

並列分散処理 最終レポート

チーム E 大城 慶知 眞榮城 隆守 伊波卓浩 宮良友也

July 21, 2018

最終報告書に載せるやつ (あとで消すやつ)

演習の背景、目的、方法、結果、考察を A410 ページ以内で適切にまとめる。個々のメンバーの役割分担を明記する。記載がない場合、あるいは、曖昧な場合には、減点の対象となる。例えば、あるタスクを複数名で担当した場合でも、個々のメンバーの役割をできる限り区別して説明する。最終報告書にはソースコードの github リポジトリも記載する。

1 演習の背景

「講義で説明した並列分散処理を実践し、他者に有益な情報となるように共有せよ。」という課題が渡された。

b3 前期はメンバーが忙しく時間も取れないので軽量かつ有益な情報をということで、GPU マシンを使った並列処理を断腸の思いで断念し、Python における非同期処理を用いた I/O の並列処理を行うことにした。

2 目的

Python のスレッドと concurrent を用いて並列化を行うとともに、async await の使うことで非同期処理を行いさらに速度向上を目指す。

3 Python の事前知識

3.1 スレッドの制約

Python では、GIL(Global Interpreter Lock) と呼ばれる制約がある。GIL とは、Figure 1 と Figure2 のように、CPU バウンドで Python を実行した際に一つだけしかスレッドのリソースを起動できない制約である。そのため、Python の CPU バウンドの並列処理はプロセスを使って、I/O バウンドの並列処理はスレッドを行うことが多い。

