

# Hands-on: *Archer Cray XC30* NPB-MZ-MPI / bt-mz\_C.8

---

VI-HPS Team

# Tutorial exercise objectives

---

- Familiarise with usage of VI-HPS tools
  - complementary tools' capabilities & interoperability
- Prepare to apply tools productively to *your* applications(s)
- Exercise is based on a small portable benchmark code
  - unlikely to have significant optimisation opportunities
- Optional (recommended) exercise extensions
  - analyse performance of alternative configurations
  - investigate effectiveness of system-specific compiler/MPI optimisations and/or placement/binding/affinity capabilities
  - investigate scalability and analyse scalability limiters
  - compare performance on different HPC platforms
  - ...

## Compiler and MPI modules (Archer Cray XC30)

---

- Select appropriate PrgEnv: GNU is recommended for tutorial

```
% module switch PrgEnv-cray PrgEnv-gnu
```

- Set-up and load the required modules

```
% module use /home/y07/y07/scalasca/modules  
% module load scalasca  
% module load must
```

- Copy tutorial sources to your \$WORK directory

```
% cd $WORK  
% tar zxvf /work/y14/shared/tutorial/NPB3.3-MZ-MPI.tar.gz  
% cd NPB3.3-MZ-MPI
```

## NPB-MZ-MPI Suite

---

- 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

```
% 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 is ready to “make” one or more of the benchmarks and install them into a (tool-specific) “bin” subdirectory

# Building an NPB-MZ-MPI Benchmark

```
% 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=C NPROCS=8 *
*****
```

- Type "make" for instructions

# Building an NPB-MZ-MPI Benchmark

```
% make bt-mz CLASS=C NPROCS=8
make[1]: Entering directory `BT-MZ'
make[2]: Entering directory `sys'
cc -o setparams setparams.c -lm
make[2]: Leaving directory `sys'
../sys/setparams bt-mz 8 C
make[2]: Entering directory `../BT-MZ'
ftn -c -O3 -fopenmp      bt.f
[...]
ftn -c -O3 -fopenmp      mpi_setup.f
cd ../common; ftn -c -O3 -openmp      print_results.f
cd ../common; ftn -c -O3 -openmp      timers.f
ftn -O3 -fopenmp -o ../bin/bt-mz_C.8 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
make[2]: Leaving directory `BT-MZ'
Built executable ../bin/bt-mz_C.8
make[1]: Leaving directory `BT-MZ'
```

- Specify the benchmark configuration
  - benchmark name: **bt-mz**, lu-mz, sp-mz
  - the number of MPI processes: **NPROCS=8**
  - the benchmark class (S, W, A, B, C, D, E): **CLASS=C**

Shortcut: `% make suite`

## NPB-MZ-MPI / BT (Block Tridiagonal Solver)

---

- What does it do?
  - Solves a discretized version of the 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
  - 8 processes each with 6 threads should be reasonable for 2 compute nodes of Archer
  - bt-mz\_B.8 should run in around 10 seconds
  - bt-mz\_C.8 should run in around 30 seconds



## NPB-MZ-MPI / BT Reference Execution

```
% cd bin
% cp ../jobscript/archer/run.pbs .
% less run.pbs
% qsub run.pbs

% cat run_mzmpibt.o<job_id>
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:      8
Total number of threads:      48 ( 6.0 threads/process)

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

BT-MZ Benchmark Completed.
Time in seconds = 28.78
```

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

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



## Tutorial Exercise Steps

---

- Edit [config/make.def](#) to adjust build configuration
  - Modify specification of compiler/linker: [MPIF77](#)
- Make clean and then build new tool-specific executable

```
% make clean
% make bt-mz CLASS=C NPROCS=8
Built executable ../bin.scorep/bt-mz_C.8
```

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

```
% cd bin.scorep
% cp ../jobscript/archer/scorep.pbs .
% qsub scorep.pbs
```

## NPB-MZ-MPI / BT: config/make.def

```
#           SITE- AND/OR PLATFORM-SPECIFIC DEFINITIONS.
#
#-----
#-----
# Configured for Cray with PrgEnv compiler-specific OpenMP flags
#-----
#COMPILER = -homp           # Cray/CCE compiler
COMPILER = -fopenmp        # GCC compiler
#COMPILER = -openmp        # Intel compiler

...
#-----
# The Fortran compiler used for MPI programs
#-----
MPIF77 = ftn

# Alternative variant to perform instrumentation
#MPIF77 = scorep --user ftn

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

```

Default (no instrumentation)

Hint: uncomment a compiler wrapper to do instrumentation

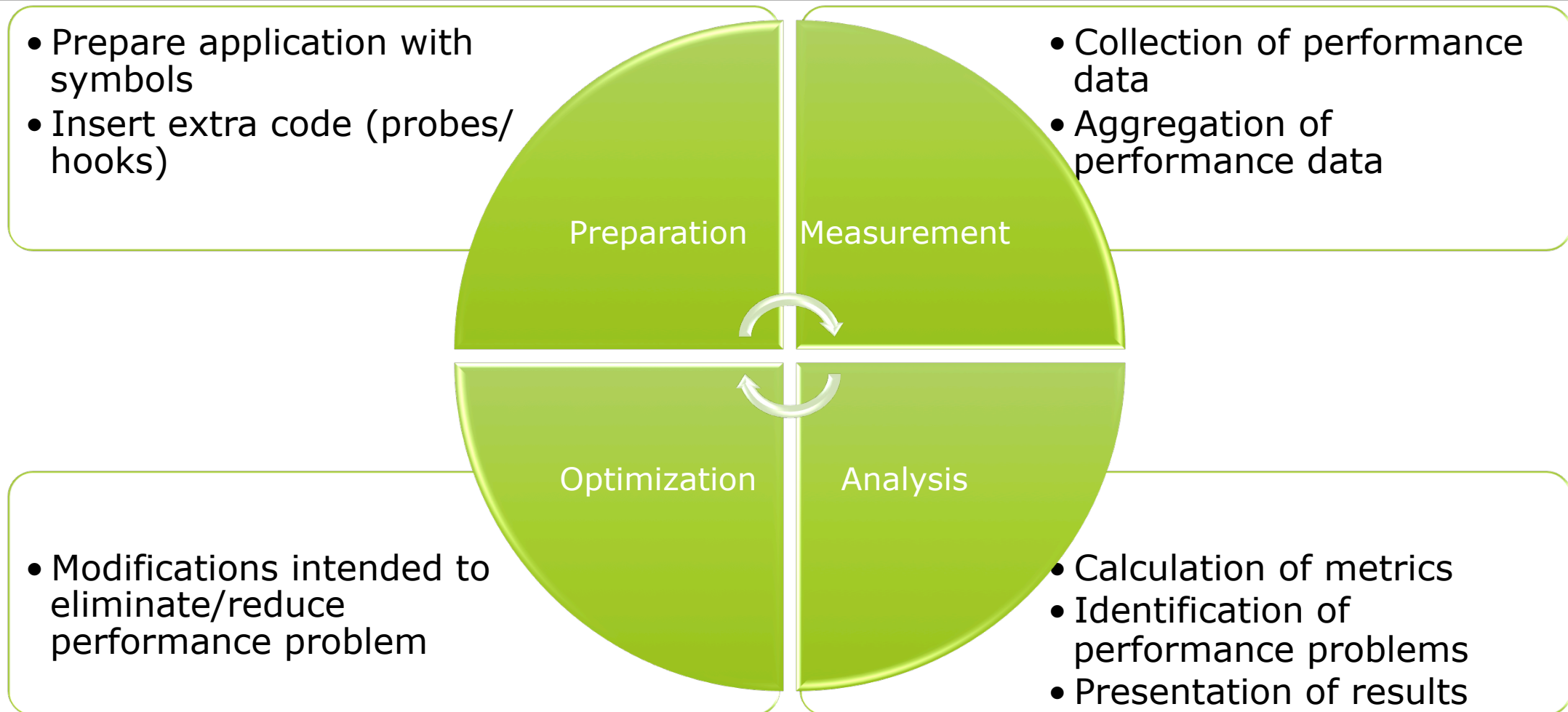
# Hands-On Exercise: Measuring Application Performance with Score-P

---

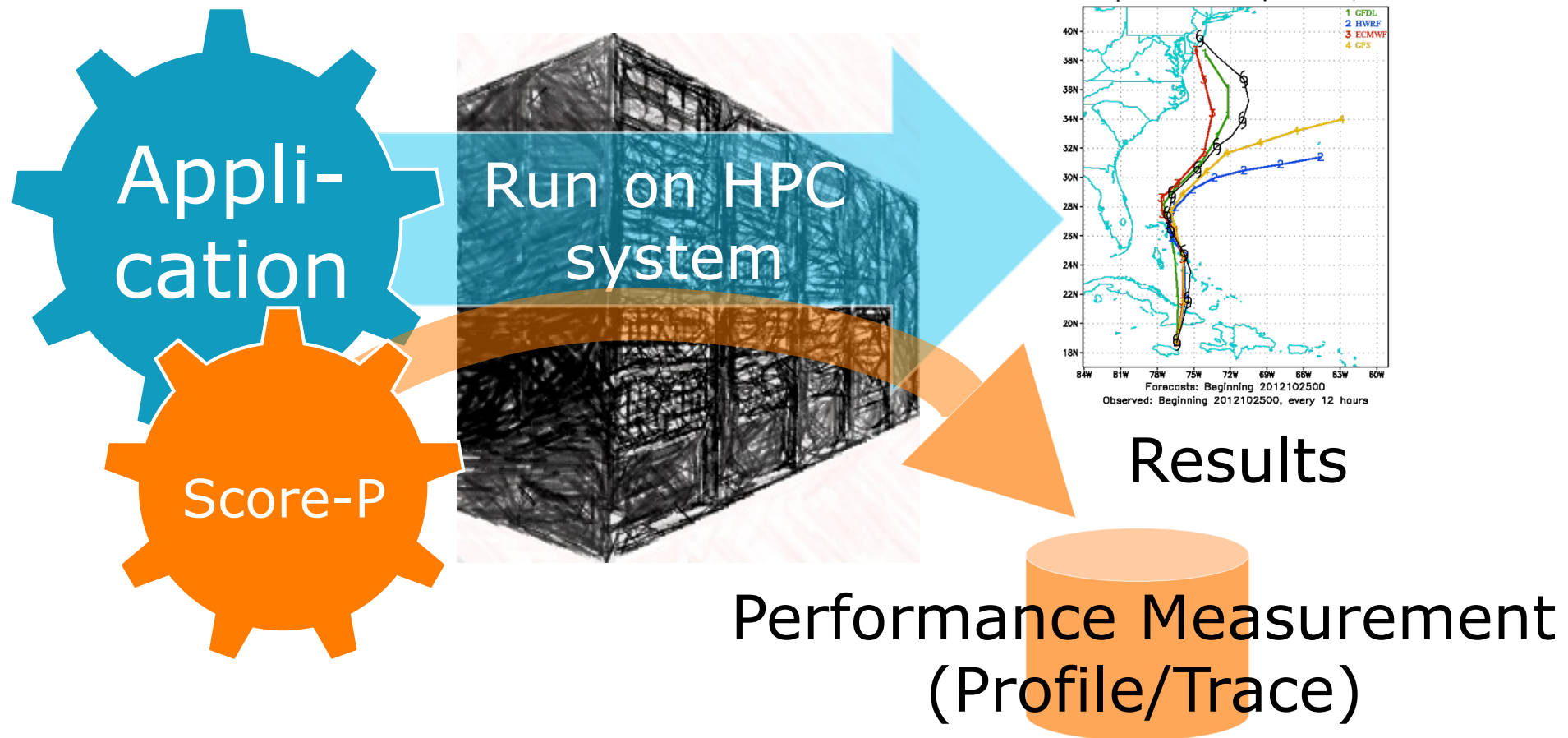
VI-HPS Team



# Performance engineering workflow



# Runtime Performance Measurement



## Fragmentation of Tools Landscape

---

- 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 Project Idea

- A community effort for a common infrastructure
- Developer perspective:
  - Save manpower by sharing development resources
  - Save efforts for maintenance, testing, porting, support, training
- User perspective:
  - Single learning curve
  - Single installation, fewer version updates
  - Interoperability and data exchange



UNIVERSITY OF OREGON

Vampir

Scalasca

TAU

Periscope

Score-P



## Score-P Functionality

---

- Provide typical functionality for HPC performance tools
- Support all fundamental concepts of partner's tools
  
- Instrumentation (various methods)
- Flexible measurement without re-compilation:
  - Basic and advanced profile generation
  - Event trace recording
  - Online access to profiling data
  
- MPI/SHMEM, OpenMP/Pthreads, and hybrid parallelism (and serial)
- Enhanced functionality (CUDA, OpenCL, highly scalable I/O)

## Hands-on: NPB-MZ-MPI / BT

---



# Performance Analysis Steps

---

- 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

# NPB-MZ-MPI / BT Instrumentation – Make the tools available

---

## ▪ COSMA

```
% module switch \  
  intel_comp intel_comp/c4/2015  
% module load scalasca \  
  scorep intel_mpi  
% cd <...>/NPB3.3-MZ-MPI
```

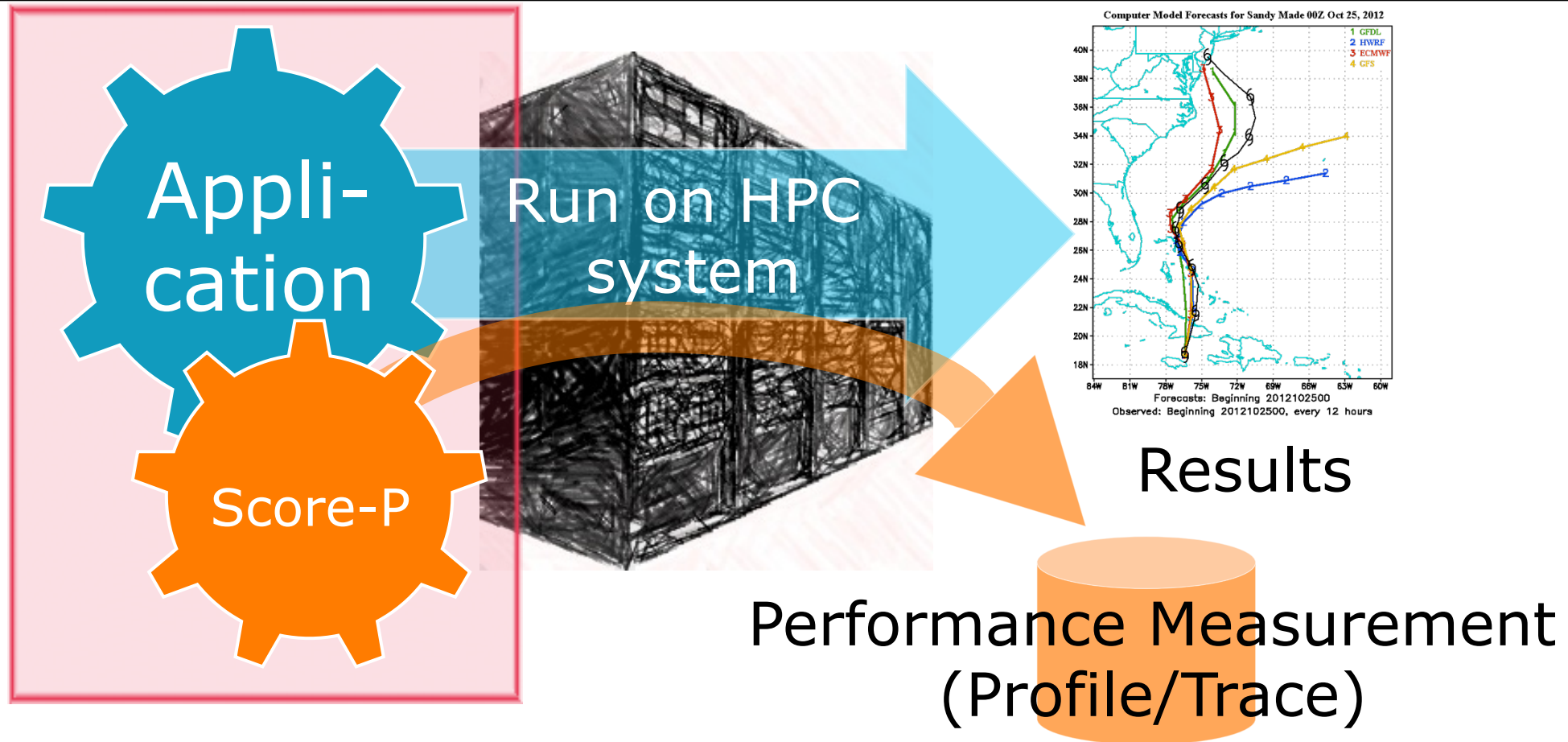
## ▪ Hamilton

```
% module load scalasca  
% cd <...>/NPB3.3-MZ-MPI
```

## ▪ Archer

```
% module use \  
  /home/y07/y07/scalasca/modules  
% module switch \  
  PrgEnv-cray PrgEnv-gnu  
% module load scalasca  
% cd <...>/NPB3.3-MZ-MPI
```

## Overview – Next: Attach Score-P



# NPB-MZ-MPI / BT Instrumentation – Link the tool to the application

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

## ▪ COSMA and Hamilton

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

# Alternative variants to perform instrum.
...
MPIF77 = scorep --user mpiifort
...
```

## ▪ Archer

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

# Alternative variants to perform instrum.
...
MPIF77 = scorep --user ftn
...
```

Uncomment and adapt  
Score-P compiler  
wrapper specification



## NPB-MZ-MPI / BT Instrumented – Build with presence of Score-P

If you run on the frontend of COSMA/  
Hamilton, use “B” and 4 procs!

