

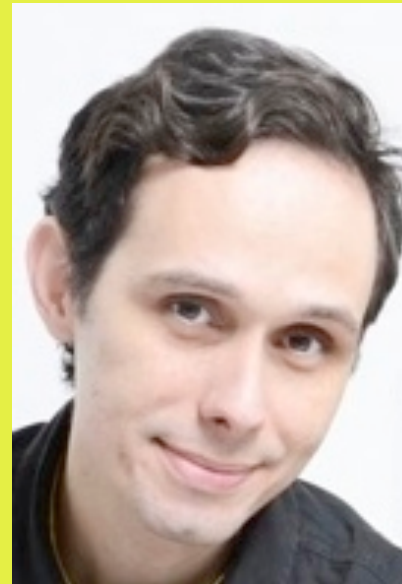
Characterizing the Energy Efficiency of Java's Thread-Safe Collections in a Multi-Core Environment



Gustavo Pinto



Kenan Liu



Fernando Castor



David Liu



Motivation (1/3)



- First, **energy consumption** is a concern for unwired devices and also for data centers
- Second, there is a large body of work in hardware/architecture, OS, runtime systems
- However, little is known about the application level



Motivation (2/3)



- First, **multicore** CPUs are ubiquitous
- Second, more cores used → more power dissipated
- However, little is known about the energy-efficiency of multicore programs



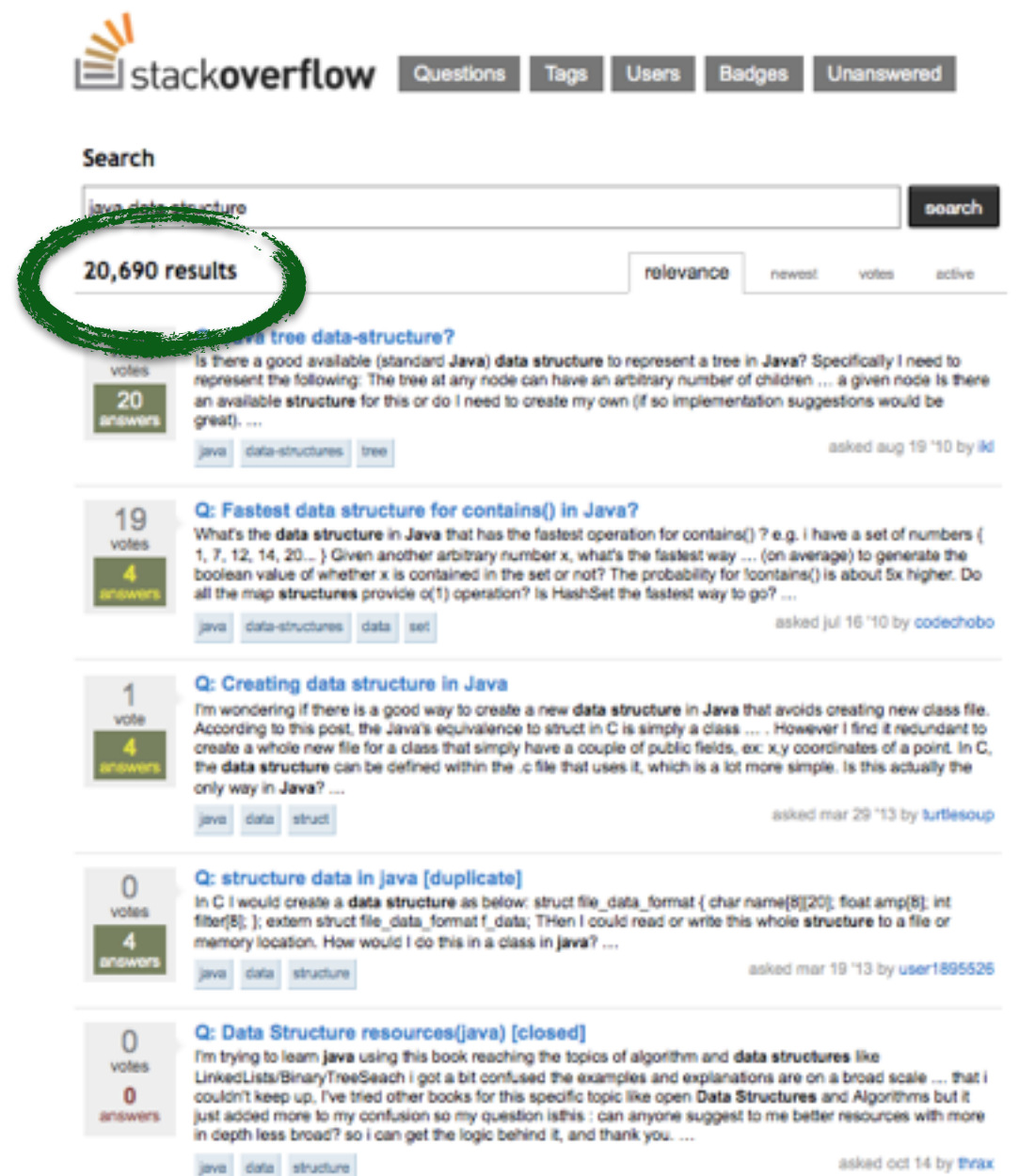
Motivation (3/3)

- Data structures are the fundamentals of computer programming

Bad programmers worry about the code.
Good programmers worry about **data structures** and their relationships.



Linus Torvalds

A screenshot of the Stack Overflow website showing search results for the query 'java data structure'. The search bar at the top contains the text 'java data structure' and the search button is labeled 'search'. Below the search bar, the number '20,690 results' is displayed and circled in green. The results list several questions related to Java data structures, including 'Is there a good available (standard Java) data structure to represent a tree in Java?', 'Q: Fastest data structure for contains() in Java?', 'Q: Creating data structure in Java', 'Q: structure data in java [duplicate]', and 'Q: Data Structure resources(java) [closed]'. Each question entry shows the number of votes, the number of answers, and the tags associated with the question.

