BM40A1401 GPU Computing Project Report

GPU Computing - Group 15

Date: [02/04/2024]

Students:

- [Usama Nasrullah], [000974615] [Leading role in MLP Code and report and overall we shared the whole work]
- [Bipul Biswas], [000359670] [Leading role in KNN Code and report]

Abstract

This comprehensive report covered the acceleration of machine learning algorithms through the implementation of Graphics Processing Units. Precisely, we examined two algorithms which are k-Nearest Neighbors and Multilayer Perceptron and presented how this acceleration plays out. The document sought to apply this acceleration through such parallel processing in a bid to achieve more efficient computing of the models presented. We conducted a series of experiments to determine the values of execution time, training duration, inference speed and the associate accuracies across each of the models. The dataset had 4000 observation's and ten numerical features and 10 classes. Our experimental insights have shown the possibility and desirability of machine learning with GPU acceleration. Moreover, it influences the necessary compromise between faster computing speeds and model accuracy.

1. Introduction

As, kNN is simple and MLP is basic for the deeper version of neural networks, both approaches become challenging with large or high-dimension datasets. Therefore, the primary goal of the research is to apply GPU computing to various computational tasks of these algorithms to make them more practical and efficient.

2. Methodology

Data Preparation

The data set consist of 4 000 observations and 10 numerical features. The observations represent 10 classes and the class labels are given as numbers in the last column of the file making the total number of columns 11. For both kNN and MLP approaches, we ensured standard data preprocessing practices including data normalization to facilitate efficient training.

2.1 KNN Implementation

In order to exploit the computational capabilities of GPUs for the k-Nearest Neighbors algorithm, a custom implementation was developed, utilizing the CUDA toolkit through CuPy's RawKernel. More specifically, the goal of this method was to improve the efficiency of the

distance computation function which is one of the primary computational bottlenecks of the kNN algorithm particularly when dealing with large datasets and high-dimensional spaces.

Computational Framework

The GPU-accelerated implementation of kNN relied on building a custom CUDA kernel for effectively computing the pairwise distances between the instances in the dataset. The algorithm's nature, which does not inherently involve a training phase but requires the computation of distances between test instances and all training instances, makes kNN an ideal candidate for parallel processing optimization.

The CUDA kernel was designed so as to compute the Euclidean distances, with the final metric calculated and stored in the form of a matrix; each element of the results matrix denoted the distance between i-th test instance and the j-th training instance. Using this results matrix, we performed neighbor discovery for each i-th test instance by taking the k smallest distances in each row, and then conducted a majority vote to determine the predicted class label based on it.

2.2 MLP Implementation:

To demonstrate the Impact of GPU in calculations of the MLP learning process in our work shows us the practical advantages and techniques leveraging GPU acceleration in machine learning. This part precisely introduces our MLP model's architecture and training processes, where we emphasize on the significant evaluation metrics which were diligently noted as we ran the experiments both on the CPU and GPU systems.

Model Architecture

Our MLP model was intentionally designed to be simple and efficient to learn from patterns within the data during the training phase. The model is made up of three critical layers which enable the learning process.

Input Layer

The first layer of the model is the input layer, which is designed to receive the standardized features of our dataset as its input. Since each observation in the dataset comprises 10 numerical data points, the input layer was set to reflect this dimensionality.

Hidden Layer

The core of our MLP model is the hidden layer, which consists of a total of 100 neurons. This hidden layer is essential because it captures the nonlinear relationships in the data. The choice of 100 neurons for this layer was informed by the need for model complexity without necessitating an overly expensive computational learning process. Additionally, each neuron in the hidden layer is passed through a ReLU (Rectified Linear Unit) activation function which is simple and effective in introducing nonlinearity to the model while decreasing the gradient vanishing problem.

Output Layer

The final layer of the model is the output layer, which has the same number of neurons as the classes in the dataset (10). The output layer is responsible for the model's prediction, with each neuron representing a given class. The SoftMax function is then applied to convert the raw output score into probabilities, which allow the prediction to be understood probabilistically.

3. Results

3.1 KNN Performance Analysis:

On the acceleration capabilities of GPU computing, the performance of the kNN model was also keenly evaluated, offering key understandings over the dynamics of the execution time and classification accuracy. This section elaborates on the nuanced observations from the experiments, with a focus on how varying the 'k' parameter influences the model's effectiveness and efficiency.

