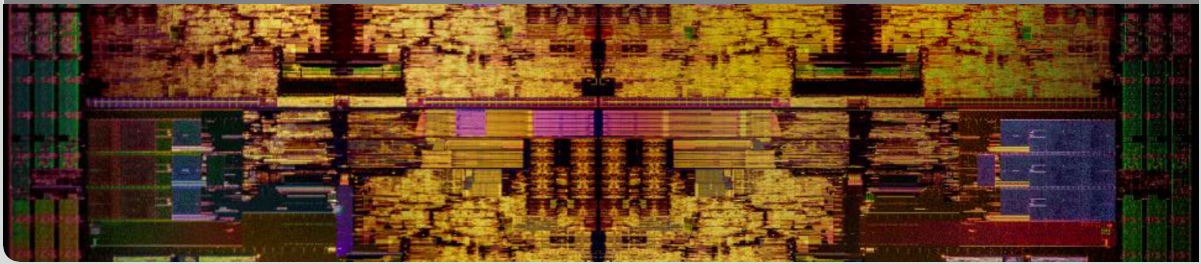
Karlsruhe Institute of Technology

Analysis and Optimization of Dynamic Voltage and Frequency Scaling for AVX Workloads Using a Software-Based Reimplementation

Yussuf Khalil | 25 September 2019

OPERATING SYSTEMS GROUP



Motivation

- AVX: modern vector processing extension for x86 processors
- Intel CPUs reduce clock frequency when executing AVX code
- Lowered frequencies retained after last AVX instruction
  - $\Rightarrow$ negative impact on performance in heterogeneous workloads

Figure: nginx throughput with different ChaCha20-Poly1305 implementations[1]

[1] Vlad Krasnov. *On the dangers of Intel's frequency scaling*. 2017. URL: https://blog.cloudflare.com/on-the-dangers-of-intels-frequency-scaling/.

Motivation

- AVX: modern vector processing extension for x86 processors
- Intel CPUs reduce clock frequency when executing AVX code
- Lowered frequencies retained after last AVX instruction
  - $\Rightarrow$ negative impact on performance in heterogeneous workloads

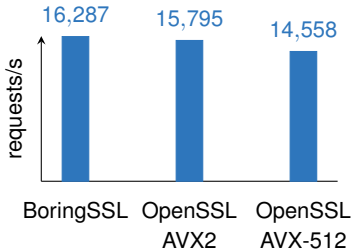


Figure: nginx throughput with different ChaCha20-Poly1305 implementations[1]

[1] Vlad Krasnov. *On the dangers of Intel's frequency scaling*. 2017. URL: https://blog.cloudflare.com/on-the-dangers-of-intels-frequency-scaling/.

Idea

- Frequency switches are costly
  - $\Rightarrow$ wait before raising frequency after AVX execution
- What if we could predict when AVX code will be executed again?
- Could we raise the frequency earlier and improve performance during scalar phases?
- Similar approaches are used for device power management

Try different reclocking algorithms!

Idea

- Frequency switches are costly
  - $\Rightarrow$ wait before raising frequency after AVX execution
- What if we could predict when AVX code will be executed again?
- Could we raise the frequency earlier and improve performance during scalar phases?
- Similar approaches are used for device power management

Try different reclocking algorithms!

Idea

- Frequency switches are costly
  - $\Rightarrow$ wait before raising frequency after AVX execution
- What if we could predict when AVX code will be executed again?
- Could we raise the frequency earlier and improve performance during scalar phases?
- Similar approaches are used for device power management

Try different reclocking algorithms!

