Chrome File Edit View History Bookmarks Profiles Tab Window Help 7:38

Multi-processing & Multi-threading in Python (for ML/AI)

scaler.com

1 / 1

Chrome File Edit View History Bookmarks Profiles Tab Window Help 8:14

Agenda:

1. Multiprocessing & Multithreading
   - intuition
   - → when
   - → why

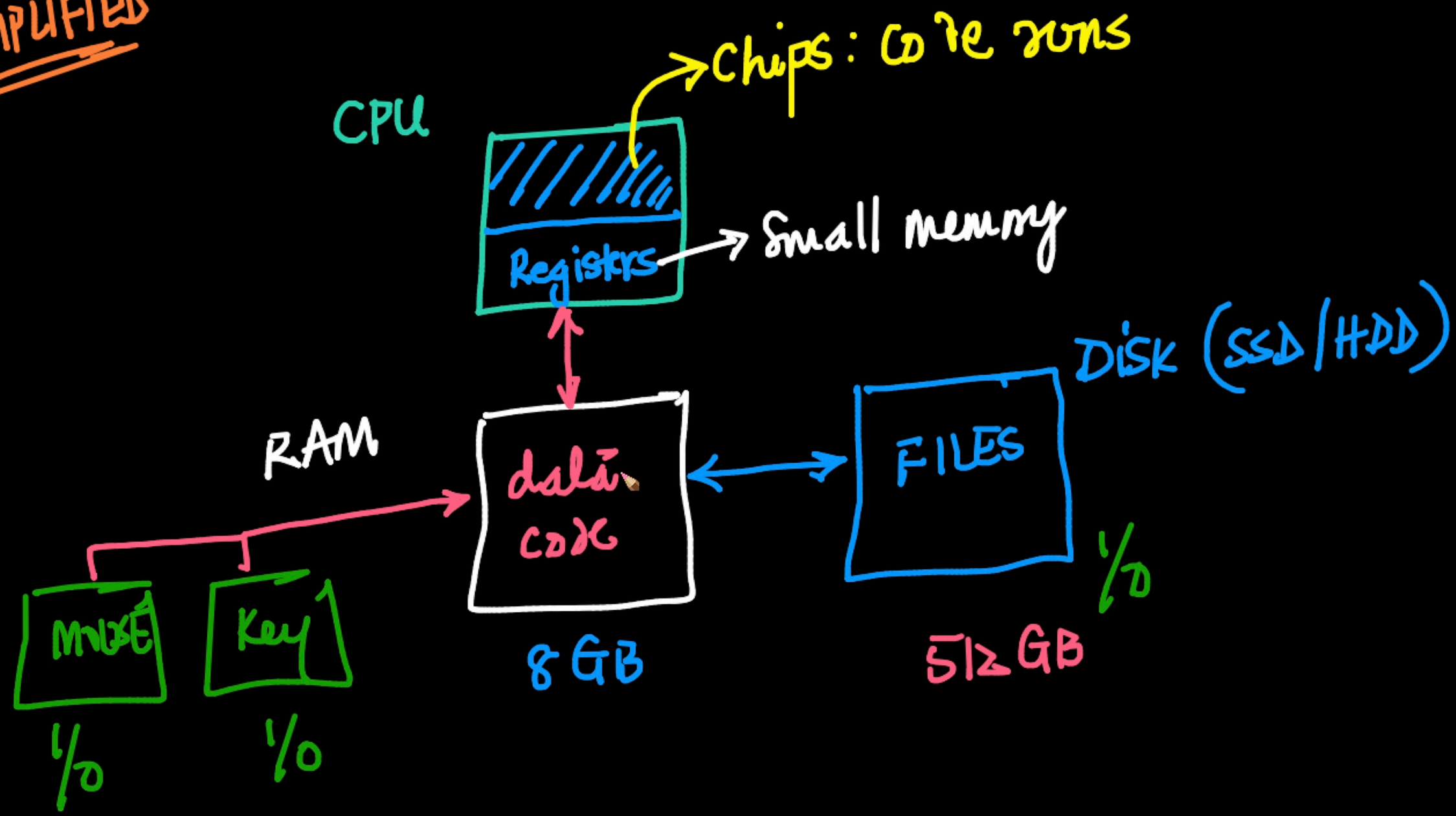
   Code + diagrams

2. Libraries — Python

2 / 2

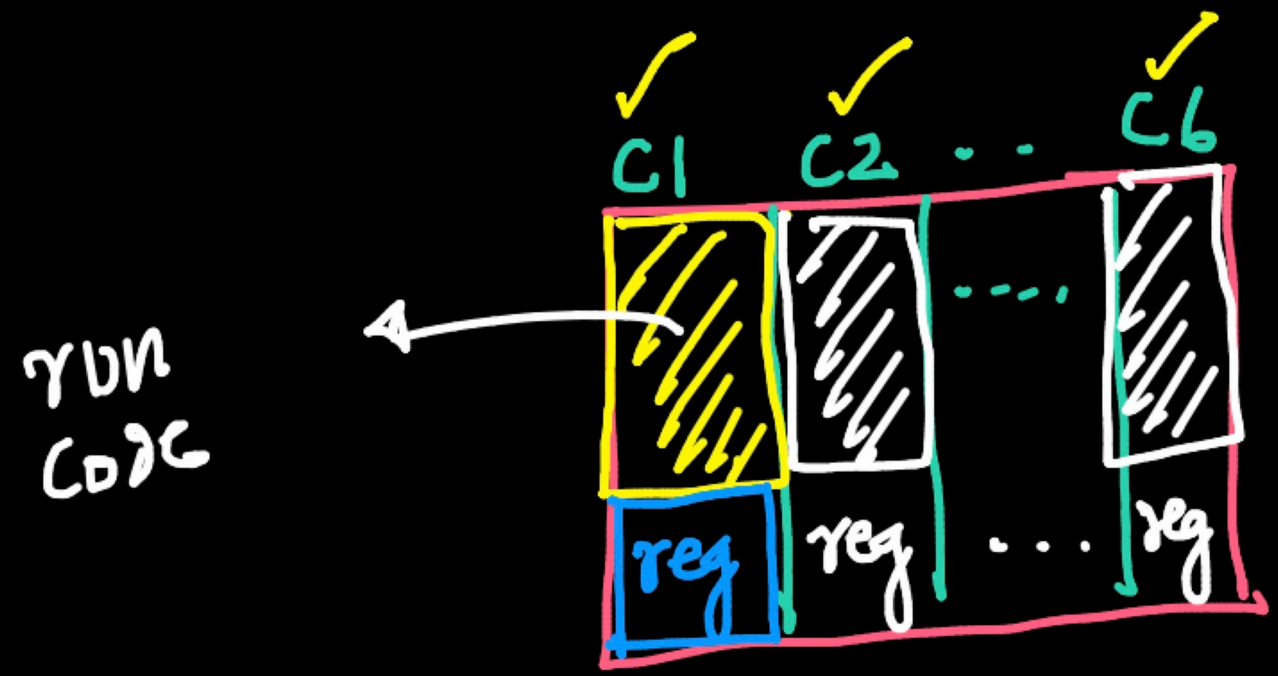
Chrome File Edit View History Bookmarks Profiles Tab Window Help 8:25

