What's new in Caliper?

2026 Scalable Tools Workshop

July 27, 2026

David Boehme
LLNL

Prepared by LLNL under Contract DE-AC52-07NA27344.

Caliper: An Instrumentation and Profiling Library

- Instrumentation API for marking source-code regions

```
#include <caliper/cali.h>

void main() {
  CALI_MARK_BEGIN(”init”);
  do_init();
  CALI_MARK_END(”init”);
}
```

- Runtime-configurable performance measurements

```
$ CALI_CONFIG=runtime-report basic_example
Path        Time (E) Time (I) Time % (E) Time % (I)
main        0.000011 0.000195   5.817424 100.000000
  setup     0.000178 0.000178  91.070359  91.070359
  main loop 0.000004 0.000006   2.017642   3.112217
    foo     0.000002 0.000002   0.784838   0.784838
    bar     0.000001 0.000001   0.309737   0.309737
```

Caliper Goals and Advantages

- Robust instrumentation and profiling for large HPC codes
  - Provide **human-readable labels** for code regions
- Support run-to-run **performance comparisons**
  - Manual region annotations **isolate logical program structure** from modifications of interest
- Wide range of performance analysis functionality
  - Measurement modes: profiling, tracing, sampling, …
  - Programming models: MPI, OpenMP, CUDA, ROCm, …
  - Features: timers, HW counters, allocation tracking, …

New: Region levels in instrumentation API

"phase" annotations mark higher-level regions

```
#include <caliper/cali.h>

void main() {
  CALI_MARK_PHASE_BEGIN("hydro");

  CALI_MARK_BEGIN("setup");
  hydro_setup();
  CALI_MARK_END("setup");

  CALI_MARK_BEGIN("solve");
  hydro_solve();
  CALI_MARK_END("solve");

  CALI_MARK_PHASE_END("hydro");
}
```

- Region levels support instrumentation and measurements at different granularity

```
$ CALI_CONFIG=runtime-report,level=phase basic_example
```

New: Python instrumentation API

Function decorators

```python
from pycaliper.high_level import annotate_function
from pycaliper.instrumentation import begin_region, end_region

@annotate_function()
def foo(i: int) -> float:
    print("Hello")
    return 0.5 * i

def main():
    begin_region("main")

    t = 1
    for i in range(args.iterations):
        t *= foo(i)

    end_region("main")
```

Begin/End region annotations

ROCm support

- ROCm analysis support via rocprofiler-sdk
  - Can record HIP API, GPU activities, rccl, roctx, HW counters, allocations

HIP API profiling

Per-instance GPU activity statistics

```
$ CALI_CONFIG=runtime-report,profile.hip,rocm.activity.stats ./square.out.rocm720
Path              Time (E) Time (I) Time % (E) Time % (I) Kernel           GPU invoc. Nsec/invoc (min) Nsec/invoc (avg) Nsec/invoc (max)
main              0.003346 0.665528   0.502749 100.000000
  copy_h~~device  0.000131 0.573692   0.019639  86.200996
    hipMemcpy     0.573561 0.573561  86.181357  86.181357                        1            78080            78080            78080
  hipMalloc       0.083623 0.083623  12.564887  12.564887
  mainloop        0.000113 0.001725   0.016951   0.259228
    square        0.000056 0.001612   0.008396   0.242277
      hipL~~rnel
        |-                                                 void ve~~ne .kd]          8             4160             5360            11920
        |-          0.001557 0.001557   0.233880   0.233880
  copy_d~~e2host  0.000014 0.003142   0.002045   0.472141
    hipMemcpy     0.003129 0.003129   0.470096   0.470096                        1            72960            72960            72960
```

