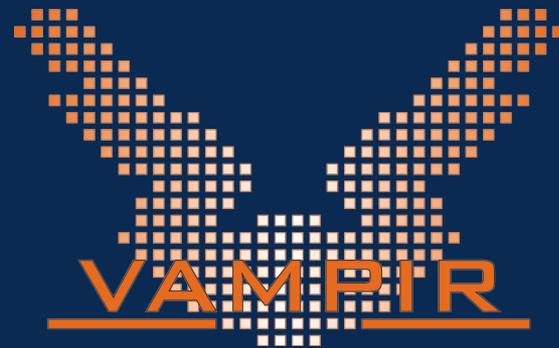


Score-P & Vampir

Comprehensive Multi-Paradigm Performance Analysis



Ronny Brendel
ORNL / TU Dresden
brendelr@ornl.gov

Argonne Training Program on Extreme-Scale Computing (ATPESC), St. Charles,
Illinois, 8th August 2017

Introduction > Methods

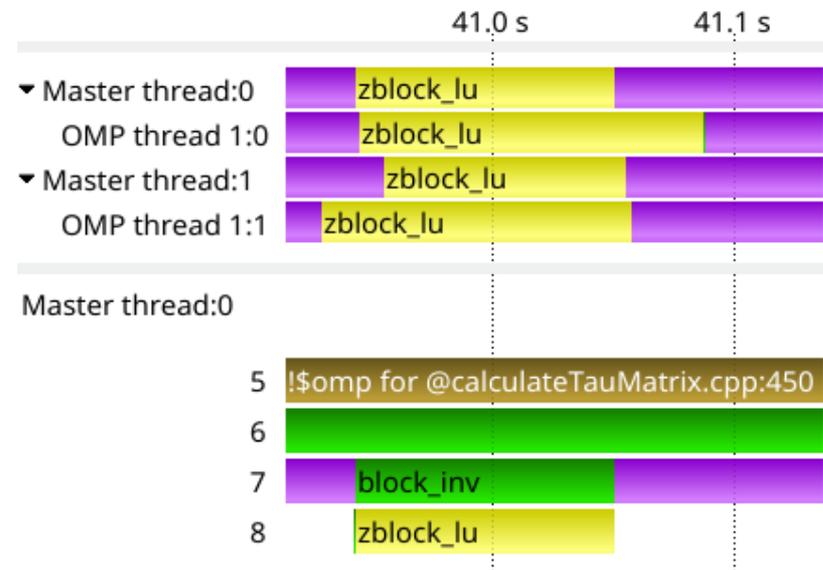
Profile

- Information accumulated into buckets
- Typically small overhead
- Static representation

Time (%)	Time (s)	Function name
5.44	1.21	QListData::isEmpty
2.96	0.66	QHash::findNode
2.67	0.60	QList::last
1.71	0.38	handleEnter
0.58	0.13	QHash::find

