



Please cite the Published Version

Akintoye, Samson B, Han, Liangxiu , Zhang, Xin , Chen, Haoming and Zhang, Daoqiang (2022) A hybrid parallelization approach for distributed and scalable deep learning. IEEE Access, 10. pp. 77950-77961. ISSN 2169-3536

DOI: <https://doi.org/10.1109/access.2022.3193690>

Publisher: Institute of Electrical and Electronics Engineers (IEEE)

Version: Published Version

Downloaded from: <https://e-space.mmu.ac.uk/630155/>

Usage rights:  [Creative Commons: Attribution 4.0](https://creativecommons.org/licenses/by/4.0/)

Additional Information: This is an Open Access article which appeared in IEEE Access, published by IEEE

Enquiries:

If you have questions about this document, contact openresearch@mmu.ac.uk. Please include the URL of the record in e-space. If you believe that your, or a third party's rights have been compromised through this document please see our Take Down policy (available from <https://www.mmu.ac.uk/library/using-the-library/policies-and-guidelines>)

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

A Hybrid Parallelization Approach for Distributed and Scalable Deep Learning

SAMSON B. AKINTOYE¹, LIANGXIU HAN^{1*}, 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 computationally 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 synchronisation increases, 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. Similarly, 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 $2.2\times$ 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-mentioned 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

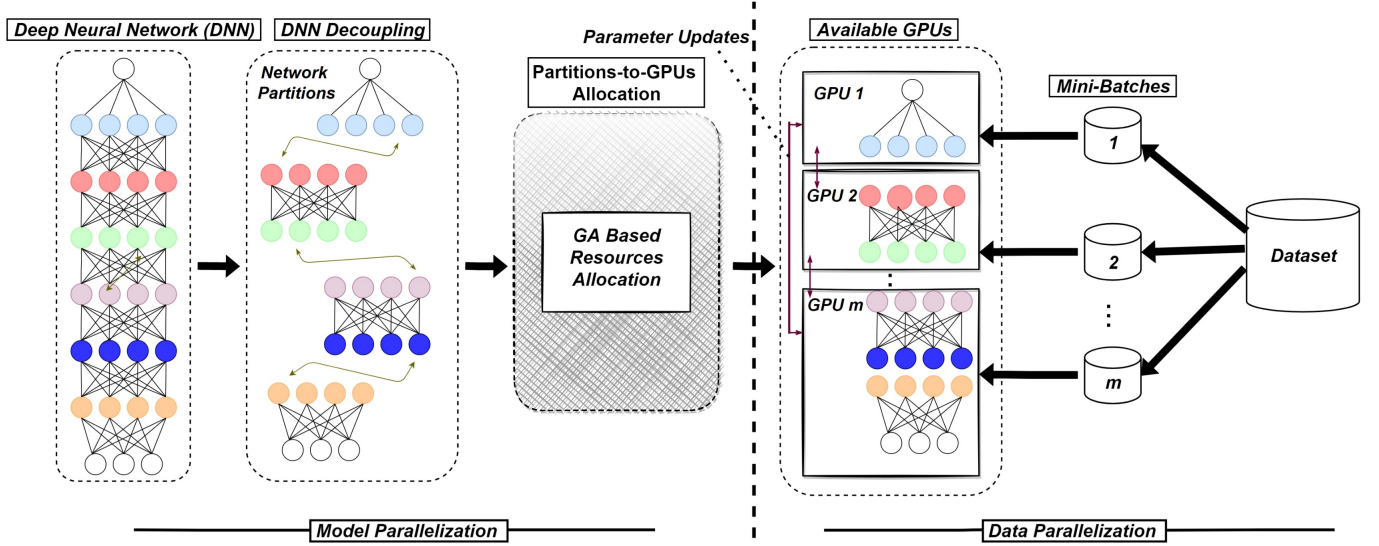


FIGURE 1: The high level architecture of the proposed hybrid parallelization approach

