Conquering Noise with Hardware Counters on HPC Systems

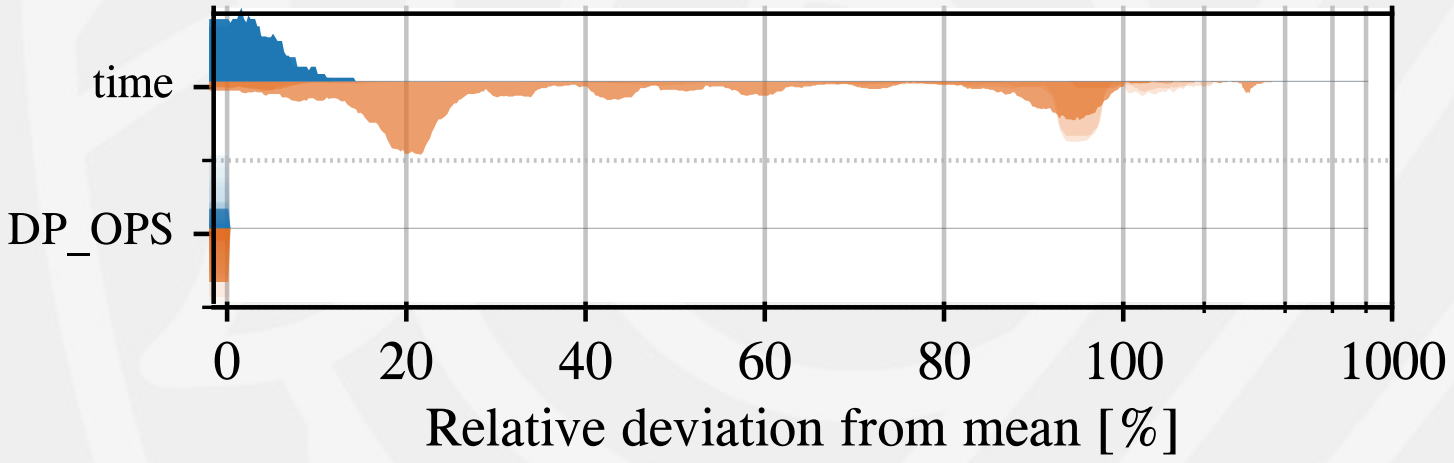
Marcus Ritter[1], Ahmad Tarraf[1], Alexander Geiß[1], Nour Daoud[2], Bernd Mohr[2], **Felix Wolf**[1]

[1]Technical University of Darmstadt

[2]Forschungszentrum Jülich GmbH

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming



EXTRA NOISE

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 **performance bottlenecks** early

→ Use **empirical performance modeling** (e.g., Extra-P)

Problem:

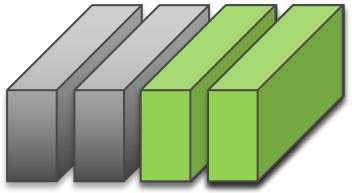
In **noisy** environments → difficult to create accurate performance models

- Strong run-to-run variations of the underlying measurements
- Irreproducible and misleading
- Deviation from *intrinsic* application behavior

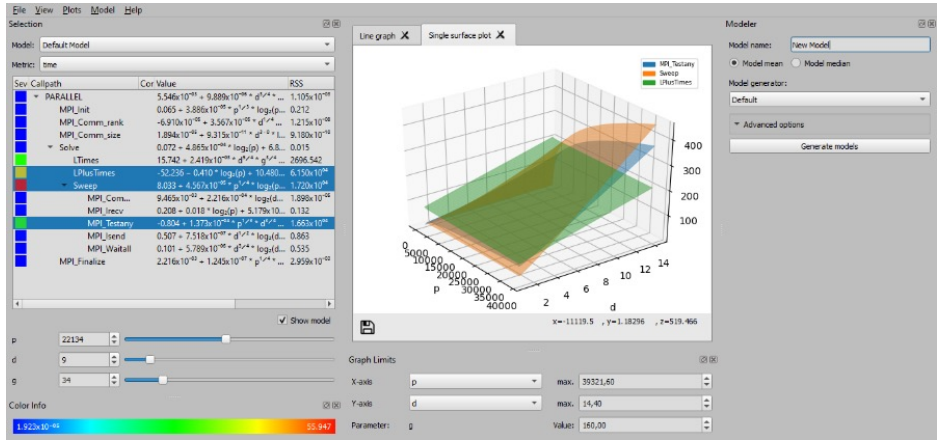
Empirical performance modeling

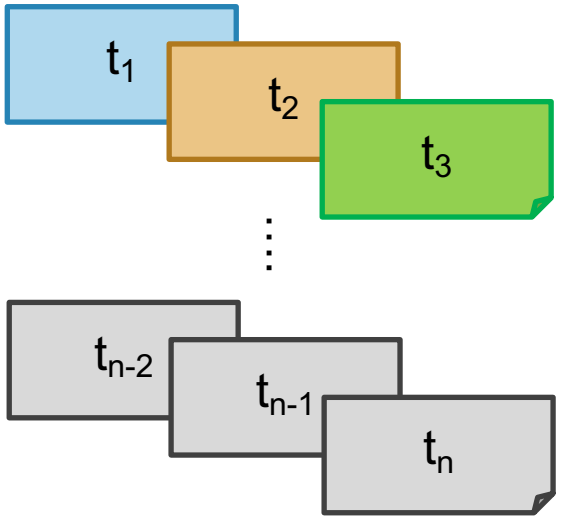
TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming



Performance measurements with different execution parameters $\mathbf{x_1,\ldots,x_n}$





Machine learning

$$t = f(x_1, \ldots, x_n)$$

Alternative metrics:
Hard- and software counters

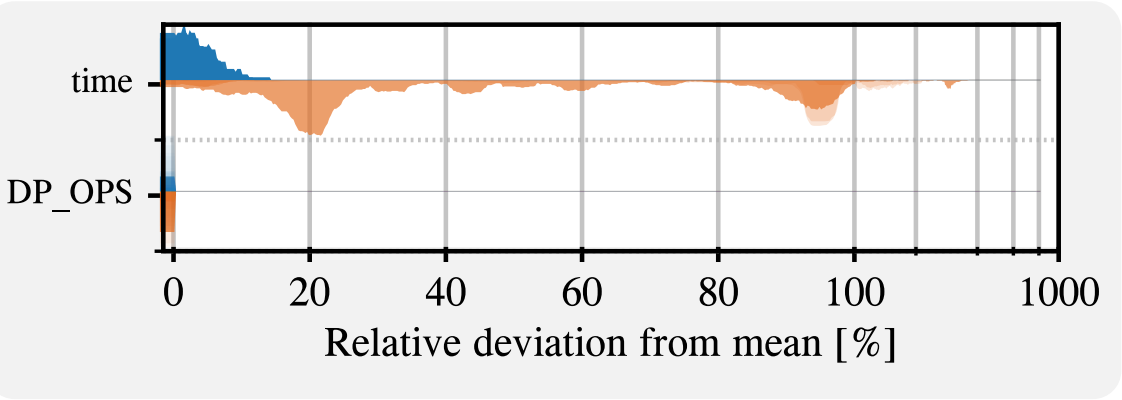
Motivation

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., floating-point operations
- Selecting the right counters requires a thorough analysis
- Once found, the counters can be used for performance modeling

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
  - Using NOIGENA (NOIse GENerator Application) developed in Jülich
  - NOIGENA processes were running on the odd processors
  - Inject different noise patterns using NOIGENA

