ImageNet Analysis

1 Introduction

1.1 Background

Deep learning models, notably CNNs, are pivotal in pushing the boundaries of AI, but their resource-intensive nature demands optimization for the hardware they run on, which is crucial for organizations aiming to leverage these technologies efficiently.

1.2 Purpose

Project 1 - ImageNet Analysis aims to examine the performance of AlexNet and ResNet18 models on the ImageNet dataset across NVIDIA's A100 and V100 GPUs, focusing on optimizing computational efficiency and memory usage.

1.3 Approach

The project employs a hybrid analysis method, starting with theoretical FLOPs and memory estimations using THOP(pen and paper estimation), followed by real-world measurements via NVIDIA's profiling tools, leading to a comparative roofline modelling to visualize and optimize GPU resource utilization.

2 Methodology

2.1 Experimental Setup

The experiments were conducted on NYU's HPC, utilizing NVIDIA's A100 and V100 GPUs to assess and compare the performance of the AlexNet and ResNet18 models.

2.2 Theoretical Estimation

Using THOP, preliminary estimations for computational load and memory usage were performed to set a benchmark for expected performance.

2.3 Empirical Profiling

The PyTorch ImageNet repository provided the groundwork for profiling, which was augmented with custom modifications to facilitate detailed analysis via NVIDIA's profiling tools.

2.4 Roofline Modelling

Performance data was visualized through roofline models in Google Colab, offering a succinct representation of each model's efficiency relative to GPU capabilities.

3 Experiment Design

3.1 Structuring the Experiments

Our experimental framework was methodically structured, designed to evaluate the operational performance of different neural network architectures on advanced GPU hardware.

3.2 GPU Flavours/Models

NVIDIA A100

The NVIDIA A100, built on the cutting-edge Ampere architecture, represents the pinnacle of NVIDIA's GPU technology, designed to accelerate the most demanding computational tasks in data science, scientific computing, and AI. Its robust architecture offers significant improvements in throughput and efficiency for both training and inference over its predecessors, making it an ideal choice for handling complex models like ResNet18 and AlexNet.

NVIDIA V100

The NVIDIA V100 GPU, based on the previous-generation Volta architecture, has been a stalwart in accelerating deep learning and high-performance computing tasks. It provides substantial computational power, facilitating rapid model training and inference, albeit with slightly lower performance metrics compared to the A100 when pushing the boundaries of the most resource-intensive applications.

ResNet18

ResNet18 is part of the Residual Network family, known for its deep yet efficient architecture that addresses the vanishing gradient problem through the use of skip connections. Its relatively simpler structure compared to deeper ResNets makes it a versatile model for a wide range of image recognition tasks, balancing performance and computational demands effectively across different GPU platforms.

AlexNet

AlexNet is considered a breakthrough model in the field of deep learning for image recognition, credited with winning the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012. Its architecture, while simpler than many modern CNNs, laid the groundwork for subsequent advancements in deep learning. AlexNet's design includes several convolutional layers followed by fully connected layers, making it computationally intensive relative to its size but manageable on advanced GPUs like the A100 and V100.

3.3 Hypotheses Formulation

We hypothesized that the A100's advanced architecture would outperform the V100, particularly in training deep models like ResNet18, and that architectural complexity would significantly influence computational load and efficiency.

