Problme 2

We are going to experiment with PyTorch's DataParallel Module, which is PyTorch's Synchronous SGD implementation across a number of GPUs on the same server. In particular, we will train ResNet-18 implementation from https://github.com/kuangliu/pytorch-cifar with num workers=2, running up to 4 GPUs with DataParallel (DP) Module. Use SGD optimizers with 0.1 as the learning rate, momentum 0.9, weight decay 5e-4. For this question, you need to do experiment with multiple GPUs on the same server. You may need to execute this on NYU Greene Cluster.

Create a PyTorch program with a DataLoader that loads the images and the related labels from torchvision CIFAR10 dataset. Import CIFAR10 dataset for the torchvision package, with the following sequence of transformations:

• Random cropping, with size 32x32 and padding 4

• Random horizontal flipping with a probability 0.5

• Normalize each image's RGB channel with mean(0.4914, 0.4822, 0.4465) and variance (0.2023, 0.1994, 0.2010)

The DataLoader for the training set uses a minibatch size of 128 and 3 IO processes (i.e., num workers=2). The DataLoader for the testing set uses minibatch size of 100 and 3 IO processes (i.e., num workers =2). Create a main function that creates the DataLoaders for the training set and the neural network.

1

Measure how long does it take to compete 1 epoch of training using different batch size on a single GPU. Start from batch size 32, increase by 4-fold for each measurement (i.e., 32, 128, 512 ...) until single GPU memory cannot hold the batch size. For each run, run 2 epochs, the first epoch is used to warmup CPU/GPU cache; and you should report the training time (excluding data I/O; but including data movement from CPU to GPU, gradients calculation and weights update) based on the 2nd epoch training. (5)

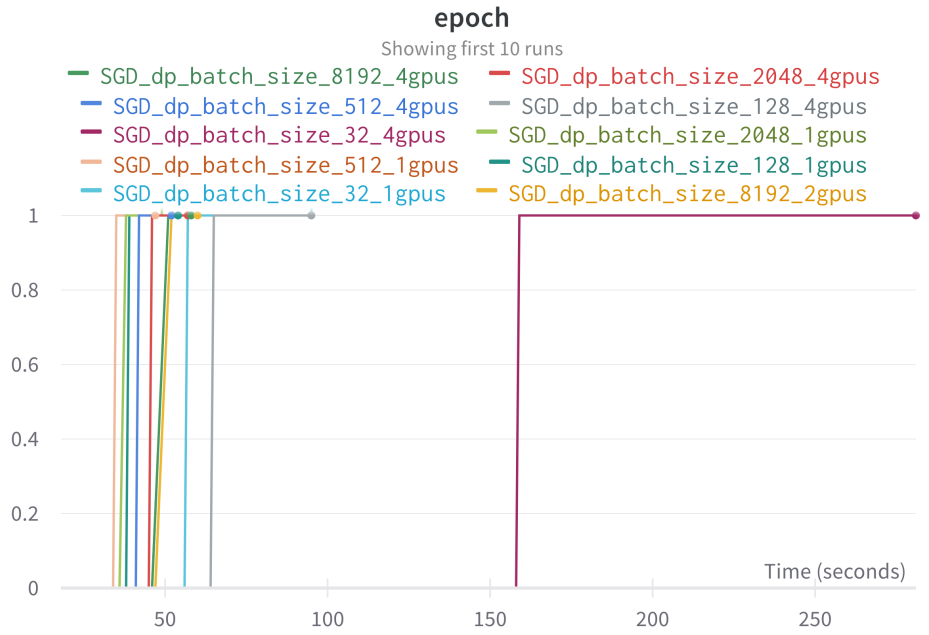
In [ ]:

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision.transforms as transforms
import torchvision.datasets as datasets

train_transform = transforms.Compose([transforms.RandomCrop(32, padding=4),#r
                                      transforms.RandomHorizontalFlip(0.5),#r
                                      transforms.Normalize((0.4914, 0.4822, 0
                                      transforms.ToTensor(),])#convert to ten
train_dataset = datasets.CIFAR10("./cached_datasets/CIFAR10", train=True, dow
train_dataloader = torch.utils.data.DataLoader(train_dataset, batch_size=128,
```

```
Files already downloaded and verified
```

Please refer to the code in problem2 folder



We use the straight line to calculate the time for each configuration.

In [ ]:
```
import pandas as pd
data = pd.read_csv("./problem2/2_1.csv")
```

In [ ]:
```
data
```

Out[ ]:

	Relative Time (Wall)	SGD_dp_batch_size_8192_4gpus - _step	SGD_dp_batch_size_8192_4gpus - _step__MIN	SGD_dp_batch_
0	NaN	7.5	0	

1 rows × 121 columns

In [ ]:
```
import wandb
import pandas as pd
api = wandb.Api()

# Project is specified by <entity/project-name>
runs = api.runs("xiang-pan/NYU_DL_Sys-HW3_problem2")
summary_list = []
config_list = []
name_list = []
res = {}
for run in runs:
    data = run.history()
    log_name = run.name
    res[log_name] = data
```

In [ ]:
```
res.keys()
```

Out[ ]:
```
dict_keys(['SGD_dp_batch_size_512_2gpus', 'SGD_dp_batch_size_2048_2gpus', 'SGD
_dp_batch_size_8192_4gpus', 'SGD_dp_batch_size_2048_4gpus', 'SGD_dp_batch_size
_512_4gpus', 'SGD_dp_batch_size_128_4gpus', 'SGD_dp_batch_size_32_4gpus', 'SGD
_dp_batch_size_8192_1gpus', 'SGD_dp_batch_size_2048_1gpus', 'SGD_dp_batch_size
```

