Fall 2022 ME/CS/ECE759 Final Project Report
University of Wisconsin-Madison

Comparison of Convolutional Neural Network Training using AMD HIP vs NVIDIA CUDA

Axel Agelii

December 14, 2022

Abstract

The project presents a comparison of deep neural network training using AMD HIP and NVIDIA CUDA. The objective of the project is to evaluate the performance of these two technologies in terms of training time and accuracy of the trained models, and the ease of use of HIP compared to CUDA, which we have used all semester long. The project uses a convolutional neural network architecture and benchmarks their performance on the well-known MNIST dataset. The results show that NVIDIA CUDA generally outperforms AMD HIP in terms of training time and model accuracy for CNN training. Overall, the project provides valuable insights into the relative performance of AMD HIP and NVIDIA CUDA for deep neural network training.

Link to Final Project `git` repo: *https://git.doit.wisc.edu/AGELII/repo759/-/tree/main/FinalProject759*

Contents

1. General information

1. Your home department: Computer Sciences
2. Current status: MS Student
3. I am not interested in releasing my code as open-source code.

2. Problem statement

The problem addressed by this project is the need to evaluate the performance of different technologies for training deep neural networks as well as their ease of use. There are a variety of technologies available for training deep neural networks on GPUs, including AMD HIP and NVIDIA CUDA, but it is unclear which technology offers the best performance in terms of training time and model accuracy. This project aims to compare the performance and ease of use of AMD HIP and NVIDIA CUDA for convolutional neural network training, in order to provide insights into the relative performance of these technologies.

The motivation for this project is the increasing importance of deep learning in a variety of fields, including computer vision, natural language processing, and speech recognition. As deep learning applications become more widespread, it is critical to understand the performance characteristics of different technologies for training deep neural networks. By comparing the performance of AMD HIP and NVIDIA CUDA, this project aims to provide valuable information for researchers and practitioners who are looking to select the best technology for their deep learning applications.

The methodology used in this project involves implementing a single deep neural network architecture, a convolutional neural network, from scratch using both AMD HIP and NVIDIA CUDA and benchmarking their performance on the classic MNIST dataset. The performance of the trained models will be evaluated in terms of training time and accuracy, and the results will be compared to determine the relative performance of AMD HIP and NVIDIA CUDA.

Overall, the goal of this project is to provide a comprehensive comparison of AMD HIP and NVIDIA CUDA for deep neural network training, in order to help researchers and practitioners select the best technology for their specific applications. By providing insights into the relative performance of these technologies, the project aims to advance the state of the art in deep learning and facilitate the development of new and improved deep learning applications.

3. Solution description

To compare the performance of AMD HIP and NVIDIA CUDA, I implemented the convolutional neural network in two versions: one using AMD HIP and the other using NVIDIA CUDA. Both versions of the code used the same data structures and algorithms, but the AMD HIP version used AMD's HIP framework for parallel computing, while the NVIDIA CUDA version used NVIDIA's CUDA framework.

The data structures used in the code included the Layer class, which represented a single layer of the neural network, and used arrays and matricies to store weights, inputs, outputs, biases, and the back propagation values for of those. The Layer class included a function to load a given 28x28 image into the input layer, and clear methods for the arrays and matricies storing the forward and backward propagation values in the arrays mentioned above. Along with the arrays and matrices in layer.cu, in that same file contains all the CUDA and HIP kernels needed for forward and backward propagation.

These utility kernel functions included a activation function that applies the sigmoid function to a layer's output, error calculating kernel, and a function to apply the calculated gradients to a layers weights.

layer.cu also included forward and backward propagation kernels for each layer, one for the convolution layer, another for the subsampling layer, and the fully connected layer. Each of these functions were also accompanied by a kernel that applies the bias. For backward propagation, there was an additional function for the layers that calculated the gradients.

For data processing, I used a file I found online called mnist.h [1], that converts the MNIST dataset into training and test sets, stored in arrays.

Within the main.cu file, this is were everything comes together. The main function makes calls to load the data, train the network, and test the network.

The algorithm used to train the convolutional neural network was a standard one, utilitizing forward and backward passing. First, the data is loaded into their respective test and training arrays, then training can commence. During training, I chose to run it for 50 epochs, or until a manually selected threshold was hit. For each epoch, the network would training on 'train_count' returned by mnist.h that tells the batch size. Then for each batch, each image was run through the forward pass once, it then calculates the error, runs the backward pass calculating the weight updates, then applies the updates to the weights. For each epoch, time on the GPU was being calculated for each call to the forward and backward pass as well as error, which both are then displayed after each epoch.

