VI-HPS

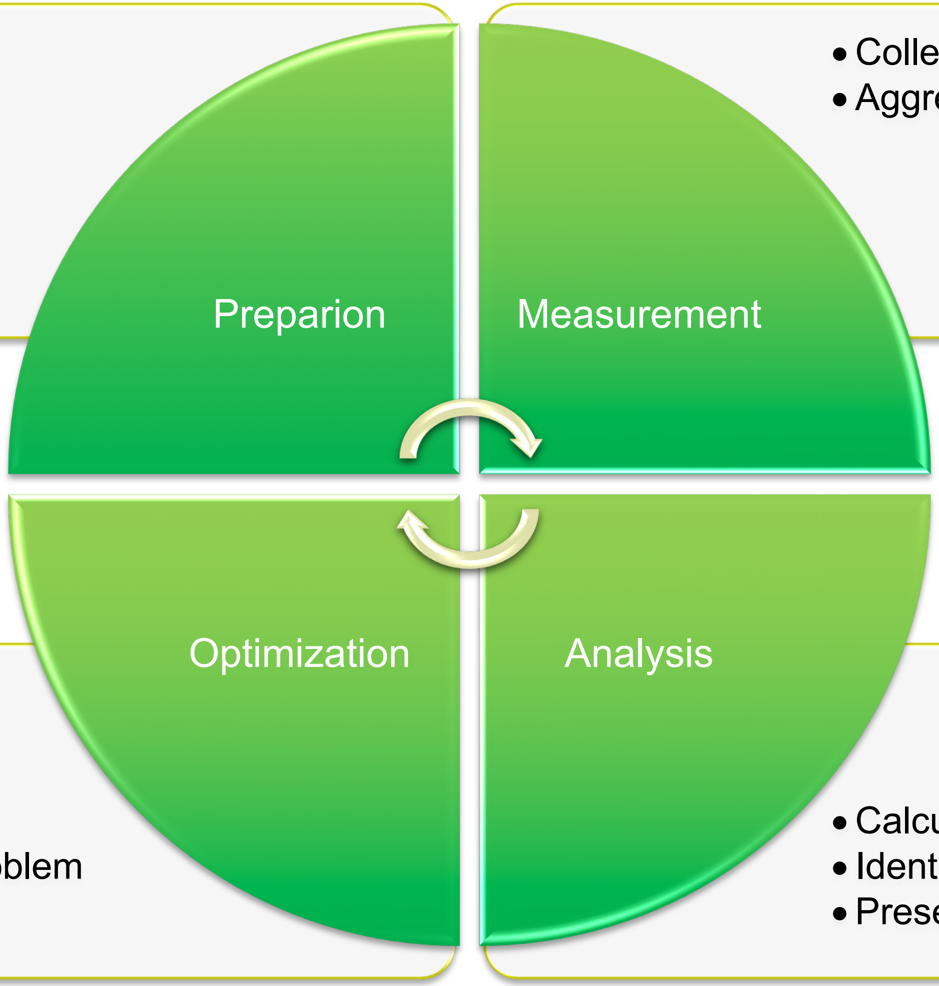
VIRTUAL INSTITUTE – HIGH PRODUCTIVITY SUPERCOMPUTING

Measurement with Score-P

Bill Williams, TU Dresden

Score-P

Performance engineering workflow



Preparion

- Prepare application with symbols
- Insert extra code (probes/hooks)

Measurement

- Collection of performance data
- Aggregation of performance data

Analysis

- Calculation of metrics
- Identification of performance problems
- Presentation of results

