並列分散処理

最終課題発表(07/31)

A班　石原巧斗　真地長一郎　銘苅理斗
玉城優太朗　大城紳之亮　西原康貴

課題テーマ

並列化による配列ソートの性能比較

並列化することによって、非並列時より配列ソートの速度は上がるのか？

また、言語別(Python,C++,Go)での並列化の速度の違いはどのくらいあるのか？

動作環境

MacBook Pro (2016)

プロセッサ	2Ghz Intel Core i5
メモリ	8GB 1867 MHz LPDDR3
OS	mac0S Mojave ver10.14.3

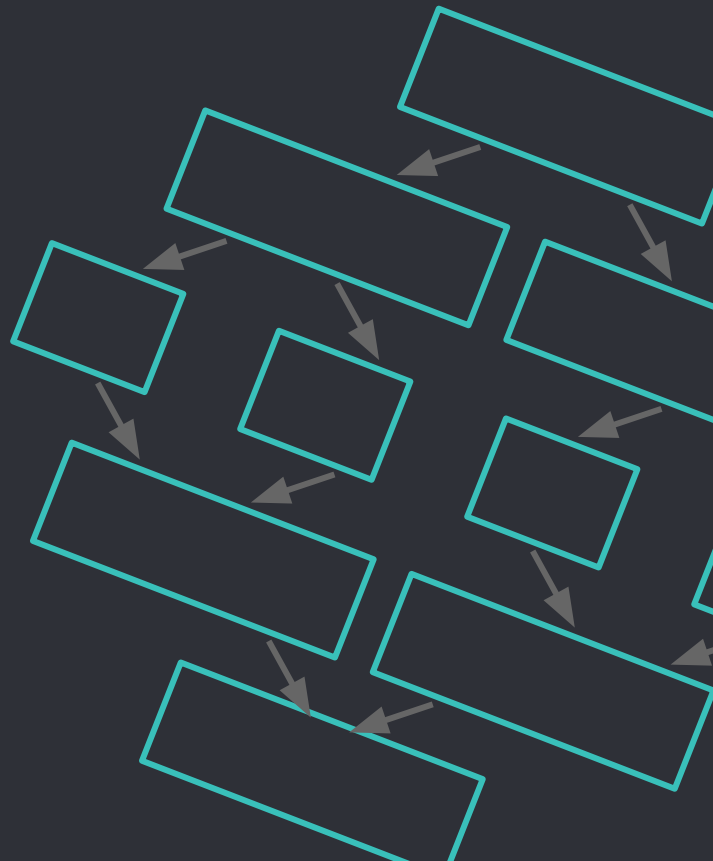
実験概要

1. それぞれの言語でマージソート、バケットソートを並列化あり、なしの処理時間を(計測5回の平均値)計測し比較する。

   計測はデータ分割からソートのみの時間とする。処理する際のコア数は2で固定とし、ソートする配列データは共通のものを使用する。

2. データを元にグラフを生成し、考察する。

1 Merge Sort編



Pythonによる実行

使用パッケージ：　multiprocessing　　使用モジュール：　Pool,time,sys,csv

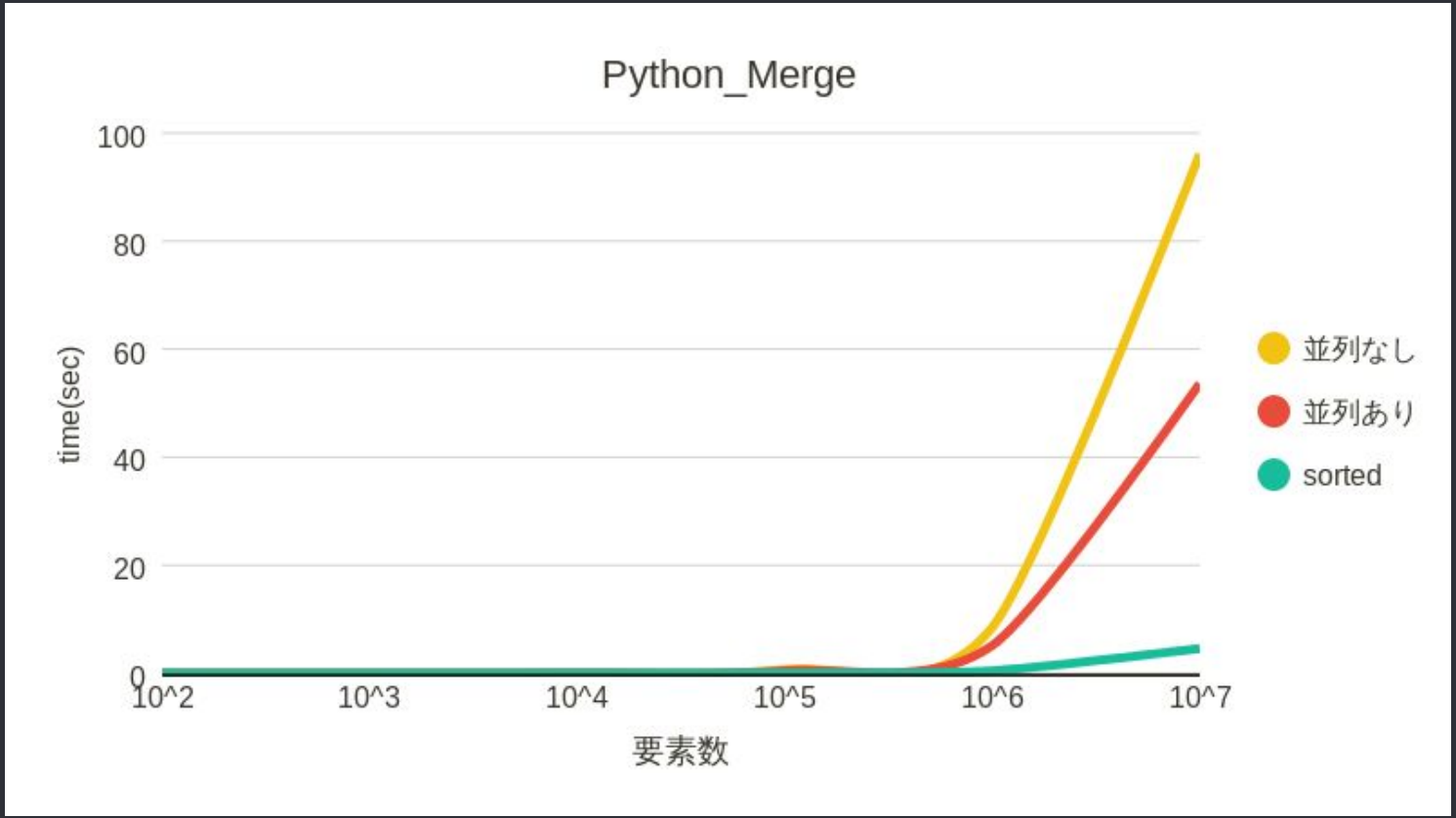
sortedとは…　iterableの要素を並び替えた新たなリストを返す組み込み関数である。