Benchmarks

16 Java Collections

List
ArrayList
Vector
Collections.synchronizedList()
CopyOnWriteArrayList

Set
LinkedHashSet
Collections.synchronizedSet()
CopyOnWriteArraySet
ConcurrentSkipListSet
ConcurrentHashSet
ConcurrentHashSetV8

Map
LinkedHashMap
Hashtable
Collections.synchronizedMap()
ConcurrentSkipListMap
ConcurrentHashMap
ConcurrentHashMapV8

16 Java Collections

List
ArrayList
Vector
Collections.synchronizedList()
CopyOnWriteArrayList

Set
LinkedHashSet
Collections.synchronizedSet()
CopyOnWriteArraySet
ConcurrentSkipListSet
ConcurrentHashSet
ConcurrentHashSetV8

Map
LinkedHashMap
Hashtable
Collections.synchronizedMap()
ConcurrentSkipListMap
ConcurrentHashMap
ConcurrentHashMapV8

 Non thread-safe

 Thread-safe

16 Java Collections

List
ArrayList
Vector
Collections.synchronizedList()
CopyOnWriteArrayList

Set
LinkedHashSet
Collections.synchronizedSet()
CopyOnWriteArraySet
ConcurrentSkipListSet
ConcurrentHashSet
ConcurrentHashSetV8

Map
LinkedHashMap
Hashtable
Collections.synchronizedMap()
ConcurrentSkipListMap
ConcurrentHashMap
ConcurrentHashMapV8

x 3 Operations

Traversal	Insertion	Removal
-----------	-----------	---------

2 Real-world Benchmarks

Tomcat

- > A web server
- > More than 170K lines of Java code
- > More than 300 Hashtables

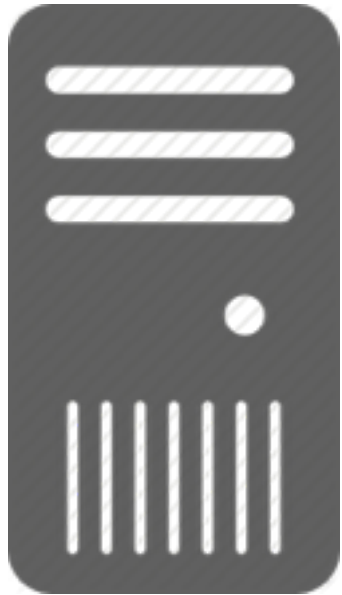


Xalan

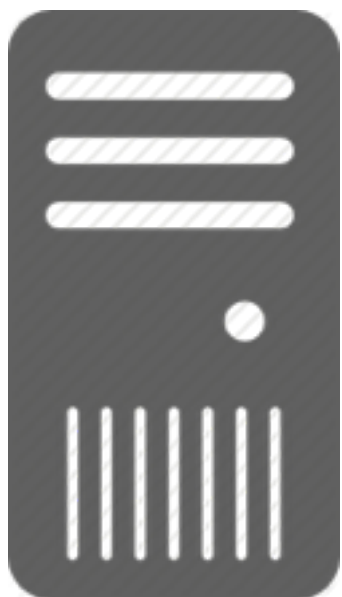
- > Parses XML in HTML documents
- > More than 188K lines of Java code
- > More than 140 Hashtables



Experimental Environments

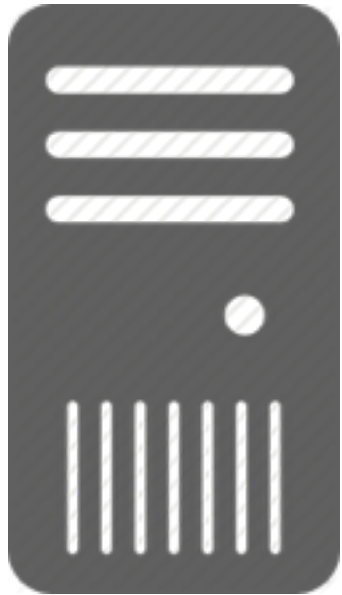


AMD CPU: A 2×16-core, running Debian, 2.4 GHz, 64GB of memory, JDK version 1.7.0 11, build 21.

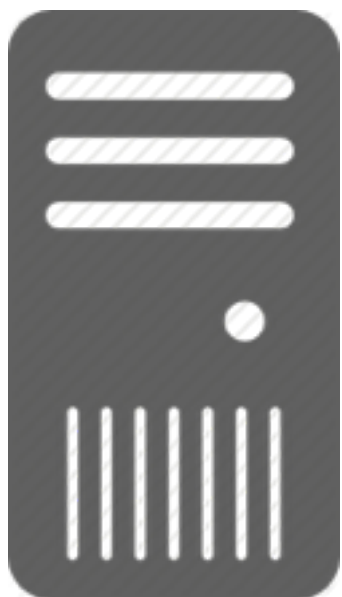


Intel CPU: A 2×8-core (32-cores w/ hyper-threading), running Debian, 2.60GHz, with 64GB of memory, JDK version 1.7.0 71, build 14.

