OUTLINE

- Accelerate a Python code

OUTLINE

- Accelerate a Python code
  - Using Numpy

OUTLINE

- Accelerate a Python code
  - Using Numpy
  - Using Cython

OUTLINE

- Accelerate a Python code
  - Using Numpy
  - Using Cython
  - Using Numba

OUTLINE

- Accelerate a Python code
  - Using Numpy
  - Using Cython
  - Using Numba
  - Using Pyccel

OUTLINE

- Accelerate a Python code
  - Using Numpy
  - Using Cython
  - Using Numba
  - Using Pyccel
- Some Benchmarks

ACCELERATE A PYTHON CODE: NUMPY

- Library for scientific computing in Python,

ACCELERATE A PYTHON CODE: NUMPY

- Library for scientific computing in Python,
- High-performance multidimensional array object,

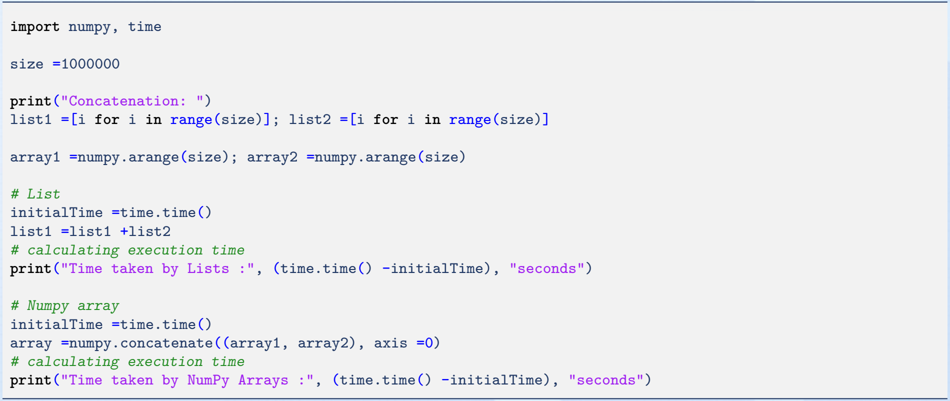
ACCELERATE A PYTHON CODE: NUMPY

- Library for scientific computing in Python,
- High-performance multidimensional array object,
- Integrates C, C++, and Fortran codes in Python,

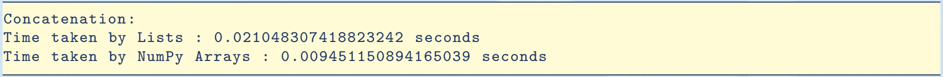
ACCELERATE A PYTHON CODE: NUMPY

- Library for scientific computing in Python,
- High-performance multidimensional array object,
- Integrates C, C++, and Fortran codes in Python,
- Uses multithreading.

ACCELERATE A PYTHON CODE: NUMPY VS LISTS

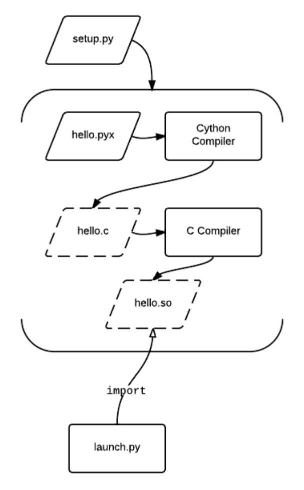


```python
import numpy, time

size =1000000

print("Concatenation: ")
list1 =[i for i in range(size)]; list2 =[i for i in range(size)]

array1 =numpy.arange(size); array2 =numpy.arange(size)

# List
initialTime =time.time()
list1 =list1 +list2
# calculating execution time
print("Time taken by Lists :", (time.time() -initialTime), "seconds")

# Numpy array
initialTime =time.time()
array =numpy.concatenate((array1, array2), axis =0)
# calculating execution time
print("Time taken by NumPy Arrays :", (time.time() -initialTime), "seconds")
```

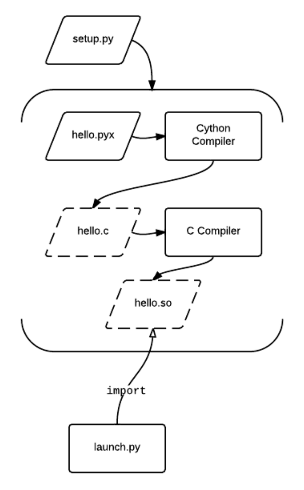


```
Concatenation:
Time taken by Lists : 0.021048307418823242 seconds
Time taken by NumPy Arrays : 0.009451150894165039 seconds
```

