NOISE-RESILIENT PERFORMANCE MEASUREMENT AND ANALYSIS OF HPC APPLICATIONS WITH SCORE-P + SCALASCA

JUNE 20, 2022 I BERND MOHR

Mitglied der Helmholtz-Gemeinschaft

EXTRANOISE

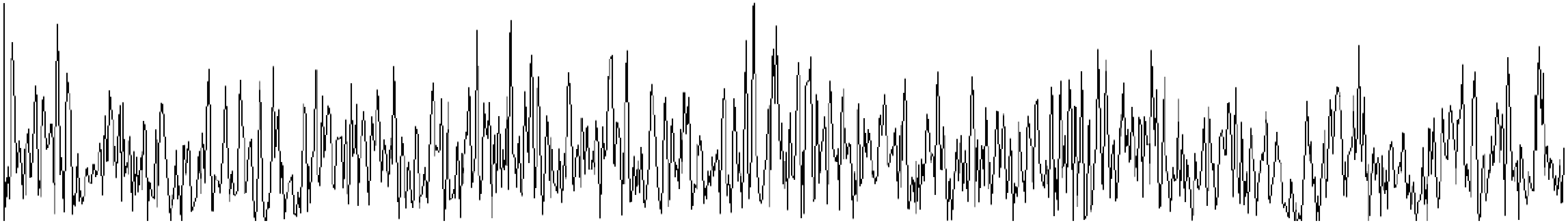
JÜLICH Forschungszentrum | JÜLICH SUPERCOMPUTING CENTRE

The DFG Project ExtraNoise

NOISE

- On many systems, execution times show many huge run-to-run variation
  - Often between 5% to 30%, but higher values have been reported too

- Sources
  - Node level: operation system, dynamic frequency scaling, manufacturing variability, shared resources like caches, memory channels or NICs
  - System level: network and file-system congestion



PROBLEM FOR PERFORMANCE ANALYSIS

- Goal
  - Understand performance behavior to identify optimization opportunities
  - Often based on performance measurements
- In noisy environments
  - Several repetitions required
  - Trends derived with statistical methods
  - Reproducibility?

- Problem
  - Expensive
  - Potentially misleading because variations may follow irregular patterns

PROJECT OBJECTIVES

- Make application performance analysis on noisy systems both cheaper and more reliable → noise resilient
- Improve typical performance analysis techniques
  - Raw performance measurements (profiling, tracing) → **Score-P**
  - Trace analysis → **Scalasca**
  - Empirical performance modeling → **Extra-P**
- Better understand noise patterns and noise sensitivity of applications
- Derive strategies of how to lower the active and passive interference potential of applications

Score-P

Scalable performance measurement infrastructure for parallel codes

- Community-developed open-source
- Replaced tool-specific instrumentation and measurement components of partners
- http://www.score-p.org