Idea

- AVX reclocking is controlled solely by CPU itself
  - We can not modify the hardware
    - But we can (mostly) disable automatic reclocking

  $\Rightarrow$ reimplement algorithm in software
    - Allows to derive and test alternative algorithms
    - Needed to measure overhead of approach
- Intel only provides vague description of algorithm[2]
  - Downclocking after $\leq 500\,\mu s$
  - Upclocking after 2 ms
  - Two AVX frequency levels (“turbo licenses”)

  $\Rightarrow$ need to analyze what processors really do to create reimplementation

[2] *Intel 64 and IA-32 Architectures Optimization Reference Manual.* Intel Corporation. Apr. 2019.

Idea

- AVX reclocking is controlled solely by CPU itself
  - We can not modify the hardware
    - But we can (mostly) disable automatic reclocking

  $\Rightarrow$ reimplement algorithm in software
    - Allows to derive and test alternative algorithms
    - Needed to measure overhead of approach
- Intel only provides vague description of algorithm[2]
  - Downclocking after $\leq 500\,\mu s$
  - Upclocking after 2 ms
  - Two AVX frequency levels (“turbo licenses”)

  $\Rightarrow$ need to analyze what processors really do to create reimplementation

[2] *Intel 64 and IA-32 Architectures Optimization Reference Manual*. Intel Corporation. Apr. 2019.

Methodology

We want to find out

- when exactly a core will reduce or raise its frequency,
- how long it needs to do that,
- and whether Intel's description is correct and complete.

How can we measure that?

- Use performance counters
- Run synthetic code snippets designed to trigger specific behavior

Methodology

We want to find out

- when exactly a core will reduce or raise its frequency,
- how long it needs to do that,
- and whether Intel's description is correct and complete.

How can we measure that?

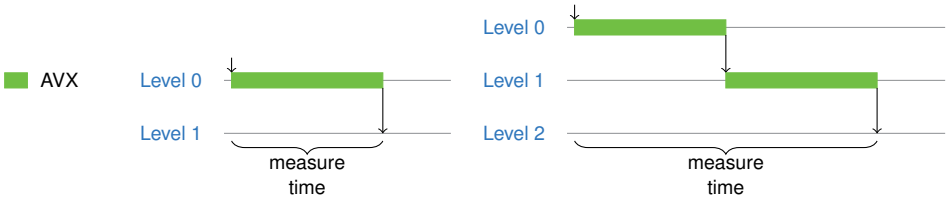
- Use performance counters
- Run synthetic code snippets designed to trigger specific behavior

Framework

- Linux kernel module
  - configures Performance Monitoring Unit (PMU)
  - handles interrupts
- User-space program
  - instructs kernel component
  - conducts measurements

Measurement Modes

Downclocking

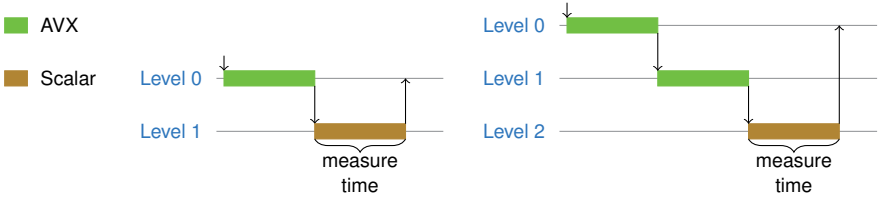


(a) Level 1

(b) Level 2

Measurement Modes

Upclocking

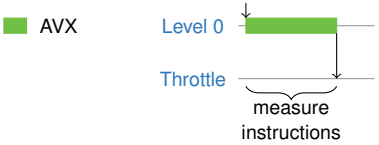


(a) Level 1

(b) Level 2

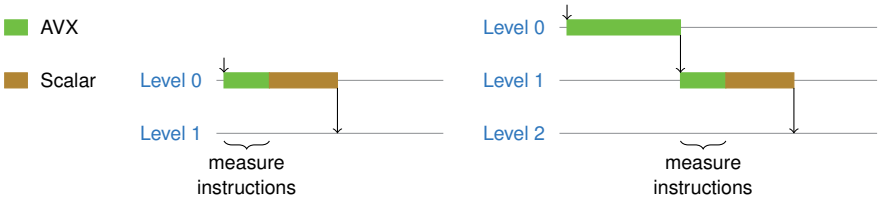
Measurement Modes

Throttle Activation



Measurement Modes

Trigger Instructions



(a) Level 1

(b) Level 2

Findings

Coarse behavior (Intel Core i9-7940X):

