Conquering Noise with Hardware Counters on HPC Systems

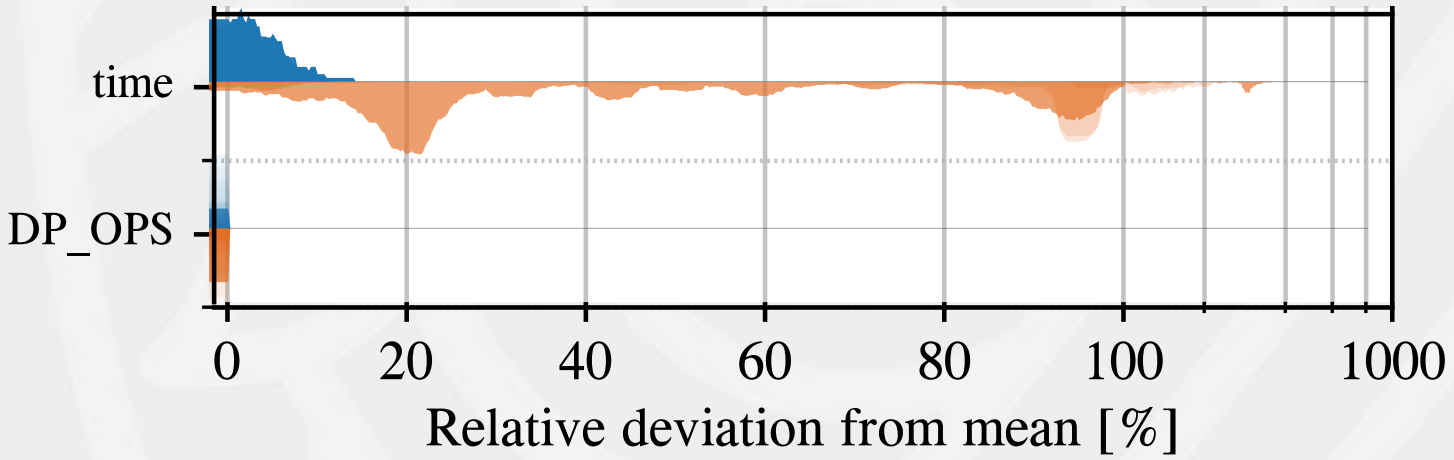
Marcus Ritter[1], **Ahmad Tarraf**[1], Alexander Geiß[1], Nour Daoud[2], Bernd Mohr[2], Felix Wolf[1]

[1]Technical University of Darmstadt

[2]Forschungszentrum Jülich GmbH

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming



Conquering Noise with Hardware Counters on HPC Systems

Marcus Ritter[1], **Ahmad Tarraf**[1], Alexander Geiß[1], Nour Daoud[2], Bernd Mohr[2], Felix Wolf[1]

[1]Technical University of Darmstadt

[2]Forschungszentrum Jülich GmbH

Motivation

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

Performance and complexity of HPC systems are constantly increasing

→ Important to examine the scaling behavior of an application and identify early **performance bottlenecks**

→ Use empirical performance modeling

Problem:

In **noisy** environments → difficult to create accurate performance models

- Strong variations in the measurements
- Measurements irreproducible and misleading
- Strong deviations from the actual application behavior

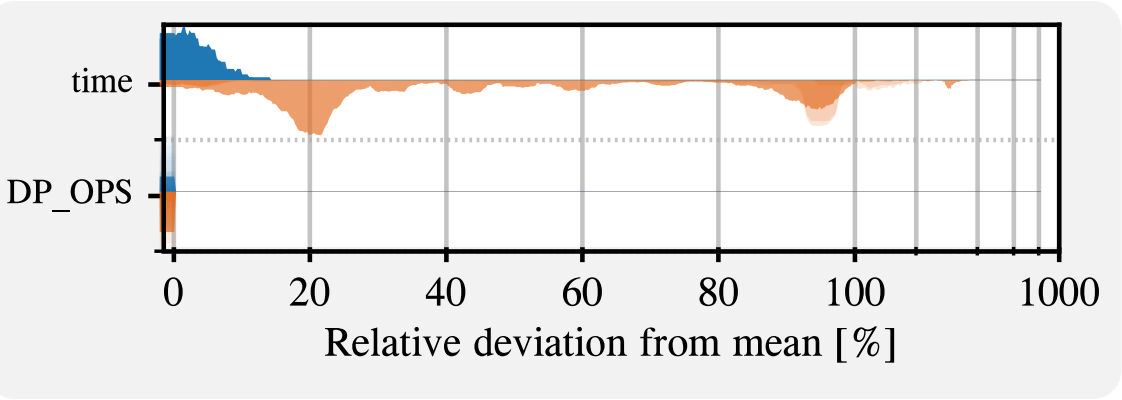
Motivation

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

Problem (cont.):

- **Application runtime** affected by noise
  - Most common performance metric



Solution:

Use hardware counters

- Noise has little impact on some **hardware counters**
  - e.g., double precision operations
- Selecting the right counters requires a thorough analysis

Contributions

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

A detailed noise analysis on various hardware counters on different systems:

→ Total of **26950** experiments (PAPI preset events only):

- **Five** systems
- **With** and **without** injected noise
- **Four** hardware architectures
- **Multiple** resource configurations (number of nodes)
- **Three** applications
- **Five** repetitions per setup

Categorized the counters across the different systems according to their noise resilience and provided a **user guide**

Analysis Methodology

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

Find noise-resilient hardware counters:

- Examine if counters' values change when repeating the measurements
- Expose the counters to different levels of noise
  - Inject different noise patterns using NOIGENA
  - NOIGENA processes were running on the odd processors

```
PATTERN_1:
  Sequence:
    - REPEATED_NOISE:
        REPEAT: inf
        Sequence:
          - NETWORK_NOISE: 2
          - NO_NOISE: 2
          - MEMORY_NOISE: 2
          - NO_NOISE: 2
```

Noise pattern used by NOIGENA to configure the amount and duration of generated noise.

Analysis Methodology

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

Compare counter values of different call paths from three applications for the repeated experiments

→ Calculate the **relative deviation** from the arithmetic mean in percent:

$$\frac{|v_i - \bar{v}|}{\bar{v}} * 100\%$$

For the a^{th} application kernel, p^{th} MPI rank, t^{th} OpenMP thread, and i^{th} repetition,

→ Find the arithmetic mean across the **repetitions**:

$$\bar{v}_{a,p,t} = \text{mean}(v_{a,p,t,i}) \rightarrow \bar{v} = \text{mean}(v_i)$$

We do this for all counters on all systems with and without noise

Application Benchmarks

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

MiniFE

- Mini-app that mimics: finite element generation, assembly, and solution
- Unstructured grid problem, required by many engineering applications

LULESH

- Proxy app solves a Sedov blast problem with analytic answers
- Represents algorithms, data motion, and programming style typical in scientific applications

LAMMPS

- Classical molecular dynamics code focusing on materials modeling
- We use the atomic fluid, Lennard-Jones (LJ) potential

For all of them, we use OpenMP and MPI for the measurements

Evaluation Systems

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

Alias	Name	Nodes	Processor	RAM	Network
CM	DEEP-EST, Cluster Module	50	2x Intel Xeon Skylake Gold 6146 CPUs (12 cores, 24 threads)	192 GB DDR4 RAM (2666 MHz)	InfiniBand EDR (100 GBit/s)
ESB	DEEP-EST, Extreme Scale Booster	75	1x Intel Xeon Cascade Lake Silver 4215 CPU (8 cores, 16 threads)	48 GB DDR4 RAM (2400 MHz),	InfiniBand EDR (100 GBit/s)
Jureca	JURECA DC Module, std. compute nodes	480	2x AMD EPYC 7742 CPUs (64 cores, 128 threads)	512 GB DDR4 RAM (3200 MHz)	InfiniBand HDR100 (100 GBit/s)
Jetson	OACISS, Franken-cluster Jetson ARM64	12x Jetson Tegra TX1	1x Quad-Core ARM Cortex®-A57 MPCore (4 cores, 4 threads)	4 GB 64-bit LPDDR4 RAM	1 GBit/s ethernet
Cyclops	OACISS, Franken-cluster Cyclops	1	2x 20c IBM Power9 CPUs (20 cores, 80 threads)	384 GB of RAM	BNX2 10G Ethernet NICs, 2x Infiniband EDR (25 GBit/s)

