

取り組んだ内容

実装結果

実装結果

取り組んだ内容
その 2

取り組んだ内容
その 2

今後やっていく
こと

補助内容

水上 和秀

富山県立大学 電子・情報工学科

February 4, 2022

分散処理の環境構築

Dask（分散処理をするためのライブラリ）について調べて、Daskの環境構築をした。

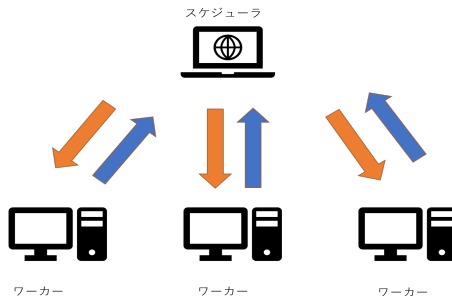


図 1: イメージ図

分散処理の実装

また、Dask を使って一つのプログラムを複数の pc で処理できるようにした。研究室の pc を使い、1つの pc、2つの pc、4つの pc の3つのパターンでプログラムを動かし、処理にかかった時間を調べた。



pc1
スレッド数:4
メモリー: 3.96Gib



pc2
スレッド数:4
メモリー: 3.98Gib



pc3
スレッド数:4
メモリー: 3.96Gib



pc4
スレッド数:4
メモリー: 3.95Gib

Table 1: 使用した pc

取り組んだ内容

実装結果

実装結果

取り組んだ内容
その 2

取り組んだ内容
その 2

今後やっていく
こと

■ 実行コンソール出力 - py NGAL_optimize

23	34500	6.00000E-01	2.18461E-02	1	0.00000E+00	f
24	36000	6.00000E-01	2.04875E-02	1	0.00000E+00	f
25	37500	6.00000E-01	1.92884E-02	1	0.00000E+00	f
26	39000	5.50000E-01	1.81709E-02	1	1.36000E+03	ideal
27	40500	5.50000E-01	1.71637E-02	1	0.00000E+00	f
28	42000	5.50000E-01	1.60337E-02	1	0.00000E+00	f
29	43500	3.50000E-01	1.50763E-02	1	2.16000E+02	ideal
30	45000	3.50000E-01	1.41709E-02	1	0.00000E+00	f
31	46500	3.50000E-01	1.32726E-02	1	0.00000E+00	f
32	48000	1.50000E-01	1.24176E-02	1	3.32000E+02	ideal
33	49500	1.50000E-01	1.16328E-02	1	0.00000E+00	f
34	51000	1.50000E-01	1.08482E-02	1	0.00000E+00	f
35	52500	6.800000000	1.00787E-02	1	2.16100E+03	ideal
36	54000	6.800000000	9.31788E-01	1	0.00000E+00	f
37	55500	0.00000E+00	8.62940E-01	1	1.09000E+03	ideal
38	57000	0.00000E+00	8.00793E-01	1	0.00000E+00	f
39	58500	0.00000E+00	7.51263E-01	1	0.00000E+00	f
40	60000	0.00000E+00	7.03377E-01	1	0.00000E+00	f
41	61500	0.00000E+00	6.55510E-01	2	1.000000000	ideal
42	63000	0.00000E+00	6.08805E-01	2	0.00000E+00	f
43	64500	0.00000E+00	5.63866E-01	3	0.440369972	ideal
44	66000	0.00000E+00	5.24744E-01	3	0.00000E+00	f
45	67500	0.00000E+00	4.82104E-01	4	0.240796020	ideal
46	69000	0.00000E+00	4.36575E-01	4	0.683750000	ideal
47	70500	0.00000E+00	4.28497E-01	4	0.00000E+00	f
48	72000	0.00000E+00	3.98387E-01	4	0.00000E+00	f
49	73500	0.00000E+00	3.70281E-01	5	0.657892868	ideal
50	75000	0.00000E+00	3.44179E-01	6	0.00000E+00	f
51	76500	0.00000E+00	3.20874E-01	6	0.00000E+00	f
52	78000	0.00000E+00	2.99003E-01	4	0.333333333	ideal
53	79500	0.00000E+00	2.75806E-01	4	0.00000E+00	f
54	81000	0.00000E+00	2.60179E-01	3	0.381235154	ideal
55	82500	0.00000E+00	2.39474E-01	3	0.00000E+00	f
56	84000	0.00000E+00	2.22009E-01	3	0.336485422	ideal
57	85500	0.00000E+00	2.03292E-01	3	0.00000E+00	f
58	87000	0.00000E+00	1.85921E-01	3	0.140615504	f
59	88500	0.00000E+00	1.72280E-01	3	0.00000E+00	f
60	90000	0.00000E+00	1.56895E-01	3	0.00000E+00	f