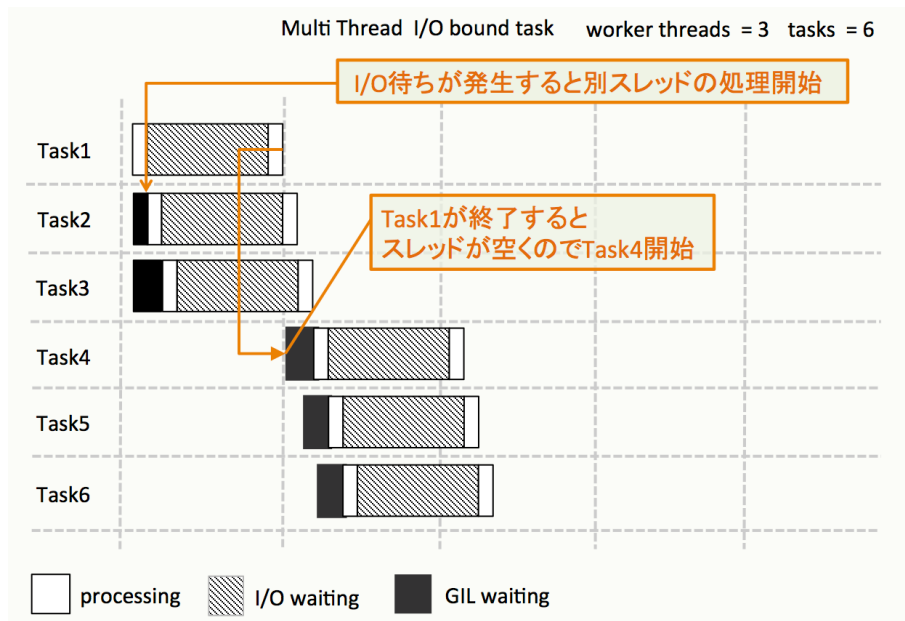


Figure 1: マルチスレッド I/O バウンド

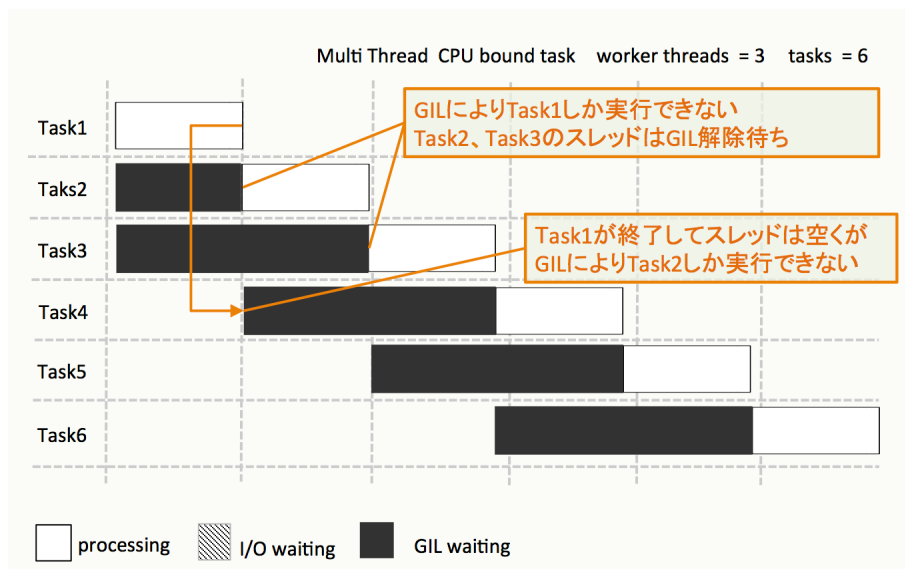


Figure 2: マルチスレッド CPU バウンド

3.2 async と await

async と await がどのような処理を行うのか以下のコードを実行して動作を確認した。

コード 1: 非同期処理の FizzBuzz

```
1 # -*- coding: utf-8 -*-
2 # from asyncio import Queue
3 # from queue import Queue
```

```

4 import asyncio
5
6 def fizzbuzz(i):
7     if i == 15:
8         return 'FizzBuzz'
9     if i % 5 == 0:
10        return 'Buzz'
11    if i % 3 == 0:
12        return 'Fizz'
13    return str(i)
14
15 async def task_fizzbuzz(prefix):
16     for x in range(1, 31):
17         await asyncio.sleep(0.3)
18         print(prefix + "{}:".format(str(x)) + fizzbuzz(x))
19     return None
20
21 loop = asyncio.get_event_loop()
22 # コルーチン個生成 1000
23 tasks = asyncio.wait([task_fizzbuzz(str(i) + ":") for i in range(1, 1000)])
24 loop.run_until_complete(tasks)
25 loop.close()

```

コード 2: FizzBuzz 実行結果

```

1 75219:6:Fizz
2 8282:6:Fizz
3 57464:6:Fizz
4 75220:6:Fizz
5 8283:6:Fizz
6 57465:6:Fizz
7 75221:6:Fizz
8 8284:6:Fizz
9 57466:6:Fizz
10 75222:6:Fizz
11 8285:6:Fizz
12 57467:6:Fizz
13 75223:6:Fizz

```

実行結果から FizzBuzz のコードをスリープさせているのにもかかわらず、

4 実験方法

HTTP の GET を用いて実験を行った。GET を複数回実行する場合を考えると、逐次処理の場合ではレスポンスがあるまで次の GET を送信することができない。これを並列処理によりレスポンスを待つことなく次の GET を実行した。これにより効率よく GET を実行し、結果を受け取ることができると想定した。

コード 3: 逐次処理

```

1 # Example 1: synchronous requests
2 import requests
3 import time
4
5 start = time.time()
6
7 num_requests = 20
8
9 def http_get():
10     requests.get('https://ie.u-ryukyu.ac.jp/')
11     print(time.time() - start)
12
13 responses = [
14     http_get()
15     for i in range(num_requests)
16 ]

```

コード 4: async-await のみ

```

1 # Example 2: asynchronous requests
2 import asyncio
3 import requests
4 import time
5
6 start = 0
7
8 async def main():
9     start = time.time()
10
11     loop = asyncio.get_event_loop()
12     futures = [
13         loop.run_in_executor(
14             None,
15             requests.get,
16             'https://ie.u-ryukyu.ac.jp/'
17         )
18         for i in range(20)
19     ]
20     for response in await asyncio.gather(*futures):
21         print(time.time() - start)
22         pass
23
24 loop = asyncio.get_event_loop()
25 loop.run_until_complete(main())

```

コード 5: async-await のみ (5 回ずつ)

```

1 # Example 2: asynchronous requests
2 import asyncio
3 import requests
4 import time
5
6 start = 0
7
8 async def main():
9     start = time.time()
10
11     for num in range(4):
12         loop = asyncio.get_event_loop()
13         futures = [
14             loop.run_in_executor(
15                 None,
16                 requests.get,
17                 'https://ie.u-ryukyu.ac.jp/'
18             )
19             for i in range(5)
20         ]
21         for response in await asyncio.gather(*futures):
22             print(time.time() - start)
23             pass
24
25 loop = asyncio.get_event_loop()
26 loop.run_until_complete(main())

```

コード 6: async-await + MultiPoolExector

```

1 # Example 3: asynchronous requests with larger thread pool
2 import asyncio
3 import concurrent.futures
4 import requests
5 import time
6
7 start = 0
8

```

```

9  async def main():
10     start = time.time()
11
12     with concurrent.futures.ThreadPoolExecutor(max_workers=20) as executor:
13
14         loop = asyncio.get_event_loop()
15         futures = [
16             loop.run_in_executor(
17                 executor,
18                 requests.get,
19                 'https://ie.u-ryukyu.ac.jp/'
20             )
21             for i in range(20)
22         ]
23         for response in await asyncio.gather(*futures):
24             print(time.time() - start)
25         pass
26
27
28 loop = asyncio.get_event_loop()
29 loop.run_until_complete(main())

```

5 実行結果

example1.py は逐次処理をしてくれるスクリプト。
example2.py はコルーチンを 20 個一気に呼び出す。
example3.py はコルーチンを 5 個ずつ 4 回呼び出している。
example4.py はスレッドを使用。

