COMPARATIVE ANALYSIS OF VERBAL LANGUAGES FOR COMPUTING TO DETERMINE WHICH LANGUAGE POSSESSES THE HIGHEST EFFICACY AND COMPUTATIONAL LOGIC FOR COMPILER OPTIMIZATION

Bodle, W. L., McKenney, M., Segura, D., & Trafton, C.

Department of Computer Science, California Baptist University,Riverside, CA 92504

ABSTRACT

This paper presents a comparative analysis of natural languages to discern their impact on the speed and optimization of compilation times within Java and C++ compilers. There lies a dominance of English in programming languages and limited research in this area. Our study explores potential barriers for non-English speakers and aims to identify correlations between written language and compilation efficiency. As we plan to unravel the impact of different natural languages on the speed and optimization of compilation time in C++ and Java we will be opening the door to a new idea of research. These insights contribute to the advancement of diversity and accessibility in computer software engineering as well as improved compilation times.

KEYWORDS

Compiler Optimization, Natural Languages, Compilation Time, Linguistic Factors on Compilation, Dominance of English in Programming

1. INTRODUCTION

In the world of contemporary software development, we are constantly seeking optimization. This extends to many different areas of software engineering, but for this paper we will be focusing on the functionality of compilers. As we continue the relentless pursuit for quicker speeds, reduced timelines, and raised system performance we will be introducing different natural languages on the effect of compilation time.

The first compiler in the modern sense was developed in 1952 at the University of Manchester. Since then, compilers are constantly being optimized to increase speedup, reduce development time, and improve overall system performance. Doing so enhances developer productivity and allows for quicker feedback for developers.[1] While compilers have been optimized through machine learning and modern programming techniques, most programming languages, libraries, and documentations are in English.

This poses a difficulty for non-English speakers when reading instructional materials or learning to code.[2] While non English-speaking programmers have trouble programming at a high level we will similarly be studying the effect of different natural languages on the speed and optimization of compilers. To study the effect of different languages on compilation time we will be conducting tests on different written languages on a Java and C++ compiler. By comparing the

compilation time of the same program in different languages we should be able to determine which written language is the easiest for a compiler to handle.

Our approach involves an examination of the compilation times of identical programs on both a Java compiler and a C++ compiler. We aim to discern which written language proves to be the most accommodating for compilers. By comparing the compilation times of our languages with identical programs, we hope to discover a pattern that indicates which written language can be handled more efficiently by compilers. The findings from our study could offer valuable insights for developers, language designers, and educators. Having an optimized compiler leads to improved productivity, efficient iteration, enhanced user experience, and resource utilization. As we uncover which written languages are handled more efficiently by compilers, we aim to pave the way for advancements that empower non-English speaking programmers and foster a more inclusive and efficient programming ecosystem.

2. BACKGROUND

An article, "Program energy efficiency: The impact of language, compiler, and implementation choices," by Sarah Abdulsalam, et al. explores the task of reducing energy usage in computing systems. Their goal is also to optimize compilers to help programmers write more efficient code.[3] By optimizing compilers we can improve system performance which is a goal we both align on. We both recognize the significance of improving efficiency in software development although we have a different focus. Our study focuses on the effect of different natural languages on compilation time while they focus on writing more energy-efficient code. Their method for determining a more energy efficient program was based on three factors, appropriate language, optimization flag, and data structure. Our research focuses only on appropriate language, both programming and written languages.

