Python Interpreter Performance Deconstructed

Gergö Barany, Technische Universität Wien

The Python programming language is known for performing poorly on many tasks. While to some extent this is to be expected from a dynamic language, it is not clear how much each dynamic feature contributes to the costs of interpreting Python. In this study we attempt to quantify the costs of language features such as dynamic typing, reference counting for memory management, boxing of numbers, and late binding of function calls. We use an experimental compilation framework for Python that can make use of type annotations provided by the user to specialize the program as well as elide unnecessary reference counting operations and function lookups. The compiled programs run within the Python interpreter and use its internal API to implement language semantics. By separately enabling and disabling compiler optimizations, we can thus measure how much each language feature contributes to total execution time in the interpreter. We find that a boxed representation of numbers as heap objects is the single most costly language feature on numeric codes, accounting for up to 43 % of total execution time in our benchmark set. On symbolic object-oriented code, late binding of function and method calls costs up to 30 %. Redundant reference counting, dynamic type checks, and Python's elaborate function calling convention have comparatively smaller costs.