GPU metrics are linked to launch region

New: ROCm hardware counters

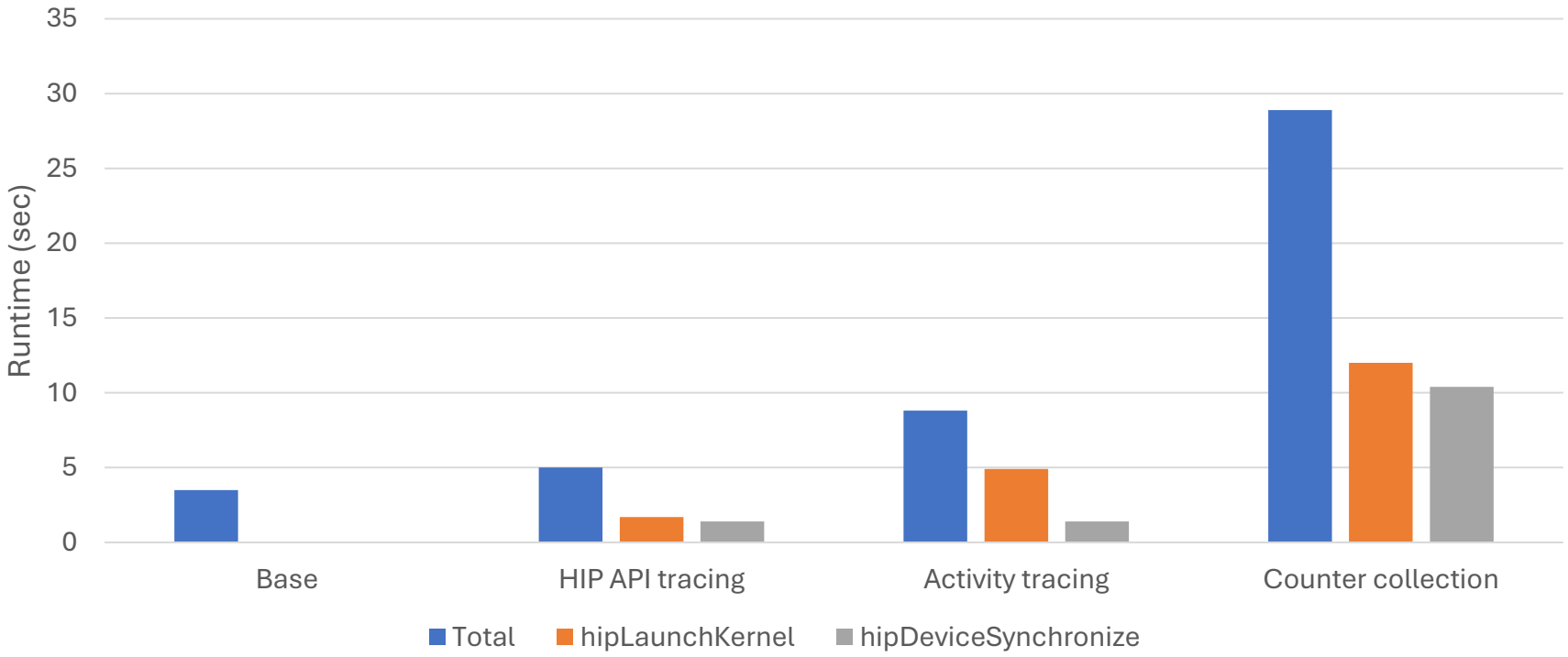
- Collect any rocprofv3 ROCm performance counters
  - Dispatch mode (counters are per kernel launch), sampling mode planned

```
$ CALI_CONFIG=rocm-activity-report,rocm.counters=SQ_WAVES ./square.out.rocm720
Path                  Host Time GPU Time GPU %    SQ_WAVES
main                   0.396782 0.000201 0.050647
  copy_host2device     0.390005 0.000077 0.019631
    hipMemcpy          0.389954 0.000077 0.019633
  hipMalloc            0.000983
  mainloop             0.001942 0.000050 2.564520
    square             0.001844 0.000050 2.700310
      hipLaunchKernel  0.001800 0.000050 2.766939 16384.000000
  copy_device2host     0.001677 0.000075 4.449370
    hipMemcpy          0.001668 0.000075 4.472470
```

ROCm's internal helper threads can cause issues

- ROCm can launch kernels from internal helper threads
  - Helper threads lack region context information: cannot link kernel back to instrumented region
  - Setting **HIP_LAUNCH_BLOCKING=1** helps

ROCm measurement overheads



Tuolumne (MI300A), 592’000 kernel launches, ROCm 7.2.0, Caliper v2.14

New: Intel Topdown analysis support

- Support for Intel topdown counters via either PAPI or perf
  - PAPI topdown component has issues in v7.2 – fixed in github master

```
$ CALI_CONFIG=runtime-report,papi.topdown.toplevel ./cali-mbench
Path           Time (E) Time (I) Time % (E) Time % (I) Retiring  Bad spec FE bound  BE bound
main                    3.923604            100.000000
  mainloop     0.000127 3.923604   0.003241 100.000000  8.382353 7.990196 35.343138 48.823530
    triad      0.407367 0.407367  10.382477  10.382477 18.823530 0.784314 11.078432 69.411767
    matmul_f64 3.516110 3.516110  89.614282  89.614282 81.568629 5.196078 12.352942  1.176471
```

