Karlsruhe Institute of Technology

Mitigating AVX-Induced Performance Variability with Core Specialization

Mathias Gottschlag, Frank Bellosa | October 18, 2018

KARLSRUHE INSTITUTE OF TECHNOLOGY (KIT) - OPERATING SYSTEMS GROUP

Motivation

- Web server benchmark at Cloudflare:
  - ChaCha20-Poly1305 encryption, OpenSSL vs. BoringSSL
  - In isolation, OpenSSL much faster: 2.89 GB/s vs. 1.46 GB/s
  - OpenSSL system overall 10% slower!
  - Only ≈2.5% of time spent on encryption

- Only difference:
  - OpenSSL: 512-bit vectorization (AVX-512)
  - BoringSSL: 256-bit vectorization (AVX2)

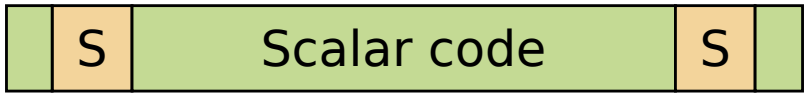
- What happened?

Vlad Krasnov: *On the dangers of Intel's frequency scaling*. Blog post, Nov. 2017

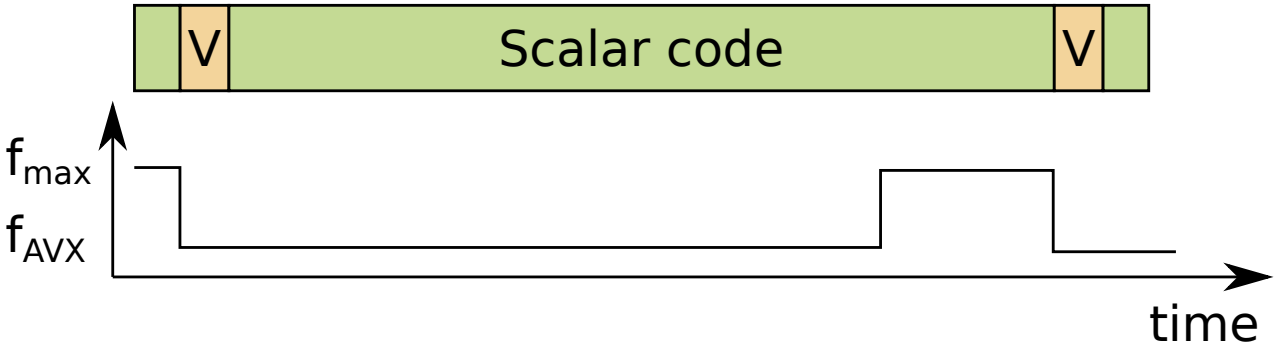
AVX512 Performance Variability (1)

- AVX-512 units draw increased power
- CPU reduces frequency (for at least 2 ms)

No vectorization:



AVX-512:

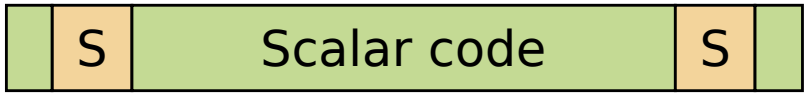


- Same effect for AVX2, but weaker
- ➜ **Slows down unrelated scalar code**

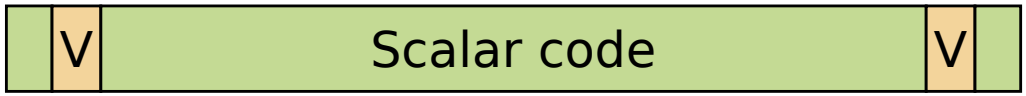
Intel®64 and IA-32 Architectures Optimization Reference Manual, April 2018

AVX512 Performance Variability (2)

No vectorization:



AVX-512:



- Development effort
  - Effect depends on workload $\Rightarrow$ Might not notice
  - Misleading profiling results
- Dependability
  - A simple library update can break your system!
- Fairness
  - Isolation on multi-tenant systems?

Only on Intel CPUs?

- Future chips: Lots of dark silicon

➜ Intensive use of accelerators

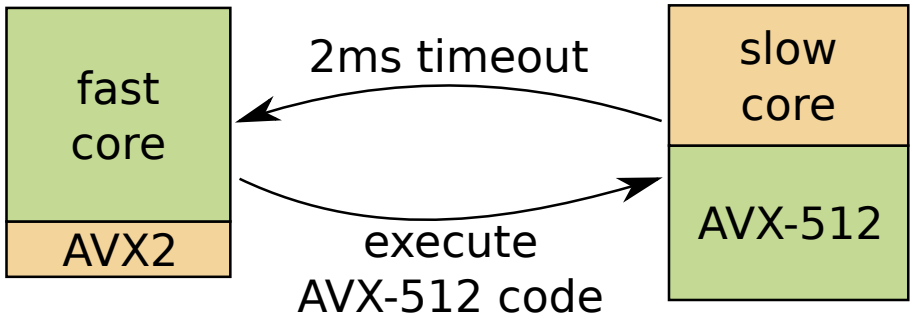
- Accelerators consume additional power

➜ **Future chips will (likely) show similar behaviour.**

Michael B. Taylor: *Is Dark Silicon Useful? Harnessing the Four Horsemen of the Coming Dark Silicon Apocalypse.* DAC'12

