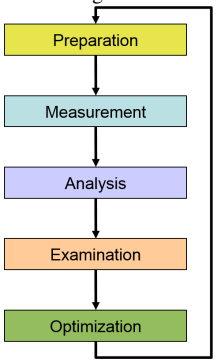
Score-P Cheat Sheet

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



Application Instrumentation

- Prefix all compile/link commands with `scorep`
- Compile as usual
- Advanced instrumentation options available to further adjust the measurement configuration

Application Measurement

Set Score-P environment variables

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

... and many more (see manual or run `scorep-info config-vars --full`)
Run application as usual:

```
% export SCOREP_ENABLE_TRACING=false
% export SCOREP_ENABLE_PROFILING=true
% export SCOREP_EXPERIMENT_DIRECTORY=scorep_run
% export OMP_NUM_THREADS=4
% mpirun -np 4 ./binary_scorep
```

Profile Examination with CUBE and Filter File Creation

Analyze profile with CUBE

```
% cube scorep_run/profile.cubex
```

Create filter file with hints from `scorep-score`

```
% scorep-score -r scorep_run/profile.cubex
% scorep-score -r -f ./scorep.filt scorep_run/profile.cubex
```

Create profile with filter applied

```
% export SCOREP_EXPERIMENT_DIRECTORY=scorep_run_filter
% export SCOREP_FILTERING_FILE=scorep.filt
% mpirun -np 4 ./binary_scorep
```

Automatic Trace Analysis with Scalasca

Run the application using Scalasca with trace collection and analysis

```
% export SCOREP_EXPERIMENT_DIRECTORY=scorep_run_trace
% export OMP_NUM_THREADS=4
% export SCOREP_TOTAL_MEMORY=25M
% scan -f ./scorep.filt -t mpirun -np 4 ./binary_scorep
```

Produces and examine trace analysis report

```
% square scorep_run_trace
```

Interactive Performance Analysis with Vampir

Open small traces direclty in Vampir

```
% vampir scorep_run_trace/traces.otf2
```

Open large traces using VampirServer

1. Launch analysis server on remote machine

```
% ssh remote-machine
% vampirserver start -n 4
Running 4 analysis processes...  (abort with vampirserver stop 17950)
VampirServer <17950> listens on:  node123:30085
```

2. Open SSH tunnel to connect remote VampirServer with GUI on your local machine

```
% ssh -L30000:node123:30085 mymachine
```

3. Open Vampir and connect to VampirServer (listening on localhost:30000 via SSH tunnel)

```
% vampir localhost:30000:scorep_run_trace/traces.otf2
```

4. Shutdown VampirServer on remote machine when finished

```
% ssh remote-machine
% vampirserver stop
```

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.