



Automatic trace analysis with Scalasca

Alan O’Cais

Jülich Supercomputing Centre

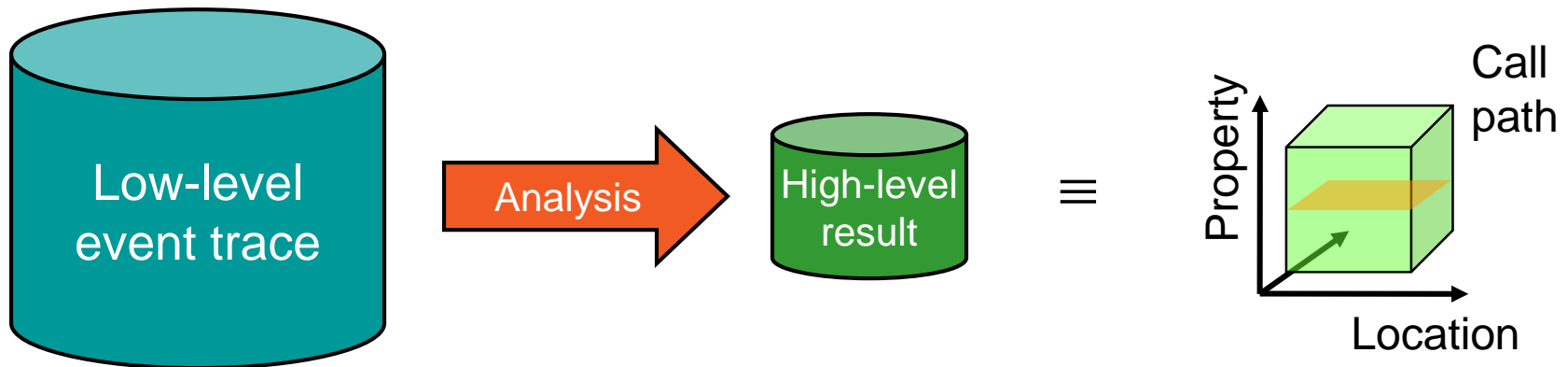
(with content from tutorials by Markus Geimer,
Brian Wylie, David Böhme /JSC)

Live notes:

<http://supercomputing.cyi.ac.cy/index.php/live>



- Idea
 - Automatic search for patterns of inefficient behavior
 - Classification of behavior & quantification of significance



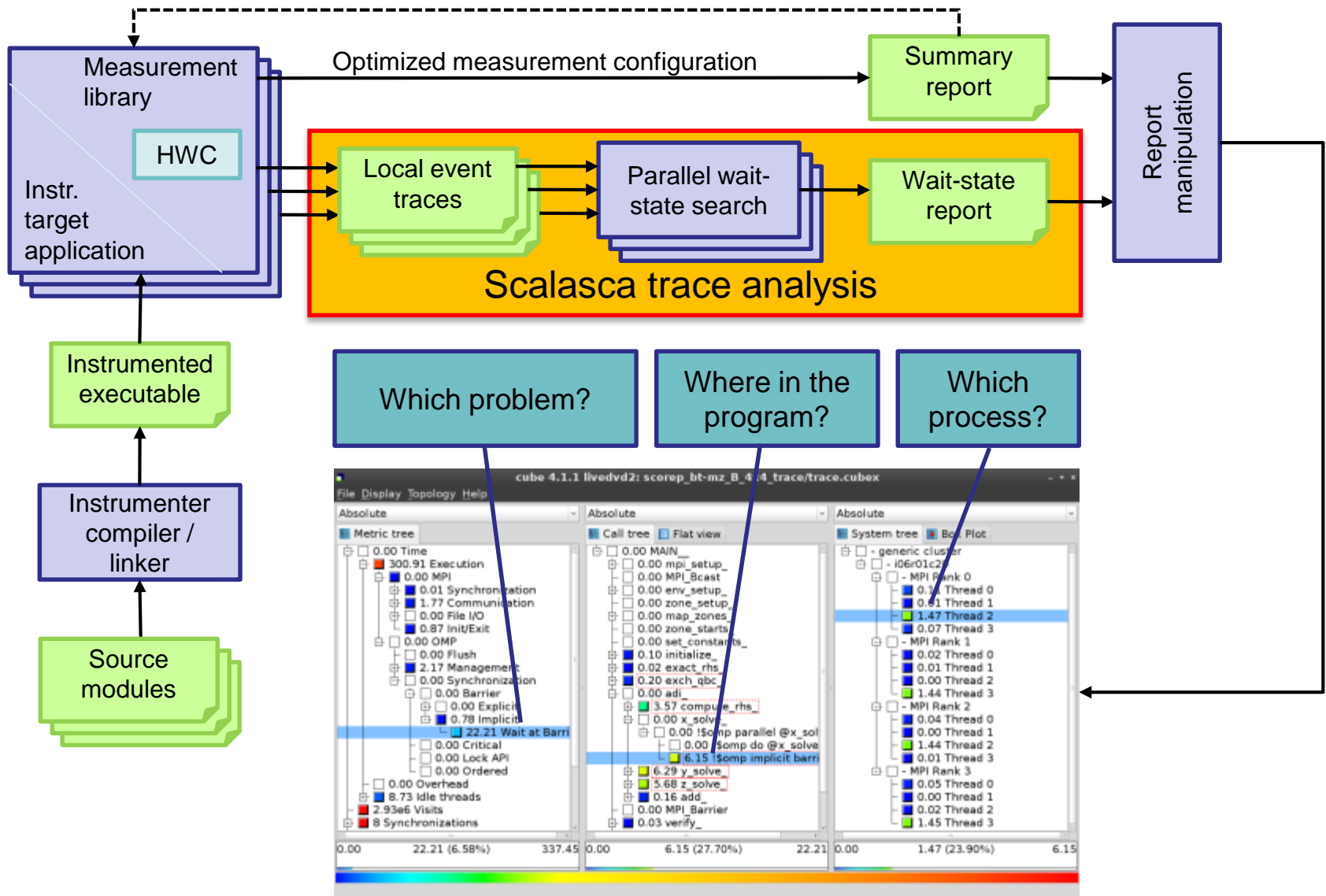
- Guaranteed to cover the entire event trace
- Quicker than manual/visual trace analysis
- Parallel replay analysis exploits available memory & processors to deliver scalability

