JKAU: Comp. IT. Sci., **Vol. 9** No. 1, pp: 29 – 44 (1441 A.H. / 2020 A.D.)

Doi: 10.4197/Comp. 9-1.3

Programming Languages and Energy Consumption: A Survey

Eythar Alghamdi [1,2] and **Ahad Alloqmani** [1]

[1]Computer Science Department, Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah, and [2] Computer Science Department, Faculty of Computing and Information Technology, University of Bisha, Bisha, Saudi Arabia

ealghamdi0219@stu.kau.edu.sa

Abstract. Programming languages have a powerful role to develop and implement highly effective programs and systems. Energy consumption is becoming a key criterion when choosing programming languages instead of fast execution. Five papers and five popular programming languages (Haskell, Java, C#, JavaScript, and PHP) were reviewed, to answer whether the fast execution programs are also energy-efficient programs, or not, and if optimizing a program for one of them that affects another one, negatively and positively with considerate the difference between programming languages. The programming languages were classified into three categories to compare between each language in the same category and compare languages in different categories. From our study, the result was there is no winner, as no language exceeds the rest in all study cases. It is clear that different programming languages classes, and even languages within the same class have a completely different impact on energy consumption based on the used data type, the size of the data, the used approach, and other reasons. Also, most energy-efficient languages are not always the fastest.

Keywords: Energy Consumption, Haskell, Java, C#, JavaScript, PHP.

1. Introduction

Programming languages have powerful mechanisms and methods to develop and implement highly efficient programs and systems. Formerly, such mechanisms and methods aim to produce fast programs by reducing run time; therefore, the less time they take to run, the better they perform. Nowadays, a lot of things have changed in software programming and engineering. Instead of fast execution, software energy consumption is becoming a key concern for everyone related to the computer like computer manufacturers and even regular computer users. That is especially with the demands and the growing need to preserve the environment and the emergence of the concept of green computing that aims to use computers and their resources in the way making them eco-friendly as a part of environmentally responsible [1].

The main question that frequently arises in the energy consumption field is whether the fast execution programs are also energy-efficient programs, or not, and if optimizing a program for one of them that affects another one, negatively and positively.

Regarding some research, energy consumption depends on execution time and many other factories, such as API calls, code obfuscation, constructs for concurrent execution, object-oriented code refactorings, and data types [1][2].

Nevertheless, it's a complicated task. The reasons for that are programs written in different languages that implement the same computing problem may use different algorithms, and the performance of a language is influenced by many factors like a virtual machine, the quality of its compiler, available libraries, garbage collectors, etc. Indeed, as theoretical and practical studies proved, a software program may become faster by improving its source code, but also by just optimizing its libraries and/or its compiler [3].

In this paper, five papers and five popular programming languages (Haskell, Java, C#, JavaScript, and PHP) is reviewing, in order to answer the previous questions, considering the difference between programming languages by classifying them into three categories to compare between each language in the same category and compare between languages in different categories.

The rest of the paper is organized as follows: Section 2 describes the classification used in this paper. Section 3 presents the analysis and discussion; Section 4 shows the results. Finally, Section 5 concludes our work.

2. Programming Languages Classification

Programming languages can be divided into different classes depending on different features. In this paper, the programming languages were split into three classes: functional languages, object-oriented languages and scripting languages.

This classification is beneficial to understand the final results of the experiments since each language class has its own features and methods to write a program.

2.1 Functional Languages

Functional programming was described as the programming style that the primary method of computation is the application of functions to arguments [3-4]. Haskell is a purely functional programming language, named after the American mathematician and logician Haskell Curry and it was initiated by an international committee of programming language researchers [4]. In 1987, Philip Wadler and others developed the concept of type classes to support overloading and handle effects, which are the leading unconventional features of Haskell in the 1990s [4]. In 2010, a revised and updated version of the Haskell Report was published. Since then, the language has continued to evolve [4].

2.2 Object-Oriented Programming Language (OOP)

OOP was defined as the language that provides support for three main language features: inheritance, abstract data types, and dynamic binding of method calls to methods [5].

Java and C# were designed to support object-oriented programming and do not support other programming paradigms; they still employ some of the basic imperative structures and have the appearance of the older imperative languages [5]. The OOP approach can demean performance and raise the power consumption of software (as compared to the classical procedure programming) because of its additional abstraction and encapsulation layers and mechanisms [6].

2.3 Scripting Language

Scripting languages were used to insert a list of commands into a file for interpretation, and they started as a small series of commands interpreted as calls to machine subprograms executing utility functions, such as file management and quick retrieval of files. Then, variables were added, declarations of control flow, features, and numerous other functionalities, resulting in a full programming language [5].

JavaScript has undergone significant development through the introduction of many new features and capabilities. In the late 1990s, a language specification was developed for JavaScript. However, a JavaScript interpreter may be used in many different applications; it is embedded in Web browsers for its most common use [5]. JavaScript code is embedded in the HTML documents and interpreted when the browser displays the documents. JavaScript's primary uses in Web programming are validating input form data and creating dynamic HTML documents [5].

PHP is an HTML-embedded server-side scripting language specifically designed for web applications. PHP code is translated on the Web Server when a browser has submitted an HTML document in which it is inserted [5]. PHP code generates an output HTML code, which replaces the PHP code in the HTML document. A Web browser thus never sees the PHP code [5].

3. Analysis and Discussion

Five papers that are relevant directly and indirectly to the subject of the study were analyzed. In this section, these papers were discussed in terms of the methodologies used, the languages were studied, and results.