Idea: Reconfigurable System

- Observation: Core is actually reconfigurable
  - Fast core without AVX-512
  - Slow core with AVX-512

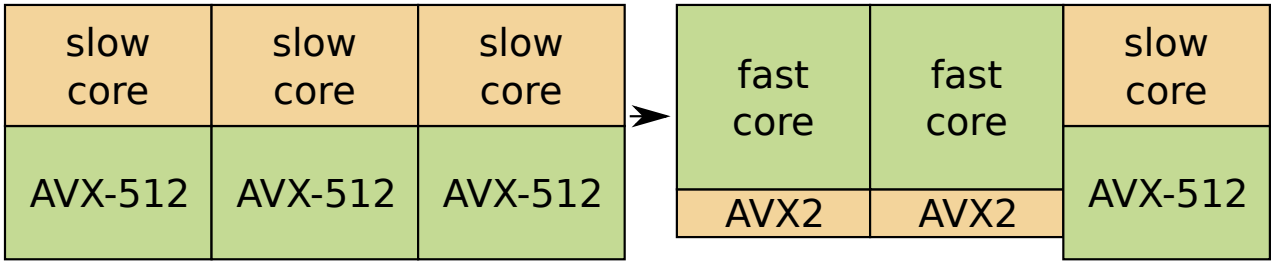


- Reconfig. mechanism: Execute any AVX-512 instruction

- But: Reconfigurable system theory not applicable
- Approaches focus on choosing one core type for a job
  - E.g., emulate AVX-512 instructions
- **Each core type only good for parts of the code**

Idea: Asymmetric System

- Observation: We have multiple cores
- Asymmetric systems perform well for heterogeneous workloads



- **Dedicate some (most?) cores to scalar code**

Mechanism: Instrumentation

- Need to modify existing software

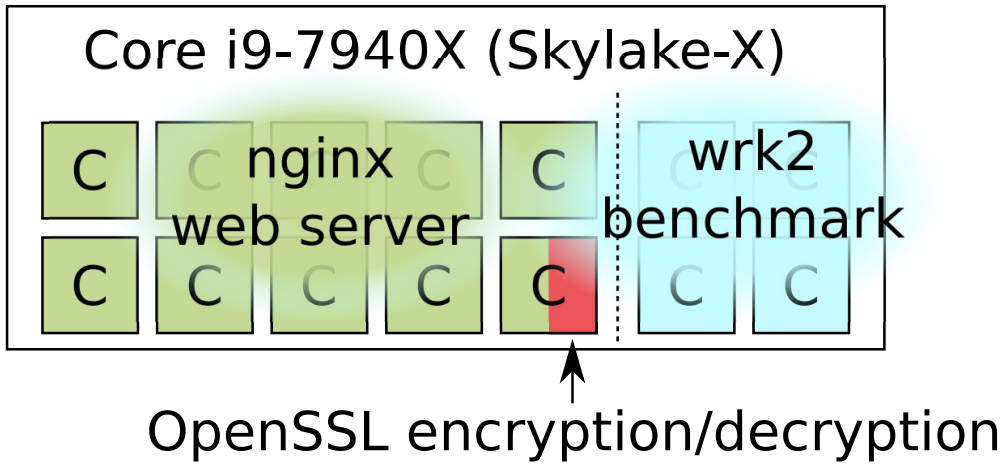
➜ Manual instrumentation of problematic functions

- Library to move function pointer and context to thread pool

```
1 run_on_avx_core(function(){
2         avx_code();
3 });
```