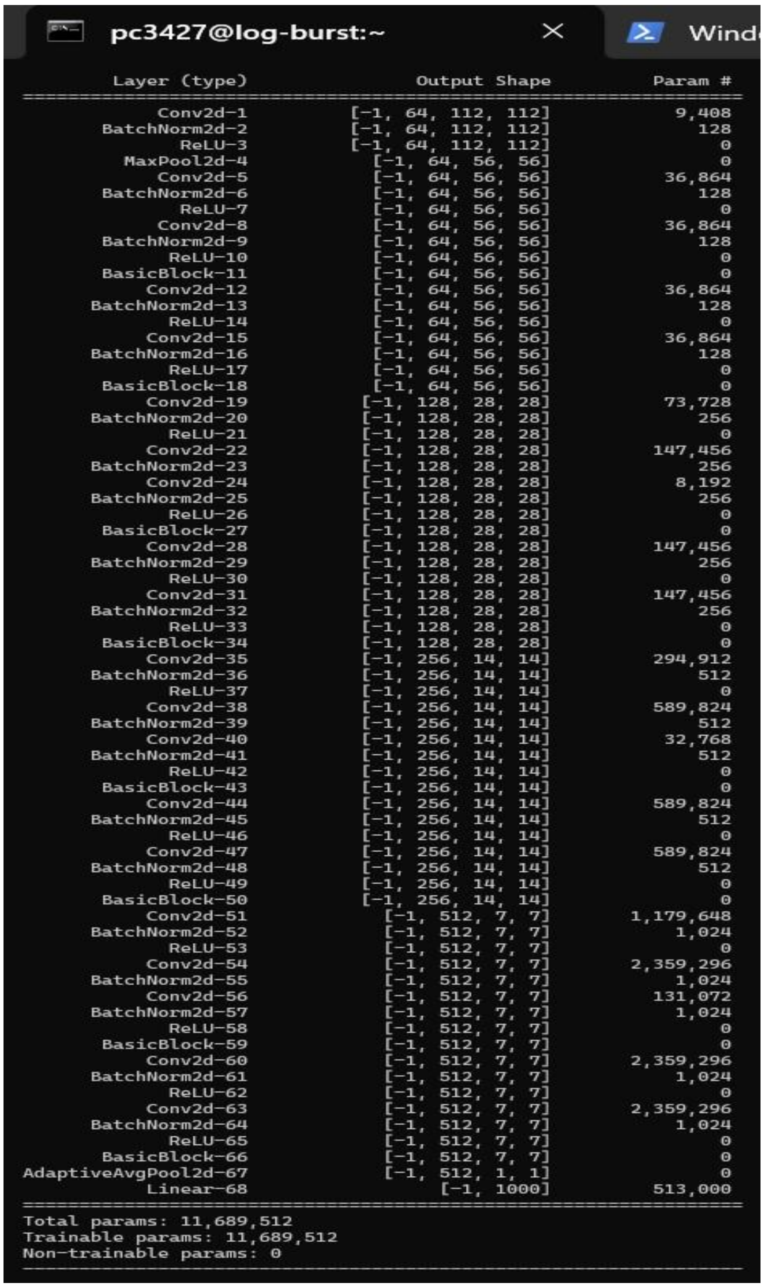
3.4 Dataset and Data Handling

Access to the ImageNet dataset was secured through NYU HPC, from which a representative subset was meticulously curated to balance the breadth of data diversity with the practicality of training time.

