Optimizing Indic Translation System Inference Using Data and Model Parallelism

Pranav Janjani
COE-CNDS
VJTI
Mumbai, India
janjanipranav@gmail.com

Jenil Shah
COE-CNDS
VJTI
Mumbai, India
jenils2003@gmail.com

Faruk Kazi
COE-CNDS
VJTI
Mumbai, India
fskazi@el.vjti.ac.in

***Abstract*—This study investigates the effectiveness of data parallelism in accelerating the inference process for a machine translation system. We demonstrate that data parallelism offers a significant advantage over traditional CPU-based processing for models with a relatively small memory footprint (1 GB in this case). Our findings reveal a processing time reduction of 99.5% when employing data parallelism. Processing 176,000 samples on a dual V100 CPU system with 512 GB of RAM was estimated to take approximately 2,000 hours (83 days). Conversely, utilizing data parallelism on two A6000 GPUs with 48 GB of memory each yielded a processing time of only 10 hours for the same workload. Notably, processing on a system with limited CPU RAM (16 GB) resulted in repetitive kernel crashes, highlighting the limitations of CPU-based inference for computationally intensive tasks. These results strongly support the efficacy of data parallelism in accelerating translation system inference, particularly for models with manageable memory requirements**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

Machine translation systems bridge language gaps and provide the ability for meaningful communication in diverse linguistic environments [1]. They function as automated processes that convert text from one language to another using complex algorithms and advanced neural network architectures [2], [3]. Their efficiency is very dependent on the power of the used computational hardware, mainly Central Processing Units (CPUs) and Graphics Processing Units (GPUs). The CPUs are designed for a limited number of high-performance cores that handle tasks through threads and processes, executing the instructions sequentially or in parallel. This architecture excels at handling complex and varied operations, offering versatility across numerous applications.

In comparison, a GPUs feature thousands of much smaller cores, designed for parallel processing. In contrast to CPUs, which are optimized to run a few powerful cores that execute many different complex tasks, GPUs specialize in executing many simpler tasks simultaneously. The architecture design in a GPU makes it highly optimized for parallel operations over large amounts of data; therefore, it's much more suitable for running the huge number of parallelizable operations involved in neural network training and inference. This makes the machine translation system, therefore, very accurate and speedy, more so when dealing with vast amounts of data and complex computations that characterize deep learning models. The capability of parallel processing in a GPU massively accelerates the processes of training and inference, thus making them quite effective while dealing with complicated and data-intensive tasks compared to a CPU.

Data parallelism and model parallelism strengthen the machine translation system's capability even further. In Data Parallelism, data is split into chunks over many processing units, allowing them to compute the same model concurrently on different subsets of data. This overlaps the processing capability of many units during training and makes the inference faster [4]. In contrast, model parallelism decomposes a model into many units and lets each unit handle some of the model's computations. Model parallelism is especially helpful in the handling of very large models that will not even fit into a single processing unit's memory, thus allowing more complex and bigger models to be handled.

Understanding the workings of translation systems reveals their computational demands. These systems typically use encoder-decoder architectures where the encoder processes the input text into a representation from which the decoder is likely to translate into output text. In particular, fine-tuning of these models on Indic languages requires adjusting the model to capture unique linguistic features. The morphologically rich and syntactically diverse Indic languages require proper tuning of model parameters based on exposure to language-specific data so that the translated output is accurate and relevant. This fine-tuning of model parameters of the Machine Translation model brings them closer to optimal translation accuracy and effectiveness for specific language pairs. Fine-tuning processes in NLP adapt or adjust pre-trained language models to target tasks or languages. It means the process whereby fine-tuning a model's performance can be made to understand and generate relevant special domains of text or languages. The major challenge in fine-tuning LLMs in Indic languages like Hindi, Marathi, Bengali, or Tamil is scarcity of domain-specific training data [5], [6]. While in the English language, there exist a lot of such instruction sets, their availability over these Indic languages remains critically limited.

II. Related Work

Parallelism techniques have made a huge impact in different areas of machine learning, enabling improvements that would otherwise have been computationally infeasible. In image recognition, Krizhevsky et al. showed just how powerful data parallelism can be in their work on ImageNet classification using Deep Convolutional Neural Networks [7]. This distribution of data across multiple GPUs gives large increases in speed for the training phase, keeping them from having massive datasets to train with for greater accuracy. Jia et al. used data parallelism in developing their Deep learning framework - Caffe, which offers a clean and extendable framework for multimedia scientists and practitioners powered by state-of-the-art deep learning algorithms and a library of reference models [8]. Balaji et al. expounded on how Message Passing Interface is being implemented on a million processors to help in distributing such complex calculations across hundreds of processors [9].