Desk: 2046, 271, 320, 04599 sec.
Function value: [1360, 29352]

図 2: pc1 のみで処理をした時の実行結果

実行コマンド: python3 pc1.py -p 1360 29352

23	34500	5.00000E+01	2.18463E+02	1	0.00000E+00	f
24	36000	5.00000E+01	2.04875E+02	1	0.00000E+00	f
25	37500	5.00000E+01	1.92884E+02	1	0.00000E+00	f
26	39000	5.00000E+01	1.81795E+02	1	1.36000E+03	ideal
27	40500	5.00000E+01	1.71037E+02	1	0.00000E+00	f
28	42000	5.00000E+01	1.60337E+02	1	0.00000E+00	f
29	43500	3.50000E+01	1.50763E+02	1	2.16000E+02	ideal
30	45000	3.50000E+01	1.41709E+02	1	0.00000E+00	f
31	46500	3.50000E+01	1.32726E+02	1	0.00000E+00	f
32	48000	3.50000E+01	1.24178E+02	1	0.00000E+00	ideal
33	49500	3.50000E+01	1.16328E+02	1	0.00000E+00	f
34	51000	3.50000E+01	1.08480E+02	1	0.00000E+00	f
35	52500	6.80000E+00	1.00787E+02	1	2.16000E+03	ideal
36	54000	6.80000E+00	9.31789E+01	1	0.00000E+00	f
37	55500	0.00000E+00	8.62840E+01	1	1.08000E+03	ideal
38	57000	0.00000E+00	8.00793E+01	1	0.00000E+00	f
39	58500	0.00000E+00	7.51263E+01	1	0.00000E+00	f
40	60000	0.00000E+00	7.03377E+01	1	0.00000E+00	f
41	61500	0.00000E+00	6.55510E+01	2	1.00000E+00	ideal
42	63000	0.00000E+00	6.08802E+01	2	0.00000E+00	f
43	64500	0.00000E+00	5.63966E+01	3	0.440369E+02	ideal
44	66000	0.00000E+00	5.24744E+01	3	0.00000E+00	f
45	67500	0.00000E+00	4.92184E+01	4	0.240796E+02	ideal
46	69000	0.00000E+00	4.58575E+01	4	0.093750E+00	ideal
47	70500	0.00000E+00	4.24847E+01	4	0.00000E+00	f
48	72000	0.00000E+00	3.90367E+01	4	0.00000E+00	f
49	73500	0.00000E+00	3.70291E+01	6	0.057692E+03	ideal
50	75000	0.00000E+00	3.44179E+01	6	0.00000E+00	f
51	76500	0.00000E+00	3.20674E+01	6	0.00000E+00	f
52	78000	0.00000E+00	2.99003E+01	4	0.333333E+03	ideal
53	79500	0.00000E+00	2.75809E+01	4	0.00000E+00	f
54	81000	0.00000E+00	2.60175E+01	3	0.381235E+04	ideal
55	82500	0.00000E+00	2.39474E+01	3	0.00000E+00	f
56	84000	0.00000E+00	2.23099E+01	3	0.338485E+02	ideal
57	85500	0.00000E+00	2.03292E+01	3	0.00000E+00	f
58	87000	0.00000E+00	1.85921E+01	3	0.146515E+04	f
59	88500	0.00000E+00	1.72280E+01	3	0.00000E+00	f
60	90000	0.00000E+00	1.58595E+01	3	0.00000E+00	f

Rank: 1436.865564532236 sec
Function value: [1360.29352]