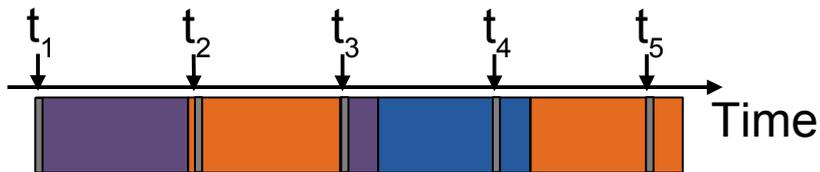
Trace

- Event log
- Possibly large overhead
- Interactive presentation



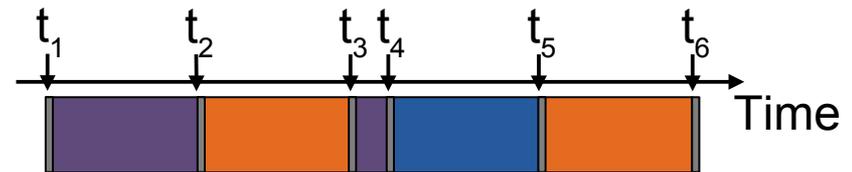
Introduction > Methods

Sampling



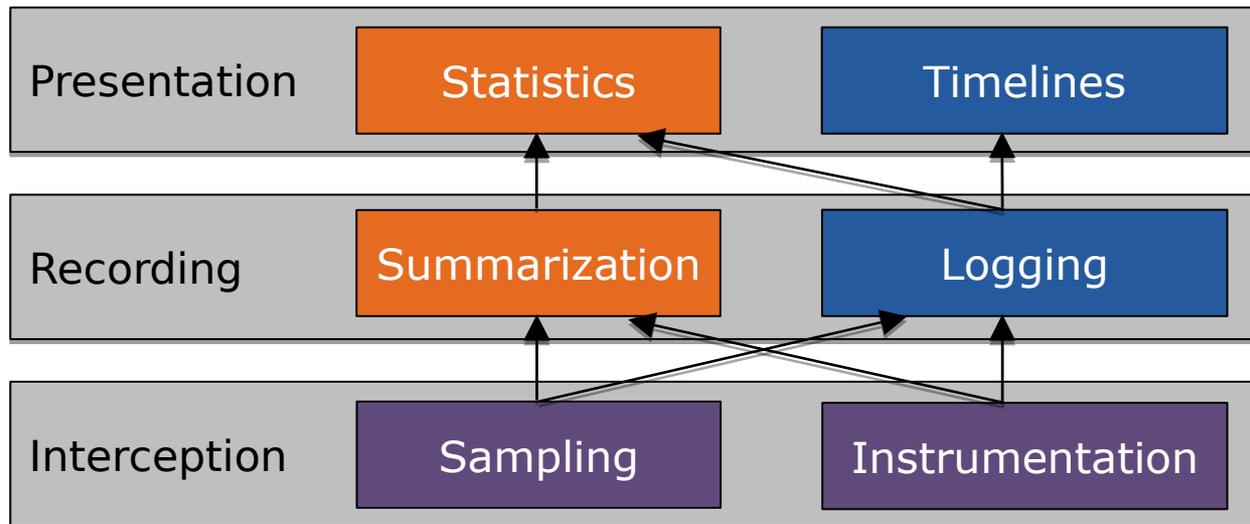
- Interrupt with given interval (typically $\sim 10\text{ms}$)
- Statistical guarantees

Instrumentation

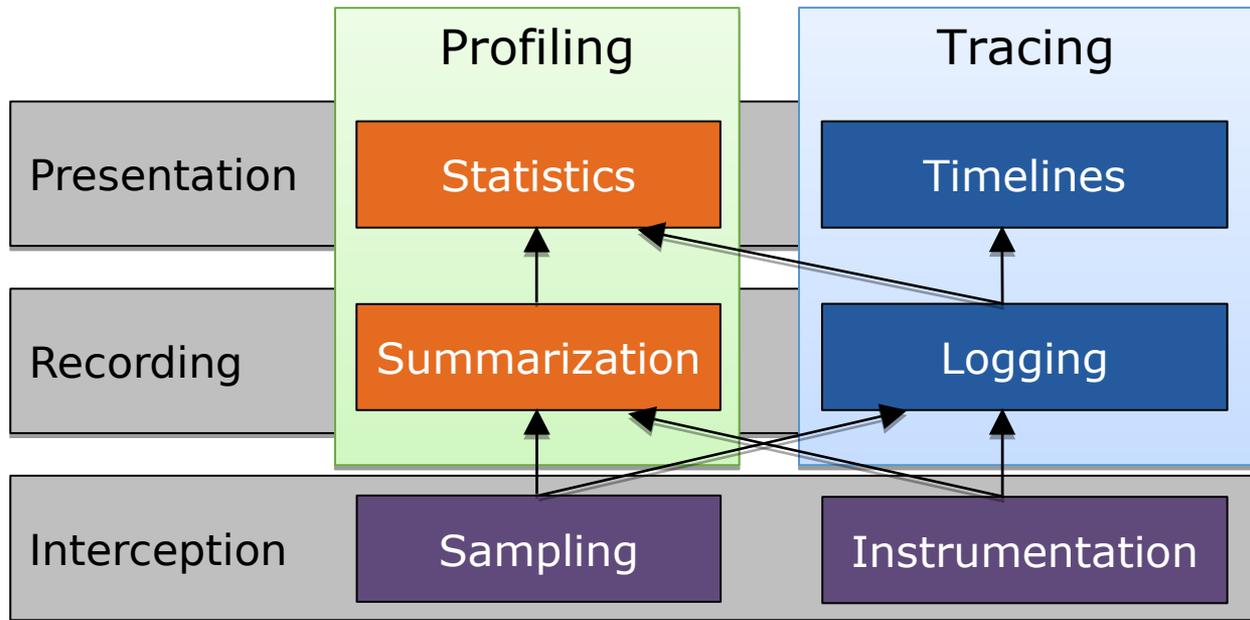


- Callback before/after event
- Exact time and call counts
- Wrappers have access to function arguments

Introduction > Methods



Introduction > Methods



Introduction > Methods

- Myth 1: Tracing has a giant overhead
 - **It depends on the event rate**
 - E.g. an MPI-only trace has very low overhead
 - Admittedly:
 - Main problem 1: Compiler instrumentation
 - → Compiler plugins to the rescue
 - Wrappers are mostly fine and widely used
 - Main problem 2: Filtering workflows are inconvenient
 - Tool-specific problem, not a general „Tracing“-problem

Introduction > Methods

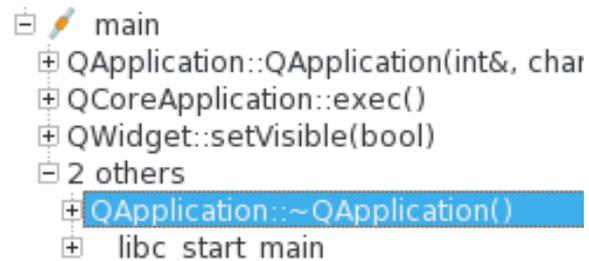
- Myth 2: Tracing produces giant recordings
 - Analogous to Myth 1: It depends on the number of events
 - Score-P has a simple filtering workflow that copes with that
 - The majority of our archived tracefiles are below 1 GB
- Personally, I always configure Score-P so that the trace is easy to handle on my laptop

