Performance Analysis with Hatchet

2 May 2022

Stephanie Brink, Olga Pearce

LLNL-PRES-813307

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

Getting Hatchet Tutorial Materials

- The SPOT container includes a sample Jupyter notebook, Hatchet 2022.1.0 install, and Lulesh datasets.
  - Alternatively, the sample Jupyter notebook and the Lulesh datasets are available directly at https://github.com/llnl/hatchet-tutorial. This repository is integrated with BinderHub, which will create a local interactive environment for you to run the notebook.

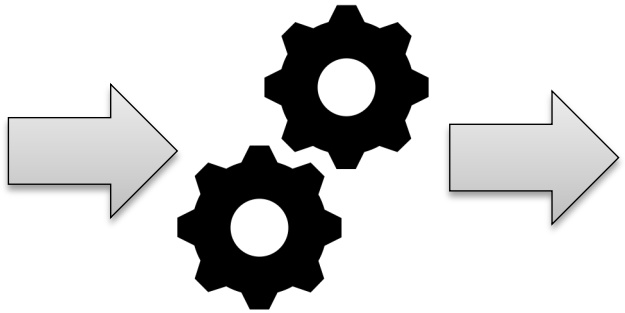
- Following this tutorial, you can substitute your own SPOT/Caliper data files into the example notebook.

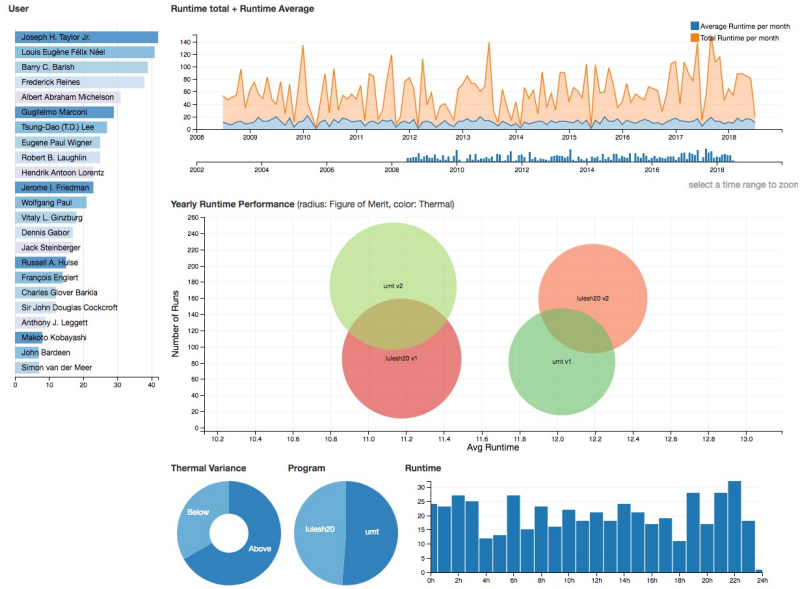
- We'll use this material in the live demo portion of the tutorial.