図 3: pc1 と pc2 で処理をした時の実行結果

実行コマンド: python3 pc2.py

26	39000	5.00000E+01	1.81795E+02	1	1.36000E+03	ideal
27	40500	5.00000E+01	1.71037E+02	1	0.00000E+00	f
28	42000	5.00000E+01	1.60337E+02	1	0.00000E+00	f
29	43500	3.50000E+01	1.50763E+02	1	2.16000E+02	ideal
30	45000	3.50000E+01	1.41709E+02	1	0.00000E+00	f
31	46500	3.50000E+01	1.32726E+02	1	0.00000E+00	f
32	48000	3.50000E+01	1.24178E+02	1	3.32000E+02	ideal
33	49500	3.50000E+01	1.16328E+02	1	0.00000E+00	f
34	51000	3.50000E+01	1.08480E+02	1	0.00000E+00	f
35	52500	6.80000E+00	1.00787E+02	1	2.16000E+03	ideal
36	54000	6.80000E+00	9.31789E+01	1	0.00000E+00	f
37	55500	0.00000E+00	8.62840E+01	1	1.08000E+03	ideal
38	57000	0.00000E+00	8.00793E+01	1	0.00000E+00	f
39	58500	0.00000E+00	7.51263E+01	1	0.00000E+00	f
40	60000	0.00000E+00	7.03377E+01	1	0.00000E+00	f
41	61500	0.00000E+00	6.55510E+01	2	1.00000E+00	ideal
42	63000	0.00000E+00	6.08802E+01	2	0.00000E+00	f
43	64500	0.00000E+00	5.63966E+01	3	0.440369E+02	ideal
44	66000	0.00000E+00	5.24744E+01	3	0.00000E+00	f
45	67500	0.00000E+00	4.92184E+01	4	0.240796E+02	ideal
46	69000	0.00000E+00	4.58575E+01	4	0.093750E+00	ideal
47	70500	0.00000E+00	4.24847E+01	4	0.00000E+00	f
48	72000	0.00000E+00	3.90367E+01	4	0.00000E+00	f
49	73500	0.00000E+00	3.70291E+01	6	0.057692E+03	ideal
50	75000	0.00000E+00	3.44179E+01	6	0.00000E+00	f
51	76500	0.00000E+00	3.20674E+01	6	0.00000E+00	f
52	78000	0.00000E+00	2.99003E+01	4	0.333333E+03	ideal
53	79500	0.00000E+00	2.75809E+01	4	0.00000E+00	f
54	81000	0.00000E+00	2.60175E+01	3	0.381235E+04	ideal
55	82500	0.00000E+00	2.39474E+01	3	0.00000E+00	f
56	84000	0.00000E+00	2.23099E+01	3	0.338485E+02	ideal
57	85500	0.00000E+00	2.03292E+01	3	0.00000E+00	f
58	87000	0.00000E+00	1.85921E+01	3	0.146515E+04	f
59	88500	0.00000E+00	1.72280E+01	3	0.00000E+00	f
60	90000	0.00000E+00	1.58595E+01	3	0.00000E+00	f

Rank: 1089.1242501735687 sec
Function value: [1360.29352]

図 4: 4 つの pc で処理をした時の実行結果

実装結果

- ・ 1つのpcでプログラムを動かしたときのプログラムの処理時間は約34分, 2つのpcでプログラムを動かしたときの処理時間は約24分, 4つのpcでプログラムを動かしたときのプログラムの処理時間は約18分であった。