TECHNISCHE UNIVERSITÄT DRESDEN

JÜLICH Forschungszentrum | JÜLICH SUPERCOMPUTING CENTRE

German Research School for Simulation Sciences

RWTH AACHEN UNIVERSITY

TUM Technische Universität München

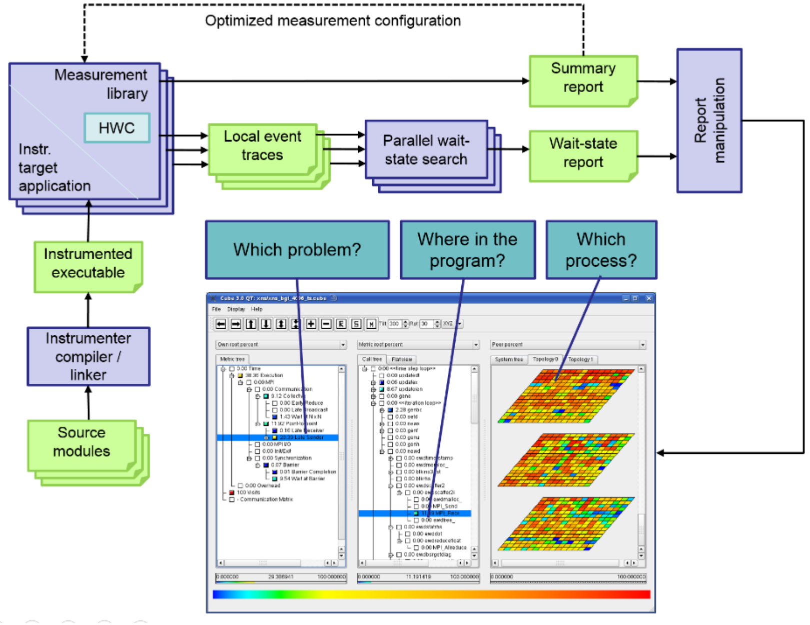
UNIVERSITY OF OREGON

SCALASCA

http://www.scalasca.org/

- Scalable Analysis of Large Scale Applications

- Approach
  - **Instrument** C, C++, and Fortran parallel applications (with Score-P)
  - Option 1: **scalable call-path profiling**
  - Option 2: **scalable event trace analysis**
    - **Collect** event traces
    - **Process trace in parallel**
      - Wait-state analysis
      - Delay and root-cause analysis
      - Critical path analysis
    - **Categorize and rank** results



