Analysing the Energy Usage of the Erlang BEAM

Gharbi Youssef, Melinda Tóth and István Bozó

ELTE, Eötvös Loránd University, Faculty of Informatics, Budapest, Hungary

Abstract

The evolution of programming languages and their runtime systems has been a continuous journey toward efficiency and performance. One such significant milestone in this journey is the introduction of Just-In-Time (JIT) compilation in the Erlang/Open-Telecom-Platform (OTP). This paper presents an in-depth investigation into the impact of JIT compilation in Erlang/OTP applications. We demonstrate our cross-platform measurement framework and evaluate a few problems on Windows and Linux with different BEAM versions. The findings emphasise the critical role of both proficient algorithm design and runtime optimisations in minimizing energy consumption and enhancing performance.

Keywords

Green computing, efficiency, Green software, Erlang, Optimisation, Compiler, JIT

1. Introduction

Erlang is a general-purpose, concurrent, functional high-level programming language that Ericsson developed in the late 1980s [1]. It can be used for a wide range of applications, some of its uses are in telecoms, banking, e-commerce, computer telephony and instant messaging. The runtime system of Erlang (the BEAM) inherently supports concurrency, distribution, and fault tolerance. Erlang/Open-Telecom-Platform (OTP) [1] provides a set of libraries and design principles providing middle-ware to develop massively scalable soft real-time systems with requirements on high availability.

The introduction of the Just-In-Time (JIT) compiler in Erlang OTP 24 has been one of the most exciting additions to the BEAM, aiming for simplicity and trading slight inefficiencies in the generated code for ease of implementation [2, 3].

JIT compilation plays a crucial role in modern programming languages. It aims to minimise or maximise some attributes of an executable computer program, such as execution time, memory footprint, storage size, and power consumption. The importance of JIT lies in its ability to improve the performance of a program by making it consume fewer resources.

The evolution of Erlang OTP has been marked by continuous improvements in its runtime system, libraries, and tools. These improvements have not only enhanced the performance of applications built on OTP but also contributed to reducing energy consumption. However, the specific details of how Erlang OTP has helped reduce energy consumption require further investigation, as the available literature does not provide explicit information on this aspect. This paper aims to analyse the effect of the new Erlang JIT on the runtime and energy usage of the Erlang programs. The study focuses on the performance of various algorithms, with a particular emphasis on the *fannkuch-redux*, *mandelbrot*, *n-body* problems, and a *server* problem. The paper also explores the inherent efficiency of certain implementations of the *N queens* problem.

Specifically, we aim to address the following research questions:

(1) What is the impact of the Erlang JIT on the energy consumption of Erlang programs compared to previous versions without the JIT compiler? This question seeks to evaluate how the new JIT affects the energy efficiency of Erlang programs. Energy consumption is a crucial factor in performance analysis, and understanding how JIT compilation influences it compared to traditional methods will provide insights into the practical benefits of this new technology.

SQAMIA 2024: Workshop on Software Quality, Analysis, Monitoring, Improvement, and Applications, September 9–11, 2024, Novi Sad, Serbia

ikycue@inf.elte.hu (G. Youssef); tothmelinda@elte.hu (M. Tóth); bozoistvan@elte.hu (I. Bozó)
0009-0005-3183-0343 (G. Youssef); 0000-0001-6300-7945 (M. Tóth); 0000-0001-5145-9688 (I. Bozó)

(2) How does the input size and implementation behave when used with and without JIT compilation? This question focuses on the performance implications of varying input sizes and implementations under both JIT and non-JIT compilation conditions. By analyzing how different factors influence performance, we aim to identify any significant variations and understand the conditions under which JIT provides the most benefit.

Besides the analysis of the BEAM JIT, we are presenting a cross-platform measurement framework for Erlang built as an extension of our Scaphandre-based [4] framework for Windows [5]. Scaphandre is a monitoring solution designed to track and analyze energy consumption metrics. We use this framework for analysing different OTP versions on Windows and Linux.

The rest of the paper is structured as follows. We start with context and related work presented in 2. In Section 3 we summarise our previous results on Erlang green coding measurements. In Section 4 we show the extension of our measurement framework originally developed for Windows. Section 5 details the performed measurement on Windows and Linux and discusses our main findings. Finally, Section 6 concludes the paper.

