004946-bdcc2324-py-multiprocessing

April 25, 2024

[Big Data and Cloud Computing]

1 Daniela Tomás, up202004946

2 Parallel programming in python3 using the multiprocessing module

Spark and several other modules in python give you tools that can automatically distribute and map threads or processes across processors and disks (using single or multiple machines), but very often it is necessary to have more control over your parallel tasks, understanding the backend implementation or implementing your own parallel code. In this class, we will use some alternatives to program in parallel using the `multiprocessing` module of python and take advantage of your multicore machine. Be aware that this does not work with ***threads***, but with ***processes***.

Material for these practical exercises was taken from this site.

Every piece of code is timed in order that you have an idea of how execution time differs among the choices for parallelization.

Note: When using the method `pool.apply_async` the function to be invoked may need to be defined in a separate file and be imported to the program, otherwise your code may not work.

References:

- python3 multiprocessing
- Programming Guidelines for multiprocessing
- Timing and profiling using colab (You may find this useful)
- Differences between pool.apply and others

Let's start with the basics: importing the relevant python3 module...(remember: we will be working with processes and not with threads)

[ ]:
```
import multiprocessing as mp
```

For this exercise we will need some extra modules.

[ ]:
```
import numpy as np
from time import time
```

We will also create some synthetic random data (a `numpy` array) but you can use your own data. Notice that I reduced the data dimension in order that you can better understand the sequence of

results produced by the sequence of operations executed (sequentially and in parallel).

[ ]:
```
# Prepare data
# value range
r = 10
# number of rows
m = 1000
#m = 5
# number of columns
n = 10000
#n = 10

np.random.seed(100)
arr = np.random.randint(0, r, size=[m, n])
data = arr.tolist()
print(data[:10])
```

Now, let's write a very simple sequential program that defines a function to count the number of values of a row of the array created above that falls in a given interval.

[ ]:
```
# import this only to have access to process number/id
import multiprocessing as mp

# Sequential code: Solution Without Parallelization

def howmany_within_range(row, minimum, maximum):
    """Returns how many numbers lie within `maximum` and `minimum` in a given␣
 ↪`row`"""
    #print(mp.current_process(),' ',row)
    count = 0
    for num in row:
        if minimum <= num <= maximum:
            count = count + 1
    return count

# begin timing
start_time = time()

results = []
for row in data:
    results.append(howmany_within_range(row, minimum=4, maximum=8))

# end timing
print(round(time() - start_time,8),'seconds')

print(results[:10])

# m = 5 n = 10:
```

```
# 0.00017881 seconds
# [6, 3, 5, 4, 4]
```

```
0.94633937 seconds
[5062, 4978, 4943, 5078, 5016, 5064, 5006, 5011, 4977, 4903]
```

Let's check some characteristics of our machine.

[ ]:
```
!lsb_release -a
```

```
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 22.04.3 LTS
Release:        22.04
Codename:       jammy
```

[ ]:
```
!uname -a
```

```
Linux 5db3858b5a36 6.1.58+ #1 SMP PREEMPT_DYNAMIC Sat Nov 18 15:31:17 UTC 2023
x86_64 x86_64 x86_64 GNU/Linux
```

[ ]:
```
!cat /proc/cpuinfo
```

```
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 79
model name      : Intel(R) Xeon(R) CPU @ 2.20GHz
stepping        : 0
microcode       : 0xffffffff
cpu MHz         : 2199.998
cache size      : 56320 KB
physical id     : 0
siblings        : 2
core id         : 0
cpu cores       : 1
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
pat pse36 clflush mmx fxsr sse sse2 ss ht syscall nx pdpe1gb rdtscp lm
constant_tsc rep_good nopl xtopology nonstop_tsc cpuid tsc_known_freq pni
pclmulqdq ssse3 fma cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt aes xsave avx
f16c rdrand hypervisor lahf_lm abm 3dnowprefetch invpcid_single ssbd ibrs ibpb
stibp fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms invpcid rtm rdseed adx
smap xsaveopt arat md_clear arch_capabilities
```

```
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds
swapgs taa mmio_stale_data retbleed
bogomips        : 4399.99
clflush size    : 64
cache_alignment : 64
address sizes   : 46 bits physical, 48 bits virtual
power management:

processor       : 1
vendor_id       : GenuineIntel
cpu family      : 6
model           : 79
model name      : Intel(R) Xeon(R) CPU @ 2.20GHz
stepping        : 0
microcode       : 0xffffffff
cpu MHz         : 2199.998
cache size      : 56320 KB
physical id     : 0
siblings        : 2
core id         : 0
cpu cores       : 1
apicid          : 1
initial apicid  : 1
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
pat pse36 clflush mmx fxsr sse sse2 ss ht syscall nx pdpe1gb rdtscp lm
constant_tsc rep_good nopl xtopology nonstop_tsc cpuid tsc_known_freq pni
pclmulqdq ssse3 fma cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt aes xsave avx
f16c rdrand hypervisor lahf_lm abm 3dnowprefetch invpcid_single ssbd ibrs ibpb
stibp fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms invpcid rtm rdseed adx
smap xsaveopt arat md_clear arch_capabilities
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds
swapgs taa mmio_stale_data retbleed
bogomips        : 4399.99
clflush size    : 64
cache_alignment : 64
address sizes   : 46 bits physical, 48 bits virtual
power management:
```

In what follows, we will be using first the `Pool` class. Next, we will use the `Process` class. When using `Pool`, threads or processes get launched as soon as the Pool is initialized (it happens in Pool___init___() - there is no need to submit tasks for this to happen) and wait for tasks. When a task arrives and is executed, threads or processes **do not exit**, they just go back to waiting state waiting for more work to come.

You can define it to work differently, though. You can add the `maxtasksperchild` parameter to your pool. As soon as a worker completes this amount of tasks, it exits, and a new worker is immediately launched (no need to give it a task first, it gets launched as soon as a worker exits). This is managed in the `Pool` class `Pool._maintain_pool()` and `Pool._repopulate_pool()` functions in the source code. `Pool` can use several different methods to distribute tasks. We will see some of them next.

In order to help parallelizing the code we will be using some mapping functions: `apply`, `map`, `starmap`, `apply_async` etc.

Let's parallelize our program that counts values within a range.

3 Option #1: using `pool.apply`

[ ]:
```
# Parallelizing using Pool.apply()

import multiprocessing as mp

# Is this actually running with multiple cpus?
num_cpus = mp.cpu_count()
print('Num cpus = ', num_cpus)

# begin timing
start_time = time()

pool = mp.Pool(mp.cpu_count())
# end timing init
print('Time to create pool: ',round(time() - start_time,8), 'seconds')

# Step 2: `pool.apply` the `howmany_within_range()`
results = [pool.apply(howmany_within_range, args=(row, 4, 8)) for row in data]

# Step 3: Don't forget to close
pool.close()

# end timing
print('Total time: ',round(time() - start_time,8), 'seconds')

print(results[:10])

# m = 5 n = 10:
# Num cpus =  2
# Time to create pool:  0.02744389 seconds
# Total time:  0.03518176 seconds
# [6, 3, 5, 4, 4]
```

```
Num cpus =  2
Time to create pool:  0.07190919 seconds
```

```
Total time:  3.0060575 seconds
[5062, 4978, 4943, 5078, 5016, 5064, 5006, 5011, 4977, 4903]
```

4 Q1: Is your parallel program slower than the sequential? Why?

The parallel program using `pool.apply` is slower than the sequential program because it is synchronous and incurs the overhead of starting and managing processes. To be efficient, parallelism requires bigger data dimensions and more complex tasks.

Let's parallelize this program using an alternative function.

5 Option #2: using `pool.map`

[ ]:
```python
# Parallelizing using Pool.map()
import multiprocessing as mp

# Redefine, with only 1 mandatory argument.
def howmany_within_range_rowonly(row, minimum=4, maximum=8):
  #print(mp.current_process(),' ',row) # this will print the process object and␣
 ↪the item it is working with
  count = 0
  for num in row:
     if minimum <= num <= maximum:
        count = count + 1
  return count

# begin timing
start_time = time()

pool = mp.Pool(mp.cpu_count())

# begin timing
start_time = time()

pool = mp.Pool(mp.cpu_count())
results = pool.map(howmany_within_range_rowonly, [row for row in data])

pool.close()

# end timing
print(round(time() - start_time,8), 'seconds')

print(results[:10])

# m = 5 n = 10:
# 0.03856659 seconds
# [6, 3, 5, 4, 4]
```

```
1.30773854 seconds
[5062, 4978, 4943, 5078, 5016, 5064, 5006, 5011, 4977, 4903]
```

6 Q2: What is the difference between Option #1 and Option #2? In other words, what is the difference between `apply` and `map`? Which one is slower? Why?

The difference between Option #1 (`pool.apply`) and Option #2 (`pool.map`) is in the way tasks are divided and executed across worker processes.

`apply` issues one task to a worker process, and the main process blocks until each task is complete.

`map` issues multiple tasks to the process pool simultaneously. The tasks are divided among the available worker processes, and the main process blocks until all tasks are completed.

In general, `apply` is slower because `map` provides efficient parallelization. However, in this case, due to the reduced data size and task simplicity, they have similar times.

7 Q3: Try increasing the dimension of your data. Do you see any improvement in performance or not? Why?

As we have increased the data dimension but not the complexity of the tasks, there is no improvement in performance between parallel and sequential programs. Where we might notice differences is between `apply` and `map`, where `map` seems slightly faster.

8 Option #3: using `pool.starmap`

[ ]:
```python
# Parallelizing with Pool.starmap()
import multiprocessing as mp

# begin timing
start_time = time()

pool = mp.Pool(mp.cpu_count())

results = pool.starmap(howmany_within_range, [(row, 4, 8) for row in data])

pool.close()

# end timing
print(round(time() - start_time,8), 'seconds')

print(results[:10])
```

```
1.26990151 seconds
[5062, 4978, 4943, 5078, 5016, 5064, 5006, 5011, 4977, 4903]
```

9 Option #4: using `pool.apply_async`

Let's try a little bit different parallelization approach where we let processes run asynchronously.