SIMPLIFIED



3 / 3

Chrome File Edit View History Bookmarks Profiles Tab Window Help 8:30



6-core Intel

Multi-processing

4 / 5

Chrome File Edit View History Bookmarks Profiles Tab Window Help 8:34

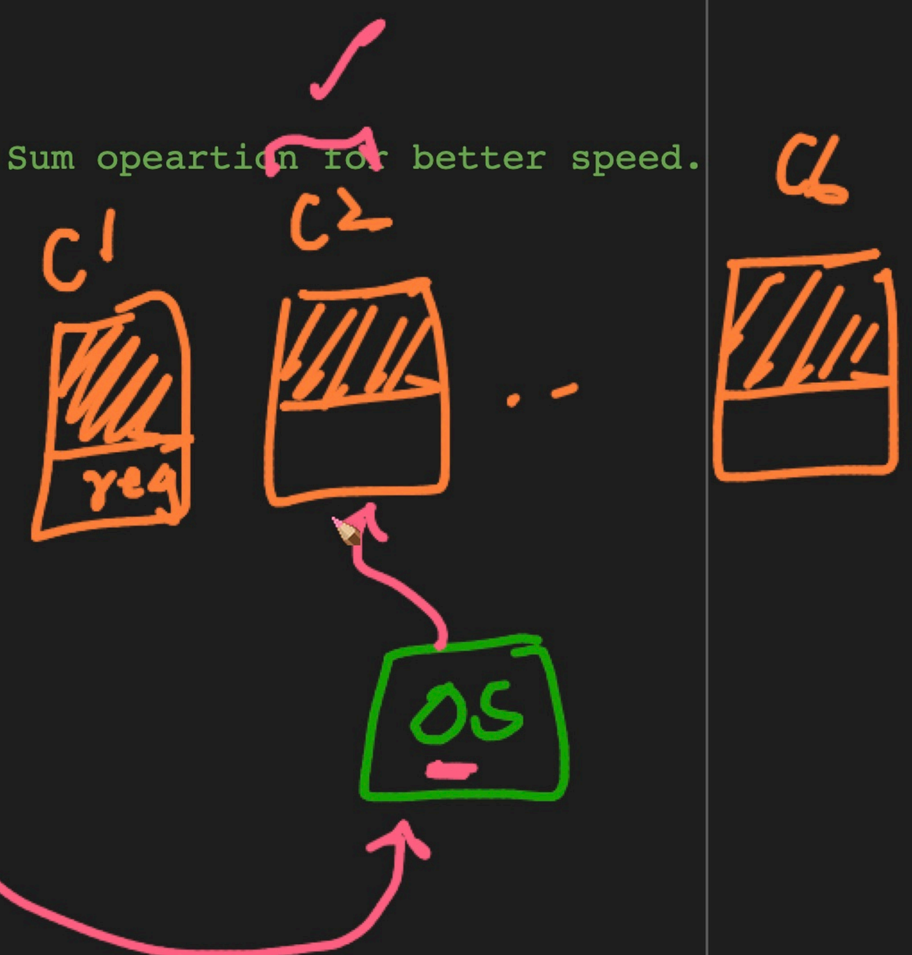
Copy of Multithreading_i | Find out how many cores | Multithreading (compute | multiprocessing - What | Numba: A High Performa

colab.research.google.com/drive/1n5JkAWDr_VRCQjlr6BVWdDpUgW4eZODr#scrollTo=okpN0WRo9qYk

+ Code + Text Connect

```
(100000000,)
```

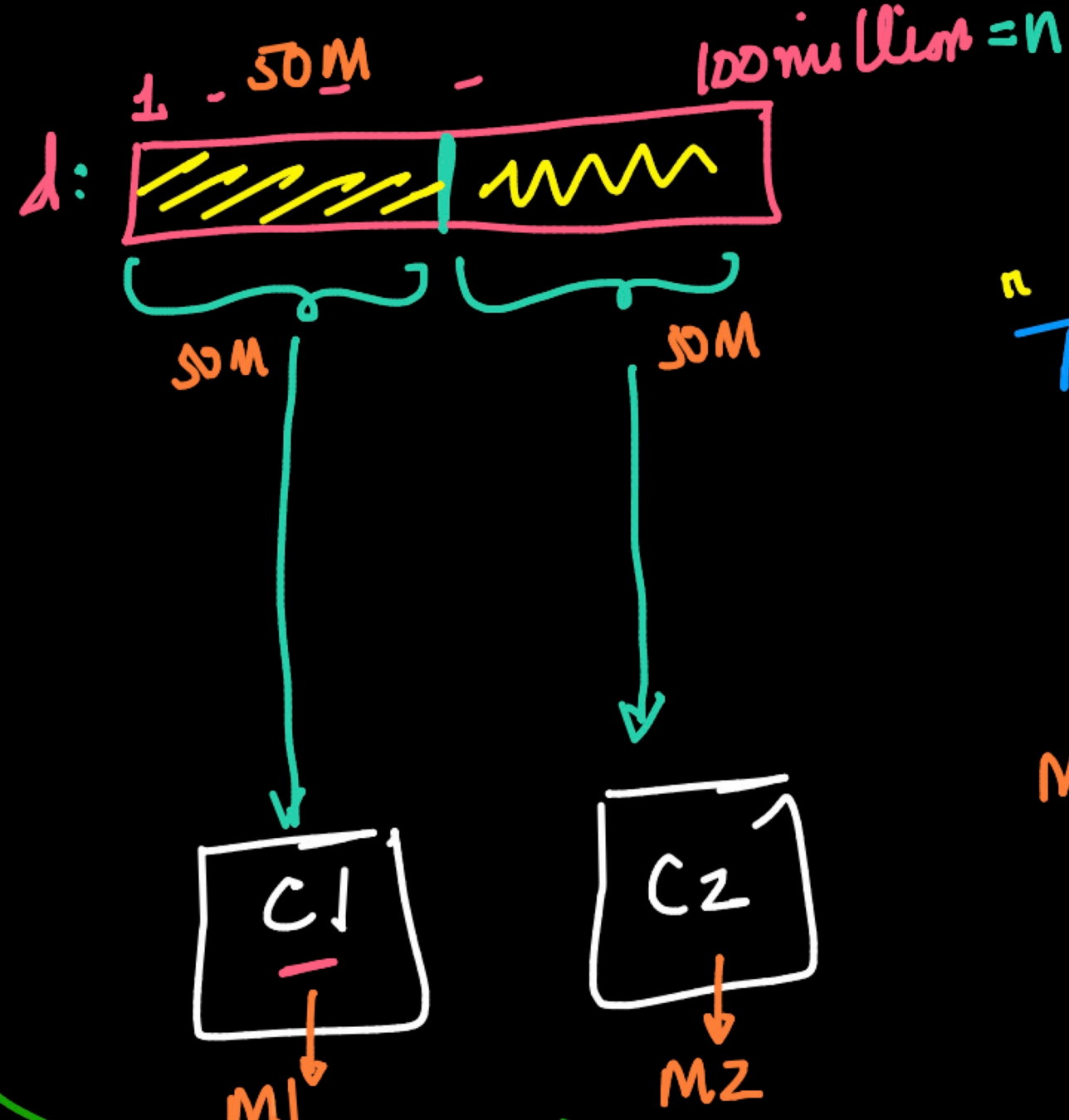
```python
import time
def mean():

  #Sum using for loops. We can use inbuilt NumPy Sum opeartion for better speed.
  sum = 0
  n=d.size
  for i in range(n):
    sum +=d[i]

  #Mean
  mean = sum/n
  return mean


#Time the execution
start_time = time.time()
m = mean() # compute mean of 100MM numbers.
end_time = time.time()
print (end_time-start_time)
print(m)
```



20.90

5 / 5

Task: avg



"Trivially" Parallelize

Process-Safe

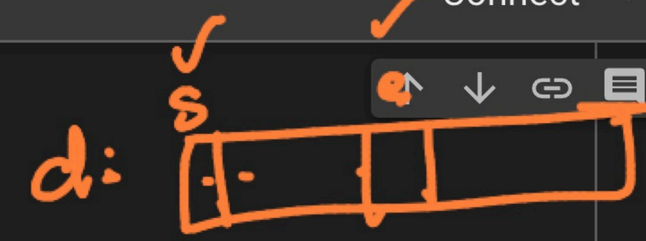
RAM

Queue:

$$M = \frac{M1 + M2}{2}$$

Chrome File Edit View History Bookmarks Profiles Tab Window Help 8:39
Copy of Multithreading_i | Find out how many cores | Multithreading (compute | multiprocessing - What | Numba: A High Performa
colab.research.google.com/drive/1n5JkAWDr_VRCQjlr6BVWdDpUgW4eZODr#scrollTo=f4m9X5dw-Pvh
Update
+ Code + Text
Connect

