A COMPILER-ASSISTED WORKFLOW FOR EFFICIENCY-GUIDED SELECTIVE TRACING

POP Webinar 11.06.2026

Sebastian Kreutzer[1], Valentin Seitz[2], Joan Vinyals[2], Tim Heldmann[1], Christian Iwainsky[1], Marta Garcia[2], Jesus Labarta[2], Christian Bischof[1]

[1]Scientific Computing Group, TU Darmstadt
[2]Barcelona Supercomputing Center

sebastian.kreutzer@tu-darmstadt.de

SCIENTIFIC COMPUTING

TECHNISCHE UNIVERSITÄT DARMSTADT

MOTIVATION

- Size of HPC systems continues to increase
- Full-scale tracing produces prohibitively large data volumes
  - Cumbersome to post-process and analyze
- Smaller runs may not be representative
- But: only some of this data is relevant to the analysis

- Solutions:
  - Post-processing (cutting and filtering)
  - Trace compression
  - Online/in-situ analysis
  - Selective tracing

SCIENTIFIC COMPUTING
TECHNISCHE UNIVERSITÄT DARMSTADT

SELECTIVE TRACING

- Broad definition: restrict data collection in selected parts of the execution
- Three axes:
  - **Structural**: where to record?
    → Instrumented code regions of interest
  - **Temporal**: when to record?
    → Selected region invocations
  - **Detail**: what to record?
    → Kinds and detail level of performance data

SCIENTIFIC COMPUTING

TECHNISCHE UNIVERSITÄT DARMSTADT

TOOL COMPARISON

Table 1: Comparison of selective tracing axes across HPC performance tools.

Tool	Structural Axis	Temporal Axis	Detail Axis
Score-P	Compiler instrumentation (via filter files), source annotations	Configurable invocation ranges	On/off
TAU	Compiler/dynamic instrumentation (via filter files), source annotations	Source code API	On/off
Extrae	Source annotations, dynamic instrumentation	Source code API	On/off, burst mode

SCIENTIFIC COMPUTING

TECHNISCHE UNIVERSITÄT DARMSTADT

KEY CHALLENGES

- **Instrumentation**: granularity and overhead management
- **Temporal control**: configurability and ease-of-use
- **Decision making**: tooling infrastructure to support the analyst
- **Visibility gap**: “on/off” mode hides spurious issues or degradation over time

SC SCIENTIFIC COMPUTING

TECHNISCHE UNIVERSITÄT DARMSTADT

OUR APPROACH

End-to-end toolchain using compiler support, building on top of existing tool infrastructure

Instrumentation

→ Selective instrumentation based on call graph analysis coupled with dynamic filtering

Temporal control

→ Runtime system allowing full temporal control, based on multi-fidelity selective tracing model

Decision making

→ Two-phase workflow: semi-automatic selection based on parallel efficiency metrics

Visibility gap:

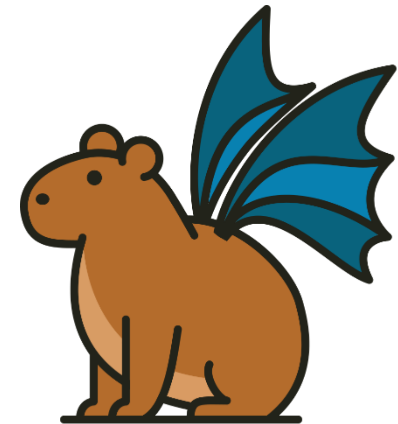
→ Multi-fidelity: expose tool-specific measurement detail levels

SC SCIENTIFIC COMPUTING

TECHNISCHE UNIVERSITÄT DARMSTADT

WORKFLOW

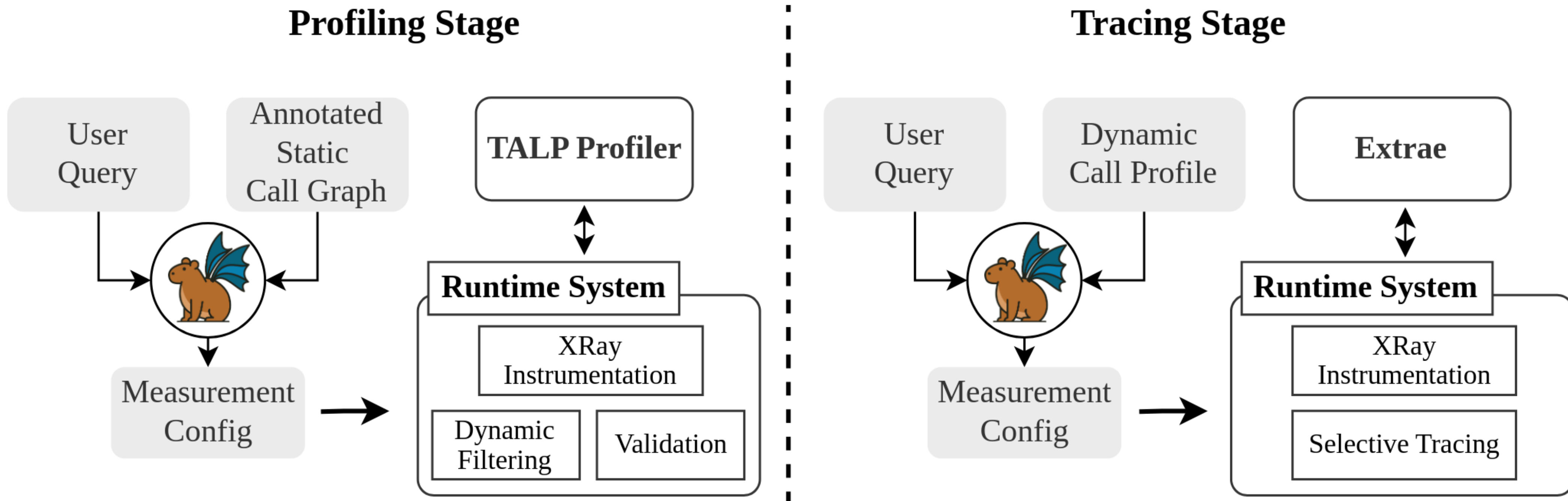
- Two phases mirroring typical analysis workflow:
  - **Profiling:** generate efficiency metrics and fine-tune instrumentation
  - **Tracing:** selective trace based on profiling results
- Both phases configured by user query
  - Running on annotated static/dynamic call graph
  - Using the *Compiler-assisted Performance Instrumentation (CaPI)* tool
- Single-build workflow without source modifications
  - Using LLVM XRay for adaptable instrumentation