Automated Application Performance Analysis: Caliper → SPOT → Hatchet

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

static inline
void LagrangeElements(Domain&
domain, Index_t numElem)
{
   CALI_CXX_MARK_FUNCTION;
// ...
```





Caliper instrumentation in the application

At runtime: Performance and Metadata Collection

Web-based Visualization and Analysis Tools

Analyze caliper datasets in Python

c/o D Boehme

SPOT and Hatchet

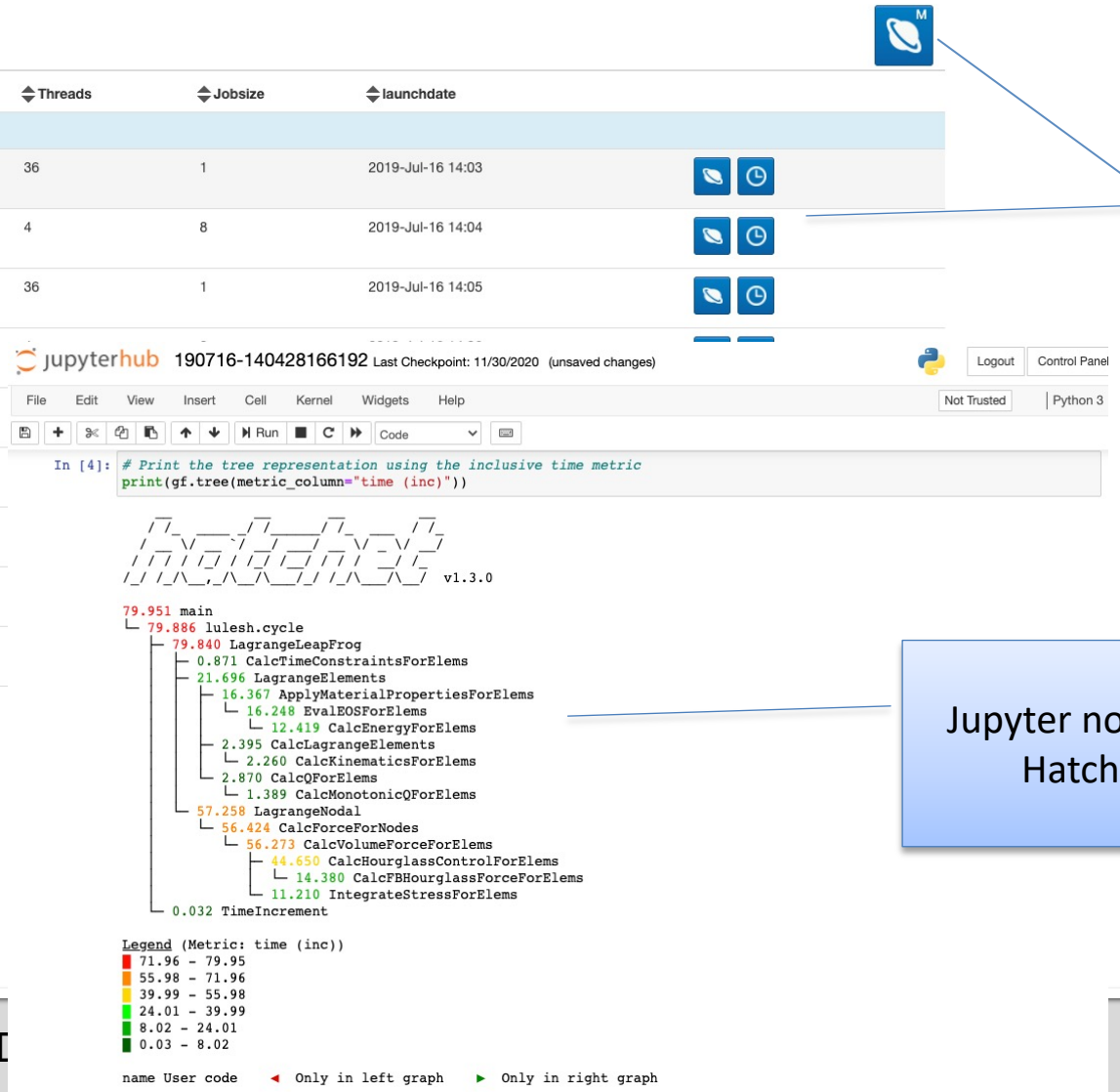
*Hatchet can analyze other datasets (HPCToolkit, gprof, TAU, Ascent (WIP))

SPOT Web Interface: Run Table and Jupyter Notebooks

All records selected. Please click on the graph to apply filters.

TABLE | COMPARE

Figure_of_merit	Problem_size	Threads	Jobsize	launchdate
1				
5491.526855	60	36	1	2019-Jul-16 14:03
12412.877512	30	4	8	2019-Jul-16 14:04
5525.153912	60	36	1	2019-Jul-16 14:05
11339.841229	30			
5100.923858	60			
11197.780437	30			
5071.148951	60			
10247.931707	30			
18468.799766	60			
18666.324557	60			



Buttons bring up Jupyter notebook or specialized analysis views

Jupyter notebook contains Hatchet functions

c/o D Boehme

Hatchet is a performance analysis tool for parallel profiles

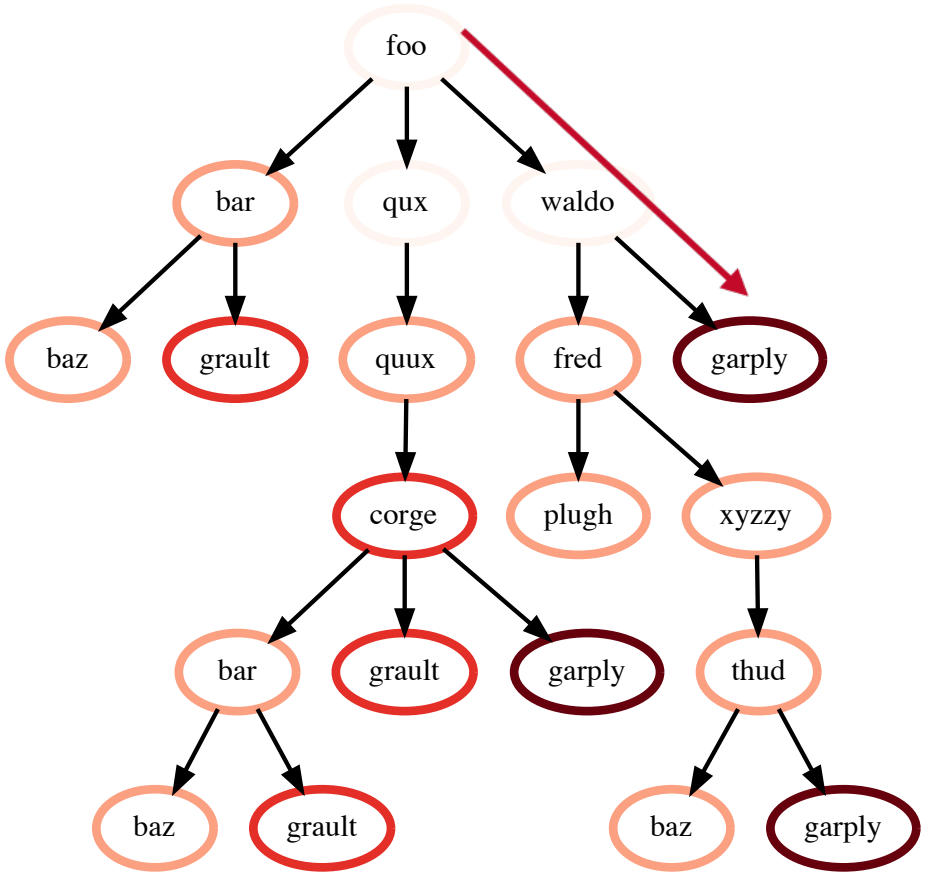
- Identify performance bottlenecks to enhance application development