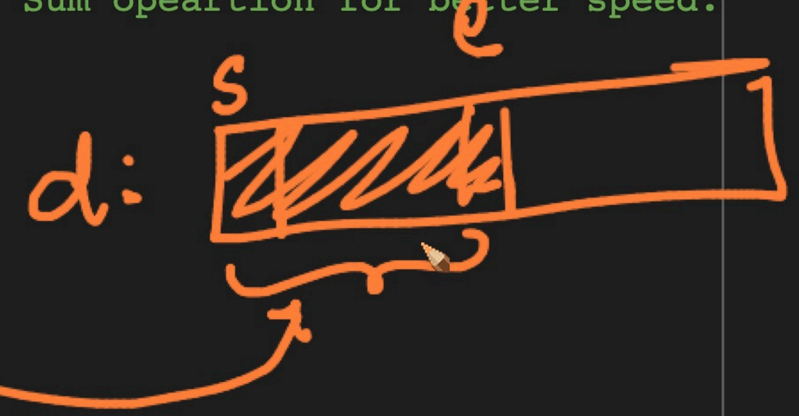
```python
from multiprocessing import Process, Queue
import math

def mean_MP(s, e, q ):

  #Sum using for loops. We can use inbuilt NumPy Sum opeartion for better speed.
  sum = 0
  for i in range(s,e+1):
    sum +=d[i]

  #Mean
  mean = sum/(e-s+1)
  q.put(mean)
  return

n1 = math.floor(n/2)

q = Queue() #Queues are thread and process safe. For communicating between processes and threads

p1 = Process(target=mean_MP, args=(0, n1,q ))
p2 = Process(target=mean_MP, args=(n1+1,n-1, q))


#Time the execution
start_time = time.time()
```



7 / 7

Chrome File Edit View History Bookmarks Profiles Tab Window Help 8:40
Copy of Multithreading_i | Find out how many cores | Multithreading (compute | multiprocessing - What | Numba: A High Performa
colab.research.google.com/drive/1n5JkAWDr_VRCQjlr6BVWdDpUgW4eZODr#scrollTo=f4m9X5dw-Pvh
+ Code + Text Connect

```python
from multiprocessing import Process, Queue
import math

def mean_MP(s, e, q ):

  #Sum using for loops. We can use inbuilt NumPy Sum opeartion for better speed.
  sum = 0
  for i in range(s,e+1):
    sum +=d[i]

  #Mean
  mean = sum/(e-s+1)
  q.put(mean)
  return

n1 = math.floor(n/2)

q = Queue() #Queues are thread and process safe. For communicating between processes and threads

p1 = Process(target=mean_MP, args=(0, n1,q ))
p2 = Process(target=mean_MP, args=(n1+1,n-1, q))


#Time the execution
start_time = time.time()
```



8 / 8

Chrome File Edit View History Bookmarks Profiles Tab Window Help 8:43
Copy of Multithreading_i | Find out how many cores | Multithreading (compute | multiprocessing - What | Numba: A High Performa
colab.research.google.com/drive/1n5JkAWDr_VRCQjlr6BVWdDpUgW4eZODr#scrollTo=f4m9X5dw-Pvh
Update
+ Code + Text
Connect

```python
from multiprocessing import Process, Queue
import math

def mean_MP(s, e, q ):

  #Sum using for loops. We can use inbuilt NumPy Sum opeartion for better speed.
  sum = 0
  for i in range(s,e+1):
    sum +=d[i]

  #Mean
  mean = sum/(e-s+1)
  q.put(mean)
  return

n1 = math.floor(n/2)

q = Queue() #Queues are thread and process safe. For communicating between processes and threads

p1 = Process(target=mean_MP, args=(0, n1,q ))
p2 = Process(target=mean_MP, args=(n1+1,n-1, q))


#Time the execution
start_time = time.time()
```

9 / 9

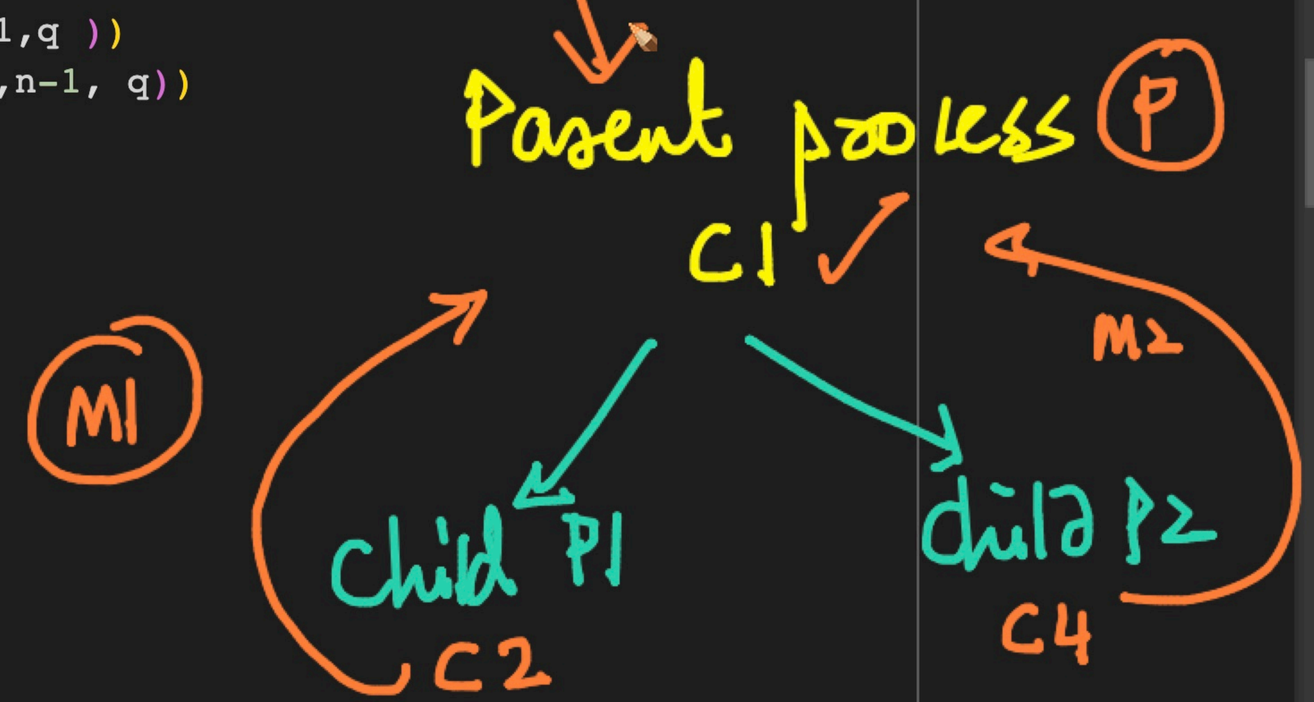
Chrome File Edit View History Bookmarks Profiles Tab Window Help 8:47
Copy of Multithreading_i | Find out how many cores | Multithreading (compute | multiprocessing - What | Numba: A High Performa
colab.research.google.com/drive/1n5JkAWDr_VRCQjlr6BVWdDpUgW4eZODr#scrollTo=f4m9X5dw-Pvh
+ Code + Text Connect

```
    sum +=d[i]

  #Mean
  mean = sum/(e-s+1)
  q.put(mean)
  return

n1 = math.floor(n/2)

q = Queue() #Queues are thread and process safe. For communicating between processes and threads

p1 = Process(target=mean_MP, args=(0, n1,q ))
p2 = Process(target=mean_MP, args=(n1+1,n-1,q))


#Time the execution
start_time = time.time()

p1.start()
p2.start()

p1.join() # Wait till p1 finishes
p2.join()
```

10 / 10

Chrome File Edit View History Bookmarks Profiles Tab Window Help 8:50
Copy of Multithreading_i | Find out how many cores | Multithreading (compute | multiprocessing - What | Numba: A High Performa
colab.research.google.com/drive/1n5JkAWDr_VRCQjlr6BVWdDpUgW4eZODr#scrollTo=f4m9X5dw-Pvh
+ Code + Text Connect

```
    #Mean
    mean = sum/(e-s+1)
    q.put(mean)
    return

n1 = math.floor(n/2)

q = Queue() #Queues are thread and process safe. For communicating between processes and threads

p1 = Process(target=mean_MP, args=(0, n1,q ))
p2 = Process(target=mean_MP, args=(n1+1,n-1, q))


#Time the execution
start_time = time.time()

p1.start()
p2.start()

p1.join() # Wait till p1 finishes
p2.join()

m=0;
while not q.empty():
    m += q.get()
```