https://github.com/tudasc/capi

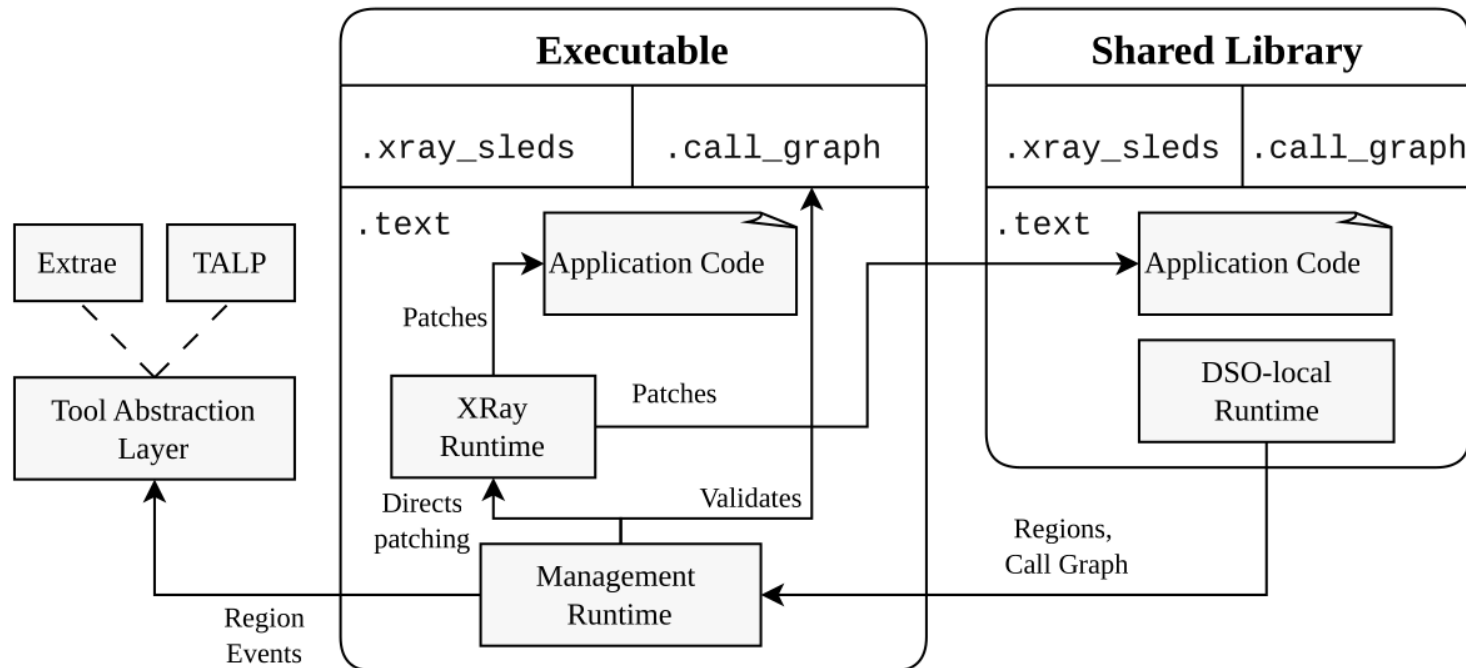
SC SCIENTIFIC COMPUTING | TECHNISCHE UNIVERSITÄT DARMSTADT

WORKFLOW STAGES



SCIENTIFIC COMPUTING
TECHNISCHE UNIVERSITÄT DARMSTADT

BINARY PREPARATION



Usability is critical!

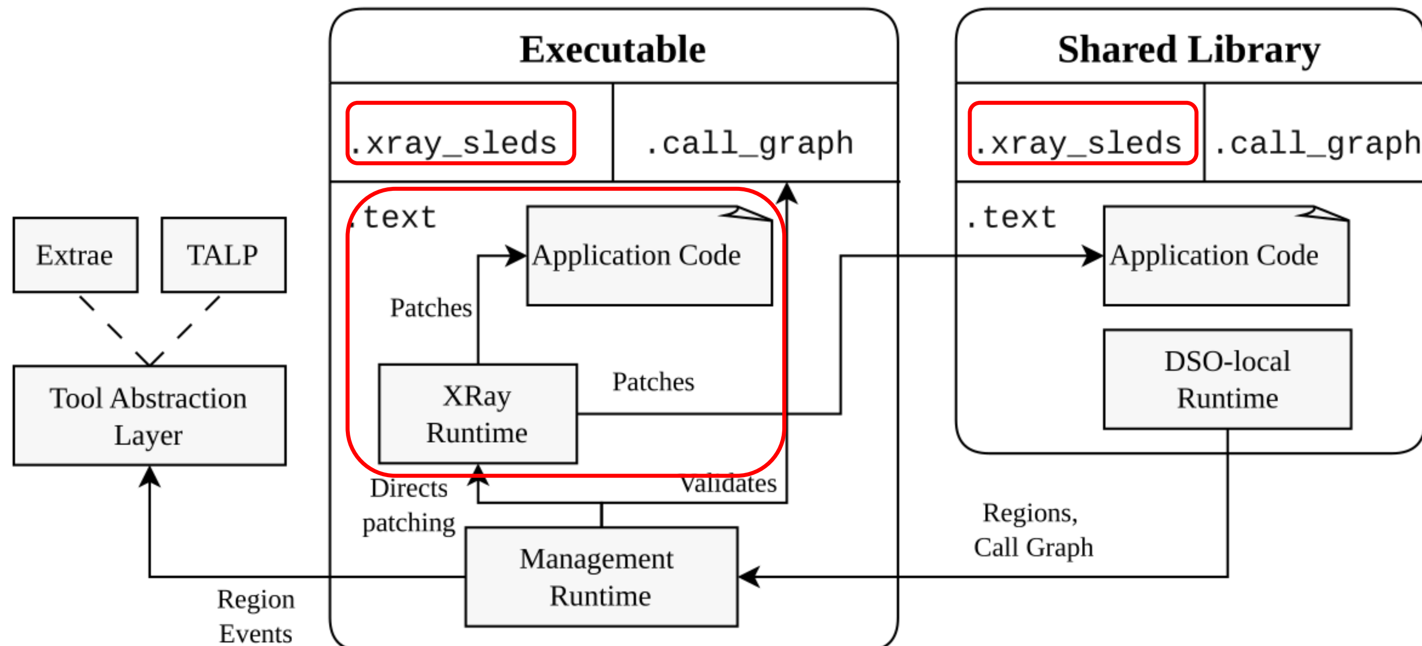
→ complexity hidden behind compiler wrapper

