scalasca

Tutorial Exercise NPB-MZ-MPI/BT

Brian Wylie & Markus Geimer
Jülich Supercomputing Centre
scalasca@fz-juelich.de
August 2012

Performance analysis steps

0. Reference preparation for validation
1. Program instrumentation: `skin`
2. Summary measurement collection & analysis: `scan [-s]`
3. Summary analysis report examination: `square`
4. Summary experiment scoring: `square -s`
5. Event trace collection & analysis: `scan -t`
6. Event trace analysis report examination: `square`

- Configuration & customization
  - Instrumentation, Measurement, Analysis, Presentation

NPB-MZ-MPI/BT

- Intermediate-level tutorial example
- Available in MPI, OpenMP & **hybrid OpenMP/MPI** variants
  - also MPI File I/O variants (collective & individual)
- Summary measurement collection & analysis
  - Automatic instrumentation
    - OpenMP, MPI & application functions
  - Summary analysis report examination
  - PAPI hardware counter metrics
- Trace measurement collection & analysis
  - Filter determination, specification & configuration
  - Automatic trace analysis report patterns
- Manual and PDT instrumentation
- Measurement configuration
- Analysis report algebra

Load the Scalasca module

- Load the module

```
% module load UNITE
UNITE loaded
% module load scalasca
scalasca/1.4.2 loaded
```

- ... and run `scalasca` for brief usage information

```
% scalasca
Scalasca 1.4.2
Toolset for scalable performance analysis of large-scale applications
usage: scalasca [-v][-n] {action}
    1. prepare application objects and executable for measurement:
       scalasca -instrument <compile-or-link-command>       # skin
    2. run application under control of measurement system:
       scalasca -analyze <application-launch-command>       # scan
    3. interactively explore measurement analysis report:
       scalasca -examine <experiment-archive|report>        # square

   -v: enable verbose commentary
   -n: show actions without taking them
   -h: show quick reference guide (only)
```

NPB instrumentation

- Prefix compile/link commands in Makefile definitions (config/make.def) with the Scalasca instrumenter

```
MPIF77 = scalasca -instrument mpif77
FLINK = $(MPIF77)
FFLAGS = -O -fopenmp

bt-mz: $(OBJECTS)
        $(FLINK) $(FFLAGS) -o bt-mz $(OBJECTS)
.f.o:
        $(MPIF77) $(FFLAGS) -c $<
```

- or use PREP macro as customizable preparation preposition

```
MPIF77 = $(PREP) mpif77
```

  - By default, PREP macro is not set and no instrumentation is performed for a regular “production” build
  - Specifying a PREP value in the Makefile or on the make command line uses it as a preposition, e.g., for instrumentation
    - % make PREP=“scalasca -instrument” ...

```
scalasca -instrument  mpif77 -O -fopenmp -c bt.f
```

NPB-MZ-MPI/BT instrumented build

- Return to root directory and clean-up

```
% make clean
```

- Re-build specifying Scalasca instrumenter as PREP

```
% make bt-mz CLASS=B NPROCS=4 PREP="scalasca -instrument"
cd BT-MZ; make CLASS=B NPROCS=4 VERSION=
gmake: Entering directory 'BT-MZ'
cd ../sys; cc  -o setparams setparams.c
../sys/setparams bt-mz 4 B
scalasca -instrument mpif77 -c  -O -fopenmp  bt.f
...
scalasca -instrument mpif77 -c  -O -fopenmp  setup_mpi.f
cd ../common;  scalasca -instrument mpif77 -c -O -fopenmp timers.f
scalasca -instrument mpif77 -O -fopenmp -o ../bin.scalasca/bt-mz_B.4 \
bt.o make_set.o initialize.o exact_solution.o exact_rhs.o \
set_constants.o adi.o define.o copy_faces.o rhs.o solve_subs.o \
x_solve.o y_solve.o z_solve.o add.o error.o verify.o setup_mpi.o \
../common/print_results.o ../common/timers.o
INFO: Instrumented executable for OMP+MPI measurement
gmake: Leaving directory 'BT-MZ'
```

BT-MZ summary measurement

- Run the application using the Scalasca measurement collection & analysis nexus prefixed to launch command

```
% cd bin.scalasca
% OMP_NUM_THREADS=4  scalasca -analyze  mpiexec -np 4 ./bt-mz_B.4
S=C=A=N: Scalasca 1.3 runtime summarization
S=C=A=N: ./epik_bt-mz_B_4x4_sum experiment archive
S=C=A=N: Sun Mar 29 16:36:31 2009: Collect start
mpiexec -np 4 ./bt-mz_B.4
[00000]EPIK: Created new measurement archive ./epik_bt-mz_B_4x4_sum
[00000]EPIK: Activated ./epik_bt-mz_B_4x4_sum [NO TRACE] (0.006s)

         [... Application output ...]

[00000]EPIK: Closing experiment ./epik_bt-mz_B_4x4_sum
[00000]EPIK: 164 unique paths (178 max paths, 7 max frames, 0 unknown)
[00000]EPIK: Unifying... done (0.023s)
[00000]EPIK: Collating... done (0.049s)
[00000]EPIK: Closed experiment ./epik_bt-mz_B_4x4_sum (0.073s)
S=C=A=N: Sun Mar 29 16:36:45 2009: Collect done (status=0) 57s
S=C=A=N: ./epik_bt-mz_B_4x4_sum complete.
```

- Produces experiment directory ./epik_bt-mz_B_4x4_sum

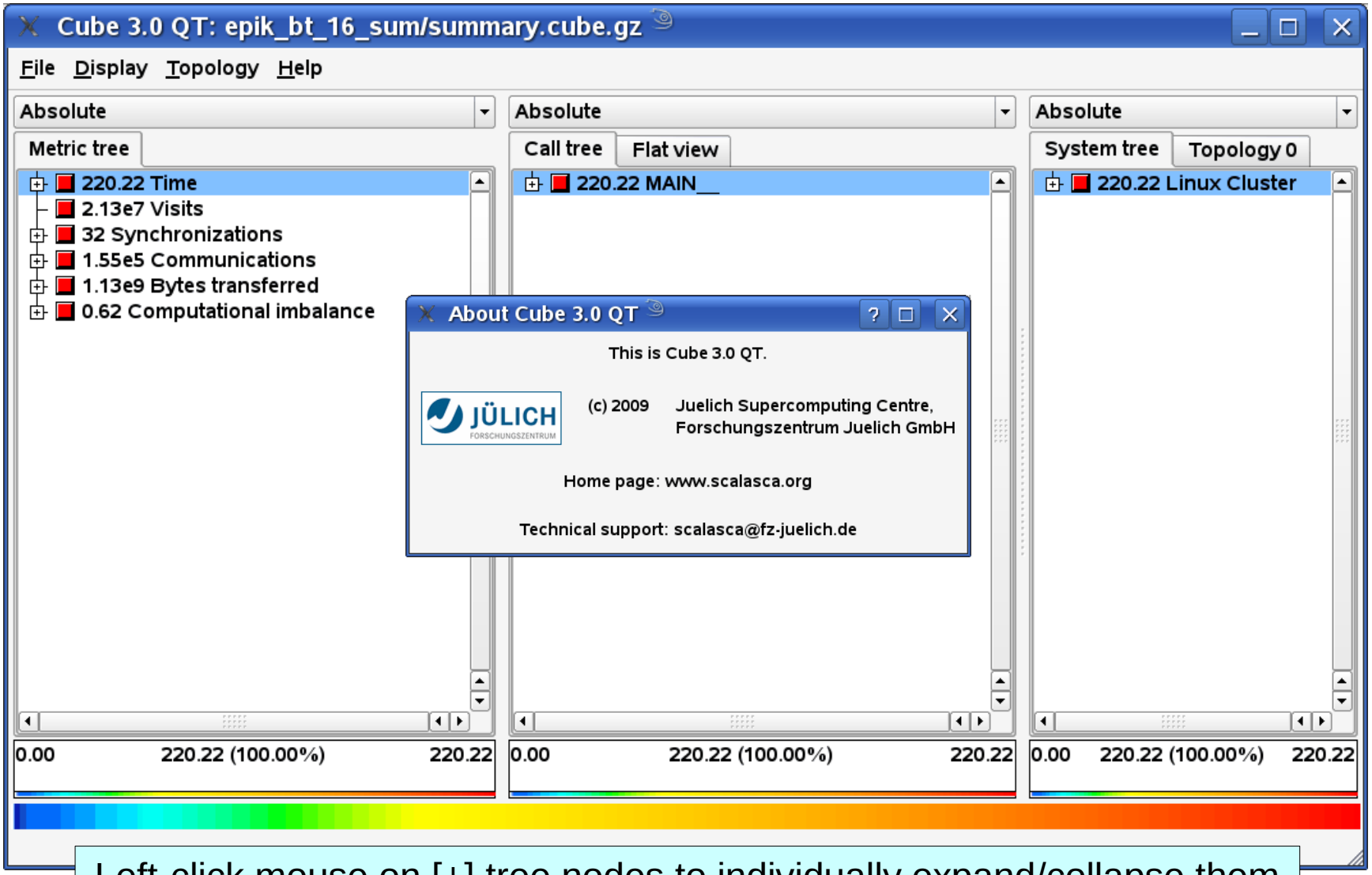
BT-MZ summary analysis report examination

- Interactive exploration with Scalasca GUI

```
% scalasca -examine  epik_bt-mz_B_4x4_sum
INFO: Post-processing runtime summarization result...
INFO: Displaying ./epik_bt-mz_B_4x4_sum/summary.cube...

                  [GUI showing summary analysis report]
```

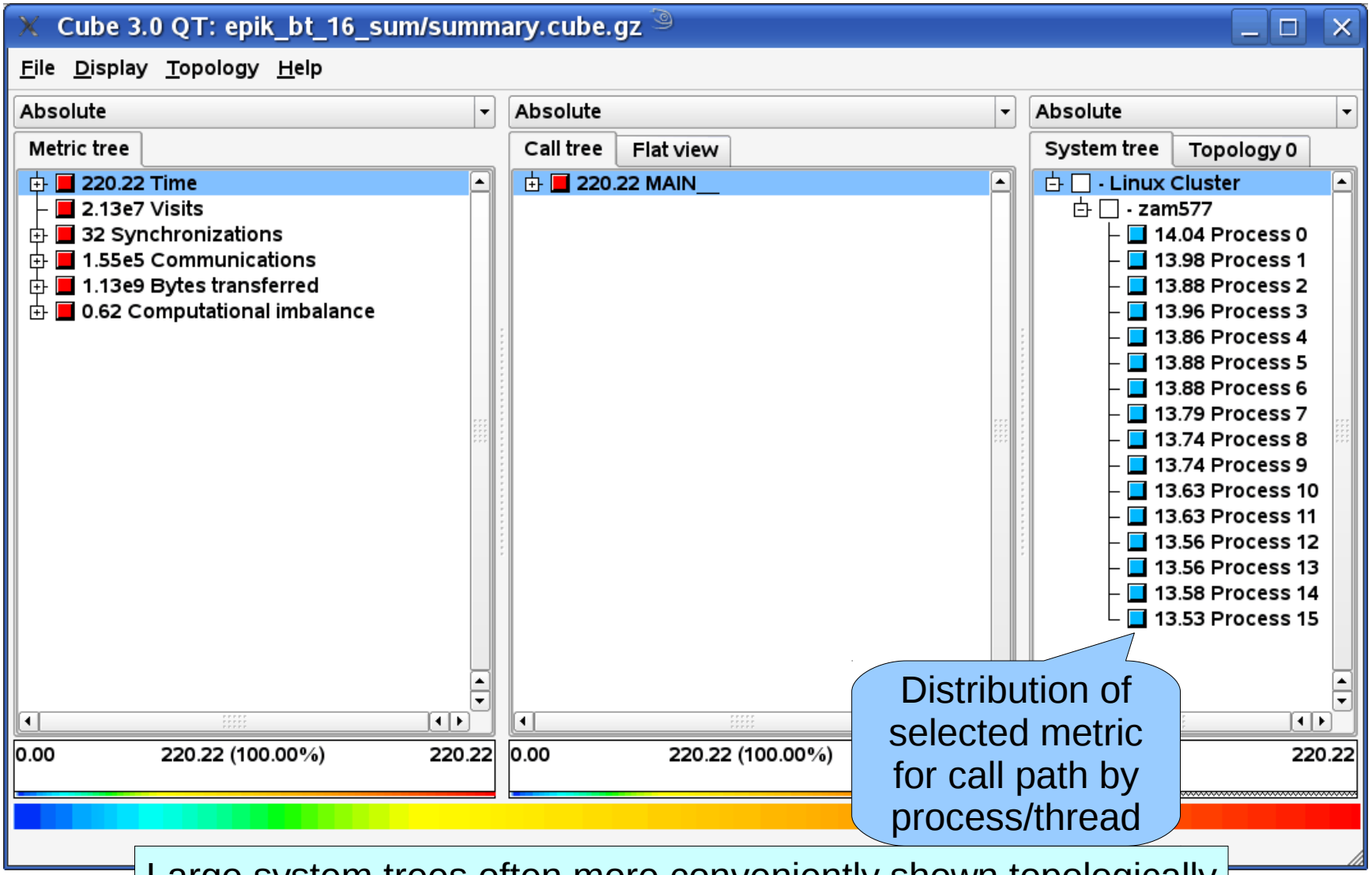
- The measurement archive directory ultimately contains
  - a copy of the execution output (epik.log)
  - a record of the measurement configuration (epik.conf)
  - the basic analysis report that was collated after measurement (epitome.cube)
  - the complete analysis report produced during post-processing (summary.cube.gz)

