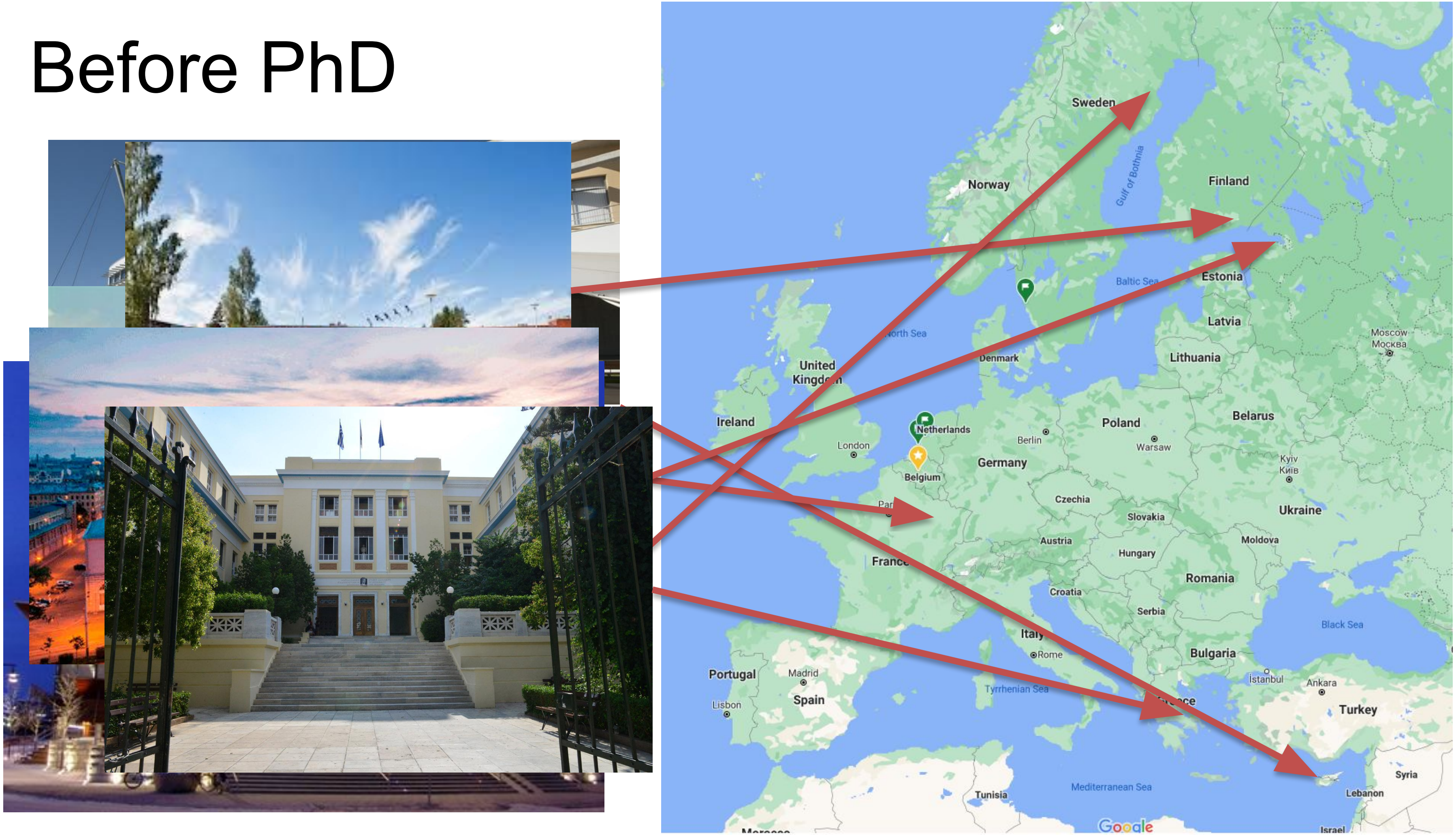
Saving Energy in Software Development by Making the Right Choices

By **Stefanos Georgiou**

4th of March 2022
Sustainable Software Engineering

Before PhD


Sweden
Norway
Finland
Gulf of Bothnia
Estonia
Baltic Sea
Latvia
Lithuania
Moscow
Москва
North Sea
Denmark
United Kingdom
Ireland
Netherlands
Belgium
London
Berlin
Poland
Warsaw
Belarus
Germany
Kyiv
Київ
Czechia
Slovakia
Ukraine
Austria
Hungary
Moldova
France
Romania
Croatia
Serbia
Black Sea
Italy
Rome
Bulgaria
Istanbul
Ankara
Portugal
Madrid
Spain
Lisbon
Tyrrhenian Sea
Turkey
Syria
Mediterranean Sea
Tunisia
Lebanon
Israel
Google

Next to my PhD studies

SingularLogic

Post Doctoral Fellow at Queen’s University

Spare time


33 followers · 14 following · 60
Athens University of Economics and Bu...
Budapest, Hungary
sgeorgiou@aueb.gr
https://stefanos1316.github.io/my_port...
@StefanosGeorgi1
Highlights
Arctic Code Vault Contributor
PRO
Organizations
8,389 contributions in the last year
Contribution settings
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Jan
Feb
Mon
Wed
Fri
@AUEB-BALab
Activity overview
Contributed to
23
NRS


START

Agenda

- Introduction
- Key terms
- Existing Challenges
- Programming Languages
- Inter-Process Communication Technologies
- Security Mechanisms
- Deep Learning
- What are we missing?

Introduction

23%

14%

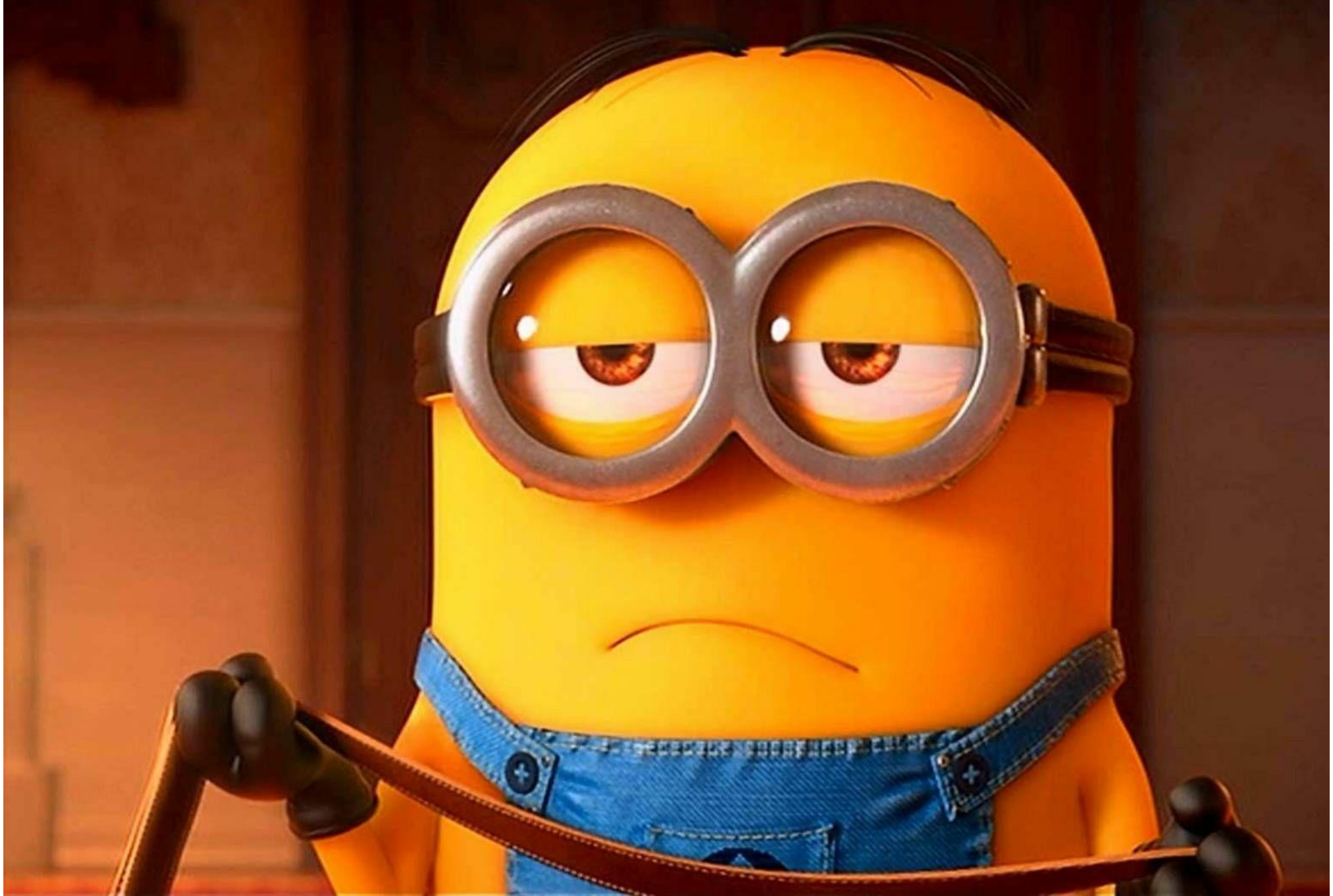
Annualized Total Bitcoin Footprints

Carbon Footprint	Electrical Energy	Electronic Waste
114.06 Mt CO2	**204.50 TWh**	**32.14 kt**
Comparable to the carbon footprint of **Czech Republic**.	Comparable to the power consumption of **Thailand**.	Comparable to the small IT equipment waste of **the Netherlands**.

Single Bitcoin Transaction Footprints

Carbon Footprint	Electrical Energy	Electronic Waste
1278.36 kgCO2	**2291.94 kWh**	**360.20 grams**
Equivalent to the carbon footprint of **2,833,277** VISA transactions or **213,059** hours of watching Youtube.	Equivalent to the power consumption of an average U.S. household over **78.56** days.	Equivalent to the weight of **2.20** iPhones 12 or **0.74** iPads. (Find more info on e-waste here.)

Reducing energy consumption to combat Climate change



Sustainable software development?

Energy-efficient vs Run-time-efficient

Computer	**A**	**B**
Energy (in Joules)	30	20
Time (in seconds)	10	20

Software Development Life Cycle for Energy-Efficiency: Techniques and Tools

1

Software Development Life Cycle for Energy-Efficiency: Techniques and Tools

STEFANOS GEORGIOU, Athens University of Economics and Business, Singular Logic S.A.
STAMATIA RIZOU, Singular Logic S.A.
DIOMIDIS SPINELLIS, Athens University of Economics and Business

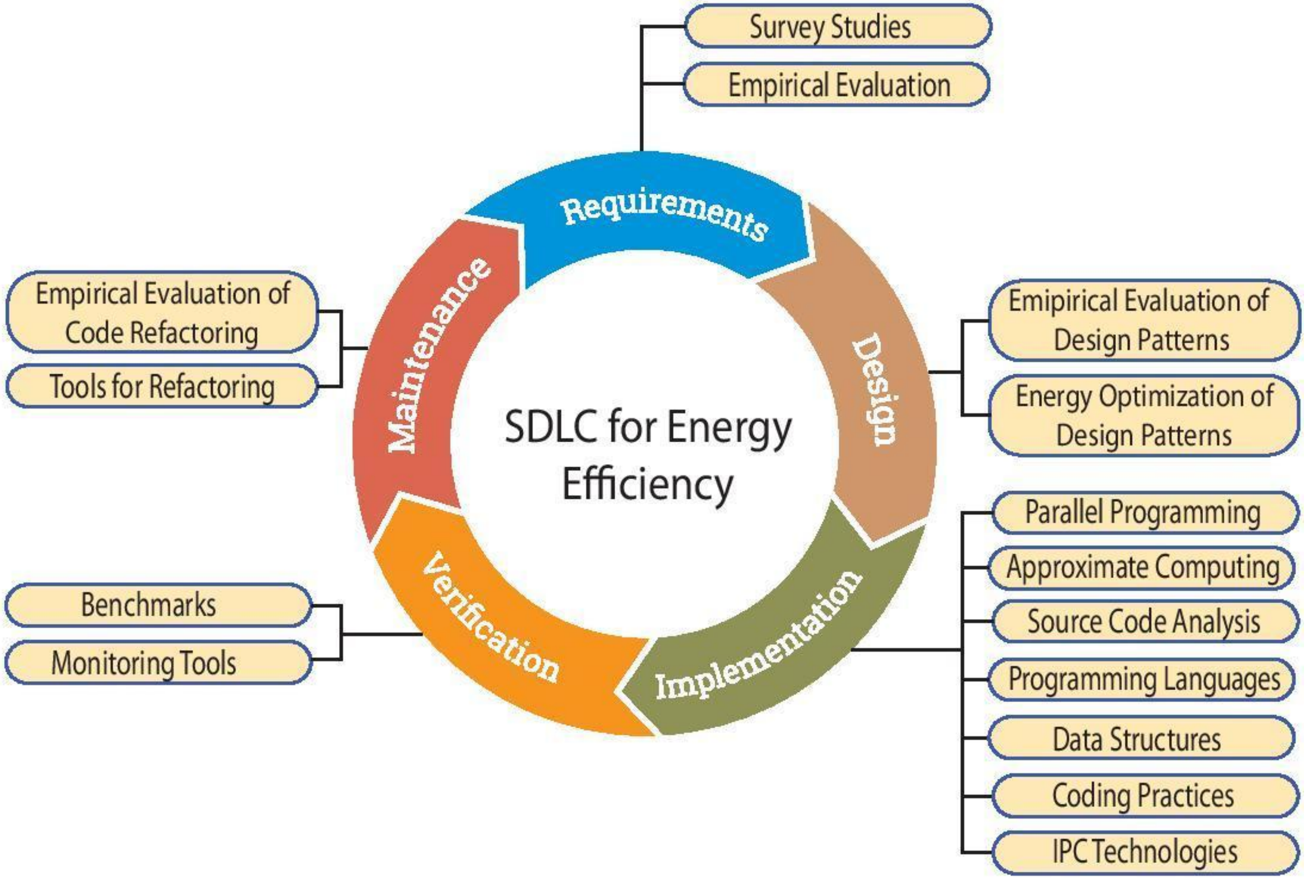
Motivation: In modern IT systems the increasing demand for computational power is tightly coupled with ever higher energy consumption. Traditionally, energy efficiency research has focused on reducing energy consumption at the hardware level. Nevertheless, the software itself provides numerous opportunities for improving energy efficiency.
Goal: Given that energy efficiency for IT systems is a rising concern, we investigate existing work in the area of energy-aware software development and identify open research challenges. Our goal is to reveal limitations, features, and trade-offs regarding energy-performance for software development and provide insights on existing approaches, tools, and techniques for energy-efficient programming.
Method: We analyze and categorize research work mostly extracted from top-tier conferences and journals concerning energy efficiency across the software development life cycle phases.
Results: Our analysis shows that related work in this area has focused mainly on the implementation and verification phases of the software development life cycle. Existing work shows that the use of parallel and approximate programming, source code analyzers, efficient data structures, coding practices, and specific programming languages can significantly increase energy efficiency. Moreover, the utilization of energy monitoring tools and benchmarks can provide insights for the software practitioners and raise energy-awareness during the development phase.

CCS Concepts:•**Software and its engineering** → **Software maintenance tools; Requirements analysis; Software design engineering; Software verification and validation;** *Extra-functional properties; Data types and structures; Classes and objects; Software design tradeoffs;* Concurrent programming structures; Patterns; Software implementation planning;•**Computing methodologies** → *Parallel computing methodologies;*

Additional Key Words and Phrases: GreenIT, Energy Efficiency, Energy Optimization, Energy Profiling, Design Patterns, Parallel Programming, Code Refactoring, Source Code Analysis, Coding Practices, Approximate Programming, Software Development Life Cycle

ACM Reference Format:
Stefanos Georgiou, Stamatia Rizou, Diomidis Spinellis, 2019. Software Development Life Cycle for Energy Efficiency: Techniques and Tools. *ACM Comput. Surv.* 1, 1, Article 1 (January 2019), 35 pages.
DOI: 10.1145/3337773

Survey Study: Context



Survey Study: Research Challenges

RC1. Limited investigation on diverse programming languages.

RC2. Limited investigation on diverse remote Inter-Process Communication technologies.

RC3. Limited tooling support for finding which data structures are the most energy-efficient for specific case.

RC4. Selection of configurations and parameters (parallel and approximate programming).

What are your Programming Language Energy-Delay Product Implications?

What Are Your Programming Language's Energy-Delay Implications?

Stefanos Georgiou
Athens University of Economics and Business
sgeorgiou@aueb.gr

Maria Kechagia
Delft University of Technology
m.kechagia@tudelft.nl

Panos Louridas
Athens University of Economics and Business
louridas@aueb.gr

Diomidis Spinellis
Athens University of Economics and Business
dds@aueb.gr

ABSTRACT

Motivation: Even though many studies examine the energy efficiency of hardware and embedded systems, those that investigate the energy consumption of software applications are still limited, and mostly focused on mobile applications. As modern applications become even more complex and heterogeneous a need arises for methods that can accurately assess their energy consumption.
Goal: Measure the energy consumption and run-time performance of commonly used programming tasks implemented in different programming languages and executed on a variety of platforms to help developers to choose appropriate implementation platforms.
Method: Obtain measurements to calculate the Energy Delay Product, a weighted function that takes into account a task's energy consumption and run-time performance. We perform our tests by calculating the Energy Delay Product of 25 programming tasks, found in the Rosetta Code Repository, which are implemented in 14 programming languages and run on three different computer platforms, a server, a laptop, and an embedded system.
Results: Compiled programming languages are outperforming the interpreted ones for most, but not for all tasks. C, C#, and JavaScript are on average the best performing compiled, semi-compiled, and interpreted programming languages for the Energy Delay Product, and Rust appears to be well-placed for I/O-intensive operations, such as file handling. We also find that a good behaviour, energy-wise, can be the result of clever optimizations and design choices in seemingly unexpected programming languages.

CCS CONCEPTS

• **Hardware → Power estimation and optimization**; • **Software and its engineering** → *Software libraries and repositories*; *Software design tradeoffs*;

KEYWORDS

Programming Languages; Energy-Delay-Product; Energy-Efficiency;

ACM Reference Format:
Stefanos Georgiou, Maria Kechagia, Panos Louridas, and Diomidis Spinellis. 2018. What Are Your Programming Language's Energy-Delay Implications?. In *MSR '18: MSR '18: 15th International Conference on Mining Software Repositories , May 28–29, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3196398.3196414

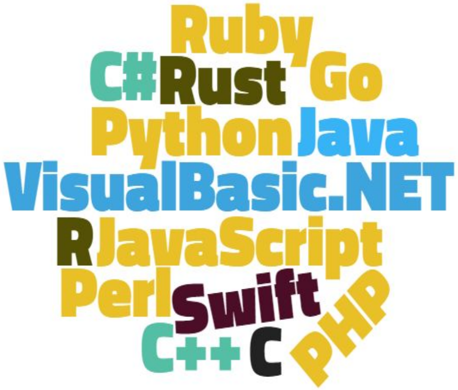
1 INTRODUCTION

Nowadays, energy consumption[1] mattes more than ever before—given that modern software applications should be able to run on devices with particular characteristics (e.g., regarding their main memory and processor). Although hardware design and utilization is undoubtedly a key factor affecting energy consumption, there is much evidence that software can also significantly influence the energy usage of computer platforms [7, 16, 18].

Today, software practitioners can select from a large pool of programming languages to develop software applications and systems. Each of these programming languages comes with a number of features and characteristics that can affect the energy consumption and run-time performance of programming tasks implemented in such languages. With the advent of cloud computing, data centers, and mobile platforms, the same programming tasks can run on distinct platforms consuming energy in different ways. In this context, there is limited work available that examines the energy and performance implications of particular programming tasks that are written in different languages and run on different platforms.

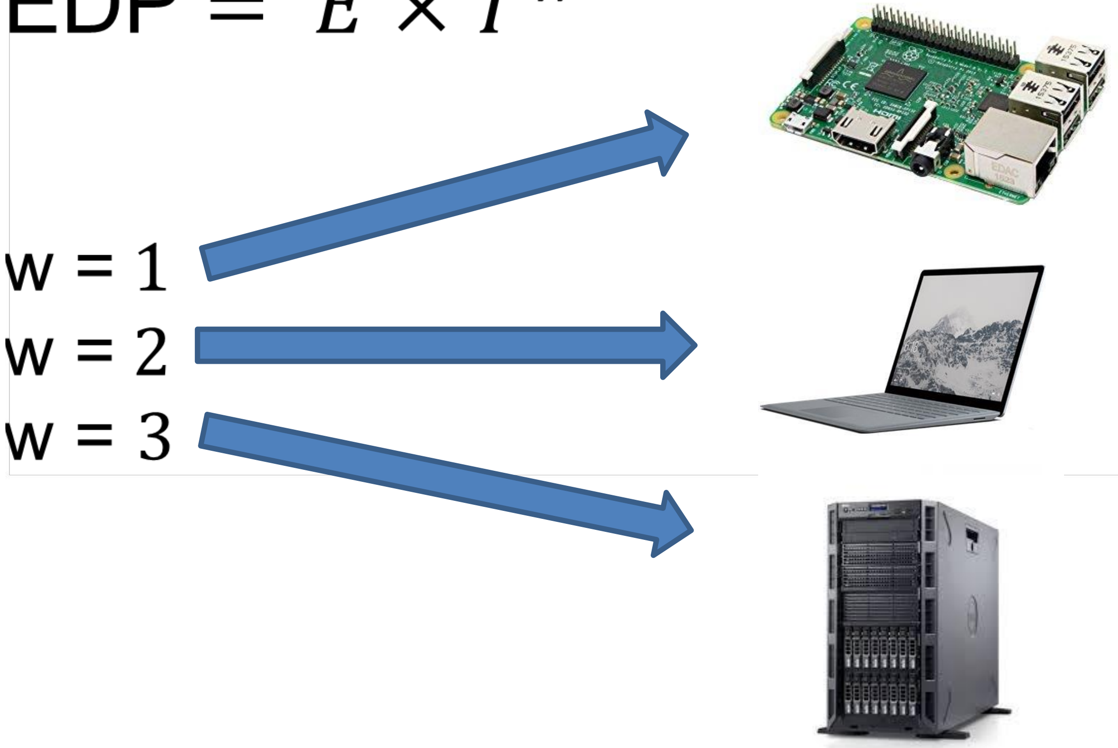
In this paper, we measure the Energy Delay Product (EDP), a weighted function of the energy consumption and run-time performance product, for a sample of commonly used programming tasks. We do this to identify which *programming language implementations* (i.e., programming tasks developed in particular programming

Motivation


Ruby
C# Rust Go
Python Java
VisualBasic.NET
R JavaScript
Perl Swift
C++ C PHP

Energy Delay Product

$$\text{EDP} = E \times T^{w}$$



Research Questions

RQ1: Which programming languages are the most EDP efficient and inefficient for particular tasks?

RQ2: Which type of programming languages are, on average, more EDP efficient and inefficient for each of our selected platforms?

RQ3: How much does the EDP of each programming language differ among the selected platforms?