→ required data embedded into binary

SCIENTIFIC COMPUTING

TECHNISCHE UNIVERSITÄT DARMSTADT

BINARY PREPARATION

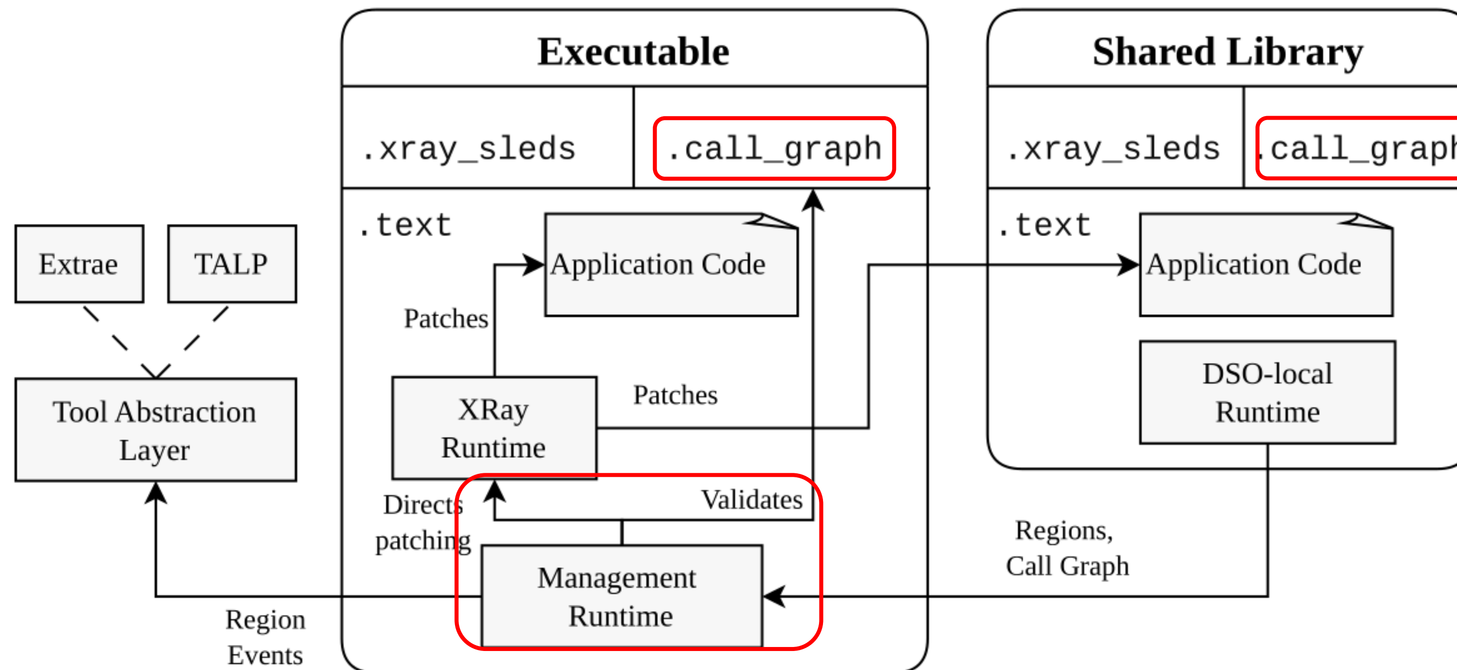


LLVM XRay for runtime-configurable instrumentation:

- Inserts patchable `NOP` sleds
- Can be toggled dynamically
- Near-zero overhead when inactive
- Extended with shared library support in prior work [1]

[1] S. Kreutzer, C. Iwainsky, M. Garcia-Gasulla, V. Lopez, and C. Bischof, “**Runtime-Adaptable Selective Performance Instrumentation**,” in *2023 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, May 2023, pp. 423–432. doi: 10.1109/IPDPSW59300.2023.00073.