Data parallelism in traditional methods, involved splitting a single training iteration among various cores, suffer from diminishing efficiency with greater numbers of cores. Dryden et al. quantized gradient updates to reduce communication bandwidth requirements and compared this with traditional MPI allreduce implementations, showing that adaptive quantization is able to obtain near-linear speedup in data-parallel training without accuracy loss [10]. Zhang et al. further introduced a new cyclic data parallelism paradigm, reducing memory consumption and balancing gradient communications by switching from concurrent to sequential micro-batch execution, and demonstrated the approach on CIFAR-10 and ImageNet datasets [11].

Model parallelism enables complex model training by partitioning it across multiple GPUs. These limitations were addressed by Chen et al., who showed that pipelined model parallelism not only solves inter-GPU communication issues but can be much more efficient [12]. The approach of Zheng et al. to hierarchical model parallelism emphasizes the automated generation of parallel execution plans that are as effective as possible for different architectures [13]. Moreover, Narayanan et al. proposed an approach that integrated interbatch pipelining with intrabatch parallelism, allowing increased training throughput by improving computation-communication overlap [14].

Indic languages, though highly used, have very few high-quality annotated data which can be used for fine-tuning purposes. Due to the fact that a great majority of NLP research and development activities are focused on high-resource languages like English, the same gap exists for languages like Hindi, Marathi, Bengali, Tamil, etc. Yet, despite all the challenges, in recent years a lot of effort has been spent by researchers in developing Indic language models for translation tasks. Shah et al. presented one neural machine translation system for Hindi and Gujarati, an encoder-decoder with an attention mechanism. This model shows rather promising results across most evaluation metrics, outperforming their baseline Google Translate by a margin of 6 BLEU points on EN-GU translation [15].

Feature	Google Translate	IndicTransTokenizer (IIT Madras, Ai4Bharat)
Focus	General-purpose translation service	Tailored for Indian languages, including Marathi
Marathi Support	Limited vocabulary, may not handle technical terms	Rich vocabulary for Marathi, including technical terms
Accuracy	More inaccurate for Indic Languages	More accurate for technical translations in Marathi
Customization	No customization options available	Can be fine-tuned for specific domains or terminology
Open Source	No	Yes (refer to indic-nlp github repository)
Control	Limited control over translation process	More control over the translation process
Privacy	Data privacy concerns, as data is sent to Google servers	Can be deployed on-premise for privacy-sensitive data

III. Employed Translation System

A. Difference between Translation Systems

We compare the strengths and weaknesses of Google Translate about Indian translation technologies, specifically Indic NLP's Indic Transliterator. Both services are machine translation (MT) tools that automatically translate text from one language to another. However, the focus differs between the two. Google Translate is a general-purpose service, encompassing a broad range of languages. Conversely, Indic Transliterator is specifically designed for Indian languages, including Marathi.

1) General Translation vs. Focus on Indian Languages: General-purpose MT, like Google Translate, employs a statistical or rule-based approach to translate text. A large corpus of text and its translations is analyzed to identify patterns between languages. This approach can be successful for translating widely used languages with vast amounts of training data. However, it may struggle with less common languages or nuanced text, such as technical terminology. Indic Transliterator, on the other hand, is designed to address this shortcoming. By specifically tailoring the service to Indian languages, it can incorporate domain-specific vocabulary and terminology, resulting in more accurate translations for technical fields. This tailored approach is known as domain adaptation and is a growing area of research in MT.
2) Accuracy and Customization: Google Translate may have lower accuracy for Indian languages, particularly for technical translations in Marathi. This is likely due to the aforementioned limitations of general-purpose MT. Indic Transliterator, being designed for Indian languages, can potentially achieve higher accuracy in these areas. Furthermore, Indic Transliterator offers customization options, allowing users to fine-tune the translation process for specific domains or terminology. This customization, known as domain adaptation, can further enhance the accuracy of translations. Google Translate, on the other hand, does not offer this level of customization.
3) Open-Source vs. Proprietary and Privacy: Indic Transliterator is open-source software, whereas Google Translate is proprietary. Open-source software offers transparency and allows for community modification and improvement. Google Translate, being proprietary, does not offer this benefit. There is a potential privacy concern with Google Translate. Because it is a cloud-based service, user data is sent to Google servers for translation.

