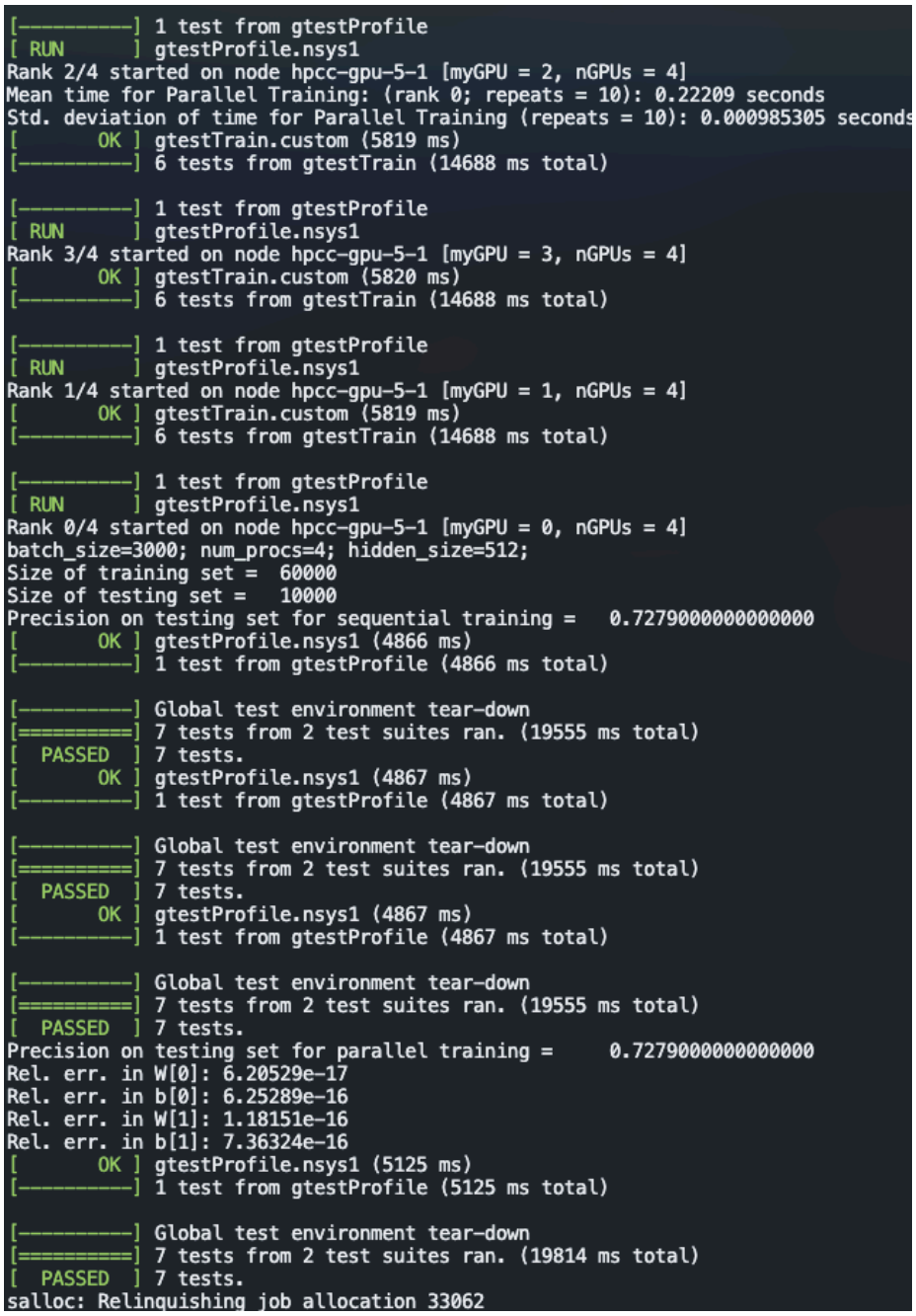
Problem1

Question1.1

The code passes all the tests in single-precision:

```
[----------] 1 test from gtestProfile
[ RUN      ] gtestProfile.nsys1
Rank 1/4 started on node hpcc-gpu-5-1 [myGPU = 1, nGPUs = 4]
Mean time for Parallel Training: (rank 0; repeats = 10): 0.101922 seconds
Std. deviation of time for Parallel Training (repeats = 10): 0.00129016 seconds
[       OK ] gtestTrain.custom (3719 ms)
[----------] 6 tests from gtestTrain (9096 ms total)

[----------] 1 test from gtestProfile
[ RUN      ] gtestProfile.nsys1
Rank 2/4 started on node hpcc-gpu-5-1 [myGPU = 2, nGPUs = 4]
[       OK ] gtestTrain.custom (3719 ms)
[----------] 6 tests from gtestTrain (9096 ms total)

[----------] 1 test from gtestProfile
[ RUN      ] gtestProfile.nsys1
Rank 3/4 started on node hpcc-gpu-5-1 [myGPU = 3, nGPUs = 4]
[       OK ] gtestTrain.custom (3719 ms)
[----------] 6 tests from gtestTrain (9096 ms total)

[----------] 1 test from gtestProfile
[ RUN      ] gtestProfile.nsys1
Rank 0/4 started on node hpcc-gpu-5-1 [myGPU = 0, nGPUs = 4]
batch_size=3000; num_procs=4; hidden_size=512;
Size of training set =  60000
Size of testing set =   10000
Precision on testing set for sequential training =   0.7279000282287598
[       OK ] gtestProfile.nsys1 (3167 ms)
[----------] 1 test from gtestProfile (3167 ms total)

[----------] Global test environment tear-down
[==========] 7 tests from 2 test suites ran. (12263 ms total)
[  PASSED  ] 7 tests.
[       OK ] gtestProfile.nsys1 (3167 ms)
[----------] 1 test from gtestProfile (3167 ms total)

[----------] Global test environment tear-down
[==========] 7 tests from 2 test suites ran. (12263 ms total)