```
% make clean

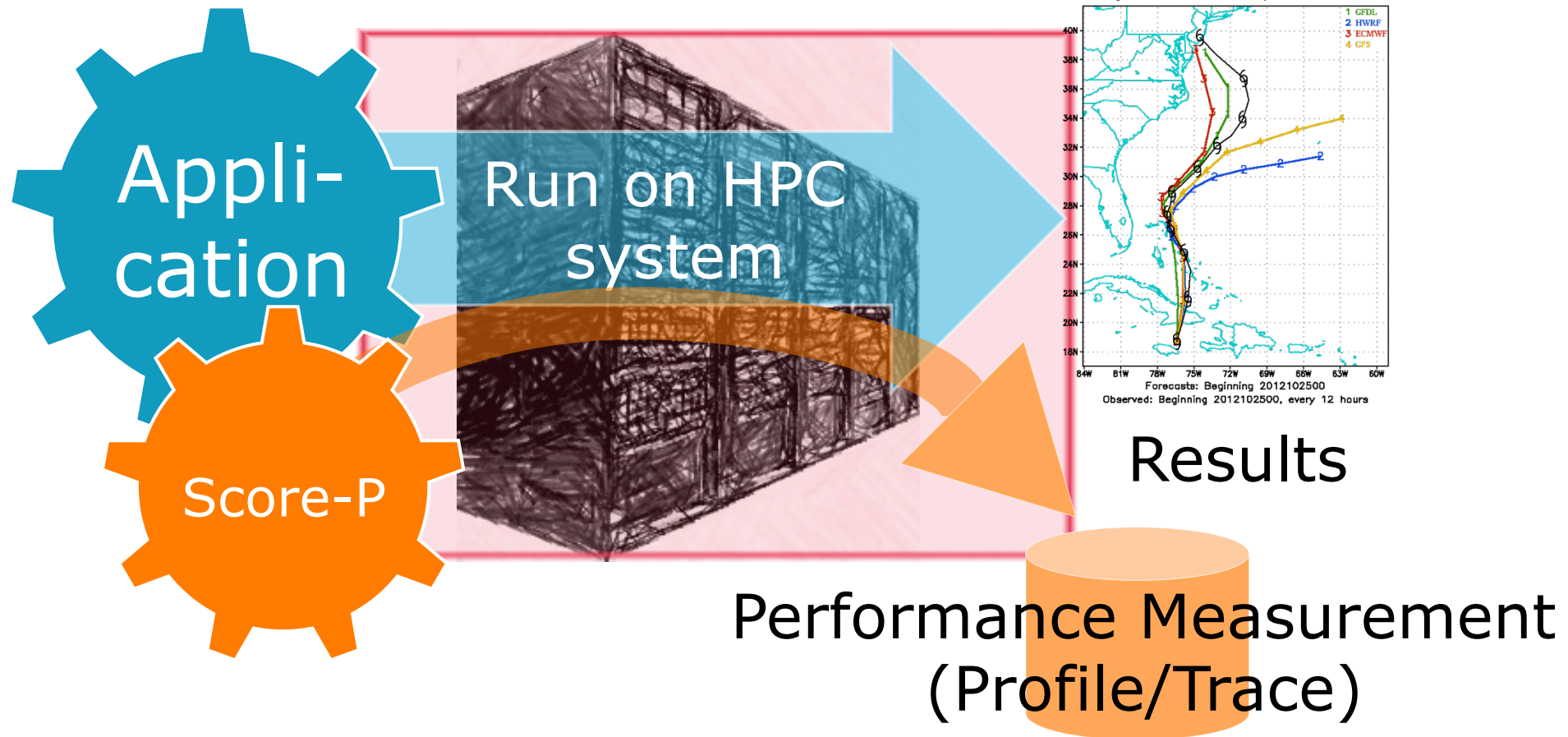
% make bt-mz CLASS=C NPROCS=8
cd BT-MZ; make CLASS=C NPROCS=8 VERSION=
make: Entering directory 'BT-MZ'
cd ../sys; cc -o setparams setparams.c -lm
../sys/setparams bt-mz 8 C
scorep mpiifort -c -O3 -openmp bt.f
[...]
```

```
cd ../common; scorep mpiifort -c -O3 -openmp timers.f
scorep mpiifort -O3 -fopenmp -o ../bin.scorep/bt-mz_C.8 \
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_C.8
make: Leaving directory 'BT-MZ'
```

- Clean-up
- Re-build executable with NPB build system (this is unrelated to Score-P and simply part of the NPB benchmarks)



## Overview – Next: Run with Score-P attached (Initial run)



# Measurement Configuration: scorep-info

---

```
% scorep-info config-vars --full
SCOREP_ENABLE_PROFILING
  Description: Enable profiling
  [...]
SCOREP_ENABLE_TRACING
  Description: Enable tracing
  [...]
SCOREP_TOTAL_MEMORY
  Description: Total memory in bytes for the measurement system
  [...]
SCOREP_EXPERIMENT_DIRECTORY
  Description: Name of the experiment directory
  [...]
SCOREP_FILTERING_FILE
  Description: A file name which contain the filter rules
  [...]
SCOREP_METRIC_PAPI
  Description: PAPI metric names to measure
  [...]
SCOREP_METRIC_RUSAGE
  Description: Resource usage metric names to measure
  [...]
  [...] More configuration variables ...]
```

- Score-P measurements are configured via environmental variables:

## Summary Measurement Collection – First execution with Score-P

- Change to the directory containing the new executable (bin.scorep)
- COSMA and Hamilton
- Archer

```
% cd bin.scorep
% export OMP_NUM_THREADS=4
% export SCOREP_EXPERIMENT_DIRECTORY=\
    scorep_4x4_sum
% mpirun -np 4 ./bt-mz_B.4
```

Runs directly on frontend – Use a jobscript if you have access to quick to react queues  
Example jobscripts available in:  
`../jobscripts/{cosma/hamilton}/`

```
% cd bin.scorep
% cp ../jobscript/archer/scorep.pbs ./
% nano scorep.pbs
...
#PBS -A y14
...
export OMP_NUM_THREADS=6
PROCS=8
CLASS=C
EXE=./bt-mz_${CLASS}.${PROCS}
export SCOREP_EXPERIMENT_DIRECTORY=\
scorep_${NPROCS}x${OMP_NUM_THREADS}_sum
#export SCOREP_FILTERING_FILE=../config/scorep.filt
#export SCOREP_METRIC_PAPI=PAPI_TOT_INS,PAPI_TOT_CYC
#export SCOREP_TOTAL_MEMORY=300M
...
% qsub -q short scorep.pbs
```

Adapt!

Keep them commented

# Summary Measurement Collection – First execution with Score-P

---

```
% less <Jobscript/Shell-Output>

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:      8

Use the default load factors with threads
Total number of threads:      32  (  4.0 threads/process)

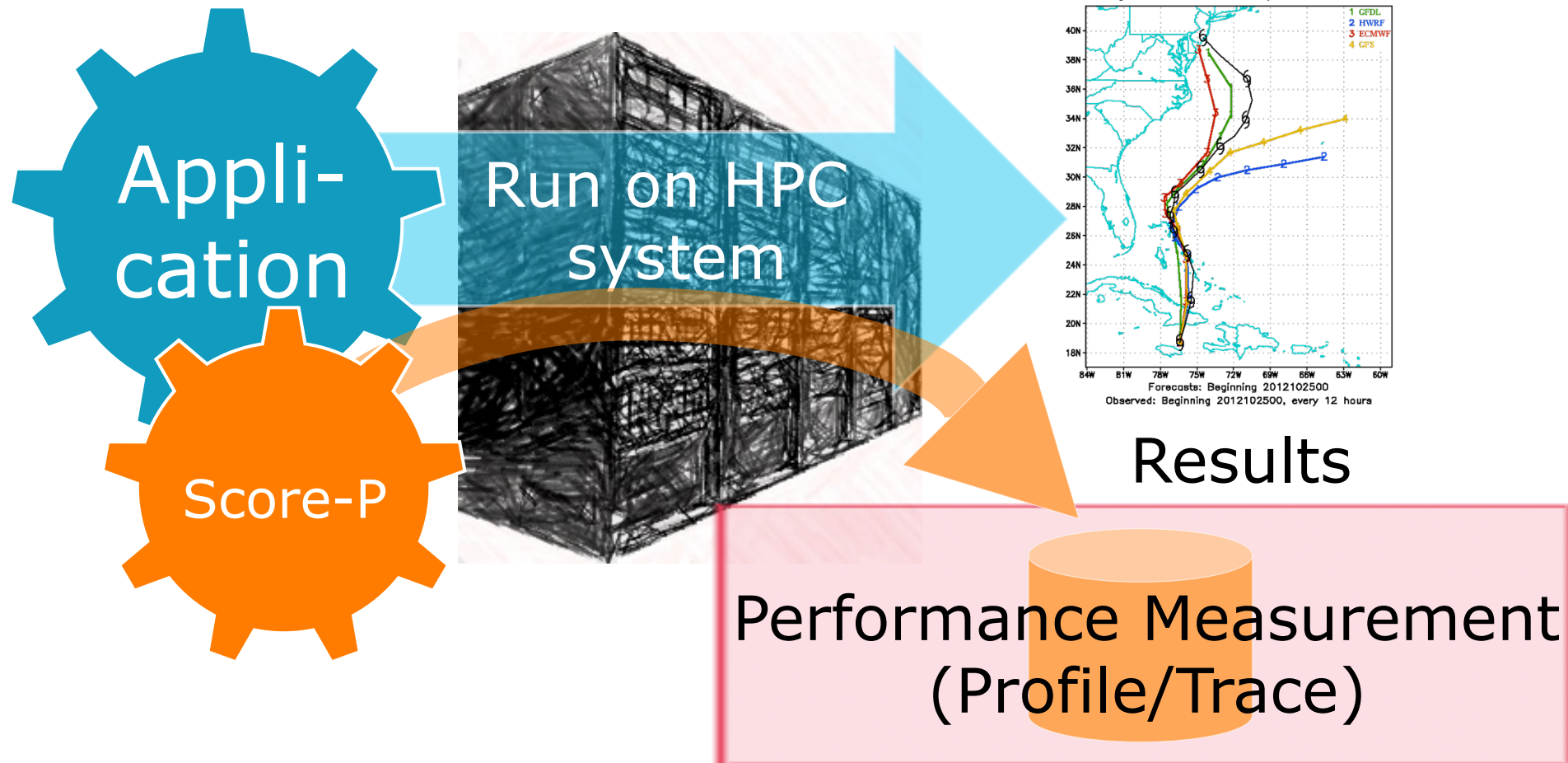
Calculated speedup =      31.99

Time step      1

[... More application output ...]
```

- Check the output of the application run

## Overview – Next: Run with Score-P attached (Initial run)



# BT-MZ Summary Analysis Report Examination

or 4x4

```
% ls
bt-mz_C.8  mzmplibt.o2969889  scorep_8x6_sum
% ls scorep_8x6_sum
profile.cubex  scorep.cfg