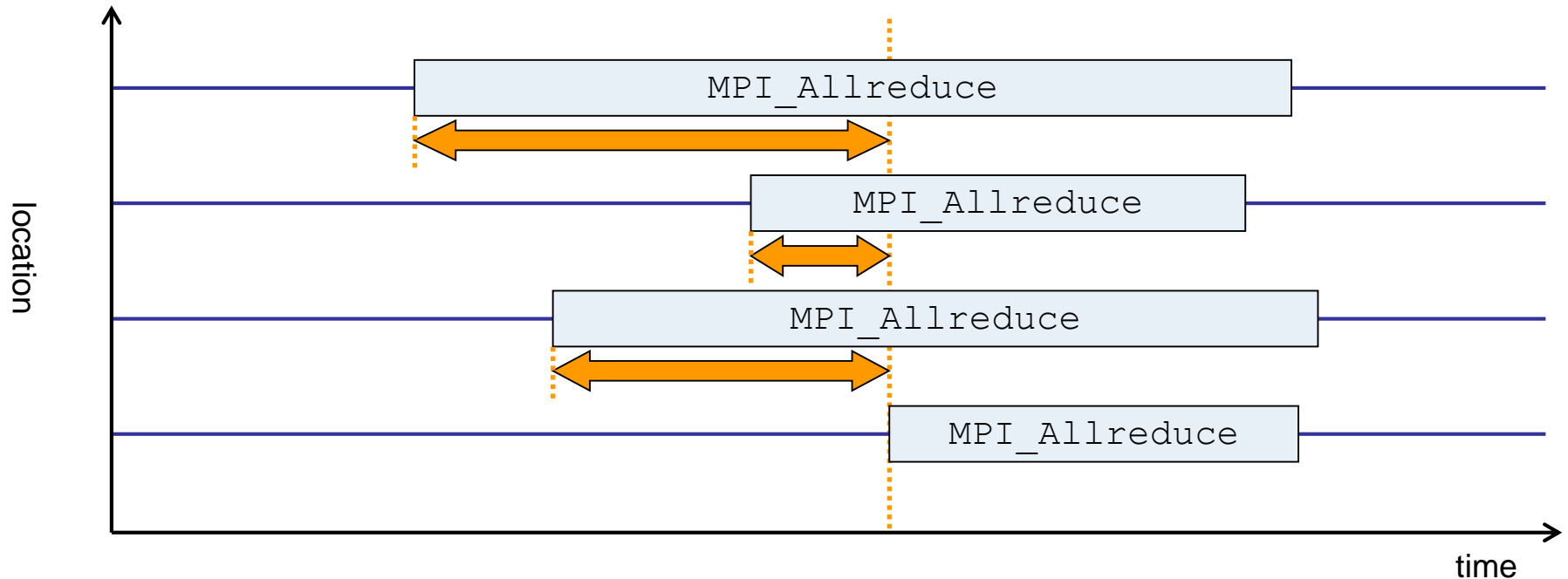
- Project started in 2006
 - Initial funding by Helmholtz Initiative & Networking Fund
 - Many follow-up projects
- Follow-up to pioneering KOJAK project (started 1998)
 - Automatic pattern-based trace analysis
- Now joint development of
 - Jülich Supercomputing Centre
 - German Research School for Simulation Sciences



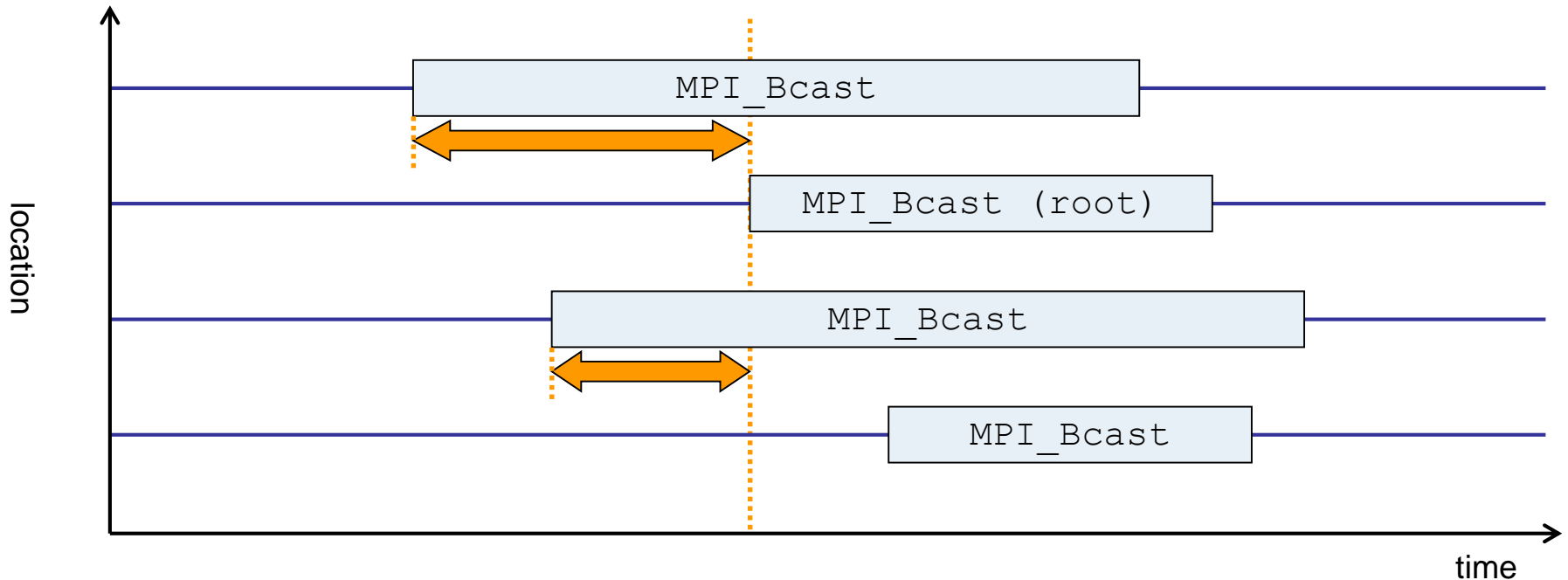
- Development of a **scalable** performance analysis toolset for most popular parallel programming paradigms
- Specifically targeting **large-scale** parallel applications
 - such as those running on IBM BlueGene or Cray XT systems with one million or more processes/threads
- Latest release:
 - Scalasca v2.1 with Score-P support

- Open source, New BSD license
- Fairly portable
 - IBM Blue Gene, IBM SP & blade clusters, Cray XT, SGI Altix, Solaris & Linux clusters, ...
- Uses Score-P instrumenter & measurement libraries
 - Scalasca 2.1 core package focuses on trace-based analyses
 - Supports common data formats
 - Reads event traces in OTF2 format
 - Writes analysis reports in CUBE4 format
- Current limitations:
 - No support for nested OpenMP parallelism and tasking
 - Unable to handle OTF2 traces containing CUDA events