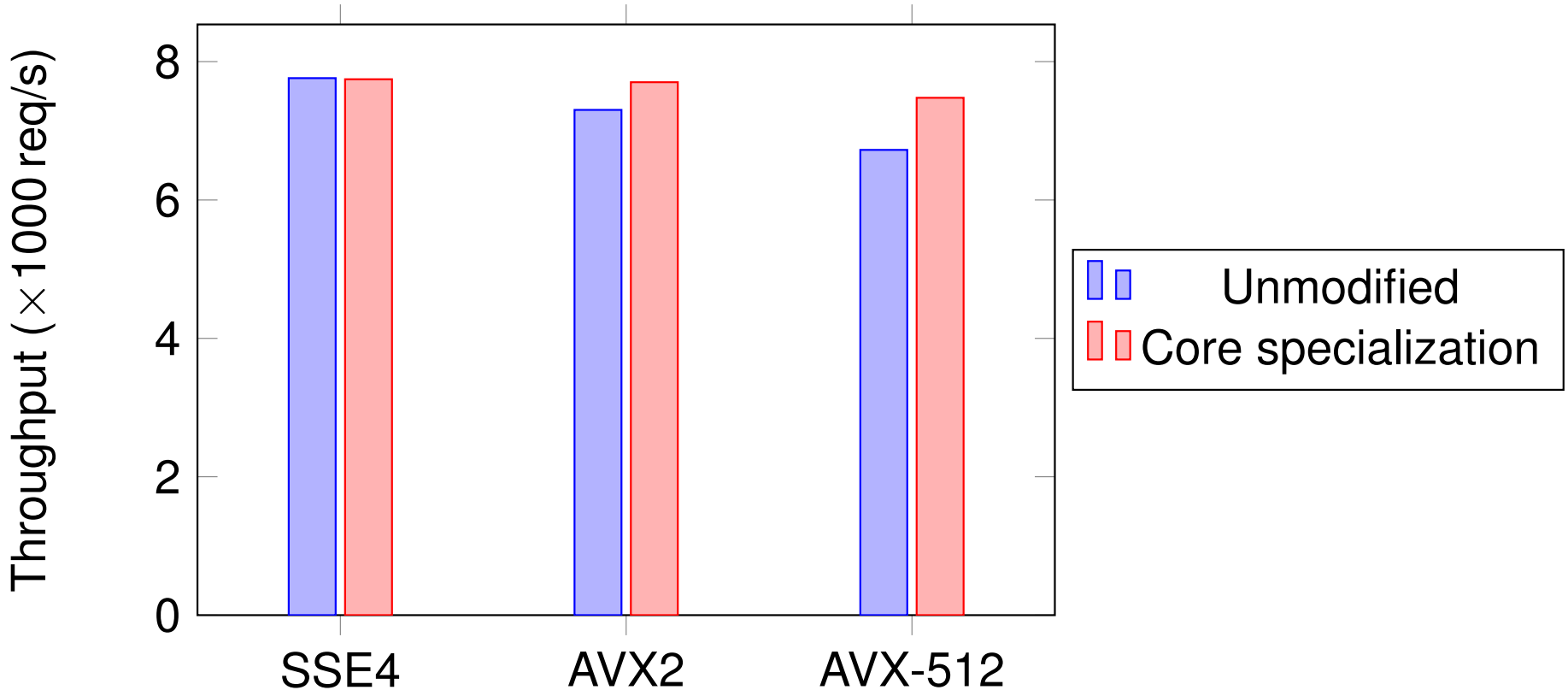
Prototype

- Cloudflare scenario replicated
  - nginx web server, OpenSSL, wrk2 benchmark client
  - Static web page, on-the-fly compression
- OpenSSL encryption/decryption on separate cores



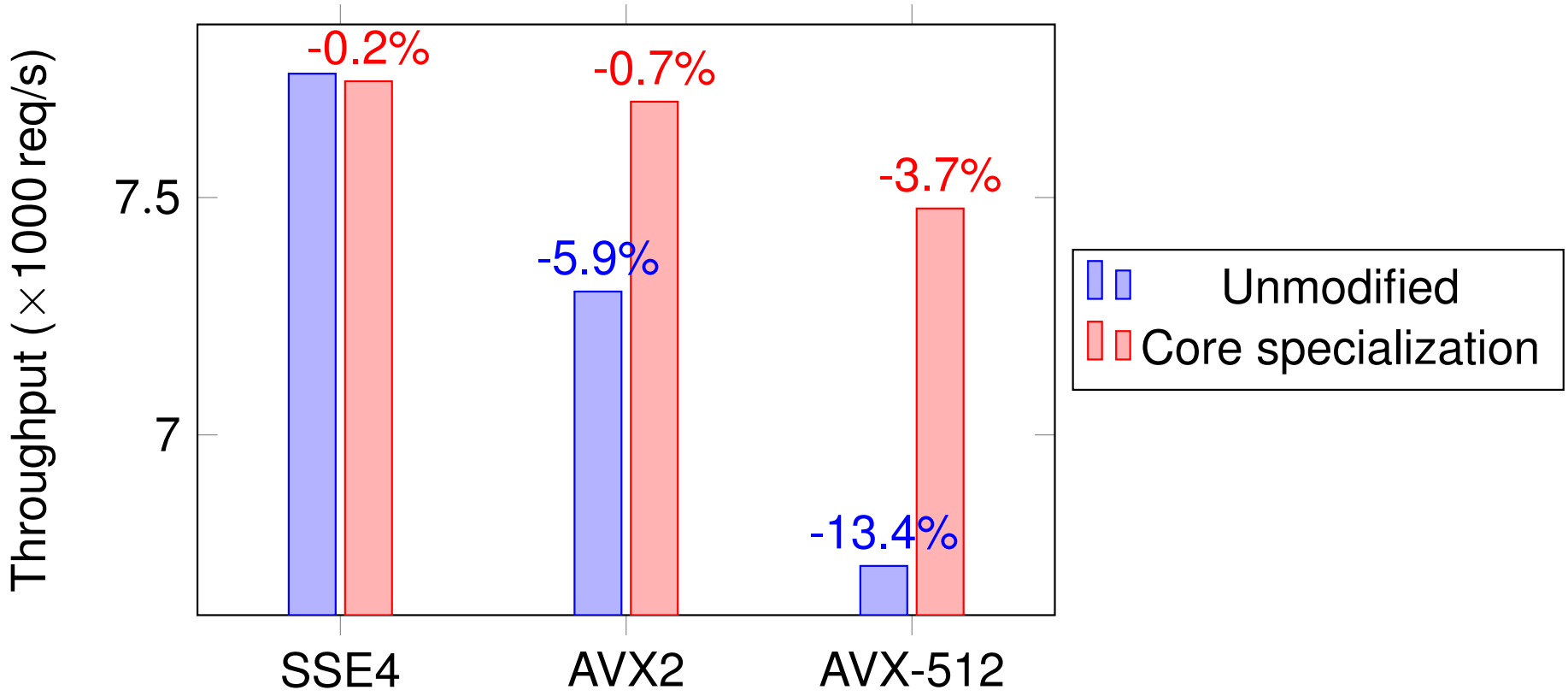
- 5% vectorized code $\Rightarrow$ one hyperthread
- Other hyperthread: Scalar code, slowed down