% cube scorep_8x6_sum/profile.cubex

[CUBE GUI showing summary analysis report]
```

- Creates experiment directory
  - A record of the measurement configuration (scorep.cfg)
  - The analysis report that was collated after measurement (profile.cubex)
- Interactive exploration with CUBE



# Congratulations!?

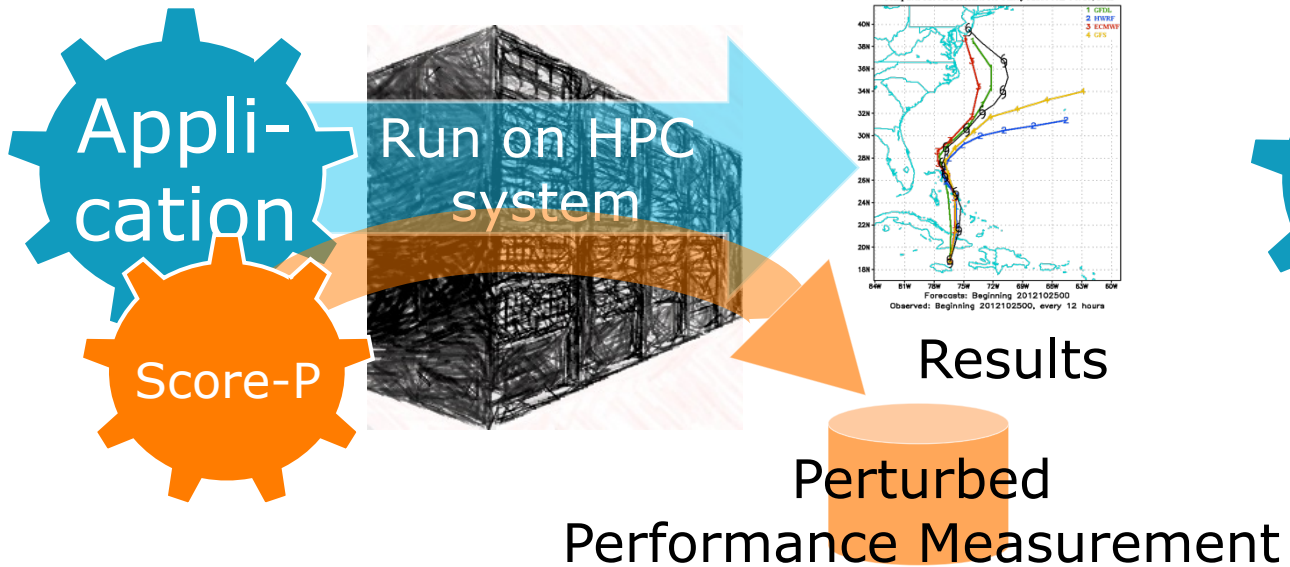
---

- If you made it this far, you successfully used Score-P to
  - instrument the application
  - analyze its execution with a summary measurement, and
  - examine it with one the interactive analysis report explorer GUIs
- ... revealing the call-path profile annotated with
  - the “Time” metric
  - Visit counts
  - MPI message statistics (bytes sent/received)
- ... 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

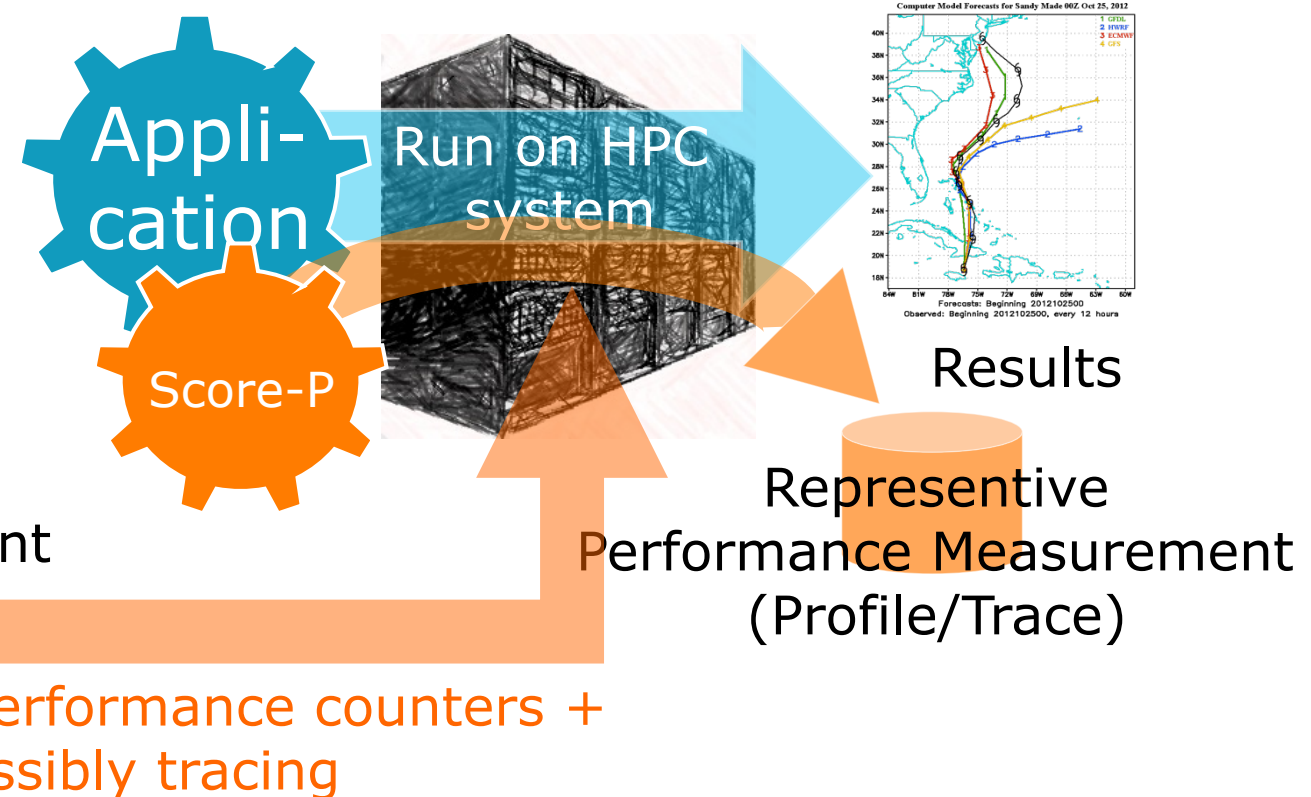


## Overview – Next: Filtering

### ▪ First profiling run



### ▪ Second filtered run (possibly tracing)



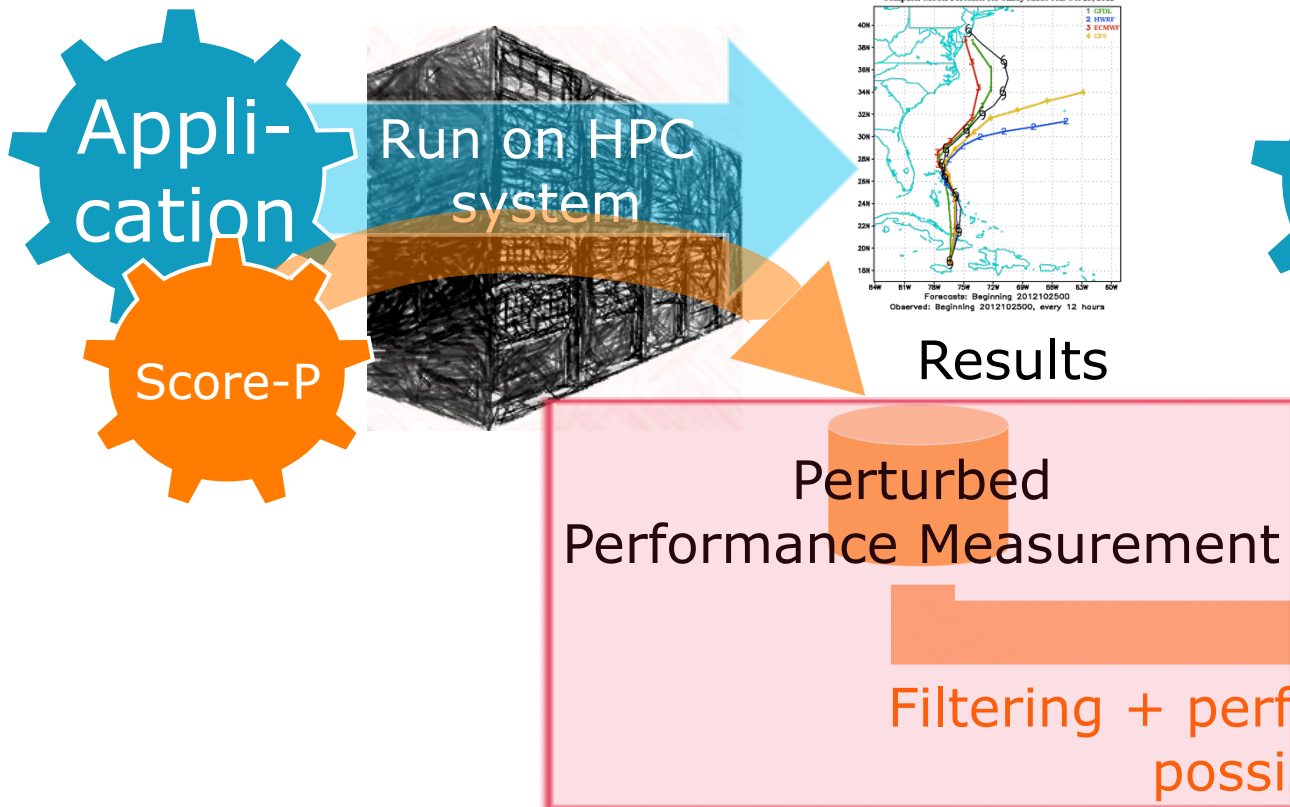
# Performance Analysis Steps

---

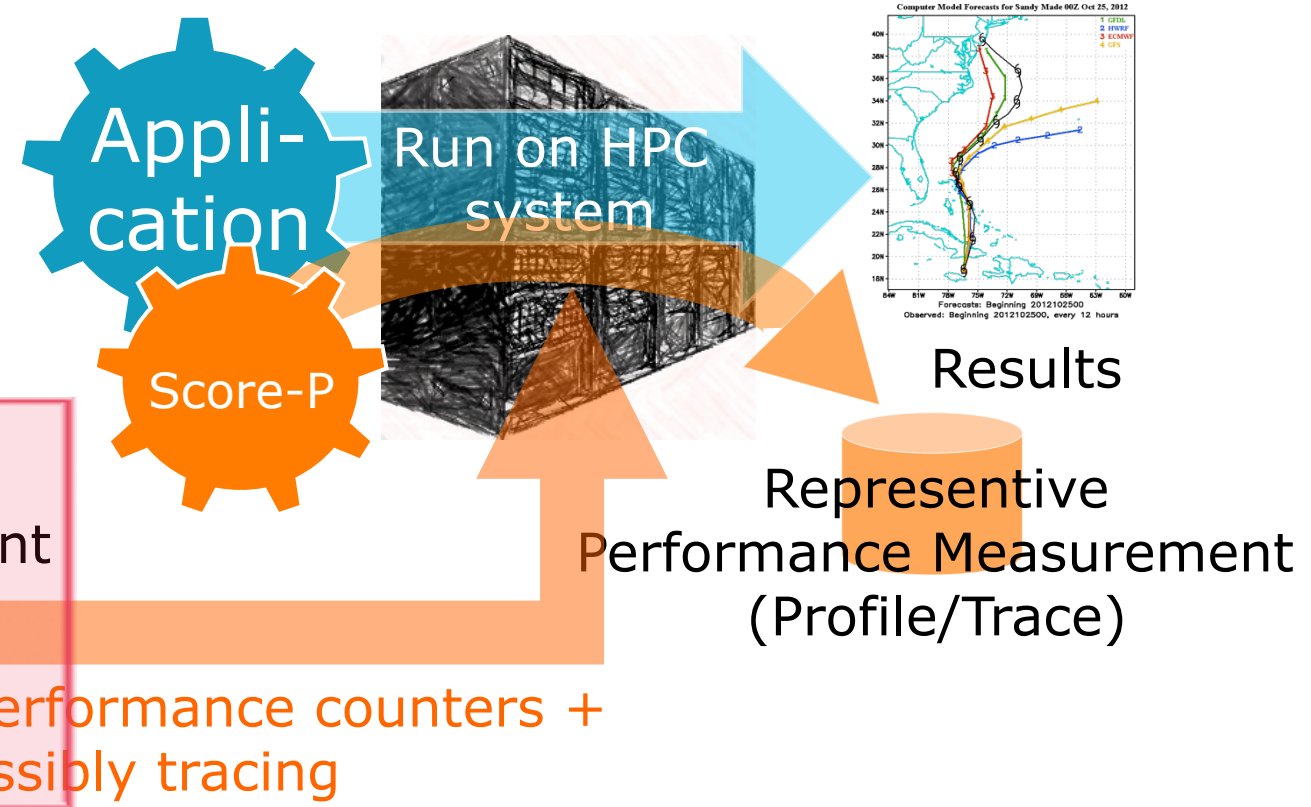
- 0.0 Reference preparation for validation
- 1.0 Program instrumentation
  - 1.1 Summary measurement collection
  - 1.2 Summary analysis report examination
- 2.0 Summary experiment scoring
  - 2.1 Summary measurement collection with filtering
  - 2.2 Filtered summary analysis report examination
- 3.0 Event trace collection
  - 3.1 Event trace examination & analysis

## Overview – Next: Filtering

### ▪ First profiling run



### ▪ Second filtered run (possibly tracing)



## BT-MZ Summary Analysis Result Scoring

```
% scorep-score scorep_8x6_sum/profile.cubex
```

Estimated aggregate size of event trace:

Estimated requirements for largest trace buffer (max\_buf):

Estimated memory requirements (SCOREP\_TOTAL\_MEMORY):

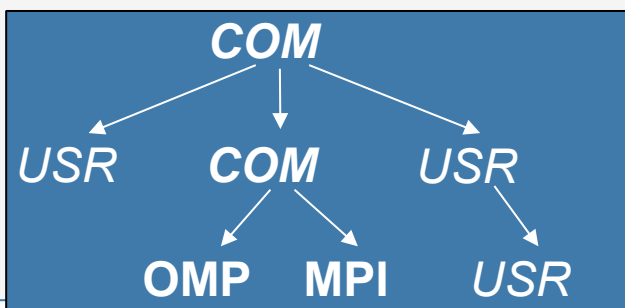
(hint: When tracing set SCOREP\_TOTAL\_MEMORY=20GB to avoid intermediate flushes or reduce requirements using USR regions filters.)

flt	type	max_buf[B]	visits	time[s]	time[%]	time/visit[us]	region
	ALL	21,377,442,117	6,554,106,201	4946.18	100.0	0.75	ALL
	USR	21,309,225,314	6,537,020,537	2326.51	47.0	0.36	USR
	OMP	65,624,896	16,327,168	2607.63	52.7	159.71	OMP
	COM	2,355,080	724,640	2.49	0.1	3.43	COM
	MPI	236,827	33,856	9.56	0.2	282.29	MPI

159 GB  
20 GB  
20 GB

- Report scoring as textual output

159 GB total memory  
20 GB per rank!



- Region/callpath classification
  - MPI** pure MPI functions
  - OMP** pure OpenMP regions
  - USR** user-level computation
  - COM** "combined" USR+OpenMP/MPI
  - ANY/ALL** aggregate of all region types

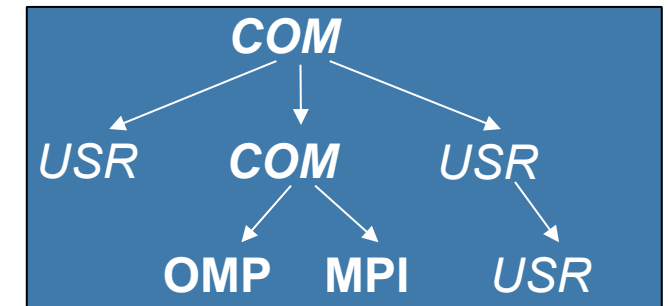
# BT-MZ Summary Analysis Report Breakdown

```
% scorep-score -r scorep_8x6_sum/profile.cubex
```

```
[...]
```

flt	type	max_buf[B]	visits	time[s]	time[%]	time/visit[us]	region
ALL		21,377,442,117	6,554,106,201	4946.18	100.0	0.75	ALL
USR		21,309,225,314	6,537,020,537	2326.51	47.0	0.36	USR
OMP		65,624,896	16,327,168	2607.63	52.7	159.71	OMP
COM		2,355,080	724,640	2.49	0.1	3.43	COM
MPI		236,827	33,856	9.56	0.2	282.29	MPI

USR	6,883,222,086	2,110,313,472	651.44	13.2	0.31	matvec_sub_
USR	6,883,222,086	2,110,313,472	720.38	14.6	0.34	matmul_sub_
USR	6,883,222,086	2,110,313,472	881.32	17.8	0.42	binvrhs_
USR	293,617,584	87,475,200	29.93	0.6	0.34	binvrhs_
USR	293,617,584	87,475,200	33.03	0.7	0.38	lhsinit_
USR	101,320,128	31,129,600	7.78	0.2	0.25	exact_solution_



More than  
18 GB just for these 6  
regions

## BT-MZ Summary Analysis Score

---

- Summary measurement analysis score reveals
  - Total size of event trace would be ~159 GB
  - Maximum trace buffer size would be ~20 GB per rank
    - smaller buffer would require flushes to disk during measurement resulting in substantial perturbation
  - 99.8% of the trace requirements are for USR regions
    - purely computational routines never found on COM call-paths common to communication routines or OpenMP parallel regions
  - These USR regions contribute around 32% of total time
    - however, much of that is very likely to be measurement overhead for 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

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

% scorep-score -f ../config/scorep.filt -c 2 \
>scorep_8x6_sum/profile.cubex

Estimated aggregate size of event trace: 521 MB
Estimated requirements for largest trace buffer (max_buf): 66 MB
Estimated memory requirements (SCOREP_TOTAL_MEMORY): 78 MB
(hint: When tracing set SCOREP_TOTAL_MEMORY=78MB to avoid \
>intermediate flushes
or reduce requirements using USR regions filters.)
```

- Report scoring with prospective filter listing 6 USR regions

521 MB of memory in total,  
66 MB per rank!

(Including 2 metric values)



# BT-MZ Summary Analysis Report Filtering

```
% scorep-score -r -f ../config/scorep.filt \
> scorep_8x6_sum/profile.cubex
flt type      max_buf[B]      visits time[s] time[%] time/visit[us] region
-   ALL 21,377,442,117 6,554,106,201 4946.18 100.0 0.75 ALL
-   USR 21,309,225,314 6,537,020,537 2326.51 47.0 0.36 USR
-   OMP 65,624,896 16,327,168 2607.63 52.7 159.71 OMP
-   COM 2,355,080 724,640 2.49 0.1 3.43 COM
-   MPI 236,827 33,856 9.56 0.2 282.29 MPI
*   ALL 68,216,855 17,085,673 2622.30 53.0 153.48 ALL-FLT
+   FLT 21,309,225,262 6,537,020,528 2323.88 47.0 0.36 FLT
-   OMP 65,624,896 16,327,168 2607.63 52.7 159.71 OMP-FLT
*   COM 2,355,080 724,640 2.49 0.1 3.43 COM-FLT
-   MPI 236,827 33,856 9.56 0.2 282.29 MPI-FLT
*   USR 52 9 2.63 0.1 292158.12 USR-FLT

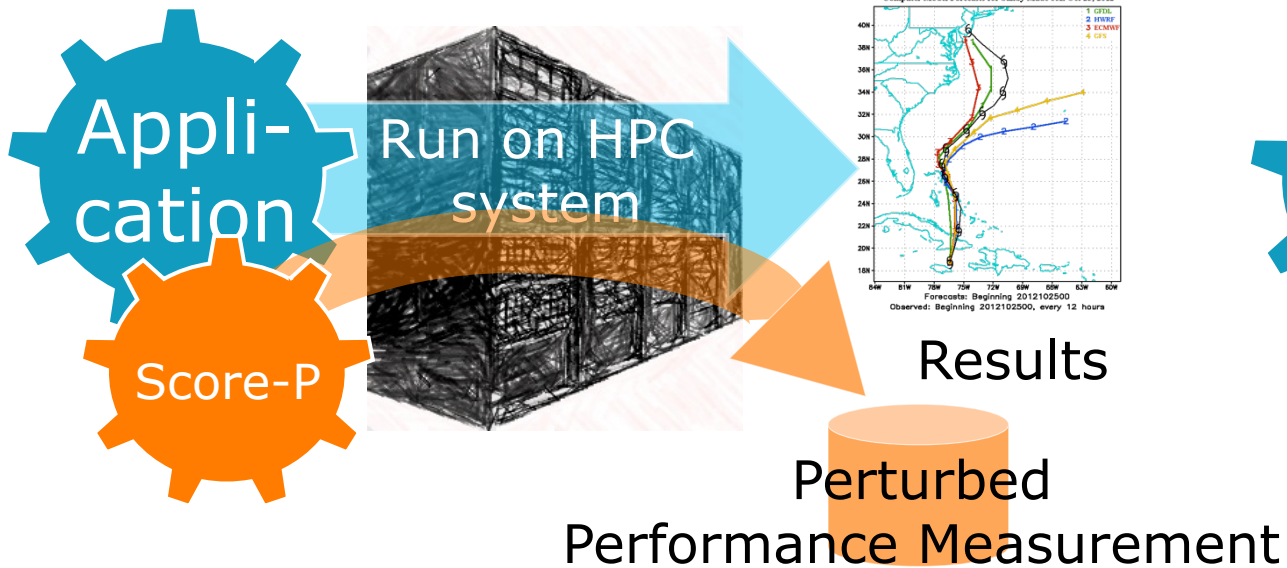
+   USR 6,883,222,086 2,110,313,472 651.44 13.2 0.31 matvec_sub_
+   USR 6,883,222,086 2,110,313,472 720.38 14.6 0.34 matmul_sub_
+   USR 6,883,222,086 2,110,313,472 881.32 17.8 0.42 binvcrhs_
+   USR 293,617,584 87,475,200 29.93 0.6 0.34 binvrhs_
+   USR 293,617,584 87,475,200 33.03 0.7 0.38 lhsinit_
+   USR 101,320,128 31,129,600 7.78 0.2 0.25 exact_solution_
```

