TECHNISCHE UNIVERSITÄT DRESDEN

Performance Analysis at Scale using Score-P and Vampir

Score-P
Scalable performance measurement infrastructure for parallel codes

VAMPIR

Scaling your Science on Mira Workshop 2016

Frank Winkler (frank.winkler@tu-dresden.de)

ZIH
Center for Information Services & High Performance Computing

Disclaimer

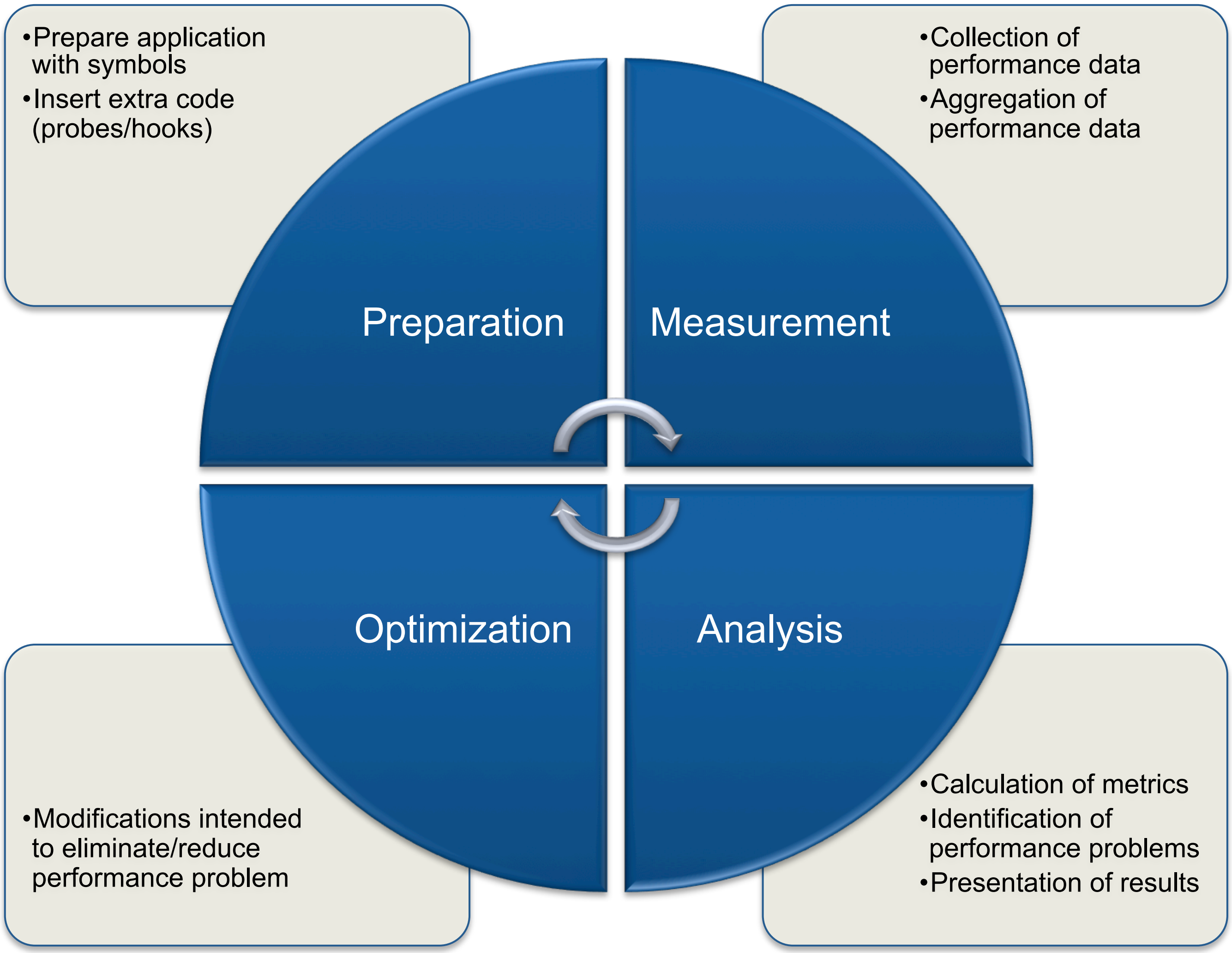
It is extremely easy to waste performance!

- Bad MPI (50-90%)
- No node-level parallelism (94%)
- No vectorization (75%)
- Bad memory access pattern (99%)
- In sum: 0.008% of the peak performance (785 gigaflops of Mira)

Disclaimer (2)

Performance tools will not automatically make your code run faster. They help you understand, what your code does and where to put in work.

Performance engineering workflow



Agenda

Performance Analysis Approaches

- Sampling vs. Instrumentation
- Profiling vs. Tracing

Score-P: Scalable Performance Measurement Infrastructure for Parallel Codes

- Architecture
- Workflow
- Cube

Vampir: Event Trace Visualization

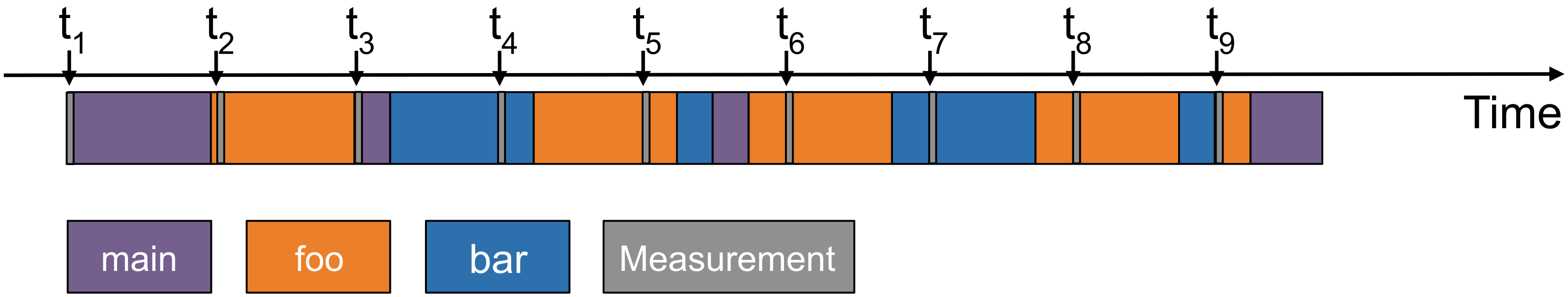
- Mission
- Visualization Modes
- Performance Charts

Demo

- Performance Analysis of NPB-MZ-MPI / BT on Mira

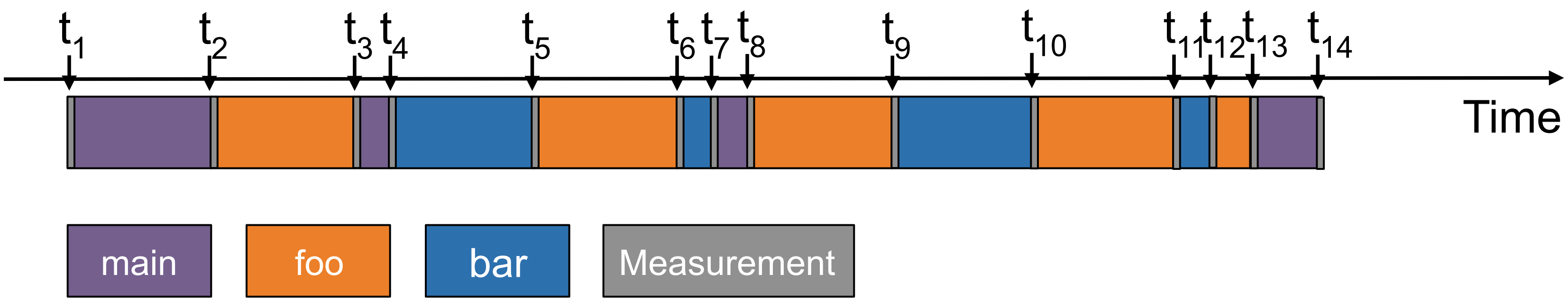
Conclusions

Sampling



- Running program is periodically interrupted to take measurement
- Statistical inference of program behavior
  - Not very detailed information on highly volatile metrics
  - Requires long-running applications
- Works with unmodified executables

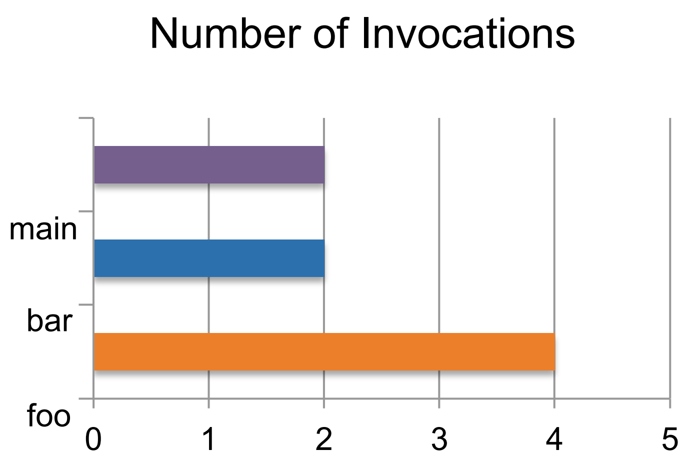
Instrumentation

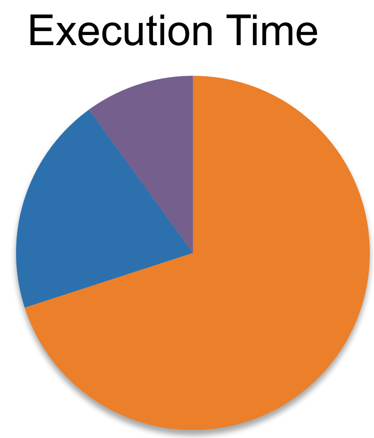


- Measurement code is inserted such that every event of interest is captured directly
- Advantage:
  - Much more detailed information
- Disadvantage:
  - Processing of source-code / executable necessary
  - Large relative overheads for small functions