Analysis report exploration (opening view)



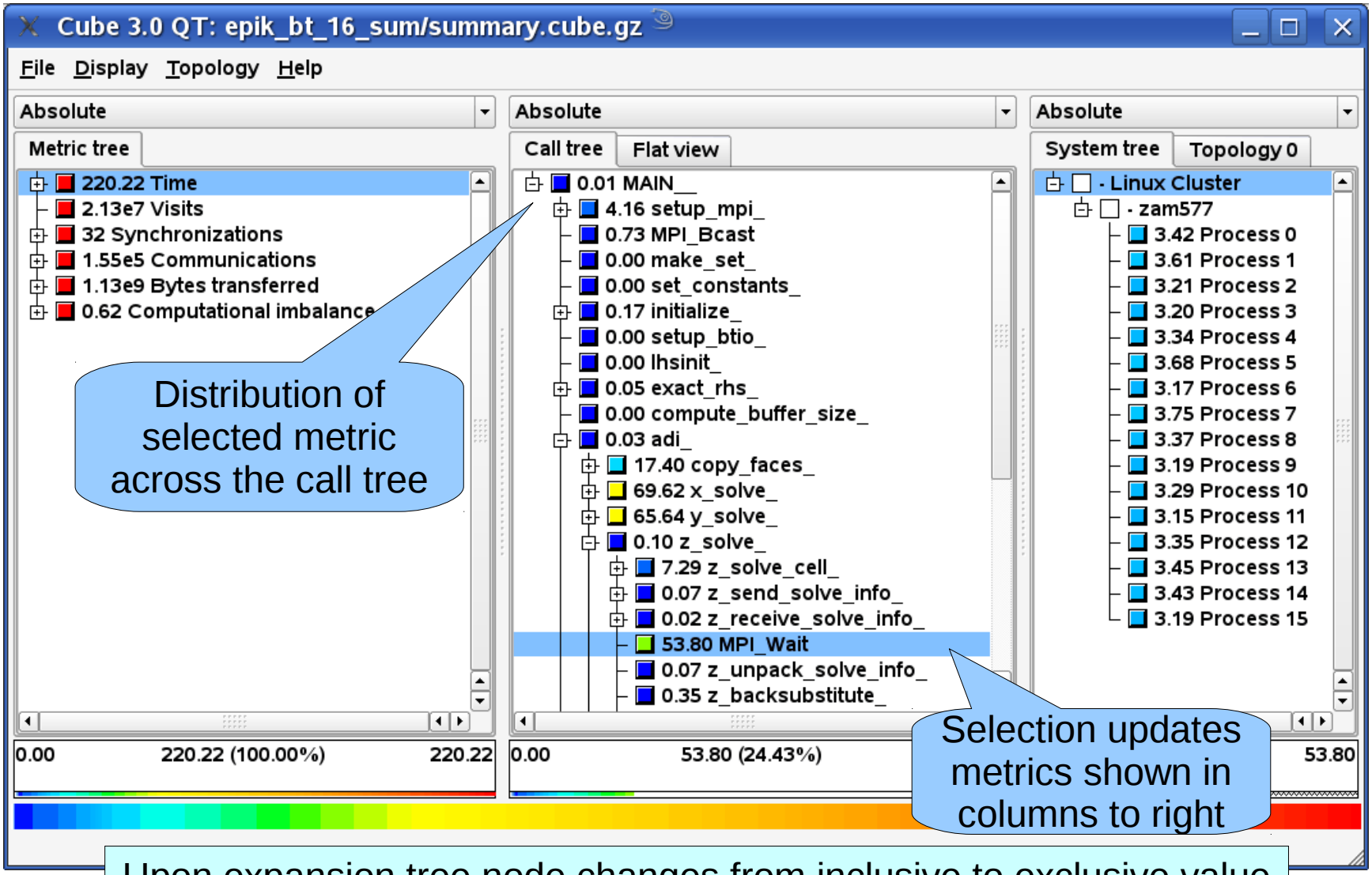
Left-click mouse on [+] tree nodes to individually expand/collapse them

Analysis report exploration (system tree)

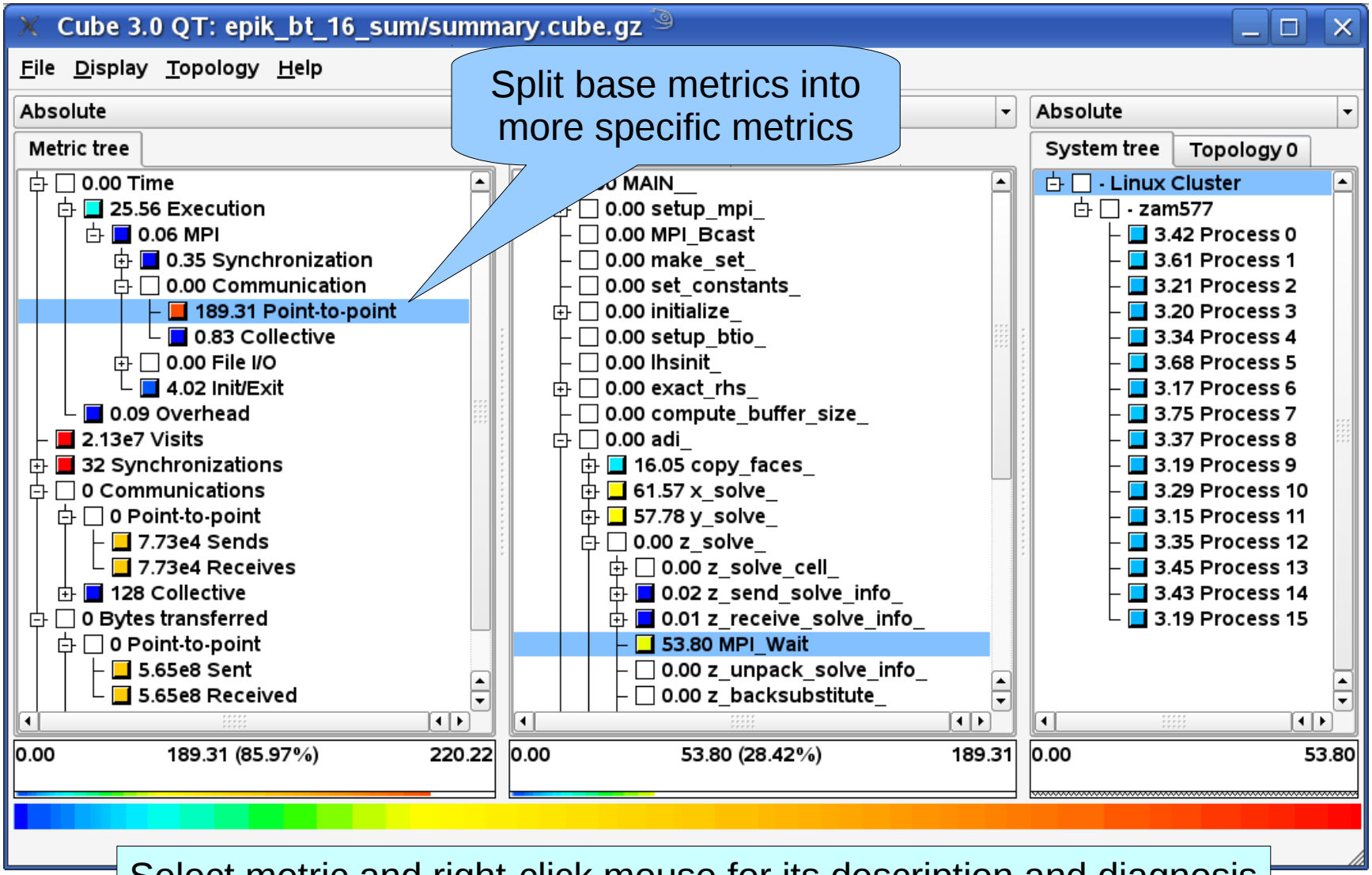


Large system trees often more conveniently shown topologically

Analysis report exploration (call tree)

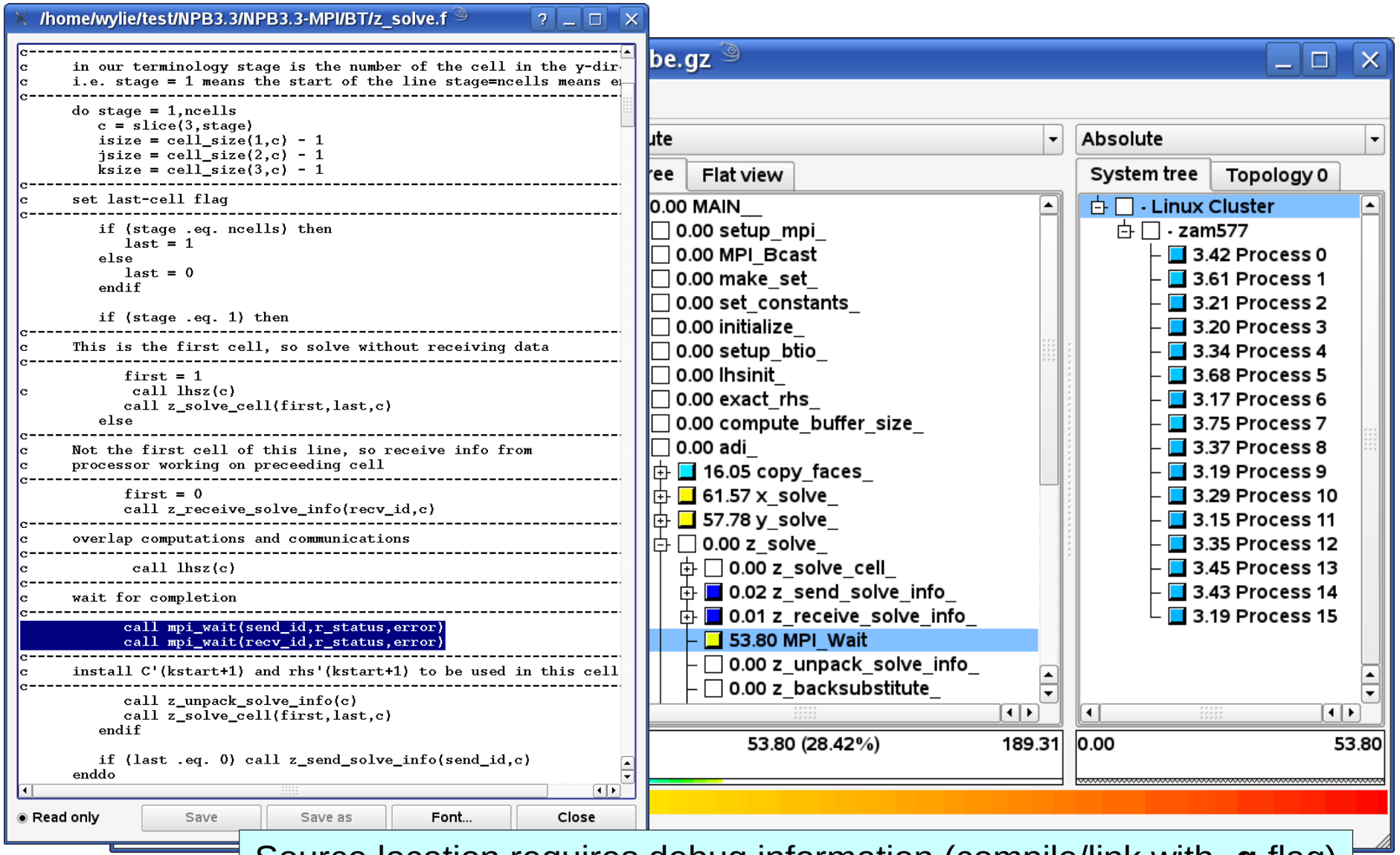


Analysis report exploration (metric tree)



Select metric and right-click mouse for its description and diagnosis

Analysis report exploration (source browser)



Source location requires debug information (compile/link with **-g** flag)

Congratulations!?