3.1 Functional Languages

For functional languages, two papers about energy consumption in the Haskell language were studied and compared their results [2][3].

Lima, L. G. et al., 2016 [2], attempted to highlight the energy attitude in some programs that are written in a purely functional language, like Haskell [2]. They had developed two existing performance analysis tools to become aware of energy behavior, the Criterion benchmarking library and the profiler that comes with the Glasgow Haskell Compiler [2]. They made two kinds of comparison: compare energy consumption in different data structures and compare concurrent constructs [2].

In the first comparison, using Edison, a library provides multiple implementations for several families. They analysed the efficiency and energy behaviour of several test operations across 15 separate implementations of the three different types of data structures as shown in Table 1 [2].

Table 1. The functions that Edison provides and used by researchers to implement the operations [2].

	Sequences	**Sets**	**Heaps**	**Associative Collections**
add	lcons, rcons	insert	insert	insert
addAll	append	union	union	union
clear	null, ltail	difference	minView, delete	difference
contains	null, filter	member	member	member
containsAll	foldr, map	subset	null, membe, minView	submap
iterator	map	foldr	fold	map
remove	null, ltail	deleteMin	deleteMin	null, deleteMin
removeAll	filter	difference	minView, delete	difference
retainAll	filter	intersection	filter, member	Intersection-With
toArray	toList	foldr	fold	foldrWithKey

They analyzed the results and found that following [2]:

- *For Sequences:* The time of execution has a strong influence on energy consumption. The measured proportions for all processes and applications differ at most 1.9 percent.
- *For Associative Collections:* Energy consumption was equal to the time of operation. The AssocList was less effective for almost all. StandardMap costs between 40 percent to 85 percent more time and energy than AssocList. The amount of energy consumed was 1% higher than the percentage of time spent on the add operation only.

- *For Collections:*

o Sets: Further time execution often requires further energy consumption for each combination of implementation and benchmarking. The UnbalancedSet is less effective than the StandardSet for all test operations but Contains. Comparing the time and energy usage percentages have shown that -for all benchmark operations- the difference between the proportion of time and energy consumption is always less than 1.49 percent.

o Heaps: Energy consumption is equal to the time of operation. Overall, the LazyPairingHeap was found to be the most effective in all test operations except for Add. The SkewHeap and SplayHeap were the least efficient in 5 operations for each. The proportions of execution time and energy consumption for any operation in any Heaps operation are at most 2.16 percent different.

In the second comparison, researchers aimed at determining the energy efficiency of Haskell's concurrent programming constructs by analyzing the concurrent programming primitives functions such as forkIO, forkOn, and forkOS [2].

They tested three separate thread control structures and three data sharing primitives utilizing nine benchmarks and multiple experimental configurations and found that [2]:

- *Small changes lead to significant savings:* One of the key findings of this study is that easy refactoring, such as flipping between thread control systems, can have a significant impact on energy consumption. For example, using forkOn instead of forkOS with TVar can save between 25% and 57% of energy.

- *Faster is not always greener:* For 6 out of 9 benchmarks, in at least two variants of each, there are times when faster execution time leads to more energy consumption.

- *There is no overall winner:* Generally, no thread management construct or data sharing, or a mixture of both always gives the best result.

Therefore, we can conclude that the connection between energy consumption and efficiency is not always evident. In general, high performance leads to low energy consumption, particularly in sequential benchmarks. Even so, when considering concurrency, we didn't find a definite relationship.

Couto, M, et al., 2017 [3], presented the research of the runtime, memory, and energy consumption of twenty-seven pretty-known software languages; one of them was Haskell [3]. The primary motivation and main focus of this study are to understand energy efficiency across the programming languages [3].

They used the Computer Language Benchmarks Game (CLBG) as a framework for execution, testing and comparison implemented solutions to a set of well-known programming problems such as Hashtable update and k-nucleotide strings, Allocate, traverse and deallocate many binary trees. All programming problems that were implemented were shown in Table 2. They then collected the fastest version of the code for each of the benchmark issues [3].

The CLBG also provides calculated knowledge on both runtime and energy consumption, but for precise energy projections, they used the RAPL tool for measuring the energy consumption [3]. Each benchmark solution has been implemented and measured ten times, in order to obtain ten energy consumption and run time samples, to minimize the effect of cold start, caching effects, and avoid outliers [3].

As a result, they identified different energy consumption behaviors and execution times in several languages and tests. In the binary-tree benchmark, Haskell consumed 270.15J to execute the solutions. In the fannkuch-redux benchmark, it consumed 433.68J to execute the solutions. In the fasta benchmark, it consumed 205.52J to execute the solutions. But when comparing the energy and time, we found that Haskell is one of four languages that preserve the same energy consumption and execution time [3].

Table 2. CLBG framework [3].

Benchmark	Description	Input
n-body	Double precision N-body simulation	50M
fannkuch-redux	Indexed access to a tiny integer sequence	12
spectral-norm	Eigenvalue using the power method	5,500
mandelbrot	Generate Mandelbrot set portable bitmap file	16,000
pidigits	Streaming arbitrary precision arithmetic	10,000
regex-redux	Match DNA 8mers and substitute magic patterns	fasta output
fasta	Generate and write random DNA sequences	25M
k-nucleotide	Hashtable update and k-nucleotide strings	fasta output
reverse-complement	Read DNA sequences, write their reverse-complement	fasta output
binary-trees	Allocate, traverse and deallocate many binary trees	21
chameneos-redux	Symmetrical thread rendezvous requests	6M
meteor-contest	Search for solutions to shape packing puzzle	2,098
thread-ring	Switch from thread to thread passing one token	50M

Comparing with other languages, we grouped them by the paradigms; we found that the imperative languages averagely took less energy consumption, then object-oriented languages, then Haskell and the rest of the functional languages, and nothing worse than functional languages except scripting languages.

