Score-P & Vampir
Comprehensive Multi-Paradigm Performance Analysis

Ronny Brendel
Technische Universität Dresden
ronny.brendel@tu-dresden.de

Argonne Training Program on Extreme-Scale Computing (ATPESC), St. Charles, IL, 7th August 2018
Introduction
Why Bother Analyzing Performance

• There are countless ways to leave performance on the table
  - Lots of little function calls due to e.g. constructors/destructors
  - Inefficient parallelization
  - Lack of vectorization
  - Bad memory access patterns / cache usage
  - Bad file I/O usage
  - …

• Many performance tools are really easy to use
  - Just try it out on your code or pet project

• There are also things performance tools cannot help you with
  - Different/better algorithms
Introduction

Methods

Sampling

- Interrupt in given intervals (typically ~10ms)
- Statistical correctness guarantees

Instrumentation

- Callback before and after event
- Exact times and counts
- Wrappers have easy access to function arguments
Introduction

Methods

- Presentation
  - Statistics
  - Timelines

- Recording
  - Summarization
  - Logging

- Interception
  - Sampling
  - Instrumentation
**Introduction**

Methods

- Presentation
  - Profiling
    - Statistics
  - Recording
    - Summarization
- Interception
  - Sampling
  - Instrumentation
- Tracing
  - Timelines
  - Logging
Introduction
Methods

Profile

- Information accumulated into buckets
- Typically small overhead
- Typically Static representation

Trace

- Event log
- Possibly large overhead
- Interactive representation

<table>
<thead>
<tr>
<th>Time (%)</th>
<th>Time (s)</th>
<th>Function name</th>
<th>Time (s)</th>
<th>Function name</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.44</td>
<td>1.21</td>
<td>QListData::isEmpty</td>
<td>4.68</td>
<td>QListData::isEmpty</td>
</tr>
<tr>
<td>2.96</td>
<td>0.66</td>
<td>QHash::findNode</td>
<td>3.95</td>
<td>QHash::findNode</td>
</tr>
<tr>
<td>2.67</td>
<td>0.60</td>
<td>QList::last</td>
<td>2.67</td>
<td>QList::last</td>
</tr>
<tr>
<td>1.71</td>
<td>0.38</td>
<td>handleEnter</td>
<td>1.71</td>
<td>handleEnter</td>
</tr>
<tr>
<td>0.58</td>
<td>0.13</td>
<td>QHash::find</td>
<td>0.58</td>
<td>QHash::find</td>
</tr>
</tbody>
</table>
Introduction
Methods

• Trade-offs
  - Ease of use
  - Run time overhead and recording size
  - Accuracy
  - API semantics (e.g. MPI_Send's sender and receiver processes and transferred bytes)

• Most tools combine both sampling (+ call stack unwinding) and instrumentation of library events (e.g. MPI, OpenMP and CUDA library functions)
  - To avoid problems with some techniques while gathering enough information
Introduction

Vampir

- Comprehensive, powerful performance data visualization
- Developed since 1996
- Commercial
Introduction
Score-P

- Jointly developed performance data collector
- Developed since 2009
- Open-source (3-clause BSD)
- Partners:
  - TU Dresden, GER
  - FZ Jülich, GER
  - TU München, GER
  - University of Oregon, USA
  - RWTH Aachen; TU Darmstadt; Gesellschaft für numerische Simulation mbH; German Research School for Simulation Sciences GmbH (all GER)
Introduction
Score-P

• Supports:
  - C, C++, Fortran
  - MPI, SHMEM
  - OpenMP, PThreads
  - CUDA, OpenACC, OpenCL
  - Compilers: Cray, GNU, IBM, Intel, Pathscale, PGI, (LLVM)
Tutorial
Data Collection with Score-P
Tutorial
Data Collection with Score-P

- Load Score-P at ANL
  
  \$ module load scorep

- Compile & Link with MPI
  
  \$ scorep ... gcc ... main.c

- Compile & Link with SHMEM
  
  \$ scorep ... mpicc ... main.c

- CMake
  
  \$ SCOREP_WRAPPER=OFF cmake -DCMAKE_C_COMPILER=scorep-gcc ..
  \$ SCOREP_WRAPPER_INSTRUMENTER_FLAGS="..." SCOREP_WRAPPER_COMPILER_FLAGS="..." make

- Autotools
  
  \$ SCOREP_WRAPPER=OFF ../configure CC=scorep-gcc MPICC=scorep-mpicc ..
  \$ SCOREP_WRAPPER_INSTRUMENTER_FLAGS="..." SCOREP_WRAPPER_COMPILER_FLAGS="..." make
**Tutorial**