- Score report breakdown by region

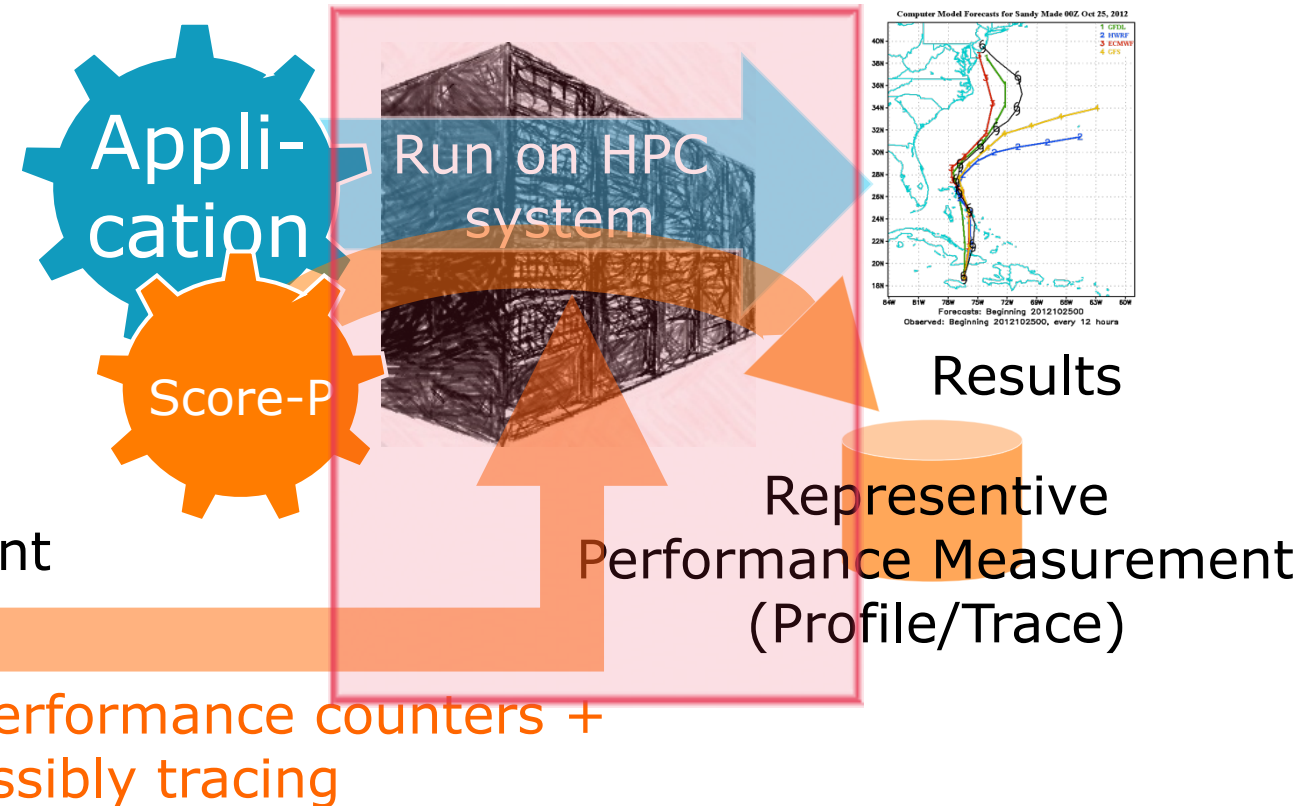
Filtered routines marked with '+'

## Overview – Next: Filtering

### ▪ First profiling run



### ▪ Second filtered run (possibly tracing)



## Summary Measurement Collection – Score-P w/ Filter

- Set new experiment directory and re-run measurement with new filter configuration
- COSMA and Hamilton
- Archer

```
% cd bin.scorep
% export OMP_NUM_THREADS=4
% export SCOREP_EXPERIMENT_DIRECTORY=\
    scorep_4x4_sum_filter
% export SCOREP_FILTERING_FILE=\
    ../config/scorep.filt
% mpirun -np 4 ./bt-mz_B.4
```

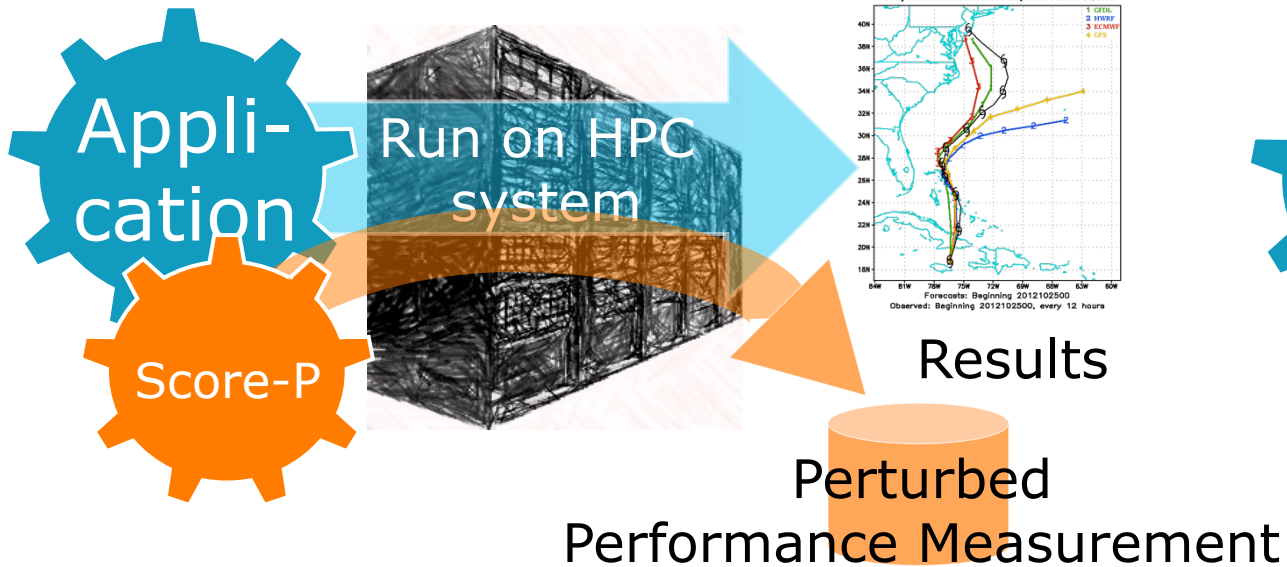
Runs directly on frontend – Use a jobscript if you have access to quick to react queues  
Example jobscripts available in:  
`../jobscripts/{cosma/hamilton}/`

```
% cd bin.scorep
% cp ../jobscript/archer/scorep.pbs ./
% nano scorep.pbs
...
#PBS -A y14
...
export OMP_NUM_THREADS=6
PROCS=8
CLASS=C
EXE=./bt-mz_${CLASS}.${PROCS}
export SCOREP_EXPERIMENT_DIRECTORY=\
scorep_${NPROCS}x${OMP_NUM_THREADS}_sum_filter
export SCOREP_FILTERING_FILE=../config/scorep.filt
#export SCOREP_METRIC_PAPI=PAPI_TOT_INS,PAPI_TOT_CYC
#export SCOREP_TOTAL_MEMORY=300M
...
% qsub -q short scorep.pbs
```

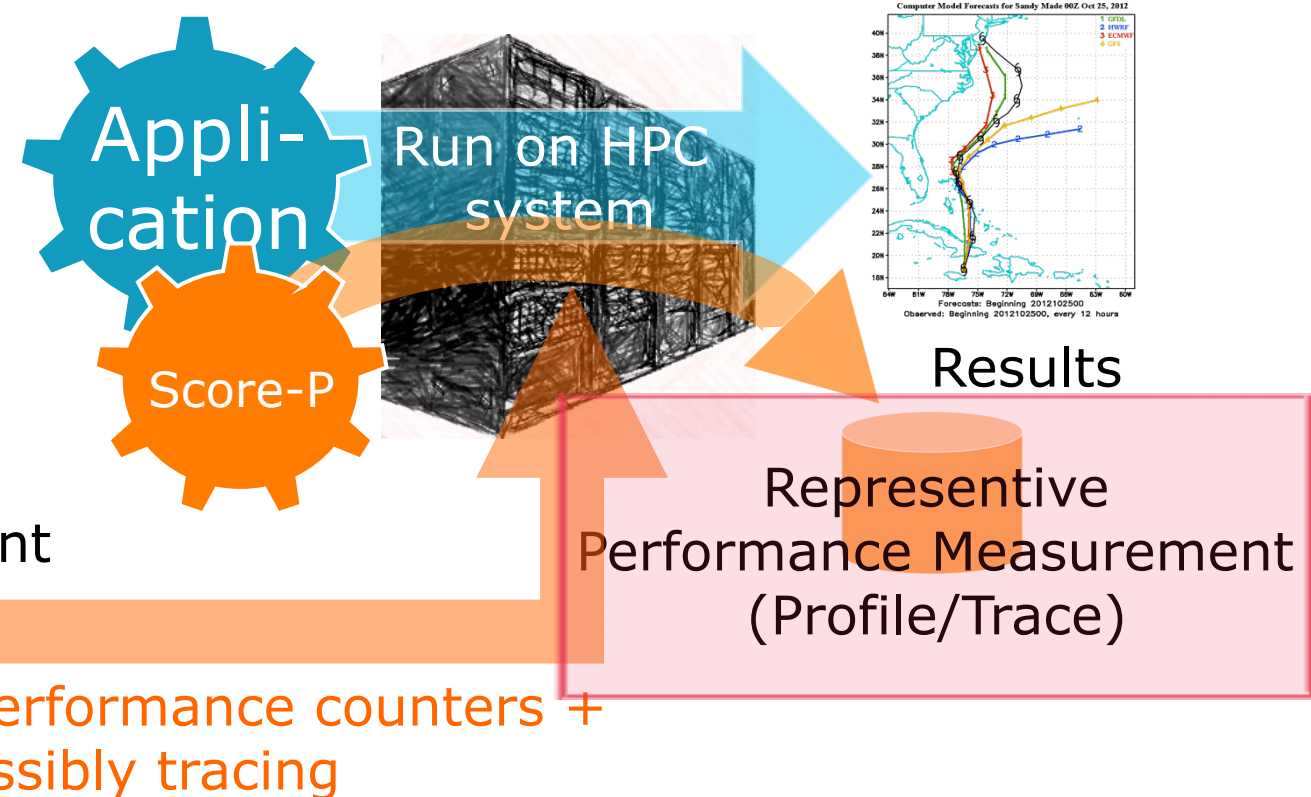
Adapt!

## Overview – Next: Filtering

### ▪ First profiling run



### ▪ Second filtered run (possibly tracing)



## BT-MZ Summary Analysis Report Examination – With Filter

or 4x4

```
% cube scorep_8x6_sum_filter/profile.cubex
```

```
[CUBE GUI showing summary analysis report]
```

- Interactive exploration with CUBE
- This time reported times are representative of the actual application behavior

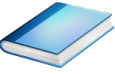


## Score-P: Advanced Measurement Configuration





# Advanced Measurement Configuration: Metrics



- Available PAPI metrics
  - Preset events: common set of events deemed relevant and useful for application performance tuning
    - Abstraction from specific hardware performance counters, mapping onto available events done by PAPI internally

```
% papi_avail
```

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

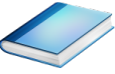
```
% papi_native_avail
```

## Note:

Due to hardware restrictions

- number of concurrently recorded events is limited
- there may be invalid combinations of concurrently recorded events

# Advanced Measurement Configuration: Metrics



```
% man getrusage
struct rusage {
    struct timeval ru_utime; /* user CPU time used */
    struct timeval ru_stime; /* system CPU time used */
    long ru_maxrss; /* maximum resident set size */
    long ru_ixrss; /* integral shared memory size */
    long ru_idrss; /* integral unshared data size */
    long ru_isrss; /* integral unshared stack size */
    long ru_minflt; /* page reclaims (soft page faults) */
    long ru_majflt; /* page faults (hard page faults) */
    long ru_nswap; /* swaps */
    long ru_inblock; /* block input operations */
    long ru_oublock; /* block output operations */
    long ru_msgsnd; /* IPC messages sent */
    long ru_msrvcv; /* IPC messages received */
    long ru_nsignals; /* signals received */
    long ru_nvcsw; /* voluntary context switches */
    long ru_nivcsw; /* involuntary context switches */
};
```

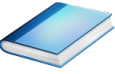
- Available resource usage metrics

- Note:

- (1) Not all fields are maintained on each platform.
- (2) Check scope of metrics (per process vs. per thread)

# Advanced Measurement Configuration: CUDA

---



- Record CUDA events with the CUPTI interface

```
% export SCOREP_CUDA_ENABLE=gpu,kernel,idle
```

- All possible recording types
  - runtime      CUDA runtime API
  - driver        CUDA driver API
  - gpu           GPU activities
  - kernel        CUDA kernels
  - idle          GPU compute idle time
  - memcpy        CUDA memory copies

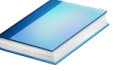
# Score-P User Instrumentation API

---



- Can be used to mark initialization, solver & other phases
  - Annotation macros ignored by default
  - Enabled with [--user] flag
- Appear as additional regions in analyses
  - Distinguishes performance of important phase from rest
- Can be of various type
  - E.g., function, loop, phase
  - See user manual for details
- Available for Fortran / C / C++

# Score-P User Instrumentation API (Fortran)



```
#include "scorep/SCOREP_User.inc"

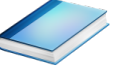
subroutine foo(...)
  ! Declarations
  SCOREP_USER_REGION_DEFINE( solve )

  ! Some code...
  SCOREP_USER_REGION_BEGIN( solve, "<solver>", \
                           SCOREP_USER_REGION_TYPE_LOOP )

  do i=1,100
    [...]
  end do
  SCOREP_USER_REGION_END( solve )
  ! Some more code...
end subroutine
```

- Requires processing by the C preprocessor

# Score-P User Instrumentation API (C/C++)



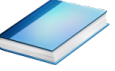
```
#include "scorep/SCOREP_User.h"

void foo()
{
    /* Declarations */
    SCOREP_USER_REGION_DEFINE( solve )

    /* Some code... */
    SCOREP_USER_REGION_BEGIN( solve, "<solver>",
                             SCOREP_USER_REGION_TYPE_LOOP )
    for (i = 0; i < 100; i++)
    {
        [...]
    }
    SCOREP_USER_REGION_END( solve )
    /* Some more code... */
}
```



# Score-P User Instrumentation API (C++)



```
#include "scorep/SCOREP_User.h"

void foo()
{
    // Declarations

    // Some code...
    {
        SCOREP_USER_REGION( "<solver>",
                           SCOREP_USER_REGION_TYPE_LOOP )
        for (i = 0; i < 100; i++)
        {
            [...]
        }
    }
    // Some more code...
}
```