3.5 Experimental Parameters

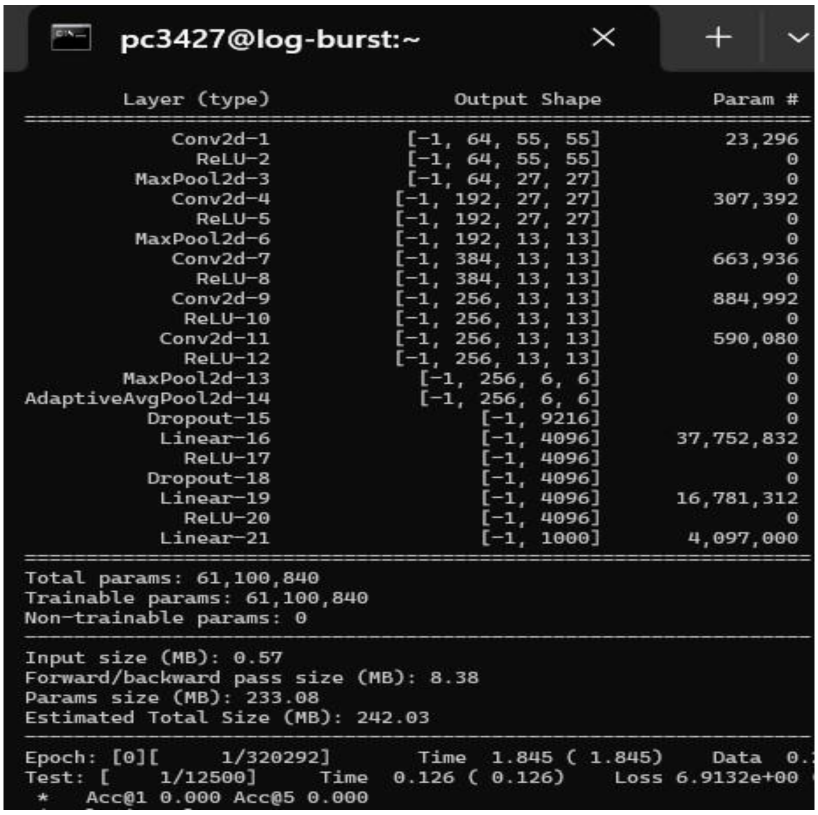
Training parameters were conscientiously chosen, reflecting common practice benchmarks: batch sizes were standardized, and a limited number of epochs were run to capture trends without necessitating full model convergence. Detailed model parameters were derived from **torchsummary** outputs to inform our FLOPs and memory usage estimations.

ResNet18



```
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 112, 112]           9,408
       BatchNorm2d-2         [-1, 64, 112, 112]             128
              ReLU-3         [-1, 64, 112, 112]               0
         MaxPool2d-4           [-1, 64, 56, 56]               0
            Conv2d-5           [-1, 64, 56, 56]          36,864
       BatchNorm2d-6           [-1, 64, 56, 56]             128
              ReLU-7           [-1, 64, 56, 56]               0
            Conv2d-8           [-1, 64, 56, 56]          36,864
       BatchNorm2d-9           [-1, 64, 56, 56]             128
             ReLU-10           [-1, 64, 56, 56]               0
       BasicBlock-11           [-1, 64, 56, 56]               0
           Conv2d-12           [-1, 64, 56, 56]          36,864
      BatchNorm2d-13           [-1, 64, 56, 56]             128
             ReLU-14           [-1, 64, 56, 56]               0
           Conv2d-15           [-1, 64, 56, 56]          36,864
      BatchNorm2d-16           [-1, 64, 56, 56]             128
             ReLU-17           [-1, 64, 56, 56]               0
       BasicBlock-18           [-1, 64, 56, 56]               0
           Conv2d-19          [-1, 128, 28, 28]          73,728
      BatchNorm2d-20          [-1, 128, 28, 28]             256
             ReLU-21          [-1, 128, 28, 28]               0
           Conv2d-22          [-1, 128, 28, 28]         147,456
      BatchNorm2d-23          [-1, 128, 28, 28]             256
           Conv2d-24          [-1, 128, 28, 28]           8,192
      BatchNorm2d-25          [-1, 128, 28, 28]             256
             ReLU-26          [-1, 128, 28, 28]               0
       BasicBlock-27          [-1, 128, 28, 28]               0
           Conv2d-28          [-1, 128, 28, 28]         147,456
      BatchNorm2d-29          [-1, 128, 28, 28]             256
             ReLU-30          [-1, 128, 28, 28]               0
           Conv2d-31          [-1, 128, 28, 28]         147,456
      BatchNorm2d-32          [-1, 128, 28, 28]             256
             ReLU-33          [-1, 128, 28, 28]               0
       BasicBlock-34          [-1, 128, 28, 28]               0
           Conv2d-35          [-1, 256, 14, 14]         294,912
      BatchNorm2d-36          [-1, 256, 14, 14]             512
             ReLU-37          [-1, 256, 14, 14]               0
           Conv2d-38          [-1, 256, 14, 14]         589,824
      BatchNorm2d-39          [-1, 256, 14, 14]             512
           Conv2d-40          [-1, 256, 14, 14]          32,768
      BatchNorm2d-41          [-1, 256, 14, 14]             512
             ReLU-42          [-1, 256, 14, 14]               0
       BasicBlock-43          [-1, 256, 14, 14]               0
           Conv2d-44          [-1, 256, 14, 14]         589,824
      BatchNorm2d-45          [-1, 256, 14, 14]             512
             ReLU-46          [-1, 256, 14, 14]               0
           Conv2d-47          [-1, 256, 14, 14]         589,824
      BatchNorm2d-48          [-1, 256, 14, 14]             512
             ReLU-49          [-1, 256, 14, 14]               0
       BasicBlock-50          [-1, 256, 14, 14]               0
           Conv2d-51            [-1, 512, 7, 7]       1,179,648
      BatchNorm2d-52            [-1, 512, 7, 7]           1,024
             ReLU-53            [-1, 512, 7, 7]               0
           Conv2d-54            [-1, 512, 7, 7]       2,359,296
      BatchNorm2d-55            [-1, 512, 7, 7]           1,024
           Conv2d-56            [-1, 512, 7, 7]         131,072
      BatchNorm2d-57            [-1, 512, 7, 7]           1,024
             ReLU-58            [-1, 512, 7, 7]               0
       BasicBlock-59            [-1, 512, 7, 7]               0
           Conv2d-60            [-1, 512, 7, 7]       2,359,296
      BatchNorm2d-61            [-1, 512, 7, 7]           1,024
             ReLU-62            [-1, 512, 7, 7]               0
           Conv2d-63            [-1, 512, 7, 7]       2,359,296
      BatchNorm2d-64            [-1, 512, 7, 7]           1,024
             ReLU-65            [-1, 512, 7, 7]               0
       BasicBlock-66            [-1, 512, 7, 7]               0
AdaptiveAvgPool2d-67            [-1, 512, 1, 1]               0
           Linear-68                 [-1, 1000]         513,000
================================================================
Total params: 11,689,512
Trainable params: 11,689,512
Non-trainable params: 0
----------------------------------------------------------------
```

