Vampir Performance Visualization

Argonne Training Program on Extreme-Scale Computing 2016

Matthias Weber (matthias.weber@tu-dresden.de)
It is extremely easy to waste performance!

- Bad MPI (50-90%)
- No node-level parallelism (94%)
- No vectorization (75%)
- Bad memory access pattern (99%)
- In sum: 0.008% of the peak performance (785 GFLOPs of mira)
Performance tools will not automatically make your code run faster. They help you understand, what your code does and where to put in work.
Welcome to the Vampir Tool Suite

- Mission
- Event Trace Visualization
- Parallel Performance Analysis Approaches

The Vampir Workflow

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

Vampir Performance Charts

Vampir Demo

- Tracing and Visualizing NPB-MZ-MPI / BT

Conclusions
Visualization of dynamics of concurrent processes

Two components / steps
- 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?
Event Trace Visualization with Vampir

- Show dynamic run-time behavior graphically at a fine level of detail
- Provide summaries (profiles) on performance metrics

Timeline charts
- Show application activities and communication along a time axis

Summary charts
- Provide quantitative results for the currently selected time interval
Sampling

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

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 + Instrumentation

Function Instrumentation:

Sampling:
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
Profiling vs. Tracing

Statistics

Number of Invocations

<table>
<thead>
<tr>
<th>Function</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>2</td>
</tr>
<tr>
<td>bar</td>
<td>4</td>
</tr>
<tr>
<td>foo</td>
<td>5</td>
</tr>
</tbody>
</table>

Execution Time

Timelines

At ATPESC 2016 - Vampir Performance Visualization
Terms Used and How They Connect

Data Presentation
Data Recording
Data Acquisition
Analysis Layer

Analysis Technique

Profiling
- Statistics
- Summarization

Tracing
- Timelines
- Logging
- Event-based Instrumentation

Sampling
So what is the right choice?

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! What kind of professional are you?

Scientist

Engineer

Select Magnification

Low

Med

High

Get + install the tools

Run in profiling mode

Use compiler wrappers + filters

Instrument the critical parts

Find interesting spots

Slacker

Go use gprof or the like...
Agenda

Welcome to the Vampir Tool Suite
- Parallel Performance Analysis Approaches
- Mission
- Event Trace Visualization

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

Vampir Performance Charts

Vampir Hands-on
- Tracing and Visualizing NPB-MZ-MPI / BT

Conclusions
Tracing – Overview

Workflow:

- Attach Score-P to application
- Run with the attached monitor
  ⇒ Result: trace/profile data
- Analyze the trace with Vampir

Repeat to:

- Adapt instrumentation (“what you measure”)
- Evaluate result of a change
Score-P: Workflow / Instrumentation

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

To see all available options for instrumentation:

```bash
$ 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.
```
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
  [...]  
```

Example for generating a profile:

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

$ mpirun <instrumented binary>
```
Score-P: Sampling + Instrumentation

- Score-P since 2.0 supports a combination of:
  - Instrumentation for MPI/OpenMP events
  - Sampling for everything else

Simple configuration, e.g.:

```
% export SCOREP_ENABLE_TRACING=true
% export SCOREP_ENABLE_UNWINDING=true
% export SCOREP_SAMPLING_EVENTS=perf_cycles@2000000
```
Tracing – Score-P Architecture

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

**Event traces (OTF2)**

**Call-path profiles (CUBE4, TAU)**

**Score-P measurement infrastructure**

**Hardware counter** (PAPI, rusage, PERF, plugins)

**Online interface**

**Instrumentation wrapper**

- **Process-level parallelism** (MPI, SHMEM)
- **Thread-level parallelism** (OpenMP, Pthreads)
- **Accelerator-based parallelism** (CUDA, OpenCL, OpenACC)
- **Source code instrumentation** (Compiler, PDT, User)
- **Sampling interrupts** (PAPI, PERF)

**Application**
Tracing – Data Sources

User Code
- Compiler instrum.
- Compiler plugin
- Sampling *NEW*
- Manual
- Java (*Experimenal*)