Application Configurations

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

App	System	Experiment configuration
MiniFE	CM	$n = [1, 2, 4, 8]$, $p = n$, $t = 12p$, $s = 20^3 p$
	ESB	$n = [1, 2, 4, 8, 12, 16, 32]$, $p = n$, $t = 8p$, $s = 50^3 p$
	Jureca	$n = [4, 8, 12, 16, 20, 24]$, $p = n$, $t = 128p$, $s = 20^3 p$
	Jetson	$n = [2, 3, 4, 5, 6]$, $p = n$, $t = 2p$, $s = 50^3 p$
	Cyclops	$n = 1$, $p = [4, 8, 12, 16, 20]$, $t = 4p$, $s \approx 24^3 p$
LULESH	CM	$n = [1, 8, 27]$, $p = n$, $t = 12p$, $s = 10^3 p$
	ESB	$n = [1, 8, 27]$, $p = n$, $t = 8p$, $s = 30^3 p$
	Jureca	$n = [1, 8, 27]$, $p = n$, $t = 128p$, $s = 10^3 p$
	Jetson	$n = 8$, $p = n$, $t = 2p$, $s = 15^3 p$
	Cyclops	$n = 1$, $p = [1, 8, 27, 64]$, $t = p$, $s = 5^3 p$
LAMMPS	CM	$n = [1, 2, 4, 8]$, $p = n$, $t = 12p$, $s = 20^3 p$
	ESB	$n = [1, 2, 4, 8, 12, 16, 32]$, $p = n$, $t = 8p$, $s = 20^3 p$
	Jureca	$n = [1, 2, 4, 8, 16]$, $p = n$, $t = 128p$, $s = 20^3 p$
	Jetson	$n = [2, 3, 4, 5, 6]$, $p = n$, $t = 2p$, $s = 20^3 p$
	Cyclops	$n = 1$, $p = [4, 8, 12, 16, 20]$, $t = 4p$, $s = 20^3 p$

Evaluation Results

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

Visualizing the results for **each counter** is not an easy task:

- Three different application
- Thousands of different call paths
- Different resource configuration
- Several iterations per setup

→ Count relative metric (deviation from the mean)

→ Intensity (importance to the overall behavior)

- Five different systems
- Several hardware counters

→ Distinct plots

- Presence and absence of noise

→ Separate axes

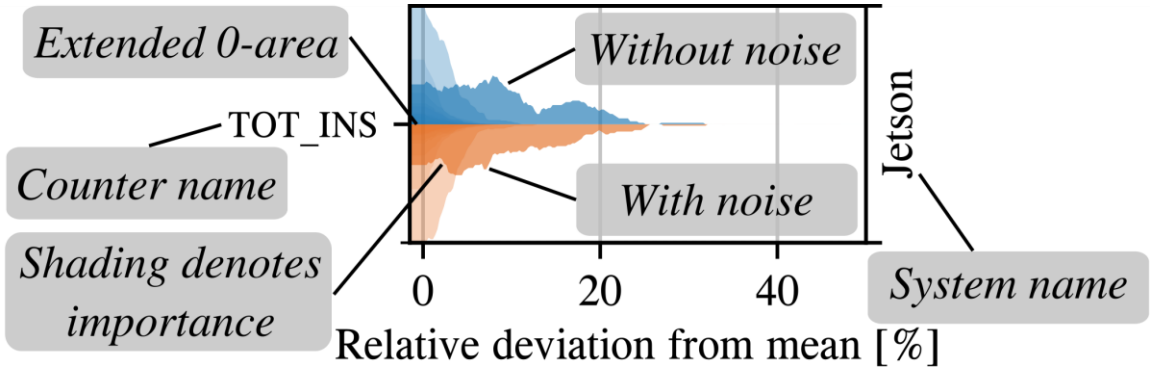
To compare distinct counter → scale the plots with the peak occurrence of the relative deviation

Evaluation Results

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

- The height of each distribution plot describes how often the corresponding relative deviation occurs
- Maximum value of the peak relative deviation was used to scale all callpaths
- We only show callpaths that resulted in more than 1% of the total counter value

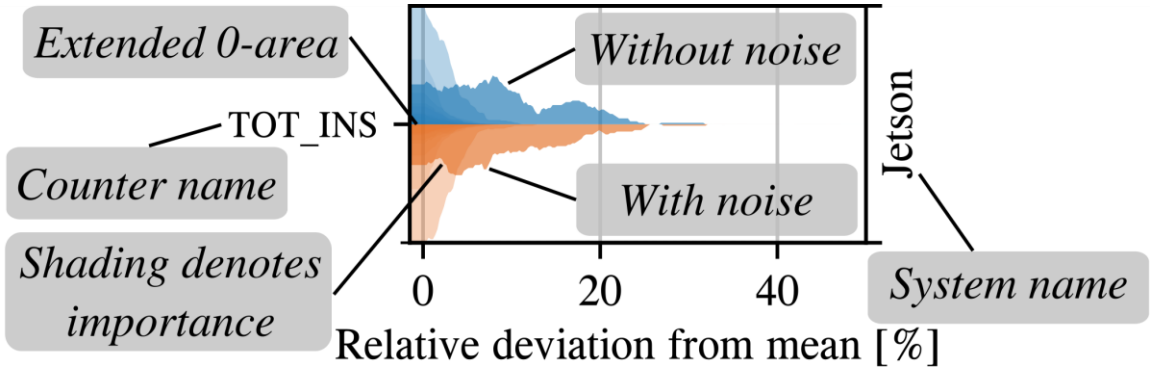


Evaluation Results

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

- The intensity (alpha) shows the importance of the region to the overall behavior
- The importance is the callpath's share of the total counted value across all applications per counter
- Large values indicate the parts of the application that exhibit more of the behavior described by that counter



Evaluation Results: How to Interpret the Results

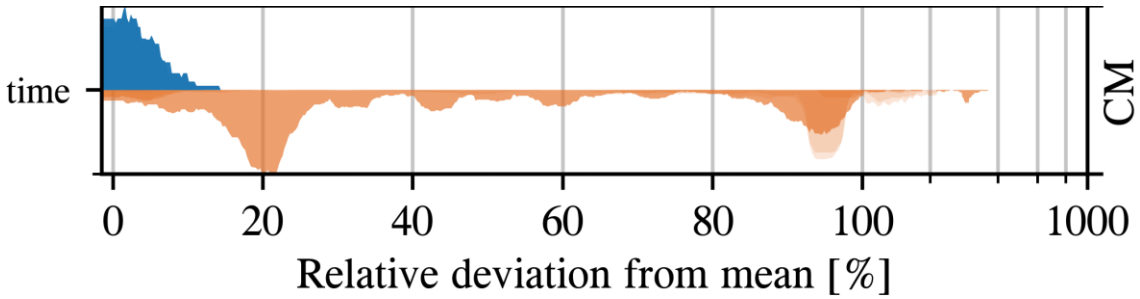
TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