[ ]:
```
# Parallel processing with Pool.apply_async()

import multiprocessing as mp

# begin timing
start_time = time()

pool = mp.Pool(mp.cpu_count())

results = []

# Step 1: Redefine, to accept `i`, the iteration number
def howmany_within_range2(i, row, minimum, maximum):
    """Returns how many numbers lie within `maximum` and `minimum` in a given␣
 ↪`row`"""
    count = 0
    for num in row:
        if minimum <= num <= maximum:
            count = count + 1
 #  print(str(i) + ' ' + count)
    return (i, count)

# Step 2: Define callback function to collect the output in `results`
def collect_result(result):
    global results
    #print(result)
    results.append(result)

# Step 3: Use loop to parallelize
for i, row in enumerate(data):
    pool.apply_async(howmany_within_range2, args=(i, row, 4, 8, ),␣
 ↪callback=collect_result)

# Step 4: Close Pool and wait for all processes to complete
pool.close()
pool.join()  # postpones the execution of next line of code until all processes␣
 ↪in the queue are done.

# end timing
print(round(time() - start_time,8), 'seconds')

# Step 5: Sort results [OPTIONAL]
results.sort(key=lambda x: x[0])
```

```
results_final = [r for i, r in results]

print(results_final[:10])
```

```
Process ForkPoolWorker-15:
Process ForkPoolWorker-16:
Traceback (most recent call last):
  File "/usr/lib/python3.10/multiprocessing/process.py", line 314, in _bootstrap
    self.run()
Traceback (most recent call last):
  File "/usr/lib/python3.10/multiprocessing/process.py", line 108, in run
    self._target(*self._args, **self._kwargs)
  File "/usr/lib/python3.10/multiprocessing/process.py", line 314, in _bootstrap
    self.run()
  File "/usr/lib/python3.10/multiprocessing/pool.py", line 114, in worker
    task = get()
  File "/usr/lib/python3.10/multiprocessing/process.py", line 108, in run
    self._target(*self._args, **self._kwargs)
  File "/usr/lib/python3.10/multiprocessing/queues.py", line 367, in get
    return _ForkingPickler.loads(res)
  File "/usr/lib/python3.10/multiprocessing/pool.py", line 114, in worker
    task = get()
AttributeError: Can't get attribute 'howmany_within_range2' on <module
'__main__'>
  File "/usr/lib/python3.10/multiprocessing/queues.py", line 367, in get
    return _ForkingPickler.loads(res)
AttributeError: Can't get attribute 'howmany_within_range2' on <module
'__main__'>
Process ForkPoolWorker-18:
Process ForkPoolWorker-17:
Traceback (most recent call last):
  File "/usr/lib/python3.10/multiprocessing/process.py", line 314, in _bootstrap
    self.run()
Traceback (most recent call last):
  File "/usr/lib/python3.10/multiprocessing/process.py", line 314, in _bootstrap
    self.run()
  File "/usr/lib/python3.10/multiprocessing/process.py", line 108, in run
    self._target(*self._args, **self._kwargs)
  File "/usr/lib/python3.10/multiprocessing/pool.py", line 114, in worker
    task = get()
  File "/usr/lib/python3.10/multiprocessing/process.py", line 108, in run
    self._target(*self._args, **self._kwargs)
  File "/usr/lib/python3.10/multiprocessing/pool.py", line 114, in worker
    task = get()
  File "/usr/lib/python3.10/multiprocessing/queues.py", line 365, in get
    res = self._reader.recv_bytes()
  File "/usr/lib/python3.10/multiprocessing/queues.py", line 364, in get
```

```
    with self._rlock:
  File "/usr/lib/python3.10/multiprocessing/connection.py", line 216, in
recv_bytes
    buf = self._recv_bytes(maxlength)
  File "/usr/lib/python3.10/multiprocessing/connection.py", line 414, in
_recv_bytes
    buf = self._recv(4)
  File "/usr/lib/python3.10/multiprocessing/connection.py", line 379, in _recv
    chunk = read(handle, remaining)
KeyboardInterrupt
  File "/usr/lib/python3.10/multiprocessing/synchronize.py", line 95, in
__enter__
    return self._semlock.__enter__()
KeyboardInterrupt
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-40-7849d349cf74> in <cell line: 34>()
     32 # Step 4: Close Pool and wait for all processes to complete
     33 pool.close()
---> 34 pool.join()  # postpones the execution of next line of code until all␣
 ↪processes in the queue are done.
     35
     36 # end timing

/usr/lib/python3.10/multiprocessing/pool.py in join(self)
    663         elif self._state not in (CLOSE, TERMINATE):
    664             raise ValueError("In unknown state")
--> 665         self._worker_handler.join()
    666         self._task_handler.join()
    667         self._result_handler.join()

/usr/lib/python3.10/threading.py in join(self, timeout)
   1094
   1095         if timeout is None:
-> 1096             self._wait_for_tstate_lock()
   1097         else:
   1098             # the behavior of a negative timeout isn't documented, but

/usr/lib/python3.10/threading.py in _wait_for_tstate_lock(self, block, timeout)
   1114
   1115         try:
-> 1116             if lock.acquire(block, timeout):
   1117                 lock.release()
   1118                 self._stop()
```

```
KeyboardInterrupt:
```

This does not run! The reason is because when you use the method `pool.apply_async` the function to be invoked needs to be defined in a separate file and be imported to the program, otherwise your code will not work (**why??**). Let's do this. (One way of doing it is to upload the file with the function `howmany_within_range2` to your drive, and then copying it to this colab machine. I created a file called `howmany.py`, uploaded it to my drive and copied to my current directory at the colab machine, as shown next).

[ ]:
```
!cp "/content/drive/MyDrive/Colab Notebooks/howmany2.py" .
!ls -la
```

```
total 28
drwxr-xr-x 1 root root 4096 Apr 25 14:26 .
drwxr-xr-x 1 root root 4096 Apr 25 13:44 ..
drwxr-xr-x 4 root root 4096 Apr 23 13:22 .config
drwx------ 5 root root 4096 Apr 25 14:23 drive
-rw------- 1 root root  293 Apr 25 14:38 howmany2.py
drwxr-xr-x 2 root root 4096 Apr 25 14:28 __pycache__
drwxr-xr-x 1 root root 4096 Apr 23 13:23 sample_data
```

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

Corrected `pool.async`, where the function goes in to a separate file.

[ ]:
```
# Parallel processing with Pool.apply_async()

import multiprocessing as mp

pool = mp.Pool(mp.cpu_count())

results = []

# Step 1: function howmany... is defined in another file
from howmany2 import *

# Step 2: Define callback function to collect the output in `results`
def collect_result(item):
    #print(item)
    return item

# begin timing
start_time = time()
```

```
results = []
# Step 3: Use loop to parallelize
for i, row in enumerate(data):
    result = pool.apply_async(howmany_within_range2, args=(i, row, 4, 8, ),␣
 ↪callback=collect_result).get()
    results.append(result)

# Step 4: Close Pool and wait for all processes to complete
pool.close()
pool.join()  # postpones the execution of next line of code until all processes␣
 ↪in the queue are done.

# end timing
print(round(time() - start_time,8), 'seconds')

# Step 5: Sort results [OPTIONAL]
#results.sort(key=lambda x: x[0])
#results_final = [r for i, r in results]

#print(results_final[:10])
print(results[:10])
```

```
1.86734796 seconds
[(0, 5062), (1, 4978), (2, 4943), (3, 5078), (4, 5016), (5, 5064), (6, 5006),
(7, 5011), (8, 4977), (9, 4903)]
```

[ ]:
```
# Parallel processing with Pool.apply_async() same model and comp pattern as␣
 ↪the others
# apply_async does not work like apply in parallel
# if implemented like here, without a loop to spawn multiple tasks, it does not␣
 ↪run

import multiprocessing as mp

pool = mp.Pool(mp.cpu_count())

results = []

# Step 1: function howmany... is defined in another file
from howmany2 import *

# Step 2: Define callback function to collect the output in `results`
def collect_result(result):
    global results
    print(result)
    results.append(result)
#    return results
```

```
# begin timing
start_time = time()

# Step 3: Use loop to parallelize
# for i, row in enumerate(data):
#    pool.apply_async(howmany.howmany_within_range2, args=(i, row, 4, 8, ),␣
 ↪callback=collect_result)

results_final = [pool.apply_async(howmany_within_range2, args = (row, 4, 8))␣
 ↪for row in data]

# Step 4: Close Pool and wait for all processes to complete
pool.close()
pool.join()  # postpones the execution of next line of code until all processes␣
 ↪in the queue are done.

# end timing
print(round(time() - start_time,8), 'seconds')

# Step 5: Sort results [OPTIONAL]
# results.sort(key=lambda x: x[0])
# results_final = [r for i, r in results]
print(results_final[0].get())
```

```
0.69138026 seconds
```

```
---------------------------------------------------------------------------
RemoteTraceback                           Traceback (most recent call last)
RemoteTraceback:
"""
Traceback (most recent call last):
  File "/usr/lib/python3.10/multiprocessing/pool.py", line 125, in worker
    result = (True, func(*args, **kwds))
TypeError: howmany_within_range2() missing 1 required positional argument:␣
 ↪'maximum'
"""

The above exception was the direct cause of the following exception:

TypeError                                 Traceback (most recent call last)
<ipython-input-7-74e1a63a5629> in <cell line: 40>()
     38 # results.sort(key=lambda x: x[0])
     39 # results_final = [r for i, r in results]
---> 40 print(results_final[0].get())

/usr/lib/python3.10/multiprocessing/pool.py in get(self, timeout)
```

```
    772             return self._value
    773         else:
--> 774             raise self._value
    775
    776     def _set(self, i, obj):

TypeError: howmany_within_range2() missing 1 required positional argument:␣
 ↪'maximum'
```