Parallel Paradigms
- MPI
- Pthreads
- OpenMP (via Opari)
  - via OMPT (*Experimental*)
    ⇒ XeonPhi Offload
  - XeonPhi Native *NEW*
- CUDA
- OpenACC/OpenCL *NEW*
- OpenShmem (+Cray)

Hardware
- Performance counters (PAPI)
- Plugin counters

Operating System
- Resource usage
Tracing – Instrumentation

Data Sources

**Source level**
- Compiler
- Manual
- OpenMP with Opari2
- Tau PDT

**MPI Profiling Interface**
- MPI

**Runtime/Library**
- Pthreads
- NVIDIA CUDA
- OpenSHMEM (Cray-SHMEM)
- External counters
- Plugin counters
- OpenCL
- OpenACC

**Operating System**
- Resource usage

**Hardware**
- Performance counters
Tracing – The Hard Part

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

- Temporarily stored in main memory
  => Limited size

Event tracing requires trade-off’s:

- Only add the data sources you need
- Limit granularity (i.e., filtering)

Thus: Score-P default is a profiling experiment
Score-P: Workflow / Filtering

• Use scorep-score to define a filter
  – Exclude short frequently called functions from measurement
    • For profiling: reduce measurement overhead (if necessary)
    • For tracing: reduce measurement overhead and total trace size

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

Estimated aggregate size of event trace: 40GB
Estimated requirements for largest trace buffer (max_buf): 10GB
Estimated memory requirements (SCOREP_TOTAL_MEMORY): 10GB

```
Flt type max_buf[B] visits time[s] time[%] time/visit[us] region

USR 3,421,305,420 522,844,416 144.46 13.4 0.28 matmul_sub
USR 3,421,305,420 522,844,416 102.40 9.5 0.20 matvec_sub
USR 3,421,305,420 522,844,416 200.94 18.6 0.38 binvcrhs
USR 150,937,332 22,692,096 5.58 0.5 0.25 binvrhs
USR 150,937,332 22,692,096 13.21 1.2 0.58 lhsinit
```

• Filter file:

```bash
$ vim scorep.filt
```

```
SCOREP_REGION_NAMES_BEGIN EXCLUDE
matmul_sub
matvec_sub
binvcrhs
```

ATPESC 2016 - Vampir Performance Visualization
Traces can become large; Their handling challenging
- Trace size proportional to: number of processes/threads (width), duration (length), and measurement detail (depth)

Intermediate flushes => High degree of perturbation
- Either use less detail or larger trace buffers

Traces should be written to a parallel file systems
- E.g. “/work”, “/scratch”, or “/p1scratchc”

Moving large traces can become challenging
- However, systems with more memory/core can analyze larger traces
- Alternatively, undersubscribe for increased memory/core ratios
Vampir – Visualization Modes (1)

Directly on front end or local machine

% vampir
Vampir – Visualization Modes (2)

On local machine with remote VampirServer

% vampirserver start -n 12

% vampir

VampirServer

Score-P

Many-Core Program

LAN/WAN

Large Trace File (stays on remote machine)

MPI parallel application

Trace File (OTF2)
Visualization: After Tracing

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

Trace File (OTF2)
Visualization: Most simple (Analysis on Desktop)

- Minimal setup (no installations, no batch job)
  - Copying of traces to desktop
  - Only small traces
Visualization: Best Option (Analysis on HPC system)

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

**Analysis:**
- **VampirServer**
- **TCP Socket connection**

**Visualization:**
- **Vampir**
Visualization: Alternative (Analysis on HPC system)

- Simpler setup, no installation on desktop
- X11 forwarding needed (use: ssh –XC …)
- Bandwidth and response time can be critical

Analysis: VampirServer

Visualization: Vampir

TCP

I/O System

Compute Nodes (Batch jobs)

Login Nodes

Desktop System

Trace File (OTF2)
Visualization: Most simple (Analysis on Frontend)

+ Minimal setup (no installations, no batch job)
- X11 forwarding, bandwidth, and response
- Only small traces