Depending on that, we can say that although Haskell maintains the same energy consumption and time rank doesn't mean that energy is affected by the time execution. But there are many scenarios where a software engineer has to choose Haskell (or not) to develop an algorithm depending on functional or non-functional requirements. Still, in general, Haskell is not the best functional language if we consider (energy and time execution), or (energy and memory), or (energy, time, and memory), but it is still one of the best. Table 3 presents the three multi-objective rankings. For each category, each line represents a set that includes the languages identical to each other for the underlying goals [3].

Table 3. The optimal solutions for various group of objectives [3].

Energy & Time	Energy & Memory	Energy & Time & Memory
C	C • Pascal	C • Pascal • Go
Rust	Rust • C++ • Fortran • Go	Rust • C++ • Fortran
C++	Ada	Ada
Ada	Java • Chapel • Lisp	Java • Chapel • Lisp • OCaml
Java	OCaml • Swift • Haskell	Swift • Haskell • C#
Pascal • Chapel	C# • PHP	Dart • F# • Racket • Hack • PHP
Lisp • OCaml • Go	Dart • F# • Racket • Hack • Python	JavaScript • Ruby • Python
Fortran • Haskell • C#	JavaScript • Ruby	TypeScript • Erlang
Swift	TypeScript	Lua • JRuby • Perl
Dart • F#	Erlang • Lua • Perl	
JavaScript	JRuby	-
Racket	-	-
TypeScript • Hack	-	-
PHP	-	-

3.2 *Object-Oriented Programming Languages*

For Object-oriented Programming languages, three papers about energy consumption in Java and C# languages were studied and compared their results [3][7][8].

Hasan, S. et al., 2016 [7], profiles were provided for the Java Collections Framework, Apache Commons Collections, and Trove [7]. They collected energy usage data using the GreenMiner framework to measure actual energy consumed in Joule (J). They created energy consumption profiles for commonly used API methods for variables from three types of collections data types: List, Map and Set implementations, and recorded how this varies with input sizes [7]. The collections classes that studied are shown in Table 4.

Table 4. Profiled Collections Classes [7].

Library	List	Map	Set
Java Collections Framework (JCF)	ArrayList LinkedList	HashMap TreeMap	HashSet TreeSet LinkedHashSet
Apache Collections Framework (ACC)	TreeList	HashedMap LinkedMap	ListOrderedSet MapBackedSet
Trove	TIntArrayList TIntLinkedList	TIntIntHashMap	TIntHashSet

They have discussed these six research questions [7]:

1. What is the most energy-efficient List implementation for insertions, iteration, and random access?

2. What is the most energy-efficient Map implementation for insertions, iteration, and random query?

3. What is the most energy-efficient Set implementation for insertions, iteration, and random query?

4. How the input size affects the energy consumption of the collections?

5. How does storing different elements affect the energy consumption of the collections?

6. How can we use the profiles to choose the most energy efficient implementation of List, Map, and Set?

1, 2 and 3 compared to the energy profiles that have been created for the implementation of List, Map and Set; 4 and 5 relate to the measurement of the impact of sizes and types of input data; 6 on the use of the results as a guide for developers [7].

The answers to research questions as follows [7]:

As for the first question: JCF's LinkedList consumes the least energy in insertions at the beginning, followed by Trove's LinkedList.

Trove's ArrayList is the most energy-efficient for insertions in the middle and at the end, followed by JCF's ArrayList. Energy does not differ when it expands. for iteration and random access, there is no much difference between them.

As for the second question: For insertions and random queries, HashMap is the most energy-efficient. ACC's LinkedMap is a little better on insertions than JCF's LinkedHashMap in case the order of insertion must be kept. TreeMap is bad energy consumption and must be averted unless explicitly needed. For iteration is the same for almost all implementations.

As for the third question: For insertions and random queries, HashSet is the most energy-efficient. For iterations, ACC's ListOrderedSet is the most energy-efficient Set. TreeSet is bad energy consumption and must be averted unless explicitly needed.

As for the fourth question: For input sizes from 1 to 500, all alternative implementations of List, Map, and Set perform equally. It does not display a significant difference in energy consumption. But, when we deal with more elements, the differences become large and significant.

As for the fifth question: Inserting small object types in a list is the most energy-efficient than operations on primitive data types in lists that consume more energy.

As for the sixth question: Generally, for list implementation, TIntArrayList is the most energy-efficient followed by ArrayList. For map implementation, HashMap is the best. For Set implementation, TIntHashSet is the most energy-efficient with HashSet as a close second.

Figure 1 shows the difference in energy efficiency between the implementation of List, Map, and Set. Where each color refers to the rank: Green determines the most efficient implementation, while the red color indicates the worst among alternatives. On each table, the row with the greenest is the best [7].

In the study of Couto, M. et al., 2017 [3], C# and Java were included. In the binary-tree benchmark, Java consumed 111.84J to execute the solutions, and C# consumed 189.74J. In the fannkuch-redux benchmark, Java consumed 311.38J, and C# consumed 399.33J. Java was achieving the fifth-best value by consumed 35.86J, and C# consumed 45.35J [3].

Considering the different combinations of objectives, we found, as shown in Table 3, that Java is the second-best object-oriented language in all cases, then C#.