Programming Languages

- Monthly index rating based on languages popularity
- Data retrieved from 25 search engines using search query
- Programming Languages criteria:
  1. At least, 5000 hits on Google
  2. Turing complete
  3. Wikipedia page

Selected Programming Languages

Categories	Programming Languages	Compilers & Interpreters		
		Embedded	Laptop	Server
Compiled	C	6.3.0	6.4.1	6.4.1
	C++	6.3.0	6.4.1	6.4.1
	Go	1.4.3	1.7.6	1.7.6
	Rust	1.20.0	1.18.0	1.21.0
	Swift	3.1.1	3.0.2	3.0.2
Semi-Compiled	C#	4.6.2	4.6.2	4.6.2
	VB.NET	4.6.2	4.6.2	4.6.2
	Java	1.8.0	1.8.0	1.8.0
Interpreted	JavaScript	9.0.4	8.9.3	8.9.3
	Perl	5.24.1	5.24.1	5.24.1
	PHP	5.6.30	7.0.25	7.0.25
	Python	2.7.23	2.7.13	2.7.13
	R	3.3.3	3.4.2	3.4.2
	Ruby	2.4.2	2.4.1	2.4.1

Rosetta Code Repository

ROSETTACODE.ORG

Create account Log in

Search

Community Explore

Main page Discussion View source History

I'm working on modernizing Rosetta Code's infrastructure. Starting with communications. Please accept this time-limited open invite to RC's Slack.. --Michael Mol (talk) 20:59, 30 May 2020 (UTC)

Rosetta Code

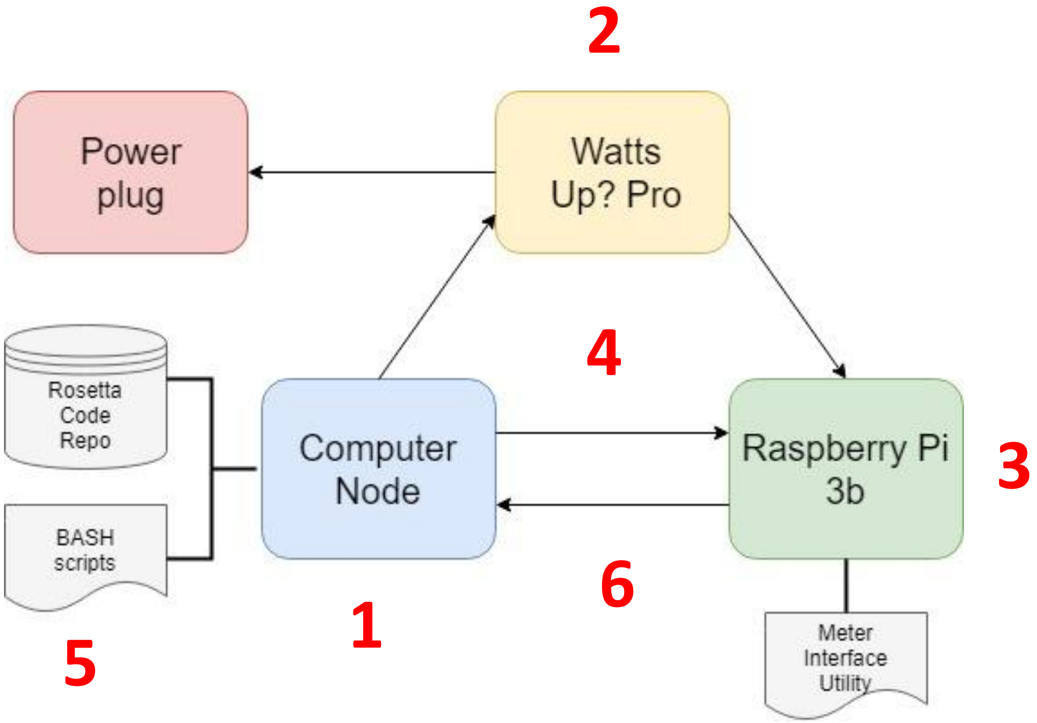
Rosetta Code is a programming chrestomathy site. The idea is to present solutions to the same task in as many different languages as possible, to demonstrate how languages are similar and different, and to aid a person with a grounding in one approach to a problem in learning another. Rosetta Code currently has 1,083 tasks, 226 draft tasks, and is aware of 813 languages, though we do not (and cannot) have solutions to every task in every language.

Data Set

Categories	Tasks
Arithmetic	*exponentiation-operator and numerical-integration*
Compression	*huffman-coding and lzw-compression*
Concurrent	*concurrency-computing and synchronous-concurrency*
Data structures	*array-concatenation and json*
File handling	*file-input-output*
Recursion	*Factorial, ackermann-function and palindrome-detection*
Regular Expression	*regular expression*
Sorting algorithms	*selection, insertion, merge, bubble, and quick*
String manipulation	*url-encoding/decoding*
Object-Oriented	*inheritance single/multiple, class, and call-an-object-method*
Functional	*function-composition*

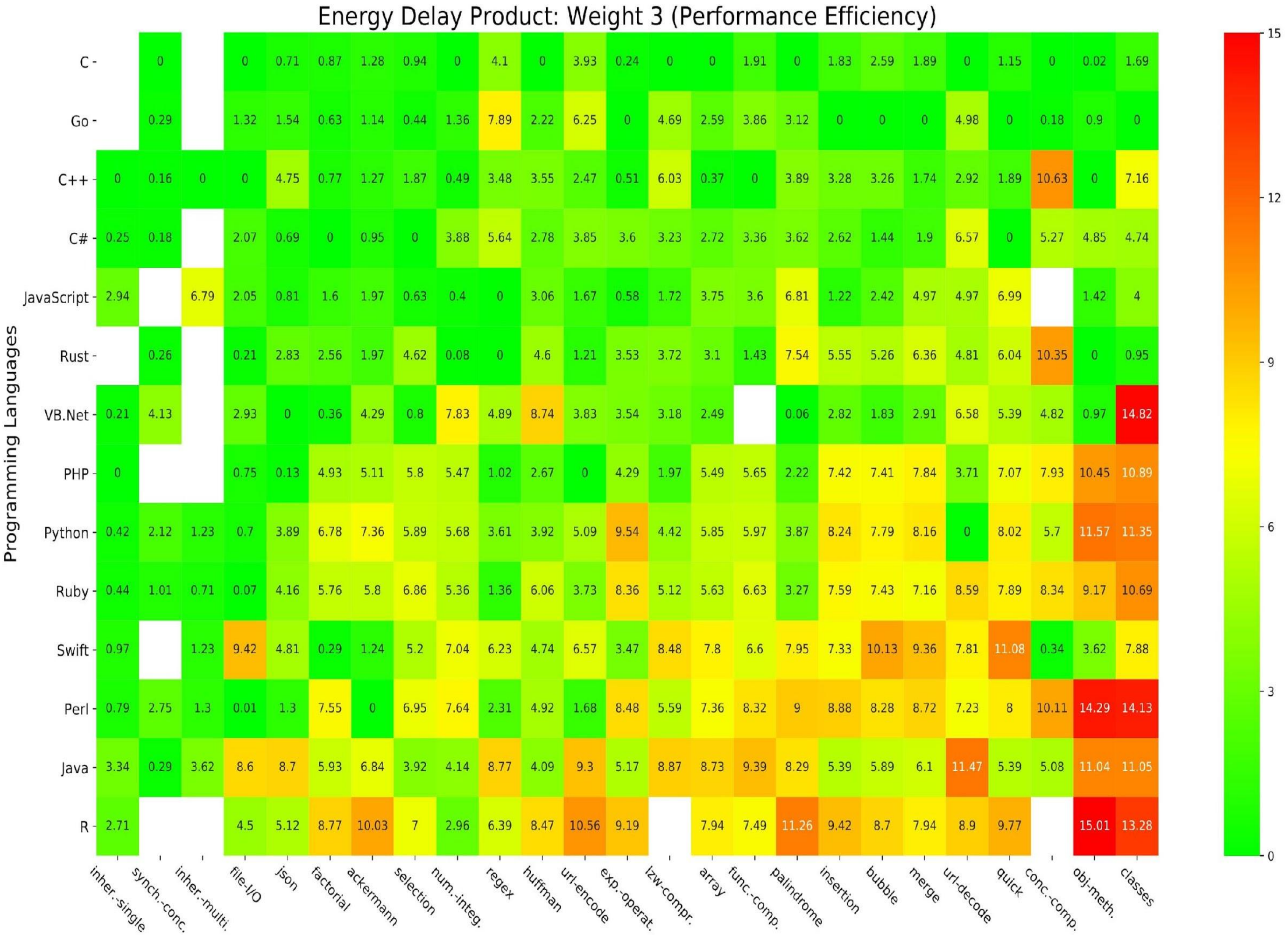
Experimental Platform

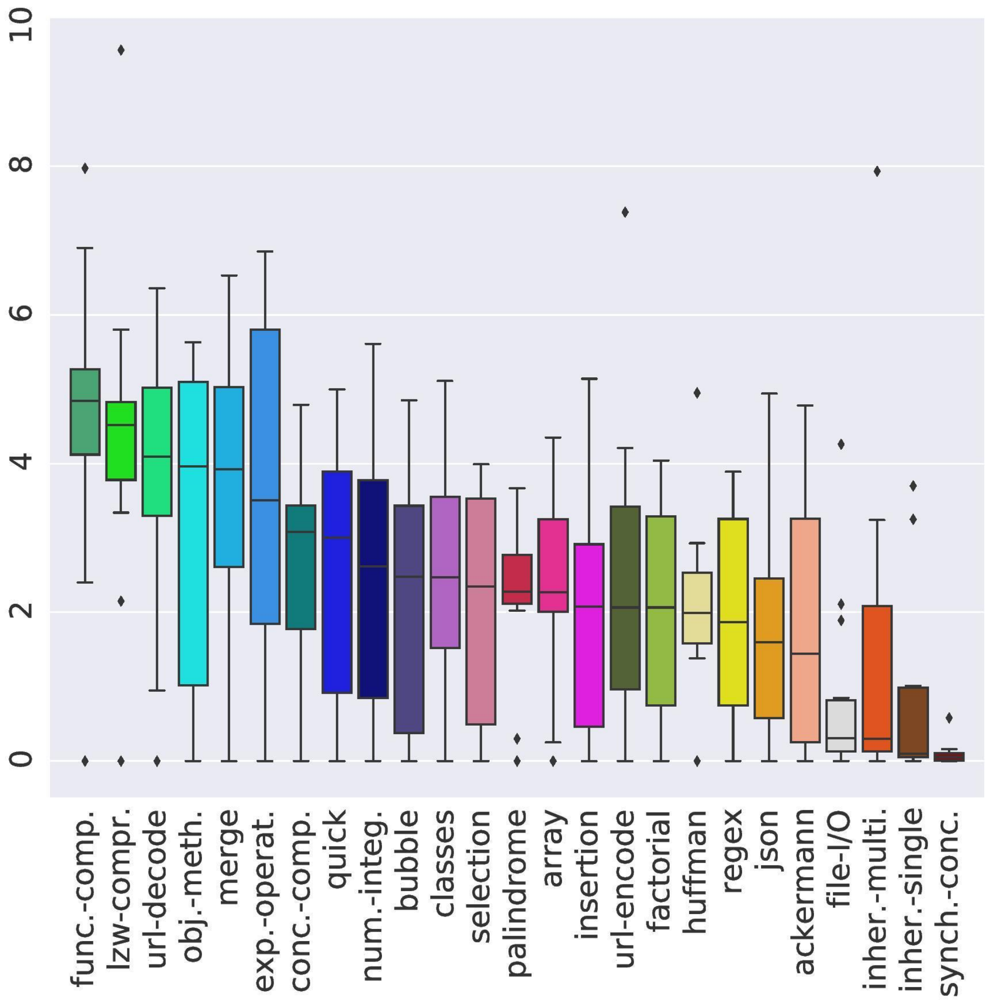
Execution Process



Questions

1. Drawback of using a hardware or software-based energy profiler?
2. Which factors can affect our experiment setup?


Energy Delay Product: Weight 3 (Performance Efficiency)
Programming Languages
C 0 0 0.71 0.87 1.28 0.94 0 4.1 0 3.93 0.24 0 0 1.91 0 1.83 2.59 1.89 0 1.15 0 0.02 1.69
Go 0.29 1.32 1.54 0.63 1.14 0.44 1.36 7.89 2.22 6.25 0 4.69 2.59 3.86 3.12 0 0 0 4.98 0 0.18 0.9 0
C++ 0 0.16 0 0 4.75 0.77 1.27 1.87 0.49 3.48 3.55 2.47 0.51 6.03 0.37 0 3.89 3.28 3.26 1.74 2.92 1.89 10.63 0 7.16
C# 0.25 0.18 2.07 0.69 0 0.95 0 3.88 5.64 2.78 3.85 3.6 3.23 2.72 3.36 3.62 2.62 1.44 1.9 6.57 0 5.27 4.85 4.74
JavaScript 2.94 6.79 2.05 0.81 1.6 1.97 0.63 0.4 0 3.06 1.67 0.58 1.72 3.75 3.6 6.81 1.22 2.42 4.97 4.97 6.99 1.42 4
Rust 0.26 0.21 2.83 2.56 1.97 4.62 0.08 0 4.6 1.21 3.53 3.72 3.1 1.43 7.54 5.55 5.26 6.36 4.81 6.04 10.35 0 0.95
VB.Net 0.21 4.13 2.93 0 0.36 4.29 0.8 7.83 4.89 8.74 3.83 3.54 3.18 2.49 0.06 2.82 1.83 2.91 6.58 5.39 4.82 0.97 14.82
PHP 0 0.75 0.13 4.93 5.11 5.8 5.47 1.02 2.67 0 4.29 1.97 5.49 5.65 2.22 7.42 7.41 7.84 3.71 7.07 7.93 10.45 10.89
Python 0.42 2.12 1.23 0.7 3.89 6.78 7.36 5.89 5.68 3.61 3.92 5.09 9.54 4.42 5.85 5.97 3.87 8.24 7.79 8.16 0 8.02 5.7 11.57 11.35
Ruby 0.44 1.01 0.71 0.07 4.16 5.76 5.8 6.86 5.36 1.36 6.06 3.73 8.36 5.12 5.63 6.63 3.27 7.59 7.43 7.16 8.59 7.89 8.34 9.17 10.69
Swift 0.97 1.23 9.42 4.81 0.29 1.24 5.2 7.04 6.23 4.74 6.57 3.47 8.48 7.8 6.6 7.95 7.33 10.13 9.36 7.81 11.08 0.34 3.62 7.88
Perl 0.79 2.75 1.3 0.01 1.3 7.55 0 6.95 7.64 2.31 4.92 1.68 8.48 5.59 7.36 8.32 9 8.88 8.28 8.72 7.23 8 10.11 14.29 14.13
Java 3.34 0.29 3.62 8.6 8.7 5.93 6.84 3.92 4.14 8.77 4.09 9.3 5.17 8.87 8.73 9.39 8.29 5.39 5.89 6.1 11.47 5.39 5.08 11.04 11.05
R 2.71 4.5 5.12 8.77 10.03 7 2.96 6.39 8.47 10.56 9.19 7.94 7.49 11.26 9.42 8.7 7.94 8.9 9.77 15.01 13.28
inher.-single
synch.-conc.
inher.-multi.
file-I/O
json
factorial
ackermann
selection
num.-integ.
regex
huffman
url-encode
exp.-operat.
lzw-compr.
array
func.-comp.
palindrome
insertion
bubble
merge
url-decode
quick
conc.-comp.
obj-meth.
classes
15
12
9
6
3
0


0
2
4
6
8
10
func.-comp.
lzw-compr.
url-decode
obj.-meth.
merge
exp.-operat.
conc.-comp.
quick
num.-integ.
bubble
classes
selection
palindrome
array
insertion
url-encode
factorial
huffman
regex
json
ackermann
file-I/O
inher.-multi.
inher.-single
synch.-conc.

RQ1. Which programming languages are the most EDP efficient and inefficient for particular tasks?

Task categories	Most efficient/inefficient
Arithmetic	C/R, VB.NET
Compression	C/VB.NET, Java
Concurrent	C/VB.NET, Perl
File Handling	Rust/VB.NET
Regular Expressions	JavaScript/Java
Sorting	Go/Swift, R
Functional	C++/Swift, Perl

RQ2. Which types of programming languages are, on average, more EDP efficient and inefficient for each of the selected platforms?

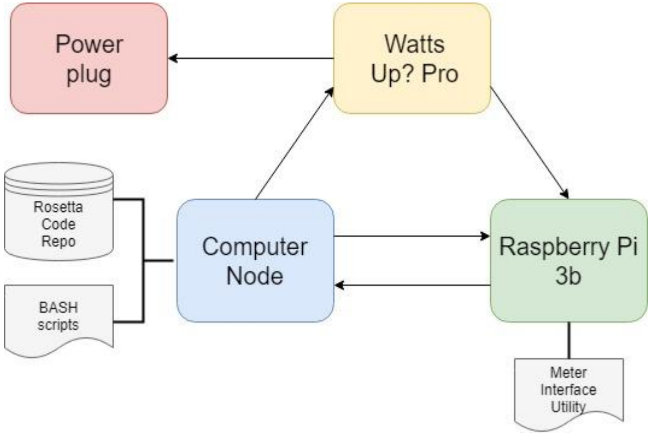
Rank	Embedded	Laptop	Server
1	C	C	C
2	C++	Go	Go
3	Go	C++	C++
4	Rust	JavaScript	C#
5	JavaScript	Rust	JavaScript
6	C#	C#	Rust
7	VB.NET	VB.NET	VB.NET
8	PHP	PHP	PHP
9	Ruby	Ruby	Python
10	Python	Swift	Ruby
11	Perl	Python	Swift
12	Java	Perl	Perl
13	Swift	Java	Java
14	R	R	R

RQ3. How much does the EDP of each programming language differ among the selected platforms?

- ***Hypothesis H0:*** *A programming language's average EDP, does not have a statistically important difference between the measurement platforms.*

In some cases, there is a significant difference between the average EDP in embedded and laptop platforms.

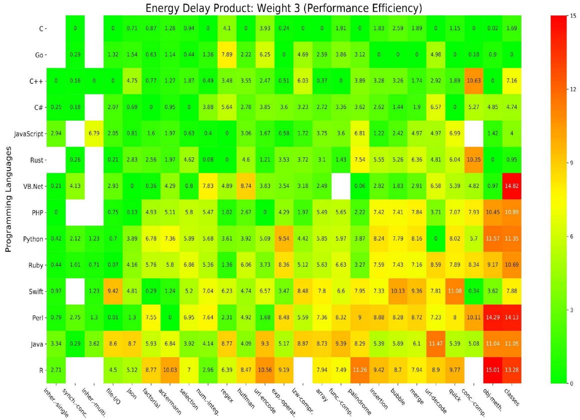
Takeaways



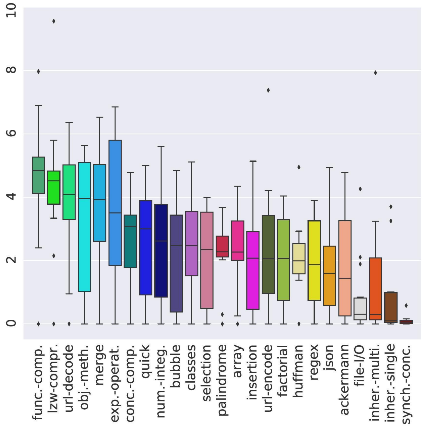
Power plug
Watts Up? Pro
Rosetta Code Repo
BASH scripts
Computer Node
Raspberry Pi 3b
Meter Interface Utility

strace !!

Android Developers Blog
01 JUNE 2016
Back to developer.android.com
SEARCH
Search
ARCHIVE
Android Developer Story: Vietnamese games developer Divmob finds more users with localized pricing on Google Play
Posted by Lily Sheringham, Google Play team
Based in Ho Chi Minh City in Vietnam, games developer Divmob has grown quickly from an original team of five people to 40 employees since it was founded three years ago. Divmob now has over 40 million downloads across its various titles, including the popular game, Epic Heroes War.



Energy Delay Product: Weight 3 (Performance Efficiency)
Programming Languages
C
Go
C++
C#
JavaScript
Rust
VB.Net
PHP
Python
Ruby
Swift
Perl
Java
R
inher.-single
synch.-conc.
inher.-multi.
file-I/O
json
factorial
ackermann
selection
num.-integ.
regex
huffman
url-encode
exp.-operat.
lzw-compr.
array
func.-comp.
palindrome
insertion
bubble
merge
url-decode
quick
conc.-comp.
obj.-meth.
classes



func.-comp.
lzw-compr.
url-decode
obj.-meth.
merge
exp.-operat.
conc.-comp.
quick
num.-integ.
bubble
classes
selection
palindrome
array
insertion
url-encode
factorial
huffman
regex
json
ackermann
file-I/O
inher.-multi.
inher.-single
synch.-conc.

Energy-Delay Investigation of Remote Inter Process Communication (IPC)Technologies

Energy-Delay Investigation of Remote Inter-Process Communication Technologies

Stefanos Georgiou[a], Diomidis Spinellis[a]

[a]Department of Management Science and Technology, Athens University of Economics and Business, Patision 76, Athina 10434

ARTICLE INFO

Keywords:
Energy Efficiency
Programming Languages
Remote Inter-Process Communication
System Calls

ABSTRACT

Most modern information technology devices use the Internet for creating, reading, updating, and deleting shared data through remote inter-process communication (IPC). To evaluate the energy consumption of IPC technologies and the corresponding run-time performance implications, we performed an empirical study on popular IPC systems implemented in Go, Java, JavaScript, Python, PHP, Ruby, and C#. We performed our experiments on computer platforms equipped with Intel and ARM processors. We observed that JavaScript and Go implementations of gRPC offer the lowest energy consumption and execution time. Furthermore, by analysing their system call traces, we found that inefficient use of system calls can contribute to increased energy consumption and poor execution time.

1. Introduction

The energy consumption,[1] for the IT-related products, is an evergrowing matter that has caught the attention of academic researchers and industry. This is primarily due to the increasing costs, as IT-related energy consumption is estimated to reach 15% of the world's total by 2020 [42]. Environmental impact is another major concern, as IT's total greenhouse gas emissions are expected to reach 2.3% by the same year [11]. Energy consumption of IT systems is particularly important in two areas. First, the data centres, one of the vital contributors of IT sector's global energy consumption and greenhouse gas emissions. These are housing large number of server nodes communicating with clients through energy-intensive remote inter-process communication (IPC) technologies. Second, the blossoming field of IoT, where low energy performance is critical, has multiple embedded devices connected with hyper-physical systems to exchange, share, and transmit data. To this end, providing sustainable solutions, by reducing energy consumption, to ensure data centres' and IoT infrastructures environmental sustainability and business growth is of paramount importance.

Researchers have carried out studies on different aspects and granularity of software artifacts to investigate the energy consumption of data structures [32, 12, 27, 28, 44], different programming languages [26, 3, 35, 18], multi-threaded applications [31, 29, 30], and coding practices [41, 19, 37, 39, 33]. In terms of remote IPC technologies, prior work [13, 8, 7, 22, 23] focused on investigating the energy consumption and run-time performance of smart phones and embedded systems on Java implementations for remote IPC such as RPC, REST, SOAP, and WebSockets. However, IPC technologies have not been investigated in terms of energy consumption and run-time performance for different programming language implementations.

In this work, we research computer platforms equipped with Intel and ARM processors using three different IPC technologies available in Java, JavaScript, Go, Python, PHP, Ruby, and C#. We try to identify which programming language and IPC technology implementations offer the best energy and run-time performance when invoking remote procedures. Furthermore, we focus on pointing out the reasons behind our results to help software developers, specifically those concerned with IPC library development, build more energy and run-time performance-efficient implementations.

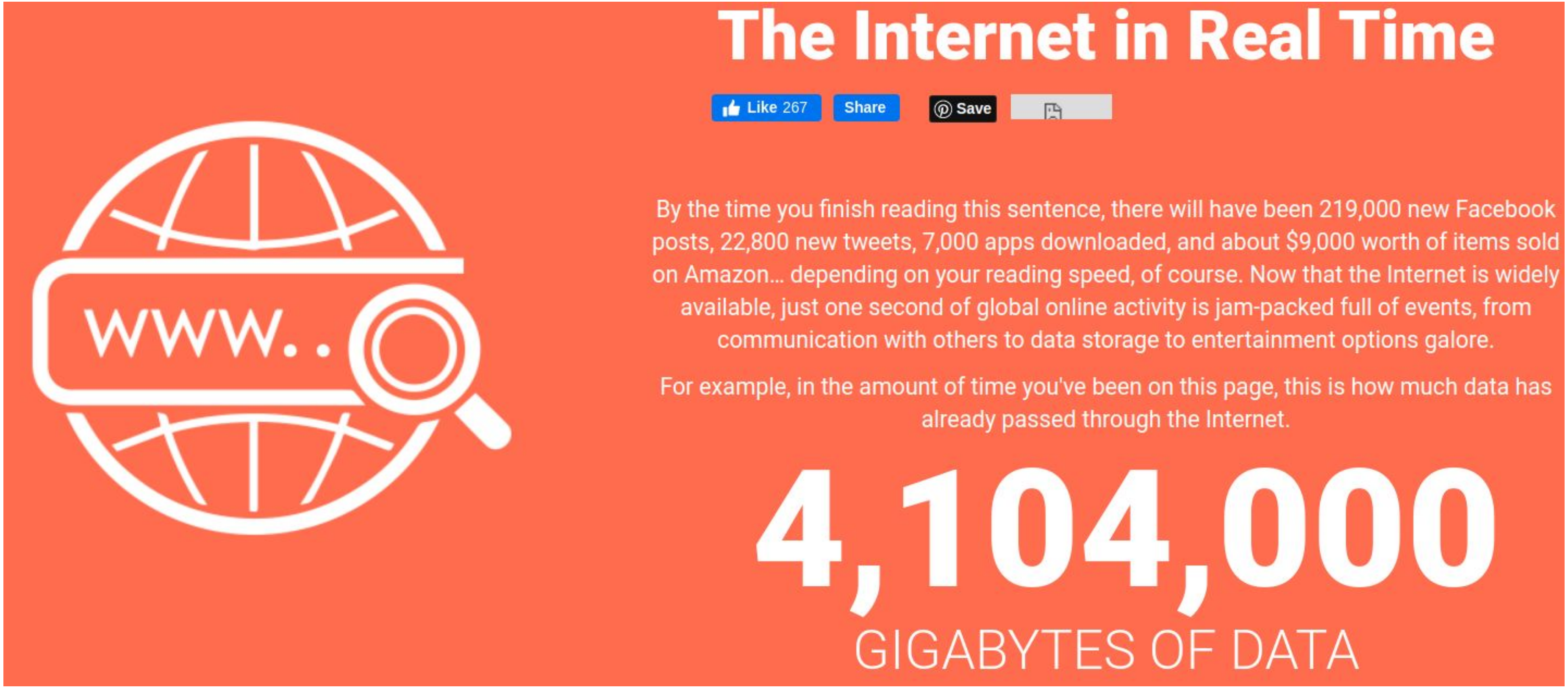
To accomplish this, we perform an empirical study on the selected computer systems on seven popular programming languages that offer implementations of three well known remote IPC technologies and investigate their energy and run-time performance cost. Our results highlight the efficiency of different implementations and libraries. We also examine whether the energy consumption of IPC technologies is proportional to the run-time performance or the systems' resource usage.