[  PASSED  ] 7 tests.
[       OK ] gtestProfile.nsys1 (3167 ms)
[----------] 1 test from gtestProfile (3167 ms total)

[----------] Global test environment tear-down
[==========] 7 tests from 2 test suites ran. (12263 ms total)
[  PASSED  ] 7 tests.
Precision on testing set for parallel training =     0.7279000282287598
Rel. err. in W[0]: 3.22303e-08
Rel. err. in b[0]: 3.60887e-07
Rel. err. in W[1]: 6.63224e-08
Rel. err. in b[1]: 5.83023e-07
[       OK ] gtestProfile.nsys1 (3303 ms)
[----------] 1 test from gtestProfile (3303 ms total)

[----------] Global test environment tear-down
[==========] 7 tests from 2 test suites ran. (12400 ms total)
[  PASSED  ] 7 tests.
salloc: Relinquishing job allocation 33060
```

The code also passes all tests in double-precision (with smaller relative error)



Question1.2

I am using single-precision.

hidden_size=512, num_epochs = 1, learning_rate = 1e-2:

Precision on testing set for sequential training = 0.7279000282287598
Precision on testing set for parallel training = 0.7279000282287598

Increase hidden_size (to 2048), the precision is higher:

hidden_size=2048, num_epochs = 1, learning_rate = 1e-2:

Precision on testing set for sequential training = 0.8328999876976013
Precision on testing set for parallel training = 0.8328999876976013

Increase num_epochs (to 4), the precision is higher:

hidden_size=512, num_epochs = 4, learning_rate = 1e-2

Precision on testing set for sequential training = 0.8492000102996826
Precision on testing set for parallel training = 0.8492000102996826

Increase learning rate (to 1e-1), the test gtestProfile.nsys1 fails:

hidden_size=512, num_epochs = 1, learning_rate = 1e-1:

Precision on testing set for parallel training = 0.8374999761581421
Rel. err. in W[0]: 0.00109976
Rel. err. in b[0]: 0.0069902
main.cpp:253: Failure
Expected: (rel_err_W) < (1e-6), actual: 0.00109975913 vs 1e-06

main.cpp:254: Failure
Expected: (rel_err_b) < (1e-6), actual: 0.00699019665 vs 1e-06

Rel. err. in W[1]: 0.000702886
Rel. err. in b[1]: 0.00586254
main.cpp:253: Failure
Expected: (rel_err_W) < (1e-6), actual: 0.000702886085 vs 1e-06

main.cpp:254: Failure
Expected: (rel_err_b) < (1e-6), actual: 0.00586254383 vs 1e-06

[FAILED] gtestProfile.nsys1 (3377 ms)
[----------] 1 test from gtestProfile (3377 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (3378 ms total)
[PASSED] 0 tests.
[FAILED] 1 test, listed below:
[FAILED] gtestProfile.nsys1

Higher learning rates can lead to numerical instability, causing the model parameters (weights and biases) to change more drastically. This can result in more differences between the GPU and CPU implementations, especially since the GPU and CPU might handle floating-point arithmetic slightly differently due to their architectures.

It shouldn’t matter at this point because the difference is not large enough to cause the model to not converge.

Question1.3

learning_rate = 5*1e-3:
Precision on testing set for parallel training = 0.6211000084877014

learning_rate = 1e-2:

Precision on testing set for parallel training = 0.7279000282287598

Learning_rate = 2*1e-2:

Precision on testing set for parallel training = 0.7932000160217285

Learning_rate = 4*1e-2:

Precision on testing set for parallel training = 0.8442000150680542

Learning_rate = 6*1e-2:
Precision on testing set for parallel training = 0.8623999953269958

Learning_rate = 8*1e-2:
Precision on testing set for parallel training = 0.8404999971389771

The precision is the highest when learning_rate = 6*1e-2 for one iteration (i.e., lead to the fastest convergence for one iteration).

Starting from the starting point of learning_rate = 1e-2, when you decrease learning rate, then convergence is slower; when you increase the learning rate, the accuracy goes up, then goes down. So we need to tune the learning rate.

Question1.4

hidden_size=512:

Par loss at iteration 0 of epoch 0/ 200 = 2.30644011497497558594
Par loss at iteration 400 of epoch 20/ 200 = 0.32801288366317749023
Par loss at iteration 800 of epoch 40/ 200 = 0.23216548562049865723
Par loss at iteration 1200 of epoch 60/ 200 = 0.19068795442581176758
Par loss at iteration 1600 of epoch 80/ 200 = 0.16516509652137756348
Par loss at iteration 2000 of epoch 100/ 200 = 0.14717754721641540527
Par loss at iteration 2400 of epoch 120/ 200 = 0.13370348513126373291
Par loss at iteration 2800 of epoch 140/ 200 = 0.12308893352746963501
Par loss at iteration 3200 of epoch 160/ 200 = 0.11391963064670562744
Par loss at iteration 3600 of epoch 180/ 200 = 0.10609452426433563232

Precision on testing set for parallel training = 0.9517999887466431

The lowest loss I can get is 0.106. It is converging (almost converged, but not yet). The difference between losses of 400 iterations is decreasing, so it's converging.

If we increase hidden_size to 2048:

Par loss at iteration 0 of epoch 0/ 200 = 2.36607766151428222656
Par loss at iteration 400 of epoch 20/ 200 = 0.25472176074981689453
Par loss at iteration 800 of epoch 40/ 200 = 0.18716853857040405273
Par loss at iteration 1200 of epoch 60/ 200 = 0.15272243320941925049
Par loss at iteration 1600 of epoch 80/ 200 = 0.12963405251502990723
Par loss at iteration 2000 of epoch 100/ 200 = 0.11244009435176849365
Par loss at iteration 2400 of epoch 120/ 200 = 0.09906069934368133545
Par loss at iteration 2800 of epoch 140/ 200 = 0.08838729560375213623
Par loss at iteration 3200 of epoch 160/ 200 = 0.07962150126695632935
Par loss at iteration 3600 of epoch 180/ 200 = 0.07223895192146301270

Precision on testing set for parallel training = 0.9606999754905701

If we increase hidden size, then the model converges to a smaller loss (0.07), and the convergence can happen with fewer iterations. But for the same iterations, the computation takes longer.