Results: Throughput



⇒ **Core specialization reduces variablity**

Results: Throughput



⇒ **Core specialization reduces variablity**

- Always all HW threads of a core affected
- Dedicating whole core to AVX would reduce utilization

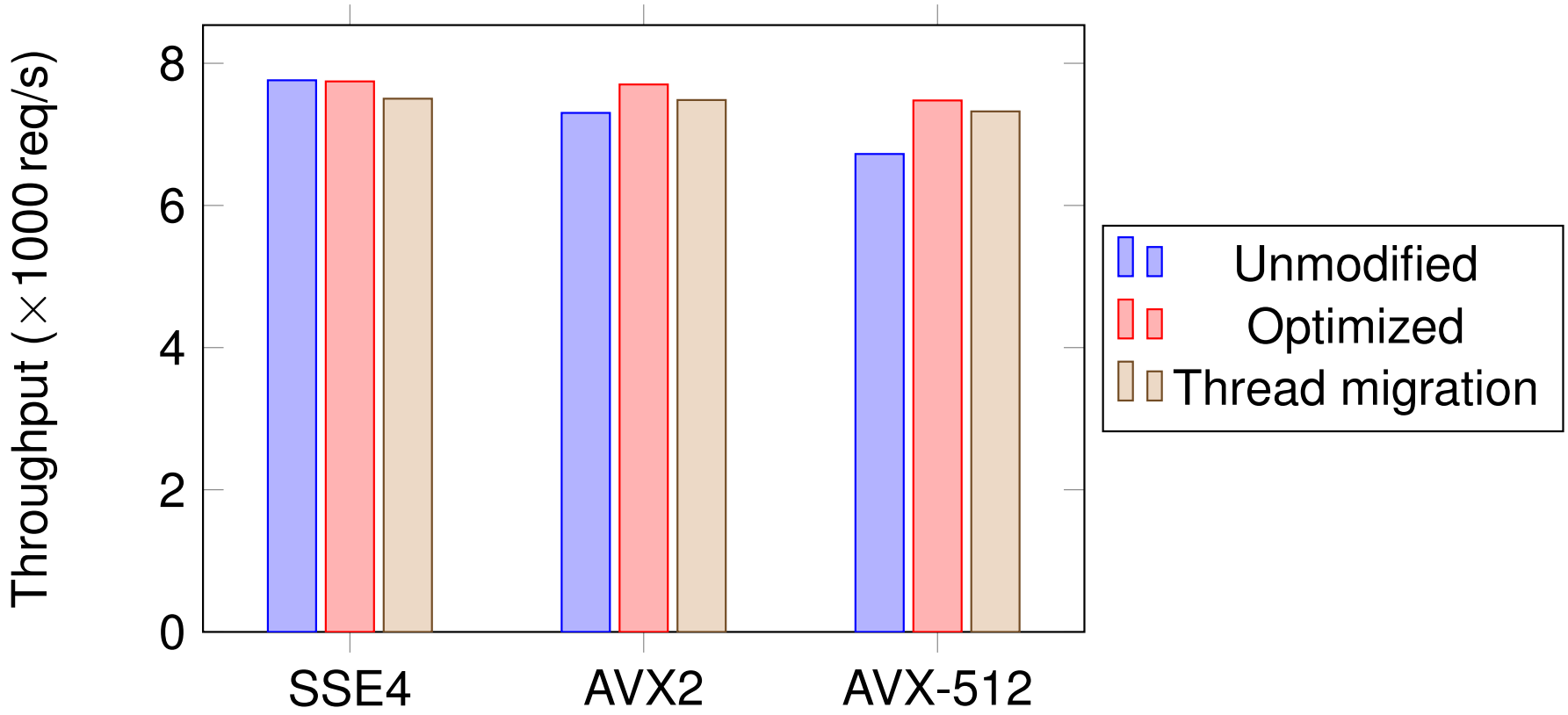
Mechanism Design Space

- Better: Trap problematic instructions
  (Intel: Restrict storage for thread context?)
- Automatically migrate thread
- Easier to use, but likely higher overhead

- Prototype for performance evaluation:

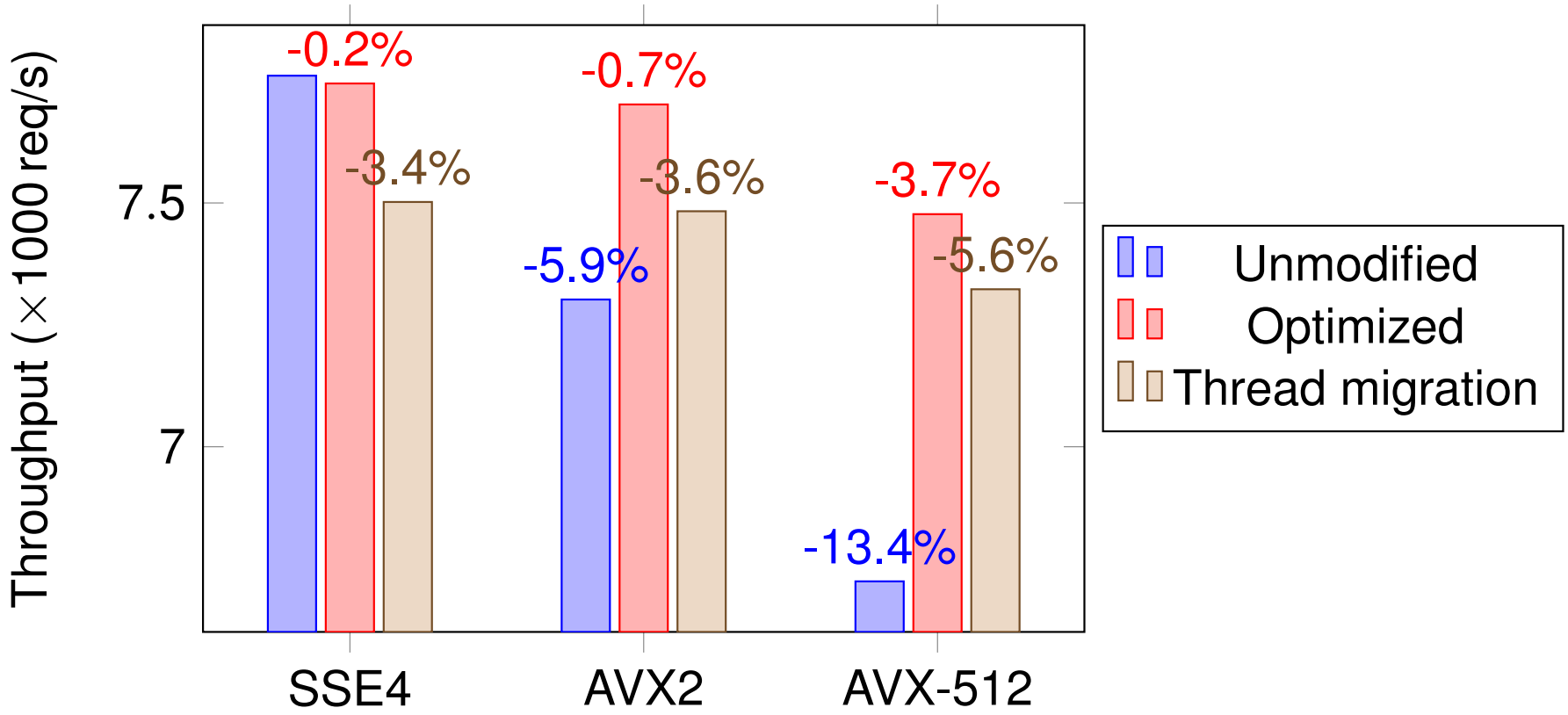
```
1 sched_setaffinity(my_pid, AVX_CORES);
2 avx_code();
3 sched_setaffinity(my_pid, SCALAR_CORES);
```

Results: Throughput



- Thread migration causes (moderate) overhead

Results: Throughput



- Thread migration causes (moderate) overhead
- **Performance advantage still significant**

Future Work

- Prototype: Information about problematic code available
- Practice: Limited knowledge/too expensive

- Instrumentation: How to pinpoint problematic code?
  - Not all vector instructions cause equal frequency change
  - Frequency change happens significantly after problematic code
  - Use last-branch record to move back in time

- Automatic approach: When to migrate back to scalar core?
  - Migrate back after fixed time
  - Iterative approach: Change timeout, measure avg. frequency, repeat.

- Better hardware: Want to be notified *before* reconfiguration?

