Akintoye, Samson B, Han, Liangxiu ORCID logoORCID: https://orcid.org/0000-0003-2491-7473, Zhang, Xin ORCID logoORCID: https://orcid.org/0000-0001-7844-593X, Chen, Haoming and Zhang, Daqiang (2022) A hybrid parallelization approach for distributed and scalable deep learning. IEEE Access, 10. pp. 77950-77961. ISSN 2169-3536

Downloaded from: https://e-space.mmu.ac.uk/630155/
Version: Published Version
Publisher: Institute of Electrical and Electronics Engineers (IEEE)
DOI: https://doi.org/10.1109/access.2022.3193690
Usage rights: Creative Commons: Attribution 4.0

Please cite the published version
A Hybrid Parallelization Approach for Distributed and Scalable Deep Learning

SAMSON B. AKINTOYE1, LIANGXIU HAN*, XIN ZHANG1, HAOMING CHEN2, AND DAOQIANG ZHANG3

1Department of Computing and Mathematics, Manchester Metropolitan University, UK (e-mail: s.akintoye@mmu.ac.uk; l.han@mmu.ac.uk; x.zhang@mmu.ac.uk)
2Department of Computer Science, University of Sheffield, UK (e-mail: hchen78@sheffield.ac.uk)
3College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, P.R.China (e-mail: dqzhang@nuaa.edu.cn)

* Corresponding author: L. Han (e-mail: l.han@mmu.ac.uk).

ABSTRACT Recently, Deep Neural Networks (DNNs) have recorded significant success in handling medical and other complex classification tasks. However, as the sizes of DNN models and the available datasets increase, the training process becomes more complex and computationally intensive, usually taking longer to complete. In this work, we have proposed a generic full end-to-end hybrid parallelization approach combining model and data parallelism for efficiently distributed and scalable training of DNN models. We have also proposed a Genetic Algorithm Based Heuristic Resources Allocation (GABRA) mechanism for optimal distribution of partitions on the available GPUs for computing performance optimization. We have applied our proposed approach to a real use case based on 3D Residual Attention Deep Neural Network (3D-ResAttNet) for efficient Alzheimer Disease (AD) diagnosis on multiple GPUs and compared with the existing state-of-the-art parallel methods. The experimental evaluation shows that our proposed approach is 20% averagely better than existing parallel methods in terms of training time and achieves almost linear speedup with little or no differences in accuracy performance when compared with the existing non-parallel DNN models.

INDEX TERMS Deep Learning, Genetic Algorithm, Data Parallelization, Model Parallelization

I. INTRODUCTION In recent time, Deep Neural Networks (DNNs) have gained popularity as an important tool for solving complex tasks ranging from image classification [1], speech recognition [2], medical diagnosis [3], [4], to the recommendation systems [5] and complex games [6], [7]. However, training a DNN model requires a large volume of data, which is both data and computational intensive, leading to increased training time. To overcome this challenge, various parallel and distributed computing methods [8] have been proposed to scale up the DNN models to provide timely and efficient learning solutions. Broadly, it can be divided into data parallelism, model parallelism, pipeline parallelism and hybrid parallelism (a combination of data and model parallelism). Data parallelism is a parallelization method that trains replicas of a model on individual devices using different subsets of data, known as mini-batches [9], [10]. In data parallel distributed training, each computing node or a worker contains a neural network model replica and a churn of dataset, and compute gradients which are shared with other workers and used by the parameter server to update the model parameters [11]. However, as parameters increases, the overhead for parameter synchronization inceases, leading to performance degradation. In addition, when a DNN model size is too big, it couldn’t be executed on a single device. Hence it is not possible to perform data parallelization. Model parallelism is a parallelization method where a large model is split, running concurrent operations across multiple devices with the same mini-batch [8]. It can help to speed up the DNN training either through its implementation or algorithm. In model parallelism, each node or a worker has distinct parameters and computation of layer of a model, and also updates weight of allocated model layers. Pipelining parallelism splits the DNN models training tasks into a sequence of processing stages [56]. Each stage takes the result from the previous stage as input, with results being passed downstream immediately. Recently, the combination of model and data parallelization methods known as Hybrid parallelization has been explored to leverage the benefits of both methods to minimize communication overhead in the multi-device parallel training of DNN models [15], [18], [19]. Despite the performance of the existing parallelization meth-
ods, they are still subject to further improvement by optimally allocating the model computations and data partitions to the available devices for better model training performance. In this paper, we have proposed a generic hybrid parallelization approach for parallel training of DNN in multiple Graphics Processing Units (GPUs) computing environments, which combines both model and data parallelization methods. Our major contributions are as follows:

- Development of a generic full end-to-end hybrid parallelization approach for the multi-GPU distributed training of a DNN model.
- Model parallelization by splitting a DNN model into independent partitions, formulating the network partitions-to-GPUs allocation problem as a 0-1 multiple knapsack model, and proposing a Genetic Algorithm based heuristic resources allocation (GABRA) approach as an efficient solution to optimize the resources allocation.
- Exploitation of data parallelization based on the All-reduced method and asynchronous stochastic gradient descent across multiple GPUs for further acceleration of the overall training speed.
- Evaluation of the proposed approach through a real use case study – by parallel and distributed training of a 3D Residual Attention Deep Neural Network (3D-ResAttNet) for efficient Alzheimer’s disease diagnosis.

