Score-P Cheat Sheet

General Workflow Loop

- **Preparation**: instrument target application and set up measurement environment
- **Measurement**: run application with measurement infrastructure enabled
- **Analysis**: analyse generated performance data
- **Examination**: find possible cause of performance anomalies in the code
- **Optimization**: apply optimizations to eliminate bottleneck
- **Repeat**: apply analysis workflow loop until acceptable performance achieved

Performance Analysis Procedure

- Create a profile with full instrumentation
- Compare runtime to uninstrumented run to determine overhead
- (Incrementally) create filter file using hints from the `scorep-score` tool
- Create an optimized profile with filter applied
- Investigate profile with CUBE
- For in-depth analysis, generate a trace **with filter applied** and examine it using Scalasca and than Vampir

LLNL Environment Setup

Dotkits for Score-P and Scalasca are available on OCF Linux clusters for both Intel and GCC compilers. Both require and will automatically load mvapich2-2.2.

- `use scorep-mvapich2-gnu-2.2` (for GNU Compilers)
- `use scorep-mvapich2-intel-2.2` (for Intel Compilers)

To load the Scalasca automatic trace analyzer, use:

- `use scalasca-2.3.1-mvapich2-gnu-2.2` (for GNU Compilers)
- `use scalasca-2.3.1-mvapich2-intel-2.2` (for Intel Compilers)

Application Instrumentation

- Prefix all compile/link commands with `scorep`, e.g.

  ```
  scorep mpicc $CFLAGS -c foo.c
  scorep mpicc -o foo foo.o $LDFLAGS
  ```

- Compile as usual
- Advanced instrumentation options available to further adjust the measurement configuration

Application Measurement

Set Score-P environment variables

`SCOREP_EXPERIMENT_DIRECTORY=<path>`	Name of the experiment directory
`SCOREP_ENABLE_PROFILING=true/false`	Enable generation of profiles (default=true)
`SCOREP_ENABLE_TRACING=true/false`	Enable the generation of traces (default=false)
`SCOREP_TOTAL_MEMORY=<size>`	Total memory in bytes used for Score-P per process (default=16M)
`SCOREP_FILTERING_FILE=<filter-file>`	Name of file containing filter rules

... and many more (see manual or run `scorep-info config-vars -full`)
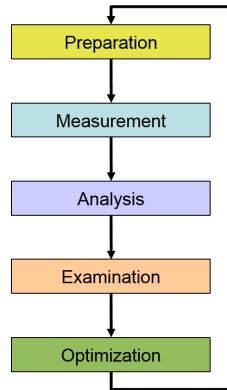
Summary Collection

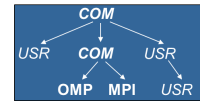Setup measurement configuration and run application as usual:

```
export SCOREP_EXPERIMENT_DIRECTORY=scorep_bt-mz_W_4x4_sum
export OMP_NUM_THREADS=4
srun -n 4 ./bt-mz_W.4
```

Profile Examination with CUBE and Filter File Creation

Launch CUBE report explorer (GUI):

```
cube scorep_bt-mz_W_4x4_sum/profile.cubex
```

Cube region/callpath classification:

MPI	pure MPI library functions
OMP	pure OpenMP functions/regions
USR	user-level source local computation
COM	USR on callpath to OpenMP/MPI
ANY/ALL	aggregate of all region types

Create filter file with hints from `scorep-score`
```
% scorep-score -r scorep_bt-mz_W_4x4_sum/profile.cubex
% scorep-score -r -f ../config/scorep.filt scorep_bt-mz_W_4x4_sum/profile.cubex
```
Create profile with filter applied
```
% export SCOREP_EXPERIMENT_DIRECTORY=scorep_bt-mz_W_4x4_sum_filter
% export SCOREP_FILTERING_FILE=../config/scorep.filt
% srun -n 4 ./bt-mz_W.4
```

Trace Analysis with Scalasca
```
% use scalasca-2.3.1-mvapich2-gnu-2.2
% export SCOREP_EXPERIMENT_DIRECTORY=scorep_bt-mz_W_4x4_trace
% export SCOREP_TOTAL_MEMORY=16M
% scan srun -n 4 ./bt-mz_W.4
```

Trace Analysis Report Exploration
```
% square scorep_bt-mz_W_4x4_trace
```

Interactive Performance Analysis with Vampir

Open trace directly in Vampir
```
% use vampir
% vampir scorep_bt-mz_W_4x4_trace/traces.otf2
```

