

Distributed Computing and Introduction to High Performance Computing

Imad Kissami¹, Nouredine Ouhaddou¹

¹Mohammed VI Polytechnic University, Benguerir, Morocco



Outline of this lecture

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

Accelerate a Python code

What is Numpy

- The NumPy library is the core library for scientific computing in Python.
- It provides a high-performance multidimensional array object, and tools for working with these arrays.
- The NumPy package integrates C, C++, and Fortran codes in Python. These programming languages have very little execution time compared to Python.
- The NumPy package breaks down a task into multiple fragments and then processes all the fragments parallelly.

Accelerate a Python code

Numpy vs python benchmarks

```
1 import numpy, time
2
3 size = 1000000
4
5 print("Concatenation: ")
6 list1 = [i for i in range(size)]
7 list2 = [i for i in range(size)]
8
9 array1 = numpy.arange(size)
10 array2 = numpy.arange(size)
11
12 # List
13 initialTime = time.time()
14 list1 = list1 + list2
15 # calculating execution time
16 print("Time taken by Lists :", (time.time() - initialTime), "seconds")
17
18 # Numpy array
19 initialTime = time.time()
20 array = numpy.concatenate((array1, array2), axis = 0)
21 # calculating execution time
22 print("Time taken by NumPy Arrays :", (time.time() - initialTime), "seconds")
```

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

Accelerate a Python code

Numpy vs python benchmarks

```
1 import numpy, time
2
3 ...
4
5 dot = 0
6 print("\nDot Product:")
7
8 # List
9 initialTime = time.time()
10 for a, b in zip(list1, list2):
11     dot = dot + (a * b)
12 print("Time taken by Lists :", (time.time() - initialTime), "seconds")
13
14 # Numpy array
15 initialTime = time.time()
16 array = numpy.dot(array1, array2)
17 print("Time taken by NumPy Arrays :", (time.time() - initialTime), "seconds")
```

```
1 Dot Product:
2 Time taken by Lists : 0.13322114944458008 seconds
3 Time taken by NumPy Arrays : 0.0025365352630615234 seconds
```

Accelerate a Python code

Numpy vs python benchmarks

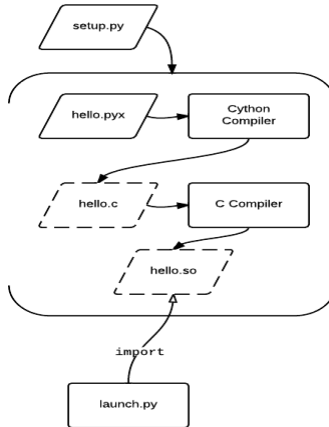
```
1 import numpy, time
2
3 ...
4
5 print("\nDeletion: ")
6
7 # List
8 initialTime = time.time()
9 del(list1)
10 print("Time taken by Lists :", (time.time() - initialTime), "seconds")
11
12 # NumPy array
13 initialTime = time.time()
14 del(array1)
15 print("Time taken by NumPy Arrays :", (time.time() - initialTime), "seconds")
```

```
1 Deletion:
2 Time taken by Lists : 0.016112804412841797 seconds
3 Time taken by NumPy Arrays : 9.512901306152344e-05 seconds
```

Accelerate a Python code

What is Cython

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



Accelerate a Python code

Cython example

■ Python

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

■ Cython

```
1 def mandelbrot_cython(int[:,::1] m, int size, int iterations):
2     cdef int i, j, n
3     cdef complex z, c
4     for i in range(size):
5         for j in range(size):
6             c = -2 + 3./size*j + 1j*(1.5-3./size*i)
7             z = 0
8             for n in range(iterations):
9                 if z.real**2 + z.imag**2 >= 100:
10                     z = z*z + c; m[i, j] = n
11                 else:
12                     break
```


