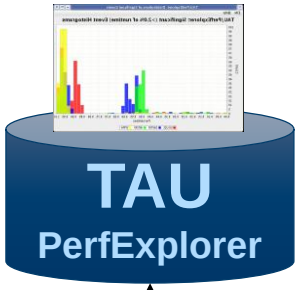
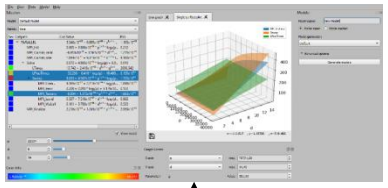
TOOLS DEMO: BT-MZ WITH SCORE-P

MAY 24, 20234 I MICHAEL KNOBLOCH I M.KNOBLOCH@FZ-JUELICH.DE

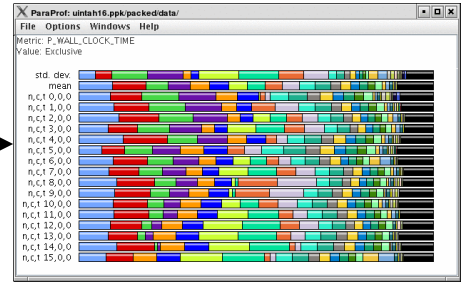
Score-P TOOL ECOSYSTEM

JÜLICH Forschungszentrum | JÜLICH SUPERCOMPUTING CENTRE





Extra-P



TAU ParaProf

Instrumented target application

Score-P

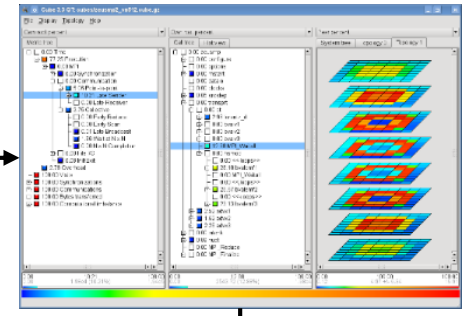
PAPI

CUBE4 report

CUBE4 report

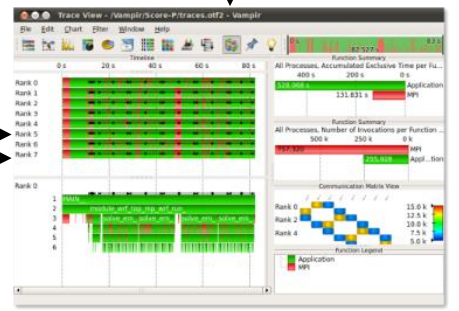
Scalasca parallel trace analysis

OTF2 traces



CUBE

Remote | Guidance



Vampir

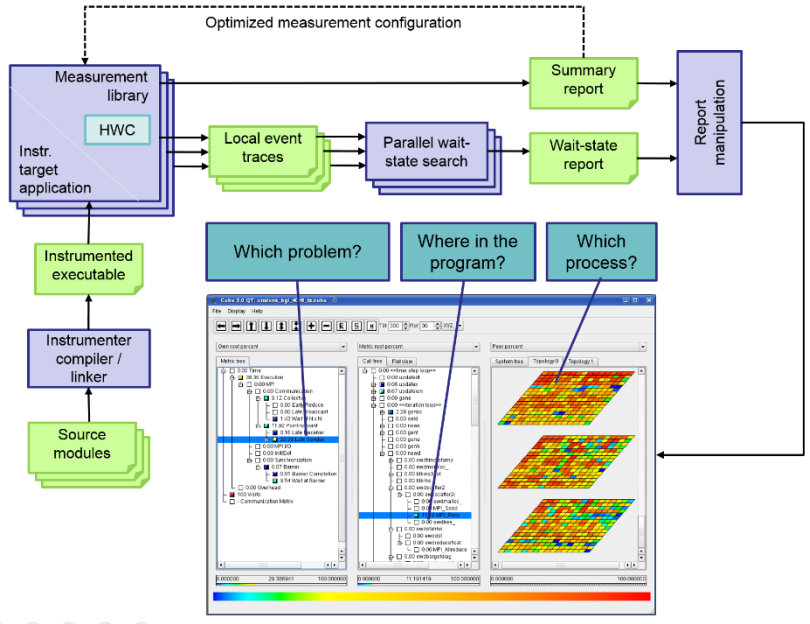
SCALASCA

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



TYPICAL PERFORMANCE ANALYSIS PROCEDURE

- Do I have a performance problem at all?
  - Time / speedup / scalability measurements
- ***What*** is the key bottleneck (computation / communication)?
  - MPI / OpenMP / flat profiling
- ***Where*** is the key bottleneck?
  - Call-path profiling, detailed basic block profiling
- ***Why*** is it there?
  - Hardware counter analysis
  - Trace selected parts (to keep trace size manageable)
- Does the code have scalability problems?
  - Load imbalance analysis, compare profiles at various sizes function-by-function, performance modeling

WHAT IS THE KEY BOTTLENECK?

- Generate flat MPI profile using Score-P/Scalasca
  - Only requires re-linking
  - Low runtime overhead

- Provides detailed information on MPI usage
  - How much time is spent in which operation?
  - How often is each operation called?
  - How much data was transferred?

- Limitations:
  - Computation on non-master threads and outside of MPI_Init/MPI_Finalize scope ignored

