Honors 499Y Project Proposal

Comparing and Contrasting Performance Metrics for RUST and C/C++, by Detecting and Analyzing Phase Change Metrics.

# Kushagra Srivastava

## Under Guidance of Prof. Meng-Chieh Chiu

# Basic Form Information

* I am registering a research manuscript
* Semester Fall 2023
* For 4 credits (expected is 3 but I have more time to dedicate)
* I will meet my supervisor for 1 hour per week (expectation is ½ hours, time can vary)
* I need these credits to reach full time status (12 credits): No
* Committee Chair: Meng Chieh Chiu
* This work is a continuation of a prior Independent Study (Not ISH)
* This work does not involve human/animal subjects

# 

# Statement of Objectives

The following Research Proposal draws inspiration from and is used as an extension of the paper, "Real-Time Program-Specific Phase Change Detection for Java Programs", authored by Prof. Meng-Chieh Chiu, Prof. Eliot Moss, and Prof. Benjamin Marlin.

During the compilation of a program in a given programming language, the program goes through different phases, such as: initialization, parsing, semantic analysis, optimization, and code generation. By analyzing phase changes in different programming languages, we can notably infer how quick and robust a given programming language can be in these stages. This data may help us in comparing performance metrics between different programming languages, as well as provide tools for other developers or researchers for further implementations.

The main idea behind this project would be to detect Real-time Phase Changes in the compilation and execution of different programming languages in different languages. Notably, we plan to target newer programming languages (RUST, in our case), and compare their performance against established System Programming languages like C/C++, to infer their performance across different metrics of code compilation and code execution.

The following research is divided into two main segments. The first part is building tracing tools for x86 Assembly, and a tracing tool for the RUST version of assembly, to compare performance between C and RUST. This part mainly gives an insight on how refactoring legacy C-based systems to RUST may be more efficient and beneficial on a large scale, especially since logically the exact same code would be tested on two different compilers. The second part is to build a Phase Change Detection tool for RUST to dive deeper into the code compilation process to understand how efficient RUST can be for the purposes of developing newer systems level software, such as operating systems. We gain a better perspective on compiler level operations, thus providing a more holistic view of the RUST compilation process, and how systems can benefit from migrating to RUST.

For the first phase, we plan to derive the performance metrics of C and RUST through generating a trace of the x86 Assembly output of similar programs written in both languages. After similar programs are written across the different programming languages, we generate the Assembly output from the two languages, and create breakpoints at certain places to trace the compilation process. Generating the trace will give an insight on how much computationally intensive resources were used by each compiler, and how efficient each language is; essentially giving an in-depth look into how many resources were used and operations executed at the machine level to achieve the desired output. Thus, metrics on comparative performance and compiler efficiency can be derived in this manner.

The second phase dives more into the actual compiler execution for RUST, and goes over each of the 5 different phases for compilation, starting with code and ending at producing the executable file. We plan to detect the real-time phase changes for each step of the compilation as discussed in the paper mentioned above (Chiu *et. al*), and follow a similar approach. We detect phase changes through recording various time intervals between different phases, and to cluster them based on the similarity of their feature vectors: the number of similar properties that specific phase consists of (which would tell how similar or different a given phase is). These metrics would then be analyzed using the Gaussian Mixture Model (GMM): which is a “probabilistic model generalizing k-means clustering to incorporate information about the covariance structure of the clusters.” (Chiu *et al.*, 2016).

The benchmarks which would be used under this project are still under consideration, but we believe that since we are dealing with a relatively newer language, there may exist a greater flexibility in terms of observable properties, thus leading to a better understanding and wider range of data collection. At best, we plan to reduce it to machine-level code, generating something akin to x86 Assembly with the RUST compiler, and tracing benchmark points before we calculate overall how better/worse each language performed with respect to each other, to achieve the same task.

The following study would help us get a better picture on how newer languages may be better than older languages, especially when both languages are used for very specific and similar purposes (Systems Programming, in the case of this project). Both RUST and C offer a high degree of flexibility in terms of user control over processes, memory, and are very strongly typed. However, RUST benefits from the fact that it is relatively newer, has better memory management, and focuses explicitly on preventing memory leaks and minimizing user error. Thus, it may be interesting to see how a newly developed language performs, as compared to an older, established language.

Such an insight would help programmers working in Systems gain a better understanding of the robustness of newer languages as compared to older ones, and may help in choosing the best fit for their use case. Moreover, this project would also act as a means of contributing valuable RUST tools for programmers and researchers.

