Accelerate Python with Numba

A Quick Introduction

Haoran Peng

Edinburgh Cohort Meeting, 17 August 2020

Table of Contents

Table of Contents

What is Numba?

Here is the description from Numba's website:

What is Numba?

Here is the description from Numba's website:

- Compiles Python code just-in-time (JIT)

What is Numba?

Here is the description from Numba's website:

- Compiles Python code just-in-time (JIT)
- Supports a subset of Python and NumPy

What is Numba?

Here is the description from Numba's website:

- Compiles Python code just-in-time (JIT)
- Supports a subset of Python and NumPy
- Translates directly to machine code

Toy (Unfair) Example: Sum from 1 to N

```
def sum_py(n):
    ans = 0
    for i in range(1, n+1):
        ans += i
    return ans
```

Toy (Unfair) Example: Sum from 1 to N

```
def sum_py(n):
    ans = 0
    for i in range(1, n+1):
        ans += i
    return ans
```

```
from numba import jit

@jit
def sum_nb(n):
    ans = 0
    for i in range(1, n+1):
        ans += i
    return ans
```

Table of Contents

Premature Optimization is the Root of All Evil

I am doing a project on constituency parsing and need to use the CYK algorithm.

You don't need to know the details about CYK, it is just a dynamic programming algorithm with $O(n^3|G|)$ complexity, i.e. 3 for loops.

n is the length of the sentence to be parsed.

$|G|$ is the size of the grammar.

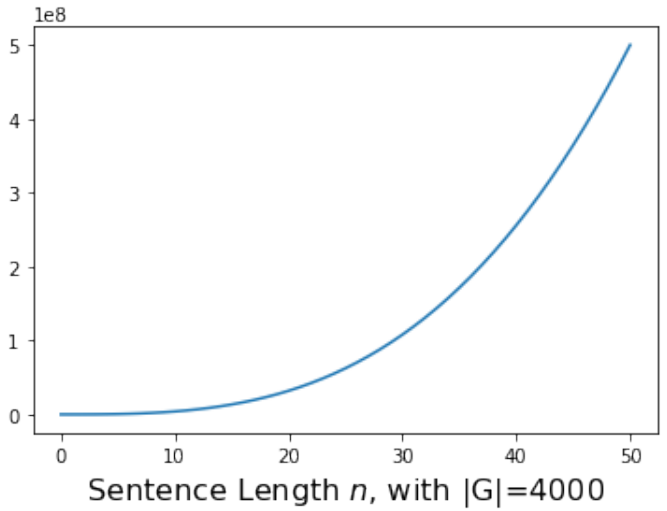
Premature Optimization is the Root of All Evil

I am doing a project on constituency parsing and need to use the CYK algorithm.

You don't need to know the details about CYK, it is just a dynamic programming algorithm with $O(n^3|G|)$ complexity, i.e. 3 for loops.

n is the length of the sentence to be parsed.

$|G|$ is the size of the grammar.



Parsing **cannot be done in reasonable time** using pure Python.

Code to Accelerate

We are going to tackle the edit distance problem which is very hard for the compiler to optimise:

```
def edit_dist(s , t):
        m, n = len(s), len(t)
        dp = [[0]*(n + 1) for _ in range(m + 1)]
        for i in range(m + 1):
            dp[i][0] = i
        for j in range(n + 1):
            dp[0][j] = j
        for i in range(1, m + 1):
            for j in range(1, n + 1):
                if s[i - 1] == t[j - 1]:
                    substitution_cost = 0
                else:
                    substitution_cost = 1
                dp[i][j] = min(dp[i - 1][j] + 1,
                               dp[i][j - 1] + 1,
                               dp[i - 1][j - 1] + substitution_cost)
        return dp[m][n]
```

		S	a	t	u	r	d	a	y
	0	1	2	3	4	5	6	7	8
S	1	0	1	2	3	4	5	6	7
u	2	1	1	2	2	3	4	5	6
n	3	2	2	2	3	3	4	5	6
d	4	3	3	3	3	4	3	4	5
a	5	4	3	4	4	4	4	3	4
y	6	5	4	4	5	5	5	4	3

Image taken from Wikipedia.

Table of Contents

`@jit(nopython=True)` and `@njit`

It would be nice if we can just `@jit` every function and it would magically run much faster, but it isn't the case.

`@jit(nopython=True)` and `@njit`

It would be nice if we can just `@jit` every function and it would magically run much faster, but it isn't the case.

Use `@njit` only, because we don't want our "accelerated" code to run slower.

`@jit(nopython=True)` and `@njit`

It would be nice if we can just `@jit` every function and it would magically run much faster, but it isn't the case.

Use `@njit` only, because we don't want our "accelerated" code to run slower.

Although you can call into the Python interpreter in nopython mode using the `objmode` context-manager (currently experimental).

Re-write Unsupported Functions

Numba only supports a subset of Python and NumPy, thus it is usually necessary to re-write some functions for `@njit` to work.

[1] As of 15/08/2020 when reflected list is still supported.

Re-write Unsupported Functions

Numba only supports a subset of Python and NumPy, thus it is usually necessary to re-write some functions for `@njit` to work.

Though our edit distance code can be `@njit`:ed directly.[1]

[1] As of 15/08/2020 when reflected list is still supported.