Another paper written by Aidan Hall aligns with our research, which explores the impact of natural languages on compilation times in Java and C++ compilers. He critiques the prevailing ASCII text-based programming languages, emphasizing the challenges they pose for non-native English speakers and suggesting improvements for greater accessibility. Our study, too, recognizes the dominance of English in programming languages and seeks to identify patterns in compilation efficiency across different languages. The proposed enhancements, such as Unicode operators and identifiers based on non-English natural languages, resonate with our aim of understanding the linguistic factors influencing compiler optimization. Both works share a common goal of fostering inclusivity in programming by addressing language-related barriersand advancing the evolution of programming languages for improved efficiency and expressiveness.[4]

If we can reveal the best language for compiler optimization it could lead to an easy method to speedup programs. This is assuming that the world would be willing to break away from the programming norm where documentation and languages are predominantly written in English. This could also give non-English speakers an argument toward having more documentation written in their language to aid growth of technological fields in places that would otherwise require people to learn English in order to program efficiently and effectively.

3. METHOD

Our method to address this is rooted in recognizing that while there is constant work done in the optimization of compilers to enhance performance, the impact of different natural languages on compilation time is unexplored. Most languages used for programming are written in English

with all documentation also written in English, leaving very little room for growth in programming with different languages. This can create a barrier for non-English speakers but may also be affecting our programming and compilation speed due to the way programming has been set in English.

The hypothesis we have reached is that the choice of a written language may influence the speed and optimization of compilers. We plan to investigate this and determine if there really is a correlation between natural language and compilation speed. We plan to test both Java and C++ compilers, with the goal of comparing compilation times for the same program but written in different languages. The compilers we will be using are the javac 21 for Java and the Apple Clang version 15.0.0 for C++. We hope to reveal patterns in which different written language is handled more efficiently.

To prove the validity of our approach, we must statistically analyze the compilation time data. For this we will let μ_i represent the compilation time for each program in language i where i ranges over the five languages we will be studying. English, Chinese, French, Italian, and Russian. We can create a test to determine whether there are significant differences in compilation times between languages. We will be using a one-way analysis of variance (ANOVA), where we will use this for the null hypothesis:
$H0: \mu_1=\mu_2= \mu_3=\mu_4 =\mu_5$

ANOVA stands for Analysis of Variance and is the statistical test used to analyze the difference and significance between the means of more than two groups. We will be using five different groups across two separate ANOVA tests for program one and two. It works by comparing the means of each group including spreading out the variance into diverse sources. Using ANOVAwe will calculate an F-statistic, using that we can indicate if there are significant differences between the language group means.

The alternative would be that at least one i is different. We will be collecting compilation time data against different programs and languages; we will then perform a statistical test to assess whether the observed differences in compilation time are proven to be statistically different. If our p-value is below 5% we can argue to reject the null hypothesis. If our F-statistic is large and the p-value is small this is an indicator of significant differences in compilation times among languages. To achieve this we will be using a python program to output our statistical data for four different sections. Java and C++ in program one which is a loop of arithmetic and Java and C++ in program two which is a typical “Hello World” Program.

4. RESULTS

After concluding our experiments, we were able to get data from two different programs with three runs each against five different languages. Table 1 displays the data found in tabular form. From this data we can notice slight patterns between each language and the compilation time in Java and C++. Using a script, we were able to time the compilation time of each of the five languages over four different programs, two being in Java and two in C++. The script we wrote is designed to measure the compilation time of Java and C++ programs for each language. We first started by getting a timestamp now of runtime using the gdate command and storing it in a variable. We then ran a java compiler, for example the English file was compiled using “javac English.java”. After the program had compiled, we stored a new timestamp in another variable and outputted the difference between the two. This gave us an easy and accurate way to measure the time it took each file to compile. We wrote and compiled all the programs on the same machine and development environment. Our data shows that English typically takes longer to compile than other languages.

