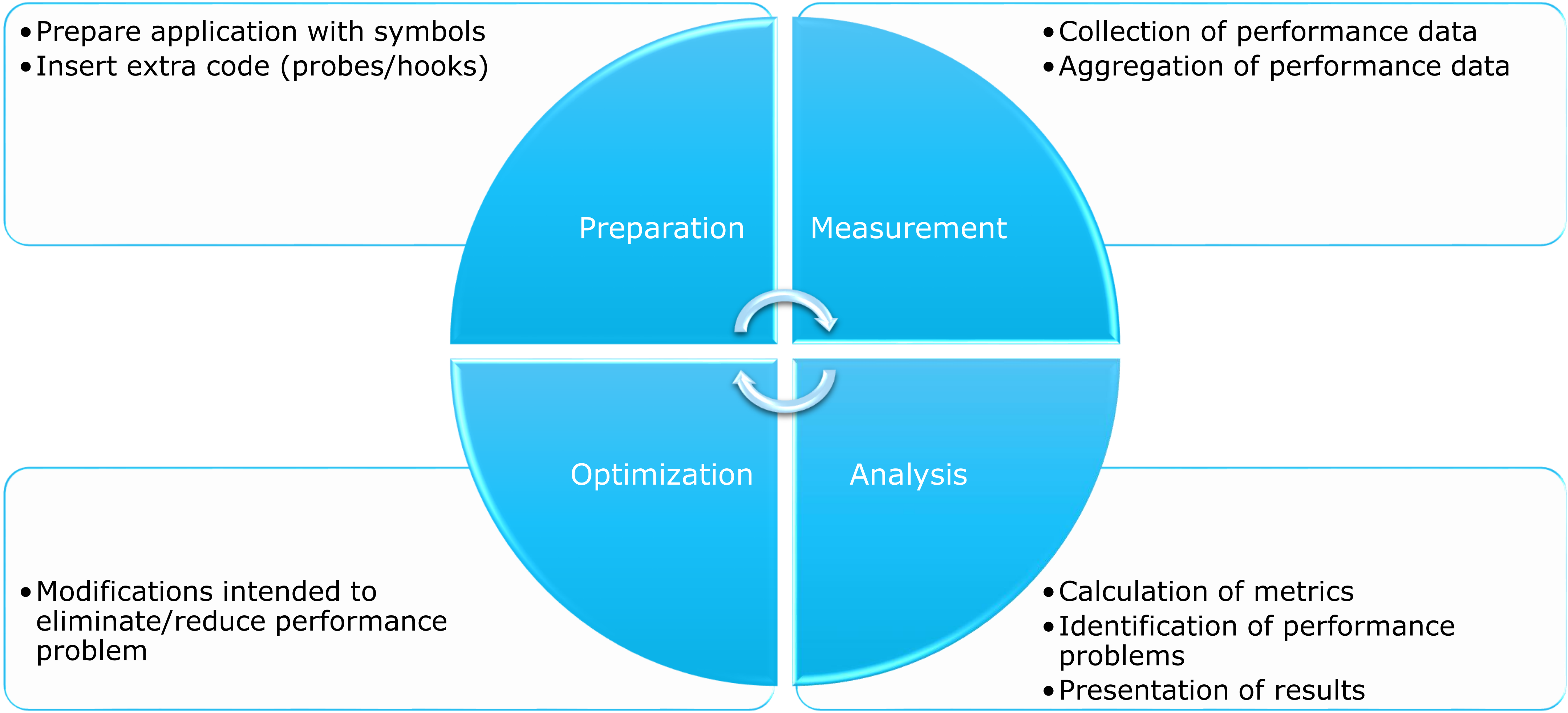
Score-P – A Joint Performance Measurement Run-Time Infrastructure for Scalasca, TAU, and Vampir

