Trace Based Performance Analysis at Large Scale on Jaguar and Titan

Guido Juckeland (guido.juckeland@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.
Agenda

- Introduction
- The Vampir Workflow
- VampirTrace
  - Instrumentation & Run-Time Measurement
- The Vampir Displays
  - “Seeing” Performance Bottlenecks
- Vampir on Jaguar and Titan
  - Scalability & GPU Support
  - How to use Vampir at OLCF
- Conclusions
Introduction

Why bother with performance analysis?

• Efficient usage of expensive and limited resources
• Scalability to achieve next bigger simulation

Profiling and Tracing

• Have an optimization phase
  • just like testing and debugging phase
• Use tools!
• Avoid *do-it-yourself-with-printf* solutions, really!
Sampling vs. Tracing

Sampling

foo bar foo bar foo

2011/06/30 10:15:12.672865 Enter foo
2011/06/30 10:15:12.894341 Leave foo

Tracing

Foo: Total Time 0.0815
Bar: Total Time 0.4711

Titan Summit @ ORNL, Aug 16, 2011 – Guido Juckeland – Slide 5
Program profiles

Statistics

Number of invocations

- foo
- bar

Execution time

- foo
- bar

Timelines

- foo
- bar
- foo
- bar
- foo
What do you need to do for it?

- Use VampirTrace

Instrumentation (automatic with compiler wrappers)

<table>
<thead>
<tr>
<th>Compiler</th>
<th>VampirTrace</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC</td>
<td>icc</td>
</tr>
<tr>
<td>CXX</td>
<td>icpc</td>
</tr>
<tr>
<td>F90</td>
<td>ifc</td>
</tr>
<tr>
<td>MPICC</td>
<td>mpicc</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Compiler</th>
<th>VampirTrace</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC</td>
<td>vtcc</td>
</tr>
<tr>
<td>CXX</td>
<td>vtcxx</td>
</tr>
<tr>
<td>F90</td>
<td>vtf90</td>
</tr>
<tr>
<td>MPICC</td>
<td>vtcc -vt:cc mpicc</td>
</tr>
</tbody>
</table>

Re-compile & re-link

Trace Run (run with appropriate test data set)

More details later
What does VampirTrace do in the background?

### Instrumentation:

- Via compiler wrappers
- By underlying compiler with specific options
- MPI instrumentation with replacement lib
- OpenMP instrumentation with Opari
- Also binary instrumentation with Dyninst
- Partial manual instrumentation
What does VampirTrace do in the background?

- Event data collection
- Precise time measurement
- Parallel timer synchronization
- Collecting parallel process/thread traces
- Collecting performance counters (from PAPI, memory usage, POSIX I/O calls and fork/system/exec calls, CUDA, and more ...)
- Monitor GPU usage
- Filtering and grouping of function calls
Event Trace Visualization

- Alternative and supplement to automatic analysis
- Show dynamic run-time behavior graphically
- Provide statistics and performance metrics
  - Master timeline for parallel processes/threads
  - Process timeline plus performance counters
  - Statistics summary display
  - Message statistics
  - more
- Interactive browsing, zooming, selecting
  - Adapt statistics to zoom level (time interval)
  - Also for very large and highly parallel traces
Vampir Workflow

- Multi-Core Program
- Vampir Trace
- Trace File (OTF)
- Vampir 7
- VampirServer
- Many-Core Program
- Trace Bundle

Titan Summit @ ORNL, Aug 16, 2011 – Guido Juckeland – Slide 11
Vampir

GUI to analyze trace files (OTF)
Main concept: Timeline + statistics
GUI is QT based – available on Linux, Mac, Windows
Parallel analysis engine for Vampir

- MPI
- pthreads

Scales to > 10,000 analysis processes

Loads the entire uncompressed trace into memory
Vampir 7: Displays for a WRF trace with 64 processes
Current Scalability Features in Vampir

- **Fit to chart height feature of master timeline and performance radar**
  - Allows visualization of more processes than pixels of screen are available

- **Clustering**
  - Allows detection of outlier processes and groups with similar behavior

- **Performance radar**
  - Highlighting performance conditions of your program in a global timeline
Scalability Feature 1: Fit to chart height: Pflotran initialization + I/O
Scalability feature II: clustering: Pflotran - first iteration
Scalability feature II: clustering: Pflotran – end of initialization & first iteration
Scalability Feature III: Performance Radar