Introduction > Methods

- Myth 3: One technique is superior
 - **Not true**
- Sampling:
 - Does not give an absolutely accurate picture of a run

```
int main(int argc,
         char** args ) {
    QApplication app(argc,
                    argv);
    QWidget w;
    QImage i;
    w.show();
    return app.exec();
}
```

Sampling



Instrumentation



Introduction > Methods

- Myth 3: One technique is superior
 - **Not true**
- Sampling:
 - Does not give an absolutely accurate picture of a run
 - Cannot count function calls
 - Cannot record exact timings
 - Cannot record exact performance counters
 - It is **statistical sampling**
- It cannot capture semantics of APIs, i.e. it cannot follow API usage and analyze passed arguments, e.g. transferred bytes

Introduction > Methods

- Myth 3: One technique is superior
 - **Not true**
- Instrumentation/Tracing:
 - Typically more difficult than using a profiler
 - Does not guarantee anything about overhead or recording size
 - (But it's not inconceivable to achieve this)

Introduction > Methods

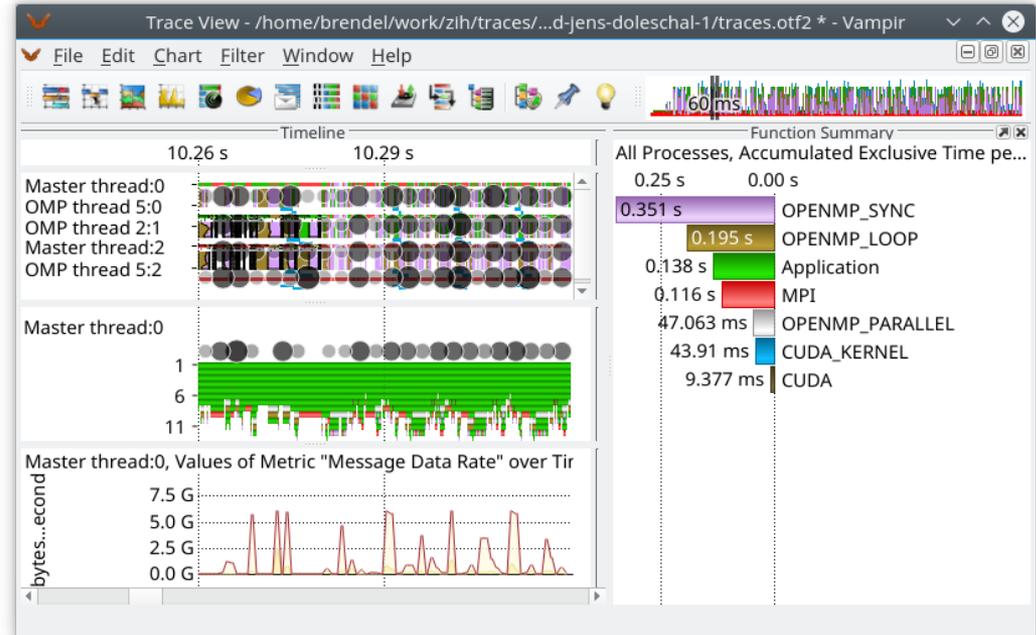
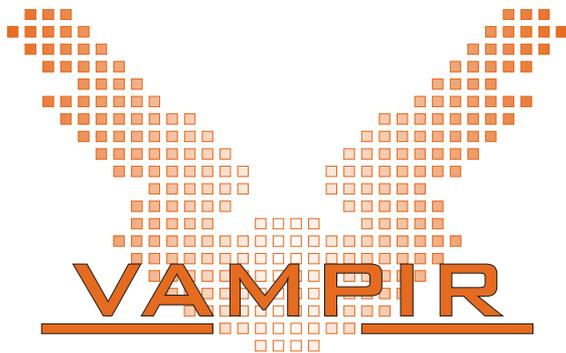
- In practice, most tools use a combination
 - Coarse-grained sampling + call stack unwinding
 - Wrappers for library functions of interest (MPI_Send, cudaMalloc, dlopen)

Introduction

Introduction > Vampir

<http://vampir.eu>

- Comprehensive, powerful performance data visualization
- Developed since 1996
- Commercial



Introduction > Score-P

<http://score-p.org>

- Jointly developed next-generation performance data collector
- Developed since 2009
- Open-source (3-clause BSD)
- Partners:

- TU Dresden, GER



- FZ Jülich, GER



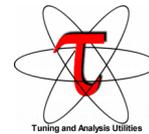
- TU München, GER

- University of Oregon, USA

- RWTH Aachen; TU Darmstadt;

Gesellschaft für numerische Simulation mbH;

German Research School for Simulation Sciences GmbH (all GER)