1. Throttle out-of-order engine after first AVX instruction
2. Switch to frequency level 1, needs $\approx 25\,\mu s$
3. Switch to frequency level 2 (only heavy AVX-512 instructions), $\approx 27\,\mu s$
4. Return to baseline frequency $\frac{2}{3}$ ms after last AVX instruction

In contrast, Intel claims

- downclocking takes “up to 500 µs”
- upclocking after 2 ms

Findings

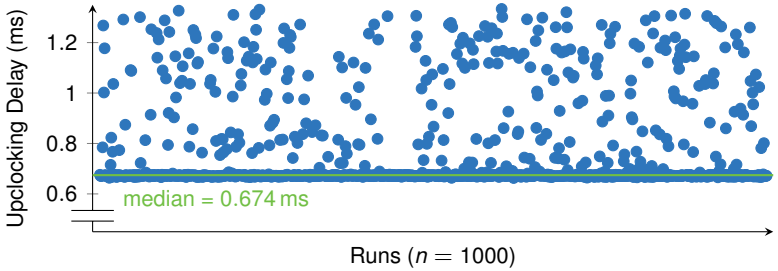
Coarse behavior (Intel Core i9-7940X):

1. Throttle out-of-order engine after first AVX instruction
2. Switch to frequency level 1, needs $\approx 25\,\mu s$
3. Switch to frequency level 2 (only heavy AVX-512 instructions), $\approx 27\,\mu s$
4. Return to baseline frequency $\frac{2}{3}$ ms after last AVX instruction

In contrast, Intel claims

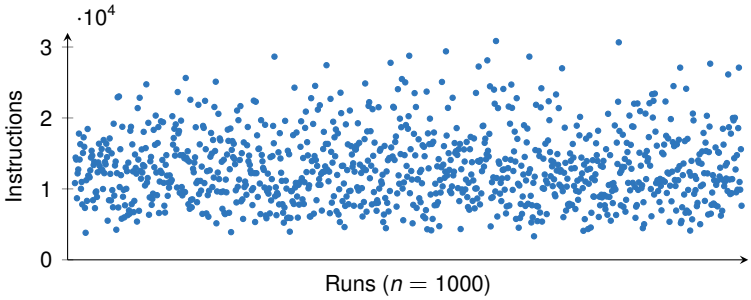
- downclocking takes “up to 500 µs”
- upclocking after 2 ms

Interesting Outlier



- Heavy AVX-512 instructions *sometimes* need up to $\frac{4}{3}$ ms to return from level 1
- We do not have an explanation for this

256 bit vs. 512 bit



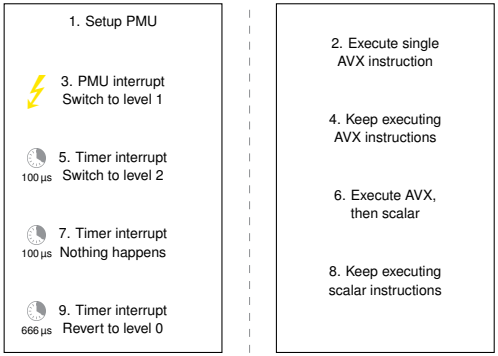
- 512-bit instructions always trigger frequency reductions after first instruction
- 256-bit ones exhibit high variance in the amount of required operations (shown above)

Design

- Modify `intel_pstate` Linux driver
  - `intel_pstate` manages core frequency
- Introduce virtual AVX frequency levels similar to hardware levels
- Use performance counters to measure executed AVX instructions
  - **Only available for floating-point**

