# Application-Level Energy Accounting with Chappie – Technical Report

Anonymous Author(s)

## 1 Overview

This document provides supplemental material for the Chappie project. Included are a technical report on our experiences working with Chappie inside a simulator, and experimental results.

## 2 Energy Simulation

In this section, we document our (unsuccessful) effort in simulating per-method energy consumption through a cycle-accurate simulator. We faced with two challenges: (1) full-system simulation for realistic Java benchmarks is extremely time-consuming; (2) The behavior of an instrumented benchmark under simulation — where method entry/exit are instrumented with special instructions for demarcating the boundary of energy simulation — significantly differs from that of its unmodified counterpart.

### 2.1 Simulation Setup

We used `gem5` version 2.0 [1] to simulate running a full Ubuntu 16.04 system. In part, we choose this approach to run the Java Virtual Machine (JVM) inside a inside the cycle-accurate simulator. `gem5` simulation statistics were piped through `mcPAT` version 1.3 [2] to simulate power numbers. Our experiments from Chappie base system were performed on a dual socket Intel E5-2630 v4 2.20 GHz CPU server. Please see the main paper Section 5.3 for full details.

We structured two experiments designed to evaluate Chappie in the context of simulation: (E1) we instrumented the top ranked methods for a selected benchmark—as determined by Chappie method energy accounting running on the base system—with code to capture the simulation cycles of the method execution. Our goal was to compare these numbers with base system energy accounting numbers; (E2) we ran Chappie for the selected benchmark on the simulator—we modified Chappie to read simulator cycles instead of energy registers and convert the simulator cycles to energy a posteriori with `mcPAT`—and compare the energy recorded in (E1) with that attributed by Chappie. Our hope was to cross validate the method energy number produced by the cycle-accurate simulate in (E1) with Chappie energy accounting using the same cycle-accurate simulation.

### 2.2 Discussion

Our efforts revealed two fundamental problems with this approach: overhead, and more importantly, application behavior change.

| benchmark | base runtime | gem5 runtime |
|---|---|---|
| `sunflow` | 1.5 (s) | 3.29 (hr) |
| `h2` | 11.4 (s) | DNF |

**Figure 1.** Benchmark Runtime Comparison for `gem5`

We show the runtime of our gem5 exploration experiments for `sunflow` and `h2` in Figure 1. `sunflow`, the smaller of the two benchmarks, completed in 3.29 hours on the gem5 simulator, with a 7895x slowdown compared with the base system. `h2` did not complete. To understand the difficulty with `h2`, we turned to our experiment to instrument one of the top methods, `compareRow` for `h2`, and found that the method is invoked for 58 million times. Based on our observations, we estimate a single run of this invocation on the simulator takes 1.5 seconds, and total execution time is likely to be 990 days. We are aware that speeding up cycle accurate simulation is an active research area [3].

Overhead, however, is the lesser of the problems for us. We implemented an early exit scheme that would collect data for an experiment until a particular top method (`compareRows` in this case) was invoked 100 times. For our experiment where we monitor with Chappie and measure method invocation (E2) in the same run, the results showed that not only was the instrumented code slowing down the application—a manifestation of the scenario described in Section 2 of our main paper—but that the methods attributed energy to were different from the base system; namely, the instrumentation code was altering the application behavior.

***Engineering Effort*** The `gem5` simulation has two methods for simulating programs: *syscall emulation* and `full system simulation`. The former will simulate running a C program through a system call interface that is run directly on the host system. True to its name, the latter loads a virtual machine image and simulates the entire system.

We needed to simulate our selected benchmarks on the simulator which required a running JVM. As such, we took the full system simulation route. A known side effect of full system simulation is the large overhead required simulate *all* aspects of the virtual machine: logging into our simulated virtual machine took 1.5 hrs. We found that irrespective of simulation time for our benchmarks, the full system simulation made iterative development very challenging. We are aware that efforts to speed up simulation in general help to address this.

## 3 Chappie Experimental Results

We include figures for all experiments for each of our benchmarks.