10 Option #5: using `Process()`

Let's try yet another option, **not using Pool**. Now, using `Process()`

[ ]:
```
# Parallelizing with Process()
import multiprocessing as mp

def howmany_within_range3(i, row, minimum, maximum):
    """Returns how many numbers lie within `maximum` and `minimum` in a given␣
 ↪`row`"""
    count = 0
    global results
    for num in row:
        if minimum <= num <= maximum:
            count = count + 1
    results[i] = count

# begin timing
start_time = time()

processes = []

for i, row in enumerate(data):
    p = mp.Process(target=howmany_within_range3, args=(i, row, 4, 8, ))
    processes.append(p)
    p.start()

for process in processes:
    process.join()

# end timing
print(round(time() - start_time,8), 'seconds')
print(results[:10])
```

```
Process Process-6:
Process Process-5:
Traceback (most recent call last):
Traceback (most recent call last):
```

```
  Output is truncated.
  File "/usr/lib/python3.10/multiprocessing/process.py", line 314, in _bootstrap
    self.run()
  File "/usr/lib/python3.10/multiprocessing/process.py", line 108, in run
    self._target(*self._args, **self._kwargs)
  File "<ipython-input-8-0f36440b386c>", line 11, in howmany_within_range3
    results[i] = count
  File "<ipython-input-8-0f36440b386c>", line 11, in howmany_within_range3
    results[i] = count
IndexError: list assignment index out of range
IndexError: list assignment index out of range
Process Process-328:
Traceback (most recent call last):
  File "/usr/lib/python3.10/multiprocessing/process.py", line 314, in _bootstrap
    self.run()
  File "/usr/lib/python3.10/multiprocessing/process.py", line 108, in run
    self._target(*self._args, **self._kwargs)
Process Process-329:
  File "<ipython-input-8-0f36440b386c>", line 11, in howmany_within_range3
    results[i] = count
IndexError: list assignment index out of range
Traceback (most recent call last):
  File "/usr/lib/python3.10/multiprocessing/process.py", line 314, in _bootstrap
    self.run()
Process Process-330:
  File "/usr/lib/python3.10/multiprocessing/process.py", line 108, in run
    self._target(*self._args, **self._kwargs)
Process Process-331:
Traceback (most recent call last):
  File "<ipython-input-8-0f36440b386c>", line 11, in howmany_within_range3
    results[i] = count
Traceback (most recent call last):
  File "/usr/lib/python3.10/multiprocessing/process.py", line 314, in _bootstrap
    self.run()
  File "/usr/lib/python3.10/multiprocessing/process.py", line 108, in run
    self._target(*self._args, **self._kwargs)
  File "/usr/lib/python3.10/multiprocessing/process.py", line 314, in _bootstrap
    self.run()
IndexError: list assignment index out of range
Process Process-332:
  File "<ipython-input-8-0f36440b386c>", line 11, in howmany_within_range3
    results[i] = count
  File "/usr/lib/python3.10/multiprocessing/process.py", line 108, in run
    self._target(*self._args, **self._kwargs)
IndexError: list assignment index out of range
Traceback (most recent call last):
  File "<ipython-input-8-0f36440b386c>", line 11, in howmany_within_range3
    results[i] = count
IndexError: list assignment index out of range
```

```
  File "/usr/lib/python3.10/multiprocessing/process.py", line 314, in _bootstrap
    self.run()
Process Process-333:
  File "/usr/lib/python3.10/multiprocessing/process.py", line 108, in run
    self._target(*self._args, **self._kwargs)
  File "<ipython-input-8-0f36440b386c>", line 11, in howmany_within_range3
    results[i] = count
Traceback (most recent call last):
Process Process-334:
  File "/usr/lib/python3.10/multiprocessing/process.py", line 314, in _bootstrap
    self.run()
  File "/usr/lib/python3.10/multiprocessing/process.py", line 108, in run
    self._target(*self._args, **self._kwargs)
  File "<ipython-input-8-0f36440b386c>", line 11, in howmany_within_range3
    results[i] = count
IndexError: list assignment index out of range
IndexError: list assignment index out of range
Traceback (most recent call last):
  File "/usr/lib/python3.10/multiprocessing/process.py", line 314, in _bootstrap
    self.run()
  File "/usr/lib/python3.10/multiprocessing/process.py", line 108, in run
    self._target(*self._args, **self._kwargs)
Process Process-335:
Process Process-336:
  File "<ipython-input-8-0f36440b386c>", line 11, in howmany_within_range3
    results[i] = count
Traceback (most recent call last):
Traceback (most recent call last):
  File "/usr/lib/python3.10/multiprocessing/process.py", line 314, in _bootstrap
    self.run()
IndexError: list assignment index out of range
  File "/usr/lib/python3.10/multiprocessing/process.py", line 108, in run
    self._target(*self._args, **self._kwargs)
  File "/usr/lib/python3.10/multiprocessing/process.py", line 314, in _bootstrap
    self.run()
  File "<ipython-input-8-0f36440b386c>", line 11, in howmany_within_range3
    results[i] = count
  File "/usr/lib/python3.10/multiprocessing/process.py", line 108, in run
    self._target(*self._args, **self._kwargs)
  File "<ipython-input-8-0f36440b386c>", line 11, in howmany_within_range3
    results[i] = count
IndexError: list assignment index out of range
Process Process-337:
Traceback (most recent call last):
  File "/usr/lib/python3.10/multiprocessing/process.py", line 314, in _bootstrap
    self.run()
IndexError: list assignment index out of range
  File "/usr/lib/python3.10/multiprocessing/process.py", line 108, in run
```

```
    self._target(*self._args, **self._kwargs)
  File "<ipython-input-8-0f36440b386c>", line 11, in howmany_within_range3
    results[i] = count
IndexError: list assignment index out of range
Process Process-338:
Traceback (most recent call last):
  File "/usr/lib/python3.10/multiprocessing/process.py", line 314, in _bootstrap
    self.run()
Process Process-339:
  File "/usr/lib/python3.10/multiprocessing/process.py", line 108, in run
    self._target(*self._args, **self._kwargs)
  File "<ipython-input-8-0f36440b386c>", line 11, in howmany_within_range3
    results[i] = count
Traceback (most recent call last):
  File "/usr/lib/python3.10/multiprocessing/process.py", line 314, in _bootstrap
    self.run()
IndexError: list assignment index out of range
Process Process-340:
Process Process-341:
  File "/usr/lib/python3.10/multiprocessing/process.py", line 108, in run
    self._target(*self._args, **self._kwargs)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-8-0f36440b386c> in <cell line: 18>()
     19     p = mp.Process(target=howmany_within_range3, args=(i, row, 4, 8, ))
     20     processes.append(p)
---> 21     p.start()
     22
     23 for process in processes:

/usr/lib/python3.10/multiprocessing/process.py in start(self)
    119                'daemonic processes are not allowed to have children'
    120         _cleanup()
--> 121         self._popen = self._Popen(self)
    122         self._sentinel = self._popen.sentinel
    123         # Avoid a refcycle if the target function holds an indirect

/usr/lib/python3.10/multiprocessing/context.py in _Popen(process_obj)
    222     @staticmethod
    223     def _Popen(process_obj):
--> 224         return _default_context.get_context().Process._Popen(process_ob )
    225
    226     @staticmethod

/usr/lib/python3.10/multiprocessing/context.py in _Popen(process_obj)
    279         def _Popen(process_obj):
```

```
    280             from .popen_fork import Popen
--> 281             return Popen(process_obj)
    282
    283     class SpawnProcess(process.BaseProcess):

/usr/lib/python3.10/multiprocessing/popen_fork.py in __init__(self, process_obj
     14
     15     def __init__(self, process_obj):
---> 16         util._flush_std_streams()
     17         self.returncode = None
     18         self.finalizer = None

/usr/lib/python3.10/multiprocessing/util.py in _flush_std_streams()
    433 def _flush_std_streams():
    434     try:
--> 435         sys.stdout.flush()
    436     except (AttributeError, ValueError):
    437         pass

/usr/local/lib/python3.10/dist-packages/ipykernel/iostream.py in flush(self)
    348                 self.pub_thread.schedule(evt.set)
    349                 # and give a timeout to avoid
--> 350                 if not evt.wait(self.flush_timeout):
    351                     # write directly to __stderr__ instead of warning␣
 ↪because
    352                     # if this is happening sys.stderr may be the proble .

/usr/lib/python3.10/threading.py in wait(self, timeout)
    605             signaled = self._flag
    606             if not signaled:
--> 607                 signaled = self._cond.wait(timeout)
    608             return signaled
    609

/usr/lib/python3.10/threading.py in wait(self, timeout)
    322             else:
    323                 if timeout > 0:
--> 324                     gotit = waiter.acquire(True, timeout)
    325                 else:
    326                     gotit = waiter.acquire(False)

KeyboardInterrupt:
```

```
Traceback (most recent call last):
Traceback (most recent call last):
  File "<ipython-input-8-0f36440b386c>", line 11, in howmany_within_range3
    results[i] = count
```

```
  File "/usr/lib/python3.10/multiprocessing/process.py", line 314, in _bootstrap
    self.run()
  File "/usr/lib/python3.10/multiprocessing/process.py", line 314, in _bootstrap
    self.run()
  File "/usr/lib/python3.10/multiprocessing/process.py", line 108, in run
    self._target(*self._args, **self._kwargs)
  File "<ipython-input-8-0f36440b386c>", line 11, in howmany_within_range3
    results[i] = count
IndexError: list assignment index out of range
  File "/usr/lib/python3.10/multiprocessing/process.py", line 108, in run
    self._target(*self._args, **self._kwargs)
IndexError: list assignment index out of range
Process Process-342:
  File "<ipython-input-8-0f36440b386c>", line 11, in howmany_within_range3
    results[i] = count
IndexError: list assignment index out of range
Traceback (most recent call last):
  File "/usr/lib/python3.10/multiprocessing/process.py", line 314, in _bootstrap
    self.run()
  File "/usr/lib/python3.10/multiprocessing/process.py", line 108, in run
    self._target(*self._args, **self._kwargs)
  File "<ipython-input-8-0f36440b386c>", line 11, in howmany_within_range3
    results[i] = count
IndexError: list assignment index out of range
Process Process-343:
Exception ignored in: <function _after_fork at 0x79cf3a98d990>
Traceback (most recent call last):
  File "/usr/lib/python3.10/threading.py", line 1622, in _after_fork
    threads.update(_dangling)
  File "/usr/lib/python3.10/_weakrefset.py", line 64, in __iter__
    with _IterationGuard(self):
  File "/usr/lib/python3.10/_weakrefset.py", line 33, in __exit__
    w._commit_removals()
  File "/usr/lib/python3.10/_weakrefset.py", line 53, in _commit_removals
    def _commit_removals(self):
KeyboardInterrupt:
Process Process-344:
Traceback (most recent call last):
  File "/usr/lib/python3.10/multiprocessing/process.py", line 314, in _bootstrap
    self.run()
  File "/usr/lib/python3.10/multiprocessing/process.py", line 108, in run
    self._target(*self._args, **self._kwargs)
  File "<ipython-input-8-0f36440b386c>", line 11, in howmany_within_range3
    results[i] = count
IndexError: list assignment index out of range
```

This was a not very good idea, was it? Too many processes are created. Let's try another way.

