Final Year Project Report

Tianhao Liu (20205784)

April 20, 2024

Contents

List of Figures

1 Introduction

In the competitive landscape of machine learning and high-performance computing, the efficiency of model training is a paramount concern that drives the adoption of innovative computational strategies. This paper presents a pioneering approach by introducing the first machine learning model that is trained using a hybrid infrastructure employing both Central Processing Units (CPUs) and Graphics Processing Units (GPUs). This novel training method capitalizes on the combined strengths of these processing units to optimize computation times and resource utilization. The integration of CPUs and GPUs for parallel computing has historically presented notable challenges, primarily due to the complexities of effective workload assignment and processor affinity. Traditional methods often failed to exploit the full potential of these heterogeneous systems due to several limitations inherent in existing tools such as improper thread placement, and sub-optimal utilization of the differing architectures of CPUs and GPUs. To overcome these hurdles, our approach incorporates OpenH, an advanced hybrid programming model that leverages an integrated use of Pthreads, OpenMP, and OpenACC to facilitate the seamless integration of CPUs and GPUs[1]. The results from the deployment of this training methodology demonstrate the robust capability of the OpenH framework in managing and executing a balanced load across the hybrid system, maximizing the utilization efficiencies and markedly reducing the computational overhead associated with machine learning model training. In the following sections, we will provide an overview of the background, design, implementation, and results of this project, along with a discussion of potential future work and concluding remarks.

2 Background

The evolution of machine learning models and their increasing complexity has led to a significant surge in computational demands. Traditional single-processor systems often fall short in effectively managing the computational burdens associated with advanced algorithms, particularly in the realm of data-intensive tasks such as training deep neural networks. As such, the exploration of more capable and efficient systems has become a crucial research endeavor in this field.

2.1 Multi-Processor System

Historically, the transition from single-processor to multi-processor systems marked a significant leap in computational capabilities. Multi-processor systems, utilizing either multiple CPUs or a combination of CPUs and GPUs, offer parallel processing capabilities that significantly accelerate computational tasks. CPUs, with their general-purpose architectures, are adept at handling complex instructions, while GPUs, originally designed for image processing, excel at executing simpler, concurrent tasks over large blocks of data. This architectural difference inherently positions GPUs as favorable for the parallel execution of numerous operations, which is a common requirement in machine learning tasks. The advantages and underlying principles of multi-processor systems have been elaborated in key studies such as those by Flynn (1995) and Hennessy et al. (2011), providing foundational knowledge in processor architectures[2] [3]. Despite the advantages, exploiting the full potential of these heterogeneous systems has been a challenging feat due to several technical and architectural hurdles. The parallelism inherent in machine learning tasks requires intricate coordination and efficient data handling capabilities between the different types of processors. Additionally, inherent differences in memory architecture and data processing paradigms between CPUs and GPUs often lead to bottlenecks, which can negate the benefits of parallel processing.

2.2 Hybrid Computing Challenges

Hybrid systems that synergize CPU and GPU capabilities aim to combine the strengths of both processor types to enhance performance and energy efficiency. However, the practical realization of efficient hybrid computing encompasses addressing significant challenges, particularly in terms of synchronization, memory management, and resource allocation. Designing an infrastructure capable of effectively distributing workloads across both CPUs and GPUs—while concurrently minimizing data transfer overheads and optimizing resource utilization—poses a complex engineering task. This complexity demands a sophisticated and adaptive programming model tailored for hybrid environments. Existing programming models such as CUDA and OpenCL have been instrumental in enabling parallel processing on GPUs, but they often lack the flexibility and ease of use required for hybrid systems. In pursuit of maximizing the potential of hybrid systems, an integrated use of tools that are inherently designed for CPU parallelization, such as OpenMP, along with those tailored for GPU acceleration like OpenACC, is a promising strategy. These tools cater specifically to the distinct architectural needs of CPUs and GPUs respectively. However, the overarching challenge lies in the coherent integration of these diverse tools, as they were not originally designed to function in tandem. To bridge this gap, there emerges a critical need for a novel programming model that can facilitate the effective integration of CPU and GPU processing capabilities within a unified hybrid system framework. This is where OpenH steps in—an innovative model designed specifically to enable the efficient orchestration of hybrid computational resources.

2.3 OpenH

OpenH provides a sophisticated toolkit designed to address the nuances of thread management, synchronization, and effective workload distribution. By mitigating the common obstacles associated with hybrid systems, such as inefficient thread placement and suboptimal processor utilization, OpenH enhances the ability of machine learning practitioners to harness the combined power of CPUs and GPUs. With OpenH, developers can seamlessly integrate Pthreads, OpenMP, and OpenACC to create a unified programming model that optimizes the execution of machine learning tasks across heterogeneous systems. This integrated approach ensures that the computational load between CPUs and GPUs can be balanced effectively by users, thereby maximizing resource utilization and minimizing computational overheads.

2.4 Ensemble Learning Method

Ensemble learning is a robust machine learning paradigm where multiple models, often referred to as "weak learners," are trained to solve the same problem and then combined to improve the robustness and accuracy of predictions. This technique capitalizes on the strength of numerous learners to achieve better performance than any single model could accomplish alone. The ensemble methods are generally classified into two main categories: bagging and boosting.