リスト型のメソッドである sort()と同様に昇順、降順にソートする。

要素数	10^2	10^3	10^4	10^5	10^6	10^7
並列化なし	0.0003954	0.0046872	0.0563916	0.66888	8.500405	96.1093378
並列化あり	0.0211262	0.245882	0.558604	0.461971	5.1284148	53.6284834
差分	+0.021	+0.241	+0.502	-0.207	-3.372	-42.481
sorted	0.0000438	0.0002682	0.0031942	0.0342698	0.4491218	4.5656456

※差分のみ小数第4位で四捨五入

Pythonによる実行



C++による実行

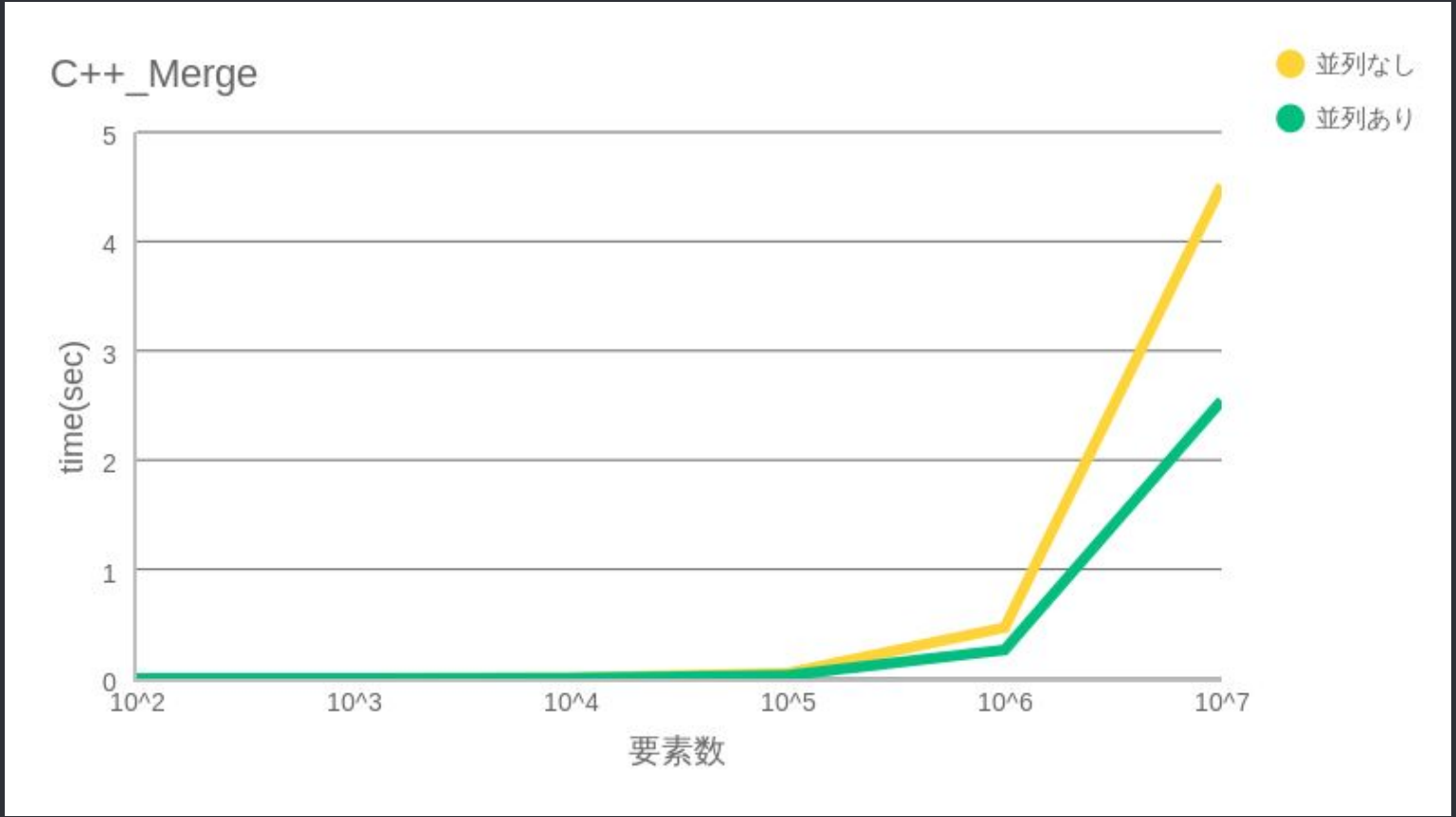
使用API： OpenMP(Open Multi-Processing)

OpenMpについて… 共有メモリ型マシンで並列プログラムを可能にする API。
ディレクティブを挿入するだけで並列化が可能。

要素数	10^2	10^3	10^4	10^5	10^6	10^7
並列化なし	0.00006961	0.00057616	0.00590601	0.04795266	0.4660998	4.513736
並列化あり	0.00017104	0.00046763	0.00407782	0.02854558	0.2613796	2.54235
差分	+0.2001	-0.0001	-0.0018	-0.0194	-0.2047	-1.9714