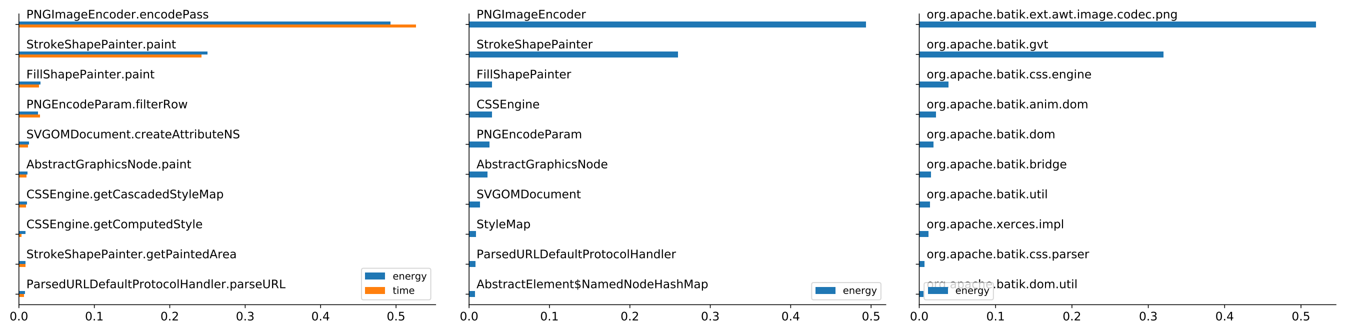

**Figure 2.** Top 10 Energy-Consuming Abstractions for `batik`

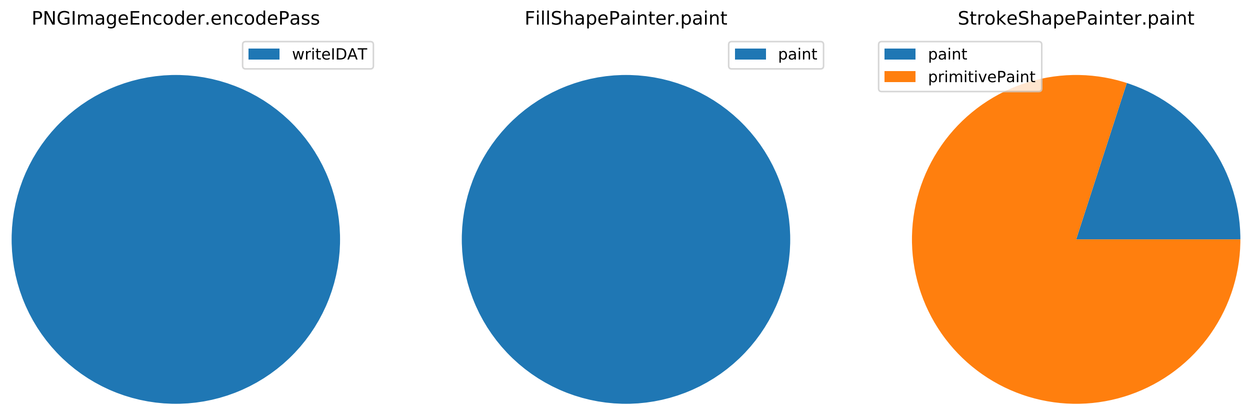

**Figure 3.** 1-CFA for `batik`

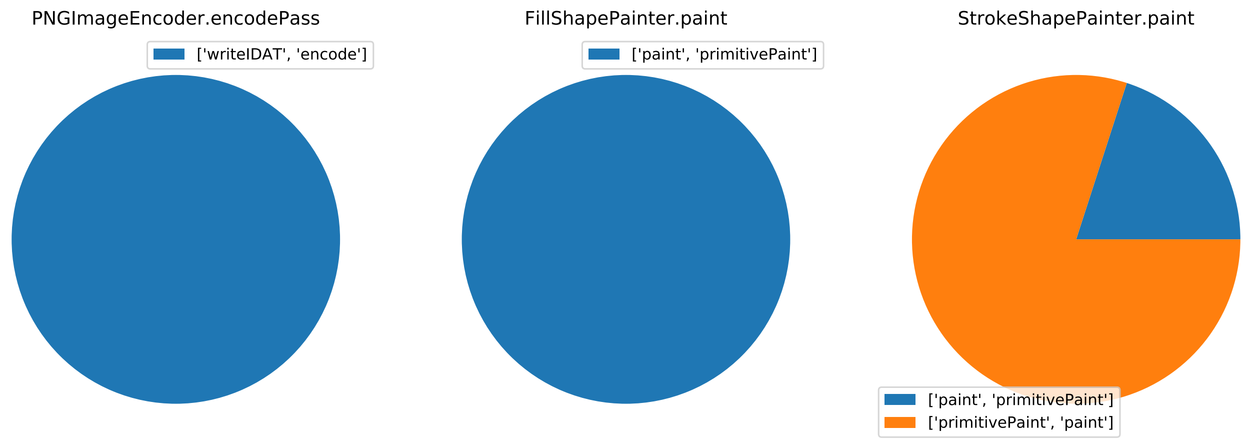

**Figure 4.** 2-CFA for `batik`

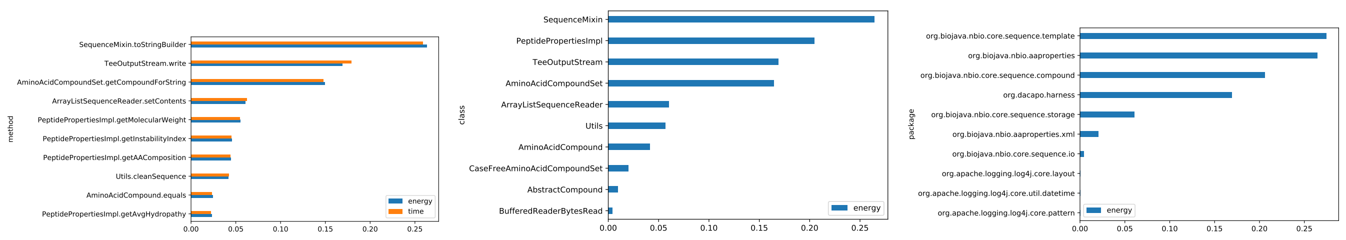

**Figure 5.** Top 10 Energy-Consuming Abstractions for `biojava`

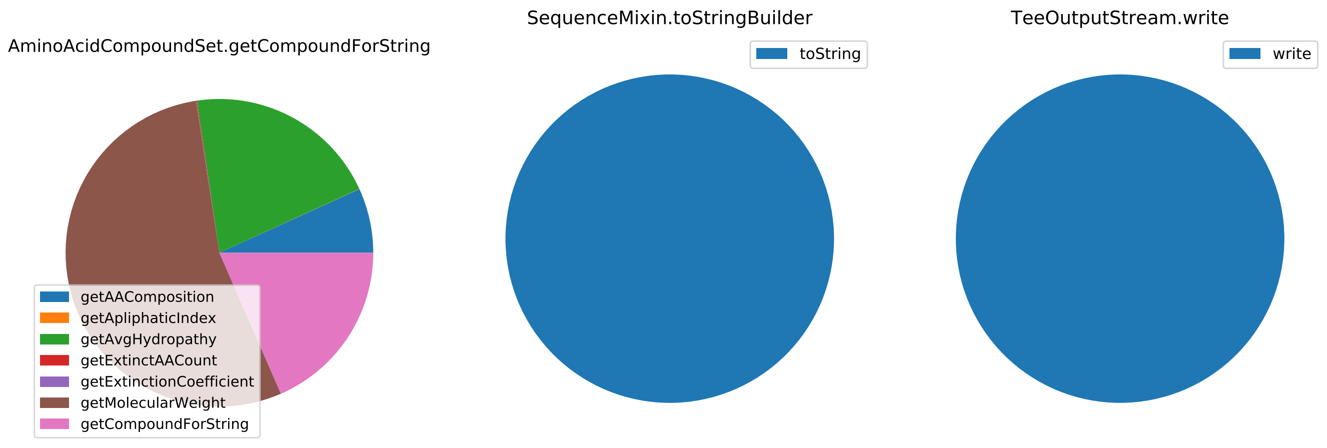

**Figure 6.** 1-CFA for `biojava`

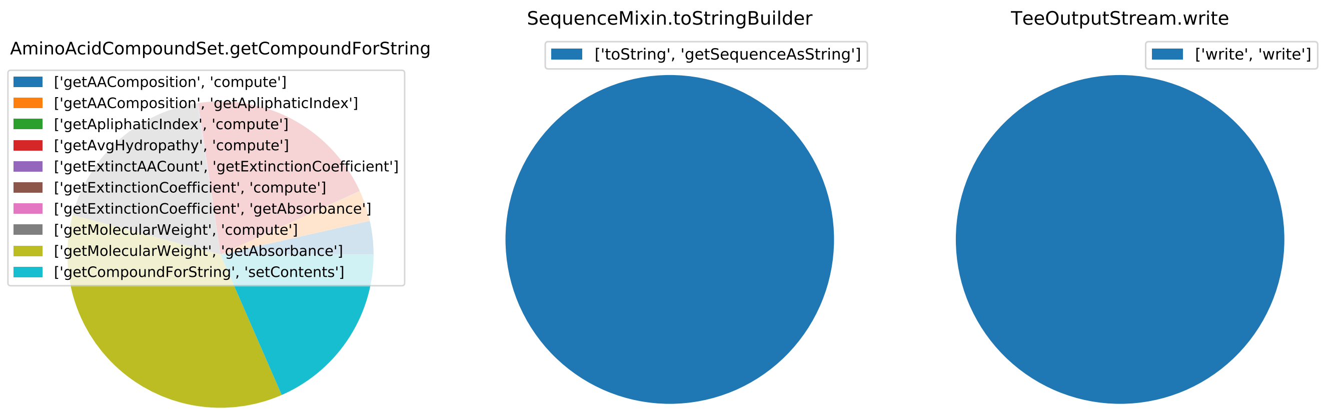

**Figure 7.** 2-CFA for `biojava`

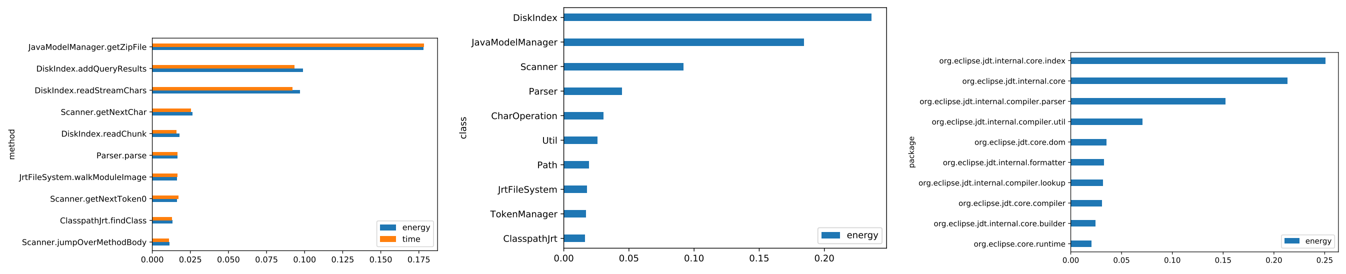

