S9347: Performance Analysis for Large Scale GPU Applications and DL Frameworks

Dr. Guido Juckeland / Robert Henschel
Head Computational Science Dept. / Director of Science CommunityTools at Indiana University
Agenda

What to expect from the next 80 minutes

- Motivation
- Generating profiles and trace files with Score-P
- Visualizing trace files with Vampir
- Looking into Deep Learning Frameworks
Disclaimer

It's extremely easy to waste performance

- Poor/no GPU usage (80-90%)
- Bad MPI (50-90%)

- Total: 1% of peak (or worse)

- Performance tools will not “automagically” make your code faster – they just point to “areas of interest”
Motivation

Performance Tuning 101
Profiling vs. Tracing

Preserving the details

Statistics

<table>
<thead>
<tr>
<th>Function</th>
<th>Number of Invocations</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>![Bar Chart for main]</td>
</tr>
<tr>
<td>bar</td>
<td>![Bar Chart for bar]</td>
</tr>
<tr>
<td>foo</td>
<td>![Bar Chart for foo]</td>
</tr>
</tbody>
</table>

Timelines

- main
- foo
- bar
- foo
Sampling

Periodic observations of your application (Pull)

- Running program is periodically interrupted to take measurement
- Statistical inference of program behavior
  - Not very detailed information on highly volatile metrics
  - Requires long-running applications
- Works with unmodified executables
Instrumentation

Modify application to deliver information (Push)

- Measurement code is inserted such that every event of interest is captured directly
- Advantage:
  - Much more detailed information
- Disadvantage:
  - Processing of source-code / executable necessary
  - Large relative overheads for small functions
Sampling vs. Tracing

Comparing both approaches visually

Function Instrumentation:

Sampling:
Sampling + Instrumentation

Combining the best of both worlds

- Long running applications:
  - Requires large buffers or heavy filtering
  - Creating a filter requires runs in advance
- Codes with many small functions (e.g.: C++):
  - Function instrumentation a challenge
- Score-P: Sampling+Tracing
Terms and How They Relate
Making sure we use the same words
Summary

Making the “right” choices

**SO, YOU HAVE DECIDED TO UNDERSTAND WHAT A PROGRAM EXACTLY DOES?**

- **CONGRATULATIONS!!!**
  - YOU ARE AHEAD OF 99% OF YOUR COLLEAGUES

- **ARE YOU SERIOUS ABOUT THIS?**
  - **YES!**
  - **SLACKER**
  - **GO USE gprof OR THE LIKE ...**

- **WHAT KIND OF PROFESSIONAL ARE YOU?**
  - **SCIENTIST**
  - **ENGINEER**

- **SELECT MAGNIFICATION**
  - LOW
  - MED
  - HIGH

- **FIND INTERESTINGS SPOTS**
  - **RUN IN PROFILING MODE**
  - **USE COMPILER WRAPPERS + FILTERS**
  - **INSTRUMENT THE CRITICAL PARTS**
Generating Traces and Profiles
with Score-P
Overall workflow

Recording and studying performance data

- Attach Score-P to application
- Run with attached monitor ==> trace/profile data
- Study trace with Vampir / profile with Cube
- Repeat to:
  - Adapt instrumentation ("what you measure")
  - Evaluate result of a change
Attaching Score-P

a.k.a. instrumenting your source code

CC       = pgcc
CXX     = pgCC
F90      = pgf90
MPICC = mpicc
NVCC = nvcc

CC       = scorep <options> pgcc
CXX     = scorep <options> pgCC
F90      = scorep <options> pgf90
MPICC = scorep <options> mpicc
NVCC = scorep <options> nvcc

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

Common options are:
...
--instrument-filter=<file>
    Specifies the filter file for filtering functions during compile-time. It applies the same syntax, as the one used by Score-P during run-time.

--user    Enables user instrumentation.
Attaching Score-P

Instrument once – change measurement via runtime variables

```
$ 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
$ export SCOREP_ENABLE_TRACING=false
$ export SCOREP_EXPERIMENT_DIRECTORY=profile

$ mpirun <instrumented binary>

Profiling Example
```
Combined Sampling+Tracing

Available since Score-P 2.0

- User code is sampled (pull)
- Runtime libraries with tracing support use events (push):
  - MPI
  - OpenMP / OpenACC / pthreads
  - CUDA / OpenCL
  - I/O

```bash
$ export SCOREP_ENABLE_TRACING=true
$ export SCOREP_ENABLE_UNWINDING=true
$ export SCOREP_SAMPLING_EVENTS=perf_cycles@2000000
```
# Things to look at

## What can Score-P record?

### User Functions
- C/C++/Fortran
- Sampling *NEW*
- Custom regions
- Java
- Python (*Experimental*)

### Parallel Paradigms
- MPI
- Pthreads
- OpenMP
- XeonPhi Native *NEW*
- CUDA
- OpenACC/OpenCL *NEW*
- OpenShmem (+Cray)
- I/O (*Experimental*)

