Progress Report

Contents

1 Abstract

Python is a programming language, popular due to its ease-of-use and flexibility. Unfortunately, it is a rather slow language, due to (at least with CPython, the default runtime) being interpreted. This project aims to provide large speed-ups to Python programs, by compiling them to Rust, a language known for its speed and safety. The furthest eventual goal will be to provide a compiler which can compile any Python program, with no changes to the source code, and provide a speed-up over CPython.

2 Introduction

2.1 Background

Python is a popular programming language, known for its ease of use and flexibility. However, it is also known for its speed; or rather, lack thereof.

Due to being an interpreter, CPython (the official Python implementation, and the most commonly used one) is significantly slower than compiled binaries such as those produced from languages such as C or Rust. This is due in part to the fact that the interpreter must maintain its own state in addition to that of the program being run, and also due to the fact that the compiled binaries of other languages are able to be optimised ahead of time, whereas CPython must attempt to optimise the code while it is being run.

Alternatives to CPython, such as PyPy or Cython, have been created in an attempt to improve the speed of Python programs; while these are certainly faster than CPython, they each have their own trade-offs. PyPy, for example, is a Just-In-Time (JIT) optimising interpreter, meaning that it can compile so called 'hot paths' - those which are run more frequently than others - either directly to machine code, or (more likely) to specialised instructions within the interpreter. Unfortunately, these optimisations take time to perform, and so PyPy is not suitable for short-lived programs. Cython, on the other hand, is a hybrid between Python and C, and can be used to generate C extension modules and even convert a source file directly into C code. However, Cython-generated code, by default, does not greatly optimise over CPython - as Cython aims to provide 100% compatibility with CPython, and will not infer types except in certain patterns, it is up to the programmer to explicitly declare functions and variables using either `cdef` or `cpdef`, meaning that the source code is no longer valid Python.

2.2 Aim

Rattlesnake aims to provide a compiler from Python to Rust, allowing Ahead-Of-Time (AOT) compilation and optimisations to be performed on Python code, while not requiring the programmer to learn new syntax. This is achieved

by translating Python source code directly into Rust, deferring most of the type-checking stage to the Rust compiler.

Eventually, Rattlesnake will be able to infer types from Python source code, and will allow using Python types from within the compiled Rust output. This will mean that Rattlesnake would be usable on (almost) any Python code base with no changes to the source code, and would require minimal changes to be made to the source code in most other cases.

Additionally, Rattlesnake will provide a run-time Python library poly-filling types, functions, and macros from Rust's standard library, allowing writing optimised Rust code from within Python, before translating it to Rust. This has the primary advantage of allowing REPL iteration, while still providing the speed of Rust.

2.3 Objectives

1. Create the run-time Rattlesnake library for Python, which will provide access to various standard Rust types, functions, and macros (and will act as a poly-fill for interpreted Python code).
2. Create the run-time Rattlesnake library for Rust, which will provide various standard Python types and functions.
3. Create the Rattlesnake compiler in Python, using the `ast` module to parse source code.
4. Compile the Python version of Rattlesnake with Rattlesnake, then optimise the output and maintain the Rust version.
    - Compiling Rattlesnake to Rust allows distributing it with Cargo; `cargo install rattlesnake` will install the compiler binary.
    - Optimising and maintaining the Rust version allows it to be faster and more memory-efficient than the plainly-compiled version, at the cost of having to maintain a separate grammar file (if using `pest`) or recompiling the `ast` module (if using Rattlesnake-compiled `ast`) on each major version.
        - However, maintaining the Rust version separately may allow lowering the MSRV/reducing dependence on Nightly features.
5. Create the associated documentation for the Rattlesnake compiler and run-time libraries.
6. Create these documents as part of the dissertation project.

2.4 Product Overview

2.4.1 Scope

The product will take as input Python 3 source code. It will translate that source code into Rust source code, or raise an error if the input contains unsupported features. The output Rust code will be able to be compiled into a native executable, which can be run without a Python interpreter. The output code

will depend on the library `rattlesnake`, which will be a Rust library which provides various standard Python types and functions.