# Score-P Measurement Control API



- Can be used to temporarily disable measurement for certain intervals
  - Annotation macros ignored by default
  - Enabled with [--user] flag

```
#include "scorep/SCOREP_User.inc"

subroutine foo(...)
  ! Some code...
  SCOREP_RECORDING_OFF()
  ! Loop will not be measured
  do i=1,100
    [...]
  end do
  SCOREP_RECORDING_ON()
  ! Some more code...
end subroutine
```

Fortran (requires Cpreprocessor)

```
#include "scorep/SCOREP_User.h"

void foo(...) {
  /* Some code... */
  SCOREP_RECORDING_OFF()
  /* Loop will not be measured */
  for (i = 0; i < 100; i++) {
    [...]
  }
  SCOREP_RECORDING_ON()
  /* Some more code... */
}
```

C / C++

## Further Information

---

- Community instrumentation & measurement infrastructure
  - Instrumentation (various methods)
  - Basic and advanced profile generation
  - Event trace recording
  - Online access to profiling data
- Available under New BSD open-source license
- Documentation & Sources:
  - <http://www.score-p.org>
- User guide also part of installation:
  - `<prefix>/share/doc/scorep/{pdf,html}/`
- Support and feedback: [support@score-p.org](mailto:support@score-p.org)
- Subscribe to [news@score-p.org](mailto:news@score-p.org), to be up to date

# Analysis report examination with CUBE

---

Brian Wylie  
Jülich Supercomputing Centre

---



# CUBE

---

Parallel program analysis report exploration tools

- Libraries for XML report reading & writing
- Algebra utilities for report processing
- GUI for interactive analysis exploration
  - requires Qt4/5

Originally developed as part of Scalasca toolset

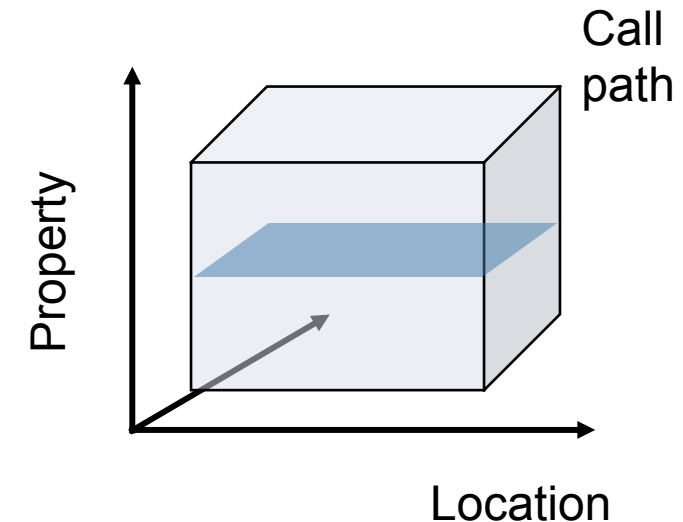
Now available as a separate component

- Can be installed independently of Score-P, e.g., on laptop or desktop
- Latest release: CUBE 4.3.2 (June 2015)

# Analysis presentation and exploration

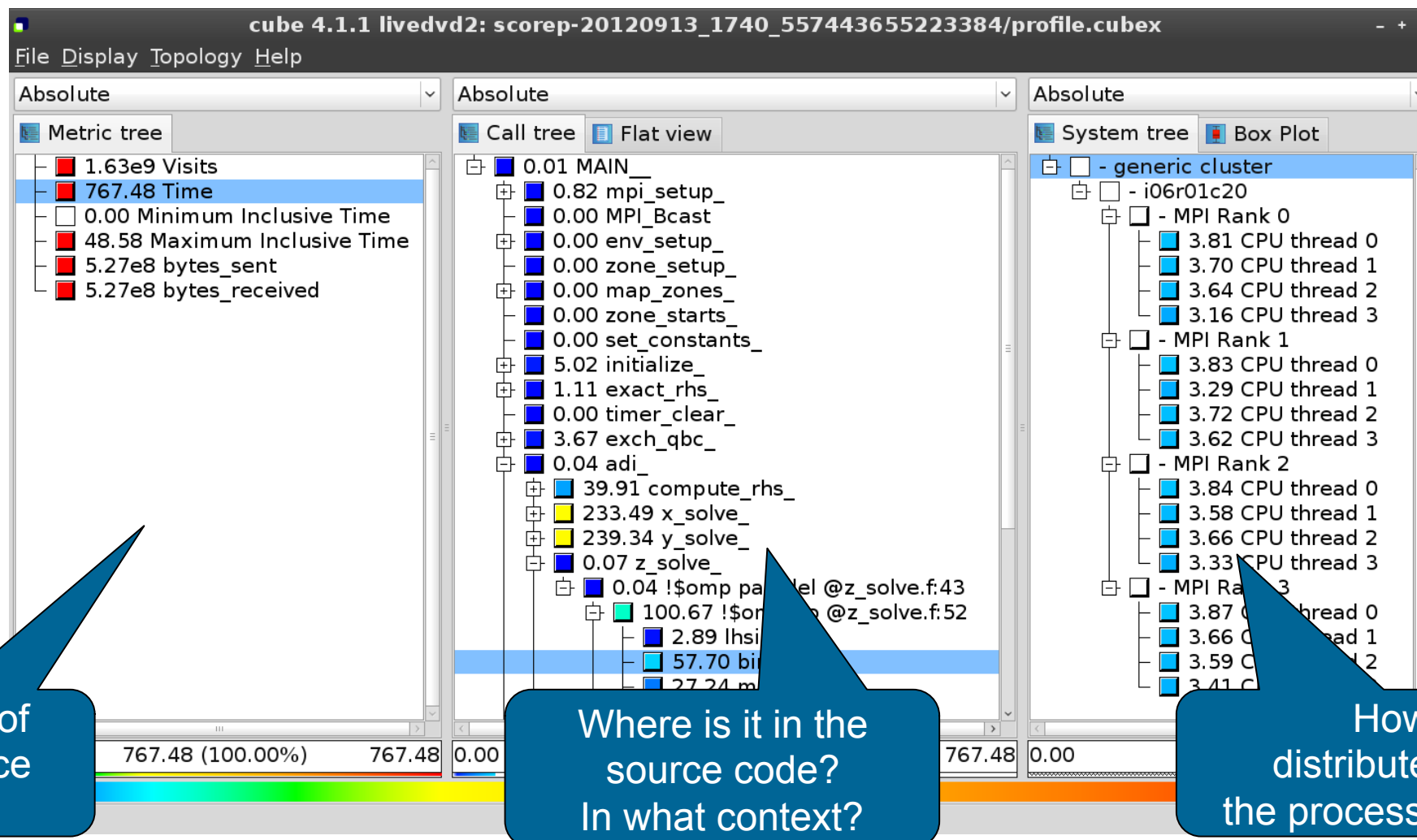
---

- Representation of values (severity matrix) on three hierarchical axes
  - Performance property (metric)
  - Call path (program location)
  - System location (process/thread)
- Three coupled tree browsers
- CUBE displays severities
  - As value: for precise comparison
  - As colour: for easy identification of hotspots
  - Inclusive value when closed & exclusive value when expanded
  - Customizable via display modes

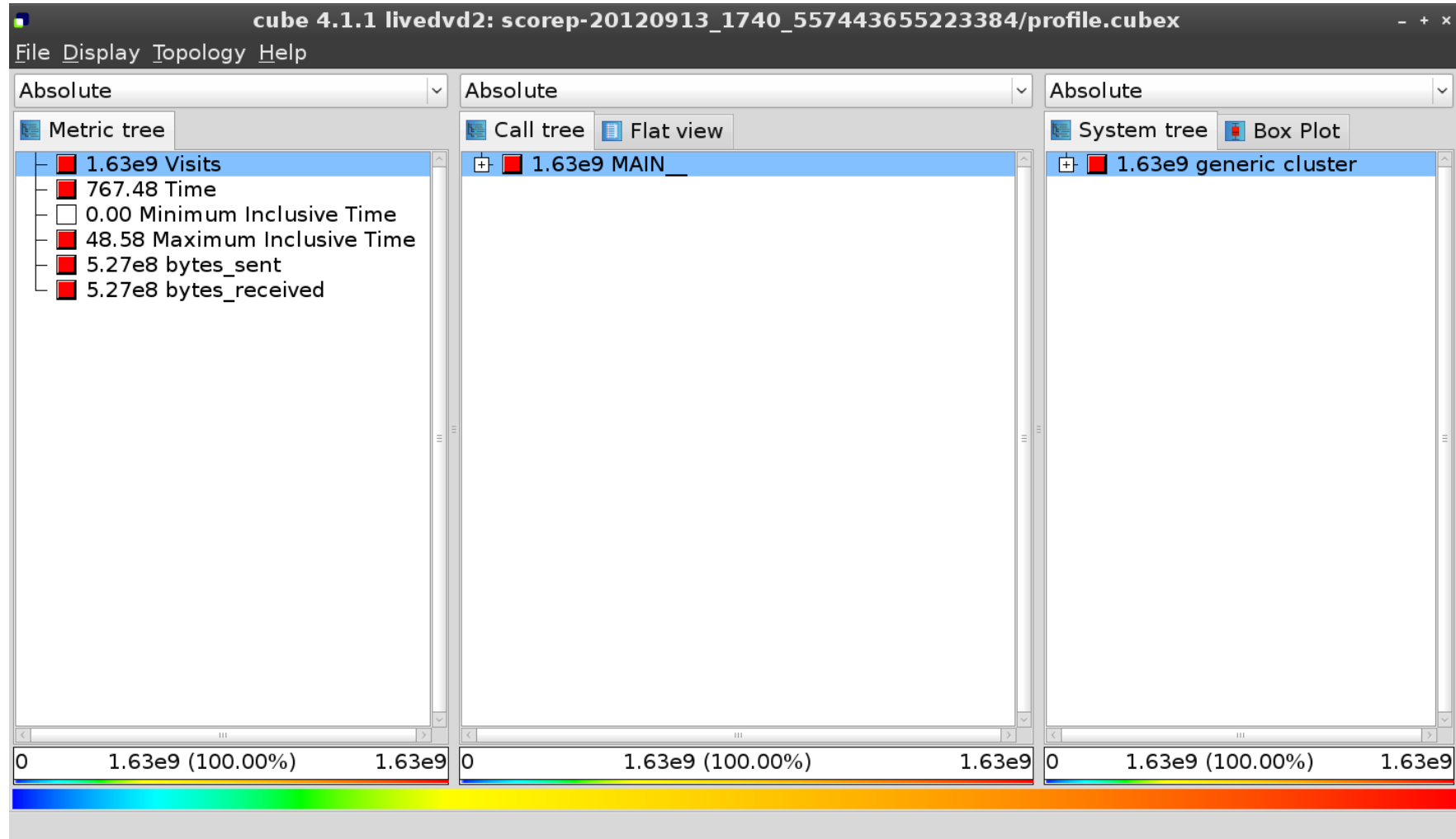




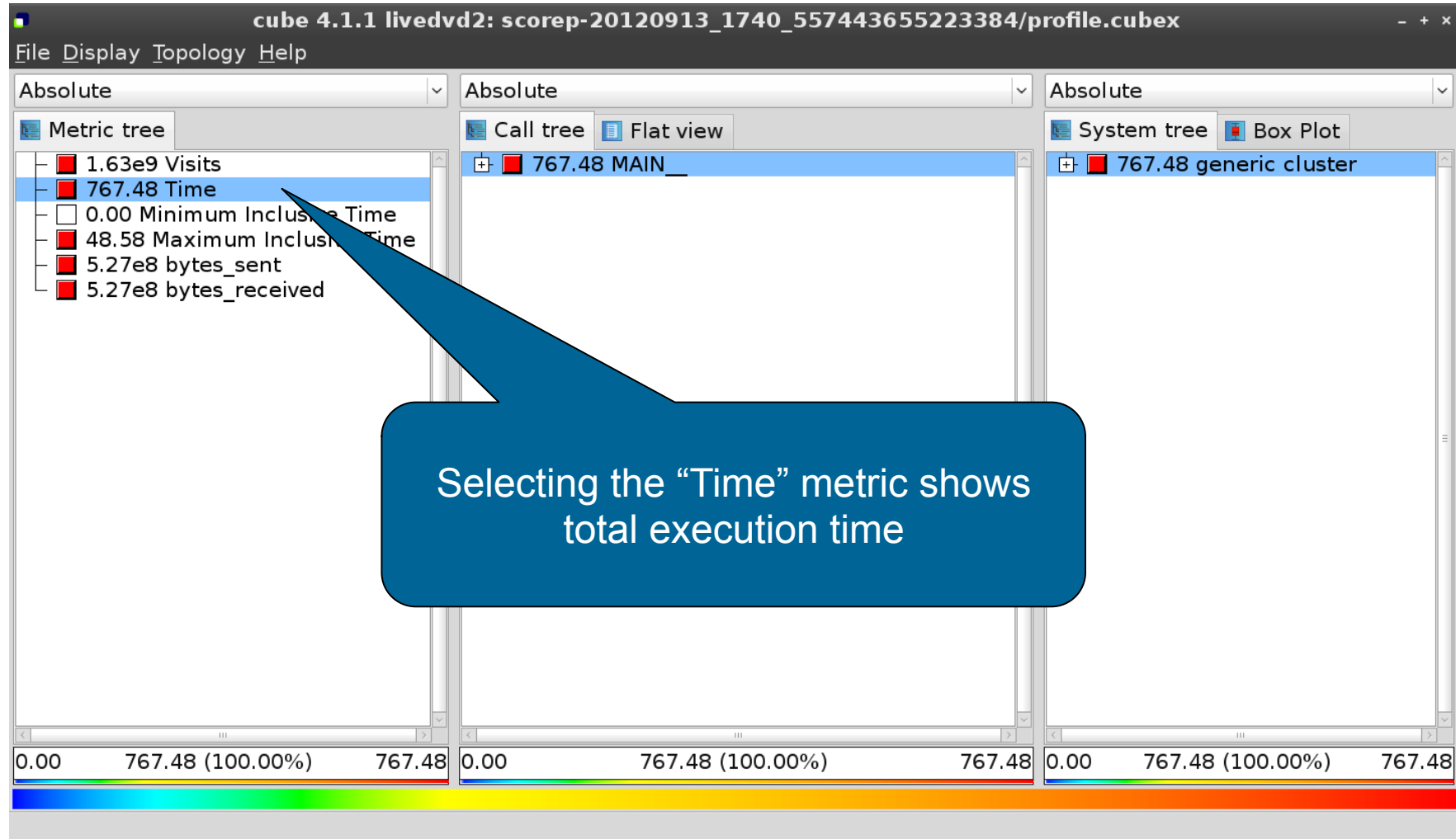
# Analysis presentation



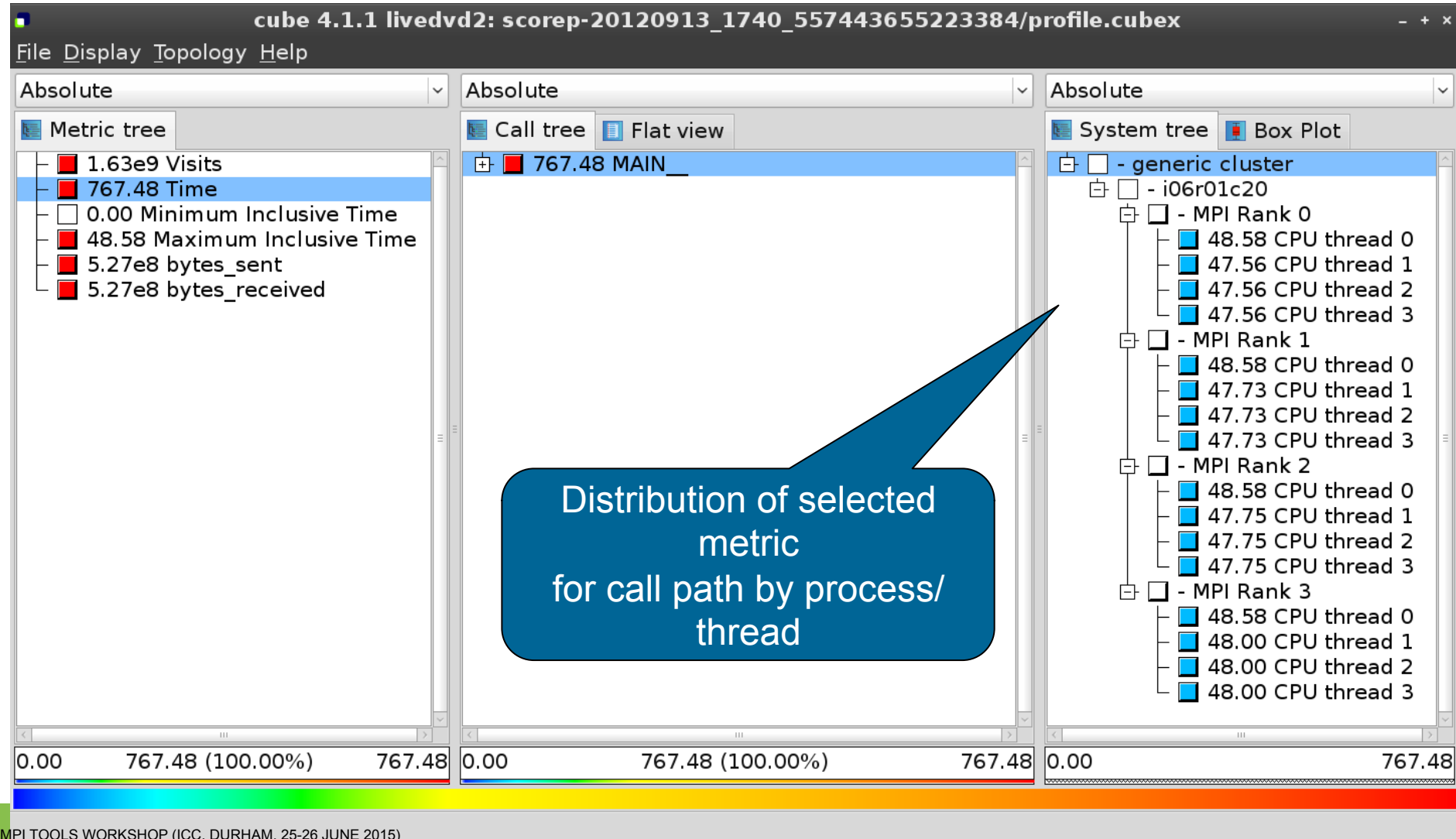
# Analysis report exploration (opening view)