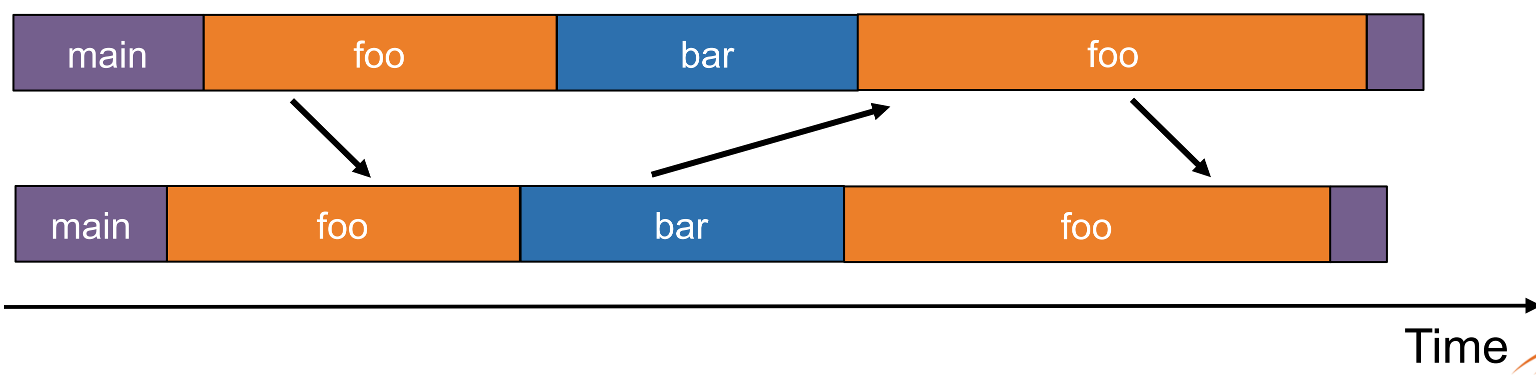
Profiling vs. Tracing

- Statistics

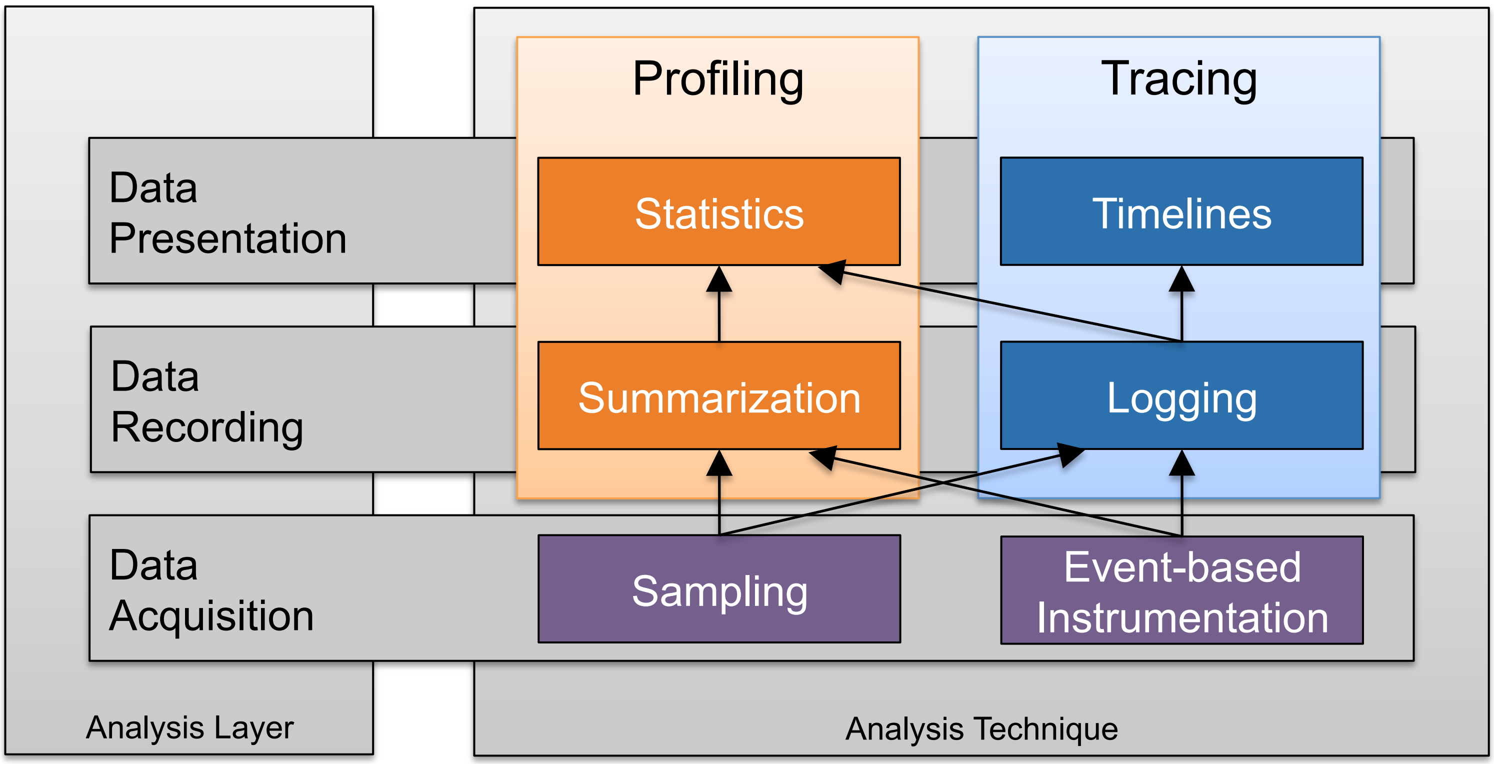




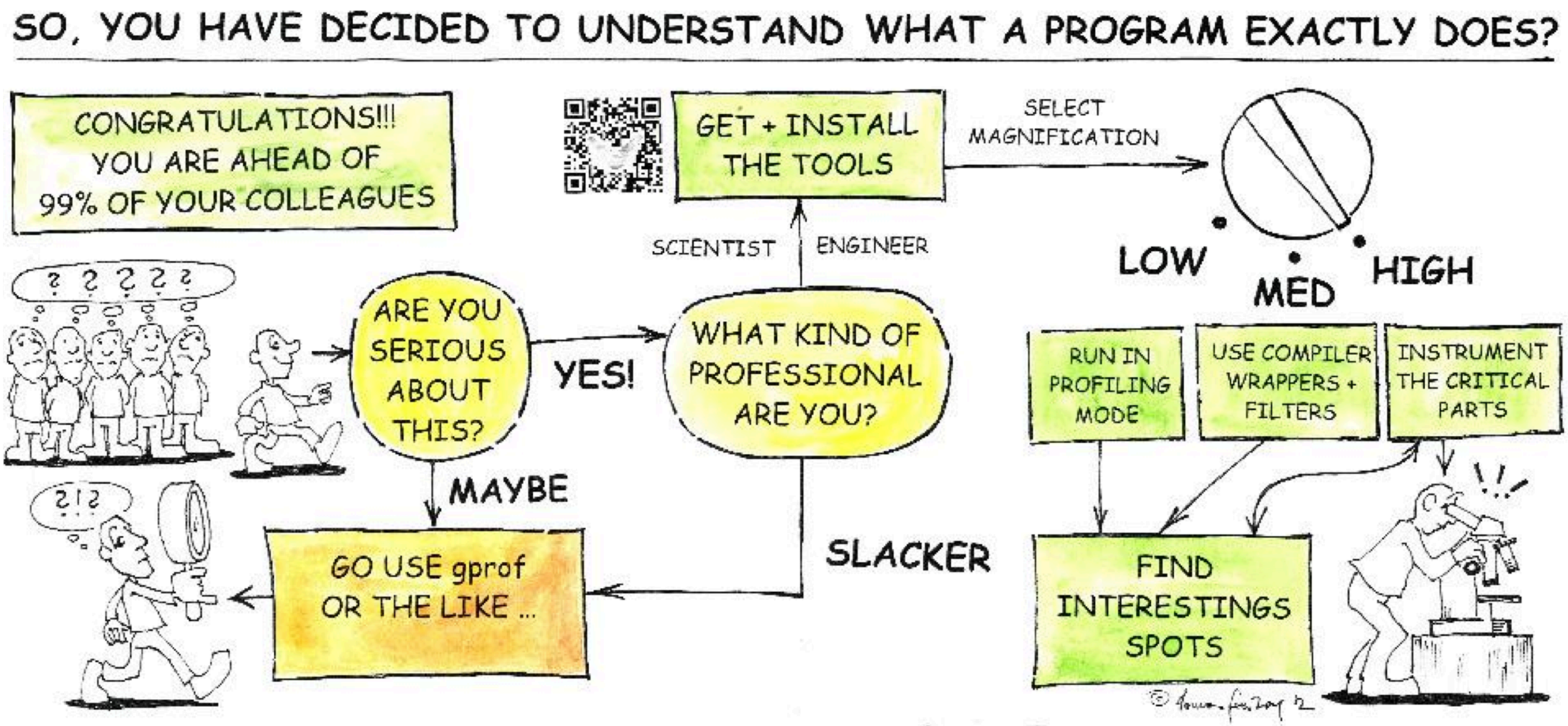
- Timelines



Terms Used and How They Connect



So what is the right choice?



Agenda

Performance Analysis Approaches

- Sampling vs. Instrumentation
- Profiling vs. Tracing

Score-P: Scalable Performance Measurement Infrastructure for Parallel Codes

- Architecture
- Workflow
- Cube

Vampir: Event Trace Visualization

- Mission
- Visualization Modes
- Performance Charts

Demo

- Performance Analysis of NPB-MZ-MPI / BT on Mira

Conclusions

Score-P: Motivation

- Several performance tools co-exist
- Separate measurement systems and output formats
- Complementary features and overlapping functionality
- Redundant effort for development and maintenance
- Limited or expensive interoperability
- Complications for user experience, support, training

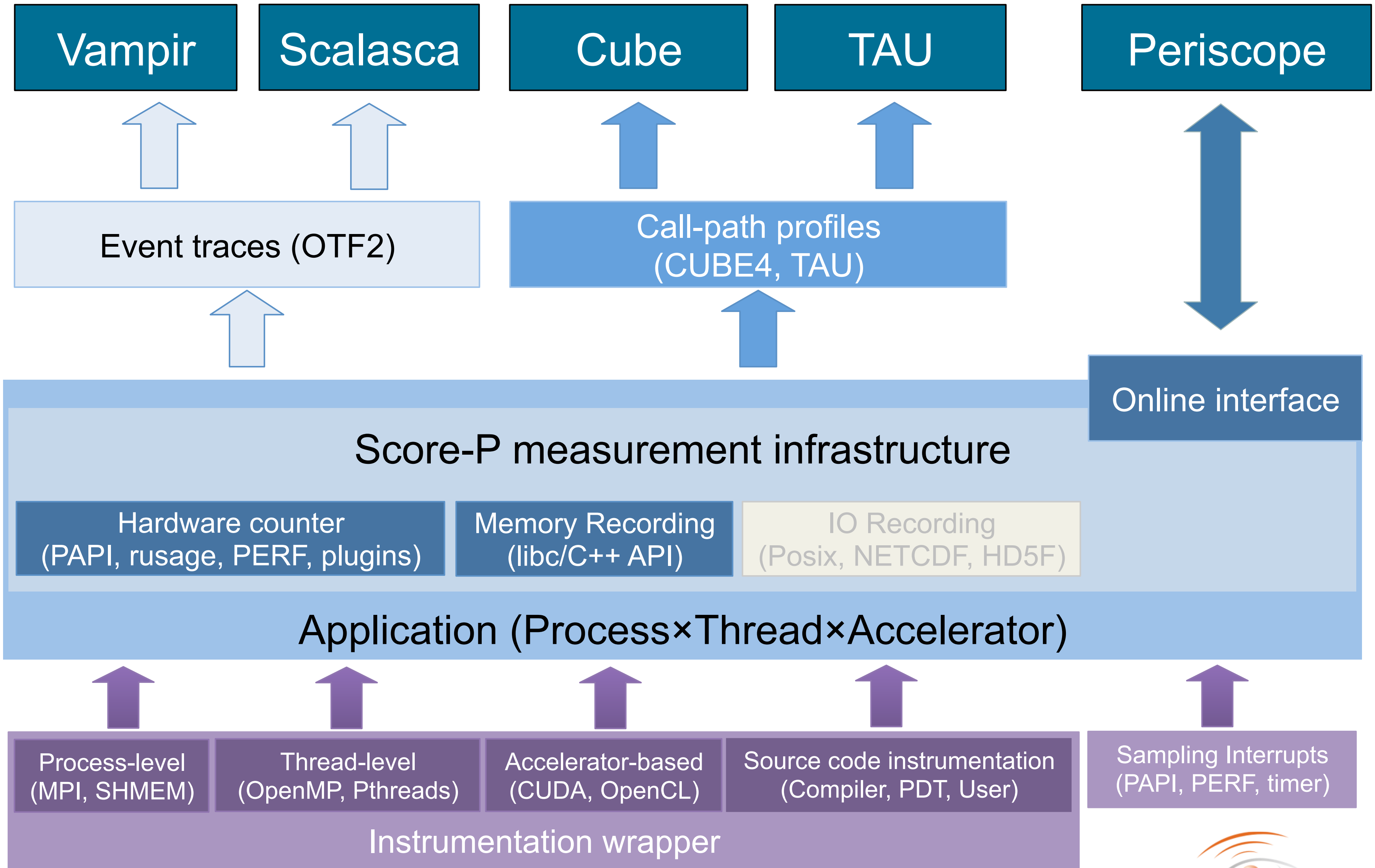
Vampir	Scalasca	TAU	Periscope
VampirTrace OTF	EPILOG / CUBE	TAU native formats	Online measurement

Score-P: Functionality

- Typical functionality for HPC performance tools
  – Instrumentation (various methods)
  – Sampling (experimental)
- Flexible measurement without re-compilation
  – Basic and advanced profile generation
  – Event trace recording
- Programming paradigms:
  – Multi-process
    - MPI, SHMEM
  – Thread-parallel
    - OpenMP, Pthreads
  – Accelerator-based
    - CUDA, OpenCL

Hybrid parallelism

Score-P: Architecture



Score-P: General Workflow

1. Perform a reference run and note the runtime
2. Instrument your application with Score-P
3. Create a profile with full instrumentation
4. Compare runtime with reference runtime to determine overhead

   If overhead is too high:

   → Create filter file using hints from scorep-score

   → Generate an optimized profile with filter applied
5. Investigate profile with Cube
6. Define (or adjust) filter file for a tracing run using scorep-score
7. Generate a trace with filter applied
8. Perform in-depth analysis on the trace data with Vampir

Score-P: Workflow / Instrumentation

```
CC     = icc
CXX    = icpc
F90    = ifc
MPICC = mpicc
```

```
CC     = scorep <options> icc
CXX    = scorep <options> icpc
F90    = scorep <options> ifc
MPICC = scorep <options> mpicc
```

- To see all available options for instrumentation:

```
$ scorep --help
This is the Score-P instrumentation tool. The usage is:
scorep <options> <original command>

Common options are:
...
  --instrument-filter=<file>
                  Specifies the filter file for filtering functions during
                  compile-time. It applies the same syntax, as the one
                  used by Score-P during run-time.

  --user          Enables user instrumentation.
```

Score-P: Workflow / Measurement

- Measurements are configured via environment variables

```
$ scorep-info config-vars --full

SCOREP_ENABLE_PROFILING
   [...]
SCOREP_ENABLE_TRACING
   [...]
SCOREP_TOTAL_MEMORY
  Description: Total memory in bytes for the measurement system
   [...]
SCOREP_EXPERIMENT_DIRECTORY
  Description: Name of the experiment directory
   [...]
```

- Example for generating a profile:

```
$ export SCOREP_ENABLE_PROFILING=true
$ export SCOREP_ENABLE_TRACING=false
$ export SCOREP_EXPERIMENT_DIRECTORY=profile

$ mpiprun <instrumented binary>
```

Score-P: Workflow / Filtering

- Use scorep-score to define a filter
  - Exclude short frequently called functions from measurement
    - For profiling: reduce measurement overhead (if necessary)
    - For tracing: reduce measurement overhead and total trace size

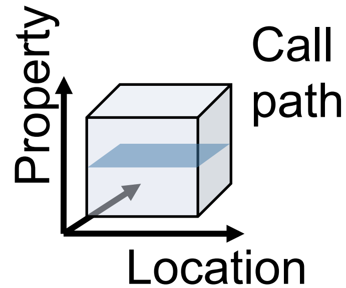
```
$ scorep-score -r profile/profile.cubex
Estimated aggregate size of event trace:                   40GB
Estimated requirements for largest trace buffer (max_buf): 10GB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       10GB
  [...]
Flt type     max_buf[B]        visits time[s] time[%] time/visit[us]  region
  [...]
     USR  3,421,305,420   522,844,416  144.46    13.4           0.28  matmul_sub
     USR  3,421,305,420   522,844,416  102.40     9.5           0.20  matvec_sub
     USR  3,421,305,420   522,844,416  200.94    18.6           0.38  binvcrhs
     USR    150,937,332    22,692,096    5.58     0.5           0.25  binvrhs
     USR    150,937,332    22,692,096   13.21     1.2           0.58  lhsinit
```

- Filter file:

```
$ vim scorep.filt
SCOREP_REGION_NAMES_BEGIN EXCLUDE
  matmul_sub
  matvec_sub
  binvcrhs
```

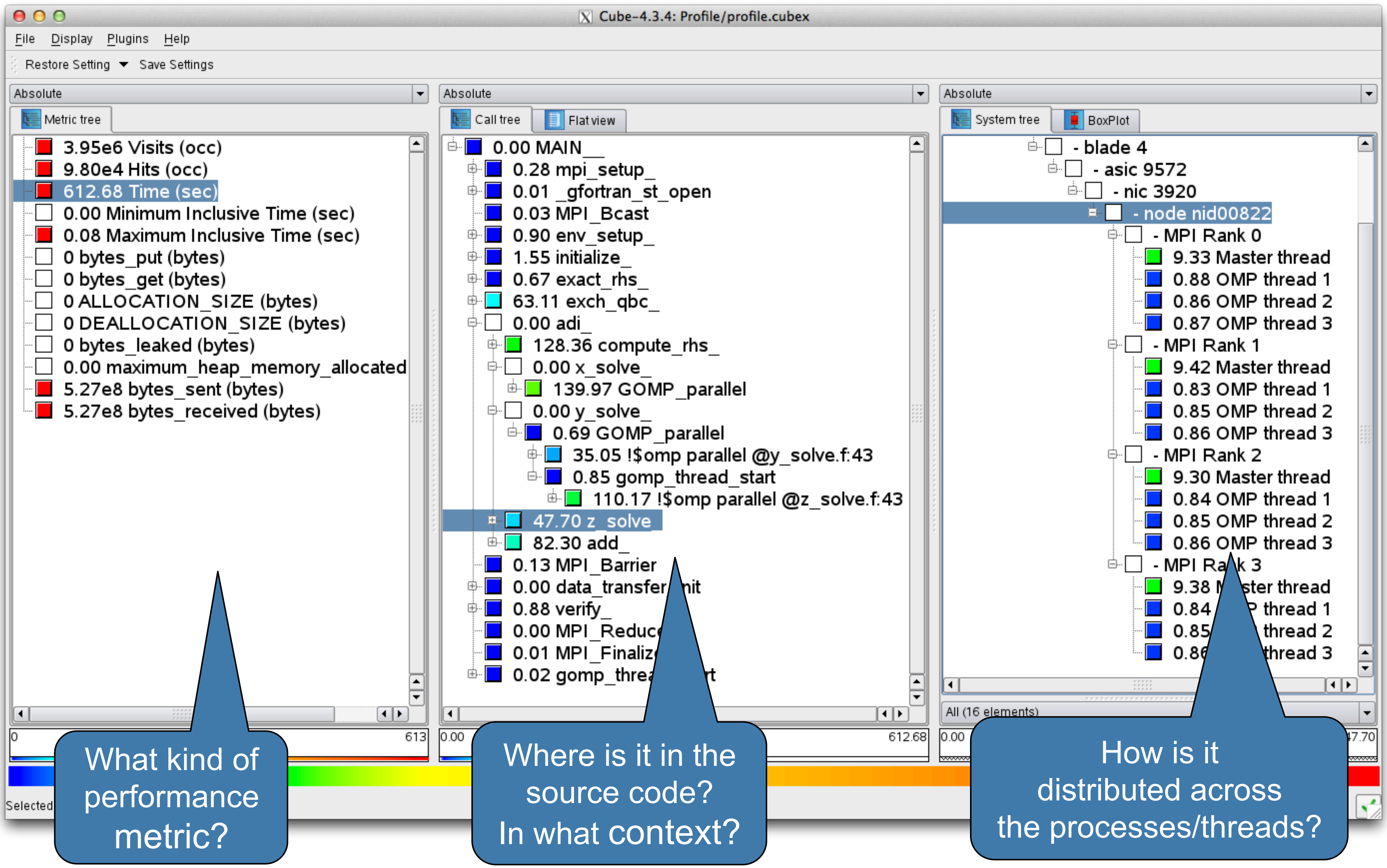
Score-P: Cube

- Profile analysis tool for displaying performance data of parallel programs
- Originally developed as part of Scalasca toolset
- Available as a separate component of Score-P
- Representation of values (severity matrix) on three hierarchical axes
  – Performance property (metric)
  – Call-tree path (program location)
  – System location (process/thread)



- Three coupled tree browsers

Score-P: Cube Analysis Presentation



Agenda

Performance Analysis Approaches

- Sampling vs. Instrumentation
- Profiling vs. Tracing

Score-P: Scalable Performance Measurement Infrastructure for Parallel Codes

- Architecture
- Workflow
- Cube

Vampir: Event Trace Visualization

- Mission
- Visualization Modes
- Performance Charts

Demo

- Performance Analysis of NPB-MZ-MPI / BT on Mira

Conclusions

Vampir: Mission

- Visualization of dynamics of complex parallel processes
- Requires two components
  - Monitor/Collector (Score-P)
  - Charts/Browser (Vampir)

Typical questions that Vampir helps to answer:

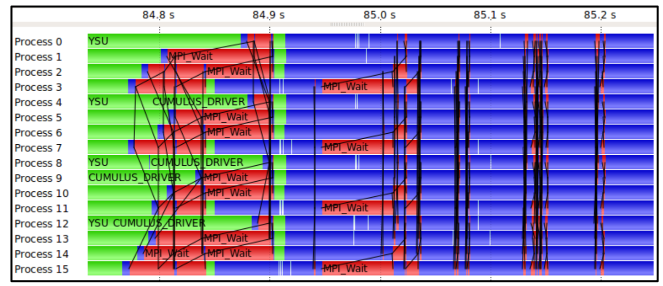
- What happens in my application execution during a given time in a given process or thread?
- How do the communication patterns of my application execute on a real system?
- Are there any imbalances in computation, I/O or memory usage and how do they affect the parallel execution of my application?