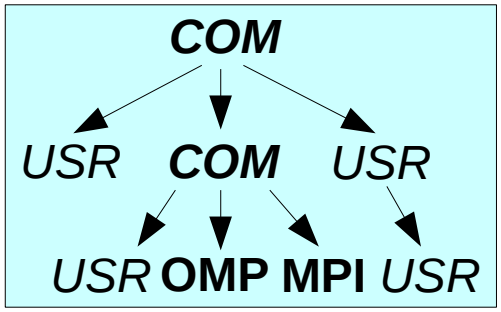
- If you made it this far, you successfully used Scalasca to
  - *instrument* the application
  - *analyze* its execution with a summary measurement, and
  - *examine* it with the interactive analysis report explorer GUI
- ... revealing the call-path profile annotated with
  - Time metrics (including MPI & OpenMP times)
  - Visit counts
  - MPI message statistics (sends/receives, bytes sent/received)
  - Computational imbalance
- ... but how ***good*** was the measurement?
  - The measured execution produced the desired valid result
  - however, the execution took rather longer than expected!
    - even when ignoring measurement start-up/completion, therefore
    - it was probably dilated by instrumentation/measurement overhead

BT-MZ summary analysis report scoring

- Report scoring as textual output

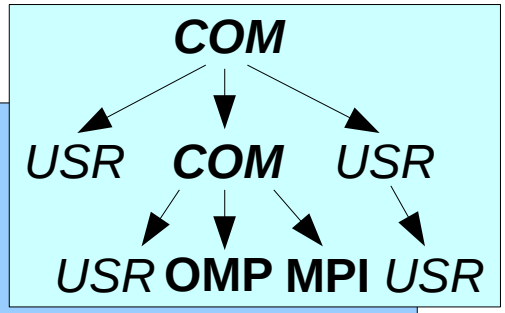
```
% scalasca -examine -s  epik_bt-mz_B_4x4_sum
cube3_score -r ./epik_bt-mz_B_4x4_sum/summary.cube
Reading ./epik_bt-mz_B_4x4_sum/summary.cube... done.
Est. aggregate size of event trace (total_tbc):  39,231,218,072 bytes
Est. size of largest process trace (max_tbc):     2,632,541,576 bytes
(When tracing set ELG_BUFFER_SIZE to avoid intermediate flushes or
 reduce requirements using filter file listing names of USR regions.)

INFO: Score report written to ./epik_bt-mz_B_4x4_sum/epik.score
```

- Region/callpath classification
  - MPI (pure MPI library functions)
  - OMP (pure OpenMP functions/regions)
  - USR (user-level source local computation)
  - COM (“combined” USR + OpenMP/MPI)
  - ANY/ALL (aggregate of all region types)



BT-MZ summary analysis report breakdown

- Score report breakdown by region




```
% less  epik_bt-mz_B_4x4_sum/epik.score

flt  type      max_tbc     time       % region
      ANY   2632541576   871.73   100.00 (summary) ALL
      MPI        73064    13.27     1.52 (summary) MPI
      OMP      5186496   496.36    56.94 (summary) OMP
      COM      1087824     3.15     0.36 (summary) COM
      USR   2626194144   358.88    41.17 (summary) USR

      USR    841575744   109.22    12.53 matmul_sub_
      USR    841575744   168.61    19.34 binvcrhs_
      USR    841575744    68.95     7.91 matvec_sub_
      USR     37120680     5.14     0.59 binvrhs_
      USR     37120680     4.10     0.47 lhsinit_
      USR     29960856     2.85     0.33 exact_solution_
      COM       308736     0.82     0.09 copy_x_face_
      COM       308736     0.81     0.09 copy_y_face_
      OMP       283008     2.07     0.24 !$omp parallel @exch_qbc.f:204
      OMP       283008     2.02     0.23 !$omp parallel @exch_qbc.f:215
      OMP       283008     2.16     0.25 !$omp parallel @exch_qbc.f:244
      OMP       283008     2.01     0.23 !$omp parallel @exch_qbc.f:255
...
```

BT-MZ summary analysis score

- Summary measurement analysis score reveals
  - Total size of event trace would be almost 40GB
  - Maximum trace buffer size would be over 2.5GB per thread
    - smaller buffer would require flushes to disk during measurement resulting in substantial perturbation
  - 99.76% of the trace requirements are for USR regions
    - purely computational routines never found on COM call-paths common to communication routines
  - These USR regions contribute around 40% of total time
    - however, much of that is very likely to be measurement overhead for a few frequently-executed small routines
- Advisable to tune measurement configuration
  - Specify an adequate trace buffer size
  - Specify a filter file listing (USR) regions not to be measured

BT-MZ summary analysis report filtering

- Report scoring with prospective filter listing USR regions

```
% scalasca -examine -s -f bt.filt  epik_bt-mz_B_4x4_sum
cube3_score -r -f bt.filt ./epik_bt-mz_B_4x4_sum/summary.cube
Applying filter "./bt.filt":
Estimated aggregate size of event trace (total_tbc):  16,852,888 bytes
Estimated size of largest process trace (max_tbc):     1,053,304 bytes

INFO: Score report written to ./epik_bt-mz_B_4x4_sum/epik.score_bt.filt

% less epik_bt-mz_B_4x4_sum/epik.score_bt.filt
flt   type    max_tbc      time       % region
  +    FLT 2626190016    358.88    41.17 (summary) FLT
  *    ANY    6351584    512.85    58.83 (summary) ALL-FLT
  -    MPI      73064     13.27     1.52 (summary) MPI-FLT
  -    OMP    5186496    496.36    56.94 (summary) OMP-FLT
  *    COM    1087824      3.15     0.36 (summary) COM-FLT
  *    USR       4152      0.00     0.00 (summary) USR-FLT

  +    USR  841575744    109.22    12.53 matmul_sub_
  +    USR  841575744    168.61    19.34 binvcrhs_
  +    USR  841575744     68.95     7.91 matvec_sub_
  +    USR   37120680      5.14     0.59 binvrhs_
  +    USR   37120680      4.10     0.47 lhsinit_
  +    USR   29960856      2.85     0.33 exact_solution_
...
```

Filtered routines marked with '+'

```
% cat bt.filt
# bt-mz filter
matmul_sub_
binvcrhs_
matvec_sub_
binvrhs_
lhsinit_
exact_solution_
timer_*
```

BT-MZ filtered summary measurement

- Rename former measurement archive directory, set new filter configuration and re-run the measurement

```
% mv epik_bt-mz_B_4x4_sum epik_bt-mz_B_4x4_sum.nofilt
% export EPK_FILTER=bt.filt
% OMP_NUM_THREADS=4  scalasca -analyze  mpiexec -np 4 ./bt-mz_B.4
S=C=A=N: Scalasca 1.3 runtime summarization
S=C=A=N: ./epik_bt-mz_4x4_sum experiment archive
S=C=A=N: Sun Mar 29 16:58:34 2009: Collect start
mpiexec -np 4 ./bt-mz_B.4
[00000.0]EPIK: Created new measurement archive ./epik_bt-mz_B_4x4_sum
[00000.0]EPIK: EPK_FILTER “bt.filt” filtered 10 of 113 functions
[00000.0]EPIK: Activated ./epik_bt-mz_B_4x4_sum [NO TRACE] (0.071s)

        [... Application output ...]

[00000.0]EPIK: Closing experiment ./epik_bt-mz_B_4x4_sum
[00000.0]EPIK: 134 unique paths (148 max paths, 7 max frames, 0 unkns)
[00000.0]EPIK: Unifying... done (0.014s)
[00000.0]EPIK: Collating... done (0.059s)
[00000.0]EPIK: Closed experiment ./epik_bt-mz_B_4x4_sum (0.075s)
S=C=A=N: Sun Mar 29 16:58:41 2009: Collect done (status=0) 36s
S=C=A=N: ./epik_bt-mz_B_4x4_sum complete.
```

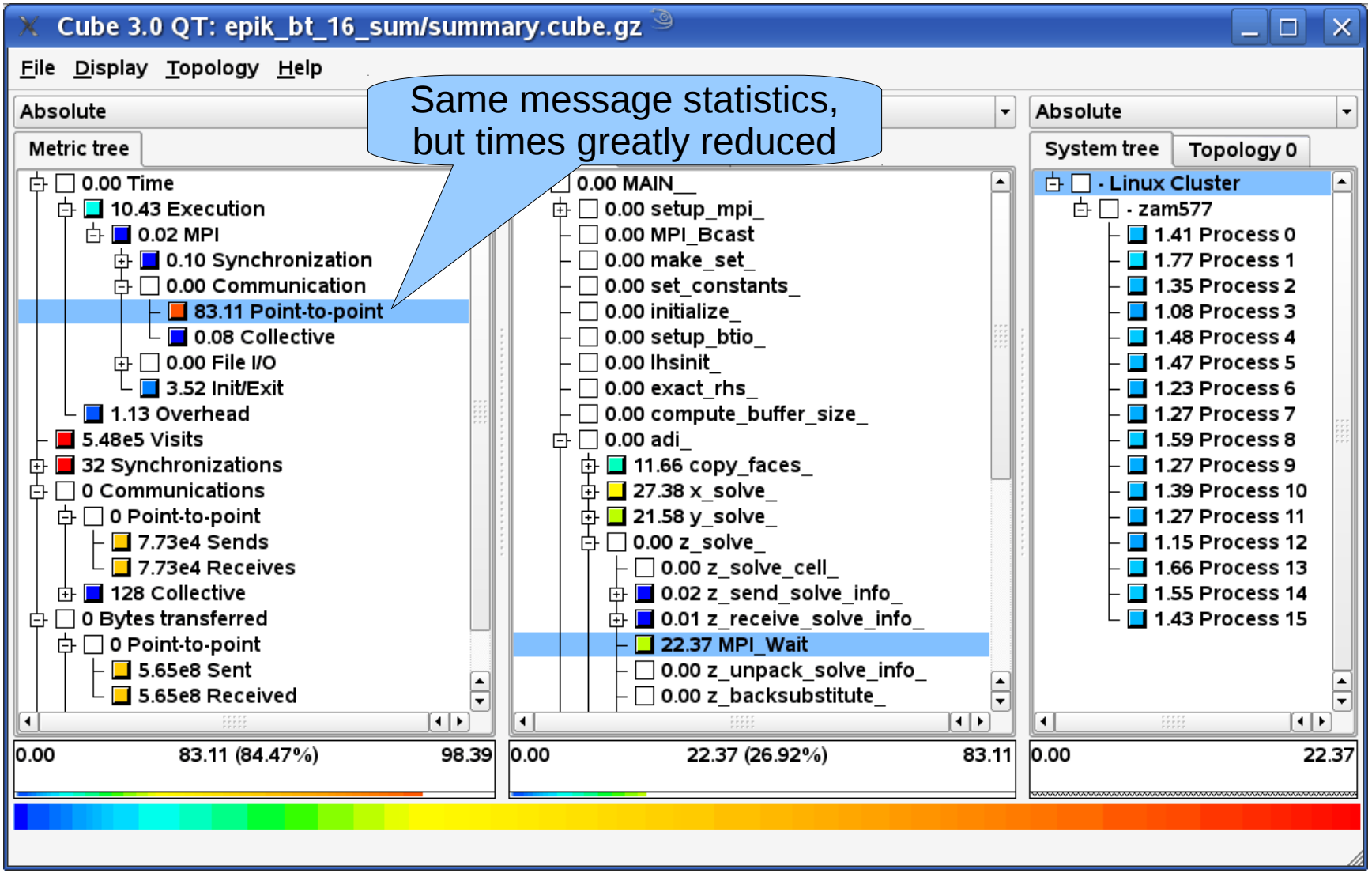
BT-MZ filtered summary analysis report score