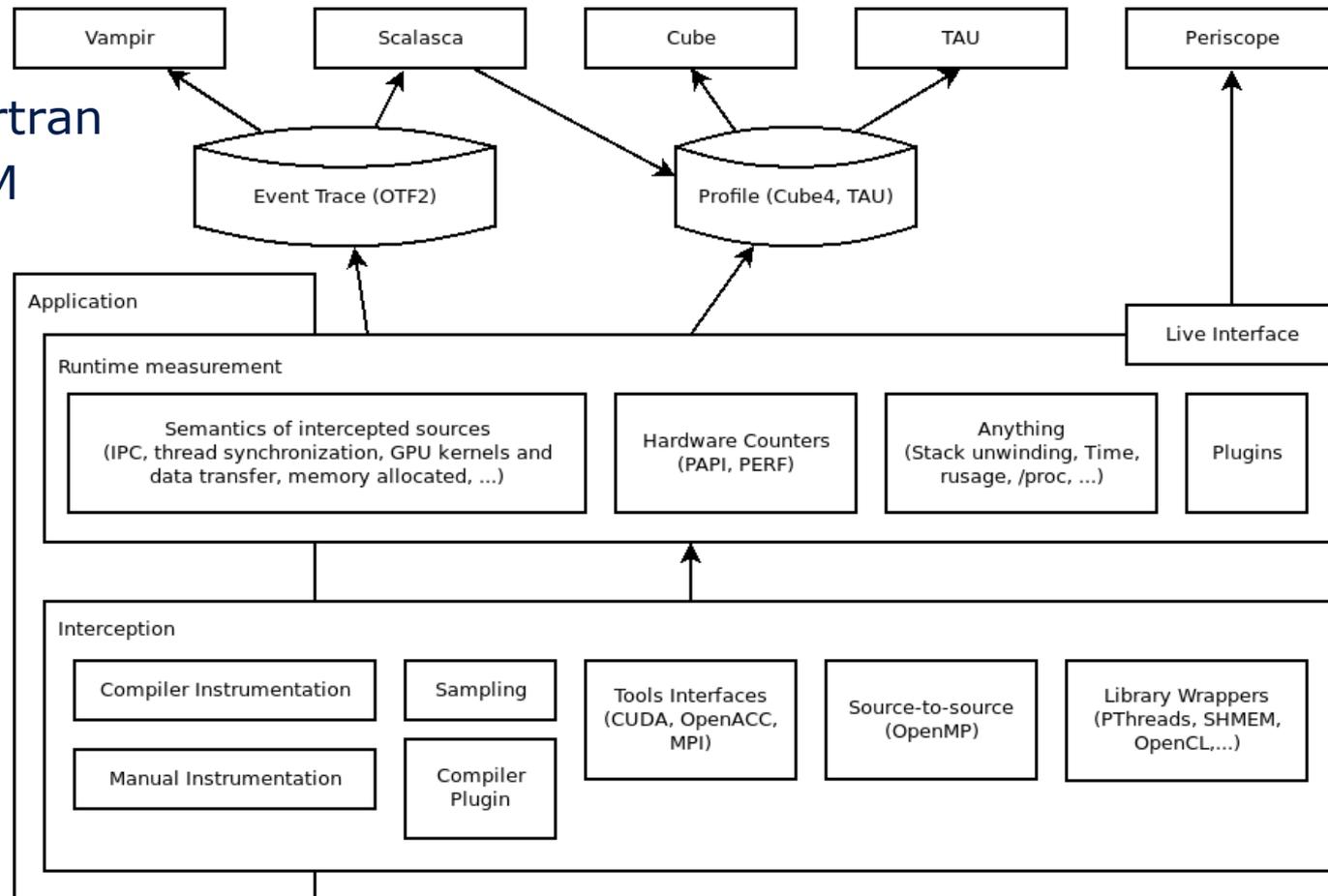


Introduction > Score-P

- Supports:

- C, C++, Fortran
- MPI, SHMEM
- OpenMP, PThreads
- CUDA, OpenACC, OpenCL

- Compilers: Cray, GNU, IBM, Intel, Pathscale, PGI, LLVM



Tutorial

Tutorial > Usage

- Load Score-P (ANL)

```
$ module load scorep
```

```
$ echo "+vampir" >> ~/.soft && resoft
```

- Compile & Link (with MPI) (with SHMEM)

```
$ scorep ... gcc ... main.c
```

```
$ scorep mpicc main.c
```

```
$ scorep oshcc main.c
```

- CMake

```
$ SCOREP_WRAPPER=OFF cmake -DCMAKE_C_COMPILER=scorep-gcc ..  
$ SCOREP_WRAPPER_INSTRUMENTER_FLAGS="..." SCOREP_WRAPPER_COMPILER_FLAGS="..." make
```

- Autotools

```
$ SCOREP_WRAPPER=OFF ../configure CC=scorep-gcc MPICC=scorep-mpicc ..  
$ SCOREP_WRAPPER_INSTRUMENTER_FLAGS="..." SCOREP_WRAPPER_COMPILER_FLAGS="..." make
```