※差分のみ小数第5位で四捨五入

C++による実行

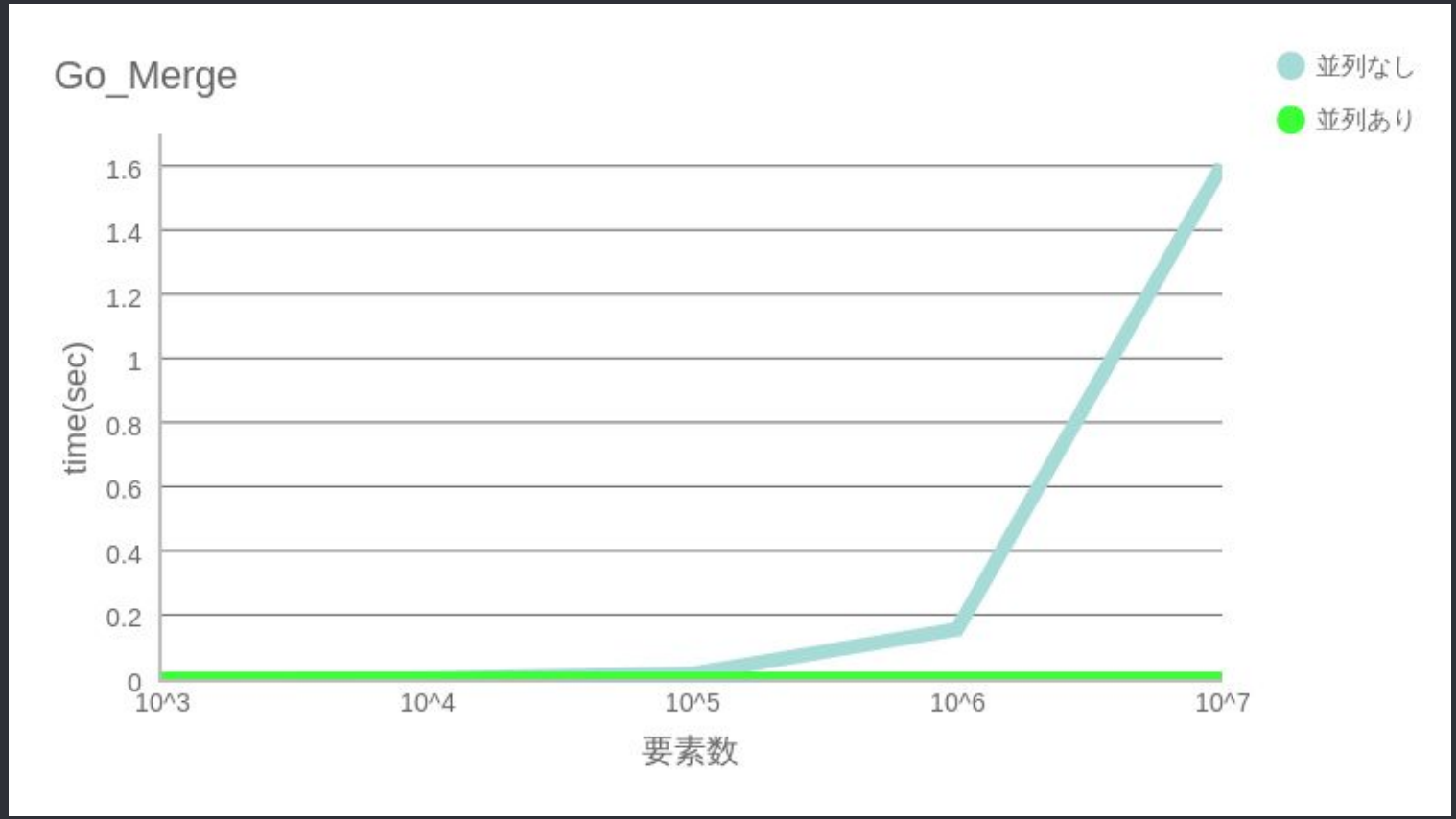


Goによる実行

Go言語について… 静的型付け、C言語の伝統に則ったコンパイル言語、メモリ安全性、ガベージコレクション、構造的型付け、CSPスタイルの並行性などの特徴を 持つ。

要素数	10^2	10^3	10^4	10^5	10^6	10^7
並列化なし	0.0000246	0.0001596	0.0017854	0.015491	0.1543366	1.6015802
並列化あり	0.0000066	0.0000114	0.0000142	0.0000116	0.0000144	0.0000214
差分	-0.000018	-0.001482	-0.0017712	-0.0154794	0.1543222	-1.6015588