```
_512_1gpus', 'SGD_dp_batch_size_128_1gpus', 'SGD_dp_batch_size_32_1gpus', 'SGD
_dp_batch_size_8192_2gpus', 'SGD_dp_batch_size_128_2gpus', 'SGD_dp_batch_size_
32_2gpus'])
```

In [ ]:
```
res
```

Out[ ]:
```
{'SGD_dp_batch_size_512_2gpus':      trainer/global_step  _step  _runtime  tra
in_loss_step  epoch  _timestamp  \
 0                       0      0        18         2.420080      0  1649128380
 1                       1      1        19         2.769084      0  1649128381
 2                       2      2        19         3.562219      0  1649128381
 3                       3      3        19         3.776740      0  1649128381
 4                       4      4        19         3.318298      0  1649128381
 ..                    ...    ...       ...              ...    ...         ...
 193                   192    193        37         1.172103      1  1649128399
 194                   193    194        37         1.230016      1  1649128399
 195                   194    195        37         1.342752      1  1649128399
 196                   195    196        37         1.132757      1  1649128399
 197                   195    197        38              NaN      2  1649128400

      train_loss_epoch
 0                 NaN
 1                 NaN
 2                 NaN
 3                 NaN
 4                 NaN
 ..                ...
 193               NaN
 194               NaN
 195               NaN
 196               NaN
 197          1.342205

 [198 rows x 7 columns],
 'SGD_dp_batch_size_2048_2gpus':     trainer/global_step  _step  _runtime  tra
in_loss_step  epoch  _timestamp  \
 0                       0      0        20         2.407045      0  1649128132
 1                       1      1        20         2.558295      0  1649128132
 2                       2      2        20         2.766725      0  1649128132
 3                       3      3        20         3.660211      0  1649128132
 4                       4      4        21         4.084877      0  1649128133
 5                       5      5        21         3.603554      0  1649128133
 6                       6      6        21         3.600378      0  1649128133
 7                       7      7        21         3.024597      0  1649128133
 8                       8      8        22         2.548291      0  1649128134
 9                       9      9        22         3.080903      0  1649128134
 10                     10     10        22         2.824992      0  1649128134
 11                     11     11        23         2.455785      0  1649128135
 12                     12     12        23         2.284315      0  1649128135
 13                     13     13        23         2.387378      0  1649128135
 14                     14     14        24         2.258087      0  1649128136
 15                     15     15        24         2.535852      0  1649128136
 16                     16     16        24         2.237878      0  1649128136
 17                     17     17        25         2.035352      0  1649128137
 18                     18     18        25         2.010104      0  1649128137
 19                     19     19        25         2.214593      0  1649128137
 20                     20     20        26         2.089128      0  1649128138
 21                     21     21        26         1.944005      0  1649128138
 22                     22     22        26         1.937898      0  1649128138
 23                     23     23        27         2.004253      0  1649128139
 24                     24     24        28         2.022934      0  1649128140
 25                     24     25        28              NaN      1  1649128140
 26                     25     26        30         1.917986      1  1649128142
```

27	26	27	30	2.046042	1	1649128142
28	27	28	30	1.983665	1	1649128142
29	28	29	31	1.961981	1	1649128143
30	29	30	31	1.930356	1	1649128143
31	30	31	31	1.869945	1	1649128143
32	31	32	32	1.867186	1	1649128144
33	32	33	32	1.846186	1	1649128144
34	33	34	32	1.829378	1	1649128144
35	34	35	33	1.847615	1	1649128145
36	35	36	33	1.775432	1	1649128145
37	36	37	33	1.822025	1	1649128145
38	37	38	34	1.789850	1	1649128146
39	38	39	34	1.758589	1	1649128146
40	39	40	34	1.721828	1	1649128146
41	40	41	35	1.739697	1	1649128147
42	41	42	35	1.705283	1	1649128147
43	42	43	35	1.711573	1	1649128147
44	43	44	36	1.695738	1	1649128148
45	44	45	36	1.679440	1	1649128148
46	45	46	37	1.677661	1	1649128149
47	46	47	37	1.728877	1	1649128149
48	47	48	37	1.725134	1	1649128149
49	48	49	37	1.675917	1	1649128149
50	49	50	38	1.621064	1	1649128150
51	49	51	38	NaN	2	1649128150

	train_loss_epoch
0	NaN
1	NaN
2	NaN
3	NaN
4	NaN
5	NaN
6	NaN
7	NaN
8	NaN
9	NaN
10	NaN
11	NaN
12	NaN
13	NaN
14	NaN
15	NaN
16	NaN
17	NaN
18	NaN
19	NaN
20	NaN
21	NaN
22	NaN
23	NaN
24	NaN
25	2.53072
26	NaN
27	NaN
28	NaN
29	NaN
30	NaN
31	NaN
32	NaN
33	NaN
34	NaN
35	NaN
36	NaN

```
 37               NaN
 38               NaN
 39               NaN
 40               NaN
 41               NaN
 42               NaN
 43               NaN
 44               NaN
 45               NaN
 46               NaN
 47               NaN
 48               NaN
 49               NaN
 50               NaN
 51           1.76056  ,
 'SGD_dp_batch_size_8192_4gpus':     trainer/global_step  _step  _runtime  tra
in_loss_step  epoch  _timestamp  \
 0                     0      0        40         2.390903      0  1648372568
 1                     1      1        41         2.618182      0  1648372569
 2                     2      2        41         2.409632      0  1648372569
 3                     3      3        42         2.382266      0  1648372570
 4                     4      4        42         2.854592      0  1648372570
 5                     5      5        44         4.384065      0  1648372572
 6                     6      6        46         4.012147      0  1648372574
 7                     6      7        46              NaN      1  1648372574
 8                     7      8        51         3.435173      1  1648372579
 9                     8      9        52         3.664042      1  1648372580
 10                    9     10        54         3.144501      1  1648372582
 11                   10     11        54         3.195889      1  1648372582
 12                   11     12        57         3.081101      1  1648372585
 13                   12     13        58         2.827945      1  1648372586
 14                   13     14        58         2.607582      1  1648372586
 15                   13     15        58              NaN      2  1648372586

