The Journal of Supercomputing (2022) 78:13455–13469
https://doi.org/10.1007/s11227-022-04399-2

Heterogeneous gradient computing optimization for scalable deep neural networks

Sergio Moreno-Álvarez[1] · Mercedes E. Paoletti[2] · Juan A. Rico-Gallego[1] · Juan M. Haut[3]

Accepted: 22 February 2022 / Published online: 19 March 2022

Abstract

Nowadays, data processing applications based on neural networks cope with the growth in the amount of data to be processed and with the increase in both the depth and complexity of the neural networks architectures, and hence in the number of parameters to be learned. High-performance computing platforms are provided with fast computing resources, including multi-core processors and graphical processing units, to manage such computational burden of deep neural network applications. A common optimization technique is to distribute the workload between the processes deployed on the resources of the platform. This approach is known as data-parallelism. Each process, known as replica, trains its own copy of the model on a disjoint data partition. Nevertheless, the heterogeneity of the computational resources composing the platform requires to unevenly distribute the workload between the replicas according to its computational capabilities, to optimize the overall execution performance. Since the amount of data to be processed is different in each replica, the influence of the gradients computed by the replicas in the global parameter updating should be different. This work proposes a modification of the gradient computation method that considers the different speeds of the replicas, and hence, its amount of data assigned. The experimental results have been conducted on heterogeneous high-performance computing platforms for a wide range of models and datasets, showing an improvement in the final accuracy with respect to current techniques, with a comparable performance.

Keywords Deep learning · Deep neural networks · High-performance computing · Heterogeneous platforms · Distributed training

✉ Sergio Moreno-Álvarez
smoreno@unex.es

[1] Department of Computer Systems Engineering and Telematics, University of Extremadura, Cáceres, Spain

[2] Department of Computer Architecture, Complutense University of Madrid, Madrid, Spain

[3] Department of Technology of Computers and Communications, University of Extremadura, Cáceres, Spain

1 Introduction

Deep neural networks (DNNs) architectures [15, 29] have leveraged important advances in research fields as image processing [12] and classification [31], natural language processing [24], medicine [19] and satellite observational studies [11], among others. DNNs are universal function approximators composed of neurons disposed in stacked layers. DNNs have the ability to learn relationships between input data as they propagate through the layers of the network to the output expected results. Each layer computes intermediate features using a set of internal trainable parameters, which are learned based on gradient optimization techniques, as stochastic gradient descent.

Taking the example of a supervised image classification algorithm, the optimization procedure works as follows. The objective of the supervised-based approaches is to learn a mapping function $f(\cdot, \Theta)$ of the images in the input dataset $\mathbb{X}$ to the corresponding labels $y_i \in \mathbb{Y}$. Input dataset is composed of examples $x_i \in \mathbb{R}^{h \times w \times c}$, where h is the height, w the width and c the number of channels of the images. By applying the map function f, each input image $x_i \in \mathbb{X}$ is assigned with an expected label $\hat{y}_i \in \mathbb{Y}$. The distance between the expected ($\hat{y}$) and actual (y) labels is measured by a *loss* function $\mathcal{L}(\Theta)$, and then it is used to adjust the model parameters Θ. The common method to update the network parameters reducing that distance is *gradient descent*, which computes the new values of the parameters using the gradient of the loss function as $\Theta' = \Theta - \alpha\, g$, where α is the *learning rate*, Θ' and Θ the current and previous parameter values, respectively, and g is the gradient vector of the loss function, computed as $g = \nabla_\Theta \mathcal{L}(\Theta)$.

Current large-scale DNNs architectures trained on large datasets and with a huge number of parameters have increased models training complexity [6]. Thus, the improvement of the performance and the accuracy of the models have become major challenges in this area. High-performance computing (HPC) platforms composed of resources with different computational capabilities (e.g., multi-core CPUs or GPUs) have addressed these challenges. Two main schemes are used to train DNN models in HPC platforms: *model-parallelism* and *data-parallelism*. Model parallelism scheme splits the model between the available computational resources, and each part is trained using the entire set of data examples. Conversely, data parallelism distributes disjoint partitions of the dataset between the processes running on the computational resources, known as *replicas*, training a copy of the model.

This work focuses on the data parallelism scheme. Each data partition $X_p \in \mathbb{X}$ assigned to a replica $p \in P$ is composed of subsets of data examples called *batches*. A batch passes through the model layers to perform a training iteration. Then, replicas compute the gradient vector and communicate each other to update the global parameter values Θ'. Parameter values are stored in centralized parameter servers, or distributed using collective communications [25]. An *epoch* is a complete pass of the batches in a data partition.