2. Related work

The concept of JIT compilation is not exclusive to Erlang/OTP but is also prevalent in other programming languages, notably Java. The Java Virtual Machine (JVM) has been a subject of extensive research and development with regards to JIT compilation and its impact on energy consumption [6, 7].

In the context of JVM, JIT compilation has been found to significantly reduce energy consumption by reducing instruction counts. However, it is worth noting that certain components such as garbage collection, which is an integral part of JVM, can incur runtime overhead that consumes more energy. Despite this, both JIT optimisation and garbage collection have been found to decrease the average power dissipated by a program[6].

A study evaluating the impact of different JVM distributions and parameters on energy consumption found that some JVM platforms can exhibit up to 100% more energy consumption [7]. Interestingly, the default configuration of the garbage collector was found to be energy efficient in only 50% of the experiments. This highlights the importance of carefully selecting and configuring the JVM distribution for energy efficiency.

In another study, the energy consumption of different Java I/O libraries and methods was compared, to reduce the overall energy consumption by substituting default I/O methods with ones that reported better energy efficiency [8].

These studies demonstrate that similar to Erlang/OTP, JIT compilation and careful configuration of the runtime environment play a crucial role in the energy efficiency of Java applications. This underlines the importance of continued research and development in this area to further optimise the energy consumption of software systems.

Compiler optimisation is another critical area that significantly impacts the performance of a program [9, 10, 11]. Optimizing compilers apply a series of transformations to the intermediate code to produce a semantically equivalent output program that uses fewer resources or executes faster. The latest OTP has seen enhancements in type-based optimisations and other performance improvements in the compiler and JIT [3].

In their comparative analysis, Sayed Najmuddin et al [12] investigated power and battery consumption in Windows operating systems and modern Linux distributions (such as Ubuntu, Fedora, Debian, Red Hat, and others). They highlighted the critical role of driver quality in Linux power management. Often, poorly written or incompatible drivers hinder efficiency. The researchers also pointed out that some Linux kernel versions are suboptimal for mobile systems, as they primarily target desktop platforms. They recommended that users should carefully choose an efficient kernel to enhance power and battery performance on Linux. These insights underscore the significant differences between Windows and Linux in terms of power management.

3. Background

Several Erlang applications are based on continuously running servers that consume lots of energy. Therefore, analysing the energy behaviour of Erlang applications has a significant impact on sustainability. Providing tools to measure the energy usage and introducing patterns to follow when creating energy aware applications are needed by the community. In our previous research [13, 5], we analysed the energy usage of different language constructs and libraries. At first, we performed measurements on Linux based on Running-Average-Power-Limit (RAPL) [14], which is a feature of Intel processors that measures and reports the CPU's energy consumption. We later adapted the framework for Windows, using the Scaphandre tool and a RAPL-driver [15] for Windows. Based on these experiments and the framework we created for Windows, we reevaluated part of the programming language comparisons presented in [16]. We showed that Erlang behaves better for concurrency related problems than it was concluded earlier in [16]. These measurements were carried out on a Windows environment.

3.1. Scaphandre based measurement framework for Windows

In our previous research, we aimed to extend the GreenErl [13] framework by integrating the Scaphandre [4] tool to enable accurate energy consumption measurements for Erlang functions on Windows. The original GreenErl tool was designed to measure the energy consumption of Erlang programs on Linux. The new GreenErl for Windows was built on top of a former GreenErl framework developed at the Eötvös Loránd University, Faculty of Informatics with the limitation of working only on Linux since it relies on the native RAPL tool [14] for energy profiling. We conducted a comprehensive measurement of energy consumption and runtime on data structures (lists, maps, dictionaries) and higher-order functions [5].

The integration of the Scaphandre tool into the GreenErl framework represents a significant milestone in enabling precise energy measurements for Erlang functions on Windows. This advancement bridges the gap left by the absence of traditional energy measurement tools like RAPL on Windows. By providing consistent and reliable data collection, the Scaphandre-based GreenErl framework serves as a vital entry point for our broader research into energy efficiency. It facilitates detailed energy profiling of various Erlang constructs. This work not only enhances our understanding of energy dynamics in software but also lays the groundwork for future research aimed at developing a cross-platform framework.

3.2. Comparing the energy usage of programming languages

Understanding the energy efficiency of different programming languages, especially in the context of JIT compilation in Erlang, is critical for developing sustainable software. Our previous work [17] utilised a subset of the Computer Language Benchmarks Game (CLBG)[1] and introduced a server implementation specifically for Erlang. These studies provide a foundation for our current research and underscore the importance of energy-efficient programming.