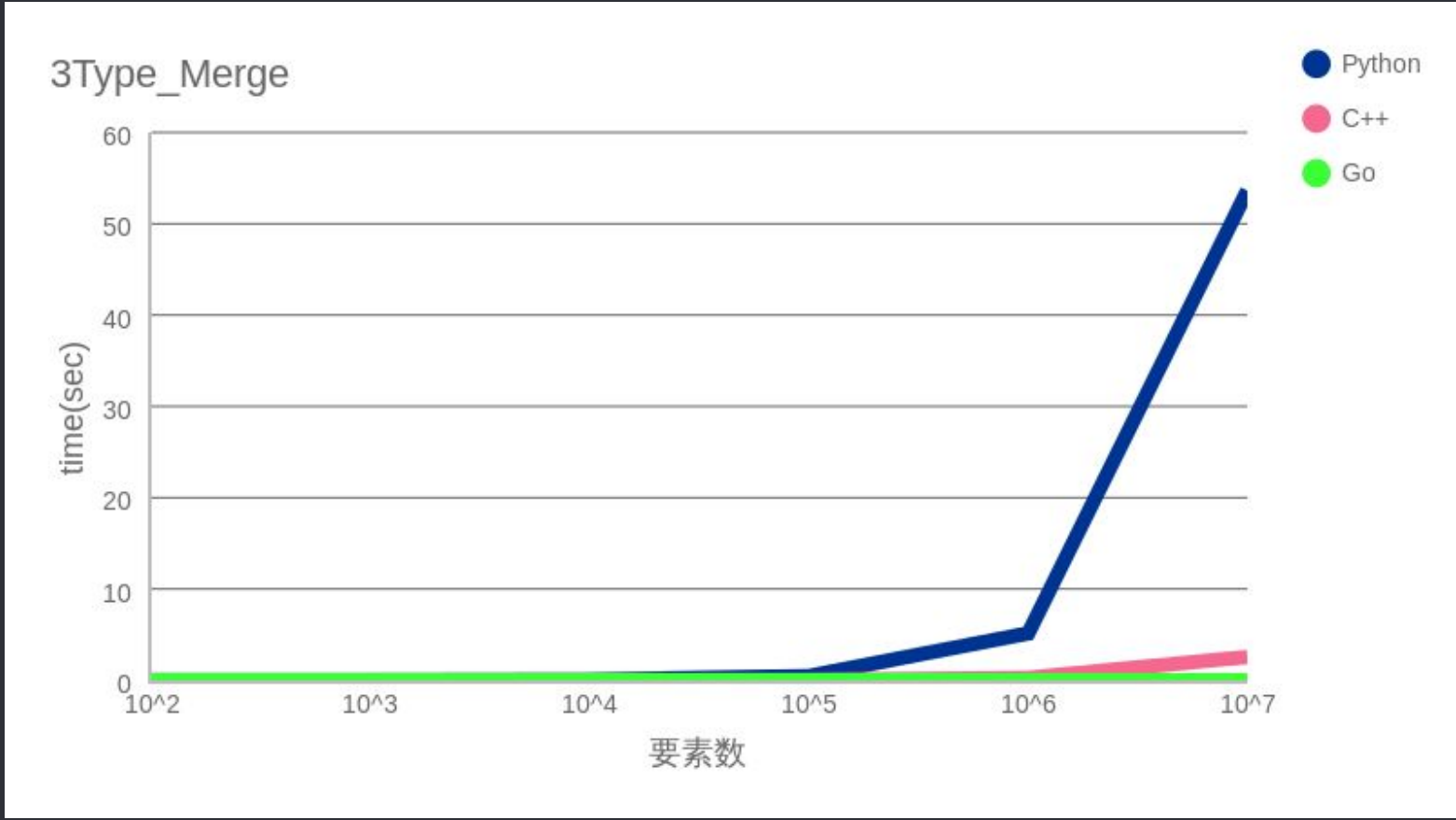
Goによる実行


Go_Merge
並列なし
並列あり
time(sec)
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
10^3
10^4
10^5
10^6
10^7
要素数

3言語の比較

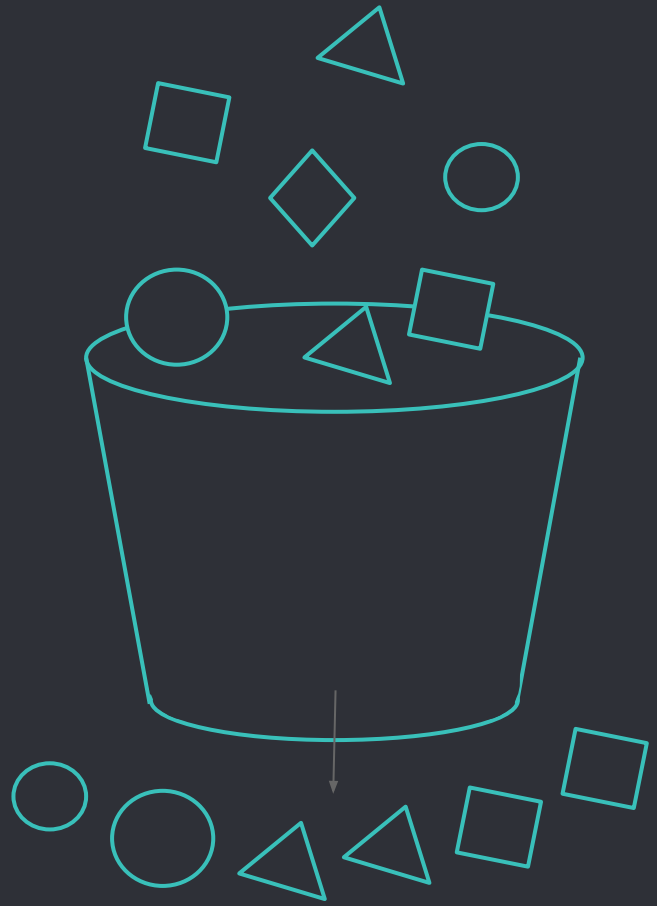
要素数	10^2	10^3	10^4	10^5	10^6	10^7
Python	0.0211262	0.245882	0.558604	0.461971	5.1284148	53.6284834
C++	0.00017104	0.00046763	0.00407782	0.02854558	0.2613796	2.54235
Go	0.0000066	0.0000114	0.0000142	0.0000116	0.0000144	0.0000214

3言語の比較


3Type_Merge
Python
C++
Go
60
50
40
30
20
10
0
time(sec)
10^2
10^3
10^4
10^5
10^6
10^7
要素数