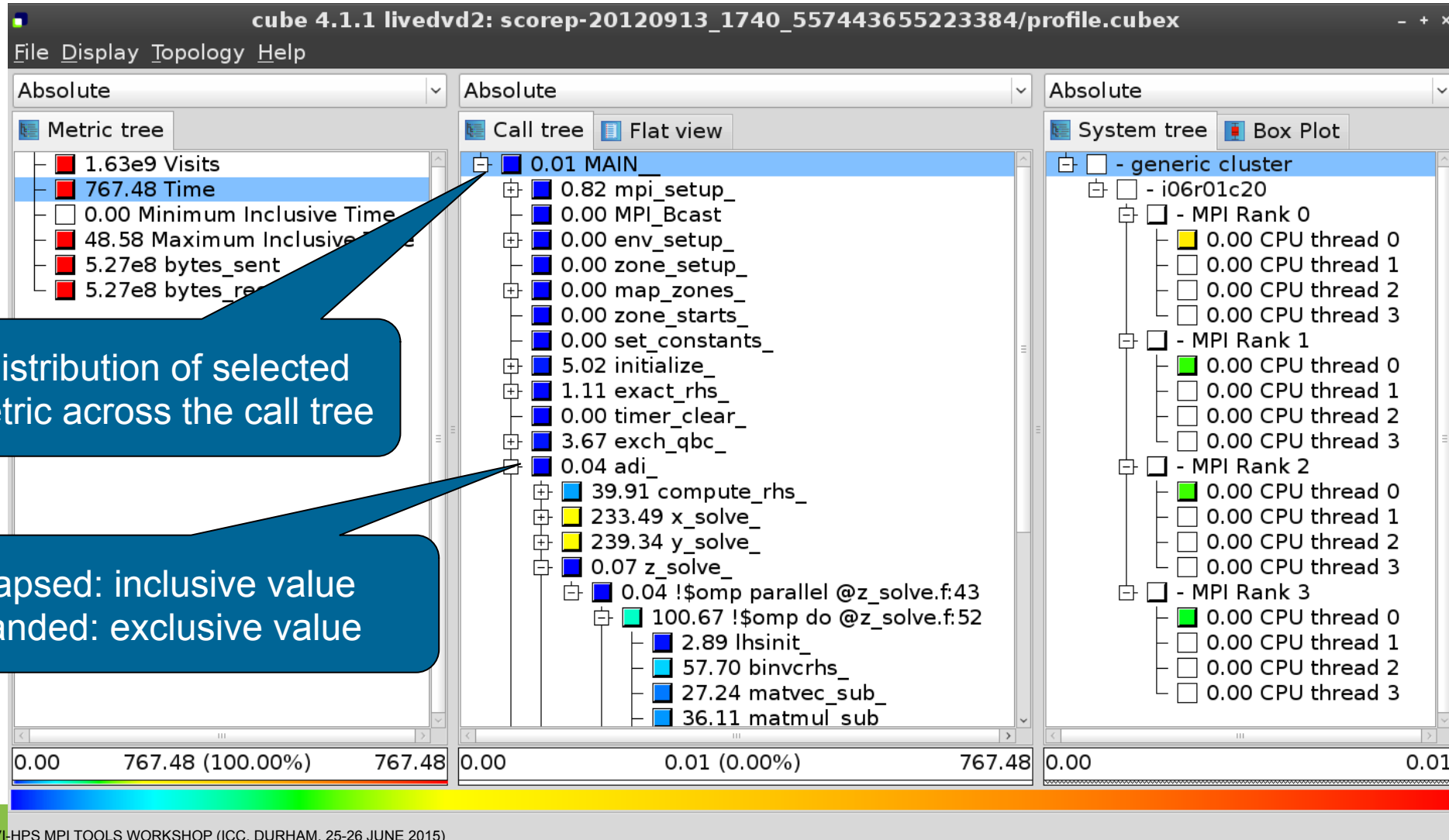
# Metric selection



# Expanding the system tree

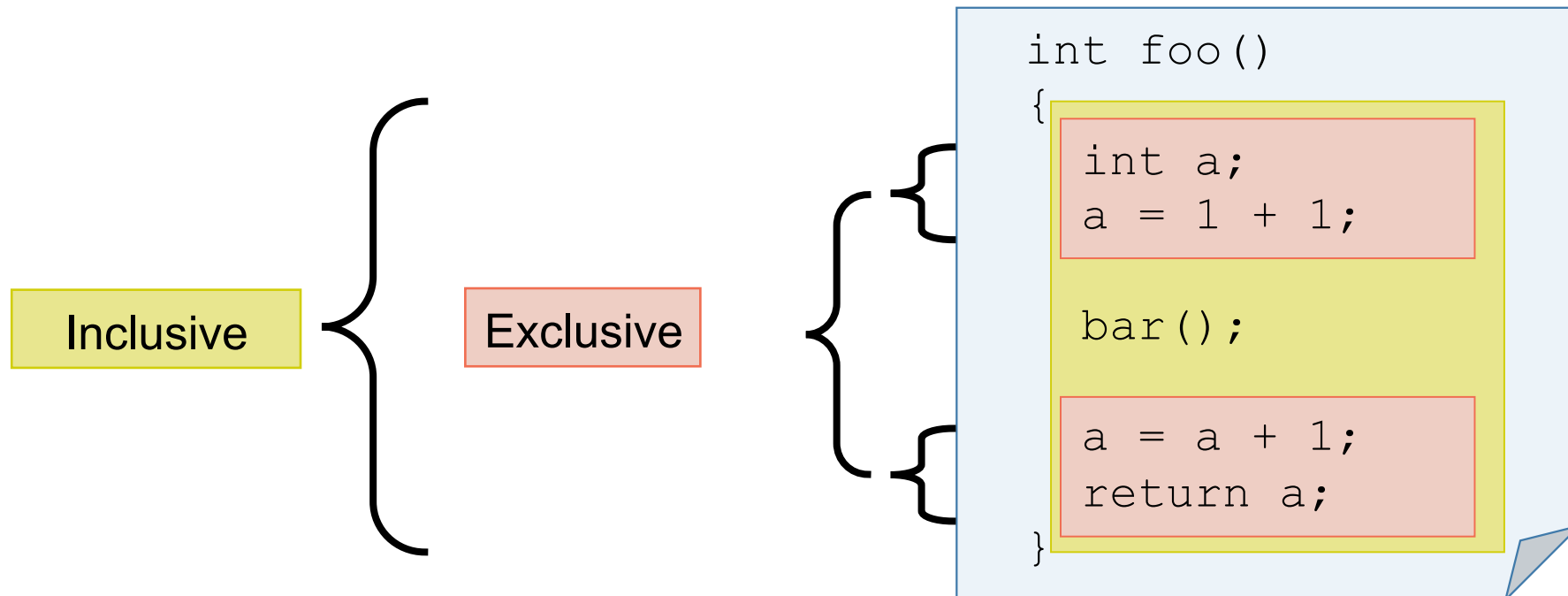


# Expanding the call tree



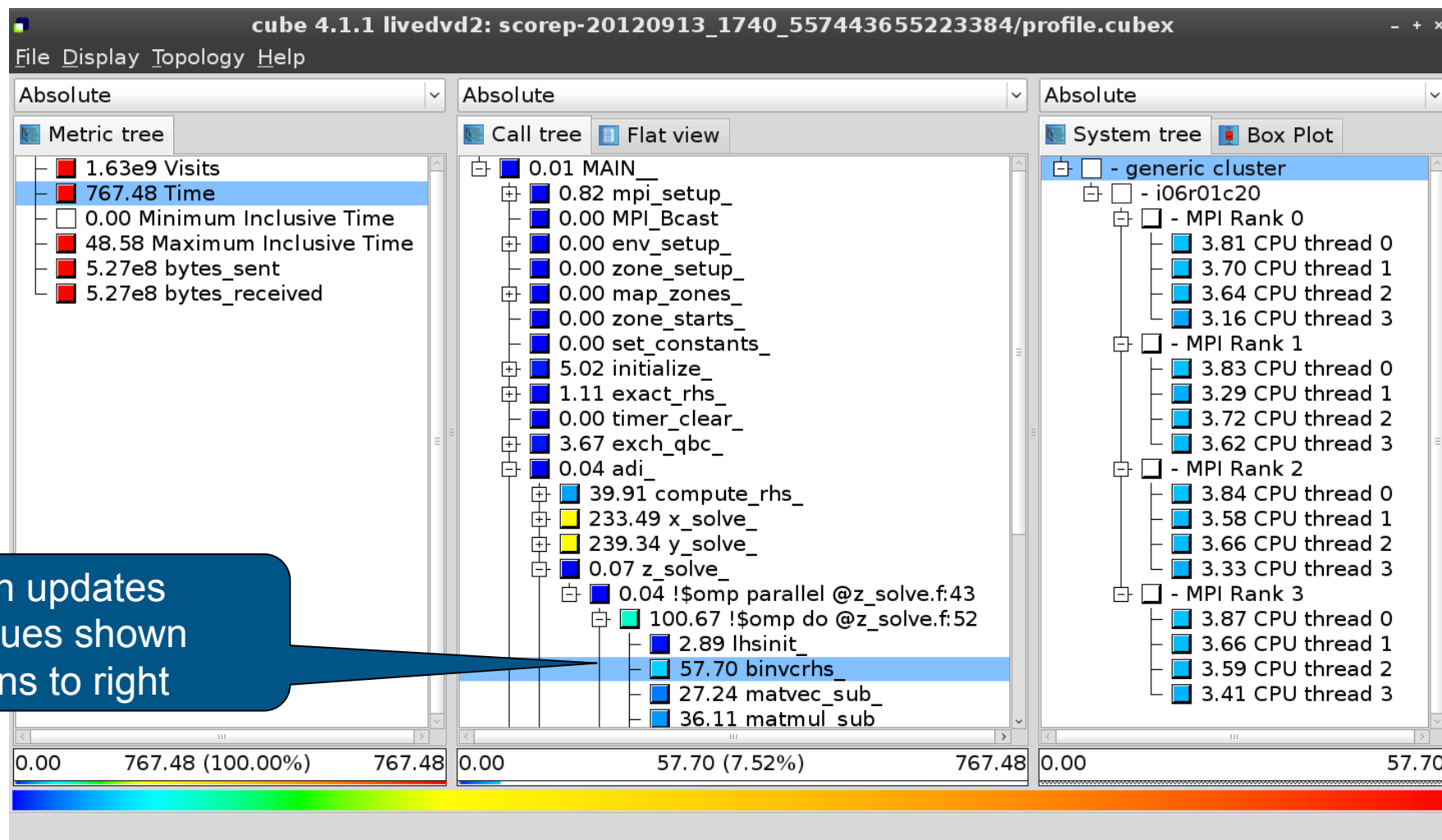
# Inclusive vs. Exclusive values

- Inclusive
  - Information of all sub-elements aggregated into single value
- Exclusive
  - Information cannot be subdivided further





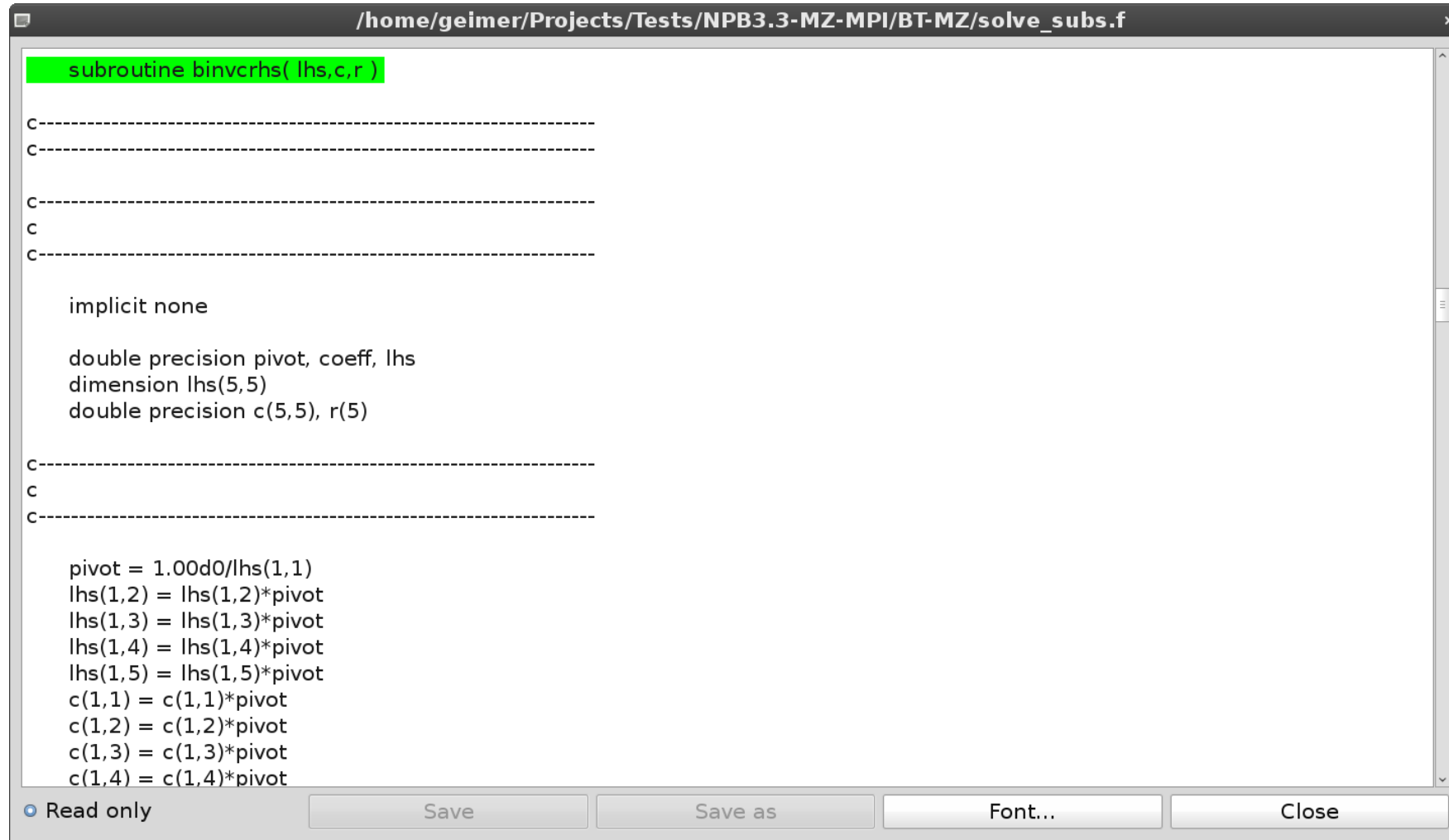
# Selecting a call path



## Source-code view via context menu

The screenshot displays the 'cube 4.1.1 livevd2' interface with three main panels: Metric tree, Call tree, and System tree. A context menu is open over the 'binvcrhs' item in the Call tree. A blue callout box points to the menu with the text 'Right-click opens context menu'. The context menu options are: Call site, Called region, Expand/collapse, Hiding, Cut call tree, Find items, Find Next, Clear found items, Copy to clipboard, and Min/max values. The 'Source code' option is highlighted, and a sub-menu is visible showing a list of CPU threads: 3.84 CPU thread 0, 3.58 CPU thread 1, 3.66 CPU thread 2, 3.33 CPU thread 3, 3.87 CPU thread 0, 3.66 CPU thread 1, 3.59 CPU thread 2, and 3.41 CPU thread 3. The status bar at the bottom shows 'Shows the source code of the clicked item'.

