COMPUTING THE POP METRICS WITH SCORE-P, SCALASCA, AND CUBE

AIXCELERATE 2022  |  DEC. 05, 2022  |  MICHAEL KNOBLOCH  |  M.KNOBLOCH@FZ-JUELICH.DE
SCORE-P

- Infrastructure for instrumentation and performance measurements
- Instrumented application can be used to produce several results:
  - Call-path profiling: CUBE4 data format used for data exchange
  - Event-based tracing: OTF2 data format used for data exchange

- Supported parallel paradigms:
  - Multi-process: MPI, SHMEM
  - Thread-parallel: OpenMP, Pthreads
  - Accelerator-based: CUDA, OpenCL, OpenACC, ROCm, Kokkos

- Open Source; portable and scalable to all major HPC systems
- Initial project funded by BMBF
- Further developed in multiple 3rd-party funded projects
SCORE-P OVERVIEW

- **Vampir**
- **Scalasca**
- **CUBE**
- **TAU**
- **Extra-P**
- **TAUdb**

**Score-P measurement infrastructure**

- Event traces (OTF2)
- Call-path profiles (CUBE4, TAU)
- Hardware counter (PAPI, rusage, PERF, plugins)

**Instrumentation wrapper**

- Process-level parallelism (MPI, SHMEM)
- Thread-level parallelism (OpenMP, Pthreads)
- Accelerator-based parallelism (CUDA, OpenACC, OpenCL, ROCm, Kokkos)
- I/O Activity Recording (Posix I/O, MPI-IO)
- Source code instrumentation (Compiler, PDT, User)
- Sampling interrupts (PAPI, PERF)
SCALASCA

• **Scalable Analysis of Large Scale Applications**

• **Approach**
  • **Instrument** C, C++, and Fortran parallel applications (with Score-P)
  • Option 1: **scalable call-path profiling**
  • Option 2: **scalable event trace analysis**
    • **Collect** event traces
    • **Process trace in parallel**
      • Wait-state analysis
      • Delay and root-cause analysis
      • Critical path analysis
    • **Categorize and rank** results

http://www.scalasca.org/
AUTOMATIC TRACE ANALYSIS

- Idea
  - Automatic search for patterns of inefficient behaviour
  - Classification of behaviour & 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
• Waiting time caused by a blocking receive operation posted earlier than the corresponding send
• Applies to blocking as well as non-blocking communication
EXAMPLE MPI WAIT STATES

(a) Late Sender

(b) Late Receiver

(c) Late Sender / Wrong Order

(d) Wait at N x N

- ENTER
- EXIT
- SEND
- RECV
- COLLEXIT
EXAMPLE: ROOT CAUSE ANALYSIS

• Root-cause analysis
  • Wait states typically caused by load or communication imbalances earlier in the program
  • Waiting time can also propagate (e.g., indirect waiting time)
  • Enhanced performance analysis to find the root cause of wait states

• Approach
  • Distinguish between direct and indirect waiting time
  • Identify call path/process combinations delaying other processes and causing first order waiting time
  • Identify original delay

---

Diagram showing an example of wait states and their causes.

• Direct wait
• Indirect wait
EXAMPLE: CRITICAL PATH

- Shows call paths and processes/threads that are responsible for the program’s wall-clock runtime
- Identifies good optimization candidates and parallelization bottlenecks
PERFORMANCE ANALYSIS WORKFLOW

1. Instrumentation

Application source files

Instrumenter
compiler / linker

Instrumented executable

2. Measurement

Instrumented application

Measurement library

Hardware counter library

Optimized measurement configuration (e.g. filter)

Summary profile

Trace analysis profile

Parallel trace analysis

Local trace files

Profile

Profile manipulation

Profile browser

3. Analysis
PERFORMANCE ANALYSIS WORKFLOW

1. Instrumentation
   - Application source files
   - Score-P instrumenter
   - Score-P library
   - PAPI library
   - Instrumented application
   - Instrumented executable

2. Measurement
   - Optimized measurement configuration (e.g. filter)
   - Summary CUBE4 profile
   - Score-P library
   - PAPI library
   - Instrumented application
   - Local OTF2 trace files

3. Analysis
   - CUBE tools
   - CUBE4 Profile
   - CUBE browser
   - Trace analysis CUBE4 profile
   - Scalasca trace analysis
PERFORMANCE ANALYSIS WORKFLOW

1. Instrumentation

- Application source files
  - Score-P instrumenter
    - Instrumented executable

2. Measurement

- Instrumented application
  - Score-P library
    - PAPI library
  - Optimized measurement configuration (e.g. filter)
    - Scan ...