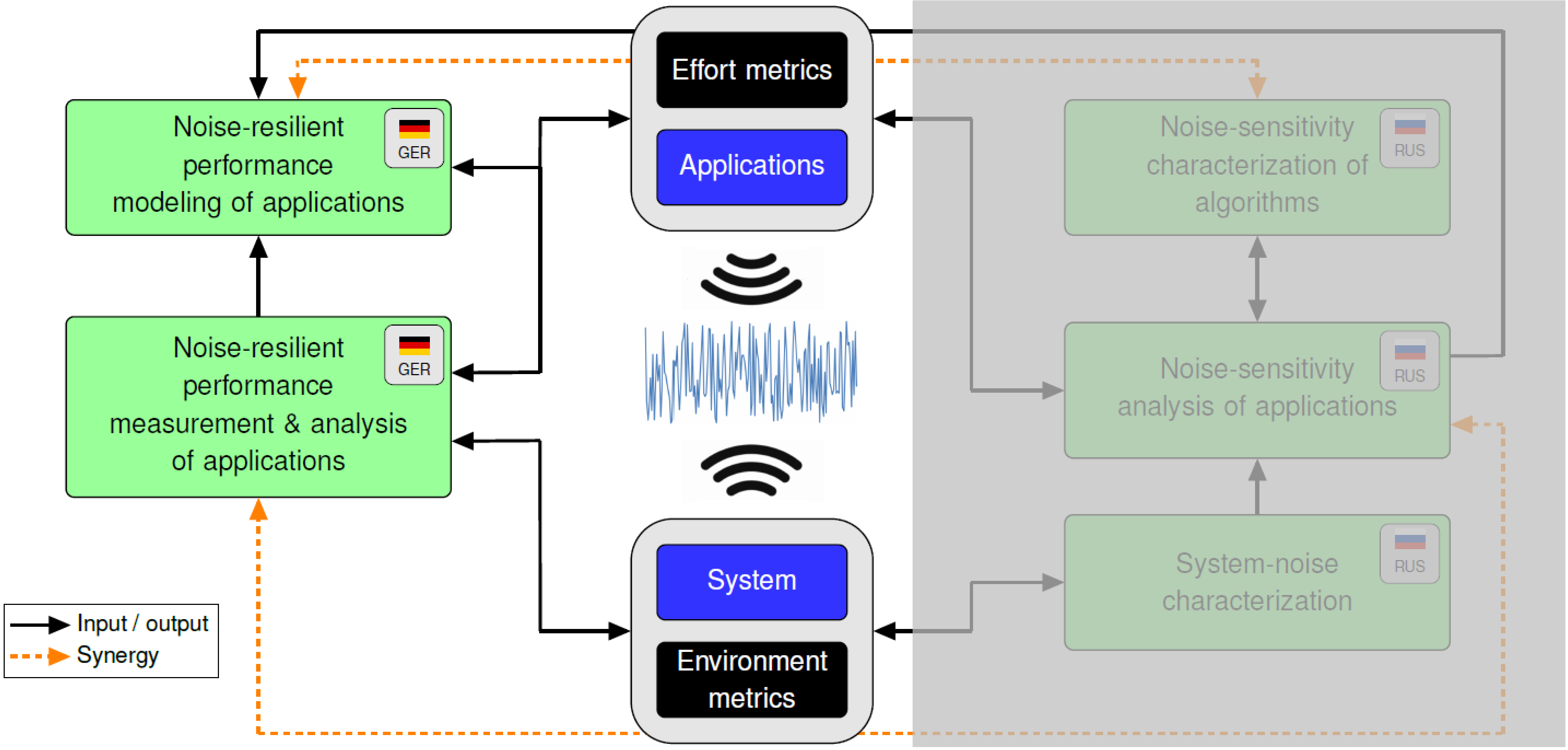
PROJECT PARTNERS

- Technical University of Darmstadt
  Laboratory for Parallel Programming
  Felix Wolf
- Forschungszentrum Jülich
  Jülich Supercomputing Centre
  Bernd Mohr
- ETH Zurich (associated)
  Scalable Parallel Computing Lab
  Thorsten Hoefler
- Moscow State University*
  Research Computing Center
  Dmitry Nikitenko

* In accordance with the DFG policy on joint projects with Russia, the collaboration with our Russian partners has been suspended.

PROJECT OVERVIEW

EXTRANOISE


Effort metrics
Applications
Noise-resilient performance modeling of applications
GER
Noise-resilient performance measurement & analysis of applications
GER
System
Environment metrics
Noise-sensitivity characterization of algorithms
RUS
Noise-sensitivity analysis of applications
RUS
System-noise characterization
RUS
Input / output
Synergy

Noise-resilient Performance Measurement and Analysis of HPC Applications

PLAN: NOISE-RESILIENT MEASUREMENT AND ANALYSIS

EXTRANOISE

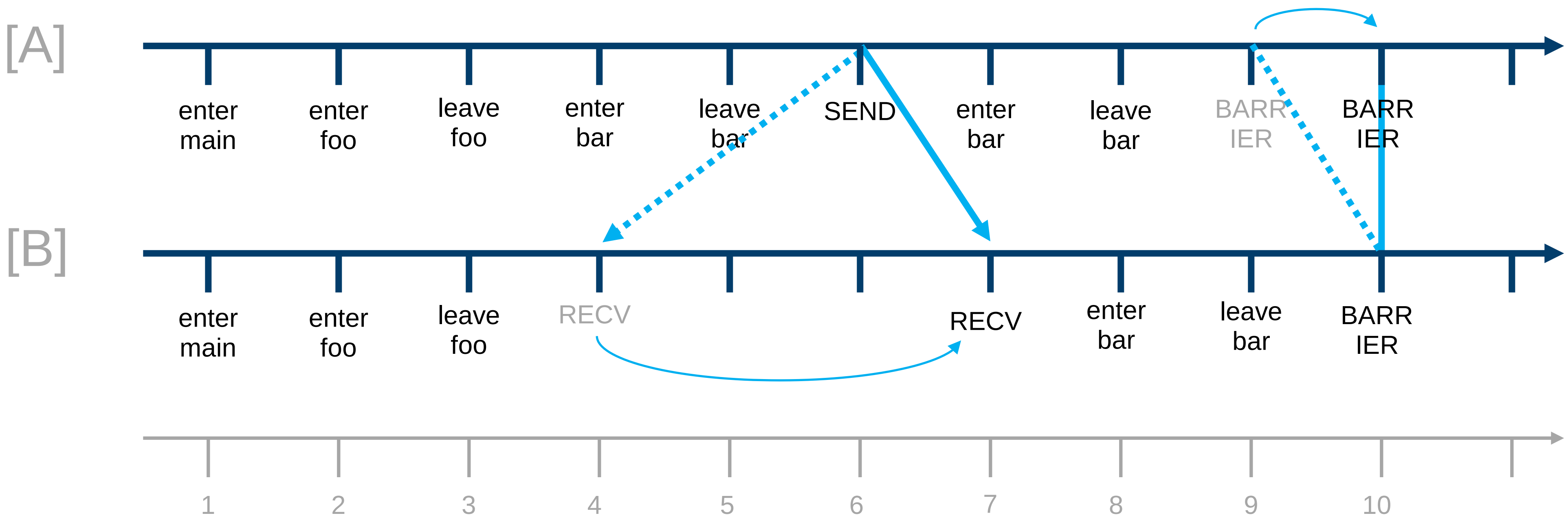
- **Design and prototype new logical timer for Score-P with 3 modes**
  1. Logical time only
     - Increment (+1) at Score-P events (function entry/exit, OpenMP+MPI constructs) ✓
     - Enforce Lamport relation at communication + synchronization events (OpenMP, MPI) ✓
  2. [A] Logical time + effort represented by loop iteration count of parallel (OpenMP) loops
     - Automatic instrumentation based on Opari ✓