Vampir: Event Trace Visualization

- Show dynamic run-time behavior graphically at a fine level of detail
- Provide summaries (profiles) on performance metrics

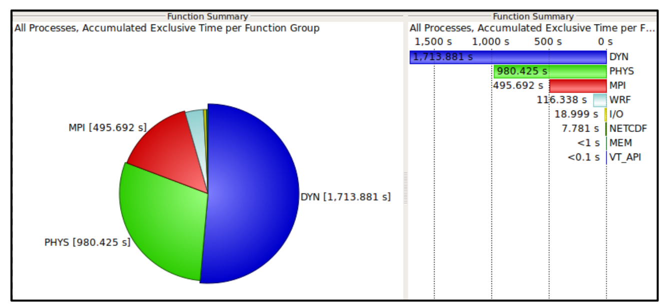
Timeline charts

- Show application activities and communication along a time axis



Summary charts

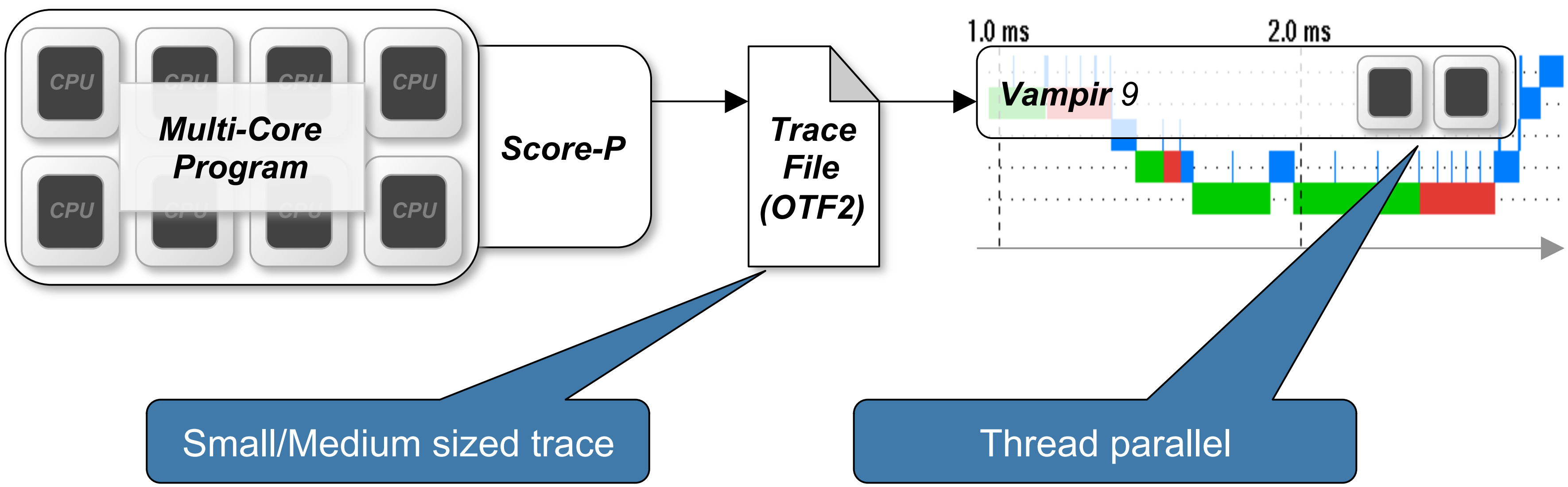
- Provide quantitative results for the currently selected time interval



Vampir: Visualization Modes (1)

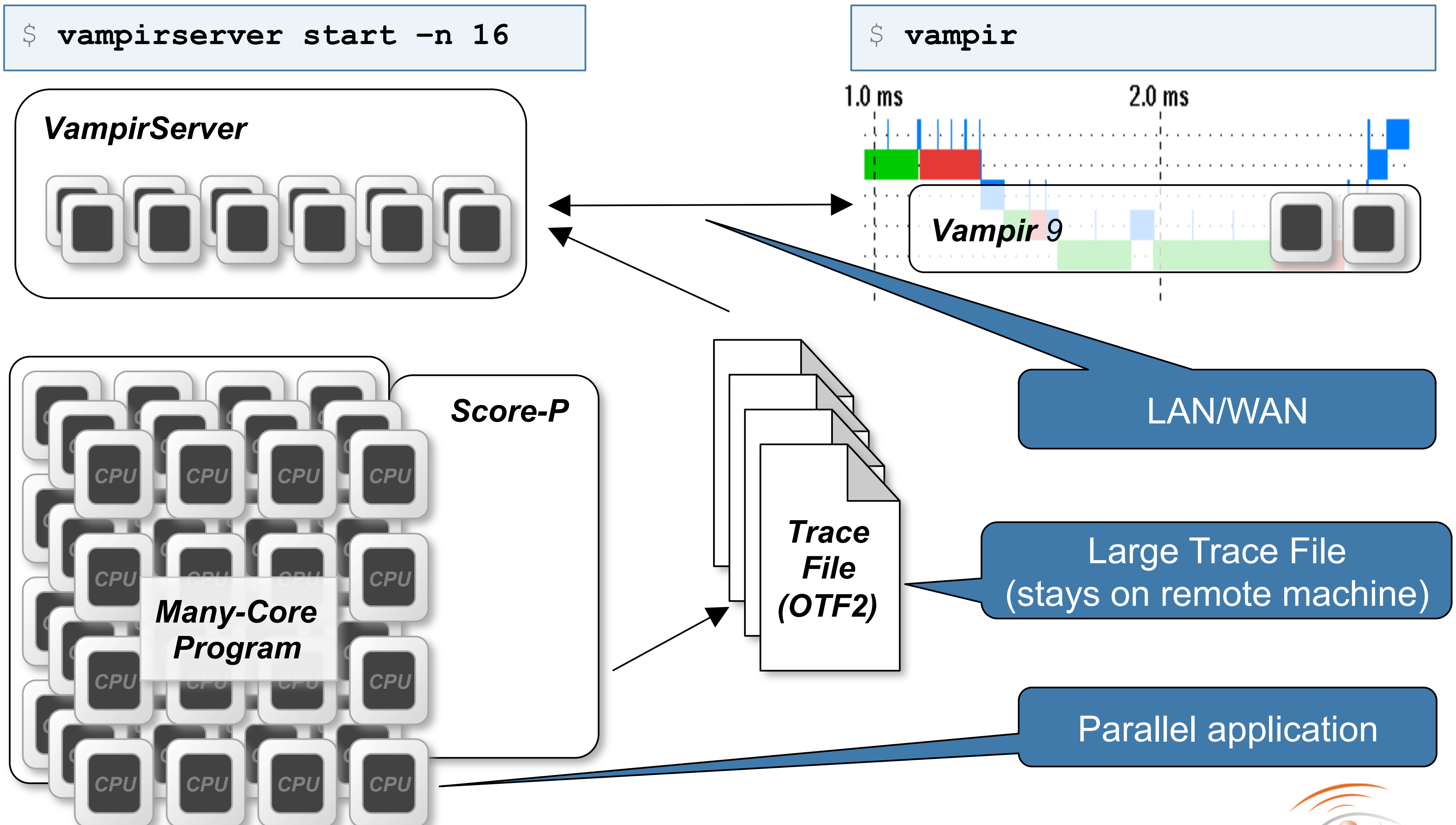
- Directly on front end or local machine

```
$ vampir
```



Vampir: Visualization Modes (2)

- On local machine with remote VampirServer



Vampir: Main Performance Charts

Timeline Charts

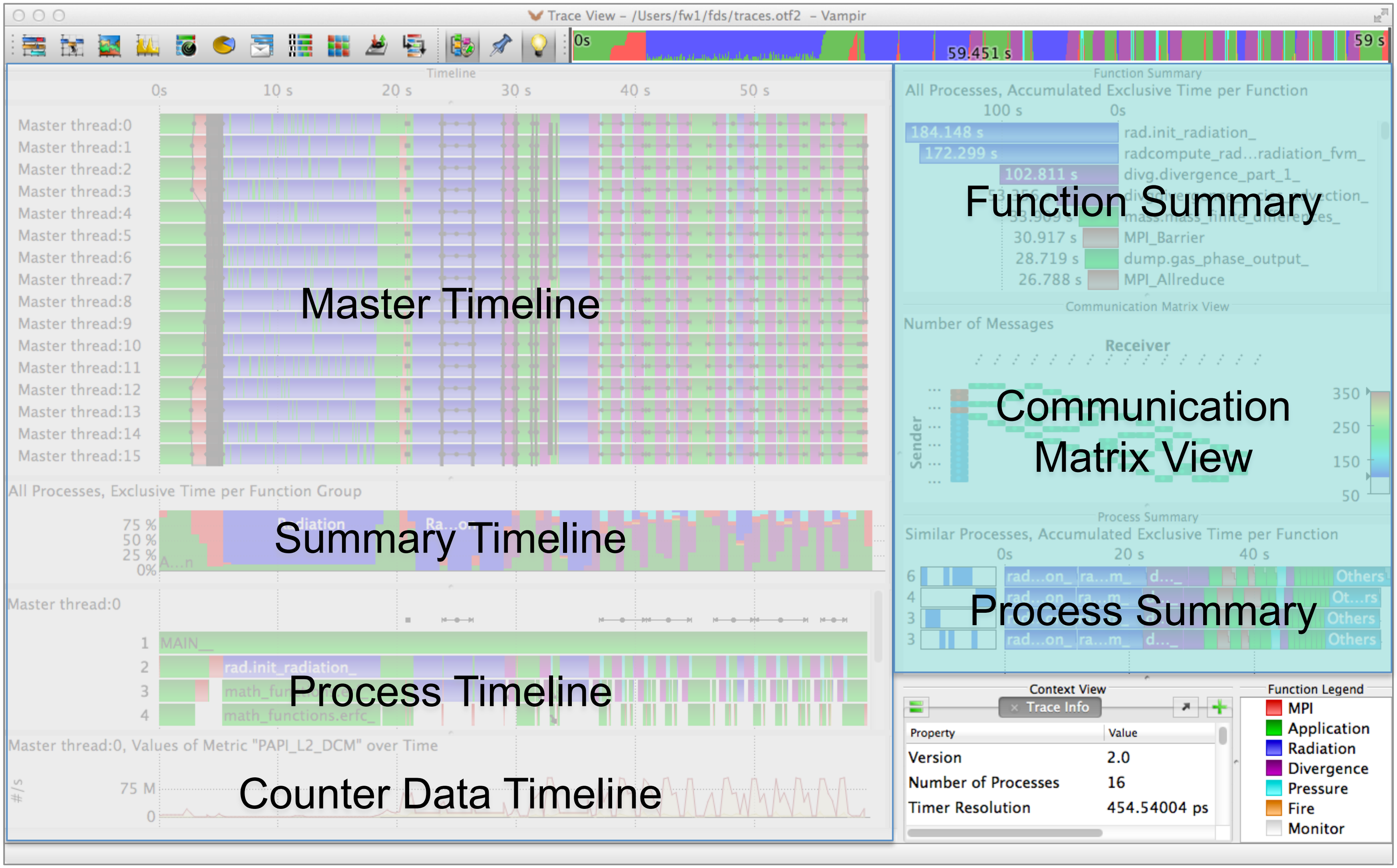
Master Timeline	➡	*all threads' activities over time per thread*
Summary Timeline	➡	*all threads activities over time per activity*
Performance Radar	➡	*all threads' perf-metric over time*
Process Timeline	➡	*single thread's activities over time*
Counter Data Timeline	➡	*single threads perf-metric over time*

Summary Charts

Function Summary	Process Summary
Message Summary	Communication Matrix View

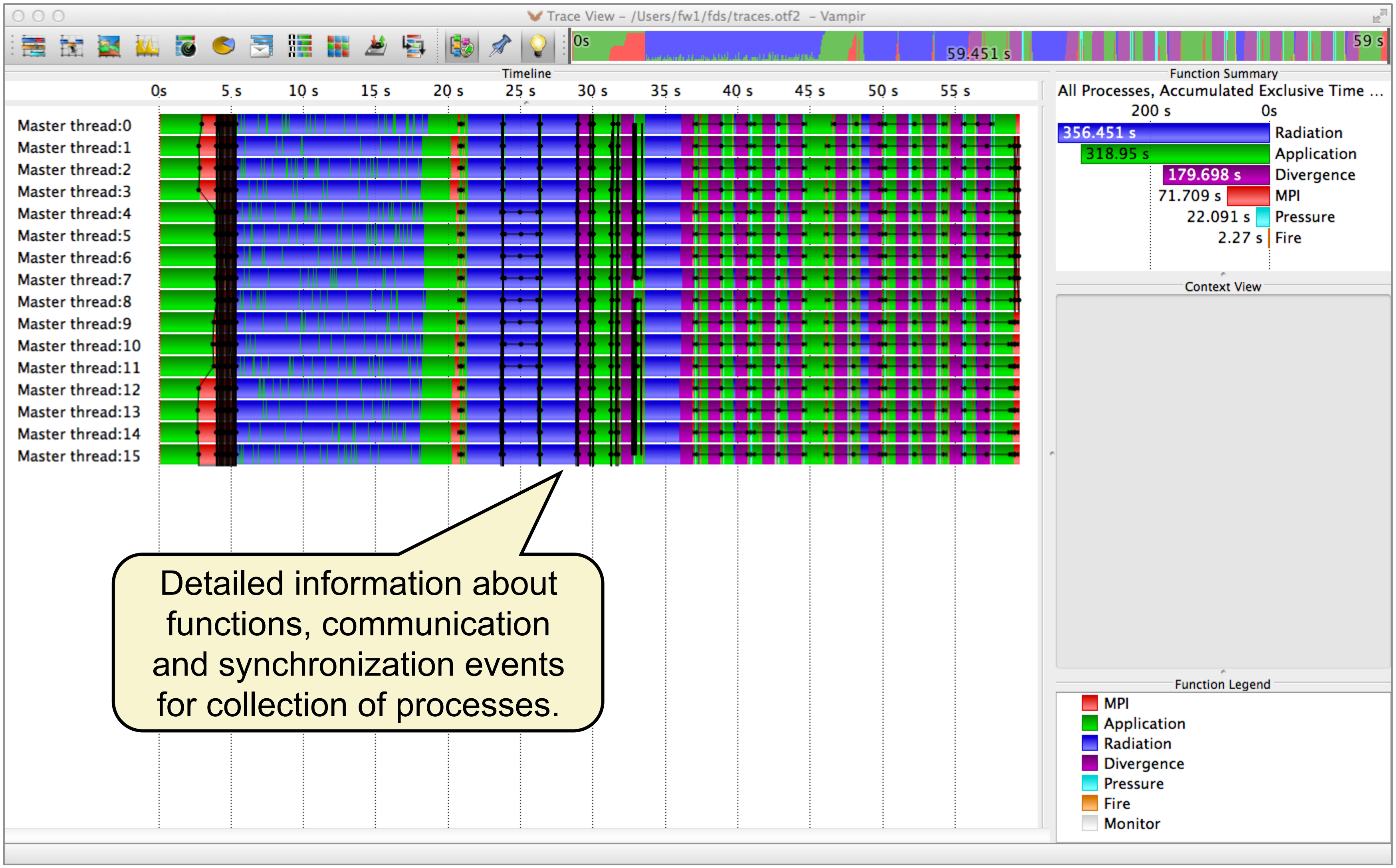
Vampir: Performance Charts

- Trace visualization of FDS (Fire Dynamics Simulator)