Topdown metrics via PAPI

```
$ CALI_CONFIG=runtime-report,perf.topdown.toplevel ./cali-mbench
Path           Time (E) Time (I) Time % (E) Time % (I) Retiring  Bad spec FE bound  BE bound
main                    3.928778            100.000000
  mainloop     0.010280 3.928778   0.261648 100.000000
    triad      0.427450 0.427450  10.879973  10.879973 17.210127 1.215839 13.585760 68.161959
    matmul_f64 3.491049 3.491049  88.858379  88.858379 81.568627 5.196025 12.156863  1.568627
```

Topdown metrics via perf

New: Memory allocation statistics

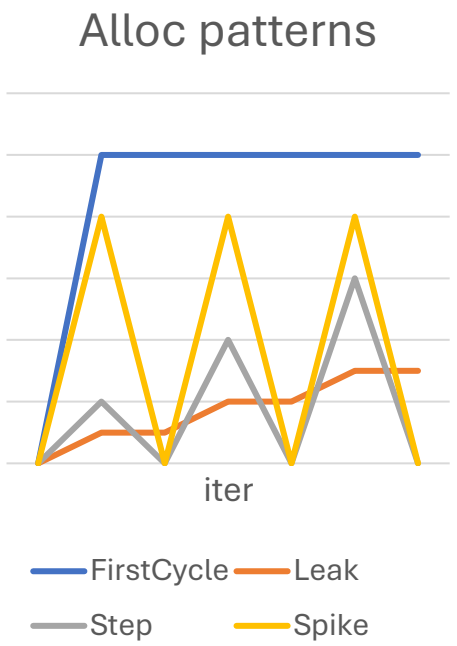
- *alloc.stats* tracks malloc/free and computes statistics by region
  - How much memory was allocated by each code region?

```
$ ./allpax -P runtime-report,alloc.stats
Path           Time (E) Time (I) Time % (E) Time % (I) Mem HWM Alloc tMax Alloc count Avg Bytes/alloc Max Bytes/alloc
main           0.000057 0.005162   1.106581 100.000000 2601032         47           2              23              31
  Setup        0.000042 0.000042   0.813350   0.813350    1304       1304          14             110             800
  cycles       0.001231 0.005063  23.847409  98.080069 2601032
    FirstCycle 0.000155 0.000155   3.002144   3.002144 2585032    1000000           1         1000000         1000000
    Split      0.000466 0.001363   9.032234  26.397793 2601032
      Alloc    0.000657 0.000657  12.723698  12.723698 2601032      16000         100           16000           16000
      Dealloc  0.000240 0.000240   4.641861   4.641861 2601032
    Leak       0.000206 0.000206   3.982952   3.982952 2601032    1600000         100           16000           16000
    Step       0.001099 0.001099  21.287785  21.287785 3385032     784000         100          392000          784000
    Spike      0.001010 0.001010  19.561985  19.561985 3601032    1000000         100         1000000         1000000
```

Region high-water mark

Region tally (how much was allocated by this region?)

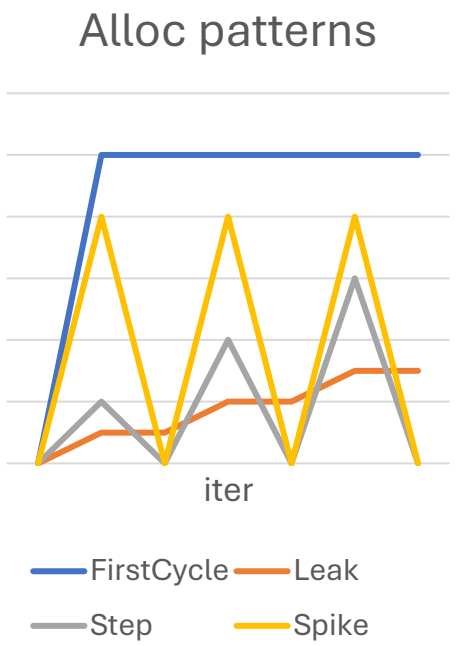
Per-allocation statistics



New: Memory page stats (Linux)