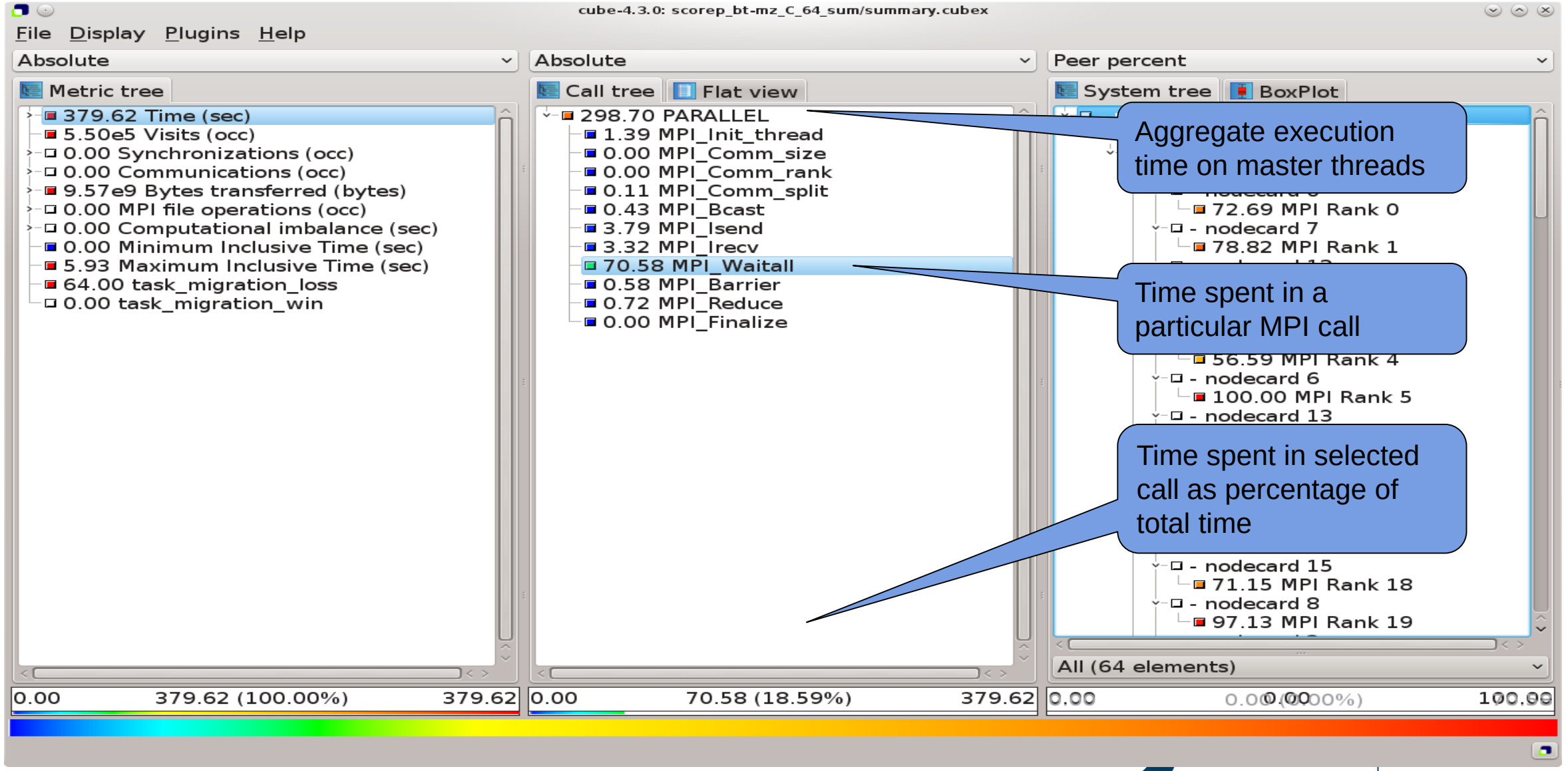
FLAT MPI PROFILE: RECIPE

1. Prefix your ***link command*** with
   "`scorep --nocompiler`"

2. Prefix your MPI ***launch command*** with
   "`scalasca -analyze`"

3. After execution, examine analysis results using
   "`scalasca -examine scorep_<title>`"

FLAT MPI PROFILE: EXAMPLE (CONT.)



WHERE IS THE KEY BOTTLENECK?

- Generate call-path profile using Score-P/Scalasca
  - Requires re-compilation
  - Runtime overhead depends on application characteristics
  - Typically needs some care setting up a good measurement configuration
    - Filtering
    - Selective instrumentation

- Option 1 (recommended for beginners):
  Automatic compiler-based instrumentation

- Option 2 (for in-depth analysis):
  Manual instrumentation of interesting phases, routines, loops

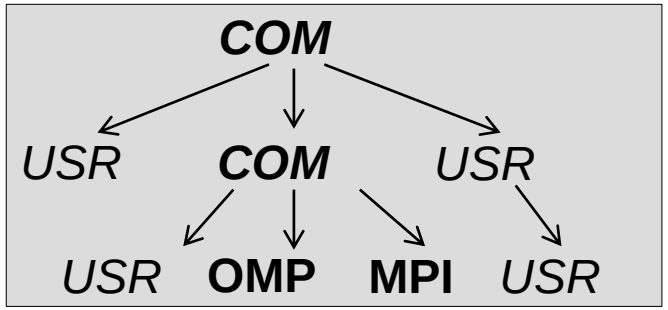
CALL-PATH PROFILE: RECIPE

1. Prefix your ***compile & link commands*** with
   "`scorep`"
2. Prefix your MPI ***launch command*** with
   "`scalasca -analyze`"
3. After execution, compare overall runtime with uninstrumented run to determine overhead
4. If overhead is too high
   1. Score measurement using
      "`scalasca -examine -s scorep_<title>`"
   2. Prepare filter file
   3. Re-run measurement with filter applied using prefix
      "`scalasca –analyze –f <filter_file>`"
5. After execution, examine analysis results using
   "`scalasca -examine scorep_<title>`"

CALL-PATH PROFILE: EXAMPLE (CONT.)

```
% scalasca -examine -s epik_myprog_Ppnxt_sum
scorep-score -r ./epik_myprog_Ppnxt_sum/profile.cubex
INFO: Score report written to ./scorep_myprog_Ppnxt_sum/scorep.score
```

- Estimates trace buffer requirements
- Allows to identify canditate functions for filtering
  - ☞Computational routines with high visit count and low time-per-visit ratio
- Region/call-path classification
  - MPI (pure MPI library functions)
  - OMP (pure OpenMP functions/regions)
  - USR (user-level source local computation
  - COM (“combined” USR + OpeMP/MPI)
  - ANY/ALL (aggregate of all region types)