Vampir: Performance Charts

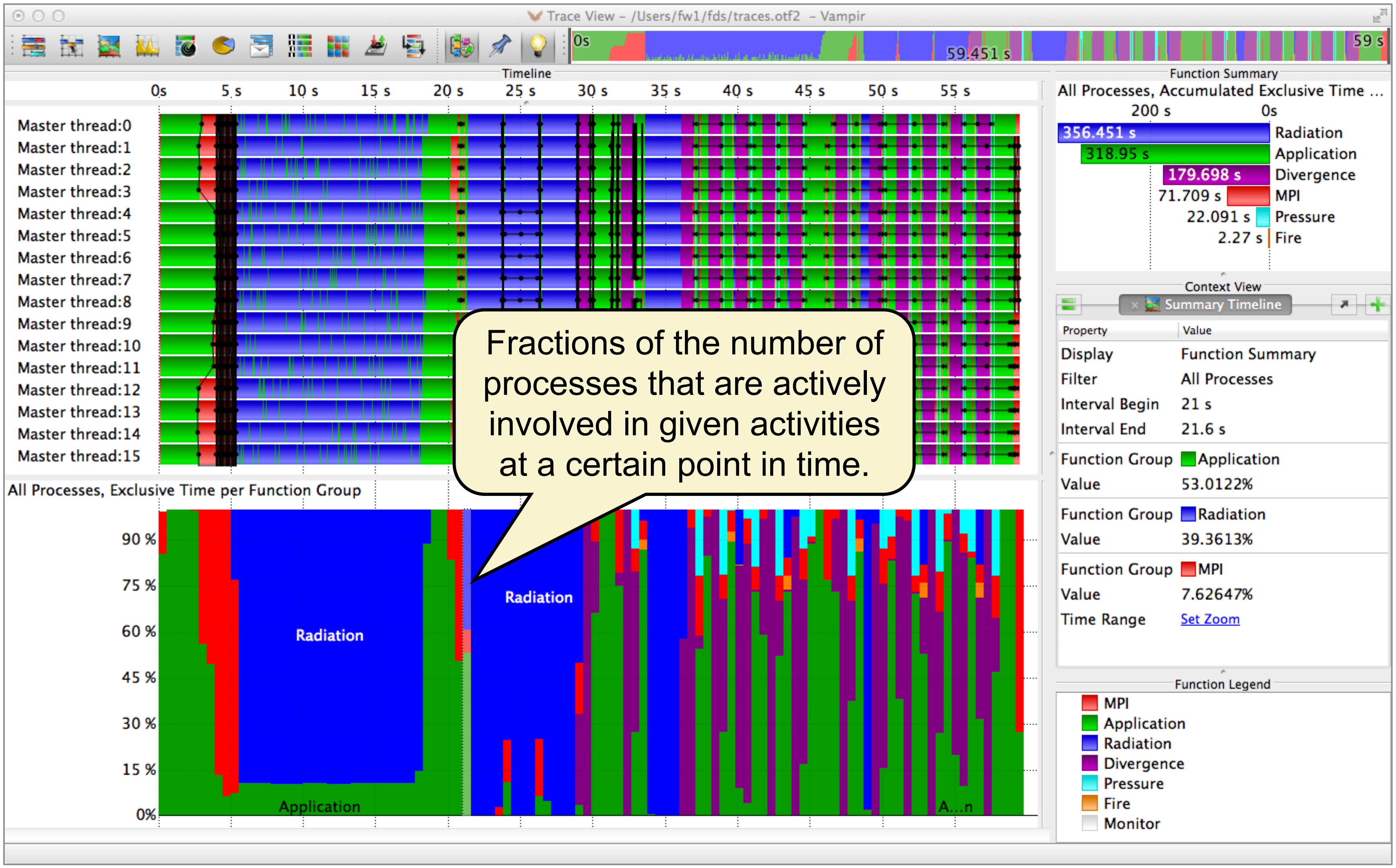
Master Timeline



Detailed information about functions, communication and synchronization events for collection of processes.

