PYTHON IS SLOW

Pasyuta Maxim

Student, Altai State Technical University. I.I. after Polzunov, Russia, Barnaul

Petrushova Natalia

senior lecturer, Altai State Technical University. I.I. after Polzunov, Russia, Barnaul

Python is a universal programming language using which you can create projects for any platform. You can build web applications, do data analysis, automate system management tasks and more. This article discusses the implementation of the CPython interpreter [1]. According to many developers Python projects work many times slower than similar software solutions in other languages. Let's try to figure it out and consider the main features of the Python programming language which can affect the speed of the program:

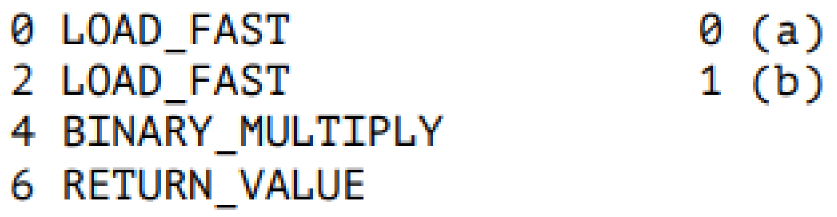
• GIL thread lock operation (Global Interpreter Lock, global interpreter lock) [1].

• Python is an interpretive rather than a compiled language [3].

• Python is a language with dynamic typing [2].

First, let's talk about what is GIL (Global Interpreter Lock). GIL is closely related to stream functioning. Modern computers have multi-core processors, and sometimes even are multi-processor systems. To use all of this computing power the operating system uses low-level structures called threads and processes (MATLAB program process) that can run multiple threads and use them appropriately. As a result, for example, if a particular process requires a lot of processor resources, its execution can be distributed across several cores. Thanks to this method, most applications increase productivity when running a program. Locks are used for multi-threaded applications. The key is that they allow such a system to protect memory while simultaneously accessing two streams (read or write) to the same cell. In Python, it is GIL that is responsible for protecting the memory of the interpreter from destruction. It implements this task by dividing all memory operations into atomic ones.

Python is an interpreted programming language. You often hear that the poor performance of Python is related to the peculiarity of building projects. Such statements are based on a gross simplification of how CPython actually works. If you run any program with the extension .py, CPython will begin a long sequence of actions which consists in reading, lexical analysis, parsing, compiling, interpreting and executing script code. As an example, consider the simplest multiplication function which will take 2 numbers as parameters and return their product. A function, like everything else in Python, is an object, and by importing it, we can list its methods using the dir () function. One of the attributes of the function is __code__, which contains a code object - an object that wraps the written code. This object has a co_code method that returns the bytecode generated by Python for this function (i.e. instructions for the Python interpreter). To bring the bytecode into readable form (i.e., to disassemble), the dis.dis () function from the dis package is used.



```
0 LOAD_FAST                 0 (a)
2 LOAD_FAST                 1 (b)
4 BINARY_MULTIPLY
6 RETURN_VALUE
```

Figure 1. Executing the dis.dis (multiply) command

This is applied not only to the scripts that we write, but also to the imported code of third-party modules. As a result, most of the time (if you do not write code that runs only once), Python will execute the finished bytecode. In figure -1 we see that the function is converted into a cycle of four instructions. Each of these instructions contains a specific action. The first is LOAD_FAST which modifies the contents of a variable in memory and pushes it on the stack. When the ‘a’ variable is pushed onto the stack, we switch to a similar command for the ‘b’ variable. The following statement, BINARY_MULTIPLY, tells the Python interpreter to take the top two values on the stack and multiply them. Please note that we can pass a value of any type to a function: an integer, a real number or a string, this bytecode does not change this. Object typing begins when Python executes an instruction. In this case, when the BINARY_MULTIPLY statement is executed, Python looks at the type of the object and, for example, if it is two integers, multiplies two integers. Next, the result of the multiplication is placed on the top of the stack. And the last statement is RETURN_VALUE which returns a variable from the top of the stack. This applies not only to the scripts that we write, but also to the imported code of third-party modules. As a result, most of the time (if you don’t write code that runs only once), Python will execute the finished bytecode.

Python is a dynamically typed programming language. Unlike static typed languages, Python does not need to declare a variable type when creating it. Languages with dynamic typing include Objective-C, Ruby, PHP, Perl, JavaScript. In these programming languages the concept of data types has the same meaning as in classical C, but the type of the variable is dynamic. This mechanism is implemented as follows. The types of variables are unknown until they have specific values at startup. Type checking and conversion are complex operations. Each time a variable is accessed, read, or written, a type check is performed. Because of this the speed of applications suffers significantly.

In conclusion I want to note that the reason for the poor performance of Python is its dynamic component, as well as its versatility. Initially this programming language was conceived by the mathematician Guido van Rossumas a simple and expressive language. The creator of Python wanted to introduce to the world a universal tool that avoids complex coding structures. He wanted to implement a programming language in which the program code would be read like regular English.

Список литературы:

1. Как устроен GIL в Python [Электронный ресурс]. – Электрон. дан. – Заглавие с экрана. – Режим доступа: URL: https://habr.com/ru/post/84629/ (дата обращения: 15.03.2020).

2. Статическая и динамическая типизация [Электронный ресурс]. – Электрон. дан. – Заглавие с экрана. – Режим доступа: URL: https://habr.com/ru/post/308484/ (дата обращения: 15.03.2020).

3. Основные принципы программирования: компилируемые и интерпретируемые языки [Электронный ресурс]. – Электрон. дан. – Заглавие с экрана. – Режим доступа: URL: https://tproger.ru/translations/programming-concepts-compilation-vs-interpretation/ (дата обращения: 15.03.2020).