- Summary CUBE4 profile
  - Trace analysis
    - Scalasca trace analysis
      - Local OTF2 trace files
  - CUBE4 profile

3. Analysis

- CUBE tools
  - CUBE4 Profile
    - CUBE browser
  - Scan –q –t ...
  - Square ...
• Measurement of simple Jacobi solver
  • Solves Poisson equation on rectangular grid assuming
    • Uniform discretization in each direction
    • Dirichlet boundary conditions

• Available in multiple variants (shipped with Score-P)
  • C, C++ or Fortran source code
  • MPI, OpenMP, or hybrid (MPI+OpenMP)
DEMO: BASE RUN OF APPLICATION

Notes
- Compile application
- Execute application with 2 threads on 2 processes
- Write down execution time for later comparison
Notes
• Make sure tools are in $PATH
• Instrument: prepend scorep
• Measure profile: prepend scan
• Compare execution time to check overhead
Notes

• Optimize measurement config: scoring with square -s
• Also does post-processing

• Potential need for filtering → see user guides
• Set SCOREP_TOTAL_MEMORY
Notes
• Measure trace: prepend scan
  • –q: profile off
  • –t: trace on

• After trace measurement, Scalasca trace analyzer runs automatically
FURTHER USEFUL INFORMATION

Extended and more detailed example based on NAS Parallel Benchmark (NPB) BT

- Scalasca documentation
  - A full workflow example
- Score-P documentation
  - Performance Analysis Workflow Using Score-P
- Slides from 33rd VI-HPS Tuning Workshop
  - Score-P instrumentation & measurement toolset
  - Score-P analysis scoring & measurement filtering
  - Score-P specialized instrumentation and measurement (Advanced)
- Scalasca automated trace analysis
PERFORMANCE ANALYSIS WORKFLOW

1. Instrumentation

Application source files → Score-P instrumenter → Instrumented executable

Score-P library → PAPI library → Instrumented application

2. Measurement

Optimized measurement configuration (e.g. filter) → Score-P library

2.1. Measurement

scan → CUBE4 profile

2.2. Measurement

scan → Trace analysis

3. Analysis

CUBE4 Profile → CUBE tools → CUBE browser

square → CUBE4 profile

CUBE browser → Local OTF2 trace files

CUBE browser → Scalasca trace analysis
SCORE-P / SCALASCA MEASUREMENT DATA

• All measured data and meta information stored in experiment directory
  • `scorep_<executable>_<size>_[sum|trace]`
  • Example: `scorep_tea_leaf_2p8x12_trace`

• Contents
  • `profile.cubex` Score-P profile measurement
  • `summary.cubex` Post-processed profile measurement
  • `scout.cubex` Scalasca trace analyzer result
  • `trace.cubex` Post-processed trace analyzer result

• Post-processed Cube files include
  • Additional derived metrics
  • Enhanced metrics hierarchy
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, e.g., on laptop or desktop
- Latest release: Cube v4.7
CUBE DATA

• Measured values organized in 3D block (“Cube”) along **three hierarchical axes**
  • Metrics (general → specific)
  • Call path (program location)
  • System location (machine → node → process → thread)

• Displayed as **three coupled tree browsers**
  • Each node displays metric value
    • As color: for easy identification of bottlenecks
    • As number: for precise comparison
  • Displayed metric value depends on state
    • Collapsed (inclusive value)
    • Expanded (exclusive value)
How is it distributed across the processes/threads?  
What kind of performance metric?  
Where is it in the source code? In what context?  
How is it distributed across the processes/threads?
CUBE BASIC COMMANDS

- Expand / Collapse tree nodes
  - Chooses level of granularity
  - Use context menu to expand / collapse whole (sub)trees

- Select tree nodes
  - Shows distribution of metric value in tree to the right
  - Use Ctrl+Click to select multiple nodes
STARTING CUBE GUI

Local scenario:

% square <experiment_directory>

- Performs post-processing if necessary
- Executes Cube GUI

Remote scenario:

% square -s <experiment_directory>

- Performs post-processing if necessary
- Copy desired Cube file to local system
- (or remotely mount file system)

% cube <cube_file>
SCALASCA CASE STUDY – TEA LEAF
CASE STUDY: TEALEAF

- HPC mini-app developed by the UK Mini-App Consortium
  - Solves the linear 2D heat conduction equation on a spatially decomposed regular grid using a 5 point stencil with implicit solvers
  - Part of the Mantevo 3.0 suite
  - Available on GitHub: https://uk-mac.github.io/TeaLeaf/

- Measurements of TeaLeaf reference v1.0 taken on Jureca cluster @ JSC
  - Using Intel 19.0.3 compilers, Intel MPI 2019.3, Score-P 5.0, and Scalasca 2.5
  - Run configuration
    - 8 MPI ranks with 12 OpenMP threads each
    - Distributed across 4 compute nodes (2 ranks per node)
    - Test problem “5”: 4000 \times 4000 cells, CG solver
SCALASCA ANALYSIS REPORT
EXPLORATION (OPENING VIEW)

Additional top-level metrics produced by the trace analysis...
SCALASCA WAIT-STATE METRICS