11 / 11

Chrome File Edit View History Bookmarks Profiles Tab Window Help 8:50
Copy of Multithreading_i | Find out how many cores | Multithreading (compute | multiprocessing - What | Numba: A High Performa
colab.research.google.com/drive/1n5JkAWDr_VRCQjlr6BVWdDpUgW4eZODr#scrollTo=f4m9X5dw-Pvh
Update
+ Code + Text Connect

```python
#Time the execution
start_time = time.time()

p1.start()
p2.start()

p1.join() # Wait till p1 finishes
p2.join()

m=0;
while not q.empty():
      m += q.get()

m /= 2;

end_time = time.time()
print (end_time-start_time)
print(m)
```

$M_1 + M_2$

```
11.001178979873657
0.4999477716457993
```

Multi-Thr
12 / 12

Chrome File Edit View History Bookmarks Profiles Tab Window Help 8:50

Copy of Multithreading_i | Find out how many cores | Multithreading (compute | multiprocessing - What | Numba: A High Performa

colab.research.google.com/drive/1n5JkAWDr_VRCQjlr6BVWdDpUgW4eZODr#scrollTo=f4m9X5dw-Pvh

+ Code + Text Connect

```
#Time the execution
start_time = time.time()

p1.start()
p2.start()

p1.join() # Wait till p1 finishes
p2.join()

m=0;
while not q.empty():
      m += q.get()

m /= 2;

end_time = time.time()
print (end_time-start_time)
print(m)

```

```
11.001178979873657
0.4999477716457993
```

13 / 13

Chrome File Edit View History Bookmarks Profiles Tab Window Help 8:52

Copy of Multithreading_i | Find out how many cores | Multithreading (compute | multiprocessing - What | Numba: A High Performa

colab.research.google.com/drive/1n5JkAWDr_VRCQjlr6BVWdDpUgW4eZODr#scrollTo=f4m9X5dw-Pvh

+ Code + Text Connect

```
#Time the execution
start_time = time.time()

p1.start()
p2.start()

p1.join() # Wait till p1 finishes
p2.join()

m=0;
while not q.empty():
    m += q.get()

m /= 2;

end_time = time.time()
print (end_time-start_time)
print(m)
```

1 Core → 21 sec

2 cores → 11 sec

overhead

```
11.001178979873657
0.4999477716457993
```

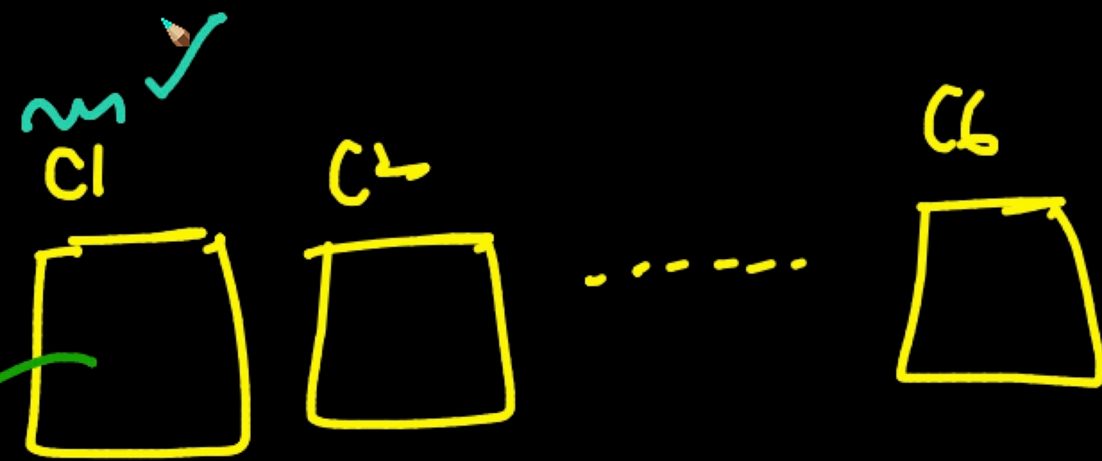
Multi-Thr

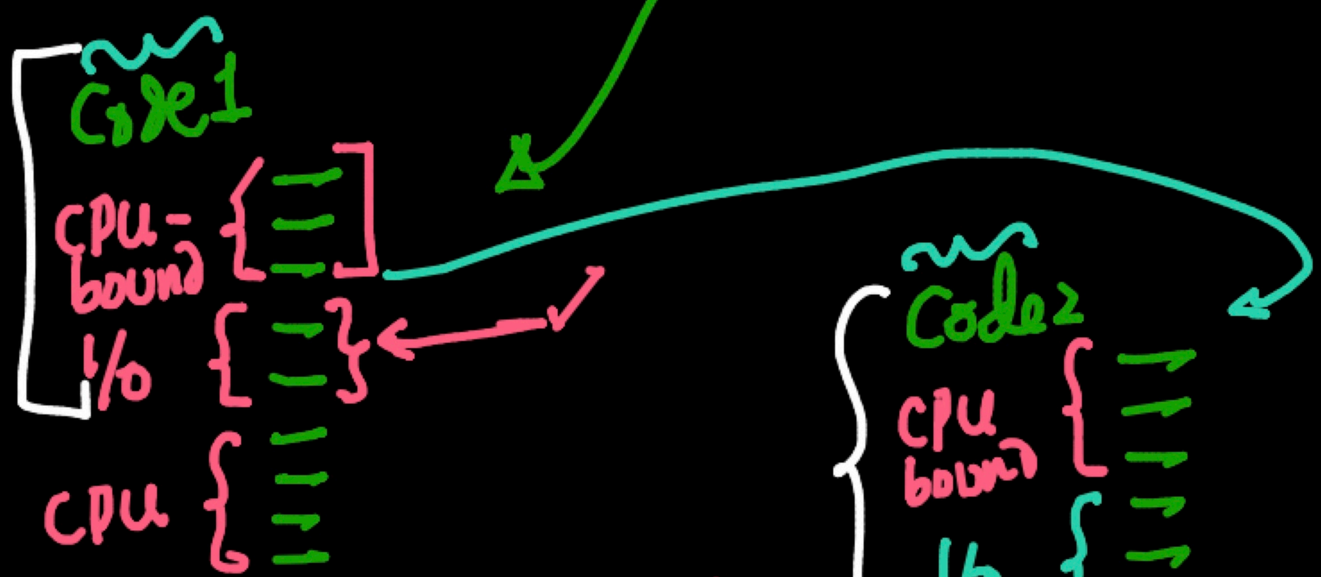
14 / 14

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:05

Multi-threading

Intel - 90's





Chase lhe
Same Dalai

1 Billion — 100M RAM
— 900M Disk

15 / 15

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:05
Copy of Multithreading_i | Find out how many cores | Multithreading (compute | multiprocessing - What | Numba: A High Performa
en.wikipedia.org/wiki/Multithreading_(computer_architecture) Update

From Wikipedia, the free encyclopedia

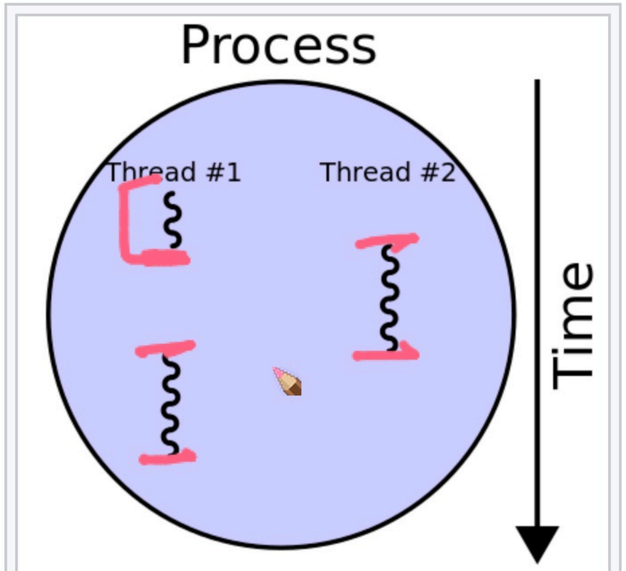
For threads in software, see Thread (computing).

