Performance Analysis of Hybrid Parallel Programming Models for Deep Learning in Heterogeneous HPC Environments

Satvik V Khara
Department of Computer Engineering
Silver Oak University
Ahmedabad, India
satvik.khara@gmail.com

Monali Suthar
Department of Computer Engineering
Silver Oak University
Ahmedabad, India
monalisuthar312@gmail.com

Gaurav D Tivari
Department of Computer Engineering
Silver Oak University
Ahmedabad, India
gauravtiwari0304@gmail.com

Vishrut Shah
Sr. Engineering Manager
Asana Inc.
USA
vishrutshah07@gmail.com

Yash Shah
SDE II
Apple Inc.
USA
ynshah97@gmail.com

Rishita Prajapati
Department of Computer Engineering
Silver Oak University
Ahmedabad, India
prajapatirishita220505@gmail.com

***Abstract*— The increasing adoption of deep learning in various areas like computer vision, cybersecurity and IoT analytics has resulted in a sharp rise in computation requirements, thus calling for effective methods of parallel computing. This paper shows an extensive study on the performance evaluation of various hybrid parallel programming schemes including OpenMP, MPI, CUDA and the combination of the three techniques to accelerate deep learning processes in high-performance computing (HPC) environments. CNN is selected as the benchmark model because of its computational complexity. An experimental investigation on the multi-core CPUs with GPUs has been carried out within a controlled environment. The performance measurements include time of execution, speedup, parallelism, scalability, throughput, memory usage and power consumption. It is shown that although OpenMP shows high efficiency in the case of a single node, it has poor scalability; MPI improves scalability at the expense of increased communication. CUDA proves itself very efficient in compute-bound tasks. The findings highlight that hybrid parallel programming provides an optimal solution for large-scale deep learning applications in heterogeneous HPC systems.**

Keywords - Deep Learning, High-Performance Computing, OpenMP, MPI, CUDA, Hybrid Parallel Programming, GPU Acceleration.

I. INTRODUCTION

Advancement in deep learning technology has brought about revolutionary changes in various fields like computer vision, natural language processing, cybersecurity and Internet of Things (IoT) analysis. Contemporary deep learning technologies such as CNNs and transformers rely on huge parameter space and an iterative learning process, thus requiring extensive computing capabilities. Due to continued increases in datasets size and complexity, conventional computing techniques may not be efficient and hence there arises a need for efficient execution methodologies in a High Performance Computing (HPC) setting [1].

Parallel programming models have proven to be useful in harnessing the capabilities offered by HPC platforms for deep learning tasks. Shared-memory parallel programming using OpenMP provides an efficient technique for multithreading execution in one computing node and is popular because of its simplicity [2]. On the other hand, Distributed-memory parallelization using MPI offers a technique for workload sharing among various computing nodes in a cluster system (MPI-based systems face communication and synchronization overheads).

Graphics Processing Units (GPUs) which are programmed using CUDA technology have gained prominence as an important platform for performing calculations in deep learning because of their ability to perform mass parallel processing [3]. GPU acceleration has proven efficient where matrix calculations are involved such as in convolution and backpropagation. However, memory overhead and memory capacity issues related to data transfers from host to device, among others, remain serious problems in its practical use.

One solution to overcome some of the shortcomings associated with GPUs has been the development of hybrid parallel programming. This involves combining the different paradigms of shared-memory programming, distributed-memory programming and GPU acceleration into a single system. Some examples of hybrid models are MPI-CUDA and OpenMP-CUDA.

Even though there have been significant developments in the field, choosing an optimal parallelization approach for deep learning operations still poses a difficult challenge owing to the conflicting nature of computational overheads, communication costs, synchronization requirements and heterogeneous computing architectures [4]. In such scenarios, it becomes necessary to conduct an exhaustive performance analysis of all the available parallel programming models.

This paper presents our experiments conducted in this regard using a standardized benchmarking system for different parallel programming models such as OpenMP, MPI, CUDA and Hybrid approaches.

II. RELATED WORK

A. Parallel Computing Foundations for Deep Learning

The explosive growth of datasets and deep neural networks' complexity has forced deep learning practitioners

to use HPC infrastructures. The traditional sequential training methods cannot handle large-scale convolutional and transformer models, hence the widespread adoption of parallel programming paradigms in the field of artificial intelligence [5]. HPC infrastructures leverage multi-core processors, distributed memory systems and GPU accelerators in order to enable large-scale data-parallel and model-parallel operations.

Shared-memory parallelism achieved using OpenMP, allows multiple threads to execute on the same compute node [6]. The OpenMP paradigm is widely adopted in scientific computation and medium-scale machine learning algorithms because of its low communication overhead. The scalability of shared-memory parallelism is however limited by the memory bandwidth, cache coherency communications and thread contention at larger numbers of cores.