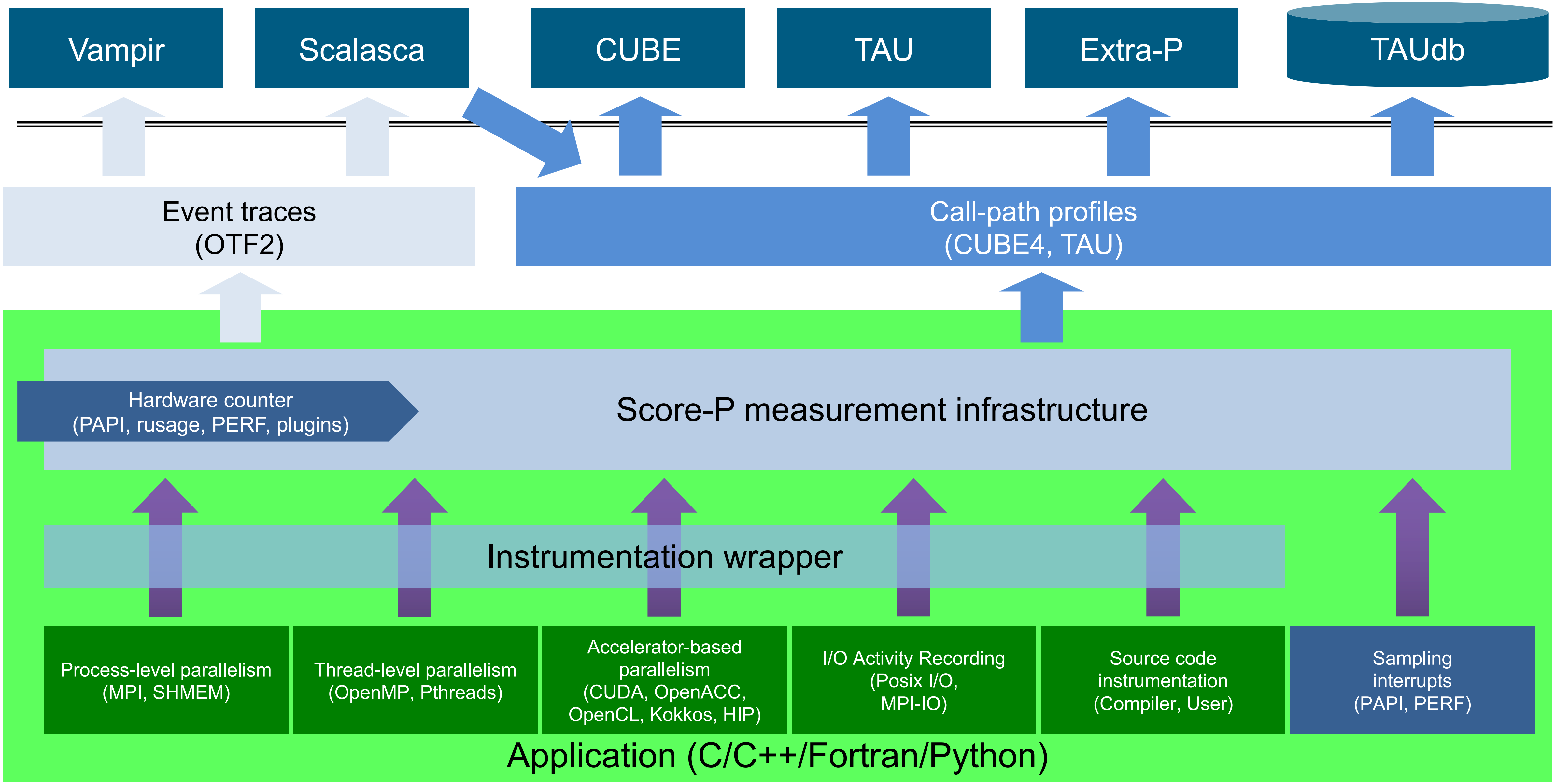
Optimization

- Modifications intended to
- eliminate/reduce performance problem

Score-P

- Infrastructure for instrumentation and performance measurements
- Instrumented application can be used to produce several results:
  - Call-path profiling: CUBE4 data format used for data exchange
  - Event-based tracing: OTF2 data format used for data exchange
- Supported parallel paradigms:
  - Multi-process: MPI, SHMEM
  - Thread-parallel: OpenMP, Pthreads
  - Accelerator-based: CUDA, ROCm, OpenCL, OpenACC
- Open Source; portable and scalable to all major HPC systems
- Initial project funded by BMBF
- Close collaboration with PRIMA project funded by DOE