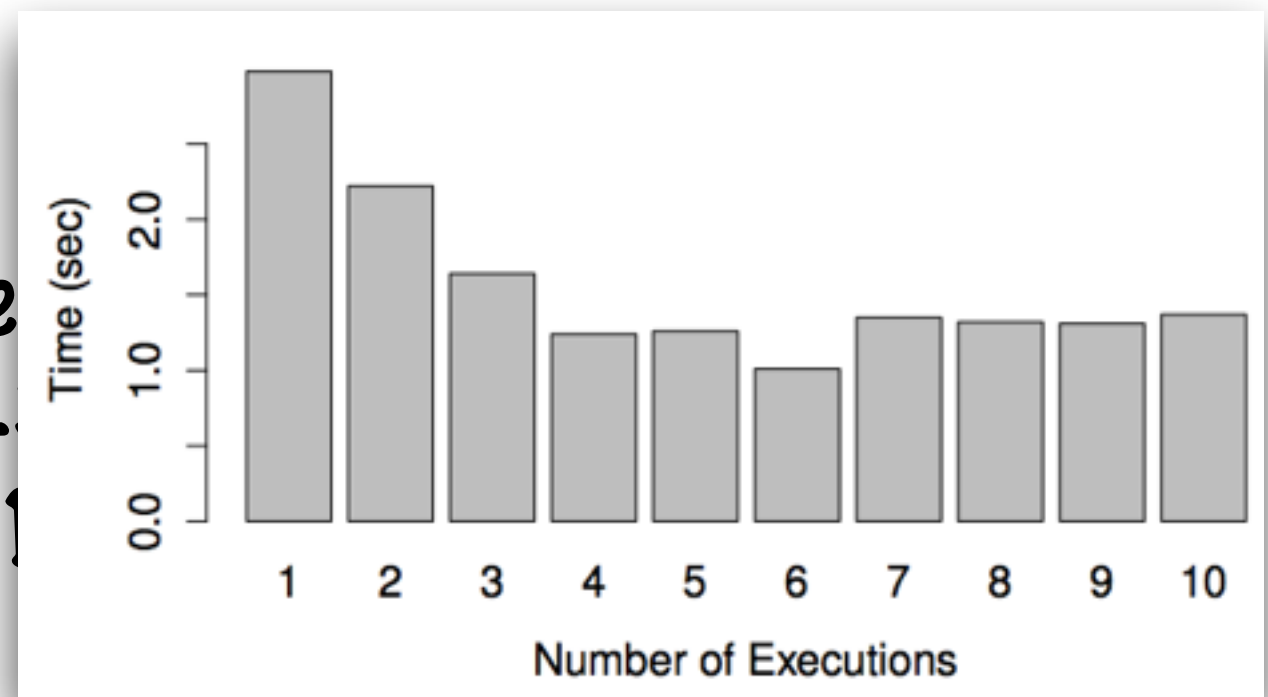
Experimental Environments



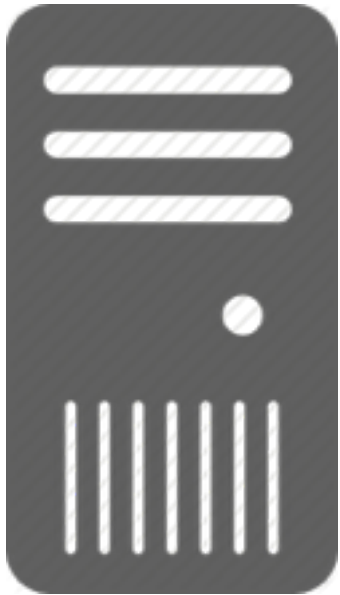
AMD CPU: A 2×16 -core, running Debian, 2.4 GHz, 64GB of memory, JDK version 1.7.0 11, build 21.



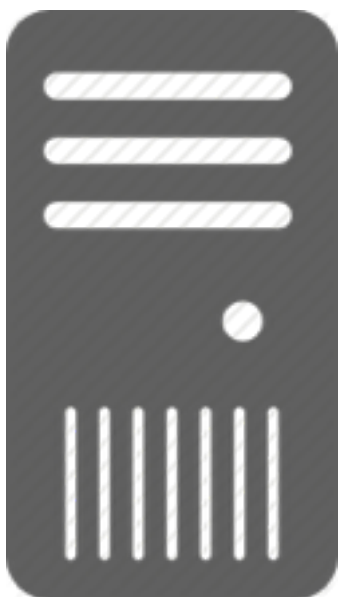
Intel CPU: A 2×8 -core running Debian, 2.60GHz, JDK version 1.7.0 71, build 1.



