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 並列処理の基礎知識

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 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事が多い。

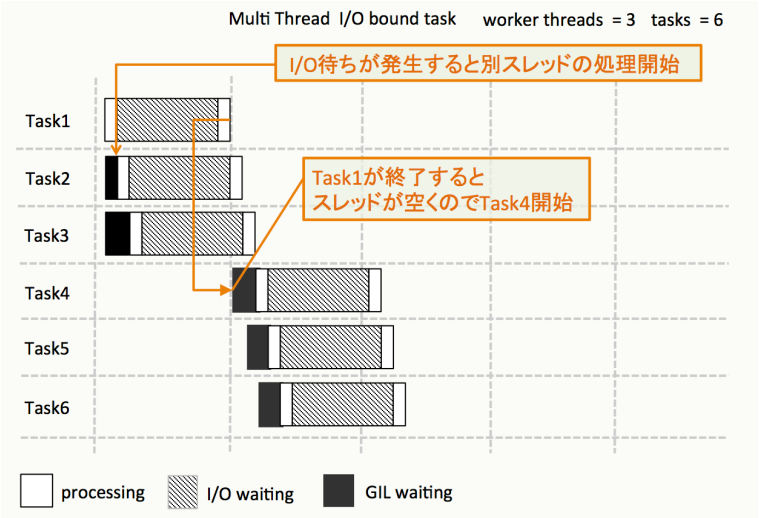

Figure 1: マルチスレッド I/O バウンド

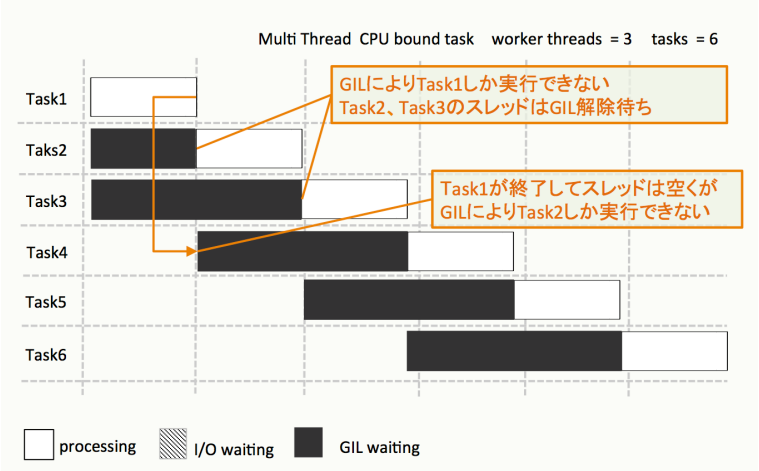

Figure 2: マルチスレッド CPU バウンド

3.2 async と await

3.3 プロセスを用いた

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。

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 1: example1.py

```
1.7703397274017334
3.2814269065856934
4.895288705825806
6.453948736190796
8.205788850784302
9.720278978347778
11.497416973114014
13.36806869506836
15.045061826705933
16.66911792755127
18.479264974594116
20.14133882522583
21.714859008789062
23.392934799194336
24.951546907424927
26.690969705581665
28.501687049865723
30.250661849975586
31.964433908462524
33.642555713653564
```

コード 2: example2.py

```
6.742665767669678
6.7426910400390625
6.742694139480591
6.742694854736328
6.742696762084961
6.7426979541778564
6.742699861526489
6.742701053619385
6.742702960968018
6.742703914642334
6.742705821990967
6.742706775665283
6.742708921432495
6.7427098751068115
6.742712020874023
6.74271297454834
6.742713928222656
6.742715835571289
6.742717742919922
6.742718935012817
```

コード 3: example3.py

```
1.8773348331451416
1.877375841140747
1.8773789405822754
1.8773798942565918
1.8773820400238037
3.4774909019470215
3.477499008178711
3.477501153945923
```

```
3.4775028228759766
3.477504014968872
5.168514013290405
5.16852593421936
5.168529033660889
5.1685309410095215
5.168533802032471
6.861561059951782
6.861573934555054
6.861576795578003
6.861578941345215
6.861582040786743
```

コード 4: example4.py

```
6.301663875579834
6.301697015762329
6.301699161529541
6.30170202255249
6.301703214645386
6.30170488835754395
6.301707029342651
6.301707983016968
6.30171012878418
6.30171203613281 25
6.301713228225708
6.301714897155762
6.301717042922974
6.30171799659729
6.301720142364502
6.301722049713135
6.301723003387451
6.301724910736084
6.301727056503296
6.301728010177612
```

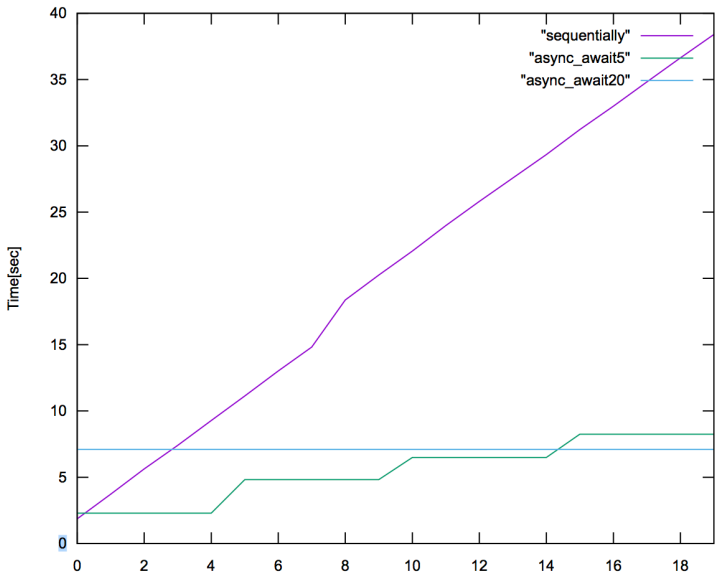

Figure 3: 実行結果

こうなりました。

6　考察

7　感想・意見

GitHub の URL

https://github.com/e165719/ParallelDistributedProcessing