Scalable Comparative Visualization of Ensembles of Call Graphs using *CallFlow*

github.com/LLNL/CallFlow

Suraj P. Kesavan[1], Harsh Bhatia[2], Abhinav Bhatele[3], Stephanie Brink[2], Olga Pearce[2], Todd Gamblin[2], Peer-Timo Bremer[2], Kwan-Liu Ma[1]

[1] University of California, Davis [2] Lawrence Livermore National Laboratory [3] University of Maryland, College Park

Motivation

- Large-scale parallel applications demand a variety of deployment modes (e.g., different optimization modes, application parameters, etc.) to identify performance bugs across versions of code or understand how different application parameters and/or initial conditions may affect the performance.
- Importance lies in comparing the execution runtimes and monitoring the performance variation across executions.
- Visualization of performance metrics along with Calling Context Tree (CCT) play an essential role to detect and understand the true causes of bottlenecks.
- However, existing tools are constrained to analysis on single CCT, thereby limiting the usability for analysis on large ensembles of profiles.

CallFlow – What's new?

CallFlow [1] is an interactive visual analysis tool that provides a high-level overview of CCTs together with semantic refinement operations to progressively explore the calling contexts. CallFlow introduces a construct called ***super graph***, created by aggregating the nodes of a CCT based on the semantic attributes, e.g., the library name, module name, or file name.

In this work, we extend CallFlow's visual design to scale performance analysis on large ensembles using ***ensemble super graph***. By combining data analysis and visualization, CallFlow's visual interface along with ensemble super graph representation enables comparison of the **performance variability** across ensembles.

Construction of Ensemble Super Graph

Performance profiles collected from HPCToolkit [2] and Caliper [3] are converted to an ensemble super graph.

