Compas'2023 : Parallélisme / Architecture/ Système
LISTIC/USMB - Annecy France, 4-7 juillet 2023

A New Parallelization Approach in Deep Learning Using CPU/GPU Architectures for Memory Optimization

Luis A. Torres N.

Universidad Industrial de Santander,
CAGE - SC3UIS
680002 Carrera 27 Calle 9 - Colombie
luis.torres@correo.uis.edu.co

Résumé
Deep Learning algorithms are highly relevant in fields such as computer vision and natural language processing. Similarly, the increasing availability of data has improved the models based on Deep Learning and therefore increased their size and efficiency. The combination of these factors has incurred the need to have increased increasingly efficient computational resources, among which are greater processing capacity and memory available for the execution of the training of these models.
As for memory consumption, the training algorithm of Deep Learning models requires a significant amount that could exceed the capacities of the GPU and, in some cases, the CPU memory. New methods have been created to train the model to resolve this issue, such as Model Parallelism, Data Parallelism, and Pipeline Parallelism. However, these methods do not necessarily reduce the memory footprint but distribute memory requirements among devices such as servers, GPUs, and TPUs.
The high processing capacities provided by GPUs and TPUs are limited by the low memory available on these devices. In search of a solution, new techniques have been developed, such as using sparse arrays, pruning the activations of the network layers, and offloading the activations to the CPU memory to reduce the memory footprint of the training process. Despite these new techniques and methods, memory remains the most significant bottleneck present in the training of Deep Learning models, which is why it has become one of the most active research fields. Many researchers have directed their studies to use the CPU as a more active actor in the training process. The new CPU architectures show some important optimizations directly aimed at training neural networks, particularly with its AVX instruction set.
It is proposed a method of parallelization of the Deep Learning training algorithm by distributing the workload between the CPU and the GPU/TPU to reduce the memory footprint present in these devices without loss of accuracy and performance. To achieve this objective, it is proposed to design an asynchronous method to execute the training distributed between the CPU and the GPU/TPU, reducing waiting times during the forward pass and backward pass balancing workloads.

---

1. Introduction

Deep learning is a complex field of artificial intelligence used to process large amounts of data and identify patterns. While it has myriad applications in fields ranging from healthcare to

finance, some challenges are associated with this technology. One of these is associated with memory. Although deep learning algorithms are very efficient at processing data, they can have difficulties during training due to the amount of data they must retain and the loss of precision due to different factors, generating errors and biases in the system.

In the Deep Learning field exists different specialized algorithms such as Multilayer Perceptron, Convolutional Neural Networks, and Recurrent Neural Networks. Models built from Convolutional Neural Networks (CNNs), such as CoogleNet[21], Resnet [11], and VGG [20], require enormous computational power to train the millions of parameters that contain them. Concerning the Natural Processing Language (NPL) area, models such as Megatron-LM[19] with 8.3 billion parameters and T5[17] with 11 billion parameters present significant restrictions on where to run the training. This problem has become one of the most challenging issues in increasing the performance and accuracy of models [4] [5].

One of the most popular solutions for training large models takes a parallel approach, dividing the model into multiple workers. Each worker trains the part of the model that corresponds to it [8]. This solution reduces the computational load on the device, but adding more devices to train the model increases the cost of communication and synchronization and could significantly decrease performance.

These solutions are possible thanks to the Deep Learning Networks training algorithm called Backpropagation Algorithm. The Backpropagation Algorithm is divided into forward and backward pass operations, composed of matrix multiplications or matrix by vectors. These operations are highly parallelizable and can be computed on accelerators such as GPUs and TPUs, a common practice today [14] [1]. However, GPU memory further increases the constraints on the model size and limits the training algorithm.

Due to this problem with memory and the growth of the models, it has been necessary to design new parallelization techniques such as Data Parallelism, Model Parallelism, and Pipeline Parallelism.

Data parallelism is the most widely used technique. However, this parallelization does not reduce the memory consumed by the model on the GPU. However, it replicates the model on each device, uses a small batch size to train each model on each device, and averages the gradients at the end of the backward pass. The network weights are updated after the gradients are averaged.