Tutorial > Usage

- Execute

```
$ ./a.out
```

```
$ mpirun -np 2 ./a.out
```

```
$ shmemrun -np 2 ./a.out
```

- Inspect

```
$ ls -R
scorep-20170323_1309_7243761919249966  a.out

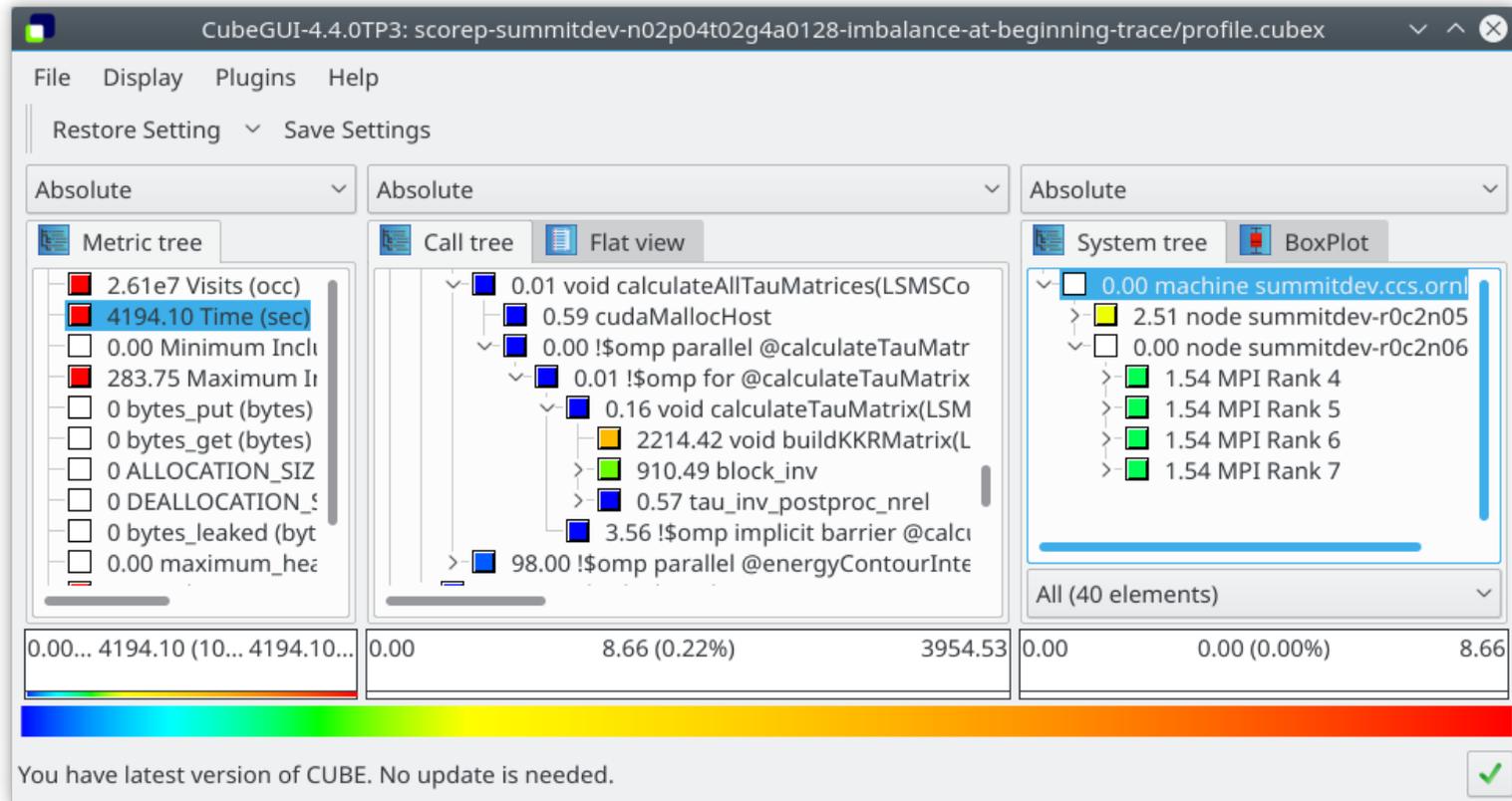
./scorep-20170323_1309_7243761919249966:
profile.cubex  scorep.cfg
```

- Inspect > Cube

```
$ cube scorep-20170323_1309_7243761919249966/profile.cubex
```

Tutorial > Usage

- Inspect > Cube



CubeGUI-4.4.0TP3: scorep-summitdev-n02p04t02g4a0128-imbalance-at-beginning-trace/profile.cubex

File Display Plugins Help
 Restore Setting Save Settings

Metric tree (Absolute):

- 2.61e7 Visits (occ)
- 4194.10 Time (sec)**
- 0.00 Minimum Incl
- 283.75 Maximum Ir
- 0 bytes_put (bytes)
- 0 bytes_get (bytes)
- 0 ALLOCATION_SIZ
- 0 DEALLOCATION_
- 0 bytes_leaked (byt
- 0.00 maximum_hez

Call tree (Absolute):