Building on the work from our study [17], our current research aims to delve deeper into the impact of JIT compilation in Erlang on energy efficiency and performance. By comparing results across different operating systems, we seek to provide a comprehensive understanding of how JIT optimizations can enhance energy efficiency in Erlang applications.

4. Cross platform framework

The integration of the Scaphandre tool into the GreenErl framework marked a significant advancement in our ability to measure energy consumption and runtime of Erlang functions not only on Windows but for enabling a cross-platform GreenErl framework. This new framework provides a robust solution for overcoming the limitations of traditional energy measurement tools on Windows, such as the absence of

[1]https://benchmarksgame-team.pages.debian.net/benchmarksgame/index.html

RAPL support. It enables precise and reliable energy consumption data collection for Erlang functions, allowing for a comprehensive analysis of their performance across multiple operating systems.

For this benchmark evaluation, we utilized a Lenovo ThinkPad X13 Gen 1 laptop, which is equipped with an Intel® Core™ i7-10610U processor featuring 8 cores and 16 threads. The system is configured with 32 GB of memory. The tests were conducted on two operating systems: Windows 11 Pro and Debian GNU/Linux 12 (Bookworm), both running in their 64-bit versions.

Scaphandre's integration ensures that energy measurements can be consistently collected on both Linux and Windows. This compatibility is crucial for conducting comparative studies and for developing energy-efficient software that performs optimally across various environments.

Additionally, the new framework allows for in-depth energy profiling of different Erlang constructs, we can obtain granular insights into the energy usage of Erlang programs, which is essential for optimising performance and reducing energy consumption.

The new GreenErl framework consists of several integral components, working in harmony to measure and analyze energy consumption in Erlang functions. At its core is a Python script that orchestrates the entire process. This script handles critical tasks. At first, function spawning creates instances of Erlang functions for measurement. Then the script initiates the Scaphandre process, which collects energy data as functions are executing, and aggregates the collected data into a structured JSON file. Once an Erlang function completes execution, the same Python script takes charge of measurement termination that stops the energy measurement process. The last step is data analysis, where the script calculates the energy consumption metrics from the JSON files and stores the results in CSV files. This streamlined approach ensures precise energy profiling and optimization across platforms.

5. Analysing the energy usage of Erlang programs

In our previous investigation [17], we conducted an in-depth analysis of Erlang's energy usage, focusing on its performance in scenarios that align with its design principles and compared it with other programming languages. We also created a framework for the energy measurements.

In this new investigation, we delve into the impact of the JIT compiler in the latest OTP 26 on Erlang's energy efficiency. We compare the performance of OTP 26 with that of OTP 23, using a few samples from our previous work such as a subset of the *Benchmark Game* and the server implementation. We also measured the n-queens problem implemented in two different ways [18]. These problems all together serve as our evaluation metrics[23]. To ensure a comprehensive comparison, we measured the same functions on identical hardware setups, with the only difference being the operating systems, Linux and Windows. This approach provides a clearer view of how the operating system influences the performance and energy efficiency of Erlang programs. However, we must state that some results show zeros, which could be attributed to the nature of the problem or the sampling tool. Futhermore, when the input size is small, the resulting energy may be so minimal that Scaphandre registers it as zero.

5.1. Selected problems

The Benchmark game and Server These implementations provide a comprehensive comparison of the performance and energy consumption of several compute-intensive programs from the Benchmark Game across different OTP versions.

The benchmark games present a set of problems that test the performance of the OTP versions under different computational tasks. In our new research, we used the *fannkuch-redux*, *mandelbrot*, and *n-body* problems. These problems provide a comprehensive evaluation of a language's computational efficiency and performance.

The server implementation simulates a typical use case for Erlang, allowing us to measure its energy efficiency in a practical context. It provides a real-world scenario where Erlang's strengths in handling

[2]https://github.com/joegharbi/SQAMIA_GREEN_JIT/tree/Linux
[3]https://github.com/joegharbi/SQAMIA_GREEN_JIT/tree/Windows

concurrency and distribution can be effectively measured.

N-Queens The *queens_list* and *n_queens* modules both solve the N-Queens problem, which involves placing N number of queens on an **NxN** chessboard such that no two queens threaten each other. The *queens_list* module, uses a list comprehension to generate the configurations to check if a queen can be placed in a certain position. The *n_queens* module uses a different approach. It uses backtracking to find a solution to the N-Queens problem.