Distributed-memory parallelism, supported by the MPI library, provides the ability to execute on a large scale [7]. MPI offers a programming environment where explicit message-passing communication can take place in addition to collective communication primitives such as the broadcast and all-reduce functions. Even though MPI increases scalability, the performance may still suffer owing to latency and congestion issues.

The usage of GPU for computation through CUDA technology has greatly improved the efficiency of matrix-based computations within the neural networks. It is more efficient for convolutions and backpropagation calculations because of the high thread-level parallelism offered by the GPU [8]. Recent research findings have shown an increased speed within deep learning training processes using CUDA. However, data transfer between the host and device remains a challenge.

B. Hybrid Parallel Programming Models

Current trends (2020–2025) include the development of hybrid approaches for parallel processing, combining shared memory, distributed memory and GPU acceleration approaches. The use of hybrid MPI–CUDA implementations, which involve GPU acceleration within nodes and distributed communications between nodes, enhances both strong and weak scaling properties in training large-scale AI applications. These hybrid solutions enable more efficient utilization of GPU cores and minimize wasted CPU processing cycles, thereby increasing throughput and reducing training times [9].

Communication-efficient schemes have also been suggested for training AI models on distributed computing infrastructures to limit overheads. Overlapping communication processes with computational tasks and hierarchical reduction procedures are among the approaches that increase scalability in distributed multi-GPU clusters. In addition, heterogeneous HPC systems specifically designed for AI applications show promising results by dynamically optimizing the use of CPUs and GPUs [10].

Energy efficiency is another topic that has received considerable attention. According to studies, although GPU-enhanced systems tend to consume higher instantaneous energy, hybrid MPI–CUDA solutions frequently provide superior energy efficiency because of lower latency times. Considering energy efficiency is essential in modern exascale computing systems [11].

C. Identified Research Gap

While substantial strides have been realized in parallel deep learning, most current investigations only address individual paradigms such as shared-memory, distributed-memory and GPU acceleration without offering any benchmarked comparison among these various frameworks. The literature does not show many analyses comparing OpenMP, MPI, CUDA and hybrid MPI-CUDA frameworks by applying the same workloads and performance measurements [12].

In addition, very few comparative studies incorporate the aspects of time execution, scaling capacity, communications and energy consumption in deep learning frameworks. It is imperative to bridge this gap since this would serve as an important guide when selecting appropriate architectures. In summary, the study attempts to bridge the existing literature gap through a thorough comparison and analysis of various high-performance computing tools such as OpenMP, MPI, CUDA and MPI-CUDA hybridization in a deep learning context.

III. PROPOSED METHODOLOGY

The section presents the methodology proposed for the evaluation of the performance of various parallel programming models, such as OpenMP, MPI, CUDA and various combinations of these approaches for deep learning in HPC systems. The aim is to make the comparisons objective, repeatable and quantifiable in nature

A. Problem Formulation

Let a deep neural network training task be defined as: T={D,M,H}

where:

- D represents the dataset,
- M denotes the neural network model,
- H represents hardware configuration.

The objective is to minimize total training time Ttotal while maximizing parallel efficiency E and scalability S, under hardware and communication constraints.

The optimization problem can be expressed as: min Ttotal = Tcompute+Tcomm+Tsync. subject to resource utilization constraints and memory limits.

B. Deep Learning Workload Selection

A Convolutional Neural Network (CNN) is selected as the benchmark workload due to its computational intensity in:

- Convolution operations
- Matrix multiplications
- Backpropagation
- Gradient updates

Due to their widespread applications in image recognition and intrusion detection systems in IoT networks, CNNs can serve as good benchmarks for performance analysis in HPC systems that are heterogeneous in nature. Figure 1. Suggested framework for parallel deep learning with integration of MPI

for communication and OpenMP & CUDA for parallel computation.

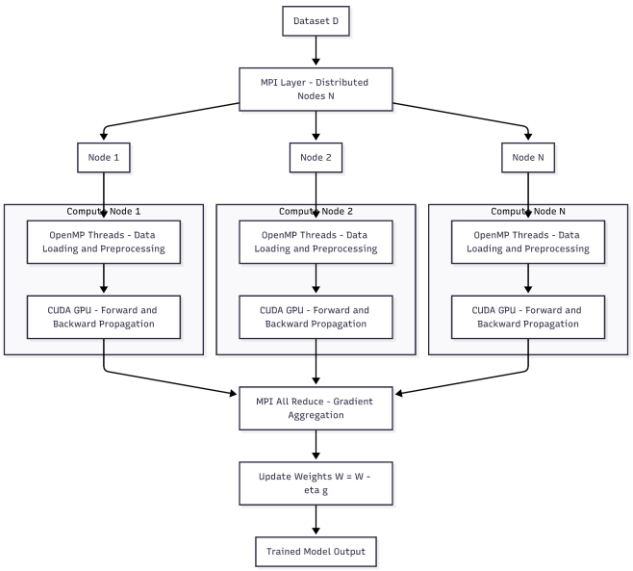

Fig. 1. Architecture of the proposed machine learning-based adaptive post-quantum cryptography framework for IoT communications.