Display objectives:

- Identification of performance relevant trace parts
- Assistance to users to navigate in trace data and to spot interesting sections
- Performance of basic arithmetics on counter data
Scalability Feature II: Performance Radar

Features:

- Detection of occurrences of functions/function groups
- Visualization of call density of functions/function groups to help to find performance relevant candidates
- Construction of filter based on function occurrences over time for further usage in calculations
Scalability Feature II: Performance Radar

Features:

- Performance of arbitrary calculations on counter data, e.g. add up all floating point operations over time, differentiation of performance counter

- Support for concatenation of several calculations

- Utilization of filter in calculations, e.g. only add up FLOPS of function $x$
Vampir I/O Analysis

enable

metadata rate

metadata server load

metadata server load

detailed per file and summary statistics

detailed per event data

Titan Summit @ ORNL, Aug 16, 2011 – Guido Juckeland – Slide 22
Finding Performance Bottlenecks

- Inefficient Communication patterns
- Load imbalance / serial parts of the application
- Memory bound computation
  - Inefficient cache usage
  - TLB misses
  - Use HW counters (PAPI) to detect
- I/O bottlenecks
- Most time consuming function
Scalability Feature I: Fit to chart height: Inefficient MPI_Allgatherv

Detection of inefficient implementation of MPI_Allgatherv
Challenges on Jaguar and Titan

**Scalability**
- Overcome I/O problems
- Find ways to show that much data

**GPU Support**
- Hitting a moving target
- Another layer of heterogenity to display
Scalability

Current limits on Jaguar

• Tracing up to 8000 Processes
  • I/O Problem (too many file creates – one per process)

Prototype already working on Jaguar

• Tracing 200,000+ processes
  • Opened with 20,000 VampirServer processes
200,000+ Processes in Vampir

Titan Summit @ ORNL, Aug 16, 2011 – Guido Juckeland – Slide 27
GPU Support: CUDA (+OpenCL)

Currently done with Library Wrapping

Application \(\rightarrow\) Library

Dynamic Linker

API

Wrapper library

Enter/Leave
Reuse known metrics

Thread = Kernel

Message = cudaMemcpy

Process A (Host) → Kernel A' (GPU)

Host-Memory → GPU-Memory

1234

"foo"

TRUE

0.98
Asynchronous Events

foo | launch | foo | Sync_wait

GPU execution queue

Host

GPU

t
Time stamps for asynchronous events

[Diagram showing time stamps for asynchronous events with notes on GPU execution queue and events.]
PIConGPU – an example for a multi-hybrid application
MPI + pthreads + CUDA
What are the GPUs doing

Green: Running a kernel
Blue: Idle
Connection to host processes
One Process & Thread Group

Titan Summit @ ORNL, Aug 16, 2011 – Guido Juckeland – Slide 38
1. Recompile application using VampirTrace

2. Run the application

3. Start VampirServer

4. Connect Vampir to VampirServer
Step 1. Recompile application using VampirTrace

First: module load vampirtrace

Use the appropriate compiler wrappers
- vtcc, vtCC, vtf77, vtf90

Pick appropriate library (seq, mpi, mt, hyb)
  - e.g. vtcc -vt:hyb (recommended)

Pick instrumentation type
  - -vt:inst compinst (default, compiler instrumentation, all functions)
  - -vt:inst manual (MPI, OpenMP, CUDA and manual)

See what's going on behind -vt:show or -vt:verbose

More details -vt:help

Tell your build system to use VampirTrace

./configure -with-CC="vtcc -vt:hyb" ...
Step 2. Running the application

Run your application as usual

Make sure the VampirTrace module is loaded

- Load module before and `qsub -V`
- `module load vampirtrace` in the runscript

Control VampirTrace by using environment variables

- `VT_PFORM_GDIR=traces`
  - `mkdir $VT_PFORM_GDIR`
  - `lfs setstripe -c 1 $VT_PFORM_GDIR`
  - place trace files in that directory
  - Use `/lustre/widow1` for large runs
Step 2. Running the application (contd.)