Results reveal that JavaScript and Go implementations of gRPC offer the most energy-efficient and best run-time performance implementations among the considered IPC technologies, while Ruby, PHP, Python, Java, and C# perform most inefficiently, for the most cases. We also found that the energy consumption and run-time performance is not proportional for all the examined IPC technology implementations. Besides, from the extracted system call traces, we were able to name certain misuse cases of computer resources that can contribute to higher energy demands and lower run-time performance.

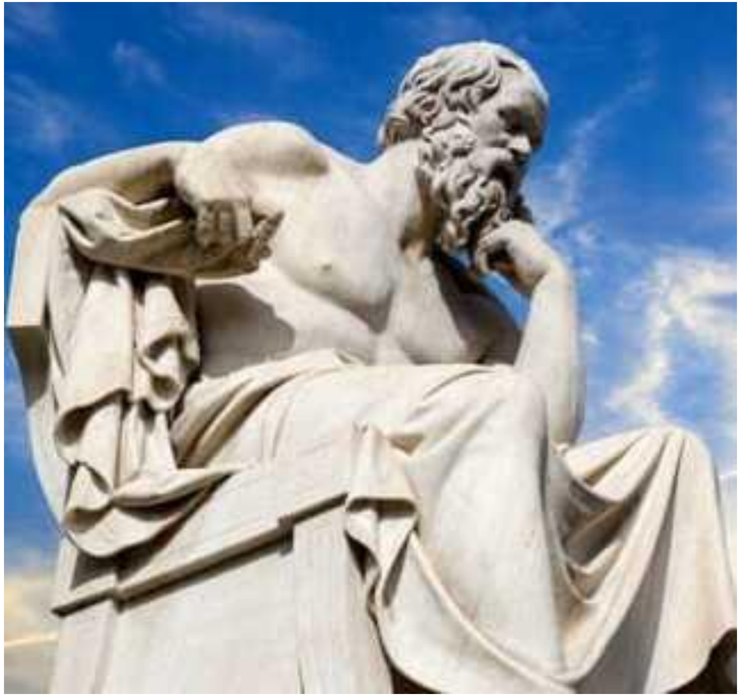
This work is organised as follows. Section 3 presents our

sgeorgiou@aueb.gr (S. Georgiou); dds@aueb.gr (D. Spinellis)
ORCID(s):

Motivation


The Internet in Real Time
Like 267
Share
Save
www..
By the time you finish reading this sentence, there will have been 219,000 new Facebook posts, 22,800 new tweets, 7,000 apps downloaded, and about $9,000 worth of items sold on Amazon... depending on your reading speed, of course. Now that the Internet is widely available, just one second of global online activity is jam-packed full of events, from communication with others to data storage to entertainment options galore.
For example, in the amount of time you've been on this page, this is how much data has already passed through the Internet.
4,104,000
GIGABYTES OF DATA

Research Questions



RQ1. Which IPC technology implementation offers the most energy and run-time performance efficient results?

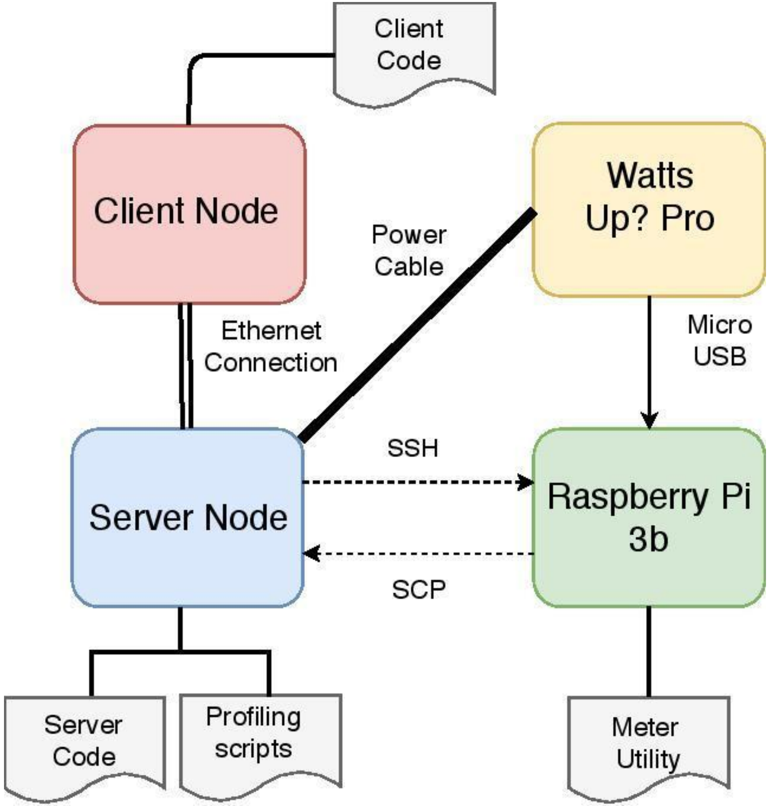
RQ2. What are the reasons that make certain IPC technologies more energy and run-time performance efficient?

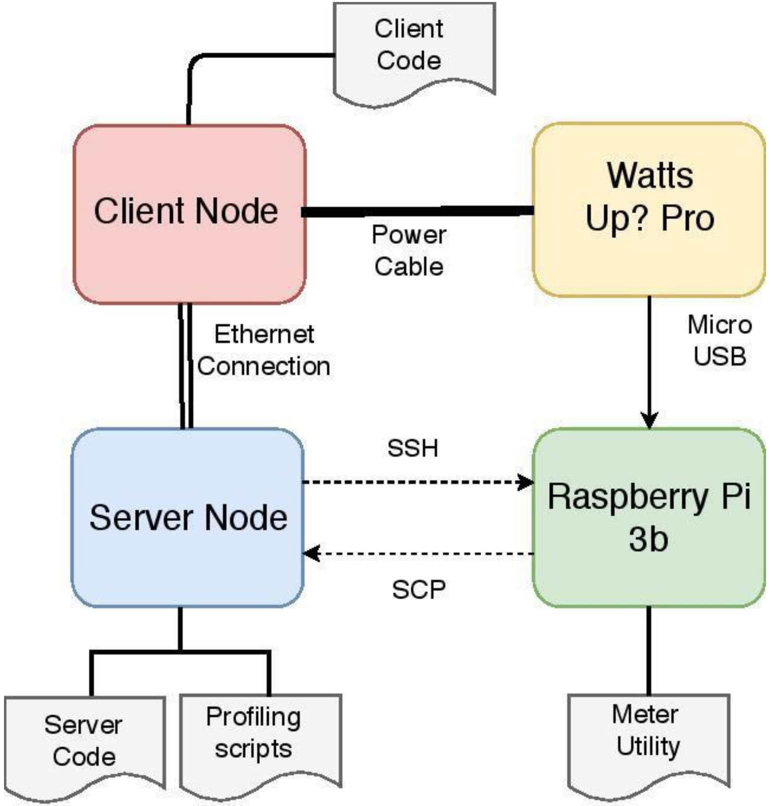
RQ3. Is the energy consumption of the IPC technologies proportional to their run-time performance or resource usage?

Subject Systems

Categories	Programming Languages	Compiler and Interpreter Versions	
		ARM Processor	Intel Processor
Compiled	Go	1.9.4	1.9.4
Semi-Compiled	Java	1.8.0	1.8.0
	C#	4.8.0	4.8.0
Interpreted	JavaScript	10.4.0	10.4.0
	Python	2.7.14	2.7.14
	PHP	7.2.12	7.2.12
	Ruby	2.5.3p	2.5.3p

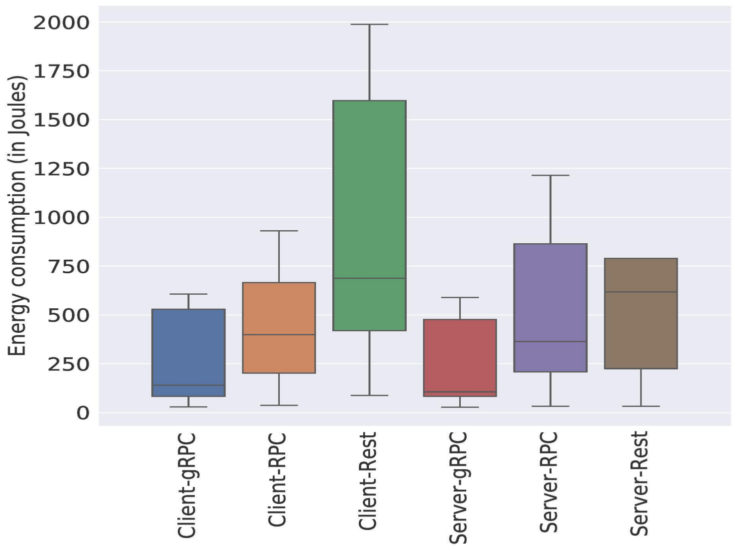
Execution Process



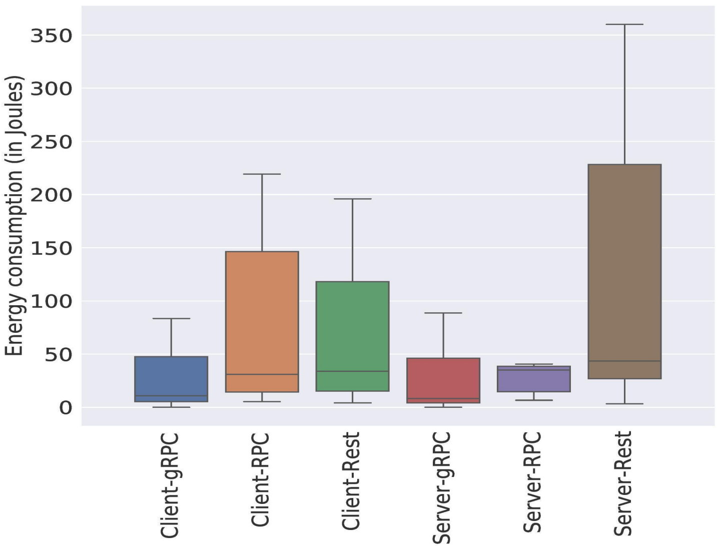


RQ1. Which IPC technology implementation offers the most energy and run-time performance-efficient results?

Intel's Platform Measurements

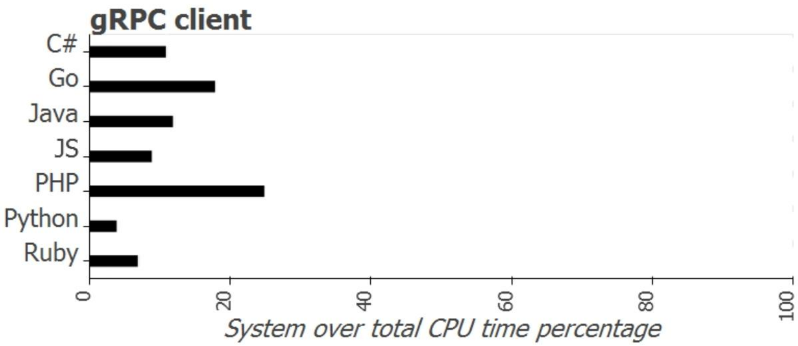


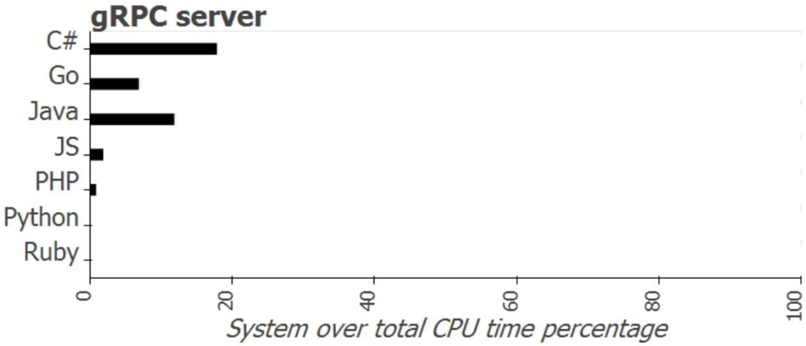
ARM's Platform Measurements



JavaScript *and* ***Go*** *are the programming languages offering the most energy and run-time performance efficient library implementations for the Intel and ARM platforms. In addition, for almost all programming language implementations, we found that* ***gRPC*** *is the IPC technology having the most efficient results.*

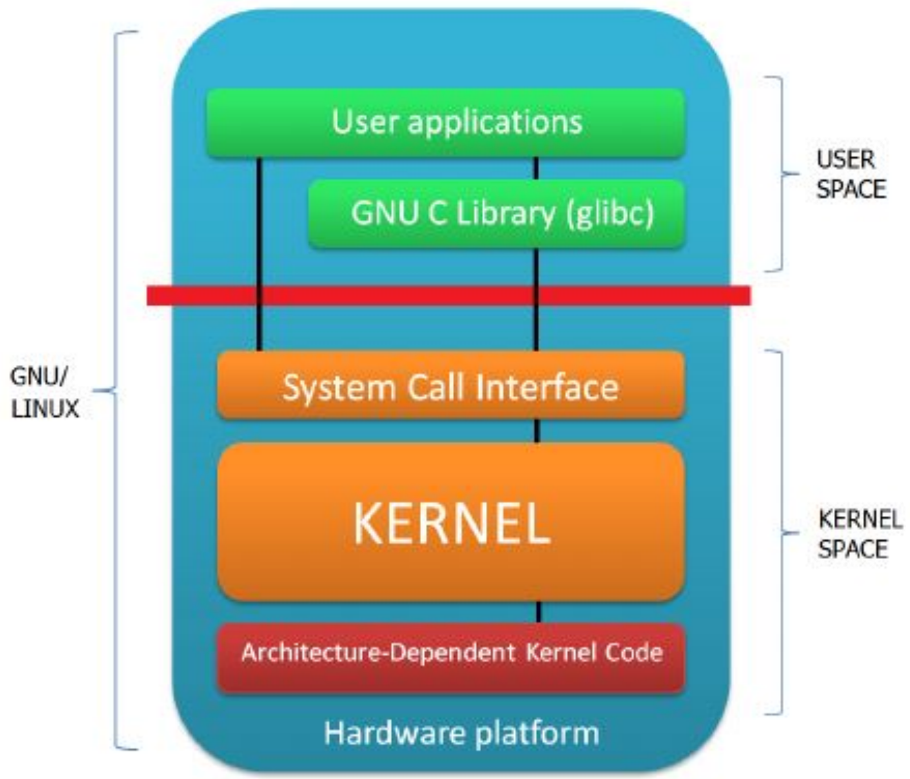
RQ2. What are the reasons that make certain IPC technologies more energy and run-time performance-efficient?





Our analysis shows the frugal opening, connecting, closing, accepting, and shutting down connections can impact the energy consumption and run-time performance of the IPC technologies. The usage of ***writev*** *system call appears in the most efficient implementations.*

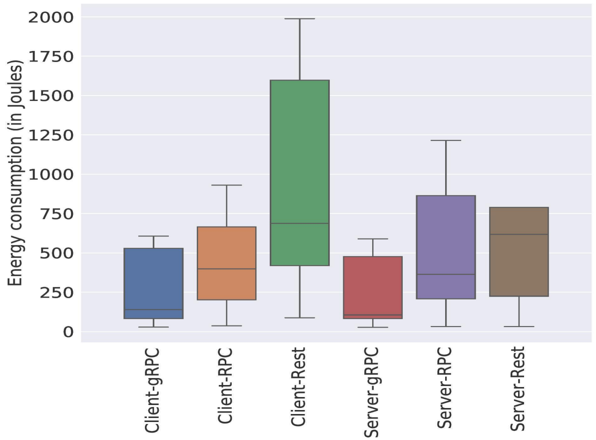
What are the system calls?



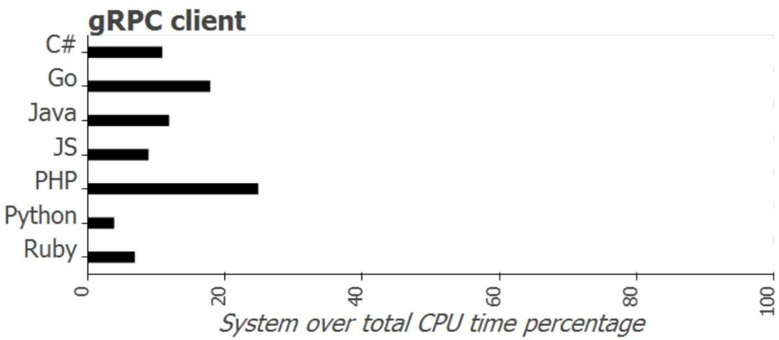
RQ3. Is the energy consumption of the IPC technologies proportional to their run-time performance or resource usage?

We found that there is a positive moderate and very strong monotonic correlation between the energy consumption and run-time performance of the Intel and ARM platforms, respectively. Also, we found a weak and very weak monotonic relationship between our energy measurements and resource usage. Therefore, none of the collected resource usage measurements can be used to justify the energy consumption results in terms of IPC technologies.

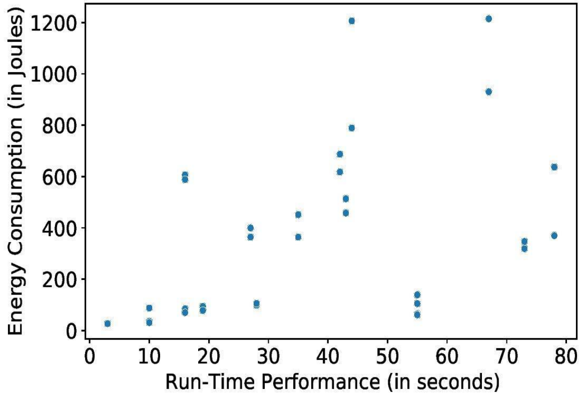
Takeaways



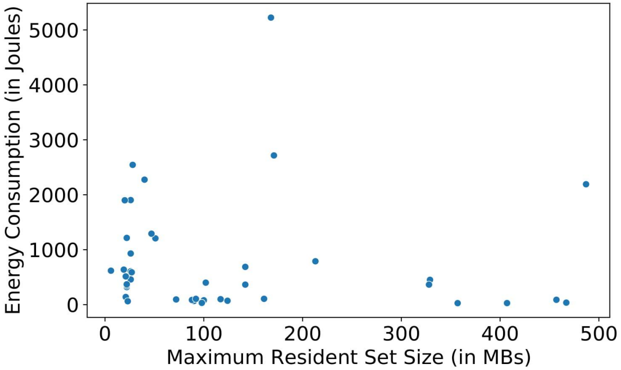
Energy consumption (in Joules)
2000
1750
1500
1250
1000
750
500
250
0
Client-gRPC
Client-RPC
Client-Rest
Server-gRPC
Server-RPC
Server-Rest



gRPC client
C#
Go
Java
JS
PHP
Python
Ruby
0
20
40
60
80
100
System over total CPU time percentage



Energy Consumption (in Joules)
1200
1000
800
600
400
200
0
0 10 20 30 40 50 60 70 80
Run-Time Performance (in seconds)



Energy Consumption (in Joules)
5000
4000
3000
2000
1000
0
0 100 200 300 400 500
Maximum Resident Set Size (in MBs)

Energy-Efficient Computing in a Secure Environment

Energy-Efficient Computing in a Secure Environment

STEFANOS GEORGIOU, Athens University of Economics and Business, Greece
DIMITRIS MITROPOULOS, Athens University of Economics and Business, Greece
DIOMIDIS SPINELLIS, Athens University of Economics and Business, Greece

An operating system (OS) typically includes numerous security mechanisms that protect the confidentiality, integrity, and availability of its data and services. However, such safeguards may impact energy consumption and encumber the run-time performance of applications running on a system. We present a large scale study, where we investigate how the various security mechanisms affect energy and run-time performance cost at the OS-level. We focus on well-known mechanisms including encrypted communication protocols, memory zeroing, GCC safeguards, and CPU vulnerability patches against critical vulnerabilities, such as Meltdown. To do so, we utilise 128 benchmarks of different application types found under the well-established Phoronix test suite. Our findings suggest that security mechanisms lead to an increased energy consumption and significantly degrade the run-time performance of various applications including web servers, databases systems, kernel operations, and disk usage. Notably, when we disabled the various security mechanisms, real-world applications such as Apache and Redis, indicated important energy (from 18% to 41%) and run-time performance (from 23% to 45%) gains. Additionally, we examined the correlation between energy consumption and performance. Our findings showed that the two are not always related. Overall, our results suggest that administrators should consider disabling such security mechanisms when a computer system runs inside a secure environment to benefit from energy and run-time performance gains.

CCS Concepts: • **Hardware → Power and energy**; • **Security and privacy → Systems security**.

Additional Key Words and Phrases: energy consumption, CPU vulnerability patches, secure environment, security mechanisms, GCC safeguards, encrypted network communications, memory zeroing, Spectre, Meltdown, MDS;

ACM Reference Format:
Stefanos Georgiou, Dimitris Mitropoulos, and Diomidis Spinellis. 2020. Energy-Efficient Computing in a Secure Environment. *ACM Trans. Comput. Syst.* 37, 4, Article 111 (June 2020), 29 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

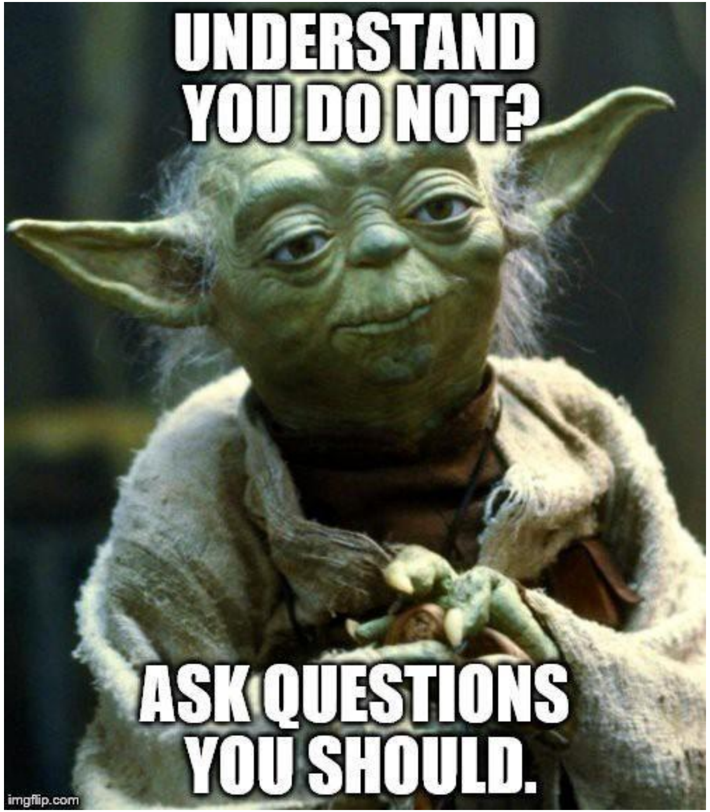
The high computational demands and services, offered by the different IT-related products, increase the energy consumption of the computer systems. The study of energy consumption in computer systems has gained vast popularity in recent years due to the advent of mobile applications and data centers that are responsible for more than 10% of the world's [18, 25, 62]. Therefore, reducing the energy footprint of IT products is of increased importance.