Name	ai4bharat/indictrans2-en-indic-dist-200M	ai4bharat/indictrans2-en-indic-dist-1B
Context Length	1280	1280
Leavel of Beam Search	5	5
Max Batch Size	100	100
Avg Inference Time/Batch	2 mins	7 mins

This may be a concern for users handling sensitive information. Indic Transliterator can be deployed on-premise, meaning it can be run on a private server without sending data to an external source. This can be a significant advantage for users working with confidential data.

Considering the trade-off between general-purpose MT and a service specifically designed for Indian languages. Google Translate offers a wider range of languages but may lack accuracy for technical translations in Indian languages. Indic Transliterator offers higher accuracy for Indian languages, customization options, and on-premise deployment for privacy concerns and is utilized for this project.

B. Difference between the AI4Bharat Models for Translations

We consider two models, ai4bharat/indictrans2-en-indic-dist-200M, and ai4bharat/indictrans2-en-indic-dist-1B which are machine translation models, that perform translation between Indic languages.

The parameters paramount for comparison between the two models are:

1) Level of Beam Search: This parameter specifies the beam search width used during inference. Beam search is a heuristic search algorithm used to find good solutions in a tree search problem. In machine translation, it helps find the most likely translation sequence by considering a set of candidate translations at each step. A higher beam search width suggests a more exhaustive search but also requires more computational resources. The table shows that both models use a beam search width of 5.
2) Context Length: This parameter specifies the maximum length of the input sequence that the model can consider during translation. A longer context length can allow the model to capture more dependencies between words in the source sentence, which can potentially improve translation quality. The table shows that both models have a context length of 1280.
3) Max Batch Size: This parameter specifies the maximum number of sentences that can be processed together in a single batch during training or inference. A larger batch size can improve training efficiency by utilizing the computational resources of the hardware more effectively. However, it can also lead to instability during training. The table shows that both models have a maximum batch size of 100.
4) Size of Model: This parameter specifies the size of the model on disk. The size of the model is primarily determined by the number of parameters in the model. A larger model can potentially capture more complex relationships between words, which can improve translation quality. However, larger models also require more computational resources to train and run. The table shows that ai4bharat/indictrans2-en-indic-dist-200M has a size of 1GB, while ai4bharat/indictrans2-en-indic-dist-1B has a size of 4.4 GB.
5) No. of Parameters: This parameter specifies the number of parameters in the model. The number of parameters is a key factor that determines the model's capacity to learn complex representations. A larger number of parameters generally allows the model to capture more complex relationships between words, which can improve translation quality. However, models with a large number of parameters also require more computational resources to train and run. The table shows that ai4bharat/indictrans2-en-indic-dist-200M has 200 million parameters, while ai4bharat/indictrans2-en-indic-dist-1B has 1.8 billion parameters.
6) Avg Inference Time/Batch: This parameter specifies the average time taken by the model to process a batch of sentences. The inference time is influenced by the size of the model, the batch size, and the hardware on which the model is running. A longer inference time suggests that the model is computationally more expensive to run. The table shows that ai4bharat/indictrans2-en-indic-dist-200M takes an average of 2 minutes to process a batch, while ai4bharat/indictrans2-en-indic-dist-1B takes 7 minutes to process a batch.

The two models have different trade-offs between model size, number of parameters, and inference speed. Ai4bharat/indictrans2-en-indic-dist-200M is a smaller model with fewer parameters and a faster inference speed, while ai4bharat/indictrans2-en-indic-dist-1B is a larger model with more parameters and a slower inference speed. The choice of which model to use would depend on the specific requirements of the application. For example, if translation quality is the primary concern and computational resources are available, then ai4bharat/indictrans2-en-indic-dist-1B may be a better choice. However, if real-time translation is required, then ai4bharat/indictrans2-en-indic-dist-200M may be a better choice due to its faster inference speed.

IV. Model Parameters & Optimization

A. Batch Size Optimization:

The Transtokenizer, a crucial component of the machine translation model, operates by processing inputs in batches, rather than individual sentences. The documentation specifies a maximum batch size of 100, exceeding which can lead to system instability. To determine the optimal batch size for our specific implementation, we conducted experiments with various configurations: 4, 16, 64, and 128 sentences per batch. Considering both inference speed and translation quality, a batch size of 64 emerged as the most effective choice, achieving a desirable balance between efficiency and accuracy.

