Бесплатная электронная книга

УЧУСЬ cython

Free unaffiliated eBook created from
Stack Overflow contributors.

#cython

1

1: cython 2

2

? 2

? 2

? 2

Examples 2

Cython 2

1: Cython 2

3

Ubuntu, Debian 3

Windows 3

2. C 3

Ubuntu, Debian 4

MAC 4

Windows 4

, 4

4

hello.pyx 4

test.py 5

setup.py 5

5

2: Cython 6

Examples 6

Cython pyinstaller 6

(Windows) 6

Numpy 7

3: C 8

Examples 8

C 8

8

test_extern.pxd 8
test_extern.pyx 8
4: C ++ 10
Examples 10
DLL: C ++ Cython Python 10
C ++ DLL : complexFunLib.h complexFunLib.cpp 10
Cython : ccomplexFunLib.pxd complexFunLib.pyx 11
Python : setup.py run.py 13
..... 15

Около

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: cython

It is an unofficial and free cython ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official cython.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

глава 1: Начало работы с cython

замечания

Что такое Китон?

Язык программирования Cython обогащает Python C-подобным статическим набором текста, возможностью прямого вызова функций C и несколькими другими функциями. Это позволяет достичь уровня C-уровня при использовании синтаксиса, подобного Python.

Как это работает?

Код Cython скомпилирован с использованием компилятора cython source-to-source для создания кода C или C ++, который, в свою очередь, может быть скомпилирован с использованием компилятора C. Это позволяет создавать расширения, которые могут быть импортированы из Python или исполняемых файлов.

Основной прирост производительности Cython может достигнуть, в отличие от чистых основ Python, минуя API CPython. Например, при добавлении двух целых чисел Python выполняет проверку типа для каждой переменной, находит функцию добавления, которая удовлетворяет найденным типам и вызывает эту функцию. В C-образном C-образном коде типы уже известны, и выполняется только один вызов функции. Следовательно, Китон особенно близок к математическим проблемам, в которых типы понятны.

Как использовать его для ускорения моего кода?

Обычный пример использования при попытке ускорить работу программы с помощью Cython заключается в том, чтобы профилировать код и перемещать вычислительно дорогие детали в компилируемые модули Cython. Это позволяет сохранить синтаксис Python для большей части кода и применить ускорение там, где это наиболее необходимо.

Examples

Установка Cython

Для использования Cython необходимы две вещи. Сам пакет Cython, который содержит `cython` source-to-source и Cython, взаимодействует с несколькими библиотеками C и Python (например, numpy). Чтобы скомпилировать код C, сгенерированный компилятором `cython` , необходим компилятор C.

Шаг 1: Установка Cython

Система Агностик

Cython можно установить с несколькими системными системами управления пакетами. Они включают:

1. PyPI через pip или easy_install:

```
$ pip install cython
$ easy_install cython
```

2. anaconda используя conda:

```
$ conda install cython
```

3. Enthought canopy с помощью менеджера пакетов enpkg:

```
$ enpkg cython
```

Также исходный код можно загрузить из github и установить вручную, используя:

```
$ python setup.py install
```

Ubuntu, Debian

Для Ubuntu `cython` пакеты `cython` и `cython3` . Обратите внимание, что они предоставляют более старую версию, чем упомянутые выше параметры установки.

```
$ apt-get install cython cython3
```

Windows

Для Windows файл .whl, который может быть установлен с помощью pip, предоставляется третьей стороной. Подробные сведения об установке файла .whl в Windows можно найти здесь .

Шаг 2. Установка компилятора C

Для компиляции файлов C, созданных Cython, необходим компилятор для C и C ++.

Компилятор gcc рекомендуется и может быть установлен следующим образом.

Ubuntu, Debian

`build-essential` пакет содержит все необходимое. Он может быть установлен из репозиториев, используя:

```
$ sudo apt-get install build-essential
```

MAC

Инструменты разработчика XCode содержат компилятор gcc.

Windows

MinGW (Minimalist GNU для Windows) содержит версию gcc для Windows. Также можно использовать компилятор из Visual Studio.

Привет, мир

Файл Cython pyx необходимо перевести на C-код (*cythonized*) и скомпилировать, прежде чем он будет использоваться с Python. Общий подход заключается в создании модуля расширения, который затем импортируется в программу Python.