# Source-code view



```
/home/geimer/Projects/Tests/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f
subroutine binvcrhs( lhs,c,r )
C-----
C-----
C-----
C
C-----

implicit none

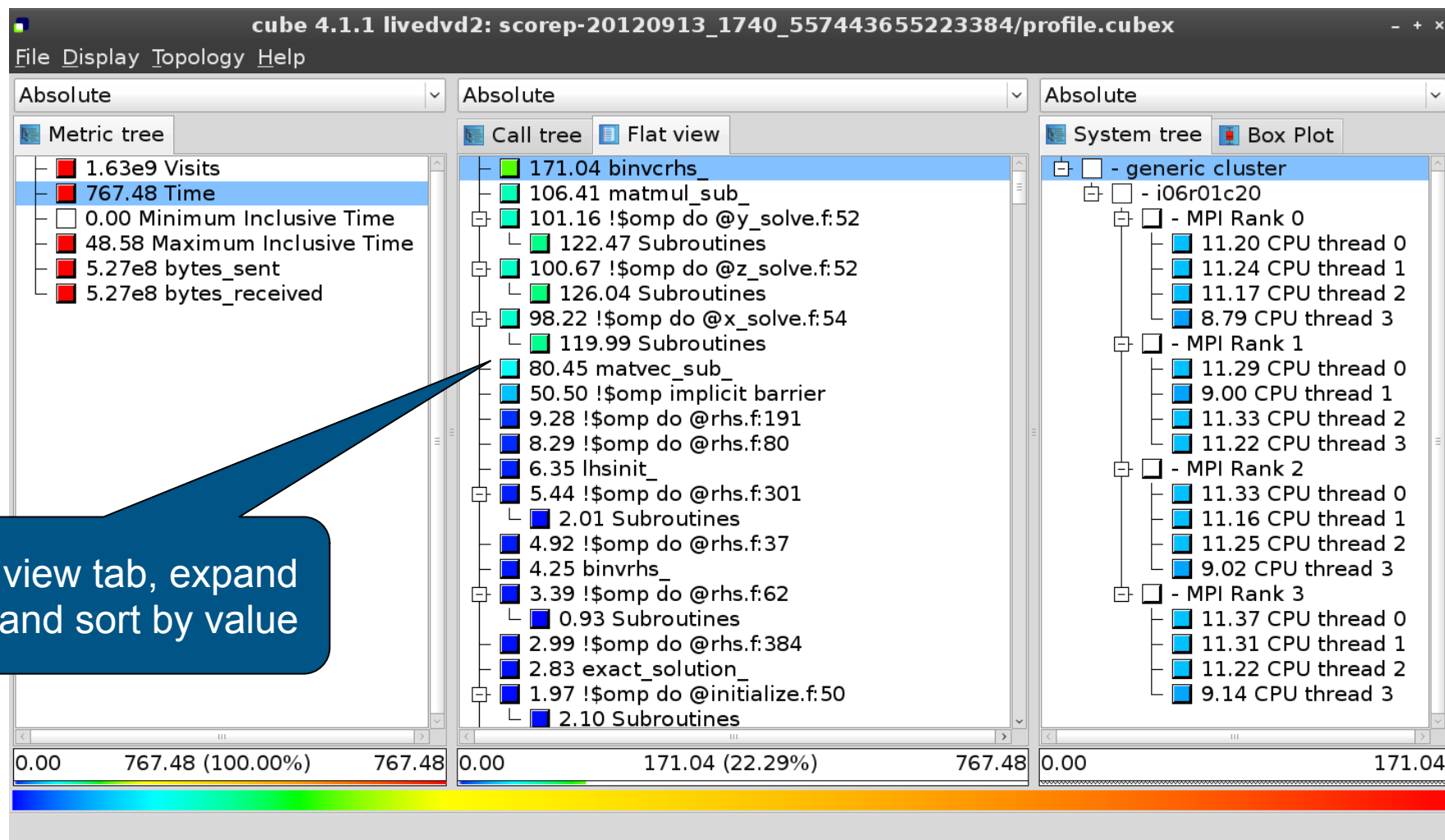
double precision pivot, coeff, lhs
dimension lhs(5,5)
double precision c(5,5), r(5)

C-----
C
C-----

pivot = 1.00d0/lhs(1,1)
lhs(1,2) = lhs(1,2)*pivot
lhs(1,3) = lhs(1,3)*pivot
lhs(1,4) = lhs(1,4)*pivot
lhs(1,5) = lhs(1,5)*pivot
c(1,1) = c(1,1)*pivot
c(1,2) = c(1,2)*pivot
c(1,3) = c(1,3)*pivot
c(1,4) = c(1,4)*pivot
```

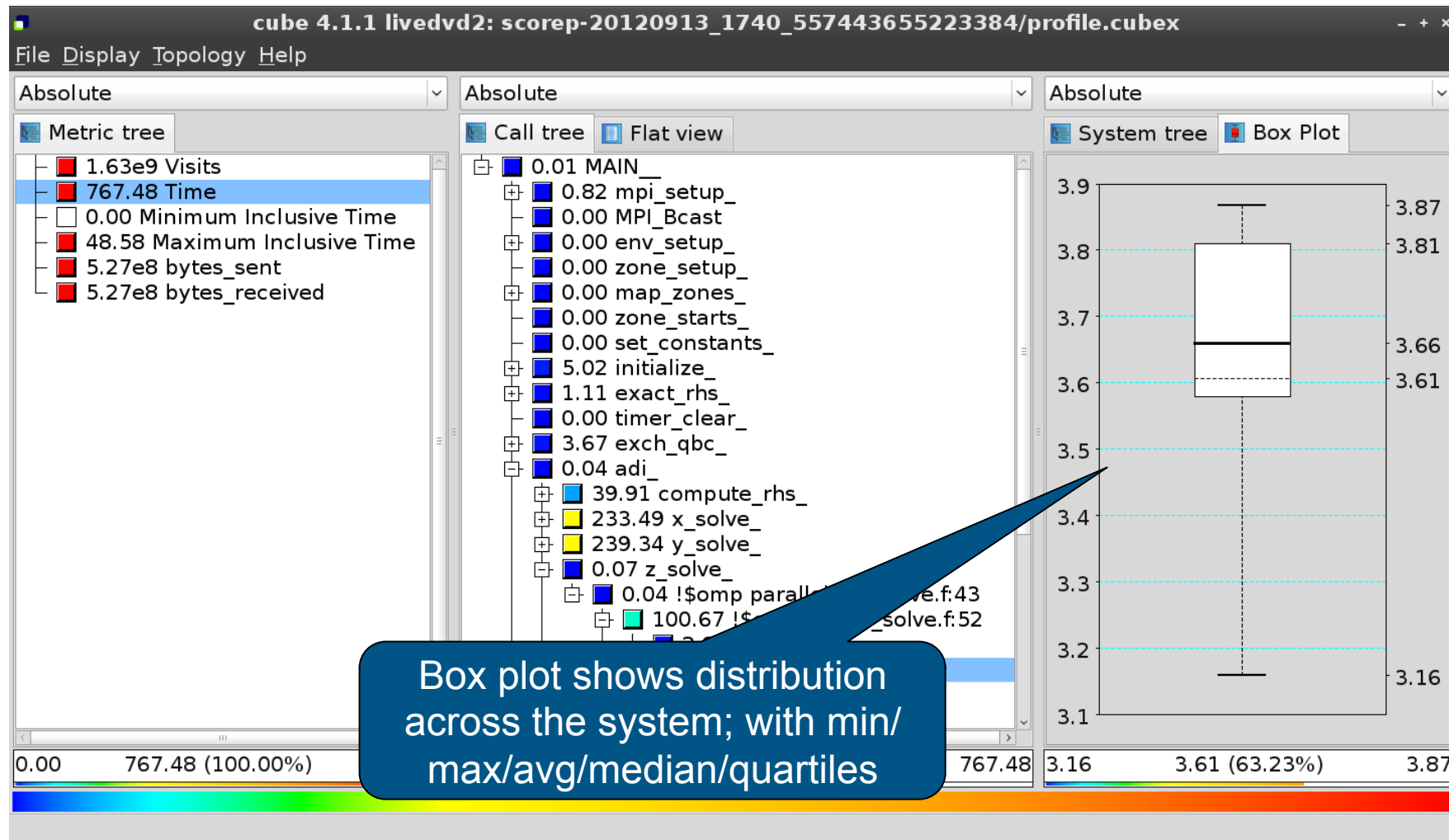
Read only    Save    Save as    Font...    Close

# Flat profile view

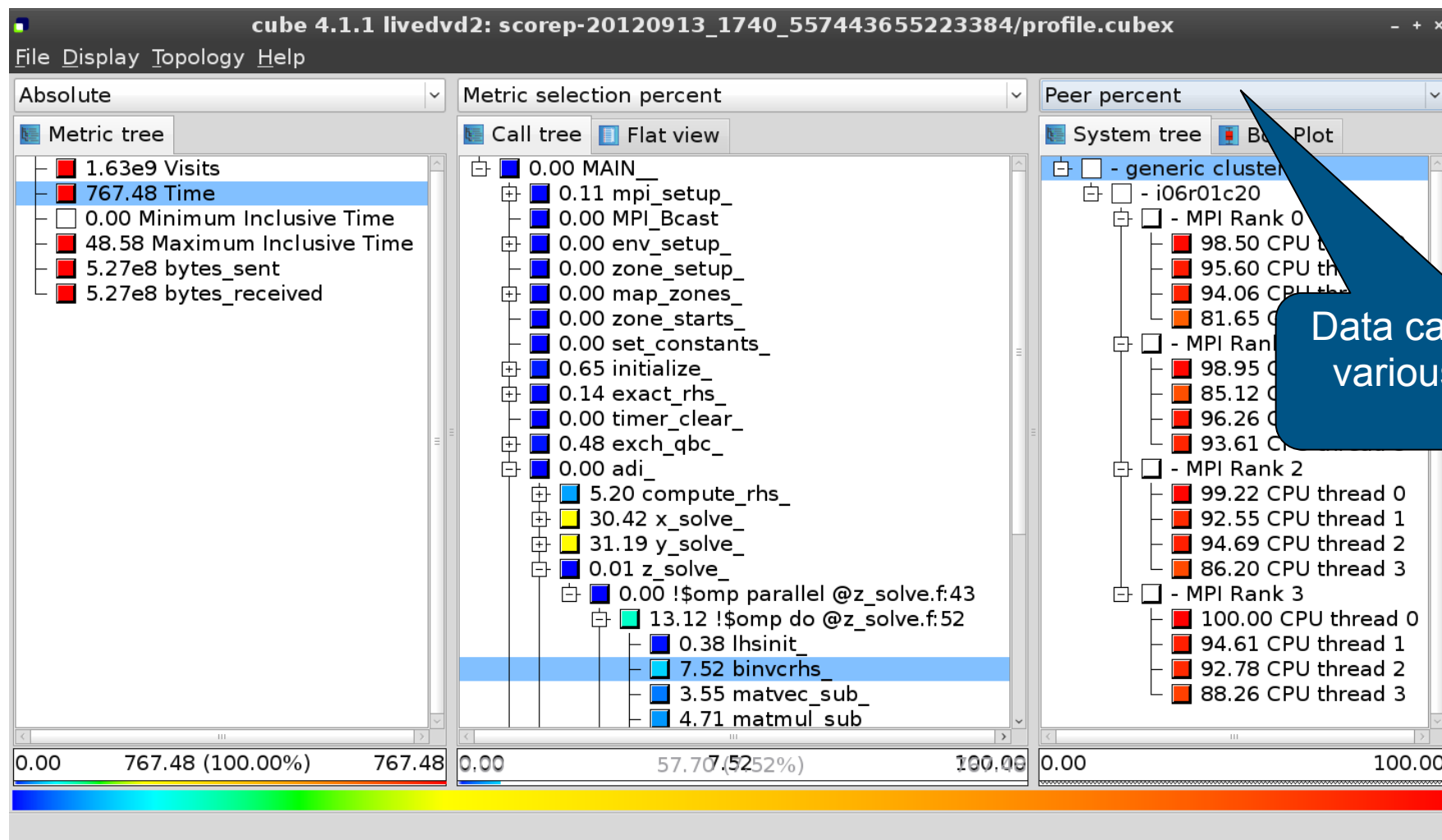


Select flat view tab, expand all nodes, and sort by value

# Box plot view



# Alternative display modes



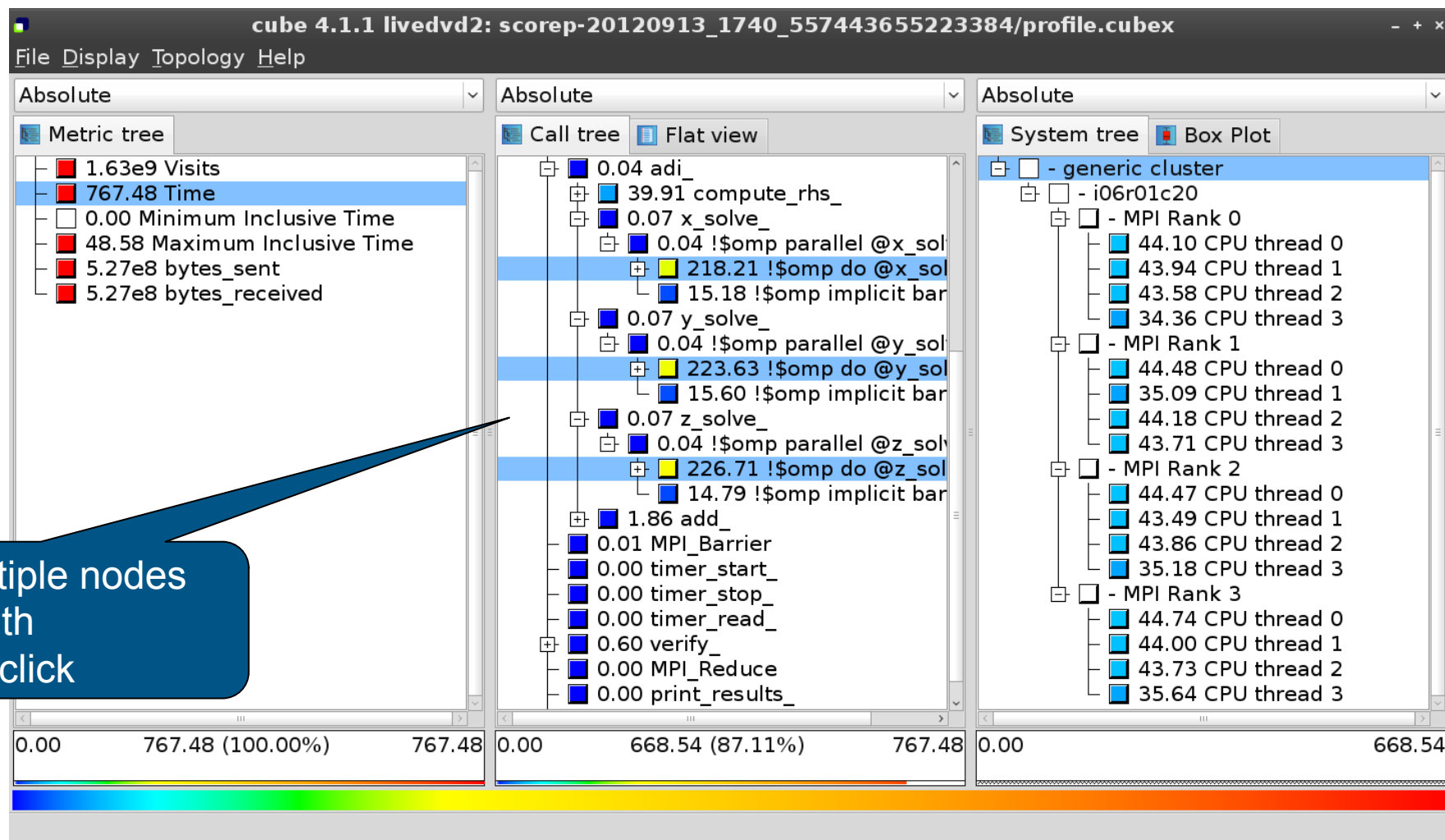


# Important display modes

---

- Absolute
  - Absolute value shown in seconds/bytes/counts
- Selection percent
  - Value shown as percentage w.r.t. the selected node  
“on the left” (metric/call path)
- Peer percent (system tree only)
  - Value shown as percentage relative to the maximum peer value