     train_loss_epoch
 0               NaN
 1               NaN
 2               NaN
 3               NaN
 4               NaN
 5               NaN
 6               NaN
 7          2.660240
 8               NaN
 9               NaN
 10              NaN
 11              NaN
 12              NaN
 13              NaN
 14              NaN
 15         3.263868  ,
 'SGD_dp_batch_size_2048_4gpus':     trainer/global_step  _step  _runtime  tra
in_loss_step  epoch  _timestamp  \
 0                     0      0        35         2.382205      0  1648372482
 1                     1      1        36         2.400140      0  1648372483
 2                     2      2        36         2.707777      0  1648372483
 3                     3      3        36         3.673821      0  1648372483
 4                     4      4        36         4.163005      0  1648372483
 5                     5      5        36         5.222578      0  1648372483
 6                     6      6        37         4.147680      0  1648372484
 7                     7      7        37         3.306381      0  1648372484
 8                     8      8        38         4.035182      0  1648372485
 9                     9      9        38         2.886489      0  1648372485
 10                   10     10        39         2.878854      0  1648372486
```

11	11	11	39	3.682711	0	1648372486
12	12	12	39	3.405432	0	1648372486
13	13	13	40	3.407504	0	1648372487
14	14	14	40	3.077529	0	1648372487
15	15	15	40	3.169637	0	1648372487
16	16	16	41	2.683454	0	1648372488
17	17	17	41	2.308244	0	1648372488
18	18	18	42	2.707833	0	1648372489
19	19	19	42	2.655234	0	1648372489
20	20	20	43	2.560152	0	1648372490
21	21	21	43	2.510918	0	1648372490
22	22	22	44	2.317803	0	1648372491
23	23	23	44	2.223610	0	1648372491
24	24	24	45	2.186810	0	1648372492
25	24	25	46	NaN	1	1648372493
26	25	26	48	2.244568	1	1648372495
27	26	27	48	2.193285	1	1648372495
28	27	28	48	2.009049	1	1648372495
29	28	29	49	2.121168	1	1648372496
30	29	30	49	2.085999	1	1648372496
31	30	31	49	2.170903	1	1648372496
32	31	32	50	1.948614	1	1648372497
33	32	33	50	2.158827	1	1648372497
34	33	34	51	2.151595	1	1648372498
35	34	35	51	2.051746	1	1648372498
36	35	36	52	1.952415	1	1648372499
37	36	37	52	2.027159	1	1648372499
38	37	38	53	1.966849	1	1648372500
39	38	39	53	2.004494	1	1648372500
40	39	40	53	2.024491	1	1648372500
41	40	41	53	1.961847	1	1648372500
42	41	42	54	1.976081	1	1648372501
43	42	43	54	1.901197	1	1648372501
44	43	44	55	1.969630	1	1648372502
45	44	45	55	1.940805	1	1648372502
46	45	46	56	1.964961	1	1648372503
47	46	47	56	1.898335	1	1648372503
48	47	48	57	1.903053	1	1648372504
49	48	49	57	1.933724	1	1648372504
50	49	50	57	1.761307	1	1648372504
51	49	51	57	NaN	2	1648372504

	train_loss_epoch
0	NaN
1	NaN
2	NaN
3	NaN
4	NaN
5	NaN
6	NaN
7	NaN
8	NaN
9	NaN
10	NaN
11	NaN
12	NaN
13	NaN
14	NaN
15	NaN
16	NaN
17	NaN
18	NaN
19	NaN
20	NaN

```
21               NaN
22               NaN
23               NaN
24               NaN
25          3.039449
26               NaN
27               NaN
28               NaN
29               NaN
30               NaN
31               NaN
32               NaN
33               NaN
34               NaN
35               NaN
36               NaN
37               NaN
38               NaN
39               NaN
40               NaN
41               NaN
42               NaN
43               NaN
44               NaN
45               NaN
46               NaN
47               NaN
48               NaN
49               NaN
50               NaN
51          1.963216  ,
 'SGD_dp_batch_size_512_4gpus':      trainer/global_step  _step  _runtime  tra
in_loss_step  epoch  _timestamp  \
 0                       0      0        30         2.447535      0  1648372405
 1                       1      1        30         2.636137      0  1648372405
 2                       2      2        31         3.141139      0  1648372406
 3                       3      3        31         4.203696      0  1648372406
 4                       4      4        31         4.988063      0  1648372406
 ..                    ...    ...       ...              ...    ...         ...
 193                   192    193        51         1.580170      1  1648372426
 194                   193    194        51         1.535819      1  1648372426
 195                   194    195        52         1.507741      1  1648372427
 196                   195    196        52         1.734261      1  1648372427
 197                   195    197        52              NaN      2  1648372427

      train_loss_epoch
 0                NaN
 1                NaN
 2                NaN
 3                NaN
 4                NaN
 ..               ...
 193              NaN
 194              NaN
 195              NaN
 196              NaN
 197          1.64051

 [198 rows x 7 columns],
 'SGD_dp_batch_size_128_4gpus':      trainer/global_step  _step  _runtime  tra
in_loss_step  epoch  _timestamp  \
 0                       0      0        34         2.454476      0  1648372290
 1                       1      1        34         3.187948      0  1648372290
 2                       2      2        34         3.783150      0  1648372290
```

```
 3                     3      3        34         4.302678      0  1648372290
 4                     4      4        34         8.475398      0  1648372290
 ..                  ...    ...       ...              ...    ...         ...
 517                 774    775        94         1.817231      1  1648372350
 518                 775    776        94         1.216002      1  1648372350
 519                 779    780        95         1.199796      1  1648372351
 520                 780    781        95         1.388824      1  1648372351
 521                 781    783        95              NaN      2  1648372351

      train_loss_epoch
 0                 NaN
 1                 NaN
 2                 NaN
 3                 NaN
 4                 NaN
 ..                ...
 517               NaN
 518               NaN
 519               NaN
 520               NaN
 521          1.463055

