Нагорных М. Э., Твардовский Г.В., Чернышёв С. А. Ускорение кода на Python // Международный журнал информационных технологий и энергоэффективности. – 2021 – Т. 6 № 3(21) с. 24-30

Международный журнал информационных технологий и энергоэффективности

Сайт журнала:
http://www.openaccessscience.ru/index.php/ijcse/

УДК 004.043:004.051

УСКОРЕНИЕ КОДА НА PYTHON

[1] Нагорных М. Э., [2] Твардовский Г.В., [3,4] Чернышёв С. А.

[1,2]Санкт-Петербургский государственный университет аэрокосмического приборостроения, Россия (190000, г. Санкт-Петербург, ул. Большая Морская, 67, лит. А),
e-mail: [1] nagornykh_max@mail.ru, [2] gtvardovsky@gmail.com
[3] Санкт-Петербургский государственный университет промышленных технологий и дизайна, Россия (191186, г. Санкт-Петербург, ул. Большая Морская, 18)
[4] Санкт-Петербургский государственный экономический университет, Россия (191023, г. Санкт-Петербург, ул. Садовая, 21), e-mail: chernyshev.s.a@bk.ru

В настоящее время Python является одним из наиболее популярных языков программирования. Несмотря на множество его преимуществ, у этого языка программирования есть существенный минус - его быстродействие. В данной статье анализируются и сравниваются различные способы решения обозначенной проблемы, такие как: использование дополнительных программных модулей, использование различных интерпретаторов и выполнение оптимизации кода. Приводится разбор каждого метода, а также продемонстрированы результаты, отражающие эффективность каждого из них.

Ключевые слова: Python, Cython, Psyco, интерпретатор, PyPy, Numba, профилирование.

ACCELERATING CODE ON PYTHON

[1]Nagornykh M.E., [2] Tvardovsky G.V., [3,4] Chernyshev S.A.

[1,2] Saint Petersburg State University of Aerospace Instrumentation, Russia (190000, Saint Petersburg, Bolshaya Morskaya st., 67, letter A),
e-mail: [1] nagornykh_max@mail.ru, [2] alexey_bykovoff@mail.ru
[3] Saint Petersburg State University of Industrial Technology and Design, Russia (191186, Saint-Petersburg, Bolshaya Morskaya st., 18)
[4] Saint Petersburg State University of Economics, Russia (191023, Saint Petersburg, Sadovaya st., 21), e-mail: chernyshev.s.a@bk.ru

Python is currently one of the most popular programming languages. Despite its many advantages, this programming language has a significant disadvantage - its performance. This article analyzes and compares various ways to solve the indicated problem, such as: using additional software modules, using different interpreters and performing code optimization. An analysis of each method is given, and the results are shown that reflect the effectiveness of each of them.

Keywords: Python, Cython, Psyco, interpreter, PyPy, Numba, profiling.

Введение

На сегодняшний день Python - самый популярный и быстроразвивающийся язык программирования (если опираться на исследование StackOverflow [1]). Разработчики его

выбирают за легкий порог вхождения, читаемость и простоту кода, а также за огромное количество постоянно обновляющихся библиотек. Код, выполняющий один и тот же функционал, написанный на языке Python, выглядит намного читаемее и компактнее, чем код на Си, Java или других языках программирования.

Например, рассмотрим простейшую программу, которая выводит строку на экран.

“Си” справится в 4 строки:

```
#include <stdio.h>
int main(int argc, char **argv) {
    printf(“Hello, world!”)
}
```

“Java” выполнит эту же программу в 5 строк, но код будет более сложный:

```
class HelloWorld {
    public static void main(String[] args) {
        System.out.println(“Hello, world!”);
    }
}
```

А вот листинг кода на Python:

```
print(‘Hello, world!’)
```

На основе приведенных примеров, можно сделать вывод, что код на языке Python более компактный и понятный, чем на других языках программирования и эта разница будет увеличиваться в разы с увеличением сложности, а также объема кода.

Однако, у языка программирования Python имеется один существенный недостаток по сравнению с другими языками высокого уровня - он имеет низкую скорость работы. Полностью избавиться от данного недостатка невозможно, но существуют различные способы ускорения работы кода [2]. В данной статье рассматриваются следующие из них: использование дополнительных программных модулей, интерпретаторов и нахождение оптимального алгоритма для решения той или иной задачи.

1. Использование дополнительных программных модулей

Существуют различные программные решения для ускорения кода на Python. Самые популярные из них: Psyco и Cython.

1.1. Cython

Cython является промежуточным слоем между Python и языком C/C++. Он позволяет писать Python-код с незначительными модификациями, который затем транслируется в C-код [3]. Для этого необходимо лишь добавить тип каждой переменной. Например, для того чтобы присвоить переменной “y” числовое значение “0,13”, необходимо явно указать ее тип: “cdef float y = 0.13”.