TABLE 1: Notations

Notations	Descriptions
Q	Number of network layers of a DNN model.
p_n	The network partitions of n size.
s_i	The i -th network layer in partition p_i .
q	A layer in a DNN model such that $q = 1, 2, \dots, Q$
d_m	The set of m GPUs.
t	Iteration.
a^t	Activation at iteration t .
w_q	The weight parameter of q layer.
b	The Batch size.
l	The Loss function.
γ_t	Learning rate at iteration t .
v	Data point.
V	Total data points of the dataset.
Ψ_c	The crossover operator.
Ψ_m	The mutation operator.
$g_{s(i)}^t$	The gradient at partition (i) at iteration t .

volume of weights and can be partitioned as a single partition for efficient parallel training performance.

Specifically, let's assume a model network contains a set of layers $\{s_1, s_2, \dots, s_Q\}$. The model network P is split into partitions $\{p_1, p_2, \dots, p_n\}$ where $p_i = \{s_i, s_i+1, \dots, s_{i+1}-1\}$, which denotes a set of layers in i partition such that $1 \leq i \leq n$. $s_i + 1$ and $s_{i+1}-1$ 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 input $a_{s_{i-1}}^t$ from partition $(i-1)$ is sent to partition i and gives activation $a_{s_{i+1}-1}^t$ at iteration t . Also, In backward

pass, the $g_{s(i+1)-1}^t$ denotes the gradient at partition $(i+1)$ at iteration t . For each layer $(s_i \leq q \leq s_{i+1}-1)$ such that $q \leq Q$, the gradient is given as: $\hat{g}_{w_q}^t = \frac{\delta a_{s(i+1)-1}^t}{\delta w_q^t} g_{s(i+1)-1}^t$ which can be updated by $w_q^{t+1} = w_q^t - \gamma_t \hat{g}_{w_q}^t$ where γ_t is learning rate.

$$\hat{g}_{w_q}^{t-i+1} = \frac{\delta a_{s(i+1)-1}^{t-i+1}}{\delta w_q^{t-i+1}} g_{s(i+1)-1}^{t-i+1} \quad (1)$$

which can be updated by:

$$w_q^{t-i+2} = w_q^{t-i+1} - \gamma_{t-i+1} \hat{g}_{w_q}^{t-i+1} \quad (2)$$

where γ_{t-i+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 find 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 $i = \{1, 2, \dots, n\}$. We also consider the capacity of a set of available GPU G , each denoted by d_j where $j = \{1, 2, \dots, m\}$ 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 :

$$c_{ij} = \frac{p_i}{d_j} \quad (3)$$

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

$$x_{ij} = \begin{cases} 1, & \text{if gpu } j \text{ is allocated to partition } i \\ 0, & \text{otherwise.} \end{cases} \quad (4)$$

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

$$\max_{p, x, c} z(X) = \sum_{i=1}^n \sum_{j=1}^m x_{ij} c_{ij} \quad (5)$$

subject to:

$$\sum_{i=1}^n p_i x_{ij} \leq d_j, \forall j \in M = \{1, 2, \dots, m\} \quad (6)$$

$$\sum_{j=1}^m x_{ij} = 1, \forall i \in N = \{1, 2, \dots, n\} \quad (7)$$

$$x_{ij} = 0 \text{ or } 1, \text{ for } i = 1, 2, \dots, n, \text{ and for } j = 1, 2, \dots, m \quad (8)$$

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 : $\{p_1, p_2, \dots, p_n\}$: computation loads of partitions
 $\{d_1, d_2, \dots, d_m\}$: capacity values of available GPUs
output: optimized solution $f(Z^*)$

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

Algorithm 2: initial population algorithm

input : $\{p_1, p_2, \dots, p_n\}$: computation loads of partitions
 $\{d_1, d_2, \dots, d_m\}$: capacity values of available GPUs
output: initial population

- 1 **for** (all partition loads) **do**
- 2 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:

$$f(\beta) = \sum_{i=1}^n c_{ij}\beta_i, \text{ and for } j = 1, 2, \dots, m \quad (9)$$