5.2. Analysing the energy usage of Erlang programs on Windows

We will start by sharing the measurement results and findings from our Windows analysis.

5.2.1. Echo Server and Benchmark Game use-cases

The Just-In-Time compiler in OTP 26 significantly optimises runtime and energy consumption in all problems measured. This is evident from the data shown in Table 1, which shows lower values for these parameters in OTP 26 compared to OTP 23. As an illustration, let us consider the *n-body* problem. In OTP 23, the energy consumption was **692.41J** and the runtime was **193.98s**. However, with the introduction of JIT in OTP 26, there was a significant improvement. The energy consumption was reduced to **312.29J** and the runtime decreased to **84.03s**. This represents a reduction in energy consumption by approximately **55%** and a reduction in runtime by approximately **57%**. This example demonstrates the efficiency improvements brought by the introduction of JIT in OTP 26.

Table 1
Windows: Benchmarks OTP 23 vs 26

		OTP 23			OTP 26		
Module	**Arg**	**Energy [J]**	**Runtime [s]**	**Power [W]**	**Energy [J]**	**Runtime [s]**	**Power [W]**
fannkuch-redux	12	1681.7	83.62	20.11	1369.93	53.43	25.64
mandelbrot	16K	2556.72	150.98	16.93	1182.25	57.0	20.74
n-body	50M	692.41	193.98	3.57	312.29	84.03	3.72
server	10K	0.06	13.94	0.0	0.04	14.0	0.0
	4K	0.02	5.87	0.0	0.01	5.52	0.0

5.2.2. Experiments with the N-Queens Problem

Table 2 summarises our measurement results for the different implementations for the N-Queens problem.

Energy Consumption The energy consumption for both implementations seems to increase with the problem size on both OTP versions. However, the *queens_list* implementation appears to be more energy-efficient than the *n_queens* implementation, especially for larger problem sizes (i.e., when the number of queens is greater than 13). This could be due to the different algorithms used in the two implementations, with the *queens_list* implementation having a more efficient algorithm for handling larger problem sizes.

Runtime Performance The runtime also increases with the problem size for both implementations on both OTP versions. However, the *n_queens* implementation seems to have a slightly worse runtime performance for smaller problem sizes. During our experimental process, we encountered an issue where the execution of the function extended to a duration of 12 hours for an input size of *N=14*. This necessitated a manual intervention to halt the measurement, as further waiting was considered

Table 2
Windows: N Queens OTP 23 vs 26

		OTP 23			OTP 26		
Module	**Arg**	**Energy [J]**	**Runtime [s]**	**Power [W]**	**Energy [J]**	**Runtime [s]**	**Power [W]**
n_queens	10	59.16	16.21	0.11	47.01	13.66	0.12
	11	7.95	4.24	0.13	0.0	3.36	na
	12	1033.08	292.0	0.14	723.11	204.71	0.14
	13	287.51	80.5	0.13	210.94	57.12	0.13
queens_list	10	0.0	1.34	na	0.0	1.23	na
	11	0.0	1.4	na	0.0	1.26	na
	12	0.0	1.73	na	0.0	1.51	na
	13	3.18	3.77	0.31	0.0	3.3	na
	14	57.06	17.93	0.14	47.16	14.44	0.13
	15	455.89	133.73	0.14	372.55	112.46	0.15
	16	5872.31	2025.7	0.17	5270.34	1864.22	0.17

unproductive. The observation suggests that for input sizes of 14 and beyond, the function *n_queens* exhibits an exponential increase in execution time. This highlights the non-linear complexity of the implementation under investigation due to the backtracking nature of the implementation.

Power Usage The power usage seems to be fairly consistent across different problem sizes for both implementations on both OTP versions. This suggests that the energy efficiency of the implementations does not significantly change with the problem size.

OTP 23 vs OTP 26 For both implementations, the energy consumption and runtime seem slightly lower on OTP 26 compared to OTP 23. This could be due to the JIT compiler's optimisations in OTP 26, which can lead to more efficient code execution and thus lower energy consumption and runtime.

In conclusion, while the *queens_list* implementation appears to be more energy-efficient, the *n_queens* implementation may offer reasonable runtime performance for smaller problem sizes. However, for larger problem sizes, the *n_queens* implementation results in significantly longer runtime, as observed with **N** larger than *14.* These findings highlight the trade-offs between energy efficiency and runtime performance in the context of the N-Queens problem implemented in two distinct ways. It also shows the improvements in the OTP 26 using the JIT compiler compared to the older OTP 23 without the JIT compiler.