Unix-like OS's are equipped with various services, background processes, and daemons [59]. Among such services are the security mechanisms that protect computer systems from attackers trying to exploit potential vulnerabilities. Such security mechanisms can affect the energy consumption and the run-time performance of a

Authors' addresses: Stefanos Georgiou, sgeorgiou@aueb.gr, Athens University of Economics and Business, Patision 76, Athens, Greece, 10434; Dimitris Mitropoulos, dimitro@aueb.gr, Athens University of Economics and Business, Patision 76, Athens, Greece, 10434; Diomidis Spinellis, dds@aueb.gr, Athens University of Economics and Business, Patision 76, Athens, Greece, 10434.

Motivation

Research Questions



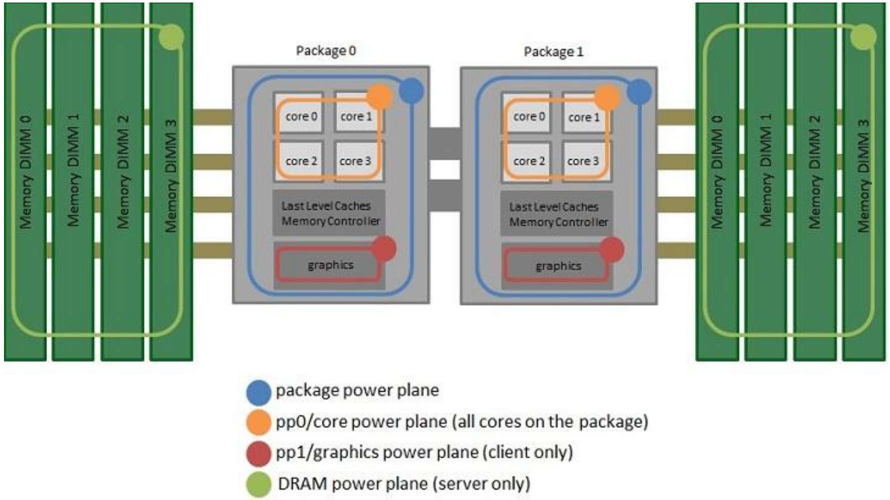
RQ1. What are the energy and run-time performance implications of the investigated security mechanisms on a computer system?

RQ2. Is the energy consumption of the examined security mechanisms proportional to their run-time performance?

RQ3. How do security mechanisms affect the energy consumption and the run-time performance of diverse applications and utilities?

Subject Systems

System	Kaby Lake (x86)
Microarchitecture	Kaby Lake
Processor/Soc	Core i7-7700
Cores × threads	4 × 2
Base Frequency	3.6 GHz
Max Frequency	4.2 GHz
Cache line size	64 B
L1-D/L1-I Cache	4 x 32 KiB 8-way
L2 cache	4 x 256 KiB 4-way
L3 cache	8 MB 16-way
I-TBL	4-KByte pages, 8-way
D-TBL	1-GB pages, 4-way
L2-TBl	1-MB, 4-way
RAM	16 GB UDIMM, DDR4 2400



Scenarios

CPU Vulnerability Patches	GCC Security Flags	HTTP/HTTPS
Stock	Stock	HTTP
Meltdown	Stack Protector	HTTPS
Spectre	FORTIFY_SOURCE	
MDS	PIC/PIE	
AllOff	RELRO	
	AllOff	

Data-set

Category	Benchmark Suites
Audio Encoding	encode-mp3, encode-flac
Video Encode/Decode	dav1d, svt-av1, svt-hevc, svt-vp9, vpxenc, x264, x265, ffmpeg
Code Compilation	build-php, build-linux-kernel, build-gcc, build-gdb, build-llvm, build2
File Compression	compress-p7zip, compress-bzip2, compress-zstd, compress-xz, lzbench
Database Suite	sqlite, redis, rocksdb, cassandra, mcperf, pymongo-insert
CPU Massive	aircrack-ng, apache, blogbench, brl-cad, byte, cloverleaf, cpp-perf-bench, crafty, dacapo-bench, ebizzy, embree, fhourstones, glibc-bench, gmpbench, himeno, hint, hmmer, hpcg, javascimark2, m-queens, minion, nero2d, nginx, node-express-loadtest, numenta-nab, phpbench, primesieve, pybench, pyperformance, rodinia, rust-prime, scimark2, stockfish, swet, sysbench, sudokut, tensorflow, xsbench, sunflow, bork, java-jmh, renaissance, tiobench, openssl, blake2s, john-the-ripper, botan, octave-bench, oidn
Disk Suite	fs-mark, iozone, dbench, postmark, aio-stress
Kernel	schbench, ctx-clock, stress-ng, osbench
Machine Learning	rbenchmark, numpy, scikit-learn, mkl-dnn
Memory Suite	ramspeed, stream, t-test1, cachebench, tinymembench, mbw
Networking Suite	iperf, network-loopback
Imaging	graphics-magick, inkscape, rawtherapee, tjbench, dcraw, darktable, rsvg, gegl
Renderers	tungsten, ospray, aobench, c-ray, povray, smallpt, ttsiod-renderer, indigobench, rays1bench, j2bench, qgears, jxrendermark
Desktop Graphics	xonotic, openarena, tesseract, paraview, unigine-valley, unigine-heaven, nexuiz, glmark2

```
while (get_time() - start_t < TIME) {
  create_files();
}
```

```
for (int i = 0; i < 1000; i++) {
  create_files();
}
```

Results

1--3.3%	3.4--6.6%
6.7--9.9%	10% >

	Stock		Meltdown		Spectre		MDS		AllOff		Difference (%)	
Tasks	Energy	Time	Energy	Time	Energy	Time	Energy	Time	Energy	Time	Energy	Time
cassandra_read	127	15	125	15	124	15	125	15	123	15	3.6	0
cassandra_write	115	13	112	13	112	13	111	13	111	13	3.6	0
mcperf_add	804	30	760	28	665	25	777	29	590	21	26.6	30
mcperf_append	752	29	710	27	632	24	712	27	555	20	26	31
mcperf_delete	550	18	521	17	441	14	508	17	387	12	29.6	33.3
mcperf_get	546	18	503	16	434	14	500	17	386	12	29.2	33.3
mcperf_prepend	748	29	710	27	631	23	711	27	555	20	25.8	31.1
mcperf_replace	747	30	710	28	630	25	720	29	553	20	25.8	31
mcperf_set	802	30	762	28	666	25	766	29	594	21	25.9	30
pymongo	883	38	890	39	866	37	870	38	867	38	1.8	0
redis_get	1972	66	1631	53	1761	58	1685	55	1166	36	40.8	45.4
redis_lpop	1985	65	1629	52	1756	58	1672	54	1167	36	41.1	45.4
redis_lpush	1988	66	1640	53	1763	58	1665	54	1166	36	41.3	45.4
redis_sadd	1981	66	1642	52	1764	58	1680	54	1169	36	40.9	45.5
redis_set	1972	66	1622	53	1761	58	1680	54	1167	36	40.8	44.6
rocksdb_fillrand	1281	31	1271	31	1243	31	1250	31	1212	31	5.3	0
rocksdb_fillseq	976	21	969	21	951	21	960	21	936	20	4.1	4.7
sqlitebench	944	134	962	136	878	125	930	134	908	130	3.8	3

Impact on Kernel operations

	Stock		Meltdown		Spectre		MDS		AllOff		Difference (%)	
Tasks	Energy	Time	Energy	Time	Energy	Time	Energy	Time	Energy	Time	Energy	Time
ctx_clock	6192	318	3523	174	6178	318	3165	169	1121	56	81.8	82.3
osb_files	659	29	631	27	593	26	622	27	542	23	17.6	20.6
osb_processes	887	22	852	20	889	22	870	21	803	19	9.4	13.6
osb_threads	588	19	549	17	574	19	550	18	494	15	15.9	21
osb_mem_alloc	514	23	503	22	508	23	501	22	480	21	6.6	8.6
osb_programs	584	11	553	10	576	11	546	10	511	9	12.4	18.1
stress-ng_fork	1232	24	1185	23	1222	24	1170	22	1108	21	10	12.5
stress-ng_matrix	799	14	759	13	775	13	786	14	752	13	5.9	7.1
stress-ng_msg	1024	22	862	18	988	21	636	13	519	10	49.2	54.5
stress-ng_sem	911	20	919	21	888	20	782	17	812	18	9.3	10
stress-ng_sock	1719	30	1392	28	1591	28	1367	28	1309	26	23.8	13.3
stress-ng_switch	1431	26	1411	25	1329	23	1204	21	1125	19	21.3	26.9
stress-ng_vec	1446	29	914	18	913	18	910	18	911	18	36.9	37.9

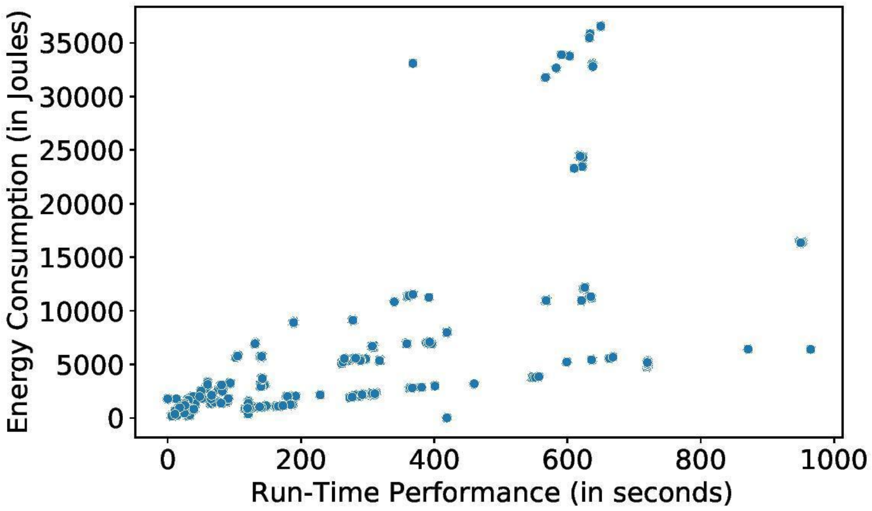
RQ1. What are the energy and run-time performance implications of the investigated security mechanisms on a computer system?

	Stock		Meltdown		Spectre		MDS		AllOff		Difference (%)	
Tasks	Energy	Time	Energy	Time	Energy	Time	Energy	Time	Energy	Time	Energy	Time
build2	10419	196	10352	197	10329	197	10282	197	10247	195	1.6	0.5
gcc	17039	359	16830	353	16898	355	16796	350	16539	345	3	3.9
gdb	5791	134	5695	131	5729	132	5666	130	5587	128	3.5	4.4
kernel	81490	1555	73093	1382	73330	1393	72572	1371	72103	1374	11.5	11.6
llvm	45784	864	45569	861	45488	862	45302	858	45052	851	1.5	1.5
php	4403	99	4380	98	4366	98	4360	98	4318	97	1.9	2

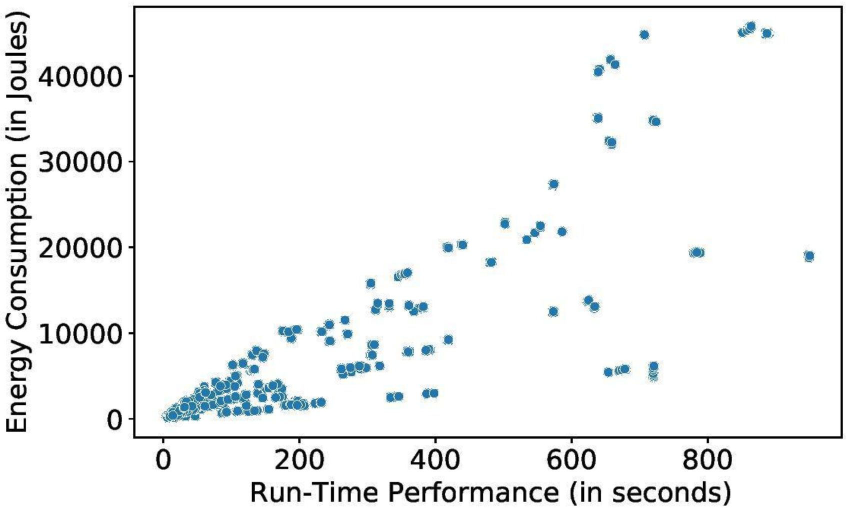
CPU vulnerability patches can ***impact the energy and run-time performance real-word applications from 18% up to 45%****, respectively. Similarly, GCC safeguards affect the energy and run-time performance of applications* ***up to 10%****. Similar results appear for the communications-related security mechanisms as well.*

RQ2. Is the energy consumption of the examined security mechanisms proportional to their run-time performance?

CPU Vulnerability Patches Measurements



GCC Security Flags Measurements



Our findings suggest that energy consumption and the run-time performance have a very strong monotonic correlation for the investigated benchmarks in the case of the CPU vulnerability patches and the GCC safeguards.

RQ3. How do security mechanisms affect the energy consumption and the run-time performance of diverse applications and utilities?

*Application types such as **database systems**, **code compilation**, **compute-intensive**, **kernel operations**, **disk usage** had the highest energy and run-time performance gains after disabling the CPU vulnerability patches. For the GCC safeguards, **compute-intensive**, **databases systems**, and **file compression** applications had the highest energy and run-time performance gains after disabling security flags.*

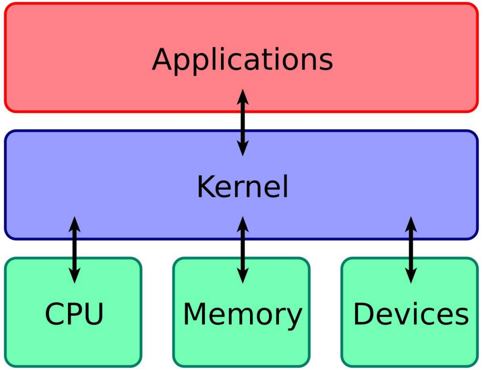
Takeaways

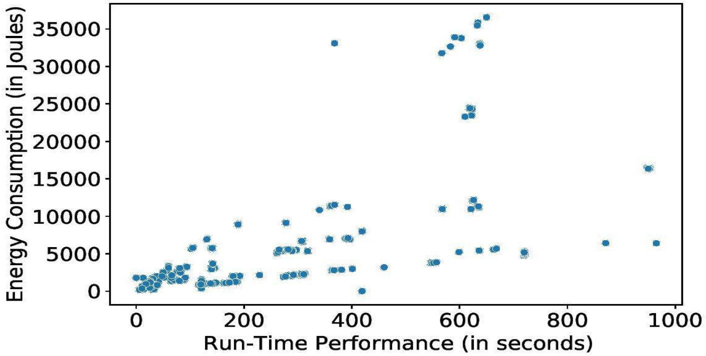
Tasks	Stock		Meltdown		Spectre		MDS		AllOff		Difference (%)	
	Energy	Time	Energy	Time	Energy	Time	Energy	Time	Energy	Time	Energy	Time
cassandra_read	127	15	125	15	124	15	125	15	123	15	3.6	0
cassandra_write	115	13	112	13	112	13	111	13	111	13	3.6	0
mcperf_add	804	30	760	28	665	25	777	29	590	21	26.6	30
mcperf_append	752	29	710	27	632	24	712	27	555	20	26	31
mcperf_delete	550	18	521	17	441	14	508	17	387	12	29.6	33.3
mcperf_get	546	18	503	16	434	14	500	17	386	12	29.2	33.3
mcperf_prepend	748	29	710	27	631	23	711	27	555	20	25.8	31.1
mcperf_replace	747	30	710	28	630	25	720	29	553	20	25.8	31
mcperf_set	802	30	762	28	666	25	766	29	594	21	25.9	30
pymongo	883	38	890	39	866	37	870	38	867	38	1.8	0
redis_get	1972	66	1631	53	1761	58	1685	55	1166	36	40.8	45.4
redis_lpop	1985	65	1629	52	1756	58	1672	54	1167	36	41.1	45.4
redis_lpush	1988	66	1640	53	1763	58	1665	54	1166	36	41.3	45.4
redis_sadd	1981	66	1642	52	1764	58	1680	54	1169	36	40.9	45.5
redis_set	1972	66	1622	53	1761	58	1680	54	1167	36	40.8	44.6
rocksdb_fillrand	1281	31	1271	31	1243	31	1250	31	1212	31	5.3	0
rocksdb_fillseq	976	21	969	21	951	21	960	21	936	20	4.1	4.7
sqlitebench	944	134	962	136	878	125	930	134	908	130	3.8	3

MEMCACHED

NGINX

Apache





Green AI: Do Deep Learning Frameworks Have Different Costs?

Green AI: Do Deep Learning Frameworks Have Different Costs?

Stefanos Georgiou*
Queen's University
stefanos.georgiou@queensu.ca

Maria Kechagia*
University College London
m.kechagia@ucl.ac.uk

Tushar Sharma
Dalhousie Uninversity
tushar@dal.ca

Federica Sarro
University College London
f.sarro@ucl.ac.uk

Ying Zou
Queen's University
ying.zou@queensu.ca

ABSTRACT

The use of Artificial Intelligence (AI), and more specifically of Deep Learning (DL), in modern software systems, is nowadays widespread and continues to grow. At the same time, its usage is energy demanding and contributes to the increased CO_2 emissions, and has a great financial cost as well. Even though there are many studies that examine the capabilities of DL, only a few focus on its *green aspects*, such as energy consumption.

This paper aims at raising awareness of the costs incurred when using different DL frameworks. To this end, we perform a thorough empirical study to measure and compare the energy consumption and run-time performance of six different DL models written in the two most popular DL frameworks, namely PyTorch and TensorFlow. We use a well-known benchmark of DL models, DeepLearningExamples, created by NVIDIA, to compare both the training and inference costs of DL. Finally, we manually investigate the functions of these frameworks that took most of the time to execute in our experiments.

The results of our empirical study reveal that there is a statistically significant difference between the cost incurred by the two DL frameworks in 94% of the cases studied. While TensorFlow achieves significantly better energy and run-time performance than PyTorch, and with large effect sizes in 100% of the cases for the training phase, PyTorch instead exhibits significantly better energy and run-time performance than TensorFlow in the inference phase for 66% of the cases, always, with large effect sizes. Such a large difference in performance costs does not, however, seem to affect the accuracy of the models produced, as both frameworks achieve comparable scores under the same configurations. Our manual analysis, of the documentation and source code of the functions examined, reveals that such a difference in performance costs is under-documented, in these frameworks. This suggests that developers need to improve the documentation of their DL frameworks, the source code of the functions used in these frameworks, as well as to enhance existing DL algorithms.

CCS CONCEPTS

• **Hardware** → **Power and energy**; • **Software and its engineering** → **Software libraries and repositories**; • **Computing methodologies** → **Machine learning**;

KEYWORDS

Energy consumption, run-time performance, deep learning, APIs

ACM Reference Format:
Stefanos Georgiou, Maria Kechagia, Tushar Sharma, Federica Sarro, and Ying Zou. 2022. Green AI: Do Deep Learning Frameworks Have Different Costs?. In *44th International Conference on Software Engineering (ICSE '22), May 21–29, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Deep learning (DL) is a field of machine learning (ML) that has recently gained significant attention from researchers and practitioners. Along with the increase of computational power and availability of data, the use of deep learning has contributed to the improvement of several applications (*e.g.*, in the medical, financial, transportation sectors) that, for instance, use speech and image recognition, machine translation, and natural language processing (NLP). The advancement in these areas would have not been possible without the great advancement in DL.

While the research community has spent a significant effort towards improving the accuracy of DL approaches, it has often overlooked their costs. As recently reported by Schwartz et al. [76], DL has been assisting in an increase in the computational costs of the state-of-the-art AI research as big as 3000,000x between 2012 and 2018. Such a dramatically increasing trend in resource consumption, dubbed as Red AI, is not just often prohibitively expensive for researchers and practitioners, but also environmentally unfriendly.

This has motivated the field of Green Software Engineering (SE) research, which aims to decrease software environmental footprints and supports, *inter alia*, Green AI [58]. Optimizing resource

Questions

- What programming languages are TensorFlow and PyTorch written in?
- Which computer component makes the biggest difference when training Deep Learning models?

Why is this important?

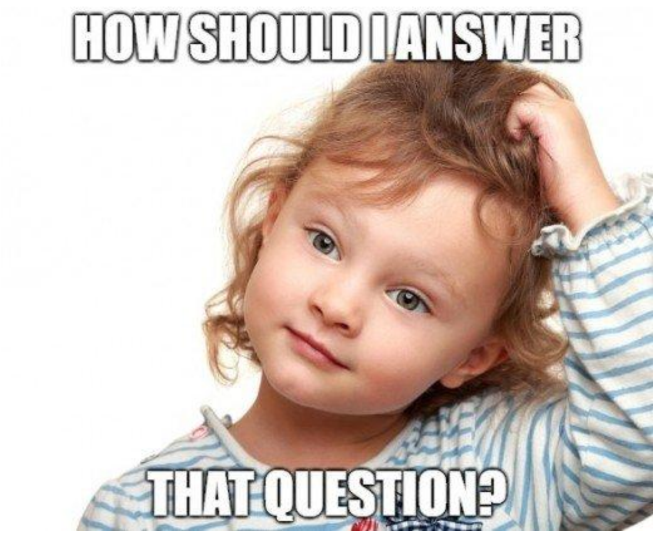
- The energy spent to train a model can vary significantly between frameworks, but how much?
- Investigate the energy and run-time performance for training and inferencing Deep Learning algorithms build in popular ML frameworks and suggest which to use in specific cases.
- What tuning parameters can affect the energy and run-time performance of the selected models?

Research Questions

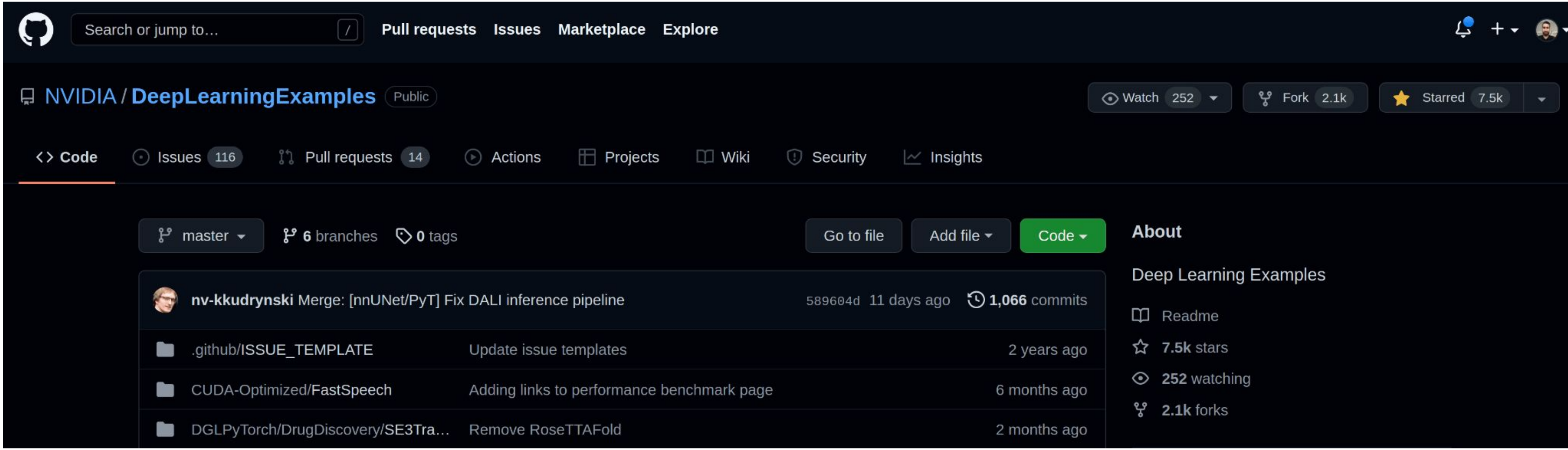
RQ1: Which is the most energy and run-time performance-efficient Deep Learning framework for the models examined?

RQ2: How much accuracy do energy and run-time performance efficient Deep Learning frameworks sacrifice for the models under examination?

RQ3: What are the most energy and run-time performance inefficient APIs of Deep Learning frameworks for the models under examination?



