CSCI-GA 3033
Cloud & Machine Learning

Project 1: ImageNet Analysis

Yuxuan Zheng — NYU (Spring 2024)

Report overview

- Compare the performances between CPU and GPU
- Compare models and analyse complexity
- Make roofline analysis

1 Introduction

Performance analysis is the key step to improve the speed of developing and deploying AI applications nowadays. This report analyses the performance of different neural network architectures, specifically the `AlexNet` and `ResNet18`, trained on the a tiny part of the ImageNet dataset, across varying compute environments with CPUs and GPUs. Complexity estimation and roofline analysis will be conducted for each setup for comparison.

2 Experiment Design

2.1 Dataset

Due to the time and resource limit, the dataset I use in the experiment is a tiny subset of the ImageNet dataset containing 0-499 classes of ImageNet. The size of the dataset is about 16GB.

2.2 Performance Comparison

For the environments, there are three setups:

1) CPU environment
   - CPU type: `Intel(R) Xeon(R) CPU @ 2.20GHz`
   - Number of cores: 6 cores
   - Memory: 83GB
2) Single GPU core environment
   - GPU type: `NVIDIA A100-SXM4-40GB`
   - Memory: 40GB
2) Dual GPU cores environment
   - GPU type (both): `NVIDIA A100-SXM4-40GB`
   - Memory (both): 40GB

Each environment is applied respectively on `AlexNet` and `ResNet18`.

We want to compare 1) the time to finish 1 epoch with default hyperparameter settings, and 2) the GPU kernel performance in terms of bandwidth and FLOPs per byte.

3 Hypothesis

We expect that the training speed under dual GPU cores environment is the fastest, whereas under CPU environment is the lowest. Since `ResNet18` is more computationally intensive than `AlexNet`, we expect that in the ideal case the kernel performance when training with `ResNet` is more on the computing-bound side, whereas the performance when training with `AlexNet` is more on the bandwidth-bound side, as shown in Figure 1.

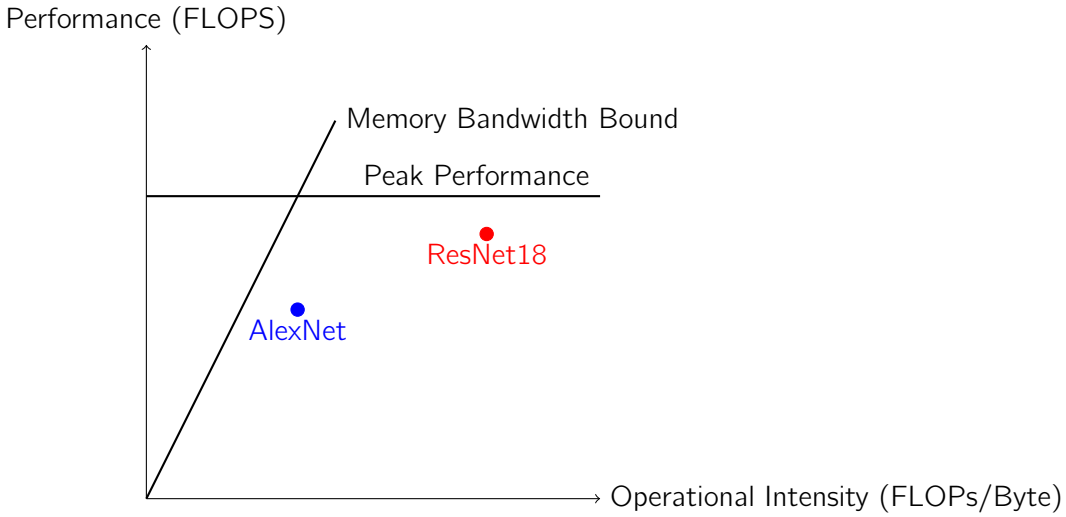

Figure 1: Expected performance with `ResNet18` and `AlexNet`

4 Complexity Analysis

4.1 Complexity Estimation

`AlexNet` consists of 5 convolutional layers followed by 3 fully connected layers. It also includes normalization and pooling layers. It has approximately 60 million parameters. In terms of FLOPs, the `AlexNet` requires roughly 5 MFLOPs to process a single 225×225 image through the network. [1]

On the other hand, `ResNet18` includes 18 layers with weights. It has around 11.7 million parameters, which is significantly less than `ResNet18` despite being deeper. `ResNet18` requires approximately 40 MFLOPs for a forward pass of a single 225×225 image. [2]

4.2 Complexity Measurement

We use `ncu` to profile the `main.py` script. The implementation is as follows.

main.py

```
...
import torch.cuda.profiler as ncu
...
parser.add_argument('--profile', default=False, type=bool, help="start profiling")
...

def train(train_loader, model, criterion, optimizer, epoch, device, args):
    ...
    for i, (images, target) in enumerate(train_loader):
        ...
        # compute output
        if args.profile:
            print("Start profiling...")
            ncu.start()
        output = model(images)
        if args.profile:
            print("End profiling...")
            ncu.stop()
            quit()
        loss = criterion(output, target)
        ...
...
```

To measures FLOPs on a single image, we need to divide the sum of FLOPs by the batch size (which is 256 by default). The script to compute the FLOPs is in Appendix A.

The result is that the number of FLOPs of a single 224×224 image going through `AlexNet` in a forward pass is 3,566,912, and that through `ResNet18` is 36,347,700. Both measurements are close to the estimated ones, and the number of FLOPS in `ResNet18` is 10 times than that in `AlexNet`.