NOIse GENerator Application

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

- Produces shared-resource contention
- Uses several benchmarks
  - Memory: Stream
  - Network: FzjLinkTest
  - I/O: IOR
- Consecutively runs configurable patterns

```
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

- **Instrument all call paths** of the applications with **Score-P** using **PAPI**
- We do the analysis **for each counter,** on **all systems, with and without injected noise** to get the Score-P measurements for the:
  - a^{th} application kernel (call path),
  - p^{th} MPI rank,
  - t^{th} OpenMP thread, and
  - i^{th} repetition:

a counter value $v_{a,p,t,i}$

Score-P
Scalable performance measurement infrastructure for parallel codes

Analysis Methodology

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

Compare counter values across the **repeated experiments** for each a^{th} application kernel (call path), p^{th} MPI rank, and t^{th} OpenMP thread

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

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

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

$$\frac{\left|v_{a,p,t,i} - \bar{v}_{a,p,t}\right|}{\bar{v}_{a,p,t}} * 100\%$$

→ Use this metric to compare the counter results

Application Benchmarks

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 used 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	$n = 1$, $p = [4, 8, 12, 16, 20]$, $t = 4p$, $s = 20^3 p$

s: problem size
n: number of computational nodes
p: the number of MPI processes
t: the number of OpenMP thread

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 configurations
