Score-P: Performance Measurement Infrastructure

18 February 2015

Frank Winkler
On-site contractor for Vampir, CSMD
Agenda

Performance Analysis Approaches
- Sampling
- Instrumentation
- Profiling
- Tracing

Score-P: Scalable Performance Measurement Infrastructure for Parallel Codes
- Motivation
- Functionality
- Architecture
- Workflow

Demo
- Performance Analysis of Jacobi Solver on Titan

Conclusions
Performance Analysis Approaches

- **Sampling**
  - Concise data sets
  - Good overview
  - Limited detail
  - No outliers

- **Instrumentation**
  - Extensive data sets
  - Most detailed

**Profile**
- Score-P
- Cube

**Event Trace**
- VAMPIR
Score-P: Motivation

• Several performance tools co-exist
• Separate measurement systems and output formats
• Complementary features and overlapping functionality
• Redundant effort for development and maintenance
• Limited or expensive interoperability
• Complications for user experience, support, training
Score-P: Functionality

• Typical functionality for HPC performance tools
  – Instrumentation (various methods)
  – Sampling (Prototype available on Titan)

• Flexible measurement without re-compilation
  – Basic and advanced profile generation
  – Event trace recording

• Programming paradigms:
  – Multi-process
    • MPI, SHMEM
  – Thread-parallel
    • OpenMP, Pthreads
  – Accelerator-based
    • CUDA, OpenCL, OpenACC (Prototype for PGI compiler)

→ Hybrid parallelism and serial programs
Score-P: Architecture

- Vampir
- Scalasca
- CUBE
- TAU
- Periscope

Event traces (OTF2)

Call-path profiles (CUBE4, TAU)

Score-P measurement infrastructure

Hardware counter (PAPI, rusage)

Application (MPI×OpenMP×CUDA)

Instrumentation wrapper

Online interface

PMPI  OPARI 2  CUDA  Compiler  PDT  User
1. Instrument your application with Score-P

**CC=cc**
**CXX=CC**
**F90=ftn**

**CC=scorep** <options> cc
**CXX=scorep** <options> CC
**F90=scorep** <options> ftn

To see all available options for instrumentation:

```
$ scorep --help
This is the Score-P instrumentation tool. The usage is:
scorep <options> <original command>

Common options are:
...
--compiler Enables compiler instrumentation.
By default, it disables pdt instrumentation.
--nocompiler Disables compiler instrumentation.
--user Enables user instrumentation.
--cuda Enables cuda instrumentation.
...
```
Score-P: Workflow

1. Instrument your application with Score-P

- For Cmake or Autoconf applications use generic wrapper

```bash
$ cat scorep_wrapper
# extract compiler name
compiler=${0##*scorep-}

if [ -n "${SCOREP_WRAPPER_OFF}" ]; then
  $compiler "$@
else
  scorep $SCOREP_WRAPPER_ARGS $compiler $SCOREP_WRAPPER_FLAGS "$@
fi

$ # symlinks to generic wrapper for all compilers
$ ln -s scorep_wrapper scorep-cc
$ ln -s scorep_wrapper scorep-CC
$ ln -s scorep_wrapper scorep-ftn
```

- Latest Score-P installation on Titan already includes scorep-cc, scorep-CC and scorep-ftn
Score-P: Workflow

1. Instrument your application with Score-P

- Example for instrumenting a CMake/Autoconf application:

```bash
$ # run cmake/configure without instrumentation if necessary
$ CC=scorep-cc CXX=scorep-CC F90=scorep-ftn SCOREP_WRAPPER_OFF=true cmake .

$ CC=scorep-cc CXX=scorep-CC F90=scorep-ftn SCOREP_WRAPPER_OFF=true ./configure

$ # run make with instrumentation
$ make SCOREP_WRAPPER_ARGS="--mpp=mpi --cuda --verbose" \ SCOREP_WRAPPER_FLAGS="-g -O0"

$ make install
```
Score-P: Workflow

2. Perform a measurement run with profiling enabled

- Measurements are configured via environment variables:

  ```bash
  $ scorep-info config-vars --full
  
  SCOREP_ENABLE_PROFILING
    [...]  
  SCOREP_ENABLE_TRACING
    [...]  
  SCOREP_TOTAL_MEMORY
    Description: Total memory in bytes for the measurement system
    [...]  
  SCOREP_EXPERIMENT_DIRECTORY
    Description: Name of the experiment directory
    [...]  
  
  $ export SCOREP_ENABLE_PROFILING=true #default
  $ export SCOREP_ENABLE_TRACING=false  #default
  $ export SCOREP_EXPERIMENT_DIRECTORY=profile
  
  $ aprun <instrumented binary>
  ```
Score-P: Workflow

3. Perform analysis on profile data

- Call tree analysis with CUBE:

```bash
$ cube profile/profile.cubex
```
3. Perform analysis on profile data