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 (ϕ), 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 Ψ_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 (Ψ_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 $cp \leftarrow \frac{Y_1}{2}$; //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 Ψ_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:

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

where f is a neural network, b is the batch size, m is the number of GPUs, ℓ 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:

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

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:

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

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

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

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:

$$S = T_s / T_p \quad (14)$$

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:

$$ACC = (TP + TN) / (TP + TN + FP + FN) \quad (15)$$

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, 488GB 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 MCI (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

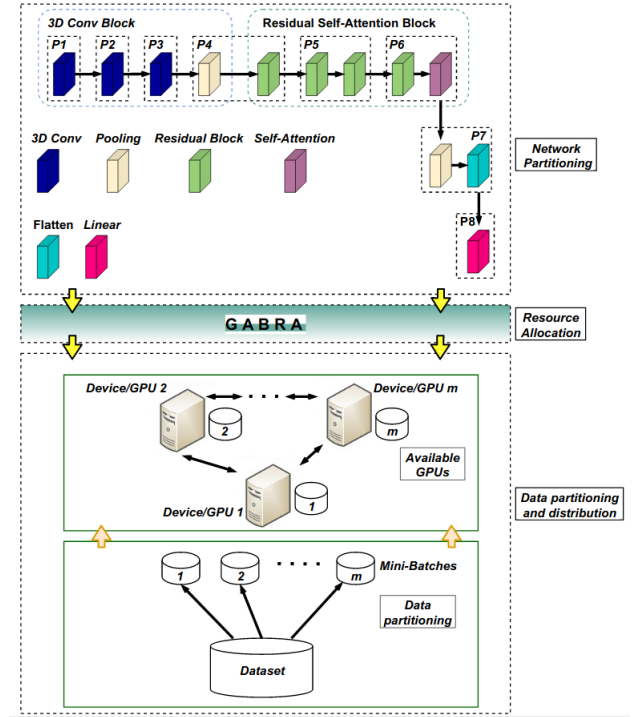


FIGURE 2: The hybrid parallelisation of 3D-ResAttNet

TABLE 2: Demographic data for the subjects from ADNI database

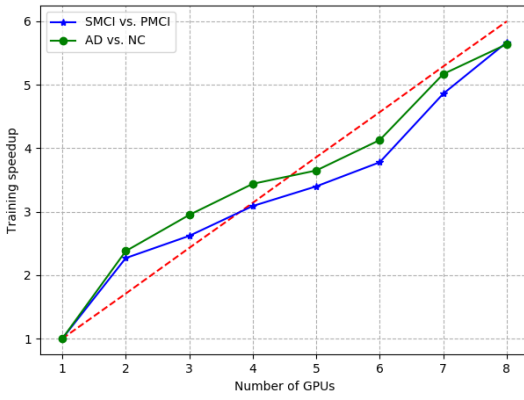
Class	Number/Size	Gender (Male/Female)	Age (Mean/Std)	MMSE (Mean/Std)
AD	389/1.4GB	202/187	75.95/7.53	23.28/2.03
pMCI	172/484MB	105/67	75.57/7.13	26.59/1.71
sMCI	232/649MB	155/77	75.71/7.87	27.27/1.78
NC	400/2.4GB	202/198	76.02/5.18	29.10/1.01

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 self-attention block combines two important network layers: Residual network layer and Self-attention layer. 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.

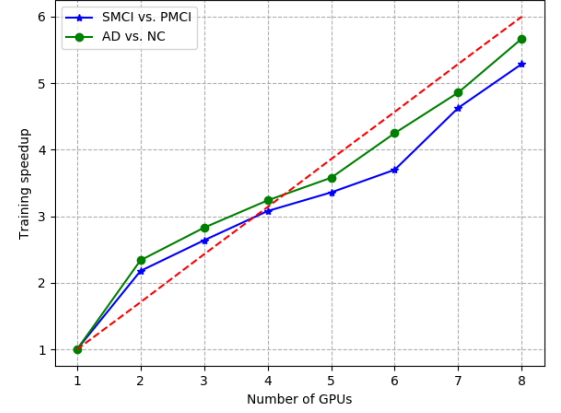
As shown in Fig. 2, the hybrid parallelization approach for 3D-ResAttNet is divided into three phases: the splitting of 3D-ResAttNet into partitions, allocation of GPUs to partitions, and data partitioning and distribution. For data parallelization, we adopt a stochastic gradient descent as well as ring 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