- Several repetitions 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

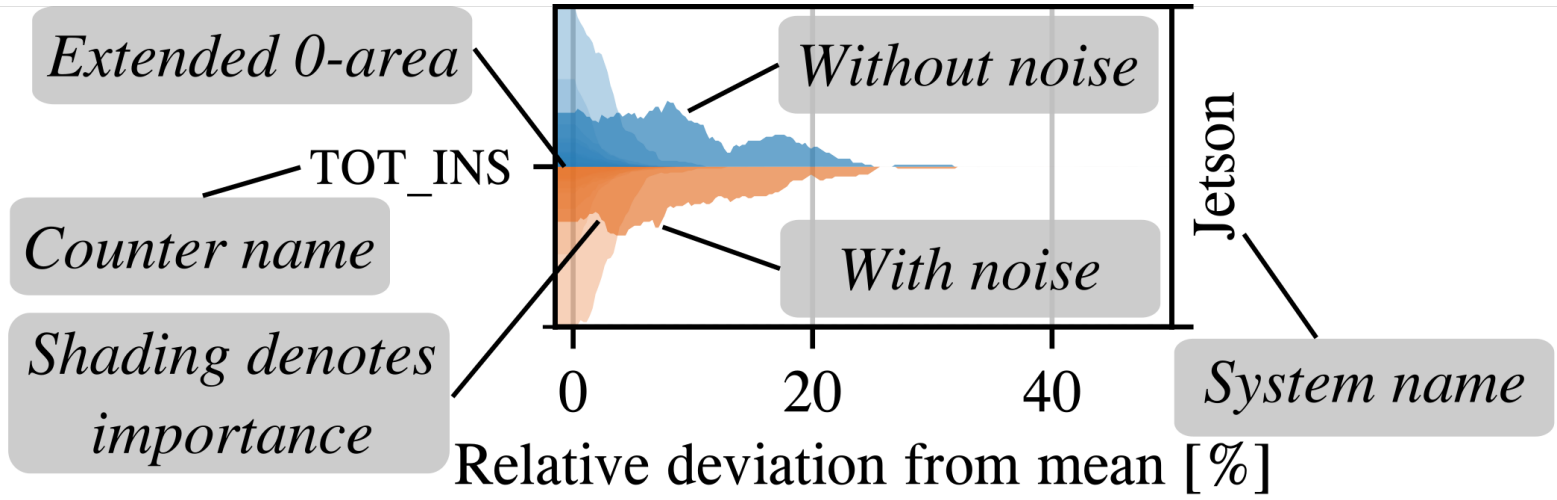
To compare distinct counter → Scale the plots with the peak occurrence of the relative deviation

Evaluation Results

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

- Height = how often the corresponding relative deviation occurs
- Intensity = share of the total counter value for that call path (importance)
- Only call paths with > 1% of the total counter value are visible



Evaluation Results: How to Interpret the Results

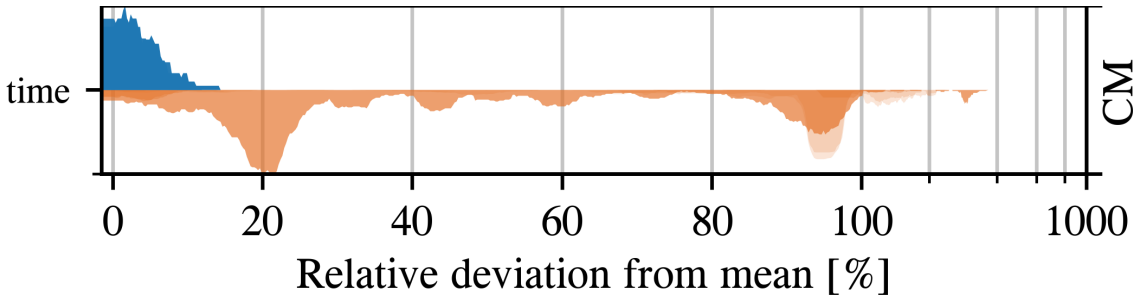
TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

Bad Counters [✗]

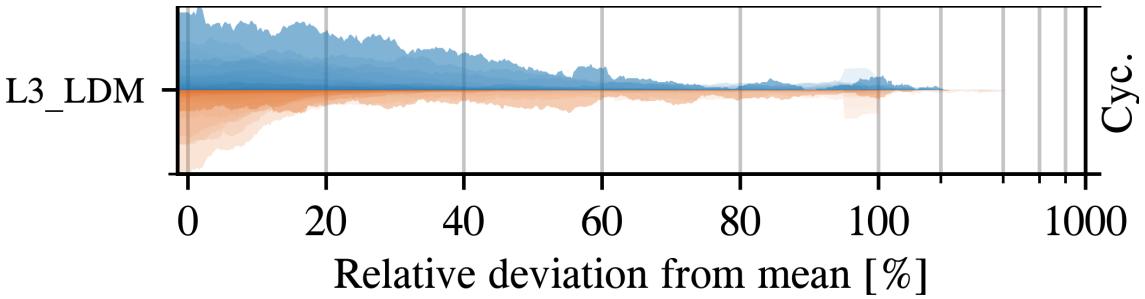
Counter strongly influenced by noise:

- Large distribution in the presence of noise, small one in the absence



Counter with large deviation:

- Large distribution disregarding the noise
- Not suited for modeling



Evaluation Results: How to Interpret the Results

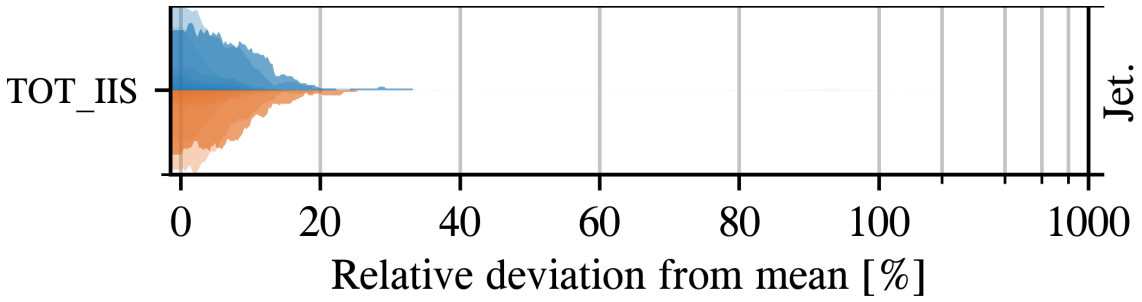
TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

Good Counters [+]

Counter robust against noise:

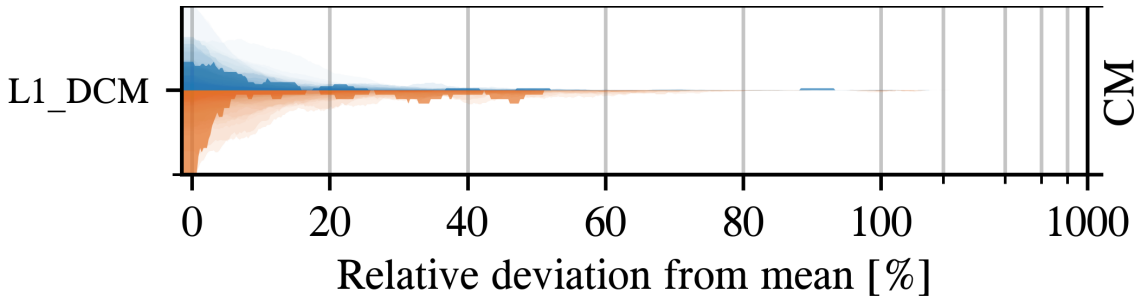
- Similar distribution in the presence and absence of noise
- Small deviation (< 20%)



OK Counters [0]

Counter to some extent robust against noise:

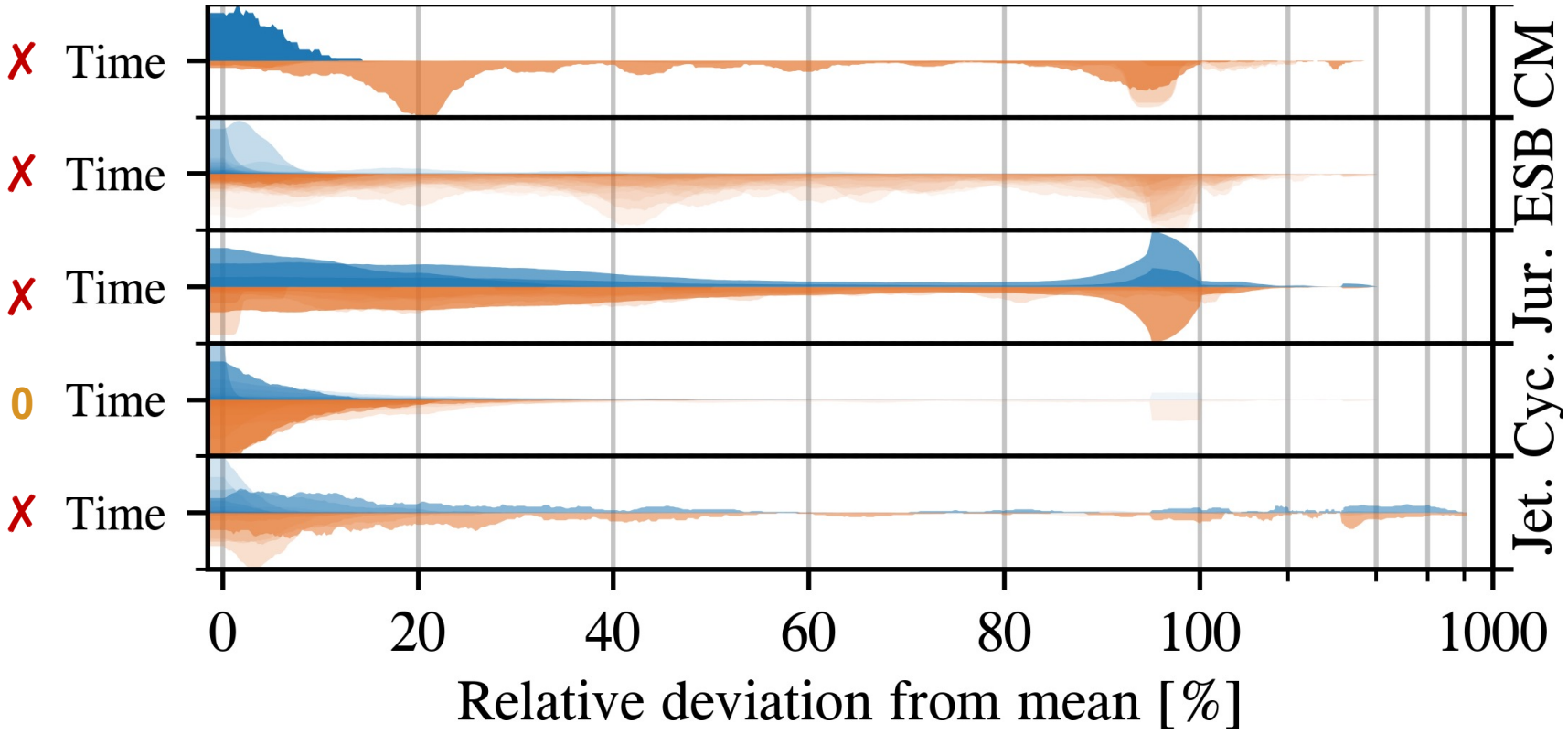
- Small deviation (< 20%) without noise
- Small deviation (< 20%) in the presence of noise for **significant call paths**



Evaluation Results: Runtime

TECHNISCHE UNIVERSITÄT DARMSTADT

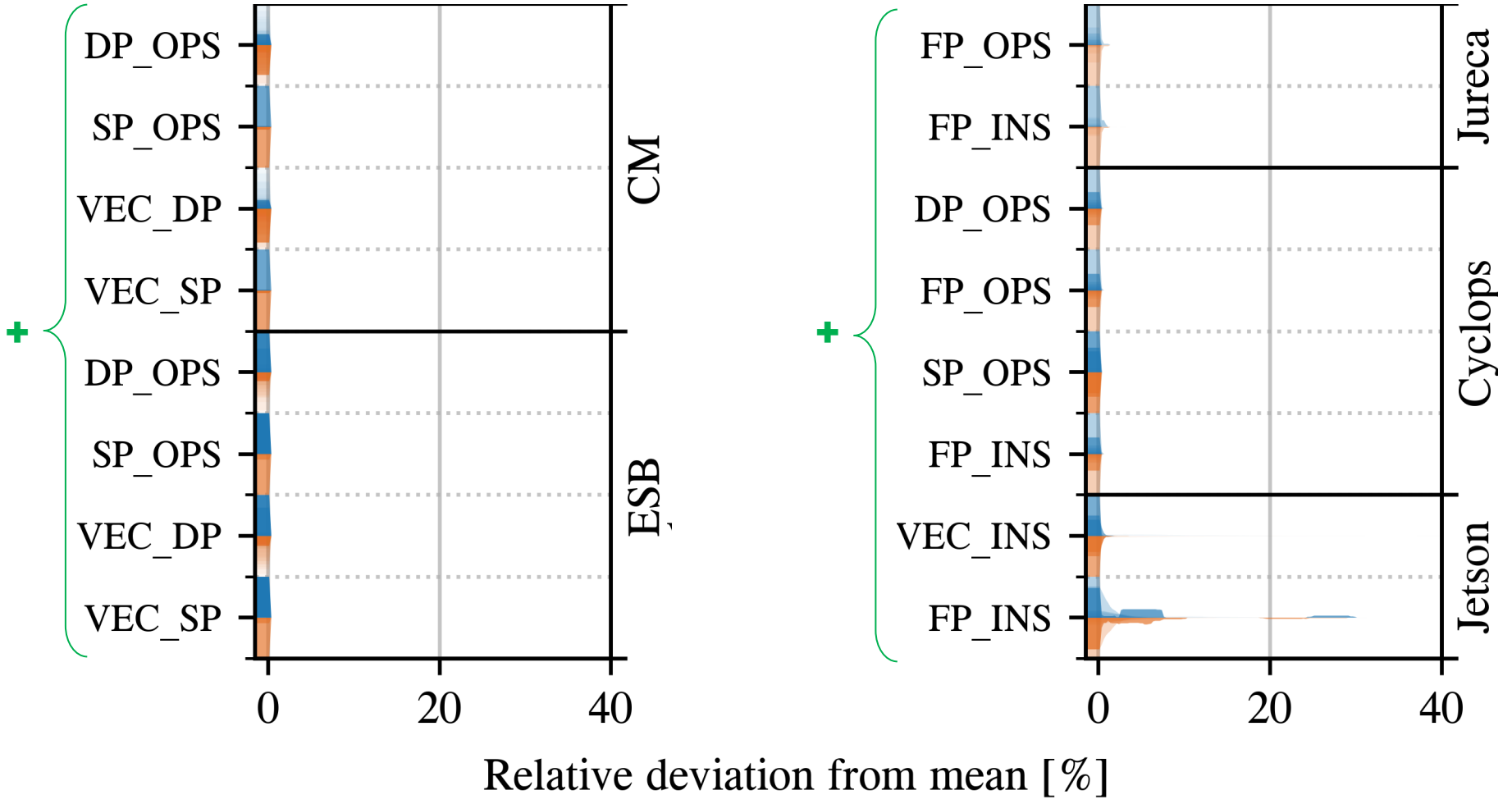
Parallel Programming



Evaluation Results: Floating Point Operations

TECHNISCHE UNIVERSITÄT DARMSTADT

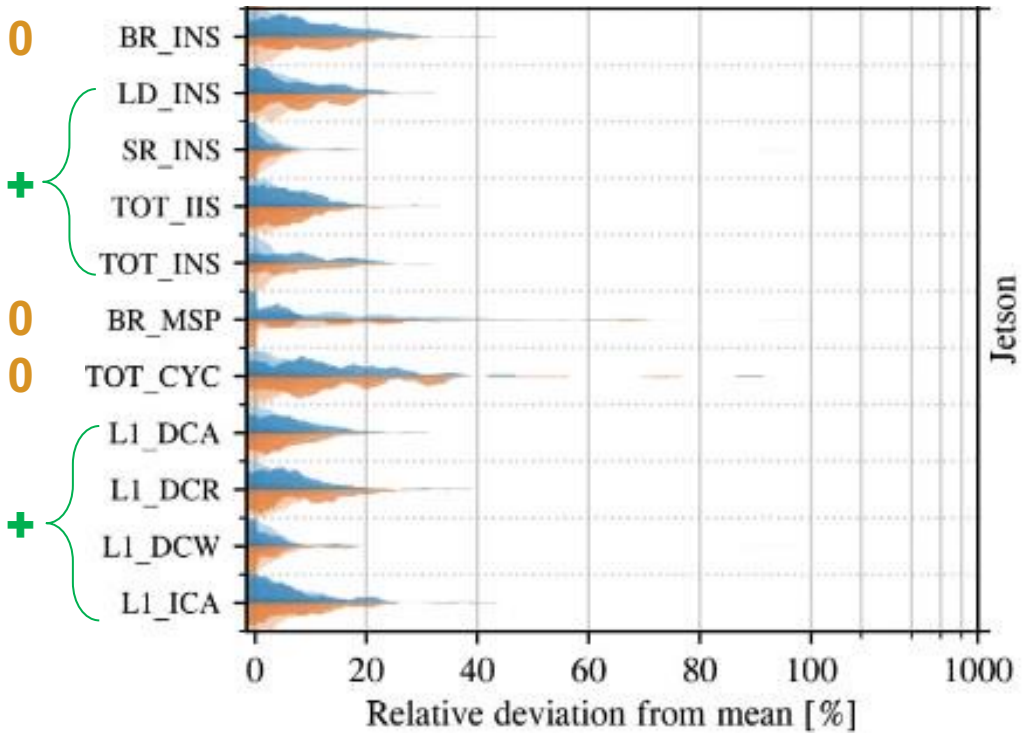