ACCELERATE A PYTHON CODE: CYTHON

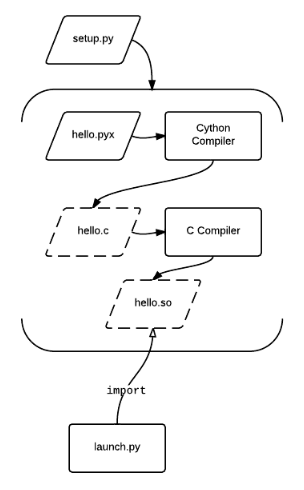


ACCELERATE A PYTHON CODE: CYTHON



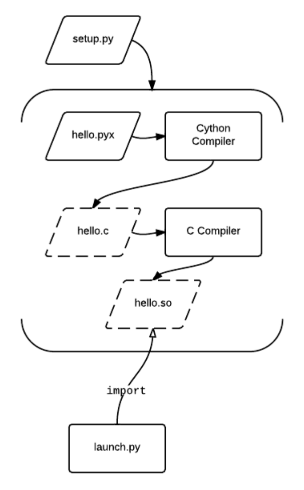
- Cython is an optimizing static compiler for:

ACCELERATE A PYTHON CODE: CYTHON



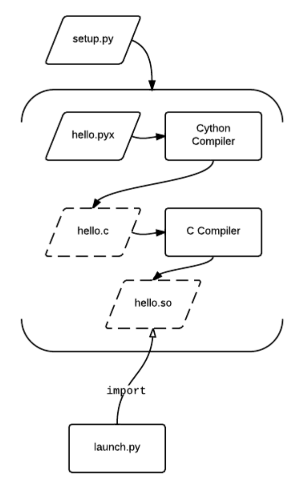
- Cython is an optimizing static compiler for:
  - Python programming language

ACCELERATE A PYTHON CODE: CYTHON



- Cython is an optimizing static compiler for:
  - Python programming language
  - Cython programming language (based on Pyrex)

ACCELERATE A PYTHON CODE: CYTHON



- Cython is an optimizing static compiler for:
  - Python programming language
  - Cython programming language (based on Pyrex)
- Cython gives you the combined power of Python.

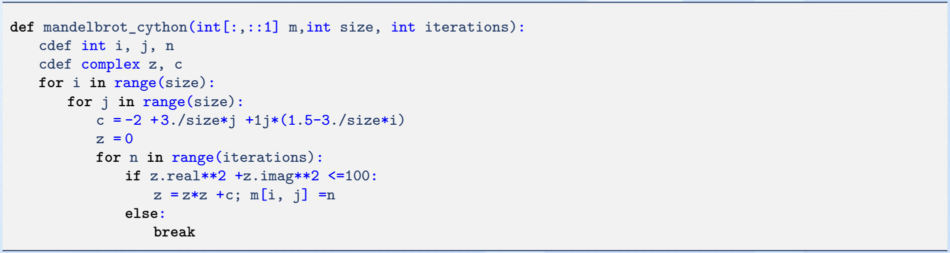
ACCELERATE A PYTHON CODE: CYTHON

- Python

```
def mandelbrot(m, size, iterations):
    for i in range(size):
        for j in range(size):
            c =-2 +3./size*j +1j*(1.5-3./size*i)
            z =0
            for n in range(iterations):
                if np.abs(z) <=10:
                    z =z*z +c; m[i, j] =n
                else:
                    break
```

ACCELERATE A PYTHON CODE: CYTHON

- Cython



```
def mandelbrot_cython(int[:,::1] m,int size, int iterations):
    cdef int i, j, n
    cdef complex z, c
    for i in range(size):
        for j in range(size):
            c =-2 +3./size*j +1j*(1.5-3./size*i)
            z =0
            for n in range(iterations):
                if z.real**2 +z.imag**2 <=100:
                    z =z*z +c; m[i, j] =n
                else:
                    break
```

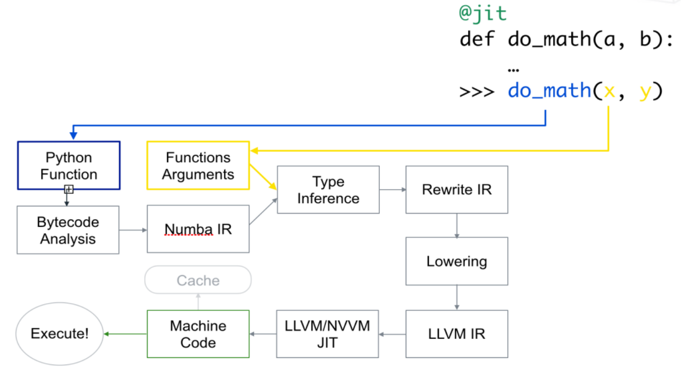
ACCELERATE A PYTHON CODE: CYTHON

- Execution time

```
%%timeit -n1 -r1
m = np.zeros(s, dtype=np.int32)
mandelbrot(m, size, iterations)
>> 12.2 s +/- 0 ns per loop (mean +/- std. dev. of 1 run, 1 loop each)


%%timeit -n1 -r1
m = np.zeros(s, dtype=np.int32)
mandelbrot_cython(m, size, iterations)
>> 29.8 ms +/- 0 ns per loop (mean +/- std. dev. of 1 run, 1 loop each)
```