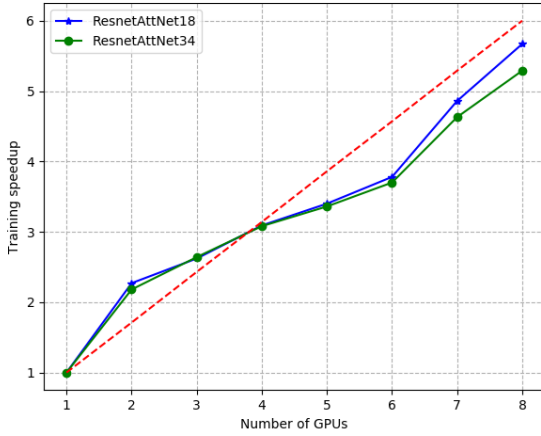
#GPUs	3D-ResAttNet18				3D-ResAttNet34			
	sMCI vs. pMCI		AD vs. NC		sMCI vs. pMCI		AD vs. NC	
	ACC	TT (mins)	ACC	TT (mins)	ACC	TT (mins)	ACC	TT (mins)
1	0.82	34	0.95	62	0.83	37	0.96	68
2	0.81	15	0.94	26	0.84	17	0.96	29
3	0.81	13	0.96	21	0.84	14	0.97	24
4	0.82	11	0.92	18	0.83	12	0.96	21
5	0.8	10	0.94	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



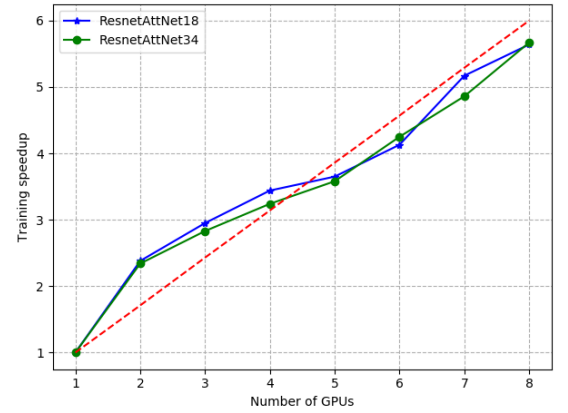
(a) sMCI vs. pMCI and AD vs. NC classification with respect to 3D-ResAttNet18



(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

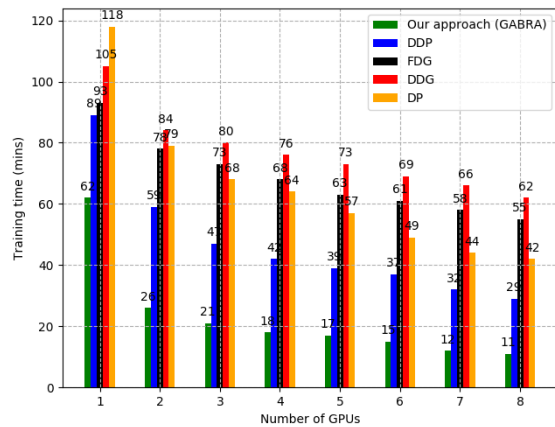
FIGURE 3: Speedup of our proposed approach

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 of $3 \times 3 \times 3$ 3D convolution layer, 3D batch normalization and rectified-linear-unit nonlinearity layer (ReLU). Moreover, a convolutional layer has higher operations with complexity $O(C_o \cdot C_i \cdot T \cdot H \cdot W \cdot K_T \cdot K_H \cdot K_W)$ where C_o and C_i denote the number of output and input