The remainder of this paper is organized as follows: Section II reviews the related work of the study. Section III discusses the details of the proposed approach. In Section IV, the experimental evaluation is described. Section V concludes the work.

II. RELATED WORK

This section provides an overview in relation to distributed training of deep neural networks and genetic algorithms for resource optimisation.

A. PARALLEL AND DISTRIBUTED TRAINING OF DEEP NEURAL NETWORKS (DNNS)

As mentioned earlier, existing efforts on parallel and distributed training of DNNs can be broadly divided into three categories, which include data parallelism, model parallelism, pipeline parallelism and hybrid parallelism.

1) Data Parallelism

In data parallelism, a dataset is broken down into mini-batches and distributed across the multiple GPUs and each GPU contains a complete replica of the local model and computes the gradient. The gradients aggregation and updates among the GPUs are usually done either synchronously or asynchronously [11]. In synchronous training, all GPUs wait for each other to complete the gradient computation of their local models, then aggregate computed gradients before being used to update the global model. On the other hand, in asynchronous training, the gradient from one GPU is used to update the global model without waiting for other GPUs to finish. The asynchronous training method has higher throughput in that it eliminates the waiting time incurred in the synchronous training method. In both asynchronous and synchronous training, aggregated gradients can be shared between GPUs through the two basic data-parallel training architectures: parameter server architecture and AllReduce architecture. Parameter server architecture [14] is a centralized architecture where all GPUs communicate to a dedicated GPU for gradients aggregation and updates. Alternately, AllReduce architecture [20] is a decentralized architecture where the GPUs share parameter updates in a ring network topology manner through the AllReduce operation.

2) Model Parallelism

In model parallelization, model layers are divided into partitions and distributed across GPUs for parallel training [21]. [22]. In model parallel training, each GPU has distinct parameters and computation of the layer of a model, and also updates weight of allocated model layers. Huo et al. [51] proposed a Decoupled Parallel Back-propagation (DDG), which splits the network into partitions and solves the problem of backward locking by storing delayed error gradient and intermediate activations at each partition. Similar, Zhuang et al. [52] adopted the delayed gradients method to propose a fully decoupled training scheme (FDG). The work breaks a neural network into several modules and trains them concurrently and asynchronously on multiple devices. However, the major challenges are how to break the model layers into partitions as well as the allocation of partitions to GPUs for efficient training performance [16]. Moreover, using model parallelization alone does not scale well to a large number of devices [17] as it involves heavy communication between workers.

3) Pipelining Parallelism

Pipelining parallelism breaks the task (data and model) into a sequence of processing stages. Each stage takes the result from the previous stage as input, with results being passed downstream immediately [53]. Various works have adopted this technique. Lee et al. [54] used the pipeline parallelism approach to overlap computation and communication for CNN training. They implement a thread in each computer server to spawn communication processes after the gradient is generated. Chen et al. [55] proposed a pipelined model parallel execution method for high GPU utilisation and used a novel weight prediction technique to achieve a robust training accuracy. However, one of the significant drawbacks of pipelining parallelism is that it is limited by the slowest stages and has limited scalability.

4) Hybrid Parallelism

Several research works have explored both data and model parallelization methods for efficient DNN models training. Yadan et al. [23] achieved 2.2× speed-up when trained a large deep convolutional neural network model with hy-
bridized data and model parallelism. Krizhevsky et al. [9] used model and data parallelization techniques to train a large deep convolutional neural network and classify 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. Shazeer et al. [25] proposed Mesh-TensorFlow where data parallelism is combined with model parallelism to improve training performance of transformer model with a huge number of parameters. In Mesh-TensorFlow, users split layers across the multi-dimensional mesh of processors and explored data parallelism technique in conjunction with All-reduced update method. Moreover, Onoufriou et al. [26] proposed Nemesyst, a novel end-to-end hybrid parallelism deep learning-based Framework, where model partitions are trained with independent data sets simultaneously. Similarly, Oyama et al. [27] proposed end-to-end hybrid-parallel training algorithms for large-scale 3D convolutional neural networks. The algorithms combine both data and model parallelisms to increase throughput and minimize I/O scaling bottlenecks. The above-aforementioned approaches adopted data, model and pipeline parallelization separately or the combination of the methods to improve the performance of DNN models training. However, none of the existing methods considered the resource utilization and allocation problem in deep learning and provided solutions for efficient distributed training performance.