- 0.01 void calculateAllTauMatrices(LSMSCo
 - 0.59 cudaMallocHost
 - 0.00 !\$omp parallel @calculateTauMatr
 - 0.01 !\$omp for @calculateTauMatrix
 - 0.16 void calculateTauMatrix(LSM
 - 2214.42 void buildKKRMatrix(L
 - 910.49 block_inv
 - 0.57 tau_inv_postproc_nrel
 - 3.56 !\$omp implicit barrier @calci
 - 98.00 !\$omp parallel @energyContourInte

System tree (Absolute):

- 0.00 machine summitdev.ccs.ornl
 - 2.51 node summitdev-r0c2n05
 - 0.00 node summitdev-r0c2n06
 - 1.54 MPI Rank 4
 - 1.54 MPI Rank 5
 - 1.54 MPI Rank 6
 - 1.54 MPI Rank 7

All (40 elements)

0.00... 4194.10 (10... 4194.10... | 0.00 | 8.66 (0.22%) | 3954.53 | 0.00 | 0.00 (0.00%) | 8.66

You have latest version of CUBE. No update is needed.

Tutorial > Usage

- Runtime Options

- Profiling (default)
- Tracing

```
$ export SCOREP_ENABLE_PROFILING=true
```

```
$ export SCOREP_ENABLE_TRACING=true
```

- Performance counters
- Filtering

```
$ export SCOREP_METRIC_PAPI=PAPI_L2_TCM,...
```

```
$ export SCOREP_FILTERING_FILE=my.filt
```

- Memory (default: 16M)

```
$ export SCOREP_TOTAL_MEMORY=1G
```

- And many more...

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

Tutorial > Usage

- Inspect > Vampir

```
$ export SCOREP_ENABLE_PROFILING=false
$ export SCOREP_ENABLE_TRACING=true
$ export SCOREP_METRIC_PAPI=PAPI_TOT_INS,PAPI_TOT_CYC

$ mpirun -np 4 ./a.out

$ ls -R
scorep-20170323_1309_7243761919249966  a.out

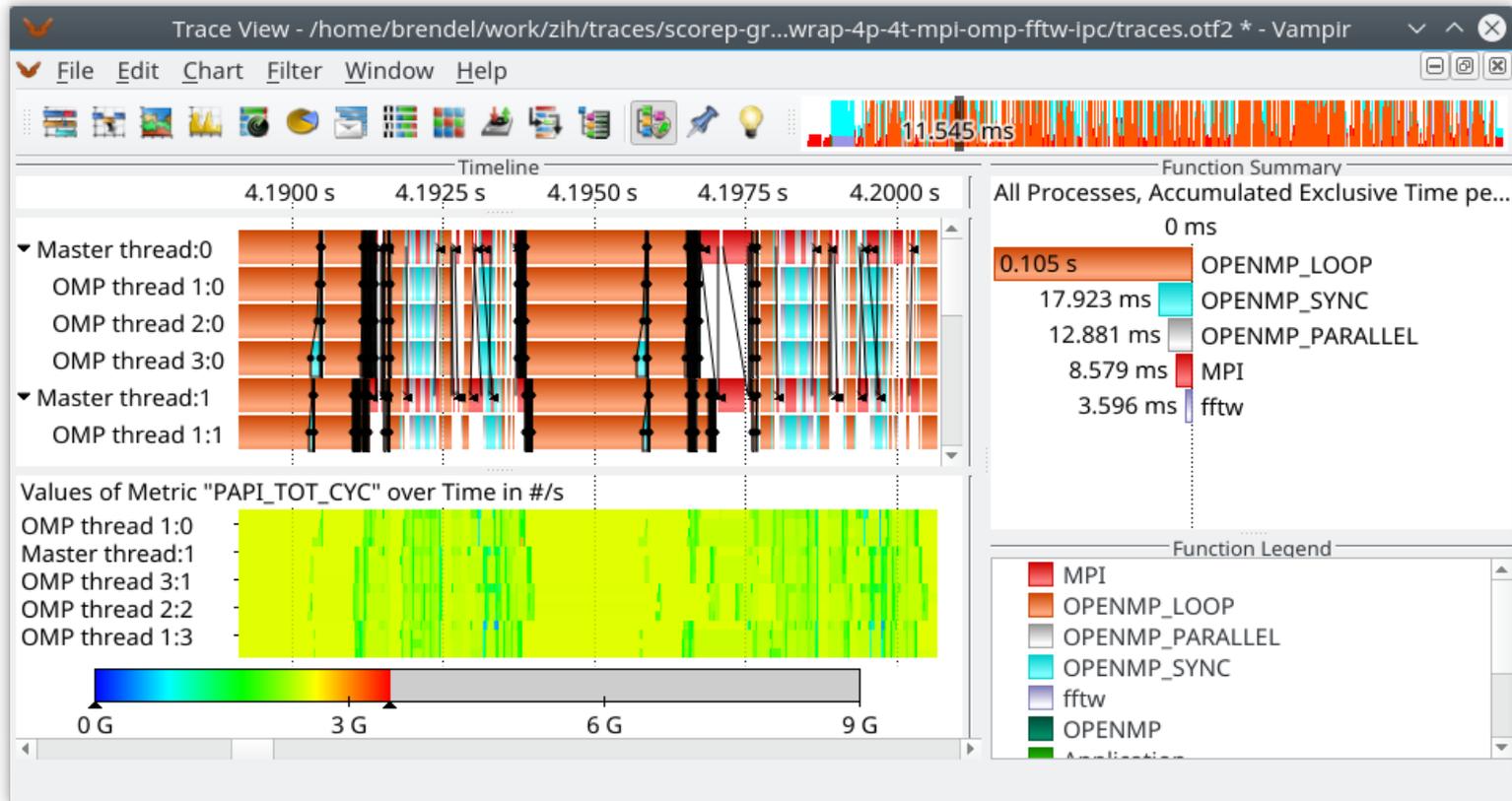
./scorep-20170323_1309_7243761919249966:
scorep.cfg  traces/  traces.def  traces.otf2

$ module load vampir

$ vampir scorep-20170323_1309_7243761919249966/traces.otf2
```