Код

В этом примере мы создаем три файла:

- `hello.pyx` содержит код Cython.
- `test.py` - это скрипт Python, который использует расширение hello.
- `setup.py` используется для компиляции кода Cython.

hello.pyx

```
from libc.math cimport pow

cdef double square_and_add (double x):
    """Compute x^2 + x as double.

    This is a cdef function that can be called from within
    a Cython program, but not from Python.
    """
    return pow(x, 2.0) + x

cpdef print_result (double x):
```

```
    """This is a cpdef function that can be called from Python."""
    print("({} ^ 2) + {} = {}".format(x, x, square_and_add(x)))
```

test.py

```
# Import the extension module hello.
import hello

# Call the print_result method
hello.print_result(23.0)
```

setup.py

```
from distutils.core import Extension, setup
from Cython.Build import cythonize

# define an extension that will be cythonized and compiled
ext = Extension(name="hello", sources=["hello.pyx"])
setup(ext_modules=cythonize(ext))
```

составление

Это можно сделать, используя `cython hello.pyx` чтобы перевести код на C, а затем скомпилировать его с помощью `gcc` . Более простой способ - позволить distutils справиться с этим:

```
$ ls
hello.pyx  setup.py  test.py
$ python setup.py build_ext --inplace
$ ls
build  hello.c  hello.cpython-34m.so  hello.pyx  setup.py  test.py
```

Файл общего объекта (.so) можно импортировать и использовать из Python, так что теперь мы можем запустить `test.py` :

```
$ python test.py
(23.0 ^ 2) + 23.0 = 552.0
```

Прочитайте Начало работы с cython онлайн: https://riptutorial.com/ru/cython/topic/2925/начало-работы-с-cython

глава 2: Комплектация Cython

Examples

Объединение программы Cython с помощью pyinstaller

Начните с программы Cython с точки входа:

```
def do_stuff():
    cdef int a,b,c
    a = 1
    b = 2
    c = 3
    print("Hello World!")
    print([a,b,c])
    input("Press Enter to continue.")
```

Создайте файл `setup.py` в той же папке:

```
from distutils.core import setup
from Cython.Build import cythonize
setup(
    name = "Hello World",
    ext_modules = cythonize('program.pyx'),
)
```

`python setup.py build_ext --inplace` его с помощью `python setup.py build_ext --inplace` вы `.pyd` библиотеку `.pyd` в `.pyd` .

После этого создайте ванильный скрипт Python с помощью библиотеки (например, `main.py`) и поместите файл `.pyd` рядом с ним:

```
import program
program.do_stuff()
```

Используйте PyInstaller для связывания его `pyinstaller --onefile "main.py"` . Это создаст вложенную папку, содержащую исполняемый файл размером 4 МБ +, содержащий библиотеку плюс время выполнения python.

Автоматическая сборка (Windows)

Для автоматизации вышеуказанной процедуры в Windows используйте `.bat` аналогичного содержимого:

```
del "main.exe"
python setup.py build_ext --inplace
del "*.c"
```

```
rmdir /s /q ".\build"
pyinstaller --onefile "main.py"
copy /y ".\dist\main.exe" ".\main.exe"
rmdir /s /q ".\dist"
rmdir /s /q ".\build"
del "*.spec"
del "*.pyd"
```

Добавление Numpy в комплект

Чтобы добавить Numpy в пакет, измените `setup.py` с помощью ключевого слова `include_dirs` и необходимо импортировать numpy в сценарии Python-оболочки для уведомления Pyinstaller.

`program.pyx` :

```
import numpy as np
cimport numpy as np


def do_stuff():
    print("Hello World!")
    cdef int n
    n = 2
    r = np.random.randint(1,5)
    print("A random number: "+str(r))
    print("A random number multiplied by 2 (made by cdef):"+str(r*n))
    input("Press Enter to continue.")
```

`setup.py` :

```
from distutils.core import setup, Extension
from Cython.Build import cythonize
import numpy

setup(
    ext_modules=cythonize("hello.pyx"),
    include_dirs=[numpy.get_include()]
)
```

`main.py` :

```
import program
import numpy
program.do_stuff()
```

Прочитайте Комплектация Cython онлайн: https://riptutorial.com/ru/cython/topic/6386/комплектация-cython

глава 3: Копирование кода C

Examples

Использование функций из пользовательской библиотеки C

У нас есть библиотека C с именем `my_random` которая производит случайные числа из пользовательского дистрибутива. Он предоставляет две функции, которые мы хотим использовать: `set_seed(long seed)` и `rand()` (и многое другое нам не нужно). Чтобы использовать их в Китоне, нам нужно

1. определить интерфейс в файле .pxd и
2. вызовите функцию в файле .pyx.