Alexnet



```
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 55, 55]          23,296
              ReLU-2           [-1, 64, 55, 55]               0
         MaxPool2d-3           [-1, 64, 27, 27]               0
            Conv2d-4          [-1, 192, 27, 27]         307,392
              ReLU-5          [-1, 192, 27, 27]               0
         MaxPool2d-6          [-1, 192, 13, 13]               0
            Conv2d-7          [-1, 384, 13, 13]         663,936
              ReLU-8          [-1, 384, 13, 13]               0
            Conv2d-9          [-1, 256, 13, 13]         884,992
             ReLU-10          [-1, 256, 13, 13]               0
           Conv2d-11          [-1, 256, 13, 13]         590,080
             ReLU-12          [-1, 256, 13, 13]               0
        MaxPool2d-13            [-1, 256, 6, 6]               0
AdaptiveAvgPool2d-14            [-1, 256, 6, 6]               0
          Dropout-15                 [-1, 9216]               0
           Linear-16                 [-1, 4096]      37,752,832
             ReLU-17                 [-1, 4096]               0
          Dropout-18                 [-1, 4096]               0
           Linear-19                 [-1, 4096]      16,781,312
             ReLU-20                 [-1, 4096]               0
           Linear-21                 [-1, 1000]       4,097,000
================================================================
Total params: 61,100,840
Trainable params: 61,100,840
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 8.38
Params size (MB): 233.08
Estimated Total Size (MB): 242.03
----------------------------------------------------------------
Epoch: [0][     1/320292]       Time  1.845 ( 1.845)    Data  0.
Test: [    1/12500]     Time  0.126 ( 0.126)    Loss 6.9132e+00 (
 *   Acc@1 0.000 Acc@5 0.000
```