[ ]:
```
# Parallelizing with Process()
import multiprocessing as mp
import math

def howmany_within_range3(row_start, row_end, minimum, maximum):
    """Returns how many numbers lie within `maximum` and `minimum` in a given␣
 ↪`row`"""
    if row_end > m:
      row_end = m
    results = []
    for row in range(row_start,row_end):
      count = 0
      for num in data[row]:
         if minimum <= num <= maximum:
            count = count + 1
      results.append(count)
    #print(results)

# begin timing
start_time = time()

processes = []

task_size = math.ceil(m / mp.cpu_count())
print(task_size)
for i in range(mp.cpu_count()):
    lower_row_index = i*task_size
    upper_row_index = i*task_size + task_size
    p = mp.Process(target=howmany_within_range3, args=(lower_row_index,␣
 ↪upper_row_index, 4, 8, ))
    processes.append(p)
    p.start()

for process in processes:
    process.join()

# end timing
print(round(time() - start_time,8), 'seconds')
print(results[:10])
```

```
500
0.9142158 seconds
[]
```

11 Q4: Write a summary about these different forms of running parallel code. In which situations would you use each one of those alternatives?

- `pool.apply`
  - Issues one task and blocks until each task is completed.
  - Useful for passing multiple arguments to a task and processing each task one at a time.
- `pool.map`
  - Issues multiple tasks simultaneously and blocks until they are completed.
  - Useful for concurrently processing with ordered results.
- `pool.starmap`
  - Similar to `map` but accepts multiple arguments.
  - Useful for passing multiple arguments to each task.
- `pool.apply_async`
  - Issue one asynchronous task to the process pool, i.e., it doesn't block the main process. It supports a callback function for the results.
  - Useful for asynchronous task execution or retrieving results when they are ready.
- `Process()`
  - Provides detailed control over specific processes, although manual handling of processes may be time-consuming and susceptible to errors.
  - Useful when we want additional control over specific processes or when using Pool is not suitable.

Next, it follows a small example of the use of threads (not using the `threading` module, but the `multiprocessing.dummy` module, which replicates the multiprocessing module to work with threads) in python. More details at: https://stackoverflow.com/questions/2846653/how-can-i-use-threading-in-python. Here, we profile the code using `cProfile`.

(Note: although you may not see much advantage of using threads for these examples, if you try an application that needs to fetch files from the network, you may notice speedups - see example here)

12 Q5: Modify these scripts to run using multiple threads instead of processes (you will need to use another module: `threading`). Compare their performance when varying the matrix size.