  - Profiling and tracing tools (*e.g.,* Caliper, HPCToolkit, TAU, Score-P, Gprof, Callgrind) provide insights into parts of the code that consume the most time

- Hatchet is an open-source python-based tool for enabling programmatic analysis of structured (or hierarchical) data

- Hatchet can be used to sub-select and focus on a specific region of the data, compare multiple execution profiles, and automate analysis in python scripts

https://github.com/llnl/hatchet/

What do profiling/tracing tools collect?



Calling Context Tree (CCT)

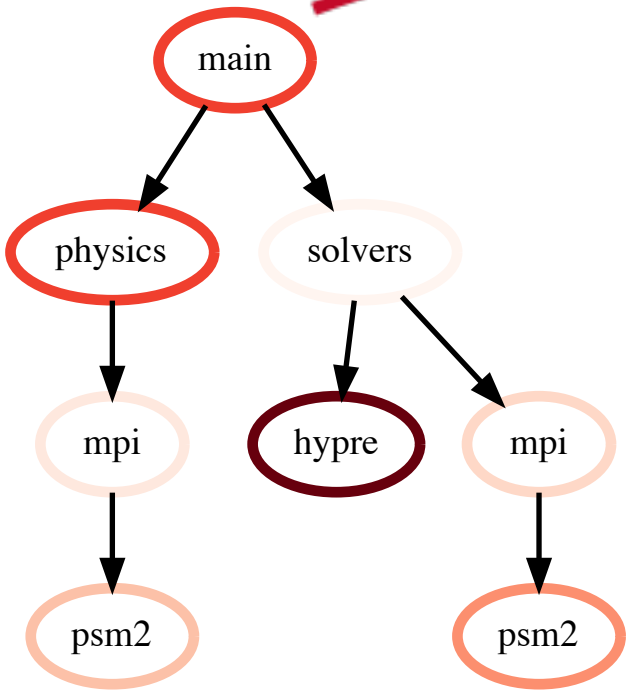
Each node may contain:

- Contextual Info
  - File
  - Line number
  - Function name
  - Callpath
  - Load module
  - Rank ID
  - Thread ID
- Performance Metrics
  - Time
  - Flops
  - Cache misses

Hatchet can read profiles from:

- Caliper
- HPCToolkit
- Gprof
- TAU
- Ascent (WIP)

Hatchet's *GraphFrame*: a Graph and a Dataframe



node	name	nid	node	time	time (inc)
main	main	0	main	40.0	200.0
physics	physics	1	physics	40.0	60.0
mpi	mpi	2	mpi	5.0	20.0
psm2	psm2	3	psm2	15.0	15.0
solvers	solvers	4	solvers	0.0	100.0
hypre	hypre	5	hypre	65.0	65.0
mpi	mpi	6	mpi	10.0	35.0
psm2	psm2	7	psm2	25.0	25.0

Graph: Stores relationships between parents and children

Pandas Dataframe: 2D table storing numerical data associated with each node (may be unique per rank, per thread)

Visualizing Hatchet's GraphFrame components