Another solution is Model Parallelism, one of the most used to train huge models. In this approach, the number of the model's layers is divided by the number of devices available, and the result is the total layer loaded in each device. The issue with this approach is that it can only scale up to the model's size and requires effective communication between each device. The results at the intra-node level are suitable where the communication bandwidth between devices is high, but its efficiency decreases when scaled to more than one node [19].

Regarding Pipeline Parallelism, it presents a form of parallelization of the model where several data streams are sent through the network to reduce idle times in the devices, a problem that occurs in the parallelism of the model. This approach increases the capacity of the system to train huge models in less time. However, it presents issues with weights updating due to the possibility of working on outdated values of the parameters [12] [16].

These solutions keep all the model parameters in the device's memory during the training process in a static way, regardless of whether they are necessary for the current phase.

Device memory restrictions make these solutions difficult to implement, particularly concerning Data Parallelism. A practice solution is to keep only the necessary data in the device(GPU) memory to execute the process that the backpropagation algorithm requires. The other pa-

rameters are to keep in the host(CPU) memory. This approach receives the name the CPU-Offloading [13]. Specifically, the data that moves between memories are the activations of the layers. This movement must be done intelligently to avoid bottlenecks caused by the data exchange through the data bus.
CPUs are more standardized than GPUs and can provide reasonable acceleration for various applications, so some researchers have been inclined to work on optimizing the training and inference of neural networks on these general-purpose devices. However, this process presents significant challenges ; one of the most important is to match the architecture of the networks with the CPU and optimize the network layers to increase the performance [15]. These challenges require a specific study of the CPU and possible optimizations that can be made when using the CPU.
This work proposes a new approach to Pipeline Parallelism where Backpropagation Algorithm can be distributed between the CPU and the GPU and subsequently parallelize each intralayer operation. The forward and backward pass process works independently. Therefore, our hypothesis states that it is possible to speed up network training and increase the model size that can be trained in a node without losing performance and accuracy. Another aim of this work is to increase the batch size and reach a faster convergence by using the CPU's memory.

2. Background

2.1. Deep Leearning

Machine Learning algorithms can find patterns in data with excellent performance but have trouble working on problems like object recognition or natural language ; when working with Machine Learning algorithms on data with high dimensionality, a phenomenon known as the Curse of Dimensionality [1], which causes the model to fail to generalize well. This issue was one of the main reasons for developing Deep Learning.
Deep Learning is a subfield of Machine Learning based on algorithms inspired by the structures and functions of the human brain. These algorithms have been applied to fields such as Computer Vision, Speech Recognition, Natural Language Processing, Bioinformatics, Medicine, and Climate Science, where the results have sometimes outperformed human experts.
There are three main types of Deep Learning Networks focussed on specific fields :

- Multilayer Perceptron (MLP) : It was the first to use Backpropagation Learning Algorithm to train a model and applied to Supervised Learning.
- Recurrent Neural Network (RNN) : Good performance on tasks that exhibit sequential time-dependent behaviors.
- Convolutional Neural Networks (CNN) : Used in Computer Vision tasks such as Image Recognition and Object Detection.

The data used in Deep Neural Networks can be classified into three main categories : inputs, weights, and activations. The inputs are the data from which patterns are extracted ; weights are the data learned during the training process by comparing the inputs with the outputs ; and the activations are the data set obtained from the output of an activation function whose input is the sum of the weights for the inputs. The amount of data depends on the complexity of the model architecture, the number of layers, and the input batch size. The size of the model depends on the amount of these data.

1. https ://deepai.org/machine-learning-glossary-and-terms/curse-of-dimensionality

2.2. Deep Learning Training Algorithm

The Backpropagation Algorithm was described in the 1970s, but until the 1986 paper by David Rumelhart, Geoffrey Hinton, and Ronald Williams titled "Learning Representation Using Backward Propagation Errors" [18] was not given proper relevance. The article describes different Neural Networks in which the Backpropagation Algorithm works better than other learning approaches and shows that networks trained with this algorithm can deal with problems that could not be addressed before.