BINARY PREPARATION

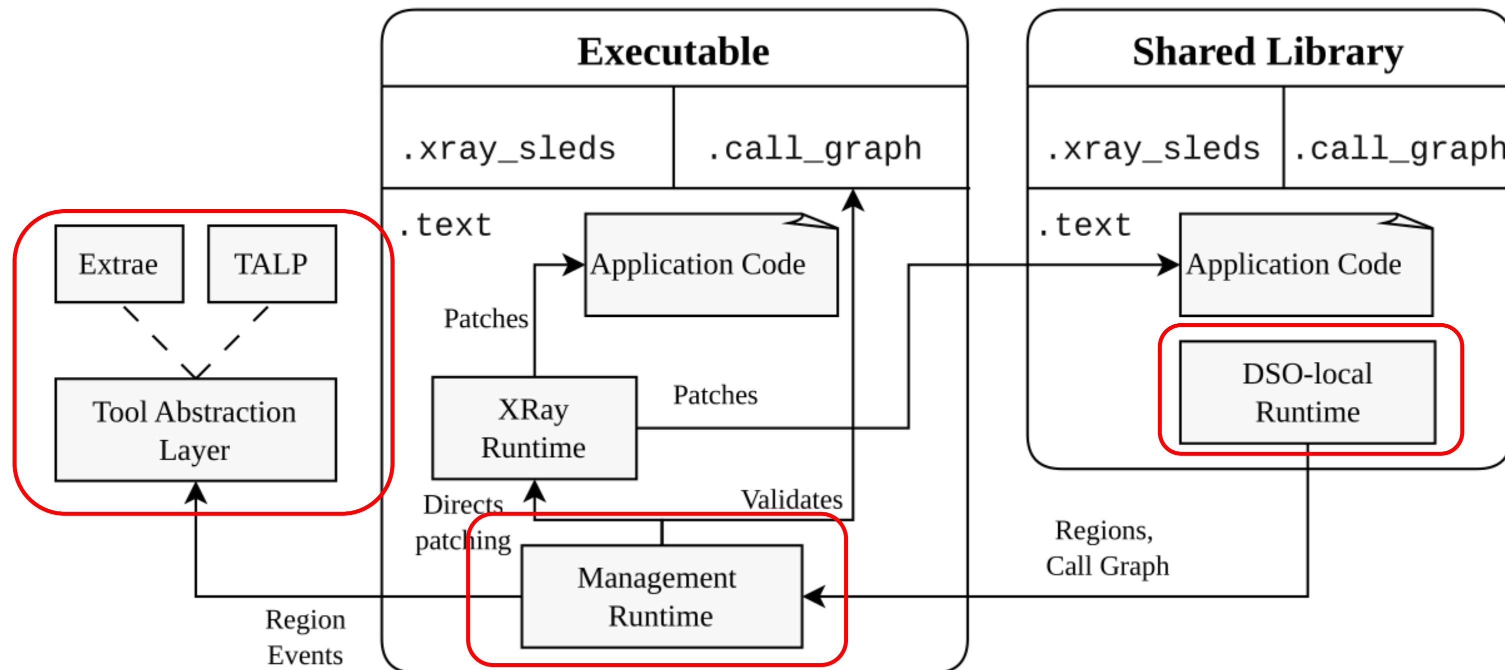


Static call graph generated with MetaCG [2]

- Extracted at link-time
- Annotated with static code metrics
- Embedded into custom ELF section
- Used for selection queries and runtime validation

[2] Jan-Patrick Lehr, Alexander Hück, Yannic Fischler, and Christian Bischof. 2020. **MetaCG: annotated call-graphs to facilitate whole-program analysis**. In Proceedings of the 11th ACM SIGPLAN International Workshop on Tools for Automatic Program Analysis (TAPAS 2020). Association for Computing Machinery, New York, NY, USA, 3–9. https://doi.org/10.1145/3427764.3428320

BINARY PREPARATION



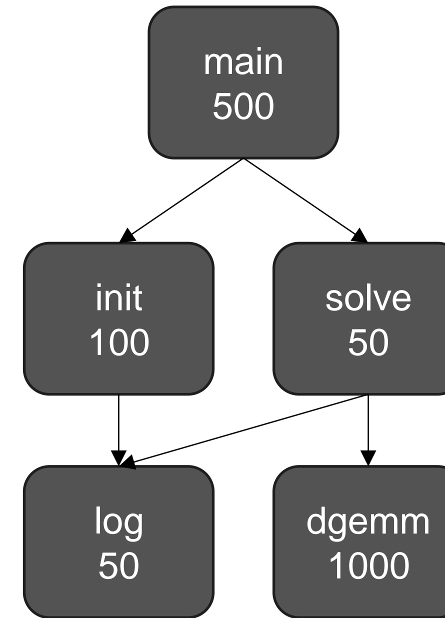
Runtime libraries:

- DSO-local runtime registers data with executable
- Main runtime:
  - Manages XRay
  - Performs call graph validation
  - Forwards region events to tool abstraction layer
- Tool abstraction layer:
  - Exchangeable tool back-end
  - Implements selective tracing semantics

STATIC SELECTION

- We rely on CaPI for static instrumentation selection
  - Operates on static whole-program call graph
  - Custom query DSL

- Selection queries based on:
  - Prior knowledge of relevant code sections
  - Function-local metrics: instruction count, flops, loops, …
  - Call path: callers/callees up to specific depth
  - Derived metrics: **inclusive instruction count**

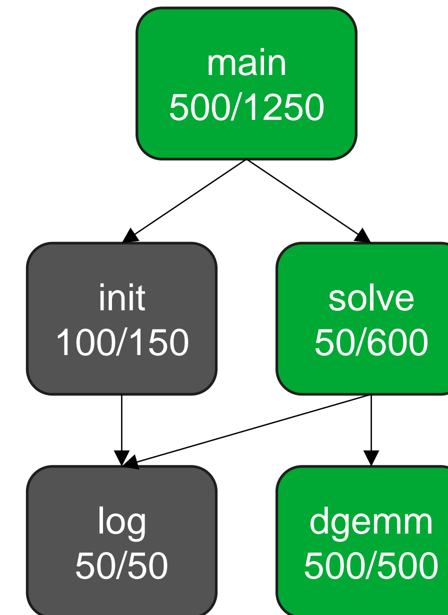


Static call graph annotated with local instruction count

STATIC SELECTION

- We rely on CaPI for static instrumentation selection
  - Operates on static whole-program call graph
  - Custom query DSL

- Selection queries based on:
  - Prior knowledge of relevant code sections
  - Function-local metrics: instruction count, flops, loops, …
  - Call path: callers/callees up to specific depth
  - Derived metrics: **inclusive instruction count**