### Hardware
- Performance counters (PAPI)
- Plugin counters

### Operating System
- Resource usage
GPU Tracing

Example CUDA and OpenACC

- Can be used in combination
- Also supports CUPTI counters

```bash
$ export SCOREP_ENABLE_TRACING=yes
$ export SCOREP_TIMER=clock_gettime
$ export SCOREP_CUDA_ENABLE=driver,kernel,memcpy,flushatexit
$ export SCOREP_OPENACC_ENABLE=yes
$ export ACC_PROFLIB=${SCOREP_LIB}/libscorep_adapter_openacc_event.so
```
Limitations

Why tracing is hard

- Event tracing requires trade-offs:
  - Only add the data sources you need
  - Limit granularity (i.e., filtering)
- Score-P is a profiling experiment

- Adds Overhead at runtime
  => Overhead must be low for meaningful performance analysis

- Temporarily stored in main memory
  Limited size

1.0 ms
DEMO:

Generating Traces and Profiles with Score-P
Visualizing

Profiles with CUBE
Traces with Vampir
Bringing it all together

Score-P + Analysis Tools

- Vampir
- Scalasca
- CUBE
- TAU
- TAUdb
- Periscope

Score-P measurement infrastructure

- Hardware counter (PAPI, rusage)
- Online interface

Instrumentation wrapper

- Process-level parallelism (MPI, SHMEM)
- Thread-level parallelism (OpenMP, Pthreads)
- Accelerator-based parallelism (CUDA, OpenCL, OpenACC)
- Source code instrumentation
- User instrumentation

Application

- Event traces (OTF2)
- Call-path profiles (CUBE4, TAU)

Accelerator-based parallelism (CUDA, OpenCL, OpenACC)
CUBE
Interactive profile analysis

What kind of performance metric?

Where is it in the source code?
In what context?

How is it distributed across the processes/threads?
Vampir

Interactive trace analysis

Large imbalance instantly visible

>50% time wasted
Vampir

Performance data visualization in a complex environment

- I/O System
- Compute Nodes (Batch jobs)
- Login Nodes
- Desktop System

Trace File (OTF2)
Simplest Approach

Use your desktop system

- Minimal setup (no installations, no batch job)
  - Copying of traces to desktop
  - Only small traces
(Re)Using the HPC Resources

Run analysis engine on compute nodes, GUI on desktop

- Best performance, low response time
  - Tunneling to connect to batch job
  - Installation on desktop system needed

Analysis: VampirServer

TCP Socket connection

Visualization: Vampir
Vampir GUI

What do the fancy colors mean?

Master Timeline
Summary Timeline
Process Timeline
Counter Data Timeline

Function Summary
Communication Matrix View
Process Summary
Vampir GUI
Timeline Charts

- Master Timeline ➞ all threads’ activities over time per thread
- Summary Timeline ➞ all threads’ activities over time per activity
- Performance Radar ➞ all threads’ perf-metric over time
- Process Timeline ➞ single thread’s activities over time
- Counter Data Timeline ➞ single threads perf-metric over time
Vampir GUI

Summary/Profile Charts

Function Summary ➡ runtime/invocation summaries
Message Summary ➡ data transfer statistics
I/O Summary ➡ I/O statistics
Process Summary ➡ Clustering of similar event streams
Communication Matrix View ➡ Pairwise communication statistics
Vampir Performance Charts in Detail

Master Timeline

Detailed information about functions, communication and synchronization events for collection of processes.
Vampir Performance Charts in Detail

Summary Timeline

Fractions of the number of processes that are actively involved in given activities at a certain point in time.
Vampir Performance Charts in Detail

Process Timeline

Detailed information about different levels of function calls in a stacked bar chart for an individual process.
Vampir Performance Charts in Detail

Counter Timeline

Detailed counter information over time for an individual process.
Vampir Performance Charts in Detail

Performance Radar

Detailed counter information over time for a collection of processes.
Vampir Performance Metrics

Where do they come from?
Overview of the accumulated information across all functions and for a collection of processes.
Vampir Performance Charts in Detail

Process Summary

Overview of the accumulated information across all functions and for every process independently.

Clustering: Grouping of similar processes by using summarized function information.
Vampir Performance Charts in Detail

Communication Matrix View
Vampir at Scale

Fit to chart height (feat. 200,000+ event streams)
Comparing Traces with Vampir
Seeing the differences
Zooming in

One iteration of solution 1

One iteration of solution 2

Computation/Communication overlap for solution 3
DEMO:
Visualizing Trace Files with Vampir
Looking into DL Frameworks
Score-P Python Bindings

Tracing/Profiling for all python programs

- Not yet included in main release
- Available on GitHub:
  - https://github.com/score-p/scorep_binding_python
- NSight/nvvp for single node DL frameworks still better (user instrumentation)
- Score-P only choice for MPI-parallel DL frameworks

```bash
$ export SCOREP_ENABLE_PROFILING=true
$ export SCOREP_ENABLE_TRACING=false
$ export SCOREP_EXPERIMENT_DIRECTORY=profile

$ python -m scorep --mpi <script.py>
```
Vampir with Python Traces

It looks all the same
Vampir is available at http://www.vampir.eu
Vampir at IU: https://kb.iu.edu/d/awbv
Get support via vampirsupport@zih.tu-dresden.de
Score-P: http://www.vi-hps.org/projects/score-p