VI-HPS Team

Score-P

Scalable performance measurement infrastructure for parallel codes

Performance engineering workflow



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

- Collection of performance data
- Aggregation of performance data

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

- Modifications intended to eliminate/reduce performance problem

Score-P

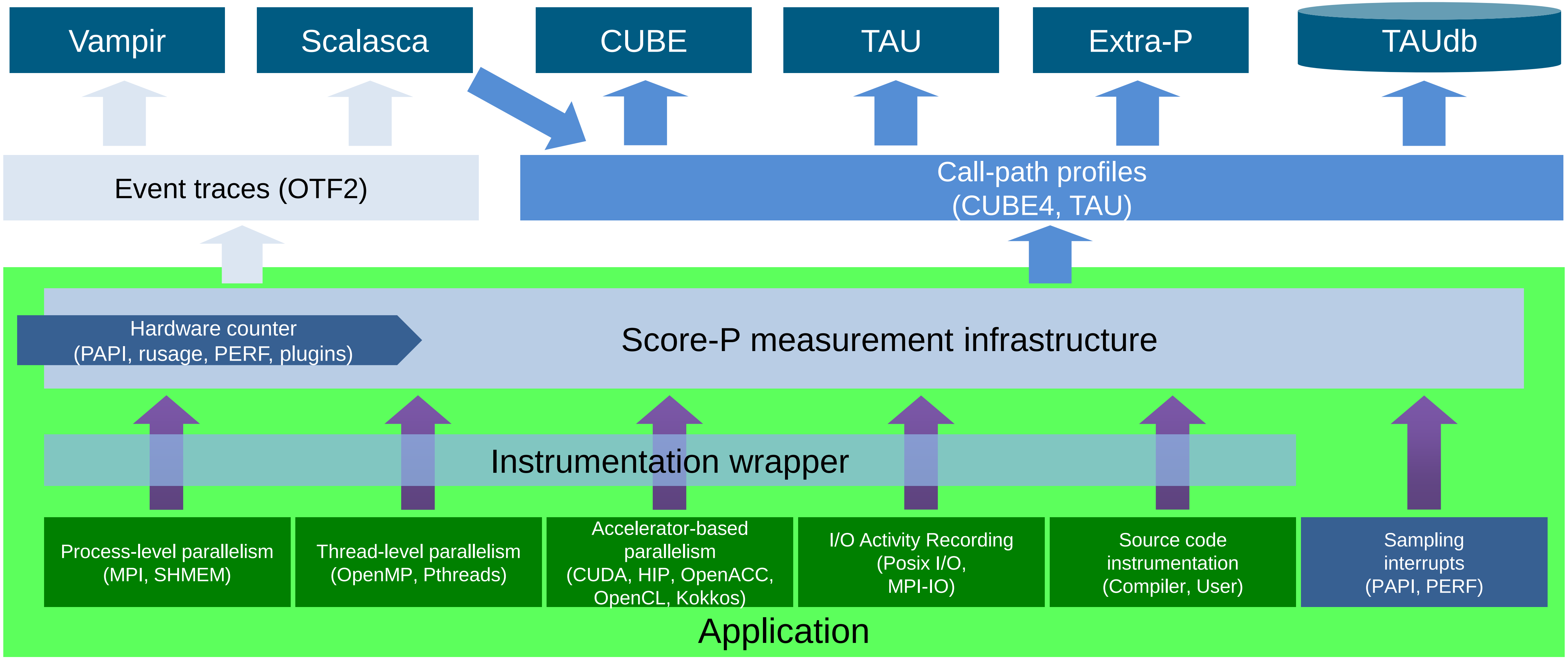
- Infrastructure for instrumentation and performance measurements
- Instrumented application can be used to produce several results:
  - Call-path profiling: CUBE4 data format used for data exchange
  - Event-based tracing: OTF2 data format used for data exchange

- Supported parallel paradigms:
  - Multi-process: MPI, SHMEM
  - Thread-parallel: OpenMP, Pthreads
  - Accelerator-based: CUDA, HIP, OpenCL, OpenACC, Kokkos