B. GENETIC ALGORITHMS FOR RESOURCE MANAGEMENT OPTIMIZATION

Resource management optimization is an important research topic in distributed computing systems [28]. Several works have been proposed, with different techniques for addressing resource management problems, such as scheduling [29] and allocation [30]. Genetic Algorithms (GA)’s are commonly used to optimize either homogenous or heterogeneous resources in distributed system environments [31] [57]. For instances, Gai et al. [32] proposed the Cost-Aware Heterogeneous Cloud Memory Model (CAHCM) to provide high performance cloud-based heterogeneous memory service offerings. It proposed the Dynamic Data Allocation Advanced (2DA) algorithm based on genetic programming to determine the data allocations on the cloud-based memories for the model. Mezache et al. [33] proposed a resource allocation method based on GA to minimize the number of hosts required to execute a set of cloudlet associated with the corresponding set of the virtual machine, thereby reducing excessive power consumption in the data centre. Furthermore, Jiang et al. [34] proposed a multi-objective model based on the non-dominated sorting genetic algorithm to minimize the expected total makespan and the expected total cost of the disassembly service under the uncertain nature of the disassembly process. Mosa and Sakellariou [35] proposed a dynamic VM placement solution used a GA to optimize the utilization of both CPU and memory with the aim to ensure better overall utilization in the cloud data centre. Devarasetty and Reddy [36] proposed an optimization method for resource allocation in the cloud with the aim to minimize the deployment cost and improve the QoS performance. They used the GA to find optimal solutions to the allocation problem. In addition to resource allocation in the cloud environment, Mata and Guardieiro [37] investigated the resource allocation in the Long-Term Evolution (LTE) uplink and proposed a scheduling algorithm based on GA to find a solution for allocating LTE resource to the user requests. Moreover, Li and Zhu [38] adopted genetic algorithm to develop a joint optimization method for offloading tasks to the mobile edge servers (MESs) in a mobile-edge computing environment under limited wireless transmission resources and MESs’ processing resources. However, none of the work mentioned above considered the resource allocation problem in deep learning and applied GA to solve the problem for efficient training performance of the DNN model.

III. THE PROPOSED APPROACH

In parallel and distributed computing, there are several considerations on efficient training of DNN models including: 1) how to decompose a model or a dataset into parts/small chunks; 2) how to map and allocate these parts onto distributed resources for efficient computation as well as reducing communication overhead between computing nodes. This work has proposed a generic full end-to-end hybrid parallelization approach for efficient training of a DNN model, which combines both data and model parallelization. For data parallelization, we have exploited data parallelization based on the All-reduced method and asynchronous stochastic gradient descent across multiple GPUs for acceleration of the overall network training speed. For model parallelization, model layers are partitioned individually with the aim to reduce communication overhead during training process. We have also designed Genetic Algorithm-based heuristic resource allocation mechanism to map and allocate partitions to appropriate resources for efficient DNN training.

Figure 1 shows the high-level architecture, including 1) model parallelization consisting of network partitions and resource allocation components; and 2) data parallelization. The details of the proposed method are presented in the following sections. The important notations in this paper are detailed in Table 1.

A. MODEL PARALLELIZATION

Model parallelization includes neural network model partitioning and Genetic Algorithm based heuristic resource allocation mechanism.

1) Network Partitioning

The principle of the network partitioning is based on the computation loads of each layer with the aim to reduce communication overhead during training process. The highly functional layers are partitioned individually as a single partition for even distribution of the DNN model layers. For instance, a convolution layer of CNN architecture has a large
gives activation a to the input (is passed to the next partition computed simultaneously, the gradient of the partition input layer of each partition. In addition to this, all partitions are ≤ 1 partitions {s1, s2, ..., sQ}.

Specifically, let’s assume a model network contains a set of layers {s1, s2, ..., sQ}. The model network P is split into partitions {p1, p2, ..., pn} where pi = {s1, s2, ..., si-1, si+1, ..., sk}, which denotes a set of layers in i partition such that 1 ≤ i ≤ n. si-1 and si+1 are the second layer and last layer of each partition. In addition to this, all partitions are computed simultaneously, the gradient of the partition input is passed to the next partition (i+1), while the partition output is sent to partition (i+1) as its new input. In forward pass, the g_{s_{i+1}}^-1 denotes the gradient at partition (i+1) at iteration t. For each layer (si ≤ q ≤ s_{i+1} - 1) such that q ≤ Q, the gradient is given as: 

\[ g_{w_q}^t = \frac{\delta a_{s_{i+1}}^t}{\delta w_q^t} g_{s_{i+1}}^t - 1 \]

which can be updated by:

\[ w_{q_{t+1}}^{t+1} = w_{q_{t+1}}^t - \gamma t g_{w_q}^t \]

where \( \gamma t \) is learning rate. 

\[ g_{w_q}^t = \frac{\delta a_{s_{i+1}}^t}{\delta w_q^t} g_{s_{i+1}}^t - 1 \]  \hspace{0.5cm} (1)

which can be updated by:

\[ w_{q_{t+1}}^{t+1} = w_{q_{t+1}}^t - \gamma t_{t+1} g_{w_q}^t - 1 \]  \hspace{0.5cm} (2)

where \( \gamma t_{t+1} \) is learning rate.

2) Genetic Algorithm Based Resource Allocation (GABRA)

To enable efficient DNN model training on multiple GPUs, we have also proposed a Genetic Algorithm-based heuristic resource allocation mechanism. A genetic algorithm (GA) is one of the evolutionary algorithms commonly used to provide efficient solutions to optimisation problems such as resource allocation problems, based on biologically inspired operators such as mutation, crossover and selection. The proposed genetic based algorithm aims to finds the best network partitions to be allocated to the available GPUs to maximise resource utilisation and minimise the computation time of network partitions for efficient and better overall training performance over the existing methods.