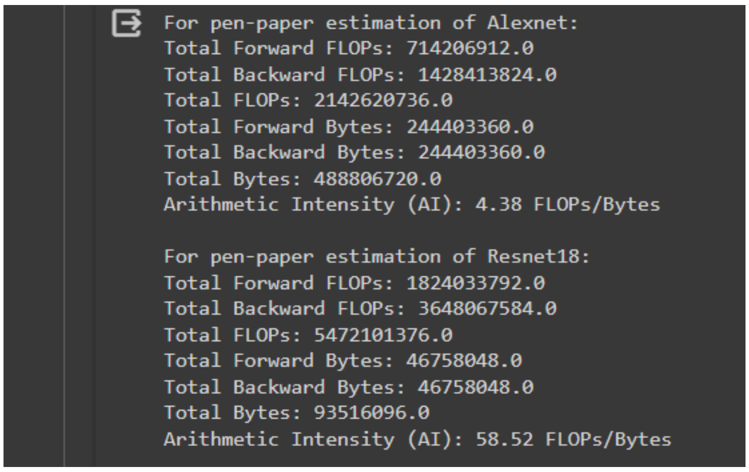
4 Complexity Estimation(Pen and Paper Method)

4.1 Estimation Approach

The complexity estimation began with a pen-and-paper strategy, utilizing the THOP library to forecast the computational intensity of our neural network models. This approach allowed us to predict the theoretical resource demands before actual profiling.

4.2 THOP Calculations

THOP provided a streamlined method to estimate the FLOPs and parameters of AlexNet and ResNet18. By injecting THOP's profiling function into the model's forward pass, we could capture the estimated computational burden without laborious manual calculations.
Thop_code



```
For pen-paper estimation of Alexnet:
Total Forward FLOPs: 714206912.0
Total Backward FLOPs: 1428413824.0
Total FLOPs: 2142620736.0
Total Forward Bytes: 244403360.0
Total Backward Bytes: 244403360.0
Total Bytes: 488806720.0
Arithmetic Intensity (AI): 4.38 FLOPs/Bytes

For pen-paper estimation of Resnet18:
Total Forward FLOPs: 1824033792.0
Total Backward FLOPs: 3648067584.0
Total FLOPs: 5472101376.0
Total Forward Bytes: 46758048.0
Total Backward Bytes: 46758048.0
Total Bytes: 93516096.0
Arithmetic Intensity (AI): 58.52 FLOPs/Bytes
```

4.3 Comparison and Analysis

These theoretical estimates were juxtaposed against empirical data from NVIDIA's profiling tools. Discrepancies between the THOP estimations and actual profiling highlighted the intricacies of in-practice model performance versus theoretical projections, offering deeper insights into model efficiency and resource allocation.

5 Roofline Modeling

5.1 Modeling Process

Roofline modeling is a method used to visualize a model's performance relative to the maximum potential of the hardware. We created roofline models to illustrate how the AlexNet and ResNet18 models use the computational power and memory bandwidth of the A100 and V100 GPUs.

Nvidia Profiling Steps:

- Cloned the PyTorch ImageNet repository from GitHub to NYU's High-Performance Computing (HPC) system.
- Implemented custom modifications to the cloned repository to enable detailed performance analysis using NVIDIA's profiling tools.
- Executed the **ncu** command, part of NVIDIA's Nsight Compute, to initiate the profiling process while running the AlexNet and ResNet18 models.
- Collected performance metrics were saved into a CSV file for further analysis.

Roofline Modeling Steps:

- Transferred the CSV file containing the performance data from the HPC system to Google Colab.
- Utilized Google Colab's data processing capabilities and visualization libraries to analyze the performance data.
- Plotted roofline models by mapping:
    - The GPUs' maximum theoretical performance and memory bandwidth as 'rooflines'.
    - The operational performance of AlexNet and ResNet18 models as data points relative to these rooflines.
- Analyzed the plots to evaluate:
    - How closely each model approached the GPUs' computational and memory bandwidth limits.
    - The efficiency of hardware utilization by each model.
    - Potential areas for optimization to enhance model performance and hardware synergy.