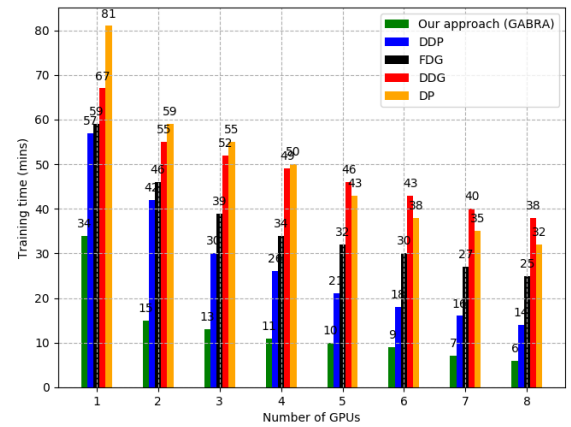
channels respectively, T , H and W are image dimension, and K_T , K_H and K_W are filter dimension. Consequently, we partitioned each Conv block individually as a single partition while other layers with less computation operations are partitioned as shown in Fig. 2.

2) Dataset Description

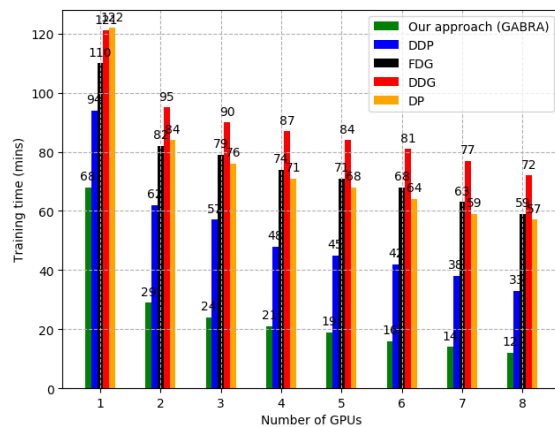
The dataset is obtained from the Alzheimer's Disease Neuroimaging Initiative (ADNI) database (<http://adni.loni.usc.edu>).



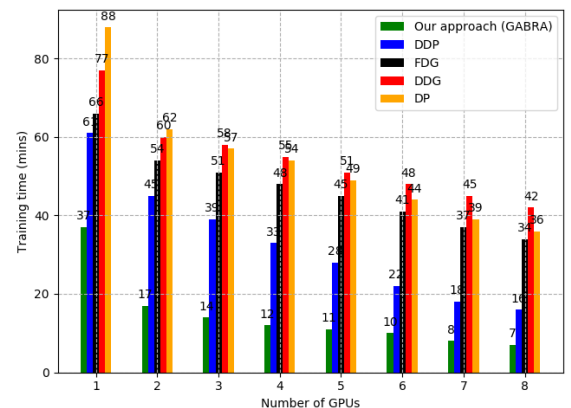
(a) Training Time of 3D-ResAttNet18 over AD vs. NC Dataset



(b) Training Time of 3D-ResAttNet18 over sMCI vs. pMCI Dataset



(c) Training Time of 3D-ResAttNet34 over AD vs. NC Dataset



(d) Training Time of 3D-ResAttNet34 over sMCI vs. pMCI Dataset

FIGURE 4: Training time of proposed approach, FDG, DDG, DDP and DP

edu), which is the same dataset previously used for validation of our 3D-ResAttNet. The dataset contains 1193 MRI scans of four classes: 389 Alzheimer's Disease (AD), 400 Normal Cohort (NC), 232 static mild cognitive impairment (sMCI) and 172 progressive mild cognitive impairment (pMCI) patients. The demographic data for this dataset is shown in Table 2.

D. EXPERIMENTS

We have conducted experiments under different strategies:

- 1) We have evaluated the model performance in the parallel setting across the number of heterogeneous GPUs.
- 2) We have compared our proposed approach with four existing data and model parallelism methods, including data parallelism - two PyTorch generic distributed training methods: *DistributedDataParallel* (DDP) and *DataParallel* (DP), and model parallelism - delayed gradient parallel methods: FDG and DDG for further evaluation. DDP is a multi-process data parallel training across GPUs either on a single machine or multiple machines, while DP is for single-process multi-parallel

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 1×10^{-4} , then reduced by 1×10^{-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.