If we decrease hidden_size to 100:

Par loss at iteration 0 of epoch 0/ 200 = 2.31282544136047363281
Par loss at iteration 400 of epoch 20/ 200 = 0.64255166053771972656

Par loss at iteration 800 of epoch 40/ 200 = 0.37497407197952270508
Par loss at iteration 1200 of epoch 60/ 200 = 0.29115310311317443848
Par loss at iteration 1600 of epoch 80/ 200 = 0.24907602369785308838
Par loss at iteration 2000 of epoch 100/ 200 = 0.22187447547912597656
Par loss at iteration 2400 of epoch 120/ 200 = 0.20232985913753509521
Par loss at iteration 2800 of epoch 140/ 200 = 0.18848386406898498535
Par loss at iteration 3200 of epoch 160/ 200 = 0.17731367051601409912
Par loss at iteration 3600 of epoch 180/ 200 = 0.16771581768989562988

Precision on testing set for parallel training = 0.9358999729156494

If we decrease the hidden size, then the model is trying to converge, after 200 iteration it is at the loss of 0.1677. For each iteration, the computation is faster, but the converge needs more iterations and may converge to a higher loss / precision at the end.

I would recommend the hidden_size of 512. Because it's a sweet spot in the trade-off space between per-iteration computation time and converged accuracy.

Problem2

Question2.1

The backward takes longer time than forward. This is a bit unexpected based on FLOPS computation from homework6: in homework6, the number of FLOPs performed in a single forward and backward pass are both approximately 2DHB.

The kernel TiledMatMul takes the most time. This is expected. Because other element-wise operations have fewer FLOPS.

The data copies from CPU to GPU takes quite a long time. I am a bit surprised about this.

Question2.2

According to the profile, 80.5% of the CUDA kernel time is spent on TiledMatMul, this suggests that we can optimize the TiledMatMul kernel to improve performance. More specifically, we can directly invoke the matrix multiplication kernel implemented in vendor libruary such as cuDNN or cuBLAS.

Another possible optimization is CUDA kernel fusion. Based on the profile, we see many small CUDA kernels, and launching each kernel has an overhead. If we could write fuse those small kernels together (e.g., the activation function can be fused into the GEMM kernel), we can further reduce the overhead of CUDA kernel launches.

Plus, using CPU pinned memory is another one that is worth trying because the data copies between CPU and GPU (e.g., copy_images_host_to_device) take a long time according to the profile. This can be done using cudaMallocHost

Question2.3

The most time-consuming **CUDA kernel** is TiledMatMul.

The number of FLOPs performed in a single forward pass is approximately 2DHB (corresponding to W(1)X).

The kernel takes 446us. D = input_size = 784; H = hidden_size = 512; B = batch_size = 3000, C=num_classes=10

So the number of flops is 2*784*512*3000 / 446us = 5.4 * 10^12 = 5.4 Teraflops

For cudaMemcpy, 77.3% of memory transfer time is spent on HostToDevice transfer. The transfer takes 382us.

Total elements transferred = D*H + H + H*C + C = 407050
Total bytes transferred = 4 * (D*H + H + H*C + C) = 1628200

For MPI_Allreduce, the total bytes transferred = 8 * (D*H + H + H*C + C) = 3256400
It takes 530us