Thus, we formulate the problem of allocating GPUs to network partitions as a 0-1 multiple knapsack problem model.
As previously illustrated, we consider computation load of a set of partitions \( p_i \), where \( i = \{1, 2, \ldots, n\} \). We also consider the capacity of a set of available GPU \( G \), each denoted by \( d_j \) where \( j = \{1, 2, \ldots, m\} \) and \( d_j \in G \). Furthermore, we assume that the GPUs can either be heterogeneous or homogeneous; with the different or same capacities items of memory. Each GPU runs at least one partition, and each partition needs to be allocated to only one GPU. Let \( C = (c_{ij}) \in \mathbb{R}^{n \times m} \) be a \( n \times m \) matrix in which \( c_{ij} \) is a profit of allocating gpu \( j \) to partition \( i \):

\[
c_{ij} = \frac{p_i}{d_j}
\]

Also, Let \( X = (x_{ij}) \in \mathbb{R}^{n \times m} \) where

\[
x_{ij} = \begin{cases} 
1, & \text{if gpu } j \text{ is allocated to partition } i \\
0, & \text{otherwise}.
\end{cases}
\]

Thus, we formulate the multiple knapsack model in terms of a function \( z \) as:

\[
\max_{p,x,c} z(X) = \sum_{i=1}^{n} \sum_{j=1}^{m} x_{ij} c_{ij}
\]

subject to:

\[
\sum_{i=1}^{n} p_i x_{ij} \leq d_j, \forall j \in M = \{1, 2, \ldots, m\}
\]

\[
\sum_{j=1}^{m} x_{ij} = 1, \forall i \in N = \{1, 2, \ldots, n\}
\]

\[
x_{ij} = 0 \text{ or } 1, \text{ for } i = 1, 2, \ldots, n, \text{ and for } j = 1, 2, \ldots, m
\]

Our goal is to find (8) that guarantees no GPU is overutilized and yields the maximum profit simultaneously. Thus, the objective function in equation 5 maximizes the sum of the profits of the selected partitions. The constraint Eq. 6 implies that each partition is allocated to at most one GPU, while constraints Eq. 7 ensures that the capacity of each available GPU is not exceeded.

Next, we present a Genetic Algorithm-Based Resources Allocation (GABRA) as an efficient solution to the model. Genetic Algorithm has been proven as a stochastic method to produce high-quality solutions for solving combinatorial optimization problems, particularly NP-hard problems [39].

The Algorithm 1 shows the pseudo-code of the GABRA for solving GUPs-to-partitions allocation problem. It consists of four major parts: input, initialization, looping and output. In the initialization part (line 3), unlike the classical GA, the set of chromosomes which also known as initial population \( \mathcal{P}(t) \) for allocating GPUs to partitions, is generated as indicated in the Algorithm 2, by randomizing the allocation of resources without exceeding their capacities with respect to the computation load of each network partition.

---

**Algorithm 1: Genetic Algorithm Based Resources Allocation (GABRA)**

**input**: \( \{p_1, p_2, \ldots, p_n\} \): computation loads of partitions

\( \{d_1, d_2, \ldots, d_m\} \): capacity values of available GPUs

**output**: optimized solution \( f(Z^*) \)

1. evaluate \( c_{ij} \leftarrow \frac{p_i}{d_j} \) for \( i = 1, 2, \ldots, n \) and \( j = \{1, 2, \ldots, m\} \);
2. set \( t \leftarrow 0 \);
3. initialise \( \mathcal{P}(t) \leftarrow \{\beta_1, \beta_2, \ldots, \beta_n\} \);
4. evaluate \( \mathcal{P}(t) : \{f(\beta_1), f(\beta_2), \ldots, f(\beta_n)\} \);
5. find \( Z^* \in \mathcal{P}(t) \) such that \( f(Z^*) \geq f(Z), \forall Z \in \mathcal{P}(t) \);
6. while \((t < t_{\text{max}}) \) do
7. select \( Y_1, Y_2 = \phi(\mathcal{P}(t)) \); // \( \phi \) is a selection function;
8. crossover \( W \leftarrow \Psi_c(Y_1, Y_2) \); // \( \Psi_c \) is a crossover function;
9. mutate \( W \leftarrow \Psi_m(W) \); // \( \Psi_m \) is a mutation function;
10. if \( W = \text{any } Z \in \mathcal{P}(t) \) then
11. \quad go to 7
12. end if
13. evaluate \( f(W) \);
14. find \( Z' \in \mathcal{P}(t) \) such that
15. \quad \( f(Z') \leq f(Z), \forall Z \in \mathcal{P}(t) \) and replace \( Z' \leftarrow W \);
16. if \( f(W) > f(Z^*) \) then
17. \quad \( Z^* \leftarrow W \); //update best fit \( Z^* \)
18. end if
19. \quad \( t \leftarrow t + 1 \);
20. end while
21. return \( Z^*, f(Z^*) \)

---

**Algorithm 2: initial population algorithm**

**input**: \( \{p_1, p_2, \ldots, p_n\} \): computation loads of partitions

\( \{d_1, d_2, \ldots, d_m\} \): capacity values of available GPUs

**output**: initial population

1. for \((\text{all partition loads}) \) do
2. \quad randomize the allocation of partitions to the number of the available GPUs
3. end for
4. return initial population
The looping part contains fitness evaluation, selection, crossover and mutation functions. The objective is to optimize the total profit of allocating GPUs to partitions. The fitness evaluation validates the optimal solution condition with respect to the optimization objectives. Thus, the fitness value of each chromosome is calculated as:

$$f(\beta) = \sum_{i=1}^{n} c_{ij} \beta_j, \text{ and for } j = 1, 2, \ldots, m$$  \hspace{1cm} (9)