This algorithm is used to train neural networks using the chain rule effectively. In simple terms, after the forward pass, the backward pass updates the parameters of the network (weights and biases) while traversing it in reverse. The parameters are adjusted to minimize the actual and desired output vector cost function. The cost function's gradients determine the adjustment level concerning those parameters. Saeed Damadi's paper explains the mathematical of the Backpropagation algorithm and its use in neural networks [7].

2.3. Parallel and Distributed Deep Learning

Deep neural networks have shown a high ability to extract meaningful patterns from data in many tasks. Sometimes the data set is vast, or the number of parameters is high due to the complexity of the model, or both. Some models can require high computational capabilities that cannot be satisfied by a single processing unit, either a node with a CPU or a GPU. In these circumstances, parallel or distributed deep learning increases the computational resources available to train this model and reduces the training time. The example by K. S. and A. Zisserman [20] shows that a VGG network can take up to 10 hours to be trained on a node with 8 CPU cores.

There are some ways to parallelize or distribute the training of the Deep Neural Networks, such as :

— **Local Training** : Both model and data are stored in a single computational node : 1) multicore processing where it is assumed that data and parameters can be stored in this node's memory and trained with all its cores. 2) GPU is used for computing-intensive, particularly for array multiplications. 3) Use CPU and GPU to train the model.

— **Distributed Training** : In some cases, storing the data and the model in a single computational node makes training impossible, so it is necessary to use multiple nodes. Three methods are used to solve this problem ; data parallelism, model parallelism, and pipeline parallelism.

2.3.1. Data Parallelism

Data Parallelism can be defined as the division of the batch for training the neural network. It is used when the model and the batch size exceed the memory capacity of the one processing unit or when it wants to increase the training speed. Data parallelism replicates the model in each processing unit, called worker, and each worker trains a model with its corresponding input batch. This method averages the gradients of each model and then performs the weights update on all workers.

The stochastic gradient descent (SGD) is inherently sequential [24] ; therefore, to perform a parallel SGD, a global model must be maintained that will be responsible for adding the gradients calculated by each worker. This gradient update can be done in two different ways :

— **Synchronous SGD** : A global model is maintained, which updates the weights with the average of all the gradients of the workers. In synchronous SGD, there are two drawbacks : 1) the update time depends on the slowest worker. 2) the scalability depends on the batch size in each worker ; that is, according to Akiba [3] and Goyal [9], a large

mini-batch size in the worker can affect the convergence rates. Several procedures for implementing this gradient update method exist Parallel SGD [24], ShuffleSGD [2], and ADMM SGD [22].

— **Asynchronous SGD** : In synchronous methods, the gradients of each worker must be gathered before updating the weights. This transfer can cause a bottleneck due to limited bandwidth, and as mentioned before, there may be another one that the slower worker causes. Asynchronous methods were developed to deal with these problems. Downpour SGD [8] was the first approach to an asynchronous SGD. In this method, each worker sends his gradients and updates the global model independently of the others.

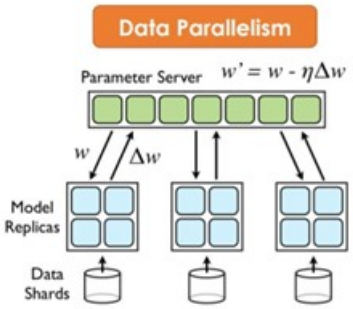

FIGURE 1 – The training data is divided into workers that contain a replica of the model. The Parameter Server collects the gradients ΔW of each worker and returns the updated W parameters[8].

Asynchronous SGD methods have two advantages : 1) performance is not affected by the slower worker, and 2) it is fault-tolerant. On the other hand, it presents a severe disadvantage; it presents the problem of delayed gradients, which can cause global model noise and convergence delays [23].

