HPC Performance Analysis Tools at ZIH
Vampir & Score-P

9th Symposium on Software Performance 2018, Hildesheim, Germany
Outline

Introduction
— The ZIH
— High Performance Computing
— Benchmarking vs. Performance Engineering

Performance Analysis Tools
— Measurement System – Score-P
— Performance Data Visualization – Vampir
— Online Monitoring – VampirLive

Tools Community
— VI-HPS
— Parallel Tools Workshop
Introduction
Providing IT infrastructure and qualified service for TU Dresden and Saxony

Data center design innovation
— Innovative cooling
— Energy efficiency
— Reliability

Open for research collaborations

New hardware (10 Mio. €) for machine learning and Big Data
Research Topics at ZIH

— Data intensive computing and data life cycle
— Scalable software tools for HPC systems
— Parallel programming, algorithms and methods
— Performance and energy efficiency analysis for innovative computer architectures
— Distributed computing and cloud computing
— Data analytics, methods and modeling
Current HPC Resources

**Bull Cluster** (Taurus)
- ~ 44,000 Intel cores
- 256 Nvidia GPUs Tesla K80 +
- 44 Nvidia GPUs Tesla K20
- 136 TB RAM, >5 PB scratch file system

**SGI UV 2000** (Venus)
- 512 Intel cores
- 10.6 TFlop/s
- 8 TB RAM NUMA
Future HPC Resources

„Fusion von Hochleistungsrechnen und Data Analytics“ (HPC-DA)

— Lustre storage system (SATA und SSD)
— 88 GPU nodes
— 828 HPC and HTC nodes
— 648 Data-Analytics nodes
— Service nodes (10/40 Gbit Ethernet)
— Redundant InfiniBand switches
— NVMe memory and object store
— Machine Learning nodes
— Accelerators and NVRAM (later)
What is a HPC Supercomputer?

Characteristics
- High (floating point) compute power (thousands of cores)
- High main memory ration per core (>2GB/core)
- Large per-node and parallel storage systems (>PB capacity)
- Fast interconnects (>40Gbit/s bandwidth, <5us latency)

Distributed Memory
- Many independent compute nodes
- Same OS image on all nodes
- Communication done through the interconnect
- Commonly programmed via MPI (Message Passing Interface)

Shared Memory
- Single OS image
- Communication through memory
- Commonly programmed via OpenMP
HPC for Measurements

Advantages
— Resource management
— Multiple hardware of the same kind
— Multiple hardware of different kind
— Different software stacks and versions

Disadvantages
— Shared resources (I/O, network, ...)
— Timing is challenging
Benchmarking vs. Performance Engineering

**Benchmarking**
- Performs a specific workload
- Compare hardware and software
- Helps in decision process
- Statistical confidence is *built-in*

**Performance Engineering**
- About a specific application
- Gaining knowledge about application behavior
- Apply knowledge to improve applications performance
- Common statistical practice not applied widely yet
Common HPC Benchmarks

**TOP500**
- Focus on parallel compute performance (no I/O)

**Green500**
- Focus on power efficiency

**Graph500**
- Focus on graph algorithms (BFS & SSSP)

**IO-500**
- Focus on I/O performance

**Standard Performance Evaluation Corporation (SPEC)**
- Single node: CPUint, CPUfloat
- Parallel: MPI, OMP, ACCEL, “MPI ACCEL” (upcoming)
TOP500 History

- **SUM**
- **#1**
- **#500**

**Laptop**

**Taurus (incl. GPU)**
Performance Analysis Tools
Classification of measurement techniques

How are performance measurements triggered?
- Sampling
- Code instrumentation

How is performance data recorded?
- Profiling / Runtime summarization
- Tracing

How is performance data analyzed?
- Post mortem
- Online
Running program is periodically interrupted to take measurement

Statistical report of program behavior

— Not very detailed information on highly volatile metrics
— Requires long-running applications

Works with unmodified executables
Methodology: 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
Methodology: Profiling vs. Tracing

--- Statistics

Number of Invocations

- main
- bar
- foo

--- Timelines

Execution Time

Time

Methodology: Profiling vs. Tracing

- Statistics

Number of Invocations

- main
- bar
- foo

--- Timelines

Execution Time

Time
Classic Post-Mortem Measurement Workflow
Score-P & Vampir

— Suggested workflow consists of multiple repeating steps
— Especially the recompile/relink step is cumbersome for many users
— Creating useful filters can become a time-consuming task
— Sometimes even filtering may not help preventing severe overhead (C++)
Developed since 2009