In the case where the optimal solution condition is not satisfied the optimization objectives, a new population is computed from an initial population of the solutions using their fitness values and genetic functions: selection, crossover and mutation functions in the looping part (lines 7 - 18). We use the selection function ($\phi$), which is based on the roulette wheel method [50] to select the best chromosomes. The selection is based on the chromosomes’ fitness values, representing the total profit of allocating partitions to the available GPUs. The chromosomes with higher fitness values are selected for the generation of the next population. The midpoint crossover function $\Psi_c$, as described in Algorithm 3, works on two-parent chromosomes $\{Y_1, Y_2\}$ with crossover probability 0.8 and produces a new individual.

**Algorithm 3: Crossover function ($\Psi_c$)**

- **input**: $Y_1, Y_2$: two parent chromosomes
- **output**: $Y_{\lambda 1}, Y_{\lambda 2}$: two offspring chromosomes
  1. $\Phi \leftarrow \text{length}(Y_1)$;
  2. $cp \leftarrow \frac{\Phi}{2}$; /*mid cross point;*/
  3. $Y_{\lambda 1} \leftarrow Y_1(1:cp) \cup Y_2(cp: \Phi)$;
  4. $Y_{\lambda 2} \leftarrow Y_1(cp: \Phi) \cup Y_2(1:cp)$;
  5. **return** $Y_{\lambda 1}, Y_{\lambda 2}$

Next, the inversion mutation functions $\Psi_m$ is adopted where a subset of genes in a chromosome is selected and inverted to form mutated offspring. In the line 14, the old chromosomes in the current population are replaced with the new chromosomes to form a new population. Finally, the algorithm terminates when the maximum number of generations is reached, or the optimal total profit of allocating GPUs-to-partitions is obtained.

**B. DATA PARALLELISATION**

To accelerate the training process, each GPU uses a different mini-batch that is GPU1 uses the first mini-batch, GPU2 uses the second mini-batch and so on. To reduce computation time, the full DNN is trained by training a partition with mini-batch in all GPUs concurrently. Furthermore, we adopt Asynchronous Stochastic Gradient Descent (ASGD) [8] as well as ring All-reduce mechanisms [40] for parameter updates to complete an iteration. The process continues until all the iterations are completed. ASGD achieves a faster training speed as there is no need to wait for the slowest GPU in every iteration for the global model updates. The ring All-reduce is an optimal communication algorithm to minimize the communication overhead among the GPUs, where all GPUs are logically arranged in a ring All-reduce topology. Each GPU sends and receives the required information to update its model parameters from the neighbour GPUs.

In all, the objective is to minimize as follows:

$$f(w; V) = \frac{1}{b \times m} \sum_{i=1}^{b \times m} \ell(w, v_i)$$  \hspace{1cm} (10)

where $f$ is a neural network, $b$ is the batch size, $m$ is the number of GPUs, $\ell$ is a loss function for each data point $v \in V$, and $w$ is the trainable parameter of the neural network.

The derivative of this objective which also referred to as the gradient is given as:

$$\frac{\partial f(w; V)}{\partial w} = \frac{1}{b \times m} \sum_{i=1}^{b \times m} \frac{\partial \ell(w, v_i)}{\partial w}$$  \hspace{1cm} (11)

In data parallelization, the gradient updates is calculated as a sum of summations each of which is the sum of derivatives over $b$ data points, and is given as:

$$\frac{\partial f(w; V)}{\partial w} = \frac{1}{m} \left( \frac{1}{b} \sum_{i=1}^{b} \frac{\partial \ell(w, v_i)}{\partial w} + \frac{1}{b} \sum_{i=b+1}^{b+2} \frac{\partial \ell(w, v_i)}{\partial w} + \cdots + \frac{1}{b} \sum_{i=b \times (m-1)+1}^{b \times m} \frac{\partial \ell(w, v_i)}{\partial w} \right)$$  \hspace{1cm} (12)

In addition, the speed of data-parallel training with $m$ GPUs can be expressed as:

$$ST_m = \frac{T_1}{T_m} \times \frac{TS_1}{TS_m} \times \frac{E_1}{E_m}$$  \hspace{1cm} (13)

where $T_1$ is the average training time per step for using one GPU, while $T_m$ is the time per step for using $m$ GPUs. $E_1$ is the number of epochs required to converge for one GPU, while $E_m$ is the number of epochs required for $m$ GPUs.