Data Collection with Score-P

- **Execute**

  ```
  $ ./a.out
  $ mpirun -np 2 ./a.out
  $ shmemrun -np 2 ./a.out
  ```

- **Inspect**

  ```
  $ ls -R
  scorep-20170323_1309_7243761919249966 a.out
  ./scorep-20170323_1309_7243761919249966:
  profile.cubex scorep.cfg
  ```

- **Inspect > Cube**

  ```
  $ cube scorep-20170323_1309_7243761919249966/profile.cubex
  ```
Tutorial
Profile Visualization with Cube
Tutorial
Data Collection with Score-P

- Runtime Options
  - Profiling (default)
    
    ```
    $ export SCOREP_ENABLE_PROFILING=true
    ```
  - Tracing
    
    ```
    $ export SCOREP_ENABLE_TRACING=true
    ```
  - Performance counters
    
    ```
    $ export SCOREP_METRIC_PAPI=PAPI_L2_TCM,...
    ```
  - Filtering
    
    ```
    $ export SCOREP_FILTERING_FILE=my.filt
    ```
  - Memory (default: 16M)
    
    ```
    $ export SCOREP_TOTAL_MEMORY=400M
    ```
  - And many more...
    
    ```
    $ scorep-info config-vars
    ```
Tutorial
Trace Visualization with Vampir

```bash
$ export SCOREP_ENABLE_PROFILING=false
$ export SCOREP_ENABLE_TRACING=true
$ export SCOREP_METRIC_PAPI=PAPI_TOT_INS,PAPI_TOT_CYC

$ mpirun -np 4 ./a.out

$ ls -R
scorep-20170323_1309_7243761919249966  a.out
./scorep-20170323_1309_7243761919249966:
  scorep.cfg  traces/  traces.def  traces.otf2

$ module load vampir

$ vampir scorep-20170323_1309_7243761919249966/traces.otf2
```
Tutorial
Trace Visualization with Vampir
Tutorial
Data Collection Overhead

- Trace size and overhead varies greatly with event rate
  - Make a reference run and check wall clock time!
  - Rule of thumb: Try to stay below 10% overhead

→ Filtering is an integral part of Score-P’s workflow
Tutorial
Data Collection with Score-P

• Score-P workflow as presented so far:
  1) Instrument & build
  2) Execute
  3) Analyze profile using Cube
Tutorial
Data Collection with Score-P

- Score-P workflow with filtering
  1) Instrument & build
  2) Execute (profiling)
  3) Analyze overhead
     If the estimated trace size is too large, filter and goto 3
  4) Execute using the filter (tracing)
  5) Analyze trace using Vampir
### Tutorial

**Data Collection with Score-P**

3) **Analyze Overhead**

```sh
$ scorep-score scorep-20170323_1309_7243761919249966/profile.cubex
```

Estimated aggregate size of event trace: **40GB**

Estimated requirements for largest trace buffer (max_buf): **6GB**

Estimated memory requirements **(SCOREP_TOTAL_MEMORY): 6GB**

(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.)

```
<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>5,383,272,006</td>
<td>1,635,443,611</td>
<td>579.23</td>
<td>100.0</td>
<td>0.35</td>
<td>ALL</td>
<td></td>
</tr>
<tr>
<td>USR</td>
<td>5,358,738,138</td>
<td>1,631,138,913</td>
<td>253.00</td>
<td>43.7</td>
<td>0.16</td>
<td>USR</td>
<td></td>
</tr>
<tr>
<td>OMP</td>
<td>23,580,522</td>
<td>4,089,856</td>
<td>318.79</td>
<td>55.0</td>
<td>77.95</td>
<td>OMP</td>
<td></td>
</tr>
<tr>
<td>COM</td>
<td>665,210</td>
<td>182,120</td>
<td>0.90</td>
<td>0.2</td>
<td>4.95</td>
<td>COM</td>
<td></td>
</tr>
<tr>
<td>MPI</td>
<td>288,136</td>
<td>32,722</td>
<td>6.55</td>
<td>1.1</td>
<td>200.11</td>
<td>MPI</td>
<td></td>
</tr>
</tbody>
</table>
```
Tutorial
Data Collection with Score-P