This article has multiple issues. Please help **improve it** or discuss these issues on the **talk page**. *(Learn how and when to remove these template messages)* [hide]

- This article **needs additional citations for verification**. *(October 2009)*
- This article **possibly contains original research**. *(October 2010)*

In computer architecture, **multithreading** is the ability of a central processing unit (CPU) (or a single core in a multi-core processor) to provide multiple threads of execution concurrently, supported by the operating system. This approach differs from multiprocessing. In a multithreaded application, the threads share the resources of a single or multiple cores, which include the computing units, the CPU caches, and the translation lookaside buffer (TLB).

Where multiprocessing systems include multiple complete processing units in one or more cores, multithreading aims to increase utilization of a single core by using thread-level parallelism, as well as instruction-level parallelism. As the two techniques are complementary, they are combined in nearly all modern systems architectures with multiple multithreading CPUs and with CPUs with multiple multithreading cores.



A process with two threads of execution, running on a single processor.

Overview [edit]

Get citation

digm has become more popular as efforts to further exploit instruction-level
d since the late 1990s. This allowed the concept of throughput computing to re-emerge from the more specialized field of
transaction process s are actually

16 / 16

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:06

Copy of Multithreading_i | Find out how many cores | Multithreading (compute | multiprocessing - What | Numba: A High Performa

colab.research.google.com/drive/1n5JkAWDr_VRCQjlr6BVWdDpUgW4eZODr#scrollTo=ipbV9r5YDFZz

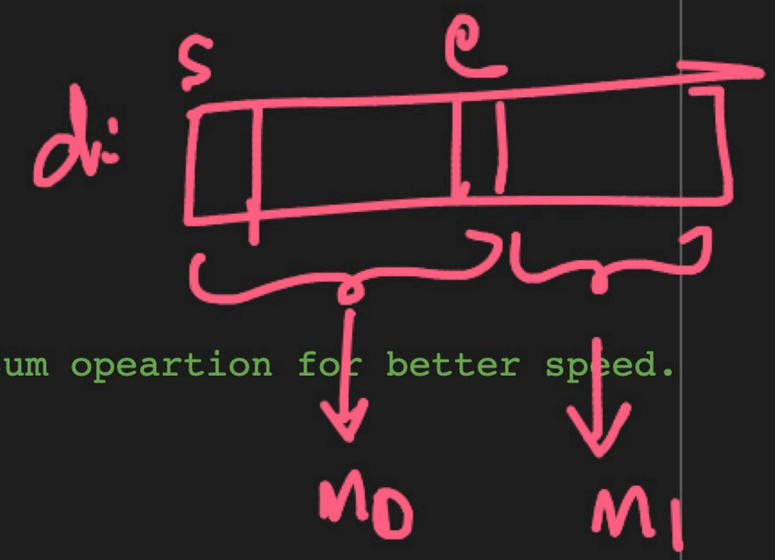
+ Code + Text Connect

```
#Refer: https://docs.python.org/3/library/threading.html
from threading import Thread

means = [0,0];

def mean_MT(s, e, threadNum ):

  #Sum using for loops. We can use inbuilt NumPy Sum opeartion for better speed.
  sum = 0
  for i in range(s,e+1):
    sum +=d[i]

  #Mean
  mean = sum/(e-s+1)
  means[threadNum] = mean; # means is a shared varibale between the threads

  return

n1 = math.floor(n/2)

t1 = Thread(target=mean_MT, args=(0, n1,0 ))  # Third apram is the thread number
t2 = Thread(target=mean_MT, args=(n1+1,n-1,1))
```

17 / 17

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:07
Copy of Multithreading_i | Find out how many cores | Multithreading (compute | multiprocessing - What | Numba: A High Performa
colab.research.google.com/drive/1n5JkAWDr_VRCQjlr6BVWdDpUgW4eZODr#scrollTo=ipbV9r5YDFZz
+ Code + Text Connect

```python
#Refer: https://docs.python.org/3/library/threading.html
from threading import Thread

means = [0,0];

def mean_MT(s, e, threadNum ):

  #Sum using for loops. We can use inbuilt NumPy Sum opeartion for better speed.
  sum = 0
  for i in range(s,e+1):
    sum +=d[i]

  #Mean
  mean = sum/(e-s+1)
  means[threadNum] = mean; # means is a shared varibale between the threads

  return

n1 = math.floor(n/2)

t1 = Thread(target=mean_MT, args=(0, n1,0 ))  # Third apram is the thread number
t2 = Thread(target=mean_MT, args=(n1+1,n-1,1))
```



18 / 18

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:08
Copy of Multithreading_i | Find out how many cores | Multithreading (compute | multiprocessing - What | Numba: A High Performa
colab.research.google.com/drive/1n5JkAWDr_VRCQjlr6BVWdDpUgW4eZODr#scrollTo=ipbV9r5YDFZz
Update
+ Code + Text
Connect

```
    sum +=d[i]

  #Mean
  mean = sum/(e-s+1)
  means[threadNum] = mean; # means is a shared varibale between the threads

  return

n1 = math.floor(n/2)

t1 = Thread(target=mean_MT, args=(0, n1,0 ))  # Third apram is the thread number
t2 = Thread(target=mean_MT, args=(n1+1,n-1,1))

#Time the execution
start_time = time.time()

t1.start()
t2.start()

t1.join() # Wait till t1 finishes
t2.join()

m = (means[0]+means[1])/2

end_time = time.time()
```

19 / 19

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:08
Copy of Multithreading_i | Find out how many cores | Multithreading (compute | multiprocessing - What | Numba: A High Performa
colab.research.google.com/drive/1n5JkAWDr_VRCQjlr6BVWdDpUgW4eZODr#scrollTo=ipbV9r5YDFZz
Update
+ Code + Text
Connect

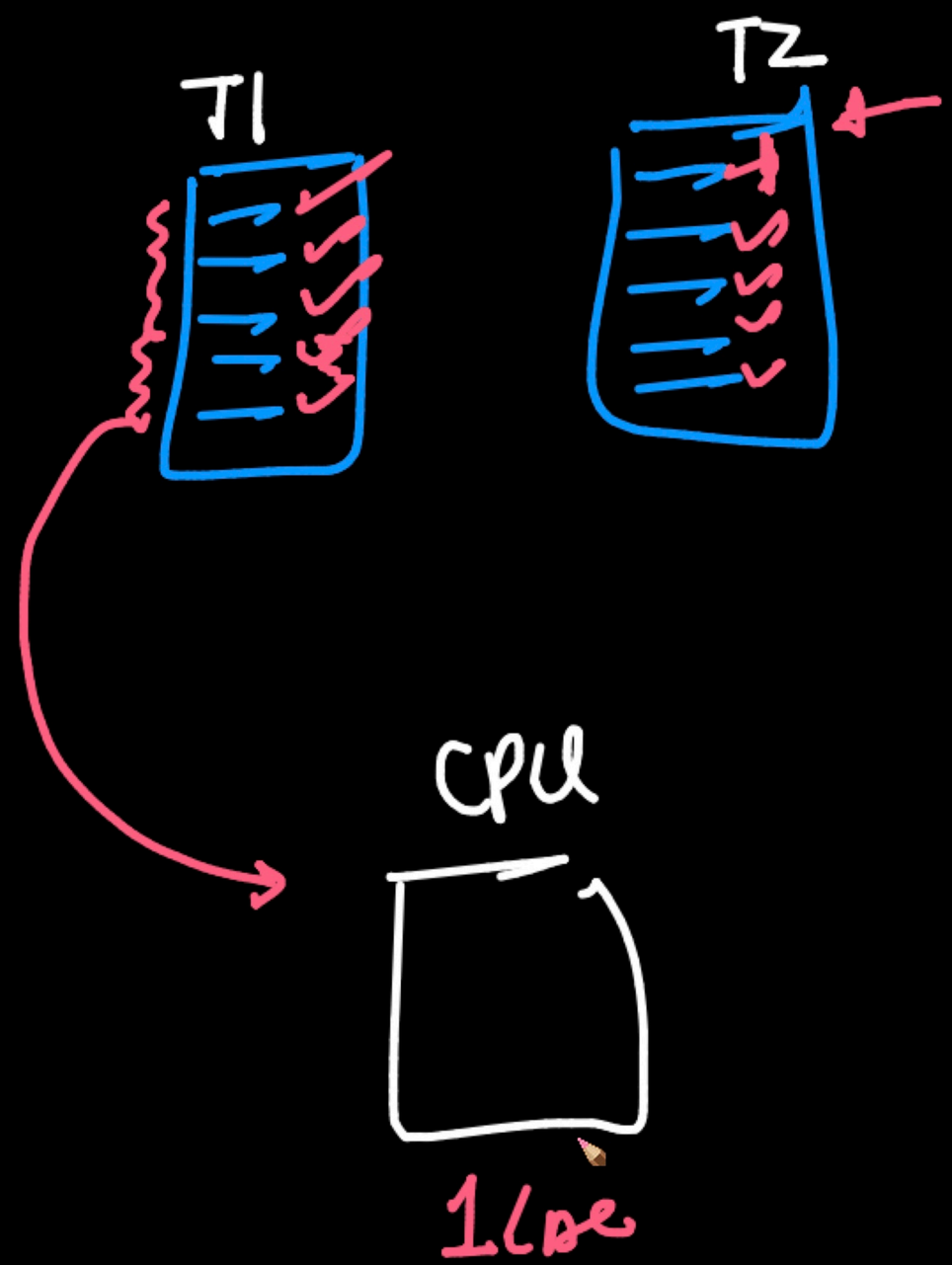
```
    sum +=d[i]

  #Mean
  mean = sum/(e-s+1)
  means[threadNum] = mean; # means is a shared varibale between the threads

  return

n1 = math.floor(n/2)

t1 = Thread(target=mean_MT, args=(0, n1,0 ))  # Third apram is the thread number
t2 = Thread(target=mean_MT, args=(n1+1,n-1,1))

#Time the execution
start_time = time.time()

t1.start()
t2.start()

t1.join() # Wait till t1 finishes
t2.join()

m = (means[0]+means[1])/2

end_time = time.time()
```