Chandra, T. B. et al., 2018 [8], the authors studied different languages including C# and Java. They implemented different sorting algorithms which are bubble sort, insertion sort, selection sort, and quick sort on these languages to compare between them for finding the language that is the most energy-efficient [8].

They used "Joulemeter" as a simulator tool to simulate the energy consumption of different sorting algorithms implemented in these languages [8]. The comparing was on both integer and double data sets with sixty thousand elements [8].

Figures 2 and 3 demonstrate the comparison results. The study calculated the values based on power consumption in watt per second [8].

Based on the results, the study found that Java is the most energy-efficient programming language, while C# consumes more power than java. Also, sorting the data elements of type double consumes more power than of the data elements of type integer. As well the energy consumption depends on the selection of sorting algorithms [8].

3.3 Scripting Languages

For scripting languages, two papers about energy consumption in PHP and JavaScript were studied and compared their results [3][9].

Kurtz, K. et al., 2017 [9], explored the effect of applying diverse web-based approaches within the execution time and energy consumption in bubble sort application [9]. They observed time and energy in PHP, JavaScript, and a Node.js implementation, then comparing the result against a java implementation. In all experiments, the array is initialized in the worst case and reiterates them thirty times for the results of statistical validity [9].

They compared three case studies: the efficiency of PHP against Java, the efficiency of JavaScript, and Node.js against Java, and a comprehensive comparison of the four approaches that are analyzed [9].

In the first case study, when an array size between 100 and 1,000, Java achieves better results. While when an array size between 10,000 and 100,000, PHP achieves a reduction in energy consumption around 15%

to 83%. Table 5 summarizes the energy consumption results for native implementation and PHP implementation [9]. Figure 4 shows the relation between array size and energy consumption in the evaluated implementations.

Table 5. Comparison between native and PHP in terms of energy [9].

Size	Implem.	Energy Consumption (mJ)	Std
100	Native	8.17	2.78
	PHP	146.8	62.30
1,000	Native	43.37	18.44
	PHP	206.93	60.70
10,000	Native	2,026.67	496.84
	PHP	1,730	346.56
100,000	Native	68,556.67	1,593.02
	PHP	11,613.33	961.58

In the second case study, Java presents the lowest consumption when an array size is between 100 and 1,000. While at an array size between 10,000 and 100,000, Node.js achieves improvements from 24% to 93% against JavaScript [9]. Table 6 summarizes the energy consumption results. Figure 5 shows the relation between array size and energy consumption in the evaluated implementations.

Table 6. Comparison between native, JavaScript, and Node.js [9].

Size	Implem.	Energy Consumption (mJ)	Std
100	Native	8.17	2.78
	JavaScript	751.47	167.83
	Node.js	724.3	143.33
1,000	Native	43.37	18.44
	JavaScript	789.97	215.32
	Node.js	770.73	189.18
10,000	Native	2,026.67	496.84
	JavaScript	1,523.17	345.44
	Node.js	1,155.57	297.68
100,000	Native	68,556.67	1,593.02
	JavaScript	18,810	119.58
	Node.js	1,361.87	340.21

In the overall comparison, when an array size between 100 and 1,000, Java achieves the best result, and Javascript and Node.js achieve the worst outcome. While when an array size is between 10,000 and 100,000, Node.js produces the best result, and Java achieves the worst result.

Therefore, we found that the efficiency of approaches is more affected by the array size. On the other side, the Node.js shows less affected by the array size; then, it is the best choice for the large array size regarding energy consumption and execution time. Figure 6 shows the relation between array size and energy consumption [9].

In Couto, M. et al., 2017 [3] study, PHP, and JavaScript were included. In the binary-tree benchmark, JavaScript consumed 312.14J to execute the solutions, PHP consumed 1,397.51J. In the fannkuch-redux benchmark, JavaScript consumed 413.90J, PHP consumed 5.731.88J. In the fasta benchmark, JavaScript consumed 64.84J, PHP consumed 430.73J. But in the regex-redux benchmark, which manipulates strings using regular expressions, PHP and JavaScript seem to be an energy-efficient choice. However, they tend to be not very energy efficient in other scenarios. Thus, clear to us that a faster language is not always the most energy-efficient [3].

Considering the different combinations of objectives, we found as shown in Table 3 that PHP is the best scripting language when we look for less energy consumption and less memory loading. Still, JavaScript is the best scripting language when we look for less energy consumption and less time execution, while they are almost the same when we look for less energy consumption, less time execution, and less memory loading [3].

4. Results

Comparison between four studies was conducted in terms of data type, elements numbers, the problem that was solved, experiment repetition, programming languages, and the best and worst cases in each language. Table 7 shows a summary of that comparison. It's clear that every language has best-case and worst-case, which are depending on data type, element number, or problem that was solved.

Then, the result cannot be generalized, as it is limited to the cases studied, and changing any criterion will, undoubtedly, change the result.

Table 8 shows the programming language order depending on the study of Couto, M, et al., 2017 [3], based on different objectives like if a programmer considers energy & time, energy & memory, or energy & time & memory. Although, the study of Couto, M, et al., 2017 [3] ordered the programming languages based on different objectives [3]. However, we cannot generalize the result in every case; it's only for operations that are provided by the CLBG framework and for the fastest version of the code for each of the benchmark issues.