Table 1. Compilation Times vs. Languages

Program 1 (Arithmetic in loop)								
	Run 1		Run 2		Run 3		Average Comilation Time	
	Java (ms)	C++ (ms)	Java (ms)	C++ (ms)	Java (ms)	C++ (ms)	Java (ms)	C++ (ms)
English	593	760	696	939	608	776	632.3	825.0
Chinese	404	358	407	357	407	363	406.0	359.3
French	395	357	396	361	414	357	401.7	358.3
Italian	396	357	402	368	398	356	398.7	360.3
Russian	415	358	413	369	422	258	416.7	328.3

Program 2 ("Hello World")								
	Run 1		Run 2		Run 3		Average Comilation Time	
	Java (ms)	C++ (ms)	Java (ms)	C++ (ms)	Java (ms)	C++ (ms)	Java (ms)	C++ (ms)
English	370	358	578	746	580	812	509.3	638.7
Chinese	380	356	378	360	380	359	379.3	358.3
French	364	356	367	355	370	371	367.0	360.7
Italian	377	356	381	358	393	356	383.7	356.7
Russian	370	357	384	360	386	360	380.0	359.0

Our study has determined that Russian has shown to be quicker in program one while Chinese was the fastest in program two. English was slower in both instances for both languages with the exception of the "Hello World" program in Java run one. French, Italian, and Russian seemed to be fairly similar in most runs without a large change in compile time. We can see that the average compilation time for English running in Java was exceptionally high at 632.3 milliseconds while the mean was 451.1 milliseconds. In Figure 1 we can identify English as an outlier in that it took almost twice as long to compile where Russian performed very well in C++ on run 3. Chinese, French, and Italian all had relatively similar compilation time averagingaround 360 milliseconds in C++ and 400 milliseconds in Java.

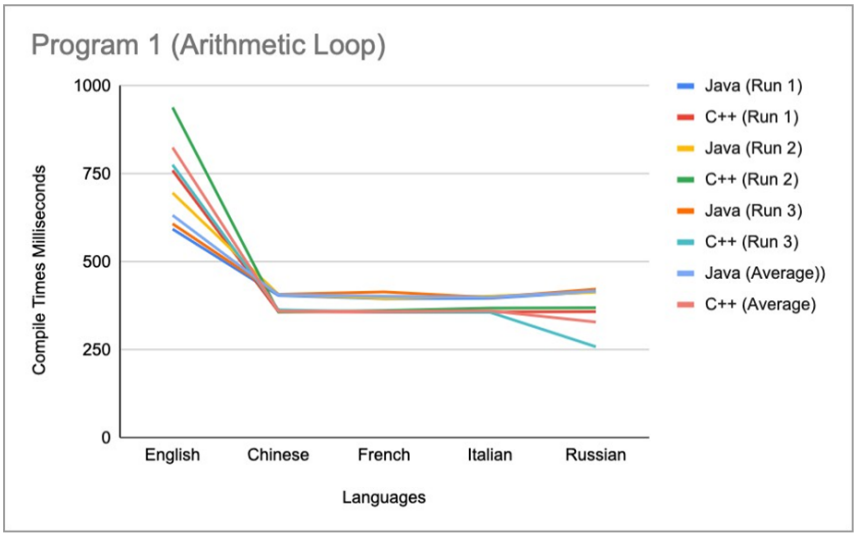

Figure 1. Program One Compilation Times vs. Language

According to our data shown in Figure 3 we do notice that the statistical difference between the two programs themselves differ as well. Program one showed a greater significant difference than program two. We would expect to see this because program two is almost the simplest program you can write while program one would model a real-world small program involving arithmetic that might be used in industry. Both were proven to be statistically significantly different but program two was just on the edge of that decision.