Visualization and analysis: **Vampir**
Agenda

Welcome to the Vampir Tool Suite
- Parallel Performance Analysis Approaches
- Mission
- Event Trace Visualization

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

Vampir Performance Charts

Vampir Demo
- Tracing and Visualizing the NPB-MZ-MPI / BT

Conclusions
**Vampir: Main Performance Charts**

**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*

**Summary Charts**
- Function Summary
- Process Summary
- Message Summary
- Communication Matrix View
- I/O Summary
Vampir: Performance Charts

Trace visualization of FDS (Fire Dynamics Simulator)

- Master Timeline
- Summary Timeline
- Process Timeline
- Counter Data Timeline
- Function Summary
- Communication Matrix View

ATPESC 2016 - Vampir Performance Visualization
Slide 34
Detailed information about functions, communication and synchronization events for collection of processes.
Summary Timeline

Fractions of the number of processes that are actively involved in given activities at a certain point in time.
Detailed information about different levels of function calls in a stacked bar chart for an individual process.
Counter Timeline

Detailed counter information over time for an individual process.
Detailed counter information over time for a collection of processes.
Vampir: Where Do the Metrics Come From?

Custom Metrics Built-In Editor

- FLOPS in User Defined
- MPI Latencies
- Message Data Rate
- Message Transfer Time
- Message Volume in
- Number of Hits
- Number of Invocations
- Simultaneous Message
- Time Spent in MPI

Description: Wait Time
Unit: 1/s

Metric: Function Duration
Operation: Add
Metric: Inclusive

Apply  Cancel  OK
Overview of the accumulated information across all functions and for a collection of processes.
Vampir: Performance Charts

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

Communication Matrix View

Number of Messages

Receiver

Sender

Master thread:0 193 193 193
Master thread:1 339 193 193 193
Master thread:2 339 193 193 193
Master thread:3 146 193 193 193
Master thread:4 339 193 193 193
Master thread:5 146 193 193 193
Master thread:6 146 193 193 193
Master thread:7 146 193 193 193
Master thread:8 146 193 193 193
Master thread:9 146 193 193 193
Master thread:10 146 193 193 193
Master thread:11 146 193 193 193
Master thread:12 146 193 193 193
Master thread:13 146 193 193 193
Master thread:14 146 193 193 193
Master thread:15 146

350 330 310 300 290 280 270 260 250 240 230 220 210 200 190 180 170 160 150 140 130
Vampir at Scale: FDS with 8192 cores

Fit to chart height feature in Master Timeline

Overview of the entire application run across all processes based on available pixels on screen.
Welcome to the Vampir Tool Suite
- Parallel Performance Analysis Approaches
- Mission
- Event Trace Visualization

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

Vampir Performance Charts

Vampir Demo
- Tracing and Visualizing NPB-MZ-MPI / BT → tonight

Conclusions
Agenda

Welcome to the Vampir Tool Suite
- Parallel Performance Analysis Approaches
- Mission
- Event Trace Visualization

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

Vampir Performance Charts

Vampir Demo
- Tracing and Visualizing NPB-MZ-MPI / BT

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 X

Score-P

- Common instrumentation and measurement infrastructure for various analysis tools
- Hides away complicated details
- Provides many options and switches for experts
The people behind Vampir, Score-P, and OTF2:

**Active**
- Dr. Holger Brunst
- Jens Doleschal
- Ronald Geisler
- Tobias Hilbrich
- Matthias Jurenz
- Dr. Andreas Knüpfer
- Dr. Hartmut Mix
- Prof. Wolfgang E. Nagel
- Ronny Tschüter
- Michael Wagner
- Matthias Weber
- Bert Wesarg
- Thomas William
- Johannes Ziegenbalg

**Retired**
- Alfred Arnold
- Laszlo Barabas
- Ronny Brendel
- Heike McCraw/Jagode
- Shino Mathukutty George
- Daniel Hackenberg
- Robert Henschel
- Dr. Matthias Müller
- Reinhard Neumann
- Frank Noack
- Michael Peter
- Heide Rohling
- Johannes Spazier
- Frank Winkler
- Manuela Winkler
Vampir is available at http://www.vampir.eu
Vampir at Argonne NL: https://www.alcf.anl.gov/vampir
Get support via vampirsupport@zih.tu-dresden.de
Score-P: http://www.vi-hps.org/projects/score-p
Agenda