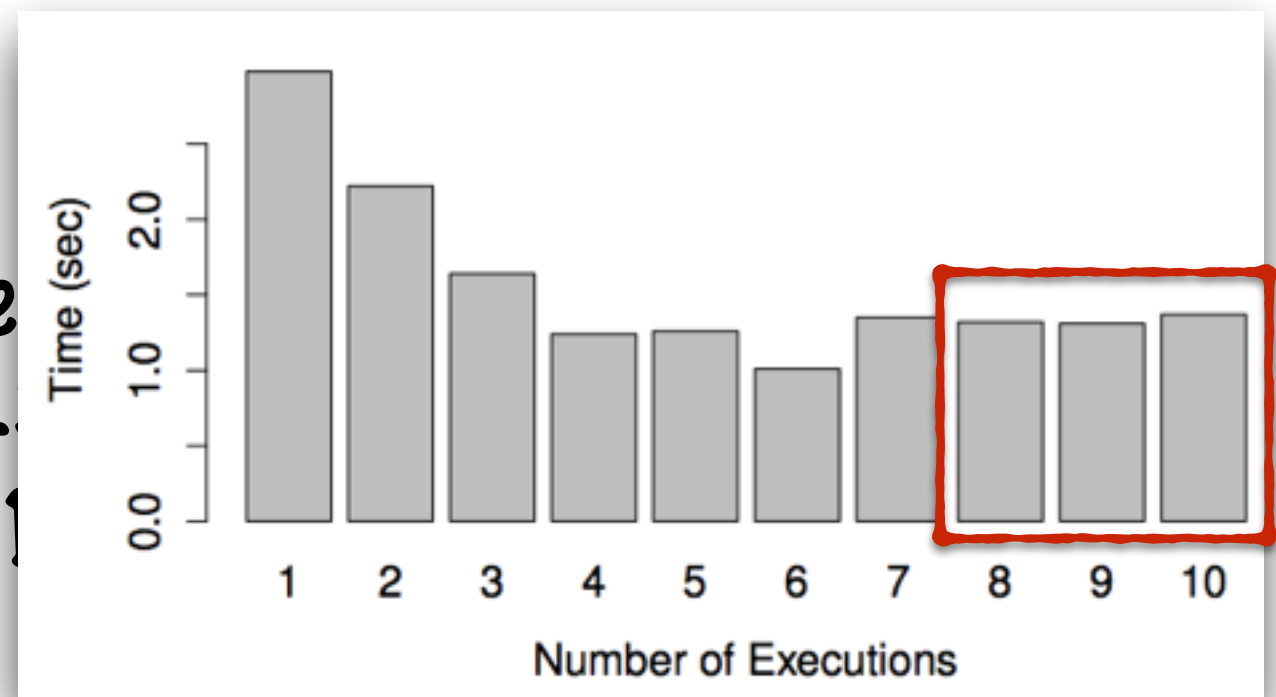
Experimental Environments

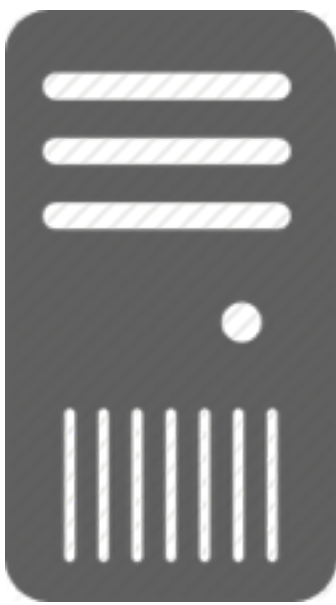


AMD CPU: A 2×16-core, running Debian, 2.4 GHz, 64GB of memory, JDK version 1.7.0 11, build 21.



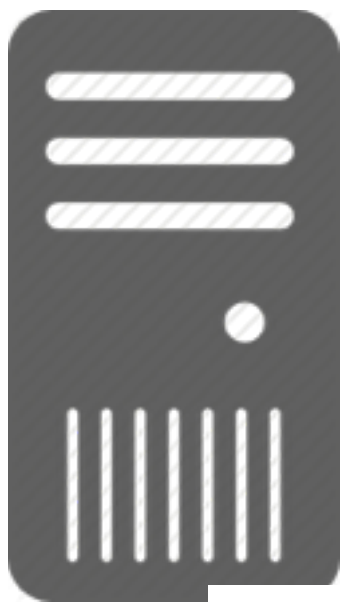
Intel CPU: A 2×8-core running Debian, 2.60GHz, JDK version 1.7.0 71, build 1.



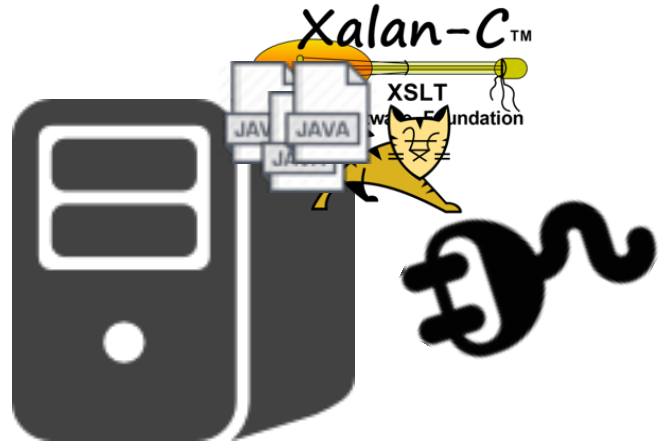


AMD CPU: A 2×16-core, running Debian, 2.4 GHz, 64GB of memory, JDK version 1.7.0 11, build 21.





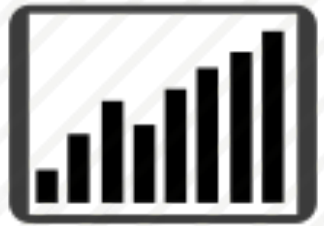
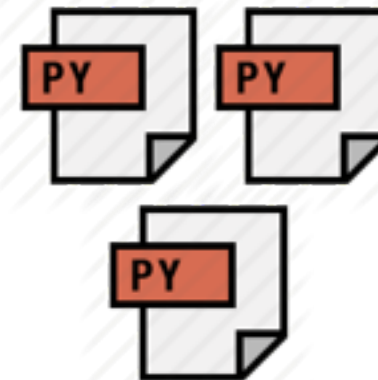
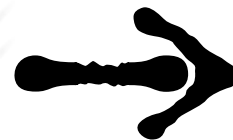
AMD CPU: A 2×16-core, running Debian, 2.4 GHz, 64GB of memory, JDK version 1.7.0 11, build 21.

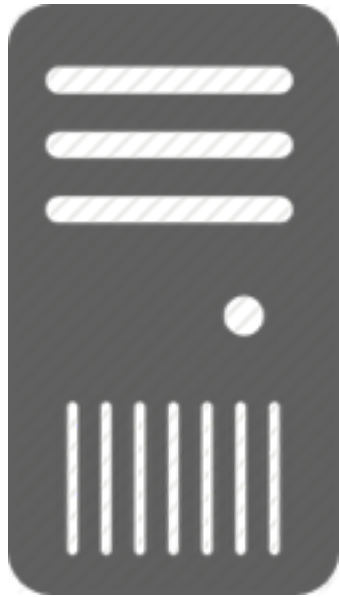


AMD CPU



NI LabVIEW
(100 samples/sec)





Intel CPU: A 2×8-core (32-cores w/ hyper-threading), running Debian, 2.60GHz, with 64GB of memory, JDK version 1.7.0 71, build 14.

jRAPL – A framework for profiling energy consumption of Java programs

What is jRAPL?

jRAPL is framework for profiling Java programs running on CPUs with Running Average Power Limit (RAPL) support.