GEFÖRDERT VOM
Bundesministerium
für Bildung
und Forschung


Vampir
Scalasca
CUBE
TAU
Extra-P
TAUdb
Event traces
(OTF2)
Call-path profiles
(CUBE4, TAU)
Hardware counter
(PAPI, rusage, PERF, plugins)
Score-P measurement infrastructure
Instrumentation wrapper
Process-level parallelism
(MPI, SHMEM)
Thread-level parallelism
(OpenMP, Pthreads)
Accelerator-based
parallelism
(CUDA, OpenACC,
OpenCL, Kokkos, HIP)
I/O Activity Recording
(Posix I/O,
MPI-IO)
Source code
instrumentation
(Compiler, User)
Sampling
interrupts
(PAPI, PERF)
Application (C/C++/Fortran/Python)

Partners

- Forschungszentrum Jülich, Germany
- Gesellschaft für numerische Simulation mbH Braunschweig, Germany
- RWTH Aachen, Germany
- Technische Universität Darmstadt, Germany
- Technische Universität Dresden, Germany
- Technische Universität München, Germany
- University of Oregon, Eugene, USA

Reference hands-on: NPB-MZ-MPI / BT

Performance analysis steps

- Reference preparation for validation

- Program instrumentation
- Summary measurement collection

- Summary experiment scoring
- Trace measurement collection with filtering

NPB-MZ-MPI / BT suite

```
% cd $VIHPS_WORKSPACE
% mkdir hands-on && cd hands-on
% tar xvzf $VIHPS_ROOT/hands-on/score-p.tar.gz
% cd score-p
% ls
bin/
bin.scorep/
BT-MZ/
common/
config/
jobscript/
LU-MZ/
Makefile
README
README.install
README.tutorial
SP-MZ/
sys/
```

- The NAS Parallel Benchmark suite (MPI+OpenMP version)
  - http://www.nas.nasa.gov/Software/NPB
- Start in the $VIHPS_WORKSPACE/hands-on/score-p directory

NPB-MZ-MPI / BT configuration

```
% <editor> config/make.def
...
MPIF77 = mpif77 -f77=ifort
...
#---------------------------------------------------------------
# Global *compile time* flags for Fortran programs
#---------------------------------------------------------------
FFLAGS  = -O3 -g $(OPENMP) -mavx -msse4.2 -march=sapphirerapids
```

- Specify classic ifort
- Tuning flags for Sapphire Rapids (not a huge difference for BT-MZ, but good practice!)

NPB-MZ-MPI / BT build

```
% make bt-mz CLASS=C NPROCS=4
cd BT-MZ; make CLASS=C NPROCS=4 VERSION=
make: Entering directory 'BT-MZ'
cd ../sys; cc  -o setparams setparams.c -lm
../sys/setparams bt-mz 4 C
mpif77 -c  -O3 -fopenmp bt.f
 [...]
cd ../common;  mpif77 -c  -O3 -fopenmp timers.f
mpif77 -O3 -fopenmp -o ../bin/bt-mz_C.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_C.4
make: Leaving directory 'BT-MZ‘
```

- Benchmark name:
  - **bt-mz**, lu-mz, sp-mz
- Number of MPI processes:
  - NPROCS=**4**
- Benchmark class:
  - S, W, A, B, **C**, D, E
  - CLASS=**C**

NPB-MZ-MPI / BT job submission

```
% cp jobscript/[barnard|claix-2023]/bt-mz.sbatch .
% cat bt-mz.sbatch

# SBATCH -J reference
...
# Generic OpenMP thread pinning
export OMP_PROC_BIND=close
export OMP_PLACES=cores
...

% sbatch bt-mz.sbatch
```

- Bring appropriate job script into main benchmark directory
- Note the job name (used to sort output) and the OpenMP thread pinning variables (for your own codes)
- Note the output locations (site-specific!)
- Run with workshop account and reservation (

NPB-MZ-MPI / BT reference execution

```
% cat reference/bt-mz.out
 NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP \
>Benchmark

 Number of zones:   16 x   16
 Iterations: 200    dt:   0.000100
 Number of active processes:     4

 Use the default load factors with threads
 Total number of threads:     48  (  12.0 threads/process)

 Calculated speedup =     47.99

 Time step    1
 [... More application output ...]
 Time step  200
 [... More application output ...]
 BT-MZ Benchmark Completed.
 Time in seconds = 10.77
```

- Launch as a hybrid MPI+OpenMP application

Save the benchmark run time to be able to refer to it later. (Beware of potential over-subscription)