# Multiple selection



## Derived metrics in Cube

---

- Value of the derived metric is not stored, but **calculated** on-the-fly
- One defines an CubePL expression, e.g.:  
**metric::time(i)/metric::visits(e)**
- Types of derived metrics:
  - **Prederived**: evaluation of the CubePL expression is done before the aggregation
  - **Postderived**: evaluation of the CubePL expression is performed after the aggregation
- Examples:
  - “Average execution time” Postderived metric with an expression:  
**metric::time(i)/metric::visits(e)**
  - “Number of FLOP per second” Postderived metric with an expression:  
**metric::FLOP()/metric::time()**

# Derived metrics in Cube GUI

Collection of derived metrics

Parameters of the derived metric

CubePL expression

# Example derived metric FLOPS based on PAPI\_FP\_OPS and time

The screenshot displays the Cube-4.3.1 interface for editing a derived metric. The left panel shows the configuration for the metric 'FLOPS', which is derived from 'PAPI\_FP\_OPS' and 'time'. The main window shows a metric tree where 'FLOPS' is highlighted with a value of 1.84e9. The tree also shows a breakdown of the 'exact\_rhs' metric, with a sub-metric '!'\$omp do @exact\_r...' highlighted at 9.65e8. The right panel shows a system tree for 'machine Linux' with a barplot view.

**Edit metric FLOPS (on froggy1)**

Select metric from collection: --- please select ---

Derived metric type: Postderived metric

Display name: FLOPS

Unique name: flops

Data type: DOUBLE

Unit of measurement:

URL:

Description:

Calculation: `metric::PAPI_FP_OPS()/metric::time()`

Share this metric with SCALASCA group

**Cube-4.3.1: scorep\_8x4\_sum/profile.cubex (on froggy1)**

Metric tree

- 1.17e7 Visits (occ)
- 1148.49 Time (sec)
- 0.00 Minimum Inclusive Time (sec)
- 41.57 Maximum Inclusive Time (...)
- 0 bytes\_put (bytes)
- 0 bytes\_get (bytes)
- 5.75e12 PAPI\_TOT\_INS (#)
- 2.69e12 PAPI\_TOT\_CYC (#)
- 2.12e12 PAPI\_FP\_OPS (#)
- 3.12e9 bytes\_sent (bytes)
- 3.12e9 bytes\_received (bytes)
- 1.84e9 FLOPS**

Call tree

- 3.17e5 MAIN\_
  - 7.04e5 mpi\_setup\_
    - 6.34e4 MPI\_Bcast
    - 2.05e5 env\_setup\_
      - 7.39e5 zone\_setup\_
        - 9.31e5 map\_zones\_
          - 9.39e4 zone\_starts\_
            - 6.16e5 set\_constants\_
              - 5.91e8 initialize\_
                - 0.00 exact\_rhs\_
                  - 145.62 !\$omp parallel @exac...
                    - 2.54e4 !\$omp do @exact\_r...
                      - 9.65e8 !\$omp do @exact\_r...**
                      - 9.62e8 !\$omp do @exact\_r...
                      - 8.14e8 !\$omp do @exact\_r...
                      - 1.21e5 !\$omp do @exact\_r...
                      - 0.00 !\$omp implicit barrier...
                    - 6.23e4 exch\_qbc\_
                      - 1.94e9 adi\_
                        - 2.19e5 MPI\_Barrier
                        - 1.92e9 <<bt\_iter>> (200 itera...
                        - 1.98e8 verify\_
                          - 1.05e5 MPI\_Reduce

System tree

      - machine Linux
        - node frog6
          - MPI Rank 0
            - 1.17e9 Master thread
            - 9.43e8 OMP thread 1
            - 9.47e8 OMP thread 2
            - 9.47e8 OMP thread 3
          - MPI Rank 1
            - 1.17e9 Master thread
            - 9.87e8 OMP thread 1
            - 9.68e8 OMP thread 2
            - 9.72e8 OMP thread 3
          - MPI Rank 2
            - 1.10e9 Master thread
            - 8.97e8 OMP thread 1
            - 8.77e8 OMP thread 2
            - 8.76e8 OMP thread 3
          - MPI Rank 3
            - 1.09e9 Master thread
            - 9.06e8 OMP thread 1
            - 9.04e8 OMP thread 2
            - 9.02e8 OMP thread 3

All (32 elements)

Selected "!"\$omp do @exact\_rhs.f:46"

# Context-sensitive help

The screenshot displays the 'cube 4.1.1' application window. The 'Help' menu is open, showing options like 'Getting started', 'Mouse and keyboard control', 'What's This?' (highlighted), and 'About'. A blue callout box points to the 'What's This?' option with the text 'Context-sensitive help available for all GUI items'. The main window shows a 'Metric tree' on the left, a central 'System tree' with a 'Box Plot' view, and a bottom status bar. The status bar shows a color gradient and the text 'Change into help mode for display components'.

cube 4.1.1 livedvd2: scorep-20120913\_1740\_557443655223384/profile.cubex

File Display Topology Help

Absolute

Metric tree

- 1.63e9 Visits
- 767.48 Time
- 0.00 Minimum I
- 48.58 Maximum
- 5.27e8 byt
- 5.27e8

System tree

- generic cluster
  - i06r01c20
    - MPI Rank 0
      - 44.10 CPU thread 0
      - 43.94 CPU thread 1
      - 43.58 CPU thread 2
      - 34.36 CPU thread 3
    - MPI Rank 1
      - 44.48 CPU thread 0
      - 35.09 CPU thread 1
      - 44.18 CPU thread 2
      - 43.71 CPU thread 3
    - MPI Rank 2
      - 44.47 CPU thread 0
      - 43.49 CPU thread 1
      - 43.86 CPU thread 2
      - 35.18 CPU thread 3
    - MPI Rank 3
      - 44.74 CPU thread 0
      - 44.00 CPU thread 1
      - 43.73 CPU thread 2
      - 35.64 CPU thread 3

0.00 767.48 (100.00%) 767.48

0.00 668.54 (87.11%) 767.48

0.00 668.54

Change into help mode for display components

# CUBE algebra utilities

---

- Extracting solver sub-tree from analysis report

```
% cube_cut -r '<<ITERATION>>' scorep_bt-mz_B_8x8_sum/profile.cubex  
Writing cut.cubex... done.
```

- Calculating difference of two reports

```
% cube_diff scorep_bt-mz_B_8x8_sum/profile.cubex cut.cubex  
Writing diff.cubex... done.
```

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



# Loop Unrolling

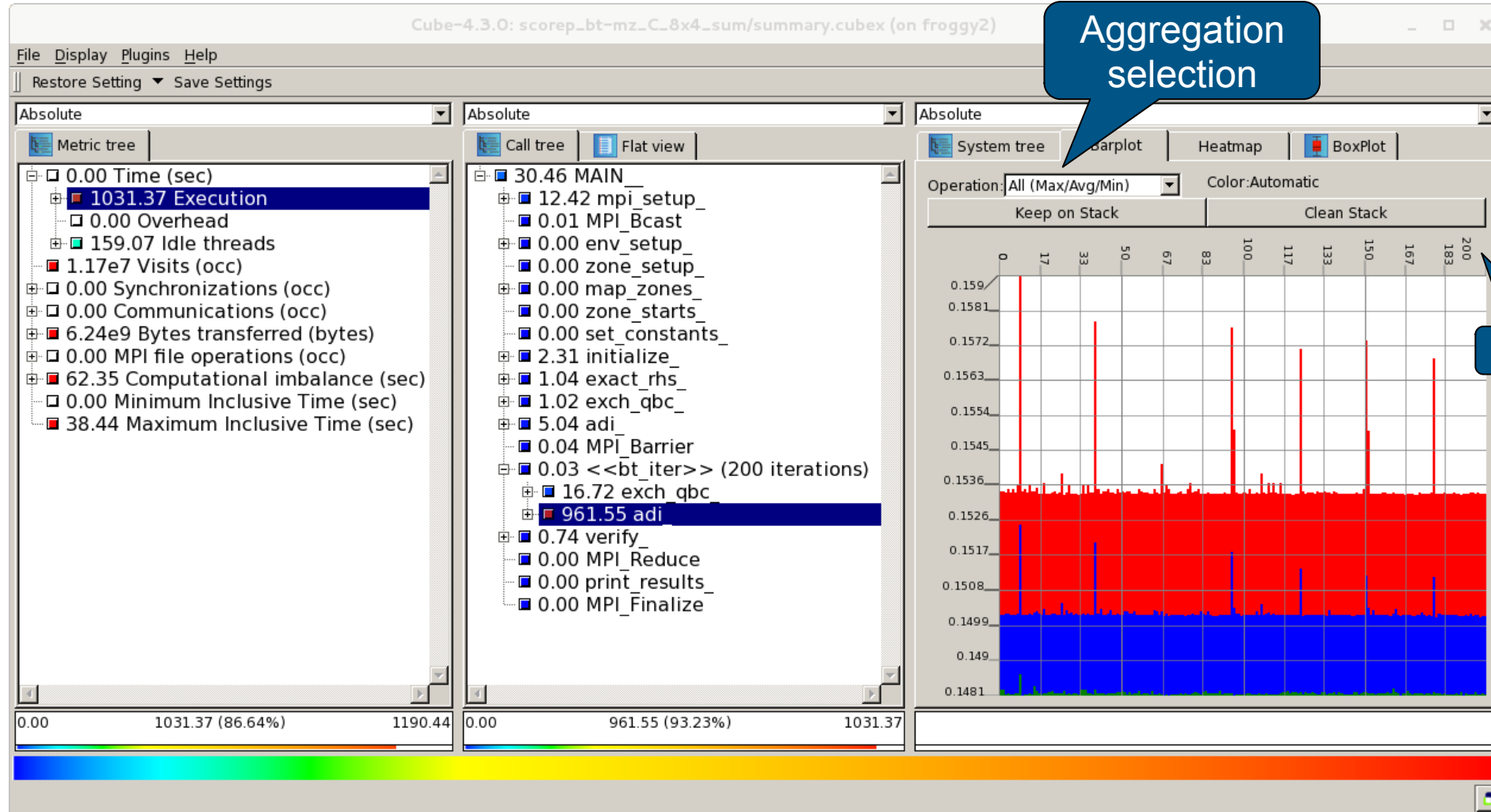
---

- Show time dependent behavior by unrolling iterations
- Preparations:
  - Mark loops by using Score-P user instrumentation in your source code

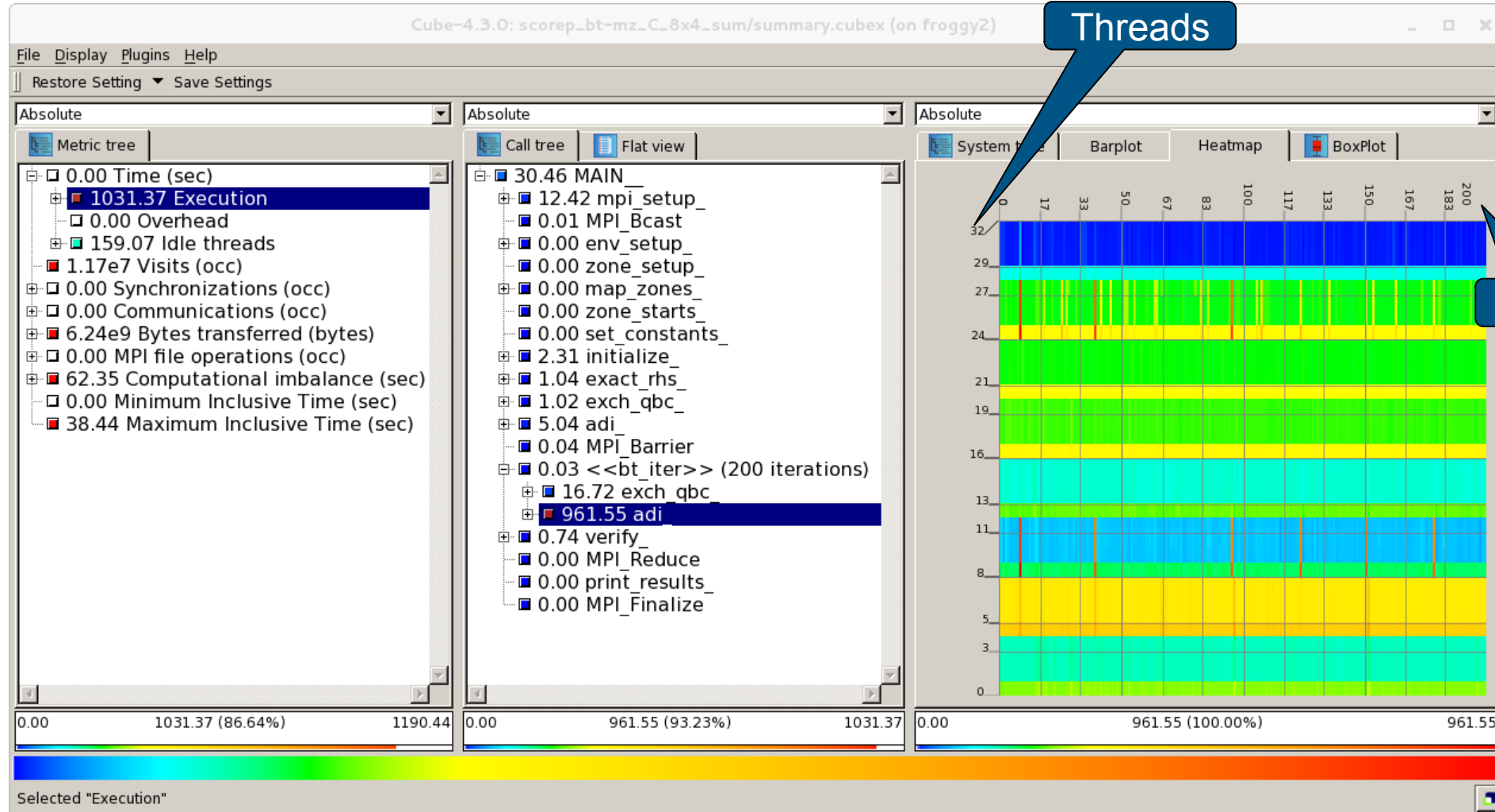
```
SCOREP_USER_REGION_BEGIN( scorep_bt_loop, "<<bt_iter>>", SCOREP_USER_REGION_TYPE_DYNAMIC )
```

- Result in the CUBE profile:
  - Iterations shown as separate call trees
    - Useful for checking results for specific iterations
  - or
  - Select your user instrumented region and mark it as loop
  - Choose hide iterations
    - View the Barplot statistics or the (thread x iterations) Heatmap

# Loop Unrolling - Barplot



# Loop Unrolling – Heatmap



## Further information

---

### CUBE

- Parallel program analysis report exploration tools
  - Libraries for XML report reading & writing
  - Algebra utilities for report processing
  - GUI for interactive analysis exploration
- Available under New BSD open-source license
- Documentation & sources:
  - <http://www.scalasca.org>
- User guide also part of installation:
  - ``cube-config --cube-dir`/share/doc/CubeGuide.pdf`
- Contact:
  - mailto: [scalasca@fz-juelich.de](mailto:scalasca@fz-juelich.de)