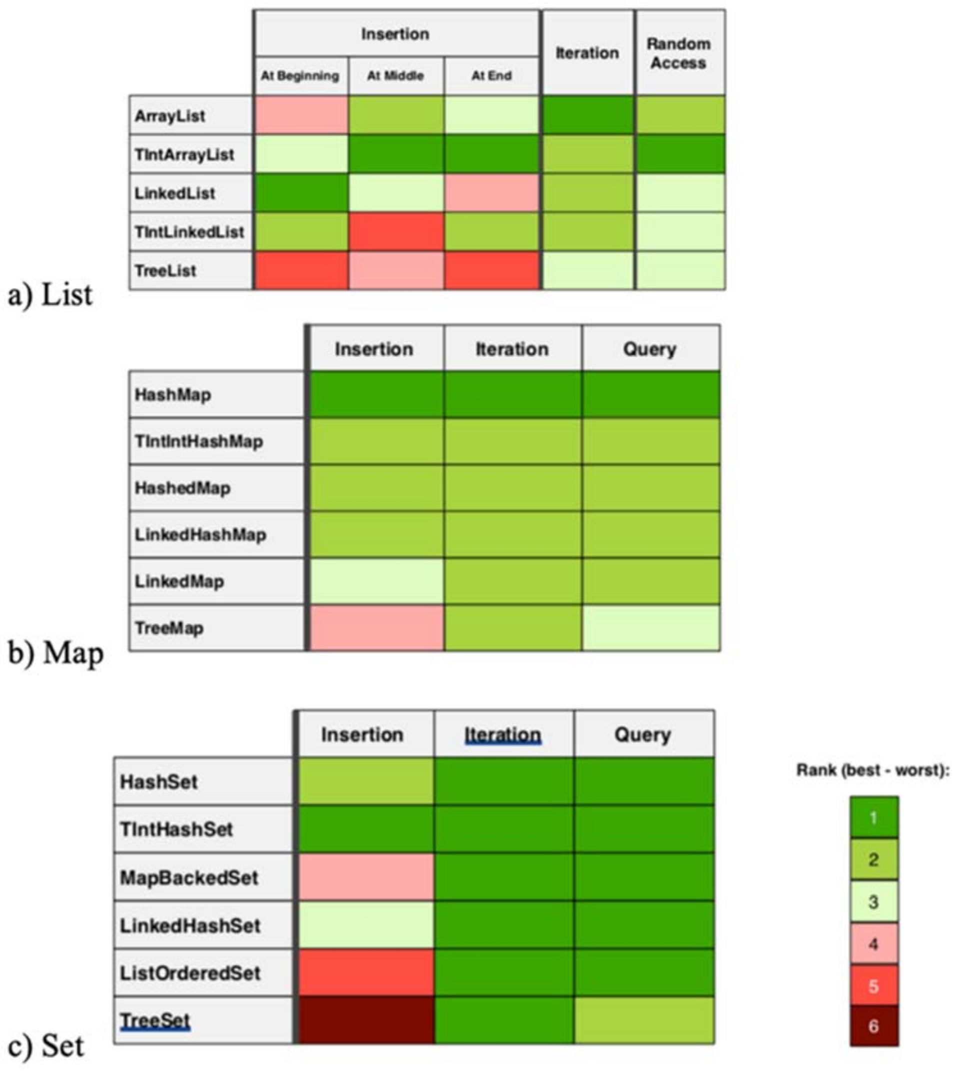


Fig. 1. Energy-efficient difference between List, Map and, Set implementation [7].

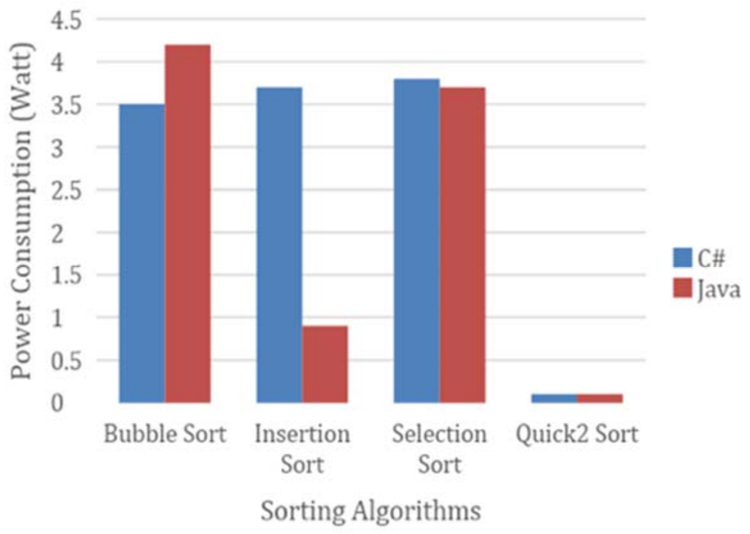


Fig. 2. Average power consumed for integer data set (W/s) [8].