There will also be a Python library `rattlesnake`, which will provide various standard Rust types, functions, and macros, and can be used by Python code in order to produce more optimised or efficient Rust code.

2.4.2 Audience

Rattlesnake is intended to be a tool for Python developers who want the speed and memory-efficiency benefits of Rust. It also serves as a rapid prototyping tool for Rust projects, and a way to simplify the distribution of Python projects as a single executable.

3 Background Review

3.1 Software Review: Interpreted Python

This section will review various software related to interpreted Python

3.1.1 PyO3

PyO3[1] is a project which provides Rust bindings for CPython. This is typically used for creating extension modules, but can also be used to run Python code from Rust at run-time.

This is not directly relevant to Rattlesnake, as it is a tool for interoperability between Rust code and a Python interpreter, but due to its similar target space, it is worth mentioning.

3.1.2 RustPython

RustPython[2] is a Python interpreter written in Rust. It is not a compiler, but rather an alternative to CPython; its biggest selling point is the ability to embed a Python script into a web page, and run it in the browser with WebAssembly. It can also be used to embed Python scripts into Rust programs, similar to PyO3.

While is isn't a compiler, it may still be useful to Rattlesnake as a source of inspiration for the object model, and potentially for implementations of some of the standard library.

3.1.3 PyPy

*Not to be confused with **PyPI**, the Python Package Index.*

[1] PyO3 Developers (2023) *PyO3: Rust bindings for the Python interpreter*. Available at: https://github.com/PyO3/pyo3 (Accessed: 18/10/2023)

[2] windelbouwman *et al.* (2023) 'RustPython'. Available at: https://rustpython.github.io/ (Accessed: 18 October 2023)

PyPy[3] is a JIT[4]-compiling Python interpreter, which is written in RPython, a restricted subset of Python which was developed alongside PyPy. PyPy's primary goal is to be a drop-in alternative to CPython, and in most cases is significantly faster at running pure-Python code.

3.2 Software Review: Compiled Python

3.2.1 Pyrex

Pyrex[5] is a Python-like language which compiles to C as a Python extension module. It was developed by Greg Ewing, and was the predecessor to Cython. It is no longer maintained, and has been superseded by Cython.

It only supported extension module targets, and as a result always depends on the CPython interpreter, rather than producing stand-alone executables.

3.2.2 Cython/Cythonize

Cython[6] and Cythonize, first released in 2007, are tools which translate Python or Cython code down to C, which can then be compiled either into a native executable or into a Python extension module. Cython is a modified dialect of Python, which changes some syntax to allow for easier translation for the tool.

While Cython and Cythonize share the same high-level goal as Rattlesnake - to make Python code faster - Cython produces code which is 100% equivalent to how CPython would run that code. This has a couple of downsides:

- Cython-generated code depends on Python's C API, which requires the Python headers to be installed, and in some cases requires a Python interpreter to be present at runtime.
- Cython-generated code contains lots of CPython calls, which can be difficult to read; this is made worse by the fact that C has no operator overloading, and so most operations require a function call.
- Because Cython produces 100% equivalent code, it is in many cases not possible to optimise the generated code for speed without changing the source Python code.

Rattlesnake, on the other hand, will produce Rust code, which is not guaranteed to be 100% equivalent to how CPython would run the code (although its behaviour will be matched on a best-effort basis). This allows for more aggressive

[3]The PyPy Team (2023) 'PyPy'. Available at: https://www.pypy.org/ (Accessed: 11 October 2023)

[4]*Just In Time*-compiling; a hybrid compiled/interpreted approach where 'hot' code which is run many times gets compiled/optimised more heavily. This is the same approach used by the Java Virtual Machine (JVM) and the Common Language Runtime (CLR) used by .NET.

[5]Ewing, G. (2010) 'Pyrex - a Language for Writing Python Extension Modules', *Pyrex*. Available at: https://www.csse.canterbury.ac.nz/greg.ewing/python/Pyrex/ (Accessed: 11 October 2023).

[6]Behnel S. *et al.* (2023) 'Cython: C-Extensions for Python'. Available at: https://cython.org/ (Accessed: 11 October 2023).

optimisations, and means that the generated code will never require a Python interpreter to be present at runtime.