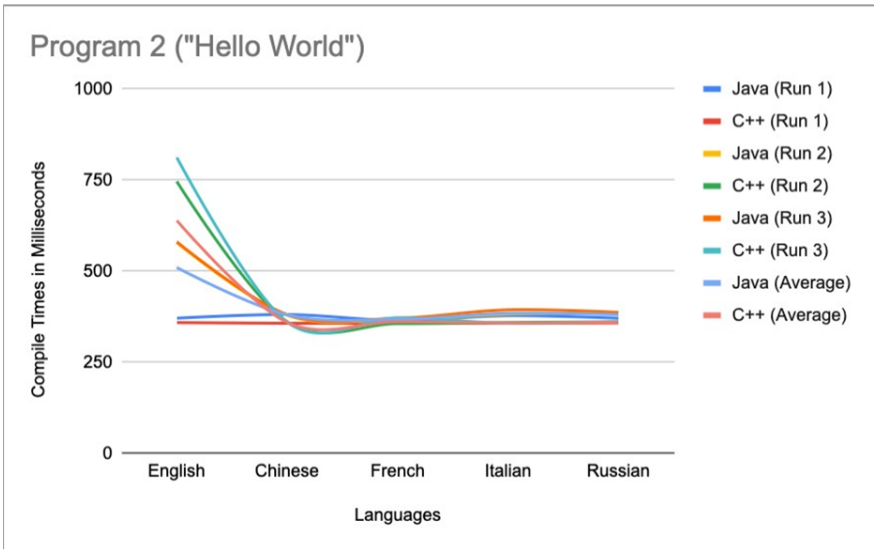

Figure 2. Program Two Compilation Times vs. Language

After obtaining the data in Figure 1 we could then decide to move to our ANOVA test to determine if this data was significantly different across languages. Using a python program, we wrote we were able to reliably get both F and P statistics for comparison to determine if our data would pass the hypothesis. Figure 2 identifies the results of our ANOVA test in the IDE console. Our python program performs ANOVA using the 'f_oneway' function in the 'scipy.stats' module. We first imported our modules, then defined our compilation times for program one and program two. The program then processed and outputted the F-statistic and p-value to the console which is displayed in Figure 3.

```
Program 1 Java
F-statistic: 47.67394207066542
P-value: 1.7662095304273512e-06

Program 1 C++
F-statistic: 49.596929029730205
P-value: 1.4657727084041403e-06

Program 2 Java
F-statistic: 3.583078525967704
P-value: 0.04625333820620936

Program 2 C++
F-statistic: 3.9037062646936853
P-value: 0.03670997708039739
```

Figure 3. ANNOVA Statistics Test

The ANOVA data was derived following these specific steps, calculating the sum of squares, determining the degrees of freedom, calculating the mean squares, calculating the F-statistic and determining the critical value. An ANOVA test is deemed as significantly different if the P- value is lower than 0.005 and the F-statistic is high. We can see this in both Java and C++ compiler times in program one meaning that the compilation times in the looped arithmetic program are significantly different. We do not see as big of a difference in program two. In program two we can see that the F-Statistic is relatively low while the P-value is just under 0.05. From this we can draw the conclusion that it is not as significantly different as programone.

5. DISCUSSION

Through the data we have collected we will now dive into its implications to fill the existing gap and relate it to the hypothesis we have previously addressed. We hypothesized that different choices of written languages such as English, French, Italian, Chinese, German, and Russian may play a factor in the speed of compilation. We ran a study on identical programs written in different languages to determine which is best for the optimization and speed of compile time. Our result is illustrated in Figure 1 and Figure 2 and supported by our calculations shown in Figure 3.

Our study provides valuable insights into the influence of natural languages on the speed and optimization of compilation time. This is a largely unexplored area in the field due to the abundance of English documentation and resources. While other works such as that of Abdulsalam et al. focuses on compiler optimization for the reason of energy efficiency there are few if any studies that reference different languages' effect on compiler optimization. This knowledge could lead to new avenues for language designers and developers with respect to the linguistic factor of compiler optimization. Our focus is on comparative analysis of different natural languages for compilation speed with respect to both Java and C++ compilers. Our work complements the existing research done by Abdulsalam et al. by expanding the scope of optimization to consider linguistic factors.

Through our analysis of English, Chinese, French, Italian, and Russian we reveal an interesting inequality. Russian demonstrated quicker compilation times in program one, while Chinese was the quickest and outperformed the others in program two. English in both experiments exhibited longer compilation times. There can be a number of different factors that lead to these conclusions, parsing complexity, language features, and language complexity. Figure 4 demonstrates the effect of language features and language complexity that may be leading to the results found.