Control VampirTrace by using environment variables

- VT_FILTERSPEC=default.filter
  - `cp $VAMPIRTRACE_DIR/etc/default.filter`.
  - Reduces resulting trace size by filtering frequently called functions

- VT_MAX_FLUSHES=20
  - Defaults to 1
  - Allows VT to flush buffer during execution

- VT_BUFFER_SIZE=100M
  - Defaults to 32M
  - Increases internal buffer size

- VT_SYNC_FLUSH=yes
  - Use that if you have collective ops in your code
  - Avoids all processes waiting for one process to do trace I/O
  - But has overhead itself (extra allreduce on all collective ops!)

- VT_METRICS=PAPI_FP_OPS:PAPI_TOT_INS
Step 3+4. Analyze small traces locally (< 100 MB)

Copy the Vampir GUI / Client to your workstation/laptop

- scp home.ccs.ornl.gov:/sw/sources/vampir/client/vampir-7.*-i386.tar.gz .
- tar -xzf vampir-7.*-i386.tar.gz
- ./vampir/bin/vampir

Client is available for:

- Linux {i386, amd64}.tar.gz
- Windows {x86, x64}.exe
- MacOS as .dmg

Copy trace files to your workstation/laptop

- scp -r jaguarpf.ccs.ornl.gov:/tmp/.../traces/ .

Start Vampir on your workstation/laptop

- Open trace and enjoy!
Step 3. Start VampirServer

VampirServer runs on the compute nodes and needs access to the generated trace files (.otf, .z)

Currently available on jaguar, smoky, lens

$ module load vampir

$ qsub -V -I -l size=120,walltime=2:0:0 -A <ACC>

$ vngd-start.sh

Number of processes depends on (size of) the trace

- Try ¼ of the cores used by the application
- Running out of memory → add nodes
- Getting strange MPI errors → reduce process count
Step 4. Connect Vampir to VampirServer (Linux/Mac)

Open an SSH tunnel to the compute node

```bash
jaguarpf$ vngd-start.sh
Launching VampirServer Version 2.3.0 with on 12 processes ...
Found license file: /tmp/work/tilsche/.vampir/etc/lic.dat
Running 11 analysis processes... (abort with Ctrl-C or vngd-shutdown)
Server listens on: nid13928:30051
```

```bash
workstation$ ssh -L 30051:nid13928:30051 jaguarpf.ccs.ornl.gov
```

Open Vampir and “remote open” to

- **Server**: localhost
- **Port**: 30051
Step 4. Connect Vampir to VampirServer (Windows)

We need an SSH tunnel to the compute node

```
jaguarpf$ vngd-start.sh
Launching VampirServer Version 2.3.0 with on 12 processes ...
Found license file: /tmp/work/tilsche/.vampir/etc/lic.dat
Running 11 analysis processes... (abort with Ctrl-C or vngd-shutdown)
Server listens on: nid13928:30051
```

Vampir “remote open” to

- **Server:** localhost
- **Port:** 30051

![PuTTY Configuration](image)
View your application trace:

“Play around” with Vampir to get a feeling about the features
Caution: Effects due to Tracing

I/O overhead (flush)
- Visibly marked in the trace
- ‘Long’ time for I/O
- Ideally only once at the end (invisible) or during barriers
- Avoid by applying runtime filters

Measurement overhead
- Overhead on function calls
- Invisible
- Avoid instrumenting tiny frequently called functions
- Compare total runtime to get an upper bound on overhead
Summary

**Vampir & VampirServer**

- interactive trace visualization and analysis
- intuitive browsing and zooming
- scalable to “quite large” trace data sizes (1.5 TByte)
- scalable to high parallelism (200,000 processes)

**Vampir available for Windows, Linux/Unix and Mac OS X**

**VampirTrace**

- convenient instrumentation and measurement infrastructure
- hides away complicated details
- provides many options and switches for experts

**VampirTrace is part of Open MPI 1.3 and higher**
Acknowledgements

This work would have been impossible without the dedication of:

- Matthias Lieber (Tracing & Analysis)
- Matthias Jurenz (VampirTrace Software & Support)
- Matthias Weber (Vampir Software & Support)

• The Vampir Team:
Further information about VampirTrace & Vampir

- www.nccs.gov/computing-resources/jaguar/software/?software=vampir
- www.vampir.eu
- www.tu-dresden.de/zh/vampirtrace