- **Bagging (Bootstrap Aggregating):** Bagging involves training multiple models in parallel, each on a slightly different data sample—typically a random subset with replacement. The individual models are often less correlated with each other, allowing the ensemble to average out their biases and reduce variance. Decision trees are commonly used in bagging setups, with Random Forest being one of the most popular bagging ensemble methods.

- **Boosting**: Boosting involves sequentially training a series of models, where each new model attempts to correct errors made by the previous ones. The models build upon each other, learning from the mistakes, thus improving the overall performance. Gradient boosting and AdaBoost are prominent examples of this approach.

2.5 Random Forest Model

The Random Forest algorithm, introduced by Breiman in 2001, is a powerful ensemble learning method that involves the aggregation of multiple decision trees to improve prediction accuracy and control over-fitting. The model operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random Forests perform well with large data sets and are inherently suitable for parallel processing due to the independence of each decision tree during both training and inference phases. This inherent characteristic makes them ideal candidates for hybrid CPU-GPU computing environments.

3 Project Design

3.1 Overview

The project is designed to implement a hybrid training approach for a Random Forest model using OpenH, integrating both CPU and GPU resources. The model's performance and resource utilization are then compared against implementations solely using OpenMP (CPU) and OpenACC (GPU).

3.2 Sytem Architecture

OpenH is used as the hybrid programming model to orchestrate the CPU and GPU resources effectively. The system architecture is designed to distribute the workload between CPUs and GPUs using thread management and synchronization mechanisms provided by OpenH.

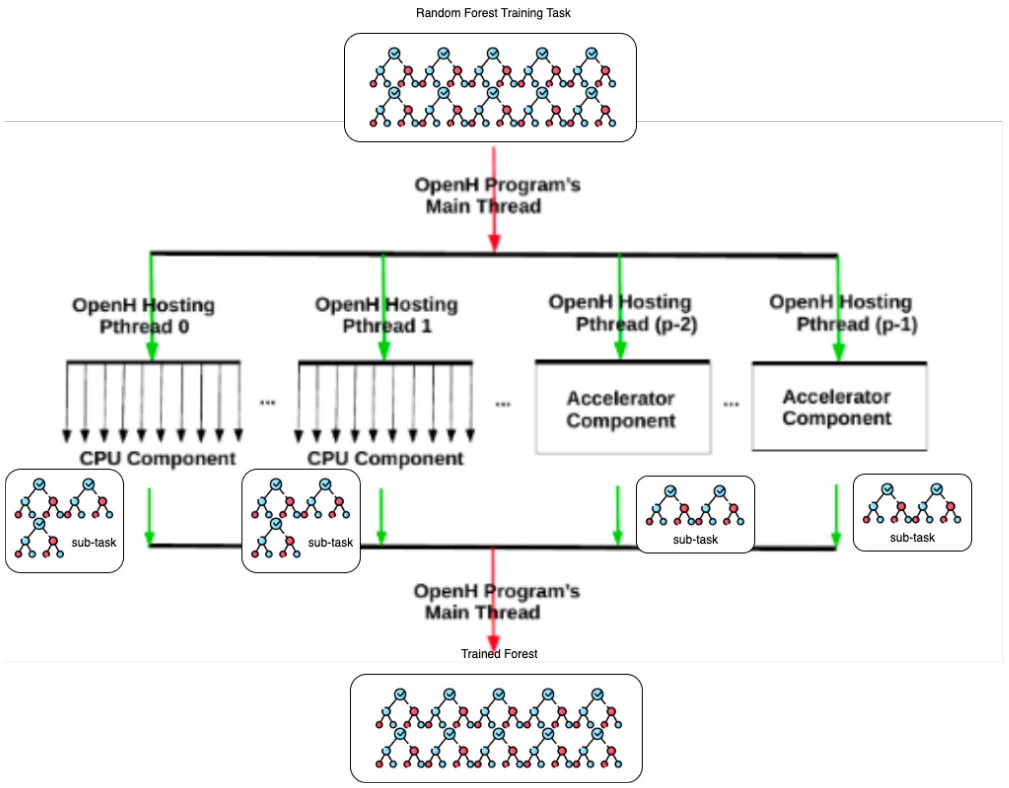

Figure 1: System Architecture

As illustrated in the figure 1, the system architecture comprises three main steps:

- **Initiation:** The system initializes the training process by creating a thread and then use OpenH to detect device capabilities and allocate resources accordingly.

- **Training:** The training phase involves transferring data to different devices, setting up the thread affinity, and executing the Random Forest training algorithm in each accelerator.
- **Gather:** The results from different devices in training phase are gathered and combined to produce the final model output.

3.3 Metrics

The performance of the hybrid training approach is evaluated by the execution time only since the primary goal is to optimize the training speed.

4 Implementation

5 Results

6 Future Work

7 Conclusion

References

[1] Farrelly, Simon, Ravi Reddy Manumachu, and Alexey Lastovetsky. "OpenH: A Novel Programming Model and API for Developing Portable Parallel Programs on Heterogeneous Hybrid Servers." *IEEE Access* 12 (2024): 23666–94.

[2] Flynn, Michael J. *Computer Architecture: Pipelined and Parallel Processor Design*. 1st ed. USA: Jones and Bartlett Publishers, Inc., 1995. ISBN: 0867202041.

[3] Patterson, David A., and John L. Hennessy. *Computer Architecture: A Quantitative Approach*. 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1990. ISBN: 1558800698.