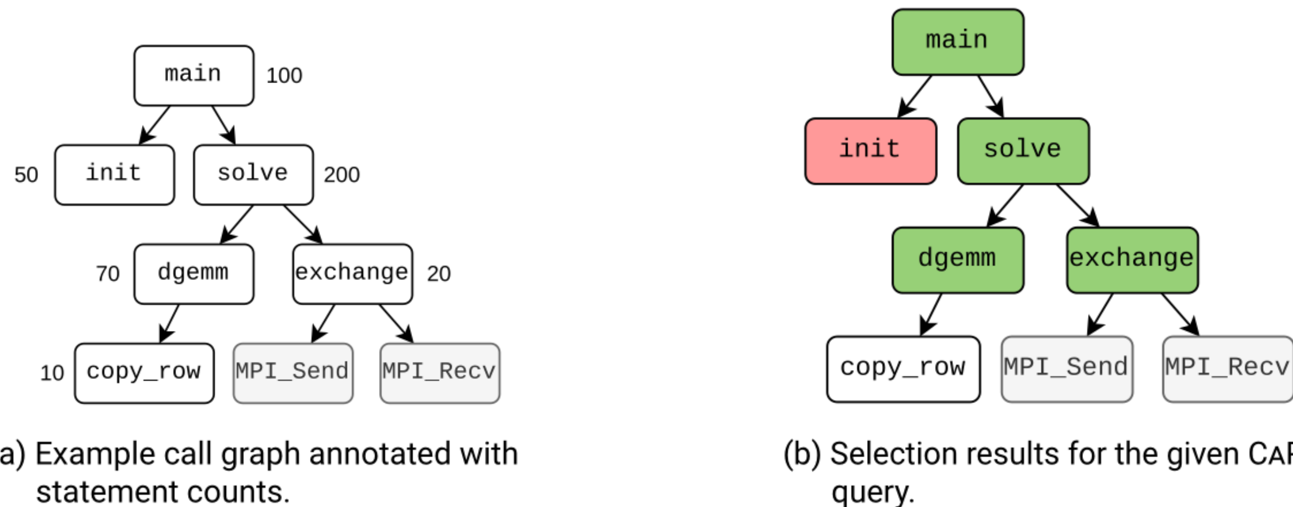


Selection with threshold 500

Running the query:

```
capi -i ‘inclusive_instruction_count(“>=”, 500)’ <program_binary> -o <output_cfg>
```

EXTENDED QUERY EXAMPLE



(a) Example call graph annotated with statement counts.

(b) Selection results for the given CaPI query.

Figure 2.2.: Example selection on a static call graph. Nodes marked green are selected for instrumentation, with `init` being excluded explicitly.

An equivalent CaPI query can be constructed as follows:

```
1 included  = (by_name("MPI_.*") | inclusive_statement_count(">=", 50))
2               |> on_call_path_to
3 excluded  = by_name("init") |> on_call_path_from
4 selection = %included - %excluded
5 !instrument(%selection)
```

Listing 2.1: Example CaPI query.

TECHNISCHE UNIVERSITÄT DARMSTADT

SCIENTIFIC COMPUTING

CALL GRAPH VALIDATION

- Indirect calls lead to missing edges in the static call graph
  - Can affect analyses
- Validation system, extending prior work [3]:
  - Instrument indirect calls:

```
__metacg_indirect_call(const char* fn, void* callee);
```

  - Resolve callee at runtime
  - Maintain patch graph of missing edges
  - At the end of execution:
    - Merge with embedded static call graph
    - Re-run selection query
    - Report difference in selection set

```
struct Data;
struct SolverCfg {
 void (*solverFn)(Data*);
};

Data* data;
void timeStep(SolverCfg* cfg) {
 cfg->solverFn(data);
}
```

[3] S. Kreutzer, S. Martens, P. Arzt, T. Heldmann, and C. Bischof, “**CGPatch: Streamlining Static Call Graph Validation Using Selective Instrumentation**,” in *High Performance Computing*, S. Neuwirth, A. K. Paul, T. Weinzierl, and E. C. Carson, Eds., Cham: Springer Nature Switzerland, 2026, pp. 287–299. doi: 10.1007/978-3-032-07612-0_22.

SCIENTIFIC COMPUTING

TECHNISCHE UNIVERSITÄT DARMSTADT

DYNAMIC FILTERING

- Static selection alone is often not sufficient to control overhead
- We adopt the **throttling** approach from TAU
  - Filter out functions after N invocations if their average execution time is below T_{min}
- Improvement: XRay enables dynamic unpatching (threadsafe!)
  → eliminates residual overhead

TECHNISCHE UNIVERSITÄT DARMSTADT

SELECTIVE TRACING MODEL

- Trace configuration assigns measurement detail level to specific invocation ranges
  - Entering a new function: level can go up, but not down

- Measurement levels for Extrae:
  - **off** (lowest): Fully disable tracing
  - **annotate**: Record only function entry/exit time stamps
  - **burst**: Record programming models in burst mode
  - **full**: Record all events explicitly
  - **forceoff** (highest): Force off mode