- ・ また、コンピュータの数が多いほど実行時間が短いことが確認できた。

- ・ このことより並列処理のプログラムは有用であることが確認できた。

並列処理の実装

`concurrent.futures`（並列処理をするためのライブラリ）について調べて、並列処理のプログラムを実装した。

また、ノート pc を使い、スレッド数が 1 から 10 のパターンでプログラムを動かし、処理にかかった時間を調べた。

取り組んだ内容

実装結果

実装結果

取り組んだ内容
その 2

取り組んだ内容
その 2

今後やっていく
こと

取り組んだ内容

実装結果

実装結果

取り組んだ内容
その 2

取り組んだ内容
その 2

今後やっていく
こと

```
表示(V) 移動(G) 実行(R) ターミナル(T) ヘルプ(H) suc3.py

suc2.py 1 suc3.py 1 X suc3.py 2

C:\Users\kazuh\Desktop> suc3.py ...
1 from multiprocessing import Pool
2 from multiprocessing import Process
3 import random as rnd
4 import time
5 from concurrent.futures import ThreadPoolExecutor
6
7 def func():
8     time.sleep(1)
9
10 for x in range(10):
11     start = time.time() #処理開始時間
12     with ThreadPoolExecutor(max_workers=10) as e:
13         #並列処理
14         for i in range(10):
15             e.submit(func)
16         #完了待ち
17     end = time.time() #処理終了時間
18     delta = end - start #処理時間
19     print('スレッド数', x+1, 'のときの処理時間:', '{:.3f}'.format(round(delta, 3)))
```

図 5: ソースコード

コマンドプロンプト

```
C:\Users\kazuh\Desktop>py suc3.py
スレッド数 1 のときの処理時間:10.098s
スレッド数 2 のときの処理時間:5.048s
スレッド数 3 のときの処理時間:4.04s
スレッド数 4 のときの処理時間:3.021s
スレッド数 5 のときの処理時間:2.011s
スレッド数 6 のときの処理時間:2.022s
スレッド数 7 のときの処理時間:2.023s
スレッド数 8 のときの処理時間:2.02s
スレッド数 9 のときの処理時間:2.031s
スレッド数 10 のときの処理時間:1.016s

C:\Users\kazuh\Desktop>
```

図 6: 実行結果

ウェブスクレイピングの実装

python で HTML からデータを引き出せる BeautifulSoup というライブラリを使ってウェブサイトからデータを抽出した
また、抽出したデータを CSV に出力した。

```
C: > Users > kazuh > Desktop > suc.py > ...
1 import csv
2 import requests
3 from bs4 import BeautifulSoup
4
5 HEADER = ['name', 'age', 'occupation', 'url']
6
7 url = 'https://talent-dictionary.com/s/jobs/3/20'
8 r = requests.get(url)
9
10 soup = BeautifulSoup(r.text, 'html.parser')
11 actors = soup.find('ul', class_='list').find_all('li')
12
13 with open('actors.csv', 'w', encoding='Shift-Jis') as f:
14     writer = csv.writer(f)
15     writer.writerow(HEADER)
16     for actor in actors:
17         prof = actor.find('div', class_='right')
18
19         name = prof.find('a', class_='title').text
20         url = prof.find('a', class_='title').get('href')
21         occupation = prof.find('a', class_='job').text
22         age = prof.find('span', class_='age').text
23
24         row = [name, age, occupation, url]
25         writer.writerow(row)
```

図 7: スクレイピングのソースコード

取り組んだ内容その2

10/11

取り組んだ内容

実装結果

実装結果

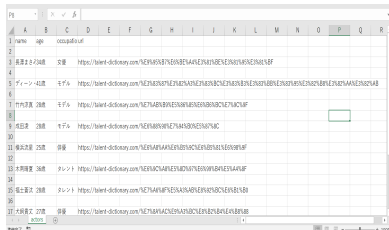
取り組んだ内容
その 2

取り組んだ内容
その 2

今後やっていく
こと



図 8: スクレイピングしたサイト



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	name	age	category															
2																		
3	長澤まさみ	24歳	女優															
4																		
5	ディーン・フジオカ	41歳	モデル															
6																		
7	竹内涼真	24歳	モデル															
8																		
9	成田凌	23歳	モデル															
10																		
11	横浜流星	25歳	俳優															
12																		
13	本間清昭	24歳	モデル															
14																		
15	藤井夏生	23歳	モデル															
16																		
17	大前元文	27歳	俳優															
18																		

図 9: 実行結果

今後やっていくこと

- ・ より高速にプログラムを動かせるようにする
- ・ 先輩の手伝いを引き続き行う
- ・ 先輩のプログラムをサーバー上で動かせるようにする。

取り組んだ内容

実装結果

実装結果

取り組んだ内容
その 2

取り組んだ内容
その 2

今後やっていく
こと