Bad Counters

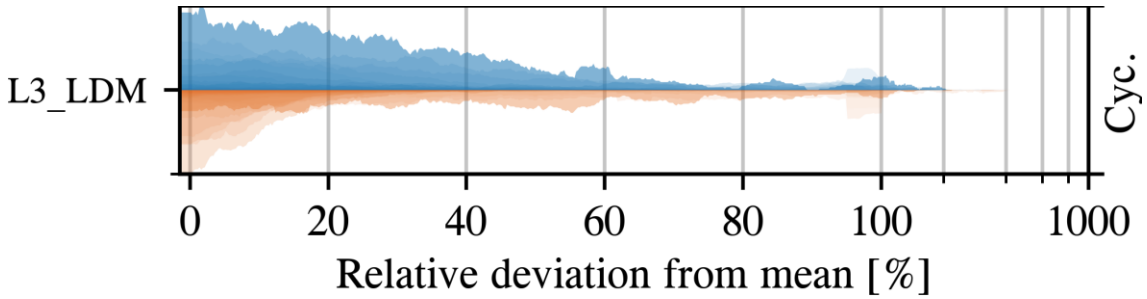
Counter strongly influenced by noise:

- Large distribution in the presence of noise, while short one in the absence



Counter with large deviation:

- Large distribution disregarding the noise
- Not suited for modeling



Evaluation Results: How to Interpret the Results

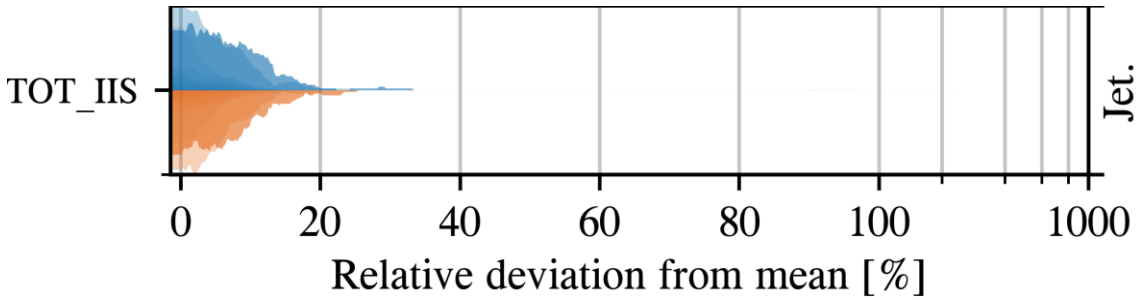
TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

Good Counters

Counter robust against noise:

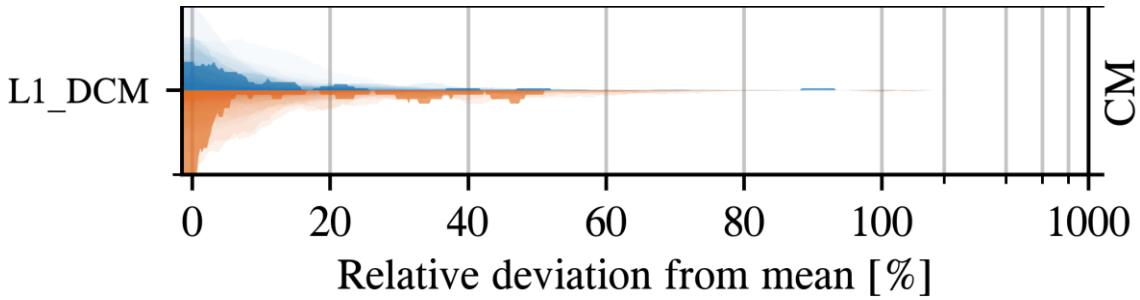
- Similar distribution in the presence and absence of noise
- Short deviation (< 20%)



OK Counters

Counter to some extent robust against noise:

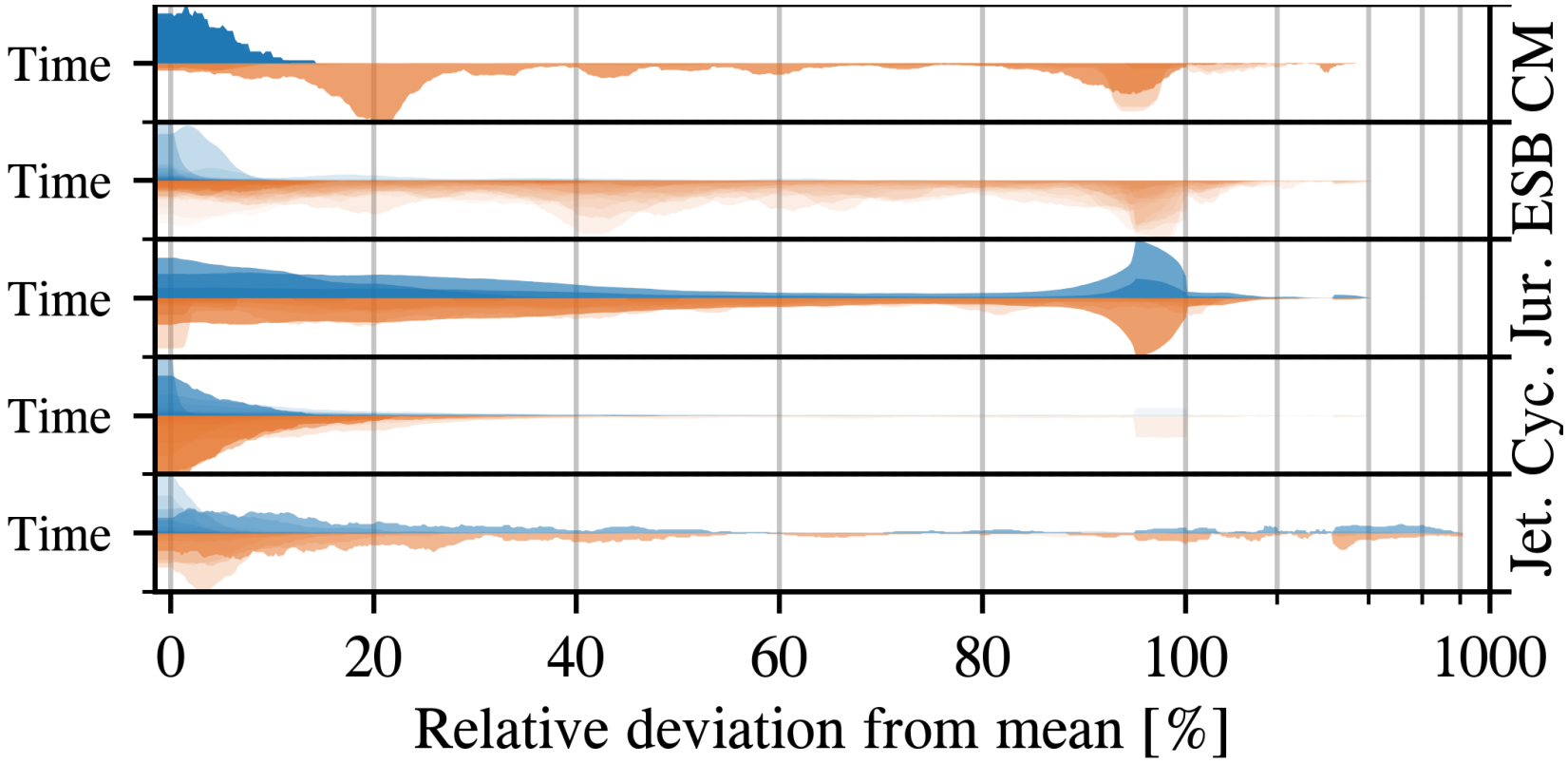
- Short deviation (< 20%) without noise
- Short deviation (< 20%) in the presence of noise for the **significant callpaths**