CALL-PATH PROFILE: EXAMPLE (CONT.)

```
% less scorep_myprog_Ppmxt_sum/scorep.score
Estimated aggregate size of event trace:                   162GB
Estimated requirements for largest trace buffer (max_buf): 2758MB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       2822MB
(hint: When tracing set SCOREP_TOTAL_MEMORY=2822MB to avoid
 intermediate flushes or reduce requirements using USR regions
 filters.)

flt type    max_buf[B]         visits  time[s] time[%] time/      region
                                                       visit[us]
     ALL 2,891,417,902 6,662,521,083 36581.51   100.0      5.49  ALL
     USR 2,858,189,854 6,574,882,113 13618.14    37.2      2.07  USR
     OMP    54,327,600    86,353,920 22719.78    62.1    263.10  OMP
     MPI       676,342       550,010   208.98     0.6    379.96  MPI
     COM       371,930       735,040    34.61     0.1     47.09  COM

     USR   921,918,660 2,110,313,472  3290.11     9.0      1.56  matmul_sub
     USR   921,918,660 2,110,313,472  5914.98    16.2      2.80  binvcrhs
     USR   921,918,660 2,110,313,472  3822.64    10.4      1.81  matvec_sub
     USR    41,071,134    87,475,200   358.56     1.0      4.10  lhsinit
     USR    41,071,134    87,475,200   145.42     0.4      1.66  binvrhs
     USR    29,194,256    68,892,672    86.15     0.2      1.25  exact_solution
     OMP     3,280,320     3,293,184    15.81     0.0      4.80  !$omp parallel
     [...]
```

CALL-PATH PROFILE: FILTERING

- In this example, the 6 most fequently called routines are of type USR
- These routines contribute around 35% of total time
  - However, much of that is most likely measurement overhead
    - Frequently executed
    - Time-per-visit ratio in the order of a few microseconds

☞Avoid measurements to reduce the overhead

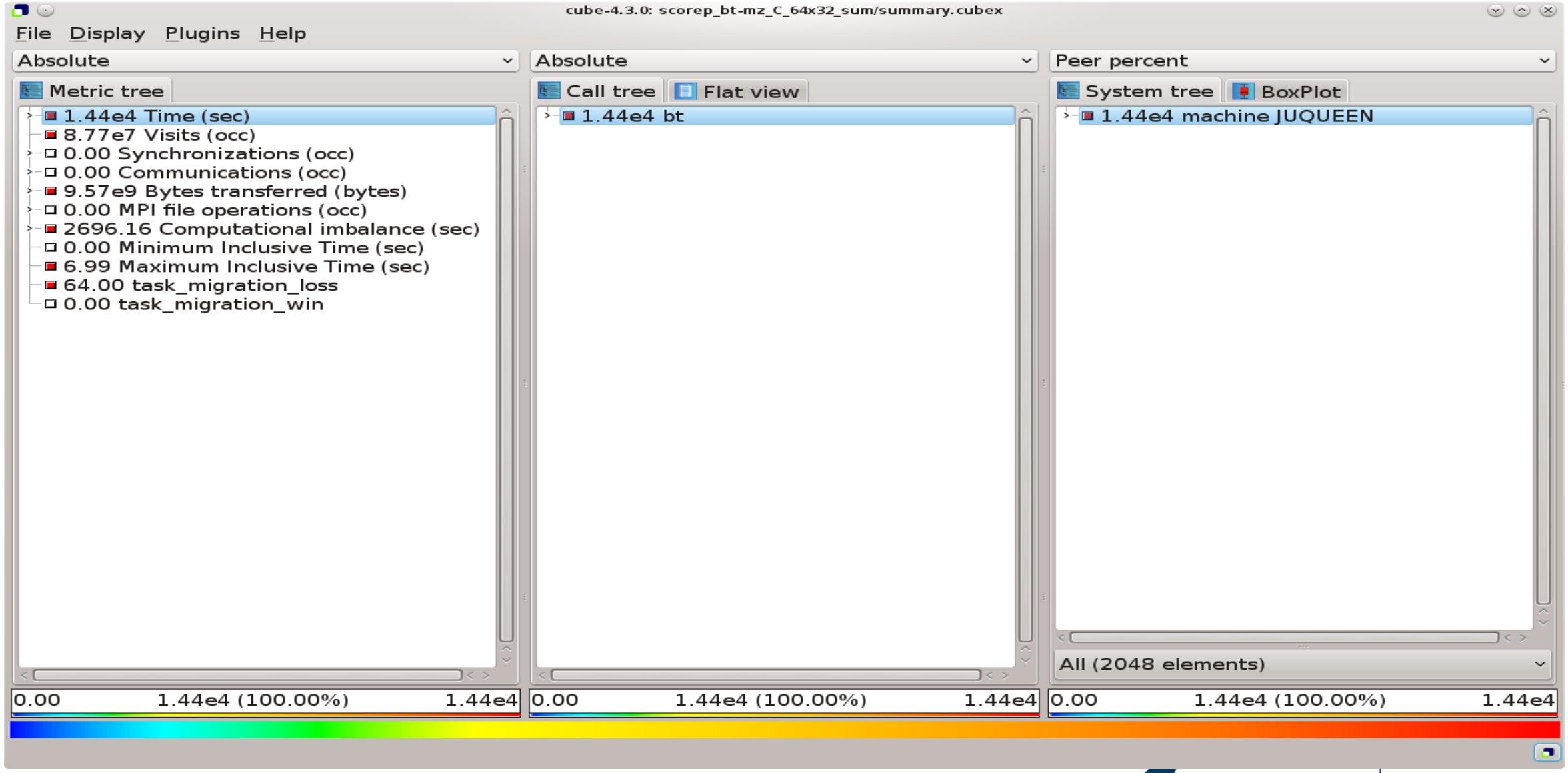
☞List routines to be filtered in simple text file