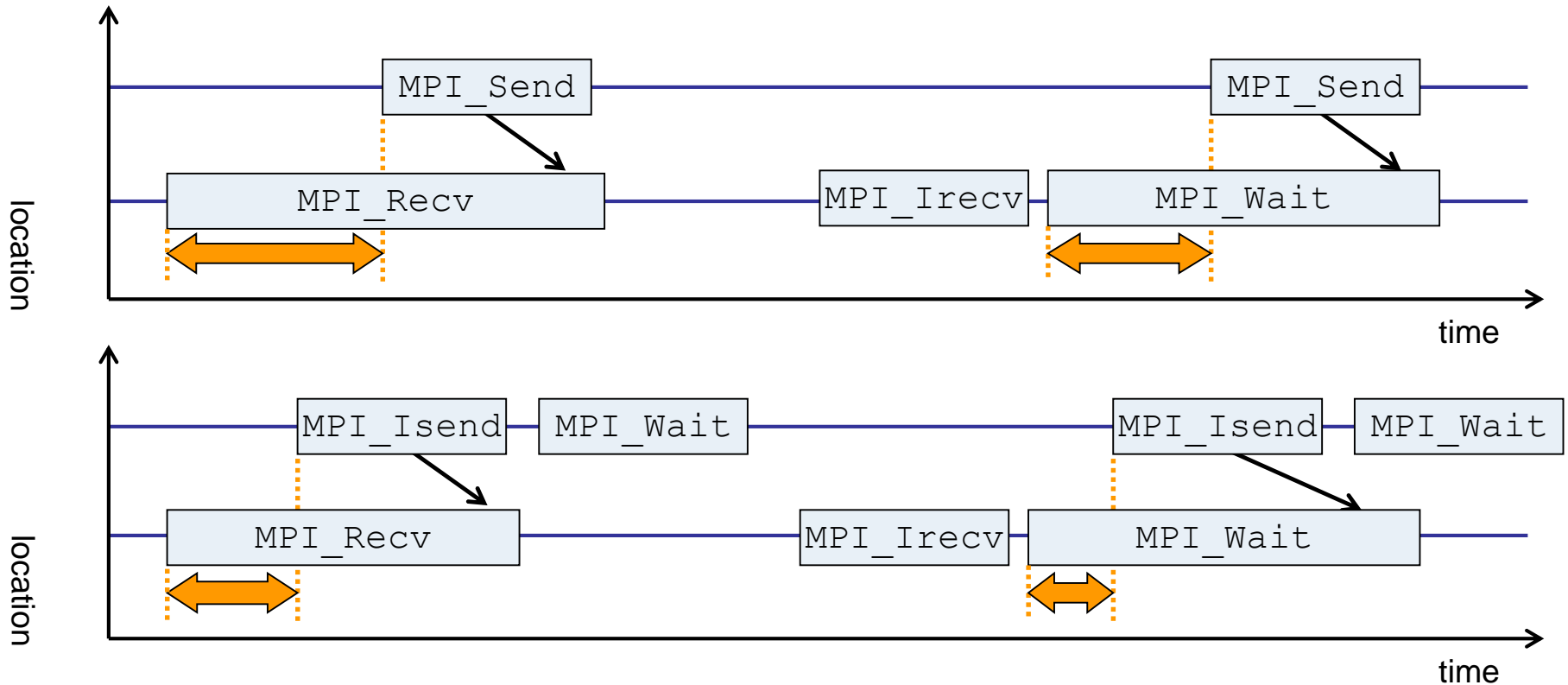




- Time spent waiting in front of synchronizing collective operation until the last process reaches the operation
- Applies to: MPI_Allgather, MPI_Allgatherv, MPI_Alltoall, MPI_Reduce_scatter, MPI_Reduce_scatter_block, MPI_Allreduce



- Waiting times if the destination processes of a collective 1-to-N operation enter the operation earlier than the source process (root)
- Applies to: MPI_Bcast, MPI_Scatter, MPI_Scatterv



- Waiting time caused by a blocking receive operation posted earlier than the corresponding send
- Applies to blocking as well as non-blocking communication

VI-HPS



Hands-on exercise (EUCLID): NPB-MZ-MPI / BT

VI-HPS Team

- Connect to EUCLID using trusted X11 forwarding

```
% ssh -YC euclid.cyi.ac.cy
```

- Check/modify modules for MPI & compilers

```
% module list
Currently loaded modules:
[...]
% module purge
% module avail
[...]
% module load Score-P Scalasca
```

- Copy tutorial sources to your work directory

```
% cp -r /tmp/PRACE_AutumnWorkshop_2014 .
% cd ./PRACE_AutumnWorkshop_2014/Scalasca/NPB3.3-MZ-MPI
```

- (When available, generally advisable to use a parallel filesystem such as \$WORK)

- The NAS Parallel Benchmark suite (MPI+OpenMP version)
 - Available from
<http://www.nas.nasa.gov/Software/NPB>
 - 3 benchmarks in Fortran77
 - Configurable for various sizes & classes
- Move into the NPB3.3-MZ-MPI root directory

```
% .../NPB3.3-MZ-MPI; ls
bin/      common/  jobscript/  Makefile  README.install  SP-MZ/
BT-MZ/    config/  LU-MZ/      README    README.tutorial  sys/
```

- Subdirectories contain source code for each benchmark
 - plus additional configuration and common code
- The provided distribution has already been configured for the tutorial, such that it's ready to “make” one or more of the benchmarks and install them into a (tool-specific) “bin” subdirectory

- Type “make” for instructions

```
% make
=====
=      NAS PARALLEL BENCHMARKS 3.3      =
=      MPI+OpenMP Multi-Zone Versions   =
=      F77                                =
=====

To make a NAS multi-zone benchmark type

    make <benchmark-name> CLASS=<class> NPROCS=<nprocs>

where <benchmark-name> is "bt-mz", "lu-mz", or "sp-mz"
     <class>           is "S", "W", "A" through "F"
     <nprocs>         is number of processes

[...]
```

```
*****
* Custom build configuration is specified in config/make.def *
* Suggested tutorial exercise configuration for HPC systems: *
*      make bt-mz CLASS=B NPROCS=4                          *
*****
```

- Specify the benchmark configuration
 - benchmark name: **bt-mz**, lu-mz, sp-mz
 - the number of MPI processes: **NPROCS=4**
 - the benchmark class (S, W, A, B, C, D, E): **CLASS=B**

```
% make bt-mz CLASS=B NPROCS=4
cd BT-MZ; make CLASS=B NPROCS=4 VERSION=
make: Entering directory 'BT-MZ'
cd ../sys; cc -o setparams setparams.c
../sys/setparams bt-mz 4 B
mpif77 -c -O3 -openmp bt.f
[...]
cd ../common; mpif77 -c -O3 -openmp timers.f
mpif77 -O3 -openmp -o ../bin/bt-mz_B.4 \
bt.o initialize.o exact_solution.o exact_rhs.o set_constants.o \
adi.o rhs.o zone_setup.o x_solve.o y_solve.o exch_qbc.o \
solve_subs.o z_solve.o add.o error.o verify.o mpi_setup.o \
../common/print_results.o ../common/timers.o
Built executable ../bin/bt-mz_B.4
make: Leaving directory 'BT-MZ'
```

