LPython: Interactive LLVM-based Python Compiler for Modern Architectures

Ondrej Certik*
GSI Technology, USA
*ondrej@certik.us

Brian Beckman
GSI Technology, USA

Naman Gera
Indian Institute of Information Technology

Smit Lunagariya
IIT BHU

Gagandeep Singh
GSI Technology, USA

Rohit Goswami
Science Institute, University of Iceland
Quansight Austin, TX, USA

Thirumalai Shaktivel
K S Institute of Technology

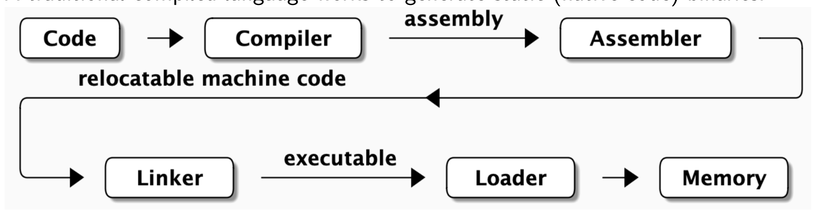
Dylon Edwards
GSI Technology, USA

Introduction

Python as a general purpose language has many strengths, stemming from duck-typing and a rich ecosystem. However, for performance critical applications such as High Performance Computing (HPC) or any other kind of numerical computing the standard `CPython` implementation is often not fast enough.
Python supports optional type hints, which, when used bring the semantics and syntax closer to that of strongly typed languages like `C++`.

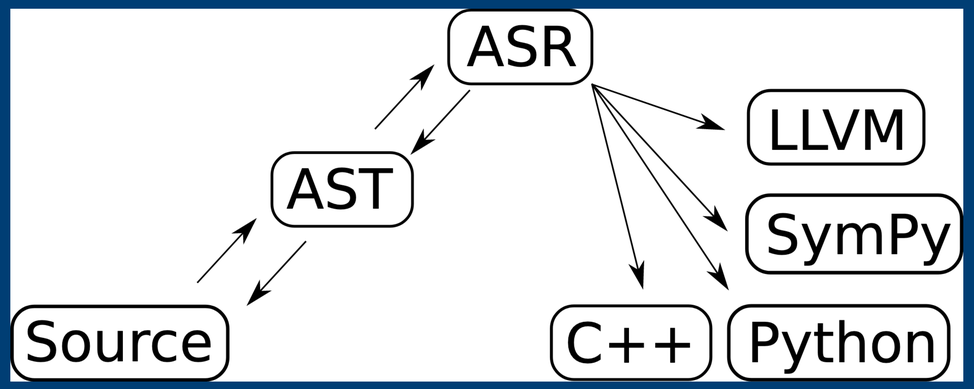
Design Principles

A traditional compiled language works to generate static (native-code) binaries:



Interpreted languages differ in their generation of bytecode for immediate execution. LPython works as an ahead of time compiler for Python with an abstract semantic representation shared with LFortran.

LPython is modern **LLVM based** compiler for a **typed subset of Python** can provide speed, portability and interactivity.



† Designed as a library with separate building blocks (parser, AST, ASR, semantic phase, codegen)

† Part of LCompilers with LFortran.

† Can transpile to other compatible languages.

† The Abstract Syntax Tree (AST) and the intermediate Abstract Semantic Representation (ASR) is represented using the ASDL domain-specific language, just like CPython's AST.

Take a picture to read more about the project

- LPython is written in C++ and it has multiple backends to generate code including LLVM and C++.
- Uses the same internal representation (ASR) as in LFortran, and both the LPython and LFortran frontends are effectively surface languages that share the same middle end and backends, as well as high and low level optimizations.
- The speed of LPython comes from high level optimizations at the ASR level, as well as the low level optimizations that LLVM can do.

Active Areas

- Webassembly conversions from the ASR
- Support for generic types
- Support for the Python standard library
- Interactive prompts
- `re2c` and `GNU Bison` for the parser generator.
- Language server protocol (LSP)

Project Goals

- The best possible performance for numerical array oriented code
- Run on all platforms
- Compile a subset of Python and be Python compatible
- Fast compilation
- Excellent user friendly diagnostic messages: error, warnings, hints, notes, etc.
- Ahead of time compilation to binaries and interactive usage (Jupyter notebook)
- Able to transform the Python code to C++, Fortran and other languages

The ASR sets LPython apart from other statically compiled Python projects like `nutika`, `pythran`, `Numba` etc. At the ASR level, it is possible to provide multiple passes which convert the ASR into more optimized variants of the ASR itself, before being passed over to the LLVM backend. Additionally, for numerical methods in particular, it is possible to provide type checking and more robust support for typing by using LPython.

References

[1] A. V. Aho and A. V. Aho, Eds., *Compilers: Principles, Techniques, & Tools*, 2nd ed. Boston: Pearson/Addison Wesley, 2007, 1009 pp.