5 Training Time Measurement

We measure the training time on the dataset of the first epoch with `AlexNet` and `ResNet18`. The results are as follows:

- `AlexNet`
  - **CPU:**: 3354 seconds
  - **GPU single core:** 1177 seconds
  - **GPU dual core:** 1026 seconds
- `ResNet18`
  - **CPU:** 2723 seconds
  - **GPU single core:** 1229 seconds
  - **GPU dual core:** 865 seconds

6 Roofline Chart

We generate the report with roofline plot using `ncu` and view it in the NVIDIA `ncu-ui` software. The script to generate the report is shown in Appendix A. The roofline plots for `AlexNet` and `ResNet18` are shown in Figure 2 and 3, respectively.

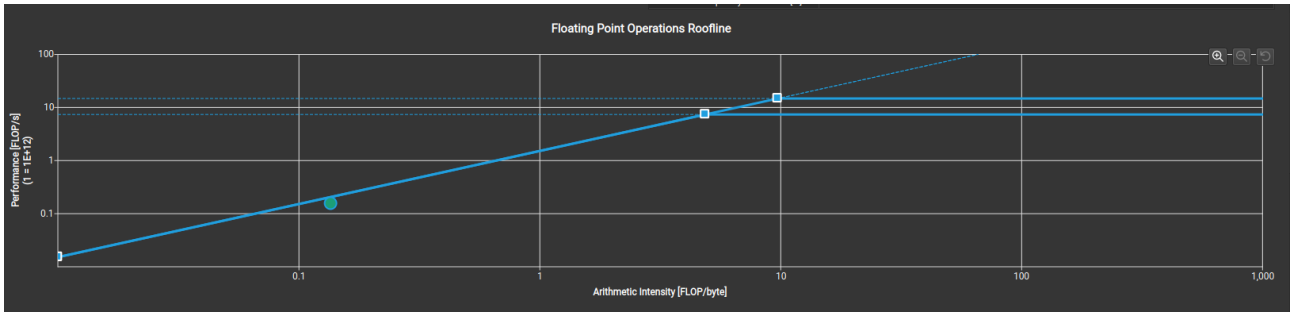

Figure 2: Roofline plot for `AlexNet`

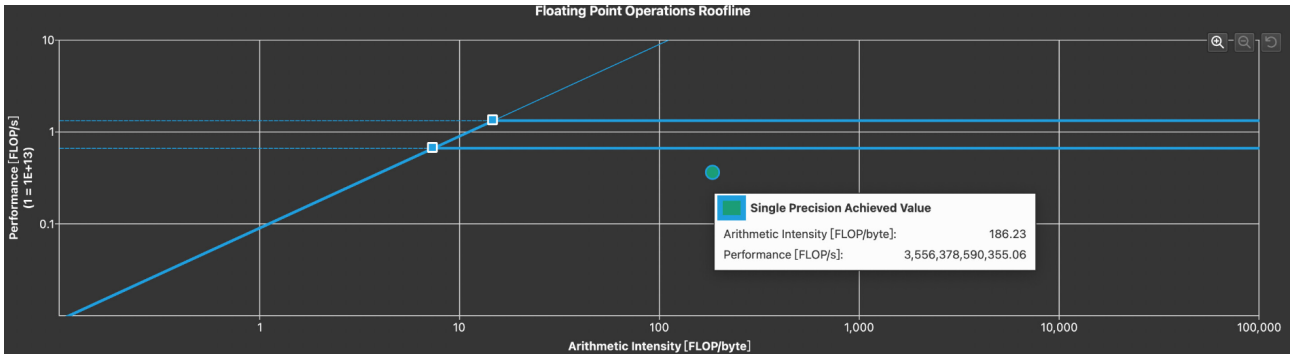

Figure 3: Roofline plot for `ResNet18`

We can see from the plots that the kernel behaves the same as what we expect. Both applications make good use of the system. Training with `AlexNet` is close to the bandwidth bound, whereas training with `ResNet18` is close to the compute bound. This verifies that the`ResNet18` network, though has much fewer number of parameters, is more computationally intensive.

7 Conclusions & Discussions

In this report, we conducted a comprehensive analysis of kernel performance across different hardware configurations—specifically, CPU, single-core GPU, and dual-core GPU—during the training of the ImageNet dataset utilizing `AlexNet` and `ResNet18` models.

Given the intricate design of `ResNet18`, it outperforms `AlexNet`, an older and simpler model, by achieving superior results with a smaller number of parameters. `ResNet18`'s architecture is notably more adept at leveraging parallel processing, a capability that is underscored by its significantly reduced training times when transitioning from a single-core to a dual-core GPU setup.

With regard to kernel performance, it is evident that the training operations for both models effectively harness the available system resources. The findings suggest that improvements on the GPUs could

particularly augment `ResNet18`'s performance further, as its training process is limited by computing bound.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

A Shell Scripts

Script for profiling:

profile.sh

```
#!/bin/bash

ncu --profile-from-start off --metrics smsp__sass_thread_inst_executed_op_fadd_pred_on \
  --target-processes all \
  python main.py --profile True --arch $1 --epochs 1
```

Script for measuring (summing up) FLOPs if we have the output of `profile.sh` saved in a file:

measure_flop.sh

```
#!/bin/bash

cat $1 | grep sum | awk '{ n += $3 } END { print n/256 }'
```

Script for generating reports (including the roofline plot):

gen_report.sh

```
#!/bin/bash

ncu --set full --profile-from-start off -o try2 python main.py --profile True --epochs 1
```