Evaluation Results: Runtime

TECHNISCHE UNIVERSITÄT DARMSTADT

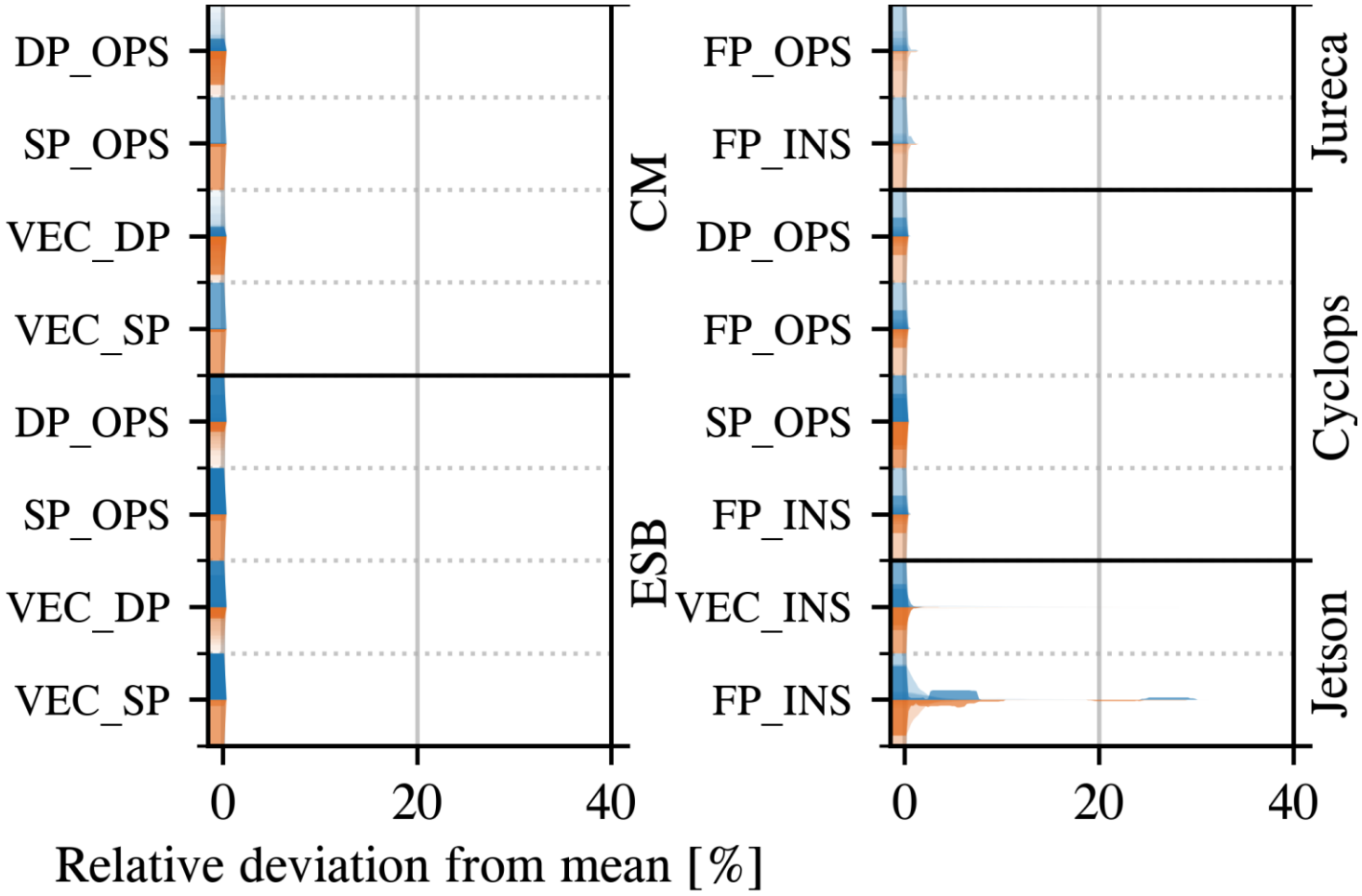
Parallel Programming



Evaluation Results: Floating Point Operations

TECHNISCHE UNIVERSITÄT DARMSTADT

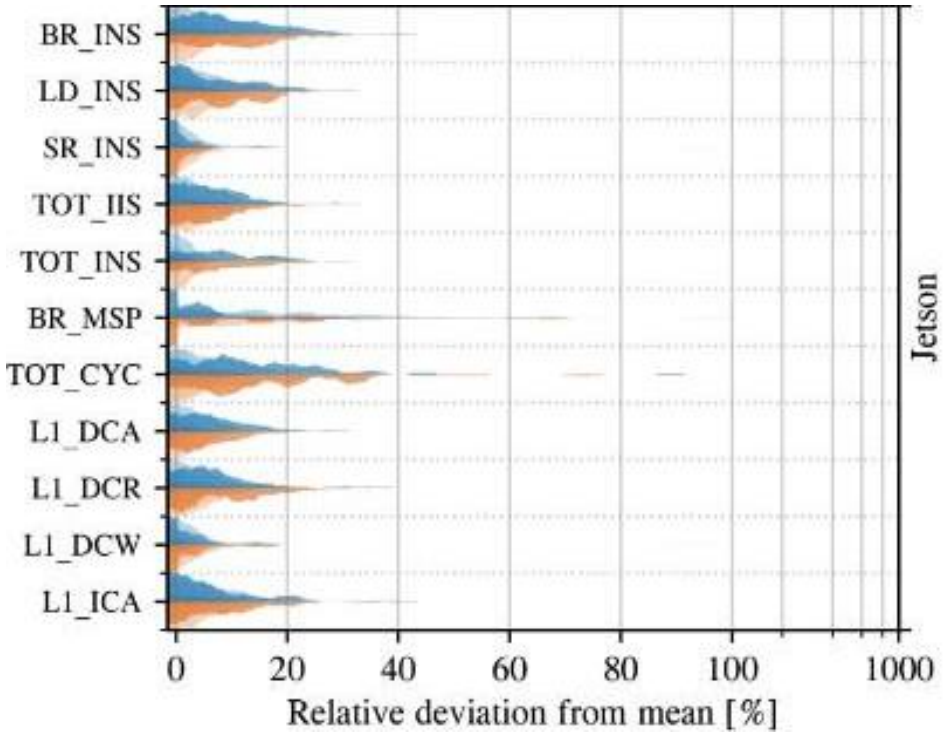
Parallel Programming