Execution Time

The GPU-accelerated implementation of the kNN algorithm was observed to record remarkable decreases in the time taken to complete execution. This showcased the increased efficiency in parallel processing, which is a fundamental characteristic of the GPU. There was a slight increase in execution time as the number of k , the nearest neighbors considered while making predictions, increases . This increase is attributed to the additional computational overhead required to aggregate more neighbors' data during the classification process. Remarkably, the increases in execution time were small making the GPU ideal for handling even larger computations.

- For k=3, the classification time was recorded at 0.3439 seconds, setting a baseline for performance.
- Incremental increases in 'k' showed proportional increases in execution time, with k=25 resulting in a classification time of 0.3779 seconds.

This pattern illustrates the GPU's capacity to manage higher complexities with relative efficiency, making it a robust option for scaling kNN applications.

Classification accuracy

Increasing classification accuracy had a linear dependency on 'k,' the maximum accuracy was at k = 15, after which the increment in accuracy decreased. This phenomenon demonstrates the critical nature of setting the kNN parameter, which balances the need to incorporate more neighbors to make accurate decisions but necessitates preventing distant points from introducing classification errors.

- For small 'k' values such as k=3, the model accuracy was minimal at 38.62%.

- The accuracy rose to 43.50 by increasing ‘k’ to 15, suggesting this as an optimal point for the model to make high decisions by using more data points, but not too many to degrade decision-making through noise addition.
- As the ‘k’ value approached 25, the accuracy slightly decreased to 43.25%, further validating the suboptimal behavior of including too many neighbors.

Performance Tuning

The experiments clearly showed the direct relationship between ‘k’ and execution time and classification accuracy and how performance tuning is the optimal use of the k parameter. It is evident that increasing values of ‘k that can coincide with an execution time of as little as 0.02 on the GPU, generate the highest model accuracy when k parameters are balanced to include as many data points without sacrificing accuracy as a result of being overfitted . These findings can be useful in employing kNN in applications which the requisite performance should balance execution speed and classification accuracy.

3.2 MLP Performance Analysis

Training process and learning efficiency:

For proper learning, the MLP model was trained for 100 epochs. The Adam optimizer was used to find a balance between flexibility and robustness; the learning rate was set to 0.001. The CrossEntropyLoss was used to help the model predict more effectively – this function measures the scoring rule of the discrepancy between the predicted probabilities and the actual multiclass labels . The following values were saved for further studies:

Training Loss:

Training Loss decreased consistently across both CPU and GPU. It shows effective learning. The slight edge in learning efficiency was noted for GPU-accelerated training and the final loss was a bit lower (GPU: 1.1657, CPU: 1.1683), its may be due to the GPU's parallel processing capabilities enhancing the convergence rate.