```
>>> print(gf.tree())      # print graph
>>> print(gf.dataframe)   # print dataframe
```

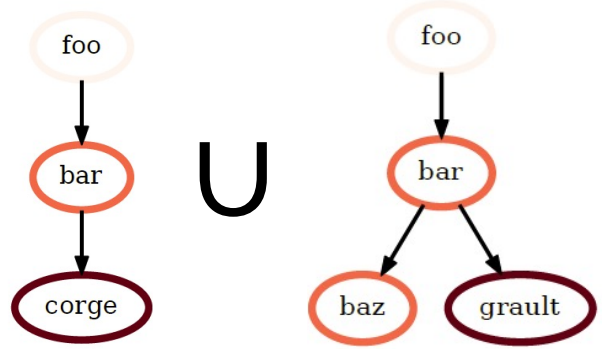
```
0.000 foo
├─ 6.000 bar
│  └─ 5.000 baz
├─ 0.000 qux
│  └─ 5.000 quux
│     ├─ 10.000 corge
│     ├─ 15.000 garply
│     └─ 1.000 grault
└─ 15.000 waldo
   ├─ 3.000 fred
   │  └─ 5.000 plugh
   └─ 15.000 garply
```

Legend (Metric: time)
- 13.50 – 15.00
- 10.50 – 13.50
- 7.50 – 10.50
- 4.50 – 7.50
- 1.50 – 4.50
- 0.00 – 1.50

name User code ◀ Only in left graph ▶ Only in right graph

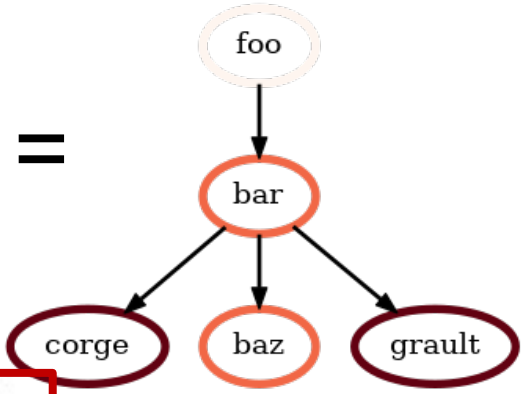
node	name	time	time (inc)
{'name': 'foo'}	foo	0.0	130.0
{'name': 'bar'}	bar	5.0	20.0
{'name': 'baz'}	baz	5.0	5.0
{'name': 'grault'}	grault	10.0	10.0
{'name': 'qux'}	qux	0.0	60.0
{'name': 'quux'}	quux	5.0	60.0
{'name': 'corge'}	corge	10.0	55.0
{'name': 'bar'}	bar	5.0	20.0
{'name': 'baz'}	baz	5.0	5.0
{'name': 'grault'}	grault	10.0	10.0
{'name': 'garply'}	garply	15.0	15.0
{'name': 'grault'}	grault	10.0	10.0

Compare GraphFrames using division (or add, subtract, multiply)