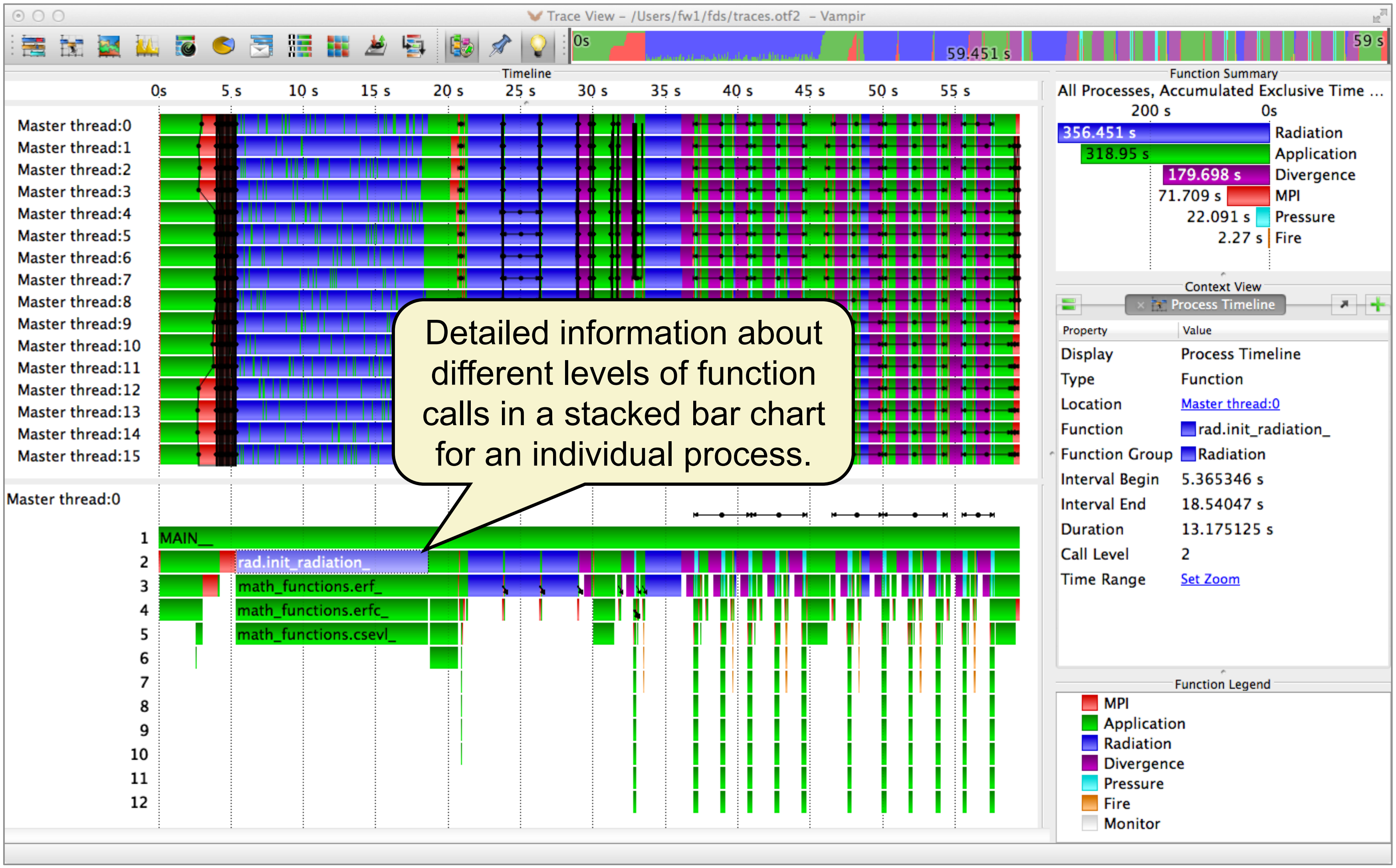
Vampir: Performance Charts

Summary Timeline



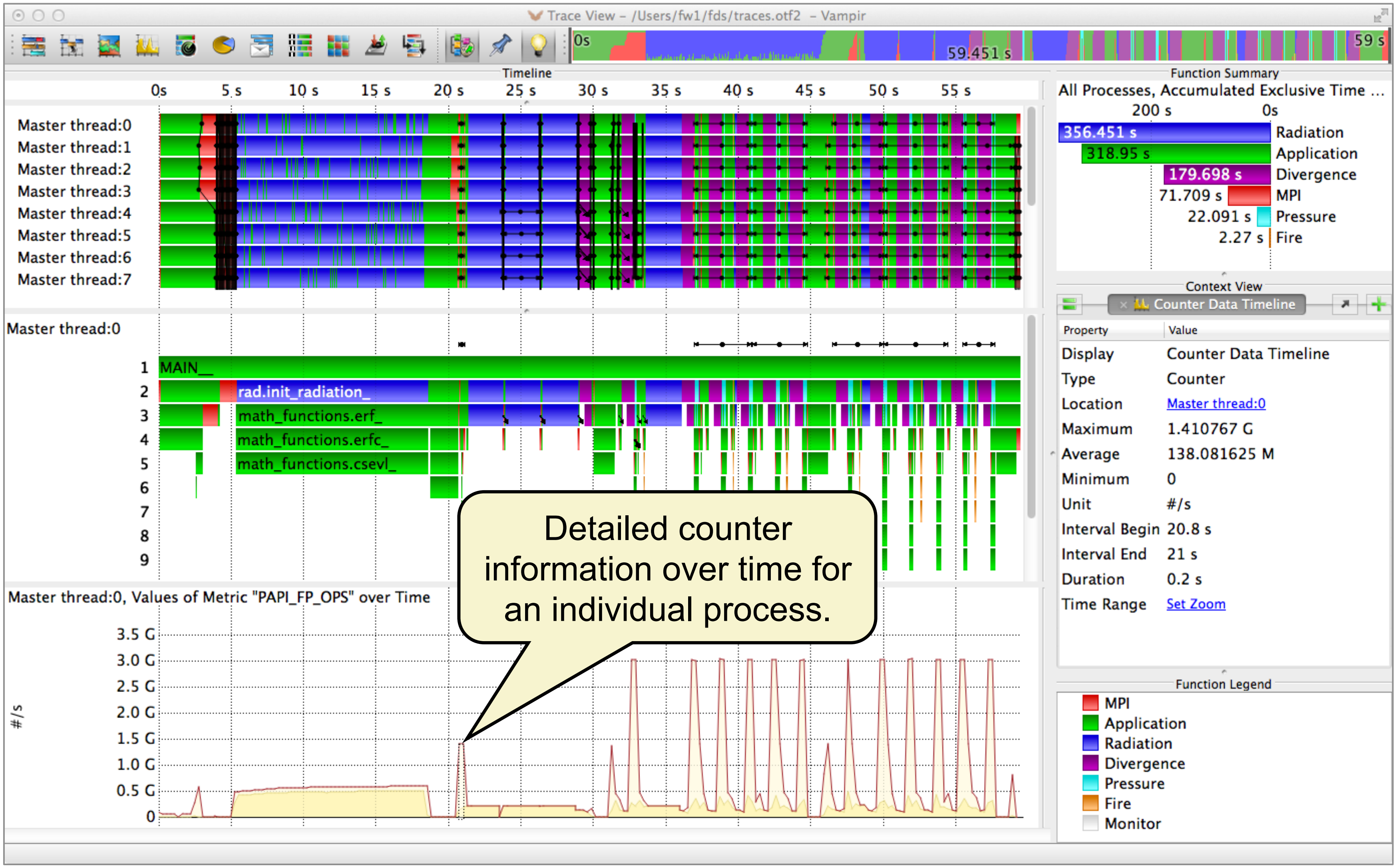
Vampir: Performance Charts

Process Timeline



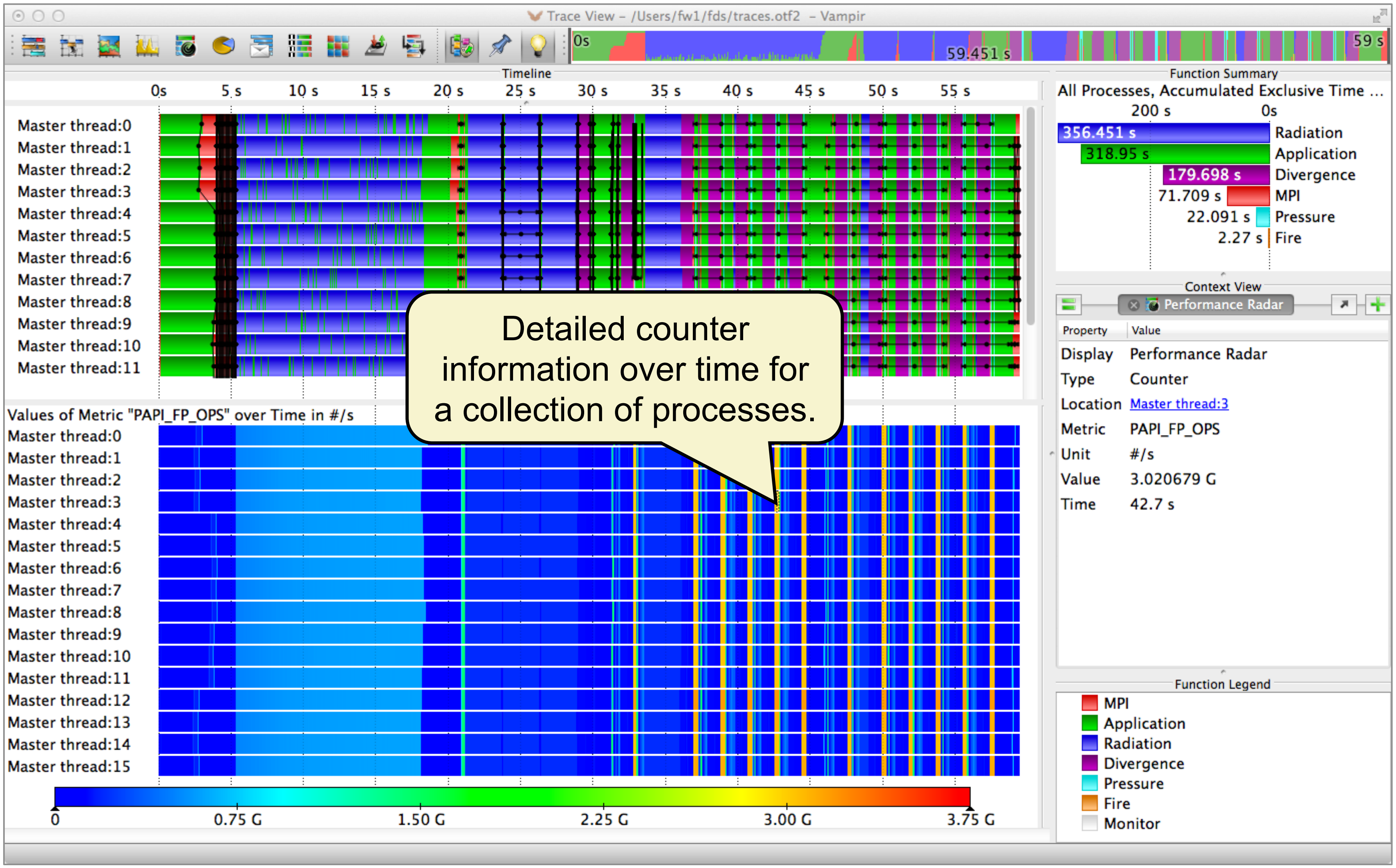
Vampir: Performance Charts

Counter Timeline



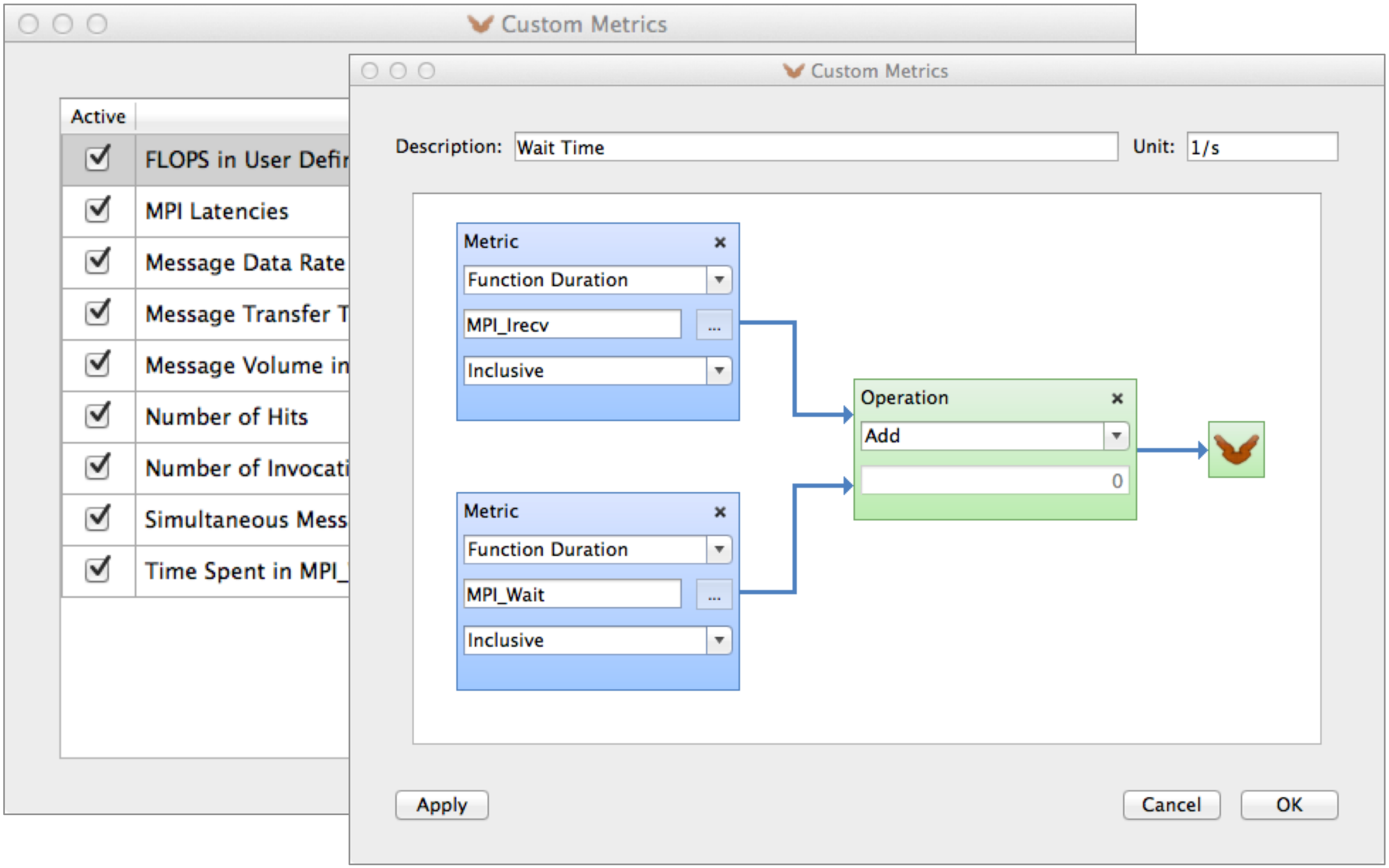
Vampir: Performance Charts

Performance Radar



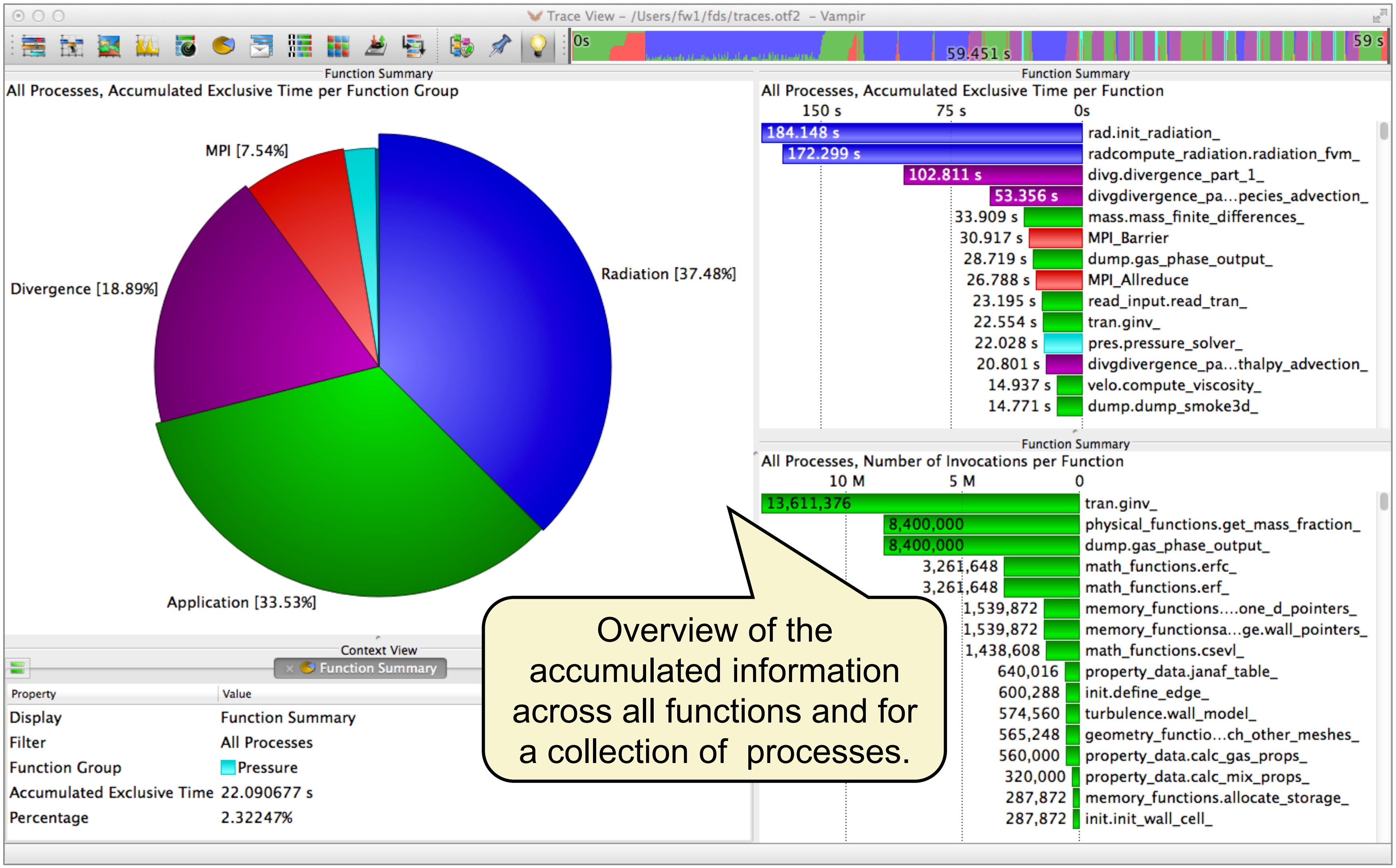
Vampir: Where Do the Metrics Come From?

- Custom Metrics Built-In Editor



Vampir: Performance Charts

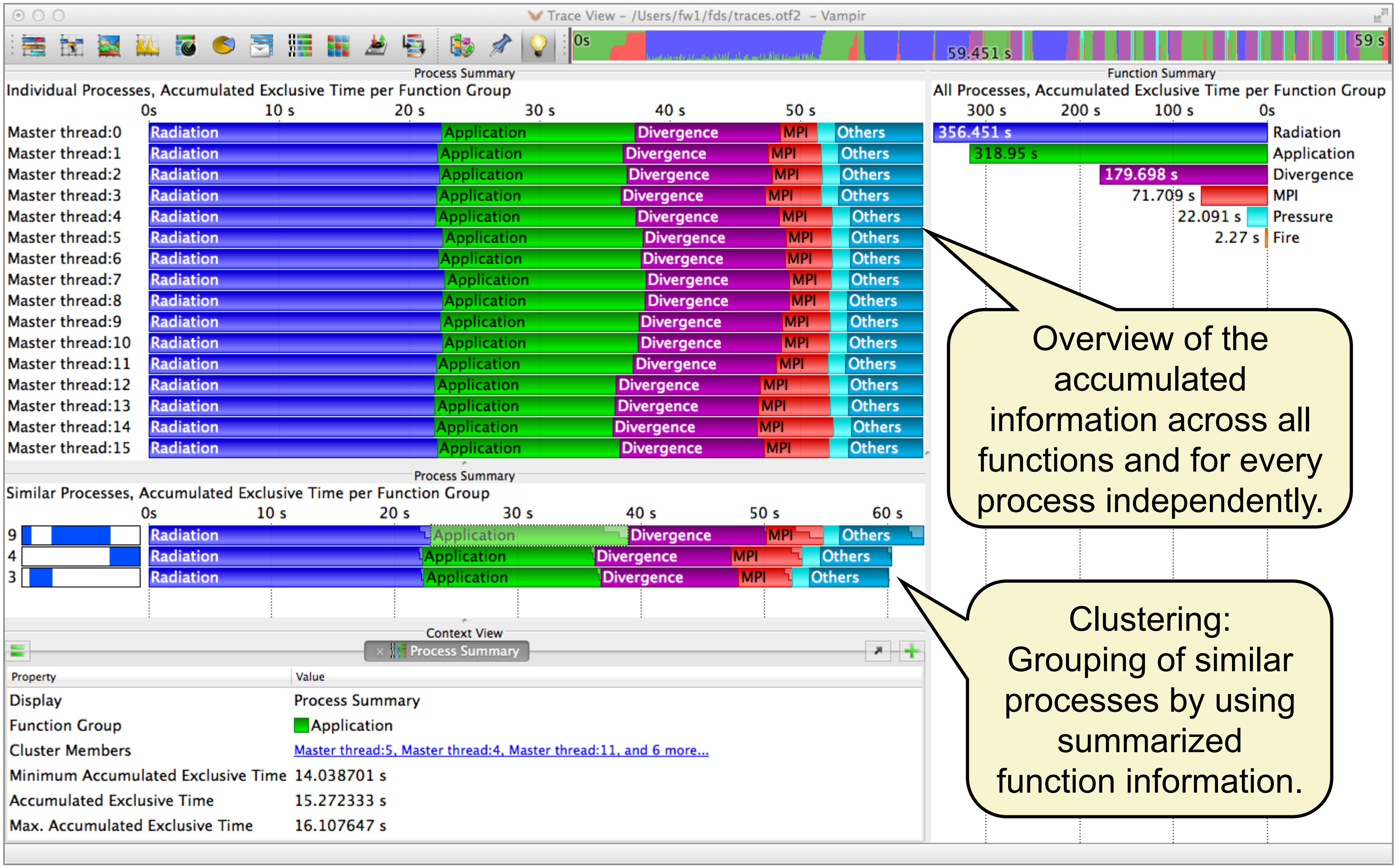
Function Summary



Overview of the accumulated information across all functions and for a collection of processes.