**IV. EXPERIMENTAL EVALUATION THROUGH A REAL USE CASE STUDY**

We have applied our approach to a real case study in Neurocomputing to evaluate the effectiveness of the proposed method in this work. Previously, we have developed a 3D explainable residual self-attention convolutional neural network (3D-ResAttNet) to automatically classify discriminative atrophy localization on sMRI image for Alzheimer’s Disease (AD) diagnosis [42]. It is a non-parallel model and runs only on a single GPU. To evaluate the proposed parallel approach, we have parallelized our previous 3D-ResAttNet model, run it on a homogenous multiple-GPUs setting and compared the performance with and without parallelization. Moreover, we have compared our approaches with the state-of-the-art methods including Distributed Data Parallel (DDP) and Data Parallel (DP) from PyTorch framework [43], FDG [51] and DDG [52].
A. EVALUATION METRICS
We have adopted standard metrics for performance evaluation including Speedup (S), Accuracy (ACC) and Training Time (TT). The Speedup (S) is to measure the scalability and computing performance. It is defined as the ratio of the serial runtime of the best sequential algorithm for solving a problem to the time taken by the parallel algorithm to solve the same problem on multiple processors (e.g., GPUs in this case). It can be calculated as:

\[ S = \frac{T_s}{T_p} \] (14)

where \( T_s \) represents computing time on a single machine or GPU. \( T_p \) refers to the computing time on multiple machines or GPUs. The Accuracy (ACC) measures the classification accuracy and is defined as:

\[ ACC = \frac{(TP + TN)}{(TP + TN + FP + FN)} \] (15)

where TP = True positive, FP = False positive, TN = True negative and FN = False negative. Training Time (TT) is the time taken for training 3D-ResAttNet using the proposed approach and other existing distributed training methods.

B. SYSTEM CONFIGURATION
We have conducted our experiments on an Amazon Web Service (AWS) EC2 P3 instances. Specifically, we used a p3.16xlarge instance consisting of homogeneous 8 NVIDIA Tesla V100 GPUs developed purposely for the deep learning and Artificial intelligent crowd to provide ultra-fast GPU to GPU communication through NVLink technology. Other hardware configuration of the p3.16xlarge instance includes 128GB GPU memory, 64 vCPUs, 488GM memory, and 25Gbps network bandwidth. Additionally, software configuration /installation include: Ubuntu 18.04, Python 3.7.3, Pytorch 1.2.0, Torchvision 0.4.0, Numpy 1.15.4, Tensorboardx 1.4, Matplotlib 3.0.1, Tqdm 4.39.0, nibabel, fastai, and NVIDIA Collective Communications Library (NCCL) CUDA toolkit 10.2 - a library of multi-GPU collective communication primitives [41].

C. A USE CASE - PARALLELIZATION OF 3D-RESATTNET FOR ALZHEIMER’S DISEASE (AD) DIAGNOSIS
As described earlier, we have applied our hybrid parallelization approach to our previous non-parallel 3D-ResAttNet for automatic detection of the progression of AD and its Mild Cognitive Impairments (MCIs) such as Normal cohort (NC), Progressive MCL (pMCI) and Stable MCI (sMCI) from sMRI scans [42]. It includes two types of classification: NC vs. AD, and pMCI vs. sMCI.

1) The High-Level Parallelization Of The System
Fig. 2 shows the high level parallelization of our previous 3D-ResAttNet model architecture based on self-attention residual mechanism and explainable gradient-based localisation class activation mapping (Grad-CAM) to improve AD diagnosis performance. The 3D-ResAttNet model consists of 3D Convolutional blocks (Conv blocks), Residual self-attention block, and Explainable blocks. Conv blocks use a 3D filter for computation of the low-level feature representations. The residual network layer comprises two Conv blocks consisting of \( 3 \times 3 \times 3 \) 3D convolution layers, 3D batch normalization and rectified-linear-unit nonlinearity layer (ReLU). The explainable block uses 3D Grad-CAM to improve the model decision.

As shown in Fig. 2, the hybrid parallelization approach for 3D-ResAttNet is divided into three phases: the splitting of 3D-ResAttNet into partitions, allocation of GPUs to partitions, and data partitioning and distribution. For data parallelization, we adopt a stochastic gradient descent as well as ring All-reduce mechanisms for parameters updates and equally distribute data parts to each GPU. For model parallelization, the network model is partitioned based on...
TABLE 3: Parallel training performance of 3D-ResAttNet using our proposed approach

| #GPUs | 3D-ResAttNet18 | 3D-ResAttNet34 |
|-------|----------------|----------------|
|       | sMCI vs. pMCI  | AD vs. NC      | sMCI vs. pMCI  | AD vs. NC      |
| ACC   | (mins)         | ACC            | (mins)         | ACC            | (mins)         |
| 1     | 0.82           | 0.95           | 0.83           | 0.96           | 62             |
| 2     | 0.81           | 15             | 0.94           | 26             | 0.84           | 17             | 0.96           | 29             |
| 3     | 0.81           | 13             | 0.96           | 21             | 0.84           | 14             | 0.97           | 24             |
| 4     | 0.82           | 11             | 0.92           | 18             | 0.83           | 12             | 0.96           | 21             |
| 5     | 0.8            | 10             | 0.94           | 17             | 0.84           | 11             | 0.97           | 19             |
| 6     | 0.82           | 9              | 0.95           | 15             | 0.82           | 10             | 0.96           | 16             |
| 7     | 0.81           | 7              | 0.95           | 12             | 0.84           | 8              | 0.96           | 14             |
| 8     | 0.81           | 6              | 0.95           | 11             | 0.84           | 7              | 0.96           | 12             |