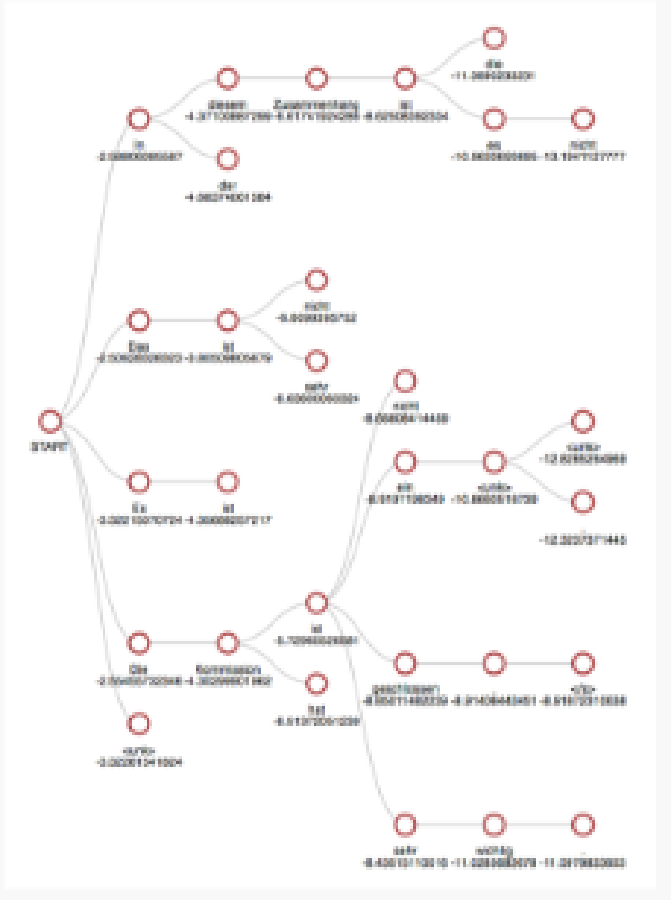

Fig. 1. Enter Caption

B. Maximizing Contextual Awareness:

A critical parameter influencing machine translation performance is the concept of maximum context length. This parameter defines the upper limit of tokens (fundamental units of text like words and punctuation) that the model can effectively analyze to produce accurate translations. Exceeding this limit can lead to a phenomenon known as context drift. In this scenario, the model loses track of earlier elements in the source text, resulting in inconsistencies and inaccuracies in the translated output. To address this challenge, we first analyzed our dataset and identified an average sentence length of 1024 tokens, with the maximum reaching 12,000 tokens. To ensure efficient processing while maintaining sufficient context, we established a maximum context length of 1280 tokens, adhering to a multiple of 256 for computational efficiency.

C. Beam Search:

Beam search serves as a pivotal decoding strategy in machine translation, aiming to identify the optimal translation for a given source sentence. It employs a left-to-right, greedy search approach, maintaining a predefined "beam width" of candidate translations at each step in the translation process.

Sr. No.	Translation Dataset	Number of Samples
1	riddle_sense	4531
2	Hello-SimpleAI/HC3	18468
3	teknium/GPTeacher-General-Instruct	88486
4	databricks/databricks-dolly-15k	13201
5	tatsu-lab/alpaca	51326
	Total	176012

This width signifies the number of most probable partial translations that are carried forward for further exploration. The model assesses the probability score for each candidate and its potential continuations, retaining only the top contenders within the designated beam width. This iterative process guarantees the exploration of diverse translation pathways while prioritizing sequences with high probability scores. While the initial configuration utilized a beam search width of 5, we observed minimal performance gains compared to a beam search width of 1. This finding suggests that a smaller beam width can achieve comparable translation quality while reducing computational overhead.

V. Data Preprocessing

Instruction fine-tuning datasets typically adhere to a well-defined structure. This structure entails three key components:

- Input Data: This element represents the core information or stimuli presented to the Large Language Model (LLM) upon which it will base its response.
- Desired Instructions: This component functions as a query or prompt directed at the LLM, guiding it towards the generation of a specific output.
- Corresponding Output: This final component captures the response produced by the LLM in response to the provided input data and desired instructions.

Encountering the Null Input Challenge: However, during our investigation, a recurring issue emerged within the datasets: the presence of null values or empty strings within the 'input' field at various indices. Processing these null or empty inputs triggered unintended consequences that negatively impacted the overall performance of the system.

Impact of Null Inputs on Translation System: The translation system, upon encountering null or empty inputs, defaulted to an unanticipated behavior. It generated a sequence of dots, often replicating the maximum context length permitted by the system. This resulted in a significant and detrimental increase in computation time. The generated output, consisting solely of dots, held no relevance to the intended translation objective and ultimately served no purpose.