Это необходимо, потому что при использовании обычного Python-кода, типы переменных определяются динамически, а для языка С необходимо явное указание типа переменной.

При работе с функциями, Cython также дает возможность объявлять их по-другому для ускорения кода. Вместо классического “def” имеется возможность объявлять функции с помощью “cdef” и ”cpdef”. “cdef” - функция, которую можно вызывать только в пределах Cython-кода. cpdef-функция дает возможность вызова не только в пределах Python-кода, а также и из С-кода.

Пример листинга кода функций с использованием Cython и без:

```
def test(x):
    y = 1
    for i in range(1, x+1):
        y *= i
    return y
```

```
cpdef int test(int x):
    cdef int y = 1
    cdef int i
    for i in range(1, x+1):
        y *= i
    return y
```

Анализ эффективности Cython продемонстрирован в Таблице 1. Как из нее видно, Cython дает возможность значительно ускорить код на Python, не прикладывая особых усилий. Результат будет зависеть от количества циклов в коде программы, а также от количества обрабатываемых данных.

1.2. Psyco

Альтернативным вариантом решения является использование модуля Psyco. Работа этого модуля основывается на его внедрение в интерпретатор Python, где он выборочно заменяет части интерпретируемого байт-кода Python оптимизированным машинным кодом.

Psyco работает исключительно во время исполнения Python. Пока интерпретатор Python выполняет приложение, Psyco делает проверки, для попытки замены операции байт-кода Python отработанным машинным кодом [4].

Применять Psyco можно как для всего приложения, тогда модуль будет в автоматическом режиме производить вставки “по месту”. Еще имеется возможность указывать в параметрах модуля определенные методы и классы, с которыми нужно работать.

Однако, Psyco требует тестирования. Он полезен для обработки многократно выполняющихся циклов, и операций, в которых задействованы целые числа и числа с плавающей запятой. Для нециклических функций и для операций над другими типами объектов, Psyco обычно добавляет накладные расходы на анализ и внутреннюю компиляцию [4].

Для программ с большим числом методов и классов, включение Psyco для всего приложения является дополнительной нагрузкой. Кроме того, имеется существенный

недостаток - поддерживается только 32-битная версия Python версии не выше 2.6. Разработчики Psyco планировали выпуск второй версии модуля, но его выход так и не состоялся. Вместо этого, проект перерос в создание интерпретатора PyPy.

2. Использование различных интерпретаторов Python

2.1. PyPy

PyPy — это совместимый интерпретатор Python, с помощью которого можно получить заметное улучшение скорости. В PyPy встроен трассирующий JIT-компилятор, которой может превращать код на языке Python в машинный код прямо во время выполнения программы [5].

Оценка скорости работы PyPy приводится в Таблице 1, по ней видно, что ускорение кода происходит примерно в 50 раз. PyPy – это быстрая и эффективная альтернатива CPython. Запустив скрипт с его помощью, можно получить значительное улучшение скорости, не внося ни одного изменения в код.

Однако, следует отметить некоторые недостатки PyPy, такие как: обработка кода заново при мельчайших изменениях, поддержка не все библиотек Python, проблемы с jit-компилятором, а точнее с его уровнями абстракций, что в свою очередь влечет появление непредвиденных конфликтов и усложнение процесса поиска ошибок.

2.2. Numba

Numba — это JIT-компилятор с открытым исходным кодом. Суть его работы заключается в переводе функции Python в оптимизированный машинный код во время выполнения программы. Для этого используется стандартные библиотеки компилятора LLVM. Скомпилированные с помощью Numba алгоритмы могут приближаться к скорости выполнения программ на C или FORTRAN. Также Numba позволяет распараллеливать выполняемые задачи, что дает прирост в скорости работы [6].

Кроме того, Numba не требует замены интерпретатора Python или отдельного запуска. Для её использования необходимо применить один из декораторов Numba к функции.

Пример использования Numba продемонстрирован ниже:

```
from numba import jit
import random

@jit(nopython=True)
def numba_test(a):
    count = 0
    for i in range(a):
        x = random.random()
        y = random.random()
        if (x ** 2 + y ** 2) < 1.0:
            count += 1
    return count
```

Анализ эффективности использования данного программного модуля и сравнение его с другими происходит в Таблице 1. Из него видно, что, используя Numba достигается увеличение скорости выполнения программы примерно в 50 раз.

Подводя итоги анализа различных программных решений, стоит отметить, что каждый из модулей показывает скорость работы превосходящую скорость выполнения без их использования. Также следует учитывать, что их работа будет особенно сильно заметна при наличии большого количества циклов, математических вычислений и объема данных.

Для сравнения скорости работы с использованием дополнительных программных модулей и без, использовалась следующая функция:

```
total = 0
for i in range(1, 10000):
    for j in range(1, 10000):
        total += i + j
```

Полученные результаты сравнения представлены в Таблице 1.