FILTERING: EXAMPLE

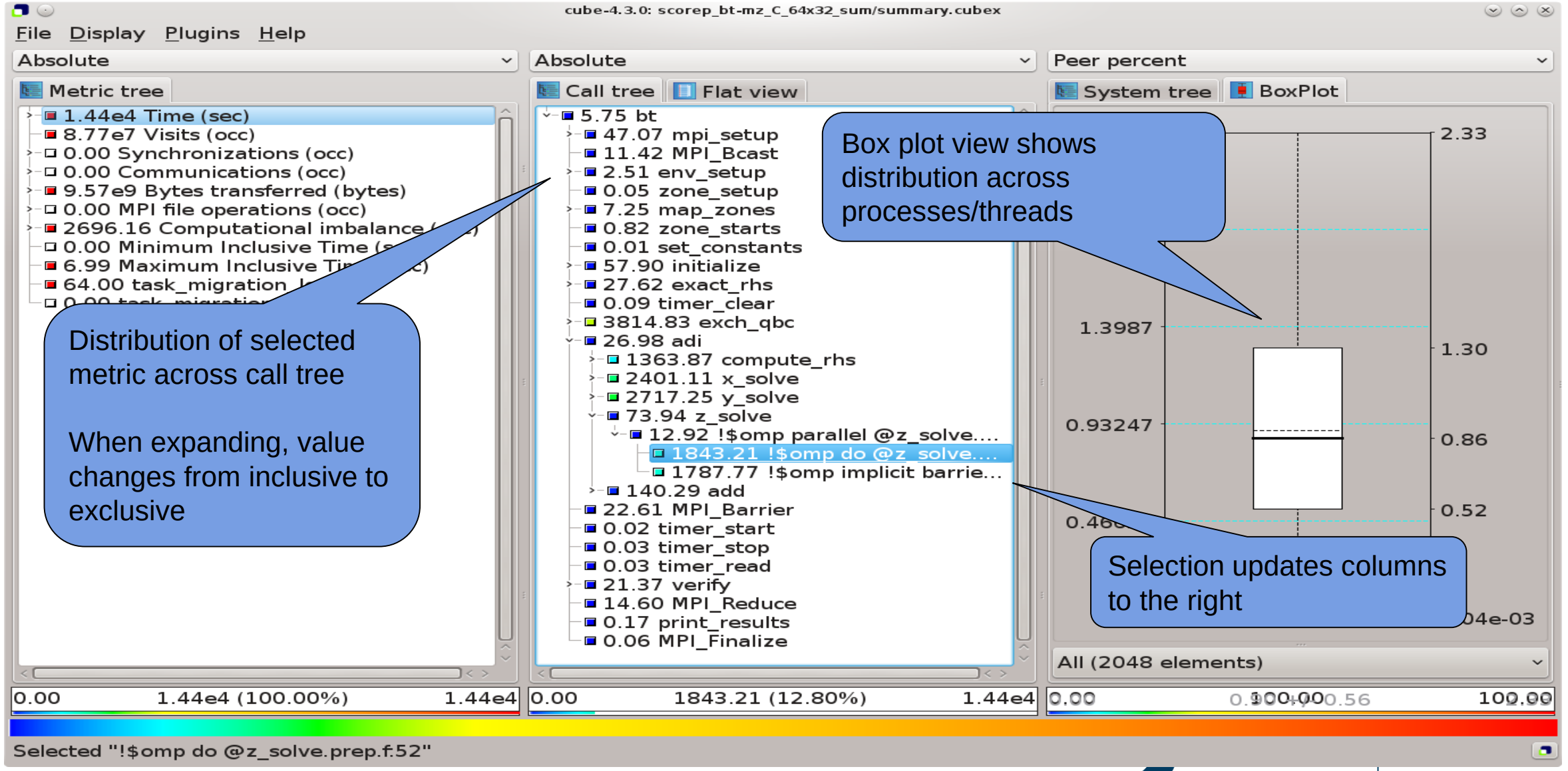
```
% cat filter.txt
SCOREP_REGION_NAMES_BEGIN
    EXCLUDE
        binvcrhs
        matmul_sub
        matvec_sub
        binvrhs
        lhsinit
        exact_solution
SCOREP_REGION_NAMES_END
```

