Performance Analysis with Vampir

Frank Winkler (frank.winkler@tu-dresden.de)
Performance tools will not automatically make you code run faster. They help you understand, what your code does and where to put in work.
Welcome to the Vampir Tool Suite

- Mission
- Profiling versus Tracing
- Event Trace Visualization

The Vampir Workflow

- Score-P: Instrumentation & Run-Time Measurement
- Vampir & VampirServer

Vampir Performance Charts

Vampir Demo

- Tracing and Visualizing the RandomAccess Benchmark

Conclusions
Mission

- Visualization of dynamics of complex parallel processes
- Requires two components
  - Monitor/Collector (Score-P)
  - Charts/Browser (Vampir)

Typical questions that Vampir helps to answer:

- What happens in my application execution during a given time in a given process or thread?
- How do the communication patterns of my application execute on a real system?
- Are there any imbalances in computation, I/O or memory usage and how do they affect the parallel execution of my application?
Profiling versus Tracing

Profile:
Summarization of events over execution interval

- Recording of aggregated information
  - Total, maximum, minimum, …
- For measurements
  - Time
  - Counts
Profiling versus Tracing

Event Trace: Chronologically ordered sequence of event records

- Recording information about significant points (events) during execution of the program
  - Enter / leave of a region
  - Send / receive a message
  - Save information in event record
    - Timestamp, location, event type
    - Plus event-specific information
Show dynamic run-time behavior graphically at any level of detail

Provide statistics and performance metrics

Timeline charts

- Show application activities and communication along a time axis

Summary charts

- Provide quantitative results for the currently selected time interval
Agenda

Welcome to the Vampir Tool Suite
- Mission
- Profiling versus Tracing
- Event Trace Visualization

The Vampir Workflow
- Score-P: Instrumentation & Run-Time Measurement
- Vampir & VampirServer

Vampir Performance Charts

Vampir Hands-on
- Tracing and Visualizing the RandomAccess Benchmark

Conclusions
Score-P: Instrumentation & Run-Time Measurement

- Scalable Performance Measurement Infrastructure for Parallel Codes
- Supports a number of analysis tools
  - Periscope, Tau, Scalasca, Vampir
- Comes together with:
  - New Open Trace Format Version 2
  - CUBE4 profiling format
  - Opari2 instrumentor

- New BSD Open Source license
Score-P: Functionality

- Provide typical functionality for HPC performance tools
- Instrumentation (various methods)
  - Score-P compiler wrapper
- Flexible measurement without re-compilation:
  - Basic and advanced profile generation
  - Event trace recording
  - Online access to profiling data

- MPI, OpenMP, CUDA, and hybrid parallelism (and serial)
- Prototype with OpenSHMEM support
Score-P: Architecture

Vampir

Scalasca

TAU

Periscope

Event traces (OTF2)

Call-path profiles (CUBE4, TAU)

Hardware counter (PAPI, rusage)

MPI

POMP2

CUDA

Compiler

TAU

User

Application (MPI×OpenMP×CUDA)

PMPI

OPARI 2

CUDA

Compiler

PDT

User

Instrumentation wrapper

Score-P measurement infrastructure

Online interface

OpenSHMEM @ Annapolis, March 4-6, 2014 – Frank Winkler – Slide 11
Score-P: Measurement Options

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
[...]
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
```

Profiles can be analyzed with **scorep-score**

- Helps to define appropriate filters for a tracing run

OpenSHMEM @ Annapolis, March 4-6, 2014 – Frank Winkler – Slide 12
Vampir Tool Suite Workflow

1. Instrument your application with Score-P

   CC=icc  
   CXX=icpc  
   F90=ifc  
   MPICC=mpicc

2. Perform a measurement run with profiling enabled

3. Use **scorep-score** to define an appropriate filter

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

5. Perform in-depth analysis on the trace data with **Vampir**
Vampir – Visualization Modes (1)

Directly on front end or local machine

% vampir
On local machine with remote VampirServer

% vampirserver start -n 12

% vampir
Welcome to the Vampir Tool Suite
- Mission
- Profiling versus Tracing
- Event Trace Visualization

The Vampir Workflow
- Score-P: Instrumentation & Run-Time Measurement
- Vampir & VampirServer

Vampir Performance Charts

Vampir Demo
- Tracing and Visualizing the RandomAccess Benchmark

Conclusions
Main Charts of Vampir

Timeline Charts:
- Master Timeline
- Process Timeline
- Counter Data Timeline
- Performance Radar

Summary Charts:
- Function Summary
- Process Summary
- Communication Matrix View
Vampir: Charts for a WRF Trace with 64 Processes
Process and Counter Timeline

Vampir - Trace View - /home/dolescha/tracefiles/feature-traces/wrf-p64-io-mem-usage/wrf.1h.otf

Process Timeline

Function Legend

Property | Value
---|---
Display | Process Timeline
Type | Function
Function Name | MPI_Wait
Function Group | MPI
Interval Begin | 18.121 s
Interval End | 18.121 s
Duration | 0 s
Source File
Source Line

Process 0, Values of Counter "MEM_APP_ALLOC" over Time

50 M
0 M

Process 63, Values of Counter "ru_utime" over Time

750 k
0 k
Welcome to the Vampir Tool Suite
• Mission
• Profiling versus Tracing
• Event Trace Visualization

The Vampir Workflow
• Score-P: Instrumentation & Run-Time Measurement
• Vampir & VampirServer

Vampir Performance Charts

Vampir Demo
• Tracing and Visualizing the RandomAccess Benchmark

Conclusions
HPC Challenge Benchmark for measuring GUPS

GUPS (Giga Updates per Second) is a measurement that profiles the memory architecture of a system (similar to MFLOPS)

GUPS is calculated by identifying the number of memory locations that can be randomly updated in one second, divided by 1 billion (1e9)
Instrumentation of RandomAccess:

- Load Score-P module:
  
  ```
  % module load scorep
  ```

- Get a list of all instrumentation options:
  
  ```
  % scorep --help
  ...
  --mpp=<paradigm>[:<variant>]
  Possible paradigms and variants are:
  none
  No multi-process support.
  mpi
  MPI support using library wrapping
  shmem
  SHMEM support using library wrapping
  ```
Vampir Demo: Score-P Profile of RandomAccess

Change compiler command in Makefile:

```
% cd gups-shmem
% vi Makefile
...
CC = gcc → CC = scorep -mpp=shmem gcc
CXX = gcc → CXX = scorep -mpp=shmem gcc
```

Compile RandomAccess:

```
% make
scorep --mpp=shmem gcc -Iinclude -I/opt/sgi/mpt/mpt-2.03/include/mpp/ \ 
-I/opt/sgi/mpt/mpt-2.03/include/ -O3 -c RandomAccess.c
...
```

Configure Score-P measurement and create a profile:

```
% export SCOREP_ENABLE_PROFILING=true
% export SCOREP_ENABLE_TRACING=false
% export SCOREP_EXPERIMENT_DIRECTORY=scorep_profile_ra_shmem
% mpirun -np 16 ./ra_shmem
```
Vampir Demo: Score-P Profile Analysis of RandomAccess

- Creates experiment directory ./scorep_profile_ra_shmem containing
  - a record of the measurement configuration (scorep.cfg)
  - the analysis report that was collated after measurement (profile.cubex)

```
% ls
... scorep_profile_ra_shmem
% ls scorep_profile_ra_shmem
profile.cubex scorep.cfg
```
Report scoring as textual output:

```bash
% scorep-score -r -c 2 scorep_profile_ra_shmem/profile.cubex
```

Estimated aggregate size of event trace:
Estimated requirements for largest trace buffer (max_tbc):

(hint: When tracing set SCOREP_TOTAL_MEMORY > max_tbc to avoid intermediate flushes or reduce requirements using file listing names of USR regions to be filtered.)

<table>
<thead>
<tr>
<th>flt type</th>
<th>max_tbc</th>
<th>time</th>
<th>% region</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>21645030</td>
<td>13.32</td>
<td>100.0</td>
</tr>
<tr>
<td>USR</td>
<td>21645030</td>
<td>13.32</td>
<td>100.0</td>
</tr>
<tr>
<td>USR</td>
<td>8653590</td>
<td>9.11</td>
<td>68.4</td>
</tr>
<tr>
<td>USR</td>
<td>4325376</td>
<td>0.42</td>
<td>3.1</td>
</tr>
<tr>
<td>USR</td>
<td>4325376</td>
<td>0.42</td>
<td>3.1</td>
</tr>
<tr>
<td>USR</td>
<td>4325376</td>
<td>0.44</td>
<td>3.3</td>
</tr>
<tr>
<td>USR</td>
<td>13728</td>
<td>0.42</td>
<td>3.2</td>
</tr>
</tbody>
</table>

350 MB total memory
About 20 MB per rank
Configure Score-P measurement and create a trace:

```bash
% export SCOREP_ENABLE_PROFILING=false
% export SCOREP_ENABLE_TRACING=true
% export SCOREP_EXPERIMENT_DIRECTORY=scorep_trace_ra_shmem
% export SCOREP_METRIC_RUSAGE=ru_utime,ru_stime
% export SCOREP_TOTAL_MEMORY=50M
% mpirun -np 16 ./ra_shmem
```

Separate trace file per thread written straight into new experiment directory ./scorep_trace_ra_shmem

Interactive trace exploration with Vampir

```bash
% module load vampir
% vampir scorep_trace_ra_shmem/traces.otf2
```
Vampir Demo: Visualizing RandomAccess

Timeline

0.00 s  0.25 s  0.50 s  0.75 s

Master thread:0
Master thread:1
Master thread:2
Master thread:3
Master thread:4
Master thread:5
Master thread:6
Master thread:7
Master thread:8
Master thread:9
Master thread:10
Master thread:11
Master thread:12
Master thread:13
Master thread:14
Master thread:15

0.908 s

Function Summary

All Processes, Accumulated Exclusivity

0 s
10 s

shmem.h

Application

Communication Matrix View

Longest Transfer Time

Ma...:0  40 ms
Ma...:4  30 ms
Ma...:8  20 ms
Ma...:12 10 ms
Ma...:0  0 ms

Process Summary

Similar Processes, Accumulated Exclusivity

0 s

shm...all
shm...all

Function Legend

Monitor
Application
shmem.h
Agenda

Welcome to the Vampir Tool Suite
- Mission
- Profiling versus Tracing
- Event Trace Visualization

The Vampir Workflow
- Score-P: Instrumentation & Run-Time Measurement
- Vampir & VampirServer

Vampir Performance Charts

Vampir Demo
- Tracing and Visualizing the RandomAccess Benchmark

Conclusions
Conclusions

Vampir & VampirServer

- Interactive trace visualization and analysis
- Intuitive browsing and zooming
- Scalable to large trace data sizes (20 TByte)
- Scalable to high parallelism (200000 processes)
- Vampir for Linux, Windows and Mac OS

Score-P

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

OpenSHMEM @ Annapolis, March 4-6, 2014 – Frank Winkler – Slide 33
Vampir is available at http://www.vampir.eu
Get support via vampirsupport@zih.tu-dresden.de
Score-P: http://www.vi-hps.org/projects/score-p