2.3.2. Model Parallelism

Deep Learning models can have billions of parameters, and assuming each occupies four bytes, the model size can reach hundreds or thousands of Gigabytes, which is a problem if it wants to store on a GPU. Model Parallelism is the solution proposed to solve this problem. The graph of a neural network can be divided so that the edges running between components ((dark lines in Figure 2 are minimal or that the among of data flowing between them is low to minimize the communication overhead. However, to decrease execution time, operations must also be executed in a particular order ensuring that outputs from one task are available when another task requires them as input (scheduling).
It is the most used solution to parallelize training but can only scale up to the model's size. By performing the division of the model, high efficiency in terms of memory use is obtained, but the computational and communication costs increase. At the intra-node level, communication cost is not that high, but when scaled to inter-nodes, the cost increases.

2.3.3. Pipeline Parallelism

The model and the training process are divided among the workers in Model Parallelism. The training is done with the support of the bidirectional backpropagation algorithm (step forward

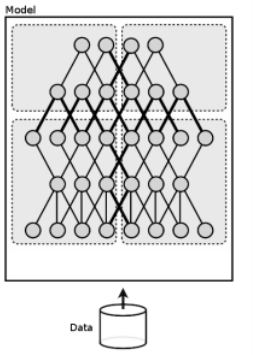

FIGURE 2 – The gray boxes represent different devices (GPU or CPU) or machines. Thicker edges show communication between machines or devices[8].

and step backward). Before fully updating the model, each mini-batch must go through the two layer-by-layer processes. Each worker is idle for some time after the mini-batch has passed through it until it does not have to perform the corresponding calculations in the backward pass. Because of this, Model Parallelism achieves a considerable underutilization of computational resources.
To solve the problem of underutilization of resources, Deepak Narayanan and Aaron Harlap [16] developed a new parallelization scheme named PipeDream and Yanping Huang with a similar approach [12] presented GPipe (Figure 3). This new form of parallelization was called Pipeline Parallelism.

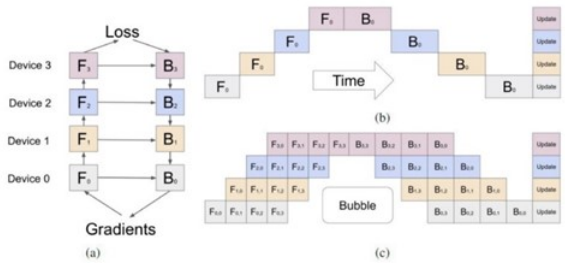

FIGURE 3 – a) Example of a neural network with sequential layers divided into four accelerators. b) Model Parallelism leads to an underutilization of resources due to sequential dependence on the network. c) Pipeline Parallelism divides the input (mini-batch) into micro-batches, allowing different devices to work in different micro-batches simultaneously. Gradients are applied synchronously at the end[12].

GPipe suffers from the problem of weight staleness caused by asynchronous updates, so PipeDream uses a weight storage technique that maintains several versions of them, one for each minibatch, to solve this problem. Each stage processes a mini-batch using the latest available weights in the forward pass. Once the forward pass is complete, PipeDream stores the weights to calculate their update with their respective gradients in the backward pass. In summary, PipeDream has better performance and acceleration than GPipe but also requires a significant amount of memory due to the storage of the weights corresponding to each mini-batch.

2.3.4. CPU Approach

Specialized hardware such as GPUs and TPUs that can accelerate tensor operations are an expensive addiction in Deep Learning projects. On the other hand, CPUs have been gaining traction due to the recent growth in the popularity of deep learning models and the high cost of accelerators [6].

Modern CPUs have evolved significantly regarding cache sizes and memory hierarchy and have added vectorization that allows massive operations (Single Instruction / Multiple Data – SIMD). Similarly, the CPU has been provided with AVX (Advantage Vector Extensions)-512, allowing the processor to process 512-bit registers in a single instruction and allowing a large number of parallel operations. Finally, more modern versions of Intel processors offer support for bfloat16, which allows low-precision arithmetic at high speed, a significant feature in neural network training.

3. Research Proposal

Memory consumption during the training of Deep Learning models requires a considerable amount that usually exceeds the capabilities of the GPU and even those of the compute node. Several methods have been created to solve this problem mentioned above. However, these methods have required increasingly specialized hardware to distribute or parallelize the training and reduce the impact on memory.