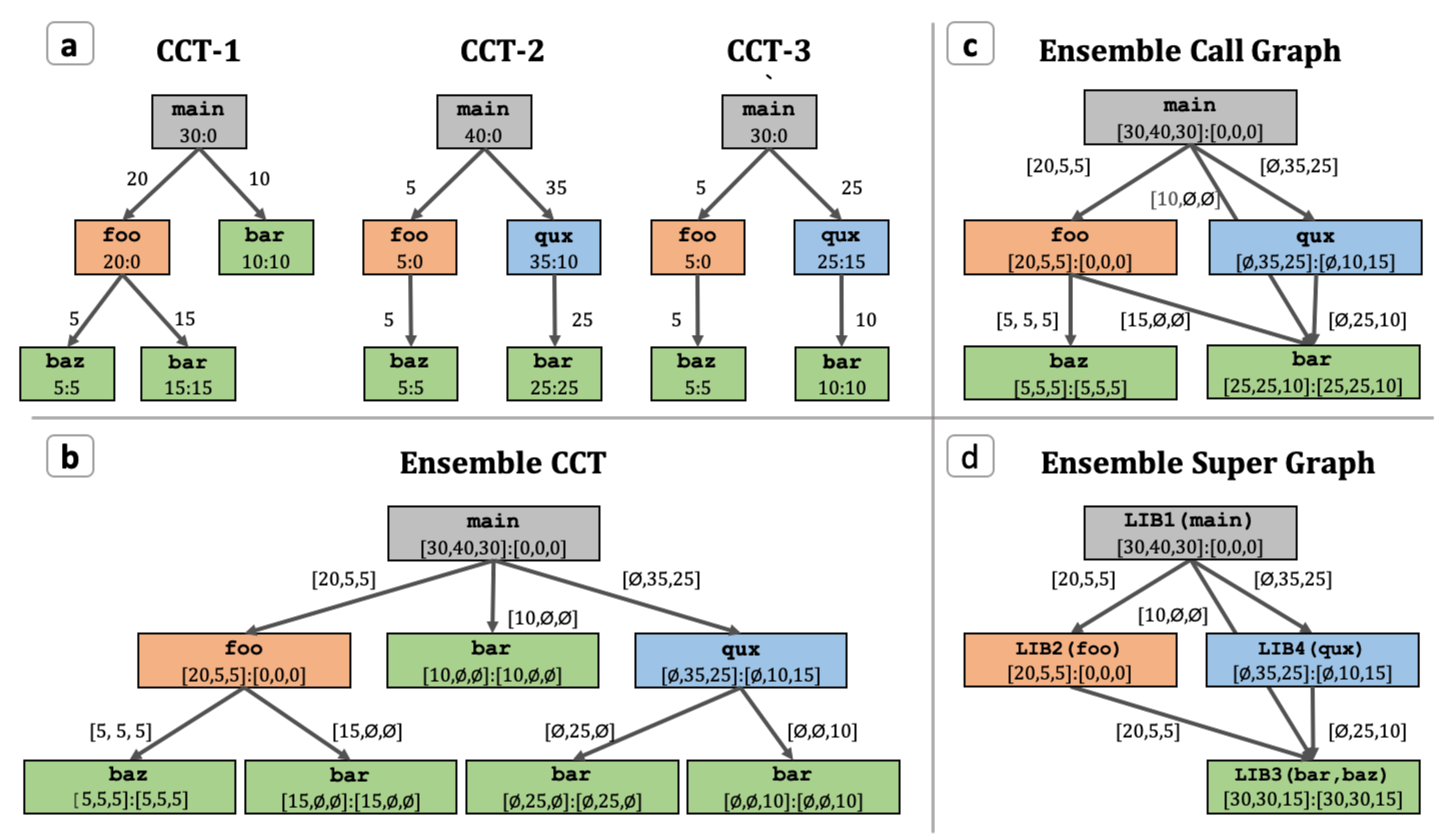


Fig. 1: **Construction of ensemble super graph**. Each node in graph represents a profiled region (e.g., **main**, **foo**), and the performance metrics ([*inclusive runtime*]:[*exclusive runtime*]). Each region is tagged to belong to a module/library (e.g., **LIB1**, **LIB2**)

Performance trends in LULESH

We demonstrate 2 comparison studies using LULESH[5] to study effects of weak scaling on modules across an ensemble of 1, 8, 27, 64, 125, 216, 343, and 512 cores.