Roofline_modeling_code

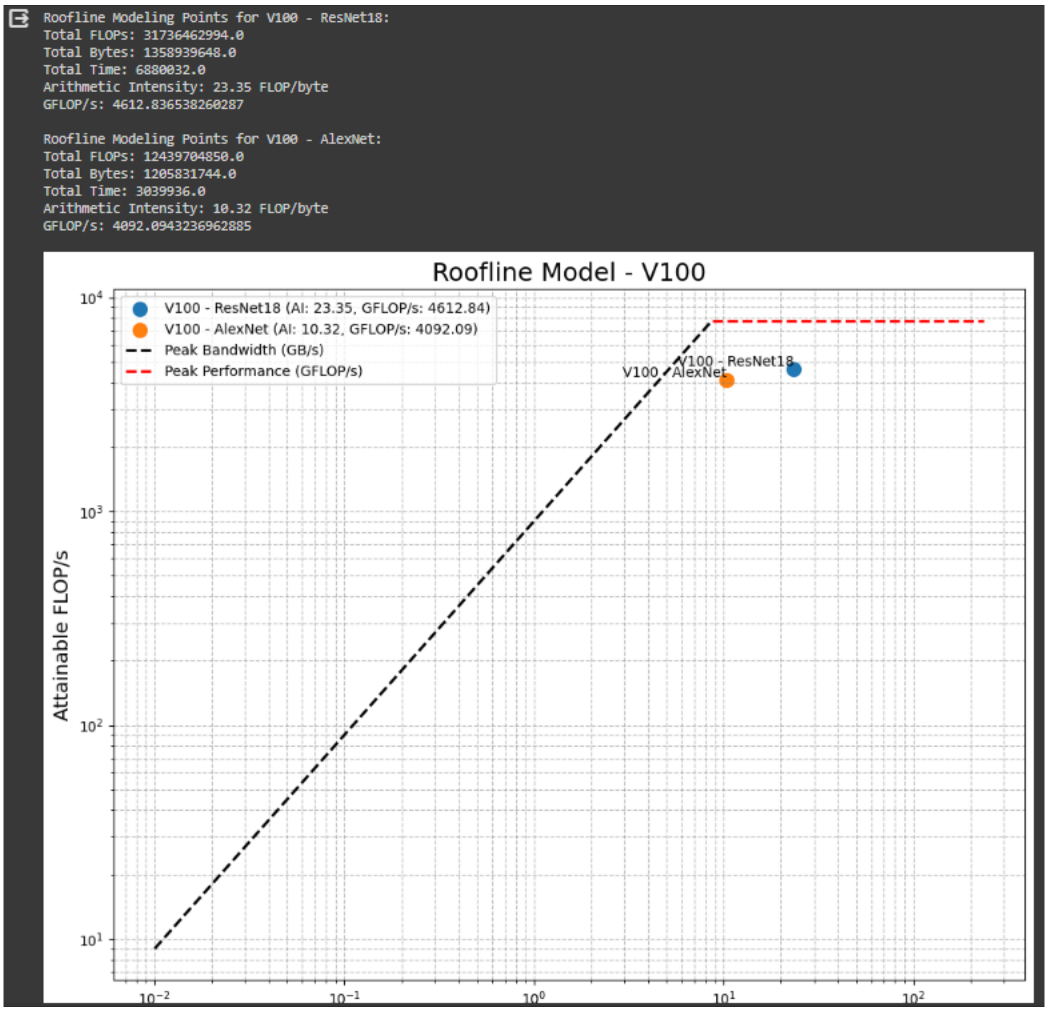
5.2 Visual Interpretation

Charts were crafted to show each model's operational point against the GPUs' peak performance. These visuals helped in identifying whether the models were compute-bound or memory-bound.

5.3 Interpretation of Models

By analyzing the roofline charts, we could discern the efficiency of each model in utilizing the available GPU resources. The A100's higher placement on the roofline indicated its superior handling of complex computations, while the V100 demonstrated robust, yet less optimal, performance.