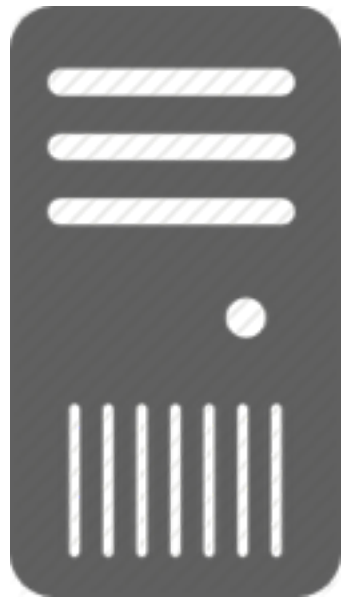
But, what is RAPL?

RAPL is a set of low-level interfaces with the ability to monitor, control, and get notifications of energy and power consumption data of different hardware levels.

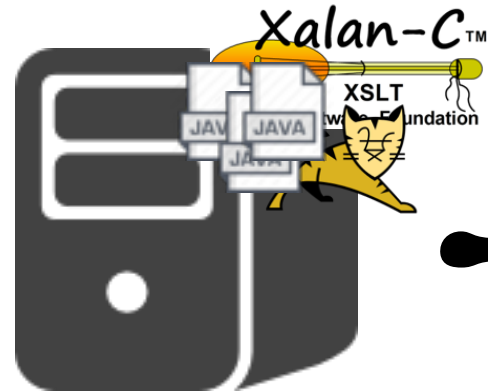
Originally designed by Intel for enabling chip-level power management, RAPL is widely supported in today's Intel architectures, including Xeon server-level CPUs and the popular i5 and i7.

```
double beginning = EnergyCheck.statCheck();  
doWork();  
double end = EnergyCheck.statCheck();
```

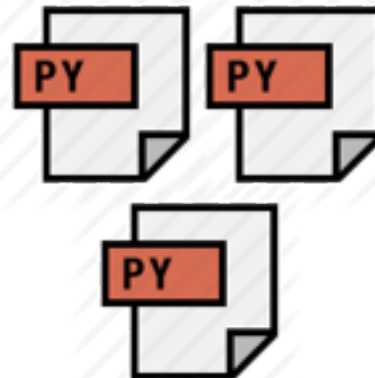
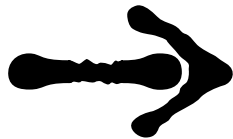
K. Liu, G. Pinto, and D. Liu, "Data-oriented characterization of application-level energy optimization," in Proceedings of the 18th International Conference on Fundamental Approaches to Software Engineering, ser. FASE'15, 2015.



Intel CPU: A 2×8-core (32-cores w/ hyper-threading), running Debian, 2.60GHz, with 64GB of memory, JDK version 1.7.0 71, build 14.



Intel CPU
(w/ jRAPL)



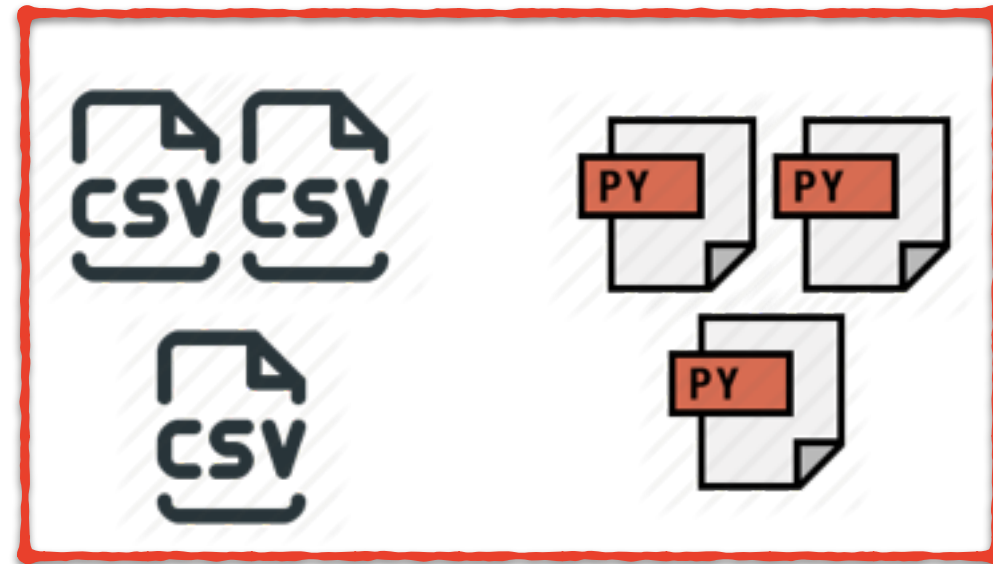
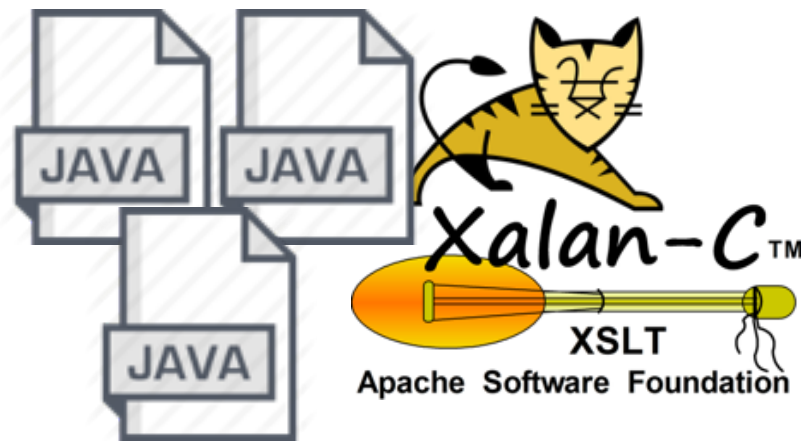
The Artifacts