Parallel Programming

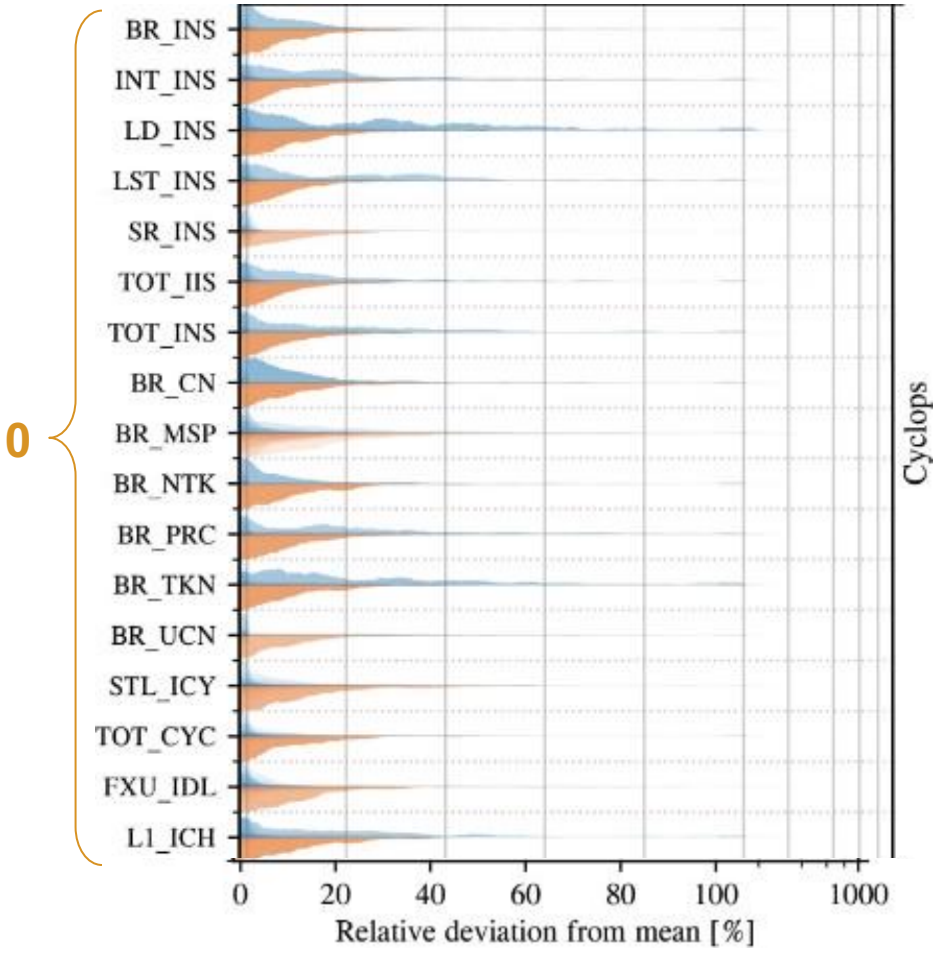


Evaluation Results: Instructions, Cycles

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

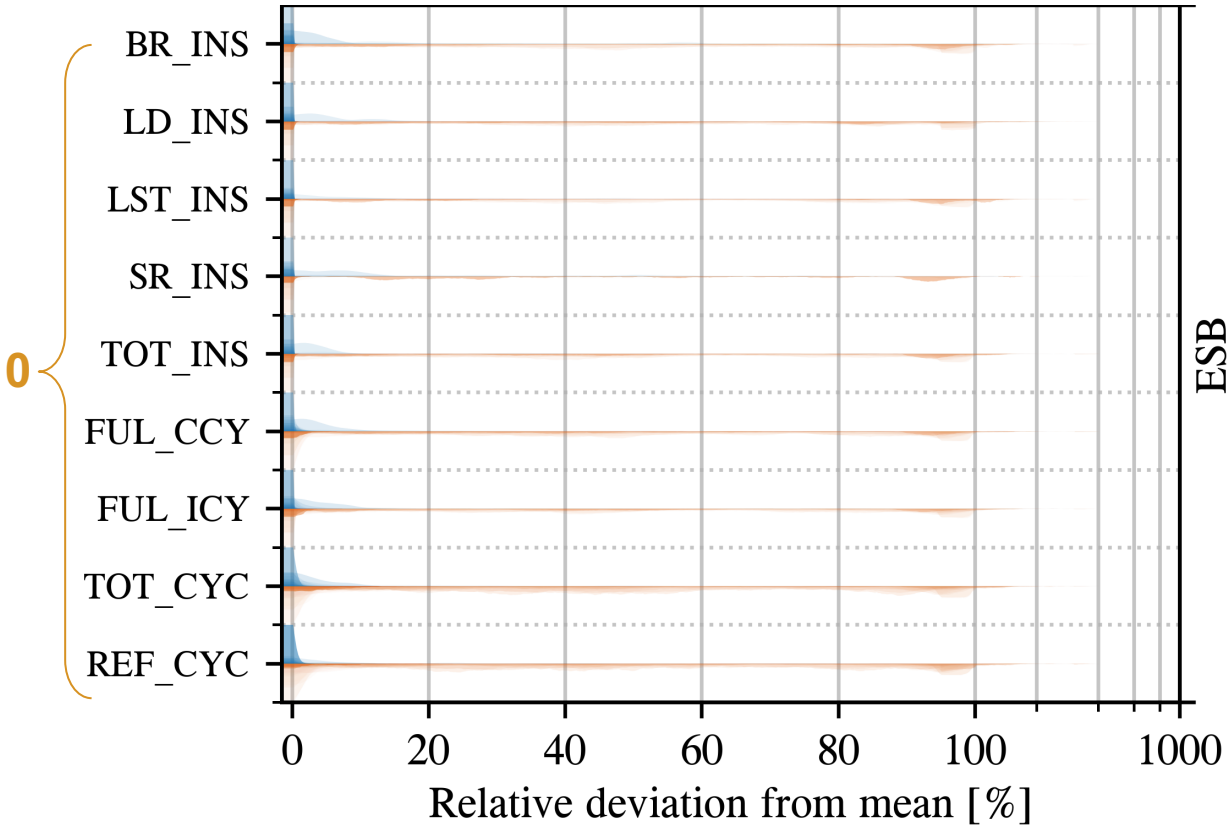


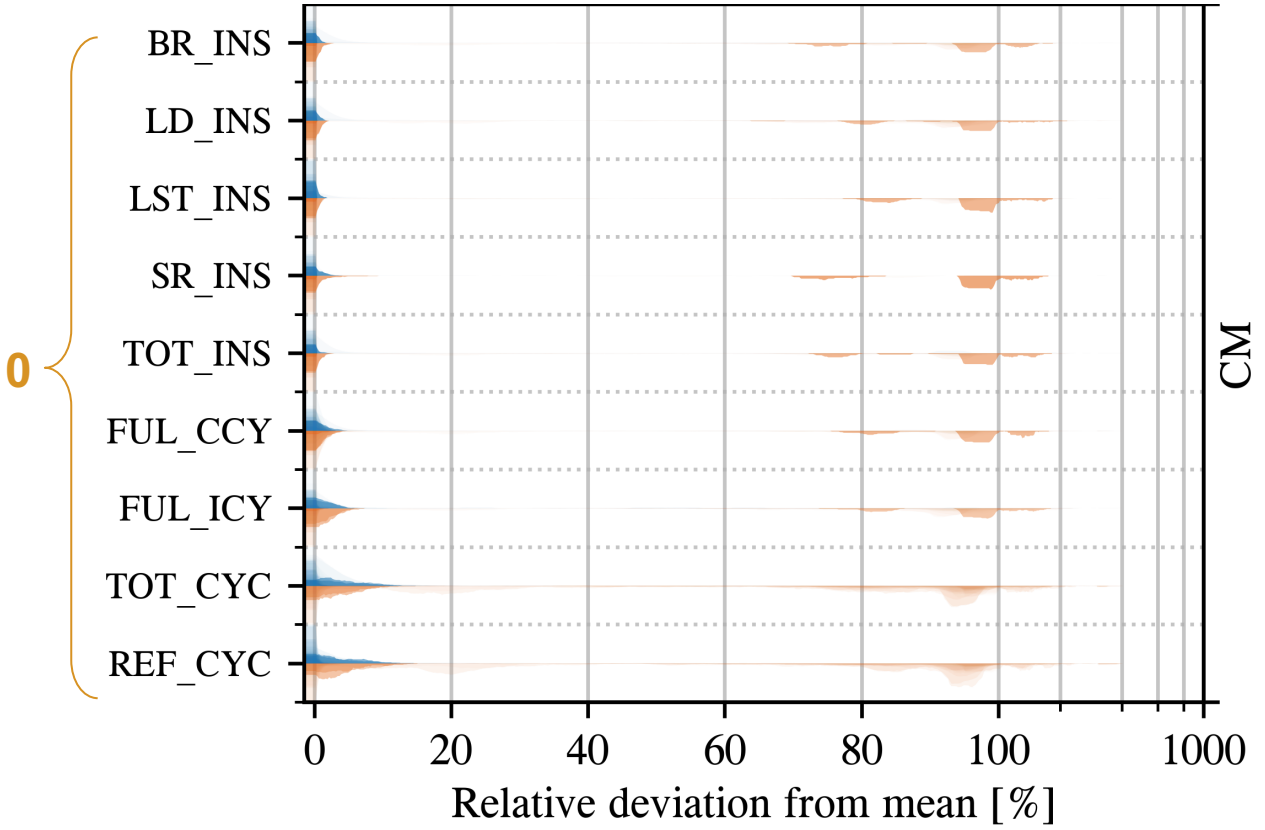


Evaluation Results: Instructions, Cycles

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

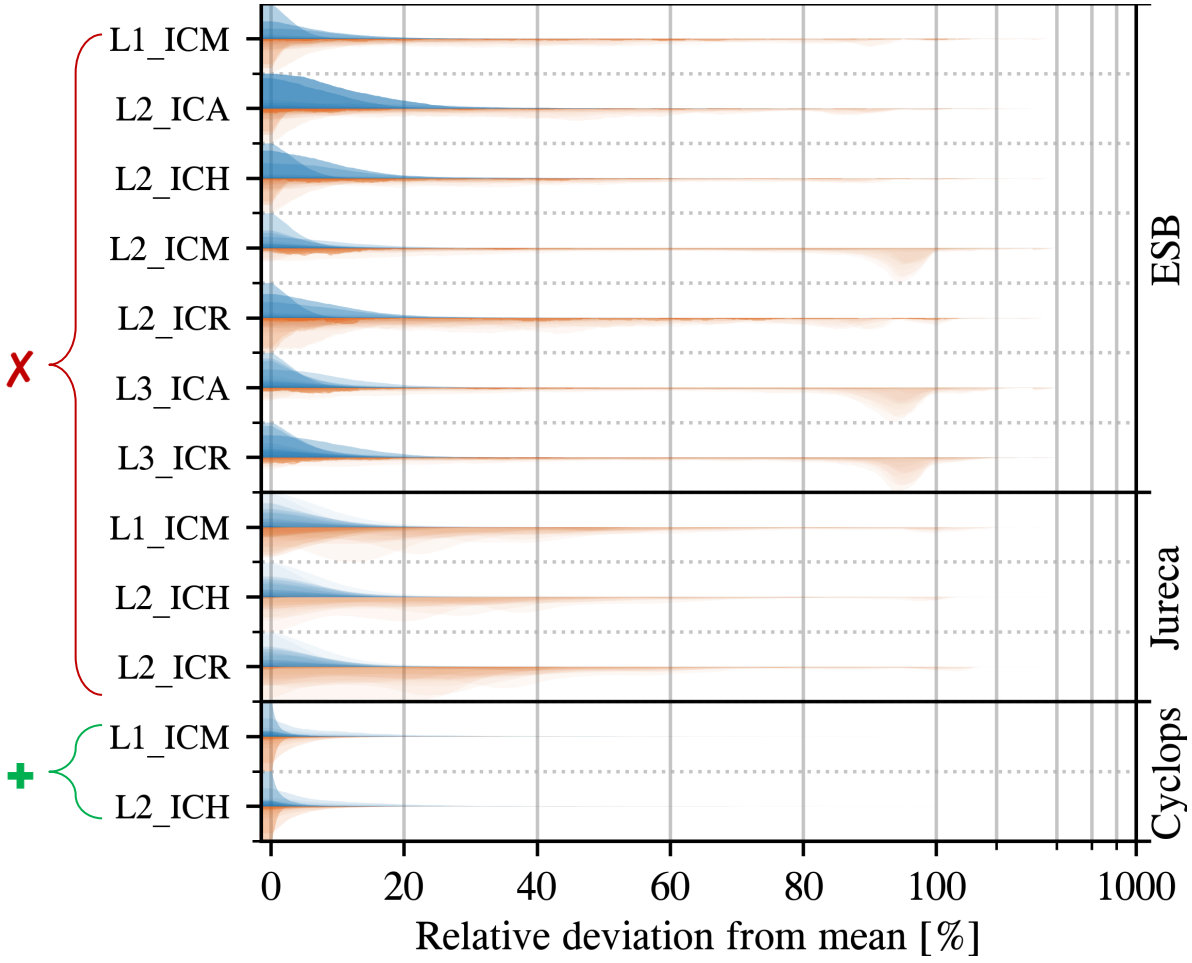


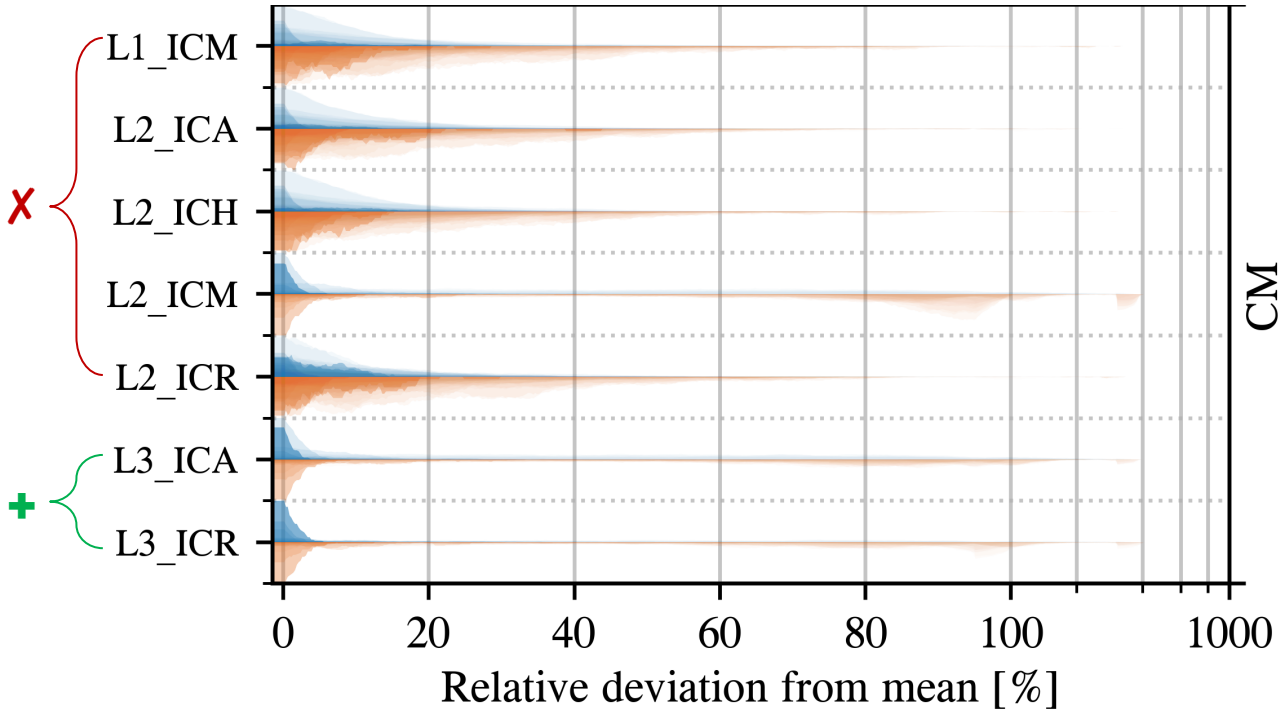


Evaluation Results: L1, L2 Instruction Cache

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

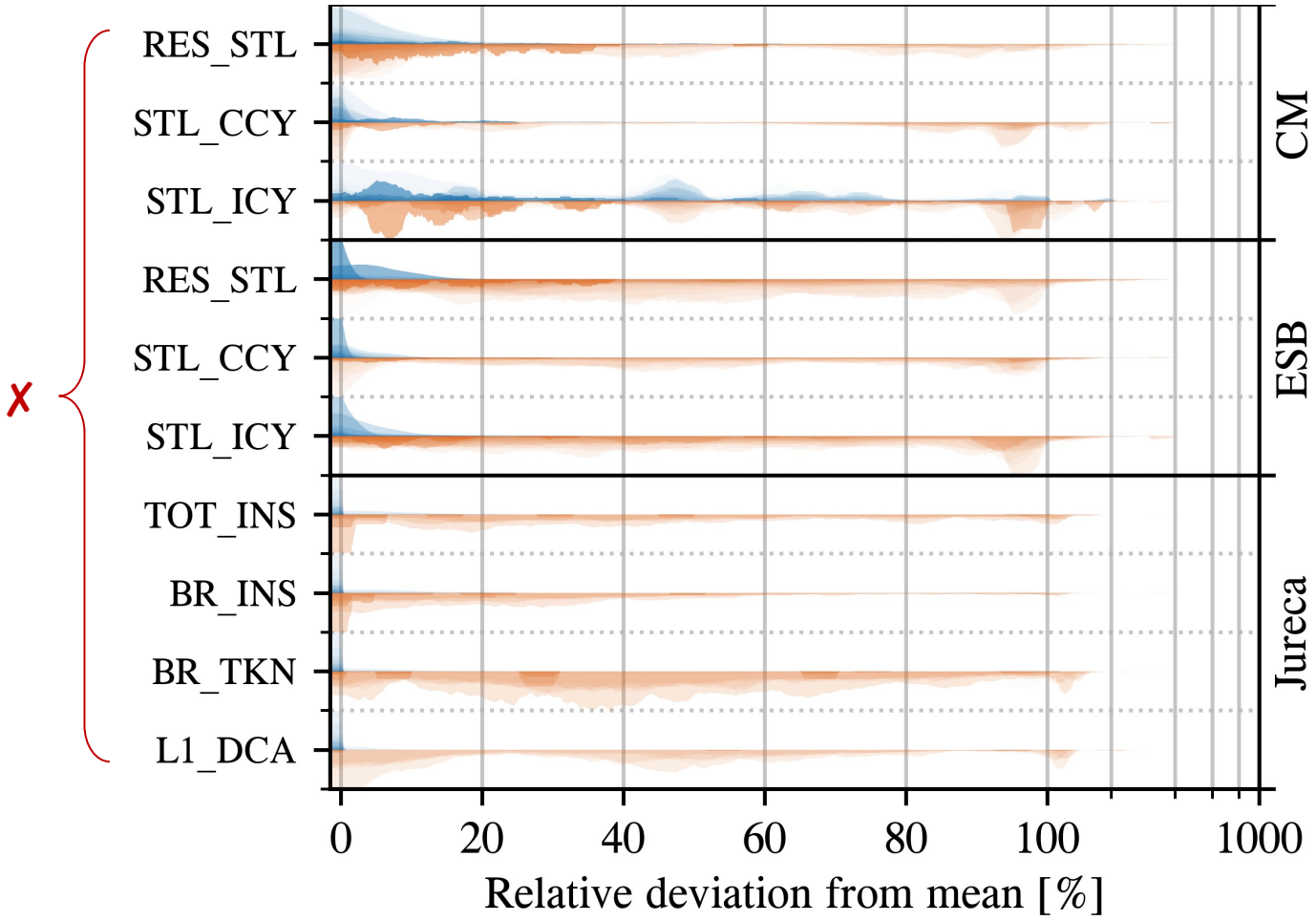




Evaluation Results: Stalls & Reference Cycles

TECHNISCHE UNIVERSITÄT DARMSTADT

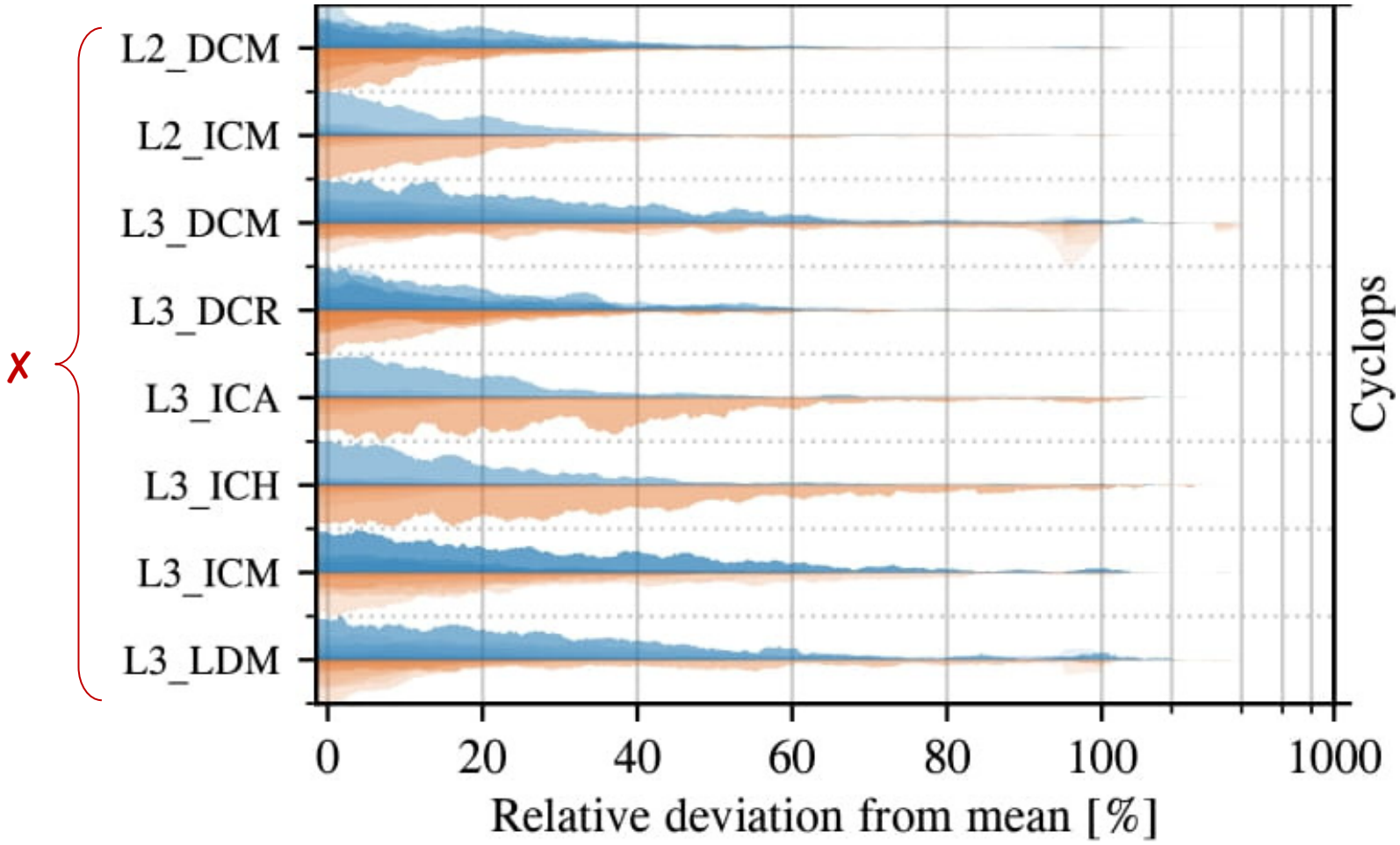
Parallel Programming



Evaluation Results: L2, L3 Cache

TECHNISCHE UNIVERSITÄT DARMSTADT

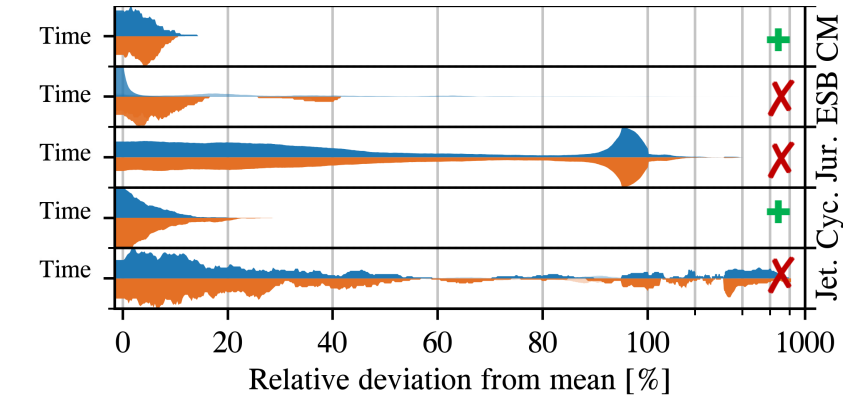