2

Bucket Sort編



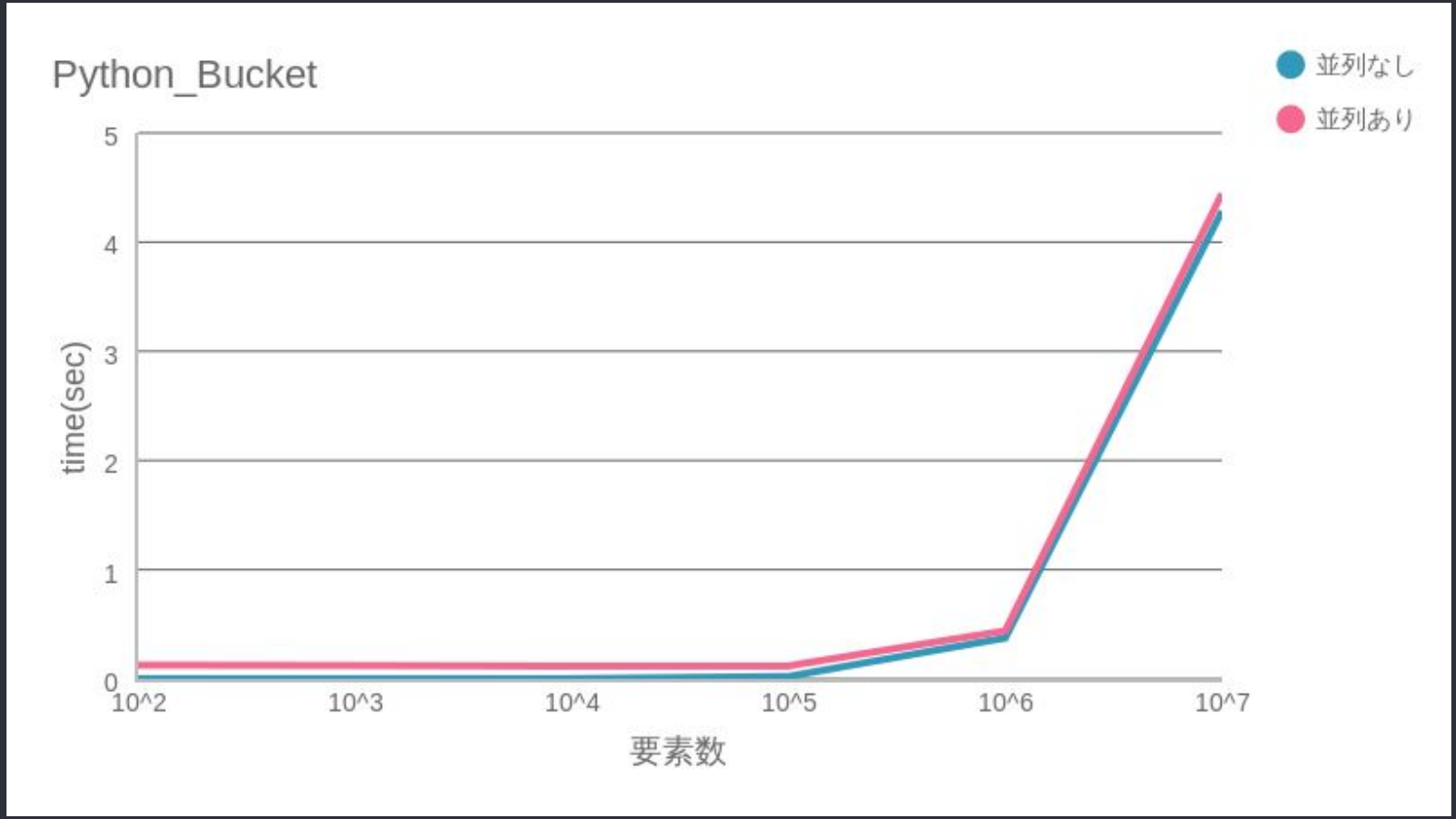
Pythonによる実行

使用パッケージ： multiprocessing　　使用モジュール： Pool,time,sys,csv

要素数	10^2	10^3	10^4	10^5	10^6	10^7
並列化なし	0.00047206	0.00018096	0.0017297	0.021328	0.374088	4.2835209
並列化あり	0.125315	0.1218361	0.116219	0.1176011	0.439441	4.4430248
差分	+0.125	+0.122	+0.114	+0.096	+0.065	+0.160