Tutorial > Usage

- Inspect > Vampir



Tutorial > Overhead

- Trace size and overhead varies greatly with event rate
 - Make a reference run and check wall clock time!
 - Rule of thumb: Try to stay below 10% overhead
- Filtering is an integral part of Score-P's workflow

Tutorial > Profiling Workflow

- 1) Instrument & build
- 2) Execute
- 3) Analyze profile using Cube

Mind that overhead can be too high. Runtime filtering does not help, if the event rate is extremely high

→ Compile-time filtering with our GCC instrumentation compiler plugin solves this

```
$ scorep --instrument-filter=<filter_file> gcc main.c
```

Tutorial > Tracing Workflow

- 1) Instrument & build
- 2) Execute (profiling)
- 3) Analyze overhead
 - If the estimated trace size is too large, filter and goto (3)
- 4) Execute using the filter (tracing)
- 5) Analyze trace using Vampir

Tutorial > Tracing Workflow

3) Analyze Overhead

```
$ scorep-score scorep-20170323_1309_7243761919249966/profile.cubex
```

```
Estimated aggregate size of event trace: 40GB
```

```
Estimated requirements for largest trace buffer (max_buf): 6GB
```

```
Estimated memory requirements (SCOREP_TOTAL_MEMORY): 6GB
```

```
(warning: The memory requirements cannot be satisfied by Score-P to avoid intermediate flushes when tracing. Set SCOREP_TOTAL_MEMORY=4G to get the maximum supported memory or reduce requirements using USR regions filters.)
```

flt	type	max_buf[B]	visits	time[s]	time[%]	time/visit[us]	region
	ALL	5,383,272,006	1,635,443,611	579.23	100.0	0.35	ALL
	USR	5,358,738,138	1,631,138,913	253.00	43.7	0.16	USR
	OMP	23,580,522	4,089,856	318.79	55.0	77.95	OMP
	COM	665,210	182,120	0.90	0.2	4.95	COM
	MPI	288,136	32,722	6.55	1.1	200.11	MPI

Tutorial > Tracing Workflow

3) Analyze Overhead

```
$ scorep-score -r scorep-20170323_1309_7243761919249966/profile.cubex
[...]
```

flt	type	max_buf[B]	visits	time[s]	time[%]	time/visit[us]	region
	ALL	5,383,272,006	1,635,443,611	579.23	100.0	0.35	ALL
	USR	5,358,738,138	1,631,138,913	253.00	43.7	0.16	USR
	OMP	23,580,522	4,089,856	318.79	55.0	77.95	OMP
	COM	665,210	182,120	0.90	0.2	4.95	COM
	MPI	288,136	32,722	6.55	1.1	200.11	MPI
	USR	1,716,505,830	522,844,416	79.32	13.7	0.15	matmul_sub_
	USR	1,716,505,830	522,844,416	53.44	9.2	0.10	matvec_sub_
	USR	1,716,505,830	522,844,416	111.47	19.2	0.21	binvrhs_
	USR	76,195,080	22,692,096	2.76	0.5	0.12	binvrhs_
	USR	76,195,080	22,692,096	4.37	0.8	0.19	lhsinit_
	USR	56,825,184	17,219,840	1.63	0.3	0.09	exact_solution_

Tutorial > Tracing Workflow

3) Filter

```
$ cat myfilter.filt
SCOREP_REGION_NAMES_BEGIN
  EXCLUDE
    matmul_sub*
    matvec_sub*
    binvcrhs*
    Binvrhs*
    exact_solution*
    lhs*init*
    timer_*
SCOREP_REGION_NAMES_END

$ scorep-score -f myfilter.filt scorep-20170323*/profile.cubex

Estimated aggregate size of event trace:                409MB
Estimated requirements for largest trace buffer (max_buf): 58MB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):    70MB
(hint: When tracing set SCOREP_TOTAL_MEMORY=70M to avoid
[...])
```