[50]:
```
import cProfile

from time import time
import numpy as np
import math
import multiprocessing as mp

# Prepare data
# value range
r = 10
# number of rows
```

```
m = 1000
# m = 5
# number of columns
n = 10000
# n = 30

np.random.seed(100)
arr = np.random.randint(0, r, size=[m, n])
data = arr.tolist()
# print(data[:10])

def howmany_within_range_rowonly(row, minimum=4, maximum=8):
  # print(mp.current_process(),' ',row) # this will print the process object␣
 ↪and the item it is working with
  count = 0
  for num in row:
     if minimum <= num <= maximum:
        count = count + 1
  return count

def mythreads_1():
  # creating 2 threads
  from multiprocessing.dummy import Pool as ThreadPool
  start_time = time()
  pool = ThreadPool(mp.cpu_count())
  results = pool.map(howmany_within_range_rowonly, [row for row in data])
  print("Using cpu_count threads: ",round(time() - start_time,8), 'seconds')
  #print(results)

def mythreads_2():
  # Other thread version, trying to divide work according to indice
  from multiprocessing.dummy import Pool as ThreadPool
  start_time = time()
  task_size = int(math.ceil(m / mp.cpu_count()))
  pool = ThreadPool(mp.cpu_count())
  print(task_size)
  for i in range(mp.cpu_count()):
    lower_row_index = i*task_size
    upper_row_index = i*task_size + task_size
    results[i] = pool.map(howmany_within_range_rowonly, [data[j] for j in␣
 ↪range(lower_row_index,upper_row_index)])
  print("Using cpu_count threads but dividing indice: ",round(time() -␣
 ↪start_time,8), 'seconds')

def seq():
  # sequential version
  start_time = time()
```

```
  results = []
  for row in data:
    results.append(howmany_within_range_rowonly(row))
  print("Sequential: ",round(time() - start_time,8), 'seconds')
    #print(results)

if __name__=='__main__':
   cProfile.run("mythreads_1()")
   cProfile.run("mythreads_2()")
#   cProfile.run("seq()")
```

```
Using cpu_count threads:  0.93803954 seconds
         586 function calls in 0.942 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        3    0.000    0.000    0.000    0.000 <frozen
importlib._bootstrap>:1053(_handle_fromlist)
        6    0.000    0.000    0.000    0.000 <frozen
importlib._bootstrap>:404(parent)
        1    0.000    0.000    0.938    0.938 <ipython-
input-50-478cd5dc192b>:31(mythreads_1)
        1    0.000    0.000    0.000    0.000 <ipython-
input-50-478cd5dc192b>:36(<listcomp>)
        1    0.000    0.000    0.942    0.942 <string>:1(<module>)
        1    0.000    0.000    0.002    0.002 __init__.py:122(Pool)
        2    0.000    0.000    0.000    0.000 __init__.py:36(__init__)
        2    0.000    0.000    0.000    0.000 __init__.py:43(start)
        5    0.000    0.000    0.000    0.000 _weakrefset.py:86(add)
        2    0.000    0.000    0.000    0.000 connection.py:117(__init__)
        2    0.000    0.000    0.000    0.000 connection.py:130(__del__)
        3    0.000    0.000    0.000    0.000 connection.py:134(_check_closed)
        3    0.000    0.000    0.000    0.000 connection.py:142(_check_writable)
        3    0.000    0.000    0.000    0.000 connection.py:181(send_bytes)
        2    0.000    0.000    0.000    0.000 connection.py:360(_close)
        3    0.000    0.000    0.000    0.000 connection.py:365(_send)
        3    0.000    0.000    0.000    0.000 connection.py:390(_send_bytes)
        1    0.000    0.000    0.000    0.000 connection.py:516(Pipe)
        1    0.000    0.000    0.000    0.000 context.py:110(SimpleQueue)
        3    0.000    0.000    0.000    0.000 context.py:187(get_context)
        2    0.000    0.000    0.000    0.000 context.py:197(get_start_method)
        1    0.000    0.000    0.000    0.000 context.py:237(get_context)
        1    0.000    0.000    0.000    0.000 context.py:41(cpu_count)
        2    0.000    0.000    0.000    0.000 context.py:65(Lock)
        7    0.000    0.000    0.000    0.000 iostream.py:195(schedule)
        6    0.000    0.000    0.000    0.000
```

```
iostream.py:308(_is_master_process)
        6    0.000    0.000    0.000    0.000 iostream.py:321(_schedule_flush)
        6    0.000    0.000    0.000    0.000 iostream.py:384(write)
        7    0.000    0.000    0.000    0.000 iostream.py:91(_event_pipe)
        1    0.000    0.000    0.000    0.000 pool.py:157(__init__)
        1    0.000    0.000    0.002    0.002 pool.py:183(__init__)
        1    0.000    0.000    0.000    0.000 pool.py:266(__del__)
        1    0.000    0.000    0.000    0.000 pool.py:273(__repr__)
        1    0.000    0.000    0.001    0.001 pool.py:305(_repopulate_pool)
        1    0.000    0.000    0.001    0.001
pool.py:314(_repopulate_pool_static)
        1    0.000    0.000    0.000    0.000 pool.py:351(_check_running)
        1    0.000    0.000    0.936    0.936 pool.py:362(map)
        1    0.000    0.000    0.000    0.000 pool.py:471(_map_async)
        1    0.000    0.000    0.003    0.003 pool.py:680(_terminate_pool)
        1    0.000    0.000    0.000    0.000 pool.py:747(__init__)
        1    0.000    0.000    0.000    0.000 pool.py:756(ready)
        1    0.000    0.000    0.936    0.936 pool.py:764(wait)
        1    0.000    0.000    0.936    0.936 pool.py:767(get)
        1    0.000    0.000    0.000    0.000 pool.py:796(__init__)
        2    0.000    0.000    0.000    0.000 pool.py:924(Process)
        1    0.000    0.000    0.002    0.002 pool.py:929(__init__)
        1    0.000    0.000    0.000    0.000 pool.py:932(_setup_queues)
        1    0.000    0.000    0.000    0.000 pool.py:938(_get_sentinels)
        1    0.000    0.000    0.000    0.000 pool.py:945(_help_stuff_finish)
        2    0.000    0.000    0.000    0.000 process.py:37(current_process)
        1    0.000    0.000    0.000    0.000 queues.py:339(__init__)
        3    0.000    0.000    0.000    0.000 queues.py:369(put)
        2    0.000    0.000    0.000    0.000 random.py:506(choices)
        2    0.000    0.000    0.000    0.000 random.py:519(<listcomp>)
        3    0.000    0.000    0.000    0.000 reduction.py:38(__init__)
        3    0.000    0.000    0.000    0.000 reduction.py:48(dumps)
        7    0.000    0.000    0.000    0.000 socket.py:543(send)
        2    0.000    0.000    0.000    0.000 synchronize.py:114(_make_name)
        2    0.000    0.000    0.000    0.000 synchronize.py:161(__init__)
        2    0.000    0.000    0.000    0.000 synchronize.py:50(__init__)
        2    0.000    0.000    0.000    0.000 synchronize.py:90(_make_methods)
        3    0.000    0.000    0.000    0.000 synchronize.py:94(__enter__)
        3    0.000    0.000    0.000    0.000 synchronize.py:97(__exit__)
        2    0.000    0.000    0.000    0.000 tempfile.py:281(rng)
        2    0.000    0.000    0.000    0.000 tempfile.py:292(__next__)
        3    0.000    0.000    0.000    0.000 threading.py:1028(_stop)
        3    0.000    0.000    0.003    0.001 threading.py:1064(join)
       11    0.000    0.000    0.003    0.000
threading.py:1102(_wait_for_tstate_lock)
        2    0.000    0.000    0.000    0.000 threading.py:1129(name)
        2    0.000    0.000    0.000    0.000 threading.py:1140(name)
        8    0.000    0.000    0.000    0.000 threading.py:1169(is_alive)
```

```
        8    0.000    0.000    0.000    0.000 threading.py:1183(daemon)
        5    0.000    0.000    0.000    0.000 threading.py:1198(daemon)
        5    0.000    0.000    0.000    0.000
threading.py:1301(_make_invoke_excepthook)
       15    0.000    0.000    0.000    0.000 threading.py:1430(current_thread)
        6    0.000    0.000    0.000    0.000 threading.py:236(__init__)
        6    0.000    0.000    0.000    0.000 threading.py:264(__enter__)
        6    0.000    0.000    0.000    0.000 threading.py:267(__exit__)
        6    0.000    0.000    0.000    0.000 threading.py:273(_release_save)
        6    0.000    0.000    0.000    0.000 threading.py:276(_acquire_restore)
        6    0.000    0.000    0.000    0.000 threading.py:279(_is_owned)
        6    0.000    0.000    0.936    0.156 threading.py:288(wait)
        6    0.000    0.000    0.000    0.000 threading.py:545(__init__)
       22    0.000    0.000    0.000    0.000 threading.py:553(is_set)
        6    0.000    0.000    0.936    0.156 threading.py:589(wait)
        5    0.000    0.000    0.000    0.000 threading.py:782(_newname)
        5    0.000    0.000    0.000    0.000 threading.py:827(__init__)
        5    0.000    0.000    0.001    0.000 threading.py:916(start)
        2    0.000    0.000    0.000    0.000 util.py:171(register_after_fork)
        1    0.000    0.000    0.000    0.000 util.py:186(__init__)
        1    0.000    0.000    0.003    0.003 util.py:205(__call__)
        1    0.000    0.000    0.000    0.000 util.py:44(sub_debug)
        9    0.000    0.000    0.000    0.000 util.py:48(debug)
        2    0.000    0.000    0.000    0.000 weakref.py:106(remove)
        2    0.000    0.000    0.000    0.000 weakref.py:165(__setitem__)
        2    0.000    0.000    0.000    0.000 weakref.py:348(__new__)
        2    0.000    0.000    0.000    0.000 weakref.py:353(__init__)
        2    0.000    0.000    0.000    0.000 weakref.py:368(__init__)
        2    0.000    0.000    0.000    0.000 weakref.py:428(__setitem__)
        2    0.000    0.000    0.000    0.000 {built-in method __new__ of type
object at 0x577961e729a0}
        3    0.000    0.000    0.000    0.000 {built-in method _struct.pack}
       12    0.000    0.000    0.000    0.000 {built-in method
_thread.allocate_lock}
       15    0.000    0.000    0.000    0.000 {built-in method
_thread.get_ident}
        5    0.000    0.000    0.000    0.000 {built-in method
_thread.start_new_thread}
        1    0.000    0.000    0.000    0.000 {built-in method _warnings.warn}
        2    0.000    0.000    0.000    0.000 {built-in method
_weakref._remove_dead_weakref}
        1    0.000    0.000    0.000    0.000 {built-in method builtins.divmod}
        1    0.000    0.000    0.942    0.942 {built-in method builtins.exec}
        3    0.000    0.000    0.000    0.000 {built-in method builtins.getattr}
        8    0.000    0.000    0.000    0.000 {built-in method builtins.hasattr}
        2    0.000    0.000    0.000    0.000 {built-in method builtins.id}
       10    0.000    0.000    0.000    0.000 {built-in method
builtins.isinstance}
```

```
       18    0.000    0.000    0.000    0.000 {built-in method builtins.len}
        6    0.000    0.000    0.000    0.000 {built-in method builtins.next}
        1    0.000    0.000    0.000    0.000 {built-in method builtins.print}
        1    0.000    0.000    0.000    0.000 {built-in method builtins.round}
       16    0.000    0.000    0.000    0.000 {built-in method math.floor}
        2    0.000    0.000    0.000    0.000 {built-in method posix.close}
        1    0.000    0.000    0.000    0.000 {built-in method posix.cpu_count}
       10    0.000    0.000    0.000    0.000 {built-in method posix.getpid}
        1    0.000    0.000    0.000    0.000 {built-in method posix.pipe}
        3    0.000    0.000    0.000    0.000 {built-in method posix.write}
        2    0.000    0.000    0.000    0.000 {built-in method time.time}
        3    0.000    0.000    0.000    0.000 {method '__enter__' of
'_multiprocessing.SemLock' objects}
        6    0.000    0.000    0.000    0.000 {method '__enter__' of
'_thread.lock' objects}
        3    0.000    0.000    0.000    0.000 {method '__exit__' of
'_multiprocessing.SemLock' objects}
        5    0.000    0.000    0.000    0.000 {method '__exit__' of
'_thread.RLock' objects}
        6    0.000    0.000    0.000    0.000 {method '__exit__' of
'_thread.lock' objects}
       35    0.939    0.027    0.939    0.027 {method 'acquire' of
'_thread.lock' objects}
        5    0.000    0.000    0.000    0.000 {method 'add' of 'set' objects}
       13    0.000    0.000    0.000    0.000 {method 'append' of
'collections.deque' objects}
        2    0.000    0.000    0.000    0.000 {method 'append' of 'list'
objects}
        3    0.000    0.000    0.000    0.000 {method 'copy' of 'dict' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of
'_lsprof.Profiler' objects}
        3    0.000    0.000    0.000    0.000 {method 'dump' of