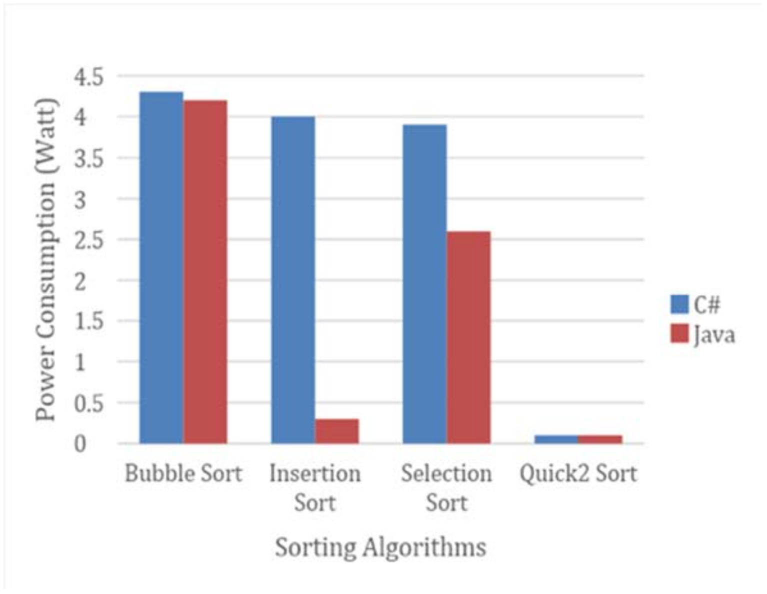
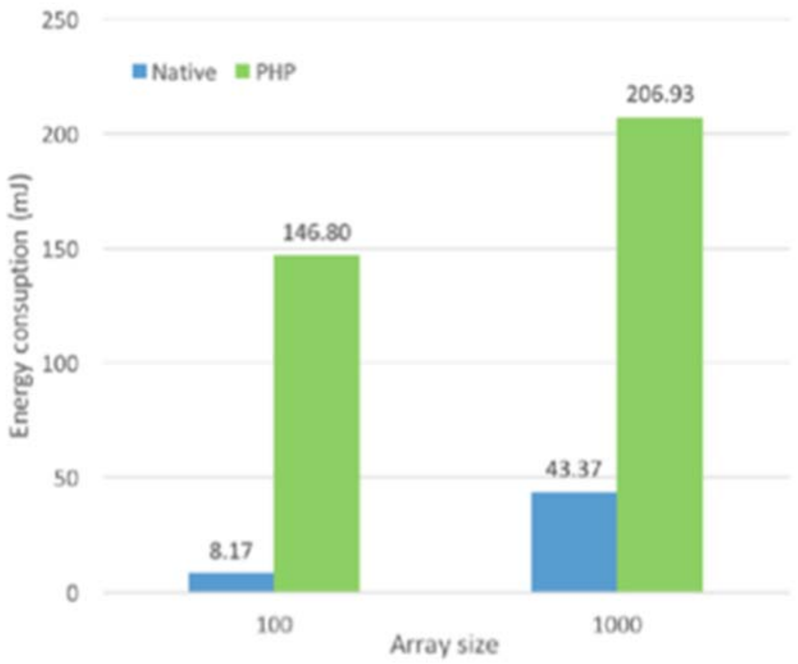
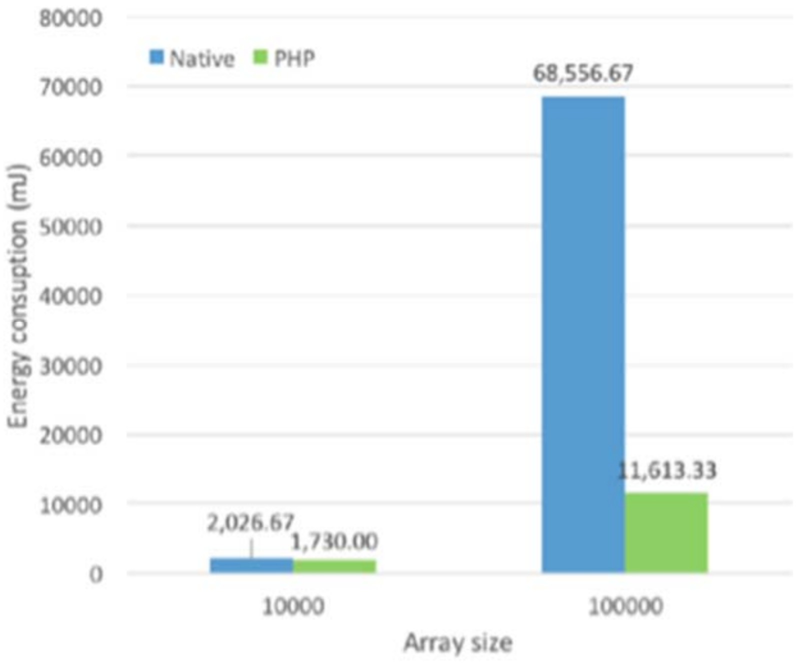


Fig. 3. Average power consumed for double data set (W/s) [8].

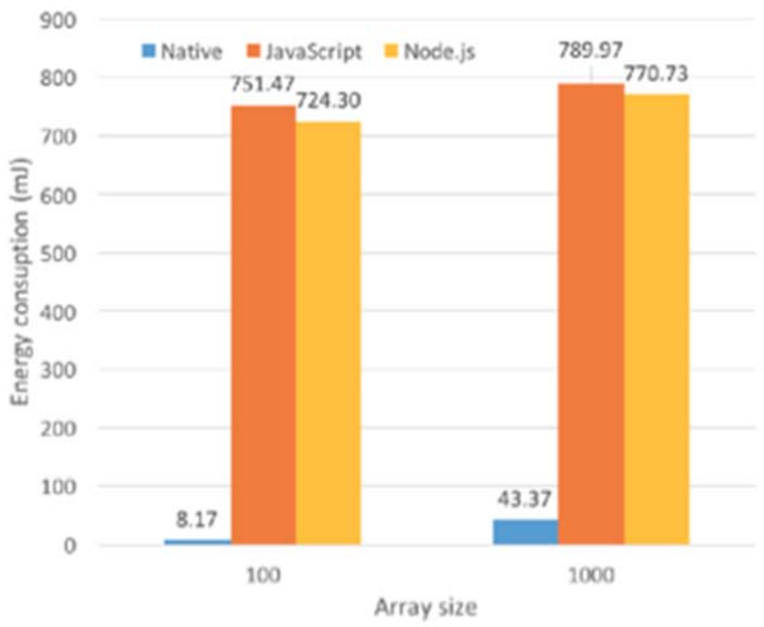


a) Arrays of 100 and 1,000 elements.