5.3. Analysing the energy usage of Erlang programs on Linux

The investigation into the energy usage of Erlang programs on Linux follows a similar methodology as the analysis on Windows. This involves comparing different versions of OTP (Open Telecom Platform) to evaluate improvements brought by the Just-In-Time (JIT) compiler introduced in OTP 24 and the further enhancements in OTP 26 and 27.

5.3.1. Echo Server and Benchmark Game use-cases

The Echo Server and Benchmark Game use cases provide practical scenarios for evaluating the energy efficiency and performance of Erlang. The focus is on typical computational tasks and server operations to understand the impact of the JIT compiler on energy consumption and runtime.

Benchmarks OTP 23 vs 26 on Linux The data presented in Table 3 indicates that OTP 26 offers significant improvements in both energy consumption and runtime for most benchmarks. The *fannkuch-redux* and *mandelbrot* modules show notable reductions in runtime and energy usage, illustrating the efficiency gains from the JIT compiler in OTP 26. The power usage shows a notable decrease, especially for *mandelbrot*, suggesting that the energy efficiency per unit of time has improved with OTP 26.

Table 3
Linux: Benchmarks OTP 23 vs 26

		OTP 23			OTP 26		
Module	**Arg**	**Energy [J]**	**Runtime [s]**	**Power [W]**	**Energy [J]**	**Runtime [s]**	**Power [W]**
fannkuch-redux	12	821.42	54.09	15.18	913.43	52.92	17.25
mandelbrot	16K	1554.38	112.28	13.84	854.62	48.02	17.79
n-body	50M	1913.58	136.62	14.00	1155.28	69.51	16.61
server	10K	4.34	9.70	0.44	4.21	9.31	0.45
	4K	1.75	3.86	0.45	1.75	3.81	0.46

Benchmarks OTP 26 vs 27 on Linux The differences between OTP 26 and OTP 27 are more nuanced (Table 4). While OTP 27 shows marginal improvements in some areas, such as the *fannkuch-redux* and *server* benchmarks, it does not consistently outperform OTP 26 across all metrics. OTP 27 generally shows slight improvements in runtime compared to OTP 26, though not uniformly across all benchmarks. The ratios indicate minor improvements or slight declines, suggesting that while there are optimisations in OTP 27, the benefits are not as pronounced as the jump from OTP 23 to OTP 26. This suggests that while further optimisations are present in OTP 27, the benefits may be more specific to certain workloads.

Table 4
Linux: Benchmarks OTP 26 vs 27

		OTP 26			OTP 27		
Module	**Arg**	**Energy [J]**	**Runtime [s]**	**Power [W]**	**Energy [J]**	**Runtime [s]**	**Power [W]**
fannkuch-redux	12	913.43	52.92	17.25	891.13	51.47	17.31
mandelbrot	16K	854.62	48.02	17.79	919.25	53.08	17.31
n-body	50M	1155.28	69.51	16.61	1166.53	72.62	16.06
server	10K	4.21	9.31	0.45	3.53	9.56	0.36
	4K	1.75	3.81	0.46	0.937	3.74	0.25

5.3.2. Experiments with the N-Queens Problem

The N-Queens problem provides another critical evaluation metric, examining both the *n_queens* and *queens_list* implementations for their energy usage and runtime performance across different OTP versions.

N Queens OTP 23 vs 26 on Linux The data presented in Table 5 shows that OTP 26 offers better performance and energy efficiency than OTP 23 across both implementations. For the *n_queens* implementation, the reductions in energy usage and runtime are particularly significant for larger problem sizes. The *queens_list* implementation also benefits from these improvements, though the differences are more pronounced in the larger problem sizes. The power usage generally improves, indicating better energy efficiency per unit of time with OTP 26.