- Open Source; portable and scalable to all major HPC systems
- Initial project funded by BMBF
- Further developed in multiple 3rd-party funded projects

GEFÖRDERT VOM
Bundesministerium
für Bildung
und Forschung

Score-P overview


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, HIP, OpenACC, OpenCL, Kokkos)
I/O Activity Recording (Posix I/O, MPI-IO)
Source code instrumentation (Compiler, User)
Sampling interrupts (PAPI, PERF)
Application

Score-P user commands

`scorep` – instrument your software by prepending to your compile command:

```
scorep <scorep-options> <compiler> <compiler-options>
```

`scorep-cc/CC/ftn` – alt. instrumentation, replaces compiler command (CMake, autotools):

```
SCOREP_WRAPPER_INSTRUMENTER_FLAGS=<scorep-options> scorep-CC <compiler-options>
```

See `scorep-wrapper --help` for details.

`scorep-info` – List (measurement) configuration options, open issues, license, and more

`scorep-score` – *Score* a profile measurement, create a filter (prepare trace measurement):

```
scorep-score [-g] [-r] [-f <filter>] profile.cubex
```

All commands support `--help`

Contributing Partners

- Forschungszentrum Jülich, Germany
- RWTH Aachen, Germany
- Technische Universität Darmstadt, Germany
- Technische Universität Dresden, Germany
- Technische Universität München, Germany
- University of Oregon, Eugene, USA
- German Aerospace Center (DLR)

Sponsors

GEFÖRDERT VOM
Bundesministerium
für Bildung
und Forschung

DEPARTMENT OF ENERGY
UNITED STATES OF AMERICA

SEVENTH FRAMEWORK
PROGRAMME

European Commission
Horizon 2020
European Union funding
for Research & Innovation

EuroHPC
Joint Undertaking

Funded by
DFG Deutsche
Forschungsgemeinschaft
German Research Foundation

HELMHOLTZ
ASSOCIATION

ITEA2

OAK
RIDGE
National Laboratory

intel.

AMD

SIEMENS

NVIDIA

Hands-on:
NPB-MZ-MPI / bt-mz_C.x

Score-P

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

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 (Archer2)

- Select modules for the PrgEnv-gnu tool chain

```
% module swap PrgEnv-cray PrgEnv-gnu
```

Default is PrgEnv-cray
Alternatives PrgEnv-aocc & PrgEnv-gnu

- Copy tutorial sources to your “WORK” directory

```
% cd /work/ta115/ta115/$USER
% tar zxvf /work/y23/shared/tutorial/NPB3.4-MZ-MPI.tar.gz
% cd NPB3.4-MZ-MPI
```

Use “WORK” filesystem
for building and submitting

- Directory for data exchange during the workshop

```
/work/ta115/ta115/shared/
```

NPB-MZ-MPI Suite

- The NAS Parallel Benchmark suite (MPI+OpenMP version)
  - Available from:

    http://www.nas.nasa.gov/Software/NPB

  - 3 benchmarks in Fortran90 (older versions Fortran77)
  - Configurable for various sizes & classes
- Move into the NPB3.4-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
  - but config/make.def may first need to be adjusted to specify appropriate compiler flags

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

```
#                SITE- AND/OR PLATFORM-SPECIFIC DEFINITIONS.
#
#---------------------------------------------------------------------------

#---------------------------------------------------------------------------
# Configured for HPE/Cray systems with PrgEnv compiler-specific OpenMP
#---------------------------------------------------------------------------
#COMPFLAGS = -fopenmp # aocc/flang
#COMPFLAGS = -homp -G2 # cce
COMPFLAGS = -fopenmp -fallow-argument-mismatch # gnu
...

#----------------------------------------------------------------------
# The Fortran compiler used for MPI programs
#----------------------------------------------------------------------
FC = ftn

# Alternative variants to perform instrumentation
...
#FC = $(PREP) ftn
#FC = scorep --user  ftn
#FC = scorep-ftn
...
```