Figure 4. UTF-16 Encodings in English vs Chinese

In the “Hello, World!” program, the only difference the compiler would recognize is the string literal being output. For this reason we can determine that the language with the smallest amount of characters per word would be the most efficient for compilation. This is most likely why Chinese has outperformed all of the other languages in program two. Chinese has a more concise representation of characters which could lead to quicker compilation times. Chinese has

a logographic writing system with fewer characters per word on average which could lead to a notable advantage. Beyond character density, there could be a parsing complexity and different language features that could be attributed to the complexity of the English language in compilation.

6. FUTURE DIRECTIONS

While our study sheds light on the different influences of natural languages on compiler optimization, there are many different ways for our research to expand. We ran our study withfive different natural languages with two different compilers, C++, and Java. By adding We could expand both the natural languages and the compilers use to deepen our understanding of the effects of natural languages on compilation time.

Another direction for compiler optimization with an emphasis with different natural languages could be to integrate machine learning techniques into our compiler optimization process. By investigating the correlation between natural languages and machine learning based compiler optimizations we could yield insights into creating an adaptive compiler that would be extremely optimized.

Apart from compilation times we could also broaden our study into the concept of developer experience. This would include different factors that lead to the overall positive or negative experience of the programmer on their environment. Some factors may include code readability, ease of debugging, and program efficiency. The user experience for a non-Englishspeaker can be greatly changed in a positive manner by giving a non-English outlet for programming and documentation which would improve all of the above factors.

Lastly, we could incorporate cross-language compilation. Some languages interact differently to the compiler and by studying this further we could be able to determine in which scenarios different natural languages would prevail. After determining the best cases for each language, we could want to study the ability of cross compilation on different languages to improve overall compilation speed and program times.

7. ETHICAL & ECONOMIC CONSIDERATIONS

When considering different languages for compiler optimization we must also study the ethical considerations of our research. Language diversity is crucial in enabling cultural diversity in programming. Any of our findings are to promote inclusivity rather than to reinforce language biases in connection with computer science and software engineering. Our research underscores the dominance of English in programming, raising ethical considerations about inclusivity in language design. As the future of computer engineering, we should aim to improve inclusivity by considering linguistic diversity in the designs of our compilers and programming languages.

Along with the ethical considerations of our research of different natural languages on compilation efficiency, we will be reviewing the economic considerations of program and compiler design in other languages. It is known that there is a correlation to the expansion of

technology companies and research with economic prosperity. For this reason, we can identify that the prevalence of certain languages in programming can influence different industry trends and innovation along with job markets and skill demands. By creating diversity in program and compilation design we can stimulate and encourage exploration of new paradigms and problem-solving approaches. Allowing for other culturally non-English speaking areas to program in their native language we could foster innovation and development that can attract investment, talent and collaborations.

Looking ahead, we can pave the way to potential avenues for technological innovation of compiler design. We could see advanced language models and compiler integration due to natural language processing. Future compiler may use different sophisticated language models

to comprehend and adapt to different diverse language structures. This could lead to compilers that dynamically tailor their optimization to different natural languages being programmed in.

8. CONCLUSION

Our study aimed to explore the impact of different natural languages on the speed and optimization of compilation times within a Java and C++ compiler. Our research was inspired by the dominance of English in programming languages with very little previous studies involved. This potentially creates barriers for non-English speakers and influences compilation efficiency. Our comparative analysis encompassed English, Chinese, French, Italian, and Russian. This revealed intriguing patterns. Russian exhibited quicker compilation times in program one, while Chinese surpassed others in program two. Conversely, English surprisingly demonstrated longer compilation times. We determined this is most likely due to Parsing complexity as illustrated in Figure 3.