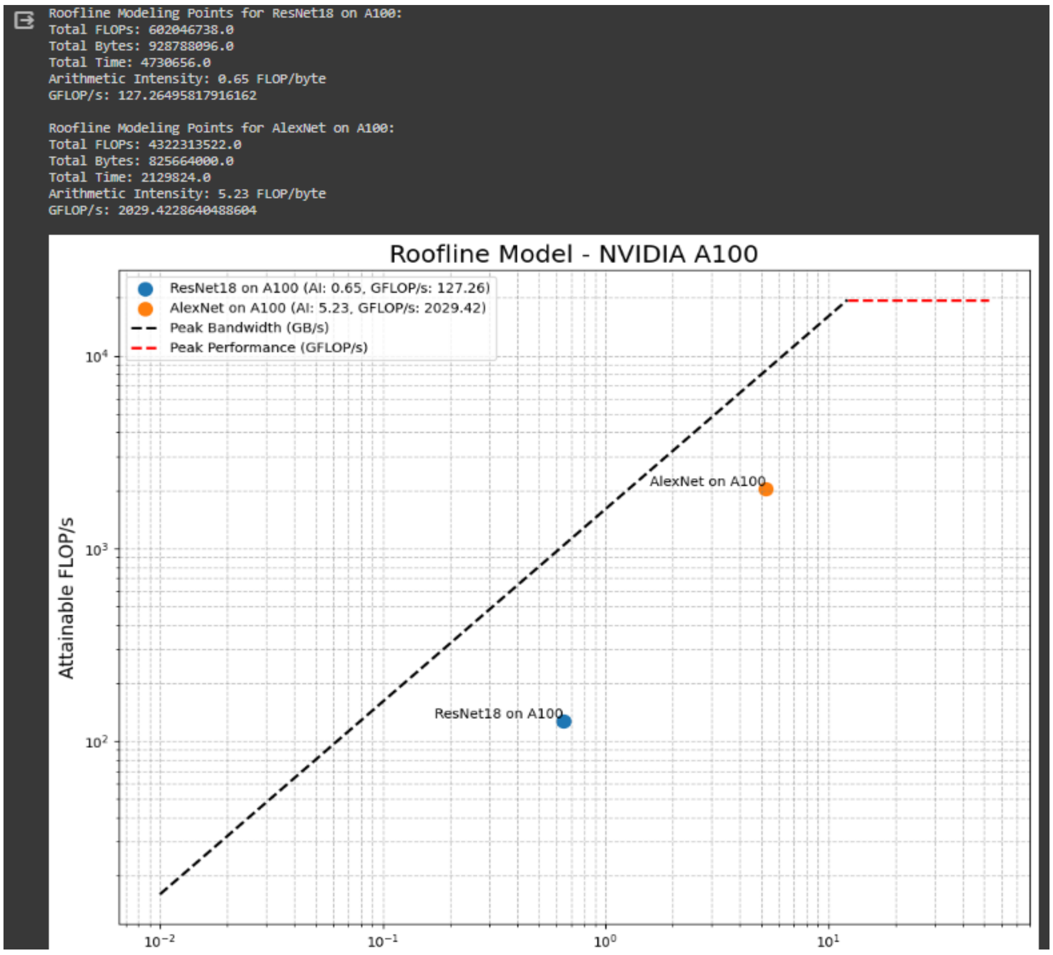
<u>V100</u>



```
Roofline Modeling Points for V100 - ResNet18:
Total FLOPs: 31736462994.0
Total Bytes: 1358939648.0
Total Time: 6880032.0
Arithmetic Intensity: 23.35 FLOP/byte
GFLOP/s: 4612.836538260287

Roofline Modeling Points for V100 - AlexNet:
Total FLOPs: 12439704850.0
Total Bytes: 1205831744.0
Total Time: 3039936.0
Arithmetic Intensity: 10.32 FLOP/byte
GFLOP/s: 4092.0943236962885
```