- Score-P filtering files support
  - Wildcards (shell globs)
  - Blacklisting
  - Whitelisting
  - Filtering based on filenames

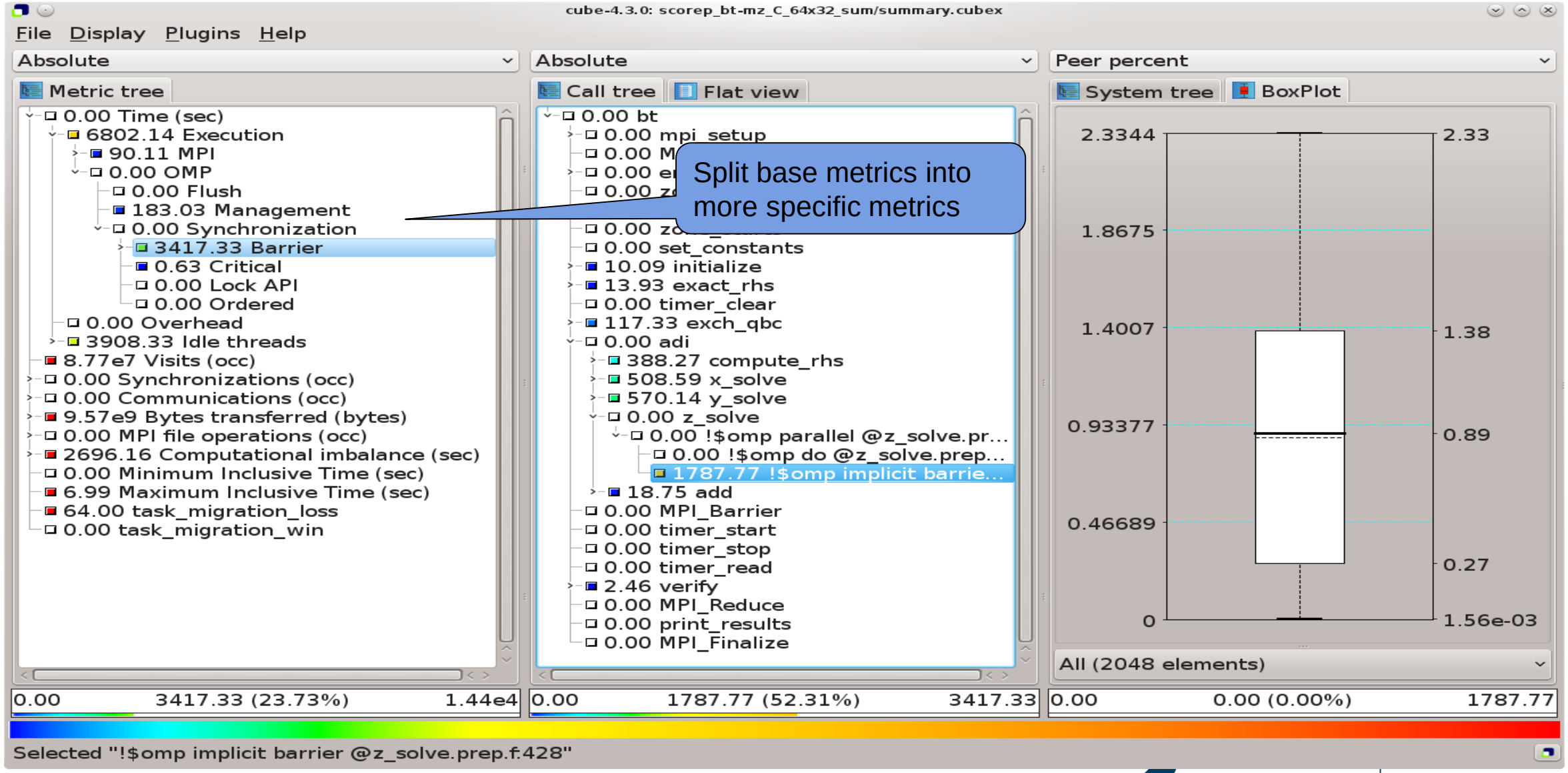
CALL-PATH PROFILE: EXAMPLE (CONT.)