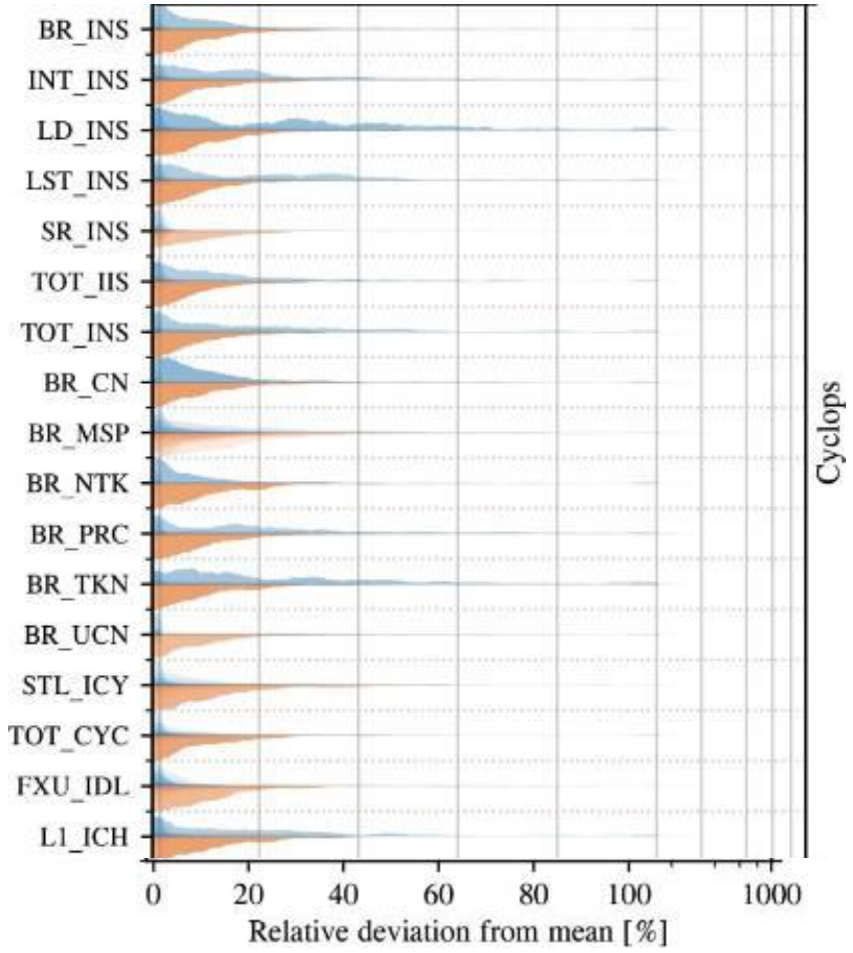


Evaluation Results: Branch Instructions, Cycles

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

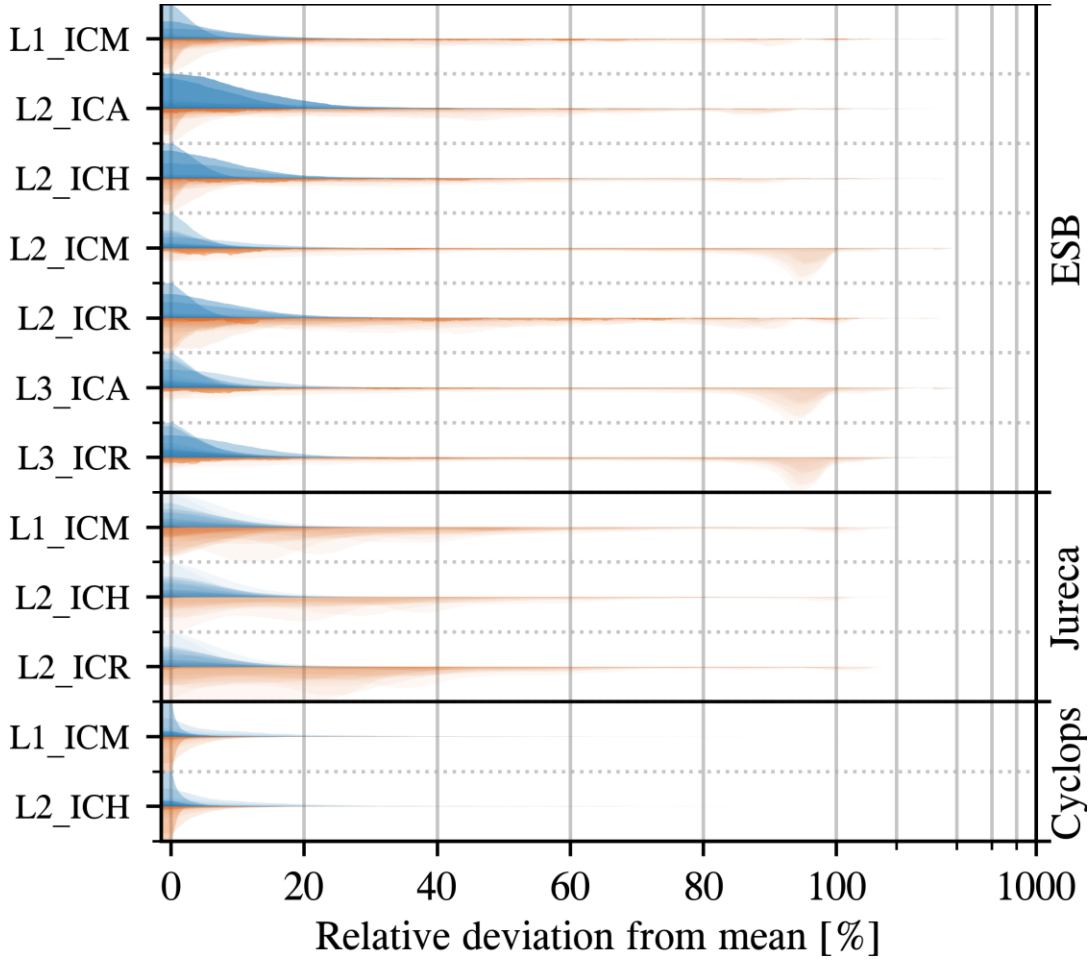


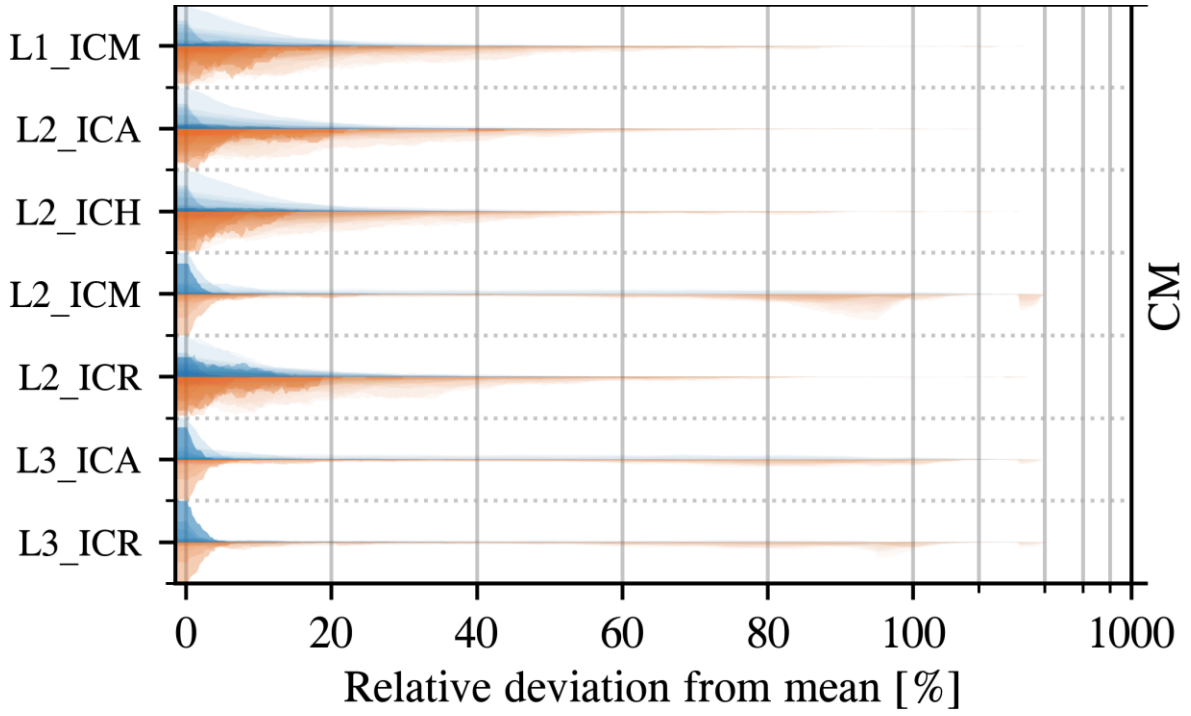


Evaluation Results: L1, L2 Instruction Cache