Hint: for default configuration:

```
% make suite
```

- What does it do?
 - Solves a discretized version of unsteady, compressible Navier-Stokes equations in three spatial dimensions
 - Performs 200 time-steps on a regular 3-dimensional grid
- Implemented in 20 or so Fortran77 source modules
- Uses MPI & OpenMP in combination
 - 2 processes with 4 threads each should be reasonable for execution on a single compute node
 - bt-mz_B.4 should run in around 50 seconds
 - bt-mz_C.4 should take around 3-4x longer

- Copy jobscript and launch as a hybrid MPI+OpenMP application

```
% cd bin
% cp ../jobscript/euclid/run.slurm .
% less run.slurm
% sbatch run.slurm
% cat job.<id>.out
NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark
Number of zones:      8 x      8
Iterations:  200      dt:  0.000300
Number of active processes:      4
Total number of threads:      16  (  4.0 threads/process)

Time step      1
Time step     20
  [...]
Time step    180
Time step    200
Verification Successful

BT-MZ Benchmark Completed.
Time in seconds = 28.74
```

Hint: save the benchmark output (or note the run time) to be able to refer to it later

- VI-HPS tools accessible through the EasyBuild framework
 - May have multiple versions and configurations
 - Limited set (for workshop) installed on Euclid, this will change!

```
% module avail  
[...]  
periscope/1.5  
scalasca/1.4.3-parastation-intel-papi-sion (default)  
scalasca/2.1-parastation-gnu  
scalasca/2.1-parastation-intel  
scorep/1.2.1-parastation-gnu-papi  
scorep/1.2.1-parastation-intel-papi (default)  
tau/2.22.2-parastation-intel-papi  
vampir/8.1.0 (default)  
vampirserver/8.1.0 (default)  
[...]
```

- Edit `config/make.def` to adjust build configuration
 - Modify specification of compiler/linker: `MPIF77`
- Make clean and build new tool-specific executable

```
% make clean
% make bt-mz CLASS=B NPROCS=4
Built executable ../bin.%(TOOL)/bt-mz_B.4
```

- Change to the directory containing the new executable before running it with the desired tool configuration

```
% cd bin.%(TOOL)
% export ...
% OMP_NUM_THREADS=4 mpirun -np 4 ./bt-mz_B.4
```

```
#          SITE- AND/OR PLATFORM-SPECIFIC DEFINITIONS
#-----
# Items in this file may need to be changed for each platform.
#-----
...
#-----
# The Fortran compiler used for MPI programs
#-----
MPIF77 = mpif77

# Alternative variants to perform instrumentation
#MPIF77 = psc_instrument -u user,mpi,omp -s ${PROGRAM}.sir mpif77
#MPIF77 = tau_f90.sh
#MPIF77 = scalasca -instrument mpif77
#MPIF77 = vtf77 -vt:hyb -vt:f77 mpif77
#MPIF77 = scorep --user mpif77

# PREP is a generic preposition macro for instrumentation preparation
#MPIF77 = $(PREP) mpif77

# This links MPI Fortran programs; usually the same as ${MPIF77}
FLINK    = $(MPIF77)
...

```

Default (no instrumentation)

Hint: uncomment one of these alternative compiler wrappers to perform instrumentation

... or use this generic variant

VI-HPS



Hands-on: NPB-MZ-MPI / BT

scalasca 

- One command for (almost) everything...

```
% scalasca
Scalasca 2.0
Toolset for scalable performance analysis of large-scale applications
usage: scalasca [-v][-n][c] {action}
  1. prepare application objects and executable for measurement:
    scalasca -instrument <compile-or-link-command> # skin (using scorep)
  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, --verbose          enable verbose commentary
-n, --dry-run         show actions without taking them
-c, --show-config     show configuration and exit
```

- The ‘scalasca -instrument’ command is deprecated and only provided for backwards compatibility with Scalasca 1.x.
- Recommended: use Score-P instrumenter directly

- Scalasca application instrumenter

```
% skin
Scalasca 2.0: application instrumenter using scorep
usage: skin [-v] [-comp] [-pdt] [-pomp] [-user] <compile-or-link-cmd>
  -comp={all|none|...}: routines to be instrumented by compiler
                        (... custom instrumentation specification for compiler)
  -pdt:  process source files with PDT instrumenter
  -pomp: process source files for POMP directives
  -user: enable EPIK user instrumentation API macros in source code
  -v:    enable verbose commentary when instrumenting

  --*:   options to pass to Score-P instrumenter
```

- Provides compatibility with Scalasca 1.x
- Recommended: use Score-P instrumenter directly

- Scalasca measurement collection & analysis nexus

```
% scan
Scalasca 2.0: measurement collection & analysis nexus
usage: scan {options} [launchcmd [launchargs]] target [targetargs]
      where {options} may include:
  -h      Help: show this brief usage message and exit.
  -v      Verbose: increase verbosity.
  -n      Preview: show command(s) to be launched but don't execute.
  -q      Quiescent: execution with neither summarization nor tracing.
  -s      Summary: enable runtime summarization. [Default]
  -t      Tracing: enable trace collection and analysis.
  -a      Analyze: skip measurement to (re-)analyze an existing trace.
  -e exptdir   : Experiment archive to generate and/or analyze.
                  (overrides default experiment archive title)
  -f filtdir  : File specifying measurement filter.
  -l lockfile  : File that blocks start of measurement.
```

- Scalasca analysis report explorer

```
% square  
Scalasca 2.1: analysis report explorer  
usage: square [-v] [-s] [-f filtfiler] [-F] <experiment archive  
           | cube file>  
-F           : Force remapping of already existing reports  
-f filtfiler : Use specified filter file when doing scoring  
-s           : Skip display and output textual score report  
-v           : Enable verbose mode
```


- **scan** configures Score-P measurement by setting some environment variables automatically
 - e.g., experiment title, profiling/tracing mode, filter file, ...
 - Precedence order:
 - Command-line arguments
 - Environment variables already set
 - Automatically determined values
- Also, **scan** includes consistency checks and prevents corrupting existing experiment directories
- For tracing experiments, after trace collection completes then automatic parallel trace analysis is initiated
 - uses identical launch configuration to that used for measurement (i.e., the same allocated compute resources)

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

```
% export SCOREP_EXPERIMENT_DIRECTORY=scorep_bt-mz_W_4x4_sum
% OMP_NUM_THREADS=4 scan mpirun -np 4 ./bt-mz_W.4
S=C=A=N: Scalasca 2.0 runtime summarization
S=C=A=N: ./scorep_bt-mz_W_4x4_sum experiment archive
S=C=A=N: Thu Sep 13 18:05:17 2012: Collect start
mpixec -np 4 ./bt-mz_W.4

NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark

Number of zones:      8 x      8
Iterations: 200      dt: 0.000300
Number of active processes:      4

[... More application output ...]

S=C=A=N: Thu Sep 13 18:05:39 2012: Collect done (status=0) 22s
S=C=A=N: ./scorep_bt-mz_W_4x4_sum complete.
```

