# Parallelism In Python

Rounak Vyas

# Table Of Contents

# Overview

Python has a terrible rep when it comes to its parallel processing capabilities. Ignoring the standard arguments about its threads and the GIL (which are mostly valid), the real problem I see with parallelism in Python isn't a technical one, but a pedagogical one. The common tutorials surrounding Threading and Multiprocessing in Python, while generally excellent, are pretty “heavy.” They start in the intense stuff, and stop before they get to the really good, day-to-day useful parts.

# Is multi-threading a scam in Python?

YES.

# How the Python Interpreter Works

Python

```
>>> import sys
>>> a = []
>>> b = a
>>> sys.getrefcount(a)
3
```

# Global Interpreter Lock (GIL)

The Python Global Interpreter Lock or GIL, in simple words, is a mutex (or a lock) that allows only one thread to hold the control of the Python interpreter.

# Impact of GIL on multi-threaded programs

Let's have a look at a simple CPU-bound program that performs a countdown:

Python

```python
# single_threaded.py
import time
from threading import Thread

COUNT = 50000000

def countdown(n):
    while n>0:
        n -= 1

start = time.time()
countdown(COUNT)
end = time.time()

print('Time taken in seconds -', end - start)
```

6.20 secs

# Impact of GIL on multi-threaded programs

Python

```python
# multi_threaded.py
import time
from threading import Thread

COUNT = 50000000

def countdown(n):
    while n>0:
        n -= 1

t1 = Thread(target=countdown, args=(COUNT//2,))
t2 = Thread(target=countdown, args=(COUNT//2,))

start = time.time()
t1.start()
t2.start()
t1.join()
t2.join()
end = time.time()

print('Time taken in seconds -', end - start)
```

6.924 secs

# Introducing: Multi-Processing

The most popular way is to use a multi-processing approach where you use multiple processes instead of threads. Each Python process gets its own Python interpreter and memory space so the GIL won’t be a problem.

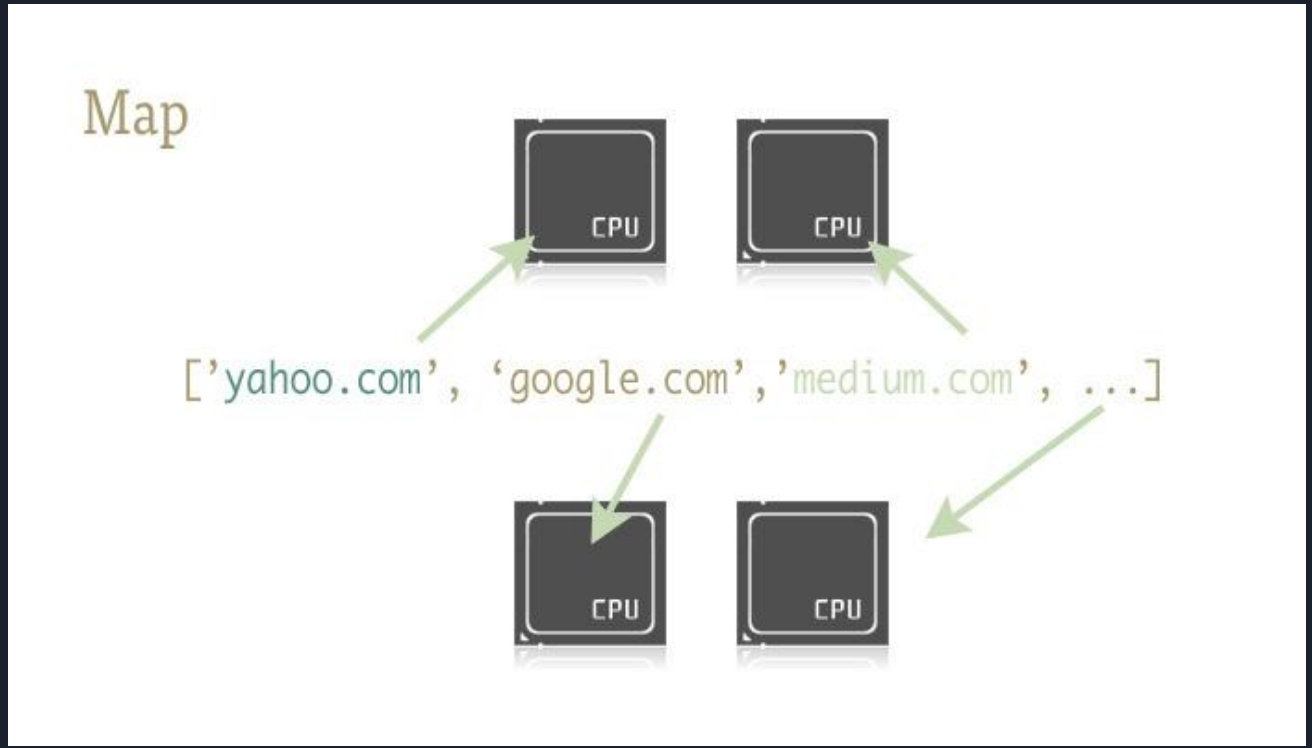
# Let’s talk about a Textbook Example

```
import urllib2
from multiprocessing.dummy import Pool as ThreadPool

urls = [
  'http://www.python.org',
  'http://www.python.org/about/',
  'http://www.onlamp.com/pub/a/python/2003/04/17/metaclasses.html',
  'http://www.python.org/doc/',
  'http://www.python.org/download/',
  'http://www.python.org/getit/',
  'http://www.python.org/community/',
  'https://wiki.python.org/moin/',
  'http://planet.python.org/',
  'https://wiki.python.org/moin/LocalUserGroups',
  'http://www.python.org/psf/',
  'http://docs.python.org/devguide/',
  'http://www.python.org/community/awards/'
  # etc..
  ]

# Make the Pool of workers
pool = ThreadPool(4)
# Open the urls in their own threads
# and return the results
results = pool.map(urllib2.urlopen, urls)
#close the pool and wait for the work to finish
pool.close()
pool.join()
```

# How Map works



# Real Life Example

## Thumbnailing thousands of images.

A common CPU bound  everyday task for someone working on vision, image processing , etc.

Includes manipulating massive image folders.

# Real Life Example

Basic single process step

```
for image in images:
  create_thumbnail(image)
```

On my machine, this took 27.9 seconds to process ~6000 images.

Parallel Map Call

```
pool = Pool()
  pool.map(create_thumbnail, images)
  pool.close()
  pool.join()
```

**5.6 seconds!**

# About Me

- Undergrad at SRM Institute of Science and Technology
- Interests :  AI/Deep Learning, Making Command Line Tools, Web Development (Django), System Design, Open-Source Enthusiast
- Previously Interned at : Thomson Reuters, Hyderabad and OpenGenus Foundation.

You can always contact me here -

- GitHub Url : https://github.com/itsron717
- LinkedIn Url: https://www.linkedin.com/in/itsron143/

Link to the Slides and Code Snippets :  https://github.com/itsron717/Talks

# Thanks!