**Figure 8.** Top 10 Energy-Consuming Abstractions for `eclipse`

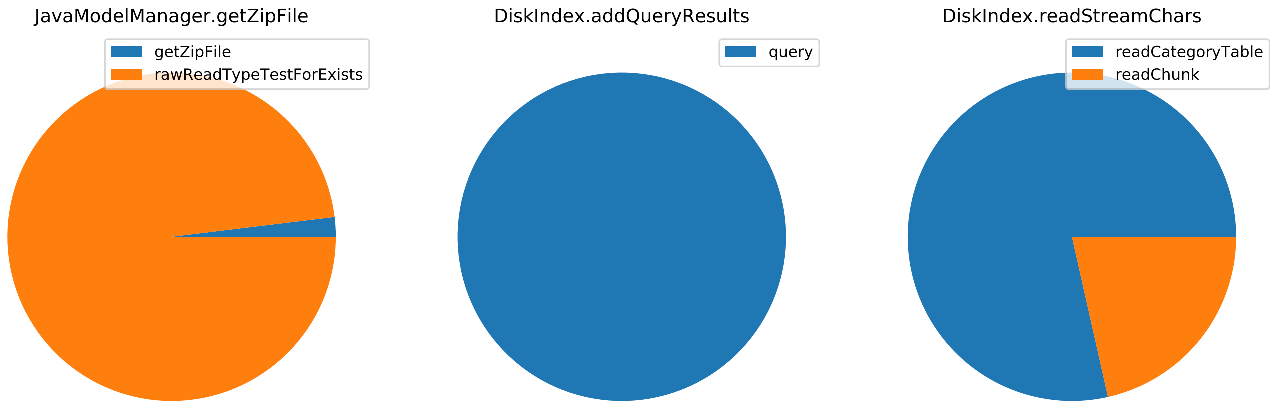

**Figure 9.** 1-CFA for `eclipse`

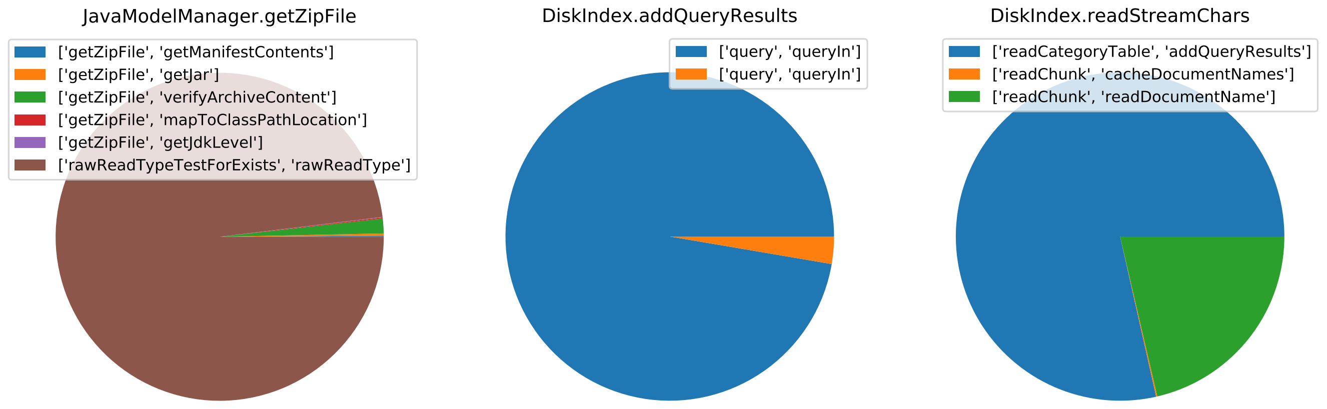

**Figure 10.** 2-CFA for `eclipse`

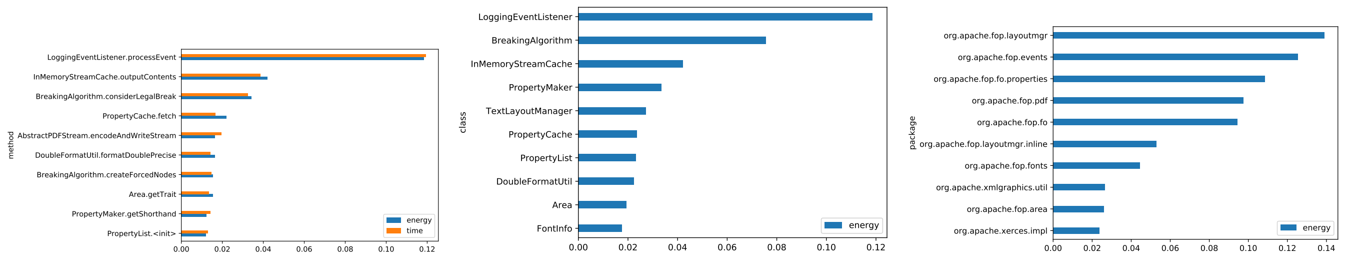

**Figure 11.** Top 10 Energy-Consuming Abstractions for `fop`

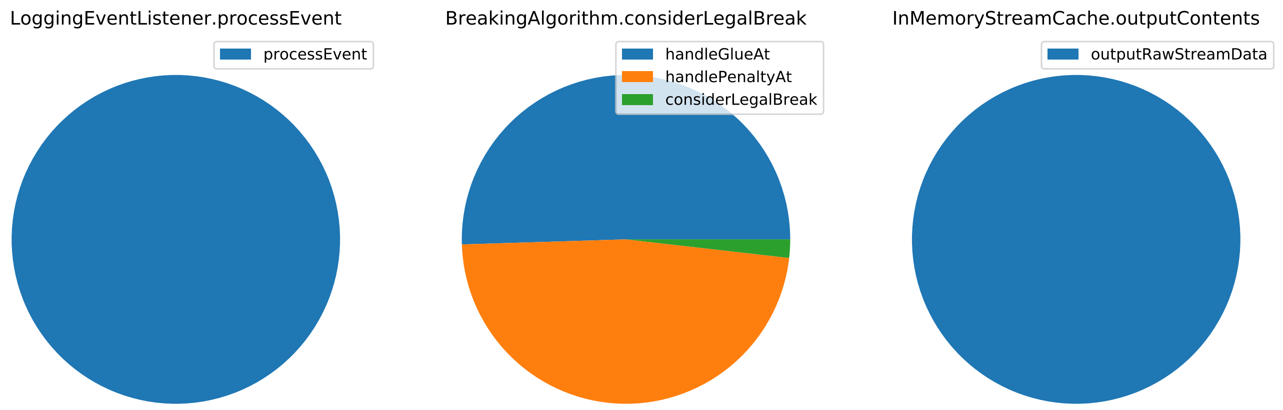

**Figure 12.** 1-CFA for `fop`

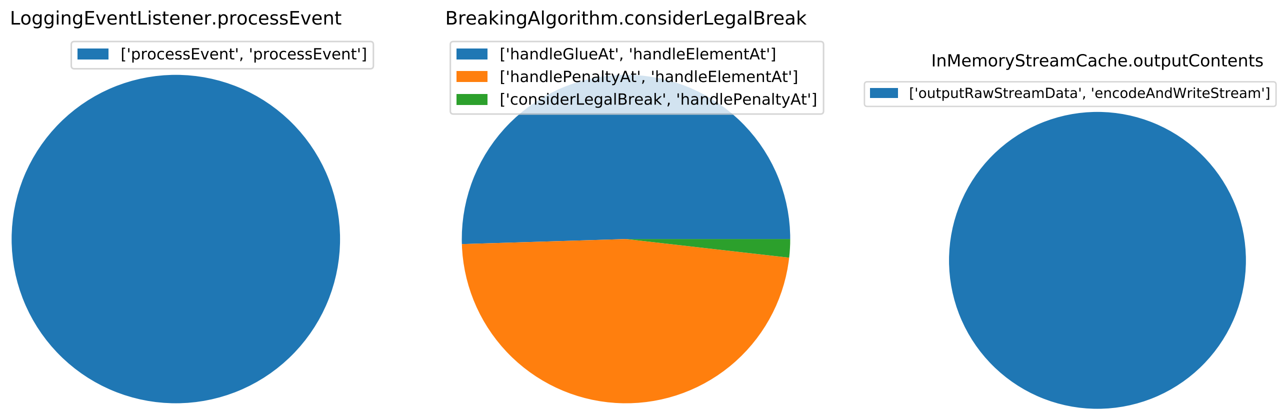

**Figure 13.** 2-CFA for `fop`

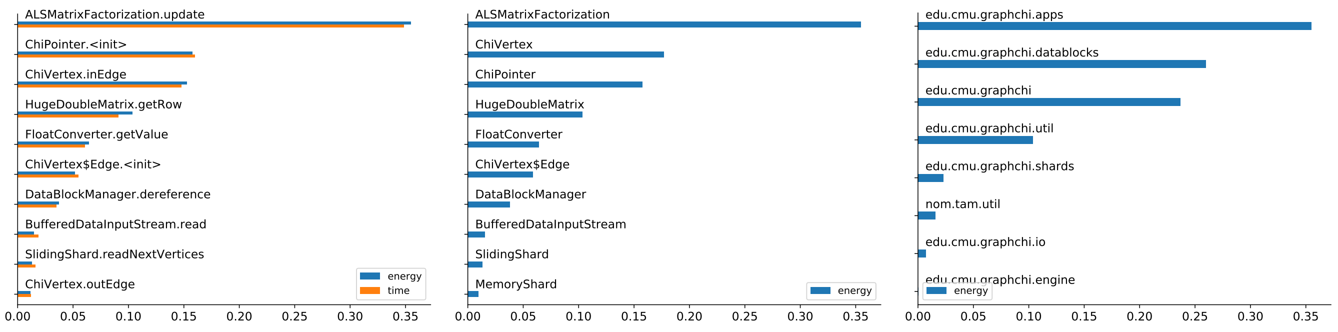

**Figure 14.** Top 10 Energy-Consuming Abstractions for `graphchi`

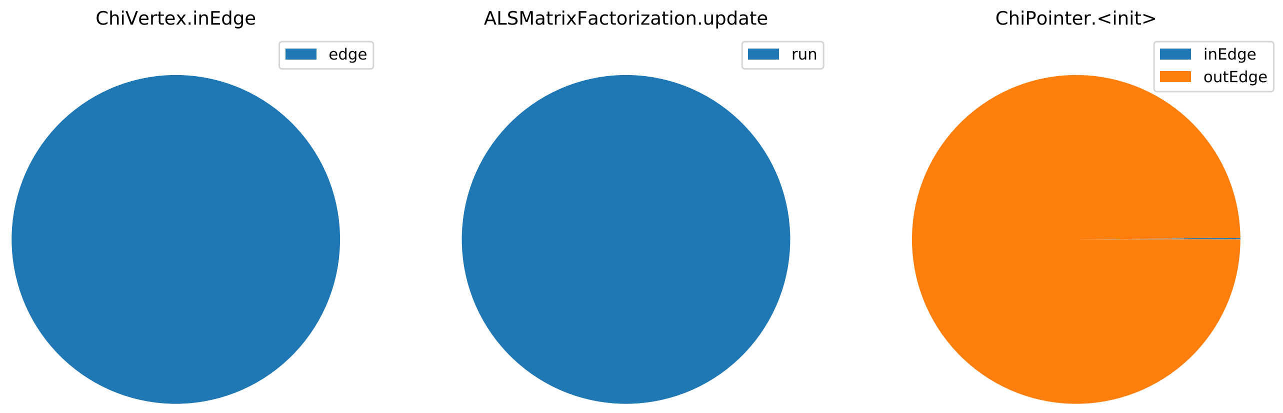