FIGURE 3: Speedup of our proposed approach

(a) sMCI vs. pMCI and AD vs. NC classification with respect to 3D-ResAttNet18

(b) sMCI vs. pMCI and AD vs. NC classification with respect to 3D-ResAttNet34

(c) 3D-ResAttNet18 vs. 3D-ResAttNet34 with respect to sMCI vs. pMCI

(d) 3D-ResAttNet18 vs. 3D-ResAttNet34 with respect to AD vs. NC

the computational complexity which usually synonymous to the number of basic operations, such as multiplications and summations, that each layer performs. Each Conv Block in the network consists $3 \times 3 \times 3$ 3D convolution layer, 3D batch normalization and rectified-linear-unit nonlinearity layer (ReLU). Moreover, a convolutional layer has higher operations with complexity $O(C_o \cdot C_1 \cdot T \cdot H \cdot W \cdot K_T \cdot K_H \cdot K_W)$ where $C_o$ and $C_1$ denote the number of output and input channels respectively, $T$, $H$ and $W$ are image dimension, and $K_T$, $K_H$ and $K_W$ are filter dimension. Consequently, we partitioned each Conv block individually as a single partition while other layers with less computation operations are partitioned as shown in Fig. 2.

2) Dataset Description

The dataset is obtained from the Alzheimer’s Disease Neuroimaging Initiative (ADNI) database (http://adni.loni.usc.8 VOLUME 4, 2016
FIGURE 4: Training time of proposed approach, FDG, DDG, DDP and DP

training using multiple GPUs on a single machine [43]. Both FDG and DDG were implemented by partitioning data and trained parallel models with sub-data across multiple GPUs.

In all experiments, we carried out several distributed training of 3D-ResAttNet18 and 3D-ResAttNet34 for two classification tasks: sMCI vs. pMCI and AD vs. NC with different number of GPUs (ranging from 1 to 8). Furthermore, we optimized model parameters with SGD, a stochastic optimization algorithm. We adopted other training parameters, including a batch size of six samples, cross-entropy as the loss function, and 50 epochs for better convergence. In addition, we set initial learning rate (LR) as $1 \times 10^{-4}$, then reduced by $1 \times 10^{-2}$ with increased iterations.

E. EXPERIMENTAL RESULTS AND DISCUSSION

1) Performance of 3D-ResAttNet in the parallel setting

We conducted the experiments on 3D-ResAttNet model for two classification tasks: sMCI vs. pMCI and AD vs. NC. The Tables 3 shows the experiment results of our parallel 3D-ResAttNet (with 18 and 34 layers respectively) in terms...
of both training time (TT) and accuracy. Based on it, we calculated the speedup, as shown in Figs. 3a, 3b, 3c, and 3d.

In all cases, it is observed that our proposed approach achieves almost linear speedup, which demonstrates the scalability of our approach in that the number of GPUs is directly proportional to the training speedup performance. For instance, in AD vs. NC classification task with 3D-ResAttNet34, the training speedup for 1, 2, 3, 4, 5, 6, 7, and 8 GPUs are 1, 2.38, 2.95, 3.44, 3.65, 4.13, 5.17, and 5.64 respectively. A similar trend is also observed in the sMCI vs. pMCI classification task with 3D-ResAttNet34, the training speedup for 1, 2, 3, 4, 5, 6, 7, and 8 GPUs are 1, 2.27, 2.62, 3.09, 3.40, 3.78, 4.86, and 5.67 respectively.

### TABLE 4: Accuracy comparison with the existing works

| References | Model       | Parallel Training | sMCI vs. pMCI | AD vs. NC |
|------------|-------------|-------------------|---------------|-----------|
| Hosseini-Asl et al [44] | CNN         | No                | N/A           | 97%       |
| Suk et al. [45] | DBM         | No                | 76%           | 95%       |
| Sarraf et al. [46] | CNN         | No                | N/A           | 96%       |
| Billones et al. [47] | CNN         | No                | N/A           | 91%       |
| Li et al. [48] | 3D CNN      | No                | 77%           | 91%       |
| Shi et al. [49] | MM-SDPN     | No                | 75%           | 95%       |
| Zhang et al. [42] | 3D-ResAttNet34 | No               | 84%           | 97%       |
| Our approach | 3D-ResAttNet34 | Yes              | 84%           | 97%       |

2) Comparison With The Existing Parallel Works

We compare our proposed approach with existing parallel and non-parallel methods regarding training time and accuracy, respectively, to affirm the robustness of the proposed method.

(i) Training speed: Our Proposed Approach And The Existing Parallel Approaches

We have compared our approach with DDP [43], DP [43], DDG [52] and FDG [51]. DDP and DP are two PyTorch generic distributed training methods. FDG and DDG are model parallelism approaches based on the delayed gradient method. The experiment results are shown in Figs. 4a, 4b, 4c, and 4d. It can be seen that our proposed approach outperforms the existing methods in terms of training time. For instance, for 3D-ResAttNet18 on AD vs. NC and sMCI vs. pMCI classification tasks, the training time incurred by our proposed approach is 20% averagely lower than DDP, DP, DDG and FDG. Similarly, there are related trends when comparing the proposed approach with the DDP, DP, DDG and FDG concerning the distributed training of 3D-ResAttNet34 for two classification tasks: AD vs. NC and sMCI vs. pMCI.

(ii) Accuracy: Comparison With The Existing Non-Parallel Works

Table 4 shows the accuracy comparison results from seven state-of-the-art deep neural networks and our methods. The best testing accuracies obtained in our approach are 97% and 84% for AD vs. NC and sMCI vs. pMCI classification respectively. The results show that our proposed approach performs efficiently when compared with the existing works in terms of accuracy. In addition, our work implements parallel distributed training of networks in a multi-GPU environment, whereas existing works are non-parallel methods.