Public Repository



Tasks

Categories	Models	Datasets
Recommender Systems	NCF	ML-20M
NLP	Transformer-XL	WikiText-103
	GNMT	WMT16 EN-DE
Computer Vision	ResNet-50	Coco 2014
	SSD	Coco 2017
	MaskRCNN	Coco 2017

Tools

```
Tue Aug  3 17:16:41 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Quadro P4000        On   | 00000000:2D:00.0 Off |                  N/A |
| 64%   82C    P0    63W / 105W |   7789MiB /  8110MiB |     98%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A      2023      G   /usr/lib/xorg/Xorg                  9MiB |
|    0   N/A  N/A      2086      G   /usr/bin/gnome-shell                5MiB |
|    0   N/A  N/A     21488      C   python                           7769MiB |
+-----------------------------------------------------------------------------+
```

cProfile

Python 3.X

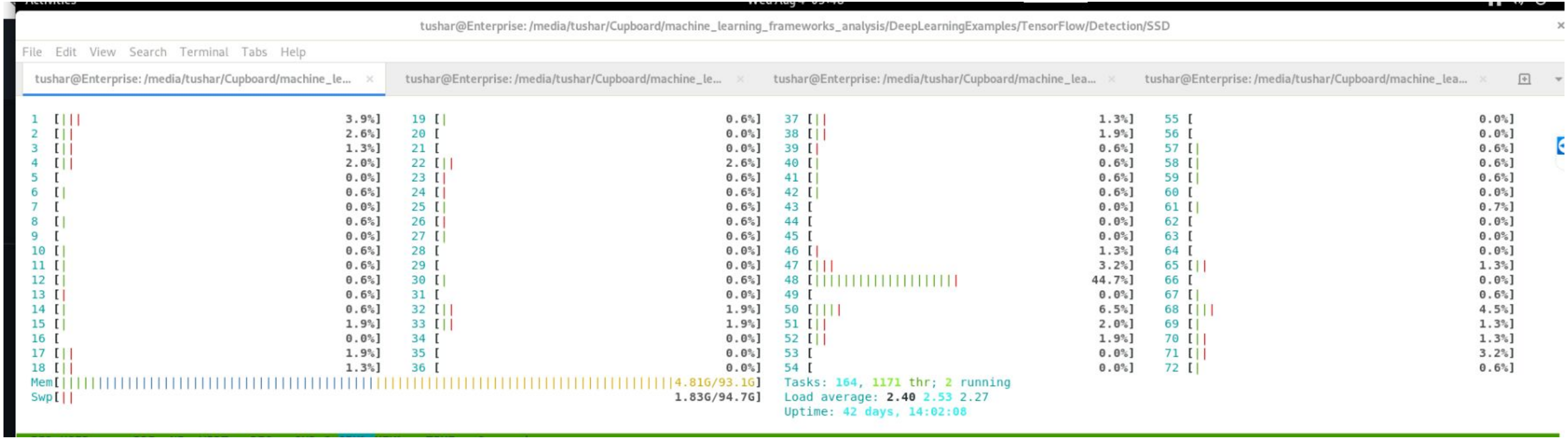
cProfile and profile provide deterministic profiling of Python programs. A profile is a set of statistics that describes how often and for how long various parts of the program executed. These statistics can be formatted into reports via the pstats module.The Python standard library provides two different implementations of the same profiling interface:

More at Python 3 Documentation

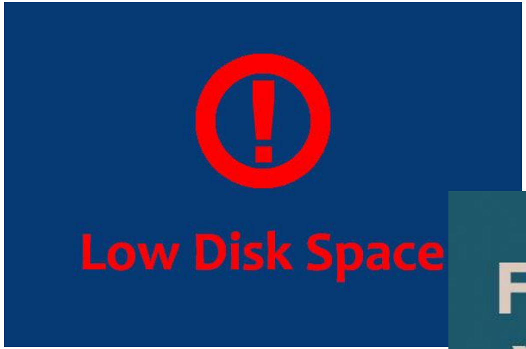
Share Feedback

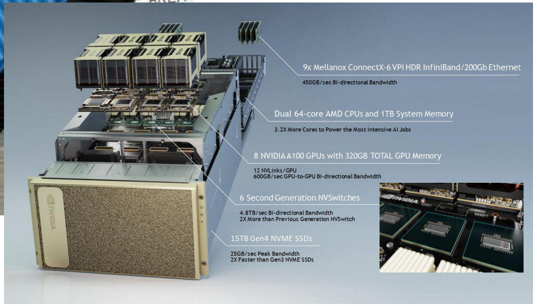
```
Display all 240 possibilities? (y or n)
tushar@Enterprise:/media/tushar/Cupboard/machine_learning_frameworks_analysis$ cat measurements/transformer_xl_PyTorch_perf_2.txt

 Performance counter stats for 'system wide':

        144,643.48 Joules power/energy-pkg/
         15,078.98 Joules power/energy-ram/

    1431.584266139 seconds time elapsed

tushar@Enterprise:/media/tushar/Cupboard/machine_learning_frameworks_analysis$ 
```

Our platform

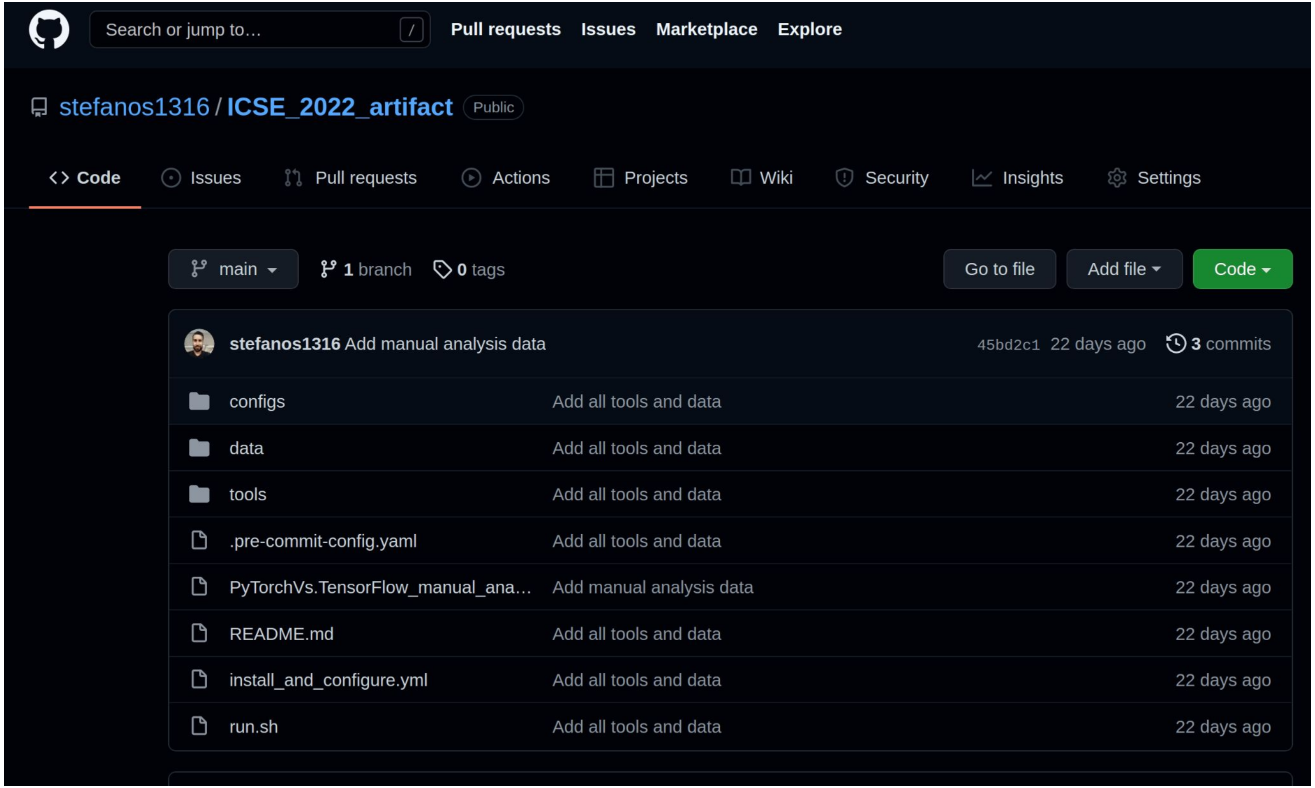


More challenges ahead





PrEngDL


Search or jump to…
Pull requests
Issues
Marketplace
Explore
stefanos1316 / ICSE_2022_artifact
Public
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
main
1 branch
0 tags
Go to file
Add file
Code
stefanos1316 Add manual analysis data
45bd2c1 22 days ago
3 commits
configs Add all tools and data 22 days ago
data Add all tools and data 22 days ago
tools Add all tools and data 22 days ago
.pre-commit-config.yaml Add all tools and data 22 days ago
PyTorchVs.TensorFlow_manual_ana… Add manual analysis data 22 days ago
README.md Add all tools and data 22 days ago
install_and_configure.yml Add all tools and data 22 days ago
run.sh Add all tools and data 22 days ago

Training results

The Recommender System, the Natural Language Processing, and the Computer Vision Category

Models	CPU & RAM Energy	*p-value (A12)*	GPU Energy	*p-value (A12)*	Run-Time	*p-value (A12)*
NCF	TensorFlow	< 0.001 (1)	TensorFlow	< 0.001 (1)	TensorFlow	< 0.001 (1)
Transformer-XL	PyTorch	< 0.001 (1)	PyTorch	< 0.001 (1)	PyTorch	< 0.001 (1)
GNMT	PyTorch	< 0.001 (1)	PyTorch	< 0.001 (1)	PyTorch	< 0.001 (1)
ResNet-50	TensorFlow	< 0.001 (1)	TensorFlow	< 0.001 (1)	TensorFlow	< 0.001 (1)
SSD	TensorFlow	< 0.001 (1)	TensorFlow	< 0.001 (1)	TensorFlow	< 0.001 (1)
Mask-RCNN	TensorFlow	< 0.001 (1)	TensorFlow	< 0.001 (1)	TensorFlow	< 0.001 (1)

Inference results

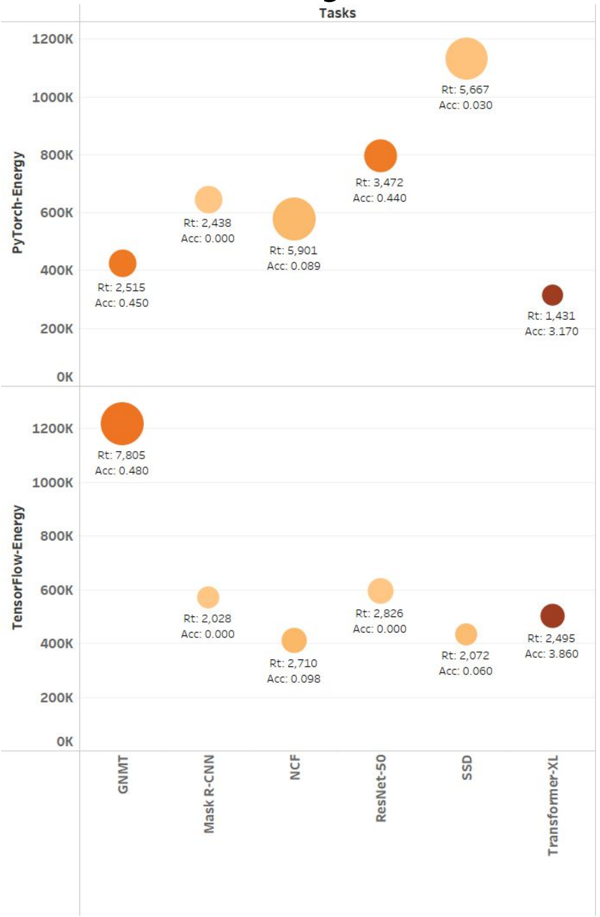
The Recommender System, the Natural Language Processing, and the Computer Vision Category

Models	CPU & RAM Energy	*p-value (A12)*	GPU Energy	*p-value (A12)*	Run-Time	*p-value (A12)*
NCF	TensorFlow	< 0.001 (1)	TensorFlow	< 0.001 (1)	TensorFlow	< 0.001 (1)
Transformer-XL	PyTorch	< 0.001 (1)	PyTorch	< 0.001 (1)	PyTorch	< 0.001 (1)
GNMT	PyTorch	< 0.001 (1)	PyTorch	< 0.001 (1)	PyTorch	< 0.001 (1)
ResNet-50	TensorFlow	< 0.001 (1)	TensorFlow	< 0.001 (1)	TensorFlow	< 0.001 (1)
SSD	PyTorch	< 0.001 (1)	PyTorch	< 0.001 (1)	PyTorch	< 0.001 (1)
Mask-RCNN	PyTorch	< 0.001 (1)	PyTorch	< 0.001 (1)	PyTorch	< 0.001 (1)

Answer to RQ1

- TensorFlow performs better for training **Recommender Systems** and **Computer Vision** tasks.
- PyTorch outperforms TensorFlow for **Natural Language Processing** tasks.
- For model inference, TensorFlow is more efficient for **Recommender Systems** and **ResNet-50**.
- Overall, TensorFlow is more energy and run-time performance efficient for **training models**, while PyTorch for models inference.

Accuracy Trade-Offs

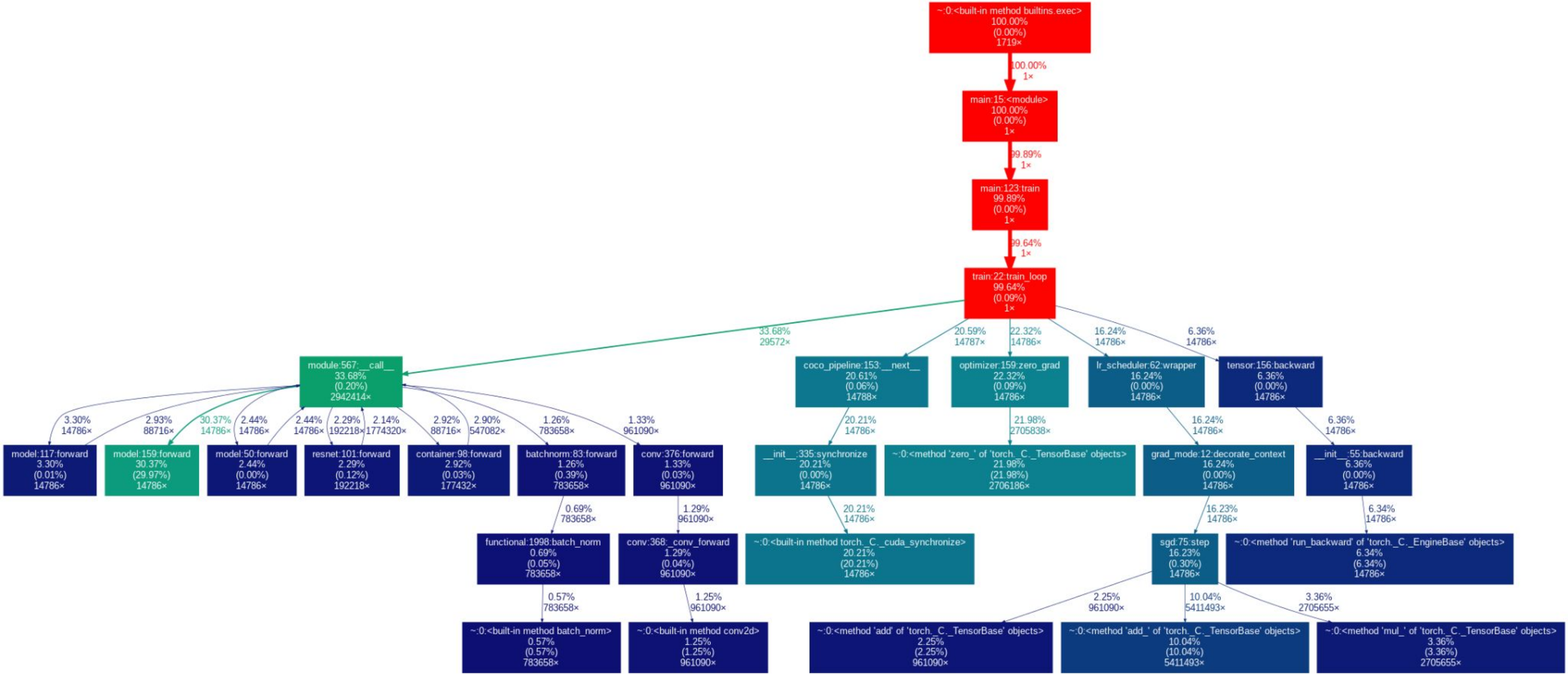


Answer to RQ2: The collected results suggest that better energy consumption and run-time performance—in most of the cases—yield better accuracy results as well. Overall, we find that TensorFlow has similar accuracy to PyTorch, under the configurations and parameter used in our study.

Identifying costly API calls – Spearman

Tuples	PyTorch	TensorFlow
PKG Energy–Run-Time	0.25	0.88
RAM Energy–Run-Time	0.88	0.94
GPU Energy–Run-Time	0.42	0.60

Deep Learning Frameworks Profiling



Symbols and their meanings

- ■ Complex calculations
- □ Complex Implementation
- ● Large data
- ◇ Device dependency
- — Unknown

PyTorch Inference

Model	Function Name	Ncalls	Run-Time	Cost	Type
Mask R-CNN	Torch.tensor	81,819	578	60.6%	◇
	Torch.nn.Conv2d	380,000	24	2.6%	■
	Torch.Tensor.nonzero	400,000	17	1.8%	▬
	Torch.Tensor.float	1,505,005	16	1.7%	●
	Torch.Tensor.to	105,312	16	1.7%	□
	Torch.Tensor.type	9,735,395	14	1.5%	▬

Takeaways

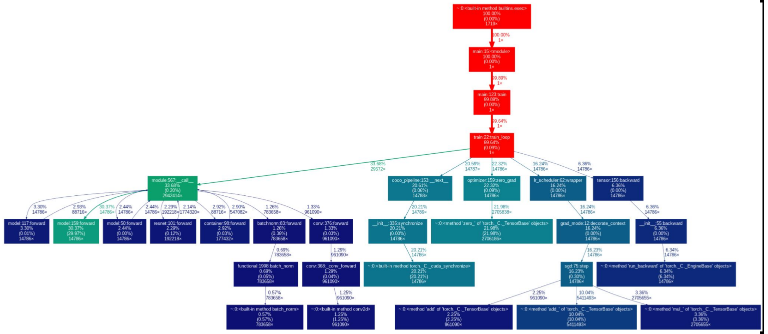
PyTorch

Table of Contents

PYTORCH DOCUMENTATION

PyTorch is an optimized tensor library for deep learning using GPUs and CPUs.

Features described in this documentation are classified by release status:



1. Test small programs energy consumption before running large scale experiments.
2. Use profiling approaches on small experiments to estimate resources to be used for large experiments afterwards.

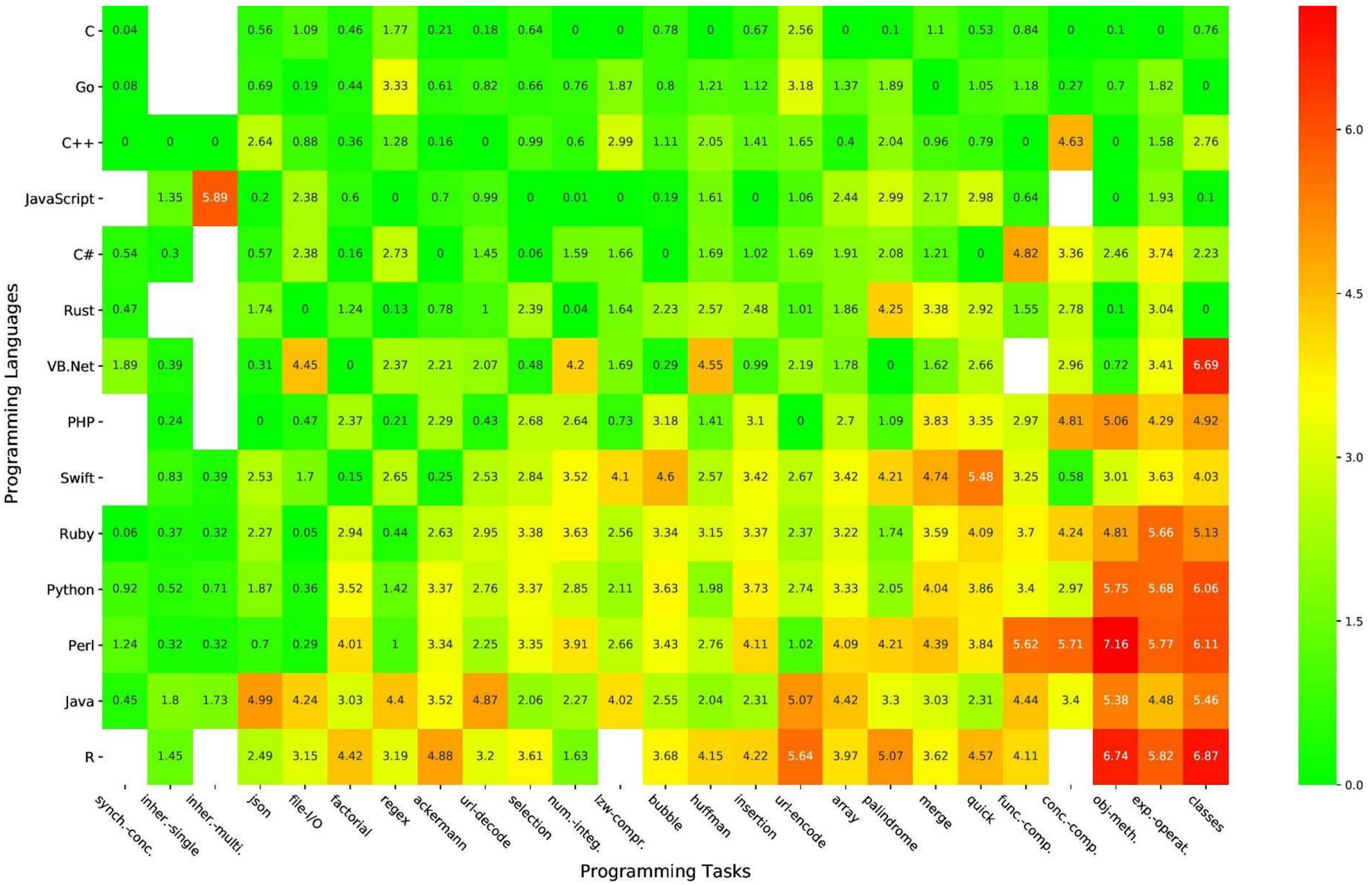
What are we missing as a community?

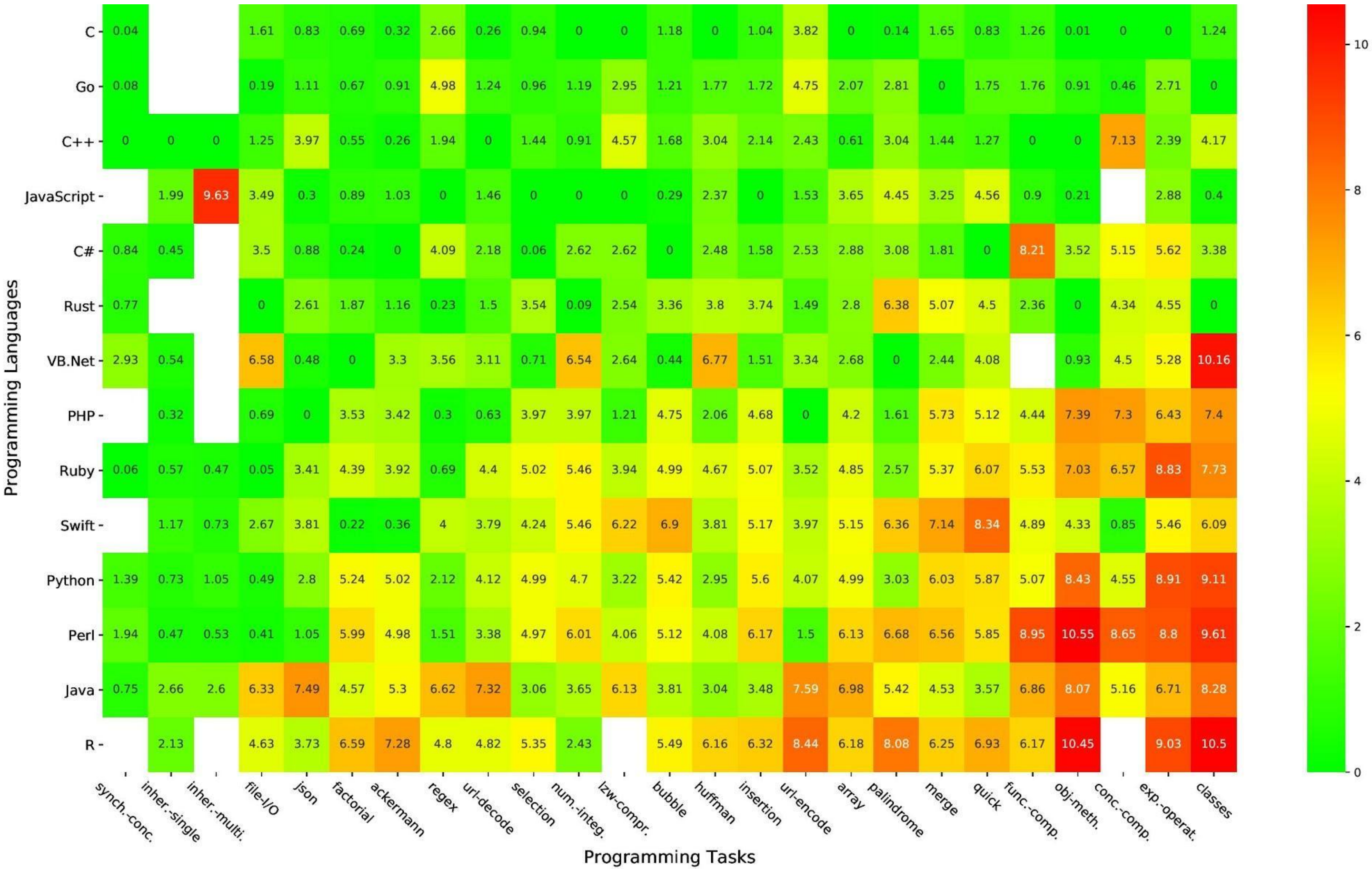
- How to convince developers which programing language, module, or framework to choose in order to reduce energy consumption?
- How different components will affect the energy and the run-time performance of applications in the long term?
- An easy way to collect measurements.