'_pickle.Pickler' objects}
        1    0.000    0.000    0.000    0.000 {method 'get' of
'_queue.SimpleQueue' objects}
        3    0.000    0.000    0.000    0.000 {method 'getbuffer' of
'_io.BytesIO' objects}
        2    0.000    0.000    0.000    0.000 {method 'join' of 'str' objects}
        3    0.000    0.000    0.000    0.000 {method 'locked' of '_thread.lock'
objects}
        4    0.000    0.000    0.000    0.000 {method 'put' of
'_queue.SimpleQueue' objects}
       16    0.000    0.000    0.000    0.000 {method 'random' of
'_random.Random' objects}
        9    0.000    0.000    0.000    0.000 {method 'release' of
'_thread.lock' objects}
        2    0.000    0.000    0.000    0.000 {method 'replace' of 'str'
objects}
```

```
        5    0.000    0.000    0.000    0.000 {method 'rpartition' of 'str'
objects}
        3    0.000    0.000    0.000    0.000 {method 'update' of 'dict'
objects}


500
Using cpu_count threads but dividing indice:  1.13929749 seconds
         655 function calls in 1.143 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        3    0.000    0.000    0.000    0.000 <frozen
importlib._bootstrap>:1053(_handle_fromlist)
        6    0.000    0.000    0.000    0.000 <frozen
importlib._bootstrap>:404(parent)
        1    0.000    0.000    1.142    1.142 <ipython-
input-50-478cd5dc192b>:40(mythreads_2)
        2    0.000    0.000    0.000    0.000 <ipython-
input-50-478cd5dc192b>:50(<listcomp>)
        1    0.000    0.000    1.143    1.143 <string>:1(<module>)
        1    0.000    0.000    0.179    0.179 __init__.py:122(Pool)
        2    0.000    0.000    0.000    0.000 __init__.py:36(__init__)
        2    0.000    0.000    0.092    0.046 __init__.py:43(start)
        5    0.000    0.000    0.000    0.000 _weakrefset.py:86(add)
        2    0.000    0.000    0.000    0.000 connection.py:117(__init__)
        2    0.000    0.000    0.000    0.000 connection.py:130(__del__)
        3    0.000    0.000    0.000    0.000 connection.py:134(_check_closed)
        3    0.000    0.000    0.000    0.000 connection.py:142(_check_writable)
        3    0.000    0.000    0.000    0.000 connection.py:181(send_bytes)
        2    0.000    0.000    0.000    0.000 connection.py:360(_close)
        3    0.000    0.000    0.000    0.000 connection.py:365(_send)
        3    0.000    0.000    0.000    0.000 connection.py:390(_send_bytes)
        1    0.000    0.000    0.000    0.000 connection.py:516(Pipe)
        1    0.000    0.000    0.001    0.001 context.py:110(SimpleQueue)
        3    0.000    0.000    0.000    0.000 context.py:187(get_context)
        2    0.000    0.000    0.000    0.000 context.py:197(get_start_method)
        1    0.000    0.000    0.000    0.000 context.py:237(get_context)
        3    0.000    0.000    0.000    0.000 context.py:41(cpu_count)
        2    0.000    0.000    0.000    0.000 context.py:65(Lock)
        9    0.000    0.000    0.003    0.000 iostream.py:195(schedule)
        8    0.000    0.000    0.000    0.000
iostream.py:308(_is_master_process)
        8    0.000    0.000    0.000    0.000 iostream.py:321(_schedule_flush)
        8    0.000    0.000    0.003    0.000 iostream.py:384(write)
        9    0.000    0.000    0.000    0.000 iostream.py:91(_event_pipe)
        1    0.000    0.000    0.000    0.000 pool.py:157(__init__)
```

```
        1    0.000    0.000    0.179    0.179 pool.py:183(__init__)
        1    0.000    0.000    0.000    0.000 pool.py:266(__del__)
        1    0.000    0.000    0.000    0.000 pool.py:273(__repr__)
        1    0.000    0.000    0.093    0.093 pool.py:305(_repopulate_pool)
        1    0.000    0.000    0.093    0.093
pool.py:314(_repopulate_pool_static)
        2    0.000    0.000    0.000    0.000 pool.py:351(_check_running)
        2    0.000    0.000    0.959    0.480 pool.py:362(map)
        2    0.000    0.000    0.000    0.000 pool.py:471(_map_async)
        1    0.000    0.000    0.001    0.001 pool.py:680(_terminate_pool)
        2    0.000    0.000    0.000    0.000 pool.py:747(__init__)
        2    0.000    0.000    0.000    0.000 pool.py:756(ready)
        2    0.000    0.000    0.959    0.479 pool.py:764(wait)
        2    0.000    0.000    0.959    0.479 pool.py:767(get)
        2    0.000    0.000    0.000    0.000 pool.py:796(__init__)
        2    0.000    0.000    0.000    0.000 pool.py:924(Process)
        1    0.000    0.000    0.179    0.179 pool.py:929(__init__)
        1    0.000    0.000    0.000    0.000 pool.py:932(_setup_queues)
        1    0.000    0.000    0.000    0.000 pool.py:938(_get_sentinels)
        1    0.000    0.000    0.000    0.000 pool.py:945(_help_stuff_finish)
        2    0.000    0.000    0.000    0.000 process.py:37(current_process)
        1    0.000    0.000    0.001    0.001 queues.py:339(__init__)
        3    0.000    0.000    0.000    0.000 queues.py:369(put)
        2    0.000    0.000    0.000    0.000 random.py:506(choices)
        2    0.000    0.000    0.000    0.000 random.py:519(<listcomp>)
        3    0.000    0.000    0.000    0.000 reduction.py:38(__init__)
        3    0.000    0.000    0.000    0.000 reduction.py:48(dumps)
        9    0.003    0.000    0.003    0.000 socket.py:543(send)
        2    0.000    0.000    0.000    0.000 synchronize.py:114(_make_name)
        2    0.000    0.000    0.000    0.000 synchronize.py:161(__init__)
        2    0.000    0.000    0.000    0.000 synchronize.py:50(__init__)
        2    0.000    0.000    0.000    0.000 synchronize.py:90(_make_methods)
        3    0.000    0.000    0.000    0.000 synchronize.py:94(__enter__)
        3    0.000    0.000    0.000    0.000 synchronize.py:97(__exit__)
        2    0.000    0.000    0.000    0.000 tempfile.py:281(rng)
        2    0.000    0.000    0.000    0.000 tempfile.py:292(__next__)
        3    0.000    0.000    0.000    0.000 threading.py:1028(_stop)
        3    0.000    0.000    0.001    0.000 threading.py:1064(join)
       13    0.000    0.000    0.001    0.000
threading.py:1102(_wait_for_tstate_lock)
        2    0.000    0.000    0.000    0.000 threading.py:1129(name)
        2    0.000    0.000    0.000    0.000 threading.py:1140(name)
       10    0.000    0.000    0.000    0.000 threading.py:1169(is_alive)
        8    0.000    0.000    0.000    0.000 threading.py:1183(daemon)
        5    0.000    0.000    0.000    0.000 threading.py:1198(daemon)
        5    0.000    0.000    0.000    0.000
threading.py:1301(_make_invoke_excepthook)
       15    0.000    0.000    0.000    0.000 threading.py:1430(current_thread)
```

```
        7    0.000    0.000    0.000    0.000 threading.py:236(__init__)
        7    0.000    0.000    0.000    0.000 threading.py:264(__enter__)
        7    0.000    0.000    0.000    0.000 threading.py:267(__exit__)
        7    0.000    0.000    0.000    0.000 threading.py:273(_release_save)
        7    0.000    0.000    0.000    0.000 threading.py:276(_acquire_restore)
        7    0.000    0.000    0.000    0.000 threading.py:279(_is_owned)
        7    0.000    0.000    1.136    0.162 threading.py:288(wait)
        7    0.000    0.000    0.000    0.000 threading.py:545(__init__)
       25    0.000    0.000    0.000    0.000 threading.py:553(is_set)
        7    0.000    0.000    1.136    0.162 threading.py:589(wait)
        5    0.000    0.000    0.000    0.000 threading.py:782(_newname)
        5    0.000    0.000    0.000    0.000 threading.py:827(__init__)
        5    0.000    0.000    0.177    0.035 threading.py:916(start)
        2    0.000    0.000    0.000    0.000 util.py:171(register_after_fork)
        1    0.000    0.000    0.000    0.000 util.py:186(__init__)
        1    0.000    0.000    0.001    0.001 util.py:205(__call__)
        1    0.000    0.000    0.000    0.000 util.py:44(sub_debug)
        9    0.000    0.000    0.000    0.000 util.py:48(debug)
        2    0.000    0.000    0.000    0.000 weakref.py:106(remove)
        2    0.000    0.000    0.000    0.000 weakref.py:165(__setitem__)
        2    0.000    0.000    0.000    0.000 weakref.py:348(__new__)
        2    0.000    0.000    0.000    0.000 weakref.py:353(__init__)
        2    0.000    0.000    0.000    0.000 weakref.py:368(__init__)
        2    0.000    0.000    0.000    0.000 weakref.py:428(__setitem__)
        2    0.000    0.000    0.000    0.000 {built-in method __new__ of type
object at 0x577961e729a0}
        3    0.000    0.000    0.000    0.000 {built-in method _struct.pack}
       14    0.000    0.000    0.000    0.000 {built-in method
_thread.allocate_lock}
       15    0.000    0.000    0.000    0.000 {built-in method
_thread.get_ident}
        5    0.000    0.000    0.000    0.000 {built-in method
_thread.start_new_thread}
        1    0.000    0.000    0.000    0.000 {built-in method _warnings.warn}
        2    0.000    0.000    0.000    0.000 {built-in method
_weakref._remove_dead_weakref}
        2    0.000    0.000    0.000    0.000 {built-in method builtins.divmod}
        1    0.000    0.000    1.143    1.143 {built-in method builtins.exec}
        3    0.000    0.000    0.000    0.000 {built-in method builtins.getattr}
        9    0.000    0.000    0.000    0.000 {built-in method builtins.hasattr}
        2    0.000    0.000    0.000    0.000 {built-in method builtins.id}
       12    0.000    0.000    0.000    0.000 {built-in method
builtins.isinstance}
       22    0.000    0.000    0.000    0.000 {built-in method builtins.len}
        7    0.000    0.000    0.000    0.000 {built-in method builtins.next}
        2    0.000    0.000    0.003    0.001 {built-in method builtins.print}
        1    0.000    0.000    0.000    0.000 {built-in method builtins.round}
        1    0.000    0.000    0.000    0.000 {built-in method math.ceil}
```

```
       16    0.000    0.000    0.000    0.000 {built-in method math.floor}
        2    0.000    0.000    0.000    0.000 {built-in method posix.close}
        3    0.000    0.000    0.000    0.000 {built-in method posix.cpu_count}
       12    0.000    0.000    0.000    0.000 {built-in method posix.getpid}
        1    0.000    0.000    0.000    0.000 {built-in method posix.pipe}
        3    0.000    0.000    0.000    0.000 {built-in method posix.write}
        2    0.000    0.000    0.000    0.000 {built-in method time.time}
        3    0.000    0.000    0.000    0.000 {method '__enter__' of
'_multiprocessing.SemLock' objects}
        7    0.000    0.000    0.000    0.000 {method '__enter__' of
'_thread.lock' objects}
        3    0.000    0.000    0.000    0.000 {method '__exit__' of
'_multiprocessing.SemLock' objects}
        5    0.000    0.000    0.000    0.000 {method '__exit__' of
'_thread.RLock' objects}
        7    0.000    0.000    0.000    0.000 {method '__exit__' of
'_thread.lock' objects}
       41    1.136    0.028    1.136    0.028 {method 'acquire' of
'_thread.lock' objects}
        5    0.000    0.000    0.000    0.000 {method 'add' of 'set' objects}
       16    0.000    0.000    0.000    0.000 {method 'append' of
'collections.deque' objects}
        2    0.000    0.000    0.000    0.000 {method 'append' of 'list'
objects}
        3    0.000    0.000    0.000    0.000 {method 'copy' of 'dict' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of
'_lsprof.Profiler' objects}
        3    0.000    0.000    0.000    0.000 {method 'dump' of
'_pickle.Pickler' objects}
        1    0.000    0.000    0.000    0.000 {method 'get' of
'_queue.SimpleQueue' objects}
        3    0.000    0.000    0.000    0.000 {method 'getbuffer' of
'_io.BytesIO' objects}
        2    0.000    0.000    0.000    0.000 {method 'join' of 'str' objects}
        3    0.000    0.000    0.000    0.000 {method 'locked' of '_thread.lock'
objects}
        5    0.000    0.000    0.000    0.000 {method 'put' of
'_queue.SimpleQueue' objects}
       16    0.000    0.000    0.000    0.000 {method 'random' of
'_random.Random' objects}
       10    0.000    0.000    0.000    0.000 {method 'release' of
'_thread.lock' objects}
        2    0.000    0.000    0.000    0.000 {method 'replace' of 'str'
objects}
        5    0.000    0.000    0.000    0.000 {method 'rpartition' of 'str'
objects}
        3    0.000    0.000    0.000    0.000 {method 'update' of 'dict'
objects}
```