N Queens OTP 26 vs 27 on Linux The transition from OTP 26 to OTP 27 showed better overall results (Table 6). However, while most scenarios demonstrate energy savings and reduced runtimes, some do not show a consistent improvement such as the *n_queens* with input 10 and 12, reflecting the

Table 5
Linux: N Queens OTP 23 vs 26

		OTP 23			OTP 26		
Module	**Arg**	**Energy [J]**	**Runtime [s]**	**Power [W]**	**Energy [J]**	**Runtime [s]**	**Power [W]**
n_queens	10	154.48	9.74	15.85	99.24	7.25	13.69
	11	44.34	3.6	12.33	0.0	2.91	0.0
	12	3792.48	266.35	14.24	2760.55	190.57	14.49
	13	1018.69	75.42	13.51	809.13	50.84	15.92
queens_list	10	0.0	1.2	0.0	0.0	1.23	0.0
	11	0.0	1.26	0.0	0.0	1.27	0.0
	12	0.0	1.53	0.0	0.0	1.48	0.0
	13	32.81	3.32	9.88	0.0	2.94	0.0
	14	223.21	15.21	14.67	164.26	12.2	13.46
	15	1526.1	114.43	13.34	1151.18	90.69	12.69
	16	10863.3	953.34	11.39	8214.54	774.58	10.61

Table 6
Linux: N Queens OTP 26 vs 27

		OTP 26			OTP 27		
Module	**Arg**	**Energy [J]**	**Runtime [s]**	**Power [W]**	**Energy [J]**	**Runtime [s]**	**Power [W]**
n_queens	10	99.24	7.25	13.69	110.13	7.08	15.56
	11	0.0	2.91	0.0	0.0	2.88	0.0
	12	2760.55	190.57	14.49	2707.43	178.56	15.16
	13	809.13	50.84	15.92	779.5	50.48	15.44
queens_list	10	0.0	1.23	0.0	0.0	0.36	0.0
	11	0.0	1.27	0.0	0.0	1.28	0.0
	12	0.0	1.48	0.0	0.0	1.47	0.0
	13	0.0	2.94	0.0	0.0	2.87	0.0
	14	164.26	12.2	13.46	150.91	11.81	12.77
	15	1151.18	90.69	12.69	1090.02	90.0	12.11
	16	8214.54	774.58	10.61	8850.71	842.9	10.5

nuanced impact of optimisations specific to different types of workloads. The ratios are also variable, reflecting mixed improvements in energy efficiency.

5.4. Findings

The analysis of energy usage and runtime performance of Erlang programs on Linux mirrors the patterns observed on Windows. The JIT compiler introduced in OTP 24 as well as the test versions 26 and 27 offers substantial improvements over OTP 23. Further enhancements in OTP 27 provide additional benefits. These findings underscore the importance of considering specific workload characteristics when evaluating the impact of compiler and runtime optimisations.

The detailed benchmarks across different Erlang modules and problem sizes illustrate the potential for significant energy savings and performance gains with newer OTP versions. However, the degree of improvement varies, indicating the complex interplay between program characteristics and runtime optimisations.

In conclusion, the move to OTP 26 and 27 on Linux demonstrates clear advantages in energy efficiency and runtime performance for many Erlang programs, aligning with the trends observed on Windows. This is not the first time we discover that Windows is generally more optimised as Sayed Najmuddin et al [12] found that Windows generally exhibits more efficient power and battery management than

Linux, largely due to better driver support and optimisation. In addition, we conclude that choice of OTP version should consider the specific workload to maximise the benefits of these advancements.

6. Conclusion and future work

In our previous research, we developed a framework for measuring the energy consumption of Erlang applications and investigated the energy behaviour of different Erlang implementations. In this paper, we demonstrated a cross-platform solution to carry out energy-related measurements of Erlang application. Our solution is based on Scaphandre and RAPL.

Using this framework we have investigated the energy behaviour of the Erlang BEAM virtual machine and compared OTP 23 (non-JIT) implementations with OTP 26 implementations (JIT). In addition, we have analysed the recent improvement to the BEAM JIT by comparing OTP 26 and OTP 27. we have addressed our first research question.

Furthermore, our analysis has demonstrated that irrespective of whether a JIT or non-JIT compiler is used, the implementation approach of the software profoundly influences its energy consumption. This finding addresses our second research question and underscores the critical role of software design in energy efficiency.

This research has made significant progress in understanding the performance of various algorithms in the Erlang runtime system. The introduction of JIT compilation in OTP 26 has shown to significantly improve the runtime and energy efficiency of these algorithms.

The research also highlighted the implicit efficiency of certain implementations, such as the *queens_list* module. These findings underscore the importance of both efficient algorithm design and runtime optimisations in reducing energy consumption and improving performance.