Accelerate a Python code

Cython example

■ Execution time

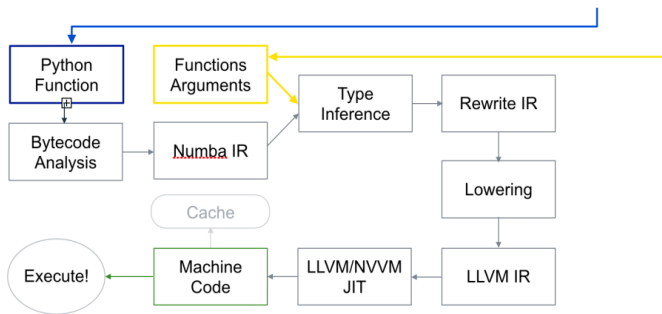
```
1 %%timeit -n1 -r1
2 m = np.zeros(s, dtype=np.int32)
3 mandelbrot(m, size, iterations)
4 >> 12.2 s +/- 0 ns per loop (mean +/- std. dev. of 1 run, 1 loop each)
5
6
7 %%timeit -n1 -r1
8 m = np.zeros(s, dtype=np.int32)
9 mandelbrot_cython(m, size, iterations)
10 >> 29.8 ms +/- 0 ns per loop (mean +/- 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 example

■ Python

```
1 import numpy as np
2
3 def do_sum():
4     acc = 0.
5     for i in range(10000000) :
6         acc += np.sqrt(i)
7     return acc
```

■ Numba

```
1 from numba import njit
2
3 @njit
4 def do_sum_numba():
5     acc = 0.
6     for i in range(10000000) :
7         acc += np.sqrt(i)
8     return acc
```

■ Execution time:

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

Accelerate a Python code

Pyccel

- Pyccel is a static compiler for Python 3, using Fortran or C as a backend language, with a focus on high-performance computing (HPC) applications.
- Public repository is now hosted on GitHub, freely available for download.
- Python function:

```
1 import numpy as np
2
3 def do_sum_pyccel():
4     acc = 0.
5     for i in range(10000000) :
6         acc += np.sqrt(i)
7     return acc
```

- Compilation using fortran:

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

Accelerate a Python code

Pyccel: Generated fortran function

```
1 module pyccel_example
2
3 use, intrinsic :: ISO_C_Binding, only : i64 => C_INT64_T , f64 => C_DOUBLE
4   implicit none
5
6   contains
7   ! .....
8   function do_sum_pyccel() result(acc)
9
10      implicit none
11      real(f64) :: acc
12      integer(i64) :: i
13
14      acc = 0.0_f64
15      do i = 0_i64, 9999999_i64, 1_i64
16         acc = acc + sqrt(Real(i, f64))
17      end do
18      return
19
20 end function do_sum_pyccel
21 ! .....
22
23 end module pyccel_example
```

■ Execution time:

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

Accelerate a Python code

Pyccel: Generated c function

```
1 #ifndef PYCCEL_EXAMPLE_H
2 #define PYCCEL_EXAMPLE_H
3
4 #include <stdlib.h>
5
6 double do_sum_pyccel(void);
7 #endif // PYCCEL_EXAMPLE_H
```

```
1 #include "pyccel_example.h"
2 #include <stdlib.h>
3 #include <math.h>
4 #include <stdint.h>
5
6 /* ..... */
7 double do_sum_pyccel(void)
8 {
9     int64_t i;
10    double acc;
11    acc = 0.0;
12    for (i = 0; i < 10000000; i += 1)
13    {
14        acc += sqrt((double)(i));
15    }
16    return acc;
17 }
18 /* ..... */
```

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 | — | $\times 111.43$ | $\times 48.57$ | $\times 110.98$ | $\times 232.94$ | $\times 353.94$ |

Black-Scholes

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

Laplace

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

Growcut

| Tool | Python | Cython | Numba | Pythran | Pyccel-gcc | Pyccel-intel |
|------------|--------|----------------------|----------------------|----------------------|----------------------|----------------------|
| Timing (s) | 54.39 | $1.02 \cdot 10^{-1}$ | $4.67 \cdot 10^{-1}$ | $8.57 \cdot 10^{-2}$ | $6.27 \cdot 10^{-2}$ | $6.54 \cdot 10^{-2}$ |
| Speedup | — | $\times 532.37$ | $\times 116.45$ | $\times 634.32$ | $\times 866.49$ | $\times 831.7$ |