Performance Analysis Tools Overview

Markus Geimer
Jülich Supercomputing Centre
Virtual Institute – High Productivity Supercomputing

- **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
    - Next multi-day tuning workshop: January 21-25 2019, BSC, Barcelona, Spain
- **Service**
  - Support email lists
  - Application engagement

http://www.vi-hps.org
Performance engineering workflow

You!
Instrumentation

- Measurement code is inserted such that every event of interest is captured directly
  - Can be done in various ways
- Advantage:
  - Much more detailed information
- Disadvantage:
  - Processing of source-code / executable necessary
  - Large relative overheads for small functions

```c
int main()
{
    int i;
    Enter("main");
    for (i=0; i < 3; i++)
        foo(i);
    Leave("main");
    return 0;
}

void foo(int i)
{
    Enter("foo");
    if (i > 0)
        foo(i - 1);
    Leave("foo");
}
```
Critical issues

- **Accuracy**
  - Intrusion overhead
    - Measurement itself needs time and thus lowers performance
  - Perturbation
    - Measurement alters program behaviour
    - E.g., memory access pattern
  - Accuracy of timers & counters

- **Granularity**
  - How many measurements?
  - How much information / processing during each measurement?

☞ Tradeoff: Accuracy vs. Expressiveness of data
Profiling / Runtime summarization

- Recording of aggregated information
  - Total, maximum, minimum, ...
- For measurements
  - Time
  - Counts
    - Function calls
    - Bytes transferred
    - Hardware counters
- Over program and system entities
  - Functions, call sites, basic blocks, loops, ...
  - Processes, threads

Profile = summarization of events over execution interval
Tracing

- Recording detailed information about significant points (events) during execution of the program
  - Enter / leave of a region (function, loop, …)
  - Send / receive a message, …

- Save information in event record
  - Timestamp, location, event type
  - Plus event-specific information (e.g., communicator, sender / receiver, …)

- Abstract execution model on level of defined events

Event trace = Chronologically ordered sequence of event records
Process A

void foo() {
    trc_enter("foo");
    ...
    trc_send(B);
    send(B, tag, buf);
    ...
    trc_exit("foo");
}

Process B

void bar() {
    trc_enter("bar");
    ...
    recv(A, tag, buf);
    trc_recv(A);
    ...
    trc_exit("bar");
}

Event tracing

Local trace A

| ... | 58 | ENTER foo |
| ... | 62 | SEND to B |
| ... | 64 | EXIT foo |

Global trace view

| ... | 58 | A | ENTER foo |
| ... | 60 | B | ENTER bar |
| ... | 62 | A | SEND to B |
| ... | 64 | A | EXIT foo |
| ... | 68 | B | RECV from A |
| ... | 69 | B | EXIT bar |

(Virtual merge)
Tracing Pros & Cons

- **Tracing advantages**
  - Event traces preserve the *temporal* and *spatial* relationships among individual events (context)
  - Allows reconstruction of *dynamic* application behavior on any required level of abstraction
  - Most general measurement technique
    - Profile data can be reconstructed from event traces

- **Disadvantages**
  - Traces can very quickly become extremely large
  - Writing events to file at runtime may cause (additional) perturbation
Score-P instrumentation and measurement infrastructure
Score-P ecosystem

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

Application

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)
Score-P features

- Open source: 3-clause BSD license
  - Commitment to joint long-term cooperation
  - Development based on meritocratic governance model
  - Open for contributions and new partners
- Portability: supports all major HPC platforms
- Scalability: successful measurements with >1M threads
- Functionality:
  - Generation of call-path profiles and event traces (supporting highly scalable I/O)
  - Using direct instrumentation and sampling
  - Flexible measurement configuration without re-compilation
  - Recording time, visits, communication data, hardware counters
  - Access and reconfiguration also at runtime
  - Support for MPI, SHMEM, OpenMP, Pthreads, CUDA, OpenCL, OpenACC and valid combinations
- Latest release: Score-P 4.1 (October 2018)
Cube performance report explorer
Cube

- Parallel program analysis report exploration tools
  - Libraries for XML+binary report reading & writing
  - Algebra utilities for report processing
  - GUI for interactive analysis exploration
    - Requires Qt4 ≥4.6 or Qt 5

- Originally developed as part of the Scalasca toolset

- Now available as a separate component
  - Can be installed independently of Score-P and Scalasca, e.g., on laptop or desktop
  - Latest releases:
    - CubeW 4.4.1, CubeLib & CubeGUI 4.4.2 (September 2018)
      - Source tarball & Windows/MacOS binaries
Analysis presentation and exploration

- Representation of values (severity matrix) on three hierarchical axes
  - Performance property (metric)
  - Call path (program location)
  - System location (process/thread)

- Three coupled tree browsers

- Cube displays severities
  - As value: for precise comparison
  - As color: for easy identification of hotspots
  - Inclusive value when closed & exclusive value when expanded
  - Customizable via display modes
Analysis presentation

What kind of performance metric?

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

How is it distributed across the processes/threads?
Scalasca Trace Tools
Automatic trace analysis

- Idea
  - Automatic search for patterns of inefficient behavior
  - Classification of behavior & quantification of significance
  - Identification of delays as root causes of inefficiencies

- Guaranteed to cover the entire event trace
- Quicker than manual/visual trace analysis
- Parallel replay analysis exploits available memory & processors to deliver scalability
Scalasca Trace Tools objectives

- Development of a **scalable trace-based** performance analysis toolset for the most popular parallel programming paradigms
  - Current focus: MPI, OpenMP, and POSIX threads

- Specifically targeting large-scale parallel applications
  - Such as those running with one million or more processes/threads

- Latest release: Scalasca v2.4 (May 2018)
Scalasca Trace Tools features

- Open source: 3-clause BSD license
- Portability: supports all major HPC platforms
- Scalability: successful analyses with >1M threads

- Uses Score-P instrumenter & measurement libraries
  - Scalasca v2 core package focuses on trace-based analyses
  - Provides convenience commands for measurement, analysis, and postprocessing
  - Supports common data formats
    - Reads event traces in OTF2 format
    - Writes analysis reports in CUBE4 format
Vampir Tool Suite

(Slides courtesy of Bert Wesarg et al., ZIH, TU Dresden)
Event Trace Visualization with Vampir

- Alternative and supplement to automatic analysis
- 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
Visualization Modes (1)
Directly on front end or local machine

% vampir
Visualization Modes (2)
On local machine with remote VampirServer

```
% vampirserver start
```

```
% vampir
```

**VampirServer**

**Score-P**

**Many-Core Program**

**Trace File**

**LAN/WAN**

Large Trace File (stays on remote machine)

Parallel application
Putting it all together

Optimized measurement configuration

Measurement library

Instr. target application

Event traces

Parallel trace analysis

Trace analysis report

Summary report

Report postprocessing

Score-P

Instrumented executable

Instruenter compiler/linker

Source modules

Instrumented executable

Vampir

Scalasca

Cube

Virtual Institute - High Productivity Supercomputing

Advanced MPI and OpenMP Course (Jülich, Germany, November 26-28, 2018)