Benchmarks

Raw Data

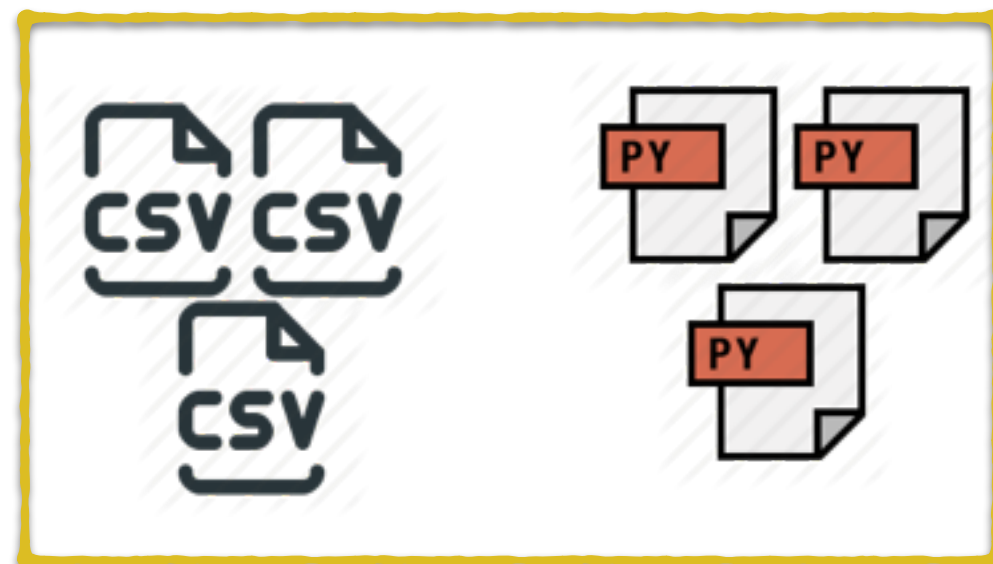
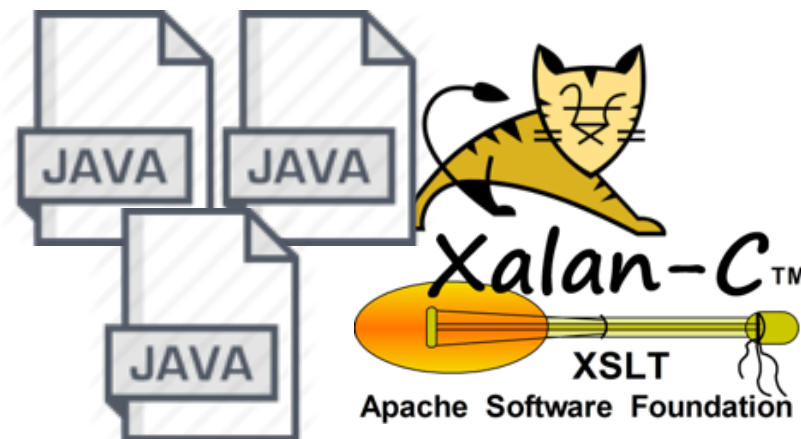
Scripts

**Intel CPU
(w/ jRAPL)**



.zip

AMD CPU



.zip

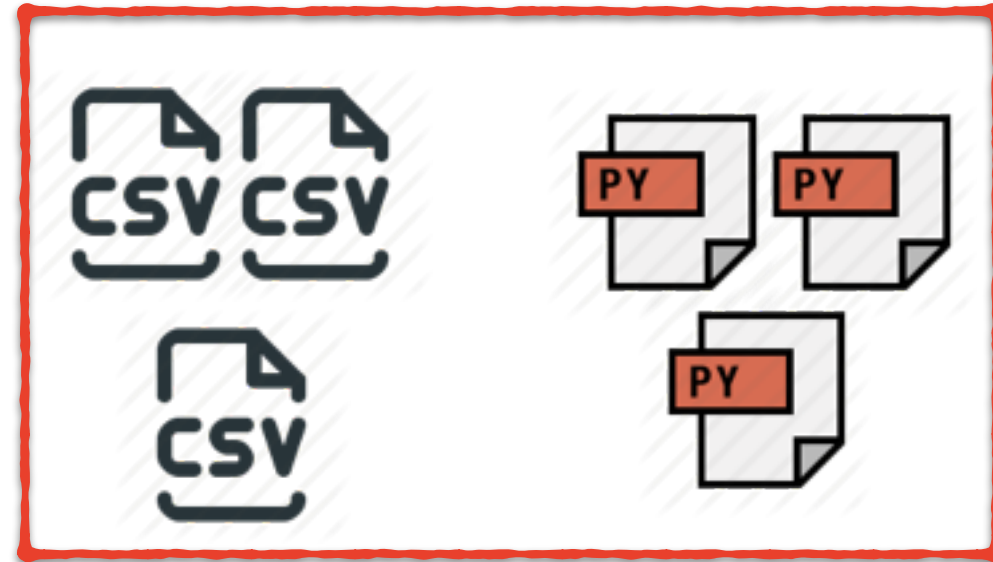
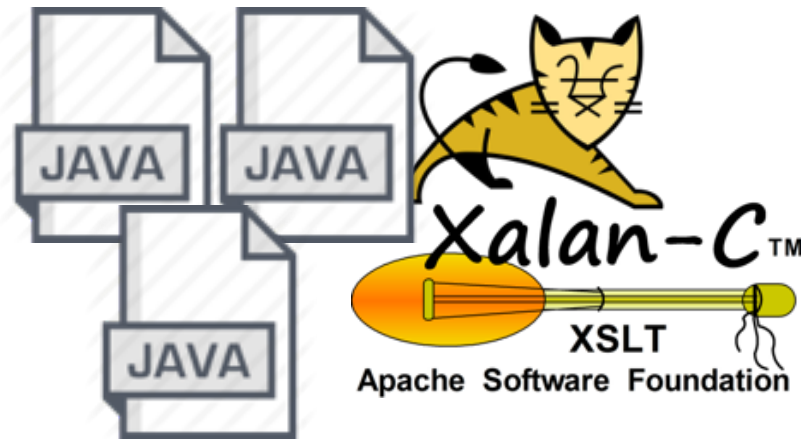
The Artifacts