     [B] Logical time + effort represented by basic block count

     [C] Logical time + effort represented by statements count

  - Automatic instrumentation (C/C++) with Clang plugin **[prototype]**
  - Logical time + effort represented by (noise-insensitive) HW counter (e.g. #instructions, #flops) **[prototype]**
     - Scaling of HPC counter values to logical clock ticks

LOGICAL/LAMPORT TIMER

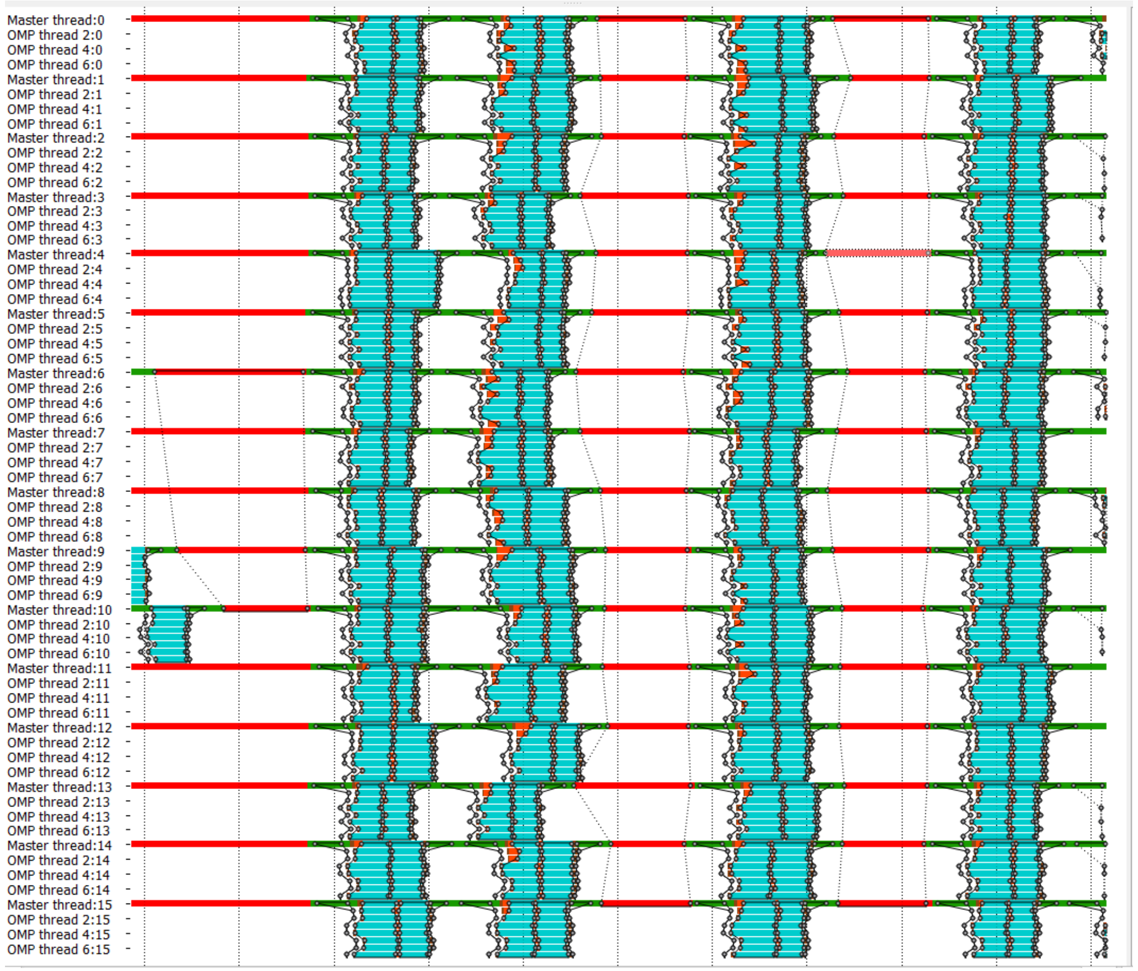


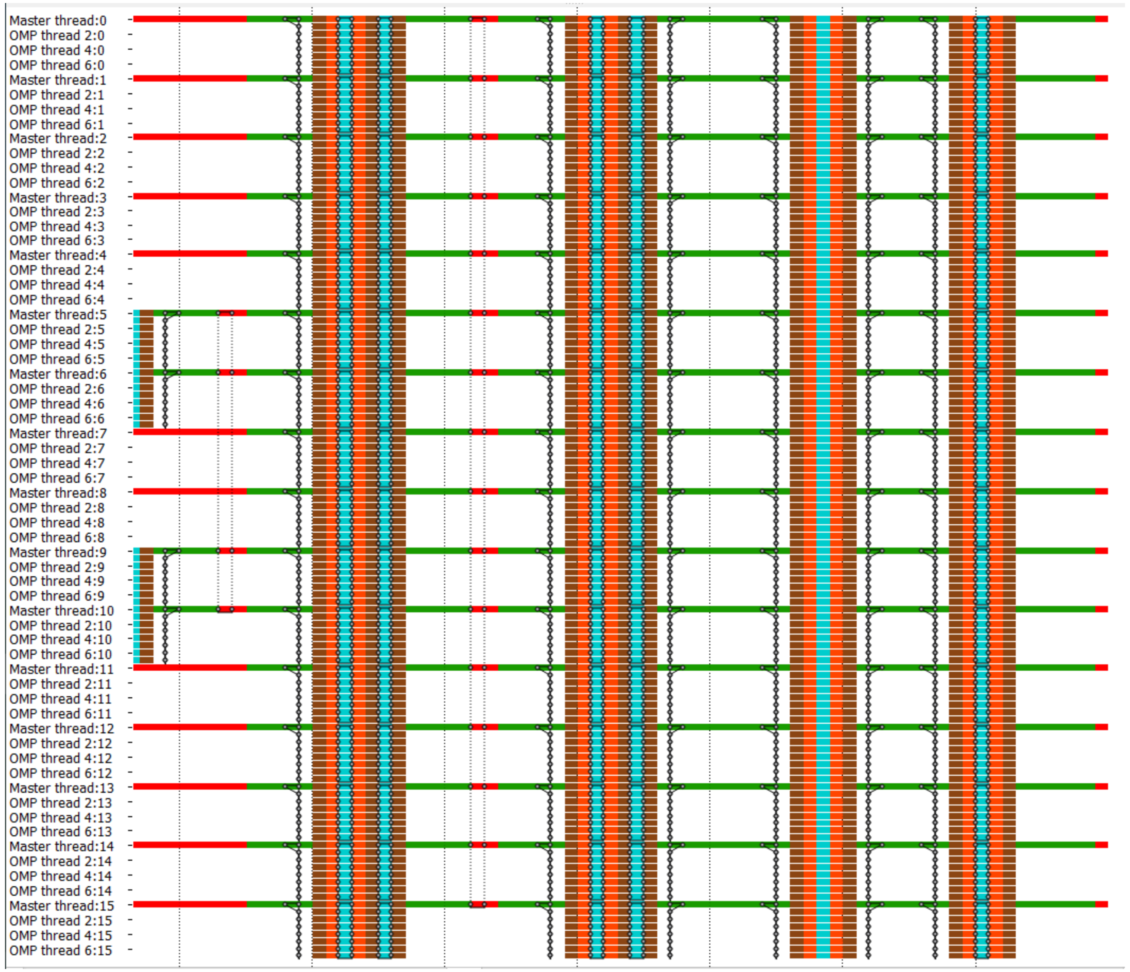
EXAMPLE CODE: TEALEAF

- **TeaLeaf Reference V1.0**
  - HPC mini-app developed by the UK Mini-App Consortium
  - Solves the linear 2D heat conduction equation on a spatially decomposed regular grid using a 5 point stencil with implicit solvers
  - https://github.com/UK-MAC/TeaLeaf_ref/archive/v1.0.tar.gz