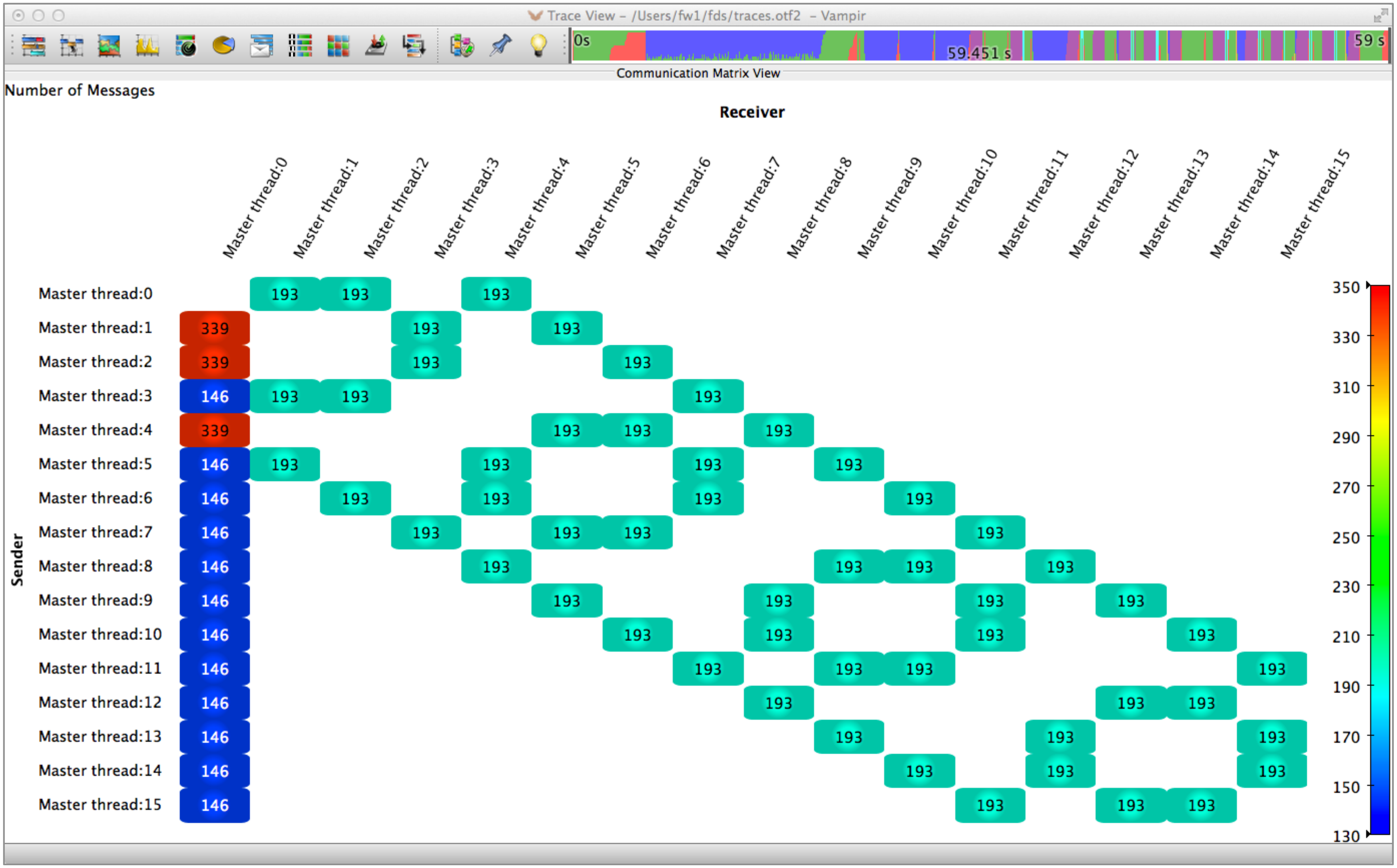
Vampir: Performance Charts

Process Summary



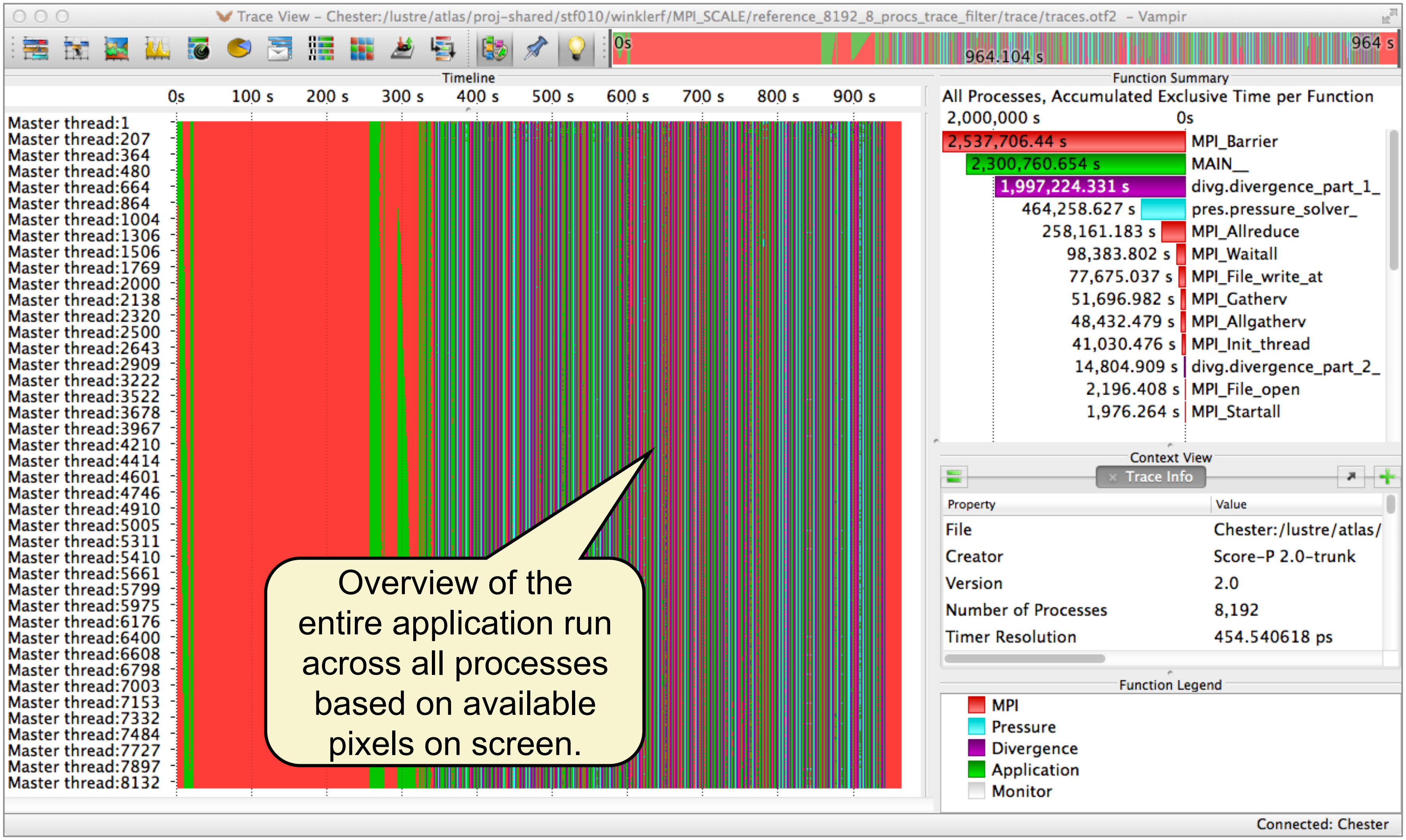
Vampir: Performance Charts

Communication Matrix View



Vampir at Scale: FDS with 8192 cores

- Fit to chart height feature in Master Timeline



Agenda

Performance Analysis Approaches

- Sampling vs. Instrumentation
- Profiling vs. Tracing

Score-P: Scalable Performance Measurement Infrastructure for Parallel Codes

- Architecture
- Workflow
- Cube

Vampir: Event Trace Visualization

- Mission
- Visualization Modes
- Performance Charts

Demo

- Performance Analysis of NPB-MZ-MPI / BT on Mira

Conclusions

Vampir Demo: NPB-MZ-MPI / BT

- The NAS Parallel Benchmark suite (MPI+OpenMP version)
  - Available from: http://www.nas.nasa.gov/Software/NPB
  - 3 benchmarks in Fortran77 (bt-mz, lu-mz, sp-mz)
  - Configurable for various sizes & classes (S, W, A, B, C, D, E)
- Benchmark configuration for demo:
  - Benchmark name: **bt-mz**
  - Number of MPI processes: **NPROCS=4**
  - Benchmark class: **CLASS=W**
  - 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

NPB-MZ-MPI / BT Build

- Connect to Mira and add Score-P to the SoftEnv system

```
% vi .soft
  +scorep
% resoft
```

- Copy sources to working directory

```
% cp /projects/Tools/scorep/tutorial/NPB3.3-MZ-MPI.tar.gz .
% tar xzvf NPB3.3-MZ-MPI.tar.gz
% cd NPB3.3-MZ-MPI
```

- Compile the benchmark

```
% make bt-mz CLASS=W NPROCS=4
cd BT-MZ; make CLASS=W NPROCS=4 VERSION=
make: Entering directory 'BT-MZ'
cd ../sys; cc  -o setparams setparams.c
../sys/setparams bt-mz 4 W
mpixlf77_r -c  -O3 -qsmp=omp -qextname=flush bt.f
 [...]
Built executable ../bin/bt-mz_W.4
make: Leaving directory 'BT-MZ'
```

NPB-MZ-MPI / BT Reference Execution

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

```
% cd bin
% cp ../jobscript/mira/run.sh .
% less run.sh
  export OMP_NUM_THREADS=4
  runjob -n 4 -p 4 --block $COBALT_PARTNAME --env-all : bt-mz_W.4
% qsub -A <projid> -t 10 -n 1 --mode script run.sh
% cat <jobid>.outpout
  NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark
  Number of zones:   4 x   4
  Iterations:  200    dt:   0.000800
  Number of active processes:     4
  Total number of threads:     16  (  4.0 threads/process)

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

  BT-MZ Benchmark Completed.
  Time in seconds = 2.27
```

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

NPB-MZ-MPI / BT Instrumentation

- Edit config/make.def to adjust build configuration
- Modify specification of compiler/linker: MPIF77

```
#              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 = mpixlf77_r

# Alternative variants to perform instrumentation
...
MPIF77 = scorep mpixlf77_r

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

Uncomment the Score-P compiler wrapper specification

NPB-MZ-MPI / BT Instrumented Build

- Return to root directory and clean-up

```
% make clean
```

- Re-build executable using Score-P compiler wrapper

```
% make bt-mz CLASS=W NPROCS=4
cd BT-MZ; make CLASS=W NPROCS=4 VERSION=
make: Entering directory 'BT-MZ'
cd ../sys; cc  -o setparams setparams.c
../sys/setparams bt-mz 4 W
scorep mpixlf77_r -c  -O3 -qsmp=omp -qextname=flush bt.f
 [...]
cd ../common; scorep mpixlf77_r -c -O3 -qsmp=omp -qextname=flush timers.f
scorep mpixlf77_r -O3 -qsmp=omp -qextname=flush -o ../bin.scorep/bt-mz_W.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.scorep/bt-mz_W.4
make: Leaving directory 'BT-MZ'
```

NPB-MZ-MPI / BT Summary Measurement Collection

- Change to the directory containing the new executable before running it and adjust configuration

```
% cd bin.scorep
% cp ../jobscript/mira/* .
% less run_profile.sh
  export SCOREP_ENABLE_TRACING=false
  export SCOREP_ENABLE_PROFILING=true
  export SCOREP_TOTAL_MEMORY=100M
  export SCOREP_EXPERIMENT_DIRECTORY=scorep_bt-mz_W_4x4_sum
  export OMP_NUM_THREADS=4
  runjob -n 4 -p 4 --block $COBALT_PARTNAME --env-all : bt-mz_W.4
% qsub -A <projid> -t 10 -n 1 --mode script run_profile.sh
% cat <jobid>.outpout
  NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark
  Number of zones:   4 x   4
    [...]
  Time step  200
  Verification Successful

  BT-MZ Benchmark Completed.
  Time in seconds = 12.74
```

Measurement overhead too high!

NPB-MZ-MPI / BT Summary Analysis Result Scoring

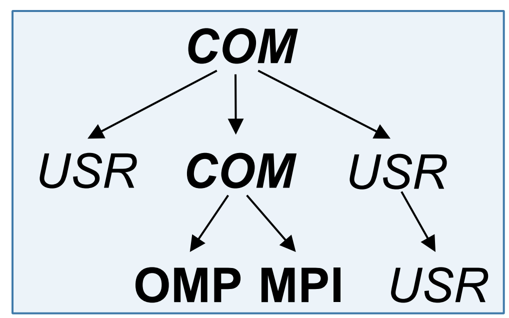
- Report scoring as textual output

1 GB of event trace
273 MB per rank!

```
% scorep-score scorep_bt-mz_W_4x4_sum/profile.cubex
Estimated aggregate size of event trace:                    1025MB
Estimated requirements for largest trace buffer (max_buf):  265MB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):        273MB

flt   type  max_buf[B]      visits time[s] time[%] time/visit[us]  region
       ALL 277,799,892 41,157,529  169.38   100.0            4.12  ALL
       USR 274,792,492 40,418,321   71.66    42.3            1.77  USR
       OMP   6,882,860    685,952   95.52    56.4          139.25  OMP
       COM     371,930     45,940    1.51     0.9           32.85  COM
       MPI     102,286      7,316    0.70     0.4           95.39  MPI
```

- 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)



NPB-MZ-MPI / BT Summary Analysis Report Breakdown

- Score report breakdown by region

```
% scorep-score -r scorep_bt-mz_W_4x4_sum/profile.cubex
 [...]
Flt type  max_buf[B]      visits time[s] time[%] t/v*[us]  region
     ALL 277,799,892 41,157,529  169.38   100.0     4.12  ALL
     USR 274,792,492 40,418,321   71.66    42.3     1.77  USR
     OMP   6,882,860    685,952   95.52    56.4   139.25  OMP
     COM     371,930     45,940    1.51     0.9    32.85  COM
     MPI     102,286      7,316    0.70     0.4    95.39  MPI

     USR  85,774,338 12,516,672   17.61    10.4     1.41  matmul_sub
     USR  85,774,338 12,516,672   19.71    11.6     1.57  matvec_sub
     USR  85,774,338 12,516,672   28.85    17.0     2.30  binvcrhs
     USR   7,974,876  1,170,624    1.86     1.1     1.59  binvrhs
     USR   7,974,876  1,170,624    2.94     1.7     2.52  lhsinit
     USR   3,473,912    526,848    0.67     0.4     1.28  exact_solution
     OMP     410,040     25,728    0.15     0.1     5.78  !$omp parallel
     OMP     410,040     25,728    0.15     0.1     5.83  !$omp parallel
     OMP     410,040     25,728    0.1      0.1     5.73  !$omp parallel
 [...]
```

More than 270 MB just for these 6 regions

42% of the total time, however, much of that is very likely measurement overhead due to short frequently called functions!

NPB-MZ-MPI / BT Summary Analysis Report Filtering

- Report scoring with prospective filter listing 6 USR regions

```
% cat ../config/scorep.filt
SCOREP_REGION_NAMES_BEGIN EXCLUDE
binvcrhs*
matmul_sub*
matvec_sub*
exact_solution*
binvrhs*
lhs*init*
timer_*

% scorep-score -f ../config/scorep.filt scorep_bt-mz_W_4x4_sum/profile.cubex
Estimated aggregate size of event trace:                   23MB
Estimated requirements for largest trace buffer (max_buf): 8MB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       16MB
(hint: When tracing set SCOREP_TOTAL_MEMORY=16MB to avoid intermediate
 flushes or reduce requirements using USR regions filters.)
```

23 MB of event trace, 16 MB per rank for measurement!

NPB-MZ-MPI / BT Summary Measurement Collection

- Generate an optimized profile with filter applied

```
% vi run_profile.sh
  export SCOREP_ENABLE_TRACING=false
  export SCOREP_ENABLE_PROFILING=true
  export SCOREP_TOTAL_MEMORY=100M
  export SCOREP_EXPERIMENT_DIRECTORY=scorep_bt-mz_W_4x4_sum_with_filter
  export SCOREP_FILTERING_FILE=../config/scorep.filt
  export OMP_NUM_THREADS=4
  runjob -n 4 -p 4 --block $COBALT_PARTNAME --env-all : bt-mz_W.4
% qsub -A <projid> -t 10 -n 1 --mode script run_profile.sh
% cat <jobid>.outpout
  NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark
  Number of zones:   4 x   4
    [...]
  Time step  200
  Verification Successful

  BT-MZ Benchmark Completed.
  Time in seconds = 3.58
```

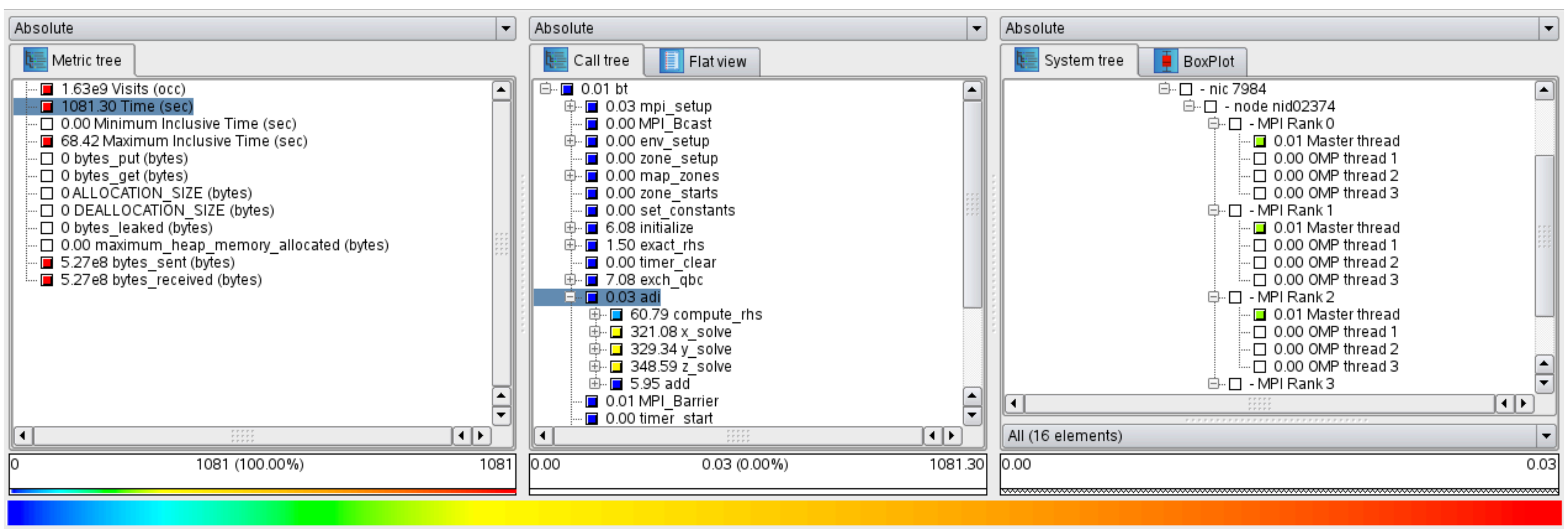
NPB-MZ-MPI / BT Profile Analysis

- Flat profile analysis with cube_stat:

```
$ cube_stat -t 20 -p scorep_bt-mz_W_4x4_sum_with_filter/profile.cubex
cube::Region              NumberOfCalls      ExclusiveTime  InclusiveTime
!$omp do @y_solve.f:52    12864.000000           10.843637      10.843637
!$omp do @z_solve.f:52    12864.000000           10.545983      10.545983
!$omp do @x_solve.f:54    12864.000000           9.567538        9.567538
...
```

- Call-path profile analysis with Cube:

```
$ cube scorep_bt-mz_W_4x4_sum_with_filter/profile.cubex
```



NPB-MZ-MPI / BT Trace Measurement Collection

- Perform measurement run with tracing enabled and filter applied

```
% cd bin.scorep
% less run_trace.sh
  export SCOREP_ENABLE_TRACING=true
  export SCOREP_ENABLE_PROFILING=false
  export SCOREP_FILTERING_FILE=../config/scorep.filt
  export SCOREP_TOTAL_MEMORY=100M
  export SCOREP_EXPERIMENT_DIRECTORY=scorep_bt-mz_W_4x4_trace
  export SCOREP_METRIC_PAPI=PAPI_FP_OPS,PAPI_L1_DCM
  export OMP_NUM_THREADS=4
  runjob -n 4 -p 4 --block $COBALT_PARTNAME --env-all : bt-mz_W.4
% qsub -A <projid> -t 10 -n 1 --mode script run_trace.sh
% cat <jobid>.output
  NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark
  Number of zones:   4 x   4
     [...]
  Time step  200
  Verification Successful