Addressing the Null Input Challenge through Preprocessing: To mitigate this challenge and optimize system performance, we implemented a dedicated pre-processing step specifically designed to cleanse the data by eliminating null or empty strings within the 'input' field. During this process, meticulous attention to data point indexing is paramount. Any inaccuracies or mismatches during pre-processing have the potential to induce catastrophic consequences. A misstep could lead to a complete misalignment within the fine-tuning set,

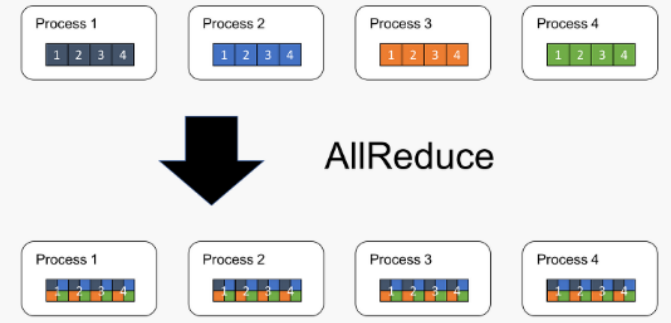

Fig. 2. Enter Caption

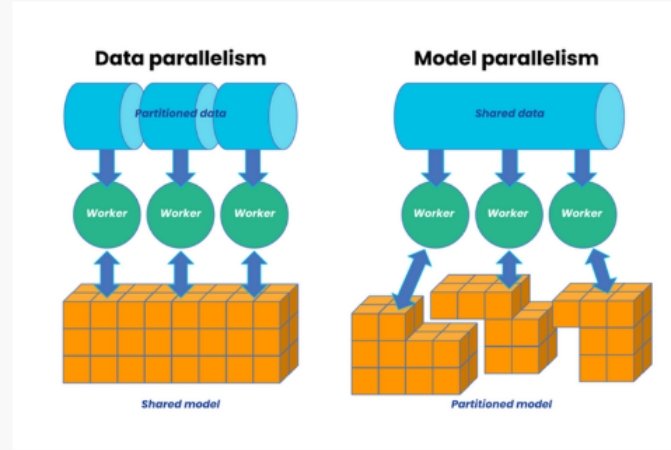

Fig. 3. Enter Caption

where instructions, outputs, and inputs become erroneously associated with one another. This misalignment would render the dataset unusable for its intended purpose.

VI. Parallelism & Algorithms Employed

In the ever-evolving realm of deep learning, achieving pinnacle model performance often necessitates harnessing the immense computational prowess of multiple machines in a coordinated fashion. Distributed Data Parallel (DDP), a cornerstone paradigm within the PyTorch deep learning framework, emerges as a powerful maestro, orchestrating a symphony of algorithms to facilitate efficient and scalable model training across numerous devices. This paper delves into the intricate workings of DDP, unveiling the interplay between its core mechanisms that empowers researchers and practitioners to unleash the true potential of their deep learning models.

Delving into the Algorithmic Symphony: A Breakdown of DDP's Core Mechanisms

The cornerstone of DDP's functionality lies in the elegant concept of data parallelism. This meticulous strategy acts as the conductor, meticulously segmenting the colossal training dataset into manageable and digestible batches. These batches are then strategically distributed across all participating devices, typically high-performance Graphics Processing Units (GPUs), in a well-rehearsed and equitable manner. Each GPU serves as a virtuoso performer, independently processing its assigned batch and meticulously calculating the corresponding gradients. These gradients, akin to the subtle nuances in a musical piece, represent the direction for optimizing the model's parameters, ultimately leading to a more refined and accurate model.

Ensuring Equitable Distribution: The Distributed Sampler as the Meticulous Conductor

To guarantee efficient data allocation during training, DDP employs a specialized tool known as the DistributedSampler. This ingeniously crafted sampler acts as a meticulous conductor, meticulously dividing the dataset into non-overlapping subsets, ensuring each device receives a unique batch during each training iteration. This meticulous division prevents redundant computations and maximizes resource utilization, akin to ensuring that each musician in the orchestra has their distinct part to play, eliminating unnecessary repetition and creating a harmonious performance.

Synchronizing the Symphony: The All-Reduce Algorithm and Automatic Differentiation

The successful convergence of a distributed training process hinges upon the seamless synchronization of gradients computed across all participating devices. Imagine an orchestra where each section refines its part in isolation; without proper coordination, a cacophony would ensue. DDP addresses this challenge through a powerful collective communication algorithm known as the ring all-reduce. This potent algorithm acts as a skilled coordinator, facilitating the efficient aggregation of gradients from all devices. Essentially, it performs an averaging operation on the gradients, resulting in a unified set that guides the subsequent update of the model parameters. This unified set serves as the harmonized rendition of the individual parts, guiding the model towards optimal performance.