 [522 rows x 7 columns],
 'SGD_dp_batch_size_32_4gpus':      trainer/global_step  _step  _runtime  trai
n_loss_step  epoch  _timestamp
 0                     12     12        38         4.564947      0  1648371922
 1                     44     44        40         2.453320      0  1648371924
 2                     46     46        40         2.263949      0  1648371924
 3                     54     54        41         2.240820      0  1648371925
 4                     66     66        42         2.397463      0  1648371926
 ..                   ...    ...       ...              ...    ...         ...
 485                3089   3090       278         1.161531      1  1648372162
 486                3093   3094       278         0.834570      1  1648372162
 487                3098   3099       279         0.499158      1  1648372163
 488                3114   3115       280         2.188230      1  1648372164
 489                3118   3119       280         0.931003      1  1648372164

 [490 rows x 6 columns],
 'SGD_dp_batch_size_8192_1gpus': Empty DataFrame
 Columns: []
 Index: [],
 'SGD_dp_batch_size_2048_1gpus':     trainer/global_step  _step  _runtime  tra
in_loss_step  epoch  _timestamp  \
 0                     0      0        23         2.378599      0  1648371712
 1                     1      1        24         2.419253      0  1648371713
 2                     2      2        24         2.778703      0  1648371713
 3                     3      3        25         3.354625      0  1648371714
 4                     4      4        25         4.705425      0  1648371714
 5                     5      5        26         5.723037      0  1648371715
 6                     6      6        26         6.090533      0  1648371715
 7                     7      7        27         6.604627      0  1648371716
 8                     8      8        27         8.788239      0  1648371716
 9                     9      9        28         8.495030      0  1648371717
 10                   10     10        28         7.792514      0  1648371717
 11                   11     11        29         6.351874      0  1648371718
 12                   12     12        29         4.802438      0  1648371718
 13                   13     13        30         3.963925      0  1648371719
 14                   14     14        30         6.134727      0  1648371719
 15                   15     15        31         5.324747      0  1648371720
 16                   16     16        31         4.129182      0  1648371720
 17                   17     17        32         3.113030      0  1648371721
 18                   18     18        32         3.248461      0  1648371721
 19                   19     19        32         2.751126      0  1648371721
 20                   20     20        33         2.790308      0  1648371722
```

21	21	21	33	2.772371	0	1648371722
22	22	22	34	2.586316	0	1648371723
23	23	23	34	2.596285	0	1648371723
24	24	24	36	2.469417	0	1648371725
25	24	25	36	NaN	1	1648371725
26	25	26	38	2.522918	1	1648371727
27	26	27	39	2.351709	1	1648371728
28	27	28	39	2.585978	1	1648371728
29	28	29	40	2.398835	1	1648371729
30	29	30	40	2.346667	1	1648371729
31	30	31	41	2.326970	1	1648371730
32	31	32	41	2.376224	1	1648371730
33	32	33	41	2.290313	1	1648371730
34	33	34	42	2.323020	1	1648371731
35	34	35	42	2.267548	1	1648371731
36	35	36	43	2.282800	1	1648371732
37	36	37	43	2.251382	1	1648371732
38	37	38	44	2.217711	1	1648371733
39	38	39	44	2.229835	1	1648371733
40	39	40	45	2.261669	1	1648371734
41	40	41	45	2.194469	1	1648371734
42	41	42	46	2.201263	1	1648371735
43	42	43	46	2.248072	1	1648371735
44	43	44	47	2.173002	1	1648371736
45	44	45	47	2.177445	1	1648371736
46	45	46	48	2.208236	1	1648371737
47	46	47	48	2.183602	1	1648371737
48	47	48	49	2.229931	1	1648371738
49	48	49	49	2.221301	1	1648371738
50	49	50	49	2.225251	1	1648371738
51	49	51	50	NaN	2	1648371739

	train_loss_epoch
0	NaN
1	NaN
2	NaN
3	NaN
4	NaN
5	NaN
6	NaN
7	NaN
8	NaN
9	NaN
10	NaN
11	NaN
12	NaN
13	NaN
14	NaN
15	NaN
16	NaN
17	NaN
18	NaN
19	NaN
20	NaN
21	NaN
22	NaN
23	NaN
24	NaN
25	4.535004
26	NaN
27	NaN
28	NaN
29	NaN
30	NaN

```
 31               NaN
 32               NaN
 33               NaN
 34               NaN
 35               NaN
 36               NaN
 37               NaN
 38               NaN
 39               NaN
 40               NaN
 41               NaN
 42               NaN
 43               NaN
 44               NaN
 45               NaN
 46               NaN
 47               NaN
 48               NaN
 49               NaN
 50               NaN
 51          2.285252  ,
 'SGD_dp_batch_size_512_1gpus':      trainer/global_step  _step  _runtime  tra
in_loss_step  epoch  _timestamp  \
 0                      0      0        21         2.410742      0  1648371678
 1                      1      1        21         2.752490      0  1648371678
 2                      2      2        21         2.737628      0  1648371678
 3                      3      3        21         3.834259      0  1648371678
 4                      4      4        21         5.095662      0  1648371678
 ..                   ...    ...       ...              ...    ...         ...
 193                  192    193        46         1.473887      1  1648371703
 194                  193    194        47         1.604234      1  1648371704
 195                  194    195        47         1.577804      1  1648371704
 196                  195    196        47         1.381406      1  1648371704
 197                  195    197        47              NaN      2  1648371704

      train_loss_epoch
 0                NaN
 1                NaN
 2                NaN
 3                NaN
 4                NaN
 ..               ...
 193              NaN
 194              NaN
 195              NaN
 196              NaN
 197         1.619874