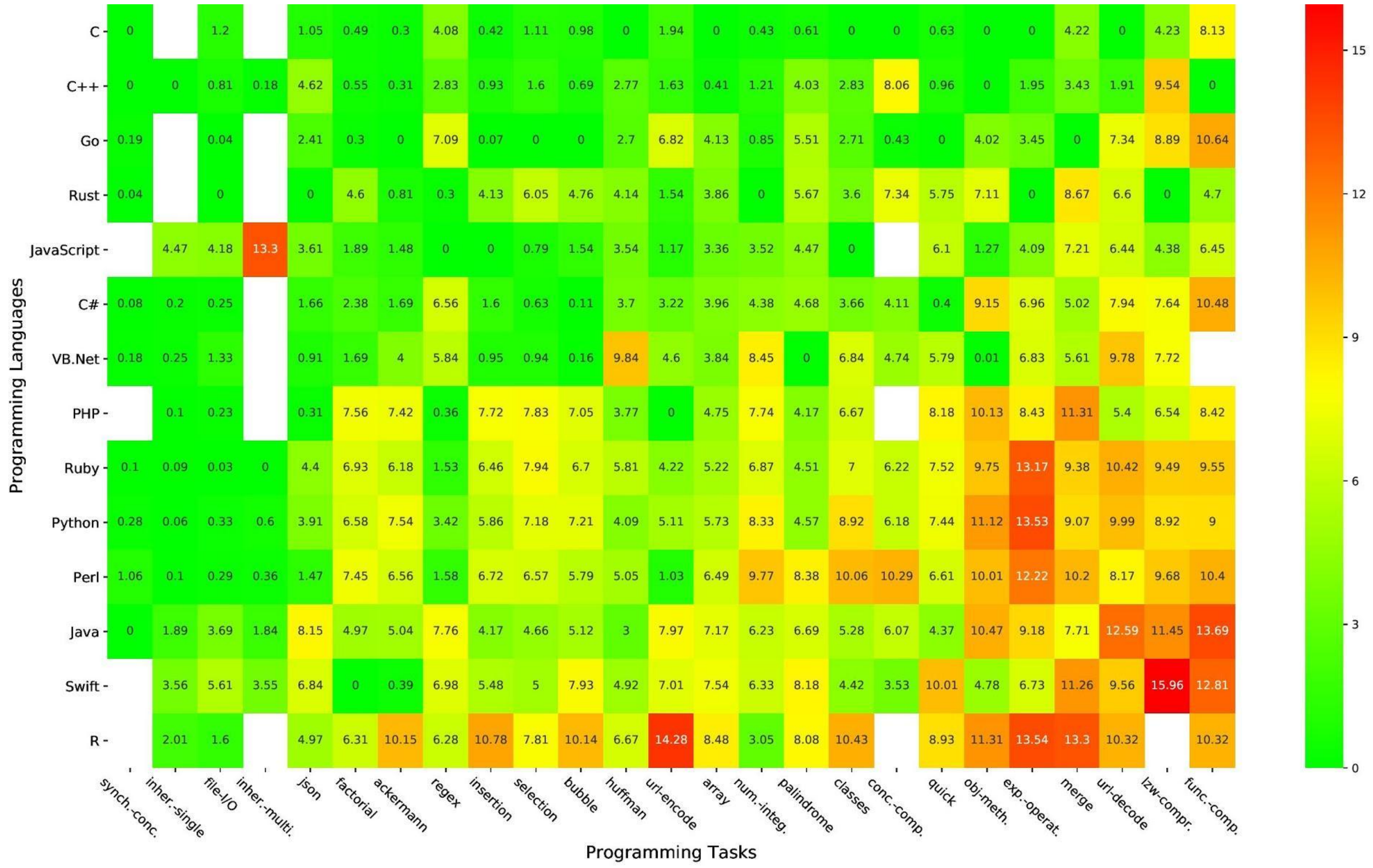
Thank you for your attention!!!

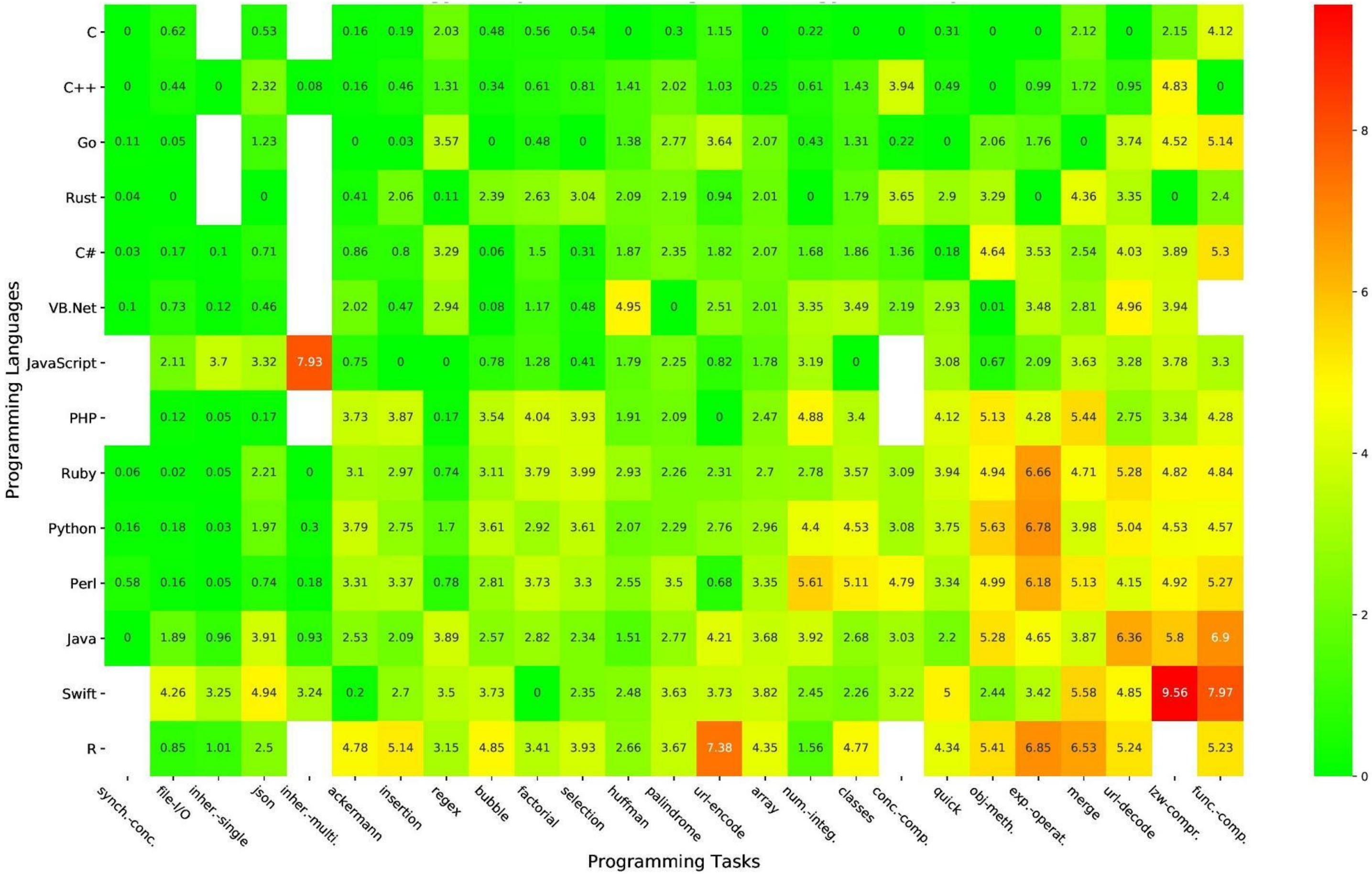
Email: stefanos1316@gmail.com
GitHub: stefanos1316

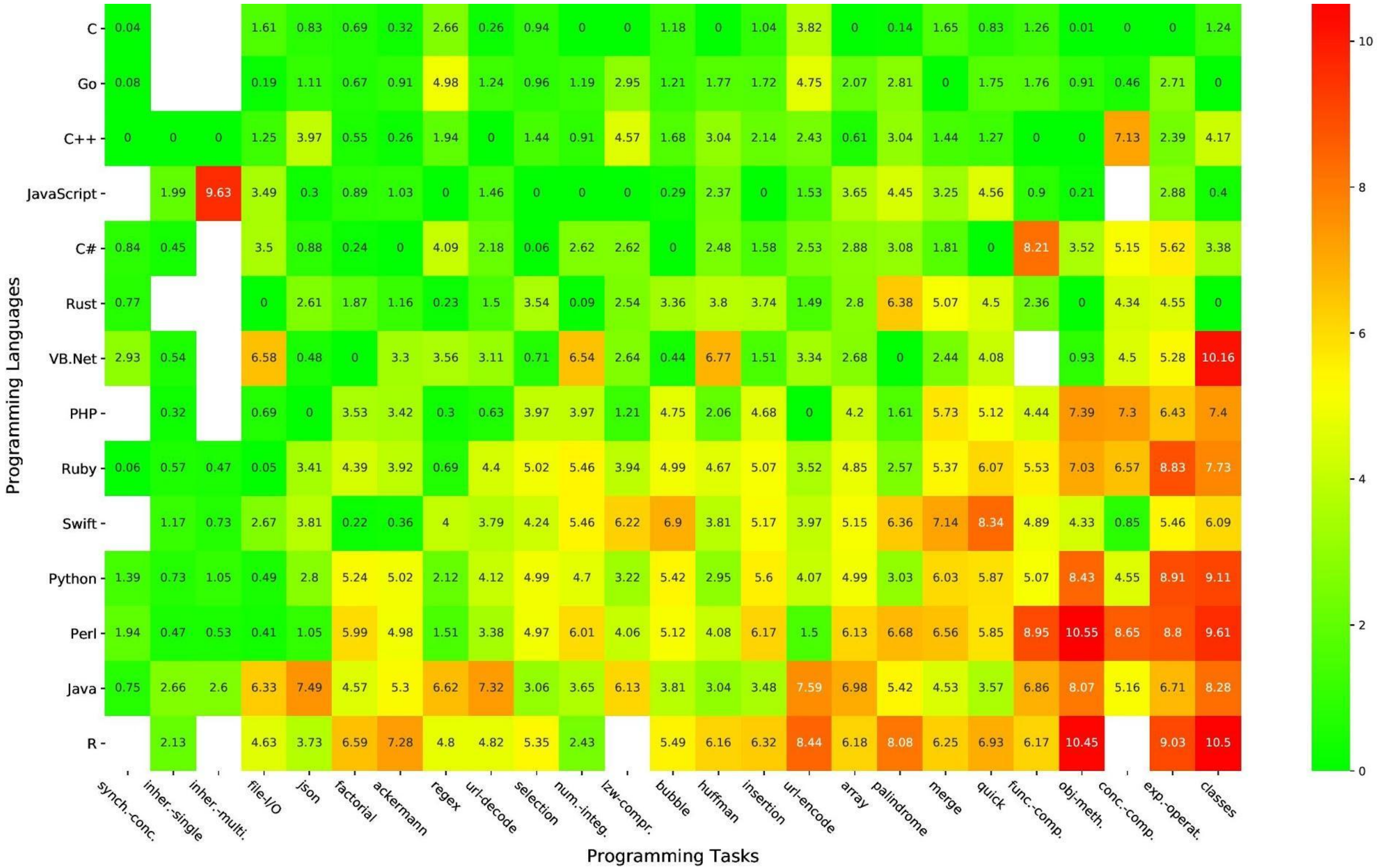
Backup slides

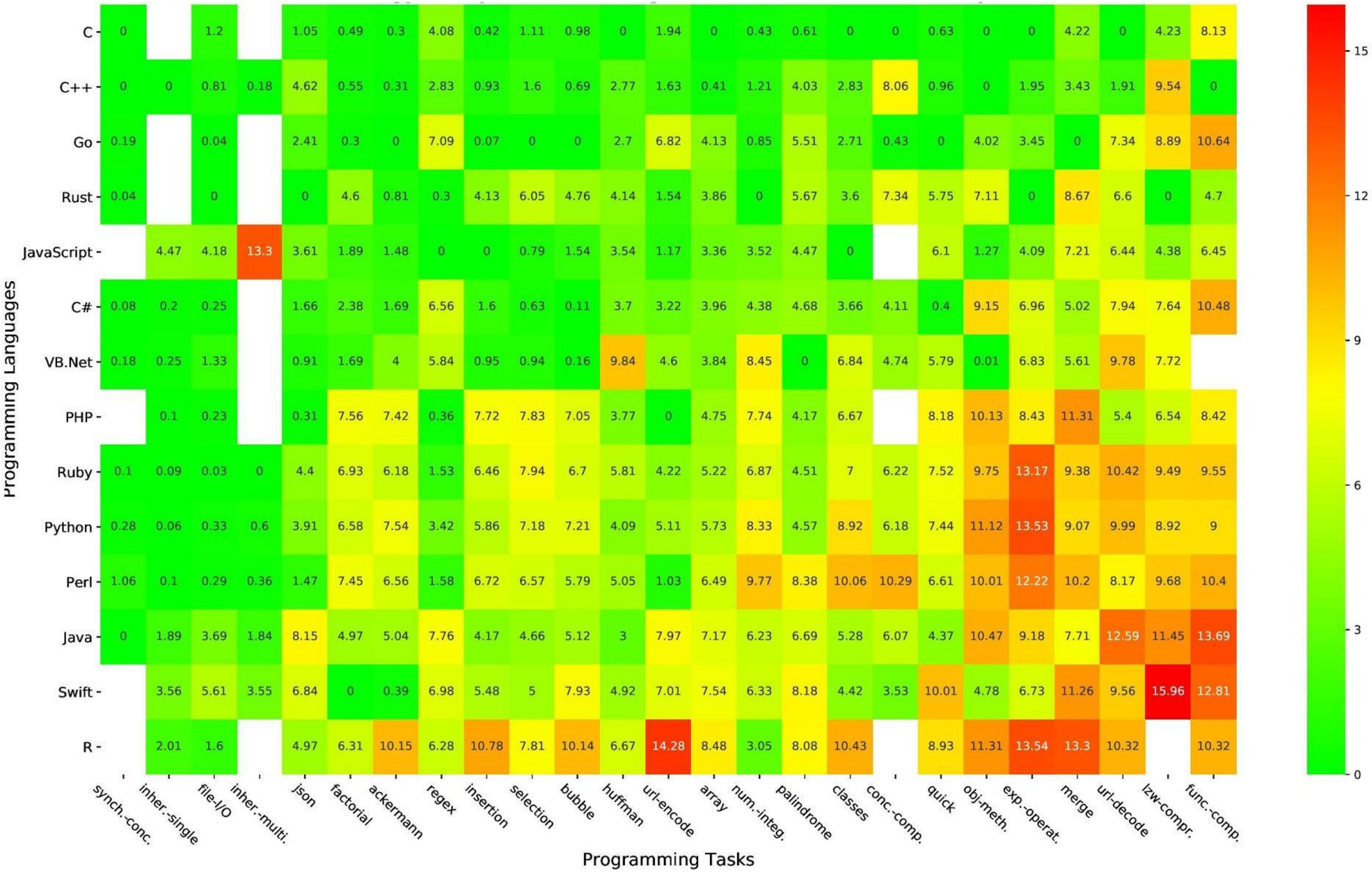

Programming Languages
Programming Tasks
synch.-conc. inher.-single inher.-multi. json file-I/O factorial regex ackermann url-decode selection num.-integ. lzw-compr. bubble huffman insertion url-encode array palindrome merge quick func.-comp. conc.-comp. obj-meth. exp.-operat. classes
C 0.04 0.56 1.09 0.46 1.77 0.21 0.18 0.64 0 0 0.78 0 0.67 2.56 0 0.1 1.1 0.53 0.84 0 0.1 0 0.76
Go 0.08 0.69 0.19 0.44 3.33 0.61 0.82 0.66 0.76 1.87 0.8 1.21 1.12 3.18 1.37 1.89 0 1.05 1.18 0.27 0.7 1.82 0
C++ 0 0 0 2.64 0.88 0.36 1.28 0.16 0 0.99 0.6 2.99 1.11 2.05 1.41 1.65 0.4 2.04 0.96 0.79 0 4.63 0 1.58 2.76
JavaScript 1.35 5.89 0.2 2.38 0.6 0 0.7 0.99 0 0.01 0 0.19 1.61 0 1.06 2.44 2.99 2.17 2.98 0.64 0 1.93 0.1
C# 0.54 0.3 0.57 2.38 0.16 2.73 0 1.45 0.06 1.59 1.66 0 1.69 1.02 1.69 1.91 2.08 1.21 0 4.82 3.36 2.46 3.74 2.23
Rust 0.47 1.74 0 1.24 0.13 0.78 1 2.39 0.04 1.64 2.23 2.57 2.48 1.01 1.86 4.25 3.38 2.92 1.55 2.78 0.1 3.04 0
VB.Net 1.89 0.39 0.31 4.45 0 2.37 2.21 2.07 0.48 4.2 1.69 0.29 4.55 0.99 2.19 1.78 0 1.62 2.66 2.96 0.72 3.41 6.69
PHP 0.24 0 0.47 2.37 0.21 2.29 0.43 2.68 2.64 0.73 3.18 1.41 3.1 0 2.7 1.09 3.83 3.35 2.97 4.81 5.06 4.29 4.92
Swift 0.83 0.39 2.53 1.7 0.15 2.65 0.25 2.53 2.84 3.52 4.1 4.6 2.57 3.42 2.67 3.42 4.21 4.74 5.48 3.25 0.58 3.01 3.63 4.03
Ruby 0.06 0.37 0.32 2.27 0.05 2.94 0.44 2.63 2.95 3.38 3.63 2.56 3.34 3.15 3.37 2.37 3.22 1.74 3.59 4.09 3.7 4.24 4.81 5.66 5.13
Python 0.92 0.52 0.71 1.87 0.36 3.52 1.42 3.37 2.76 3.37 2.85 2.11 3.63 1.98 3.73 2.74 3.33 2.05 4.04 3.86 3.4 2.97 5.75 5.68 6.06
Perl 1.24 0.32 0.32 0.7 0.29 4.01 1 3.34 2.25 3.35 3.91 2.66 3.43 2.76 4.11 1.02 4.09 4.21 4.39 3.84 5.62 5.71 7.16 5.77 6.11
Java 0.45 1.8 1.73 4.99 4.24 3.03 4.4 3.52 4.87 2.06 2.27 4.02 2.55 2.04 2.31 5.07 4.42 3.3 3.03 2.31 4.44 3.4 5.38 4.48 5.46
R 1.45 2.49 3.15 4.42 3.19 4.88 3.2 3.61 1.63 3.68 4.15 4.22 5.64 3.97 5.07 3.62 4.57 4.11 6.74 5.82 6.87
0.0 1.5 3.0 4.5 6.0


Programming Languages
C
Go
C++
JavaScript
C#
Rust
VB.Net
PHP
Ruby
Swift
Python
Perl
Java
R
synch.-conc.
inher.-single
inher.-multi.
file-I/O
json
factorial
ackermann
regex
url-decode
selection
num.-integ.
lzw-compr.
bubble
huffman
insertion
url-encode
array
palindrome
merge
quick
func.-comp.
obj-meth.
conc.-comp.
exp.-operat.
classes
Programming Tasks
0
2
4
6
8
10


Programming Languages
Programming Tasks
C 0 1.2 1.05 0.49 0.3 4.08 0.42 1.11 0.98 0 1.94 0 0.43 0.61 0 0 0.63 0 0 4.22 0 4.23 8.13
C++ 0 0 0.81 0.18 4.62 0.55 0.31 2.83 0.93 1.6 0.69 2.77 1.63 0.41 1.21 4.03 2.83 8.06 0.96 0 1.95 3.43 1.91 9.54 0
Go 0.19 0.04 2.41 0.3 0 7.09 0.07 0 0 2.7 6.82 4.13 0.85 5.51 2.71 0.43 0 4.02 3.45 0 7.34 8.89 10.64
Rust 0.04 0 0 4.6 0.81 0.3 4.13 6.05 4.76 4.14 1.54 3.86 0 5.67 3.6 7.34 5.75 7.11 0 8.67 6.6 0 4.7
JavaScript 4.47 4.18 13.3 3.61 1.89 1.48 0 0 0.79 1.54 3.54 1.17 3.36 3.52 4.47 0 6.1 1.27 4.09 7.21 6.44 4.38 6.45
C# 0.08 0.2 0.25 1.66 2.38 1.69 6.56 1.6 0.63 0.11 3.7 3.22 3.96 4.38 4.68 3.66 4.11 0.4 9.15 6.96 5.02 7.94 7.64 10.48
VB.Net 0.18 0.25 1.33 0.91 1.69 4 5.84 0.95 0.94 0.16 9.84 4.6 3.84 8.45 0 6.84 4.74 5.79 0.01 6.83 5.61 9.78 7.72
PHP 0.1 0.23 0.31 7.56 7.42 0.36 7.72 7.83 7.05 3.77 0 4.75 7.74 4.17 6.67 8.18 10.13 8.43 11.31 5.4 6.54 8.42
Ruby 0.1 0.09 0.03 0 4.4 6.93 6.18 1.53 6.46 7.94 6.7 5.81 4.22 5.22 6.87 4.51 7 6.22 7.52 9.75 13.17 9.38 10.42 9.49 9.55
Python 0.28 0.06 0.33 0.6 3.91 6.58 7.54 3.42 5.86 7.18 7.21 4.09 5.11 5.73 8.33 4.57 8.92 6.18 7.44 11.12 13.53 9.07 9.99 8.92 9
Perl 1.06 0.1 0.29 0.36 1.47 7.45 6.56 1.58 6.72 6.57 5.79 5.05 1.03 6.49 9.77 8.38 10.06 10.29 6.61 10.01 12.22 10.2 8.17 9.68 10.4
Java 0 1.89 3.69 1.84 8.15 4.97 5.04 7.76 4.17 4.66 5.12 3 7.97 7.17 6.23 6.69 5.28 6.07 4.37 10.47 9.18 7.71 12.59 11.45 13.69
Swift 3.56 5.61 3.55 6.84 0 0.39 6.98 5.48 5 7.93 4.92 7.01 7.54 6.33 8.18 4.42 3.53 10.01 4.78 6.73 11.26 9.56 15.96 12.81
R 2.01 1.6 4.97 6.31 10.15 6.28 10.78 7.81 10.14 6.67 14.28 8.48 3.05 8.08 10.43 8.93 11.31 13.54 13.3 10.32 10.32
synch.-conc. inher.-single file-I/O inher.-multi. json factorial ackermann regex insertion selection bubble huffman url-encode array num.-integ. palindrome classes conc.-comp. quick obj-meth. exp.-operat. merge url-decode lzw-compr. func.-comp.
0 3 6 9 12 15