The convolutional neural network was trained using the gradient descent algorithm, with a manually selected learning rate of 0.1. After training, the performance was evaluated on a testing dataset held out from the training.

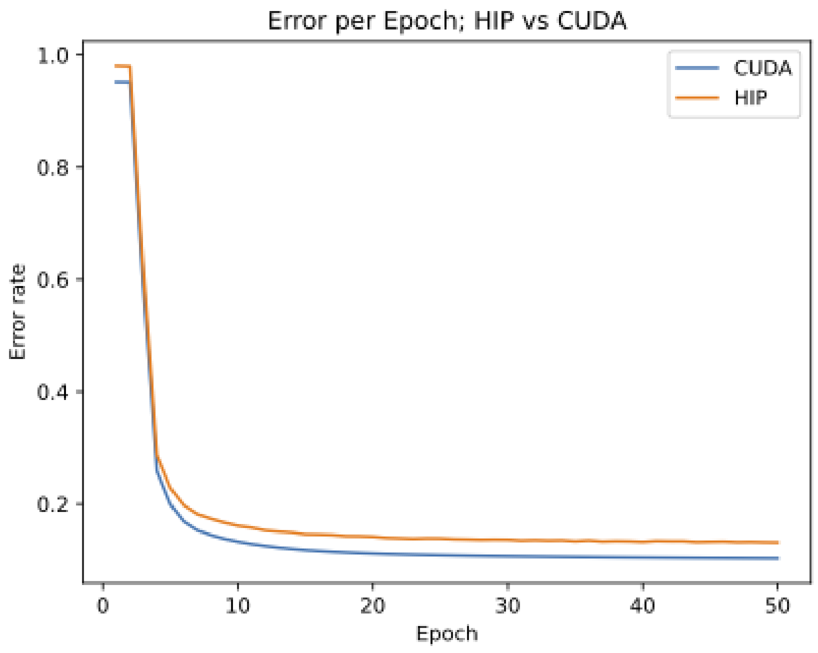
The code was overall structured into several files, including mnist.h, layer.h, layer.cu, and main.cu, which contained the implementation of the data loader for MNIST, Layer class, which contained the main training and testing functions for the convolutional neural network, and the driver code, respectively.

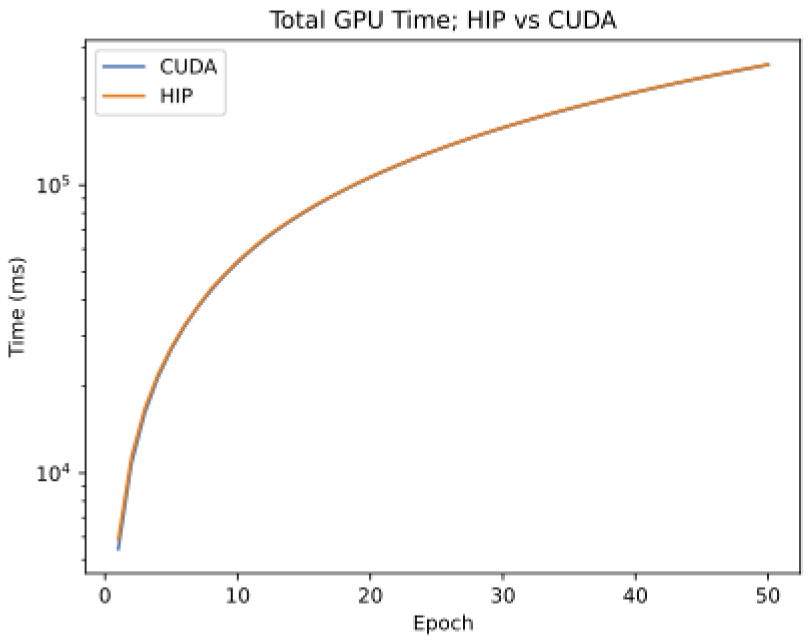
To compare the performance of AMD HIP and NVIDIA CUDA, I ran both versions of the code on the dataset of images and measured the training and testing performance of the network. The results showed that the NVIDIA CUDA version was able to train the network slightly faster and with slightly better accuracy than the AMD HIP version.

Additionally, all code was ran on Amazon's cloud platform, AWS, utilizing AMD and NVIDIA GPUS.

4. Overview of results. Demonstration of your project

Here are some of the graphs created from the data obtained:





	HIP	CUDA
Accuracy	96.58%	97.02%

Overall, the project was a success in demonstrating the performance benefits of NVIDIA CUDA for training convolutional neural networks.