 [198 rows x 7 columns],
 'SGD_dp_batch_size_128_1gpus':      trainer/global_step  _step  _runtime  tra
in_loss_step  epoch  _timestamp  \
 0                      1      1        21         3.322361      0  1648371640
 1                      4      4        21         5.161662      0  1648371640
 2                      5      5        21         5.269036      0  1648371640
 3                      9      9        22         3.224431      0  1648371641
 4                     10     10        22         2.732107      0  1648371641
 ..                   ...    ...       ...              ...    ...         ...
 502                  776    777        54         1.222841      1  1648371673
 503                  777    778        54         1.118677      1  1648371673
 504                  778    779        54         1.203329      1  1648371673
 505                  780    781        54         1.263283      1  1648371673
 506                  781    783        54              NaN      2  1648371673

      train_loss_epoch
```

```
 0                 NaN
 1                 NaN
 2                 NaN
 3                 NaN
 4                 NaN
 ..                ...
 502               NaN
 503               NaN
 504               NaN
 505               NaN
 506          1.468338

 [507 rows x 7 columns],
 'SGD_dp_batch_size_32_1gpus':      trainer/global_step  _step  _runtime  trai
n_loss_step  epoch  _timestamp
 0                      14     14        18         5.188556      0  1648371592
 1                      17     17        18         3.954767      0  1648371592
 2                      25     25        18         3.013579      0  1648371592
 3                      35     35        18         3.005083      0  1648371592
 4                      38     38        18         2.163107      0  1648371592
 ..                    ...    ...       ...              ...    ...         ...
 491                 3086   3087        94         1.174098      1  1648371668
 492                 3090   3091        95         1.645498      1  1648371669
 493                 3097   3098        95         1.174476      1  1648371669
 494                 3098   3099        95         1.003709      1  1648371669
 495                 3105   3106        95         1.125607      1  1648371669

 [496 rows x 6 columns],
 'SGD_dp_batch_size_8192_2gpus':     trainer/global_step  _step  _runtime  tra
in_loss_step  epoch  _timestamp  \
 0                      0      0        39         2.343246      0  1648371067
 1                      1      1        40         2.417241      0  1648371068
 2                      2      2        41         2.485400      0  1648371069
 3                      3      3        42         2.576402      0  1648371070
 4                      4      4        43         3.028359      0  1648371071
 5                      5      5        44         3.302346      0  1648371072
 6                      6      6        47         4.070184      0  1648371075
 7                      6      7        47              NaN      1  1648371075
 8                      7      8        52         4.775397      1  1648371080
 9                      8      9        53         4.856526      1  1648371081
 10                     9     10        55         4.949286      1  1648371083
 11                    10     11        56         4.184242      1  1648371084
 12                    11     12        59         3.392799      1  1648371087
 13                    12     13        60         2.816032      1  1648371088
 14                    13     14        60         2.903746      1  1648371088
 15                    13     15        60              NaN      2  1648371088

     train_loss_epoch
 0                NaN
 1                NaN
 2                NaN
 3                NaN
 4                NaN
 5                NaN
 6                NaN
 7           2.721693
 8                NaN
 9                NaN
 10               NaN
 11               NaN
 12               NaN
 13               NaN
 14               NaN
 15          4.130736  ,
```

```
 'SGD_dp_batch_size_128_2gpus':     trainer/global_step  _step  _runtime  trai
n_loss_step  epoch  _timestamp  \
 0                    49      0        29         2.181488      0  1648370297
 1                    99      1        32         2.085283      0  1648370300
 2                   149      2        34         1.959760      0  1648370302
 3                   199      3        37         1.864811      0  1648370305
 4                   249      4        39         1.824337      0  1648370307
 5                   299      5        42         1.748104      0  1648370310
 6                   349      6        44         1.689489      0  1648370312
 7                   390      7        47              NaN      1  1648370315
 8                   399      8        48         1.787462      1  1648370316
 9                   449      9        50         1.464020      1  1648370318
 10                  499     10        53         1.419646      1  1648370321
 11                  549     11        55         1.453597      1  1648370323
 12                  599     12        58         1.254473      1  1648370326
 13                  649     13        60         1.219815      1  1648370328
 14                  699     14        63         1.319156      1  1648370331
 15                  749     15        65         1.492566      1  1648370333
 16                  781     16        67              NaN      2  1648370335

    train_loss_epoch
 0               NaN
 1               NaN
 2               NaN
 3               NaN
 4               NaN
 5               NaN
 6               NaN
 7          1.987961
 8               NaN
 9               NaN
 10              NaN
 11              NaN
 12              NaN
 13              NaN
 14              NaN
 15              NaN
 16         1.462412  ,
 'SGD_dp_batch_size_32_2gpus':     trainer/global_step  _step  _runtime  train
_loss_step  epoch  _timestamp  \
 0                    49      0        24         3.000532      0  1648370289
 1                    99      1        27         3.825331      0  1648370292
 2                   149      2        29         2.502269      0  1648370294
 3                   199      3        31         2.015135      0  1648370296
 4                   249      4        33         1.945419      0  1648370298
 ..                  ...    ...       ...              ...    ...         ...
 59                 2949     59       153         1.278652      1  1648370418
 60                 2999     60       155         2.211522      1  1648370420
 61                 3049     61       157         1.409628      1  1648370422
 62                 3099     62       159         1.205655      1  1648370424
 63                 3125     63       161              NaN      2  1648370426

    train_loss_epoch
 0               NaN
 1               NaN
 2               NaN
 3               NaN
 4               NaN
 ..              ...
 59              NaN
 60              NaN
 61              NaN
 62              NaN
 63          1.50301
```