PAPI Hardware Performance Counters

```
% export SCOREP_METRIC_PAPI=PAPI_L2_DCM:!CPU_TEMP1
```
CPU_TEMP1 is provided by the lm-sensors component.
See `papi_avail` and `papi_native_avail` for available counter.

```
PAPI_L[1|2|3]_[D|I|T]C[M|H|A|R|W]
                Level 1/2/3 data/instruction/total cache
                misses/hits/accesses/reads/writes
PAPI_L[1|2|3]_[LD|ST]M
                Level 1/2/3 load/store misses
PAPI_CA_SNP     Requests for a snoop
PAPI_CA_SHR     Req. for excl. access to shared cache line
PAPI_CA_CLN     Req. for excl. access to clean cache line
PAPI_CA_INV     Requests for cache line invalidation
PAPI_CA_ITV     Requests for cache line intervention
PAPI_BRU_IDL    Cycles branch units are idle
PAPI_FXU_IDL    Cycles integer units are idle
PAPI_FPU_IDL    Cycles floating point units are idle
PAPI_LSU_IDL    Cycles load/store units are idle
PAPI_TLB_DM     Data translation lookaside buffer misses
PAPI_TLB_IM     Instruction transl. lookaside buffer misses
PAPI_TLB_TL     Total translation lookaside buffer misses
PAPI_BTAC_M     Branch target address cache misses
PAPI_PRF_DM     Data prefetch cache misses
PAPI_TLB_SD     Translation lookaside buffer shootdowns
PAPI_CSR_FAL    Failed store conditional instructions
PAPI_CSR_SUC    Successful store conditional instructions
PAPI_CSR_TOT    Total store conditional instructions
PAPI_MEM_SCY    Cycles Stalled Waiting for memory accesses
PAPI_MEM_RCY    Cycles Stalled Waiting for memory Reads
PAPI_MEM_WCY    Cycles Stalled Waiting for memory writes
PAPI_STL_ICY    Cycles with no instruction issue
PAPI_FUL_ICY    Cycles with maximum instruction issue
PAPI_STL_CCY    Cycles with no instructions completed
PAPI_FUL_CCY    Cycles with maximum instructions completed
PAPI_BR_UCN     Unconditional branch instructions
PAPI_BR_CN      Conditional branch instructions
PAPI_BR_TKN     Conditional branch instructions taken
PAPI_BR_NTK     Conditional branch instructions not taken
PAPI_BR_MSP     Conditional branch inst. mispredicted
PAPI_BR_PRC     Cond. branch inst. correctly predicted
PAPI_FMA_INS    FMA instructions completed
PAPI_TOT_IIS    Instructions issued
PAPI_TOT_INS    Instructions completed
PAPI_INT_INS    Integer instructions
PAPI_FP_INS     Floating point instructions
PAPI_LD_INS     Load instructions
PAPI_SR_INS     Store instructions
PAPI_BR_INS     Branch instructions
PAPI_VEC_INS    Vector/SIMD instructions
PAPI_LST_INS    Load/store instructions completed
PAPI_SYC_INS    Synchronization instructions completed
PAPI_FML_INS    Floating point multiply instructions
PAPI_FAD_INS    Floating point add instructions
PAPI_FDV_INS    Floating point divide instructions
PAPI_FSQ_INS    Floating point square root instructions
PAPI_FNV_INS    Floating point inverse instructions
PAPI_RES_STL    Cycles stalled on any resource
PAPI_FP_STAL    Cycles the FP unit(s) are stalled
PAPI_FP_OPS     Floating point operations
PAPI_TOT_CYC    Total cycles
PAPI_HW_INT     Hardware interrupts
```

Resource Usage Counters

The Unix system call `getrusage` provides information about consumed resources and operating system events.
```
% export SCOREP_METRIC_RUSAGE=ru_stime:ru_majflt
```

Name	Unit	Linux	Description
ru_utime	ms	x	Total amount of user time used.
ru_stime	ms	x	Total amount of system time used.
ru_maxrss	kB		Maximum resident set size.
ru_ixrss	kB/s		Integral shared memory size (text segment).
ru_idrss	kB/s		Integral data segment memory used over runtime.
ru_isrss	kB/s		Integral stack memory used over the runtime.
ru_minflt	#	x	Number of soft page faults.
ru_majflt	#	x	Number of hard page faults.
ru_nswap	#		# times process was swapped out of phys. mem.
ru_inblock	#		Number of input operations via the file system.
ru_oublock	#		Number of output operations via the file system.
ru_msgsnd	#		Number of IPC messages sent.
ru_msgrcv	#		Number of IPC messages received.
ru_nsignals	#		Number of signals delivered.
ru_nvcsw	#	x	Number of voluntary context switches.
ru_nivcsw	#	x	Number of involuntary context switches.