※差分のみ小数第4位で四捨五入

Pythonによる実行



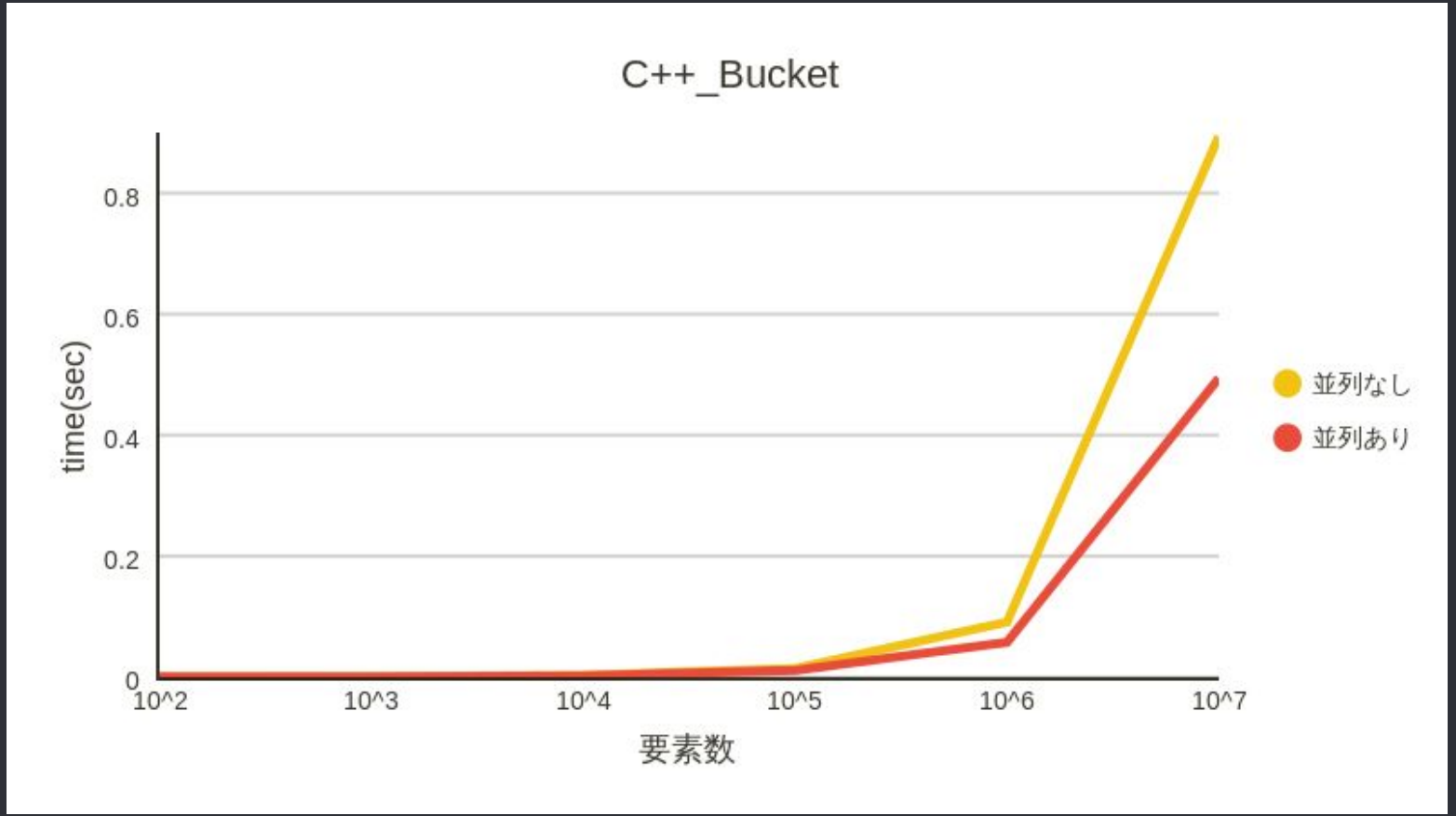
C++による実行

使用API： OpenMP(Open Multi-Processing)

並列化したいタスクの前にOpenMP指示文を書くことで実装

要素数	10^2	10^3	10^4	10^5	10^6	10^7
並列化なし	0.000069	0.000297	0.002887	0.0141	0.090874	0.893702
並列化あり	0.000365	0.000439	0.002057	0.010855	0.057421	0.494106
差分	0.000296	0.000142	-0.00083	-0.003245	-0.033453	-0.399596

C++による実行


C++_Bucket
time(sec)
0.8
0.6
0.4
0.2
0
10^2
10^3
10^4
10^5
10^6
10^7
要素数
並列なし
並列あり

バケツ数とスレッド数について

緑：バケツ数＝スレッド数 赤：バケツ数＜スレッド数

スレッド数＼バケツ数	1	2	4	8	16	32
1	0.941	0.894	0.825	0.744	0.678	0.616
2	0.940	0.494	0.465	0.425	0.317	0.365
4	0.934	0.495	0.296	0.278	0.266	0.245
8	0.959	0.522	0.324	0.230	0.217	0.205
16	0.989	0.526	0.333	0.230	0.220	0.206
32	0.973	0.534	0.321	0.228	0.224	0.206

データを分割した数以上にスレッドを生成するとスレッドが不活性状態になり，　処理時間を短縮できない．

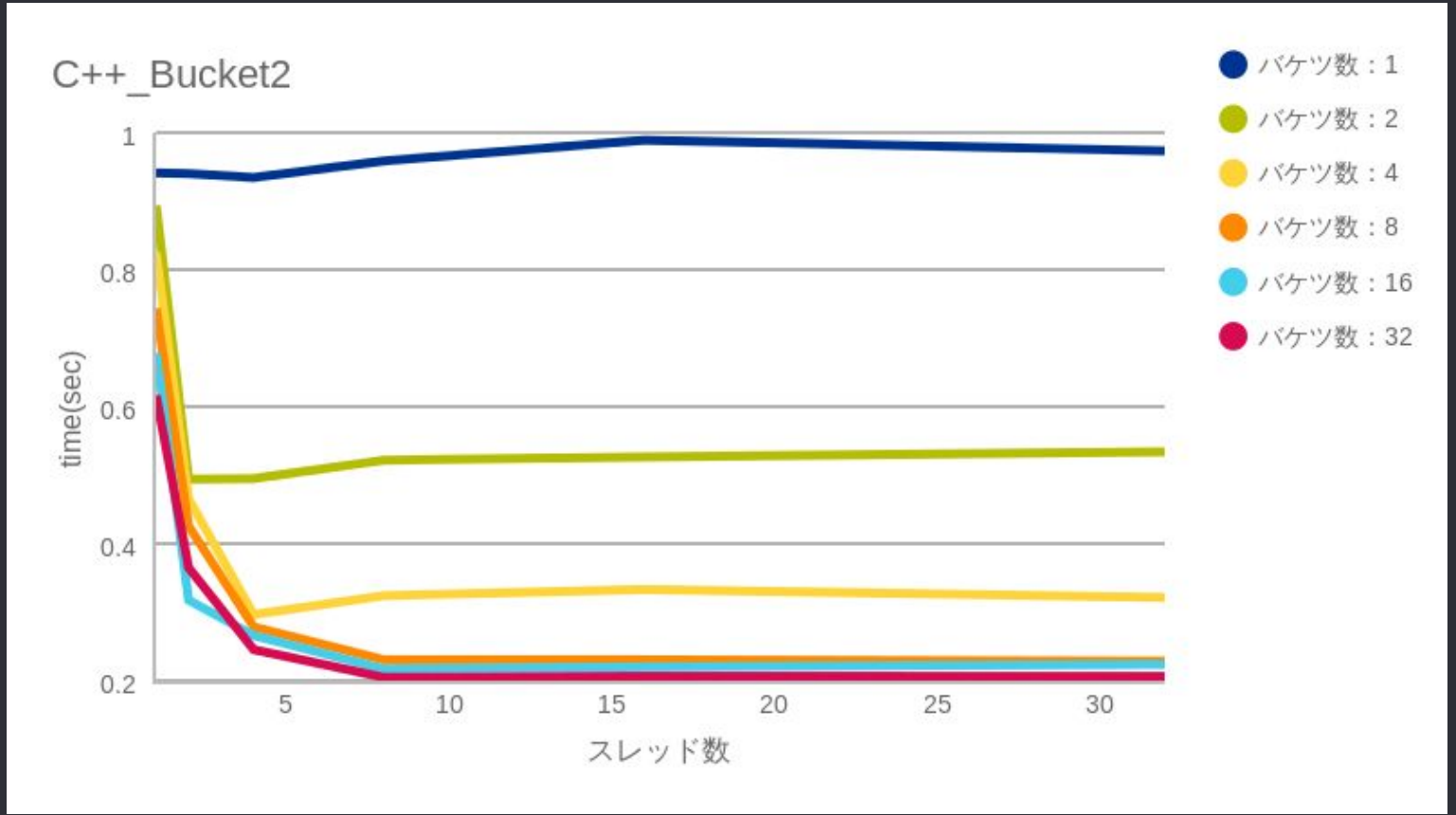
バケツ数とスレッド数について

実行ハード：4コア8スレッド　　緑：スレッド数8　赤：スレッド数16，32

スレッド数＼バケツ数	1	2	4	8	16	32
1	0.941	0.894	0.825	0.744	0.678	0.616
2	0.940	0.494	0.465	0.425	0.317	0.365
4	0.934	0.495	0.296	0.278	0.266	0.245
8	0.959	0.522	0.324	0.230	0.217	0.205
16	0.989	0.526	0.333	0.230	0.220	0.206
32	0.973	0.534	0.321	0.228	0.224	0.206