- *mem.pages* reads VmSize, VmRSS, Data size from /proc/self/statm

```
$ ./allpax -P runtime-report,mem.pages,mem.highwatermark
Path           Time (E) Time (I) Time % (E) Time % (I) VmSize VmRSS Data Mem HWM MB
main           0.000074 0.005379   1.369260 100.000000   6310  3306 2032   2.601032
  Setup        0.000049 0.000049   0.916781   0.916781   5401  3072 1123   0.001304
  cycles       0.001938 0.005256  36.025999  97.713959   6310  3306 2032   2.601032
    FirstCycle 0.000294 0.000294   5.468227   5.468227   6274  3304 1996   2.585032
    Split      0.000826 0.001575  15.363518  29.277587   6274  3304 1996   2.601032
      Alloc    0.000418 0.000418   7.780000   7.780000   6274  3304 1996   2.601032
      Dealloc  0.000330 0.000330   6.134069   6.134069   6274  3304 1996   2.601032
    Leak       0.000371 0.000371   6.895699   6.895699   6274  3304 1996   2.601032
    Step       0.000553 0.000553  10.287679  10.287679   6274  3305 1996   3.385032
    Spike      0.000525 0.000525   9.758768   9.758768   6310  3306 2032   3.601032
```



New: MPI communication pattern statistics

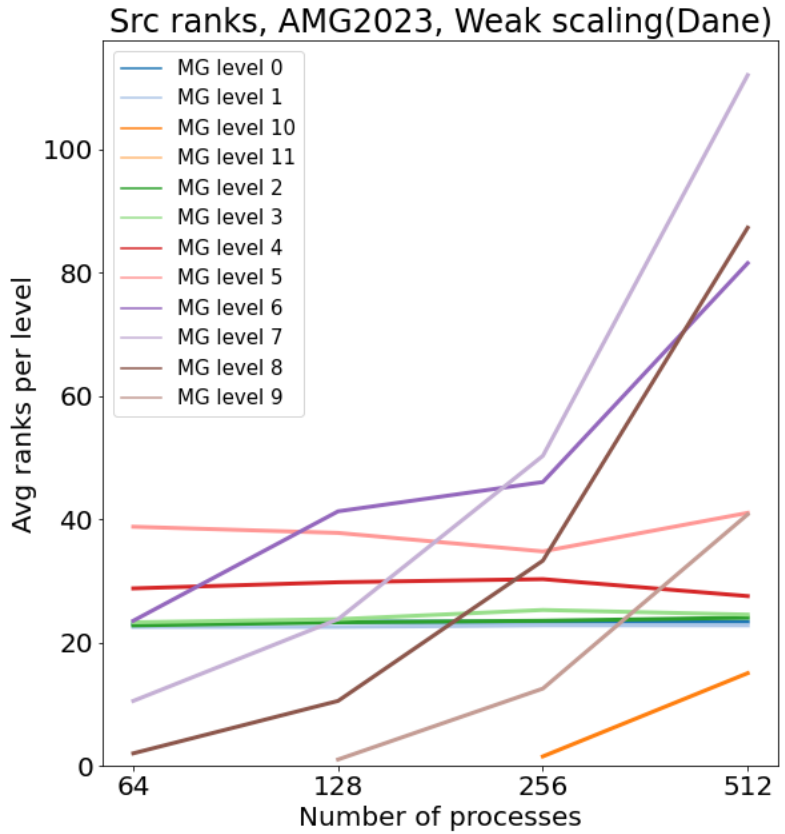
- Special region markers distinguish **logical** communication phases
- *comm.stats* computes per-pattern communication statistics