The heterogeneity of in HPC platforms is an issue that should be considered in order to efficiently train models under the data-parallelism scheme. The

resources that compose the platforms have different computational capabilities, which lead to uneven replica training times. The consequence is that assuming collective communication at the communication stage, faster processes wait for slower processes, highly degrading performance. Parameter servers together with the asynchronous SGD (ASGD) [9] technique partially solve the performance problem of processes synchronization at the communication stages. Nevertheless, as the training progresses, faster replicas will compute their gradients using an obsolete version of the global parameters. The distance between local parameters employed to compute gradients in a replica and their global values is called *staleness*. While staleness reduces the performance impact of the communication in the overall training performance, the fact is that it has an impact in the final accuracy of the model.

Our approach is to unevenly distribute the data between the replicas according to its relative computational speed [22]. This mechanism ensures similar training times in each replica, and hence, it diminishes the waiting times at communication stages, and furthermore, it avoids staleness. Each replica p is assigned with a data partition X_p proportional to its computational speed. We establish a global batch size B as an hyperparameter to the training. The size of the batch (number of examples) feeding the model in each replica is computed as $b_p = B \times s_p$, hence, proportional to the relative speed of the replica s_p. The total size of the partition assigned to each replica is $X_p = X \times s_p$. Note that the number of iterations per epoch in each replica is constant, and equal to X_p/b_p. A decisive step is to figure out the relative speeds of the replicas s_p in the platform. As example, the FuPerMod tool [8] represents the performance profile of P processes as a vector $S = \{s_1, s_2, \ldots, s_P\}$, with s_p the inverse of the time spent in the execution of a benchmark provided by the user. The benchmark should reflect the computation performed in the actual training code, and hence it is usually a subset of the entire model.

While former approach have demonstrated to improve performance of training models in heterogeneous HPC platforms, the fact is that replicas use a different amount of data to train its local models, and compute the gradients based on different feature information. Thus, replicas should contribute unevenly to the parameters updating. A main contribution of this work is to introduce a methodology to improve the accuracy of the models based on weighted gradient computations, according to the speeds of the replicas involved in the training process. Assuming a dedicated heterogeneous HPC platform, the overall methodology is as follows. Data are distributed between replicas according to their relative speeds. Such speeds are figured out in a previous stage to the training and the amount of data assigned to each replica is computed. Then, replicas train its copies of the model on assigned data avoiding waiting times at communication stages. Collective communication operations are used to share the gradients. Parameter updating is achieved using weighted gradient computations, ensuring that each replica have an proportional impact on the weights depending on its speed.

As a summary, the proposed work implements the following solutions to the described problems.

- To speed up the computation phase, the distribution of the workload has been measured based on the speeds of each resource using popular methods available in the literature.
- To improve the accuracy of the model, a methodology is proposed to ensure that the training information obtained in each replica has an impact relative to the assigned data workload. In this way, it is controlled that the replicas with low amount of data does not interfere negatively in the steps of training and weight updating.
- In order to evaluate the proposed method, two heterogeneous platforms composed of multicore CPUs and GPUs have been tested. We compare the method with a baseline unweighted gradient computation using several scientific datasets, as CIFAR-10, CIFAR-100 and Mini-ImageNet. The proposed method obtains a higher accuracy than the baseline approach with a comparable performance for a variety of simple and deeper networks.

The organization of the paper is as follows. Section 2 explores related approaches proposed in the literature. Section 3 describes the proposed weighted gradient methodology. Section 4 evaluates the method in different platforms and datasets, and analyze the obtained results showing the viability of the proposal. Finally, Sect. 5 concludes.

2 Related work

The convergence of high-performance computing and deep learning model training, together with the development of parallelization techniques, has been studied in multiple works [16, 32]. In [18], both model and data parallelism techniques are proposed as main schemes of parallelization for deep neural networks on HPC platforms. A thorough study and evaluation of the such parallelization techniques is performed in [2]. Model parallelism enhancements have been proposed for particular platforms in the literature [21]. Meanwhile, data parallelism have gained interest in different scientific research fields due to the performance improvements and its ease of use for different research fields. As examples, the work [11] enumerates the possibilities of the distributed data parallelism schemes over HSI image classification for different synchronous or asynchronous approaches, and the work [23] evaluates the performance of the data parallel approach in pattern recognition applications. HSI classification methods has been greatly enhanced due to the significant improvements in image processing and analysis techniques. Nevertheless, these techniques should be efficiently parallelized in order to optimize their performance. For instance, Danfeng et al. [13] proposed a multimodal deep neural architecture for feature representation learning using fully connected (FCs) and convolutional (CNNs) networks to extract relevant pixel-wise and spatial information, respectively. Meanwhile, the work proposed by Danfeng et al. [14] extracts detailed spectral representations using a sequential network to learn group-wise spectral information of the different HSI materials.