Welcome to the Vampir Tool Suite
- Parallel Performance Analysis Approaches
- Mission
- Event Trace Visualization

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

Vampir Performance Charts

Vampir Demo
- Tracing and Visualizing NPB-MZ-MPI / BT → tonight

Conclusions
The NAS Parallel Benchmark suite (MPI+OpenMP version)

- Available from: http://www.nas.nasa.gov/Software/NPB
- 3 benchmarks in Fortran77 (bt-mz, lu-mz, sp-mz)
- Configurable for various sizes & classes (S, W, A, B, C, D, E)

Benchmark configuration for demo:

- Benchmark name: **bt-mz**
- Number of MPI processes: **NPROCS=4**
- Benchmark class: **CLASS=W**
- What does it do?
  - Solves a discretized version of unsteady, compressible Navier-Stokes equations in three spatial dimensions
  - Performs 200 time-steps on a regular 3-dimensional grid
Connect to Mira and add Vampir to the SoftEnv system

% vi .soft
+vampir
% resoft

Copy sources to working directory

% cp /projects/Tools/vampir/tutorial/NPB3.3-MZ-MPI.tar.gz .
% tar xzvf NPB3.3-MZ-MPI.tar.gz
% cd NPB3.3-MZ-MPI

Compile the benchmark:

% make bt-mz CLASS=W NPROCS=4
cd BT-MZ; make CLASS=W NPROCS=4 VERSION=
make: Entering directory 'BT-MZ'
cd ../sys; cc -o setparams setparams.c
../sys/setparams bt-mz 4 W
mpixlf77_r -c -O3 -qsmp=omp -qextname=flush bt.f
[...] Built executable ..../bin/bt-mz_W.4
make: Leaving directory 'BT-MZ'
Copy jobscript and launch as a hybrid MPI+OpenMP application

```bash
% cd bin
% cp ../jobscript/mira/run.sh .
% less run.sh
export OMP_NUM_THREADS=4
runjob -n 4 -p 4 --block $COBALT_PARTNAME --env-all : bt-mz_W.4
% qsub -A <projid> -t 10 -n 1 --mode script run.sh
% cat <jobid>.outpout
NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark
Number of zones: 4 x 4
Iterations: 200 dt: 0.000800
Number of active processes: 4
Total number of threads: 16 (4.0 threads/process)

Time step 1
Time step 20
[...]
Time step 200
Verification Successful

BT-MZ Benchmark Completed.
Time in seconds = 2.27
```

Hint: save the benchmark output (or note the run time) to be able to refer to it later
Edit `config/make.def` to adjust build configuration

Modify specification of compiler/linker: MPIF77

```fortran
# SITE- AND/OR PLATFORM-SPECIFIC DEFINITIONS
#-----------------------------------------------
# Items in this file may need to be changed for each platform.
#-----------------------------------------------
...
#-----------------------------------------------
# The Fortran compiler used for MPI programs
#-----------------------------------------------
#MPIF77 = mpxlf77_r

# Alternative variants to perform instrumentation
...
MPIF77 = scorep mpxlf77_r

# This links MPI Fortran programs; usually the same as `${MPIF77}`
FLINK   = $(MPIF77)
...
```

Uncomment the Score-P compiler wrapper specification
Return to root directory and clean-up