5. CONCLUSION AND FUTURE WORK

In this work, we have proposed a hybrid parallelization approach that combines both model and data parallelization for parallel training of a DNN model. The Genetic Algorithm based heuristic resources allocation mechanism (GABRA) has also been developed for optimal distribution of network partitions on the available GPUs with the same or different capacities for performance optimization. Our proposed approach has been compared with the existing state-of-the-art parallel methods and evaluated with a real use case based on our previous 3D-ResAttNet model developed for efficient AD diagnosis. The experiment results show that the proposed approach achieves linear speedup, which demonstrates its scalability and efficient computing capability with little or no differences in accuracy performance (when compared with the existing non-parallel DNN models).

Future work will be focused on further improvement of parallelization approach for efficient training performance.

ACKNOWLEDGMENT

The work reported in this paper has formed part of the project by Royal Society - Academy of Medical Sciences Newton Advanced Fellowship (NAF/180371).

REFERENCES

[1] Y. Li, Y. Zhang and Z. Zhu, Error-Tolerant Deep Learning for Remote Sensing Image Scene Classification, in IEEE Transactions on Cybernetics, vol. 51, no. 4, pp. 1756–1768, April 2021, doi: 10.1109/TCYB.2020.2989241.

[2] B. J. Abbasiachian, D. Sierra-Sosa, A. Elmaghraby, Deep learning techniques for speech emotion recognition, from databases to models, Sensors, 2021, (4).

[3] M. Liu, J. Zhang, C. Lian and D. Shen, Weakly Supervised Deep Learning for Brain Disease Diagnosis Using MRI and Incomplete Clinical Scores, in IEEE Transactions on Cybernetics, vol. 50, no. 7, pp. 3381–3392, July 2020, doi: 10.1109/TCYB.2019.2904186.

[4] M. F. J. Acosta, L. Y. C. Tovar, M. B. Garcia-Zapirain, et al., Melanoma diagnosis using deep learning techniques on dermatoscopic images, BMC Med Imaging 2021 (6).

[5] M. Schell, Deep learning in music recommendation systems, Frontiers in Applied Mathematics and Statistics 5 (2019) 44.

[6] D. J. N. J. Soemers, V. Mella, C. Browne, O. Teytaud, Deep learning for general game playing with ludii and polygames, ArXiv, 2021, abs/2101.09562.

[7] H. Tembine, "Deep Learning Meets Game Theory: Bregman-Based Algorithms for Interactive Deep Generative Adversarial Networks," in IEEE Transactions on Cybernetics, vol. 50, no. 4, pp. 1132–1145, March 2020, doi: 10.1109/TCYB.2018.2868238.

[8] Michael Diskin and Alexey Bukhtiyarov and Max Ryabinin and Lucile Saulnier and Quentin Lhoest and Anton Sinitsin and Dmitry Popov and Dmitriy Pyrkin and Maxim Kashirin and Alexander Borzunov and Albert Villanova del Moral and Denis Mazur and Ilia Kobelev and Yacine Jernite and Thomas Wolf and Gennady Pekhimenko, Distributed Deep Learning In Open Collaborations, in: Advances in Neural Information Processing Systems, 2021.

[9] A. Križhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, Commun. ACM 60 (6) (2017) 84–90.
tions in the Cloud. Concurrency and Computation: Practice and Experience, 2018.

SAMSON B. AKINTOYE received the Ph.D. degree in Computer Science from University of the Western Cape, South Africa, 2019. He is currently working as a research associate in the Department of Computing and Mathematics, Manchester Metropolitan University, United Kingdom. His current research interests include parallel and distributed computing, deep learning, and cloud computing.

LIANGXIU HAN received the Ph.D. degree in computer science from Fudan University, Shanghai, China, in 2002. She is currently a Professor of computer science with Department of Computing and Mathematics, Manchester Metropolitan University. Her research areas mainly lie in the development of novel big data analytics and development of novel intelligent architectures that facilitates big data analytics (e.g., parallel and distributed computing, Cloud/Service-oriented computing/ data intensive computing) as well as applications in different domains using various large datasets (biomedical images, environmental sensor, network traffic data, web documents, etc.). She is currently a Principal Investigator or Co-PI on a number of research projects in the research areas mentioned above.

XIN ZHANG is associate researcher in Manchester Metropolitan University (MMU), he received the B.S degree from The PLA Academy of Communication and Commanding, China, in 2009 and Ph.D. degree in Cartography and Geographic Information System from Beijing Normal University(BNU), China, in 2014. His current research interests include remote sensing image processing and deep learning.

HAOMING CHEN is studying for a master’s degree in Computer Science and Artificial Intelligence in University of Sheffield. His current research interests include machine learning and Artificial Intelligence.

DAOQIANG ZHANG received the B.Sc. and Ph.D. degrees in computer science from Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 1999 and 2004, respectively. He is currently a Professor in the Department of Computer Science and Engineering, Nanjing University of Aeronautics and Astronautics. His current research interests include machine learning, pattern recognition, and biomedical image analysis. In these areas, he has authored or coauthored more than 100 technical papers in the refereed international journals and conference proceedings.

***