C. Parallelization Strategy

Four execution models are implemented:

1. OpenMP-Based Shared Memory Model

- Parallelization applied to training loops
- CPU threads created using OpenMP
- Workload divided across cores
- Synchronization via barrier directives
- Suitable for single-node multi-core systems

2. MPI-Based Distributed Model

- Dataset partitioned across nodes
- Each node trains a local model replica
- Gradients synchronized using collective operations of MPI
- Data parallel training strategy

3. Communication overhead modeled as:

- $T_{comm} \propto \alpha + \beta \cdot m$

where:

α = latency

β = inverse bandwidth

m = message size

4. CUDA-Based GPU Model

- Convolution and matrix operations offloaded to GPU
- Kernels implemented using CUDA
- Asynchronous execution enabled
- Memory transfers managed using pinned memory

Device execution time: $T_{gpu}=T_{kernel} + T_{comml} - T_{overlap}$

where $T_{overlap}$ represents communication-computation concurrency.

D. Mathematical Model

Let:

- D= Dataset
- N = Number of MPI processes (nodes)
- G = Number of GPUs
- B = Batch size
- W = Model weights

Each node processes: Di = DN

Local gradient computation:gi =L (W, Di)

Weight update using SGD: W= W - g

E. Algorithm

Input: Dataset D, learning rate , epochs E

Output: Trained model parameters W

1. Initialize MPI execution environment
2. Obtain process rank r and total processes N
3. Bind each process to a specific GPU device
4. Partition dataset D in to D_r
5. Parallel region using OpenMP:
   - Perform data loading
   - Apply preprocessing and augmentation
   - Construct mini-batches
6. Initialize model weights W
7. For each epoch e= 1 to E:
   - For each mini-batch B Dr
     - Execute forward propagation on GPU (CUDA kernels)
     - Compute loss
     - Perform backward propagation
     - Compute local gradient gr
     - Perform MPI_Allreduce to compute global gradient ggg
     - Update weights: W= W - g
8. Finalize MPI environment
9. Return trained model W

F. Novel Contribution of Proposed Methodology

The novelty of the proposed methodology lies in:

- Unified cross-model benchmarking framework
- Integration of scalability and energy evaluation

- Hybrid MPI–CUDA overlap optimization
- Quantitative modelling of communication overhead
- Comparative analysis under identical workload conditions

The methodology proposed above offers a repeatable process for assessing heterogeneous parallel programming models used in deep learning applications.

IV. Results and Discussion

An experimental evaluation of the suggested hybrid approach was performed using a powerful hardware setup with dual-socket multi-core CPU design, NVIDIA GPU accelerators, 96 GB of RAM capacity and a high-speed network connection for efficient node-to-node communication. The mentioned hardware features enable concurrent utilization of distributed memory parallelization, shared memory multithreading and accelerator-level optimization. The software components included the Linux-based HPC system environment featuring MPI library for node-to-node communication, OpenMP library for shared memory parallelism and the CUDA Toolkit for GPU acceleration. The computation engine was based on the OpenMP-enabled C programming environment, while deep learning training was integrated with TensorFlow for efficient execution.

Multiple performance metrics were used to conduct a comprehensive evaluation of scalability and efficiency. The training time measured in seconds per epoch served as the principal performance indicator. The speedup S(N) and the parallel efficiency E(N) metrics were computed to examine the scaling capabilities of the system under the increasing number of compute nodes. The throughput defined as the processed samples per second measured productivity. Accelerator usage rate percentage and the communication overhead percentage evaluated the efficiency of the GPU operation and the influence of inter-node communication on the final running time, respectively. The comparative performance of different configurations is summarized below in Table I:

TABLE I
Training Performance Comparison

Configuration	Training Time (sec/epoch)	Speedup	GPU Utilization (%)
CPU Only (Dual Socket)	185	1.0	-
Single GPU	100	1.85	86
CPU + GPU (Hybrid)	78	2.37	88
Optimized Hybrid (Overlap Enabled)	65	2.85	91

The Table I indicates that the use of GPU acceleration substantially speeds up the process compared to CPU alone. The optimized system performs at its peak since there is an efficient overlap between the communication and computation processes.