実行結果を以下に示す。

コード 7: example1.py

```

1  1.7703397274017334
2  3.2814269065856934
3  4.895288705825806
4  6.453948736190796
5  8.205788850784302
6  9.720278978347778
7  11.497416973114014
8  13.36806869506836
9  15.045061826705933
10 16.66911792755127
11 18.479264974594116
12 20.14133882522583
13 21.714859008789062
14 23.392934799194336
15 24.951546907424927
16 26.690969705581665
17 28.501687049865723
18 30.250661849975586
19 31.964433908462524
20 33.642555713653564

```

コード 8: example2.py

```

1  6.742665767669678
2  6.7426910400390625
3  6.742694139480591
4  6.742694854736328
5  6.742696762084961
6  6.7426979541778564
7  6.742699861526489

```

```

8 | 6.742701053619385
9 | 6.742702960968018
10 | 6.742703914642334
11 | 6.742705821990967
12 | 6.742706775665283
13 | 6.742708921432495
14 | 6.7427098751068115
15 | 6.742712020874023
16 | 6.74271297454834
17 | 6.742713928222656
18 | 6.742715835571289
19 | 6.742717742919922
20 | 6.742718935012817

```

コード 9: example3.py

```

1 | 1.8773348331451416
2 | 1.877375841140747
3 | 1.8773789405822754
4 | 1.8773798942565918
5 | 1.8773820400238037
6 | 3.4774909019470215
7 | 3.477499008178711
8 | 3.477501153945923
9 | 3.4775028228759766
10 | 3.477504014968872
11 | 5.168514013290405
12 | 5.16852593421936
13 | 5.168529033660889
14 | 5.1685309410095215
15 | 5.168533802032471
16 | 6.861561059951782
17 | 6.861573934555054
18 | 6.861576795578003
19 | 6.861578941345215
20 | 6.861582040786743

```

コード 10: example4.py

```

1 | 6.301663875579834
2 | 6.301697015762329
3 | 6.301699161529541
4 | 6.30170202255249
5 | 6.301703214645386
6 | 6.3017048835754395
7 | 6.301707029342651
8 | 6.301707983016968
9 | 6.30171012878418
10 | 6.3017120361328125
11 | 6.301713228225708
12 | 6.301714897155762
13 | 6.301717042922974
14 | 6.30171799659729
15 | 6.301720142364502
16 | 6.301722049713135
17 | 6.301723003387451
18 | 6.301724910736084
19 | 6.301727056503296
20 | 6.301728010177612

```

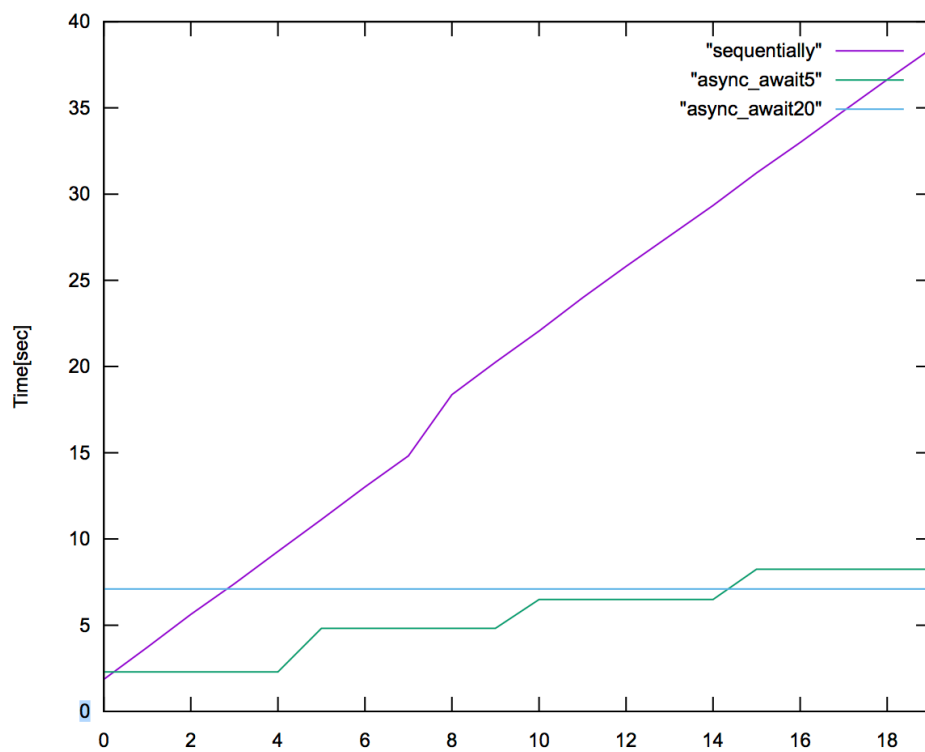


Figure 3: 実行結果

こうなりました。

6 考察

7 感想・意見

GitHub の URL

<https://github.com/e165719/ParallelDistributedProcessing>