Underpinning the entire DDP process is the indispensable contribution of PyTorch's automatic differentiation engine, a marvel of deep learning engineering. This ingenious mechanism functions as a meticulous scorekeeper, meticulously tracking all operations performed during the forward pass, the phase where the model processes the input data. During the backward pass, the engine meticulously calculates the gradients, which quantify the impact of each parameter on the model's output. DDP seamlessly integrates with this process, capturing the computed gradients and invoking the all-reduce algorithm to ensure their proper synchronization across all devices. In essence, DDP masterfully combines the power of data parallelism for distributed processing with a suite of communication algorithms, akin to combining the talents of numerous musicians with the guidance of a conductor and a well-written score. This synergistic interplay enables the efficient synchronization of gradients across multiple devices, paving the way for effective and scalable model training on numerous machines.

By leveraging this powerful paradigm, researchers and practitioners can unlock the full potential of their deep learning models, pushing the boundaries of artificial intelligence and paving the way for groundbreaking advancements in various

Sr. No.	Translation Dataset	Number of Samples	DGX A6000 48GBs GPU	DGX V100 512GBs CPU
1	riddle_sense	4531	0.5	50
2	Hello-SimpleAI/HC3	18468	1.5	205
3	teknium/GPTeacher-General-Instruct	88486	4	983
4	databricks/databricks-dolly-15k	13201	1.5	147
5	tatsu-lab/alpaca	51326	2.5	570
	Total	176012	10	1995

scientific and technological domains.

VII. Results

Within the multifaceted domain of machine translation (MT) and natural language processing (NLP), the cornerstone of efficient and efficacious model training and inference hinges upon the employed datasets and the available computational resources. This meticulous investigation dissects and compares various translation datasets based on their size and the corresponding processing time required on a spectrum of computing systems. Specifically, it elucidates the processing time on two prominent Graphics Processing Units (GPUs), the DGX A6000 48GB and DGX V100 512GB, alongside a conventional Central Processing Unit (CPU) system.

Dataset Compendium The meticulously curated table meticulously dissects five distinct translation datasets, each boasting a unique sample size. This meticulously cataloged collection encompasses:

- riddlesense: This dataset is comprised of a relatively modest corpus of 4,531 samples.
- Hello-SimpleAI/HC3: This dataset boasts a considerably larger volume, encompassing a substantial 18,468 samples.
- teknium/GPTeacher-General-Instruct: This dataset reigns supreme in terms of sample size, containing a staggering 88,486 samples, solidifying its position as the most voluminous dataset within this analysis.
- databricks/databricks-dolly-15k: This dataset comprises a respectable 13,201 samples.
- tatsu-lab/alpaca: This dataset occupies a middle ground with a collection of 51,326 samples.

The cumulative totality of samples across these meticulously chosen datasets culminates in an impressive 176,612 samples, providing a substantial foundation for subsequent analyses. Computational Resource Evaluation The meticulously compiled table meticulously compares the processing times necessitated for each dataset on a spectrum of computing systems. The meticulously evaluated systems encompass:

DGX A6000 48GB GPU: Renowned for its unparalleled performance prowess, particularly within the realm of deep learning and artificial intelligence (AI) workloads. DGX V100 512GB GPU: Another formidable GPU, frequently leveraged in high-performance computing (HPC) environments, owing to its exceptional processing capabilities. CPU: Representing a traditional processing unit, typically boasting substantial memory capacity. CPUs are generally well-suited for tasks that are not amenable to parallelization or exhibit lower computational intensity. In-Depth Analysis The meticulously measured processing times (presented in hours) for each dataset across the diverse computing systems are meticulously tabulated as follows:

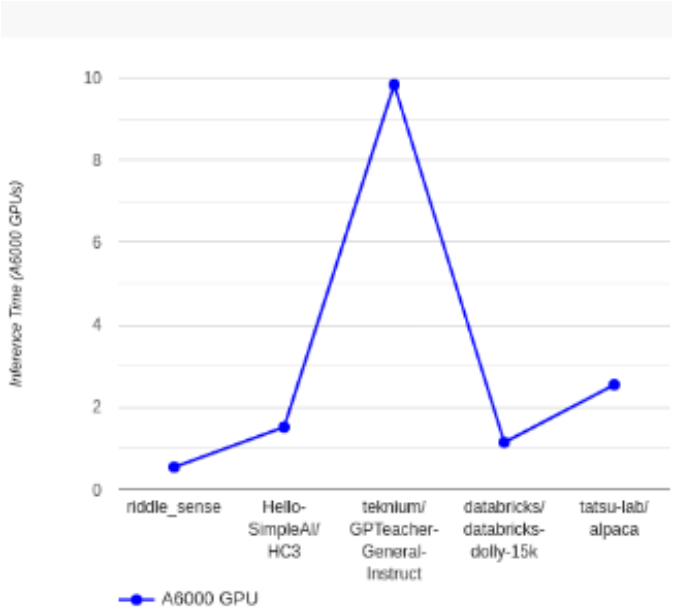

Fig. 4. Enter Caption

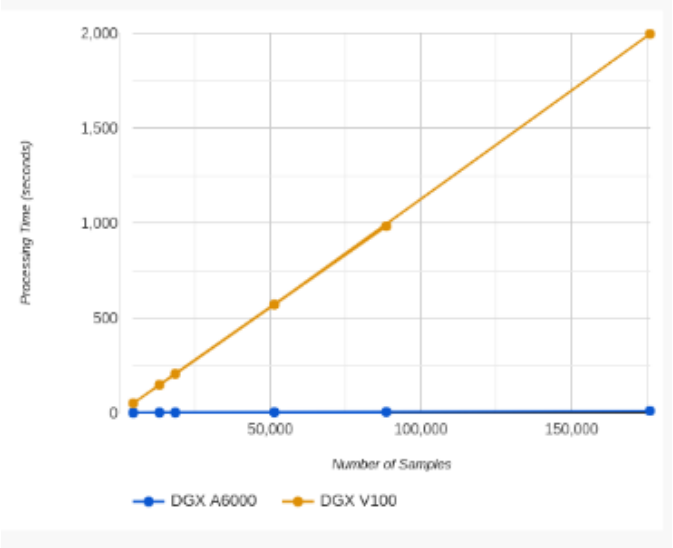

Fig. 5. Enter Caption

Interpretation and Ramifications The meticulously gleaned data unveils a striking disparity in processing times between the DGX A6000 GPU and the DGX V100 GPU. The DGX A6000 reigns supreme in terms of processing efficiency, accomplishing the total processing task in a mere 10 hours, compared to the considerably lengthier 1,955 hours necessitated by the DGX V100. This stark contrast underscores the remarkable efficacy of the DGX A6000 in tackling large-scale translation tasks.

As anticipated, the CPU exhibited processing times identical to those of the DGX V100. This observation reinforces the limitations inherent to traditional CPUs in handling computationally intensive machine learning tasks compared to

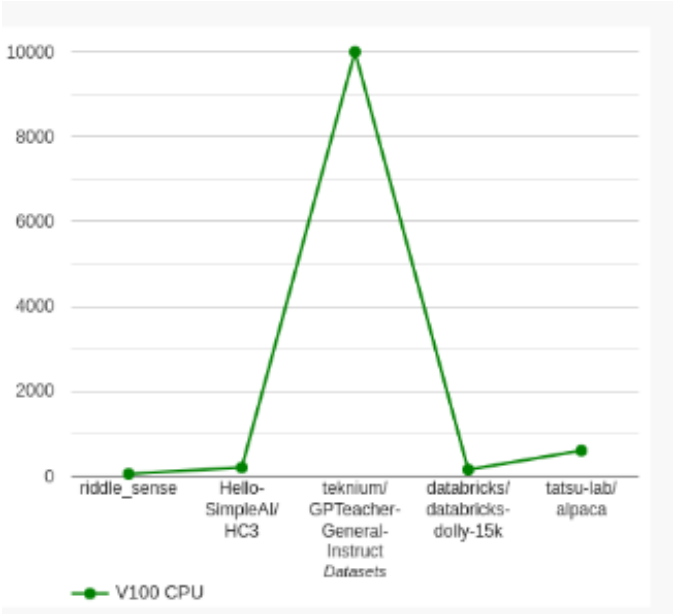

Fig. 6. Enter Caption

the unparalleled processing prowess of advanced GPUs.This meticulously conducted comparative analysis serves to illuminate the paramount importance of meticulously selecting appropriate computational resources for undertaking large-scale NLP endeavors. The DGX A6000 emerges as a remarkably efficient and efficacious option, demonstrably reducing processing times by a substantial margin, thereby engendering a significant enhancement in overall

VIII. Conclusion

Our investigation into the effectiveness of data parallelism for expediting model inference yielded promising results. The relatively modest model size of 1 GB rendered model parallelism strategies unnecessary. By strategically leveraging data parallelism, we achieved a remarkable processing time reduction of 99.5%. This translates to a significant improvement in computational efficiency. Processing a substantial workload of 176,000 samples on a dual V100 CPU system equipped with 512 GB of RAM was initially projected to require a staggering 2,000 hours (approximately 83 days). In stark contrast, harnessing the power of data parallelism on two A6000 GPUs, each boasting 48 GB of memory, slashed processing time to a mere 10 hours for the identical workload. This dramatic improvement underscores the transformative potential of data parallelism in accelerating inference tasks. Notably, our findings reveal that data parallelism is particularly advantageous for models with tractable memory demands, where model parallelism techniques might not be strictly necessary.