Performance analysis steps

- Reference preparation for validation

- Program instrumentation
- Summary measurement collection

- Summary experiment scoring
- Trace measurement collection with filtering

NPB-MZ-MPI / BT instrumentation

```
% make clean
% ml Score-P
```

- Start in the *Tutorial* directory again and clean-up the build
- Load the Score-P module (should be the matching classic Intel one by default)

NPB-MZ-MPI / BT instrumentation

```
#              SITE- AND/OR PLATFORM-SPECIFIC DEFINITIONS
#---------------------------------------------------------------
# Items in this file may need to be changed for each platform.
#---------------------------------------------------------------
OPENMP = -fopenmp
...
#---------------------------------------------------------------
# The Fortran compiler used for MPI programs
#---------------------------------------------------------------
#MPIF77 = mpif77 -f77=ifort

# Alternative variants to perform instrumentation
...
MPIF77 = $(PREP) mpif77 -f77=ifort

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

- Edit *config/make.def* to adjust build configuration
  - Modify specification of compiler/linker: MPIF77
  - Prefix compiler with `scorep` command (or use compiler wrappers, see reference material)

NPB-MZ-MPI / BT instrumented build

```
% make PREP=scorep bt-mz CLASS=C NPROCS=4
cd BT-MZ; make CLASS=C NPROCS=4 VERSION=
make: Entering directory 'BT-MZ'
cd ../sys; cc  -o setparams setparams.c -lm
../sys/setparams bt-mz 4 W
mpif77 -c  -O3 -fopenmp bt.f
 [...]
cd ../common;  scorep mpif77 -c  -O3 -fopenmp timers.f
scorep mpif77 –O3 -fopenmp -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_C.4
make: Leaving directory 'BT-MZ‘
```

- Re-build executable prefixing the compiler with the `scorep` command

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 environment variables

NPB-MZ-MPI / BT summary measurement collection

```
% <editor> bt-mz.sbatch
...
#SBATCH -J profile
...
BUILD=.scorep
...
export SCOREP_ENABLE_PROFILING=true
# change NOTES as desired to reflect measurement settings
NOTES=profile
export SCOREP_EXPERIMENT_DIRECTORY=\
 $OUTDIR/scorep-$SLURM_NPROCS-$OMP_NUM_THREADS-$NOTES
...
<save and exit>
% sbatch bt-mz.sbatch
```

- Point the script to the instrumented executable
- Configure measurement variables
- Run instrumented application

NPB-MZ-MPI / BT summary analysis report examination

```
% ls profile
bt-mz.out bt-mz.err scorep-4-12-profile/
% ls profile/scorep-4-12-profile/
MANIFEST.md  profile.cubex  scorep.cfg

% less profile/bt-mz.out
...
Time in seconds =                    44.60
...



% # optional
% cube profile/scorep-4-12-profile/profile.cubex
% paraprof profile/scorep-4-12-profile/profile.cubex

  [CUBE or TAU ParaProf GUI showing summary analysis report]
```

- Creates experiment directory including
  - Experiment directory overview (MANIFEST.md)
  - A record of the measurement configuration (scorep.cfg)
  - The analysis report that was collated after measurement (profile.cubex)

Congratulations!?

- If you made it this far, you successfully used Score-P to
  - instrument the application
  - record its execution with a summary measurement, and
  - [optional] 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

Performance analysis steps

- Reference preparation for validation

- Program instrumentation
- Summary measurement collection

- Summary experiment scoring
- Trace measurement collection with filtering

Goals of scoring and filtering

- Evaluate how *expensive* measurement of various regions is
  - Time cost is roughly fixed per event
  - Short functions are relatively more expensive
  - Space cost for tracing is linear in number of events
- Determine which expensive regions are *unnecessary* to measure
  - Frequently called, short execution, and non-scaling behavior
- Repeat the measurement, with those regions *filtered* at runtime to reduce overhead
  - Reduce space cost to zero and time cost to a hash comparison and a couple of branches
- (Optional) Apply the filter at *compile-time* in order to further reduce overhead
  - Eliminates the hash and branches; often not needed

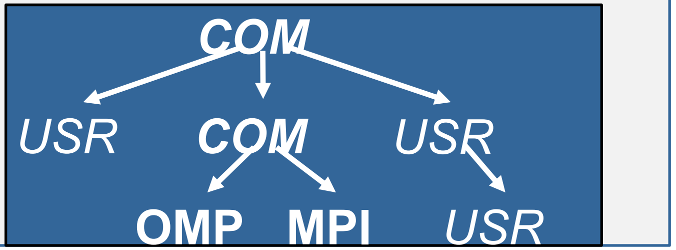
NPB-MZ-MPI / BT summary analysis result scoring