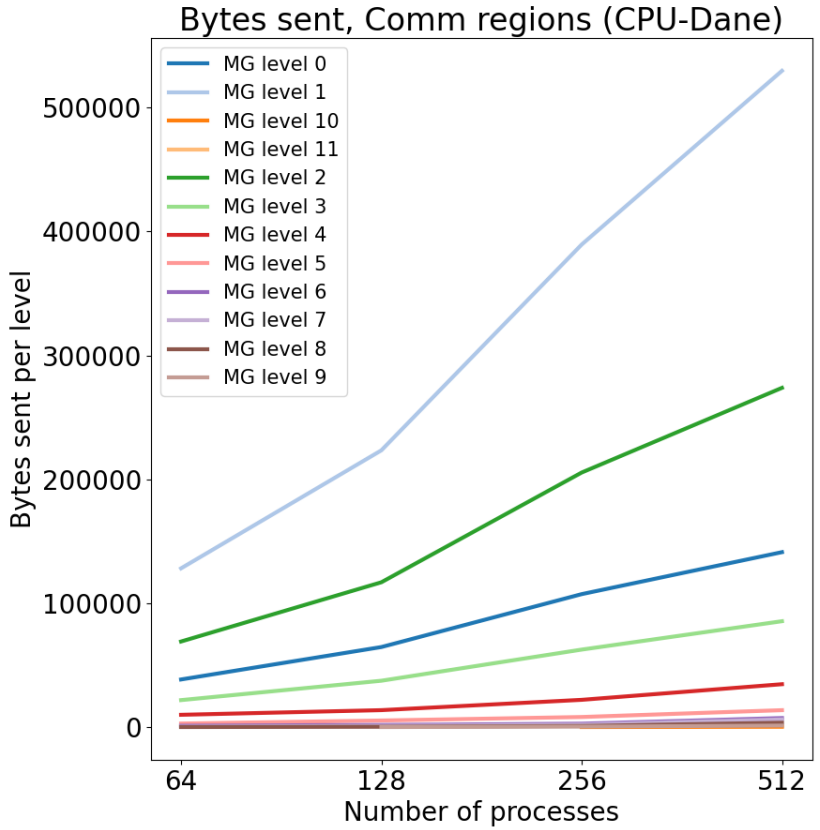
```
#include <caliper/cali.h>

void halo_exchange(...) {
  CALI_MARK_COMM_REGION_BEGIN("halo_exchange");
  MPI_Irecv(...);
  MPI_Isend(...);
  MPI_Waitall(...);
  CALI_MARK_COMM_REGION_END("halo_exchange");
}
```

Metrics	Description
#sends, #recvs	Messages sent/received
dest/source ranks	Distinct destination/source ranks
bytes sent/received	Total bytes sent/received
Irecv gap	Time between Irecv/wait

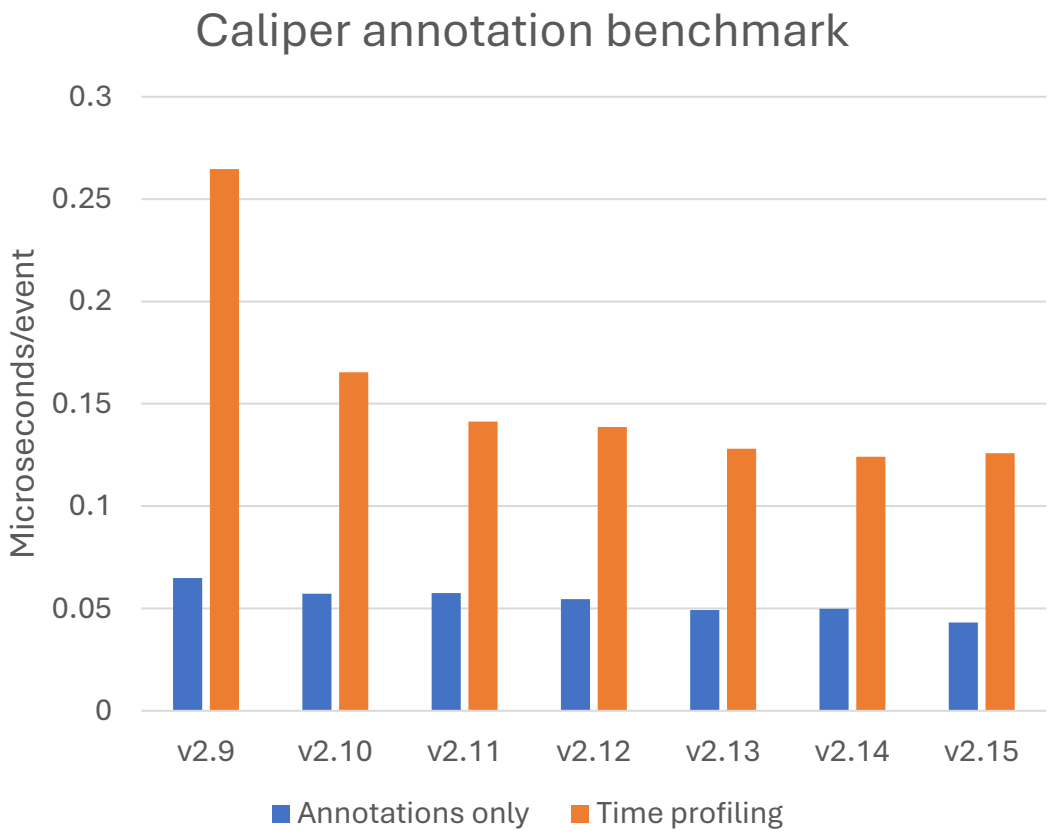
Example: Multi-grid level communication in AMG