Pairwise Comparison of Profiles using *Diff View*

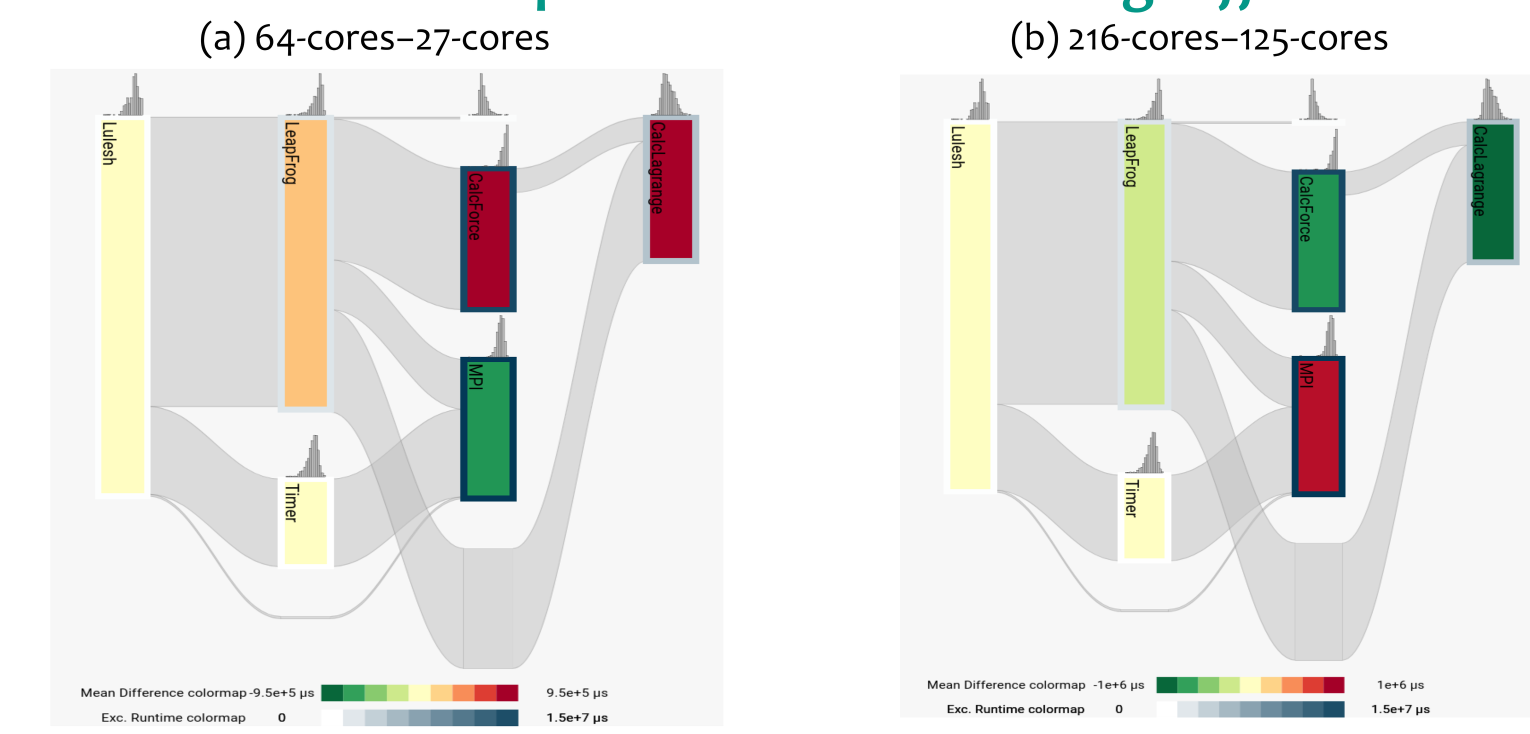


Fig. 3: **Diff view:** Pairwise differences (subtract operation) of the runtimes between 2 selected runs (i.e., (a) 64 vs 27, (b) 216 vs 125 cores) are colored using a green-red color map.

- *Diff View* highlights the slower modules, and communicates the relative degree of performance degradation easily
- Red hues highlight code regions that cause a performance slow down (e.g., CalcForce and CalcLagrange in (a), and MPI in (b)).
- Green hues highlight the performance speedup (e.g., MPI in (a), and CalcForce, and CalcLagrange in (b)).