  BT-MZ Benchmark Completed.
  Time in seconds = 3.49
```

NPB-MZ-MPI / BT Interactive Trace Analysis with Vampir

- Download and install VampirClient for target platform

```
# Linux 64bit
$ scp <user>@mira.alcf.anl.gov:/soft/perftools/vampir/downloads/vampir*x86_64-setup.bin .
$ scp <user>@mira.alcf.anl.gov:/soft/perftools/vampir/license/vampir-remote.license .
$ bash ./vampir-*.bin
```

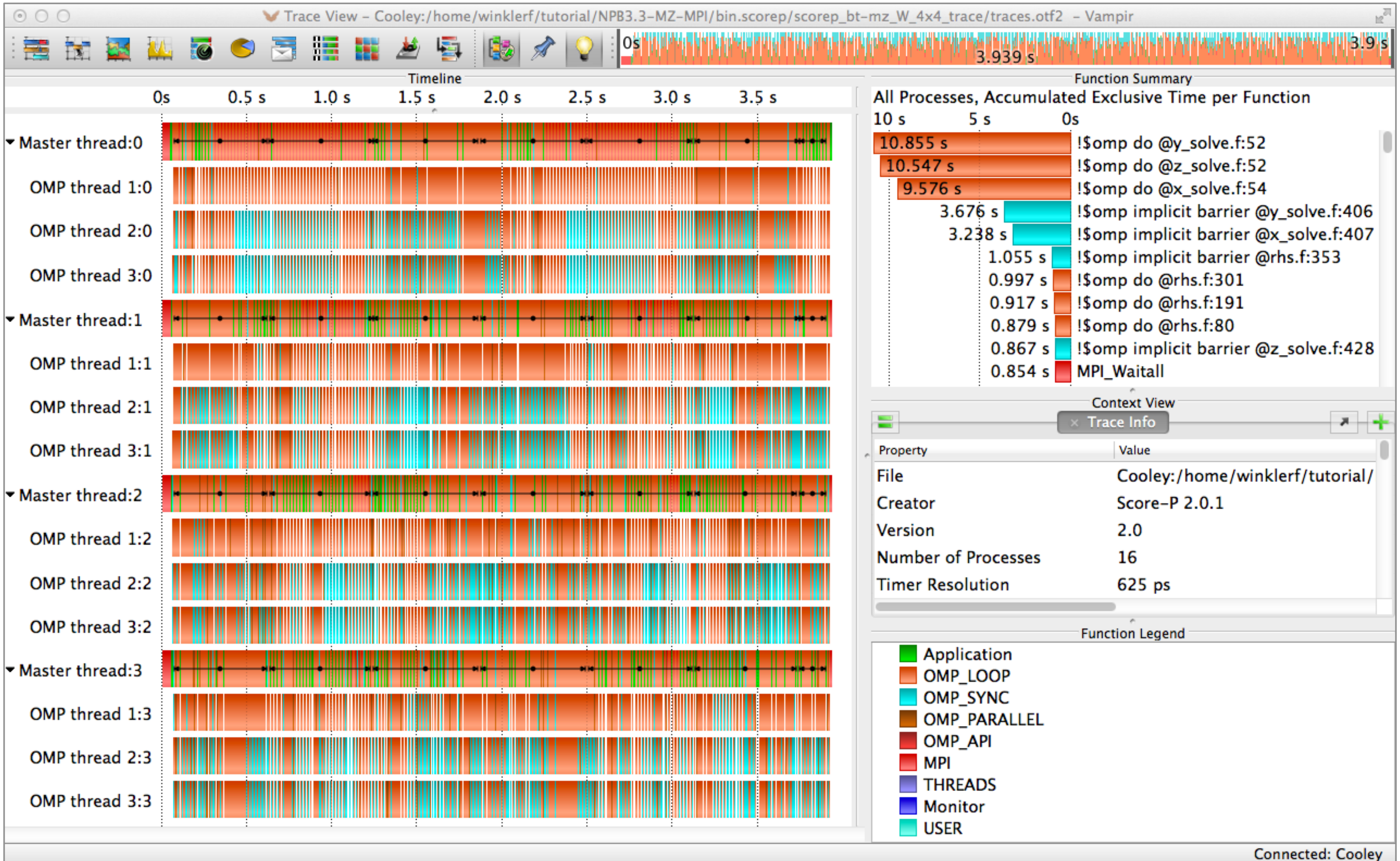
- Start VampirServer on Cooley and follow output instructions

```
$ vampirserver start -n 4 -- -A Tools -w 30
Launching VampirServer...
Submitting PBS batch job (this might take a while)...
** Project 'tools'; job rerouted to queue 'prod-short'
VampirServer 9.0.0  (r9950)
Licensed to Argonne NL
Running 3 analysis processes... (abort with vampirserver stop 23286)
VampirServer <23286> listens on: cc123:30097

Please run:
    ssh -L 30001:cc123:30097 <user>@cooley.alcf.anl.gov
on your desktop to create ssh tunnel to VampirServer.

Start vampir on your desktop and choose 'Open Other -> Remote File'
    Description: cooley,   Server: localhost,   Port: 30001
    Authentication: None
    Connection type: Socket
    Ignore "More Options"
```

NPB-MZ-MPI / BT Trace Analysis with Vampir



Agenda

Performance Analysis Approaches

- Sampling vs. Instrumentation
- Profiling vs. Tracing

Score-P: Scalable Performance Measurement Infrastructure for Parallel Codes

- Architecture
- Workflow
- Cube

Vampir: Event Trace Visualization

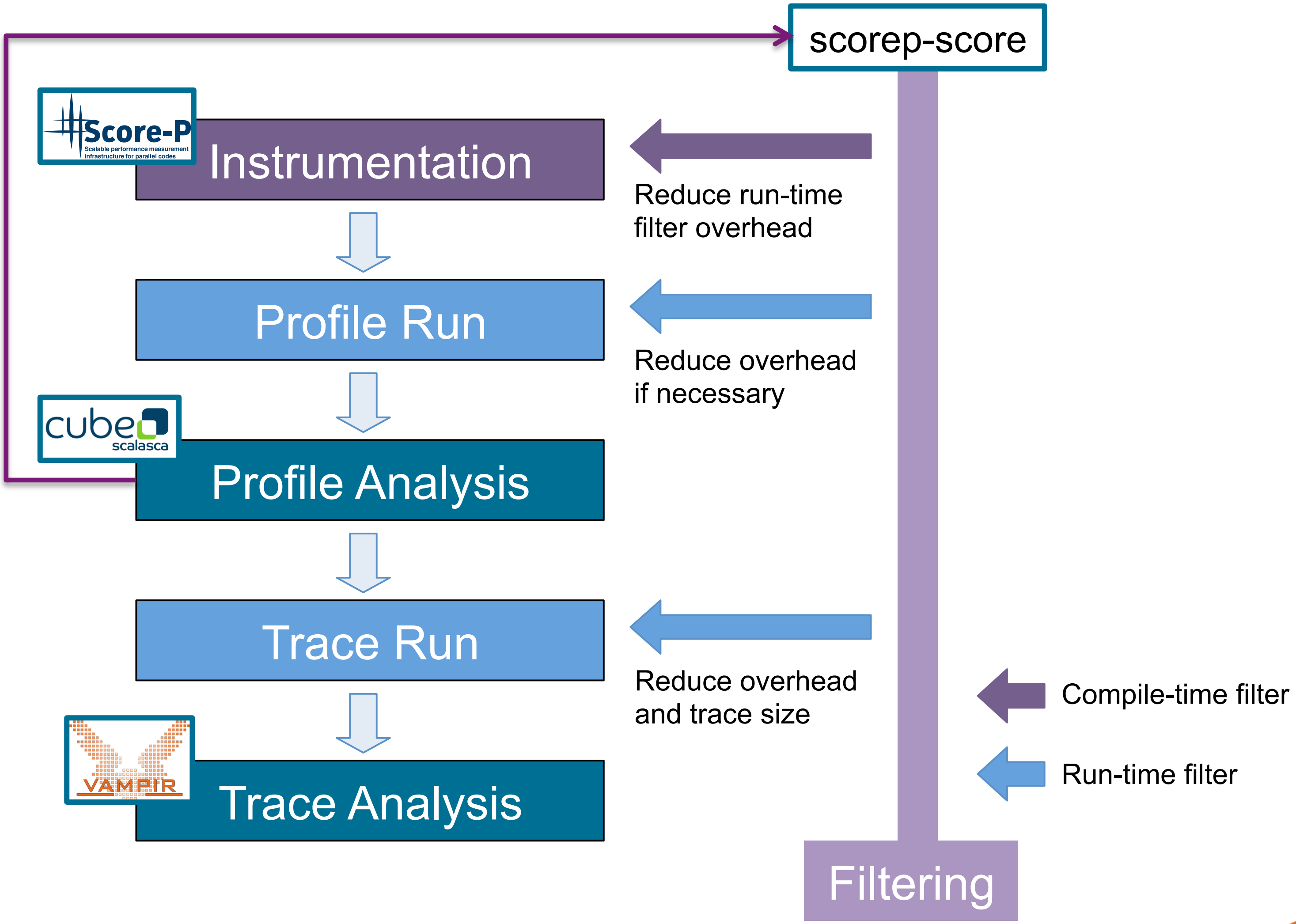
- Mission
- Visualization Modes
- Performance Charts

Demo

- Performance Analysis of NPB-MZ-MPI / BT on Mira

Conclusions

Conclusions: Score-P Workflow



Conclusions

Score-P

- Common instrumentation and measurement infrastructure for various analysis tools
- Hides away complicated details
- Provides many options and switches for experts

Vampir & VampirServer

- Interactive event trace visualization and analysis
- Intuitive browsing and zooming
- Scalable to large trace data sizes (20 TByte)
- Scalable to high parallelism (200000 processes)
- Vampir for Linux, Windows and Mac OS

Score-P is available at:

http://www.vi-hps.org/projects/score-p

Get support via support@score-p.org

Vampir is available at http://www.vampir.eu

Get support via vampirsupport@zih.tu-dresden.de

Score-P: Workflow / Advanced Instrumentation

- For CMake and autotools based build systems it is recommended to use the scorep-wrapper script instances

Disable instrumentation

```
#CMake

SCOREP_WRAPPER=OFF cmake .. \
   -DCMAKE_C_COMPILER=scorep-icc \
   -DCMAKE_CXX_COMPILER=scorep-icpc \
   -DCMAKE_Fortran_COMPILER=scorep-ifc
```

```
#Autotools

SCOREP_WRAPPER=OFF ../configure \
     CC=scorep-icc \
     CXX=scorep-icpc \
     FC=scorep-ifc \
     --disable-dependency-tracking
```

- Pass instrumentation and compiler flags at make

```
make SCOREP_WRAPPER_INSTRUMENTER_FLAGS="--user" \
     SCOREP_WRAPPER_COMPILER_FLAGS="-g -O2"
```

scorep --user <your_compiler> –g –O2

Score-P Advanced Features: Metrics

- Available PAPI metrics
  - Preset events: common set of events deemed relevant and useful for application performance tuning

```
$ papi_avail
```

  - Native events: set of all events that are available on the CPU (platform dependent)

```
$ papi_native_avail
```

- Available resource usage metrics

```
$ man getrusage
    [... Output ...]

  struct rusage {
    struct timeval ru_utime; /* user CPU time used */
    struct timeval ru_stime; /* system CPU time used */
    [... More output ...]
```

Score-P Advanced Features: Metrics (2)

- Recording hardware counters via PAPI

```
$ export SCOREP_METRIC_PAPI=PAPI_TOT_INS,PAPI_FP_INS
```

- Recording operating system resource usage

```
$ export SCOREP_METRIC_RUSAGE=ru_maxrss,ru_stime
```

Score-P Advanced Features: Sampling

- Alternative to compiler instrumentation to generate profiles or traces
- Regulate the trade-off between overhead and correctness
- Libunwind/1.1 to capture current stack
- Sampling interrupt sources:
  - Interval timer
  - PAPI
  - Perf
- Example for enabling sampling for measurement run:

```
$ export SCOREP_ENABLE_UNWINDING=true
$ export SCOREP_SAMPLING_EVENTS=PAPI_TOT_CYC@1000000
```

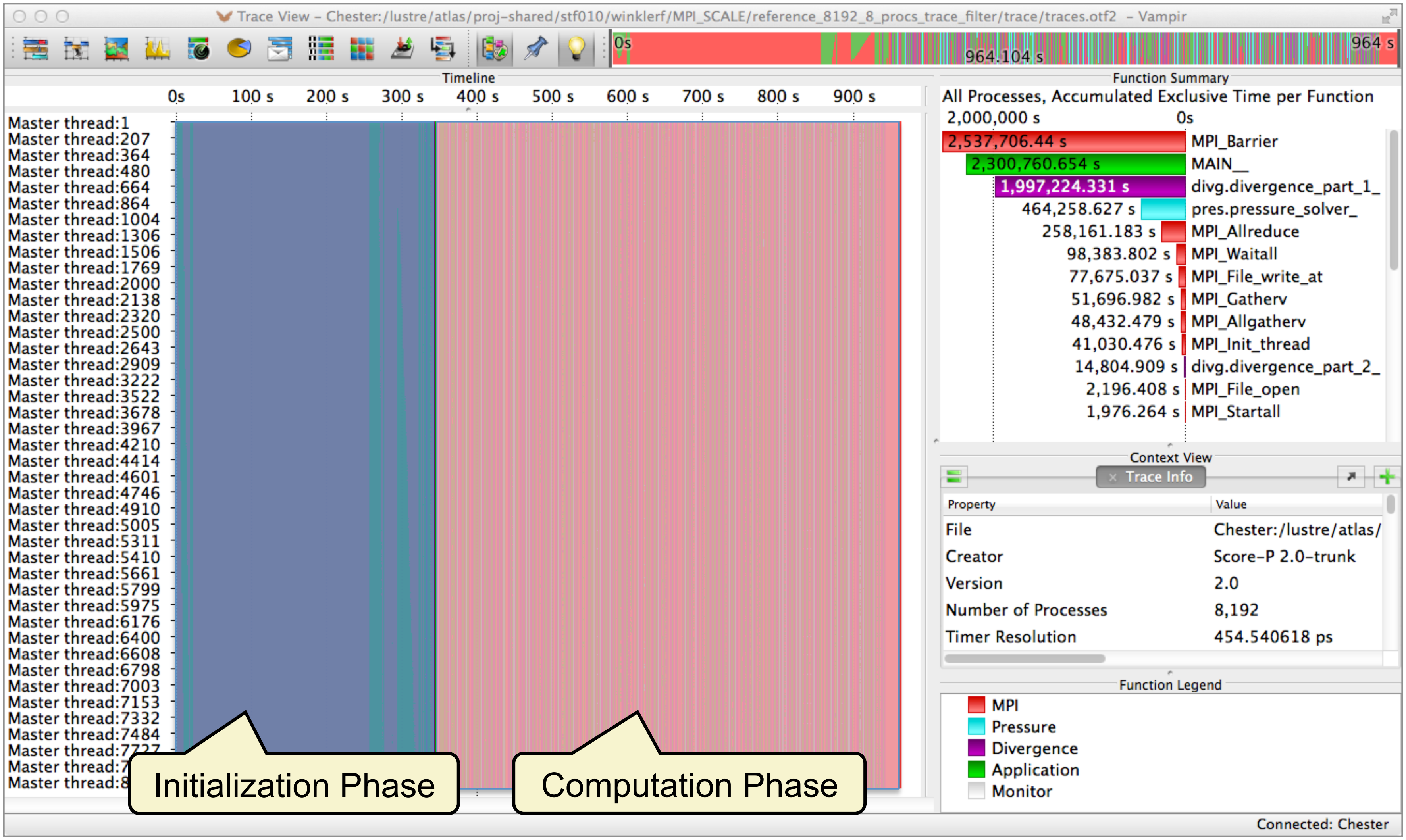
Score-P Advanced Features: Memory Recording

- Memory (de)allocations are recorded via the libc/C++ API
- Recording of memory location's call-site in sampling mode
  - Debugging symbols required (-g)
- Interplay of memory usage and application's execution
  - CUBE: (De)allocation size, maximum heap memory, leaked bytes
  - Vampir: Memory usage in "Counter Timelines"
- Enabling memory recording for measurement run:

```
$ export SCOREP_MEMORY_RECORDING=true
```

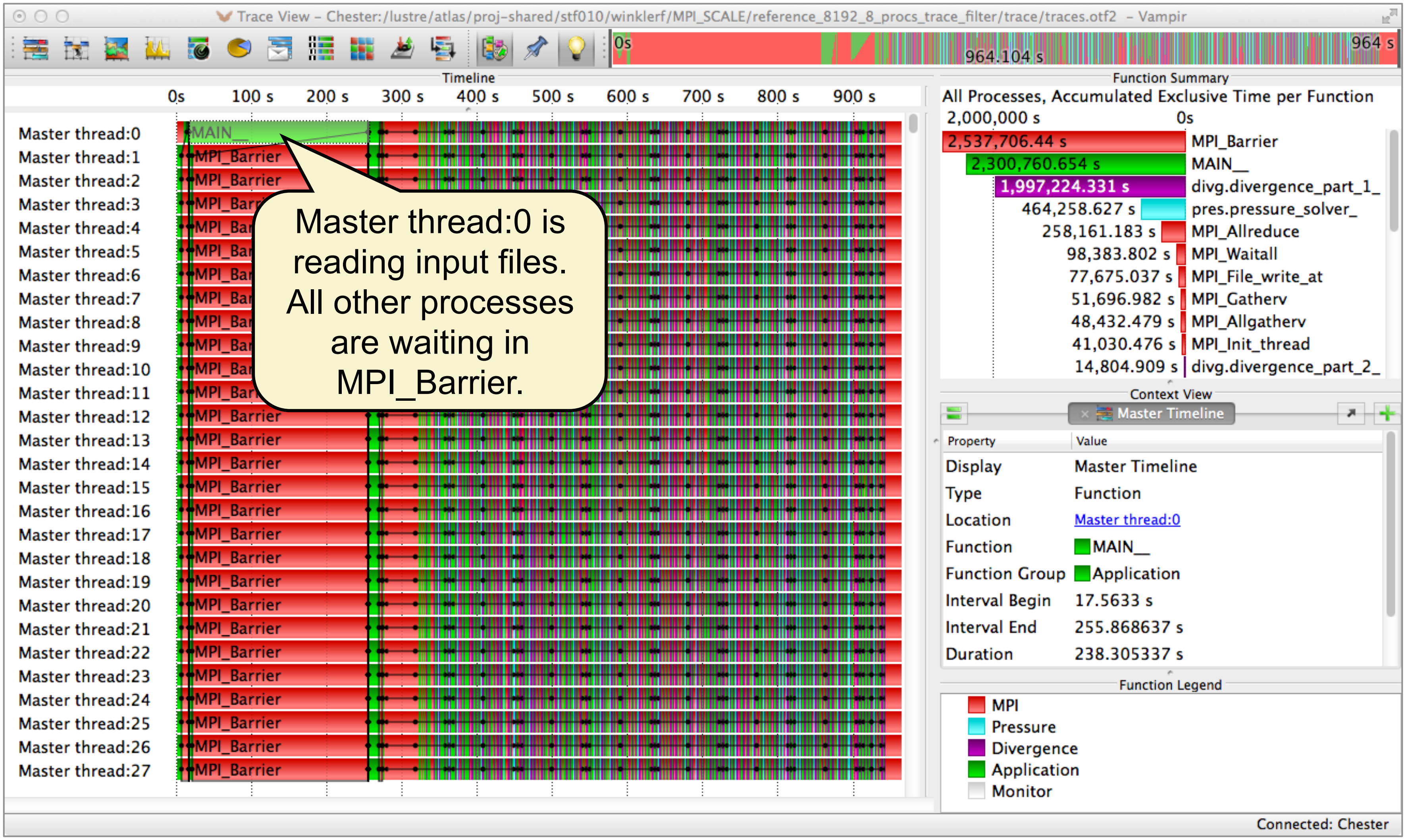
Vampir Bonus: Case Study of FDS

- Indentification of program phases



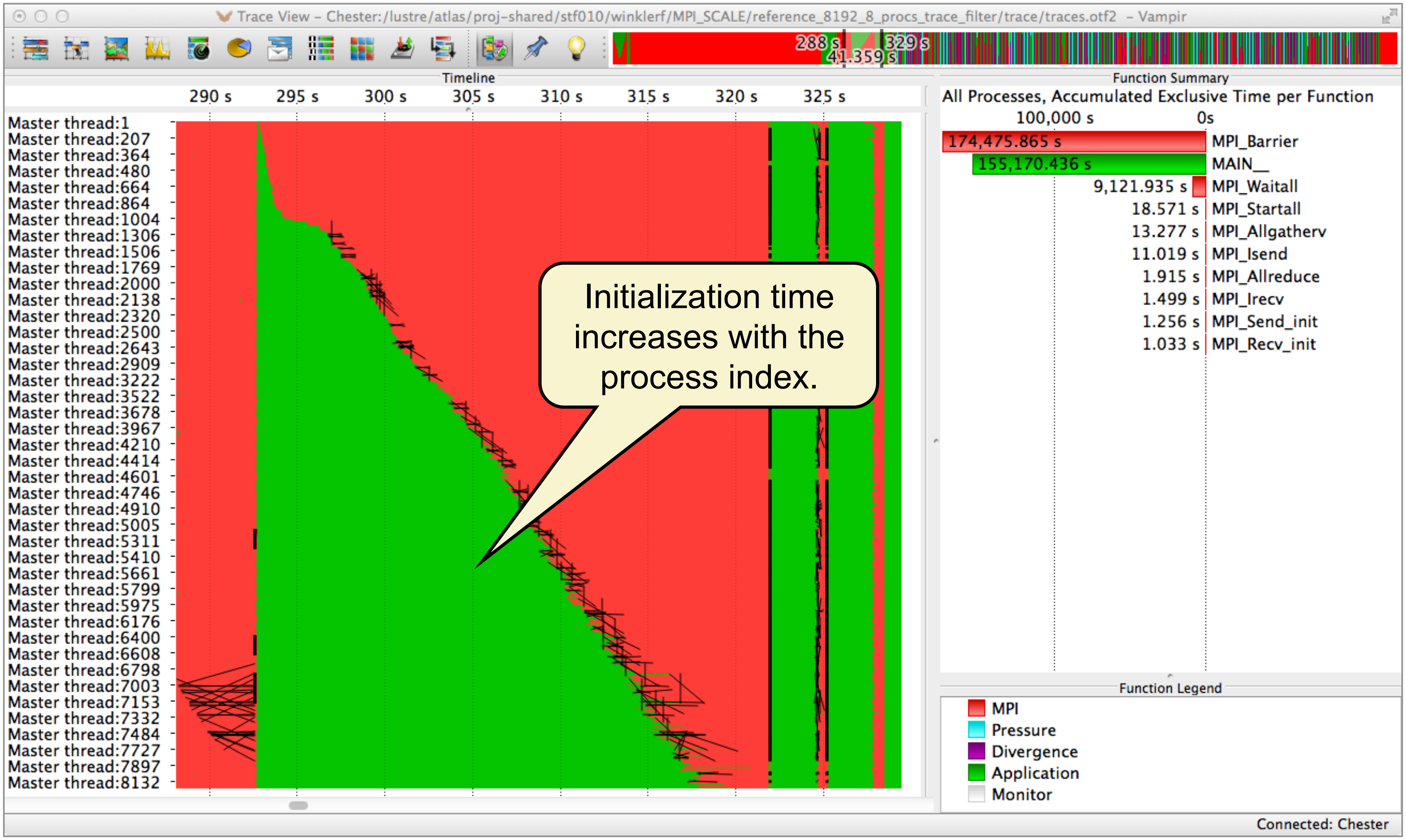
Vampir Bonus: Case Study of FDS

- Load imbalance in initialization phase



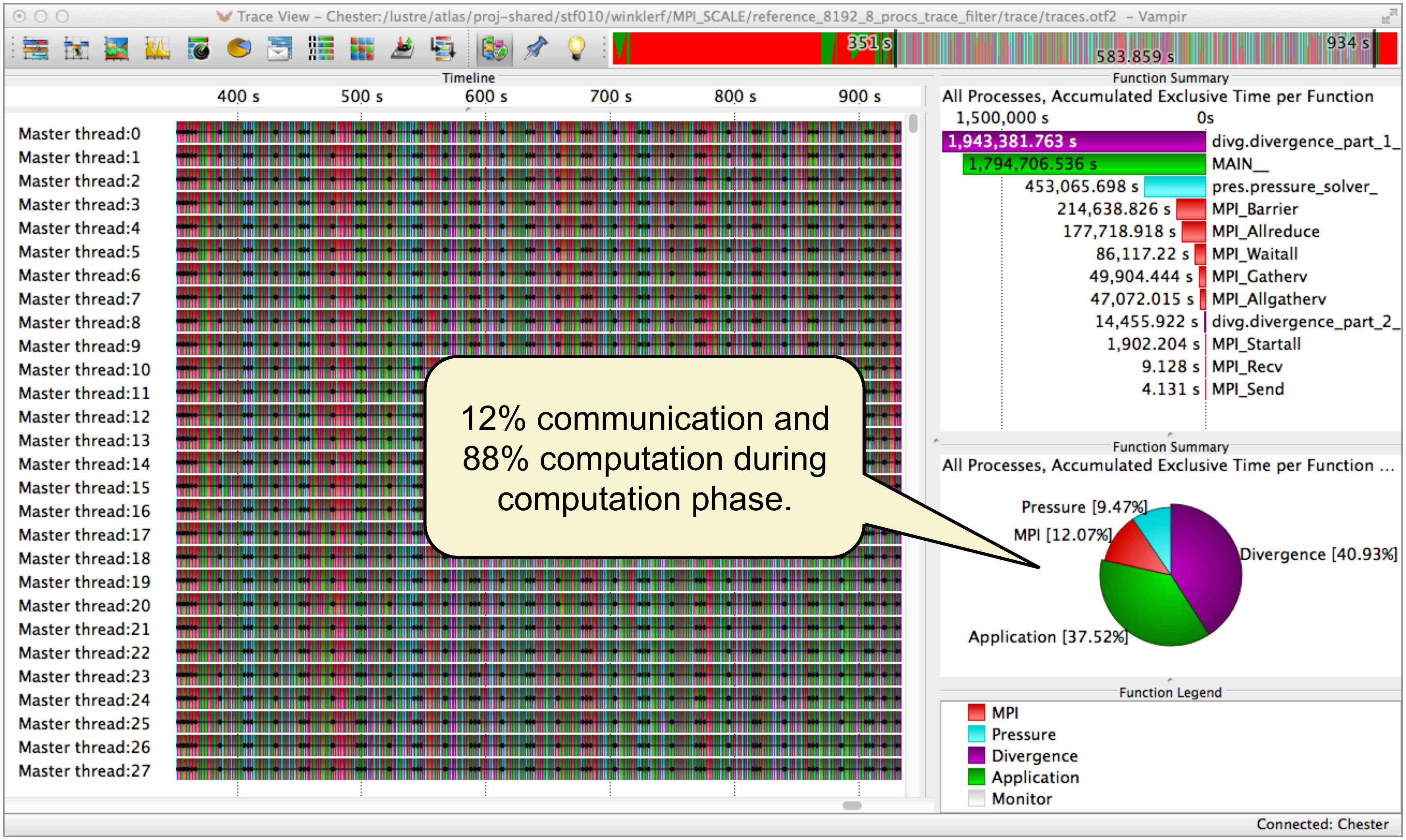
Vampir Bonus: Case Study of FDS

- Load imbalance in initialization phase (2)



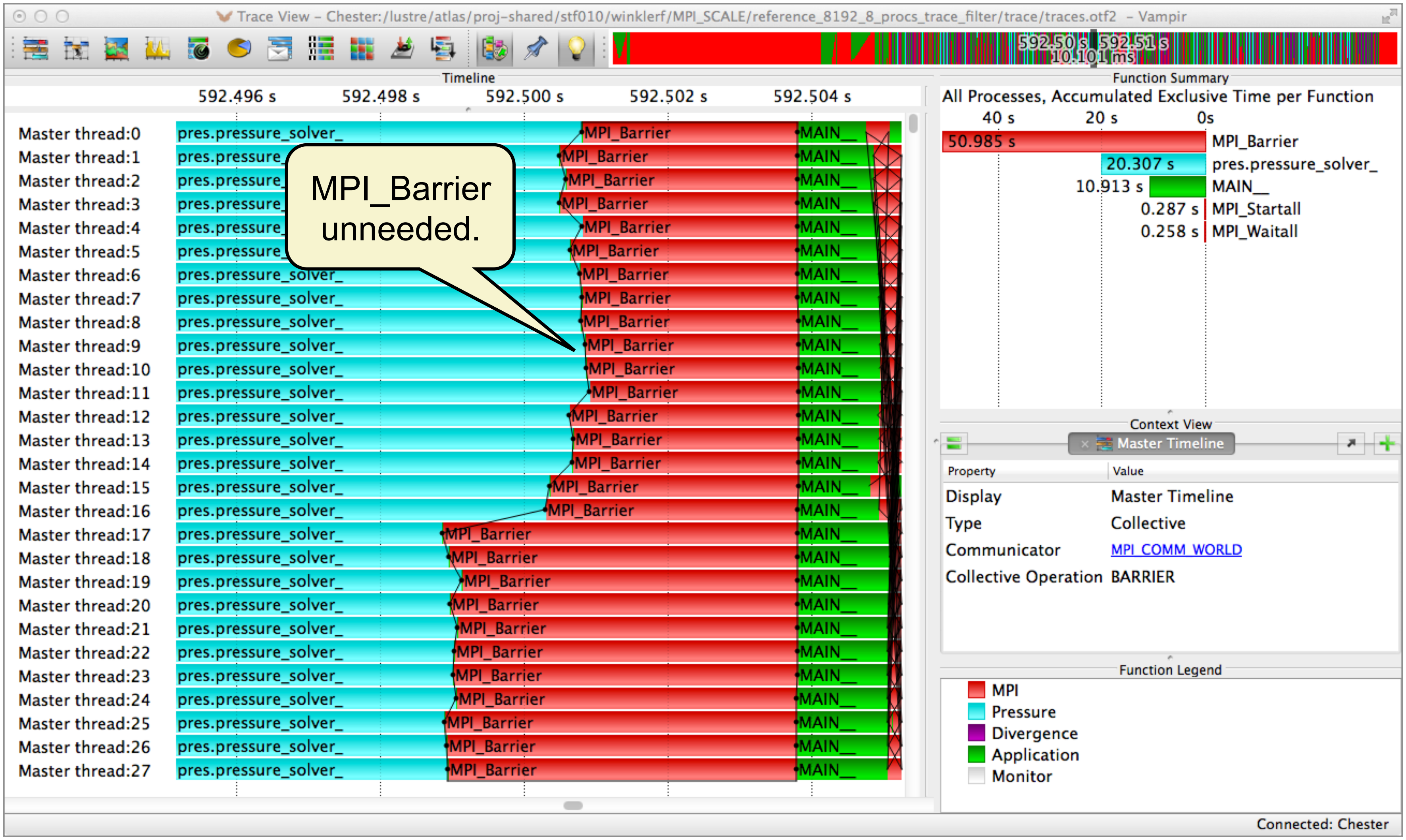
Vampir Bonus: Case Study of FDS

- Computation phase



Vampir Bonus: Case Study of FDS

- Unnecessary synchronization in computation phase



Vampir Bonus: Case Study of FDS

- Inefficient cache usage in computation phase