s e M[0]

s e M[1]

20 / 20

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:12



21 / 23

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:13

1 core → 21 sec

2 cores → 11 sec

1 core + 2 threads → 19.55 sec

≈

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:27

Copy of Multithreading_i | Find out how many cores | Multithreading (compute | multiprocessing - What | Numba: A High Performa

colab.research.google.com/drive/1n5JkAWDr_VRCQjlr6BVWdDpUgW4eZODr#scrollTo=ipbV9r5YDFZz

Update

+ Code + Text Connect

Joblib

Caching of function output values

[ ] ↳ 2 cells hidden

median → sort:

→ middle

Simple Parallel programming for Loops

[ ] ↳ 4 cells hidden

std-dev

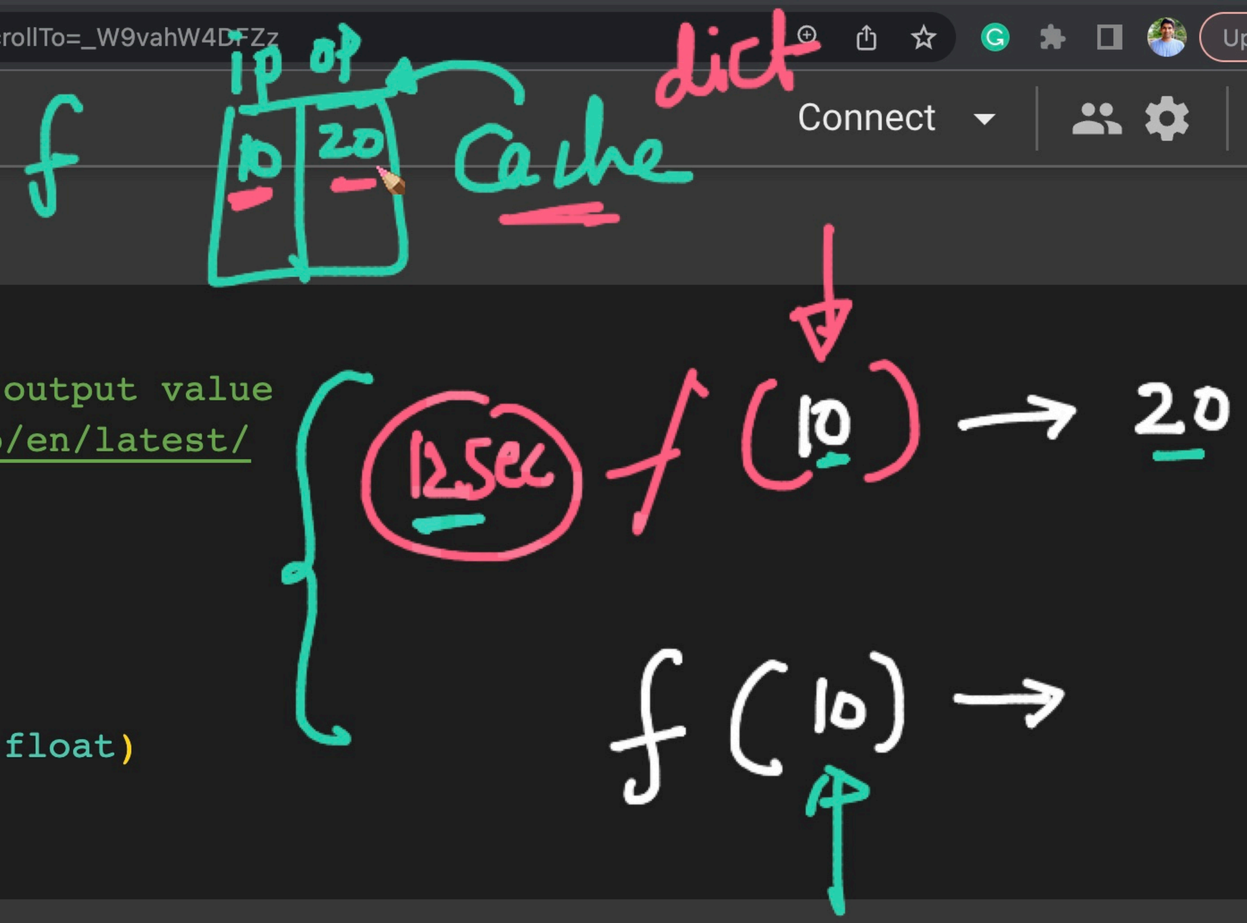
$$\frac{(x_i - m)^2}{n}$$

Numba: https://numba.pydata.org/

23 / 23

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:32
Copy of Multithreading_i | Find out how many cores | Multithreading (compute | multiprocessing - What | Numba: A High Performa
colab.research.google.com/drive/1n5JkAWDr_VRCQjlr6BVWdDpUgW4eZODr#scrollTo=_W9vahW4DFZz
+ Code + Text Connect

Caching of function output values




```
[ ]
#Transparent and fast disk-caching of output value
# Refer: https://joblib.readthedocs.io/en/latest/
from joblib import Memory
cachedir = './'
mem = Memory(cachedir)

import numpy as np
a = np.vander(np.arange(3)).astype(np.float)
square = mem.cache(np.square)
b = square(a)
```

```
[Memory] Calling square...
square(array([[0., 0., 1.],
       [1., 1., 1.],
       [4., 2., 1.]]))
_____________________________________________square - 0.0s, 0.0min
<ipython-input-1-5aa631b74919>:8: DeprecationWarning: `np.float` is a deprecated alias for the built
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-nc
  a = np.vander(np.arange(3)).astype(np.float)
```

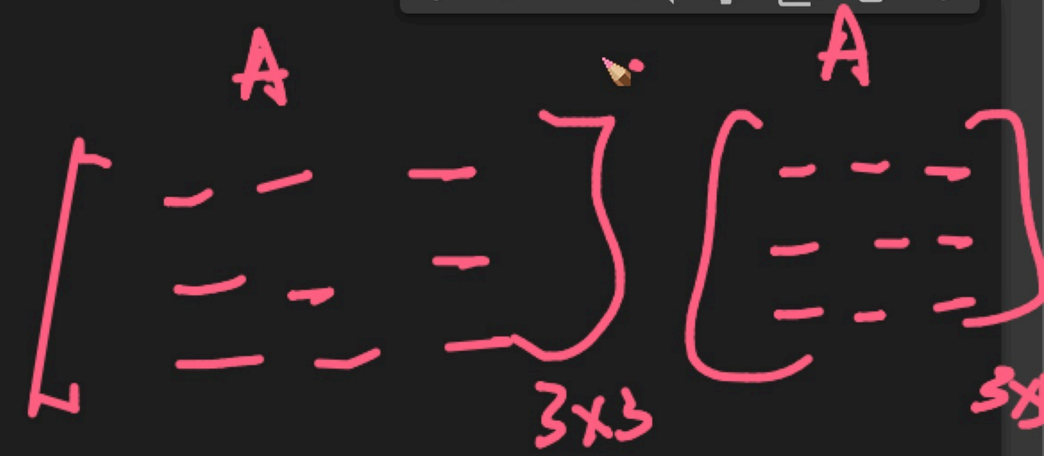
Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:34