V. 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\R1\180371).

REFERENCES

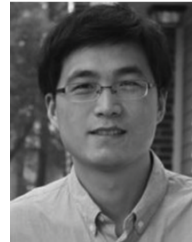
- [1] Y. Li, Y. Zhang and Z. Zhu, *Error-Tolerant Deep Learning for Remote Sensing Image Scene Classification*, in IEEE Transactions on Cybernetics, vol. 51, no. 4, pp. 1756–1768, April 2021, doi: 10.1109/TCYB.2020.2989241.
- [2] B. J. Abbaschian, D. Sierra-Sosa, A. Elmaghraby, *Deep learning techniques 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 Prognosis 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. Schedl, *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. 3, pp. 1132–1145, March 2020, doi: 10.1109/TCYB.2018.2886238.
- [8] Michael Diskin and Alexey Bukhtiyarov and Max Ryabinin and Lucile Saulnier and Quentin Lhoest and Anton Sinitits and Dmitry Popov and Dmitriy Pyrkun 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. Krizhevsky, I. Sutskever, G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, Commun. ACM 60 (6) (2017) 84–90.

- [10] J. George, P. Gurram, Distributed deep learning with event-triggered communication, ArXiv, 2019, abs/1909.05020.
- [11] S. Kim, G.-I. Yu, H. Park, S. Cho, E. Jeong, H. Ha, S. Lee, J. S. Jeong, Byung-Gon Chun, Parallax: Sparsity-aware data parallel training of deep neural networks, in: Fourteenth EuroSys Conference 2019 (EuroSys19), Dresden, Germany., 2019, p. 15.
- [12] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, Z. Zhang, Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems, ArXiv, 2015, abs/1512.01274.
- [13] PyTorch, Pytorch deep learning framework that puts python first, <http://pytorch.org/>, 2020 (accessed 16 Dec. 2020).
- [14] Zhen Song, Yu Gu, Zhigang Wang, Ge Yu. DRPS: efficient disk-resident parameter servers for distributed machine learning. Front. Comput. Sci. 16, 164321 (2022).
- [15] J. Ono, M. Utiyama, E. Sumita, Hybrid data-model parallel training for sequence-to-sequence recurrent neural network machine translation, ArXiv, 2019, abs/1909.00562.
- [16] Swapnil Gandhi and Anand Padmanabha Iyer, P3: Distributed Deep Graph Learning at Scale, in: Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation, OSDI'21, July, 2021, pp. 551–568.
- [17] A. Mirhoseini, H. Pham, Q. V. Le, B. Steiner, R. Larsen, Y. Zhou, N. Kumar, M. Norouzi, S. Bengio, J. Dean, Device placement optimization with reinforcement learning, in: ICML, 2017.
- [18] M. Wang, C. Huang, J. Li, Unifying data, model and hybrid parallelism in deep learning via tensor tiling, ArXiv, 2018, abs/1805.04170.
- [19] L. Song, J. Mao, Y. Zhuo, X. Qian, H. Li, Y. Chen, HyPar: Towards hybrid parallelism for deep learning accelerator array, 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA), 2019.
- [20] A. Sergeev, M. D. Balso, Horovod: fast and easy distributed deep learning in tensorflow, ArXiv, 2018, abs/1802.05799.
- [21] Z. Jia, M. Zaharia, A. Aiken, Beyond data and model parallelism for deep neural networks, ArXiv, 2019, abs/1807.05358.
- [22] A. Mirhoseini, A. Goldie, H. Pham, B. Steiner, Q. V. Le, J. Dean, A hierarchical model for device placement, in: International Conference on Learning Representations, 2018.
- [23] O. Yadan, K. Adams, Y. Taigman, M. Ranzato, Multi-gpu training of convnets, CoRR, 2014, abs/1312.5853.
- [24] Y. Huang, Y. Cheng, D. Chen, H. Lee, J. Ngiam, Q. V. Le, Z. Chen, Gpipe: Efficient training of giant neural networks using pipeline parallelism, ArXiv, 2019 abs/1811.06965.