Particular aspects of the data parallelism scheme have been addressed in several contributions. The work [26] evaluates the impact on the distributed training for different hyperparameter configurations. The work [17] proposes the *Stale Synchronous Parallel* method to address the problem of the staleness on homogeneous data distributions using stochastic gradient descent (SGD) as optimizer. In addition, the staleness issue has been addressed in other works. The work [20] proposes an asynchronous SGD (ASGD) approach to reduce the variance of the gradient values during the optimization process, while [10] proposes to modify the learning rate depending on the current staleness value for ASGD. In the former work, a discussion is performed about the feasibility of modifying different hyperparameters of the model and its impact on the training. Authors in [7] propose to avoid excessive staleness impact on the model accuracy by benefiting the gradients from the fastest processes, discarding slower processes ones. Nevertheless, this approach causes a significant loss of information from slower processes.

Regarding workload distribution in data-parallelism scheme, a dynamic workload distribution scheme is proposed in [5], to adapt the assigned batch size to each replica in every iteration. A recurrent neural network (RNN) is used in order to measure the speed of each replica. On the other hand, static load balancing approaches are used to avoid the impact of the speed calculation time in every training iteration. In this sense, the work [22] performs a unique workload distribution computation in a prior stage to the training stage.

Notwithstanding heterogeneous data partitioning improves training performance on heterogeneous platforms, its impact on the accuracy of the final model is not addressed extensively in the literature. The work [30] proposes a method called TernGrad based on scaling down the gradient norms in order to improve the convergence and speed up of training. Method convergence and performance have been evaluated using SGD under a parameter localization scheme. In this scheme, workers holds its own copy of the parameters locally. Additionally, it proposes the use of ternary values to reduce communication. Ternary values quantifies gradients (32-bits) before being communicated using a ternary vector with values $\in \{-1, 0, 1\}$ that reduces the communication volume by a 'x' factor of $x = 32/\log_2(3)$.

The reduction in the communications is studied in [28] using different gradient quantization techniques in the loss calculation step for the MNIST and CIFAR-10 datasets. The work [30] improves former quantization in each computation element for more complex Imagenet dataset. The relationship between the quantization of the gradients and the accuracy obtained in the model is evaluated in [1].

In brief, most of the works refer to the management of the gradient focusing on performance. Since the performance problem has been solved using computational balancing techniques, weighting methods gained importance to improve accuracy, by computing model parameters based on the replicas gradient directions. The importance of weighting methods for DNNs is deeply studied in the literature demonstrating how these methods influence the convergence and network predictions [3]. Additionally, weighted methods are also used to differently assigning a weight to each dataset class [4].

In this sense, our proposal addresses data-parallelism models training on heterogeneous data partitions. Gradient contributions from each replica to the global

parameter computations are weighted using the speed of each replica. Therefore, our objective in terms of performance and specially final accuracy is to outperform current baseline method with an unweighted gradient computation.

3 *Hetgrad* optimization methodology

This section details the methodology followed to solve the described problems derived from the pointed challenges. In this context, the implemented method proposes a static mapping technique to distribute the data among the available resources prior to the execution of the deep neural model, seeking for the static workload balancing and introducing a new loss balancing to compensate the heterogeneous partitioning previously performed. The introduced balanced (or weighted) gradient calculation considerably improves the convergence of the neural model during the training phase, which conducts a more accurate parameter tuning, leading to a better accuracy.

Following previous research [22], the strategy for the performance optimization of deep networks has been based primarily on data parallelization approaches, where a replica of the full model is loaded and run by each computational resource using non-overlapped data subsets. While in those clusters with homogeneous resources, data partitioning is done directly, giving the same partition size to each resource, in heterogeneous clusters, a fair partitioning must be done. Indeed, at equal partition, differences in the computational capacities of the resources on current heterogeneous platforms can induce long waiting times when synchronously combining gradients, leading to a degradation of overall performance. In order to overcome this drawback, the data partitions have to balance the load between the different resources according to their computational capacities (particularly in proportion to the speed of the computing device), minimizing the computational time.