Код

test_extern.pxd

```
# extern blocks define interfaces for Cython to C code
cdef extern from "my_random.h":
    double rand()
    void c_set_seed "set_seed" (long seed) # rename C version of set_seed to c_set_seed to
avoid naming conflict
```

test_extern.pyx

```
def set_seed (long seed):
    """Pass the seed on to the c version of set_seed in my_random."""
    c_set_seed(seed)

cpdef get_successes (int x, double threshold):
    """Create a list with x results of rand <= threshold

    Use the custom rand function from my_random.
    """
    cdef:
        list successes = []
        int i
    for i in range(x):
        if rand() <= threshold:
            successes.append(True)
        else:
            successes.append(False)
    return successes
```

Прочитайте Копирование кода C онлайн: https://riptutorial.com/ru/cython/topic/3626/

копирование-кода-с

глава 4: Обтекание C ++

Examples

Обтекание DLL: C ++ до Cython до Python

Это демонстрирует нетривиальный пример переноса C ++ dll с Cython. Он будет охватывать следующие основные этапы:

- Создайте пример DLL с C ++ с помощью Visual Studio.
- Оберните DLL с помощью Cython, чтобы он мог быть вызван в Python.

Предполагается, что Cython установлен и может успешно импортировать его в Python.

Для этапа DLL также предполагается, что вы знакомы с созданием DLL в Visual Studio.

Полный пример включает создание следующих файлов:

1. `complexFunLib.h` : Заголовочный файл для источника DLL на C ++
2. `complexFunLib.cpp` : CPP-файл для источника DLL на C ++
3. `ccomplexFunLib.pxd` : файл заголовка Cython
4. `complexFunLib.pyx` : файл Cython "wrapper"
5. `setup.py` : установочный файл Python для создания `complexFunLib.pyd` с Cython
6. `run.py` : Пример файла Python, который импортирует скомпилированную, обернутую Cython DLL

C ++ DLL Источник: `complexFunLib.h` и `complexFunLib.cpp`

Пропустите это, если у вас уже есть файл с исходным кодом DLL и заголовком. Во-первых, мы создаем источник C ++, из которого DLL будет скомпилирована с использованием Visual Studio. В этом случае мы хотим выполнить быстрые вычисления массивов со сложной экспоненциальной функцией. Следующие две функции выполняют вычисление `k*exp(ee)` на массивах `k` и `ee` , где результаты сохраняются в `res` . Существуют две функции для размещения как одиночной, так и двойной точности. Обратите внимание, что в этих примерных функциях используется OpenMP, поэтому убедитесь, что OpenMP включен в параметрах Visual Studio для проекта.

Файл H

```
// Avoids C++ name mangling with extern "C"
#define EXTERN_DLL_EXPORT extern "C" __declspec(dllexport)
#include <complex>
#include <stdlib.h>
```

```
// Handles 64 bit complex numbers, i.e. two 32 bit (4 byte) floating point numbers
EXTERN_DLL_EXPORT void mp_mlt_exp_c4(std::complex<float>* k,
                                     std::complex<float>* ee,
                                     int sz,
                                     std::complex<float>* res,
                                     int threads);

// Handles 128 bit complex numbers, i.e. two 64 bit (8 byte) floating point numbers
EXTERN_DLL_EXPORT void mp_mlt_exp_c8(std::complex<double>* k,
std::complex<double>* ee,
                                     int sz,
                                     std::complex<double>* res,
                                     int threads);
```

Файл CPP

```
#include "stdafx.h"
#include <stdio.h>
#include <omp.h>
#include "complexFunLib.h"

void mp_mlt_exp_c4(std::complex<float>* k,
                   std::complex<float>* ee,
                   int sz,
                   std::complex<float>* res,
                   int threads)
{
    // Use Open MP parallel directive for multiprocessing
    #pragma omp parallel num_threads(threads)
    {
        #pragma omp for
        for (int i = 0; i < sz; i++) res[i] = k[i] * exp(ee[i]);
    }
}

void mp_mlt_exp_c8(std::complex<double>* k,
                   std::complex<double>* ee,
                   int sz, std::complex<double>* res,
                   int threads)
{
    // Use Open MP parallel directive for multiprocessing
    #pragma omp parallel num_threads(threads)
    {
        #pragma omp for
        for (int i = 0; i < sz; i++) res[i] = k[i] * exp(ee[i]);
    }
}
```

Cython Источник: `ccomplexFunLib.pxd` и `complexFunLib.pyx`

Затем мы создаем исходные файлы Cython, необходимые для оболочки библиотеки C ++. На этом этапе мы делаем следующие предположения:

- Вы установили Cython
- У вас есть рабочая DLL, например, описанная выше

Конечной целью является создание этих исходных файлов Cython в сочетании с оригинальной DLL для компиляции файла `.pyd` который может быть импортирован как модуль Python и раскрывает функции, написанные на C ++.