- Scoring of new analysis report as textual output

```
% scalasca -examine -s  epik_bt-mz_B_4x4_sum
INFO: Post-processing runtime summarization result...
cube3_score ./epik_bt-mz_B_4x4_sum/summary.cube
Estimated aggregate size of event trace (total_tbc):  83,920,952 bytes
Estimated size of largest process trace (max_tbc):     6,351,584 bytes
...

INFO: Score report written to ./epik_bt-mz_B_4x4_sum/epik.score

flt  type       max_tbc         time       % region
      ANY       6351584       531.69  100.00 (summary) ALL
      MPI         73064        13.27    2.50 (summary) MPI
      OMP       5186496       515.11   96.88 (summary) OMP
      COM       1087824         3.22    0.61 (summary) COM
      USR          4152         0.00    0.00 (summary) USR
```

- Significant reduction in runtime (measurement overhead)
  - Not only reduced time for USR regions, but OMP reduced too!
- Further measurement tuning (filtering) may be appropriate
  - e.g., "`timer_*`" filters `timer_start_`, `timer_read_`, etc.

Summary analysis report exploration (filtered)



BT-MZ trace measurement collection ...

- Re-run the application using Scalasca nexus with “-t” flag

```
% OMP_NUM_THREADS=4 scalasca -analyze -t  mpiexec -np 4 ./bt-mz_B.4
S=C=A=N: Scalasca trace collection and analysis
S=C=A=N: ./epik_bt-mz_B_4x4_trace experiment archive
S=C=A=N: Sun Apr  5 18:50:57 2009: Collect start
mpiexec -np 4 ./bt-mz_B.4
[00000.0]EPIK: Created new measurement archive ./epik_bt-mz_B_4x4_trace
[00000.0]EPIK: EPK_FILTER "npb.filt" filtered 10 of 113 functions
[00000.0]EPIK: Activated ./epik_bt-mz_B_4x4_trace [10000000 bytes] (0.051s)

          [... Application output ...]

[00000.0]EPIK: Closing experiment ./epik_bt-mz_B_4x4_trace [0.069GB] (max 18466028)
[00000.0]EPIK: Flushed 6351570 bytes to file ./epik_bt-mz_B_4x4_trace/ELG/00000
[00000.0]EPIK: 134 unique paths (148 max paths, 7 max frames, 0 unknowns)
[00000.0]EPIK: Unifying... done (0.021s)
[00003.0]EPIK: Flushed 6351570 bytes to file ./epik_bt-mz_B_4x4_trace/ELG/00003
...
[00001.0]EPIK: Flushed 6351570 bytes to file ./epik_bt-mz_B_4x4_trace/ELG/00001
[00000.0]EPIK: 1flush=0.001GB@2.582MB/s, Pflush=0.015GB@35.458MB/s
[00000.0]EPIK: Closed experiment ./epik_bt-mz_B_4x4_trace (0.178s)
S=C=A=N: Sun Apr  5 18:51:05 2009: Collect done (status=0) 41s
[.. continued ...]
```

- Separate trace file per MPI rank written straight into new experiment directory ./epik_bt-mz_B_4x4_trace

BT-MZ trace measurement … analysis

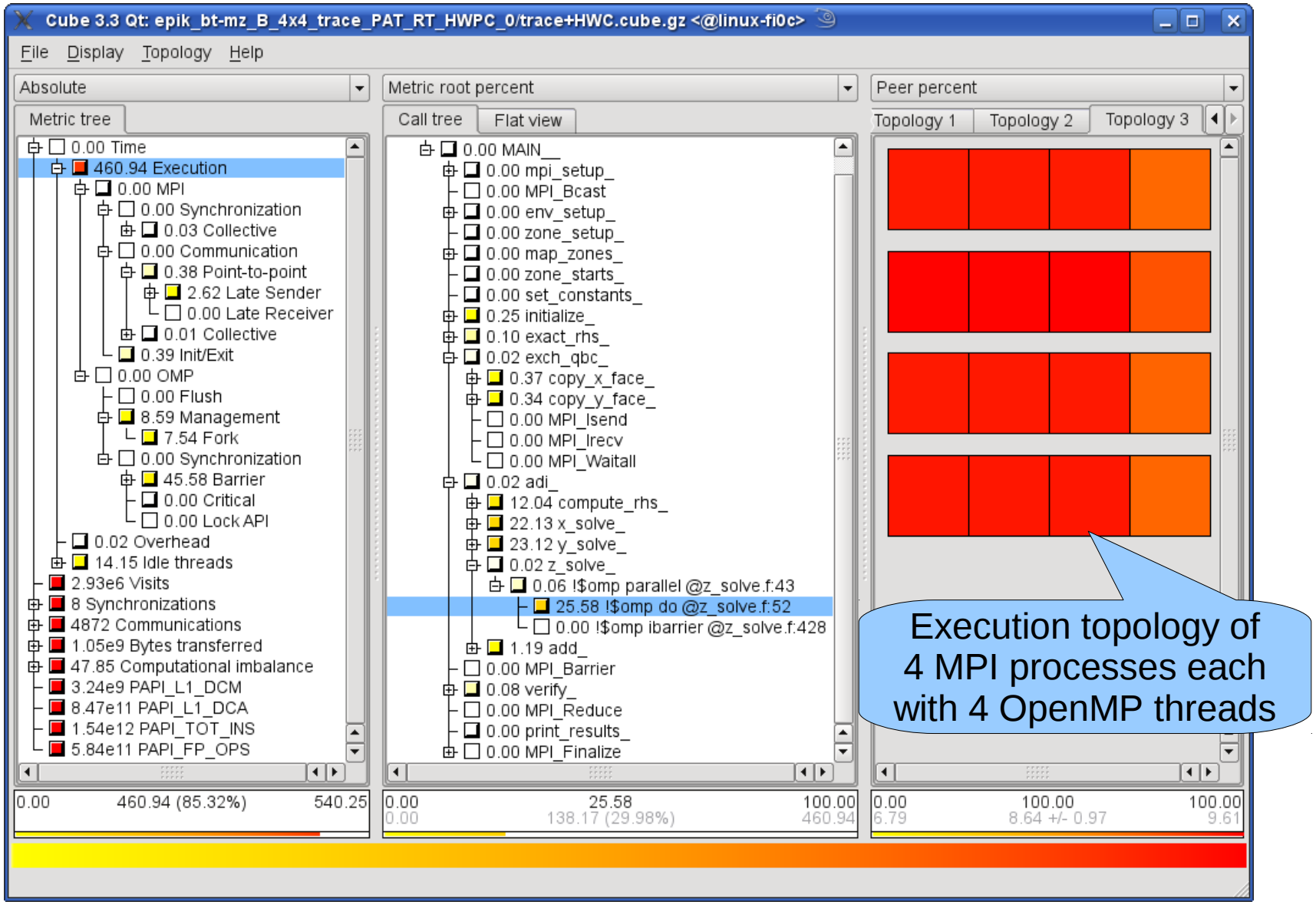
- Continues with automatic (parallel) analysis of trace files

```
S=C=A=N: Sun Apr  5 18:51:05 2009: Analyze start
mpiexec -np 4 scout.hyb ./epik_bt-mz_B_4x4_trace
SCOUT   Copyright (c) 1998-2009 Forschungszentrum Juelich GmbH

Analyzing experiment archive ./epik_bt-mz_B_4x4_trace

Reading definitions file  ... done (0.563s).
Reading event trace files ... done (0.495s).
Preprocessing             ... done (0.134s).
Analyzing event traces    ... done (2.186s).
Writing CUBE report       ... done (0.160s).

Total processing time     : 3.737s
Max. memory usage         : 47.504MB

S=C=A=N: Sun Apr  5 18:51:09 2009: Analyze done (status=0) 4s
S=C=A=N: ./epik_bt-mz_B_4x4_trace complete.
```

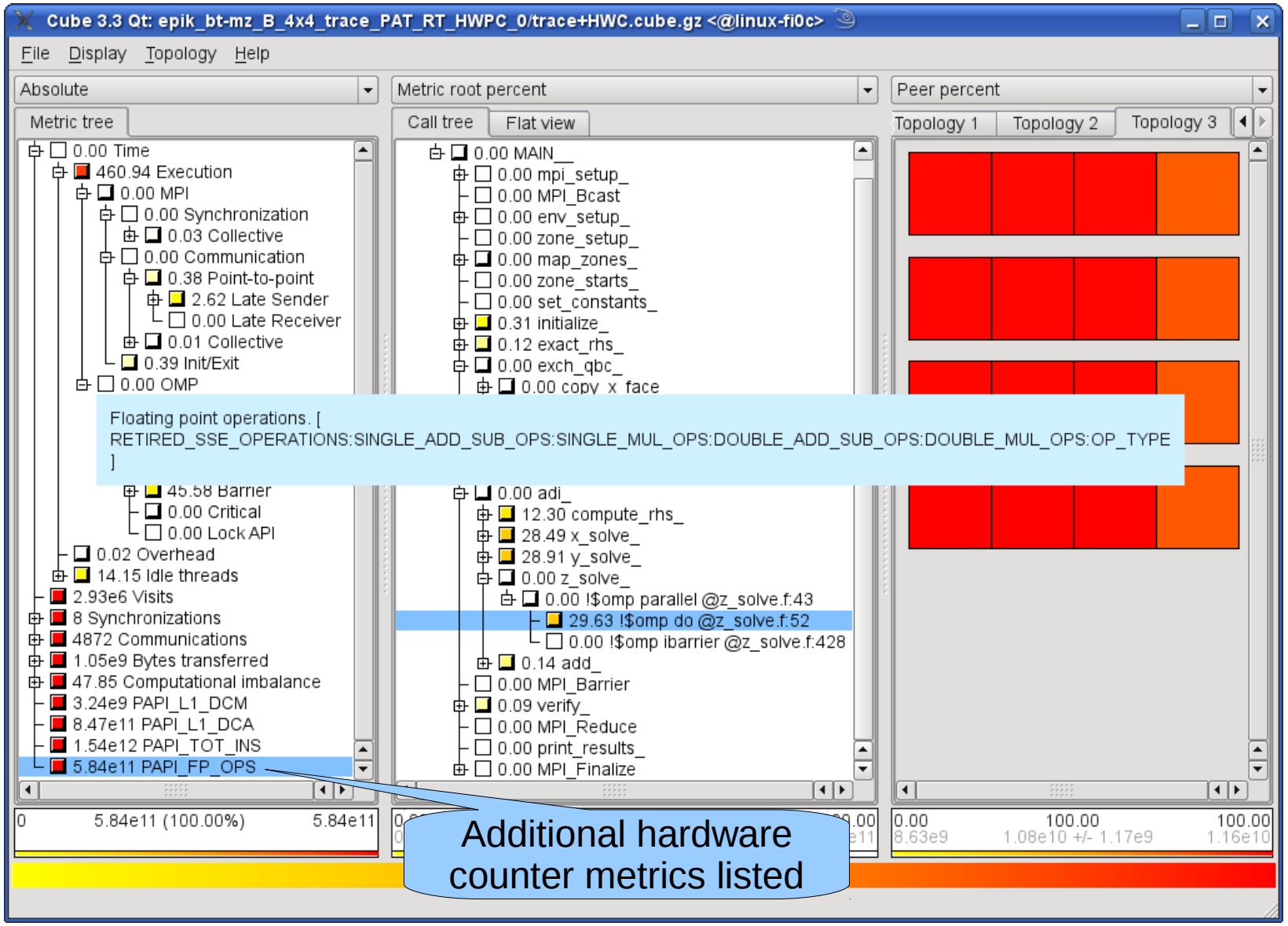
- Produces trace analysis report in experiment directory

```
% scalasca -examine epik_bt-mz_B_4x4_trace
INFO: Post-processing runtime summarization result...
INFO: Post-processing trace analysis report ...
INFO: Displaying ./epik_bt-mz_B_4x4_trace/trace.cube...

                  [GUI showing trace analysis report]
```