Parallel Programming

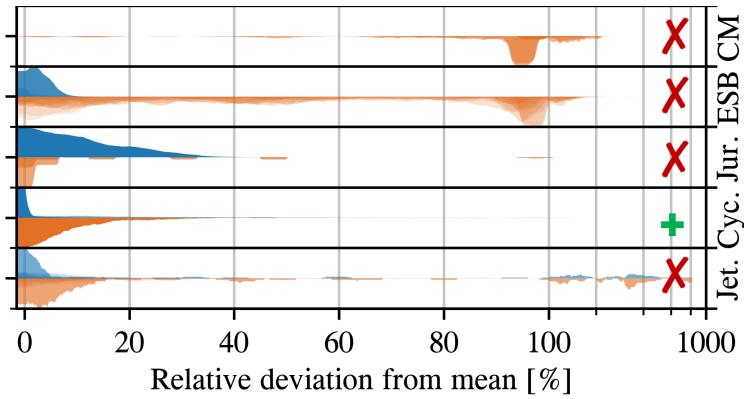


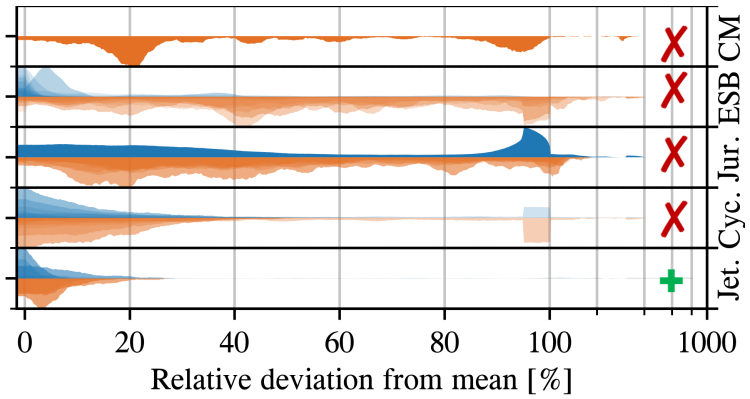
Impact of the Application: Time and Load Inst.

TECHNISCHE UNIVERSITÄT DARMSTADT

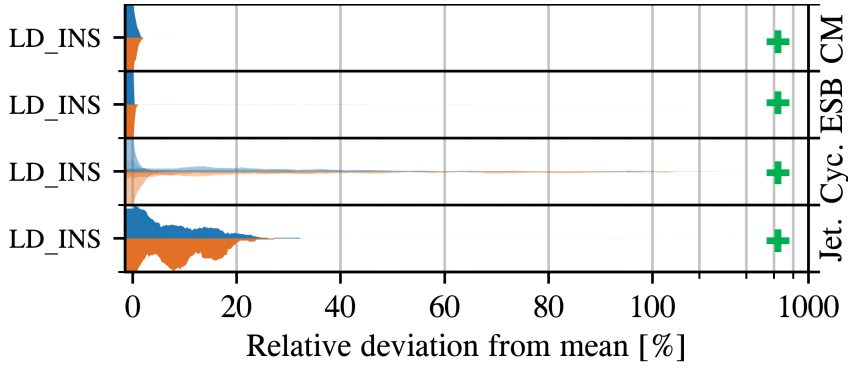
Parallel Programming



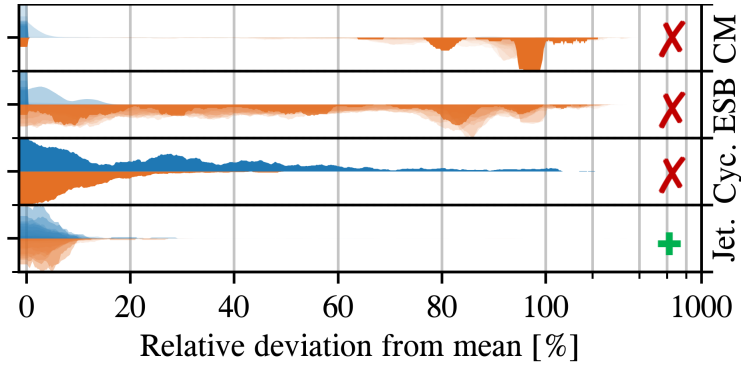




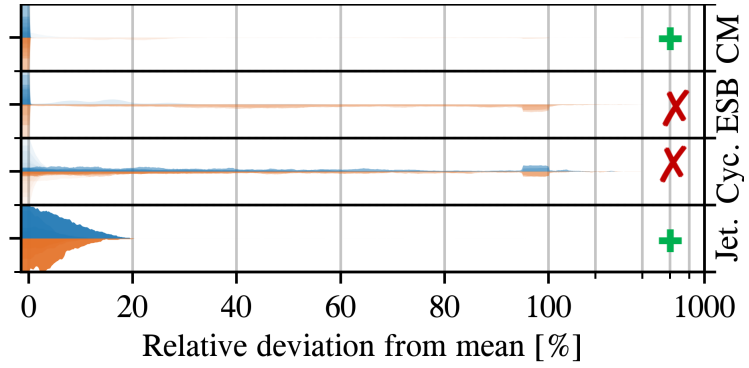
LAMMPS



LULESH



MiniFE



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		✗	✗		✗

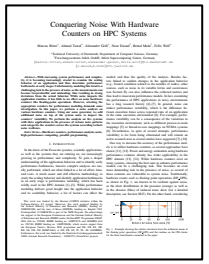
Legend	
++	very good
+	good
0	ok
✗	bad

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 presets and a selection of native events
- Most hardware counters are affected by noise, but still less than the runtime
- Floating-point operations or instructions are noise resilient on all systems
- Variability significantly depends on the system architecture
- Best practice guide helps choose suitable counters for given system


Conquering Noise With Hardware Counters on HPC Systems

Marcus Ritter, Ahmad Tarraf, Alexander Geiß, Nour Daoud, Bernd Mohr, Felix Wolf: Conquering Noise With Hardware Counters on HPC Systems. In *Proc. of the Workshop on Programming and Performance Visualization Tools (ProTools), held in conjunction with the Supercomputing Conference (SC22)*, pages 1–10, IEEE, 2022.

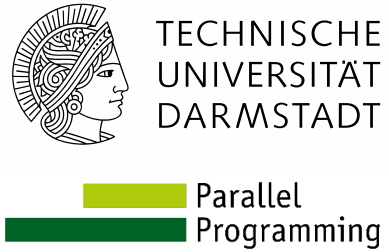
Future Work

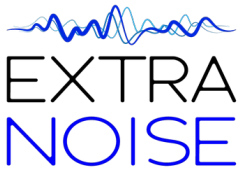
TECHNISCHE UNIVERSITÄT DARMSTADT

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 variation
- Started working with the PAPI developers

Acknowledgment





Federal Ministry
of Education
and Research

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel Programming

Questions?

Thank you for your attention!