**Figure 15.** 1-CFA for `graphchi`

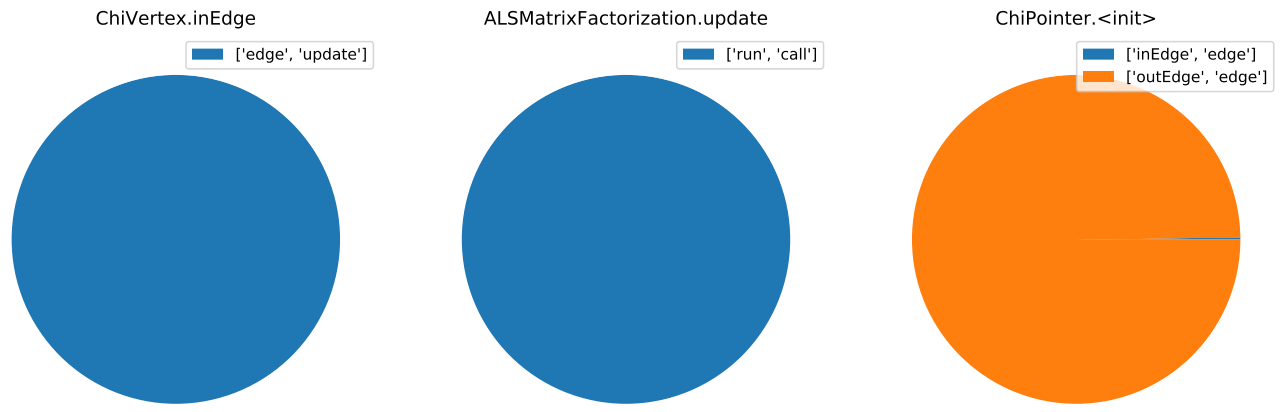

**Figure 16.** 2-CFA for `graphchi`

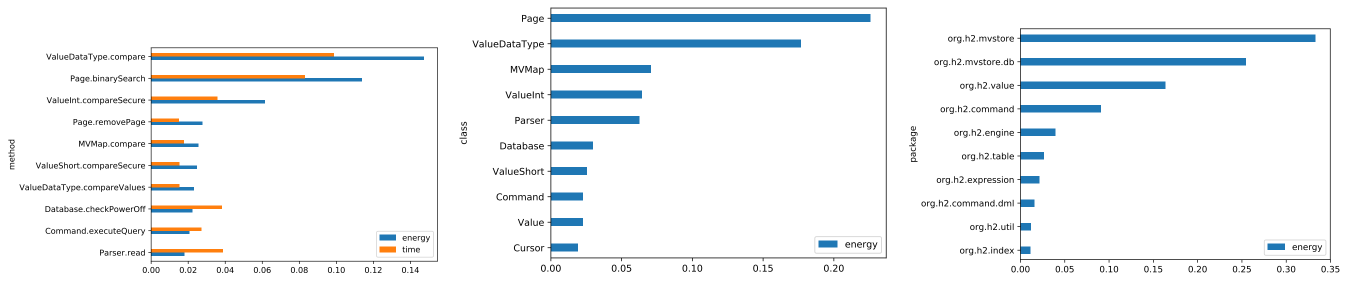

**Figure 17.** Top 10 Energy-Consuming Abstractions for h2

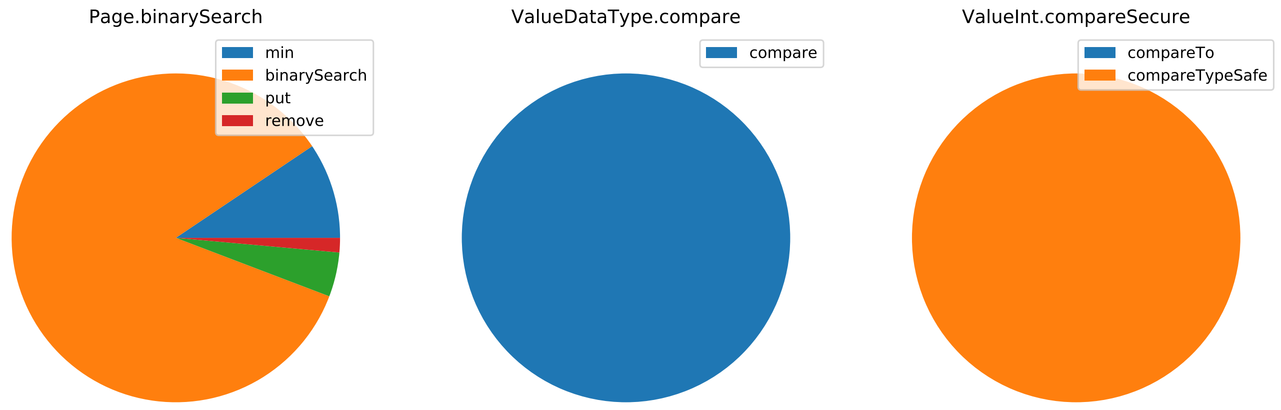

**Figure 18.** 1-CFA for h2

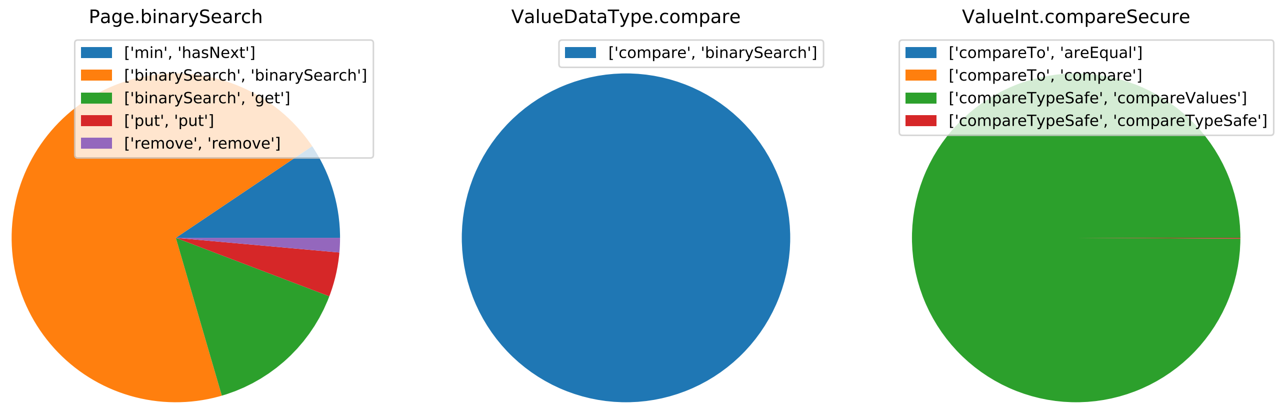

**Figure 19.** 2-CFA for h2

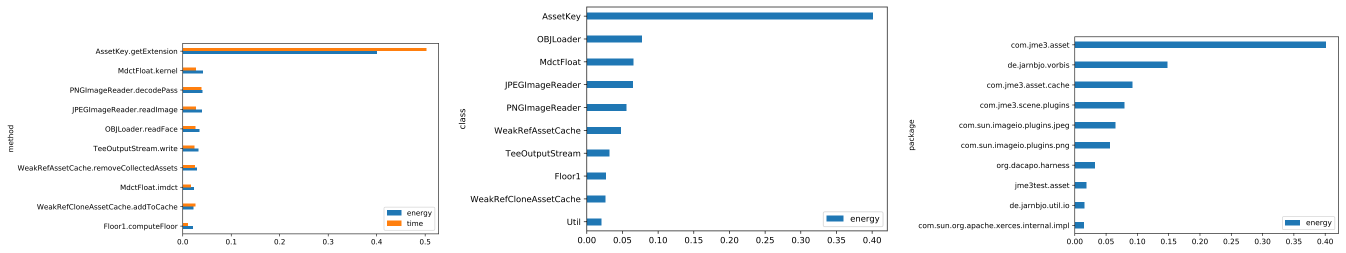

**Figure 20.** Top 10 Energy-Consuming Abstractions for `jme`

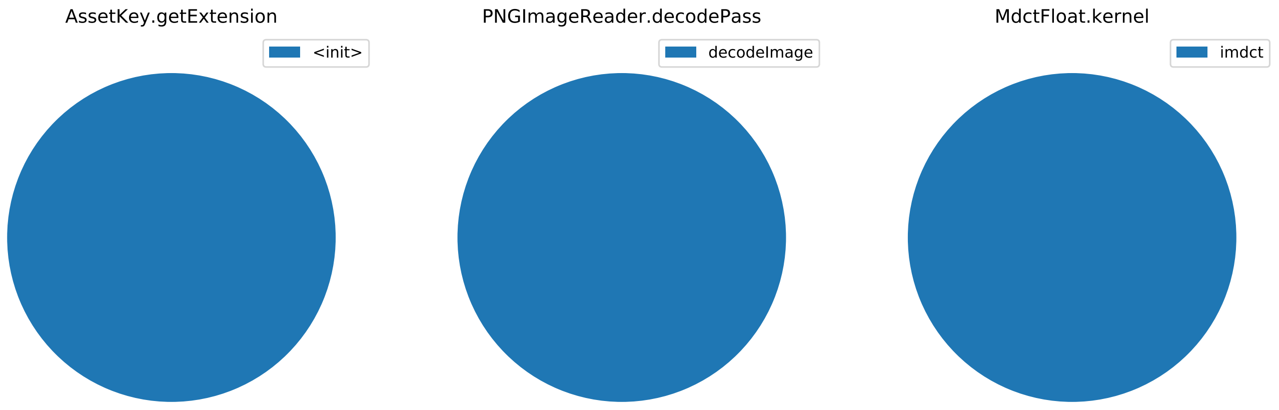

**Figure 21.** 1-CFA for `jme`

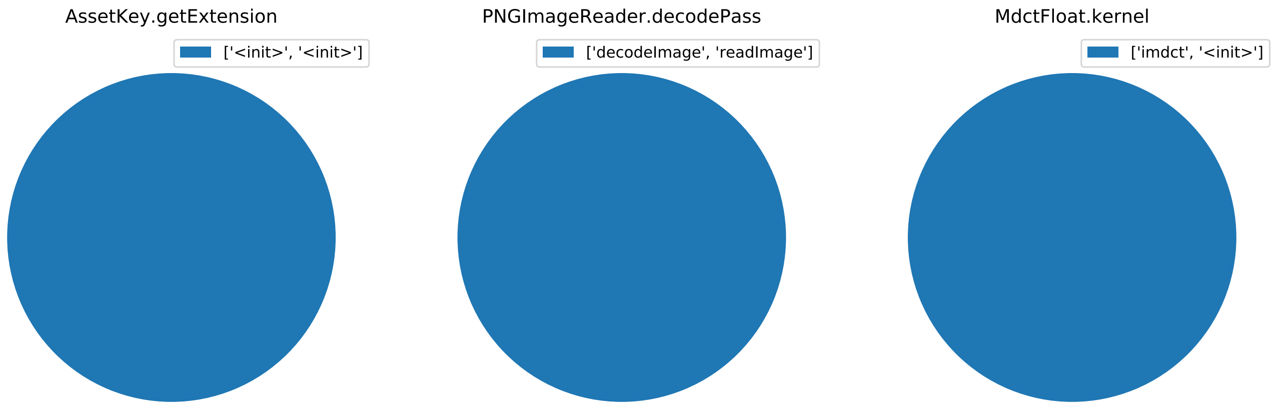