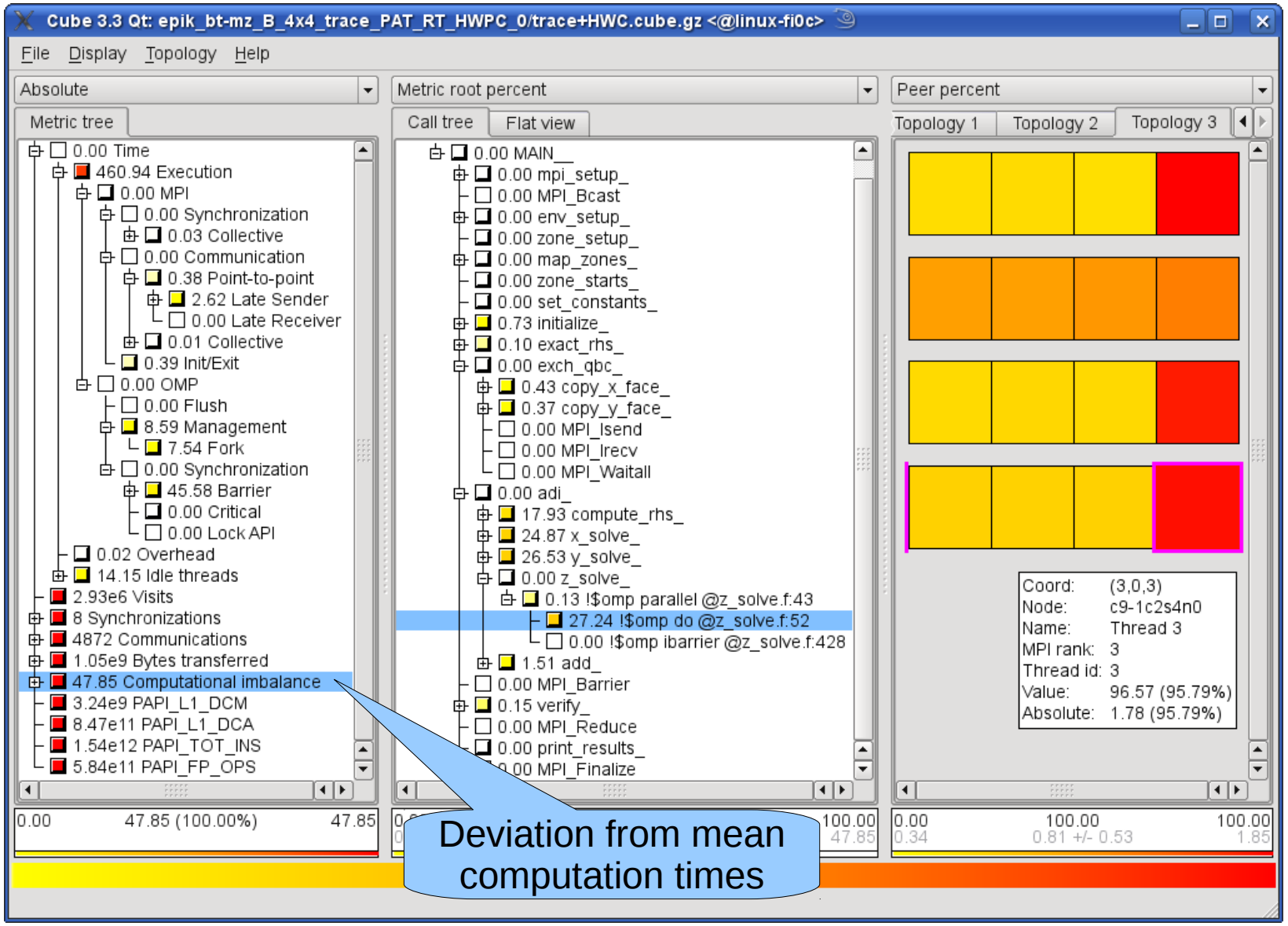
Scalasca topological presentation



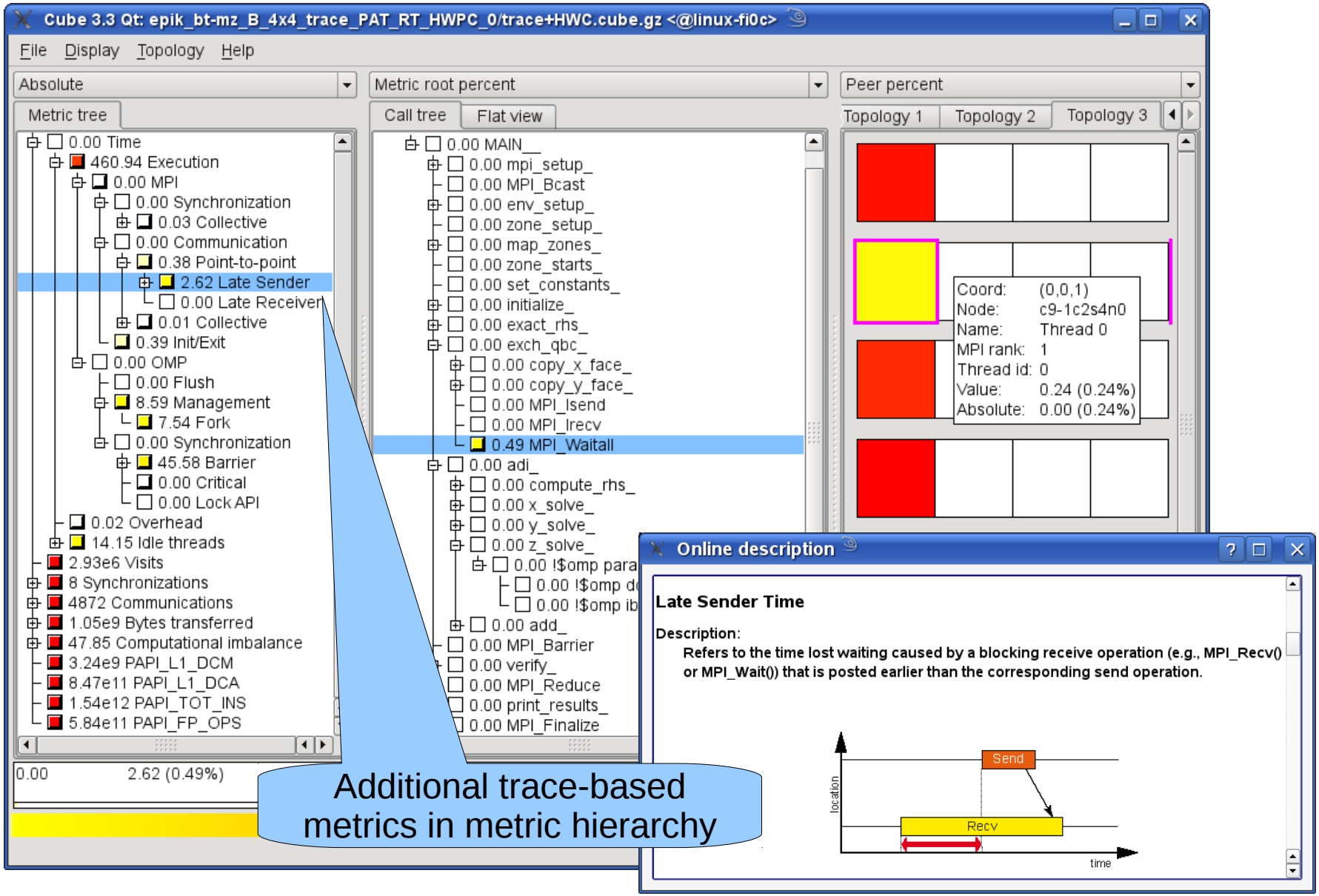
Scalasca hardware counter metrics



Scalasca computational imbalance heuristic



Scalasca trace analysis report exploration



Further examination of Scalasca experiments

- Scalasca analysis reports can be viewed with *ParaProf* for a multitude of interactive 2D & 3D graphical profiles

```
% paraprof epik_bt-mz_B_4x4_trace_PAT_RT_HWPC_0/trace.cube.gz
```

- Scalasca traces can be viewed directly with *Vampir7* for interactive timeline and communication matrix displays

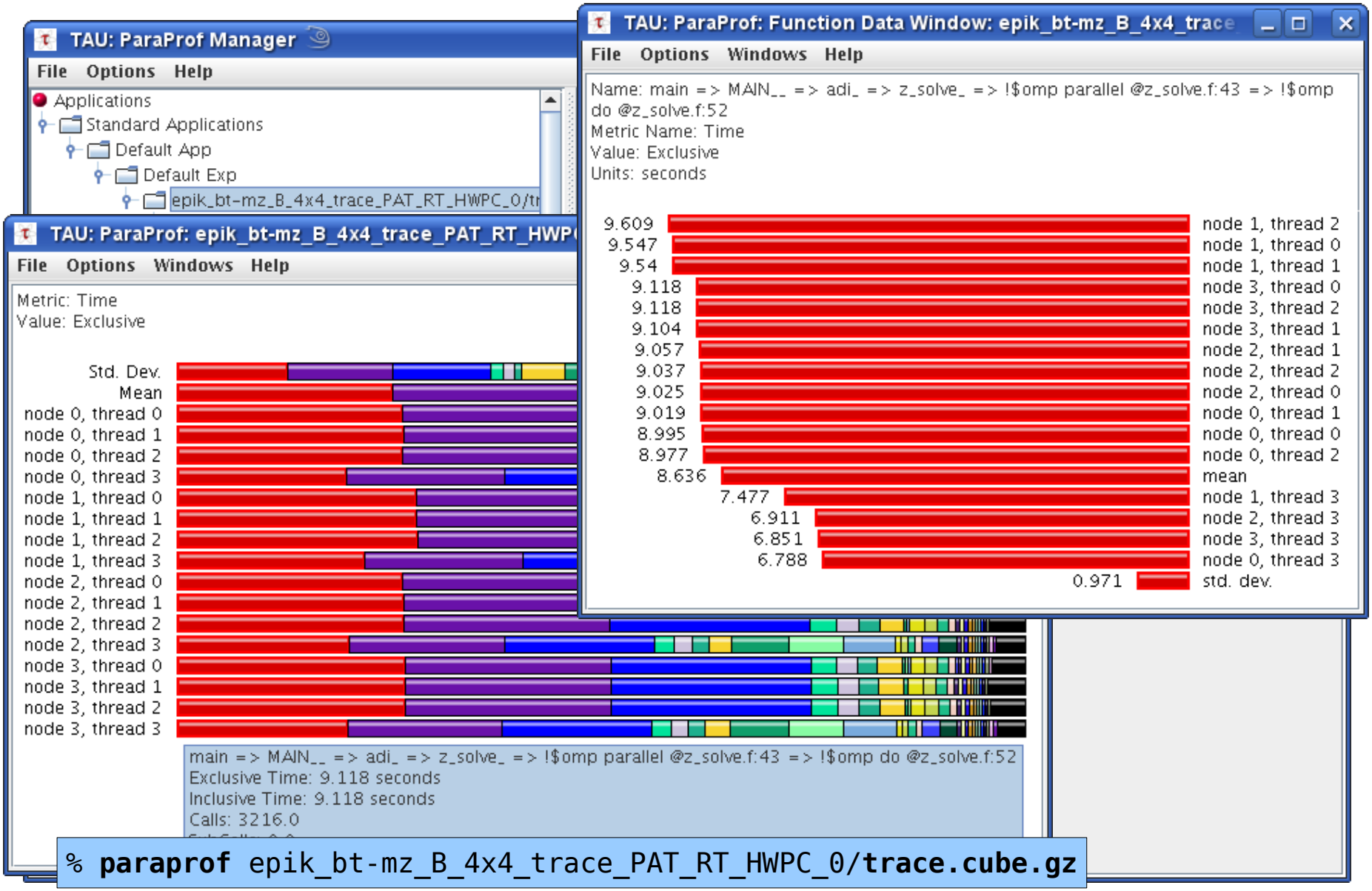
```
% vampir epik_bt-mz_B_4x4_trace_PAT_RT_HWPC_0/epik.esd
```

