Programmatic Analysis of Performance Data: APIs, and Uses

Scalable Tools Workshop 2023

Group Lead: John Mellor-Crummey

https://bit.ly/stw-pa

Topics not discussed

- Visualization Strategies
- Data Collection

Participants

- John Mellor-Crummey (Rice University)
- Marty Itzkowitz (Rice University)
- Wileam Phan (Rice University)
- Dragana Grbic (Rice University)
- Laksono Adhianto (Rice University)
- Dave Montoya (Trenza)
- Mahesh Rajan (Trenza)
- Abhinav Bhatele (University of Maryland)
- Matt Legendre (LLNL)
- Kathleen Shoga (LLNL)
- Stephanie Brink (LLNL)
- David Boehme (LLNL)
- Kate Isaacs (University of Utah)
- Shadmaan Hye (University of Utah)
- Rahat Zaman (University of Utah)
- Connor Scully-Allison (University of Utah)
- Martin Schulz (TU-Munich)
- Bert Wesarg (TU-Dresden)
- Jesus Labarta (BSC)
- Thomas Gruber (NHR@FAU)
- Jan Laukemann (NHR@FAU)
- Qidong Zhao (NCSU)
- Keren Zhou (OpenAI)

Programmatic Analysis of Performance Data

Why?

- Automated performance analysis and diagnosis
- Comparative analysis of multiple executions
- Analysis of very large scale executions
  - Zooming and scrolling with a GUI isn't enough
- Automatic feedback and tuning of applications and systems
  - E.g. rank reordering, feedback to compiler
- Cheaper than building custom GUIs
- Ad-hoc investigation of unexpected phenomena
- Evaluate conjectures about program performance
- Tool validation
- Provide input to ChatGPT when looking for performance guidance
- Fusion of information sources, e.g. Caliper data, machine organization

Programmatic Analysis of Performance Data

What?

- Data model for performance data
    - E.g. event model for traces
    - PERIXML: http://cscads.rice.edu/2008-07-snowbird-perixml.pdf
- Data representations for performance data
    - Goal: usable at scale
- Operations on performance data
    - Ensemble analysis within an execution (e.g. across ranks, GPUs)
    - Ensemble analysis across different executions
    - Summarization of traces or sections or subsets (e.g. convert to profiles)
    - Binning of prominent performance features
        - E.g. communication with particular characteristics
    - Analysis of time-variant behavior
    - Analysis at multiple levels of abstraction
        - Hourglass model (Tennessee) doi:10.1145/3274770
- Presentation/Visualization

Programmatic Analysis of Performance Data

How?

- Parsing of profiles
- Parsing of trace data
- Merging profiles from different executions
- Analysis of time-variant behavior
    - Strategies to identify timesteps
        - Application instrumentation
        - Fourier analysis
        - Auto-correlation

What do we want from traces?

Trace selection

- Subset of traces
- Time window in trace
- Match patterns in trace lines
  - May include performance or semantic attributes of items in the trace
    - e.g. long mallocs
- Sequenced-before, Happens-before and Happens-after
  - Multiple resolutions of analysis
- Query language for traces in Perfetto

Multiple different query levels

- User-level analysis queries
- Hatchet or Pipit query layer
- Data extraction/access query layer
- ScrubJay
    - Bring in data from multiple sources, scale time

Hatchet has three levels of query languages

- String-based dialect (easiest to read)
- Object-based dialect
- Base syntax (more specific, low-level)
- I. Lumsden eScience’22: https://www.osti.gov/servlets/purl/1893573

Base Syntax

```
query = QueryMatcher().match(
     “.”,
     lambda row: re.match(
         “MPI_.*”,
         row[“name”] )
     is not None
     and row[“PAPI_L2_TCM”] > 5
  ).rel(“*”)
```

+ Support any query
- Require Python libs knowledge
- Work with Python only

Object-based Dialect

```
query = [
    (“.”, {
     “name”: “MPI_.*”,
     “PAPI_L2_TCM”: “> 5”
    }),
    “*”
]
```

+ Use built-in Python objects
- Support limited queries
- Work with Python only

String-based Dialect

```
query = """
MATCH (".", p)->("*")
WHERE p."name" =~ "MPI_.*"
AND p."PAPI_L2_TCM" > 5
"""
```

+ Work with any language
- Support limited queries

Issues - 1

- Metadata, metadata, metadata
  - Describe each GPU in the system
  - What nodes am I running on
  - What compiler is used for the code ranges in the application
  - See Adiak (https://github.com/llnl/adiak)
    - Machine name, attributes in environment variables, MPI attributes (size), time of launch, user name
  - Perhaps things like system load
  - MachineState (https://github.com/RRZE-HPC/MachineState)
    - Machine names, env variables MPI settings, temperature, system settings, OS settings, frequencies, power limits, co-processor infos, system topology, user information, loaded modules, linked libs to application, load, IB network, file systems, …
    - Allows comparison of two machine states / meta data sets
  - Application-level info
    - Static: what physics packages enabled, problem size
    - Performance measure for a run: FOM
  - Dynamic machine information, e.g., changing resources, cache sizes, topology mapping, etc. - captured with sys-sage (https://github.com/caps-tum/sys-sage)

Issues - 2

- Data representation
  - Scalability
    - Sparse
    - Out-of-core data
    - Lazy reading of requested slices on demand
      - Filtering in multiple dimensions
        - Selective fetch
        - Filtering of trace data
    - Data server that performs read operations
- Query language design
  - Commonality among different tools, Pipit, Hatchet, Thicket, lower-level layers
- Memory management for scalability

Types of tools

- Online
  - Chimbuko - trace collection as needed for coarse-grain epochs
  - Vampir online collection and analysis
  - System tuning
    - Collection of window of metric trace data
- Post-mortem
  - Hatchet
  - Thicket

Action Items

- How HPCToolkit data source might provide data for Hatchet/Thicket/Pipit
- Perhaps have an aggregate data only mode for hpcprof
- Query language design
- Common data extraction API for different kinds of data
    - Hpctoolkit
    - Trenza survey data
    - Nsight systems
    - Rocprof, omniperf, omnitrace, uProf
    - VTune