- Measurements performed on Jureca cluster @ JSC
  - **Run configuration**
    - 8 MPI ranks with 12 OpenMP threads each
    - Distributed across 4 compute nodes (2 ranks per node)
    - Test problem “5”: 4000 × 4000 cells, CG solver

EARLY RESULTS: LOGICAL TIME MEASUREMENT AND ANALYSIS OF TEALEAF MINI-APP





VERY EARLY RESULTS: LOGICAL TIME MEASUREMENT AND ANALYSIS OF TEALEAF MINI-APP

Execution	Real	Logical	Logical+ Loop	Location
Computation	69.78%	26.02%	76.71%	
MPI	2.12%	2.50%	1.53%	
OpenMP	11.41%	27.12%	9.00%	
Idle Threads	16.70%	44.35%	21.76%	✓

- Undercounting of computation time with Logical time
- Much better with Logical time + parallel loop iter counts
- Hopefully better with effort counting
  - Basic blocks
  - Statements
  - Instructions

VERY EARLY RESULTS: LOGICAL TIME MEASUREMENT AND ANALYSIS OF TEALEAF MINI-APP

Patterns	Real	Logical	Logical+ Loop	Location
Wait at MPI Barrier	0.01%	0.00%	0.00%	
Late Sender	11.00%	0.38%	0.49%	✓
Wait at NxN	0.99%	1.15%	0.73%	✓
Fork	3.78%	10.30%	3.23%	✓
Wait at OpenMP Barrier	2.58%	0%	0.50%	✓
ALL OTHER PATTERNS	0%	0%	0%	

- (Almost) found same bottlenecks
- Bottleneck found are at same location in code
- Underestimation of some patterns
  - Should be better with effort counting, needs more investigation

Noise Generation: NOIGENA

PLAN: NOISE-RESILIENT MEASUREMENT AND ANALYSIS

EXTRANOISE

- **Find / create reliable (→ reproducible) noisy system execution environment**
  - Use nodes with high core count (JURECA, JUSUF: 2 sockets each 64 cores) ✓
  - Run application + noise generator side-by-side with synchronized start ✓
    - Split by socket
    - Split by even/odd cores
  - Noise generator **NOIGENA** **[prototype]**
    - Based on MPI (FZJ_linktest), memory (stream), I/O (ior) benchmarks
    - Configurable but reproducible pattern+duration of different noise phases

EXAMPLE NOIGENA CONFIGURATION FILE (YAML)