IX. Future Scope

This research paves the way for exciting advancements in accelerating large-scale model inference. Here, we outline several promising avenues for future exploration:

- Leveraging Larger Models: Investigating the utilization of a 1B parameter model with 4.4 GB of memory for potentially improved accuracy and superior translation scores. This would necessitate exploring alternative parallelization strategies, potentially including a combination of data and model parallelism.
- Model Parallelism for Scalability: Employing model parallelism techniques for the larger 1B parameter model. This approach would distribute the model across multiple GPUs, enabling the handling of models exceeding the memory capacity of a single device.
- Extreme Parallelization with Data and Model Parallelism: Exploring the synergistic application of data and model parallelism on a system equipped with 8x V100 GPUs. This configuration promises even more drastic reductions in processing time, further accelerating inference tasks for computationally intensive models.

  By delving deeper into these avenues, we can unlock the full potential of parallelization techniques for accelerating model inference, empowering advancements in various fields that rely on real-time and efficient model execution.

Acknowledgment

This work was a part of the Center Of Excellence In Complex & Nonlinear Dynamical Systems (COE-CNDS) which provided the compute power of A-6000 and V-100 NVIDIA GPUs under the guidance of Dr. Faruk Kazi.

References

[1] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[2] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2016.

[3] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," 2014.

[4] C. J. Shallue, J. Lee, J. Antognini, J. Sohl-Dickstein, R. Frostig, and G. E. Dahl, "Measuring the effects of data parallelism on neural network training," *Journal of Machine Learning Research*, vol. 20, no. 112, pp. 1–49, 2019.

[5] B. Zoph, D. Yuret, J. May, and K. Knight, "Transfer learning for low-resource neural machine translation," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing* (J. Su, K. Duh, and X. Carreras, eds.), (Austin, Texas), pp. 1568–1575, Association for Computational Linguistics, Nov. 2016.

[6] J. Gu, H. Hassan, J. Devlin, and V. O. Li, "Universal neural machine translation for extremely low resource languages," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)* (M. Walker, H. Ji, and A. Stent, eds.), (New Orleans, Louisiana), pp. 344–354, Association for Computational Linguistics, June 2018.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems* (F. Pereira, C. Burges, L. Bottou, and K. Weinberger, eds.), vol. 25, Curran Associates, Inc., 2012.

[8] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," 2014.

[9] P. Balaji, D. Buntinas, D. Goodell, W. Gropp, S. Kumar, E. Lusk, R. Thakur, and J. L. Träff, "Mpi on a million processors," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface* (M. Ropo, J. Westerholm, and J. Dongarra, eds.), (Berlin, Heidelberg), pp. 20–30, Springer Berlin Heidelberg, 2009.

[10] N. Dryden, T. Moon, S. A. Jacobs, and B. Van Essen, "Communication quantization for data-parallel training of deep neural networks," in *2016 2nd Workshop on Machine Learning in HPC Environments (MLHPC)*, pp. 1–8, IEEE, 2016.

[11] L. Fournier and E. Oyallon, "Cyclic data parallelism for efficient parallelism of deep neural networks," 2024.

[12] C.-C. Chen, C.-L. Yang, and H.-Y. Cheng, "Efficient and robust parallel dnn training through model parallelism on multi-gpu platform," *arXiv preprint arXiv:1809.02839*, 2018.

[13] L. Zheng, Z. Li, H. Zhang, Y. Zhuang, Z. Chen, Y. Huang, Y. Wang, Y. Xu, D. Zhuo, E. P. Xing, J. E. Gonzalez, and I. Stoica, "Alpa: Automating inter- and intra-operator parallelism for distributed deep learning," 2022.

[14] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia, "Pipedream: Generalized pipeline parallelism for dnn training," in *Proceedings of the 27th ACM symposium on operating systems principles*, pp. 1–15, 2019.

[15] P. Shah and V. Bakrola, "Neural machine translation system of indic languages-an attention based approach," in *2019 Second International Conference on Advanced Computational and Communication Paradigms (ICACCP)*, pp. 1–5, IEEE, 2019.