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

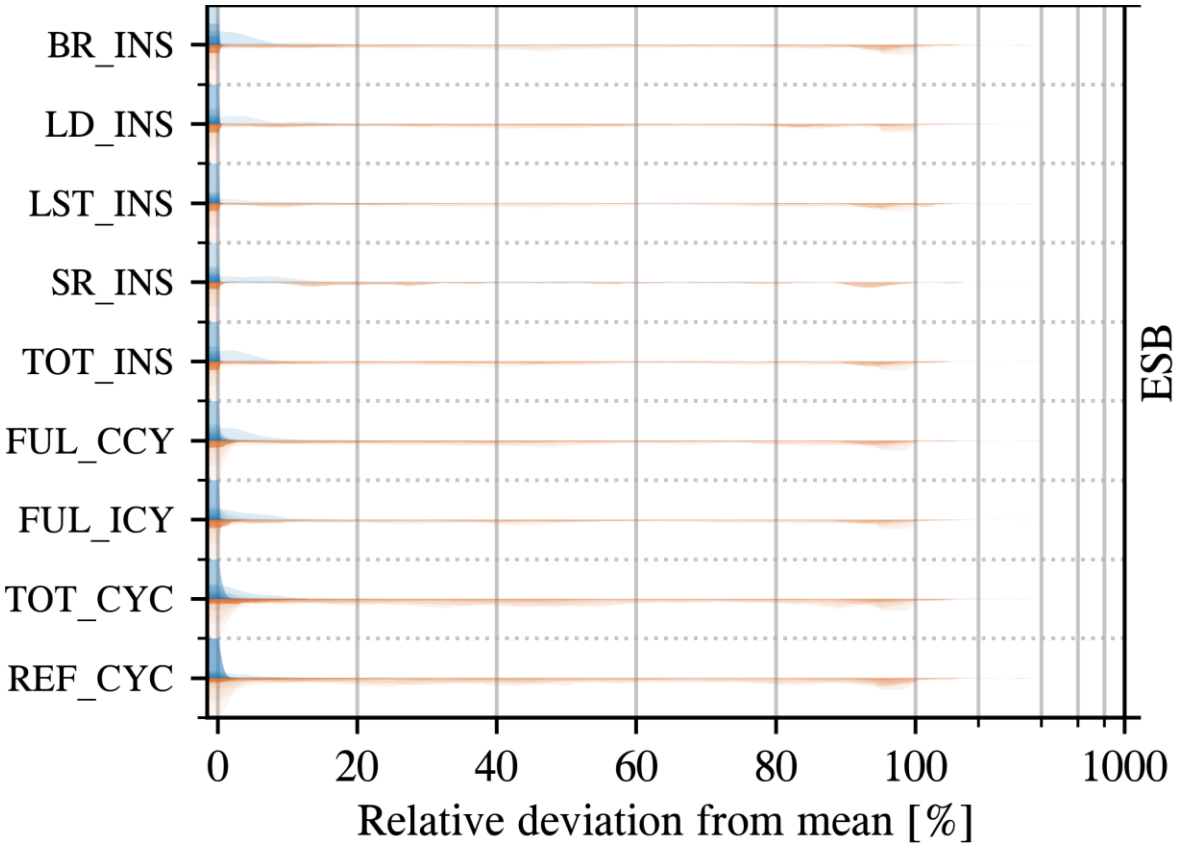


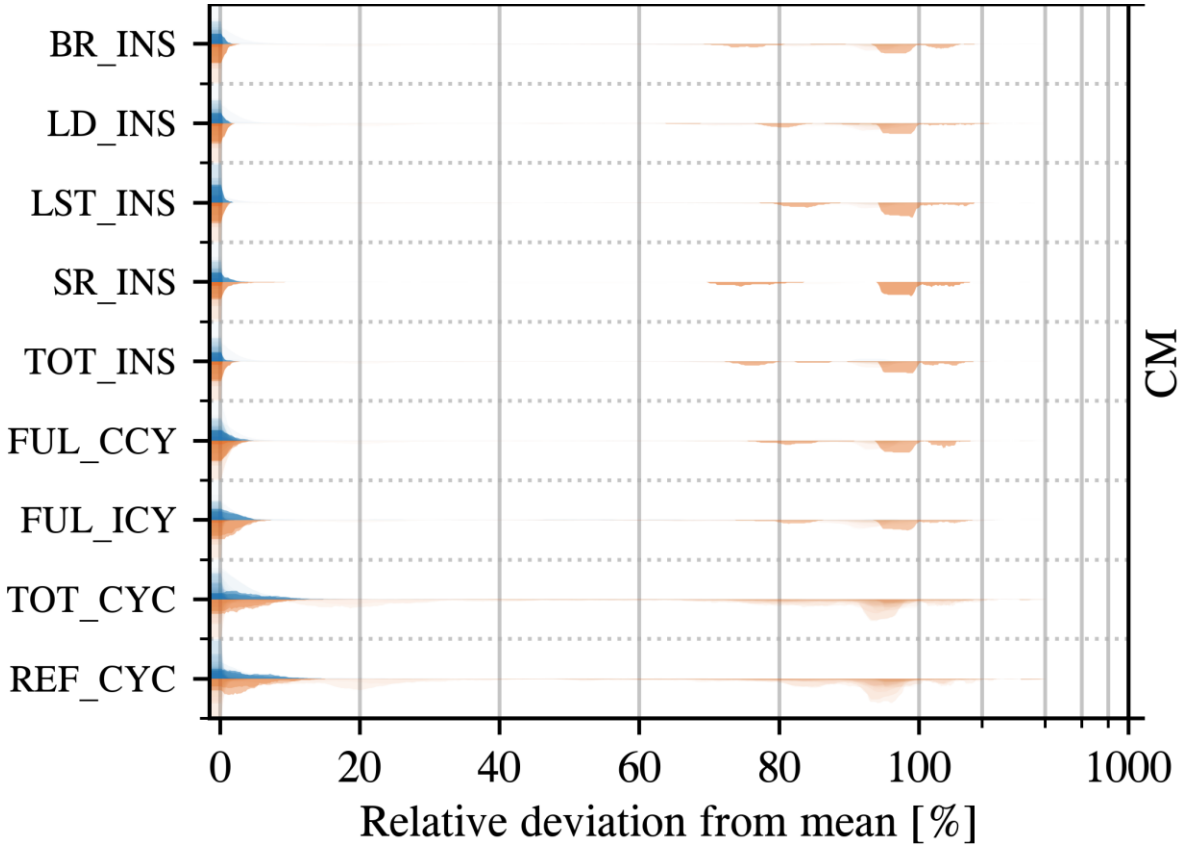


Evaluation Results: Branch Instructions, Cycles

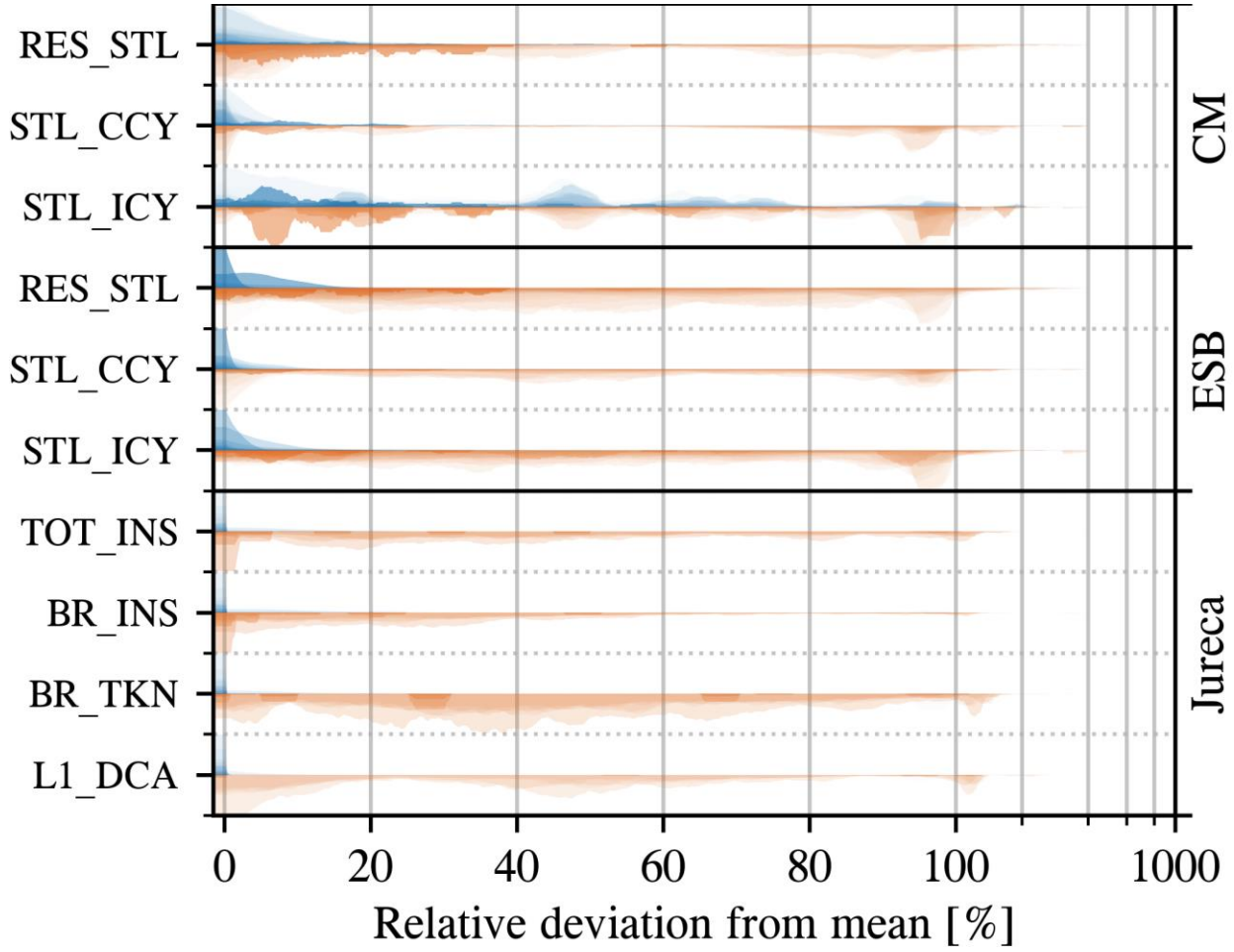
TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming





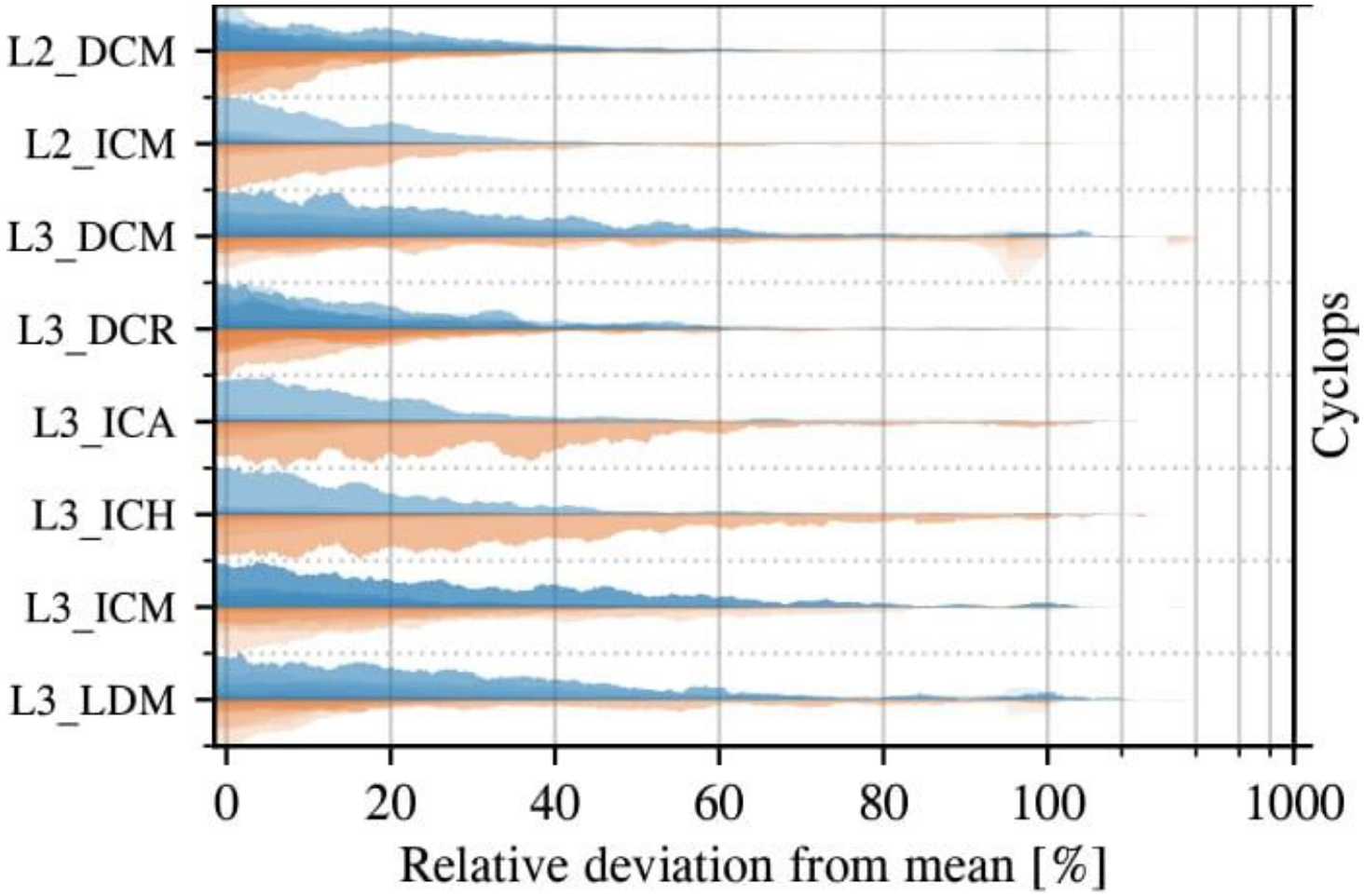
Evaluation Results: Stalls



Evaluation Results: L2, L3 Cache

TECHNISCHE UNIVERSITÄT DARMSTADT

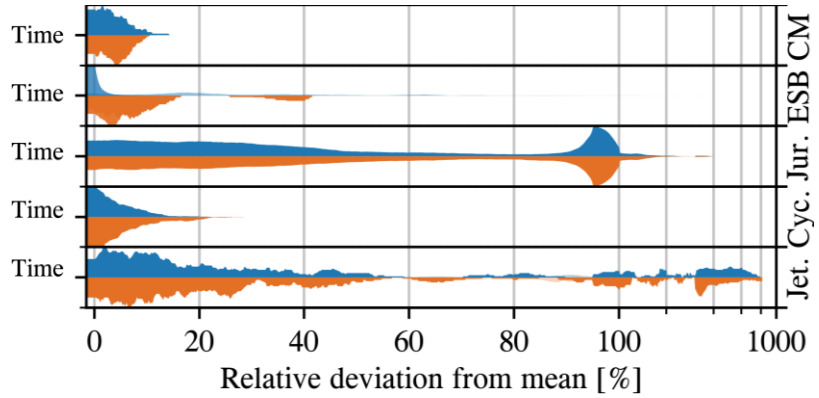
Parallel Programming

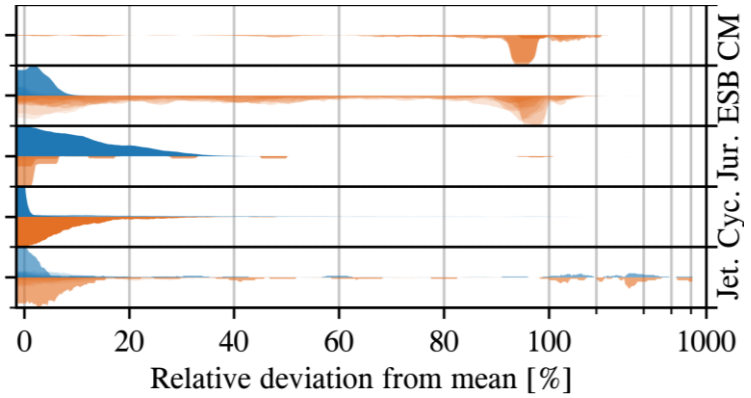


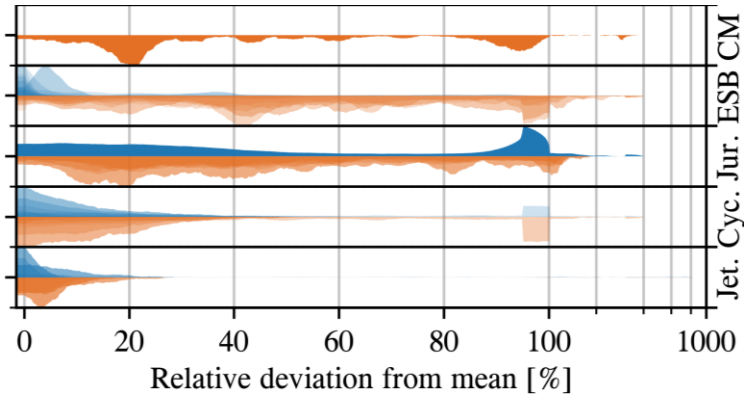
Impact of the Application: Total Cycles and Load Inst.

TECHNISCHE UNIVERSITÄT DARMSTADT

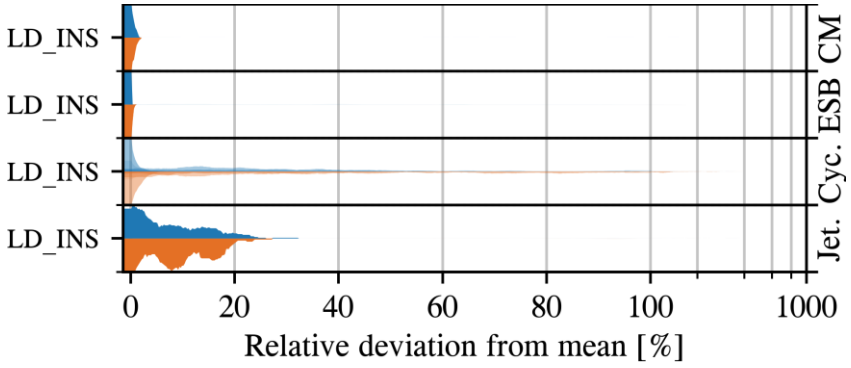
Parallel Programming



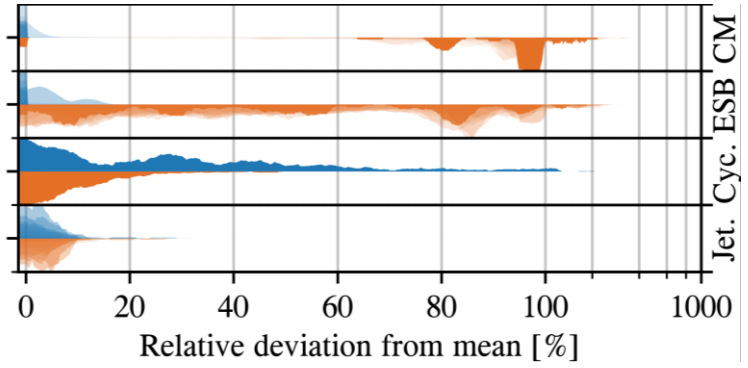




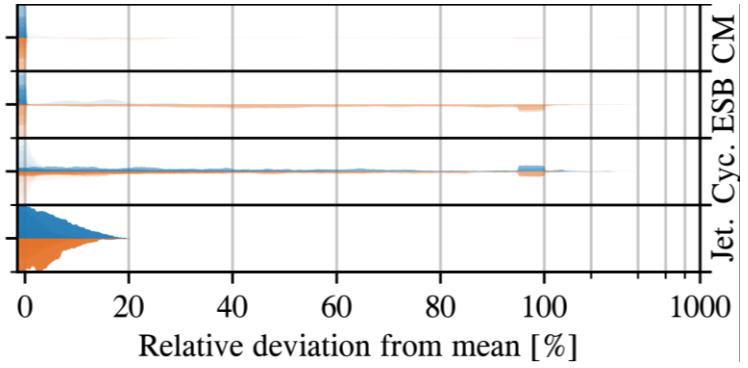
LAMMPS



LULESH



MiniFE



Best Practice User Guide General

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

In a noisy environment, the best counters independent of the system architecture are

- All counters measuring floating point operations, e.g., DP_OPS, VEC_SP
- All counters measuring floating point instructions, e.g., FP_INS

Best Practice User Guide Intel CPUs

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

Good

- Instructions, Cycles
  - Strong deviation for less important callpaths
  - Small deviation for important callpaths

Bad

- Stall and reference cycles
  - Because of large deviations
- L3 counters and TLB_DM
  - Accurate only in the absence of noise

Best Practice User Guide AMD CPUs

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

Good

- L2 cache accesses and misses, TLB_DM, TLB_IM
  - Have a small deviation with a maximum of 20%

Bad

- Total and branch instructions have less deviation than time
  - Still too much to be useful for modeling

Best Practice User Guide ARM CPUs

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

Good

- Instruction and cycle counters
  - Only small deviations in noisy environments