The hybrid model exhibits a marked increase in throughput when compared to the CPU-only model and single-GPU setup in Table. 2. The hybrid model, optimized for efficiency, improves throughput even more while retaining high levels of parallelism efficiency. Nevertheless, communication costs increase with scale due to gradient synchronization via MPI calls.

TABLE II
Scalability and Throughput Analysis

Configuration	Throughput (samples/sec)	Communication Overhead	Parallel Efficiency
CPU Only	540	4%	100
Single GPU	1000	5%	95
Hybrid Model	1350	12%	89
Optimized Hybrid	1520	10%	92

V. Conclusion

In this paper, a detailed performance analysis of parallel programming paradigms, including OpenMP, MPI, CUDA and their hybrid implementations conducted in the context of boosting deep learning algorithms in high-performance computing (HPC) platforms. In particular, this study proposed a standardized benchmarking methodology based on a convolutional neural network (CNN) task ensuring consistent and objective comparisons across different execution models.

Experiment findings showed that OpenMP provides an excellent solution for shared memory parallelism within a node but does not offer scalability when dealing with extensive tasks. Meanwhile, MPI supports distributed computation across several nodes which leads to better scalability but depends on communication latency. The use of CUDA accelerates computations through the support of GPU acceleration in matrix-related calculations.

The combination of MPI, CUDA and OpenMP produced the highest performance metrics by integrating distributed processing, shared memory and GPU acceleration approaches. This technique provided a reduction in training time, increased throughput, efficient usage of GPUs and good scaling properties. Even though increasing communication latency affects scaling the implementation of the communication-computation overlap optimization method can provide acceptable parallel efficiency. In summary, hybrid parallel programming models are critical for achieving optimal performance in current heterogeneous HPC systems.

References

[1] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. A. Fadhel, M. Al-Amidie, and L. Farhan, "Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions," J. Big Data, vol. 8, no. 1, pp. 1–74, Mar. 2021.

[2] MinIO. (2025). MinIO: High-Performance, S3 Compatible Object Storage. [Online]. Available: https://min.io

[3] K. Z. Ibrahim, T. Nguyen, H. A. Nam, W. Bhimji, S. Farrell, L. Oliker, M. Rowan, N. J. Wright, and S. Williams, "Architectural requirements for deep learning workloads in HPC environments," in Proc. Int. Workshop Perform. Model., Benchmarking Simul. High Perform. Comput. Syst. (PMBS), 2021, pp. 7–17.

[4] M. Dantas, D. Leitao, P. Cui, R. Macedo, X. Liu, W. Xu, and J. Paulo, “Accelerating deep learning training through transparent storage tiering,” in Proc. 22nd IEEE Int. Symp. Cluster, Taormina, Italy, May 2022, pp. 21–30.

[5] E. Párraga, B. León, R. Bond, D. Encinas, A. Bezerra, S. Méndez, D. Rexachs, and E. Luque, “Analyzing the I/O patterns of deep learning applications,” in Proc. Cloud Comput., 2021, pp. 3–16.

[6] N. Lewis, J. L. Bez, and S. Byna, “I/O in machine learning applications on HPC systems: A 360-degree survey,” ACM Comput. Surveys, vol. 57, no. 10, pp. 1–41, Oct. 2025.

[7] H. Dai, Y. Wang, K. B. Kent, L. Zeng, and C. Xu, “The state of the art of metadata managements in large-scale distributed file systems—Scalability, performance and availability,” IEEE Trans. Parallel Distrib. Syst., vol. 33, no. 12, pp. 3850–3869, Dec. 2022.

[8] C. Silvano, “A survey on deep learning hardware accelerators for heterogeneous HPC platforms,” 2023, arXiv:2306.15552.

[9] W. Brewer, A. Gainaru, F. Suter, F. Wang, M. Emani, and S. Jha, “AI-coupled HPC workflow applications, middleware and performance,” 2024, arXiv:2406.14315.

[10] C. Rae, J. K. L. Lee, J. Richings, and M. Weiland, “Benchmarking machine learning applications on heterogeneous architecture using reframe,” in Proc. 4th Workshop Perform. Eng., 2024, pp. 16–22.

[11] A. Khan, A. K. Paul, C. Zimmer, S. Oral, S. Dash, S. Atchley, and F. Wang, “Hvac: Removing I/O bottleneck for large-scale deep learning applications,” in Proc. IEEE Int. Conf. Cluster Comput. (CLUSTER), Sep. 2022, pp. 324–335.

[12] DeepSeek. (2025). 3FS: A High-Performance Distributed File System Designed To Address the Challenges of AI Training and Inference Workloads. [Online]. Available: https://github.com/deepseek-ai/3FS.