ハードウェアの性能以上のスレッドを生成しても処理速度は上昇しない．

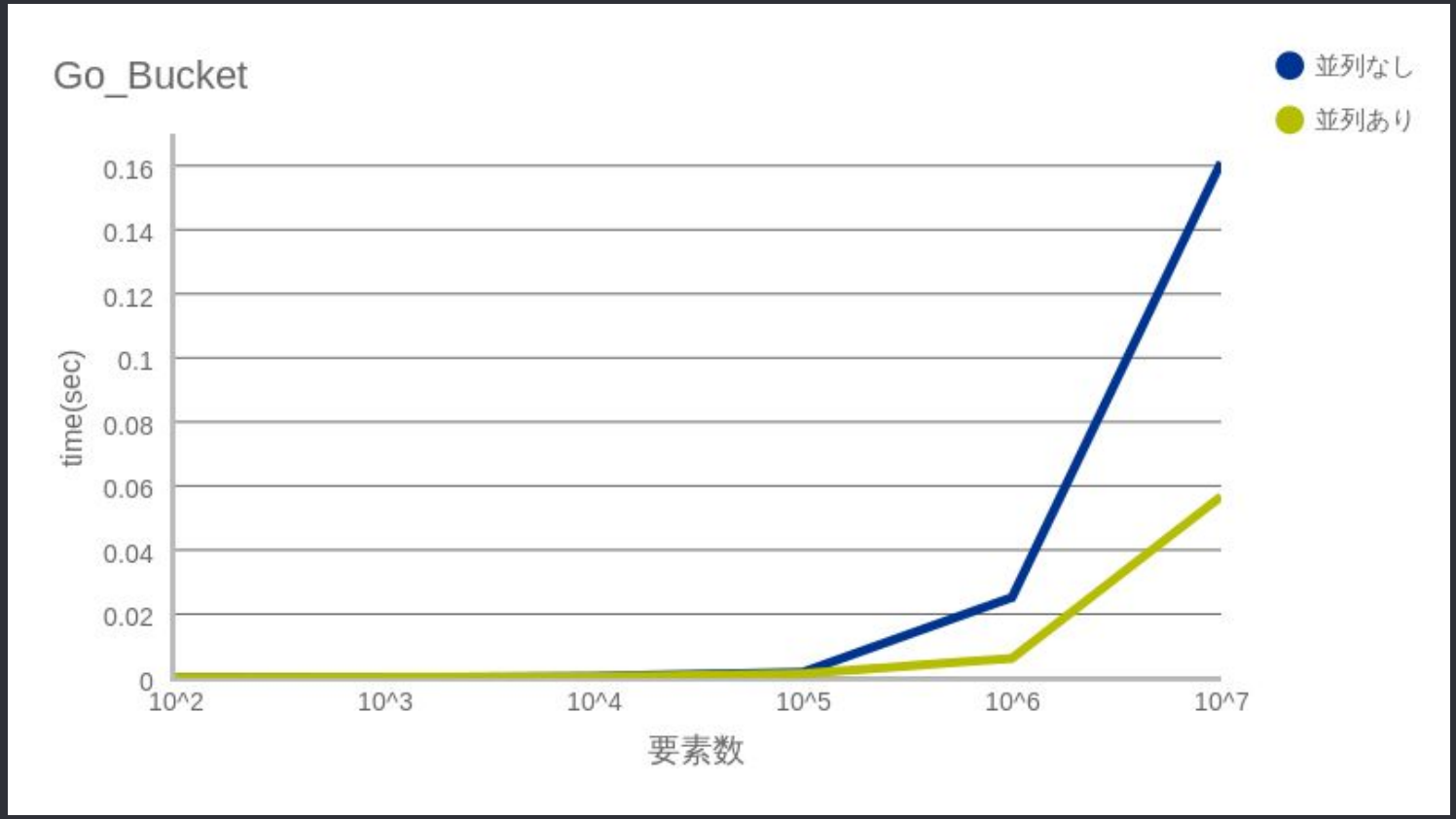
バケツ数・スレッド数について



Goによる実行

要素数	10^2	10^3	10^4	10^5	10^6	10^7
並列化なし	0.0000936	0.0001352	0.0005262	0.0018016	0.025185	0.1610546
並列化あり	0.0001494	0.0002604	0.0003884	0.0013668	0.0061688	0.0567012
差分	+0.0000558	+0.0001252	-0.0001378	-0.0004348	-0.0190162	-0.1043534