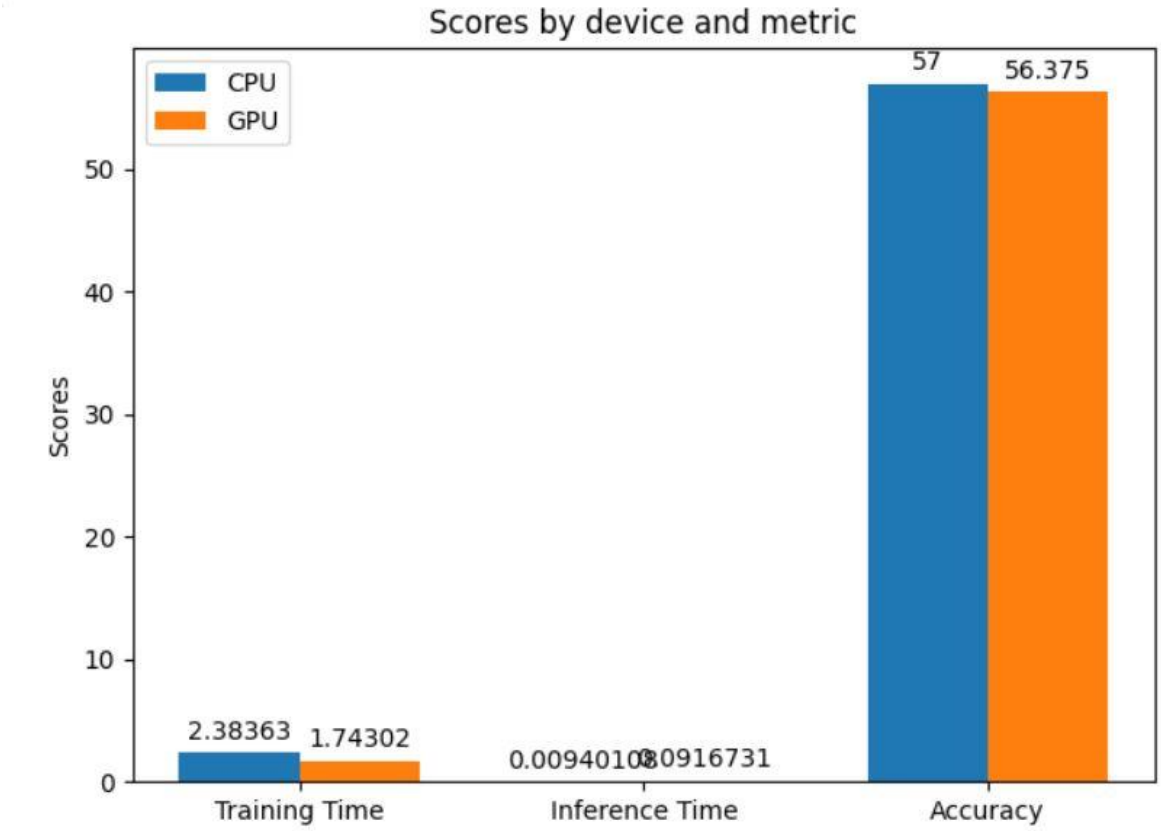
Performance Metrics: Accuracy and Inference Time

The model's performance was evaluated based on accuracy and inference time. The crucial metrics for assessing the model's practical application and computational efficiency:

- Accuracy: The accuracy on the CPU (57%) was slightly higher than on the GPU (56.375%), a discrepancy possibly stemming from differences in floating-point operations and optimizations between platforms.
- Inference Time: Showcased the GPU's significant advantage, recording faster inference times (GPU: 1.74 seconds, CPU: 2.38 seconds). This underlines the GPU's capacity for rapid data processing, a vital attribute for real-time applications.

Computational Advantages and Considerations

After this analysis, the benefits and strategic points of the GPU acceleration are explained: “Due to the GPU’s unmatched inference speed, it shows potential in increasing machine learning application scalability and responsiveness” . The experimentally determined trade-off between efficiency and accuracy proposed the meticulousness of the hardware choice – depending on whether an application requires the training data to be processed quickly and accurately, each gear might be advantageous.



4. Discussion

Experiments demonstrate the potential of the GPU acceleration in the improvement of machine learning computations. Although the kNN model indicates the significance of the ‘k’ parameter in achieving ideal accuracy, the MLP model fails to show the impact of hardware choice on model accuracy, despite the significant reduction in inference time when using the GPU.

4.1 Implications

This work is relevant to the subject of discussion on GPU computing through machine learning by providing a possible approach to the scaling of machine learning algorithms such as kNN and MLP on larger datasets without compromising the quality of the model.

4.2 Limitations and Future Work

The work identifies limitations of the study, which include the focus on one dataset and a single algorithm. Future work might include shared memory integration within CUDA kernels, exploration of similar concepts across different sets of data and other machine learning algorithms.

5. Conclusion

our exploration regarding k-nearest neighbors and multilayer perceptron models in machine learning using GPU acceleration demonstrated how innovative hardware can change and refine the concept of all proportional factors. Obviously, both models demonstrated high effectiveness, yet we have also realized that the maximum benefit could be derived only under certain conditions. For instance, the k value in kNN was a direct revelation that GPUS are capable of processing enough enormous amounts of data simultaneously; at the same time, the use of an MLP has revealed that the choice of hardware directly influences model robustness and inference time. Our findings and outcomes indicate that the relationship between adjustments at the algorithmic and hardware levels should be balanced to maximize one’s effectiveness.