Copy of Multithread | Find out how many | Multithreading (con | multiprocessing - W | Numba: A High Perf | numpy.vander — Nu

colab.research.google.com/drive/1n5JkAWDr_VRCQjlr6BVWdDpUgW4eZODr#scrollTo=5s3HE97FDFZ0

+ Code + Text Connect

```
#Transparent and fast disk-caching of output value
# Refer: https://joblib.readthedocs.io/en/latest/
from joblib import Memory
cachedir = './'
mem = Memory(cachedir)

import numpy as np
a = np.vander(np.arange(3)).astype(np.float)
square = mem.cache(np.square)
b = square(a)
```




```
_____________________________________________________________________________
[Memory] Calling square...
square(array([[0., 0., 1.],
       [1., 1., 1.],
       [4., 2., 1.]]))
_____________________________________________square - 0.0s, 0.0min
<ipython-input-1-5aa631b74919>:8: DeprecationWarning: `np.float` is a deprecated alias for the built
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-nc
  a = np.vander(np.arange(3)).astype(np.float)
```

```
c = square(a)
```

25 / 25

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:34

Copy of Multithread | Find out how many | Multithreading (con | multiprocessing - W | Numba: A High Perf | numpy.vander — Nu

colab.research.google.com/drive/1n5JkAWDr_VRCQjlr6BVWdDpUgW4eZODr#scrollTo=5s3HE97FDFZ0

+ Code + Text Connect

```
#Transparent and fast disk-caching of output value
# Refer: https://joblib.readthedocs.io/en/latest/
from joblib import Memory
cachedir = './'
mem = Memory(cachedir)

import numpy as np
a = np.vander(np.arange(3)).astype(np.float)
square = mem.cache(np.square)
b = square(a)
```

```
________________________________________________________________________________
[Memory] Calling square...
square(array([[0., 0., 1.],
       [1., 1., 1.],
       [4., 2., 1.]]))
_____________________________________________________________square - 0.0s, 0.0min
<ipython-input-1-5aa631b74919>:8: DeprecationWarning: `np.float` is a deprecated alias for the built
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-nc
  a = np.vander(np.arange(3)).astype(np.float)
```

```
c = square(a)
```

26 / 26

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:37

Copy of Multithread | Find out how many | Multithreading (con | multiprocessing - W | Numba: A High Perf | numpy.vander — Nu

colab.research.google.com/drive/1n5JkAWDr_VRCQjlr6BVWdDpUgW4eZODr#scrollTo=sl-n_-HBDFZ1

+ Code + Text Connect

```python
# Refer: https://joblib.readthedocs.io/en/latest/generated/joblib.Parallel.html

import time
from math import sqrt # inbuilt fucntion

def f(i):

    # some computations  that take time
    x=10000
    p =1;
    for j in range(x):
        for k in range(j):
            p *= k

    return sqrt(i ** 2);

# Find f(x) of first n numbers
n=10;

start_time = time.time()

for i in range(n):
    f(i)

```

27 / 27

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:37

Copy of Multithread | Find out how many | Multithreading (con | multiprocessing - W | Numba: A High Perf | numpy.vander — Nu

colab.research.google.com/drive/1n5JkAWDr_VRCQjlr6BVWdDpUgW4eZODr#scrollTo=sl-n_-HBDFZ1

+ Code + Text Connect

```python
# Refer: https://joblib.readthedocs.io/en/latest/generated/joblib.Parallel.html

import time
from math import sqrt # inbuilt fucntion

def f(i):

    # some computations  that take time
    x=10000
    p =1;
    for j in range(x):
        for k in range(j):
            p *= k

    return sqrt(i ** 2);

# Find f(x) of first n numbers
n=10;

start_time = time.time()

for i in range(n):
    f(i)

```

28 / 28

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:38

Copy of Multithread | Find out how many | Multithreading (con | multiprocessing - W | Numba: A High Perf | numpy.vander — Nu

colab.research.google.com/drive/1n5JkAWDr_VRCQjlr6BVWdDpUgW4eZODr#scrollTo=sl-n_-HBDFZ1

+ Code + Text Connect

```
def f(i):

    # some computations  that take time
    x=10000
    p =1;
    for j in range(x):
        for k in range(j):
            p *= k

    return sqrt(i ** 2);

# Find f(x) of first n numbers
n=10;

start_time = time.time()

for i in range(n):
    f(i)

end_time = time.time()
print (end_time-start_time)
```

f(i)

1,2,3, . 9

```
17.116321802139282
```

+ Code + Text

29 / 29

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:41

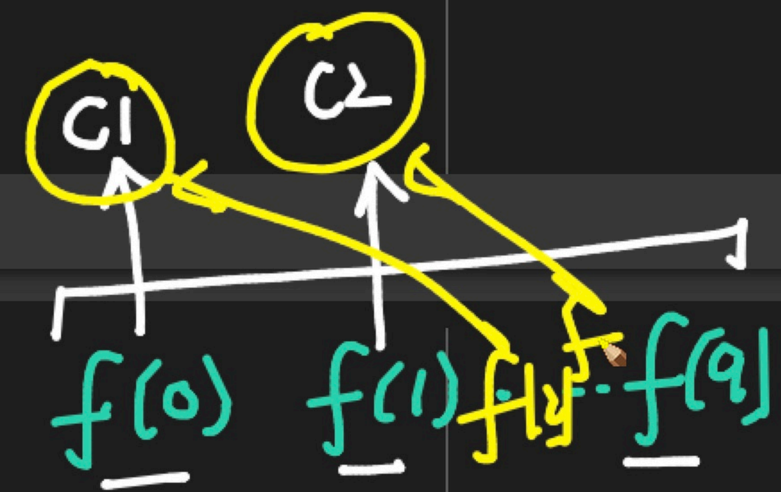
Copy of Multithread | Find out how many | Multithreading (con | multiprocessing - W | Numba: A High Perf | numpy.vander — Nu

colab.research.google.com/drive/1n5JkAWDr_VRCQjlr6BVWdDpUgW4eZODr#scrollTo=sl-n_-HBDFZ1

+ Code + Text Connect

```
for i in range(n):
    f(i)

end_time = time.time()
print (end_time-start_time)
```

```
17.116321802139282
```




```
from joblib import Parallel, delayed

start_time = time.time()

a = Parallel(n_jobs=2)(delayed(f)(i) for i in range(n))

# Why we need dealyed(): https://stackoverflow.com/questions/42220458/what-does-the-delayed-func

end_time = time.time()
print (end_time-start_time)
```

```
9.58085012435913
```

```
# Multi threading
start_time = time.time()
```

30 / 31

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:42

Copy of Multithread | Find out how many | Multithreading (con | multiprocessing - W | Numba: A High Perf | numpy.vander — Nu

colab.research.google.com/drive/1n5JkAWDr_VRCQjlr6BVWdDpUgW4eZODr#scrollTo=sl-n_-HBDFZ1

+ Code + Text Connect

```
for i in range(n):
    f(i)

end_time = time.time()
print (end_time-start_time)
```

```
17.116321802139282
```

1 core → 17.11 sec

2 cores → 9.58 sec

```
from joblib import Parallel, delayed

start_time = time.time()

a = Parallel(n_jobs=2)(delayed(f)(i) for i in range(n))

# Why we need dealyed(): https://stackoverflow.com/questions/42220458/what-does-the-delayed-func

end_time = time.time()
print (end_time-start_time)
```

```
9.58085012435913
```

```
# Multi threading
start_time = time.time()
```