Looking ahead, there are several promising directions for future research. One potential path is to extend this analysis to other algorithms in Erlang, to gain a more comprehensive understanding of the performance characteristics of the Erlang runtime system. The findings of this research provide a solid foundation for future investigations in this dynamic and rapidly progressing field.

Acknowledgments

Project no. TKP2021-NVA-29 has been implemented with the support provided by the Ministry of Culture and Innovation of Hungary from the National Research, Development and Innovation Fund, financed under the TKP2021-NVA funding scheme.

References

[1] F. Cesarini, S. Thompson, Erlang Programming: A Concurrent Approach to Software Development, O'Reilly Media, Inc., 2009. Https://www.oreilly.com/library/view/erlang-programming/9780596803940/.

[2] J. Högberg, A first look at the JIT - Erlang/OTP, 2020. URL: https://www.erlang.org/blog/a-first-look-at-the-jit/.

[3] B. Gustavsson, More Optimizations in the Compiler and JIT - Erlang/OTP, 2023. URL: https://www.erlang.org/blog/more-optimizations/.

[4] Hubblo, Scaphandre documentation, 2023. URL: https://hubblo-org.github.io/scaphandre-documentation/index.html.

[5] Y. Gharbi, I. Bozó, M. Tóth, Green computing for Erlang, in: To appear in the Proceedings of the 3rd Workshop on Resource AWareness of Systems and Society (RAW 2024), CEUR, 2024.

[6] S. Hu, L. K. John, Impact of virtual execution environments on processor energy consumption and hardware adaptation, in: Proceedings of the 2nd International Conference on Virtual Execution Environments, VEE '06, Association for Computing Machinery, New York, NY, USA, 2006, p. 100–110. doi:10.1145/1134760.1134775.

[7] Z. Ournani, M. C. Belgaid, R. Rouvoy, P. Rust, J. Penhoat, Evaluating the Impact of Java Virtual Machines on Energy Consumption, in: Proceedings of the 15th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), ESEM '21, Association for Computing Machinery, New York, NY, USA, 2021. doi:10.1145/3475716.3475774.

[8] Z. Ournani, R. Rouvoy, P. Rust, J. Penhoat, Comparing the Energy Consumption of Java I/O Libraries and Methods, in: ICSME 2021 - 37th International Conference on Software Maintenance and Evolution, Proceedings of the 37th International Conference on Software Maintenance and Evolution (ICSME), Luxembourg / Virtual, Luxembourg, 2021. URL: https://inria.hal.science/hal-03269129.

[9] S. Muchnick, Advanced compiler design and implementation, Morgan Kaufmann, 1997.

[10] S. Wong, Super optimization, 2020. URL: https://www.cs.cornell.edu/courses/cs6120/2020fa/blog/super-optimization/.

[11] GCC, Optimize options (using the gnu compiler collection (gcc)), 2014. URL: https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html.

[12] S. Najmuddin, Z. Atal, R. A. Ziar, Comparative analysis of power consumption of the linux and its distribution operating systems vs windows and mac operating systems., Kardan journal of engineering and technology (2021). doi:10.31841/KJET.2021.21.

[13] G. Nagy, A. A. Mészáros, I. Bozó, M. Tóth, Tools supporting green computing in Erlang, in: Proceedings of the 18th ACM SIGPLAN International Workshop on Erlang, Erlang 2019, Association for Computing Machinery, New York, NY, USA, 2019, p. 30–35. doi:10.1145/3331542.3342570.

[14] V. M. Weaver, Reading RAPL energy measurements from Linux, http://web.eece.maine.edu/~vweaver/projects/rapl/, 2015.

[15] Hubblo, Windows driver to get RAPL metrics, 2023. URL: https://github.com/hubblo-org/windows-rapl-driver.

[16] R. Pereira, M. Couto, F. Ribeiro, R. Rua, J. Cunha, J. P. Fernandes, J. Saraiva, Energy efficiency across programming languages: how do energy, time, and memory relate?, Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering - SLE 2017 (2017). doi:10.1145/3136014.3136031.

[17] Y. Gharbi, M. Tóth, I. Bozó, Measuring and Analysing Erlang's Energy Usage, in: 2024 47th MIPRO ICT and Electronics Convention (MIPRO), 2024, pp. 1993–1998. doi:10.1109/MIPRO60963.2024.10569977.

[18] N-queens problem - rosetta code, 2024. URL: https://rosettacode.org/wiki/N-queens_problem#Erlang.