Question2.4

Pflops = 2*784*512*3000 / 446us = 5.4 * 10^12 = 5.4 Teraflops

Ppci = 1628200 / 382us = 4.26 * 10^9 = 4.26 GB/s

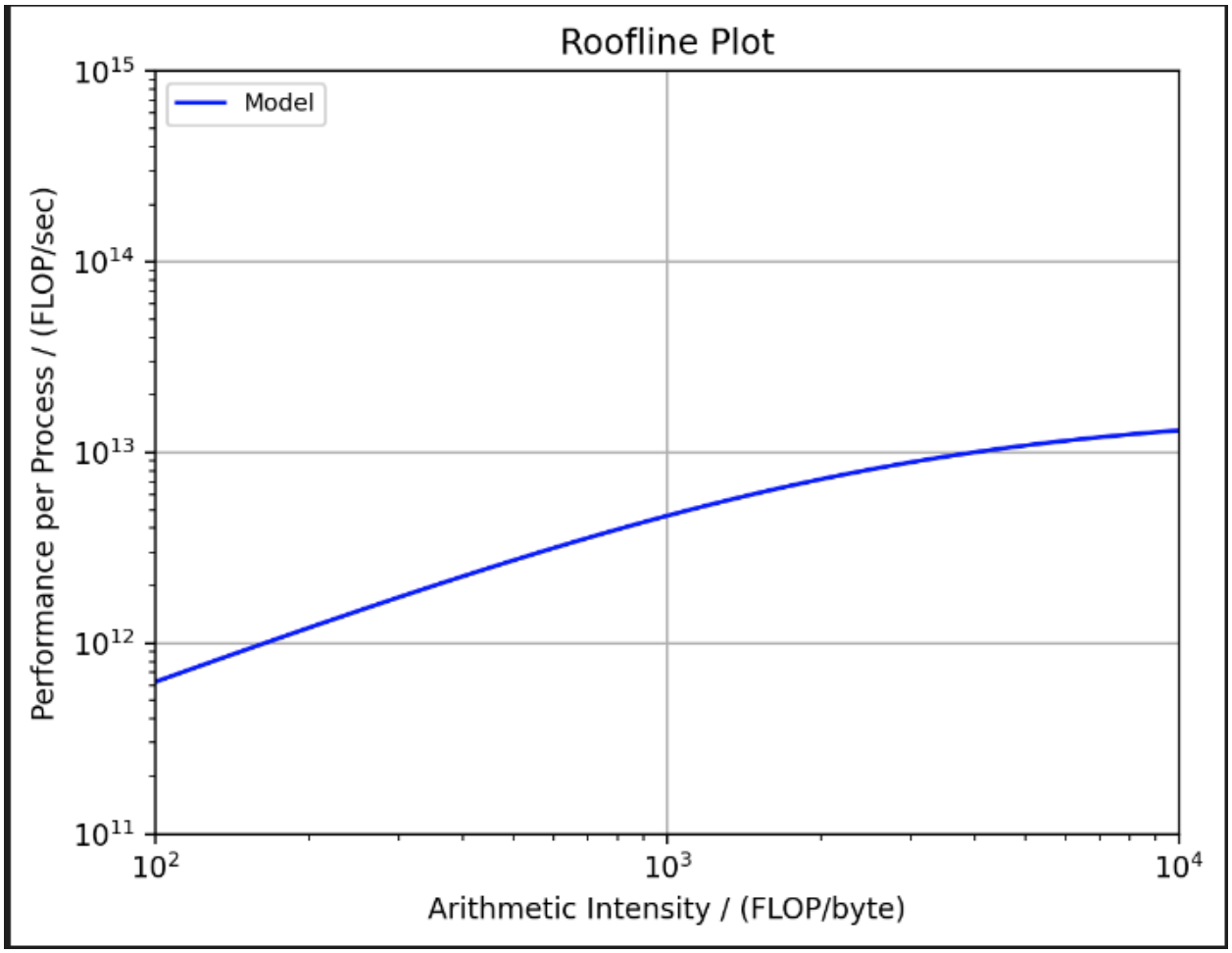
Pmpi = 3256400 / 530us = 6.1 * 10^9 = 6.1 GB/s