Файл PXD

Этот файл соответствует заголовочному файлу C ++. В большинстве случаев вы можете скопировать-вставить заголовок в этот файл с незначительными специфическими изменениями в Cython. В этом случае использовались конкретные типы комплекса Cython. Обратите внимание на добавление `c` в начале `ccomplexFunLib.pxd` . Это необязательно, но мы обнаружили, что такое соглашение об именах помогает поддерживать организацию.

```
cdef extern from "complexFunLib.h":
    void mp_mlt_exp_c4(float complex* k, float complex* ee, int sz,
                       float complex* res, int threads);
    void mp_mlt_exp_c8(double complex* k, double complex* ee, int sz,
                       double complex* res, int threads);
```

Файл PYX

Этот файл соответствует исходному файлу C ++ `cpp` . В этом примере мы будем передавать указатели на объекты Numpy `ndarray` на функции DLL импорта. Также возможно использовать встроенный объект `memoryview` Cython для массивов, но его производительность может быть не такой хорошей, как объекты `ndarray` (однако синтаксис значительно чище).

```
cimport ccomplexFunLib  # Import the pxd "header"
# Note for Numpy imports, the C import most come AFTER the Python import
import numpy as np  # Import the Python Numpy
cimport numpy as np  # Import the C Numpy

# Import some functionality from Python and the C stdlib
from cpython.pycapsule cimport *

# Python wrapper functions.
# Note that types can be delcared in the signature

def mp_exp_c4(np.ndarray[np.complex64_t, ndim=1] k,
              np.ndarray[np.complex64_t, ndim=1] ee,
              int sz,
              np.ndarray[np.complex64_t, ndim=1] res,
              int threads):
    '''
    TODO: Python docstring
    '''
    # Call the imported DLL functions on the parameters.
    # Notice that we are passing a pointer to the first element in each array
    ccomplexFunLib.mp_mlt_exp_c4(&k[0], &ee[0], sz, &res[0], threads)

def mp_exp_c8(np.ndarray[np.complex128_t, ndim=1] k,
              np.ndarray[np.complex128_t, ndim=1] ee,
              int sz,
              np.ndarray[np.complex128_t, ndim=1] res,
```

```
            int threads):
    '''
    TODO: Python docstring
    '''
    ccomplexFunLib.mp_mlt_exp_c8(&k[0], &ee[0], sz, &res[0], threads)
```

Python Источник: `setup.py` и `run.py`

setup.py

Этот файл представляет собой файл Python, который выполняет компиляцию Cython. Его цель - сгенерировать скомпилированный файл `.pyd` который затем может быть импортирован модулями Python. В этом примере мы сохранили все необходимые файлы (т.е. `complexFunLib.h` , `complexFunLib.dll` , `ccomplexFunLib.pxd` и `complexFunLib.pyx`) в том же каталоге, что и `setup.py` .

После создания этого файла он должен запускаться из командной строки с параметрами: `build_ext --inplace`

Как только этот файл будет выполнен, он должен создать файл `.pyd` без каких-либо ошибок. Обратите внимание, что в некоторых случаях, если есть ошибка, может быть создан `.pyd` , но недействителен. Убедитесь, что при `.pyd` `setup.py` не было ошибок при использовании сгенерированного `.pyd` .

```
from distutils.core import setup
from distutils.extension import Extension
from Cython.Distutils import build_ext
import numpy as np

ext_modules = [
    Extension('complexFunLib',
              ['complexFunLib.pyx'],
              # Note here that the C++ language was specified
              # The default language is C
              language="c++",
              libraries=['complexFunLib'],
              library_dirs=['.'])
    ]

setup(
    name = 'complexFunLib',
    cmdclass = {'build_ext': build_ext},
    ext_modules = ext_modules,
    include_dirs=[np.get_include()]  # This gets all the required Numpy core files
)
```

run.py

Теперь `complexFunLib` может быть импортирован непосредственно в модуль Python и вызваны связанные функции DLL.

```
import complexFunLib
```

```
import numpy as np

# Create arrays of non-trivial complex numbers to be exponentiated,
# i.e. res = k*exp(ee)
k = np.ones(int(2.5e5), dtype='complex64')*1.1234 + np.complex64(1.1234j)
ee = np.ones(int(2.5e5), dtype='complex64')*1.1234 + np.complex64(1.1234j)
sz = k.size  # Get size integer
res = np.zeros(int(2.5e5), dtype='complex64')  # Create array for results

# Call function
complexFunLib.mp_exp_c4(k, ee, sz, res, 8)

# Print results
print(res)
```

Прочитайте Обтекание C ++ онлайн: https://riptutorial.com/ru/cython/topic/3525/обтекание-c-plusplus

кредиты

S. No	Главы	Contributors
1	Начало работы с cython	Community, J.J. Hakala, Keith L, m00am
2	Комплектация Cython	Andrii Magalich
3	Копирование кода C	m00am
4	Обтекание C ++	J.J. Hakala, Keith L, Kevin Pasquarella