Multiple threads:

[55]:
```
import cProfile
from time import time
import numpy as np
import math
#import multiprocessing as mp
import threading

# Prepare data
# value range
r = 10
# number of rows
m = 1000
# m = 5
# number of columns
n = 10000
# n = 30

np.random.seed(100)
arr = np.random.randint(0, r, size=[m, n])
data = arr.tolist()
# print(data[:10])

def howmany_within_range_rowonly(row, minimum=4, maximum=8):
  # print(mp.current_process(),' ',row) # this will print the process object␣
 ↪and the item it is working with
  count = 0
  for num in row:
     if minimum <= num <= maximum:
        count = count + 1
  return count

def mythreads_1():
  # creating 2 threads
  start_time = time()
  results = []
  threads = []
  #for item in [row for row in data]:
    #results.append(howmany_within_range_rowonly(item))
  for row in data:
    thread = threading.Thread(target=results.append,␣
 ↪args=(howmany_within_range_rowonly(row),))
    threads.append(thread)
    thread.start()
```

```
  for thread in threads:
    thread.join()
  print("Using cpu_count threads: ",round(time() - start_time,8), 'seconds')
  #print(results)

def mythreads_2():
  # Other thread version, trying to divide work according to indice
  start_time = time()
  task_size = int(math.ceil(m / threading.active_count()))
  print(task_size)
  results = []
  threads = []
  for i in range(threading.active_count()):
    lower_row_index = i*task_size
    upper_row_index = i*task_size + task_size
    thread = threading.Thread(target=results.extend,␣
 ↪args=([howmany_within_range_rowonly(j) for j in data[lower_row_index:
 ↪upper_row_index]],))
    threads.append(thread)
    thread.start()
  for thread in threads:
    thread.join()
  print("Using cpu_count threads but dividing indice: ",round(time() -␣
 ↪start_time,8), 'seconds')

def seq():
  # sequential version
  start_time = time()
  results = []
  for row in data:
    results.append(howmany_within_range_rowonly(row))
  print("Sequential: ",round(time() - start_time,8), 'seconds')
    #print(results)

if __name__=='__main__':
   cProfile.run("mythreads_1()")
   cProfile.run("mythreads_2()")
#   cProfile.run("seq()")
```

```
Using cpu_count threads:  1.7264843 seconds
         51068 function calls in 1.730 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     1000    1.286    0.001    1.286    0.001 <ipython-
input-55-788bb9bc2302>:23(howmany_within_range_rowonly)
```

```
        1    0.009    0.009    1.727    1.727 <ipython-
input-55-788bb9bc2302>:31(mythreads_1)
        1    0.002    0.002    1.729    1.729 <string>:1(<module>)
      992    0.000    0.000    0.001    0.000 _weakrefset.py:39(_remove)
     1000    0.002    0.000    0.003    0.000 _weakrefset.py:86(add)
        7    0.000    0.000    0.000    0.000 iostream.py:195(schedule)
        6    0.000    0.000    0.000    0.000
iostream.py:308(_is_master_process)
        6    0.000    0.000    0.000    0.000 iostream.py:321(_schedule_flush)
        6    0.000    0.000    0.000    0.000 iostream.py:384(write)
        7    0.000    0.000    0.000    0.000 iostream.py:91(_event_pipe)
        7    0.000    0.000    0.000    0.000 socket.py:543(send)
     1000    0.001    0.000    0.004    0.000 threading.py:1028(_stop)
     1000    0.002    0.000    0.008    0.000 threading.py:1064(join)
     1007    0.001    0.000    0.005    0.000
threading.py:1102(_wait_for_tstate_lock)
        7    0.000    0.000    0.000    0.000 threading.py:1169(is_alive)
     2000    0.001    0.000    0.001    0.000 threading.py:1183(daemon)
     1000    0.003    0.000    0.003    0.000
threading.py:1301(_make_invoke_excepthook)
     2000    0.002    0.000    0.002    0.000 threading.py:1430(current_thread)
     1000    0.009    0.000    0.009    0.000 threading.py:236(__init__)
     1000    0.022    0.000    0.023    0.000 threading.py:264(__enter__)
     1000    0.001    0.000    0.001    0.000 threading.py:267(__exit__)
      999    0.001    0.000    0.001    0.000 threading.py:273(_release_save)
      999    0.001    0.000    0.001    0.000 threading.py:276(_acquire_restore)
      999    0.001    0.000    0.002    0.000 threading.py:279(_is_owned)
      999    0.006    0.000    0.102    0.000 threading.py:288(wait)
     1000    0.002    0.000    0.012    0.000 threading.py:545(__init__)
     2007    0.001    0.000    0.001    0.000 threading.py:553(is_set)
     1000    0.216    0.000    0.342    0.000 threading.py:589(wait)
     1000    0.003    0.000    0.003    0.000 threading.py:782(_newname)
     1000    0.001    0.000    0.002    0.000
threading.py:800(_maintain_shutdown_locks)
     1000    0.001    0.000    0.001    0.000 threading.py:810(<listcomp>)
     1000    0.013    0.000    0.037    0.000 threading.py:827(__init__)
     1000    0.006    0.000    0.386    0.000 threading.py:916(start)
     1999    0.001    0.000    0.001    0.000 {built-in method
_thread.allocate_lock}
     2000    0.001    0.000    0.001    0.000 {built-in method
_thread.get_ident}
     1000    0.036    0.000    0.036    0.000 {built-in method
_thread.start_new_thread}
        1    0.000    0.000    1.730    1.730 {built-in method builtins.exec}
        6    0.000    0.000    0.000    0.000 {built-in method
builtins.isinstance}
        1    0.000    0.000    0.000    0.000 {built-in method builtins.print}
        1    0.000    0.000    0.000    0.000 {built-in method builtins.round}
```

```
        6    0.000    0.000    0.000    0.000 {built-in method posix.getpid}
        2    0.000    0.000    0.000    0.000 {built-in method time.time}
     1000    0.000    0.000    0.000    0.000 {method '__enter__' of
'_thread.lock' objects}
     1000    0.000    0.000    0.000    0.000 {method '__exit__' of
'_thread.RLock' objects}
     2000    0.001    0.000    0.001    0.000 {method '__exit__' of
'_thread.lock' objects}
     5003    0.093    0.000    0.093    0.000 {method 'acquire' of
'_thread.lock' objects}
     1000    0.001    0.000    0.001    0.000 {method 'add' of 'set' objects}
     1006    0.000    0.000    0.000    0.000 {method 'append' of
'collections.deque' objects}
     1000    0.002    0.000    0.002    0.000 {method 'append' of 'list'
objects}
     1000    0.000    0.000    0.000    0.000 {method 'difference_update' of
'set' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of
'_lsprof.Profiler' objects}
      992    0.000    0.000    0.000    0.000 {method 'discard' of 'set'
objects}
     3001    0.000    0.000    0.000    0.000 {method 'locked' of '_thread.lock'
objects}
     1999    0.000    0.000    0.000    0.000 {method 'release' of
'_thread.lock' objects}


91
Using cpu_count threads but dividing indice:  2.41104174 seconds
         1728 function calls in 2.411 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     1000    2.398    0.002    2.398    0.002 <ipython-
input-55-788bb9bc2302>:23(howmany_within_range_rowonly)
        1    0.000    0.000    2.411    2.411 <ipython-
input-55-788bb9bc2302>:47(mythreads_2)
       11    0.004    0.000    2.402    0.218 <ipython-
input-55-788bb9bc2302>:57(<listcomp>)
        1    0.000    0.000    2.411    2.411 <string>:1(<module>)
       11    0.000    0.000    0.000    0.000 _weakrefset.py:39(_remove)
       11    0.000    0.000    0.000    0.000 _weakrefset.py:86(add)
        9    0.000    0.000    0.000    0.000 iostream.py:195(schedule)
        8    0.000    0.000    0.000    0.000
iostream.py:308(_is_master_process)
        8    0.000    0.000    0.000    0.000 iostream.py:321(_schedule_flush)
        8    0.000    0.000    0.000    0.000 iostream.py:384(write)
```

```
        9    0.000    0.000    0.000    0.000 iostream.py:91(_event_pipe)
        2    0.000    0.000    0.000    0.000
pydevd_daemon_thread.py:103(<listcomp>)
        2    0.000    0.000    0.000    0.000
pydevd_daemon_thread.py:128(new_active_count)
        2    0.000    0.000    0.000    0.000
pydevd_daemon_thread.py:99(new_threading_enumerate)
        9    0.000    0.000    0.000    0.000 socket.py:543(send)
       11    0.000    0.000    0.000    0.000 threading.py:1028(_stop)
       11    0.000    0.000    0.000    0.000 threading.py:1064(join)
       20    0.000    0.000    0.000    0.000
threading.py:1102(_wait_for_tstate_lock)
        9    0.000    0.000    0.000    0.000 threading.py:1169(is_alive)
       22    0.000    0.000    0.000    0.000 threading.py:1183(daemon)
       11    0.000    0.000    0.000    0.000
threading.py:1301(_make_invoke_excepthook)
       22    0.000    0.000    0.000    0.000 threading.py:1430(current_thread)
       11    0.000    0.000    0.000    0.000 threading.py:236(__init__)
       11    0.000    0.000    0.000    0.000 threading.py:264(__enter__)
       11    0.000    0.000    0.000    0.000 threading.py:267(__exit__)
       11    0.000    0.000    0.000    0.000 threading.py:273(_release_save)
       11    0.000    0.000    0.000    0.000 threading.py:276(_acquire_restore)
       11    0.000    0.000    0.000    0.000 threading.py:279(_is_owned)
       11    0.002    0.000    0.006    0.001 threading.py:288(wait)
       11    0.000    0.000    0.000    0.000 threading.py:545(__init__)
       31    0.000    0.000    0.000    0.000 threading.py:553(is_set)
       11    0.000    0.000    0.006    0.001 threading.py:589(wait)
       11    0.000    0.000    0.000    0.000 threading.py:782(_newname)
       11    0.000    0.000    0.000    0.000
threading.py:800(_maintain_shutdown_locks)
       11    0.000    0.000    0.000    0.000 threading.py:810(<listcomp>)
       11    0.000    0.000    0.001    0.000 threading.py:827(__init__)
       11    0.000    0.000    0.007    0.001 threading.py:916(start)
       22    0.000    0.000    0.000    0.000 {built-in method
_thread.allocate_lock}
       22    0.000    0.000    0.000    0.000 {built-in method
_thread.get_ident}
       11    0.001    0.000    0.001    0.000 {built-in method
_thread.start_new_thread}
        1    0.000    0.000    2.411    2.411 {built-in method builtins.exec}
       30    0.000    0.000    0.000    0.000 {built-in method builtins.getattr}
        8    0.000    0.000    0.000    0.000 {built-in method
builtins.isinstance}
        2    0.000    0.000    0.000    0.000 {built-in method builtins.len}
        2    0.000    0.000    0.000    0.000 {built-in method builtins.print}
        1    0.000    0.000    0.000    0.000 {built-in method builtins.round}
        1    0.000    0.000    0.000    0.000 {built-in method math.ceil}
        8    0.000    0.000    0.000    0.000 {built-in method posix.getpid}
```

```
        2    0.000    0.000    0.000    0.000 {built-in method time.time}
       11    0.000    0.000    0.000    0.000 {method '__enter__' of
'_thread.lock' objects}
       13    0.000    0.000    0.000    0.000 {method '__exit__' of
'_thread.RLock' objects}
       22    0.000    0.000    0.000    0.000 {method '__exit__' of
'_thread.lock' objects}
       64    0.004    0.000    0.004    0.000 {method 'acquire' of
'_thread.lock' objects}
       11    0.000    0.000    0.000    0.000 {method 'add' of 'set' objects}
       20    0.000    0.000    0.000    0.000 {method 'append' of
'collections.deque' objects}
       11    0.000    0.000    0.000    0.000 {method 'append' of 'list'
objects}
       11    0.000    0.000    0.000    0.000 {method 'difference_update' of
'set' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of
'_lsprof.Profiler' objects}
       11    0.000    0.000    0.000    0.000 {method 'discard' of 'set'
objects}
       34    0.000    0.000    0.000    0.000 {method 'locked' of '_thread.lock'
objects}
       22    0.000    0.000    0.000    0.000 {method 'release' of
'_thread.lock' objects}
        4    0.000    0.000    0.000    0.000 {method 'values' of 'dict'
objects}
```