For more information on Cython, see:

- https://cython.readthedocs.io/en/latest/
- https://pypi.org/project/Cython/
- https://github.com/cython/cythonnsions for Python'. Available at: https://cython.org/ (Accessed: 11 October 2023).

3.2.3 RPython

RPython[7] is an AOT-compiled (*Ahead Of Time*-compiled; the traditional compilation model where all transformations occur prior to running the code) dialect of Python. It was developed for use in PyPy[8], and is used to write the PyPy interpreter itself. It has no formal specification (its official definition[9] is 'RPython is everything that our translation toolchain can accept'!).

RPython has a number of compile targets, split into the two broad categories of 'C-like memory model' and 'object-oriented memory model'. This allows the compiler to make code tailored for its target, leaning more heavily on allocation and caching for C-like targets, and translating somewhat more directly for OO targets.

3.3 Literature Review

3.3.1 Why Python?

Python is 'the most popular language in Machine Learning' (Lunnikivi, Jylkkä and Hämäläinen, 2020), due largely to the fact that '[it] tends to be readable and concise, leading to a rapid development cycle' (Behnel *et al.*, 2011).

Due to its ease-of-use, Python is also a popular choice for scripting, and is used in many places - some prominent examples are:

- GDB, the GNU Debugger, has a Python API for writing scripts to automate debugging tasks
- The Linux kernel contains lots of Python scripts[10] for automating various tasks related to kernel development, including several GDB scripts
- The Rust compiler uses a Python script for managing the bootstrap/build processes[11]

[7]The PyPy Project (2022) 'RPython Language'. Available at: https://rpython.readthedocs.io/en/latest/rpython.html (Accessed: 11 October 2023)

[8]The PyPy Team (2023) 'PyPy'. Available at: https://www.pypy.org/ (Accessed: 11 October 2023)

[9]The PyPy Project (2022) 'RPython Language'. Available at: https://rpython.readthedocs.io/en/latest/rpython.html (Accessed: 11 October 2023)

[10]https://github.com/search?q=repo%3Atorvalds%2Flinux++language%3APython&type=code

[11]https://github.com/rust-lang/rust/blob/master/x.py

- Reddit's backend is written in Python[12]

3.3.2 Why Rust?

Rust is a popular programming language - it has been the most loved programming language in the StackOverflow Developer Survey every year since 2016 and was the 3rd most loved language in the year of its release, 2015 (Stack Overflow, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023).

Rust is 'a systems programming language meant to supersede languages like C++' (Bugden and Alahmar, 2022), designed around enforcing memory safety without the use of a garbage collector, but its data model is also well-suited to most other kinds of safety, including thread-safety and type-safety, as well as Software Fault Isolation and Information Flow Control analysis (Balasubramanian *et al.*, 2017), all without sacrificing performance. Additionally, Rust has a strong focus on ergonomics, having a powerful macro system and a standard build tool, Cargo, which also handles dependency management (from a centralised repository, crates.io) and testing.

3.3.3 What is a transpiler?

A transpiler is a specialised term for a compiler which translates between two languages which are at a comparable level of abstraction (Fenton, 2012). For example, a compiler from C to assembly would not be considered a transpiler, as assembly is a much lower-level language than C, but a compiler from C++ to C could be considered a transpiler, as both C and C++ allow for many operations to be performed in a single statement. Throughout this document, the term 'compiler' will be preferred over 'transpiler', except when referring to sources which use the term 'transpiler'.

4 Technical Progress

So far I have made enough progress on the project to be able to compile some (admittedly-simple) Python programs into Rust, and have them run successfully.

I have also made a bit of progress on the Python library, with several macros implemented. The full list of implemented features can always be seen in the features document.

Additionally, a little bit of work has been implemented on the Rust side, with some standard Python functions such as `str()` and `bool()` having been implemented.

[12] https://support.reddithelp.com/hc/en-us/articles/204536739-What-is-Reddit-written-in-

5 Professional Issues and Risk

5.1 Risk

This project does not carry any associated risk, as it does not by itself handle any sensitive data, and therefore cannot have any of the associated risks of doing so. Applications transformed by Rattlesnake could potentially have these risks, but that is not the responsibility of Rattlesnake.