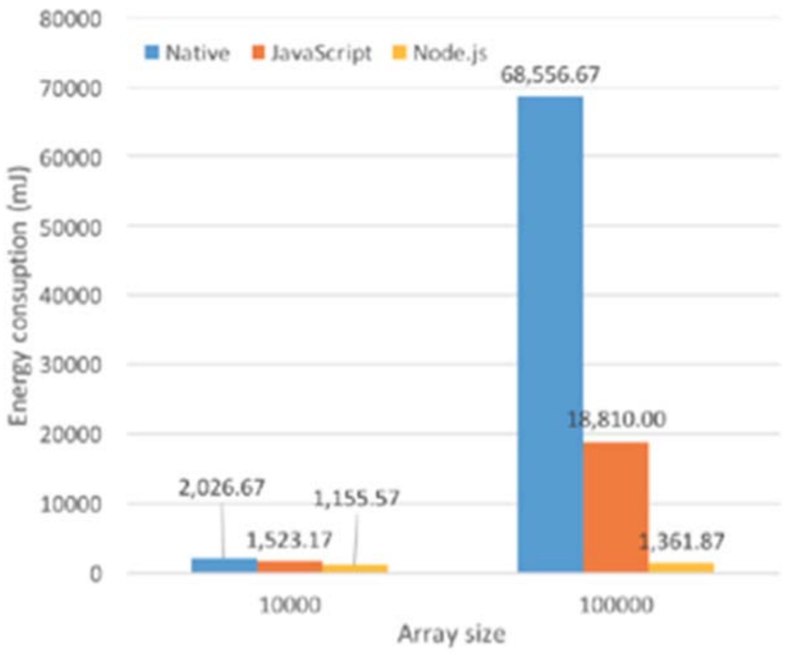


b) Arrays of 10,000 and 100,000 elements.

Fig. 4. Native vs. PHP: Energy Consumption Results [9].



a) Arrays of 100 and 1,000 elements.



b) Arrays of 10,000 and 100,000 elements.

Fig. 5. Native, JavaScript, and Node.js energy consumption results [9].

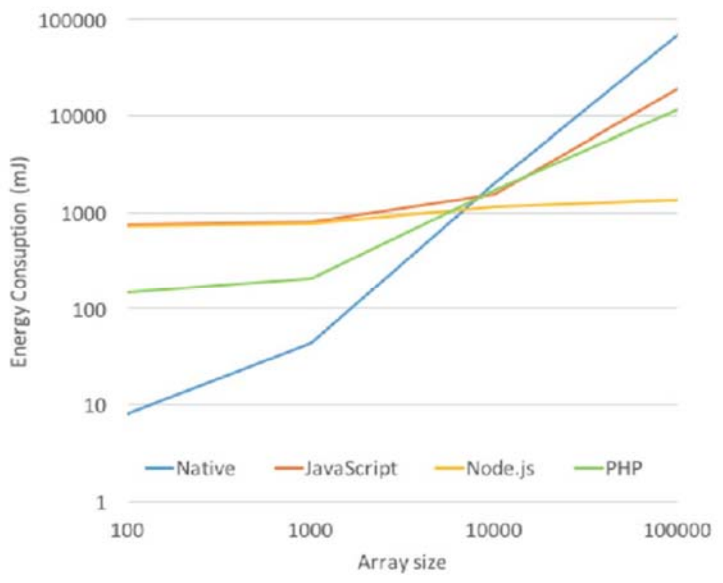


Fig. 6. The relation between array size and energy consumption [9].

Table 7. Summary comparison between the four studies.

Study	Data Type	Elements No.	Problem	Experiment Repetition	PL	Best Case	Worst Case
[2]	Sequence, Collection, Associative Collection.	-	Edison	-	Haskell	high-performance process.	low-performance process.
[7]	List, Map, Set.	from 1 to 5000	- Java Collections Framework (JCF). - Apache Collections Framework (ACC). - Trove.	20 times	Java	- List: TIntArrayList - Map: HashMap. - Set: TIntHashSet.	- List: TreeList. - Map: TreeMap. - Set: TreeSet.
[8]	Integer and double array.	60,000	Sorting	4 times	Java	- Integer data set: Quick Sort. - Double data set: Quick Sort.	- Integer data set: Bubble Sort. - Double data set: Bubble Sort.
					C#	- Integer data set: Quick Sort. - Double data set: Quick Sort.	- Integer data set: Selection Sort. - Double data set: Bubble Sort.
[9]	Array	from 100 to 100,000	Bubble Sort	30 times	Java	100 - 1,000 array size	10,000 - 100,000 array size
					JS	10,000 - 100,000 array size	100 - 1,000 array size
					PHP	10,000 - 100,000 array size	100 - 1,000 array size

Table 8. The programming languages order based on the different objectives depending on [3] study.

Objectives \ Order	1st	2nd	3rd	4th
Energy & Time	Java	Haskell C#	JS	PHP
Energy & Memory	Java	Haskell	C# PHP	JS
Energy & Time & Memory	Java	Haskell C#	PHP	JS

Obviously, there is no winner, as no language exceeds the rest in all study cases. It is clear that different programming languages classes and even languages within the same class have a completely different impact on energy consumption based on the used data type, the size of the data, the used approach, and other reasons. Also, we observed that the most energy-efficient languages are not always the fastest.