SELECTIVE TRACING MODEL

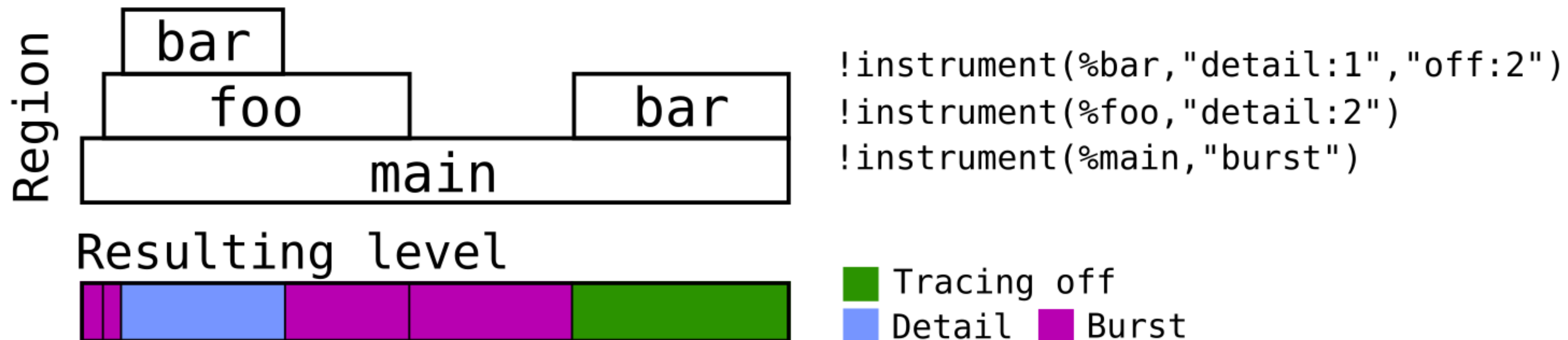


Fig. 3: Example of the selective tracing model for three regions

TRACE CONFIGURATION

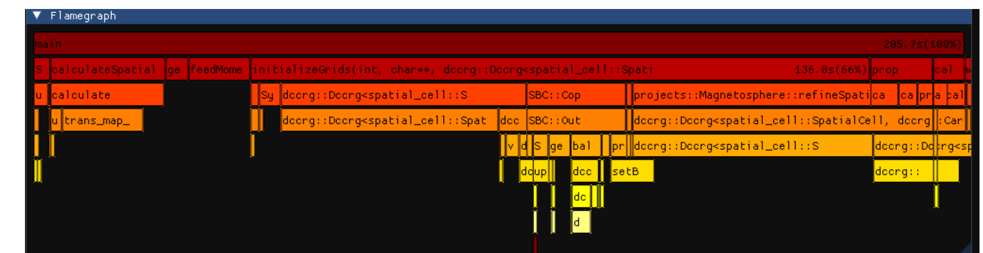
```
Dynamic Callgraph Tree

Current Measurement Config has 0 selected Functions
▼ main::internal::capi: 205.6969s
   ▶ initializeGrids: 136.8229s
   ▶ calculateSpatialTranslation: 25.4769s
   ▶ propagateFields: 13.8036s
   feedMomentsIntoFsGrid: 13.63756032s
   ▶ calculateAcceleration: 7.0113s
   getFieldsFromFsGrid: 5.063227904s
   ▶ SysBoundary::applySysBoundaryVlasovCondit
   ▶ writeGrid: 2.6103s
   computeNewTimeStep: 0.10829124s
   FsGrid::FsGrid: 0.052821136s
   FsGrid::FsGrid: 0.034842176s
   ▶ Readparameters::parse: 0.0298s
   FsGrid::FsGrid: 0.015819594s
   calculateScaledDeltasSimple: 0.010290941s
   ▶ Parameters::addParameters: 0.0009s
   ▶ Parameters::getParameters: 0.0007s
   ▶ projects::Project::addParameters: 0.0006s
   ▶ SysBoundary::addParameters: 0.0003s
   initializeDataReducers: 0.000182873s
   ▶ projects::Magnetosphere::getParameters: 0
   ▶ ObjectWrapper::addPopulationParameters: 0
   getVersion[abi:cxx11]: 4.9053e-05s
   SBC::SphericalTriGrid::initSolver: 4.6334e-0
   Readparameters::get: 4.0121e-05s
   FieldTracing::reduceData: 2.0242e-05s
   Readparameters::get: 5.996e-06s
   Readparameters::get: 4.263e-06s
   FieldTracing::calculateIonosphereFsgridCoupl
   Readparameters::get: 1.467e-06s
   Readparameters::get: 1.41e-06s
```