- Textual output:

```
$ scorep-score -r profile.cubex

Estimated aggregate size of event trace: 11GB
Estimated requirements for largest trace buffer (max_buf): 11GB
Estimated memory requirements (SCOREP_TOTAL_MEMORY): 11GB

(hint: When tracing set SCOREP_TOTAL_MEMORY=11GB to avoid intermediate flushes or reduce requirements using USR regions filters.)

<table>
<thead>
<tr>
<th>flt</th>
<th>type</th>
<th>max_buf[B]</th>
<th>visits</th>
<th>time[s]</th>
<th>time[%]</th>
<th>time/visit[us]</th>
<th>region</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>10,985,856,629</td>
<td>423,276,997</td>
<td>345.68</td>
<td>100.0</td>
<td>0.82</td>
<td>ALL</td>
<td></td>
</tr>
<tr>
<td>USR</td>
<td>10,985,844,746</td>
<td>423,276,691</td>
<td>342.88</td>
<td>99.2</td>
<td>0.81</td>
<td>USR</td>
<td></td>
</tr>
<tr>
<td>MPI</td>
<td>7,333</td>
<td>131</td>
<td>0.01</td>
<td>0.0</td>
<td>49.85</td>
<td>MPI</td>
<td></td>
</tr>
<tr>
<td>COM</td>
<td>4,550</td>
<td>175</td>
<td>2.79</td>
<td>0.8</td>
<td>15958.10</td>
<td>COM</td>
<td></td>
</tr>
</tbody>
</table>

USR  1,995,302,400  76,742,400     22.22       6.4     0.29   foo
USR  1,969,696,898  75,757,573     21.84       6.3     0.29   bar
USR  1,760,180,292  67,699,242     34.01       9.8     0.50   main
```
Score-P: Workflow

4. Define an appropriate filter

```bash
$ cat scorep.filter
SCOREP_REGION_NAMES_BEGIN
   EXCLUDE
     foo*
     bar*
SCOREP_REGION_NAMES_END
```

5. Perform a measurement run with tracing enabled and the filter applied

```bash
$ export SCOREP_ENABLE_PROFILING=false
$ export SCOREP_ENABLE_TRACING=true
$ export SCOREP_EXPERIMENT_DIRECTORY=trace
$ export SCOREP_FILTERING_FILE=scorep.filter
$ aprun <instrumented binary>
```
6. Perform analysis on the trace data with Vampir

```bash
$vampir trace/traces.otf2$
```
Demo

- Performance Analysis of Jacobi Solver on Titan

```bash
$ ssh -X titan
$ qsub -I -X -A [projid] -q debug -l nodes=2,walltime=60:00

$ module sw PrgEnv-{pgi,gnu}
$ module load scorep

$ cd $MEMBERWORK[projid]
$ cp $WORLDWORK/stf010/winklerf/jacobi.tar.gz ./
$ tar xzvf jacobi.tar.gz
$ cd jacobi

$ make scorep
$ cd bin

$ # generate profile (copy from run_scorep.pbs)
$ export OMP_NUM_THREADS=6
$ export SCOREP_ENABLE_TRACING=false
$ export SCOREP_ENABLE_PROFILING=true
$ export SCOREP_EXPERIMENT_DIRECTORY=profile
$ export SCOREP_CUDA_ENABLE=yes
$ export SCOREP_CUDA_BUFFER=32M

$ aprun -n 2 -d 6 -N 1 ./jacobi_mpi+openmp+cuda 4096 4096 0.15
```
Demo

• Performance Analysis of Jacobi Solver on Titan

```bash
$ # profile analysis with CUBE
$ cube profile/profile.cubex

$ # profile analysis with scorep-score
$ scorep-score -r profile/profile.cubex

$ # generate trace
$ export OMP_NUM_THREADS=6
$ export SCOREP_ENABLE_TRACING=true
$ export SCOREP_ENABLE_PROFILING=false
$ export SCOREP_EXPERIMENT_DIRECTORY=trace
$ export SCOREP_CUDA_ENABLE=yes
$ export SCOREP_CUDA_BUFFER=32M

$ aprun -n 2 -d 6 -N 1 ./jacobi_mpi+openmp+cuda 4096 4096 0.15

$ # trace analysis with Vampir
$ module load vampir
$ vampir trace/traces.otf2
```
Conclusions

Score-P

• Common instrumentation and measurement infrastructure for various analysis tools
• Hides away complicated details
• Provides many options and switches for experts

Workflow

• Instrument your application with Score-P
• Perform a measurement run with profiling enabled
• Perform profile analysis with CUBE or scorep-score
• Use scorep-score to define an appropriate filter
• Perform a measurement run with tracing enabled and the filter applied
• Perform in-depth analysis on the trace data with Vampir
If you have any questions or need help, please don't hesitate to contact me under winklerf@ornl.gov.

Detailed information under: http://www.vi-hps.org/projects/score-p