- Creates experiment directory `./scorep_bt-mz_W_4x4_sum`

- Score summary analysis report

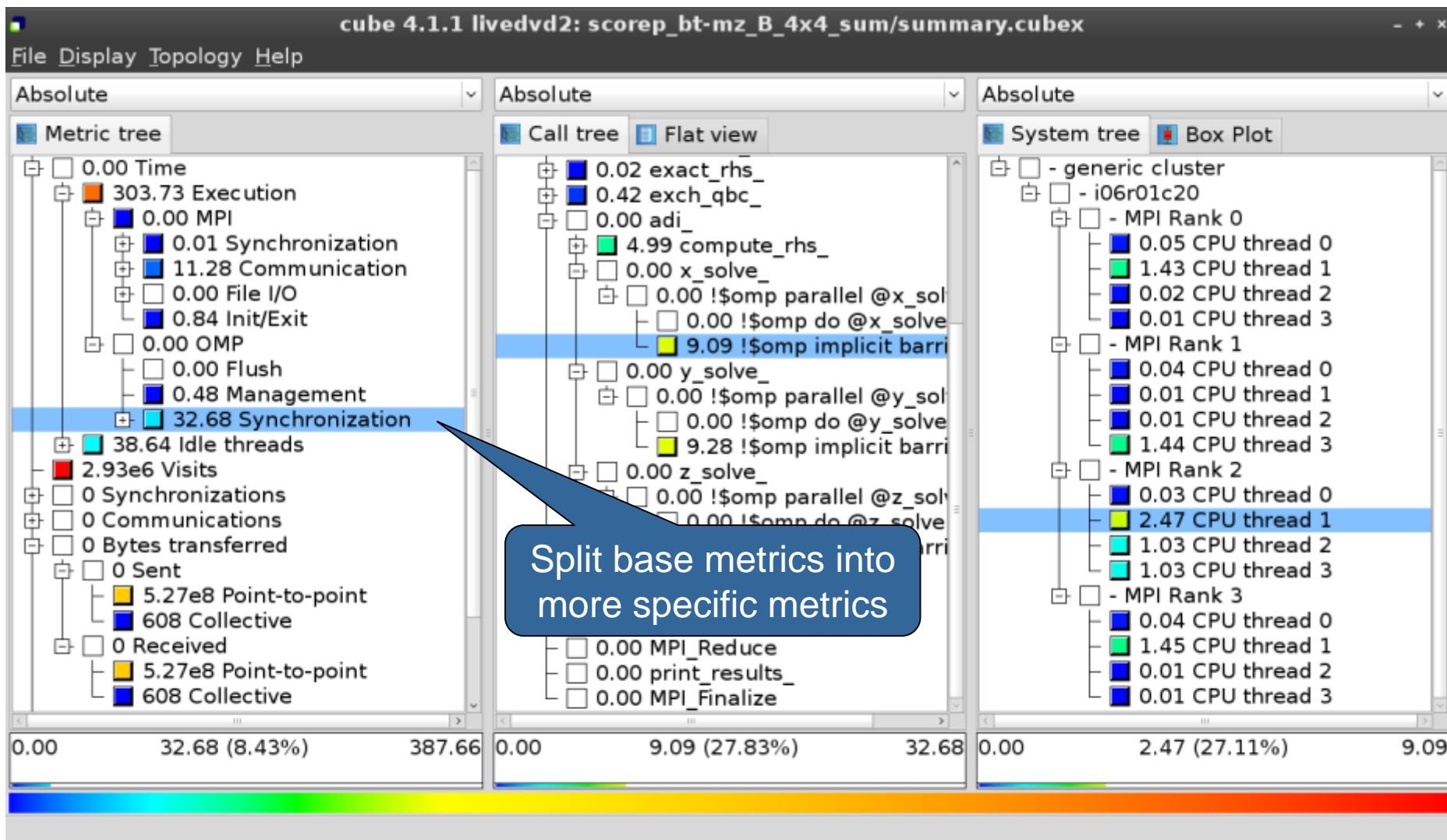
```
% square -s scorep_bt-mz_W_4x4_sum  
INFO: Post-processing runtime summarization result...  
INFO: Score report written to ./scorep_bt-mz_W_4x4_sum/scorep.score
```

- Post-processing and interactive exploration with CUBE

```
% square scorep_bt-mz_W_4x4_sum  
INFO: Displaying ./scorep_bt-mz_W_4x4_sum/summary.cubex...  
  
[GUI showing summary analysis report]
```

- The post-processing derives additional metrics and generates a structured metric hierarchy

Post-processed summary analysis report



0.0 Reference preparation for validation

1.0 Program instrumentation

1.1 Summary measurement collection

1.2 Summary analysis report examination

2.0 Summary experiment scoring

2.1 Summary measurement collection with filtering

2.2 Filtered summary analysis report examination

3.0 Event trace collection

3.1 Event trace examination & analysis

- Load modules

```
% module load Score-P/1.3-ictce-5.3.0  
Score-P/1.2.3-ictce-5.3.0 loaded  
% module load Scalasca/2.1-ictce-5.3.0  
Scalasca/2.1-ictce-5.3.0 loaded  
% module load Cube  
cube/4.3-ictce-5.3.0 loaded
```

- Change to directory containing NPB BT-MZ sources
- Existing instrumented binary in bin.scorep/ can be reused

- Change to executable directory and edit job script

```
% cd bin.scorep
% cp ../jobscript/euclid/run_scorep_with_filter_tracing.qsub .
      % vim scalasca2.0

[...]

module load ...

export SCOREP_FILTERING_FILE=../config/scorep.filt
export SCOREP_TOTAL_MEMORY=50M
export SCOREP_METRIC_PAPI=PAPI_FP_OPS

scan mpirun -np $PROCS $EXE
```