Benchmarks

Raw Data

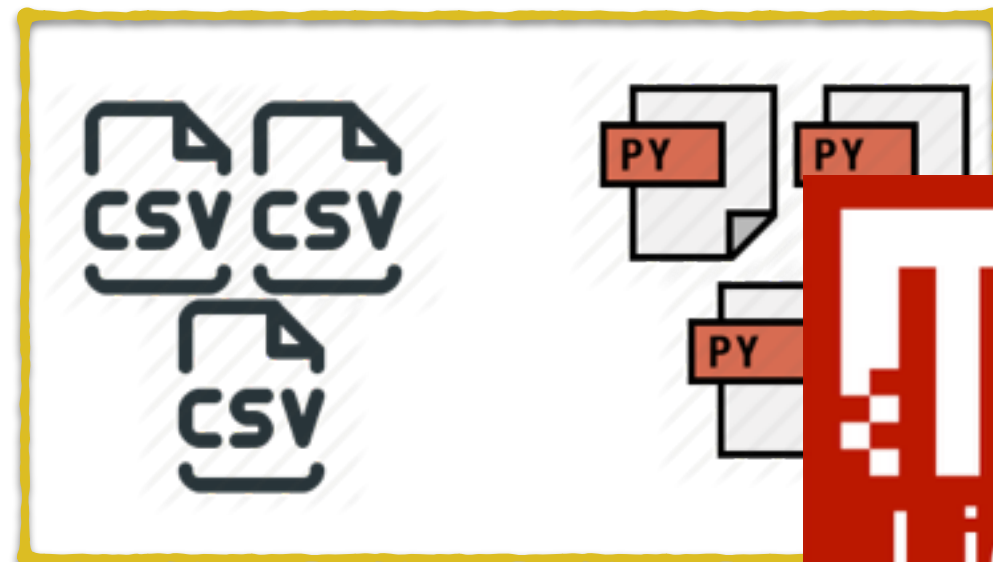
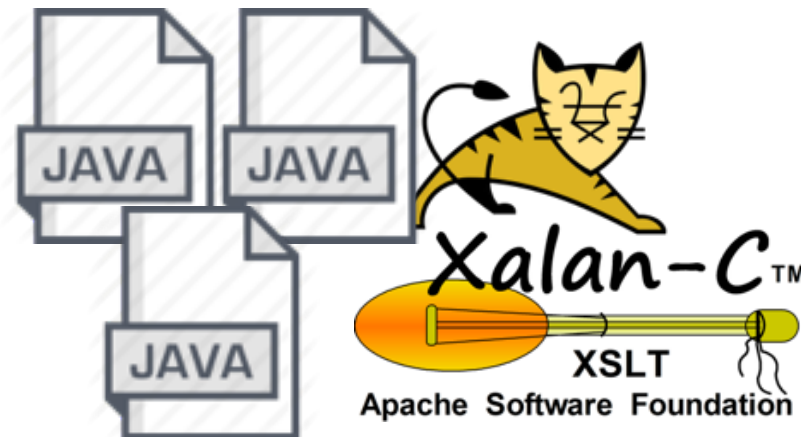
Scripts

Intel CPU
(w/ jRAPL)



.zip

AMD CPU



More details?

Artifacts for “A Comprehensive Study on the Energy Efficiency of Java’s Thread-Safe Collections”

Gustavo Pinto¹ Kenan Liu² Fernando Castor³ Yu David Liu²
¹IFPA, Brazil ²SUNY Binghamton, USA ³UFPE, Brazil

I. DESCRIPTION

A. Contact

Gustavo Pinto <gustavo.pinto@ifpa.edu.br>
Kenan Liu <kl20@binghamton.edu>
Fernando Castor <castor@cis.ufpe.br>
Yu David Liu <david@cs.binghamton.edu>

B. License

This artifact is licensed under MIT license.

C. Summary

The artifacts can be found in the following website: <https://www.dropbox.com/sh/1c0twfndmgk7m/AAAs35TepfAMQsX2aSeOyZ8Ja?dl=0>

This artifact contains the source code used in the experiments, the JRAPL¹ implementation (which helped us to measure energy consumption of the source code used), the raw energy data generated by JRAPL, as well as the figures presented in the paper, and the source code used to plot these figures using the raw energy data. Details for using JRAPL can be found at the companion website.

II. SCORECARD

A. Insightful

- *Timely (i.e., addresses a problem that is most current and most pressing)?* This artifact tackles a problem that is becoming increasingly recurring not only in software development community, but also in the agenda of software engineering researchers: energy consumption. In particular, this artifact focus on the usage of different data structures in a multi-core environment. On the one hand, data structures are one of the most important building blocks of computer programming. On the other hand, not only high-end servers but also desktop machines, smartphones, and tablets need concurrent programs to make best use of their multi-core hardware. This artifact makes it easier to reproduce the usage of 16 different data structures two different multicore machines.
- *Makes researchers “smarter” in some way (e.g., identifies and fills some significant gap in prior work)?* This artifact makes it easier to reproduce the usage of 16 different data

structures two different multicore machines. While one can reproduce the artifact by reading the research paper, the artifact greatly reduces the barriers for conducting system-related research focused on data structures. Also, the data structure source code is thoroughly integrated with JRAPL, and the JRAPL output is thoroughly integrated with the plotting scripts. Therefore, after one get acquainted with the data gathering procedure, the artifact becomes straightforward.

B. Useful

- *Serves a useful purpose?* Our artifact is useful because it makes our claims reproducible. It was created to better assist energy-aware researchers and developers. It also helped us to publicize a research tool (JRAPL) that is being used in recent software engineering studies.
- *Serves a purpose that would otherwise be tedious, prolonged, awkward, or impossible?* With our artifacts (tooling, raw energy data, and plotting scripts), one can easily not only reproduce the main results of our study, but also (1) reuse the artifacts in different energy-aware problems (e.g., with our tool), or (2) automate other system-related studies (e.g., with our scripts). When conducted manually, such activities are time consuming, tedious, and error prone.
- *Cost-effective?* There is no cost related to the usage of the artifacts.