A100



```
Roofline Modeling Points for ResNet18 on A100:
Total FLOPs: 602046738.0
Total Bytes: 928788096.0
Total Time: 4730656.0
Arithmetic Intensity: 0.65 FLOP/byte
GFLOP/s: 127.26495817916162

Roofline Modeling Points for AlexNet on A100:
Total FLOPs: 4322313522.0
Total Bytes: 825664000.0
Total Time: 2129824.0
Arithmetic Intensity: 5.23 FLOP/byte
GFLOP/s: 2029.4228640488604
```

Roofline Model - NVIDIA A100
ResNet18 on A100 (AI: 0.65, GFLOP/s: 127.26)
AlexNet on A100 (AI: 5.23, GFLOP/s: 2029.42)
Peak Bandwidth (GB/s)
Peak Performance (GFLOP/s)
Attainable FLOP/s
AlexNet on A100
ResNet18 on A100
10^4
10^3
10^2
10^{-2}
10^{-1}
10^0
10^1

6. Discussion

In line with our hypotheses, the results illuminated performance variations between the GPUs, confirming the A100's superior handling of computational tasks over the V100. The analysis also reinforced the notion that architectural complexities, such as those found in ResNet18, do indeed demand significantly more from the hardware.

Observed trends suggested a near-linear increase in resource utilization with model complexity, although not always commensurate with an increase in accuracy. Anomalies in data pointed to potential bottlenecks in memory transfer speeds, particularly in models with intensive data I/O operations.

For practitioners, these findings underline the importance of matching model selection with the appropriate hardware to optimize for both performance and cost. They offer a prescriptive insight into planning infrastructure for AI deployment, ensuring that investments in technology are both judicious and efficacious.

7. Conclusion

- **Key Findings**:
    - The empirical profiling and roofline modeling conducted on NYU's HPC system highlighted the distinct performance capabilities of NVIDIA's A100 and V100 GPUs when running AlexNet and ResNet18 models. The A100 GPU demonstrated superior efficiency, leveraging its advanced architecture to better accommodate the computational demands of both models.
    - Our analysis confirmed the hypothesis that deeper, more complex models like ResNet18 require significantly more computational resources, yet the A100's advanced tensor cores minimized this gap.
    - Roofline modeling provided a clear visualization of the models' operational efficiency, revealing that while both models leveraged the hardware effectively, there is room for optimization, especially in memory utilization and computational intensity balancing.
- **Importance of Performance Profiling**:
    - This project underscored the critical role of performance profiling in the development and deployment of neural networks. By understanding the precise computational demands and identifying bottlenecks, developers can make informed decisions on model architecture adjustments and hardware selection to optimize performance.
    - The detailed insights obtained from NVIDIA's profiling tools and THOP estimations pave the way for fine-tuning model parameters and selecting the appropriate GPU resources, ensuring both cost-effectiveness and computational efficiency.
    - As neural network applications continue to expand, the methodologies employed in this project serve as a valuable framework for maximizing the potential of available hardware, thereby driving advancements in AI research and application development.

References

1. NVIDIA Corporation. (2020). *NVIDIA A100 Tensor Core GPU Architecture*. https://www.nvidia.com/en-us/data-center/a100/
2. NVIDIA Corporation. (2017). *NVIDIA Tesla V100 GPU Architecture*. https://www.nvidia.com/en-us/data-center/tesla-v100/
3. PyTorch. (n.d.). *Examples*. https://github.com/pytorch/examples/tree/main/imagenet
4. NYU HPC. (n.d.). Getting Started on Greene. https://sites.google.com/nyu.edu/nyu-hpc/hpc-systems/greene/getting-started?authuser=0
5. Google Colab. (Prithvi.). Roofline Model Visualization. https://colab.research.google.com/drive/1SIrZ84hC2DRR4_IjTozpjl3qEpfO3-1M?usp=sharing
6. Google Colab. (Prithvi.). NVIDIA Profiling Tool Output Analysis. https://colab.research.google.com/drive/175fdLyL8leKdLuoCWTy08pljvTZgTu3l?usp=sharing