Partners
— TU Dresden, FZ Jülich, TU München, University of Oregon, RWTH Aachen, TU Darmstadt

Supports
— C/C++, Fortran (Cray, GNU, PGI, Intel, LLVM, IBM)
— MPI, SHMEM
— OpenMP, PThreads
— CUDA, OpenACC, OpenCL
Data Collection with Score-P

**Instrument (Compile & Link):**

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

- CC = scorep <options> icc
- CXX = scorep <options> icpc
- F90 = scorep <options> ifc
- MPICC = scorep <options> mpicc

**Execute:**

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

**Inspect:**

```
$ ls -R
scorep-20170323_1309_7243761919249966 a.out

./scorep-20170323_1309_7243761919249966:
profile.cubex scorep.cfg
```
Data Collection with Score-P

Measurements are configured via environment variables

Example for generating and loading a trace:

```
$ 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

$ vampir scorep-20170323_1309_7243761919249966/traces.otf2
```
Performance Data Visualization with Vampir

— Visualization of dynamic runtime behavior at any level of detail along with statistics and performance metrics
— 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?

Timeline charts
— Application activities and communication along a time axis

Summary charts
— Quantitative results for the currently selected time interval
Vampir
Main Performance Charts

Timeline Charts
- Master Timeline → all threads’ activities
- Process Timeline → single thread’s activities
- Summary Timeline → all threads’ function call statistics
- Performance Radar → all threads’ performance metrics
- Counter Data Timeline → single threads’ performance metrics
- I/O Timeline → all threads’ I/O activities

Summary Charts
- Function Summary
- Message Summary
- I/O Summary
- Process Summary
- Communication Matrix View
- Call Tree
Performance Data Visualization with Vampir

The value of seeing how an application executes on the machine

- Application code computing coulomb forces
- The workload was distributed evenly across available processes
- The user expected perfect parallelized code
- However the underlying algorithm worked differently than expected
- Visualization of the application execution instantly shows a problem in the parallelization approach

Large imbalance instantly visible

More than 50% application time wasted!
I/O Features
Operations over Time

Individual I/O operation

I/O Runtime contribution
I/O Features
Data Rates over Time

I/O data rate of single thread
### I/O Features

**Summaries with Totals**

#### I/O Features

<table>
<thead>
<tr>
<th>Operation Type</th>
<th>6 MB</th>
<th>3.8 MB</th>
<th>2.5 MB</th>
<th>1.2 MB</th>
<th>0B</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sum</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>WRITE</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>READ</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Other metrics:**
- IOPS
- I/O time
- I/O size
- I/O bandwidth
I/O Features
Summaries per File

Aggregated data for specific resource
**I/O Features**

Operations per File

- Focus on specific resource
- Show all resources
Example Analysis with Vampir
Original Application

- Weather forecast code COSMO-SPECS
- Run with 100 processes
- COSMO: weather model (METEO group)
- SPECS: microphysics for accurate cloud calculation (MP and MP_UTIL group)
- Coupling of both models done in COUPLE group
Example Analysis with Vampir

Original Application

— Compared to METEO, MP and MP_UTIL are very compute intensive, however this is due to more complex calculations and no performance issue

— Problem: >32% of time spent in MPI

— MPI runtime share increases throughout the application run
Example Analysis with Vampir
Original Application

— Zoom into the first three iterations
— MP/MP_UTIL perform four sub-steps in one iteration
— Low MPI time share
— Everything is balanced and looks okay
Example Analysis with Vampir
Original Application

- Zoom into the last three iterations
- Very high MPI time share (>50%)
- Large load imbalance caused by MPI functions around Process 54 and Process 64

Process 54 causing load imbalance
Example Analysis with Vampir

Original Application

- **PAPI_FP_OPS** counter showing higher FLOPs rates on processes causing the imbalance

- Reason for imbalance: Use of a static grid for distribution of processes. Depending on the weather, expensive cloud computations (MP group) may be only necessary on some processes

High FLOPs rates due to computation of clouds in this area
Example Analysis with Vampir
Optimized Application

— Weather forecast code COSMO-
  SPECS
— Run with 100 processes
— COSMO: weather model
  (METEO group)
— SPECS: microphysics for accurate
  cloud calculation
  (MP and MP_UTIL group)
— Coupling of both models done in
  COUPLE group
— Dynamic load balancing
  (FD4 group)
Example Analysis with Vampir
Optimized Application