Algorithm

1. Interrupt after first 512-bit instruction
2. Switch to level 1 frequency
3. Measure instruction throughput in 100 µs intervals
4. If higher than one per cycle: switch to level 2
5. Raise frequency after 666 µs with no 512-bit instructions



Oracle Mechanism

- Want to find theoretical limit for optimized algorithms
  - $\Rightarrow$ implement oracle mechanism
- System call API to tell reimplementation when to switch frequency levels
- Synthetic workload with scalar and vector phases
  - $\Rightarrow$ scalar phases of about $\frac{2}{3}$ ms expose worst case
    - hardware's worst case is best case for oracle
  - Count iterations in each phase

Reimplementation

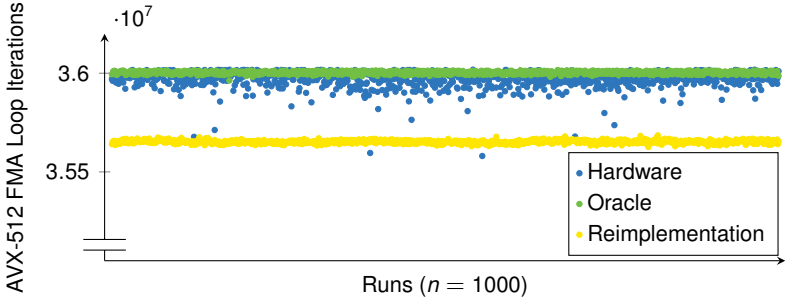
- Need to evaluate correctness and quality of reimplementation
  - $\Rightarrow$ use analysis framework, compare with hardware results
- Analysis framework and reimplementation both use performance monitoring
  - $\Rightarrow$ modify analysis kernel module and reimplementation to work together
- Measure performance overhead

Model Accuracy

- Downclocking to level 1 works as expected
- Reaching level 2 takes $\approx 104\,\mu s$
  - unlike the hardware, we can measure throughput only over time
- Upclocking after $666\,\mu s$ as designed
  - however, up to $100\,\mu s$ more possible due to limited measurement accuracy

Overhead and Oracle Performance

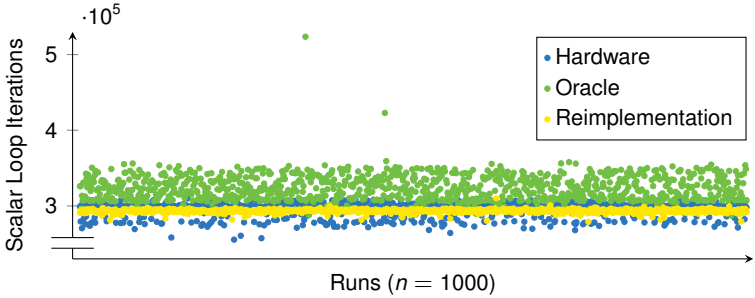
AVX phase (200 ms)



- Reimplementation is $\approx 1\,\%$ slower in AVX phases
- Oracle yields same performance as hardware

Overhead and Oracle Performance

Scalar phase (666 µs)



- Reimplementation $\approx 2\,\%$ slower
- Oracle gives $\approx 8\,\%$ median performance increase (99th percentile: $15\,\%$)

Future Work

- Implement alternative reclocking algorithms
  - Take ideas from power management
- More analysis
  - Other processors (Haswell, Icelake)
  - SMT
  - Instruction mixtures
- Reverse engineer AVX frequency offset configuration

Conclusion

- Performance of workloads with scalar and vector phases suffers from AVX reclocking
- We want to find improvements to the reclocking behavior
- Intel's claims about the algorithm are not accurate
- Improvements for heterogeneous workloads are theoretically possible
  - up to 15 % performance increase
- Possibilities for software reimplementation are limited
  - Still a potentially promising approach
  - Improvements likely possible