3) Analyze Overhead

```
$ scorep-score -r scorep-20170323_1309_7243761919249966/profile.cubex
[...]

<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>5,383,272,006</td>
<td>1,635,443,611</td>
<td>579.23</td>
<td>100.0</td>
<td>0.35</td>
<td>ALL</td>
<td></td>
</tr>
<tr>
<td>USR</td>
<td>5,358,738,138</td>
<td>1,631,138,913</td>
<td>253.00</td>
<td>43.7</td>
<td>0.16</td>
<td>USR</td>
<td></td>
</tr>
<tr>
<td>OMP</td>
<td>23,580,522</td>
<td>4,089,856</td>
<td>318.79</td>
<td>55.0</td>
<td>77.95</td>
<td>OMP</td>
<td></td>
</tr>
<tr>
<td>COM</td>
<td>665,210</td>
<td>182,120</td>
<td>0.90</td>
<td>0.2</td>
<td>4.95</td>
<td>COM</td>
<td></td>
</tr>
<tr>
<td>MPI</td>
<td>288,136</td>
<td>32,722</td>
<td>6.55</td>
<td>1.1</td>
<td>200.11</td>
<td>MPI</td>
<td></td>
</tr>
</tbody>
</table>

USR 1,716,505,830 | matmul_sub_ | 79.32 | 13.7 | 0.15 |
USR 1,716,505,830 | matvec_sub_ | 53.44 | 9.2 | 0.10 |
USR 1,716,505,830 | binvrhs_ | 111.47 | 19.2 | 0.21 |
USR 1,716,505,830 | binvrhs_ | 2.76 | 0.5 | 0.12 |
USR 1,716,505,830 | lhsinit_ | 4.37 | 0.8 | 0.19 |
USR 1,716,505,830 | exact_solution_ | 1.63 | 0.3 | 0.09 |
```
Tutorial
Data Collection with Score-P

3) Create filter

```bash
$ cat myfilter.filt
SCOREP_REGION_NAMES_BEGIN
    EXCLUDE
        matmul_sub*
        matvec_sub*
        binvcrhs*
        Binvrhs*
        exact_solution*
        lhs*init*
        timer_*
SCOREP_REGION_NAMES_END

$ scorep-score -f myfilter.filt scorep-20170323*/profile.cubex

Estimated aggregate size of event trace: 409MB
Estimated requirements for largest trace buffer (max_buf): 58MB
Estimated memory requirements (SCOREP_TOTAL_MEMORY): 70MB
(hint: When tracing set SCOREP_TOTAL_MEMORY=70M to avoid)

[...]
```
Tutorial
Data Collection with Score-P

4) Execute using the filter

```
$ exportSCOREP_ENABLE_TRACING=true
$ exportSCOREP_TOTAL_MEMORY=70M
$ exportSCOREP_FILTERING_FILE=myfilter.filt

$ mpirun -np 8 ./a.out
```

Compile-time filtering (GCC-only)

```
$ scorep --instrument-filter=myfilter.filt gcc main.c

$ exportSCOREP_ENABLE_TRACING=true
$ exportSCOREP_TOTAL_MEMORY=70M

$ mpirun -np 8 ./a.out # no runtime filtering needed
```
Tutorial
Trace Visualization with Vampir (Live)
Tutorial
Trace Visualization with Vampir

![Trace Visualization with Vampir](image-url)
Tutorial
Getting Help

- $ scorep --help
- $ scorep-wraper --help
- $ scorep-info config-vars
- VISI-HPS offers trainings (Invite them!)
  - http://www.visi-hps.org/training/tws/
  - http://www.visi-hps.org/training/material/
- https://www.alcf.anl.gov/vampir
- support@score-p.org, service@vampir.eu
- http://score-p.org
- https://vampir.eu
Conclusions, Acknowledgments
Conclusions

- Holistic, powerful and detailed software performance analysis
  - Everything in one picture
  - Extremely customizable
  - Extremely scalable
  - Advanced features
  - Very active in adopting new features

- Active research and development community

- Continuously selected by the OLCF

- Enabler for science at extreme scale
Sponsors & Projects
Contributors

- **Score-P**

- **Vampir**
  Alfred Arnold, Andreas Knüpfer, Bert Wesarg, Frank Winkler, Hartmut Mix, Heide Rohling, Holger Brunst, Jens Doleschal, Johannes Ziegenbalg, Matthias Weber, Laszlo Barabas, Michael Heyde, Michael Peter, Reinhard Neumann, Ronald Geisler, Ronny Brendel, Thomas William, Wolfgang E. Nagel
Hands-On

https://goo.gl/A12mta

or

titan.ccs.ornl.gov → $WORLDWORK/stf010/brendel/atpesc