5.2 Professional Issues

This project could foreseeably encounter problems with regards to licensing, specifically licence compatibility issues. This is primarily due to my choice to dual-license the compiler with 'GPL-3 or later' and '(custom) business licence' as the two options for licensing the compiler. The result of this is that transformed code would have to comply with both the licence of the original code, and the licence used for the compiler. This could be a problem if the original code is licensed under a licence which is incompatible with the GPL, such as the Apache Licence 1.0 or Open Watcom Licence.

6 Glossary

- AOT (adj.): Ahead-of-Time
  - A compilation model where all transformations occur prior to running the code. This is the traditional compilation model, used by languages such as C or Rust, and is the alternative to JIT.
- AST (n., acr.): Abstract Syntax Tree
  - A representation of source code which is easy to manipulate. It is often used as an intermediate representation of code in compilers as it represents the ideas written in the code without the complicating details such as whitespace.
    For example, the following Python code:
    ```
    def foo(bar):
        return bar + 1
    ```
    Could be represented with the following AST (in Lisp syntax):
    ```
    (function-definition
        (name "foo")
        (parameters
            (name "bar"))
        (body
            (return-statement
                (+ (name "bar")
                    (constant 1)))))
    ```
- Cargo, `cargo` (n.): The standard Rust package manager and build tool
  - Cargo is the standard tool used to generate, build, and test Rust projects. It is also used to manage dependencies, and is the primary way of distributing Rust projects.
- Compiler (n.): A program which takes as input code in one form, and returns as output code in another form.
  - Typically, this is translating code from a high-level language into a lower-level language - such as transforming C or Rust into machine code - but it can also be translating code between two high-level languages - such as transforming TypeScript into JavaScript, or (in this case) transforming Python into Rust.
  - See also: transpiler.
- CPython (n.): The official Python interpreter, written in C
- crates.io (n.): The official Rust package registry
  - https://crates.io/
  - Rust projects are distributed as crates, which can published to crates.io. The `cargo` tool can then be used to install these crates, either globally (installing their associated binaries) or locally (as a dependency of the current project).
  - See also: docs.rs, PyPI.
- docs.rs (n.): The official Rust documentation host
  - https://docs.rs/
  - All crates published to crates.io have their documentation built auto-

matically, and hosted on docs.rs. This allows easy access to documentation for Rust projects, without having to install the project locally, or needing to visit arbitrary third-party websites. Additionally, Rust's standard library documentation is also hosted on docs.rs, meaning that it is the single point of reference for all Rust documentation.
  - See also: crates.io.
- JIT (adj.): Just-in-Time
  - A compilation model where transformations and optimisations occur during the running of the code. Popularised by Self and Java, this is the compilation model used by most modern JavaScript engines, and is the alternative to AOT.
- Monkey-patch (n.): The result of monkey-patching
- Monkey-patch (v.): To modify a global namespace, typically to create additional global functions or data
  - This is generally considered bad practice in most languages, as doing so can lead to conflicts between different libraries which both monkey-patch the same namespace.
  - It is often used in languages like JavaScript, where poly-fills use monkey-patching to (conditionally) update the global namespace to add support for newer features.
- MSRV (n., acr.): Minimum Supported Rust Version
  - The oldest version of the Rust compiler supported by a project
- Poly-fill (n.): Code which poly-fills a specific feature
- Poly-fill (v.): To provide an implementation of a feature which is not natively supported by (this version of) the platform
  - The most common occurrence of this is in web development, where poly-fills can be used to provide support for newer features and APIs when running in older browsers.
  - Poly-fills refer specifically to monkey-patching the namespace being poly-filled, rather than providing a separate namespace for the feature, which will be implemented either by delegating to the native implementation (if available) or by providing a custom implementation (if not) - this alternative method is known as a *pony-fill*.
- PyPI (n., abbr.): Python Package Index
  - https://pypi.org/
  - The official Python package registry. It is used by the `pip` tool, and all related tools, to install Python packages.
  - See also: crates.io.
- RAD (n., acr.): Rapid Application Development
  - A software development methodology which focuses on rapid prototyping and iteration.
- Transpiler (n.): A term to refer to a compiler which translates specifically from a high-level language to another high-level language
  - See also: compiler.