# Key Readings

The following research papers would be used as key readings for the paper (will be cited in an appropriate manner):

1. Real-Time Program-Specific Phase Change Detection for Java Programs: Prof. Meng-Chieh Chiu, Prof. Eliot Moss, and Prof. Benjamin Marlin: <https://www.researchgate.net/publication/310824600_Real-Time_Program-Specific_Phase_Change_Detection_for_Java_Programs>
2. RUST Compiler Documentation: <https://rustc-dev-guide.rust-lang.org/>
3. GCC Compiler Documentation: <https://gcc.gnu.org/onlinedocs/>
4. Run-time program-specific phase prediction for python programs: Prof. Meng-Chieh Chiu, Prof. Eliot Moss: <https://www.researchgate.net/publication/327333194_Run-time_program-specific_phase_prediction_for_python_programs>

# Communication and Specialized Training

1. Meetings every week to every 2 weeks
2. Git repository and weekly reports
3. Will discuss project progress and relevant changes, as well as how it fits into the timeline.
4. No specialized training or lab equipment is required.

# Methods or Agenda

The following research project is a continuation of an Independent Study which was conducted during the Spring 2023 semester, wherein the groundwork for RUST compilation processes and how the same is carried out via the compilers was laid down. We were able to identify the key components of RUST Compilation processes, and compare them to the compilation processes in C. Moreover, various methods were tested which were determined to have contained flaws or biases, and thus we have been able to narrow our scope to what we believe may work at the moment, but is subject to change throughout the research process.

We plan to generate the lowest form of readable and executable assembly code via the compilers for each of the languages. For example, we would generate the x86 Assembly output for a GCC-compiled program. We would then identify key points of the program in that code, such as jumps or switches to different points of the program (such as in the case of an if-else statement), and add flags over there to measure and generate a trace of the program, i.e. a full overview of how said program is executed. This process would be repeated for identical code in different languages, such that we have a trace output for C and a trace output for RUST.

Once all these metrics are at hand, we plan to use a Gaussian Mixture Model to calculate time differences between the traces of the assembly code for different languages. While the methods at this point of the research are not currently well-defined, mainly due to us figuring out how the trace would look like and what exact metrics would be collected, the main idea is following a very similar path in the research paper, “Real-Time Program-Specific Phase Change Detection for Java Programs", authored by Prof. Chiu as well. This portion of the research would be worked out well as Prof. Chiu and I continue to work on the project.

Essentially, we believe that creating an Assembly tracing tool to determine the performance of each language via the tracing of said language will vastly help in determining a crucial aspect of refactoring existing Systems code from C to RUST: if resources are actually used more wisely. We believe that such a tool may help other Systems Programmers to gain a better insight into newer languages that cater to systems level programming, as well as utilize said resources into their own projects.

#### **6. Timeline and Grading**

#### **6.1. Describe a graded assignment due by the W (withdrawal) date.**

Working Assembly tracing tool, and metrics for initial programs written in both RUST and C. Moreover, detailed reports would also be submitted for work done each week across the semester, and

#### **6.2. Describe and give dates for other first-semester assignments.**

Each Friday of each week: Detailed report with progress.

Each 2 weeks: In-person meeting with Prof. Chiu, as well as going over the reports and the Git repository with source code for the project.

October 10, 2023: First initial draft of research paper with progress on tracing tool, and code for tracing tool. This would be used as a pivot point to start discussing measuring the different trace outputs to compare between different languages, and how to create the ML model to measure the same. The paper would also be used to pitch the project to Prof. Richards

#### **6.3. Give the date by which second committee member will have been selected. Give the name, if known.**

Prof. Tim Richards will be the second Committee Chair, and I plan to discuss the project and finalize the same by 10/15/2023.

#### **6.4. Give the date by which a first draft of your literature review will be submitted to your advisor.**

September 20, 2033

#### **6.5. Give the date by which a draft of your 499T/P proposal will be submitted to your faculty sponsor prior to final submission of the proposal on PATHS. Please check the PATHS homepage for the deadline to submit your final 499T/P proposal on PATHS.**

November 30, 2023

### Criteria for Evaluation, and Planned Activities (archived, taken from Independent Study Proposal)

The main criteria for evaluation for the project would be two-fold: a paper describing the different findings and analyses of the project, as well as the tools developed for the programming languages which would help us track the Phase Changes. However, since the project involves a huge development effort across various programming languages, a log and a code repository may be used for evaluation purposes in the initial stages

The project takes place across the Spring and the Fall 2023 semester, with a final paper by Spring 2024 (if the additional time is required and necessitated). A detailed plan of the study is as follows:

1. Spring 2023: Gather relevant research work in order to choose the Benchmarks needed to be tested, and design and develop the Performance Measuring Tools (the Phase Change Detector) for RUST, and confirm if it works as expected.
2. Summer/Fall 2023: Gather data for the Phase Changes for various different RUST programs. Start development for similar tools in C/C++, and conduct the same. Criteria for evaluation remains the same.
3. Fall 2023/Spring 2024: Finalize final details of project, work on compiling findings in a proper research paper.

My expectation for the project for the upcoming Spring Semester would be to conduct it for about 4 credits, which roughly translates to about 10-12 hours per week. However, I would also take this project over the Summer, and the next Fall semester as well, with a higher credit-load over the Summer as it would be my primary project. The timeline above assumes a conservative estimate, and the actual project may be completed in a shorter timespan.