Summary

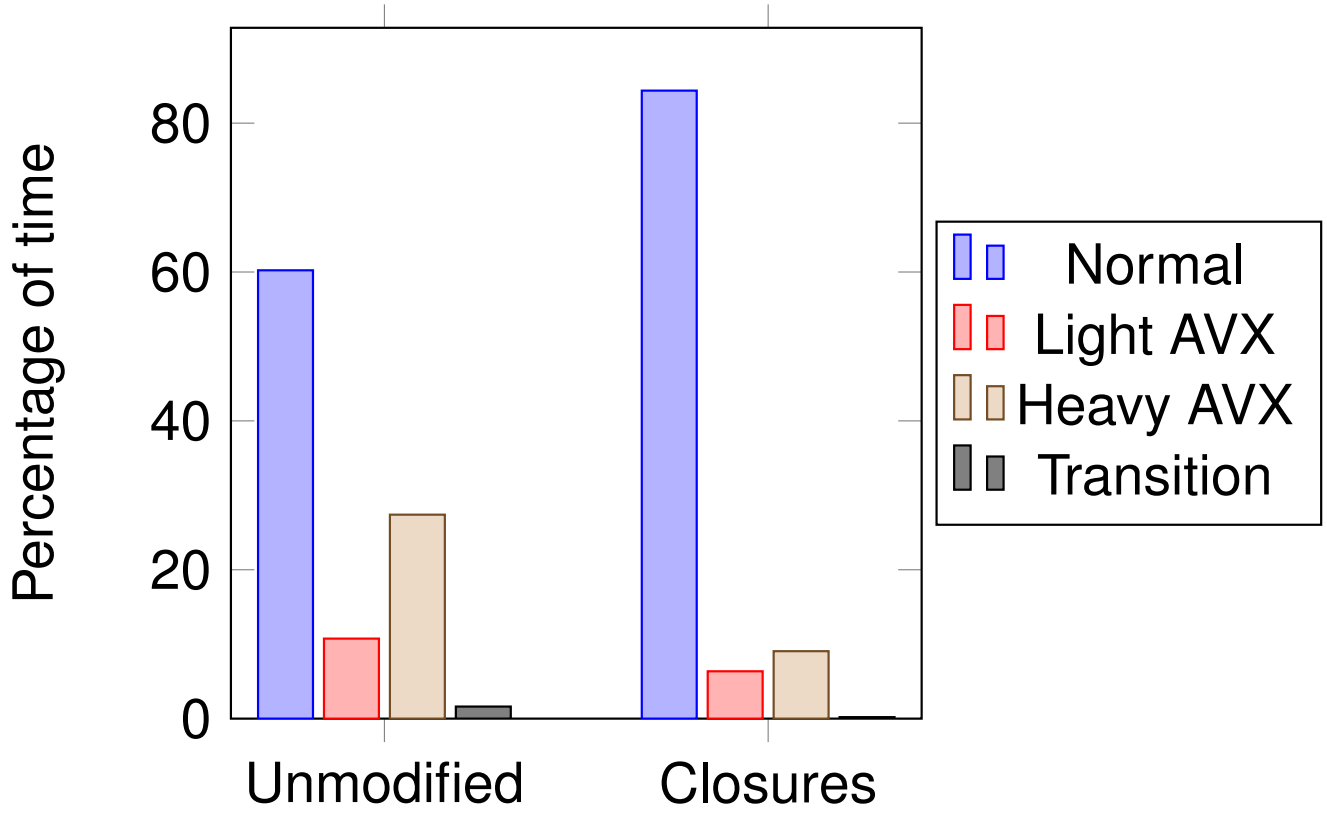
- Significantly lower AVX-512 frequency
- Frequency changes expensive $\Rightarrow$ unrelated code affected
- Performance highly variable

- Approach: Model system as reconfigurable asymmetric system
- Migrate threads between AVX and non-AVX cores