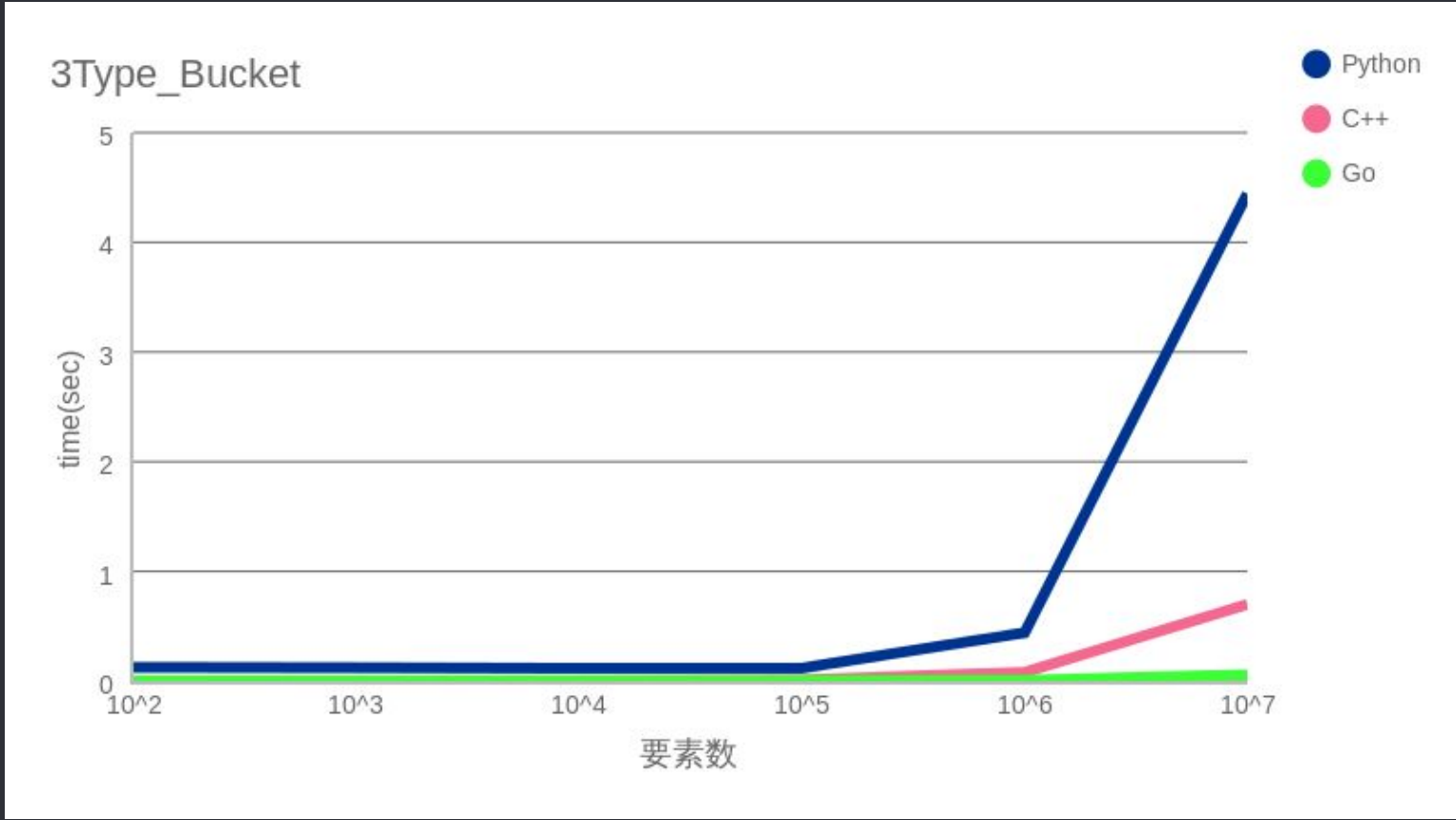
Goによる実行


Go_Bucket
並列なし
並列あり
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
time(sec)
10^2
10^3
10^4
10^5
10^6
10^7
要素数

3言語の比較

要素数	10^2	10^3	10^4	10^5	10^6	10^7
Python	0.125315	0.1218361	0.116219	0.1176011	0.439441	4.4430248
C++	0.000365	0.000439	0.002057	0.010855	0.057421	0.494106
Go	0.0001494	0.0002604	0.0003884	0.0013668	0.0061688	0.0567012

3言語の比較


3Type_Bucket
Python
C++
Go
time(sec)
0
1
2
3
4
5
10^2
10^3
10^4
10^5
10^6
10^7
要素数

考察

Pythonについて

・今回の実習ではPythonがC++,Goと比べかなり遅かった。これは Pythonが動的型付けだからではないかと思う。また、インタプリタ言語とコンパイル言語の違いも処理の速さに関係している？

・Pythonでの並列化について、並列したい関数 (処理)を指定し、使用するコア数、プ　ロセス数もしくはスレッド数などを決めるだけなので、実は書きやすいかもしれない。が、一連のプログラムのどの関数を並列化するかで、結果にかなり影響する。ましてや処理の時間がかなりかかるので注意しながら設計しなければならない。

・PythonにはGIL(Global Interpreter Lock)というインタプリタ上で一度に一つのスレッドのみが動作することを保証する機能がある。そのため、 Threadによる並列化ができないので基本的にはProcessを用いて並列化を行う。

考察

C++について

・OpenMPを用いることで単純なループを容易に並列化することができ，ソート処理を高速に行なうことができた．実用アプリケーションにおける複雑なループは OpenMP化に向いていないことがあり，複雑な処理はPthread等を活用して並列化する必要があるかもしれない．

・1つの変数に並列でアクセスする時には注意して実装する必要がある．本演習では各スレッドごとに別の変数を確保することでバグを防いだ．

・データを分割した数以上にスレッドを生成するとスレッドが不活性状態になる．生成したスレッドを最大限活かすためにデータ分割を工夫する必要がある．

・プログラム内でスレッド数を増やしてもハードウェアのコア数・スレッド数によって処理速度に限界がくる．

考察

Goについて

- GolangはpythonやC++とソートでの並列化において比較し速度が速いことがわかっ　た。理由としてGolangはワークスティール、プロセスやスレッドとは異なった　goroutine、コンテキストスイッチの軽量化などがある。
- Golangの並列化について

  goroutineはOSのスレッドでのコンテキストスイッチを少なくしているため

  コストが低い

  goroutineはスタックサイズを小さくできる、ガードページが不要そのため

  メモリを大きく消費することがない

  スレッドのモデルにM:Nモデルを使用している
- 関数の前に,構文として備え付けられている”go”をつけるだけでGoroutineが生成され　るため,並列化の実装が他言語に比べ容易　→ 並列化に向いている言語

実演

・最速処理のGoでマージソート

ご静聴
ありがとうございました