Uncomment COMPILER flags according to current environment

Default (no instrumentation)

Hint: uncomment a compiler wrapper to do instrumentation

Building an NPB-MZ-MPI Benchmark

```
% make
  ===========================================
  =      NAS PARALLEL BENCHMARKS 3.4        =
  =      MPI+OpenMP Multi-Zone Versions     =
  =      MPI/Fortran                        =
  ===========================================

  To make a NAS multi-zone benchmark type

         make <benchmark-name> CLASS=<class>

  where <benchmark-name> is “bt-mz”, “lu-mz”, or “sp-mz”
        <class>          is “S”, “W”, “A” through “F”

  [...]

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

- Type “make” for instructions

Building an NPB-MZ-MPI Benchmark

```
% make bt-mz CLASS=C
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 C
make[2]: Entering directory `../BT-MZ'
ftn -g -c  -O3 -fopenmp   bt.f90
[…]
ftn -g -c  -O3 -fopenmp   setup_mpi.f90
cd ../common; ftn -g -c  -O3 -fopenmp       print_results.f90
cd ../common; ftn -g -c  -O3 -fopenmp       timers.f90
ftn -g -O3 -fopenmp        -o ../bin/bt-mz_C.x  bt.o bt_data.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 setup_mpi.o mpinpb.o error_cond.o
 ../common/print_results.o ../common/timers.o
make[2]: Leaving directory `BT-MZ'
Built executable ../bin/bt-mz_C.x
make[1]: Leaving directory `BT-MZ'
```

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

Shortcut: `% make suite`

NPB-MZ-MPI / BT (Block Tri-diagonal 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
  - Includes verification of solution
- Implemented in 20 or so Fortran90 source modules

- Uses MPI & OpenMP in combination
  - 8 processes each with 6 threads should be reasonable for 1 compute node of ARCHER2
  - bt-mz_C.x should run in less than 15 seconds
  - Benchmark time reported as “Time in seconds”

NPB-MZ-MPI / BT Reference Execution

```
% cd bin
% cp ../jobscript/archer2/run.sbatch .
% less run.sbatch
% sbatch run.sbatch

% cat slurm-<job_id>.out
 NAS Parallel Benchmarks (NPB3.4-MZ MPI+OpenMP) - BT-MZ Benchmark
 Number of zones:  16 x  16
 Iterations:  200   dt:   0.000300
 Number of active processes:    8
 Use the default load factors with threads
 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 = 11.67
```

- 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

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

Local installation (ARCHER2)

- Setup access to Scalasca and associated tools, accessible via “other-software”
  - Required for each shell session
  - Score-P and Scalasca installations are toolchain specific (GCC11 default)

```
% module swap PrgEnv-cray PrgEnv-gnu
% module load load-epcc-module other-software
% module load scalasca/2.6.1-gcc11
```

PrgEnv-cray
PrgEnv-aocc
PrgEnv-gnu

  - Check `module avail scalasca` for alternate Score-P/Scalasca modules available

- Copy tutorial sources to your personal workspace (if not already done)

```
% cd /work/ta115/ta115/$USER
% tar zxvf /work/y23/shared/tutorial/NPB3.4-MZ-MPI.tar.gz
% cd NPB3.4-MZ-MPI
```

NPB-MZ-MPI / BT instrumentation

```
#---------------------------------------------------------------------
# The Fortran compiler used for MPI programs
#---------------------------------------------------------------------
#FC = ftn

# Alternative variants to perform instrumentation
...
# PREP is a generic macro for instrumentation preparation
# e.g. PREP="scorep --user"
FC = $(PREP) ftn