- Scalasca traces can also be merged and then converted to the formats of other analysis and visualization tools

```
% elg_merge epik_bt-mz_B_4x4_trace_PAT_RT_HWPC_0
% elg2prv   epik_bt-mz_B_4x4_trace_PAT_RT_HWPC_0
% wxparaver epik_bt-mz_B_4x4_trace_PAT_RT_HWPC_0/epik.prv
```

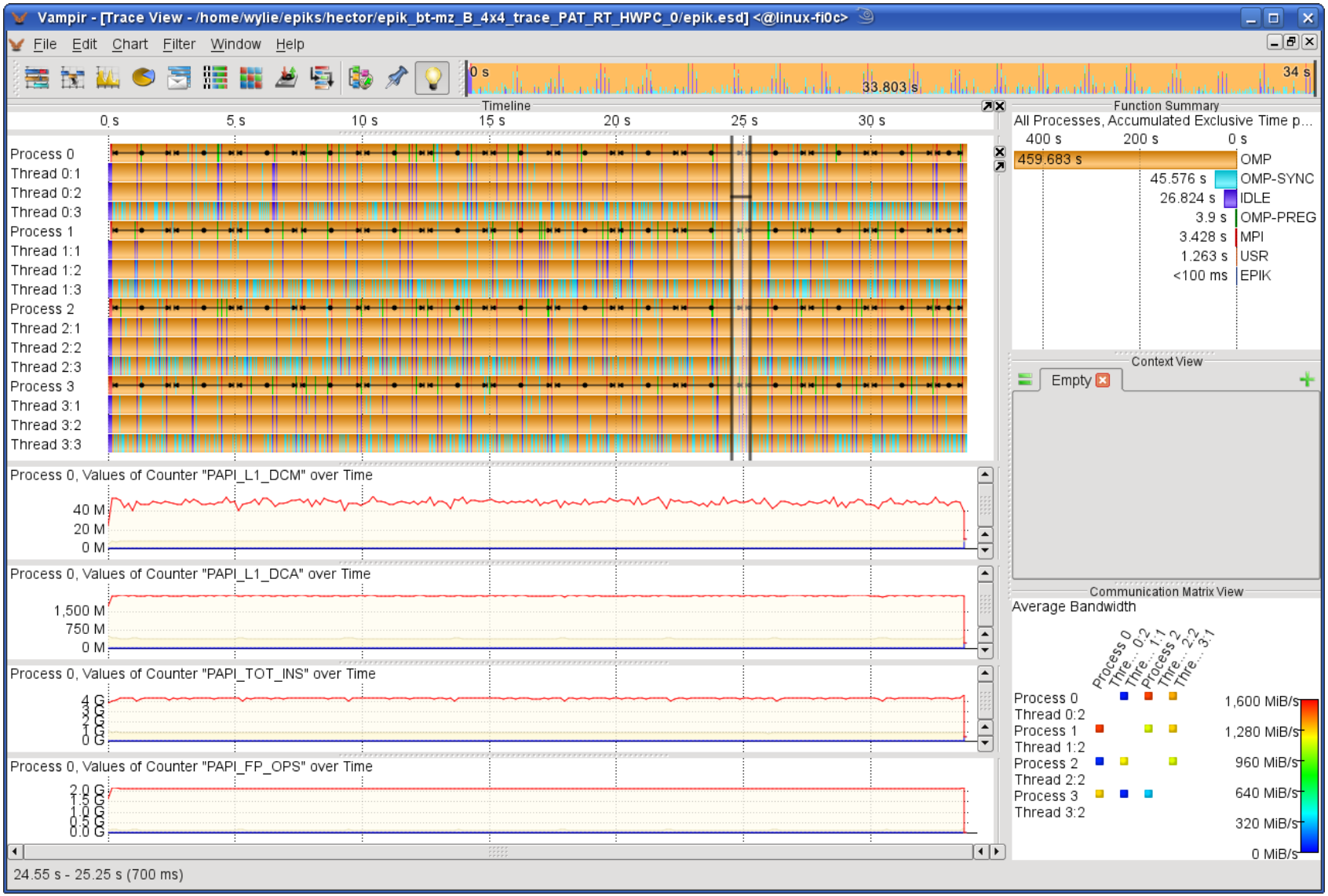
  - Trace merging and conversion are both done serially and therefore only practical for relatively small traces.
  - External tools can often manage to analyze traces that Scalasca's automatic trace analyzer can't handle

Paraprof views of Scalasca analysis report



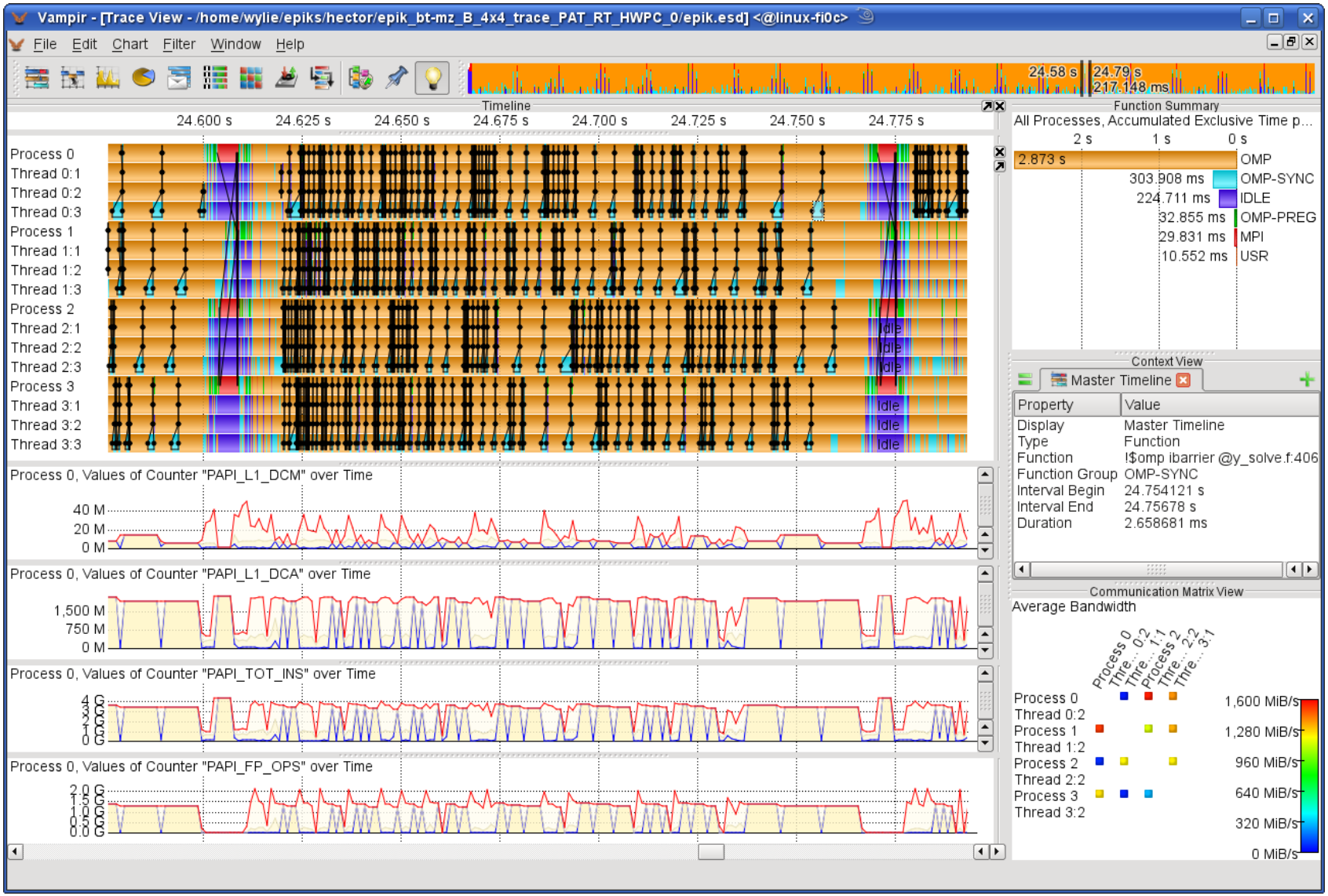

```
% paraprof epik_bt-mz_B_4x4_trace_PAT_RT_HWPC_0/trace.cube.gz
```

Vampir visual trace exploration (overview)

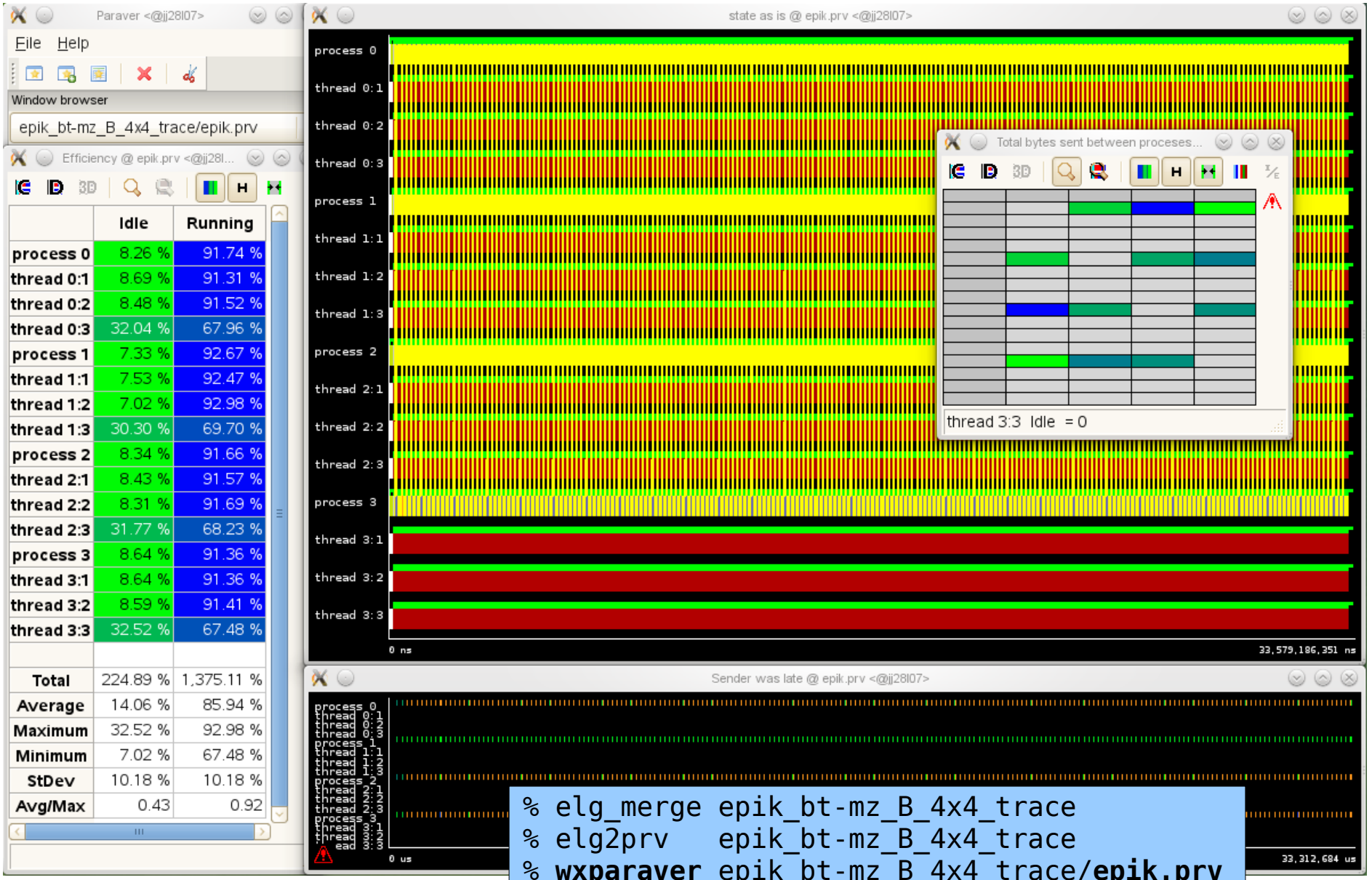



```
% vampir epik_bt-mz_B_4x4_trace_PAT_RT_HWPC_0/epik.esd
```