ACCELERATE A PYTHON CODE: NUMBA

- Open source Just-In-Time compiler for python functions.
- Uses the LLVM library as the compiler backend.



```
@jit
def do_math(a, b):
        …
>>> do_math(x, y)
```

ACCELERATE A PYTHON CODE: NUMBA

- Python

```python
import numpy as np

def do_sum():
    acc =0.
    for i in range(10000001) :
        acc +=np.sqrt(i)
    return acc
```

- Numba

```python
from numba import njit

@njit
def do_sum_numba():
    acc =0.
    for i in range(10000001) :
        acc +=np.sqrt(i)
    return acc
```

```
Time for Pure Python Function: 7.724030017852783
Time for Numba Function: 0.015453100204467773
```

ACCELERATE A PYTHON CODE: PYCCEL (F90)

- Compilation using fortran:

```
pyccel --language=fortran pyccel_example.py
```

```fortran
module pyccel_example
use, intrinsic :: ISO_C_Binding, only : i64 => C_INT64_T , f64 => C_DOUBLE
    implicit none
    contains
    !........................................
    function do_sum_pyccel() result(acc)

        implicit none
        real(f64) :: acc
        integer(i64) :: i
        acc = 0.0_f64
        do i = 0_i64, 10000000_i64, 1_i64
            acc = acc + sqrt(Real(i, f64))
        end do
        return
    end function do_sum_pyccel
!........................................
end module pyccel_example
```

```
Time for Pure Python Function: 7.400242328643799
Time for Pyccel Function: 0.01545262336730957
```

ACCELERATE A PYTHON CODE: PYCCEL (C)

- Compilation using c:

```
pyccel --language=c pyccel_example.py
```

```c
#include "pyccel_example.h"
#include <stdlib.h>
#include <math.h>
#include <stdint.h>
/*........................................*/
double do_sum_pyccel(void)
{
    int64_t i;
    double acc;
    acc = 0.0;
    for (i = 0; i < 10000001; i += 1)
    {
        acc += sqrt((double)(i));
    }
    return acc;
}
/*........................................*/
```

SOME BENCHMARKS

Rosen-Der

Tool	Python	Cython	Numba	Pythran	Pyccel-gcc	Pyccel-intel
Timing (μs)	229.85	2.06	4.73	2.07	0.98	0.64
Speedup	—	× 111.43	× 48.57	× 110.98	× 232.94	× 353.94

SOME BENCHMARKS

Rosen-Der

Tool	Python	Cython	Numba	Pythran	Pyccel-gcc	Pyccel-intel
Timing (μs)	229.85	2.06	4.73	2.07	0.98	0.64
Speedup	—	× 111.43	× 48.57	× 110.98	× 232.94	× 353.94

Black-Scholes

Tool	Python	Cython	Numba	Pythran	Pyccel-gcc	Pyccel-intel
Timing (μs)	180.44	309.67	3.0	1.1	1.04	$6.56\ 10^{-2}$
Speedup	—	× 0.58	× 60.06	× 163.8	× 172.35	× 2748.71

SOME BENCHMARKS

Rosen-Der

Tool	Python	Cython	Numba	Pythran	Pyccel-gcc	Pyccel-intel
Timing (μs)	229.85	2.06	4.73	2.07	0.98	0.64
Speedup	—	× 111.43	× 48.57	× 110.98	× 232.94	× 353.94

Black-Scholes

Tool	Python	Cython	Numba	Pythran	Pyccel-gcc	Pyccel-intel
Timing (μs)	180.44	309.67	3.0	1.1	1.04	$6.56\ 10^{-2}$
Speedup	—	× 0.58	× 60.06	× 163.8	× 172.35	× 2748.71

Laplace

Tool	Python	Cython	Numba	Pythran	Pyccel-gcc	Pyccel-intel
Timing (μs)	57.71	7.98	$6.46\ 10^{-2}$	$6.28\ 10^{-2}$	$8.02\ 10^{-2}$	$2.81\ 10^{-2}$
Speedup	—	× 7.22	× 892.02	× 918.56	× 719.32	× 2048.65