```
>>> gf3 = gf1 / gf2  # divide graphframes
```

*First, unify two trees since structure is different



gf3

```
0.000 foo
├─ 2.000 bar
│  └─ 5.000 baz
└─ inf qux
   └─ 4.000 quux
      ├─ 2.000 corge
      ├─ nan garply
      └─ nan grault
```

=

gf1

```
0.000 foo
├─ 6.000 bar
│  └─ 5.000 baz
└─ 3.000 qux
   └─ 2.000 quux
      └─ 8.000 corge
```

/

gf2

```
0.000 foo
├─ 3.000 bar
│  └─ 1.000 baz
└─ 0.000 qux
   └─ 0.500 quux
      ├─ 4.000 corge
      ├─ 15.000 garply
      └─ 0.250 grault
```

```
>>> gf3 = gf1 + gf2  # add graphframes
>>> gf3 = gf1 - gf2  # subtract graphframes
>>> gf3 = gf1 * gf2  # multiply graphframes
```

Filter the GraphFrame *by node metrics in the dataframe*

```
>>> filter_func = lambda x: x["time"] > 1          # filter function
>>> filt_gf = gf.filter(filter_func, squash=True)  # apply filter and rewire graph
```

```
0.000 foo
├─ 6.000 bar
│  └─ 5.000 baz
├─ 0.000 qux
│  └─ 5.000 quux
│     ├─ 10.000 corge
│     ├─ 15.000 garply
│     └─ 1.000 grault
└─ 15.000 waldo
   ├─ 3.000 fred
   │  └─ 5.000 plugh
   └─ 15.000 garply
```

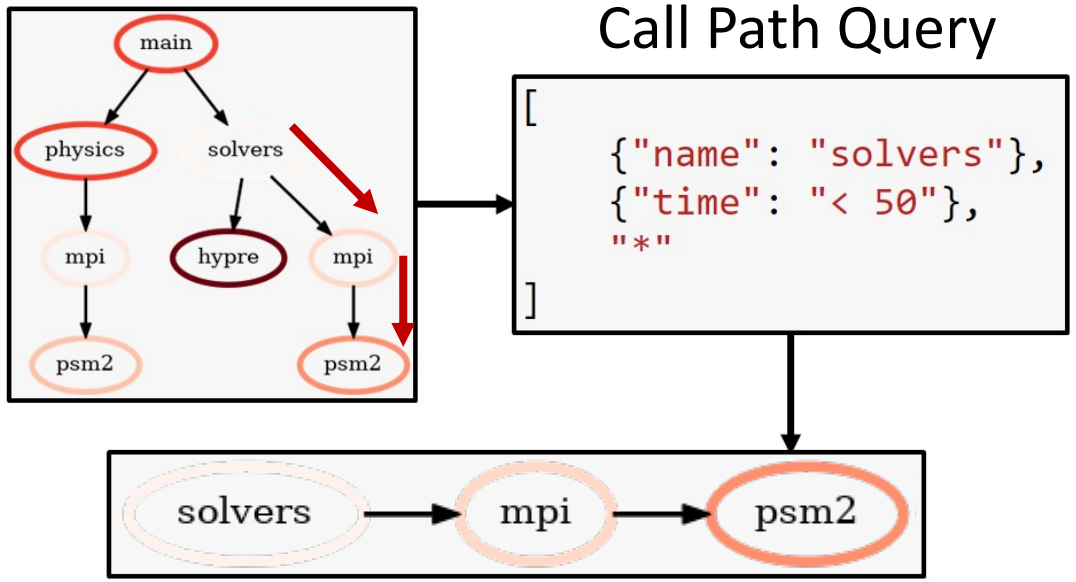
Keep only those nodes with a value greater than 1

```
6.000 bar
└─ 5.000 baz
5.000 quux
├─ 10.000 corge
└─ 15.000 garply
15.000 waldo
├─ 3.000 fred
│  └─ 5.000 plugh
└─ 15.000 garply
```

Filter the GraphFrame *using Hatchet's call path query language*

- Data reduction using *call path* pattern matching

```
# filter using call path query language
query = [
    { "name": "solvers" },
    { "time": "< 50" },
    "*"
]
filt_gf = gf.filter(query, squash=True)
```

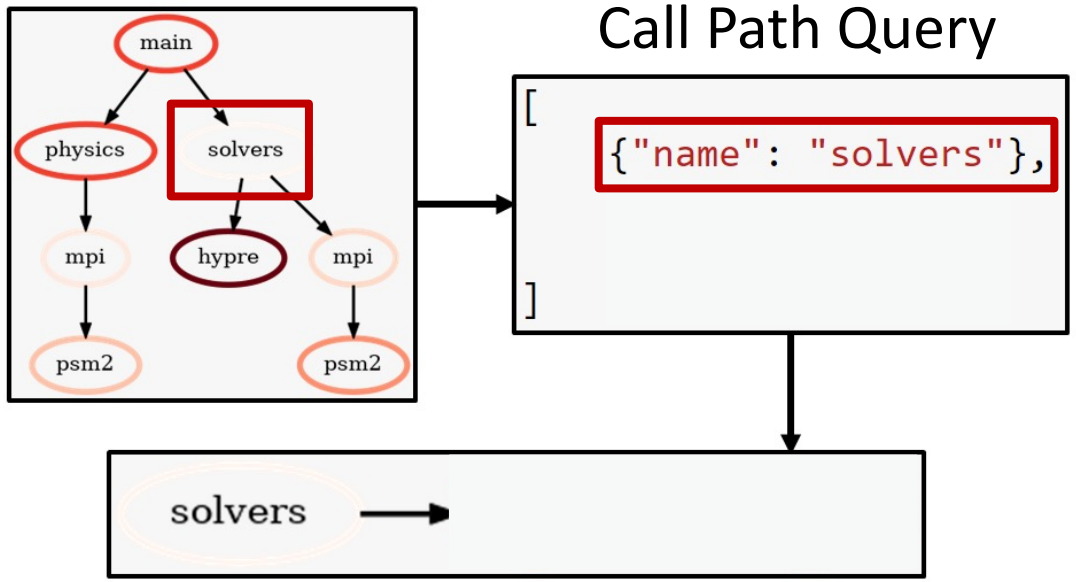


Matches a call path (1) rooted at a node with name "solvers", (2) followed by a node with a time metric value less than 50, and (3) followed by any number of children nodes.

Filter the GraphFrame *using Hatchet's call path query language*

- Data reduction using *call path* pattern matching