- [25] N. Shazeer, Y. Cheng, N. Parmar, D. Tran, A. Vaswani, P. Koanantakool, P. Hawkins, H. Lee, M. Hong, C. Young, R. Sepassi, B. A. Hechtman, Mesh-tensorflow: Deep learning for supercomputers, in: NeurIPS, 2018.
- [26] G. Onoufriou, R. Bickerton, S. Pearson, G. Leontidis, Nemesys: A hybrid parallelism deep learning-based framework applied for internet of things enabled food retailing refrigeration systems, Comput. Ind. 113, 2019.
- [27] Y. Oyama, N. Maruyama, N. Dryden, E. McCarthy, P. Harrington, J. Balewski, S. Matsuoka, P. Nugent, B. Van Essen, The case for strong scaling in deep learning: Training large 3d cnns with hybrid parallelism, IEEE Transactions on Parallel and Distributed Systems 32 (7) (2021) 1641–1652.
- [28] Z. Wesooowski, Network resource allocation in distributed systems: A global optimization framework, in: 2015 IEEE 2nd International Conference on Cybernetics (CYBCONF), 2015, pp. 267–270.
- [29] A. Velarde Martinez, Scheduling in heterogeneous distributed computing systems based on internal structure of parallel tasks graphs with meta-heuristics, Applied , 10 (18), 2020.
- [30] L. Haji, S. Zeebaree, O. Ahmed, A. Sallow, K. Jacksi, R. Zebari, Dynamic resource allocation for distributed systems and cloud computing, Test Engineering and Management 83 (2020) 22417–2242.
- [31] M. Zhang, L. Liu, S. Liu, Genetic algorithm based qos-aware service composition in multi-cloud, in: 2015 IEEE Conference on Collaboration and Internet Computing (CIC), 2015, pp. 113–118.
- [32] K. Gai, M. Qiu, H. Zhao, Cost-aware multimedia data allocation for heterogeneous memory using genetic algorithm in cloud computing, IEEE Transactions on Cloud Computing (2016) 1–1.
- [33] C. Mezache, O. Kazar, S. Bouekkache, A genetic algorithm for resource allocation with energy constraint in cloud computing, in: Proceedings of the International Conference on Image Processing, Production and Computer Science (ICIPCS-2016), 2016, pp. 62–69.
- [34] H. Jiang, J. Yi, S. Chen, X. Zhu, A multi-objective algorithm for task scheduling and resource allocation in cloud-based disassembly, Journal of Manufacturing Systems 41 (2016) 239–255.
- [35] A. Mosa, R. Sakellariou, Dynamic virtual machine placement considering cpu and memory resource requirements, in: 2019 IEEE 12th International Conference on Cloud Computing (CLOUD), 2019, pp. 196–198.
- [36] P. Devarasetty, S. Reddy, Genetic algorithm for quality of service based resource allocation in cloud computing, Evolutionary Intelligence, 2019.
- [37] S. H. da Mata, P. R. Guardieiro, A genetic algorithm based approach for resource allocation in lte uplink, in: 2014 International Telecommunications Symposium (ITS), 2014, pp. 1–5.
- [38] Z. Li, Q. Zhu, Genetic algorithm-based optimization of offloading and resource allocation in mobile-edge computing, Information 11 (2), 2020.
- [39] T. Perry, M. Bader-El-Den, S. Cooper, Imbalanced classification using genetically optimized cost sensitive classifiers, 2015 IEEE Congress on Evolutionary Computation (CEC) (2015) 680–687.
- [40] Alexander Sergeev and Mike Del Balso, Horovod: fast and easy distributed deep learning in TensorFlow, ArXiv abs/1802.05799 (2018)
- [41] Nvidia, Nccl, <https://docs.nvidia.com/deeplearning/nccl/install-guide/index.html>, 2021, (accessed 23 Jan. 2021).
- [42] X. Zhang, L. Han, W. Zhu, L. Sun, D. Zhang, An explainable 3d residual self-attention deep neural network for joint atrophy localization and alzheimer's disease diagnosis using structural mri, ArXiv, 2020, abs/2008.04024.