As is well known, GPUs are limited by the memory they have. This factor reduces the efficiency that they can offer in terms of network training. This limitation suggests that sometimes the input batch size must be reduced, even with the abovementioned methods. So the researchers have developed other techniques, such as using sparse arrays, pruning the activations of the network layers, and offloading the activations to CPU memory to reduce the memory footprint of the training process. However, memory remains the most critical bottleneck, which is why it has become one of the most active fields of study. Faced with this challenge, many researchers have directed their studies to use the GPU as an active actor in the training process; this is possible thanks to new CPU architectures, particularly those from Intel, which show essential optimizations aimed directly at training neural networks, such as AVX or AMX.

As seen above, it is proposed to build a method of parallelization of the Backpropagation algorithm through which the workloads are distributed between the CPU and the GPU to reduce the memory footprint present in the latter without further loss of precision and performance. This method will be based on PipeDream and will use a method like stochastic weighted average [10] for updating the network weights.

In order to fulfill this objective, it is proposed to analyze the memory footprint of training in some CPU and GPU neural network models to determine the factors that significantly affect each architecture. On the other hand, it is necessary to determine if instruction sets such as AVX allows obtaining similar results in training and what factors are the determinants of their differences. Once these analyses have been carried out, it is proposed to design an asynchronous method that allows to distribute of the Backpropagation algorithm between CPU and GPU and reduces the inactivity times that may occur.

Bibliographie

1. Abadi (M.), Agarwal (A.), Barham (P.), Brevdo (E.), Chen (Z.), Citro (C.), Corrado (G. S.), Davis (A.), Dean (J.), Devin (M.), Ghemawat (S.), Goodfellow (I.), Harp (A.), Irving (G.), Isard (M.), Jia (Y.), Jozefowicz (R.), Kaiser (L.), Kudlur (M.), Levenberg (J.), Mane (D.), Monga (R.), Moore (S.), Murray (D.), Olah (C.), Schuster (M.), Shlens (J.), Steiner (B.), Sutskever (I.), Talwar (K.), Tucker (P.), Vanhoucke (V.), Vasudevan (V.), Viegas (F.), Vinyals (O.), Warden (P.), Wattenberg (M.), Wicke (M.), Yu (Y.) et Zheng (X.). – Tensorflow : Large-scale machine learning on heterogeneous distributed systems, 2016.
2. Ahn (K.), Yun (C.) et Sra (S.). – Sgd with shuffling : optimal rates without component convexity and large epoch requirements, 2020.
3. Akiba (T.), Suzuki (S.) et Fukuda (K.). – Extremely large minibatch sgd : Training resnet-50 on imagenet in 15 minutes, 2017.
4. Chilimbi (T.), Suzue (Y.), Apacible (J.) et Kalyanaraman (K.). – Project adam : Building an efficient and scalable deep learning training system. – In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, *OSDI'14*, OSDI'14, p. 571–582, USA, 2014. USENIX Association.
5. Cui (H.), Zhang (H.), Ganger (G. R.), Gibbons (P. B.) et Xing (E. P.). – Geeps : Scalable deep learning on distributed gpus with a gpu-specialized parameter server. – In *Proceedings of the Eleventh European Conference on Computer Systems*, *EuroSys '16*, EuroSys '16, New York, NY, USA, 2016. Association for Computing Machinery.
6. Daghaghi (S.), Meisburger (N.), Zhao (M.), Wu (Y.), Gobriel (S.), Tai (C.) et Shrivastava (A.). – Accelerating slide deep learning on modern cpus : Vectorization, quantizations, memory optimizations, and more, 2021.
7. Damadi (S.), Moharrer (G.) et Cham (M.). – The backpropagation algorithm for a math student, 2023.
8. Dean (J.), Corrado (G.), Monga (R.), Chen (K.), Devin (M.), Mao (M.), Ranzato (M. a.), Senior (A.), Tucker (P.), Yang (K.), Le (Q.) et Ng (A.). – Large scale distributed deep networks. – In Pereira (F.), Burges (C.), Bottou (L.) et Weinberger (K.) (édité par), *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2012.
9. Goyal (P.), Dollár (P.), Girshick (R.), Noordhuis (P.), Wesolowski (L.), Kyrola (A.), Tulloch (A.), Jia (Y.) et He (K.). – Accurate, large minibatch sgd : Training imagenet in 1 hour, 2017.
10. Guo (H.), Jin (J.) et Liu (B.). – Stochastic weight averaging revisited, 2022.
11. He (K.), Zhang (X.), Ren (S.) et Sun (J.). – Deep residual learning for image recognition, 2015.
12. Huang (Y.), Cheng (Y.), Bapna (A.), Firat (O.), Chen (M. X.), Chen (D.), Lee (H.), Ngiam (J.), Le (Q. V.), Wu (Y.) et Chen (Z.). – Gpipe : Efficient training of giant neural networks using pipeline parallelism, 2018.
13. Jhu (C.-F.), Liu (P.) et Wu (J.-J.). – Data pinning and back propagation memory optimization for deep learning on gpu. – In *2018 Sixth International Symposium on Computing and Networking (CANDAR)*, pp. 19–28. IEEE, 2018.
14. Jia (Y.), Shelhamer (E.), Donahue (J.), Karayev (S.), Long (J.), Girshick (R.), Guadarrama (S.) et Darrell (T.). – Caffe : Convolutional architecture for fast feature embedding, 2014.
15. Mittal (S.), Rajput (P.) et Subramoney (S.). – A survey of deep learning on cpus : Opportunities and co-optimizations. *IEEE Transactions on Neural Networks and Learning Systems*, 2021, pp. 1–21.
16. Narayanan (D.), Harlap (A.), Phanishayee (A.), Seshadri (V.), Devanur (N. R.), Ganger (G. R.), Gibbons (P. B.) et Zaharia (M.). – Pipedream : Generalized pipeline parallelism