Our work contributes to insights in a largely unexplored area in the field, complementing existing research on compiler optimization. While previous studies like Abdulsalam et al., primarily focused on energy efficiency, our emphasis is on the linguistic factors and expands the scope of optimization considerations for language designers and developers. Our findings show the effect of different language choices and their impact on compiler efficiency. By recognizing the influence of different natural languages on compilation speed we have opened new avenues for optimizing the compilers and promoting inclusivity in the global programming landscape.

As we end our study, the avenues for research in the future on this topic are vast. Expanding our study to more programming languages we can deepen our understanding of different language effects on compilation time. We might also plan to incorporate the integration of machine learning into compiler optimization. As the world moves further toward artificial intelligence and machine learning it would only make sense that compiler design would follow a similar trend. We would also like to consider the developers experience, primarily those who are not native English speakers on code readability, ease of debugging, and access to documentation.

We must also consider the ethical and economic implications of different languages in programming and compilation design. We should direct our efforts toward creating programming languages easier and more accessible to a global audience, avoiding biases and barriers. We can create more opportunities and foster a more competitive technology sector. We may have more advanced models and integration with natural language processing which could dynamically tailer compilation optimization to different natural languages.

REFERENCES

[1] M. Demertzi, M. Annavaram and M. Hall, "Analyzing the effects of compiler optimizations on application reliability," 2011 IEEE International Symposium on Workload Characterization (IISWC), Austin, TX, USA, 2011, pp. 184-193, doi: 10.1109/IISWC.2011.6114178.

[2] Z. Wang and M. O'Boyle, "Machine Learning in Compiler Optimization," in Proceedings of the IEEE, vol. 106, no. 11, pp. 1879-1901, Nov. 2018, doi: 10.1109/JPROC.2018.2817118.

[3] S. Abdulsalam, D. Lakomski, Q. Gu, T. Jin and Z. Zong, "Program energy efficiency: The impact of language, compiler and implementation choices," International Green Computing Conference, Dallas, TX, USA, 2014, pp. 1-6, doi: 10.1109/IGCC.2014.7039169.

[4] A. Hall, "Why does English-based, 'typewritten' code remain dominant, and what problems does it present?" May 12, 2022. Retrieved from https://dcs.warwick.ac.uk/~u2106099/docs/typewritten-code.pdf

[5] L. A. Zadeh, "From computing with numbers to computing with words. From manipulation of measurements to manipulation of perceptions," in IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications, vol. 46, no. 1, pp. 105-119, Jan. 1999, doi: 10.1109/81.739259.

[6] Pandža, N. B. (1970, January 1). *Computer programming as a Second language*. SpringerLink. https://link.springer.com/chapter/10.1007/978-3-319-41932-9_36

[7] P Foster, A Tonkyn, G Wigglesworth, Measuring spoken language: a unit for all reasons, *Applied Linguistics*, Volume 21, Issue 3, September 2000, Pages 354–375, https://doi.org/10.1093/applin/21.3.354

[8] Joao, R. (2022, November 24). On the linguistic and computational requirements for creating face-to ... https://arxiv.org/pdf/2211.13804

[9] Araujo, V., Marie-Francine Moens, & Soto, A. (2023). Learning sentence-level representations with predictive coding. Machine Learning and Knowledge Extraction, 5(1), 59. doi:https://doi.org/10.3390/make5010005

[10] Wasserman, Anthony,(2023) Design of Software Representation Languages: A Historical Perspective. "https://ssrn.com/abstract=4382944"

[11] Sperber, D. (1994). Understanding verbal understanding. What is intelligence, 179-198.

[12] Tucker, A. (2003, June 13). *Very high-level language design: A viewpoint*. Computer Languages. https://www.sciencedirect.com/science/article/pii/0096055175900041

[13] Philip J. Guo. 2018. *Non-Native English Speakers Learning Computer Programming: Barriers, Desires, and Design Opportunities*. In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18). Association for Computing Machinery, New York, NY, USA, Paper 396, 1–14. https://doi.org/10.1145/3173574.3173970