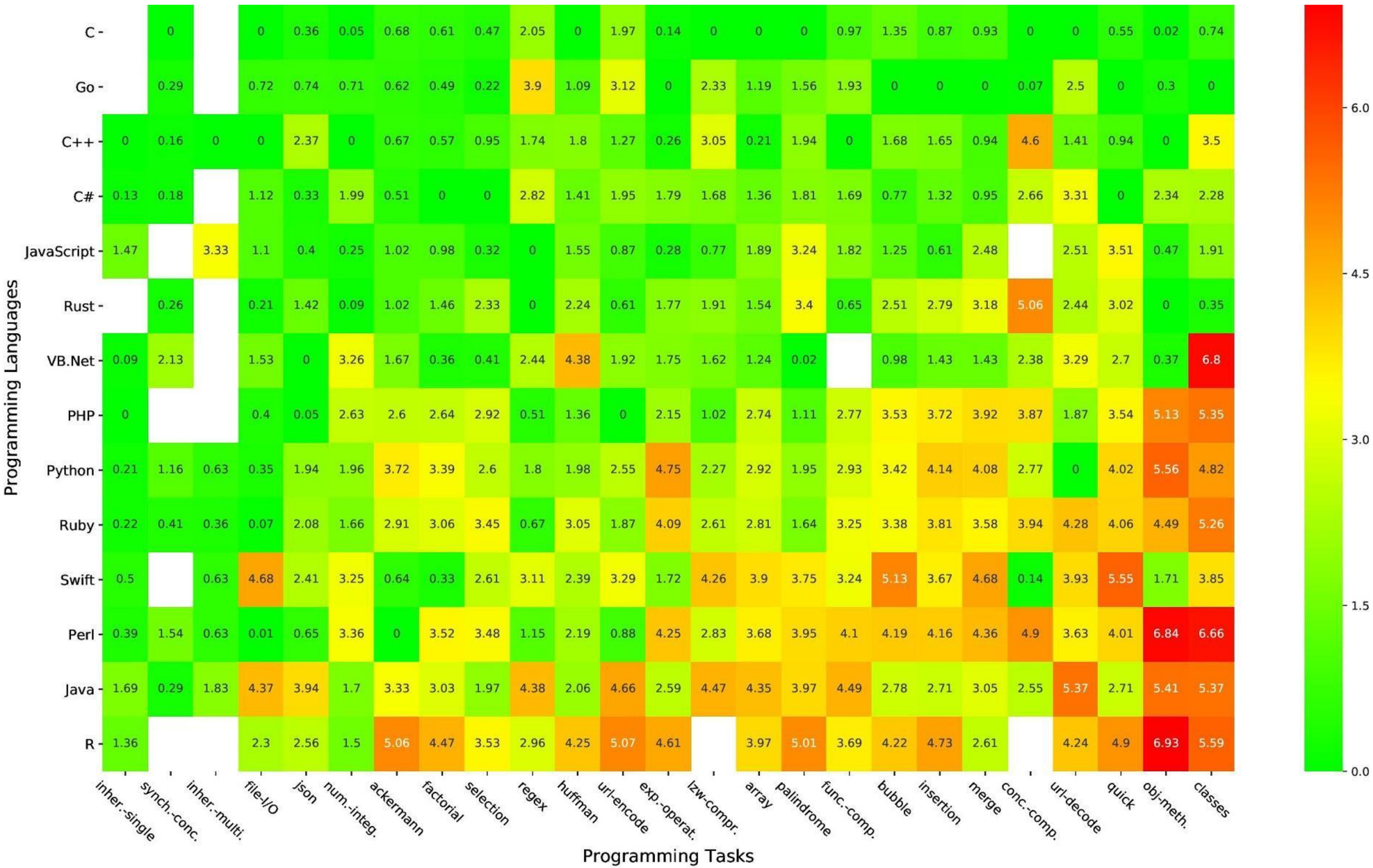

Programming Languages
C 0 0.62 0.53 0.16 0.19 2.03 0.48 0.56 0.54 0 0.3 1.15 0 0.22 0 0 0.31 0 0 2.12 0 2.15 4.12
C++ 0 0.44 0 2.32 0.08 0.16 0.46 1.31 0.34 0.61 0.81 1.41 2.02 1.03 0.25 0.61 1.43 3.94 0.49 0 0.99 1.72 0.95 4.83 0
Go 0.11 0.05 1.23 0 0.03 3.57 0 0.48 0 1.38 2.77 3.64 2.07 0.43 1.31 0.22 0 2.06 1.76 0 3.74 4.52 5.14
Rust 0.04 0 0 0.41 2.06 0.11 2.39 2.63 3.04 2.09 2.19 0.94 2.01 0 1.79 3.65 2.9 3.29 0 4.36 3.35 0 2.4
C# 0.03 0.17 0.1 0.71 0.86 0.8 3.29 0.06 1.5 0.31 1.87 2.35 1.82 2.07 1.68 1.86 1.36 0.18 4.64 3.53 2.54 4.03 3.89 5.3
VB.Net 0.1 0.73 0.12 0.46 2.02 0.47 2.94 0.08 1.17 0.48 4.95 0 2.51 2.01 3.35 3.49 2.19 2.93 0.01 3.48 2.81 4.96 3.94
JavaScript 2.11 3.7 3.32 7.93 0.75 0 0 0.78 1.28 0.41 1.79 2.25 0.82 1.78 3.19 0 3.08 0.67 2.09 3.63 3.28 3.78 3.3
PHP 0.12 0.05 0.17 3.73 3.87 0.17 3.54 4.04 3.93 1.91 2.09 0 2.47 4.88 3.4 4.12 5.13 4.28 5.44 2.75 3.34 4.28
Ruby 0.06 0.02 0.05 2.21 0 3.1 2.97 0.74 3.11 3.79 3.99 2.93 2.26 2.31 2.7 2.78 3.57 3.09 3.94 4.94 6.66 4.71 5.28 4.82 4.84
Python 0.16 0.18 0.03 1.97 0.3 3.79 2.75 1.7 3.61 2.92 3.61 2.07 2.29 2.76 2.96 4.4 4.53 3.08 3.75 5.63 6.78 3.98 5.04 4.53 4.57
Perl 0.58 0.16 0.05 0.74 0.18 3.31 3.37 0.78 2.81 3.73 3.3 2.55 3.5 0.68 3.35 5.61 5.11 4.79 3.34 4.99 6.18 5.13 4.15 4.92 5.27
Java 0 1.89 0.96 3.91 0.93 2.53 2.09 3.89 2.57 2.82 2.34 1.51 2.77 4.21 3.68 3.92 2.68 3.03 2.2 5.28 4.65 3.87 6.36 5.8 6.9
Swift 4.26 3.25 4.94 3.24 0.2 2.7 3.5 3.73 0 2.35 2.48 3.63 3.73 3.82 2.45 2.26 3.22 5 2.44 3.42 5.58 4.85 9.56 7.97
R 0.85 1.01 2.5 4.78 5.14 3.15 4.85 3.41 3.93 2.66 3.67 7.38 4.35 1.56 4.77 4.34 5.41 6.85 6.53 5.24 5.23
synch.-conc. file-I/O inher.-single json inher.-multi. ackermann insertion regex bubble factorial selection huffman palindrome url-encode array num.-integ. classes conc.-comp. quick obj-meth. exp.-operat. merge url-decode lzw-compr. func.-comp.
Programming Tasks
0 2 4 6 8

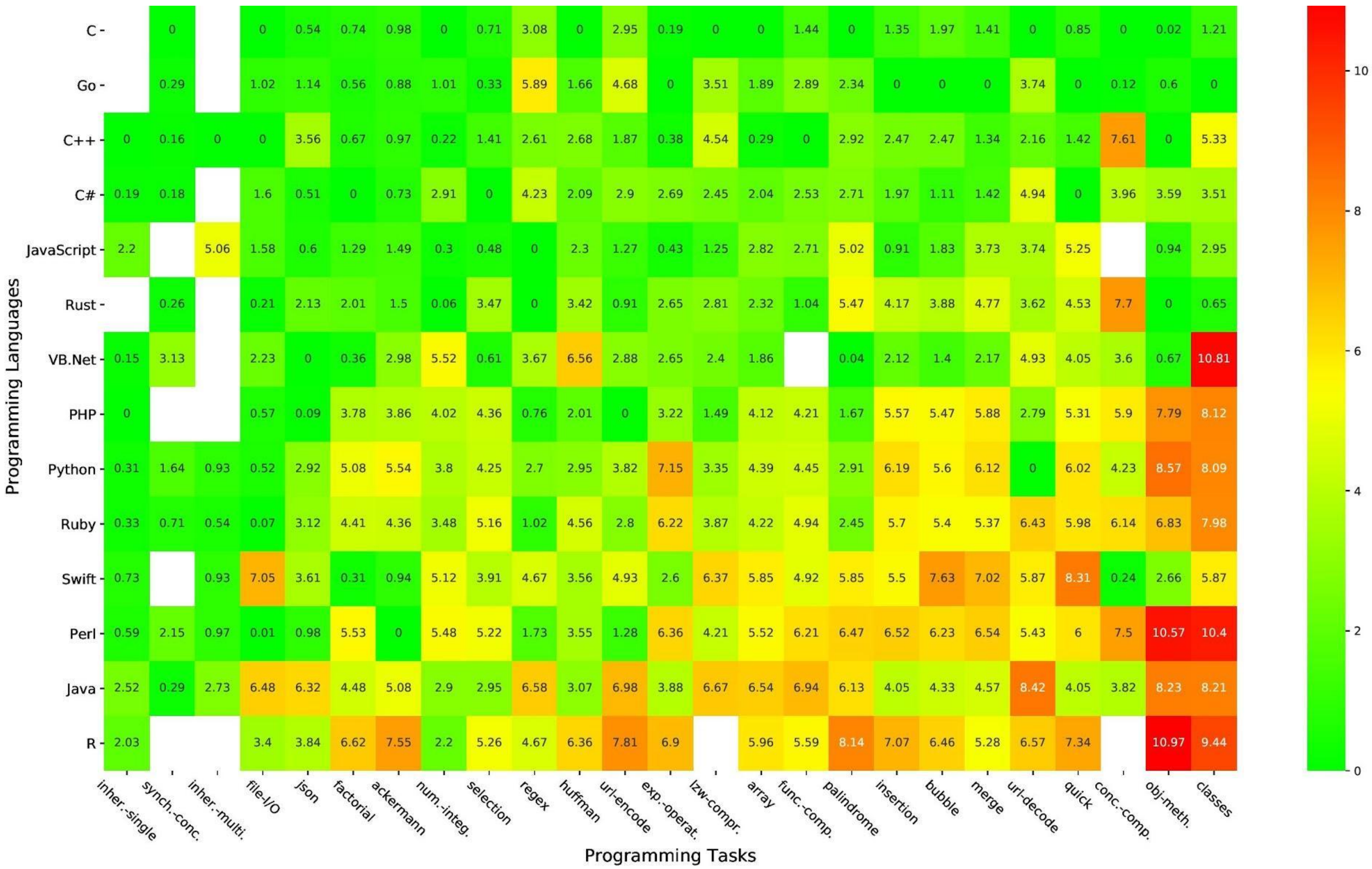

Programming Languages
Programming Tasks
synch.-conc. inher.-single inher.-multi. file-I/O json factorial ackermann regex url-decode selection num.-integ. lzw-compr. bubble huffman insertion url-encode array palindrome merge quick func.-comp. obj-meth. conc.-comp. exp.-operat. classes
C 0.04 1.61 0.83 0.69 0.32 2.66 0.26 0.94 0 0 1.18 0 1.04 3.82 0 0.14 1.65 0.83 1.26 0.01 0 0 1.24
Go 0.08 0.19 1.11 0.67 0.91 4.98 1.24 0.96 1.19 2.95 1.21 1.77 1.72 4.75 2.07 2.81 0 1.75 1.76 0.91 0.46 2.71 0
C++ 0 0 0 1.25 3.97 0.55 0.26 1.94 0 1.44 0.91 4.57 1.68 3.04 2.14 2.43 0.61 3.04 1.44 1.27 0 0 7.13 2.39 4.17
JavaScript 1.99 9.63 3.49 0.3 0.89 1.03 0 1.46 0 0 0 0.29 2.37 0 1.53 3.65 4.45 3.25 4.56 0.9 0.21 2.88 0.4
C# 0.84 0.45 3.5 0.88 0.24 0 4.09 2.18 0.06 2.62 2.62 0 2.48 1.58 2.53 2.88 3.08 1.81 0 8.21 3.52 5.15 5.62 3.38
Rust 0.77 0 2.61 1.87 1.16 0.23 1.5 3.54 0.09 2.54 3.36 3.8 3.74 1.49 2.8 6.38 5.07 4.5 2.36 0 4.34 4.55 0
VB.Net 2.93 0.54 6.58 0.48 0 3.3 3.56 3.11 0.71 6.54 2.64 0.44 6.77 1.51 3.34 2.68 0 2.44 4.08 0.93 4.5 5.28 10.16
PHP 0.32 0.69 0 3.53 3.42 0.3 0.63 3.97 3.97 1.21 4.75 2.06 4.68 0 4.2 1.61 5.73 5.12 4.44 7.39 7.3 6.43 7.4
Ruby 0.06 0.57 0.47 0.05 3.41 4.39 3.92 0.69 4.4 5.02 5.46 3.94 4.99 4.67 5.07 3.52 4.85 2.57 5.37 6.07 5.53 7.03 6.57 8.83 7.73
Swift 1.17 0.73 2.67 3.81 0.22 0.36 4 3.79 4.24 5.46 6.22 6.9 3.81 5.17 3.97 5.15 6.36 7.14 8.34 4.89 4.33 0.85 5.46 6.09
Python 1.39 0.73 1.05 0.49 2.8 5.24 5.02 2.12 4.12 4.99 4.7 3.22 5.42 2.95 5.6 4.07 4.99 3.03 6.03 5.87 5.07 8.43 4.55 8.91 9.11
Perl 1.94 0.47 0.53 0.41 1.05 5.99 4.98 1.51 3.38 4.97 6.01 4.06 5.12 4.08 6.17 1.5 6.13 6.68 6.56 5.85 8.95 10.55 8.65 8.8 9.61
Java 0.75 2.66 2.6 6.33 7.49 4.57 5.3 6.62 7.32 3.06 3.65 6.13 3.81 3.04 3.48 7.59 6.98 5.42 4.53 3.57 6.86 8.07 5.16 6.71 8.28
R 2.13 4.63 3.73 6.59 7.28 4.8 4.82 5.35 2.43 5.49 6.16 6.32 8.44 6.18 8.08 6.25 6.93 6.17 10.45 9.03 10.5
0 2 4 6 8 10

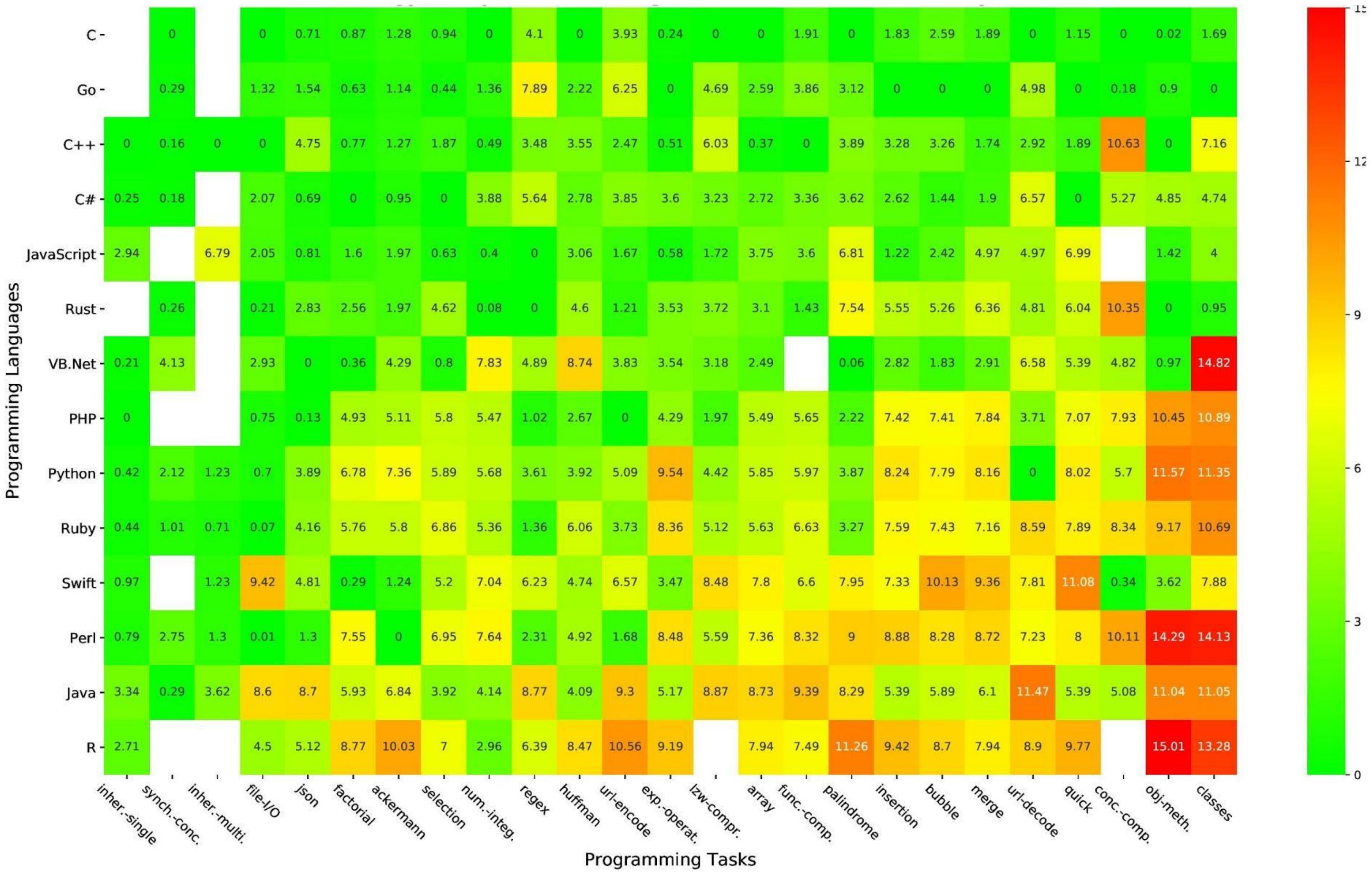


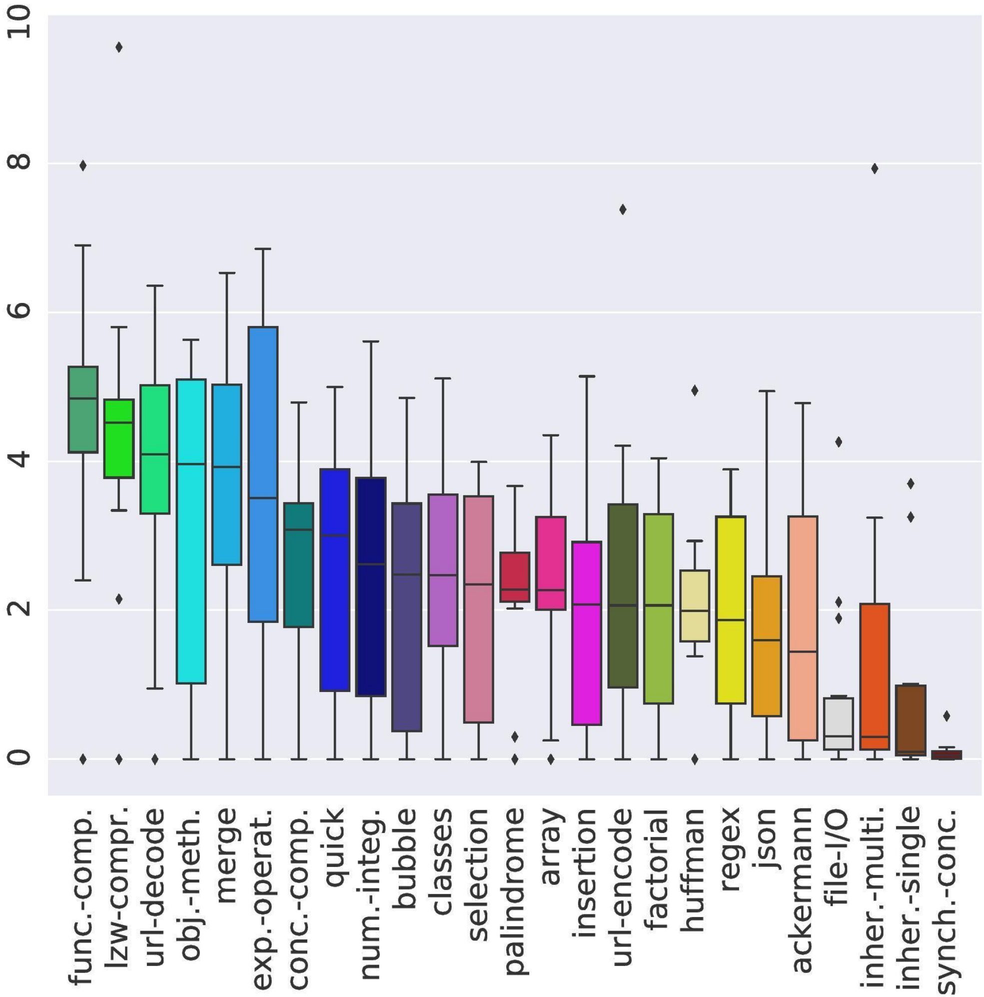
Programming Languages	synch.-conc.	inher.-single	file-I/O	inher.-multi.	json	factorial	ackermann	regex	insertion	selection	bubble	huffman	url-encode
C	0		1.2		1.05	0.49	0.3	4.08	0.42	1.11	0.98	0	1.94
C++	0	0	0.81	0.18	4.62	0.55	0.31	2.83	0.93	1.6	0.69	2.77	1.63
Go	0.19		0.04		2.41	0.3	0	7.09	0.07	0	0	2.7	6.82
Rust	0.04		0		0	4.6	0.81	0.3	4.13	6.05	4.76	4.14	1.54
JavaScript		4.47	4.18	13.3	3.61	1.89	1.48	0	0	0.79	1.54	3.54	1.17
C#	0.08	0.2	0.25		1.66	2.38	1.69	6.56	1.6	0.63	0.11	3.7	3.22
VB.Net	0.18	0.25	1.33		0.91	1.69	4	5.84	0.95	0.94	0.16	9.84	4.6
PHP		0.1	0.23		0.31	7.56	7.42	0.36	7.72	7.83	7.05	3.77	0
Ruby	0.1	0.09	0.03	0	4.4	6.93	6.18	1.53	6.46	7.94	6.7	5.81	4.22
Python	0.28	0.06	0.33	0.6	3.91	6.58	7.54	3.42	5.86	7.18	7.21	4.09	5.11
Perl	1.06	0.1	0.29	0.36	1.47	7.45	6.56	1.58	6.72	6.57	5.79	5.05	1.03
Java	0	1.89	3.69	1.84	8.15	4.97	5.04	7.76	4.17	4.66	5.12	3	7.97
Swift		3.56	5.61	3.55	6.84	0	0.39	6.98	5.48	5	7.93	4.92	7.01
R		2.01	1.6		4.97	6.31	10.15	6.28	10.78	7.81	10.14	6.67	14.28

Programming Languages	array	num.-integ.	palindrome	classes	conc.-comp.	quick	obj-meth.	exp.-operat.	merge	url-decode	lzw-compr.	func.-comp.
C	0	0.43	0.61	0	0	0.63	0	0	4.22	0	4.23	8.13
C++	0.41	1.21	4.03	2.83	8.06	0.96	0	1.95	3.43	1.91	9.54	0
Go	4.13	0.85	5.51	2.71	0.43	0	4.02	3.45	0	7.34	8.89	10.64
Rust	3.86	0	5.67	3.6	7.34	5.75	7.11	0	8.67	6.6	0	4.7
JavaScript	3.36	3.52	4.47	0		6.1	1.27	4.09	7.21	6.44	4.38	6.45
C#	3.96	4.38	4.68	3.66	4.11	0.4	9.15	6.96	5.02	7.94	7.64	10.48
VB.Net	3.84	8.45	0	6.84	4.74	5.79	0.01	6.83	5.61	9.78	7.72	
PHP	4.75	7.74	4.17	6.67		8.18	10.13	8.43	11.31	5.4	6.54	8.42
Ruby	5.22	6.87	4.51	7	6.22	7.52	9.75	13.17	9.38	10.42	9.49	9.55
Python	5.73	8.33	4.57	8.92	6.18	7.44	11.12	13.53	9.07	9.99	8.92	9
Perl	6.49	9.77	8.38	10.06	10.29	6.61	10.01	12.22	10.2	8.17	9.68	10.4
Java	7.17	6.23	6.69	5.28	6.07	4.37	10.47	9.18	7.71	12.59	11.45	13.69
Swift	7.54	6.33	8.18	4.42	3.53	10.01	4.78	6.73	11.26	9.56	15.96	12.81
R	8.48	3.05	8.08	10.43		8.93	11.31	13.54	13.3	10.32		10.32