% make clean

Re-build executable using Score-P compiler wrapper

% make bt-mz CLASS=W NPROCS=4
cd BT-MZ; make CLASS=W NPROCS=4 VERSION=
make: Entering directory 'BT-MZ'
cd ../sys; cc -o setparams setparams.c
../sys/setparams bt-mz 4 W
scorep mpixf77_r -c -O3 -qsmp=omp -qextname=flush bt.f
[...]
cd ../common; scorep mpixf77_r -c -O3 -qsmp=omp -qextname=flush timers.f
scorep mpixf77_r -O3 -qsmp=omp -qextname=flush -o ../bin.scorep/bt-mz_W.4
bt.o initialize.o exact_solution.o exact_rhs.o set_constants.o \
ad.o rhs.o zone_setup.o x_solve.o y_solve.o exch_qbc.o \nsolve_subs.o z_solve.o add.o error.o verify.o mpi_setup.o \n../common/print_results.o ../common/timers.o
Built executable ../bin.scorep/bt-mz_W.4
make: Leaving directory 'BT-MZ'
Change to the directory containing the new executable before running it and adjust configuration

```
% cd bin.scorep
% cp ../jobscript/mira/* .
% less run_profile.sh

export SCOREP_ENABLE_TRACING=false
export SCOREP_ENABLE_PROFLING=true
export SCOREP_TOTAL_MEMORY=100M
export SCOREP_EXPERIMENT_DIRECTORY=scorep_bt-mz_W_4x4_sum
export OMP_NUM_THREADS=4
runjob -n 4 -p 4 --block $COBALT_PARTNAME --env-all : bt-mz_W.4
% qsub -A <projid> -t 10 -n 1 --mode script run_profile.sh
% cat <jobid>.outpout

NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark
Number of zones: 4 x 4
    Time step 200
Verification Successful

BT-MZ Benchmark Completed.
Time in seconds = 12.74
```
Creates experiment directory `./scorep_bt-mz_W_4x4_sum` containing

- A record of the measurement configuration (`scorep.cfg`)
- The analysis report that was collated after measurement (`profile.cubex`)

```
% ls
... scorep_bt-mz_W_4x4_sum
% ls scorep_bt-mz_W_4x4_sum
profile.cubex  scorep.cfg
```
Report scoring as textual output

% scorep-score scorep_bt-mz_W_4x4_sum/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>235123428</td>
<td>419.92</td>
<td>100.0</td>
</tr>
<tr>
<td>USR</td>
<td>232516724</td>
<td>78.19</td>
<td>18.6</td>
</tr>
<tr>
<td>OMP</td>
<td>5973040</td>
<td>121.45</td>
<td>28.9</td>
</tr>
<tr>
<td>COM</td>
<td>314710</td>
<td>1.38</td>
<td>0.3</td>
</tr>
<tr>
<td>MPI</td>
<td>88898</td>
<td>218.90</td>
<td>52.1</td>
</tr>
</tbody>
</table>

868 MB total memory
224 MB per rank!
Score report breakdown by region

```
% scorep-score -r scorep_bt-mz_W_4x4_sum/profile.cubex

[...]
flt type  max_tbc  time  % region
ALL       235123428 419.92  100.0 ALL
USR       2516724   78.19   18.6 USR
OMP       5973040   121.45  28.9 OMP
COM       314710    1.38    0.3 COM
MPI       88898     218.90  52.1 MPI

USR        72578286 18.80   4.5 matmul_sub
USR        72578286 20.41   4.9 matvec_sub
USR        72578286 33.47   8.0 binvcrhs
USR        6747972  2.91    0.7 lhsinit
USR        6747972  1.88    0.4 binvrhs
USR        2939464  0.71    0.2 exact_solution
OMP        369840   0.14    0.0 !$omp parallel @exch_qbc...
OMP        369840   0.13    0.0 !$omp parallel @exch_qbc...
OMP        369840   0.13    0.0 !$omp parallel @exch_qbc...

[...]```

More than 223 MB just for these 6 regions
Report scoring with prospective filter listing 6 USR regions

```
% cat ../config/scorep.filt
SCOREP_REGION_NAMES_BEGIN EXCLUDE
  binvcrhs*
  matmul_sub*
  matvec_sub*
  exact_solution*
  binvrhs*
  lhs*init*
  timer_