- Dynamic load balancing mitigates the balance problems of the original COSMO-SPECS version
- MPI time share is reduced to <13%
- MPI time share stays constant throughout the application runtime
- Runtime reduced by factor of 2.1, from initially 578s to 276s
Example Analysis with Vampir
Optimized Application

- Zoom into last three iterations
- FD4 balances MP load (precipitation processes in clouds) across all available processes
Visualization Mode
Directly on front end or local machine

$ vampir

Multi-Core Program → Score-P → Trace File

Small/Medium sized trace

Thread parallel

1.0 ms
2.0 ms
Scalable Visualization Mode
On local machine with remote VampirServer

$ vampirserver start
$ vampir

VampirServer

Vampir

Many-Core Program

Score-P

LAN/WAN

Large Trace File (stays on remote machine)

Parallel application
Vampir – Post-Mortem Architecture

- Analysis of the Sweep 3D Benchmark with Scalasca
- Application run with 294,912 processes on a BG/P
- Application runtime: ~8 min
- Measurement overhead: 5% (~9 min runtime)
- Trace data size: 790 GB
- Create 294,912 files: 86 min, 10 min with SIONlib
- Measurement unification: 43 min (old), 13 s (new)
- Analysis replay: 11 s
- Time required for I/O dominates measurement!
VampirLive – Online Architecture

— Analysis on-the-fly during application runtime
— Captured performance data is sent directly to analysis tool and stays completely in main memory
— No I/O to file system
— Performance tool runs in parallel to application
Simplified Online Workflow

VampirLive employs a hybrid approach including sampling and instrumentation, realizing the following benefits:

— No recompiling of binary
— No filtering
— No prior profiling
— Measurement (re-)configuration on-the-fly
— Big improvement on usability!
VampirLive – Online-Visualization

- New online visualization
- Timeline, profile, and counter charts
- Aggregated performance data, both from processing and visualization perspective
- Two modes (online/detail)
- Attended/unattended analysis
- Support for long running applications
- (Re-)Connect to running applications
Synchronization of Multiple Timers
Problem

Inaccurately synchronized timers results in an erroneous representation of the program trace data:

**Qualitative error**
- Violation of the logical order of distributed events

**Quantitative error**
- Skewed performance values
Continuous Time Synchronization

- Most HPC systems lack a system-wide timer
- Different environmental effects cause continuous changes to clock properties over time
- These effects lead to false measurements and wrong conclusions
- Often these errors remain undetected by the user
- One-time as well as post-mortem synchronization approaches are insufficient as changes can vary over time
Conclusions

— Long history in development of established performance analysis tools
— Holistic, powerful and detailed software performance analysis
— Everything in one picture
— Extremely customizable
— Extremely scalable
— Advanced features
— Very active in adopting new features
Performance Tools Community
Virtual Institute – High Productivity Supercomputing
http://www.vi-hps.org

— Goal: Improve the quality and accelerate the development process of complex simulation codes running on highly-parallel computer systems
— Start-up funding (2006–2011)
  by Helmholtz Association of German Research Centres

Activities
— Development and integration of HPC programming tools
  — Correctness checking & performance analysis
— Academic workshops
— Training workshops
# Productivity Tools

<table>
<thead>
<tr>
<th>Parallel Performance</th>
<th>Debugging &amp; Correctness</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM MAP</td>
<td>ARM DDT</td>
</tr>
<tr>
<td>ARM Performance Reports</td>
<td>AutomateD</td>
</tr>
<tr>
<td>Dimemas</td>
<td>Memchecker</td>
</tr>
<tr>
<td>mpiP</td>
<td>MUST</td>
</tr>
<tr>
<td>Open</td>
<td>SpeedShop</td>
</tr>
<tr>
<td>Vampir</td>
<td></td>
</tr>
</tbody>
</table>

**Single Node Performance**
- Callgrind
- MAQAO

**Instrumentation**
- OPARI2

**Measurement**
- Extrae
- PAPI
- Score-P

**Integration**
- Component-based Tool Framework
- LaunchMON
- pMPI
- JUBE

**Visualization**
- Cube

**Infrastructure**

---

For a brief overview of tools consult the VI-HPS Tools Guide:

---

Tools Guide
June 2014
Parallel Tools Workshop

Annual joint venue between HLRS Stuttgart and ZIH Dresden

Connects researchers and practitioners that develop, maintain, or use tools that aid in the development of parallel applications

Forum to networking among tool developers and users

Next Workshop:

September 2019 in Dresden