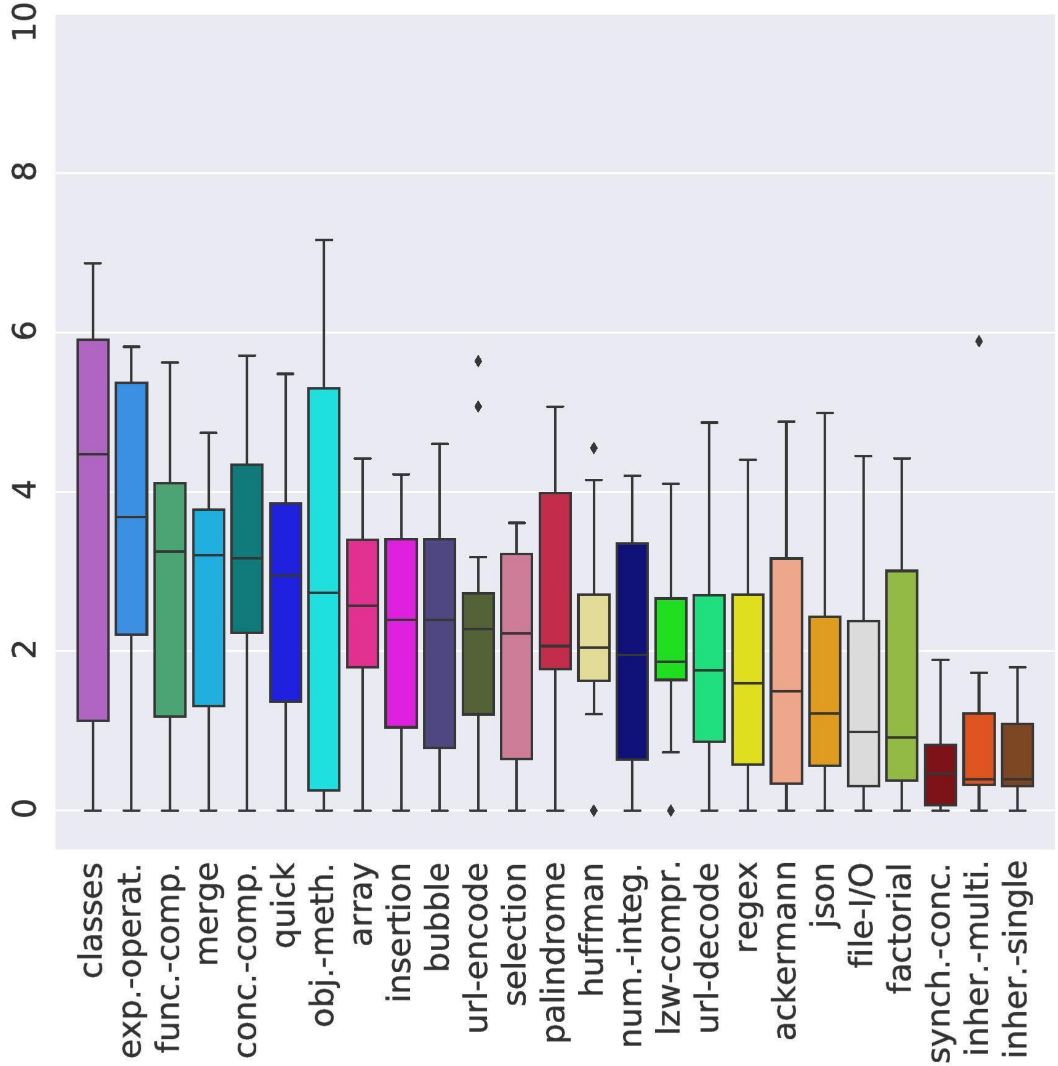

Programming Languages
Programming Tasks
C
Go
C++
C#
JavaScript
Rust
VB.Net
PHP
Python
Ruby
Swift
Perl
Java
R
inher.-single
synch.-conc.
inher.-multi.
file-I/O
json
num.-integ.
ackermann
factorial
selection
regex
huffman
url-encode
exp.-operat.
lzw-compr.
array
palindrome
func.-comp.
bubble
insertion
merge
conc.-comp.
url-decode
quick
obj-meth.
classes
0 0 0.36 0.05 0.68 0.61 0.47 2.05 0 1.97 0.14 0 0 0 0.97 1.35 0.87 0.93 0 0 0.55 0.02 0.74
0.29 0.72 0.74 0.71 0.62 0.49 0.22 3.9 1.09 3.12 0 2.33 1.19 1.56 1.93 0 0 0 0.07 2.5 0 0.3 0
0 0.16 0 0 2.37 0 0.67 0.57 0.95 1.74 1.8 1.27 0.26 3.05 0.21 1.94 0 1.68 1.65 0.94 4.6 1.41 0.94 0 3.5
0.13 0.18 1.12 0.33 1.99 0.51 0 0 2.82 1.41 1.95 1.79 1.68 1.36 1.81 1.69 0.77 1.32 0.95 2.66 3.31 0 2.34 2.28
1.47 3.33 1.1 0.4 0.25 1.02 0.98 0.32 0 1.55 0.87 0.28 0.77 1.89 3.24 1.82 1.25 0.61 2.48 2.51 3.51 0.47 1.91
0.26 0.21 1.42 0.09 1.02 1.46 2.33 0 2.24 0.61 1.77 1.91 1.54 3.4 0.65 2.51 2.79 3.18 5.06 2.44 3.02 0 0.35
0.09 2.13 1.53 0 3.26 1.67 0.36 0.41 2.44 4.38 1.92 1.75 1.62 1.24 0.02 0.98 1.43 1.43 2.38 3.29 2.7 0.37 6.8
0 0.4 0.05 2.63 2.6 2.64 2.92 0.51 1.36 0 2.15 1.02 2.74 1.11 2.77 3.53 3.72 3.92 3.87 1.87 3.54 5.13 5.35
0.21 1.16 0.63 0.35 1.94 1.96 3.72 3.39 2.6 1.8 1.98 2.55 4.75 2.27 2.92 1.95 2.93 3.42 4.14 4.08 2.77 0 4.02 5.56 4.82
0.22 0.41 0.36 0.07 2.08 1.66 2.91 3.06 3.45 0.67 3.05 1.87 4.09 2.61 2.81 1.64 3.25 3.38 3.81 3.58 3.94 4.28 4.06 4.49 5.26
0.5 0.63 4.68 2.41 3.25 0.64 0.33 2.61 3.11 2.39 3.29 1.72 4.26 3.9 3.75 3.24 5.13 3.67 4.68 0.14 3.93 5.55 1.71 3.85
0.39 1.54 0.63 0.01 0.65 3.36 0 3.52 3.48 1.15 2.19 0.88 4.25 2.83 3.68 3.95 4.1 4.19 4.16 4.36 4.9 3.63 4.01 6.84 6.66
1.69 0.29 1.83 4.37 3.94 1.7 3.33 3.03 1.97 4.38 2.06 4.66 2.59 4.47 4.35 3.97 4.49 2.78 2.71 3.05 2.55 5.37 2.71 5.41 5.37
1.36 2.3 2.56 1.5 5.06 4.47 3.53 2.96 4.25 5.07 4.61 3.97 5.01 3.69 4.22 4.73 2.61 4.24 4.9 6.93 5.59
0.0
1.5
3.0
4.5
6.0


Programming Languages
C
Go
C++
C#
JavaScript
Rust
VB.Net
PHP
Python
Ruby
Swift
Perl
Java
R
Programming Tasks
inher.-single
synch.-conc.
inher.-multi.
file-I/O
json
factorial
ackermann
num.-integ.
selection
regex
huffman
url-encode
exp.-operat.
lzw-compr.
array
func.-comp.
palindrome
insertion
bubble
merge
url-decode
quick
conc.-comp.
obj-meth.
classes
0
2
4
6
8
10

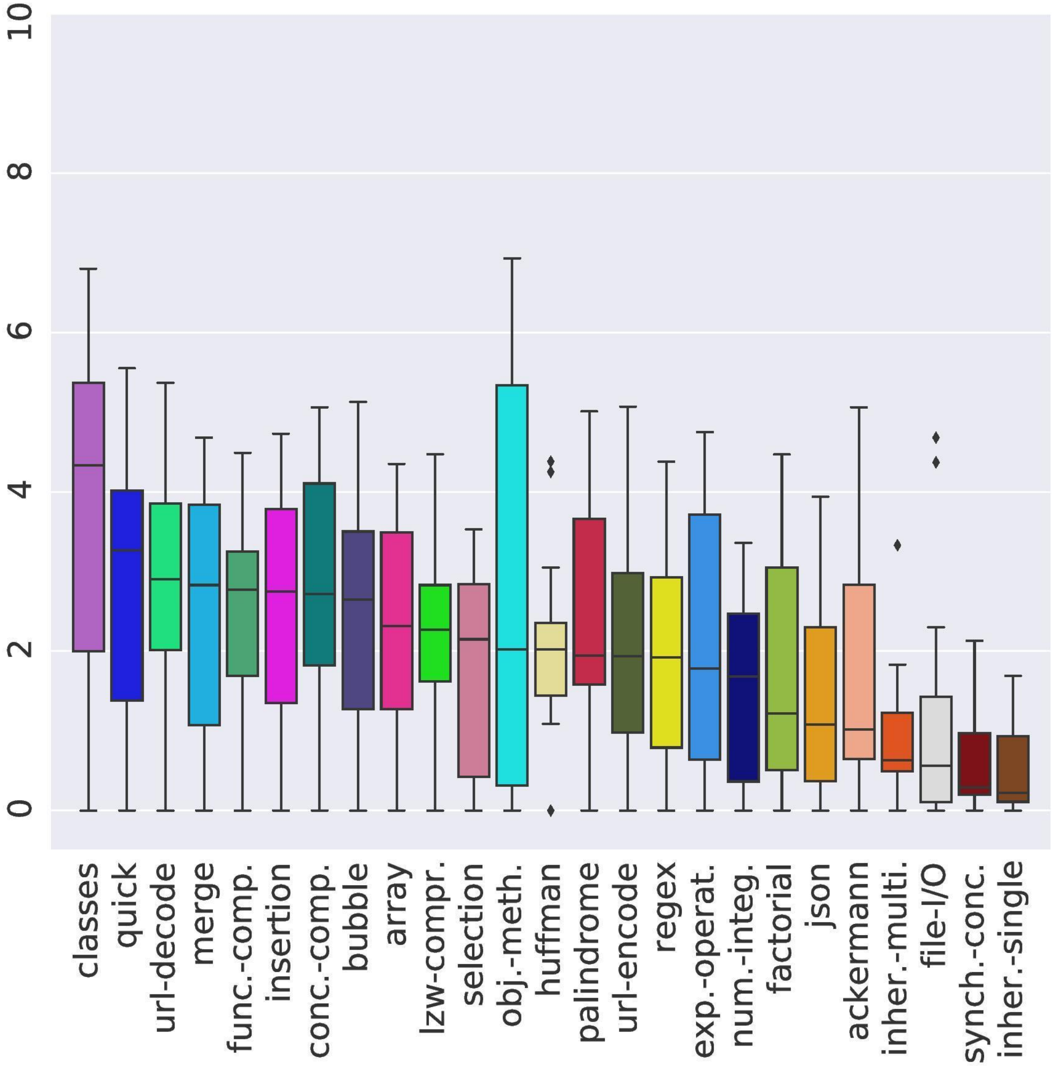

Programming Languages
Programming Tasks
C 0 0 0.71 0.87 1.28 0.94 0 4.1 0 3.93 0.24 0 0 1.91 0 1.83 2.59 1.89 0 1.15 0 0.02 1.69
Go 0.29 1.32 1.54 0.63 1.14 0.44 1.36 7.89 2.22 6.25 0 4.69 2.59 3.86 3.12 0 0 0 4.98 0 0.18 0.9 0
C++ 0 0.16 0 0 4.75 0.77 1.27 1.87 0.49 3.48 3.55 2.47 0.51 6.03 0.37 0 3.89 3.28 3.26 1.74 2.92 1.89 10.63 0 7.16
C# 0.25 0.18 2.07 0.69 0 0.95 0 3.88 5.64 2.78 3.85 3.6 3.23 2.72 3.36 3.62 2.62 1.44 1.9 6.57 0 5.27 4.85 4.74
JavaScript 2.94 6.79 2.05 0.81 1.6 1.97 0.63 0.4 0 3.06 1.67 0.58 1.72 3.75 3.6 6.81 1.22 2.42 4.97 4.97 6.99 1.42 4
Rust 0.26 0.21 2.83 2.56 1.97 4.62 0.08 0 4.6 1.21 3.53 3.72 3.1 1.43 7.54 5.55 5.26 6.36 4.81 6.04 10.35 0 0.95
VB.Net 0.21 4.13 2.93 0 0.36 4.29 0.8 7.83 4.89 8.74 3.83 3.54 3.18 2.49 0.06 2.82 1.83 2.91 6.58 5.39 4.82 0.97 14.82
PHP 0 0.75 0.13 4.93 5.11 5.8 5.47 1.02 2.67 0 4.29 1.97 5.49 5.65 2.22 7.42 7.41 7.84 3.71 7.07 7.93 10.45 10.89
Python 0.42 2.12 1.23 0.7 3.89 6.78 7.36 5.89 5.68 3.61 3.92 5.09 9.54 4.42 5.85 5.97 3.87 8.24 7.79 8.16 0 8.02 5.7 11.57 11.35
Ruby 0.44 1.01 0.71 0.07 4.16 5.76 5.8 6.86 5.36 1.36 6.06 3.73 8.36 5.12 5.63 6.63 3.27 7.59 7.43 7.16 8.59 7.89 8.34 9.17 10.69
Swift 0.97 1.23 9.42 4.81 0.29 1.24 5.2 7.04 6.23 4.74 6.57 3.47 8.48 7.8 6.6 7.95 7.33 10.13 9.36 7.81 11.08 0.34 3.62 7.88
Perl 0.79 2.75 1.3 0.01 1.3 7.55 0 6.95 7.64 2.31 4.92 1.68 8.48 5.59 7.36 8.32 9 8.88 8.28 8.72 7.23 8 10.11 14.29 14.13
Java 3.34 0.29 3.62 8.6 8.7 5.93 6.84 3.92 4.14 8.77 4.09 9.3 5.17 8.87 8.73 9.39 8.29 5.39 5.89 6.1 11.47 5.39 5.08 11.04 11.05
R 2.71 4.5 5.12 8.77 10.03 7 2.96 6.39 8.47 10.56 9.19 7.94 7.49 11.26 9.42 8.7 7.94 8.9 9.77 15.01 13.28
inher.-single
synch.-conc.
inher.-multi.
file-I/O
json
factorial
ackermann
selection
num.-integ.
regex
huffman
url-encode
exp.-operat.
lzw-compr.
array
func.-comp.
palindrome
insertion
bubble
merge
url-decode
quick
conc.-comp.
obj-meth.
classes
0
3
6
9

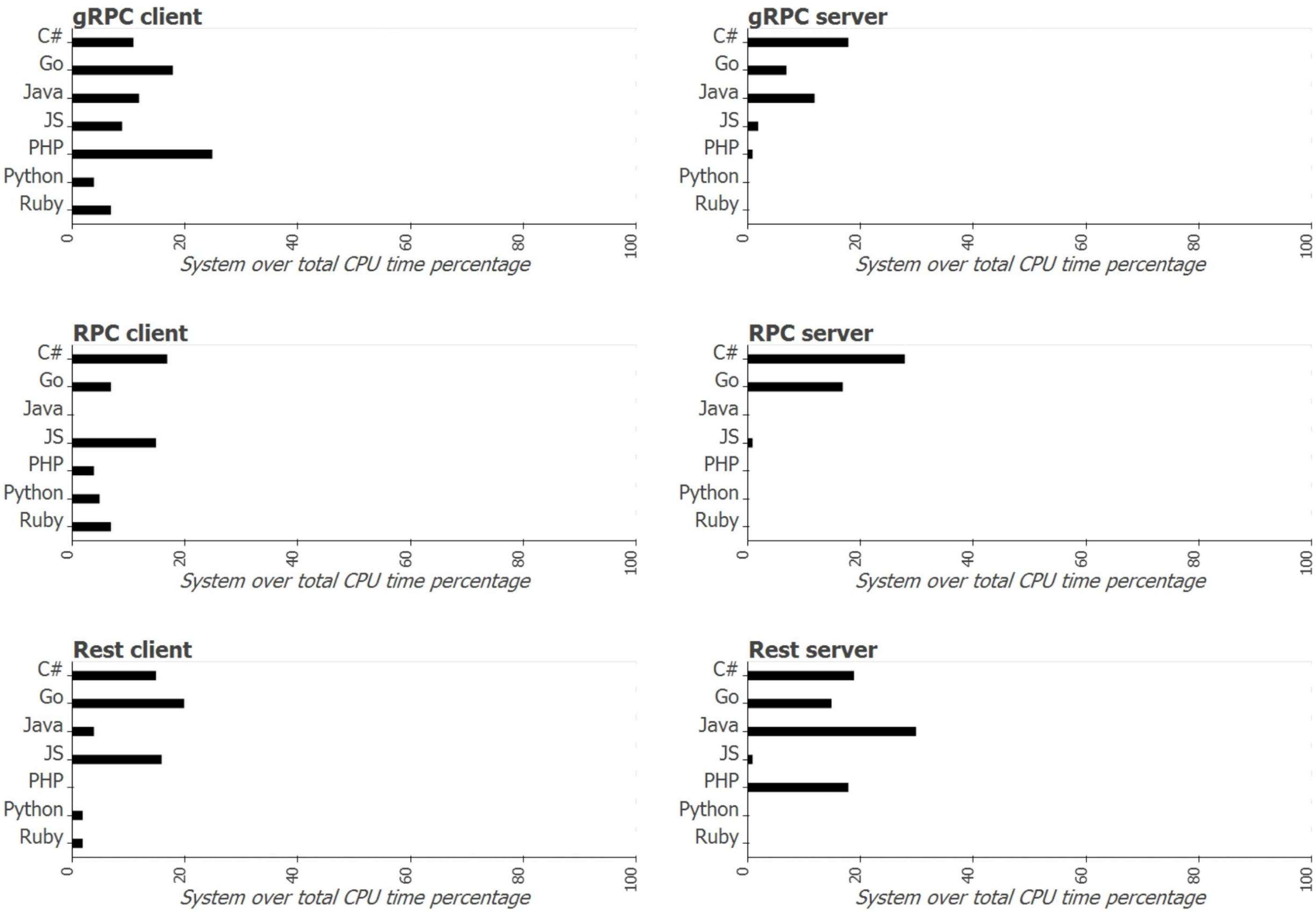


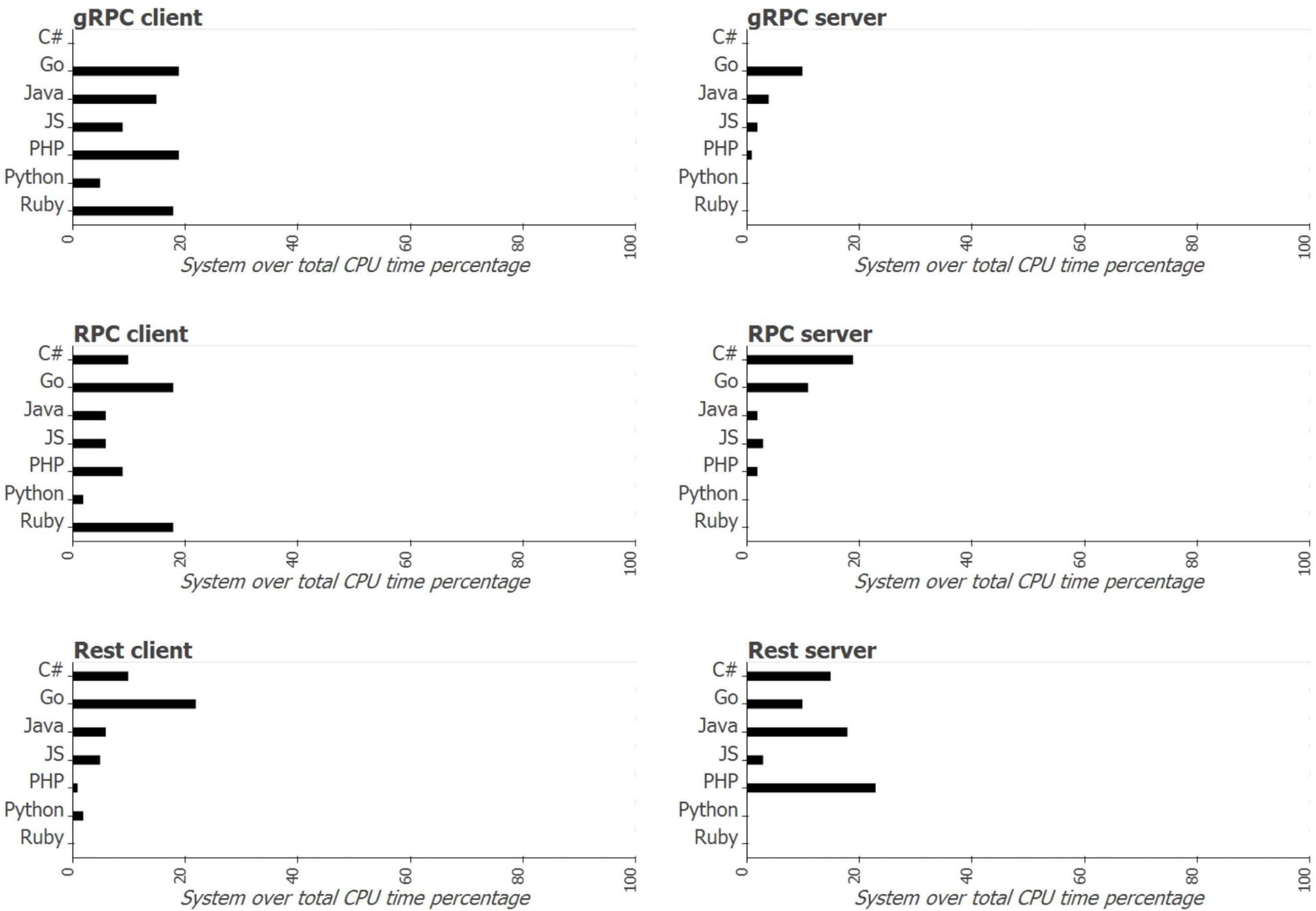
0
2
4
6
8
10
func.-comp.
lzw-compr.
url-decode
obj.-meth.
merge
exp.-operat.
conc.-comp.
quick
num.-integ.
bubble
classes
selection
palindrome
array
insertion
url-encode
factorial
huffman
regex
json
ackermann
file-I/O
inher.-multi.
inher.-single
synch.-conc.



0
2
4
6
8
10
classes
exp.-operat.
func.-comp.
merge
conc.-comp.
quick
obj.-meth.
array
insertion
bubble
url-encode
selection
palindrome
huffman
num.-integ.
lzw-compr.
url-decode
regex
ackermann
json
file-I/O
factorial
synch.-conc.
inher.-multi.
inher.-single


0
2
4
6
8
10
classes
quick
url-decode
merge
func.-comp.
insertion
conc.-comp.
bubble
array
lzw-compr.
selection
obj.-meth.
huffman
palindrome
url-encode
regex
exp.-operat.
num.-integ.
factorial
json
ackermann
inher.-multi.
file-I/O
synch.-conc.
inher.-single


gRPC client
gRPC server
RPC client
RPC server
Rest client
Rest server
C#
Go
Java
JS
PHP
Python
Ruby
0
20
40
60
80
100
System over total CPU time percentage


gRPC client
gRPC server
RPC client
RPC server
Rest client
Rest server
C#
Go
Java
JS
PHP
Python
Ruby
0
20
40
60
80
100
System over total CPU time percentage