CALL-PATH PROFILE: EXAMPLE (CONT.)



CALL-PATH PROFILE: EXAMPLE (CONT.)



WHY IS THE BOTTLENECK THERE?

- This is highly application dependent!
- Might require additional measurements
  - Hardware-counter analysis
    - CPU utilization
    - Cache behavior
  - Selective instrumentation
  - Automatic/manual event trace analysis

HARDWARE COUNTERS

- Counters: set of registers that count processor events, e.g. floating point operations or cycles
- Number of registers, counters and simultaneously measurable events vary between platforms
- Can be measured by:
  - perf:
    - Integrated in Linux since Kernel 2.6.31
    - Library and CLI
  - LIKWID:
    - Direct access to MSRs (requires Kernel module)
    - Consists of multiple tools and an API
  - PAPI (Performance API)

PAPI

- Portable API: Uses the same routines to access counters across all supported architectures
- Used by most performance analysis tools

- High-level interface:
  - Predefined standard events, e.g. PAPI_FP_OPS
  - Availability and definition of events varies between platforms
  - List of available counters: `papi_avail (-d)`
- Low-level interface:
  - Provides access to all machine specific counters
  - Non-portable
  - More flexible
  - List of available counters: `papi_native_avail`

TRACE GENERATION & ANALYSIS W/ SCALASCA

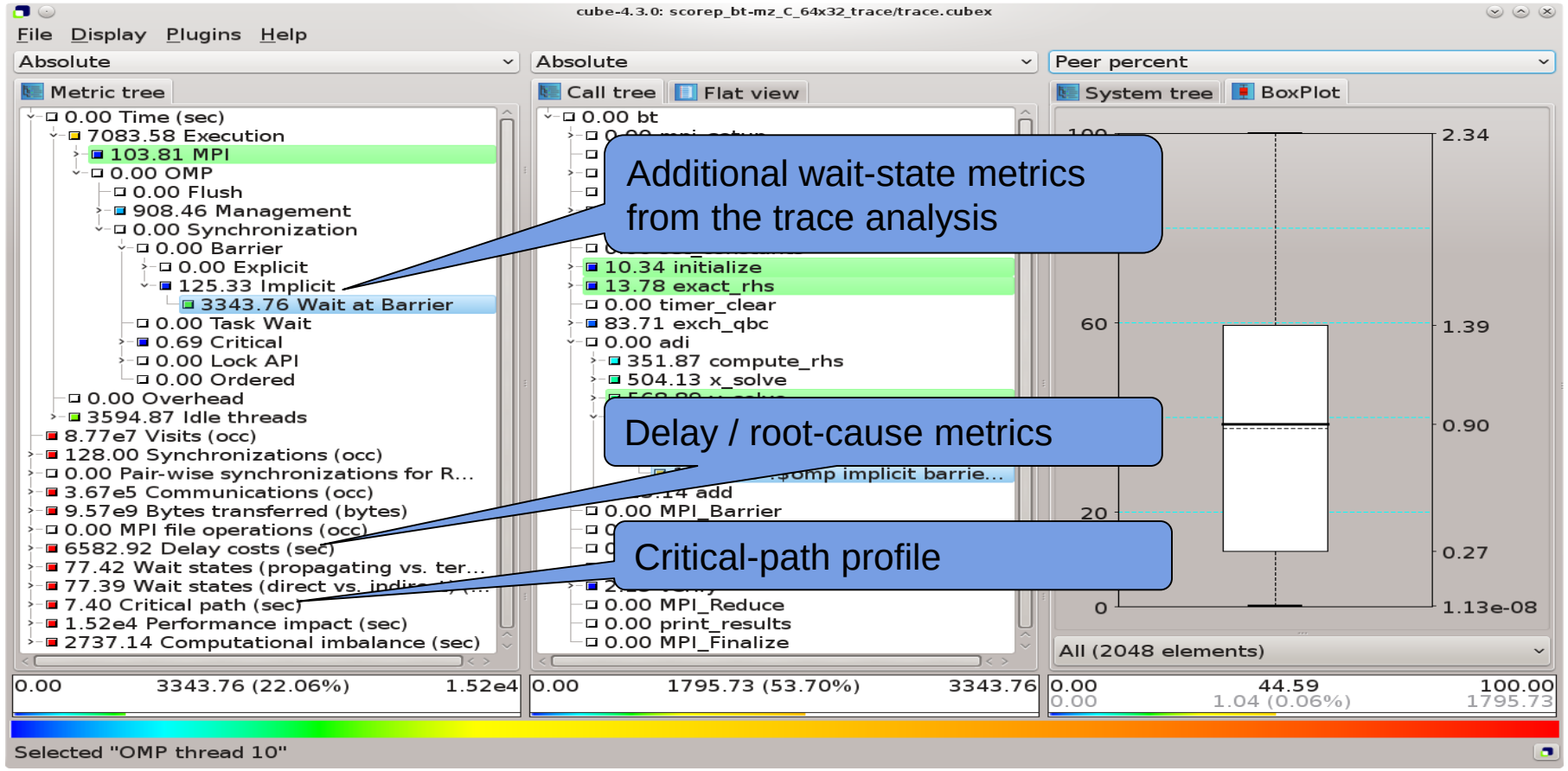
- Enable trace collection & analysis using "`-t`" option of "`scalasca -analyze`":

```
############################
##  In the job script:  ##
############################

module load ENV Score-P Scalasca
export SCOREP_TOTAL_MEMORY=120MB   # Consult score report
scalasca -analyze -f filter.txt -t \
    runjob --ranks-per-node P --np n [...] --exe ./myprog
```

- **ATTENTION:**
  - Traces can quickly become extremely large!
  - Remember to use proper filtering, selective instrumentation, and Score-P memory specification
  - Before flooding the file system, **ask us for assistance!**

SCALASCA TRACE ANALYSIS EXAMPLE



RECAP

- Performance tools provide detailed insight into application behavior
- Help identify bottlenecks and tuning potential

- Generating data is easy, getting a good measurement is not
- Think about what to record at which granularity
- Take care setting up measurement environment

- Talk to us – we‘re happy to assist you

QUESTIONS