Comparison of run-to-run slowdowns

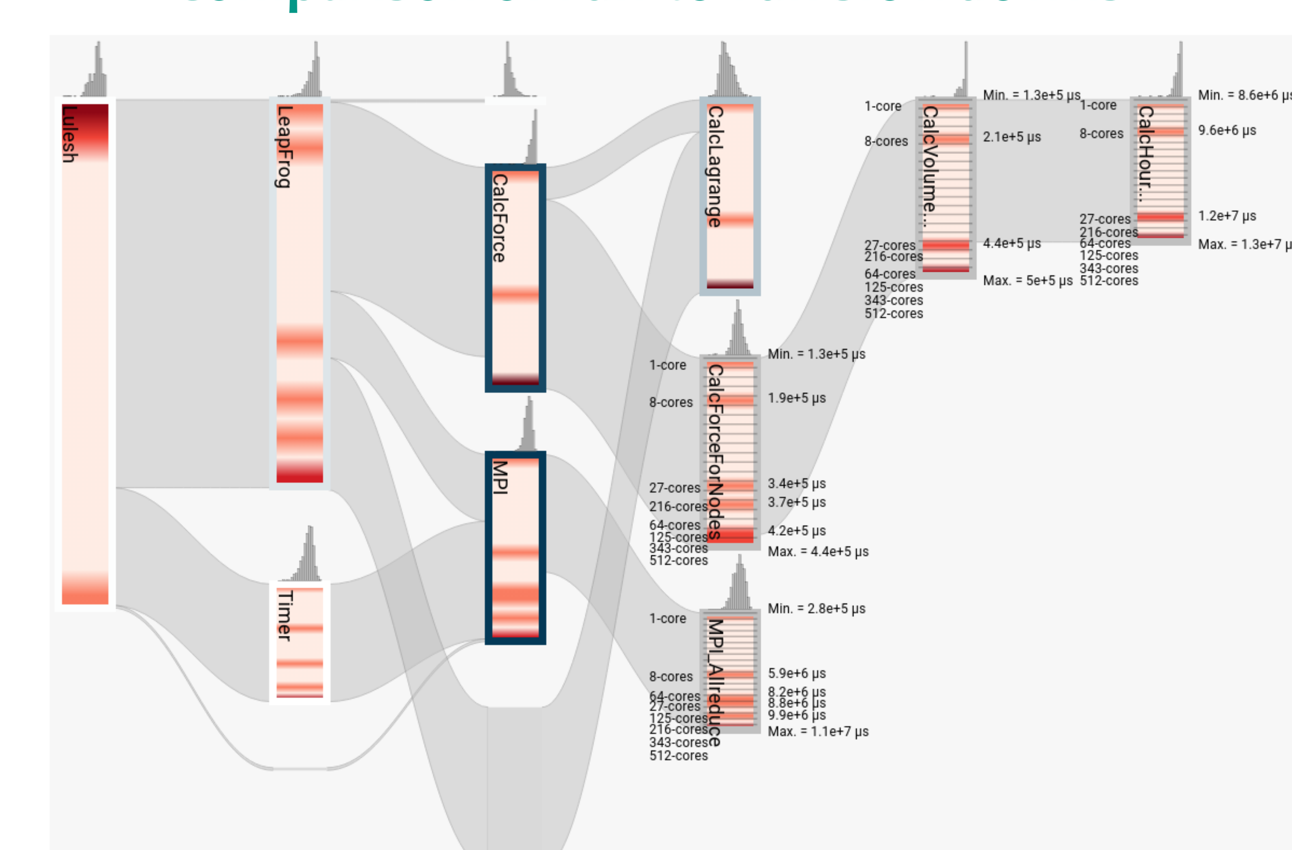
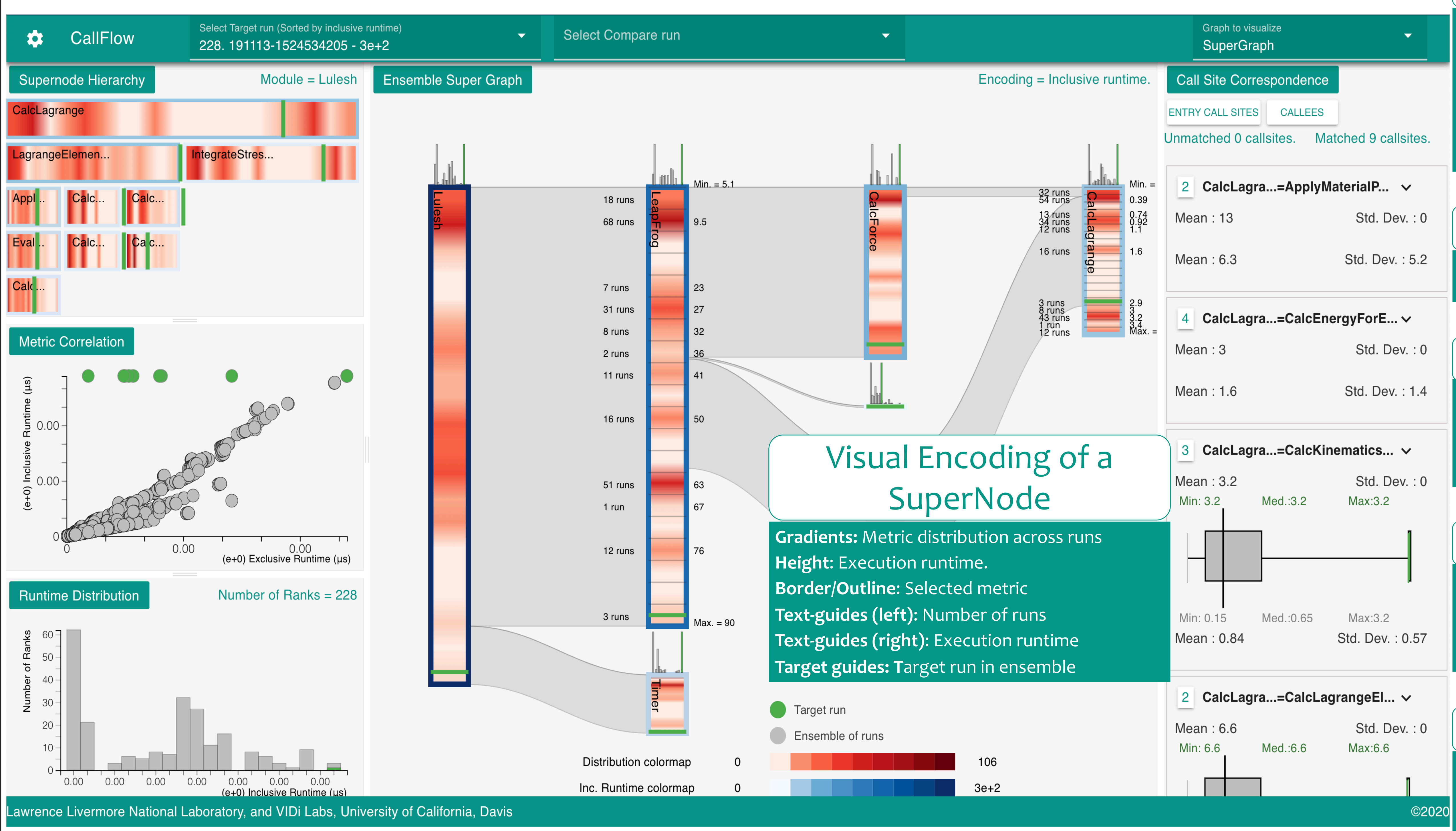