Vampir visual trace exploration (zoom)



Trace conversion & analysis with Paraver



	Idle	Running
process 0	8.26 %	91.74 %
thread 0:1	8.69 %	91.31 %
thread 0:2	8.48 %	91.52 %
thread 0:3	32.04 %	67.96 %
process 1	7.33 %	92.67 %
thread 1:1	7.53 %	92.47 %
thread 1:2	7.02 %	92.98 %
thread 1:3	30.30 %	69.70 %
process 2	8.34 %	91.66 %
thread 2:1	8.43 %	91.57 %
thread 2:2	8.31 %	91.69 %
thread 2:3	31.77 %	68.23 %
process 3	8.64 %	91.36 %
thread 3:1	8.64 %	91.36 %
thread 3:2	8.59 %	91.41 %
thread 3:3	32.52 %	67.48 %
Total	224.89 %	1,375.11 %
Average	14.06 %	85.94 %
Maximum	32.52 %	92.98 %
Minimum	7.02 %	67.48 %
StDev	10.18 %	10.18 %
Avg/Max	0.43	0.92

```
% elg_merge epik_bt-mz_B_4x4_trace
% elg2prv   epik_bt-mz_B_4x4_trace
% wxparaver epik_bt-mz_B_4x4_trace/epik.prv
```

Warnings and tips regarding tracing

- Traces can easily become extremely large and unwieldy
  - size is proportional to number of processes/threads (*width*), duration (*length*) and detail (*depth*) of measurement
- Traces containing intermediate flushes are of little value
  - uncoordinated flushes result in cascades of distortion
    - reduce size of trace such that it fits in available buffer space
- Traces should generally be written to a parallel filesystem
  - /work or /scratch are typically provided for this purpose
- Moving large traces between filesystems often impractical
  - however, systems with more memory can analyse larger traces
    - alternatively, run trace analyzer with undersubscribed nodes
- Traces can be archived or deleted after analysis completed to recover storage space
  - Scalasca binary trace data is stored in the ELG subdirectory

Further information

- Consult quick reference guide for further information

```
% scalasca -h
Scalasca 1.4 – quick reference guide
pdfview /UNITE/packages/scalasca/1.4/doc/manuals/QuickReference.pdf

                [PDF viewer showing quick reference guide]
```

- CUBE GUI provides context sensitive help and on-line metric descriptions (including problem diagnosis hints)
- EPIK archive directories contain analysis report(s), measurement collection & analysis logs, etc.
- Instrumentation, measurement, analysis & presentation can all be extensively customized
  - covered in more detailed presentation
- Visit www.scalasca.org or mail scalasca@fz-juelich.de

Scalasca usage recap

0. Reference preparation for validation
1. Program instrumentation: `skin`
2. Summary measurement collection & analysis: `scan [-s]`
3. Summary analysis report examination: `square`
4. Summary experiment scoring: `square -s`
5. Event trace collection & analysis: `scan -t`
6. Event trace analysis report examination: `square`

- General usage/help: `scalasca [-h]`
- Instrumentation, measurement, analysis & presentation can all be extensively customized
  - covered in more detailed presentation
- Visit www.scalasca.org or mail scalasca@fz-juelich.de

skin – Scalasca application instrumenter

- Prepares application objects & executables for measurement
  - ***skin*** = **scalasca -instrument**
  - skin [options] <*compile-or-link-command*>
    - defaults to automatic instrumentation of USR routines by compiler
      - available for most compilers, but not all
      - when not desired, disable with -comp=none
    - for OpenMP, includes source-level pre-processing of directives
    - for MPI, links wrappers to PMPI library routines
  - [-pdt] pre-process sources with PDToolkit (when available)
    - configurable instrumentation of specified routines (or all by default)
  - Manual instrumentation activation
    - offers complementary program structure information for analyses via user-provided annotations (e.g., phases, loops, ...)
    - [-user] enable EPIK user instrumentation API macros
    - [-pomp] enable processing of POMP region pragmas/directives

SKIN configuration variables

- Override default instrumenter configuration
  - SKIN_VERBOSE={ 0 | 1 }
    - provides verbose report of instrumenter compile and link commands
    - primarily for debugging and helpful when reporting problems
  - SKIN_MODE={ auto | MPI | MPI+OMP | OMP | serial | none }
    - the instrumenter generally determines the mode itself, but in some cases another mode may be more appropriate
    - “none” can be specified for pre-build configure tests and then unset for the actual build
  - SKIN_COMP={ all | none | … }
    - compiler instrumentation specification

MPI library instrumentation

- The MPI standard specifies the PMPI library interposition ("wrapping") mechanism based on weak symbols
  - simply requires re-linking statically-linked executables or library preloading for dynamically-linked executables
  - widely used by debugging and performance tools
  - generally negligible measurement overhead
    - event traces still grow linearly with number of MPI events!
    - certain MPI operations may be executed many times
      - e.g., MPI_Iprobe
    - local functions may have disproportionate overheads
      - e.g., MPI_Comm_size
- Scalasca can be configured to provide wrappers for almost all or subsets of MPI routines, from which subgroups can be enabled for measurements via EPK_MPI_ENABLED

Scalasca MPI limitations

- Certain MPI operations are required by the measurement library itself, particularly during initialization & finalization
  - Scalasca measurement is not possible for MPI applications which abort or don't correctly call MPI_Finalize from all ranks
- Scalasca's automatic trace analyzer relies on consistent traces of MPI communication & synchronization events
  - traces don't necessarily have to be complete, provided they are consistent for all MPI processes
- Dynamic process creation (MPI_Spawn) is not supported
- C++ bindings (deprecated by MPI-2.2) are not supported
- Fortran wrappers are based on associated C routines
  - differences in bindings may result in errors during execution or with measurement

OpenMP instrumentation

- In the absence of compiler support for OpenMP events, non-proprietary tools must rely on source instrumentation
  - OPARI parses source files for OpenMP directives/pragmas and API calls and adds corresponding “POMP” instrumentation
    - parallel regions and worksharing constructs are defined with immediate context based on source filename and linenumber(s)
    - instrumentation can be selectively disabled for locks and other synchronizations which can have potentially high overhead
  - OPARI2 uses more invasive processing to avoid problems with parallel, multi-directory and multi-executable builds
    - also can instrument ORDERED constructs and tasks

Scalasca OpenMP limitations

- Every source file containing OpenMP must be instrumented
  - OpenMP used within uninstrumented modules or libraries is at best invisible (and likely to result in measurement corruption)
  - Processing source files (particularly Fortran) is challenging and while OPARI2 is improving it is still sometimes incorrect
- Included source files, conditionally-defined code & macros are not processed by OPARI, which can lead to incomplete or inconsistent instrumentation
  - as a workaround files may be explicitly preprocessed prior to instrumentation (and OPARI processing)
- Nested OpenMP parallel regions are not supported and will result in measurement failures even if nesting is disabled
- Consult Scalasca/OPARI2 OPEN_ISSUES for more

Compiler instrumentation

- Automatic source instrumentation by the compiler [**-comp**]
  - convenient, reliable and supported by most current compilers, though interfaces and implementations differ considerably
    - instrumentation may inhibit or disable compiler optimizations
    - routines may be instrumented before or after in-lining
- By default, Scalasca instruments all routines in source file
  - instrumentation of certain source files or routines may result in excessive overheads (even with measurement filtering)
  - use **-comp=none** to disable compiler instrumentation
  - provide compiler-specific specifications with **-comp=...**
    - refer to compiler manuals for details
    - GCC: -finstrument-functions-exclude-file-list=file1.c,file2.f90 -finstrument-functions-exclude-function-list=substr1,substr2
    - IBM XL: -qfunctrace-func1:routine2:namespace::
    - Intel: -tcollect-filter=<file.txt>

PDT-based instrumentation

- Automatic source instrumentation using PDToolkit [**-pdt**]
  - only available if configured when Scalasca installed
- By default, instruments all routines in source file
  - source routines are automatically instrumented by compiler, therefore use **-comp=none** to avoid duplicate instrumentation
- Selective instrumentation of specified routines
  - -optTauSelectFile=<filename>
  - TAU-format plain text specification file
    - list names of source files and routines to include/exclude from instrumentation, using wildcard patterns
      - unsupported TAU instrumentation features are silently ignored
    - refer to TAU/PDToolkit documentation for details
    - refer to Scalasca User Guide for known limitations

PDT-based selective instrumentation file

- List routines with their PDT names one per line

```
% cat config/pdt.inst
# instrumentation specification for PDT
BEGIN_EXCLUDE_LIST
    BINVCRHS
    MATVEC_SUB
    MATMUL_SUB
    BINVRHS
    EXACT_SOLUTION
    LHS#INIT
    TIMER_#
END_EXCLUDE_LIST
```

- … and specify file when instrumenting

```
% make bt-mz CLASS=W NPROCS=4  PREP="scalasca -inst -comp=none -pdt \
                               -optTauSelectFile=$PWD/config/pdt.inst"
```

Fixed-format Fortran limitations

- PDT and EPIK user instrumentation macros expand to additional statements in program source files
  - this should be unproblematic, except for fixed-format Fortran where the default line-length limit (72 characters) may be exceeded and result in curious compilation errors
  - Fortran compilers allow extended source lines via special compile flags, e.g.,
    - CCE: `-N132`
    - GNU: `-ffixed-line-length-none`
    - Intel/Pathscale: `-extend-source`
    - PGI: `-Mextend`
  - For BT example therefore need to adjust FFLAGS

```
% make bt-mz CLASS=W NPROCS=4  PREP="scalasca -inst -comp=none -pdt" \
                              FFLAGS="-O3 -ffixed-line-length-none"
```

EPIK user instrumentation API

- EPIK user instrumentation API
  - #include “epik_user.h”
  - EPIK_USER_REG(epik_solve, “<<Solve>>”)
  - EPIK_USER_START(epik_solve)
  - EPIK_USER_END(epik_solve)
- Can be used to mark initialization, solver & other phases
  - Annotation macros ignored by default
  - Instrumentation enabled with “**-user**” flag to instrumenter
  - Also available for Fortran
    - #include “epik_user.inc” and use C preprocessor
- Appear as additional regions in analyses
  - Distinguishes performance of important phase from rest

EPIK user instrumentation: F77 example

- In NPB3.3-MZ-MPI/BT-MZ compare bt.f & bt_epik.F
  - the .F suffix indicates that it should be preprocessed
    - otherwise could specify some obscure compiler flags
- EPIK user API #include'd at the top
  - #include "epik_user.inc"
- EPIK user instrumentation macros register & mark phases "<<INIT>>", "<<STEP>>", "<<FINI>>"
- within the main routine "<<MAIN>>"
- Edit `BT-MZ/makefile` to set: `MAIN = bt_epik.F`
- Instrument specifying `-user` and extended source lines

```
% make bt-mz CLASS=W NPROCS=4  PREP="scalasca -inst -comp=none -user" \
                              FFLAGS="-O3 -ffixed-line-length-none"
```