```
[64 rows x 7 columns]}
```

In [ ]:
```python
def get_info(log_name):
    gpu_nums = log_name.split('_')[-1]
    batch_size = log_name.split('_')[-2]
    batch_size = int(batch_size)
    gpu_nums = gpu_nums.replace('gpus','')
    gpu_nums = int(gpu_nums)
    return gpu_nums, batch_size

def get_time_info(log_name):
    data = res[log_name]
    epoch_1_start_time = data[data["epoch"] == 1].iloc[0]["_timestamp"]
    epoch_1_end_time = data[data["epoch"] == 1].iloc[-1]["_timestamp"]
    t = epoch_1_end_time - epoch_1_start_time
    return t
get_time_info("SGD_dp_batch_size_8192_4gpus")
```

Out[ ]:
```
12.0
```

batch_size = 8192 can not work in my gpu.

In [ ]:
```python
df = pd.DataFrame(columns=['gpu_nums', 'batch_size', 'time'])
for key in res.keys():
    # print(key)
    if "8192" in key:
        continue
    gpu_nums, batch_size = get_info(key)
    t = get_time_info(key)
    df.loc[len(df)] = [gpu_nums, batch_size, t]
df.sort_values(by=['gpu_nums','batch_size'], ascending=True, inplace=True)
df["gpu_nums"] = df["gpu_nums"].astype(int)
df["batch_size"] = df["batch_size"].astype(int)
df.reset_index(inplace=True, drop=True)
```

In [ ]:
```python
df.to_markdown("problem2/2_1_table_src.md")
```

In [ ]:
```python
df
```

Out[ ]:

	gpu_nums	batch_size	time
0	1	32	38.0
1	1	128	16.0
2	1	512	13.0
3	1	2048	13.0
4	2	32	67.0
5	2	128	18.0
6	2	512	9.0
7	2	2048	10.0
8	4	32	121.0
9	4	128	30.0

	gpu_nums	batch_size	time
10	4	512	11.0
11	4	2048	11.0

In [ ]:
```
t = df[df["batch_size"] == 32]
t["speedup"] = t["time"].iloc[0]/t["time"]
t
```

```
/tmp/ipykernel_768624/1156332872.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  t["speedup"] = t["time"].iloc[0]/t["time"] * t["gpu_nums"]
```

Out[ ]:

	gpu_nums	batch_size	time	speedup
0	1	32	38.0	1.000000
4	2	32	67.0	1.134328
8	4	32	121.0	1.256198

In [ ]:
```
t = df[df["batch_size"] == 128]
t["speedup"] = t["time"].iloc[0]/t["time"]
t
```

```
/tmp/ipykernel_768624/3464596280.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  t["speedup"] = t["time"].iloc[0]/t["time"]
```

Out[ ]:

	gpu_nums	batch_size	time	speedup
1	1	128	16.0	1.000000
5	2	128	18.0	0.888889
9	4	128	30.0	0.533333

In [ ]:
```
t = df[df["batch_size"] == 512]
t["speedup"] = t["time"].iloc[0]/t["time"]
t
```

```
/tmp/ipykernel_768624/3178531423.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  t["speedup"] = t["time"].iloc[0]/t["time"]
```

Out[ ]:

	gpu_nums	batch_size	time	speedup
2	1	512	13.0	1.000000

	gpu_nums	batch_size	time	speedup
6	2	512	9.0	1.444444
10	4	512	11.0	1.181818

In [ ]:
```
t = df[df["batch_size"] == 2048]
t["speedup"] = t["time"].iloc[0]/t["time"]
t
```

```
/tmp/ipykernel_768624/2197254454.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  t["speedup"] = t["time"].iloc[0]/t["time"]
```

Out[ ]:

	gpu_nums	batch_size	time	speedup
3	1	2048	13.0	1.000000
7	2	2048	10.0	1.300000
11	4	2048	11.0	1.181818

In [ ]:
```
l = []
for b in [32, 128, 512, 2048]:
    t = df[df["batch_size"] == b]
    t["speedup"] = t["time"].iloc[0]/t["time"]
    t.to_markdown(f"problem2/2_1_table_src_{b}.md")
    l.append(t)
l = pd.concat(l)
```

```
/tmp/ipykernel_768624/2183533066.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  t["speedup"] = t["time"].iloc[0]/t["time"]
```

In [ ]:
```
l
```

Out[ ]:

	gpu_nums	batch_size	time	speedup
0	1	32	38.0	1.000000
4	2	32	67.0	0.567164
8	4	32	121.0	0.314050
1	1	128	16.0	1.000000
5	2	128	18.0	0.888889
9	4	128	30.0	0.533333
2	1	512	13.0	1.000000
6	2	512	9.0	1.444444
10	4	512	11.0	1.181818

	gpu_nums	batch_size	time	speedup
3	1	2048	13.0	1.000000
7	2	2048	10.0	1.300000
11	4	2048	11.0	1.181818

In [ ]:
```
l[l["batch_size"] == b]
```

Out[ ]:

	gpu_nums	batch_size	time	speedup
3	1	2048	13.0	1.000000
7	2	2048	10.0	1.300000
11	4	2048	11.0	1.181818

In [ ]:
```
# pd.set_option('max_columns',1000)
cols = []
cols.append("gpu_nums")
for b in [32, 128, 512, 2048]:
    cols.append(f"batch_{b}_time")
    cols.append(f"batch_{b}_speedup")
# print(cols)
df = pd.DataFrame(columns=cols)
for g in [1, 2, 4]:
    temp_df = pd.DataFrame(columns=cols)
    temp_df["gpu_nums"] = [g]
    for b in [32, 128, 512, 2048]:
        t = l[l["batch_size"] == b]
        t = t[t["gpu_nums"] == g]
        temp_df[f"batch_{b}_time"] = t["time"].values
        temp_df[f"batch_{b}_speedup"] = t["speedup"].values
    df = pd.concat([df, temp_df])
df.reset_index(inplace=True, drop=True)
df
```

Out[ ]:

	gpu_nums	batch_32_time	batch_32_speedup	batch_128_time	batch_128_speedup	batch
0	1	38.0	1.000000	16.0	1.000000	
1	2	67.0	0.567164	18.0	0.888889	
2	4	121.0	0.314050	30.0	0.533333	

In [ ]:
```
df.to_markdown("problem2/2_1_table_src_all.md")
df.to_csv("problem2/2_1_table_src_all.csv", index=False)
```

That speed up we use T = epoch_time, **T(1) / T(N)**.

According to the campuswire, we are suggested to use N * T(1) / T(N). But the not sure the T definition here.

To clarify, we use t = batch_time and T = epoch_time.

We can calculate using n * t(1) / t(N) if we would like to use the same problem size.

Speedup = T(1) / T(N) = n * t(1) / t(N), T is the epoch time, t is the batch time.

	gpu_nums	batch_32_time	batch_32_speedup	batch_128_time	batch_128_speedup	batch_
0	1	38	1	16	1	
1	2	67	0.567164	18	0.888889	
2	4	121	0.31405	30	0.533333	

If we use the **N * T(1) / T(N)** definition, we can get the following result:

In [ ]:
```
new_df = df.copy(deep=True)