C. Usable

- *Easy to understand?* Our artifact is well-organized. There is just one Java file for each benchmark used. Also, all Java files employ the same structure, so it is easy to move from one to another. Each Java file generates one CSV file, with the same name but different extension. The same nomenclature also applies for the plotting scripts.
- *Executable?* The artifact is ready to be executed. However, JRAPL needs to be properly installed before used (instructions at the companion website). As a limitation, though, JRAPL only works with Intel machines.

¹<http://kl20.github.io/JRAPL/>

A Comprehensive Study on the Energy Efficiency of Java’s Thread-Safe Collections

Gustavo Pinto¹ Kenan Liu² Fernando Castor³ Yu David Liu²
¹IFPA, Brazil ²SUNY Binghamton, USA ³UFPE, Brazil

Abstract—Java programmers are served with numerous choices of collections, varying from simple sequential ordered lists to sophisticated hashable implementations. These choices are well-known to have different characteristics in terms of performance, scalability, and thread-safety, and most of them are well studied. This paper analyzes an additional dimension, energy efficiency. We conducted an empirical investigation of 16 collection implementations (13 thread-safe, 3 non-thread-safe) grouped under 3 commonly used forms of collections (lists, sets, and mappings). Using micro- and real world-benchmarks (TOMCAT and XALAN), we show that our results are meaningful and impactful. In general, we observed that simple designs decisions can greatly impact energy consumption. In particular, we found that using a newer hashable version can yield a 2.19% energy savings in the micro-benchmarks and up to 17% in the real world-benchmarks, when compared to the old associative implementation. Also, we observed that different implementations of the same thread-safe collection can have widely different energy consumption behaviors. This variation also applies to the different operations that each collection implements, e.g., a collection implementation that performs traversals very efficiently can be more than an order of magnitude less efficient than another implementation of the same collection when it comes to insertions.

I. INTRODUCTION

A question that often arises in software development forums is: “since Java has so many collection implementations, which one should I use?”¹. Answers to this question come in different flavors: these collections serve for different purposes and have different characteristics in terms of performance, scalability and thread-safety. In this study, we consider one additional attribute: energy efficiency. In an era where mobile platforms are prevalent, there is considerable evidence that battery usage is a key factor for evaluating and adopting mobile applications [1]. Energy consumption estimation tools do exist [2], [3], [4], but they do not provide direct guidance on energy optimization, i.e., bridging the gap between understanding where energy is consumed and understanding how the code can be modified in order to reduce energy consumption.

Energy optimization is traditionally addressed by hardware-level (e.g., [5], [6]) and system-level approaches (e.g., [7], [8]). However, it has been gaining momentum in recent years through application-level software engineering techniques (e.g., [9], [10], [11]). The overarching premise of this emerging direction is that the high-level knowledge from software engineers on application design and implementation can make significant impact on energy consumption, as confirmed

by recent empirical studies [12], [13], [14], [15]. Moreover, energy efficiency, similarly to performance and reliability, is a systemic property. Thus, it must be tackled across multiple levels of the system stack. The space for application-level energy optimization, however, is diverse.

In this paper, we elucidate the energy consumption of different Java thread-safe collections running on parallel architectures. This is a critical direction at the junction of data-intensive computing and parallel computing, which deserves more investigation due to at least three reasons:

- Collections are one of the most important building blocks of computer programming. Multiplicity — a collection may hold many pieces of data items — is the norm of their use, and it often contributes to significant memory pressure, and performance problems in general, of modern applications where data are often intensive [16], [17].
- Not only high-end servers but also desktop machines, smartphones, and tablets need concurrent programs to make best use of their multi-core hardware. A CPU with more cores (say 32) often dissipates more power than one with fewer cores (say 1 or 2) [18]. In addition, in mobile platforms such as Android, due to responsiveness requirements, concurrency in the form of asynchronous operations is the norm [19].
- Mainstream programming languages often provide a number of implementations for the same collection and these implementations have potentially different characteristics in terms of energy efficiency [20], [21].

Through extensive experiments conducted in a multi-core environment, we correlate energy behaviors of 13 thread-safe implementations of Java collections, grouped by 3 well-known interfaces (List, Set, and Map), and their turning knobs. Our research is motivated by the following questions:

- RQ1.** Do different implementations of the same collection have different impacts on energy consumption?
- RQ2.** Do different operations in the same implementation of a collection consume energy differently?
- RQ3.** Do collections scale, from an energy consumption perspective, with an increasing number of concurrent threads?
- RQ4.** Do different collection configurations and usages have different impacts on energy consumption?

In order to answer **RQ1** and **RQ2**, we select and analyze the behaviors of three common operations — traversal, insertion and removal — for each collection implementation. To answer

¹<http://stackoverflow.com/search?q=which+data+structure+is+more+java+ic> question

Artifacts Paper

Research Paper

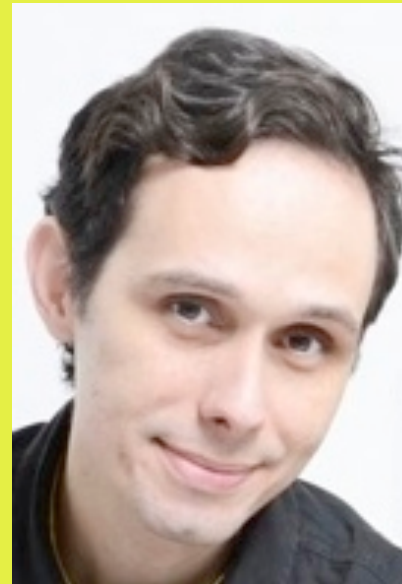
Characterizing the Energy Efficiency of Java's Thread-Safe Collections in a Multi-Core Environment



Gustavo Pinto



Kenan Liu



Fernando Castor



David Liu



jRAPL(Java Running Average Power Limit)