Fig. 4: **Ensemble view:** We perform CallFlow's split operation and reveal the nodes inside MPI and CalcForce library to investigate the out-of-order runtimes (i.e., between 27, 64, 125, and 216 cores).

- Splitting operation reveals the most expensive call sites (e.g., MPI_Allreduce, and CalcForceForNodes).
- The text guides reveal 2 cases of out-of-order runtimes (with respect to increasing core count) for MPI and CalcForce libraries.

CallFlow's Visual Interface



Visual Encoding of a SuperNode

Gradients: Metric distribution across runs
Height: Execution runtime.
Border/Outline: Selected metric
Text-guides (left): Number of runs
Text-guides (right): Execution runtime
Target guides: Target run in ensemble

Fig. 2: **Screenshot of CallFlow's interface:** Visualization of the ensemble comprising of 228 Caliper performance profiles.

Ensemble Super Graph

- Visualizes the aggregated runtimes (histograms) using *white-red color gradients.*
- Text and target guides provide detailed information of the bins in the aggregation.
- Double-clicking on the text guides recalculates the ensemble super graph of the selected subset.

Supernode Hierarchy

- Visualizes the caller-callee relationship of call sites inside a supernode.

Metric Correlations

- Shows correlations between any two metrics across ensemble members.
- Colored dots help in comparing members of a target run to its ensemble.

Runtime Distribution

- Visualizes the distribution in three modes: *call site, call graph, and MPI rank.*
- Target run's distribution (in green) is overlaid over the ensemble.

Call Site Correspondence

- Enumerates the call sites inside a selected supernode.
- Boxplots indicate the spread in the variation of runtimes.
- Visualizes outliers as circles placed along the interquartile range (IQR).
- Enables the user to perform graph splitting operations

Conclusion

We have extended CallFlow [1] to create a scalable, interactive visual analytic tool to study ensembles of CCTs. Working closely with domain experts, we formulate the definition of super graph for an ensemble of profiles and map them to concrete visual mediums to support comparative analysis.

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory (LLNL) under contract DE-AC52-07NA27344 and grant DE-SC0014917. LLNL-POST-813361

References

[1] Nguyen, H.T.P., et al. "Visualizing Hierarchical Performance Profiles of Parallel Codes using CallFlow." IEEE Trans. on Vis. and Comp, Graph. (2019).
[2] Boehme, D., et al. "Caliper: performance introspection for HPC software stacks." SC'16: Proc. of the Int. Conf. for High Performance Computing, Networking, Storage and Analysis. 2016.
[3] Adhianto, L., et al. "HPCToolkit: Tools for performance analysis of optimized parallel programs." Concurrency and Computation: Practice and Experience 22.6 (2010): 685-701.
[4] Bhatele, A., et al. "Hatchet: pruning the overgrowth in parallel profiles." Proc. of the Int. Conf. for High Performance Computing, Networking, Storage and Analysis. 2019.
[5] Karlin, I., et al. "Exploring traditional and emerging parallel programming models using a proxy application." 2013 IEEE 27th Int. Symp. on Parallel and Distributed Processing. 2013.