Tutorial > Tracing Workflow

4) Execute using the filter (Tracing)

```
$ export SCOREP_ENABLE_TRACING=true
$ export SCOREP_TOTAL_MEMORY=70M
$ export SCOREP_FILTERING_FILE=myfilter.filt

$ mpirun -np 8 ./a.out
```

4) GCC-only: Compile-time filtering

```
$ scorep --instrument-filter=myfilter.filt gcc main.c

$ export SCOREP_ENABLE_TRACING=true
$ export SCOREP_TOTAL_MEMORY=70M

$ mpirun -np 8 ./a.out           # no runtime filtering needed
```

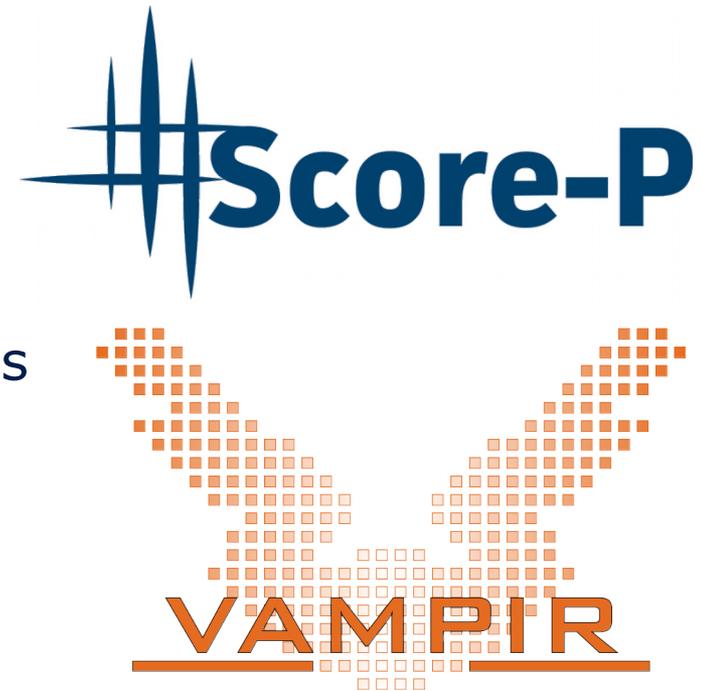
Tutorial > Vampir Demo (Live)

Tutorial > Getting Help

- `$ scorep --help`
 - `$ scorep-wrapper --help`
 - `$ scorep-info config-vars`
- <http://score-p.org>
<http://vampir.eu>
- **Manuals:** `SCOREP_DIR/share/doc/scorep/pdf/scorep.pdf`
`VAMPIR_ROOT/doc/vampir-manual.pdf`
 - <https://www.alcf.anl.gov/vampir>
 - support@score-p.org, service@vampir.eu
 - VI-HPS offers trainings (Invite them!)
 - <http://www.vi-hps.org/training/tws/>
 - <http://www.vi-hps.org/training/material/>

Conclusions

- Holistic, powerful and detailed software performance analysis
 - Everything in one picture
 - Extremely customizable
 - Extremely scalable
 - Advanced features
 - Very active in adopting new features
- Active research community
- Continuously selected by OLCF
- **Enabler for science at extreme scale**



Future Work

- User library wrapping
- I/O analysis (POSIX, ADIOS, HDF5, NetCDF)
- Non-volatile memory analysis
- POWER8/9 & Clang support
- Instrumentation compiler plugin for LLVM
- OpenMP tools interface (OMPT)
- MPI RMA analysis
- KNL-specific metrics and topology information



Sponsors & Projects



Horizon 2020
European Union funding
for Research & Innovation



Contributors

- Score-P

Andreas Knüpfer, Bert Wesarg, Christian Feld, Christian Herold, Daniel Lorenz, Dirk Schmidl, Dominic Eschweiler, Felix Schmitt, Frank Winkler, Ilya Zhukov, Johannes Spazier, Johannes Ziegenbalg, Marc Schlütter, Markus Geimer, Michael Knobloch, Michael Wagner, Pavel Saviankou, René Jäkel, Robert Dietrich, Robert Mijaković, Robert Schöne, Robin Geyer, Ronny Brendel, Ronny Tschüter, Sameer Shende, Scott Biersdorff, Sebastian Döbel, Sebastian Oeste, Suzanne Millstein, Thomas Ilsche, Yury Oleynik

- Vampir

Andreas Knüpfer, Bert Wesarg, Frank Winkler, Hartmut Mix, Heide Rohling, Holger Brunst, Jens Doleschal, Matthias Weber, Laszlo Barabas, Michael Heyde, Michael Peter, Reinhard Neumann, Ronald Geisler, Ronny Brendel, Thomas William

Hands-On

- Prepared example:

```
ssh -Y titan
```

```
cp -r /lustre/atlas/world-shared/stf010/brendel/heat ~
```

```
cd ~/heat
```

```
less instructions.txt
```

- <https://www.alcf.anl.gov/vampir>
 - **Please install Vampir on your laptop**
- (https://www.olcf.ornl.gov/kb_articles/software-scorep/)