Таблица 1 - Сравнение скорости выполнения кода с использованием различных программных модулей и без.

	Без использования дополнительных программных модулей	CPython	PyPy	Numba
Среднее время выполнения, сек	10	0.5	0.2	0.2

3. Оптимизация кода

Другой рассматриваемый метод ускорения кода на Python - его оптимизация.

3.1. Вынос кода программы в отдельные методы

Данный шаг помогает интерпретатору python проводить оптимизацию кода быстрее. Если изначальный коды выглядит так:

```
for Y in range(height):
    for X in range(width):
        #code here
```

то, изменив его на:

```
def my_func(height, width):
    for Y in xrange(height):
        for X in xrange(width):
            #code here
my_func(height, width)
```

время выполнения кода, согласно проверке через модуль time, сократится на 2.5 секунды, то есть 79%.

3.2. Профилирование

Стандартная библиотека Python'a, содержит модуль cProfile, который упрощает профилирование кода [7].

Модуль содержит множество команд и готовых решений, в статье рассматриваются лишь некоторые. Так, например, ключ интерпретатора -m позволяет запускать модули как отдельные программы, если сам модуль предоставляет такую возможность.

Пример:

```
python -m cProfile test.py
```

Результатом этой команды будет получение информации о количестве вызовах функции, время работы всех ее вызовов и каждого отдельного соответственно. С помощью этой информации можно определить места требующие оптимизации.

Также, добавив ключ “-s time”, можно отсортировать вывод профилировщика по времени выполнения.

3.3. Математика

По возможности необходимо преобразовывать сложные математические операции в более простые. Например, есть смысл заменить возведение в степень на умножение, если используется возведение в степень двойки. Такой простой шаг позволяет уменьшить временные затраты на выполнение кода примерно на 60%. Например, функция:

```
def power():
    a = 99999
    return a**2
```

выполняется в среднем за 0.002 мс, а за это же время, заменив возведение в квадрат на умножение, время выполнения функции получается равным в среднем 0.0012 мс, что может дать существенный прирост производительности, в случае большого количества подобных вычислений и усложнении операндов.

3.4. Особенности Python и решаемой задачи

При оптимизации кода следует учитывать особенности решаемой задачи - в некоторых случаях нет нужды выполнять вычисления до конца.

К общим методам оптимизации можно отнести отказ от использования глобальных переменных и глобальных объектов. Например, если в функции используется модуль math, импорт его функций можно производить в самой функции.

Пример листинга кода:

```
def my_func(height, width):
    from math import atan2, cos, sqrt
    pix = img.load()
```

Также, при написании кода, следует учитывать типизацию. Например, функция abs вернет число типа float. Поэтому, для ускорения кода, его надо сравнивать с float, а не int.

Пример оптимизации кода:

```
if abs(X) > 2:  --> if abs(X) > 2.0:
```

Заключение.

Подводя итоги, следует отметить, что в данной статье рассмотрены не все модули для оптимизации кода, а также алгоритмические решения. Однако, общий принцип работы модулей схож, а некоторые алгоритмические решения невозможно продемонстрировать без конкретных примеров. Применение модулей преимущественно носит характер оптимизации работы циклов и структур обрабатывающих большие объемы данных.

Анализируя примеры применения рассмотренных способов оптимизации кода, можно сделать вывод о том, что при правильном подходе можно добиться уменьшения времени работы программы более 80%.

Список литературы

1. Опрос разработчиков Stack Overflow. – 2019 / [Электронный ресурс]. – Режим доступа: URL: https://insights.stackoverflow.com/survey/2019.
2. Mark Lutz. Learning Python. — O'Reilly Media, 2009
3. Philipp Herron. Learning Cython Programming. — Packt, 2013
4. Psyco.Официальный сайт. – 2010 / [Электронный ресурс]. – Режим доступа: URL: http://psyco.sourceforge.net/.
5. PyPy.Официальный сайт. – 2021 / [Электронный ресурс]. – Режим доступа: URL: https://www.pypy.org/.
6. Numba. Официальный сайт. – 2021 / [Электронный ресурс]. – Режим доступа: URL: http://numba.pydata.org/.
7. Бизли Д. Python. Подробный справочник. — СПб.: Символ-Плюс, 2010

References

1. Stack Overflow Developer Poll. Available at: https://insights.stackoverflow.com/survey/2019.
2. Mark Lutz. Learning Python. — O'Reilly Media, 2009.
3. Philipp Herron. Learning Cython Programming. — Packt, 2013
4. Psyco. Official site. Available at: http://psyco.sourceforge.net/.
5. PyPy. Official site. Available at: https://www.pypy.org/.
6. Numba. Official site. Available at: http://numba.pydata.org/.
7. Beasley D. Python. Detailed reference. - Symbol-Plus, 2010