Missing the Trees for the Branches

Graphical-Scripting Interaction with Large-Scale Calling Context Trees

Connor Scully-Allison, Katherine E. Isaacs (Advisor), Olga Pearce (Advisor)
Lawrence Livermore National Lab • University of Arizona

Tension Between Scripting and Visualization in HPC Analysis

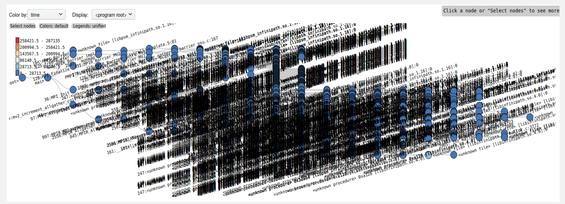
Performance analysis scripts are highly flexible, but not always intuitive.

Performance visualization can be intuitive but restrictive.

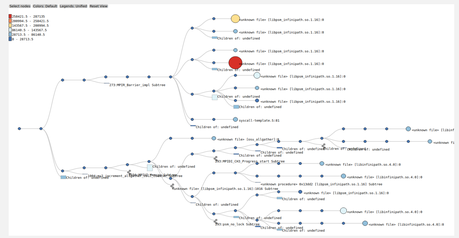
To ease this tension, we introduce a graphical-scripting interaction paradigm that combines scripting and visual analysis.

To support this new workflow in real life analysis scenarios we also introduce a scalable Calling Context Tree representation.

Scaling Node-Link Calling Context Tree Visualizations



Full CCT



With Automated Filtering and Pruning

We prune and collapse uninteresting subtrees automatically, using box-plot outlier detection. We also support further manual adjustments.

Graphical-Scripting Interaction Paradigm

Scripting: Jupyter Notebook

```
In [32]: gf1 = ht.GraphFrame.from_caliper_json("datasets/lulesh-annotation-profile-1core.json")
         gf2 = ht.GraphFrame.from_caliper_json("datasets/lulesh-annotation-profile-8cores.json")

         gf1.drop_index_levels()
         gf2.drop_index_levels()

         speedup = gf1/gf2

         speedup.dataframe["speedup"] = speedup.dataframe["time"]
         speedup.dataframe = speedup.dataframe.drop(columns=['time (inc)', 'time'])

         gf1.dataframe = gf1.dataframe \
                           .reset_index() \
                           .join( \
                               speedup.dataframe.reset_index().set_index(['nid','name']),
                               on=['nid','name'],
                               lsuffix='_l',
                               rsuffix='_r'
                           )

         gf1.dataframe = gf1.dataframe.drop(columns=['_missing_node', 'node_r']).set_index('node_l')

         gf1.exc_metrics.append("speedup")
```

Users of the performance analysis library, Hatchet, can visualize a whole or pre-filtered CCT.

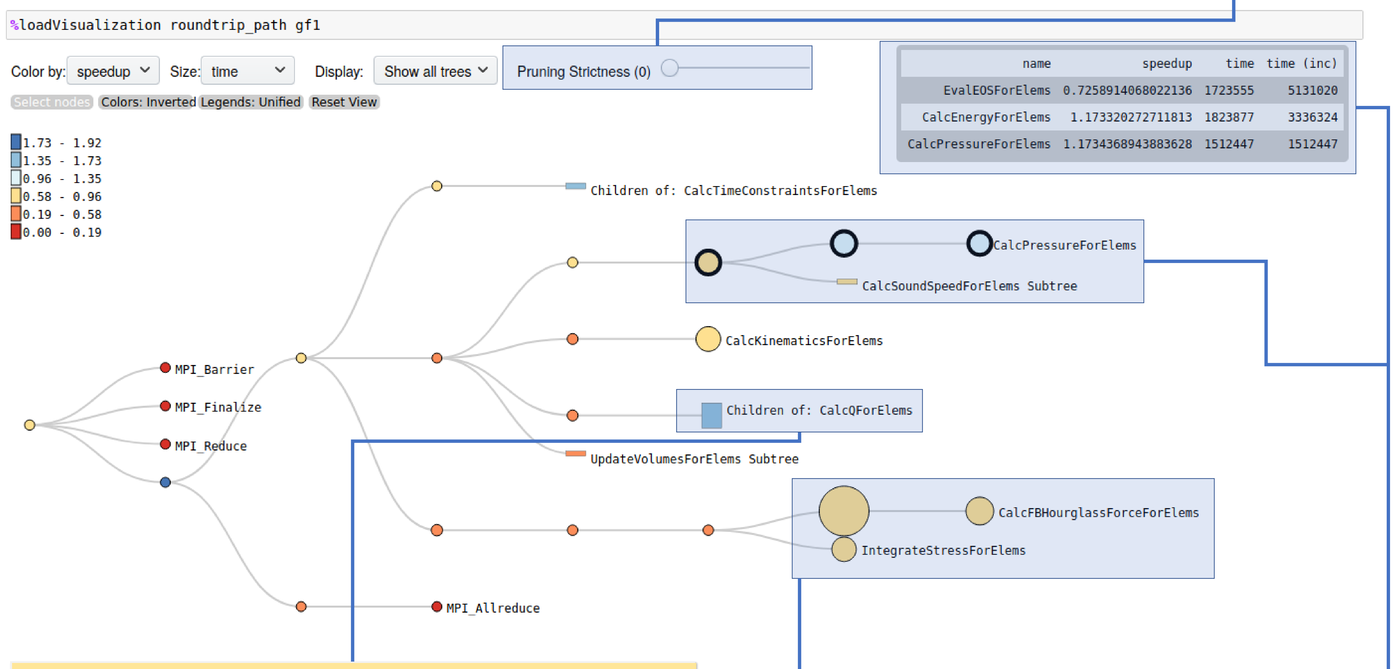
```
In [16]: # Execute this cell to populate nodeSubselection with your selection
         %fetchData nodeSubselection

         print(nodeSubselection)

         gf1.filter(myQuery);

         [{'name': 'EvalEOSForElems'}, '*', {'depth': '<= 7'}]
```

Selected nodes and filters in the visualization are brought back to scripting with a single command.

script-to-vis

vis-to-script

Graphical: Interactive Visualization



Hinting at Elided Elements

Uninteresting nodes are combined and represented as a sum over the removed metric values.

Multivariate Encoding

We show two metrics at the same time using size and color. This cluster of nodes with high runtimes and low speedup are good optimization candidates.

Select Nodes, Details on Demand

Nodes can be selected by users to spawn a tooltip containing detailed information. Selected nodes can be returned to the Jupyter notebook as a query.

Links

Hatchet Github

Roundtrip Github

Demo Video

Lawrence Livermore National Laboratory
LLNL-ABS-825366

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. as well as the United State Department of Defense through DTIC Contract FA8075-14-D-002-007, the National Science Foundation under NSF III-1656958, and the Department of Energy under DE-SC0022044.