We believe these results are useful where programmers have to choose a specific programming language to implement their software. For example, if the programmer wants to develop software for wearable devices, it is crucial to select a language with low energy consumption to help save battery. But if he is going to build software for user-interactive, it is important to choose a language that has less time execution.

4. Conclusion

This paper reviewed five papers and five popular programming languages (Haskell, Java, C#, JavaScript, and PHP) in order to answer whether the fast execution programs are also energy-efficient programs, or not, and if optimizing a program for one of them that affects another one, negatively and positively, with considerate the difference between programming languages by classifying them into three categories to compare between each language in the same category and compare between languages in different categories.

The result was that different programming languages classes and even languages within the same class have a completely different impact on energy consumption based on the used data type, the size of the data, the used approach, and other reasons. Also, most energy-efficient languages are not always the fastest, and absolutely, these results are useful where programmers have to choose a programming language to implement their software according to functional and non-functional requirements.

Acknowledgments

The authors would like to express our special thanks to Dr. Sanaa Sharaf (King Abdulaziz University), who gave us the golden opportunity to work in this research area.

References

[1] **Pereira, R., Couto, M., Ribeiro, F., Rua, R., Cunha, J., Fernandes, J. P., & Saraiva, J.** (2017, October). Energy efficiency across programming languages: How do energy, time, and memory related?. In Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering (pp. 256-267). ACM.

[2] **Lima, L. G., Soares-Neto, F., Lieuthier, P., Castor, F., Melfe, G., & Fernandes, J. P.** (2016, March). Haskell in green land: Analyzing the energy behavior of a purely functional language. In 2016 IEEE 23rd international conference on Software Analysis, Evolution, and Reengineering (SANER) (Vol. 1, pp. 517-528). IEEE.

[3] **Couto, M., Pereira, R., Ribeiro, F., Rua, R., & Saraiva, J.** (2017, September). Towards a Green Ranking for Programming Languages. In Proceedings of the 21st Brazilian Symposium on Programming Languages (p. 7). ACM.

[4] **Hutton, G.** (2016). Programming in Haskell. Cambridge University Press.

[5] **Sebesta, R. W.** (2016). Concepts of programming languages. Pearson.

[6] **Maleki, S., Fu, C., Banotra, A., & Zong, Z.** (2017). Understanding the impact of object-oriented programming and design patterns on energy efficiency. 2017 Eighth International Green and Sustainable Computing Conference (IGSC).

[7] H**asan, S., King, Z., Hafiz, M., Sayagh, M., Adams, B.,** & **Hindle, A.** (2016). Energy profiles of Java collections classes. Proceedings of the 38th International Conference on Software Engineering - ICSE 16.

[8] **Chandra, T. B., Verma, P.,** & **Dwivedi, A. K.** (2018). Impact of Programming Languages on Energy Consumption for Sorting Algorithms. Advances in Intelligent Systems and Computing Software Engineering, 93–101. doi: 10.1007/978-981-10-8848-3_9

[9] **Kurtz, K., Noguez, M., Zanini, F., Ferreira, P. R.,** & **Brisolara, L.** (2017, November). Comparing Performance and Energy Consumption of Android Applications: Native Versus Web Approaches. In 2017 VII Brazilian Symposium on Computing Systems Engineering (SBESC) (pp. 147-154). IEEE.

لغات البرمجة واستهلاك الطاقة: دراسة استقصائية

إيثار الغامدي[1،2] و عهد اللقماني[1]

[1] قسم علوم الحاسب، كلية الحاسبات وتقنية المعلومات، جامعة الملك عبد العزيز، جدة، و [2] قسم علوم الحاسب، كلية الحاسبات وتقنية المعلومات، جامعة بيشة، بيشة، المملكة العربية السعودية

emalghamdi@ub.edu.sa

المستخلص. للغات البرمجة دور هام في تطوير وتنفيذ برامج وأنظمة فعالة. أصبح استهلاك الطاقة معيارًا رئيسيًا عند اختيار لغات البرمجة بدلاً من سرعة التنفيذ. في هذه الدراسة تمت مراجعة خمس أوراق بحثية حول خمس لغات برمجة شائعة الاستخدام، هي: هاسكل وجافا وسي شارب وجافا سكريبت وبي اتش بي؛ وذلك للإجابة عما إذا كانت برامج التنفيذ السريع هي أيضًا برامج موفرة للطاقة أم لا، وإذا كان تحسين أحدها يؤثر على الآخر بشكل سلبي أو إيجابي، مع مراعاة الفروق بين لغات البرمجة. تم تصنيف لغات البرمجة إلى ثلاث فئات للمقارنة بين اللغات في نفس الفئة والمقارنة بين اللغات من فئات مختلفة. نتيجة دراستنا أثبتت أن لا تفضيل لأية لغة على الأخرى من ناحية توفير استهلاك الطاقة، حيث لا توجد لغة تفوق البقية في جميع حالات الدراسة وأيضًا معظم اللغات الموفرة للطاقة ليست دائما الأسرع. بالتالي من الواضح أن فئات لغات البرمجة المختلفة، وحتى اللغات داخل نفس الفئة لها تأثير مختلف تمامًا على استهلاك الطاقة استنادًا إلى نوع البيانات وحجمها والطريقة المستخدمة في البرمجة وغيرها من الأسباب.

الكلمات المفتاحية: استهلاك الطاقة، هاسكل، جافا، سي شارب، جافا سكريبت، بي اتش بي.