- Submit the job

```
% sbatch run_scorep_with_filter_tracing.slurm
```

- Continues with automatic (parallel) analysis of trace files

```
S=C=A=N: Fri Sep 20 15:09:59 2013: Analyze start

Analyzing experiment archive ./scorep_bt-mz_C_2p64x8_trace/traces.otf2

Opening experiment archive ... done (0.019s).
Reading definition data ... done (0.178s).
Reading event trace data ... done (2.068s).
Preprocessing ... done (3.789s).
Analyzing trace data ...
  Wait-state detection (fwd) (1/5) ... done (2.889s).
  Wait-state detection (bwd) (2/5) ... done (1.136s).
  Synchpoint exchange (fws) (3/5) ... done (0.813s).
  Critical-path & delay analysis (4/5) ... done (0.568s).
done (5.413s).
Writing analysis report ... done (1.994s).

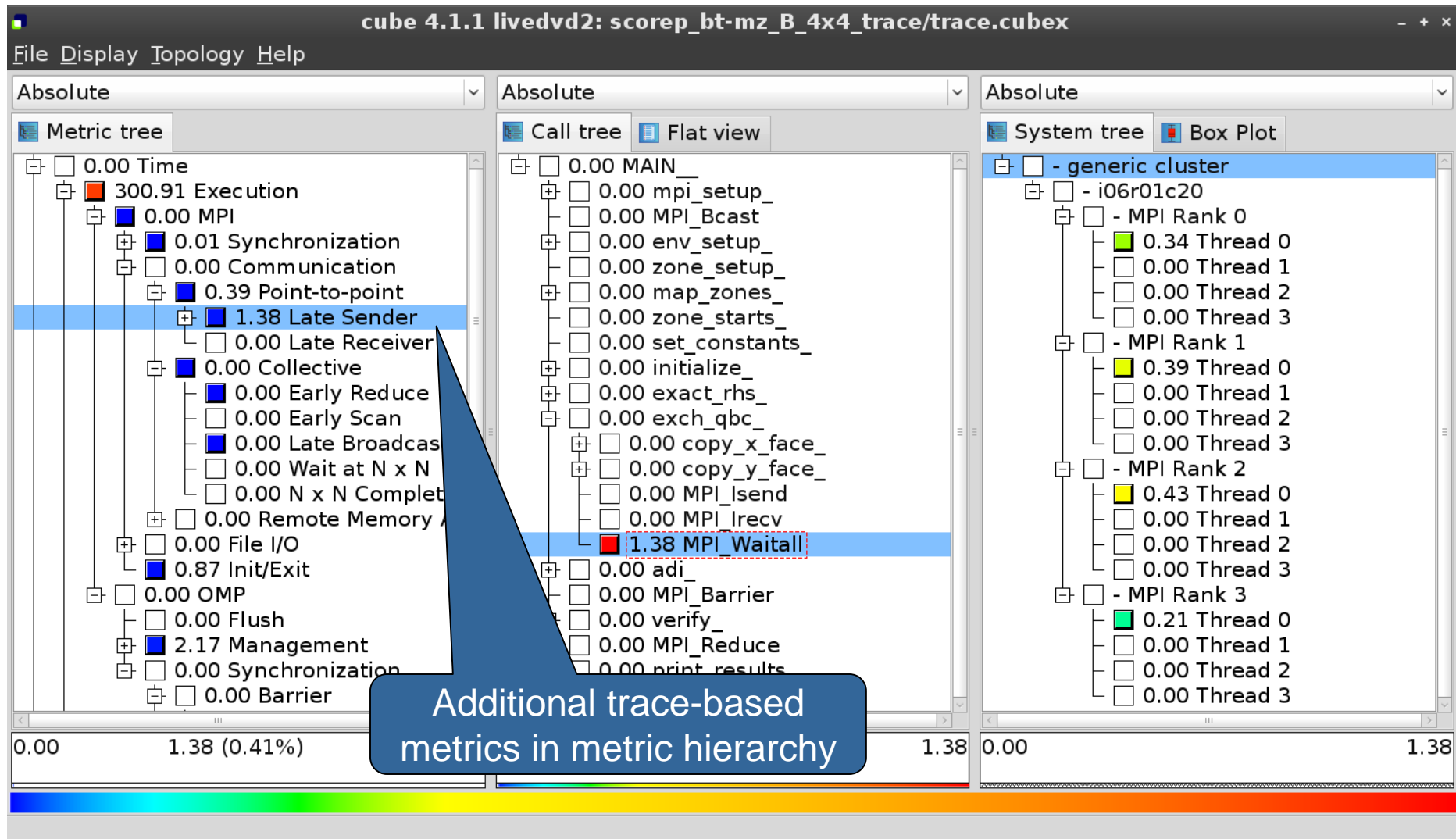
Max. memory usage : 181.066MB

Total processing time : 13.645s
S=C=A=N: Fri Sep 20 15:10:16 2013: Analyze done (status=0) 17s
```


- Produces trace analysis report in experiment directory containing trace-based wait-state metrics

```
% square scorep_bt-mz_B_32x8_trace  
INFO: Post-processing runtime summarization result...  
INFO: Post-processing trace analysis report...  
INFO: Displaying ./scorep_bt-mz_C_32x8_trace/trace.cubex...  
  
[GUI showing trace analysis report]
```

Post-processed trace analysis report



The screenshot displays the 'cube 4.1.1 livedvd2: scorep_bt-mz_B_4x4_trace/trace.cubex' application. It features three main panels:

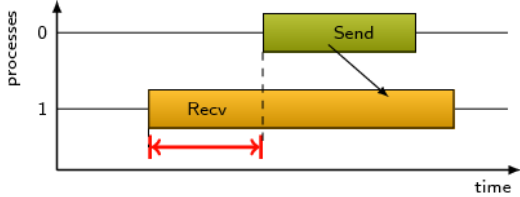
- Metric tree (Absolute):** Shows a hierarchical view of metrics. The 'Late Sender' metric (1.38) is selected, and a context menu is open over it. The menu options include: Info, Full info, **Online description**, Expand/collapse, Find items, Find Next, Clear found items, Copy to clipboard, Create derived metric..., Remove metric..., Statistics, and Max severity in trace browser.
- Call tree (Absolute):** Shows a call tree view with 'Flat view' selected. The root is 'MAIN__' (0.00), with sub-items like 'mpi_setup_', 'MPI_Bcast', 'env_setup_', 'zone_setup_', 'map_zones_', and 'zone_starts_'.
- System tree (Absolute):** Shows a system tree view with 'Box Plot' selected. The root is '- generic cluster', with sub-items for MPI Ranks 0, 1, 2, and 3, each containing Thread 0, 1, 2, and 3.

A blue callout box at the bottom right contains the text: "Access online metric description via context menu".

Performance properties

Late Sender Time

Description:
Refers to the time lost waiting caused by a blocking receive operation (e.g., `MPI_Recv` or `MPI_Wait`) that is posted earlier than the corresponding send operation.



If the receiving process is waiting for multiple messages to arrive (e.g., in an call to `MPI_Waitall`), the maximum waiting time is accounted, i.e., the waiting time due to the latest sender.