  - Can be used for performance modeling

Bad

- Exception: TOT_CYC
  - Showing a high deviation

Best Practice User Guide IBM CPUs

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

Good

- Most counters have a small deviation (less than 30%) for significant callpaths

Bad

- BR_TKN, BR_PRC, LD_INS, LST_INS
  - Should be avoided
- L3 related counters
  - High deviations in the presence of noise

Best Practice User Guide

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

	Jureca (AMD)	ESB (Intel)	CM (Intel)	Jetson (ARM)	Cyclops (IBM)
Floating point ops./instr.	++	++	++	++	++
Cycles	-	+	+	0	0
Instructions	-	+	+	+	+
L1	+	-	-	0	+
L2	+	0	-	0	+
L3		-	-		-

Conclusion

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

- Examined noise resilience of hardware counters on five systems with different architectures (Intel, AMD, ARM, IBM Power9)
- Analyzed all available preset and a selection of native events
- In general, most hardware counters are affected by noise, but still less than the runtime
- Counters measuring floating-point operations or instructions are noise resilient on all systems
- Their reliability significantly depends on the system architecture
- Our best practice guide enables developers to identify the relevant counters for performance analysis for their system

Future work

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

- Create a Tool that performs the analysis and evaluation on the system
- Examine more counters (i.e., native counters)
- Examine the correlation between the counters
- Identify the source of noise
- Started working with the PAPI developers

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

Questions?

Thank you for your attention!