[41]:
```
# Profiling each thread with yappi
!pip install yappi
```

```
Requirement already satisfied: yappi in /usr/local/lib/python3.10/dist-packages
(1.6.0)
```

[52]:
```
import yappi

from time import time
import numpy as np
import math
import multiprocessing as mp

# Prepare data
# value range
r = 10
# number of rows
m = 1000
```

```
# m = 5
# number of columns
n = 10000
# n = 30

np.random.seed(100)
arr = np.random.randint(0, r, size=[m, n])
data = arr.tolist()
# print(data[:10])

def howmany_within_range_rowonly(row, minimum=4, maximum=8):
  # print(mp.current_process(),' ',row) # this will print the process object␣
 ↪and the item it is working with
  count = 0
  for num in row:
     if minimum <= num <= maximum:
        count = count + 1
  return count

def mythreads_1():
  # creating 2 threads
  from multiprocessing.dummy import Pool as ThreadPool
  start_time = time()
  pool = ThreadPool(mp.cpu_count())
  results = pool.map(howmany_within_range_rowonly, [row for row in data])
  print("Using cpu_count threads: ",round(time() - start_time,8), 'seconds')
  #print(results)

def mythreads_2():
  # Other thread version, trying to divide work according to indice
  start_time = time()
  task_size = int(math.ceil(m / mp.cpu_count()))
  pool = ThreadPool(mp.cpu_count())
  print(task_size)
  for i in range(mp.cpu_count()):
    lower_row_index = i*task_size
    upper_row_index = i*task_size + task_size
    results[i] = pool.map(howmany_within_range_rowonly, [data[j] for j in␣
 ↪range(lower_row_index,upper_row_index)])
  print("Using cpu_count threads but dividing indice: ",round(time() -␣
 ↪start_time,8), 'seconds')

def seq():
  # sequential version
  start_time = time()
  results = []
  for row in data:
```

```
    results.append(howmany_within_range_rowonly(row))
  print("Sequential: ",round(time() - start_time,8), 'seconds')
    #print(results)

yappi.start()
mythreads_1()
yappi.stop()

# retrieve thread stats by their thread id (given by yappi)
#threads = yappi.get_thread_stats()
#for thread in threads:
    #print(
        #"\nFunction stats for (%s) (%d)" % (thread.name, thread.id)
    #)  # it is the Thread.__class__.__name__
    #yappi.get_func_stats(ctx_id=thread.id).print_all()
```

```
Using cpu_count threads:  1.29753304 seconds
```

Multiple threads:

[53]:
```
import yappi
from time import time
import numpy as np
import math
import threading

# Prepare data
# value range
r = 10
# number of rows
m = 1000
# m = 5
# number of columns
n = 10000
# n = 30

np.random.seed(100)
arr = np.random.randint(0, r, size=[m, n])
data = arr.tolist()
# print(data[:10])

def howmany_within_range_rowonly(row, minimum=4, maximum=8):
  # print(mp.current_process(),' ',row) # this will print the process object␣
 ↪and the item it is working with
  count = 0
  for num in row:
     if minimum <= num <= maximum:
        count = count + 1
```

```
  return count

def mythreads_1():
  # creating 2 threads
  start_time = time()
  results = []
  threads = []
  for row in data:
    thread = threading.Thread(target=results.append,␣
 ↪args=(howmany_within_range_rowonly(row),))
    threads.append(thread)
    thread.start()
  for thread in threads:
    thread.join()
  print("Using cpu_count threads: ",round(time() - start_time,8), 'seconds')
  #print(results)

def mythreads_2():
  # Other thread version, trying to divide work according to indice
  start_time = time()
  task_size = int(math.ceil(m / threading.active_count()))
  print(task_size)
  results = []
  threads = []
  for i in range(threading.active_count()):
    lower_row_index = i*task_size
    upper_row_index = i*task_size + task_size
    thread = threading.Thread(target=results.extend,␣
 ↪args=([howmany_within_range_rowonly(j) for j in data[lower_row_index:
 ↪upper_row_index]],))
    threads.append(thread)
    thread.start()
  for thread in threads:
    thread.join()
  print("Using cpu_count threads but dividing indice: ",round(time() -␣
 ↪start_time,8), 'seconds')

def seq():
  # sequential version
  start_time = time()
  results = []
  for row in data:
    results.append(howmany_within_range_rowonly(row))
  print("Sequential: ",round(time() - start_time,8), 'seconds')
    #print(results)

yappi.start()
```

```
mythreads_1()
yappi.stop()

# retrieve thread stats by their thread id (given by yappi)
#threads = yappi.get_thread_stats()
#for thread in threads:
    #print(
        #"\nFunction stats for (%s) (%d)" % (thread.name, thread.id)
    #)  # it is the Thread.__class__.__name__
    #yappi.get_func_stats(ctx_id=thread.id).print_all()
```

```
Using cpu_count threads:  1.80397725 seconds
```

13 Q6: For what kind of tasks should you use processes and when should you use threads?

We should use processes for tasks that must be executed in isolation and threads for tasks that need frequent communication and shared resources. Threads has a more efficient resource usage and, consequently, lower overhead.

Below, you can find results when I ran these experiments in my own machine.

Execution times and speedups running on an AMD FX(tm)-8120 Eight-Core Processor (1.4GHz), 16 GBytes RAM, for a matrix with dimension 1000 x 100000.

```
##########################################################################
Sequential  7.727450847625732 seconds  [50038, 50181, 50084, 50103, 49721, 50100,
50345, 50090, 50007, 49888] ##########################################################################
pool.apply            20.577924489974976 seconds           [50038, 50181, 50084,
50103, 49721, 50100, 50345, 50090, 50007, 49888]      Slowdown:             2.66
##########################################################################
pool.map          4.117670059204102 seconds          [50038, 50181, 50084, 50103,
49721, 50100, 50345, 50090, 50007, 49888]        Speedup:                1.88
##########################################################################
pool.starmap            4.02571177482605 seconds           [50038, 50181, 50084,
50103, 49721, 50100, 50345, 50090, 50007, 49888]      Speedup:              1.92
##########################################################################
pool.apply_async          3.945971965789795 seconds          [50038, 50181, 50084,
50103, 49721, 50100, 50345, 50090, 50007, 49888]      Speedup:              1.96
########################################################################## Process
(creating fewer processes - 2nd solution above) 1.87694931 seconds Speedup: 4.11
```

Other run by the same machine running the codes on a matrix of dimension 1000 x 500000

```
##########################################################################
Sequential   38.80920100212097 seconds   [250127, 250430, 250285, 249630, 249829,
250269, 250135, 249801, 250431, 249623] ##########################################################
pool.apply       71.33826541900635 seconds       [250127, 250430, 250285, 249630,
249829, 250269, 250135, 249801, 250431, 249623]      Slowdown:             1.83
```

```
##########################################################################
pool.map        17.257094383239746 seconds        [250127, 250430, 250285, 249630,
249829, 250269, 250135, 249801, 250431, 249623]      Speedup:              2.25
##########################################################################
pool.starmap          15.533486604690552 seconds          [250127, 250430, 250285,
249630, 249829, 250269, 250135, 249801, 250431, 249623]    Speedup:          2.50
##########################################################################
pool.apply_async         17.82588529586792 seconds         [250127, 250430, 250285,
249630, 249829, 250269, 250135, 249801, 250431, 249623]    Speedup:          2.18
##########################################################################
```

Processes (creating fewer processes - 2nd solution above) 12.95342517 seconds **Speedup: 2.99**

[ ]:
```
import yappi

from time import time
import numpy as np
import math
import multiprocessing as mp

def mythreads_1():
  # creating 2 threads
  from multiprocessing.dummy import Pool as ThreadPool
  print(mp.cpu_count())
  pool = ThreadPool(mp.cpu_count())
  start_time = time()
  results = pool.map(sum,[row for row in data])
  print("Using cpu_count threads: ",round(time() - start_time,8), 'seconds')
  #print(results)

def mythreads_2():
  # Other thread version, trying to divide work according to indice
  from multiprocessing.dummy import Pool as ThreadPool
  start_time = time()
  task_size = int(math.ceil(m / mp.cpu_count()))
  pool = ThreadPool(mp.cpu_count())
  print(task_size)
  for i in range(mp.cpu_count()):
    lower_row_index = i*task_size
    upper_row_index = i*task_size + task_size
    results[i] = pool.map(howmany_within_range_rowonly, [data[j] for j in␣
 ↪range(lower_row_index,upper_row_index)])
  print("Using cpu_count threads but dividing indice: ",round(time() -␣
 ↪start_time,8), 'seconds')

def seq():
  # sequential version
```

```
  start_time = time()
  results = []
  for row in data:
    results.append(sum(row))
  print("Sequential: ",round(time() - start_time,8), 'seconds')
    #print(results)

yappi.start()
mythreads_1()
yappi.stop()
seq()

# # retrieve thread stats by their thread id (given by yappi)
# threads = yappi.get_thread_stats()
# for thread in threads:
#     print(
#         "\nFunction stats for (%s) (%d)" % (thread.name, thread.id)
#     )  # it is the Thread.__class__.__name__
#     yappi.get_func_stats(ctx_id=thread.id).print_all()
```

```
2
Using cpu_count threads:  0.13097906 seconds
Sequential:  0.12780094 seconds
```