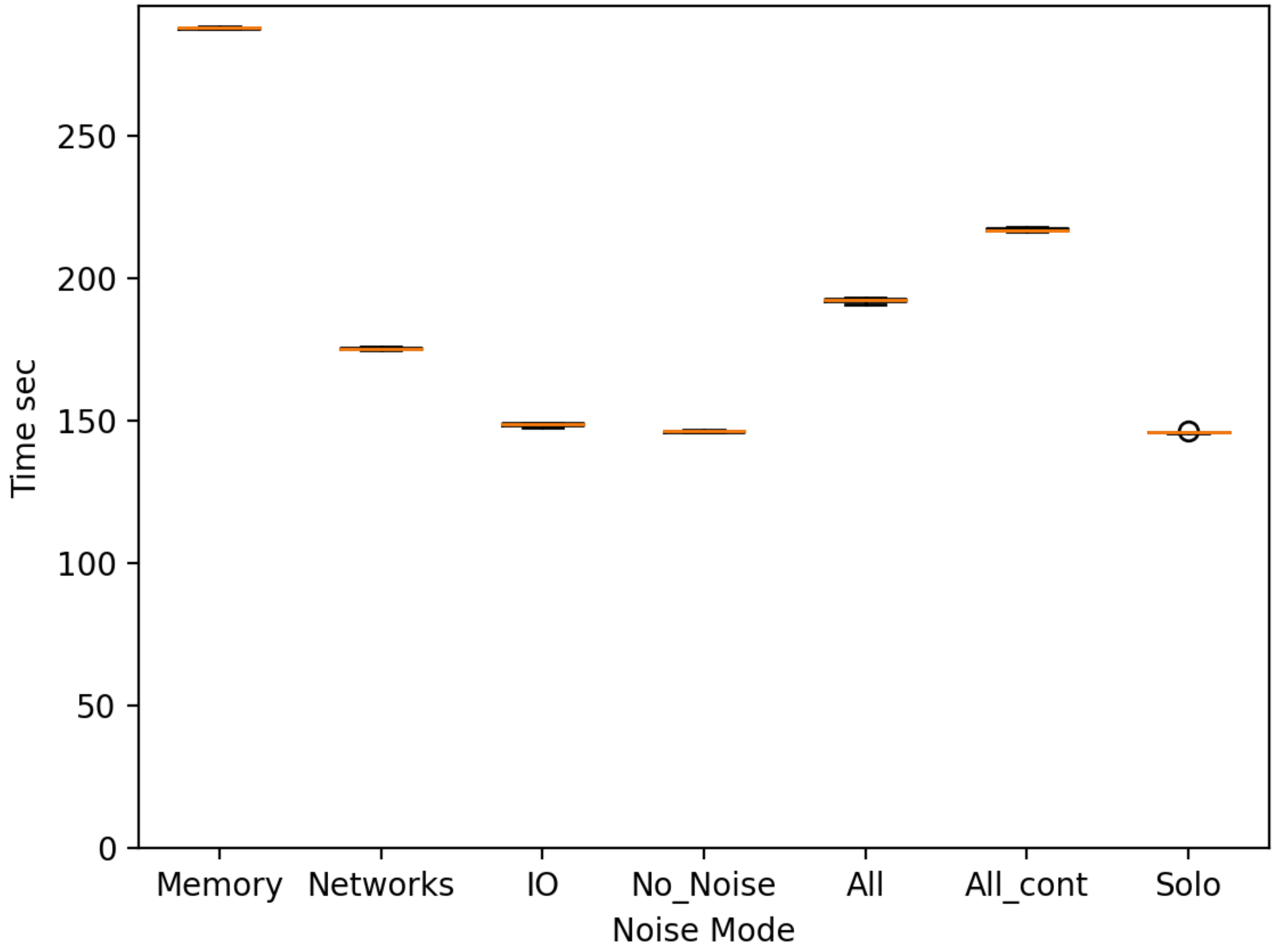
```
Benchmarks_cfg:
  Stream:
    - ENABLE: TRUE
    - array_size: 4000000000
    - OMP_use: TRUE
    - Verbose: 0
  LinkTest:
    - ENABLE: TRUE
    - num_msgs: 3
    - num_warmup_msgs: 3
    - len_msg: 13700
    - serial: 0
    - Verbose: 0
  IOR:
    - ENABLE: TRUE
    - api: MPIIO
    - block_size: 16m
    - transfer_size: 1m
    - segment_count: 16
    - num_tasks: 64
    - file_per_process: TRUE
    - reorder_tasks: TRUE
    - Verbose: 0
```

```
Runs:
  Pattern_1:
    Sequence:
      - MEMORY_NOISE: 100  # secs

  Pattern_2:
    Sequence:
      - NETWORK_NOISE: 40  # secs
      - NO_NOISE: 20       # secs
      - MEMORY_NOISE: 40   # secs
      - NO_NOISE: 20       # secs
      - IO_NOISE: 40       # secs
      - NO_NOISE: 20       # secs

  Pattern_3:
    Sequence:
      - RANDOM_NOISE:
        TIME: 100              # sec
        MEMORY_NOISE: 20       # %
        NETWORK_NOISE: 40      # %
        IO_NOISE: 30           # %
        NO_NOISE: 10           # %
```

EXAMPLE NOIGENA EFFECT ON TEALEAF MINIAPP



Solo: TeaLeaf only

All:
Memory+Network+IO
+No_Noise

All_cont:
Memory+Network+I/O

NEXT STEPS

- **Finalize prototypes**
  - Automatic instrumentation (C/C++) with Clang plugin
  - Logical time + effort represented by (noise-insensitive) HW counter
- **Measure and analyze (much) more codes**
- **Fine-tune methods further if necessary**

THANK YOU!

Want to connect to anyone interested in noise-related issues!

https://www.vi-hps.org/projects/extranoise/

Contact:

- **b.mohr@fz-juelich.de**
- **felix.wolf@tu-darmstadt.de**

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 449683531