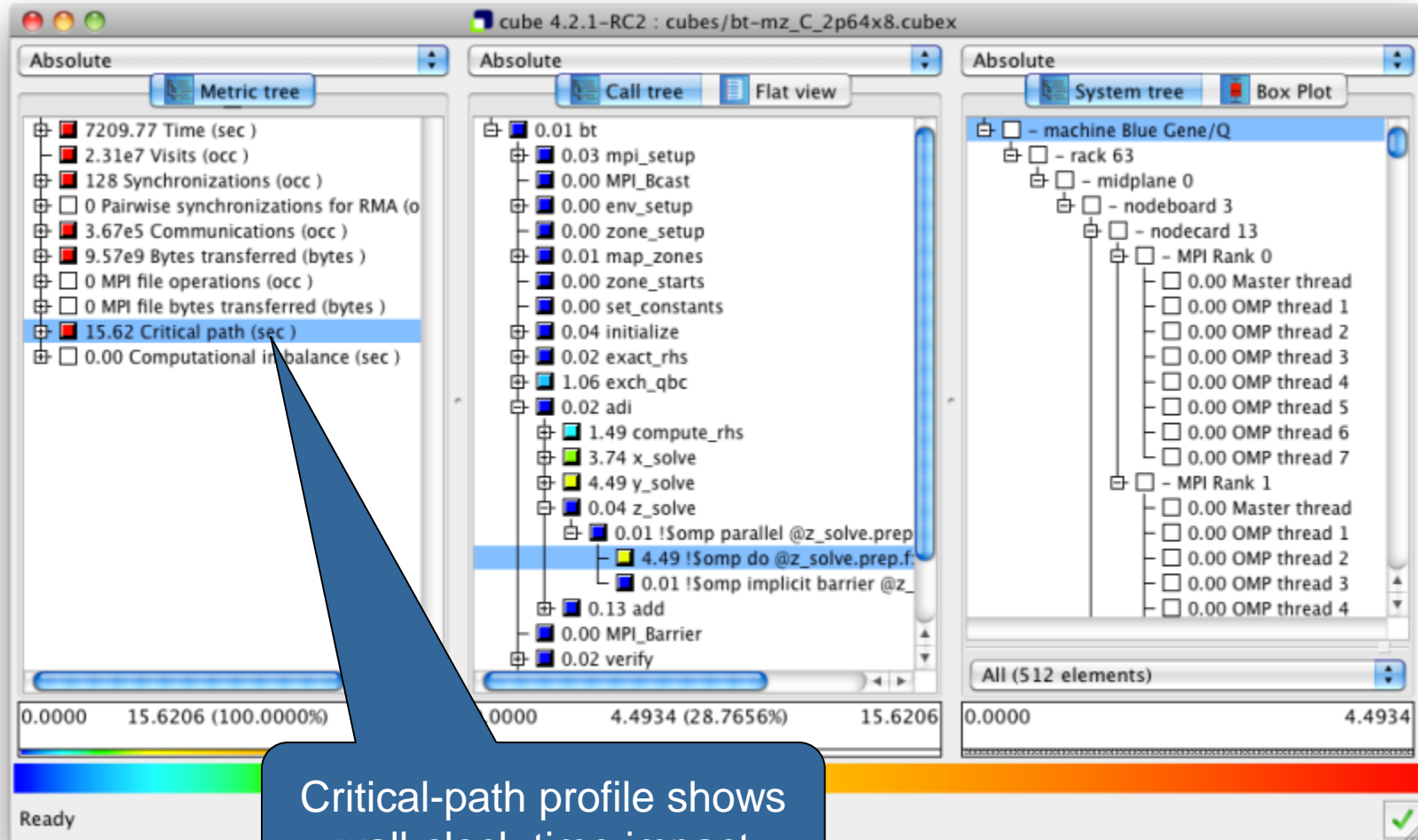
Unit:
Seconds

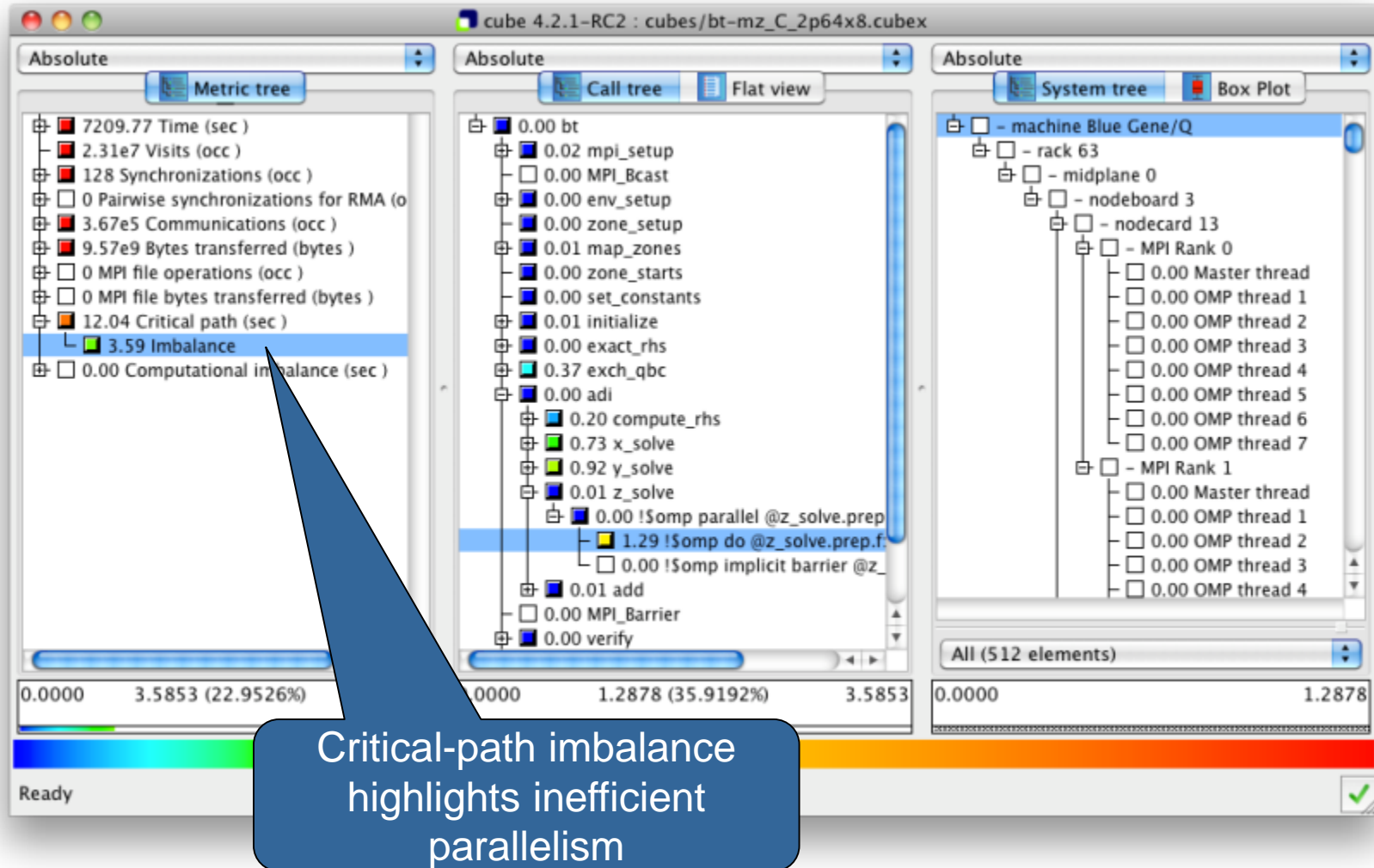
Diagnosis:
Try to replace `MPI_Recv` with a non-blocking receive `MPI_Irecv` that can be posted earlier, proceed concurrently with computation, and complete with a wait operation after the message is expected to have been sent. Try to post sends earlier, such that they are available when receivers need them. Note that outstanding messages (i.e., sent before the receiver is ready) will occupy internal message buffers, and that large numbers of posted receive buffers will also introduce message management overhead, therefore moderation is advisable.