Performance optimizations

- Many low-level optimizations
  - Aggregator rewrite (v2.10)
  - Data structure optimizations
  - Removing unnecessary locks
  - Removing unused branches and runtime options
- File I/O and query processor also improved



Automatically collecting custom run metadata

- Adiak: A small library for collecting key-value run metadata

```
adiak::collect_all(); // new: invoke all Adiak built-in metadata collectors

vector<int> ints { 1, 2, 3, 4 };
adiak::value("myvec", ints);
adiak::value("mydouble", 42.0);
adiak::value("mystring", "hi");

/* C API uses a printf-style interface */
float floats[2] = { 42.0, 7.0 };
adiak_namevalue("myint", adiak_general, NULL, "%d", 42);
adiak_namevalue("myfloats", adiak_general, NULL, "[%f32]", floats, 2);
```

- New: Add run metadata in Caliper config string or from JSON file

```
$ CALI_CONFIG=runtime-profile,metadata(experiment=test_1),metadata(file=/etc/node_info.json,keys=host.os) basic_example
```

Example: Lulesh run metadata

Adiak built-in metadata

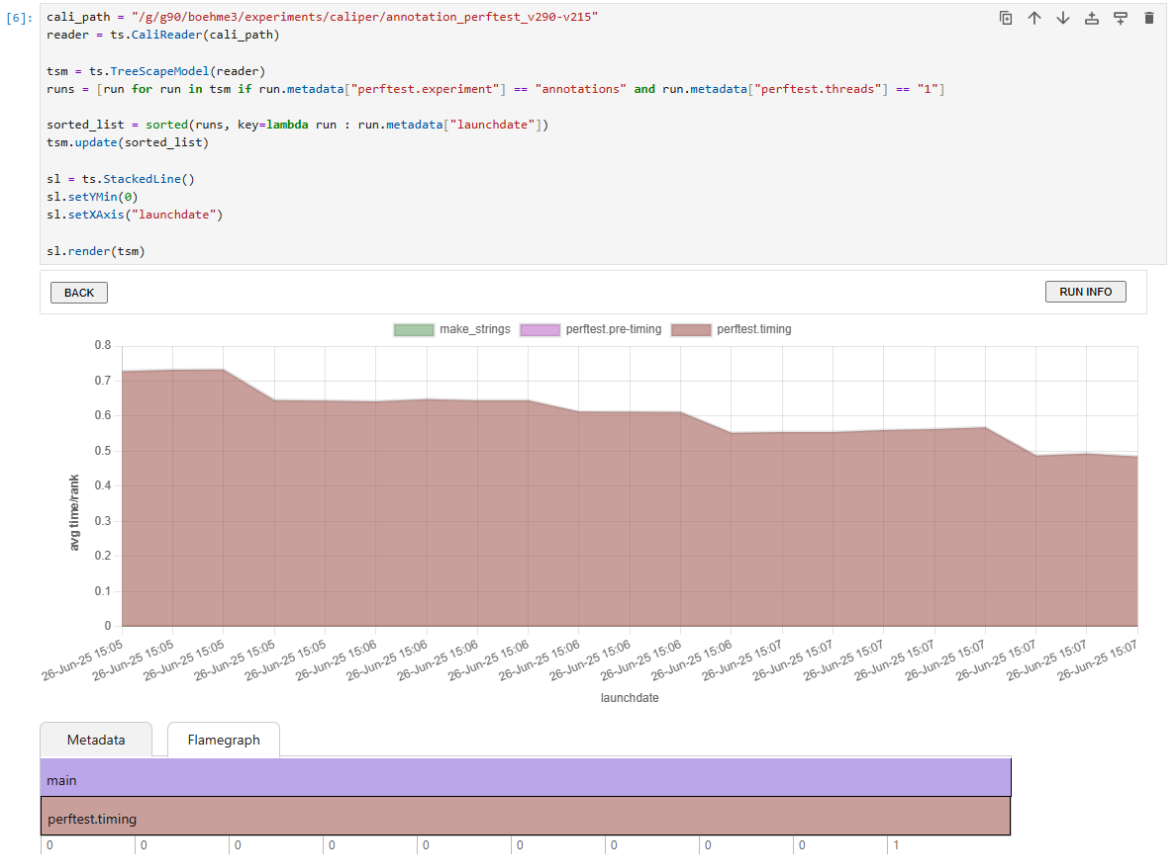
```
$ CALI_CONFIG=runtime-report,print.metadata lulesh2.0 -i 10
mpi.world.size        :          1
cali.caliper.version  : 2.15.0
adiakversion          : 0.5.0
user                  : boehme3
uid                   : ,,,
launchdate            :  1784240684
launchday             :  1784160000
executable            : lulesh2.0
executablepath        : /home/boehme3/src/caliper~~install/mpi/bin/lulesh2.0
working_directory     : /home/boehme3/src/caliper-tutorial
libraries             : [linux-vdso.so.1,/home/bo~~b/openmpi3/mca_osc_sm.so]
cmdline               : [lulesh2.0,-i,10]
hostname              : talent18
cluster               : talent
jobsize               :          1
numhosts              :          1
hostlist              : [talent18]
mpi_version           : 3.1
mpi_library_vendor    : Open MPI
mpi_library_version   : 4.1.6
threads               :          1
iterations            :         10
problem_size          :         30
num_regions           :         11
region_cost           :          1
region_balance        :          1
Compiler Name         : GNU
Compiler Version      : 13.3.0
Built by              : boehme3
Compiler Flags        :
elapsed_time          :    0.132395
figure_of_merit       : 2039.346132
opts:print.metadata   : true
opts:output.append    : true
opts:order_as_visited : true
starttime.nsec        :   953340561
starttime.sec         :  1784240684
```