**Figure 22.** 2-CFA for `jme`

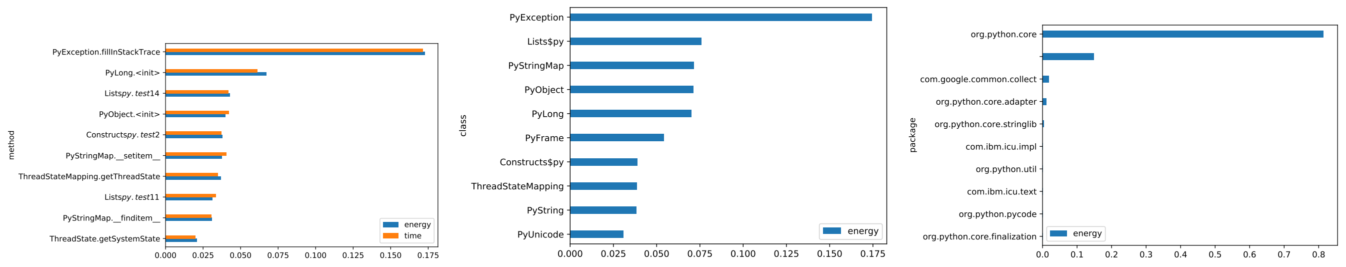

**Figure 23.** Top 10 Energy-Consuming Abstractions for `jython`

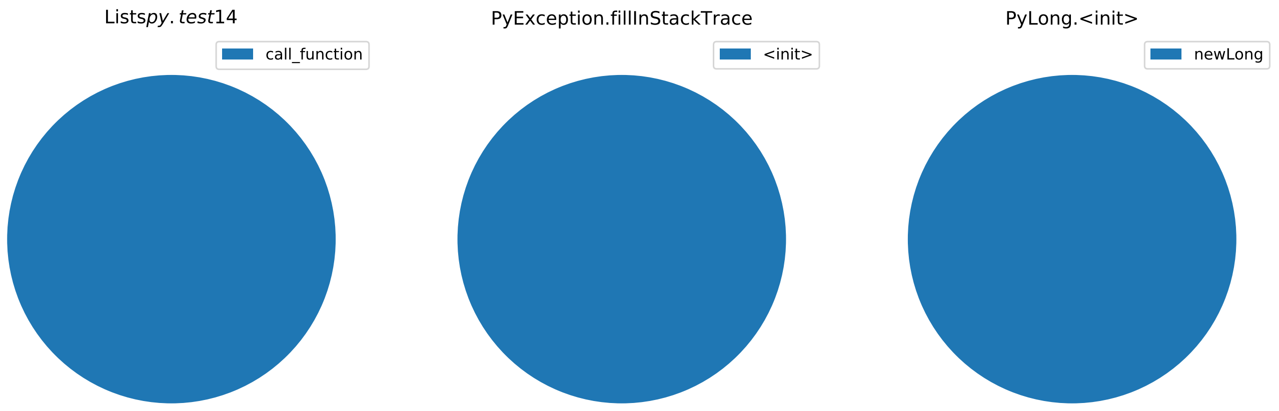

**Figure 24.** 1-CFA for `jython`

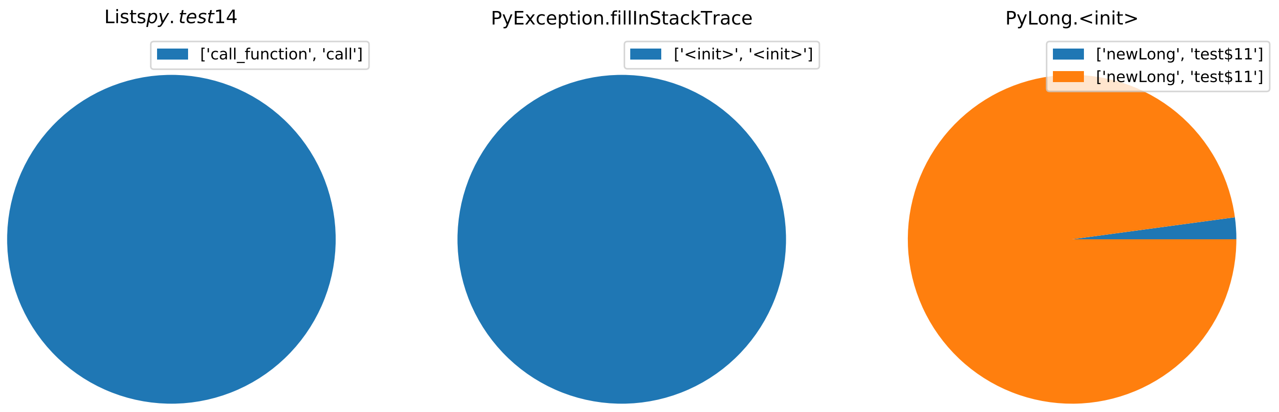

**Figure 25.** 2-CFA for `jython`

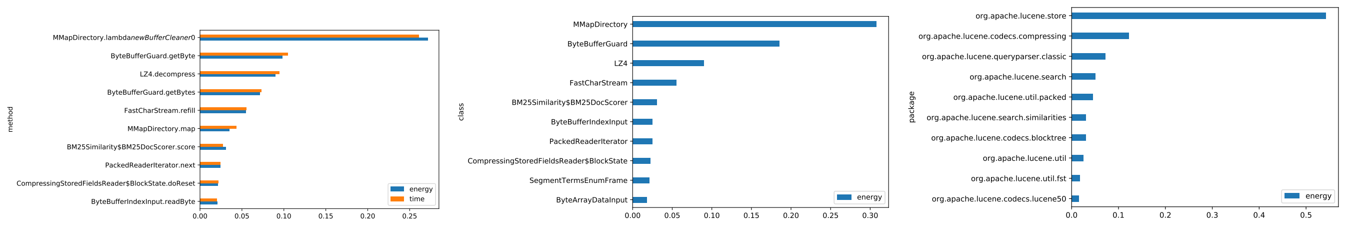

**Figure 26.** Top 10 Energy-Consuming Abstractions for `lusearch`

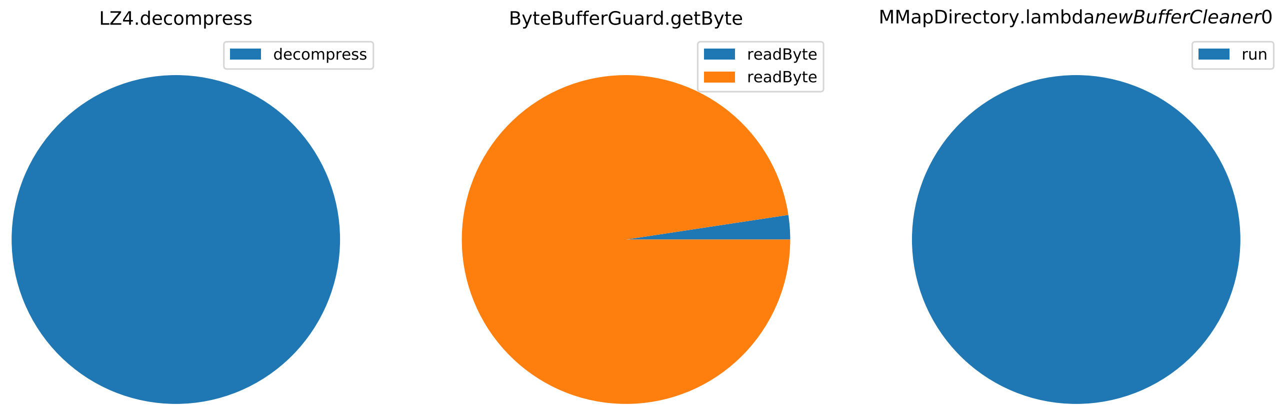

**Figure 27.** 1-CFA for `lusearch`

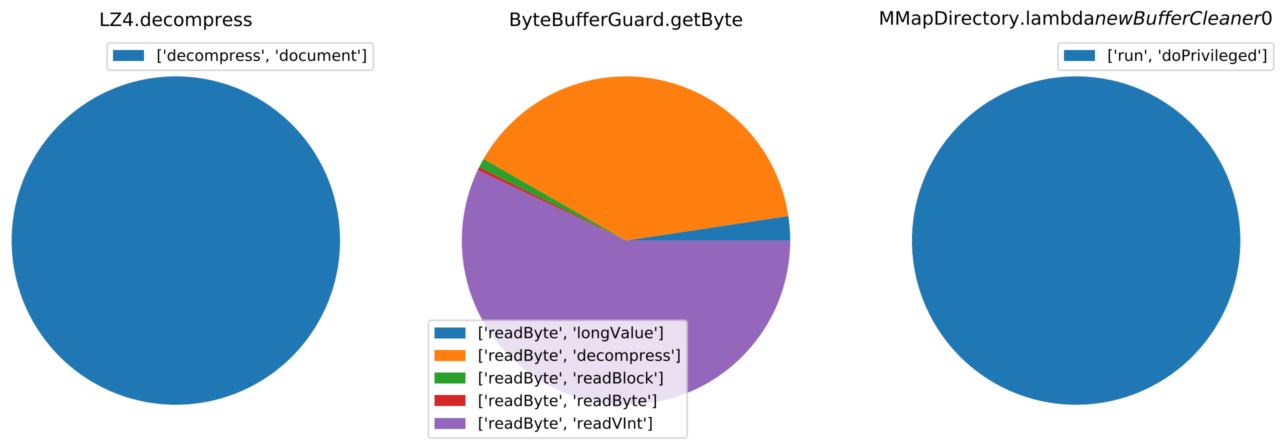

**Figure 28.** 2-CFA for `lusearch`

images/new_plots/plots_temp/deep_method_ranking.pdf images/new_plots/plots_temp/deep_class_ranking.pdf images/new_plots/plots_temp/deep_package_ran

**Figure 29.** Top 10 Energy-Consuming Abstractions for `plots`$_t$*emp*

images/new_plots/plots_temp/deep_method_1cfa_0_context.pdf images/new_plots/plots_temp/deep_method_1cfa_1_context.pdf images/new_plots/plots_temp/deep_method_1cfa

**Figure 30.** 1-CFA for `plots`$_t$*emp*

images/new_plots/plots_temp/deep_method_2cfa_0_context.pdf images/new_plots/plots_temp/deep_method_2cfa_1_context.pdf images/new_plots/plots_temp/deep_method_2cfa