Bibliography

Balasubramanian, A. *et al.* (2017) 'System Programming in Rust: Beyond Safety', *Proceedings of the 16th Workshop on Hot Topics in Operating Systems* [Preprint]. Available at: https://doi.org/10.1145/3102980.3103006.

Behnel, S. *et al.* (2011) 'Cython: The Best of Both Worlds', *Computing in Science & Engineering*, 13(2), pp. 31–39. Available at: https://doi.org/10.1109/MCSE.2010.118.

Behnel, S. *et al.* (2023) 'Cython: C-Extensions for Python'. Available at: https://cython.org/ (Accessed: 11 October 2023).

Bugden, W. and Alahmar, A. (2022) 'Rust: The Programming Language for Safety and Performance'. Available at: https://arxiv.org/abs/2206.05503.

dragostis *et al.* (2023) 'pest. The Elegant Parser'. Available at: https://pest.rs/ (Accessed: 18 October 2023).

Ewing, G. (2010) 'Pyrex - a Language for Writing Python Extension Modules', *Pyrex*. Available at: https://www.csse.canterbury.ac.nz/greg.ewing/python/Pyrex/ (Accessed: 11 October 2023).

Fenton, S. (2012) *Compiling vs Transpiling*. Available at: https://www.stevefenton.co.uk/blog/2012/11/compiling-vs-transpiling/.

Lunnikivi, H., Jylkkä, K. and Hämäläinen, T. (2020) 'Transpiling Python to Rust for Optimized Performance', in A. Orailoglu, M. Jung, and M. Reichenbach (eds) *Embedded Computer Systems: Architectures, Modeling, and Simulation*. Cham: Springer International Publishing, pp. 127–138.

PyO3 Developers (2023) 'PyO3: Rust bindings for the Python interpreter', *GitHub*. Available at: https://github.com/PyO3/pyo3 (Accessed: 18 October 2023).

Stack Overflow (2015) 'Stack Overflow Developer Survey 2015'. Available at: https://insights.stackoverflow.com/survey/2015#tech-super.

Stack Overflow (2016) 'Stack Overflow Developer Survey 2016'. Available at: https://insights.stackoverflow.com/survey/2016#technology-most-loved-dreaded-and-wanted.

Stack Overflow (2017) 'Stack Overflow Developer Survey 2017'. Available at: https://insights.stackoverflow.com/survey/2017#most-loved-dreaded-and-wanted.

Stack Overflow (2018) 'Stack Overflow Developer Survey 2018'. Available at: https://insights.stackoverflow.com/survey/2018#most-loved-dreaded-and-wanted.

Stack Overflow (2019) 'Stack Overflow Developer Survey 2019'. Available at: https://insights.stackoverflow.com/survey/2019#most-loved-dreaded-and-wanted.

Stack Overflow (2020) 'Stack Overflow Developer Survey 2020'. Available at: https://insights.stackoverflow.com/survey/2020#most-loved-dreaded-and-wanted.

Stack Overflow (2021) 'Stack Overflow Developer Survey 2021'. Available at: https://insights.stackoverflow.com/survey/2021#technology-most-loved-dreaded-and-wanted.

Stack Overflow (2022) 'Stack Overflow Developer Survey 2022'. Available at: https://survey.stackoverflow.co/2022/#section-most-loved-dreaded-and-wanted-programming-scripting-and-markup-languages.

Stack Overflow (2023) 'Stack Overflow Developer Survey 2023'. Available at: https://survey.stackoverflow.co/2023/#section-admired-and-desired-programming-scripting-and-markup-languages.

The PyPy Project (2022) 'RPython Language'. Available at: https://rpython.readthedocs.io/en/latestrpython.html/ (Accessed: 11 October 2023).

The PyPy Team (2023) 'PyPy'. Available at: https://www.pypy.org/ (Accessed: 11 October 2023).

windelbouwman *et al.* (2023) 'RustPython'. Available at: https://rustpython.github.io/ (Accessed: 18 October 2023).