- Result:
  - Frequency effect mitigated
  - Significantly increased performance

Results: Frequency

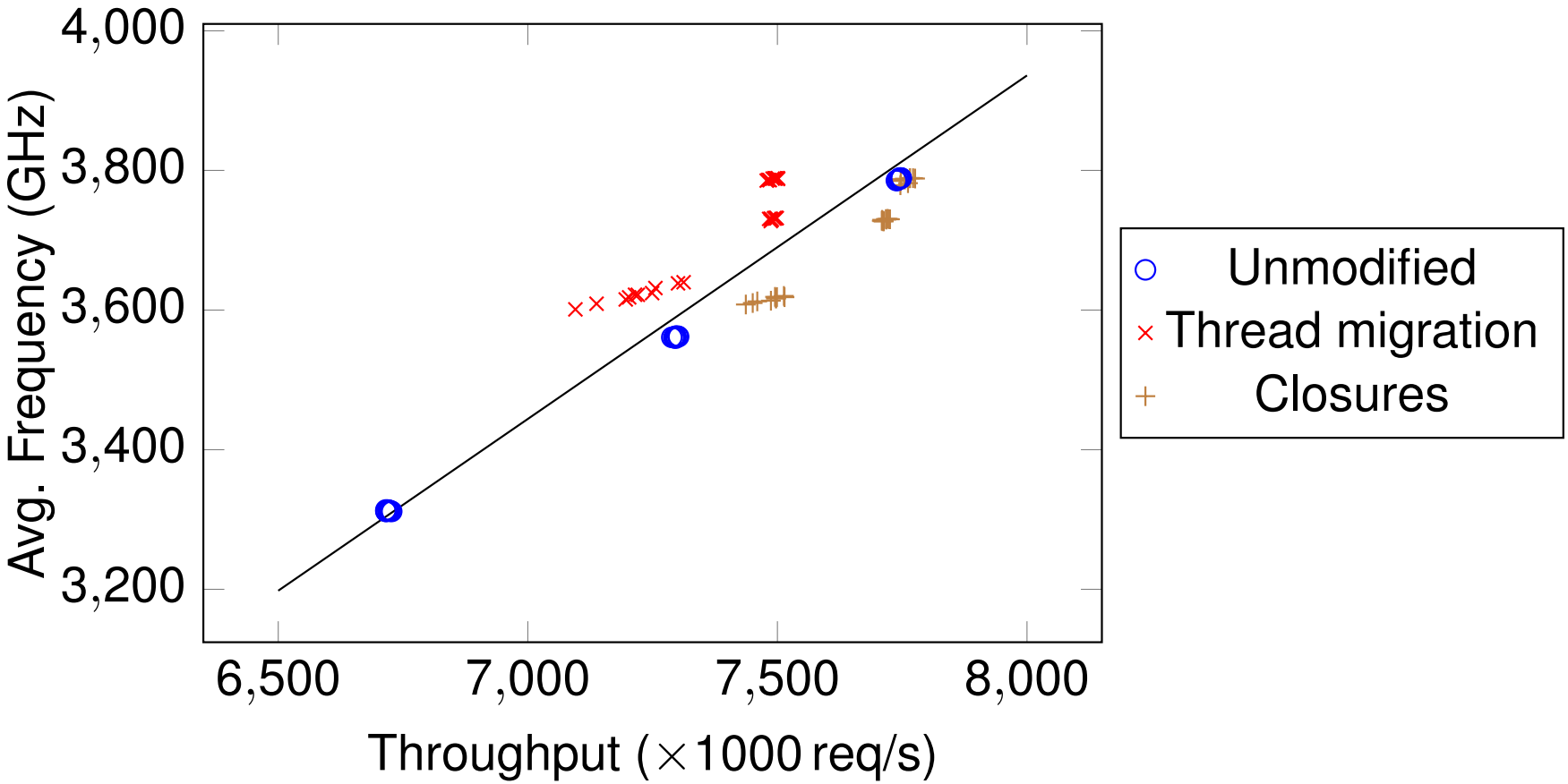
- Time spent at different frequency levels
- For OpenSSL with AVX-512:



➜ Much less time at AVX frequencies

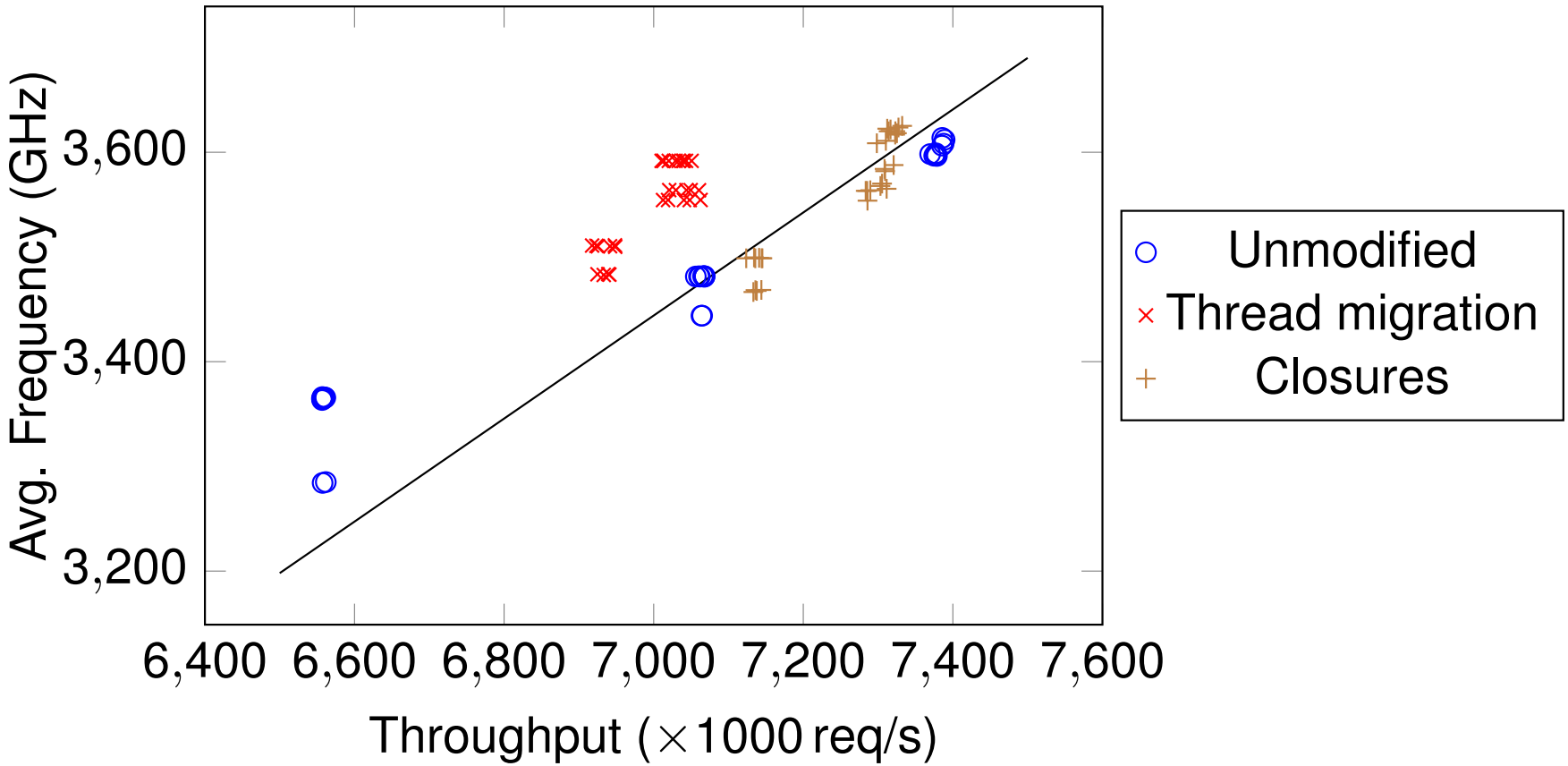
Results: Frequency

- Correlation between frequency and performance?



Results: Frequency

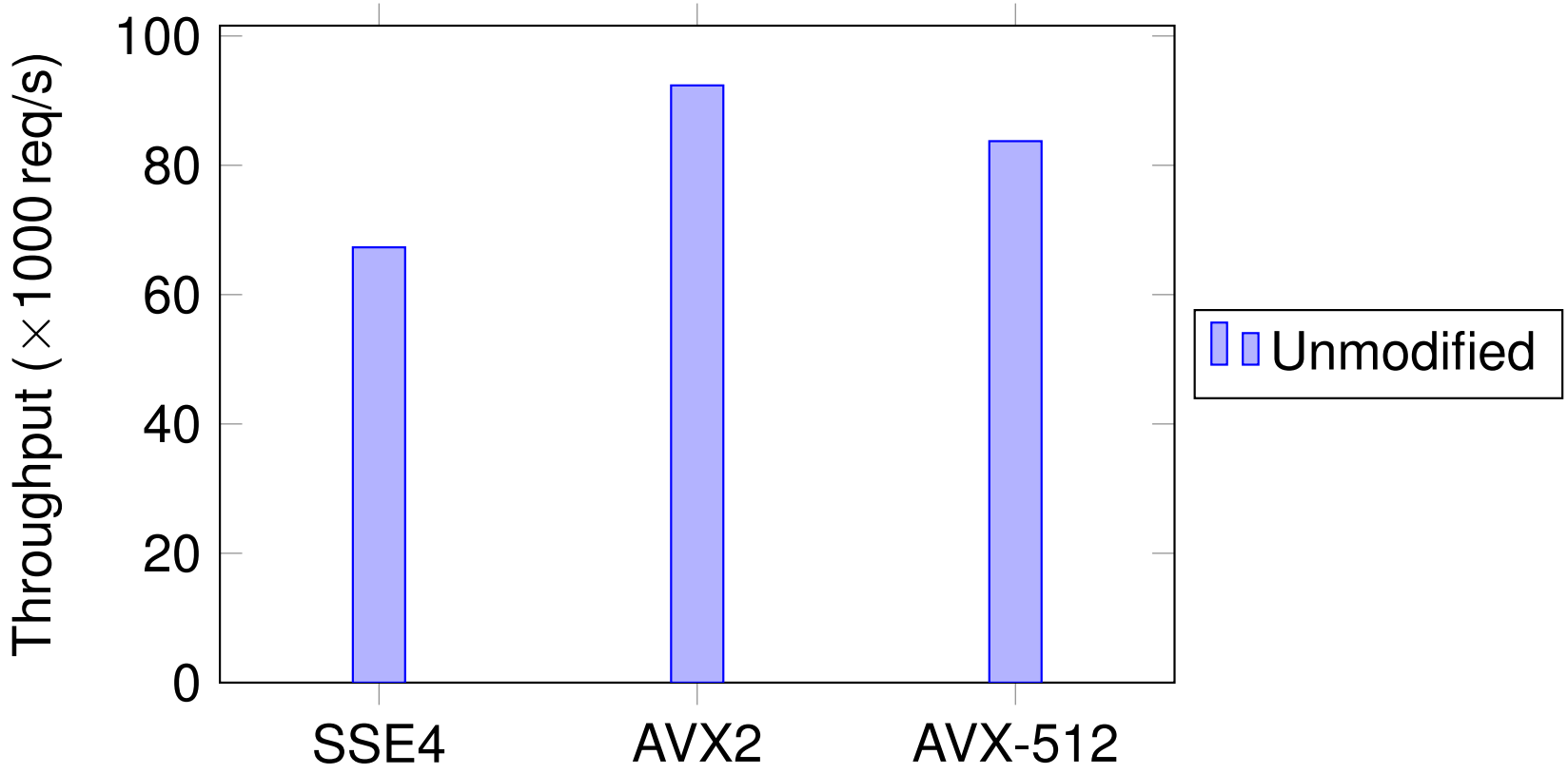
- What if C-states and cpufreq governor are disabled?



➜ Slower (less turbo), but no significant difference

Advantage of vectorization?

Different workload, less scalar CPU load:



➜ Vectorization beneficial