31 / 32

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:43

Copy of Multithread | Find out how many | Multithreading (con | multiprocessing - W | Numba: A High Perf | numpy.vander — Nu

colab.research.google.com/drive/1n5JkAWDr_VRCQjlr6BVWdDpUgW4eZODr#scrollTo=sl-n_-HBDFZ1

+ Code + Text Connect

```
from joblib import Parallel, delayed

start_time = time.time()

a = Parallel(n_jobs=2)(delayed(f)(i) for i in range(n))

# Why we need dealyed(): https://stackoverflow.com/questions/42220458/what-does-the-delayed-func

end_time = time.time()
print (end_time-start_time)
```

```
9.58085012435913
```

```
# Multi threading
start_time = time.time()

a = Parallel(n_jobs=2,prefer="threads")(delayed(f)(i ** 2) for i in range(n))

end_time = time.time()
print (end_time-start_time)
```

```
17.342177867889404
```

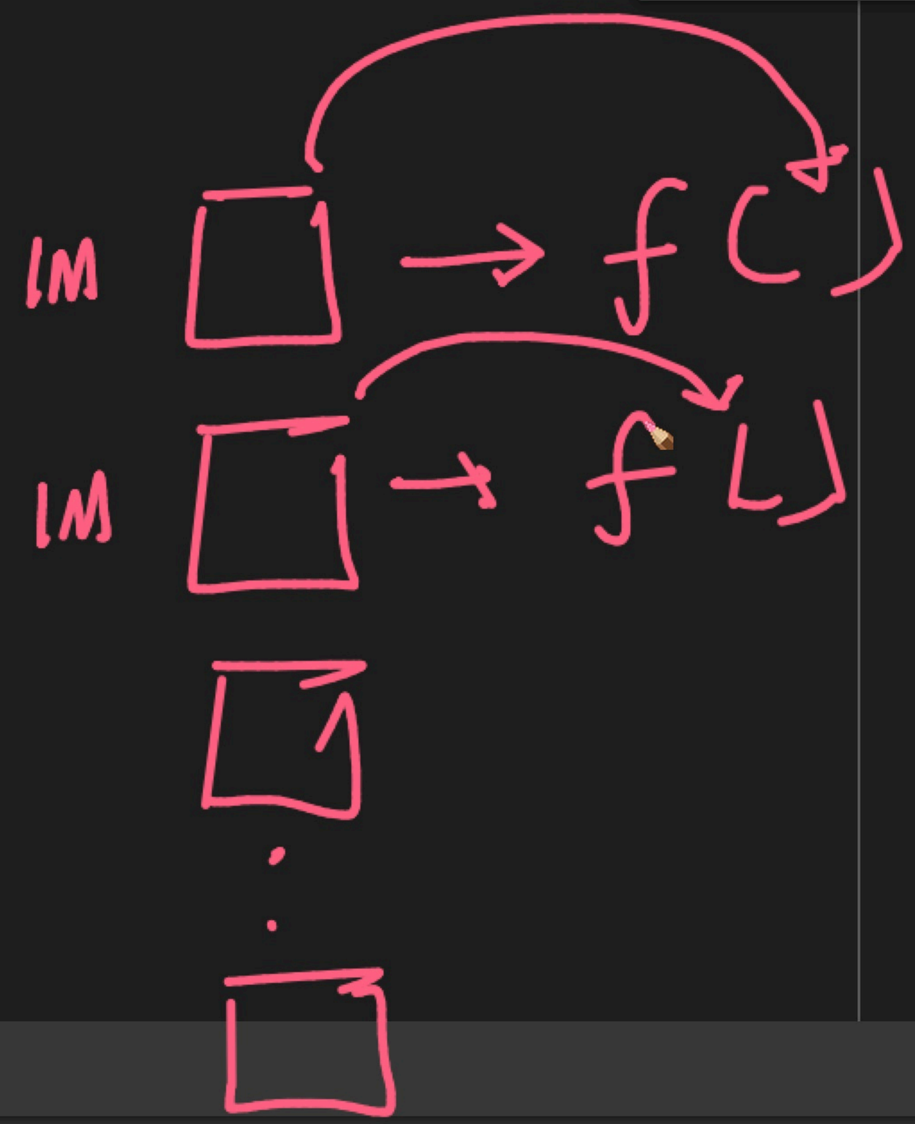
```
1
```

32 / 33

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:44
Copy of Multithread | Find out how many | Multithreading (con | multiprocessing - W | Numba: A High Perf | numpy.vander — Nu
colab.research.google.com/drive/1n5JkAWDr_VRCQjlr6BVWdDpUgW4eZODr#scrollTo=sl-n_-HBDFZ1
Update
+ Code + Text
Connect

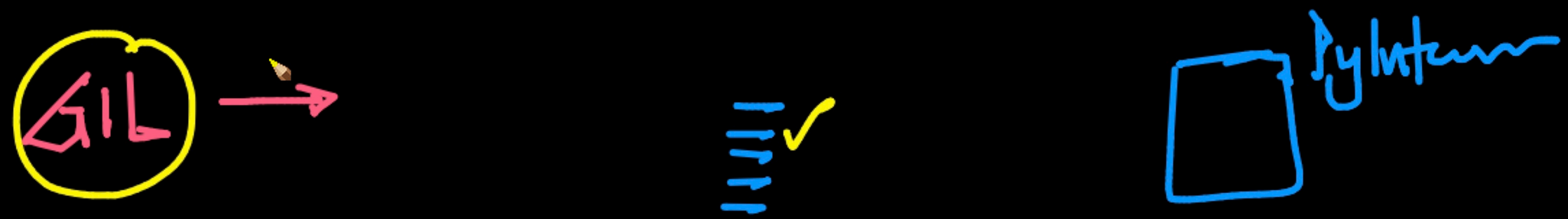
```python
def f(i):

    # some computations  that take time
    x=10000
    p =1;
    for j in range(x):
        for k in range(j):
            p *= k

    return sqrt(i ** 2);

# Find f(x) of first n numbers
n=10;

start_time = time.time()

for i in range(n):
    f(i)

end_time = time.time()
print (end_time-start_time)
```

```
17.116321802139282
```



33 / 34

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:47

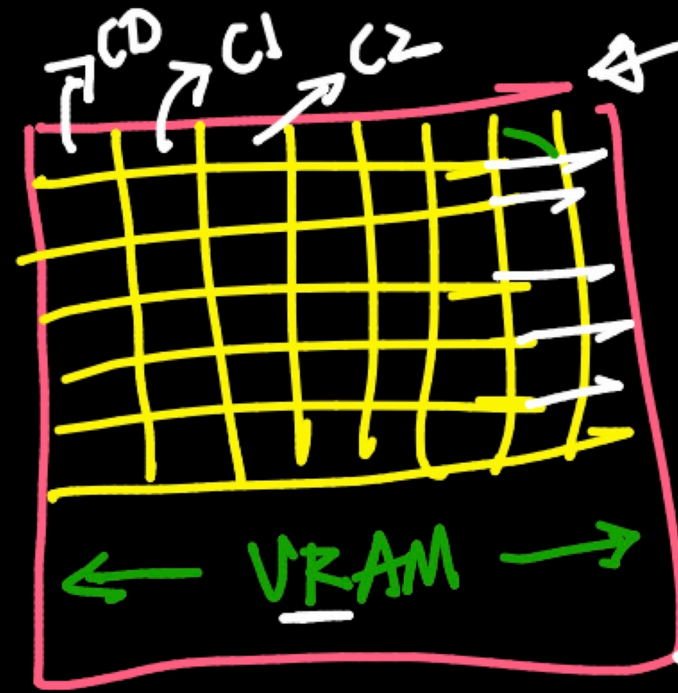


34 / 34

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:52

GPU (Nvidia)

Matrix operation



Multi-processing

1000's of cores

800 MHz

8 GB

RAM

CPU → 8 Cores : 3.2 GHz

↓

RAM

Copy of Multithreadi...

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
17.342177867889404
```

```python
# 6 jobs

from joblib import Parallel, delayed

start_time = time.time()

a = Parallel(n_jobs=6)(delayed(f)(i ** 2) for i in range(n))

# Why we need dealyed(): https://stackoverflow.com/questions/42220458/what-does-the-delayed-func

end_time = time.time()
print (end_time-start_time)
```

```
4.557589054107666
```