```
# filter using call path query language
query = [
    { "name": "solvers" },
    { "time": "< 50" },
    "*"
]
filt_gf = gf.filter(query, squash=True)
```

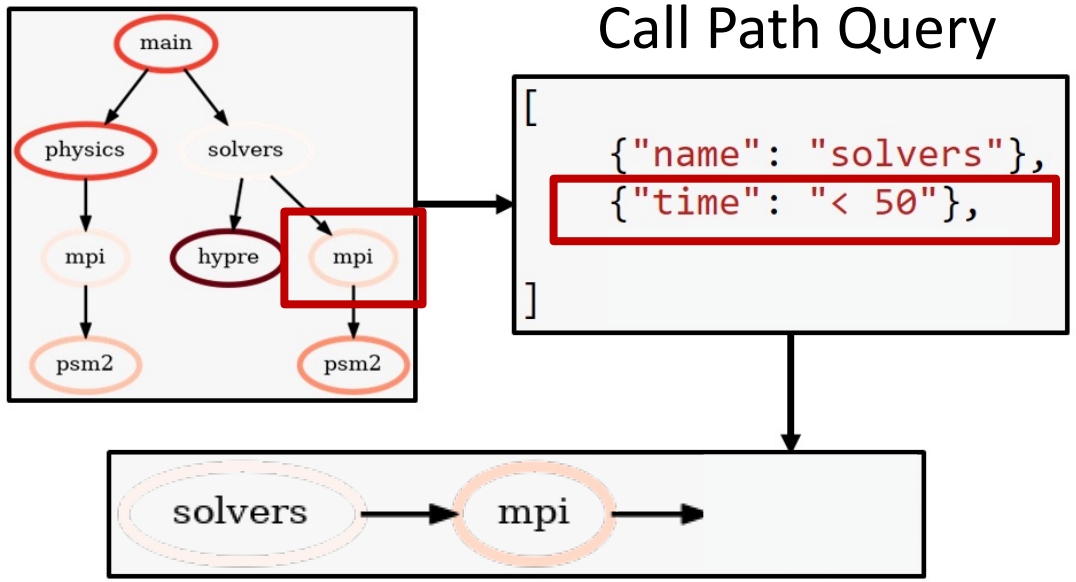


Matches a call path (1) rooted at a node with name "solvers", (2) followed by a node with a time metric value less than 50, and (3) followed by any number of children nodes.

Filter the GraphFrame *using Hatchet's call path query language*

- Data reduction using *call path* pattern matching

```
# filter using call path query language
query = [
    { "name": "solvers" },
    { "time": "< 50" },
    "*"
]
filt_gf = gf.filter(query, squash=True)
```

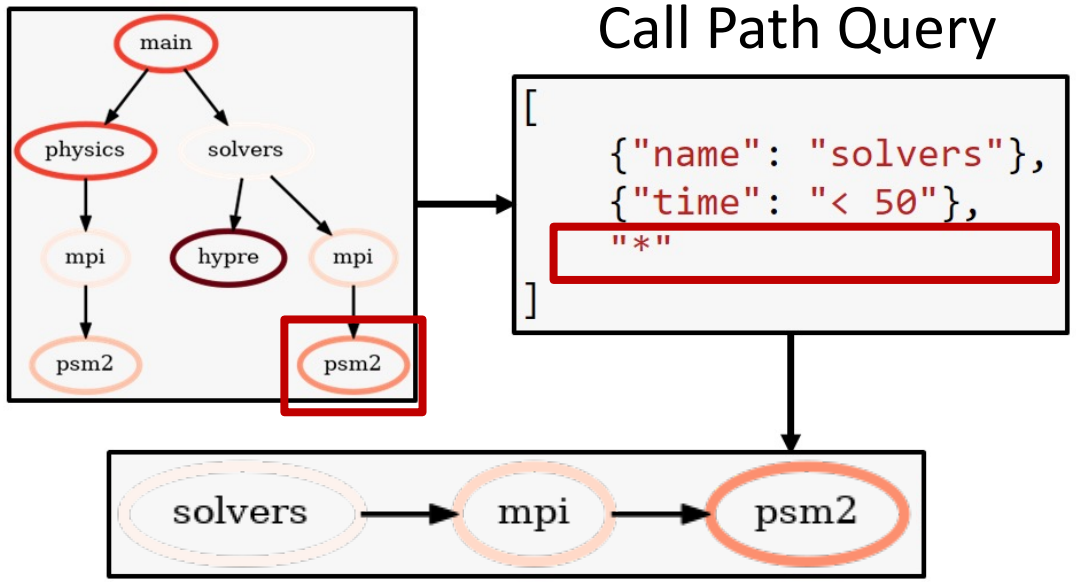


Matches a call path (1) rooted at a node with name "solvers", (2) followed by a node with a time metric value less than 50, and (3) followed by any number of children nodes.

Filter the GraphFrame *using Hatchet's call path query language*

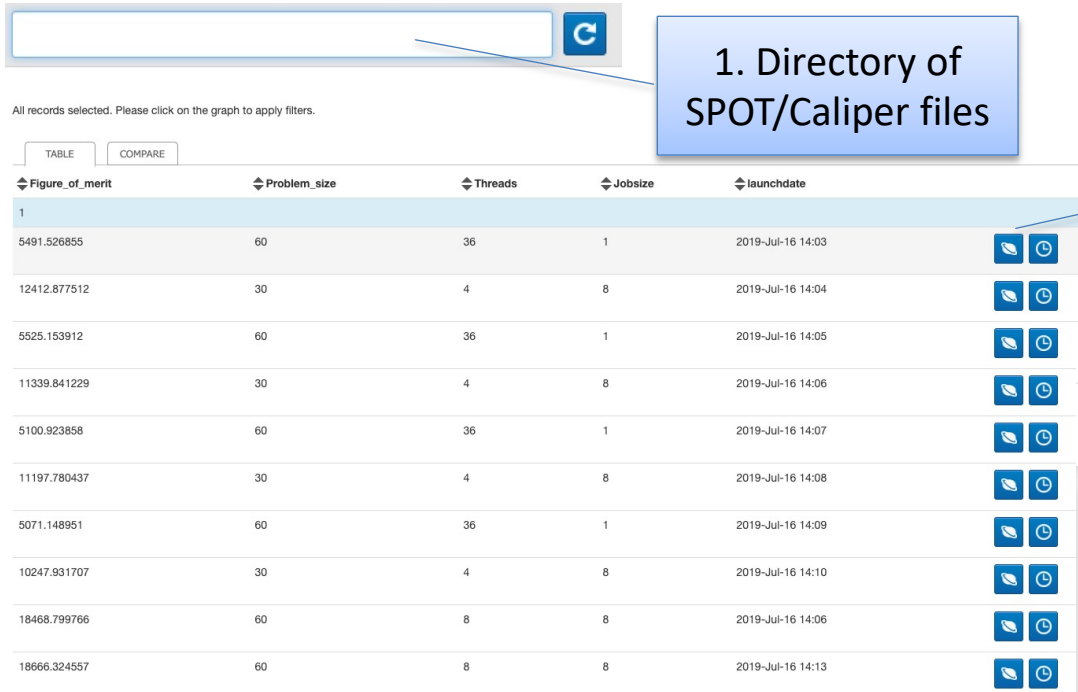
- Data reduction using *call path* pattern matching