...plus additional wait-state metrics as part of the “Time” hierarchy
While MPI communication time and wait states are small (~0.6% of the total execution time)…
...they directly cause a significant amount of the OpenMP thread idleness.
The “Wait at NxN” collective wait states are mostly caused by the first 2 OpenMP do loops of the solver (on ranks 5 & 1, resp.)...
while the MPI point-to-point wait states are caused by the 3rd solver do loop (on rank 1) and two loops in the halo exchange
Various OpenMP `do` loops (incl. the solver loops) also cause OpenMP thread idleness on other ranks via propagation.
The Critical Path also highlights the three solver loops...
...with imbalance (time on critical path above average) mostly in the first two loops and MPI communication
Computation time of 1st...
...and 2\textsuperscript{nd} do loop mostly balanced within each rank, but vary considerably across ranks...
...while the 3rd do loop also shows imbalance within each rank
TEALEAF ANALYSIS SUMMARY

- The first two OpenMP do loops of the solver are well balanced within a rank, but are imbalanced across ranks
  ➔ Requires a global load balancing strategy
- The third OpenMP do loop, however, is imbalanced within ranks,
  - causing direct “Wait at OpenMP Barrier” wait states,
  - which cause indirect MPI point-to-point wait states,
  - which in turn cause OpenMP thread idleness
  ➔ Low-hanging fruit

- Adding a SCHEDULE(guided) clause reduced
  - the MPI point-to-point wait states by ~66%
  - the MPI collective wait states by ~50%
  - the OpenMP “Wait at Barrier” wait states by ~55%
  - the OpenMP thread idleness by ~11%
  ➔ Overall runtime (wall-clock) reduction by ~5%
HOW-TO GET POP METRICS
RECAP: POP METRICS

- **Original (POP1) Metrics**
  - [Article](https://pop-coe.eu/further-information/learning-material) explaining the POP Standard Metrics for Parallel Performance Analysis
  - [Presentation](https://pop-coe.eu/further-information/learning-material) summarizing the POP Standard Metrics for Parallel Performance Analysis

- **New (POP2) Hybrid Metrics**
  - [Introduction](https://pop-coe.eu/further-information/learning-material) explaining the POP2 Standard Metrics for Performance Analysis of Hybrid Parallel Applications
  - [Cheat sheet](https://pop-coe.eu/further-information/learning-material) for Additive Hybrid Metrics
  - [Cheat sheet](https://pop-coe.eu/further-information/learning-material) for Multiplicative Hybrid Metrics
  - [In-depth explanation](https://pop-coe.eu/further-information/learning-material) of the Additive Hybrid Metrics
  - [Webinar](https://pop-coe.eu/further-information/learning-material) Identifying Performance Bottlenecks in Hybrid MPI + OpenMP Software
1. Instrument application and setup measurement parameters (e.g. filtering)
   - `scorep <comp+link+cmds>`
   - `scan <exec+cmd> ...`

2. For parallel efficiency: perform trace measurement and analysis

3. For computational scaling: perform profile measurement with suitable HW counters
   - `scan -P pop <exec+cmd>`

4. Merge profile and trace measurement

5. Post-process measurement

6. Analyze POP metrics with Cube Advisor
   - `square <measurement+archive>`

Requires
- Scalasca $\geq$ V2.6
- Cube $\geq$ V4.6
MEASUREMENT DEMO

• Measurement of simple Jacobi solver
  • Solves Poisson equation on rectangular grid assuming
    • Uniform discretization in each direction
    • Dirichlet boundary conditions

• Available in multiple variants (Shipped with Score-P)
  • C, C++ or Fortran source code
  • MPI, OpenMP, or hybrid (MPI+OpenMP)
DEMO: POP PRESET MEASUREMENT

Notes
• -P pop selects POP metrics measurement
• Automatically executes necessary trace and profile measurements
Notes
• Square recognizes POP metrics measurement
• Merges the two measurements before post-processing
• TeaLeaf Reference V1.0

• HPC mini-app developed by the UK Mini-App Consortium
  • Solves the linear 2D heat conduction equation on a spatially decomposed regular grid using a 5 point stencil with implicit solvers
  • https://github.com/UK-MAC/TeaLeaf_ref/archive/v1.0.tar.gz

• Measurements performed on Jusuf cluster @ JSC
  • Run configuration
    • 32 MPI ranks with 8 OpenMP threads each
    • Distributed across 2 compute nodes (16 ranks per node)
    • Test problem “5”: 4000 x 4000 cells, CG solver
POP METRICS + CUBE ADVISOR

Select POP metric set

Select Region of Interest

Find Advisor on General Tab
# WHERE TO GET HELP

<table>
<thead>
<tr>
<th>Tool</th>
<th>Support Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score-P</td>
<td><a href="mailto:support@score-p.org">support@score-p.org</a></td>
</tr>
<tr>
<td>Scalasca, Cube</td>
<td><a href="mailto:scalasca@fz-juelich.de">scalasca@fz-juelich.de</a></td>
</tr>
</tbody>
</table>
QUESTIONS