Use NumPy Arrays Instead of List if Possible

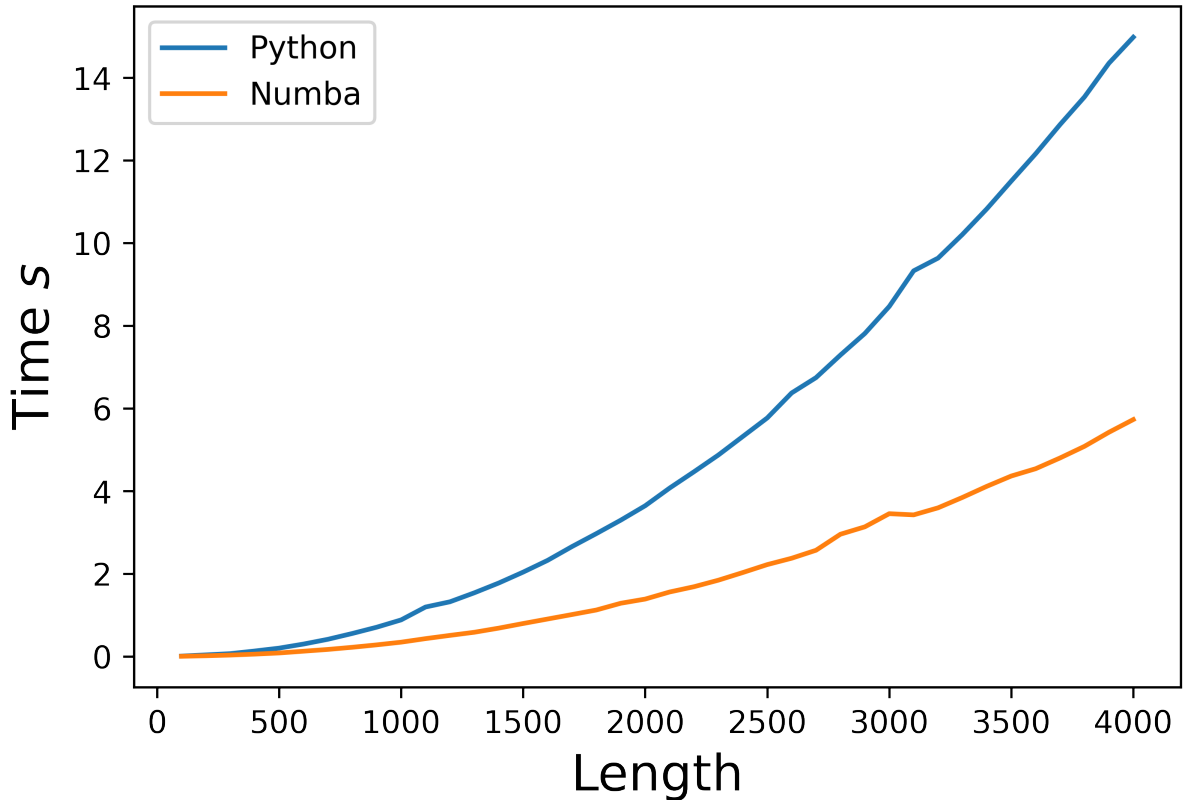
NumPy arrays are contiguous in memory, thus much faster to access. Use them instead of lists if your problem permit you to do so.

Use NumPy Arrays Instead of List if Possible

NumPy arrays are contiguous in memory, thus much faster to access. Use them instead of lists if your problem permit you to do so.

Most of the basic NumPy operations are supported in Numba.

Performance Comparison



Use Typed Data Structures

Python (reflected) list in jitted function will be deprecated soon.
Replace the list with a Numba typed-list.

Use Typed Data Structures

Python (reflected) list in jitted function will be deprecated soon.
Replace the list with a Numba typed-list.

Numba also has typed-dictionaries and (soon) typed-sets.

Multi-threading with `@njit(nogil=True)`

Because Numba code is compiled and does not depend on the Python interpreter, it can give up the GIL when executing. This makes (real) multi-threading possible in Python.

Multi-threading with `numba.prange`

The same multi-threading can often be achieved by using Numba's `prange` function. It automatically threads each iteration of the loop. But you give up some control by doing so.

Table of Contents

What more?

- Automatic nested threading using OpenMP/TBB.

What more?

- Automatic nested threading using OpenMP/TBB.
- Optimization according to your architecture with LLVM.

What more?

- Automatic nested threading using OpenMP/TBB.
- Optimization according to your architecture with LLVM.
- GPU/cuda support, can manipulate array-like objects as long as they implement `__cuda_array_interface__`.

What more?

- Automatic nested threading using OpenMP/TBB.
- Optimization according to your architecture with LLVM.
- GPU/cuda support, can manipulate array-like objects as long as they implement `__cuda_array_interface__`.
- Go check out Numba's reference manual, it's fairly well-written.

Current Python Acceleration Landscape

- Numpy, PyTorch, Tensorflow

Current Python Acceleration Landscape

- Numpy, PyTorch, Tensorflow
- Cython, Numba

Current Python Acceleration Landscape

- Numpy, PyTorch, Tensorflow
- Cython, Numba
- PyPy