EPIK user control API

- Specifies when measurement should be paused
  - events are not included within call-path summary nor trace
  - reduces measurement overhead and size of data collected
- Can be used for undesired execution phases (e.g., initialization) or particular subsets of iterations
- Pausing regions defined like other user regions
  - EPIK_PAUSE_START()
  - EPIK_PAUSE_END()
- Shown in summary and trace analyses as special region “<<PAUSING>>”
- Pause regions are not synchronized between MPI ranks and are ignored within OpenMP parallel regions
- Nesting of pause regions is not supported

scan – Scalasca collection/analysis nexus

- Runs application under control of measurement system to collect and analyze an execution experiment
  - ***scan*** = **scalasca -analyze**
  - scan [options] <*application-launch-command*>
    - e.g., scan [options] [$MPIEXEC [mpiexec-options]] [target [args]]
  - **[-s]** collect summarization experiment [default]
  - **[-t]** collect event traces and then analyze them automatically
  - Additional options
    - [-e title] specify experiment archive (directory) name: epik_*title*
    - [-f filter] specify file listing routines to ignore during measurement
    - [-m metric1:metric2:...] include hardware counter metrics
    - [-n] preview scan and perform checks but don't execute
    - [-q] quiesce (disable most) measurement collection
    - [-a] (re-)analyze a previously collected trace experiment

EPIK measurement configuration

- Via ./EPIK.CONF file

```
EPK_FILTER=smg2000.filt
EPK_MPI_ENABLED=CG:COLL:ENV:IO:P2P:RMA:TOPO
ELG_BUFFER_SIZE=40000000
```

- Via environment variables

```
% export EPK_FILTER=smg2000.filt
% export EPK_MPI_ENABLED=CG:COLL:ENV:IO:P2P:RMA:TOPO
% export ELG_BUFFER_SIZE=40000000
```

- Via command-line flags (partially)

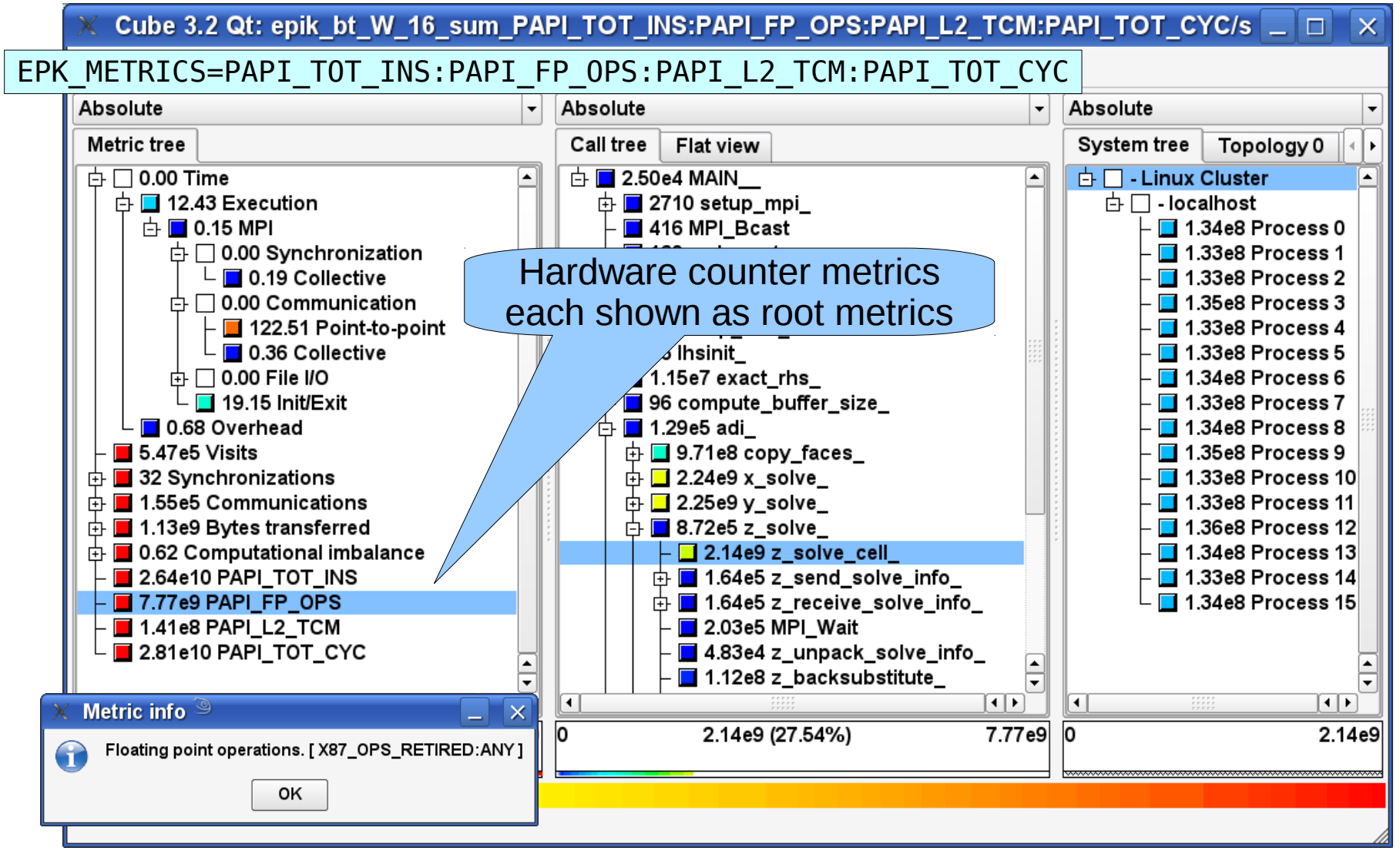
```
% scalasca -analyze -f smg2000.filt ...
```

- To show current/default configuration

```
% epik_conf
```

- Actual Scalasca measurement configuration saved in experiment archive as `epik.conf`

Summary analysis report with HWC metrics



Groups of counter metrics

- Specified as a colon-separated list in EPK_METRICS or on command-line

```
EPK_METRICS=PAPI_TOT_INS:PAPI_FP_OPS:PAPI_L2_TCM:PAPI_TOT_CYC

scan -m PAPI_TOT_INS:PAPI_FP_OPS:PAPI_L2_TCM:PAPI_TOT_CYC ...
```

- or use a predefined group from EPK_METRICS_SPEC

```
EPK_METRICS=ITANIUM2_TLB
```

- or use another file specifying groups

```
EPK_METRICS_SPEC=/opt/scalasca/example/METRICS.SPEC_POWER6
EPK_METRICS=PM_BRANCH2
```

- or define your own groups in a custom specification file

```
# Opteron groupings defined by CrayPAT
aggroup PAT_RT_HWPC_0 = PAPI_L1_DCM PAPI_L1_DCA PAPI_TOT_INS PAPI_FP_OPS
aggroup PAT_RT_HWPC_1 = PAPI_L1_DCM PAPI_L1_DCA PAPI_TLB_DM PAPI_FP_OPS
aggroup PAT_RT_HWPC_6 = PAPI_FPU_IDL PAPI_RES_STL PAPI_STL_ICY INSTRUCTION_FETCH_STALL
aggroup PAT_RT_HWPC_7 = DECODER_EMPTY DISPATCH_STALL_FOR_FPU_FULL DISPATCH_STALLS ...
aggroup PAT_RT_HWPC_10 = PAPI_L2_DCM PAPI_L1_DCA L3_CACHE_MISSES PAPI_L2_DCA
```

Hints for the Scalasca analysis nexus

- Generally, the SCAN nexus will correctly parse execution command lines, but occasionally you may need to help it
- MPI launcher arguments may need to be explicitly separated from the target application with a double-dash

```
% scalasca -analyze  mpirun -np 16  --  a.exe arg
```

- Unusual MPI launcher options may need to be quoted

```
% scalasca -analyze  mpirun -np 16  "-verbose 2"  a.exe arg
```

  - (On most systems `-verbose` doesn't take an argument)
- Explicitly specify the instrumented target executable name when using imposition commands/scripts

```
% export SCAN_TARGET=a.exe
% scalasca -analyze  imposter.sh i.arg  a.exe arg
% scan -t  mpirun -np 16  imposter.sh i.arg  a.exe arg
```

  - (*dplace*, *omplace* and *numactl* are common imposters)

SCAN configuration variables

- SCAN_ANALYZE_OPTS specifies trace analyzer options:
  - '-i' enables determination of pattern instance statistics including the worst instance
  - '-s' enables correction of timestamps from compute nodes with insufficiently synchronized clocks
- SCAN_CLEAN specifies whether to remove trace data after successful automatic trace analysis [0]
- SCAN_TRACE_ANALYZER specifies an alternative trace analyzer, or 'none' to skip automatic trace analysis
- SCAN_WAIT specifies max number of seconds to wait after measurement completion for synchronization of the filesystems on compute nodes with the launch node [0]

SCAN limitations

- Multi-executable (MPMD) executions may work fine, however, there are known limitations and Scalasca may need some assistance
  - all executables need to have been instrumented by Scalasca
    - if any are MPI+OMP, then the first executable also needs to be linked in MPI+OMP mode
  - the number of MPI processes may need to be explicitly specified via SCAN_MPI_RANKS
- MPI launcher specifications in a configuration file are not parsed (even when it is specified on the command line)

square – Scalasca analysis report examiner

- Prepares and presents measurement analysis report(s) for scoring and/or interactive exploration
  - ***square*** = **scalasca -examine**
  - square [options] <*experiment-archive*|*report*>
    - e.g., square epik_*title*
  - Post-processes intermediate measurement analysis reports
  - Launches GUI and presents default analysis report (if multiple reports available)
    - trace analysis given precedence over summary analysis
    - select other reports via File/Open menu
  - [-s] skip display and produce textual score report (epik.score)
    - estimates total trace size and maximum rank trace size
    - breakdown of USR vs. MPI/OMP vs. COM region requirements
    - add [-f test.filt] to test effect of a prospective filter file

CUBE algebra utilities

- Extracting a sub-tree from an analysis report

```
% cube3_cut  -r 'adi_'  epik_bt-mz_4x4_sum/summary.cube
Writing cut.cube... done.
```

- Calculating difference of two analysis reports

```
% cube3_diff  epik_bt_9_trace/trace.cube  epik_bt_16_trace/trace.cube
Writing diff.cube... done.
```

- Combining two or more related analysis reports

```
% cube3_merge  trace/trace.cube  HWC1/summary.cube  HWC2/summary.cube
Writing merge.cube... done.
```

- Additional algebra utilities for calculating mean, etc.
  - Default output of cube3_*utility* is a new report *utility*.cube
- Further utilities for report scoring & statistics
- Run utility with “-h” (or no arguments) for brief usage info

Tip: Merging multiple HWC analysis reports

- Example set of experiments collected with and w/o HWC

```
% ls -1d epik_*
epik_bt_B_16_sum_PAT_RT_HWPC_0/
epik_bt_B_16_sum_PAT_RT_HWPC_1/
epik_bt_B_16_sum_PAT_RT_HWPC_7/
epik_bt_B_16_sum_PAT_RT_HWPC_8/
epik_bt_B_16_trace/
```

- Ensure that each is post-processed

```
% for epik in epik_* ; do scalasca -examine -s $epik ; done
```

- Merge the HWC summary reports into the non-HWC report

```
% cube3_merge -o HWC_combo.cube \
        epik_bt_B_16_trace/trace.cube epik_bt_B_16_sum_*/summary.cube
Writing HWC_combo.cube... done.
```

- Metrics are merged as they are encountered in reports
  - already defined metrics are not modified by later versions
- Since measurements with HWCs have higher overhead, include a non-HWC measurement first

DON'T PANIC!

- Remember the Scalasca User Guide is your friend
- On the assumption that nothing terrible is going to happen, all the advice in this tutorial may be safely ignored
- But if you need more advice, mailto:scalasca@fz-juelich.de