for b in [32, 128, 512, 2048]:
    new_df[f"batch_{b}_speedup"] = new_df[f"batch_{b}_speedup"] * new_df["gpu_
new_df
```

Out[ ]:

	gpu_nums	batch_32_time	batch_32_speedup	batch_128_time	batch_128_speedup	batch
0	1	38.0	1.0	16.0	1.0	
1	2	67.0	1.134328	18.0	1.777778	
2	4	121.0	1.256198	30.0	2.133333	

2

Report for each batch size per gpu (i.e., 32, 128, 512 ...), how much time spent in computation (including CPU-GPU transferring and calculation) and how much time spent in communication in 2-GPU and 4-GPU case for one epoch. (hint You could use the training time reported in Question 1 to facilitate your calculation). (5) Expected Answer: First, describe how do you get the compute and communication time in each setup. Second, list compute and communication time in Table 2.

Comment

Expected Answer: Table 1 records the training time and speedup for different batch size up to 4 GPUs. Comment on which type of scaling we are measuring: weak-scaling or strong-scaling? Comment on if the other type scaling was used speedup number will be better or worse than what you we are measuring.

If we follow the definition, T is the epoch time, and we use T(1) / T(N) to calculate the speedup, the we use the stong-scaling (Strong scaling concerns the speedup for a fixed problem size with respect to the number of processors, and is governed by Amdahl's law) to measure the speedup, because we measure the epoch time, and the problem size for one epoch is fixed.

$$\text{Speedup } = 1/(s + p/N), \tag{1}$$

s is the serial part, p is the parallel part, N is the number of processors.

$$\text{Speedup} = T(1)/T(N) \tag{2}$$

If we use the weak-scaling (Weak scaling concerns the speedup for a scaled problem size with respect to the number of processors, and is governed by Gustafson's law.) to measure

the speedup.

$$\text{Speedup } = s + p * N \tag{3}$$

$$\text{Speedup } = N * T(1)/T(N) \tag{4}$$

Considering the larger problem size will be more efficient for more processors, if we use the weak-scaling to measure the speedup, we will get a better speedup efficiency.

Suggested Definition

If we use the **N * T(1) / T(N)** definition, T is the epoch time, then we are using the weak-scaling to measure the speedup, because we vary the problem size with respect to the number of processors.

Then if we use another definition (strong scaling), the score will be worse.

3

Report for each batch size per gpu (i.e., 32, 128, 512 ...), how much time spent in computation (including CPU-GPU transferring and calculation) and how much time spent in communication in 2-GPU and 4-GPU case for one epoch. (hint You could use the training time reported in Question 1 to facilitate your calculation). (5) Expected Answer: First, describe how do you get the compute and communication time in each setup. Second, list compute and communication time in Table 2.

For one gpu, we consider all the time is used for computation (including CPU-GPU transferring and calculation).

For two gpus or four gpus, the iteration number for each gpu is reduced, thus the computation time is reduced half of the original time, the communication time is the difference between the actual time and the reduced computation time.

In [ ]:
```
df = pd.read_csv("problem2/2_1_table_src_all.csv")
cal_col = []
for b in [32, 128, 512, 2048]:
    df.drop(columns=[f"batch_{b}_speedup"], inplace=True)
    cal_col.append(f"batch_{b}_time")
```

In [ ]:
```
df
```

Out[ ]:

	gpu_nums	batch_32_time	batch_128_time	batch_512_time	batch_2048_time
0	1	38.0	16.0	13.0	13.0
1	2	67.0	18.0	9.0	10.0
2	4	121.0	30.0	11.0	11.0

In [ ]:
```
communication_col = []
calculation_col = []
for b in [32, 128, 512, 2048]:
    communication_col.append(f"batch_{b}_communication_time")
    calculation_col.append(f"batch_{b}_calculation_time")
```

In [ ]:
```
# df["calculation_time"
communication_time = []
calculation_time = []
for g in [1, 2, 4]:
    if g == 1:
        temp_df = df[df["gpu_nums"] == g]
        temp_df = temp_df[cal_col].to_numpy()[0]
        base_df = temp_df
        communication_time.append(temp_df-temp_df)
        calculation_time.append(base_df)
        continue
    temp_df = df[df["gpu_nums"] == g]
    temp_df = temp_df[cal_col].to_numpy()[0]
    calculation_time.append(base_df / g)
    res_df = temp_df - base_df / g
    communication_time.append(res_df)
df[communication_col] = communication_time
df[calculation_col] = calculation_time
```

In [ ]:
```
df
```

Out[ ]:

	gpu_nums	batch_32_time	batch_32_speedup	batch_128_time	batch_128_speedup	batch
0	1	38.0	1.000000	16.0	1.000000	
1	2	67.0	0.567164	18.0	0.888889	
2	4	121.0	0.314050	30.0	0.533333	

In [ ]:
```
df.to_csv('./problem2/2_3.csv', index=False)
df.to_markdown('./problem2/2_3.md')
```

In [ ]:
```
display_cols = ["gpu_nums"] + communication_col
df[display_cols]
```

Out[ ]:

	gpu_nums	batch_32_communication_time	batch_128_communication_time	batch_512_com
0	1	0.0	0.0	
1	2	48.0	10.0	
2	4	111.5	26.0	

In [ ]:
```
display_cols = ["gpu_nums"] + calculation_col
df[display_cols]
```

Out[ ]:

	gpu_nums	batch_32_calculation_time	batch_128_calculation_time	batch_512_calculation_ti
0	1	38.0	16.0	13
1	2	19.0	8.0	6
2	4	9.5	4.0	3

4

Assume PyTorch DP implements the all-reduce algorithm as discussed in the class (reference below), calculate communication bandwidth utilization for each multi-gpu/batch-size-per-gpu setup. (5) Expected Answer: First, list the formula to calculate how long does it take to finish an allreduce. Second, list the formula to calculate the bandwidth utilization. Third, list the calculated results in Table 3.

References:

• PyTorch Data Parallel, Available at https://pytorch.org/docs/stable/modules/torch/nn/parallel/data_parallel.html.

• Bringing HPC Techniques to Deep Learning

Comment

We calculate the all-reduce (ring-allreduce).

P: number of processes

N: total number of model parameters

Scatter-reduce: Each process sends N/P amount of data to (P-1) learners, Total amount sent (per process): N(P-1)/P

AllGather: Each process again sends N/P amount of data to (P-1) learners

Total communication cost per process is 2N(P-1)/P

	Batch-size-per-GPU 32	**Batch-size-per-GPU 128**	**Batch-size-per-GPU 512**
	Bandwidth Utilization(GB/s)	Bandwidth Utilization(GB/s)	Bandwidth Utilization(GB/s)
2-GPU			
4-GPU			

We use the ResNet18 architecture to calculate the bandwidth utilization.

In [ ]:
```
from models.resnet import ResNet18
import numpy as np