Parent:
[MPI Point-to-point Communication Time](#)

Children:

Close





The screenshot displays the VI-HPS interface with a metric tree on the left. The tree shows a hierarchy of metrics, with '1.38 Late Sender' selected. A context menu is open over this node, listing various actions. The 'Statistics' option is highlighted. Two 'Statistics info' windows are open. The first window shows a bar chart with a y-axis from 0 to 0.035 and a single bar at approximately 0.03. The second window shows a table of statistics for the 'mpi_latesender' pattern.

Statistic	Value	Percentage
Pattern:	mpi_latesender	
Sum:	1.38	
Count:	832	
Mean:	0.00	5%
Standard deviation:	0.00	13%
Maximum:	0.03	100%
Upper quartile (Q3):	0.00	3%
Median:	0.00	3%
Lower quartile (Q1):	0.00	2%
Minimum:	0.00	0%

Context menu options:

- Info
- Full info
- Online description
- Expand/collapse
- Find items
- Find Next
- Clear found items
- Copy to clipboard
- Create derived metric...
- Remove metric...
- Statistics
- Max severity in trace browser

Statistics info window 1:

- Close

Statistics info window 2:

- To Clipboard
- Close

Callouts:

- Click to get statistics details
- Access pattern instance statistics via context menu

Connect to Vampir trace browser

The screenshot shows the Vampir trace browser interface. The 'File' menu is open, and the 'Connect to vampir...' option is selected. A dialog box titled 'Connect to vampir' is displayed, with the following fields: 'Open local file' (checked), 'Host: localhost', 'Port: 30000', and 'File: c:/supermuc_expts/scorep_bt-mz_B_4x4_trace/traces.otf2'. The 'Browse' button is highlighted. The background shows a call tree and system tree view.

To investigate most severe pattern instances, connect to a trace browser...

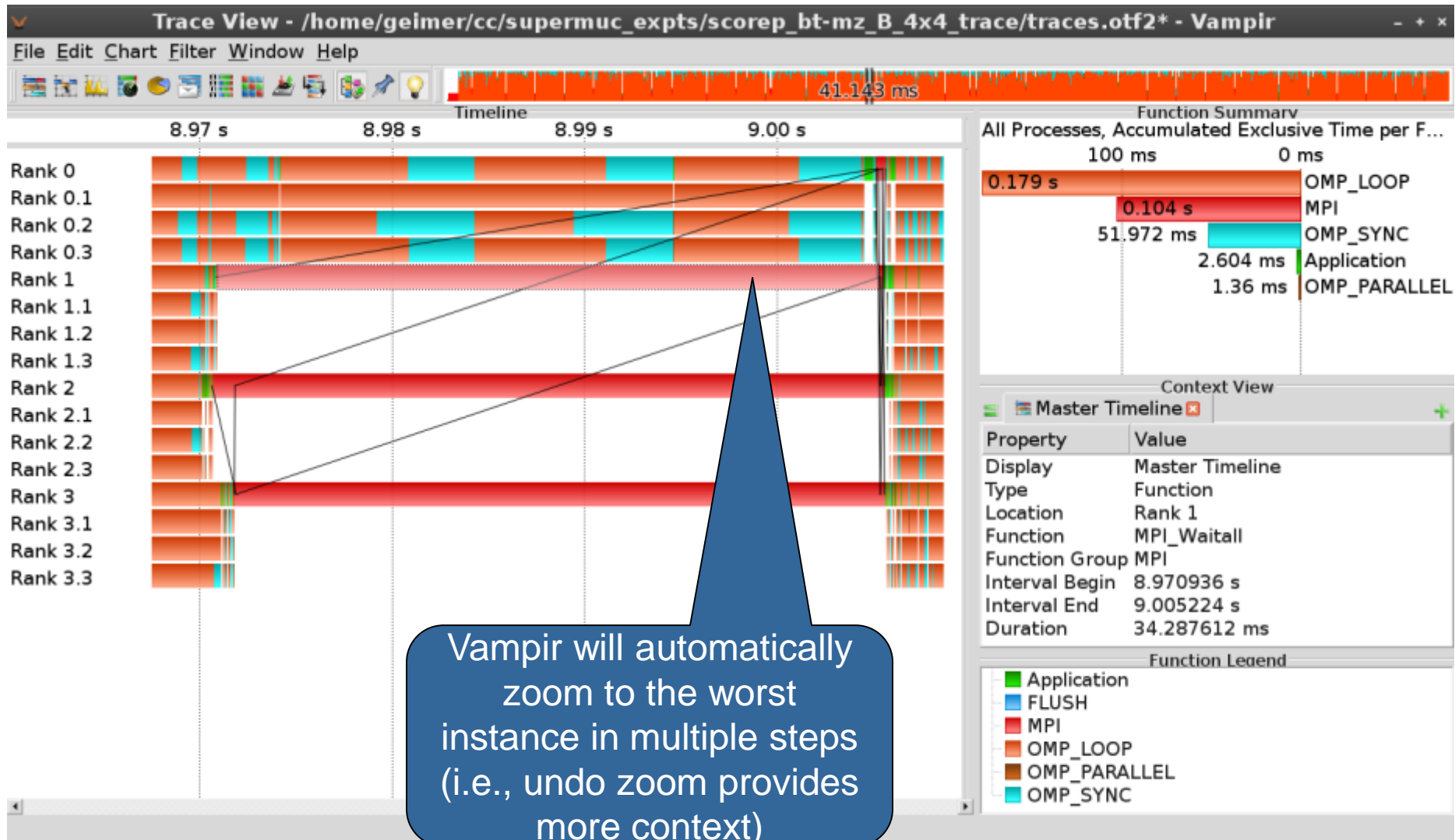
...and select trace file from the experiment directory

Connect to vampir and display a trace file

Show most severe pattern instances

The screenshot displays the VI-HPS trace browser interface with three main panels: Metric tree, Call tree, and System tree. The Metric tree on the left shows a hierarchy of performance metrics, with '1.38 Late Sender' highlighted. The Call tree in the center shows call paths, with '1.38 MPI_Waitany' highlighted and enclosed in a red frame. A context menu is open over this call path, listing various actions. The 'Max severity in trace browser' option is highlighted in blue. A blue callout bubble contains the instruction: 'Select "Max severity in trace browser" from context menu of call paths marked with a red frame'. The System tree on the right shows a hierarchical view of MPI ranks and threads.

Investigate most severe instance in Vampir



Scalable performance analysis of large-scale parallel applications

- toolset for scalable performance measurement & analysis of MPI, OpenMP & hybrid parallel applications
- supporting most popular HPC computer systems
- available under New BSD open-source license
- sources, documentation & publications:
 - <http://www.scalasca.org>
 - mailto: scalasca@fz-juelich.de