```
% scorep-score profile/scorep-4-12-profile/profile.cubex

Estimated aggregate size of event trace:                   161GB
Estimated requirements for largest trace buffer (max_buf): 41GB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       41GB
(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 43,327,477,817 6,608,030,277 1857.58   100.0           0.28  ALL
         USR 42,988,632,934 6,574,788,217  578.68    31.2           0.09  USR
         OMP    334,022,912    32,512,000 1248.29    67.2          38.39  OMP
         COM      4,697,810       722,740    2.38     0.1           3.30  COM
         MPI        124,120         7,316   28.23     1.5        3858.73  MPI
      SCOREP             41             4    0.00     0.0         295.98  SCOREP
```

160 GB total memory
41 GB per rank!



Region/callpath classification

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

NPB-MZ-MPI / BT summary analysis report breakdown

```
% scorep-score -r profile/scorep-4-12-profile/profile.cubex
  [...]
  [...]
flt     type     max_buf[B]         visits time[s] time[%] time/visit[us]  region
         ALL 43,327,477,817 6,608,030,277 1857.58   100.0           0.28  ALL
         USR 42,988,632,934 6,574,788,217  578.68    31.2           0.09  USR
         OMP    334,022,912    32,512,000 1248.29    67.2          38.39  OMP
         COM      4,697,810       722,740    2.38     0.1           3.30  COM
         MPI        124,120         7,316   28.23     1.5        3858.73  MPI
      SCOREP             41             4    0.00     0.0         295.98  SCOREP

         USR 13,812,365,034 2,110,313,472  236.59    12.7           0.11  binvcrhs_
         USR 13,812,365,034 2,110,313,472  157.46     8.5           0.07  matvec_sub_
         USR 13,812,365,034 2,110,313,472  163.54     8.8           0.08  matmul_sub_
         USR    596,197,758    87,475,200    9.95     0.5           0.11  lhsinit_
         USR    596,197,758    87,475,200    6.79     0.4           0.08  binvrhs_
         USR    447,869,968    68,892,672    4.35     0.2           0.06  exact_solution_
         OMP     26,860,032     1,234,944    0.21     0.0           0.17  !$omp parallel @exch_qbc.f:204
 [...]
```

More than
39 GB just for these 6
regions

NPB-MZ-MPI / BT summary analysis score

- Summary measurement analysis score reveals total size of event trace ~161 GB
- Maximum trace buffer size would be ~41 GB per rank
- 99.8% of the trace requirements are for USR regions
- These USR regions contribute around 20-25% of total time
- Conclusion: we need *filtering* to reduce overhead and remove uninteresting events!

NPB-MZ-MPI / BT summary analysis report filtering

```
% scorep-score -g profile/scorep-4-12-profile/profile.cubex
An initial filter file template has been generated:
'initial_scorep.filter'
To use this file for filtering at run-time, set the respective
Score-P variable:
    SCOREP_FILTERING_FILE=initial_scorep.filter
For compile-time filtering 'scorep' has to be provided with
the '--instrument-filter' option:
    $ scorep --instrument-filter=initial_scorep.filter
Compile-time filtering depends on support in the used Score-P
installation.
The filter file is annotated with comments, please check if
the selection is suitable for your purposes and add or remove
functions if needed.
```

- Report scoring with prospective filter listing 6 USR regions

NPB-MZ-MPI / BT summary analysis report filtering

```
% cat initial_scorep.filter
...
SCOREP_REGION_NAMES_BEGIN
  EXCLUDE
    # type=USR max_buf=13,812,365,034 ...
    # name='binvcrhs_'
    # file='BT-MZ/solve_subs.f'
    MANGLED binvcrhs_
    ...
SCOREP_REGION_NAMES_END

% scorep-score -f initial_scorep.filter \
>profile/scorep-4-12-profile/profile.cubex