model = ResNet18()

# calculate model parameters size
model_parameters = filter(lambda p: p.requires_grad, model.parameters())
params_count = sum([np.prod(p.size()) for p in model_parameters])
print(f"Model parameters: {params_count}, {params_count/1e6}M")

N = params_count
```

```
Model parameters: 11173962, 11.173962M
```

In [ ]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
```

```
import torch.optim as optim
import torchvision.transforms as transforms
import torchvision.datasets as datasets

train_transform = transforms.Compose([transforms.RandomCrop(32, padding=4),#r
                                      transforms.RandomHorizontalFlip(0.5),#r
                                      transforms.Normalize((0.4914, 0.4822, 0
                                      transforms.ToTensor(),])#convert to ten
train_dataset = datasets.CIFAR10("./cached_datasets/CIFAR10", train=True, dow
train_dataloader = torch.utils.data.DataLoader(train_dataset, batch_size=128,
dataset_size = len(train_dataset)
```

```
Files already downloaded and verified
```

In [ ]:

```
df
```

Out[ ]:

	gpu_nums	batch_32_time	batch_128_time	batch_512_time	batch_2048_time	batch_32_
0	1	38.0	16.0	13.0	13.0	
1	2	67.0	18.0	9.0	10.0	
2	4	121.0	30.0	11.0	11.0	

In [ ]:

```
bandwidth_cols = []
communication_count_cols = []
for b in [32, 128, 512, 2048]:
    bandwidth_cols.append(f"batch_{b}_bandwidth")
    communication_count_cols.append(f"batch_{b}_communication_count")
df[bandwidth_cols] = np.nan
df[communication_count_cols] = np.nan
```

In [ ]:

```
import math
append_df = []
# P = 1

for i,P in enumerate([1, 2, 4]):
    if i == 0:
        continue
    for b in [32, 128, 512, 2048]:
        iter_num = math.ceil(dataset_size/b)
        each_iter_communication_count = 2 * N * (P - 1) / P
        communication_count = iter_num * each_iter_communication_count
        communication_time = df["batch_" + str(b) + "_communication_time"][i]
        # print(communication_time)
        # print(f"communication_count: {communication_count}")
        bandwidth_utilization = communication_count / (communication_time * 1
        df["batch_" + str(b) + "_bandwidth"][i] = bandwidth_utilization
        df["batch_" + str(b) + "_communication_count"][i] = communication_cou
```

```
/tmp/ipykernel_768624/3602963460.py:16: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  df["batch_" + str(b) + "_bandwidth"][i] = bandwidth_utilization
/tmp/ipykernel_768624/3602963460.py:17: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
```

```
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  df["batch_" + str(b) + "_communication_count"][i] = communication_count
```

In [ ]:
```
df
```

Out[ ]:

	gpu_nums	batch_32_time	batch_128_time	batch_512_time	batch_2048_time	batch_32_
0	1	38.0	16.0	13.0	13.0	
1	2	67.0	18.0	9.0	10.0	
2	4	121.0	30.0	11.0	11.0	

In [ ]:
```
df[communication_count_cols]
```

Out[ ]:

	batch_32_communication_count	batch_128_communication_count	batch_512_communicatior
0	NaN	NaN	
1	1.746490e+10	4.369019e+09	1.0950
2	2.619735e+10	6.553529e+09	1.6425

No communication for gpu_nums = 1

In [ ]:
```
display_cols = ["gpu_nums"] + bandwidth_cols
df[display_cols]
```

Out[ ]:

	gpu_nums	batch_32_bandwidth	batch_128_bandwidth	batch_512_bandwidth	batch_2048_
0	1	NaN	NaN	NaN	
1	2	0.363852	0.436902	0.438019	
2	4	0.234954	0.252059	0.211945	

The bandwidth utilization is calculated by the following formula:

1. Calculate the iteration numer
2. Calculate the communication cost per process per iter.
3. Calculate the communication cost per process per epoch.
4. Using the epoch communication time to get the bandwidth utilization.

The above table unit is GB/s.

$$T_{train} = T_{communication} + T_{compute} \tag{5}$$
$$T_{compute} = T_1/N \tag{6}$$
$$T_{communication} = T_{train} - T_1/N \tag{7}$$

We have the T_{communication}, and we have all-reduce to get the model transfer size, we can get the bandwidth utilization by

$$\text{Bandwidth Utilization} = \frac{Size_{communication}}{T_{communication}} \tag{8}$$