- [43] S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania, S. Chintala, Pytorch distributed: Experiences on accelerating data parallel training, Proc. VLDB Endow. 13 (2020) 3005–3018.
- [44] . Hosseini-Asl, R. Keynton, A. El-Baz, Alzheimer's disease diagnostics by adaptation of 3d convolutional network, 2016 IEEE International Conference on Image Processing (ICIP) (2016) 126–130.
- [45] H. Suk, D. Shen, Deep learning-based feature representation for ad/mci classification, Medical image computing and computer-assisted intervention : MICCAI ... International Conference on Medical Image Computing and Computer-Assisted Intervention 16 Pt 2 (2013) 583–90.
- [46] S. Sarraf, G. Tofghi, Classification of alzheimer's disease using fmri data and deep learning convolutional neural networks, ArXiv, 2016, abs/1603.08631.
- [47] C. D. Billones, O. J. L. D. Demetria, D. E. D. Hostallero, P. C. Naval, Demnet: A convolutional neural network for the detection of alzheimer's disease and mild cognitive impairment, in: 2016 IEEE Region 10 Conference (TENCON), 2016, pp. 3724–3727.
- [48] H. Li, M. Habes, Y. Fan, Deep ordinal ranking for multi-category diagnosis of alzheimer's disease using hippocampal mri data, ArXiv, 2017, abs/1709.01599.
- [49] J. Shi, X. Zheng, Y. Li, Q. Zhang, S. Ying, Multimodal neuroimaging feature learning with multimodal stacked deep polynomial networks for diagnosis of alzheimer's disease, IEEE Journal of Biomedical and Health Informatics PP (2017) 1–1.
- [50] Fengrui Yu and Xueliang Fu and Honghui Li and Gaifang Dong, Improved Fitness Proportionate Selection-Based Genetic Algorithm, Proceedings of the 2016 3rd International Conference on Mechatronics and Information Technology, 2016, pp. 136–140.
- [51] Z. Huo and B. Gu and Q. Yang and H. Huang, Decoupled parallel backpropagation with convergence guarantee. In ICML, 2018.
- [52] H. Zhuang and Y. Wang and Q. Liu and Z. Lin, Fully decoupled neural network learning using delayed gradients. IEEE transactions on neural networks and learning systems, PP, 2021.
- [53] D. Narayanan and A. Harlap and A. Phanishayee and V. Seshadri and N. R. Devanur and G. R. Ganger and P. B. Gibbons and M. Zaharia, Pipedream: Generalized pipeline parallelism for dnn training. In Proceedings of the 27th ACM Symposium on Operating Systems Principles, pages 1–15, New York, NY, USA, 2019. Association for Computing Machinery.
- [54] S. Lee and D. Jha and A. Agrawal and A. Choudhary and W. Liao, Parallel deep convolutional neural network training by exploiting the overlapping of computation and communication. In 2017 IEEE 24th International Conference on High Performance Computing (HiPC), pages 183–192, Jaipur, 2017.
- [55] C.-C. Chen and C.-L. Yang and H.-Y. Cheng, Efficient and robust parallel dnn training through model parallelism on multi-gpu platform. ArXiv, abs/1809.02839, 2018.
- [56] C Kim and H Lee and M Jeong and W Baek and B Yoon and I Kim and S Lim and S Kim. torchpipe: On-the-fly pipeline parallelism for training giant models. arXiv preprint arXiv:2004.09910. 2020.
- [57] S. Li, Z. Huang and L. Han. A Genetic Algorithm Enhanced Automatic Data Flow Management Solution for Facilitating Data Intensive Applica-

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.



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.

...



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.