CAPI Data
paramters.size(): 2
isTrigger: false

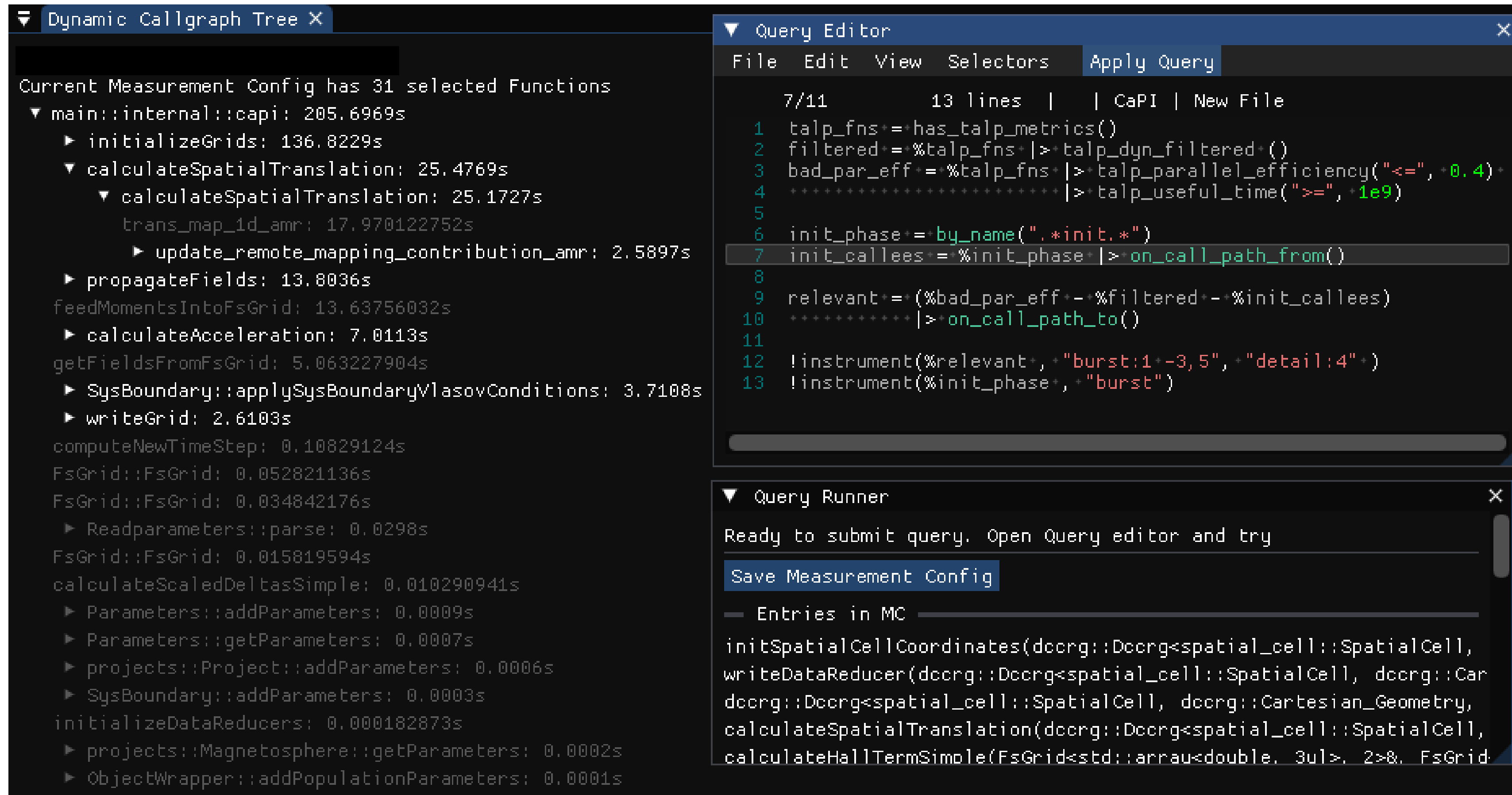
TALP Data
dynamicallyFiltered: false

Metric	Value
Parallel efficiency	0.40
MPI Parallel efficiency	0.85
MPI Communication efficiency	0.94
MPI Load balance	0.91
MPI In-node load balance	0.93
MPI Inter-node load balance	0.98
OpenMP Parallel efficiency	0.54
OpenMP Scheduling efficiency	1.00
OpenMP Load balance	1.00
OpenMP Serialization efficiency	0.54
Useful IPC	0.44
Average Frequency [GHz]	2.97
Useful instruction	5.89e+12
Useful cycles	1.34e+13
Number of Invocations	1232
Number of MPI calls	27022560
Number OpenMP parallels	17136
Number OpenMP tasks	0
Average Region Duration [ms]	9264.33
MPI calls per second per process	2367.56

- GUI provides tree view with TALP metrics
- Flame graph



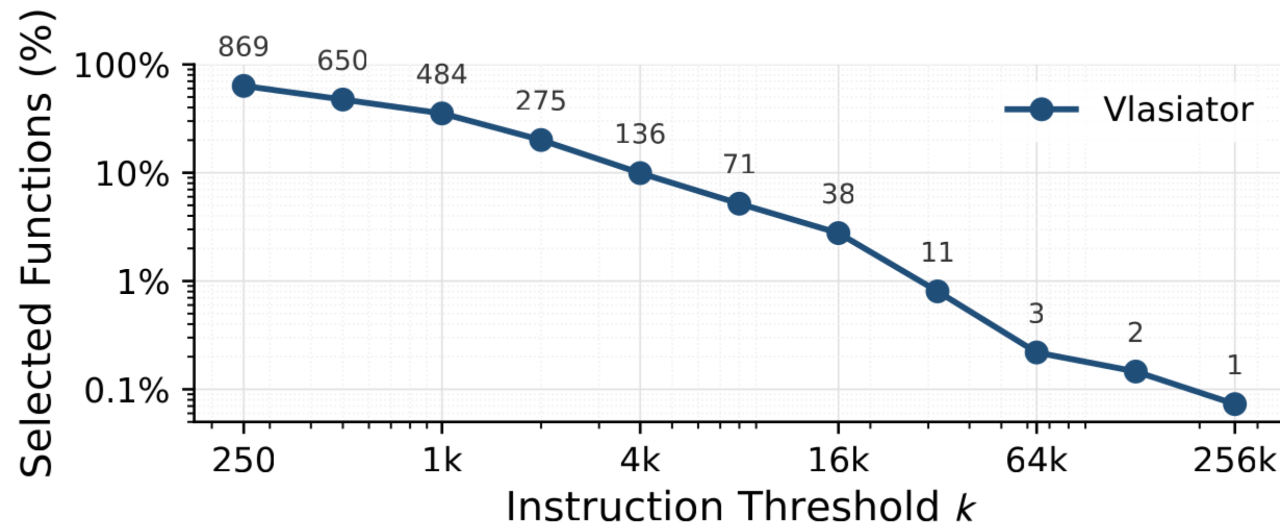
TRACE CONFIGURATION



- Selection queries can be run interactively

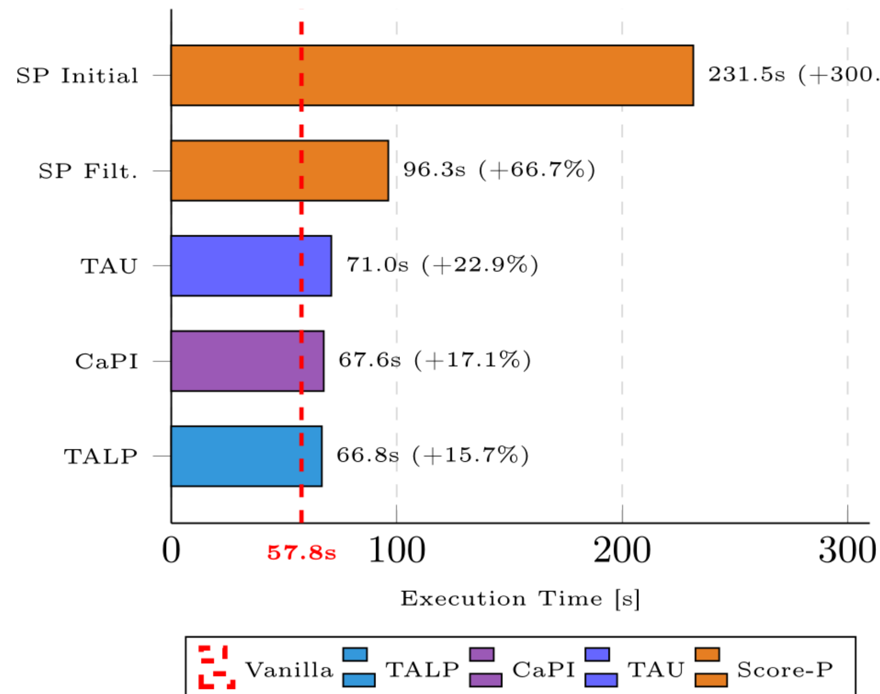
CASE STUDY: VLASIATOR

- Space plasma simulation code with hybrid MPI/OpenMP parallelism
- Built with `capicc` wrapper
  - Call graph has 2371 function nodes
- Initial instrumentation selection based on inclusive instruction count