```
# filter using call path query language
query = [
    { "name": "solvers" },
    { "time": "< 50" },
    "*"
]
filt_gf = gf.filter(query, squash=True)
```



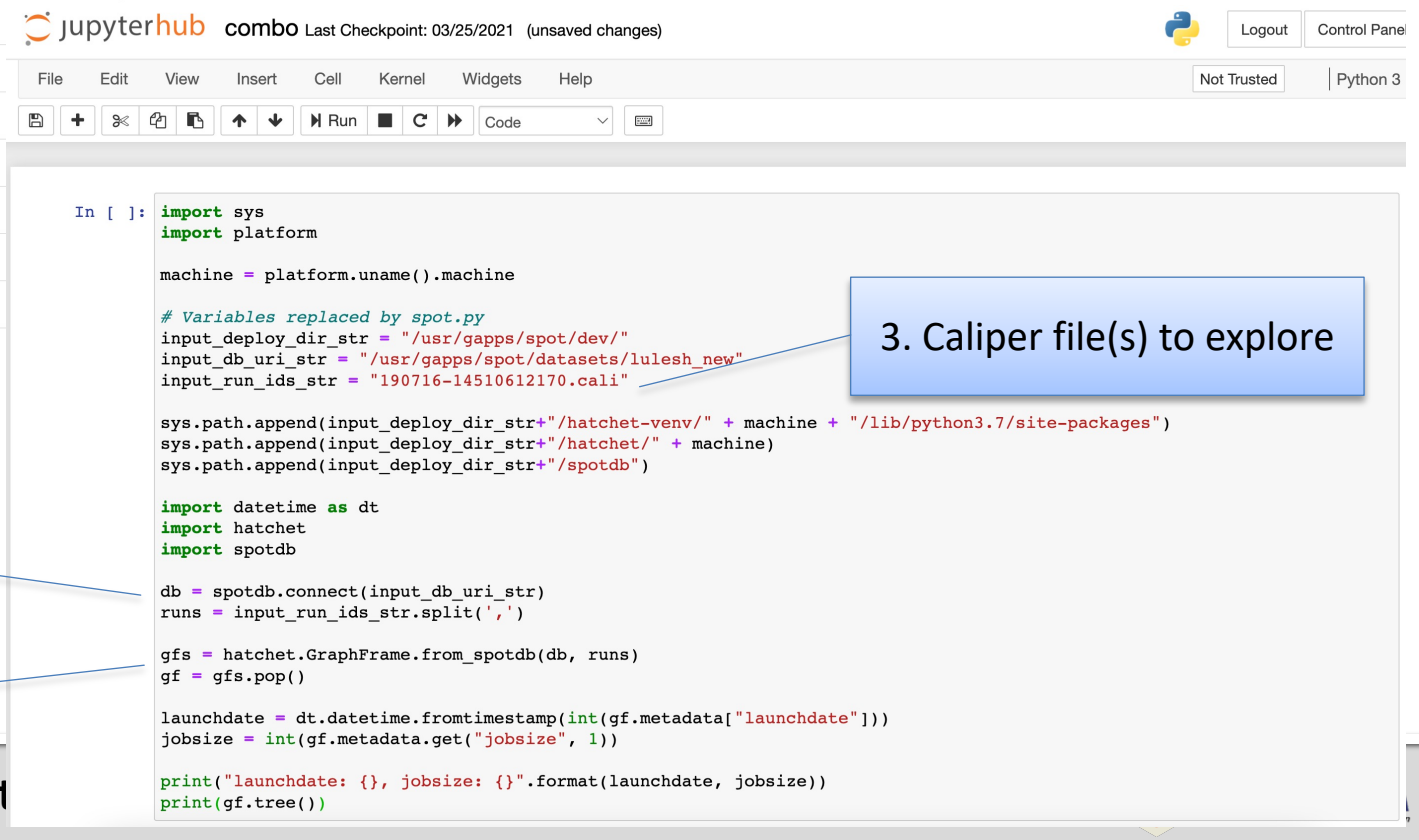
Matches a call path (1) rooted at a node with name "solvers", (2) followed by a node with a time metric value less than 50, and (3) followed by any number of children nodes.

How do I load SPOT/Caliper data into Hatchet?



1. Directory of SPOT/Caliper files

2. Buttons bring up filled-in Jupyter notebook loading 1 or many SPOT/Caliper files



3. Caliper file(s) to explore

4. Connect to SPOT database

5. Hatchet's SPOT database reader loads into Hatchet's GraphFrame object

Hands-On Time!

- The SPOT container includes a sample Jupyter notebook, Hatchet 2022.1.0 install, and Lulesh datasets.
  - Alternatively, the sample Jupyter notebook and the Lulesh datasets are available directly at https://github.com/llnl/hatchet-tutorial. This repository is integrated with BinderHub, which will create a local interactive environment for you to run the notebook.
- Following this tutorial, you can substitute your own SPOT/Caliper data files into the example notebook.
- Hop over to Jupyter to run the notebook

- We'll be walking through `hatchet_tutorial_demo.ipynb`

Review: Topics covered in today's tutorial

- Single graph:
  - **Load** SPOT/Caliper data file
  - **Visualize** tree and dataframe
  - **Filter and squash** tree