Lulesh metadata annotations

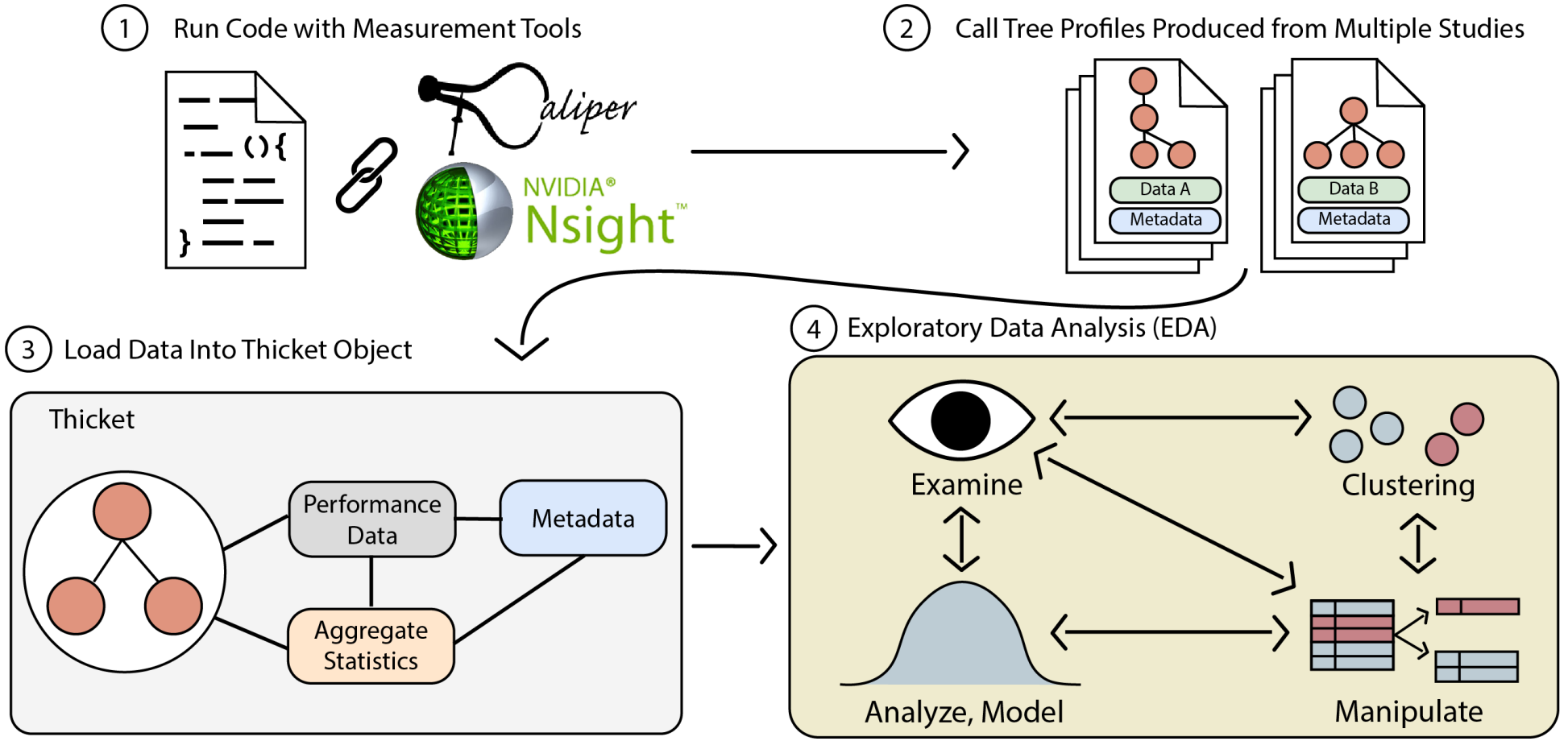
Figure of merit

New: TreeScape analysis frontend

- Jupyter-based framework for visual performance comparisons
- Replaces *spot* web frontend
- Focus on performance regression testing

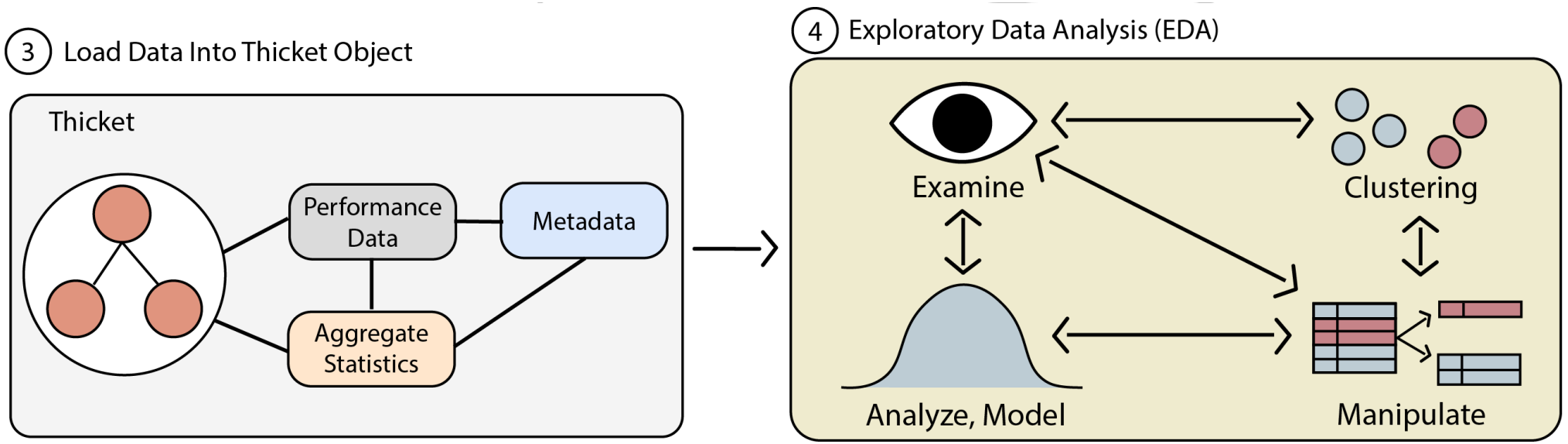


Thicket is a toolkit for exploratory data analysis of multi-run data



https://github.com/LLNL/thicket
https://github.com/LLNL/thicket-tutorial

Thicket enables exploratory data analysis of multi-run data



- Compose data from diff. sources and types
  - Different scaling (e.g., strong, weak)
  - Different application parameters
  - —Different compilers and optimization levels
  - —Different hardware types (e.g., CPUs, GPUs)
  - —Different performance tools

- Perform analysis on the thicket of runs
  - — Manipulate the set of data
  - — Visualize the dataset
  - — Perform analysis on the data
  - — Model data
  - — Leverage third-party tools in the Python ecosystem

Resource links

- Caliper: https://github.com/LLNL/Caliper
- Adiak: https://github.com/LLNL/Adiak
- Thicket: https://github.com/LLNL/Thicket
- TreeScape: https://github.com/LLNL/TreeScape

- Contact: boehme3@llnl.gov