Question2.5

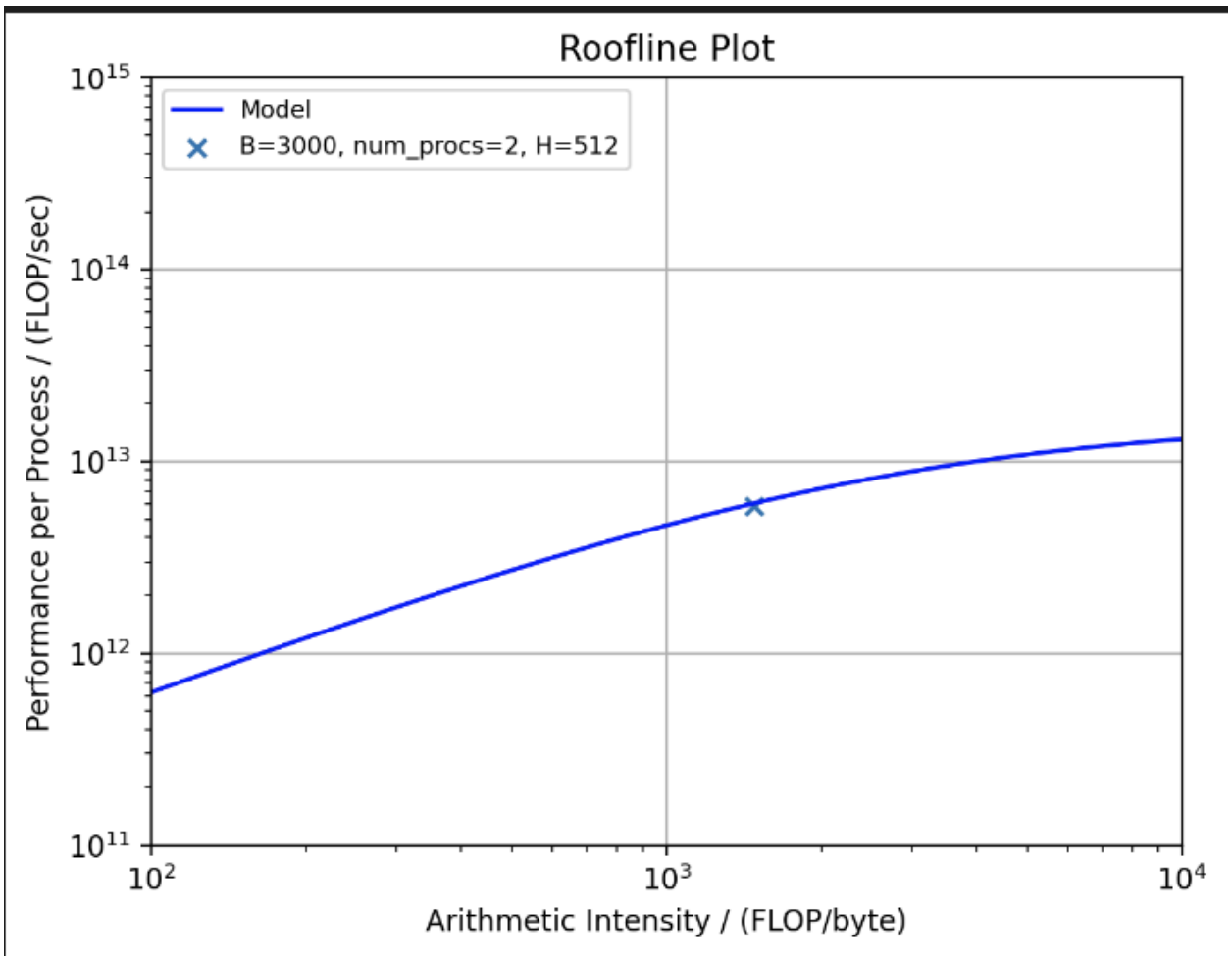
FLOPs / Total time

= FLOPs / (GPU computation time + cudaMemCopy time + MPI_collective_time)
= AI / (AI / Pflops + 1 / Ppci + 1/ Pmpi)

I don't know the theoretical maximum for Pmpi, so I assume it's 13x10^9.



Question2.6



The point is slightly below the roofline model.