% scorep-score -f ../config/scorep.filt scorep_bt-mz_W_4x4_sum/profile.cubex
```

Estimated aggregate size of event trace: 20482398 bytes
Estimated requirements for largest trace buffer (max_tbc): 6377242 bytes
(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.)

20 MB of memory in total, 6 MB per rank!
Advanced Measurement Configuration: Metrics

Available PAPI metrics

- Preset events: common set of events deemed relevant and useful for application performance tuning
  - Abstraction from specific hardware performance counters, mapping onto available events done by PAPI internally

```
% qsub -A <projid> -n 1 --mode c1 --proccount 1 -t 10 \ 
/soft/perftools/papi/bin/papi_avail
% cat <jobid>.outpout
```

- Native events: set of all events that are available on the CPU (platform dependent)

```
% qsub -A <projid> -n 1 --mode c1 --proccount 1 -t 10 \ 
/soft/perftools/papi/bin/papi_native_avail
% cat <jobid>.outpout
```

Note:
Due to hardware restrictions
- number of concurrently recorded events is limited
- there may be invalid combinations of concurrently recorded events
Re-run the application using the tracing mode of Score-P

```bash
% cd bin.scorep
% less run_trace.sh
  export SCOREP_ENABLE_TRACING=true
  export SCOREP_ENABLE_PROFILING=false
  export SCOREP_FILTERING_FILE=../config/scorep.filt
  export SCOREP_TOTAL_MEMORY=100M
  export SCOREP_EXPERIMENT_DIRECTORY=scorep_bt-mz_W_4x4_trace
  export SCOREP_METRIC_PAPI=PAPI_FP_OPS,PAPI_L1_DCM
  export OMP_NUM_THREADS=4
  runjob -n 4 -p 4 --block $COBALT_PARTNAME --env-all : bt-mz_W.4
% qsub -A <projid> -t 10 -n 1 --mode script run_trace.sh
% cat <jobid>.output
NAS Parallel Benchmarks (NPB3.3-MZ-MPI) - BT-MZ MPI+OpenMP Benchmark
Number of zones:  4 x  4
    Time step  200
Verification Successful

BT-MZ Benchmark Completed.
Time in seconds = 3.49
```
Download and install VampirClient for target platform

```bash
# Linux 64bit
$ scp <user>@mira.alcf.anl.gov:/projects/Tools/vampir/vampir-gui/vampir-*-x86_64.bin .
$ scp <user>@mira.alcf.anl.gov:/projects/Tools/vampir/vampir-gui/vampir-remote.license .
$ bash ./vampir-*.bin
```

Start VampirServer and follow output instructions

```bash
$ vampirserver start -a <projid> -n 16
Launching VampirServer...
Submitting PBS batch job (this might take a while)...
** Project 'tools'; job rerouted to queue 'prod-short'
VampirServer 8.2.1 (r8876)
Licensed to Mira Performance Boot Camp 2014
Running 15 analysis processes... (abort with vampirserver stop 28448)
VampirServer <28448> listens on: Q2G-I5-J01.mira.i2b:30066
Please run:
    ssh -L 30001:Q2G-I5-J01.mira.i2b:30066 <user>@mira.alcf.anl.gov
on your desktop to create ssh tunnel to VampirServer.
Start vampir on your desktop and choose 'Open Other -> Remote File'
    Description: mira, Server: localhost, Port: 30001
    Authentication: None
    Connection type: Socket
    Ignore "More Options"
```
NPB-MZ-MPI / BT Trace Analysis with Vampir
Vampir Bonus: Case Study of FDS

- Identification of program phases

[Diagram showing timeline with labeled phases: Initialization Phase and Computation Phase]
Vampir Bonus: Case Study of FDS

- Load imbalance in initialization phase

Master thread:0 is reading input files. All other processes are waiting in MPI_BARRIER.
Load imbalance in initialization phase (2)

Initialization time increases with the process index.
Vampir Bonus: Case Study of FDS

- Computation phase

12% communication and 88% computation during computation phase.
Vampir Bonus: Case Study of FDS

- Unnecessary synchronization in computation phase
Vampir Bonus: Case Study of FDS

- Inefficient cache usage in computation phase

Low Flops/s rate due to a higher L2 cache miss rate.