**Figure 31.** 2-CFA for `plots`$_t$*emp*

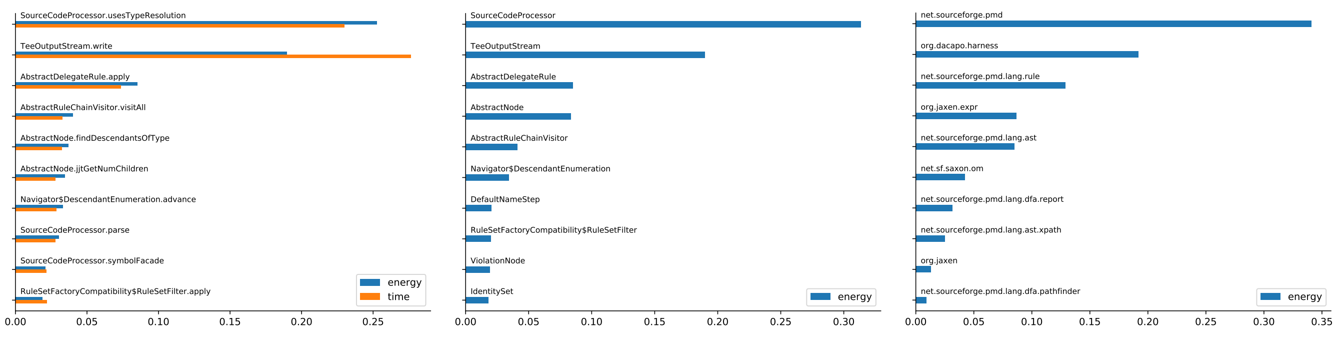

**Figure 32.** Top 10 Energy-Consuming Abstractions for pmd

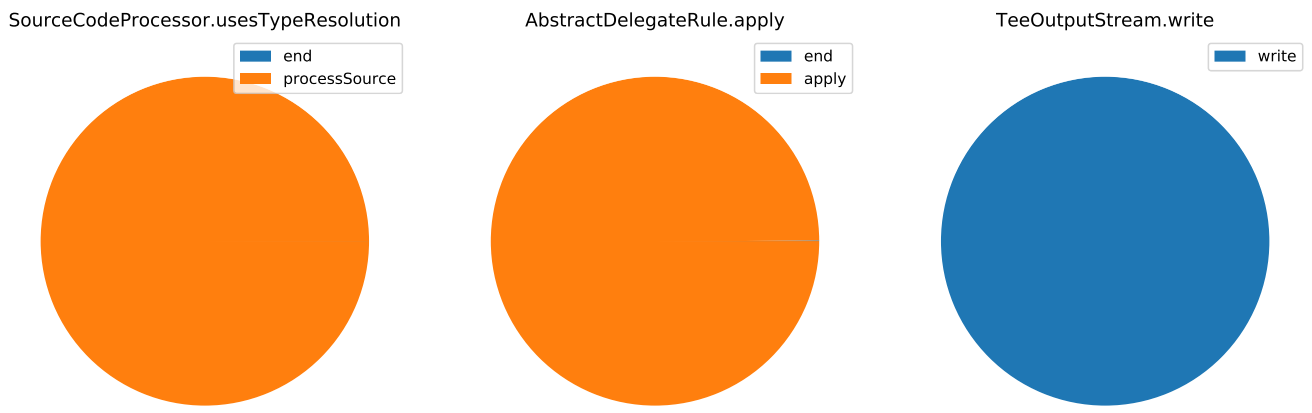

**Figure 33.** 1-CFA for pmd

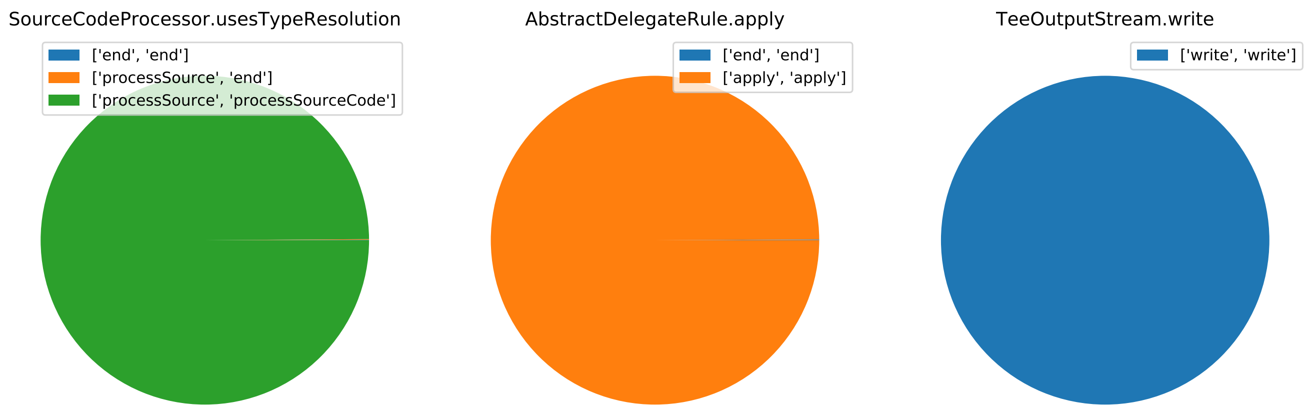

**Figure 34.** 2-CFA for pmd

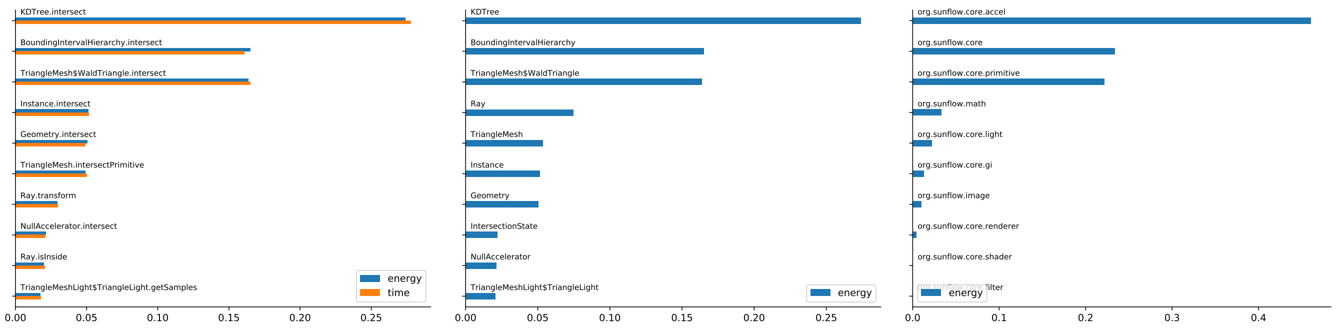

**Figure 35.** Top 10 Energy-Consuming Abstractions for `sunflow`

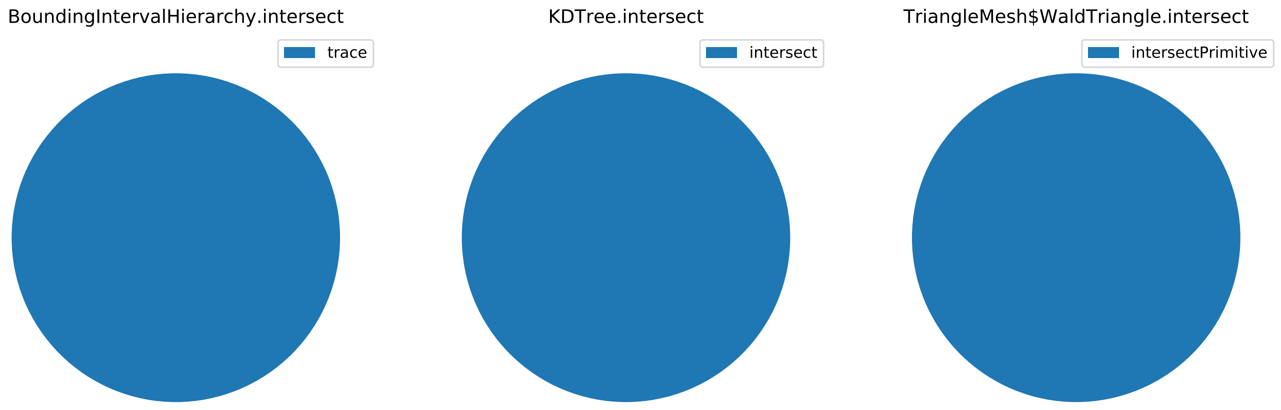

**Figure 36.** 1-CFA for `sunflow`

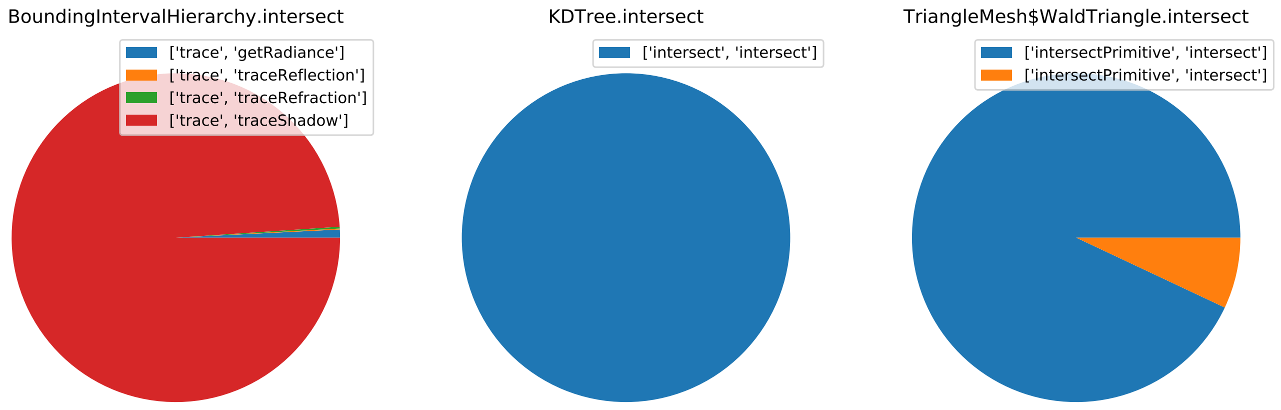

**Figure 37.** 2-CFA for `sunflow`

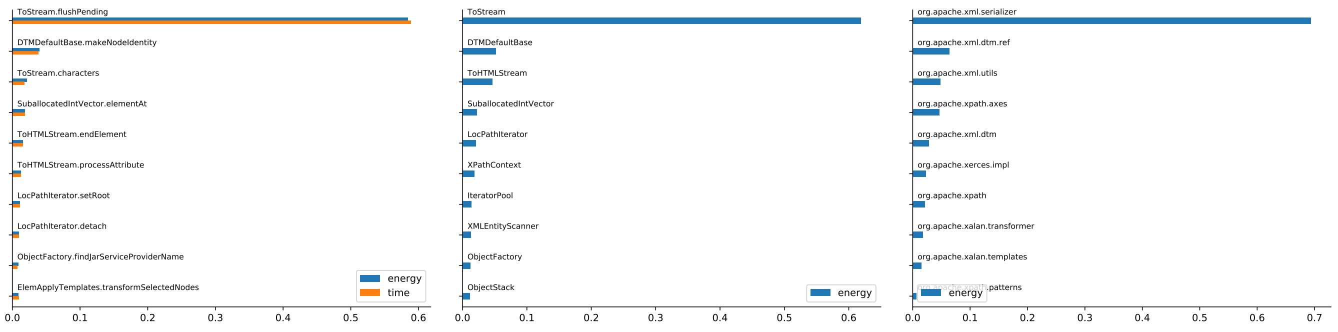

**Figure 38.** Top 10 Energy-Consuming Abstractions for `xalan`

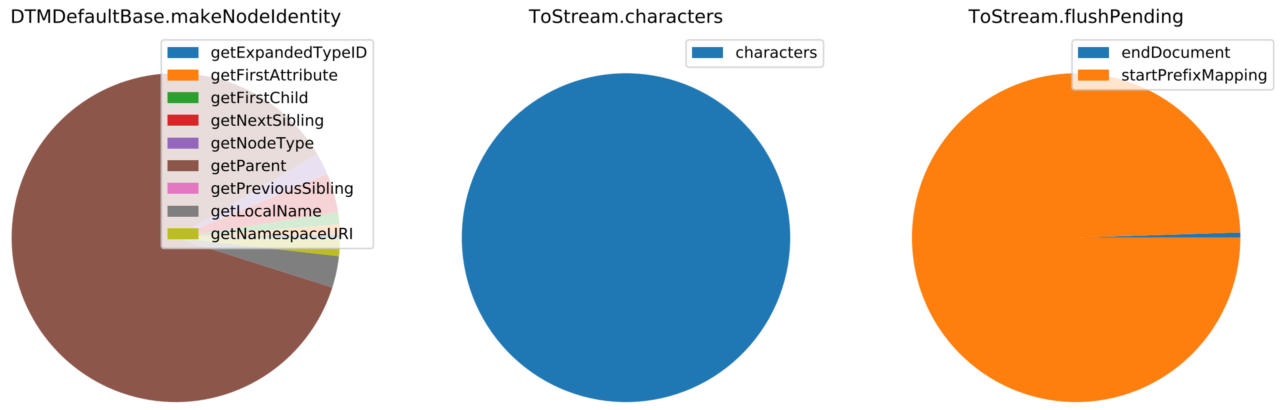

**Figure 39.** 1-CFA for `xalan`

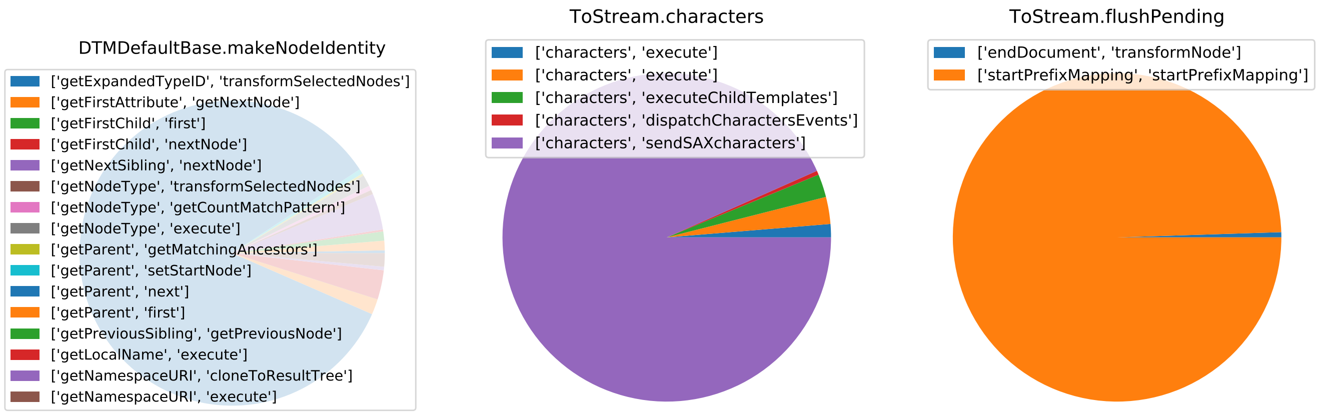

**Figure 40.** 2-CFA for `xalan`

## References

[1] Binkert, N., Beckmann, B., Black, G., Reinhardt, S. K., Saidi, A., Basu, A., Hestness, J., Hower, D. R., Krishna, T., Sardashti, S., Sen, R., Sewell, K., Shoaib, M., Vaish, N., Hill, M. D., and Wood, D. A. The gem5 simulator. *SIGARCH Comput. Archit. News 39*, 2 (Aug. 2011), 1–7.

[2] Li, S., Ahn, J. H., Strong, R. D., Brockman, J. B., Tullsen, D. M., and Jouppi, N. P. Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture* (New York, NY, USA, 2009), MICRO 42, ACM, pp. 469–480.

[3] Sandberg, A., Nikoleris, N., Carlson, T. E., Hagersten, E., Kaxiras, S., and Black-Schaffer, D. Full speed ahead: Detailed architectural simulation at near-native speed. In *2015 IEEE International Symposium on Workload Characterization* (Oct 2015), pp. 183–192.