for dnn training. – In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, *SOSP '19*, SOSP '19, p. 1–15, New York, NY, USA, 2019. Association for Computing Machinery.

17. Raffel (C.), Shazeer (N.), Roberts (A.), Lee (K.), Narang (S.), Matena (M.), Zhou (Y.), Li (W.) et Liu (P. J.). – Exploring the limits of transfer learning with a unified text-to-text transformer, 2019.
18. Rumelhart (D. E.), Hinton (G. E.) et Williams (R. J.). – *Learning Representations by Back-Propagating Errors*, p. 696–699. – Cambridge, MA, USA, MIT Press, 1988.
19. Shoeybi (M.), Patwary (M.), Puri (R.), LeGresley (P.), Casper (J.) et Catanzaro (B.). – Megatron-lm : Training multi-billion parameter language models using model parallelism, 2019.
20. Simonyan (K.) et Zisserman (A.). – Very deep convolutional networks for large-scale image recognition, 2014.
21. Szegedy (C.), Liu (W.), Jia (Y.), Sermanet (P.), Reed (S.), Anguelov (D.), Erhan (D.), Vanhoucke (V.) et Rabinovich (A.). – Going deeper with convolutions, 2014.
22. Wang (J.), Yu (F.), Chen (X.) et Zhao (L.). – Admm for efficient deep learning with global convergence. – In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, *KDD '19*, KDD '19, New York, NY, USA, 2019. Association for Computing Machinery.
23. Zheng (S.), Meng (Q.), Wang (T.), Chen (W.), Yu (N.), Ma (Z.-M.) et Liu (T.-Y.). – Asynchronous stochastic gradient descent with delay compensation, 2016.
24. Zinkevich (M. A.), Weimer (M.), Smola (A.) et Li (L.). – Parallelized stochastic gradient descent. – In *Proceedings of the 23rd International Conference on Neural Information Processing Systems - Volume 2*, *NIPS'10*, NIPS'10, p. 2595–2603, Red Hook, NY, USA, 2010. Curran Associates Inc.