```
# Read in a SPOT/Caliper file
gf = ht.GraphFrame.from_spotdb(
       <spot-database>,
       <list-of-runs>,
     )
```

```
# Print tree visualization
print(gf.tree(
   metric_column="Total time (inc)"))
```

- Subtract two trees:
  - Load two SPOT/Caliper data files
  - Compute **percent change** of two nightly test runs (two different times)
  - **Update** existing **column** in dataframe
  - **Added** new **column** to dataframe
  - Visualize resulting tree

```
# Print dataframe
print(gf.dataframe)
```

- Speedup of two trees:
  - Load two SPOT/Caliper data files
  - **Divide** two graphs for speedup comparison
  - Visualize resulting tree
  - **Generate speedup plot** for interesting functions

```
# Divide two trees
gf3 = gf2 / gf1
```

```
# Diff two trees
gf3 = (gf2 – gf1) / gf1
```

Readily available features not covered in today's tutorial

- **Add** or **multiply** two graphframes
- **Insert new column** to dataframe of metrics
  - Scale and offset "time" column by some factor: https://llnl-hatchet.readthedocs.io/en/latest/advanced_examples.html#applying-scalar-operations-to-attributes
  - Compute imbalance across MPI ranks within a single application execution: https://llnl-hatchet.readthedocs.io/en/latest/advanced_examples.html#applying-scalar-operations-to-attributes
- **Groupby-and-aggregate** nodes by other columns (e.g., function name, file name)
  - `res = gf.groupby_aggregate(["file"], {"time": np.sum})`
- For more details, please visit our User Guide: https://llnl-hatchet.readthedocs.io/en/latest/user_guide.html

Summary

- Hatchet is a performance analysis tool for parallel profiles
- It enables programmatic analysis of hierarchical data from one or multiple execution profiles
- Future Work:
  - Support other profile formats, add a format for outputting GraphFrames to disk
  - Implement a higher-level API for automating performance analysis

- Hatchet https://github.com/LLNL/hatchet
  https://github.com/LLNL/hatchet-tutorial
- Caliper https://github.com/LLNL/Caliper
- SPOT https://github.com/LLNL/spot2_container

Please contact us or submit GitHub issues for Hatchet questions, issues, or feature requests!