REPORT

"Performance Evaluation of Deep Learning Models on CPU vs GPU for MNIST Classification"

Abstract:

This project aimed to solve a computationally intensive task by leveraging GPU acceleration. We used the MNIST dataset, a well-known collection of handwritten digits, to train a Convolutional Neural Network (CNN) model. The task was to classify images of digits, and we compared the performance of GPU and CPU in terms of training time and test accuracy. The results showed that, although GPU acceleration was applied, the training times for both GPU and CPU were nearly identical, with only a slight difference in test accuracy. This report outlines the methodology, implementation, results, and analysis of this comparison.

Introduction:

Problem Statement:

In many machine learning tasks, such as image classification, deep learning models require significant computational power for training. The rapid growth in dataset sizes and model complexity has led to the widespread adoption of GPU acceleration for training deep neural networks. However, the effectiveness of GPU acceleration depends on the task's nature, model architecture, and dataset size. The problem at hand is to investigate whether GPU acceleration leads to significant improvements in training time and accuracy for a simple image classification task on a well-established dataset like MNIST.

Importance of GPU Acceleration:

Graphics Processing Units (GPUs) are designed to handle parallel computation efficiently, making them well-suited for deep learning tasks. By offloading the training process from the CPU to the GPU, significant reductions in training time can often be achieved. In comparison to the CPU, which processes tasks sequentially, GPUs handle thousands of

tasks simultaneously, making them ideal for large-scale machine learning tasks. This project explores the effectiveness of GPU acceleration in training deep learning models.

Methodology:

Dataset:

We used the MNIST dataset, which consists of 70,000 grayscale images of handwritten digits (0-9), with 60,000 images used for training and 10,000 for testing. Each image is 28x28 pixels. This dataset is often used as a benchmark for image classification tasks and is widely adopted for testing new models and algorithms.

Preprocessing:

The MNIST dataset was loaded using TensorFlow, and the images were normalized to a range of [0, 1] by dividing the pixel values by 255.0. The dataset was then reshaped to fit the input requirements of a Convolutional Neural Network (CNN), with the shape of the data adjusted to (batch_size, 28, 28, 1) to represent grayscale images.

Model:

We used a Convolutional Neural Network (CNN) with the following architecture:

Conv2D Layer: 32 filters with a kernel size of 3x3.

MaxPooling2D Layer: Pooling size of 2x2.

Conv2D Layer: 64 filters with a kernel size of 3x3.

MaxPooling2D Layer: Pooling size of 2x2.

Flatten Layer: To flatten the 2D output into 1D.

Dense Layer: 128 units.

Output Layer: 10 units (representing the 10 classes).

The model was trained using the Adam optimizer and categorical cross-entropy loss.

GPU Acceleration:

The training process was run on both GPU and CPU. We used TensorFlow with GPU support enabled to accelerate the training process on the GPU. The model was trained for 5 epochs with a batch size of 64, and the time taken for each epoch was recorded for both CPU and GPU.

Results:

Training Time:

The training times for both CPU and GPU were as follows:

Training Time on GPU: 352.21 seconds (5 epochs)

Training Time on CPU: 350.68 seconds (5 epochs)

While both times were nearly identical, it was expected that the GPU would show faster results, especially for larger models and datasets. The similarity in times here could be due to the relatively simple nature of the MNIST dataset and the model's architecture.

Test Accuracy:

The accuracy on the test set after 5 epochs was:

Test Accuracy on GPU: 99.09%

Test Accuracy on CPU: 99.27%

Again, the difference in accuracy was minimal, with the CPU slightly outperforming the GPU. The lack of a significant performance gap might be attributed to the small model size and the simplicity of the MNIST dataset.

Speedup:

The calculated speedup was:

Speedup (CPU/GPU): 1.00

This means there was no speedup in the GPU performance compared to the CPU for this task. The speedup factor was close to 1, indicating similar performance for both CPU and GPU. This could be due to the simplicity of the task or inefficiencies in utilizing the GPU for such a simple model.

CPU Usage:

The CPU utilization during the training process was recorded at:

CPU Usage during Training: 40.4%

This suggests that the CPU was not fully utilized during training, possibly due to TensorFlow’s preference to offload computation to the GPU when available.

Visualizations:

To help visualize the performance differences, we can create graphs comparing the training time and test accuracy for both CPU and GPU:

```
import matplotlib.pyplot as plt

# Training Time
labels = ['GPU', 'CPU']
training_times = [352.21, 350.68]

plt.figure(figsize=(10, 5))
plt.bar(labels, training_times, color=['blue', 'orange'])
plt.title('Training Time Comparison (5 Epochs)')
plt.ylabel('Time (seconds)')
plt.show()

# Test Accuracy
accuracies = [99.09, 99.27]

plt.figure(figsize=(10, 5))
plt.bar(labels, accuracies, color=['blue', 'orange'])
plt.title('Test Accuracy Comparison')
plt.ylabel('Accuracy (%)')
plt.show()
```

Discussion:

Analysis of Results:

The performance results suggest that, for the MNIST dataset and the relatively simple CNN architecture used, GPU acceleration did not show a significant improvement over CPU training. This is a noteworthy outcome, as many expect GPUs to drastically reduce training times for deep learning tasks. However, several factors can contribute to the lack of a clear performance boost:

- **Simple Model**: The CNN model used is relatively small and does not involve a large number of parameters or complex layers that would benefit significantly from GPU acceleration.
- **Small Dataset**: The MNIST dataset is relatively small and does not require substantial computational power for processing, making the GPU's parallel processing capabilities less impactful.
- **Inefficiencies in TensorFlow**: The way TensorFlow utilizes the GPU may not have been fully optimized for the small size of the dataset and model, resulting in comparable performance to the CPU.

Scalability and Limitations:

This approach might scale better with larger datasets and more complex models. As the model architecture grows in complexity or the dataset size increases (e.g., using a dataset like CIFAR-10 or ImageNet), the GPU is likely to demonstrate a more significant speedup. However, for small datasets like MNIST, the overhead of using a GPU may not justify the small gains in speed.

The main limitation in this experiment is that it does not account for larger-scale tasks where the advantages of GPU acceleration are more pronounced. Future studies could explore the use of larger datasets and more complex models to evaluate the full potential of GPU acceleration.

Conclusion:

This project successfully demonstrated the implementation of GPU acceleration for training a neural network on the MNIST dataset. Despite the GPU being utilized, the results indicated minimal improvement in training time and accuracy when compared to the CPU. The simplicity of the dataset and model architecture likely contributed to these findings. In future extensions, using more complex models and larger datasets could provide a clearer advantage of using GPUs over CPUs.

Future Work:

- Explore the use of more complex models (e.g., ResNet, VGG) and larger datasets (e.g., CIFAR-10, ImageNet) to better evaluate the impact of GPU acceleration.
- Optimize GPU usage for smaller models or explore other forms of parallelization (e.g., using Dask or CuPy for large-scale data processing).

References:

1. **TensorFlow Documentation**. (2024). Retrieved from https://www.tensorflow.org
2. **MNIST Dataset**. (2024). Retrieved from http://yann.lecun.com/exdb/mnist/
3. **NVIDIA CUDA Documentation**. (2024). Retrieved from https://docs.nvidia.com/cuda/