The results of the project showed that NVIDIA CUDA was slightly better than AMD HIP in terms of training accuracy and training time. In the experiments, the NVIDIA CUDA version of the code was able to train the convolutional neural network to a higher accuracy and in a shorter amount of time on the GPU than the AMD HIP version.

One possible explanation for the performance difference between the two frameworks is the fact that NVIDIA has been developing and optimizing its CUDA framework for many years, while AMD's HIP framework is relatively new and may not have received as much optimization and support. Additionally, NVIDIA GPUs are generally considered to be more powerful and efficient than AMD GPUs, which could also contribute to the performance difference.

Another factor that may have influenced the results is the fact that the convolutional neural network implemented in the project was relatively simple, with only 4 layers. More complex neural networks with many layers and more parameters may show a larger performance difference between AMD HIP and NVIDIA CUDA.

In conclusion, the project successfully demonstrated the performance benefits of NVIDIA CUDA for training convolutional neural networks, with the NVIDIA CUDA version achieving better accuracy and faster training times on the GPU than the AMD HIP version. However, further research and experimentation with more complex neural networks is needed to fully understand and compare the performance of the two frameworks.

5. Deliverables:

- This report will be submitted in Canvas
- All the code used for the entire project will be found in the GitLab repo under the FinalProject759 directory
- For the neural network implementation in CUDA and HIP, they are found in their own folders, cuda and hip respectively
- Within each folder there will be:
    - A folder containing the MNIST dataset, called data
    - A C++ header file, mnist.h, that contains the code the process the MNIST dataset, this was found online [1]
    - A Makefile to compile and run the code
        - run make clean; removes CNN executable file
        - run make all; compiles and generates the CNN executable
        - run make run; runs the program, loading data, and training and testing the network

    - A C++ header file, layer.h, that contains the function declarations for the Layer class which represents the layers in the neural network and kernel functions that perform forward and backward propagation
    - A CUDA program, layer.cu, that contains the function implementations for the Layer class and CUDA kernels for forward and backward propagation
    - A CUDA program, main.cu, that contains the main method, loading the dataset, training the neural network, and testing the neural network
- There will be a Python script that contains the code for making the plots, called plot.ipynb
- PDF images of the generated plots
- A README.md with directions to run the program

6. Conclusions and Future Work

The main lesson learned from the project is the performance benefits of NVIDIA CUDA for training convolutional neural networks. The project showed that the NVIDIA CUDA version of the code was able to train the network to a higher accuracy and in a shorter amount of time on the GPU than the AMD HIP version.

One highlight of the project was the successful implementation of a convolutional neural network from scratch, using both AMD HIP and NVIDIA CUDA frameworks. This demonstrated the flexibility and adaptability of the code, as well as the ability to compare the performance of the two frameworks. I also found that HIP can be ran on NVIDIA platforms [4].

One of the other lessons learned was that programming in HIP is not too difficult compared to CUDA, and that if you already know one of the languages, there is a very small learning curve. If, in the future, AMD GPUs become better than NVIDIA GPUs for any given kind of workload, ones transition would not be met very harshly in the new environment.

However, there is still more work to be done in the future to fully understand and compare the performance of AMD HIP and NVIDIA CUDA for training neural networks. For example, further experimentation with more complex neural networks, with more layers and more parameters, could provide more insight into the strengths and weaknesses of the two frameworks. Additionally, comparing the performance of the two frameworks on different types of datasets and different types of neural network architectures could provide a more complete picture of their relative performance.

Alongside this, in this project, I decided to implement everything from scratch, which was very timely, but did allow me to learn HIP using the functions I know from CUDA. Perhaps a future project could utilize preexisting libraries such as cuDNN and hipDNN to compare the optimized neural network processes.

Overall, the remaining work is not particularly difficult, but it will require time and effort to conduct additional experiments and analyze the results. With continued experimentation and analysis, it should be possible to gain a better understanding of the performance of AMD HIP and NVIDIA CUDA for training neural networks.

This project has leveraged ME 759 material as I implemented all of the CNN from scratch, writing many kernel functions utilizing parallelism, the power of GPU computing, and using many of the gained knowledge from the homeworks across the semester relating to the course material, especially the CUDA porition of the course.

References

[1] MNIST loader by Nuri Park - https://github.com/projectgalateia/mnist
[2] NVIDIA CUDA Documentation:
https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html
[3] AMD HIP Documentation:
https://rocmdocs.amd.com/en/latest/Programming_Guides/HIP-GUIDE.html
[4] AMD HIP FAQ: https://rocmdocs.amd.com/en/latest/Programming_Guides/HIP-FAQ.html