PROFILING PHASE

- Proceeding with inclusive instruction threshold of 1000
- Comparing run times on 4 nodes with dynamic filtering parameters $T_{min} = 10\mu s$, $N = 100$



- **SP Initial**: Score-P (LLVM plugin) w/o explicit filter
- **SP Filt.**: Dynamic filter generated with `scorep-score` utility
- **TAU**: Static instruction threshold of 1000, throttling active
- **TALP**: Global efficiency metrics without regions

CaPI results: 187 regions recorded, 10 filtered out

SCIENTIFIC COMPUTING

TECHNISCHE UNIVERSITÄT DARMSTADT

BUILDING THE QUERY

- Objectives for this trace:
  - Detect regions with subpar parallel efficiency and instrument them for one invocation
    - Record the rest in burst mode
  - Exclude the initialization phase, record that in burst mode as well
  - Annotate regions for all relevant call paths, if not already included in this set

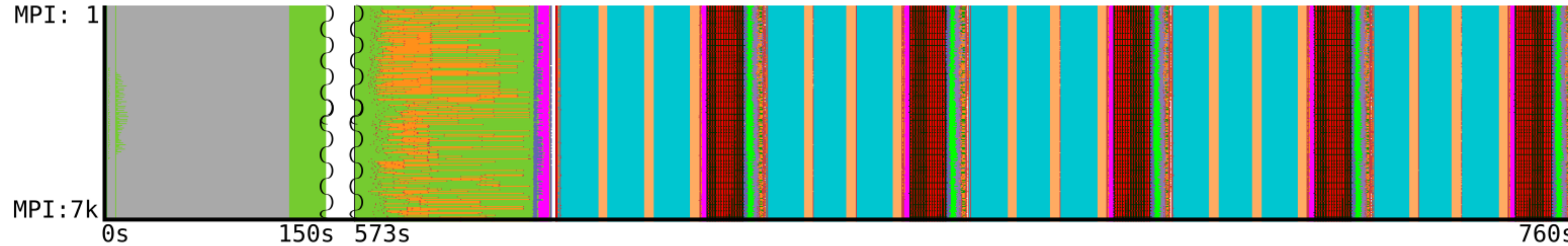
QUERY FOR VLASIATOR

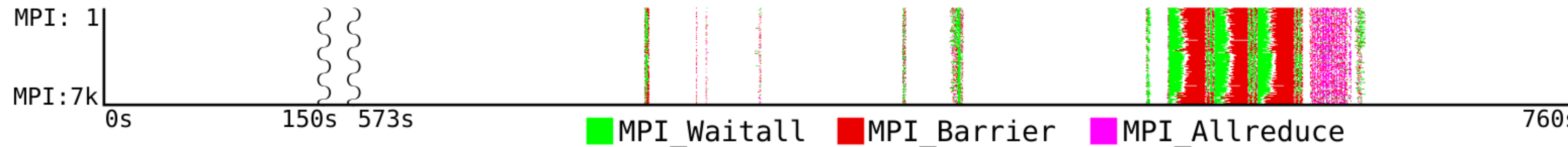
```
1  talp_fns     = has_talp_metrics()
2  bad_par_eff  = %talp_fns |> talp_parallel_efficiency("<=", 0.4)
3                           |> talp_elapsed_time(">", 2e9)
4  filtered     = %talp_fns |> talp_dyn_filtered()
5  init_fns     = by_name(".*init.*")
6  init_phase   = %init_fn  |> on_call_path_from()
7  relevant     = ((%bad_par_eff - %filtered - %init_phase)
8                           |> on_call_path_to())
9
10 !instrument(%relevant, "burst:1-3,5", "detail:4")
11 !instrument(%init_fn, "burst")
```

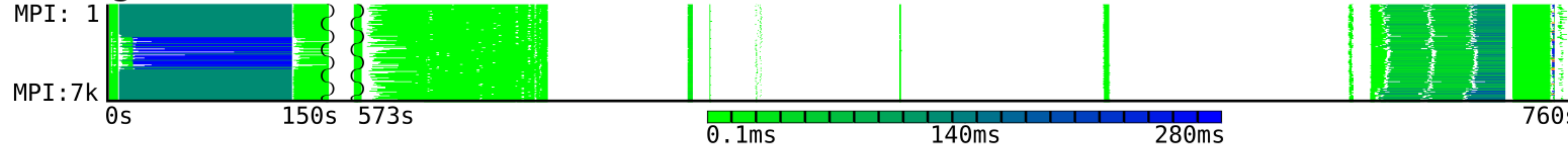
SCIENTIFIC COMPUTING

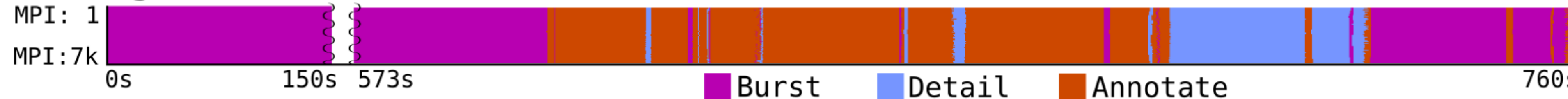
TECHNISCHE UNIVERSITÄT DARMSTADT

VLASIATOR TRACE









10x trace size reduction:
990 GB → 90 GB
(500 nodes)

SC SCIENTIFIC COMPUTING

TECHNISCHE UNIVERSITÄT DARMSTADT

FUTURE DIRECTIONS

- Evaluate with wider range of applications
- Flang support
- Custom region and loop instrumentation with XRay
- Extend selective tracing model
  - E.g. allow specifying hierarchical invocation ranges: „first 2 iterations of A within first iteration of B“
- Evaluate additional tracing backends: Score-P, Nsight systems

Container with development version:
https://github.com/sebastiankreutzer/europar26-artifact/