Estimated aggregate size of event trace:                   1293MB
Estimated requirements for largest trace buffer (max_buf): 324MB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       348MB
(hint: When tracing set SCOREP_TOTAL_MEMORY=348MB to avoid intermediate flushes
 or reduce requirements using USR regions filters.)
```

- Report scoring with prospective filter listing 6 USR regions

1.2 GB of memory in total, 348 MB per rank!

NPB-MZ-MPI / BT summary analysis report filtering

```
% scorep-score -r -f initial_scorep.filter \
> profile/scorep-4-12-profile/profile.cubex
flt     type     max_buf[B]          visits time[s] time[%] time/visit[us]  region
 -       ALL 43,327,477,817 6,608,030,277 1857.58   100.0           0.28  ALL
 -       USR 42,988,632,934 6,574,788,217  578.68    31.2           0.09  USR
 -       OMP    334,022,912    32,512,000 1248.29    67.2          38.39  OMP
 -       COM      4,697,810       722,740    2.38     0.1           3.30  COM
 -       MPI        124,120         7,316   28.23     1.5        3858.73  MPI
 -    SCOREP             41             4    0.00     0.0         295.98  SCOREP

 *       ALL    338,875,641    33,246,789 1278.90    68.8          38.47  ALL-FLT
 +       FLT 42,988,602,202 6,574,783,488  578.67    31.2           0.09  FLT
 -       OMP    334,022,912    32,512,000 1248.29    67.2          38.39  OMP-FLT
 *       COM      4,697,810       722,740    2.38     0.1           3.30  COM-FLT
 -       MPI        124,120         7,316   28.23     1.5        3858.73  MPI-FLT
 *       USR         30,758         4,729    0.00     0.0           0.22  USR-FLT
 -    SCOREP             41             4    0.00     0.0         295.98  SCOREP-FLT
[...]
```

Filtered routines marked with '+'

- Score report breakdown by region

NPB-MZ-MPI / BT filtered measurement collection

```
% <editor> bt-mz.sbatch
...
#SBATCH -J filter
...
export SCOREP_FILTERING_FILE=initial_scorep.filter
# If you want to try collecting a trace:
export SCOREP_TOTAL_MEMORY=<value from scorep-score>
export SCOREP_ENABLE_TRACING=true
# Otherwise leave the above commented out and you
# get a filtered profile
% sbatch bt-mz.sbatch
```

- Apply filter configuration and re-run measurement
- This gives you a profile with less noise
- Also, collecting a trace is now practical and easy. Remember this for later

NPB-MZ-MPI / BT filtered trace measurement collection

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

 Number of zones:   16 x   16
 Iterations: 200    dt:   0.000100
 Number of active processes:     4

 Use the default load factors with threads
 Total number of threads:     48  (  12.0 threads/process)

 Calculated speedup =     47.99

 Time step    1
 [... More application output ...]
 BT-MZ Benchmark Completed.
 Time in seconds = 13.38
```

- Output from filtered run

NPB-MZ-MPI / BT filtered results examination

```
% ls filter/scorep-4-12-profile/
MANIFEST.md  profile.cubex  scorep.cfg   scorep.filter

or

% ls filter/scorep-4-12-trace/
MANIFEST.md  profile.cubex  scorep.cfg scorep.filter  traces
traces.def  traces.otf2
```

- More about trace analysis and visualization Thursday and Friday!

Function Groups

- Frequently asked questions:
  - How do I structure my code to make it tools-comprehensible?
  - What does Score-P do automatically to make my measurement easier to read?
- Extend the USR/COM/MPI/… concept from scoring: *function groups*
- Predefined groupings:
  - Per paradigm
  - Within paradigms (MPI categories)
  - Namespace/class hierarchy

Score-P: Further information

- Scalable Performance Measurement Infrastructure for Parallel Codes
  - Instrumenter, libraries, and tools to generate profile and trace measurements
  - Bundled with OTF2 (tracing), OPARI2 (OpenMP instrumentation), CubeWriter, and CubeLib (profiling)
- Available under 3-clause BSD open-source license
- Documentation & sources:
  - https://www.score-p.org
- User guide also part of installation:
  - <prefix>/share/doc/scorep/pdf/scorep.pdf
- Contact:
  - mailto: support@score-p.org