# Alternative via Score-P compiler wrapper
# configured via SCOREP_WRAPPER_INSTRUMENTER_FLAGS="--user"
# FC = scorep-ftn

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

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

Uncomment the Score-P compiler wrapper specification
Alternatively, use compiler wrapper `scorep-ftn`

NPB-MZ-MPI / BT instrumented build

```
% make clean

% make bt-mz CLASS=C
cd BT-MZ; make CLASS=C VERSION=
make: Entering directory 'BT-MZ'
cd ../sys; cc  -o setparams setparams.c -lm
../sys/setparams bt-mz C
scorep --user ftn -g -c  -O3 -fopenmp bt.f90
 [...]
cd ../common;  scorep --user ftn -g -c  -O3 -fopenmp timers.f90
 [...]
scorep --user ftn -g -O3 -fopenmp -o ../bin.scorep/bt-mz_C.x \
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.x
make: Leaving directory 'BT-MZ`
```

- Return to root directory and clean-up
- Re-build executable using Score-P compiler wrapper

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

```
% cd bin.scorep
% cp ../jobscript/archer2/scorep.sbatch .
% cat scorep.sbatch
...
# Score-P measurement configuration
#export SCOREP_EXPERIMENT_DIRECTORY=scorep_bt-mz_sum
#export SCOREP_FILTERING_FILE=../config/scorep.filt
#export SCOREP_METRIC_PAPI=PAPI_TOT_INS,PAPI_TOT_CYC,…
#export SCOREP_TOTAL_MEMORY=100M
#export SCOREP_ENABLE_TRACING=true

# Run the application
srun  ./bt-mz_C.x

% sbatch scorep.sbatch
```

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

Leave these lines commented out for the moment

- Submit job

Summary measurement collection

```
% less slurm-<job_id>.out

 NAS Parallel Benchmarks (NPB3.4-MZ MPI+OMP) - BT-MZ Benchmark

 Number of zones:  16 x  16
 Iterations: 200    dt:   0.000100
 Number of active processes:     8
 Use the default load factors with threads
 Total number of threads:    48  (  6.0 threads/process)

 Calculated speedup = 47.97

 Time step    1

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

- Check the output of the application run

BT-MZ summary analysis report examination

```
% ls
slurm-<job_id>.out scorep_bt-mz_sum/
% ls -1 scorep_bt-mz_sum
MANIFEST.md
profile.cubex
scorep.cfg


% cube scorep_bt-mz_sum/profile.cubex
# alternatively
% square scorep_bt-mz_sum/
 [CUBE GUI showing summary analysis report]

% paraprof scorep_bt-mz_sum/profile.cubex
 [TAU ParaProf GUI showing summary analysis report]
```

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

Hint:
Copy ‘profile.cubex’ to local system (laptop) using ‘scp’ to improve responsiveness of GUI

Reference results available:
`/work/y23/shared/tutorial/examples`

BT-MZ summary analysis report remapping

```
% ls -1 scorep_bt-mz_sum/
MANIFEST.md
profile.cubex
scorep.cfg

# remap the Score-P way
% cube_remap2 -d -o scorep_bt-mz_sum/summary.cubex \
  scorep_bt-mz_sum/profile.cubex


# remap the Scalasca way
% square scorep_bt-mz_sum
INFO: Post-processing runtime summarization report (profile.cubex)...
INFO: Displaying ./scorep_bt-mz_sum/summary.cubex...

  [CUBE GUI showing summary analysis report]
```

- `profile.cubex` contains raw measurement data
- Enhance by *remapping*, i.e., transform given metric tree into metric hierarchy

Further information

- Community instrumentation & measurement infrastructure
  - Instrumentation (various methods)
  - Basic and advanced profile generation
  - Event trace recording
- Available under 3-clause BSD open-source license
- Download sources, subscribe to news mailing list:
  - http://www.score-p.org
- User guide part of installation or available online:
  - <prefix>/share/doc/scorep/{pdf,html}/
  - Online HTML / Online PDF
- Support and feedback: support@score-p.org