Emphasizing to divide the data into fair partitions that balance the workload of the heterogeneous cluster, a prior analysis of the computational features of the cluster should be conducted in order to determine the amount of data (i.e., the percentage of data) that each resource is capable of handling. Several techniques can be taken into account, such as the widely used FuPerMod framework [8]. This general-purpose data partitioning framework performs accurate and efficient benchmarking to obtain the *relative speed* of the resources that constitute the cluster, providing the load measurements for each element that optimize execution time. Also, other criteria could be taken into account to determine these measurements [5]. In this particular case, the resource speed is used to define the heterogeneity of the platform, as explained in the following Sect. 4.3 and implemented in the work [22]. As a result, a vector of measurements is obtained $\mathbf{c} = \{c_1, c_2, \ldots, c_D\}$, which is normalized taking into account the slowest device or the device with the most limited computational capability, providing a metric appropriate to the target problem. Indeed, the normalized vector $\mathbf{c}$ is providing unit of capability. Based on these measurements, the data partitioning is conducted, splitting the training data N into non-overlapped partitions with different sizes $P_1, P_2, \ldots, P_D$. To accomplish this, N is divided between the sum of units of capability, obtaining the number of samples per unit of capability

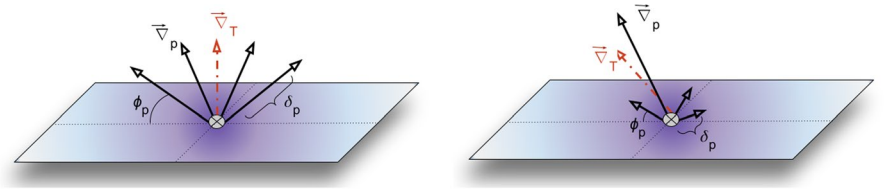
$\tilde{N} = N/\sum_{i=1}^{D} c_i$. Then, the i-th partition is obtained as $P_i = \tilde{N} \cdot c_i$. If P_i is not an integer, the data is rounded down, which may appear to disregard the remaining samples. Nevertheless, with each gradient update, data are shuffled and the partitions are refilled, thus all samples are processed at the end.

At this point, it is important to decide on the synchronization strategy. Indeed, deep neural networks optimize their parameters through the back-propagation of the gradient signal within an iterative process guided by the optimization algorithm. In particular, they conduct several training epochs, where in each epoch, the optimization algorithm pass all the training data through the network. To avoid complications in model convergence, the data passing is done iterative, grouping the training data into non-overlapped batches of identical size B that are processed separately by the network. As a result, to complete an epoch, all batches must have been processed by the neural model, so each epoch is defined by a series of iterations determined by $M = N/B$, where N indicates the number of training samples, B the batch size and M the number of iterations/batches. By having several distributed replicas, the gradients obtained by each replica must be combined in order to compose a global gradient that can be transmitted to the replicas to update their parameters. Although asynchronous communication of gradients has been proposed, it its hampered by the so-called staleness problem, complicating also the control mechanisms to obtain a suitable global gradient. In this sense, the proposed method considers a synchronized communication between the working nodes. This introduces a slight constraint when organizing the data in each partition. In fact, the number of batches required to complete an epoch M must be the same on all devices to ensure the synchronization. Thus, for each partition P_i, a batch size $B_i = P_i/M$ is set, $\forall i \in [1, D]$. Furthermore, the set of all batches in each partition must cover the total training data. This is done by introducing a *global batch size* G, which acts mimicking the total number of training data N. While setting the size of each partition is done by $P_i = \tilde{N} \cdot c_i$, batch sizes are obtained as $B_i = \tilde{G} \cdot c_i$, where $\tilde{G} = G/\sum_{i=1}^{D} c_i$ is the number of batches per unit of capability. Once again, if B_i is not an integer, is rounded to obtain the desired M iterations.

Once parameters $\{P_1, P_2, \ldots, P_D\}$, M and $\{B_1, B_2, \ldots, B_D\}$ have been determined, replicas are trained for M iterations, using completely different training samples. This avoids the same data being involved several times in the same parameters update. Furthermore, data shuffling is implemented, ensuring that all replicas are trained with the entire dataset. Focusing on the training stage, each replica performs the calculation of its gradient as a function of the loss (i.e., classification error) obtained by the model. In this regard, at the k-th iteration, the i-th replica obtains the loss of its batch $\mathbf{B}[k]$ according to its current parameters $\boldsymbol{\theta}_i^k$. As a result, each replica obtains its gradient vector $\mathcal{L}^k(\boldsymbol{\theta})_i$ and communicates it through Open-MPI *allReduce* collectives so that the gradients are combined as shown in Eq. (1).

$$\mathcal{L}^k(\boldsymbol{\theta})_{\text{global}} = \frac{1}{D} \sum_{i=1}^{D} \mathcal{L}^k(\boldsymbol{\theta})_i \tag{1}$$

Once the gradient combination is done, $\mathcal{L}^k(\boldsymbol{\theta})_{\text{global}}$ is sent to the replicas, which accordingly update their parameters using the defined learning rate. In this context,



(a) Baseline gradient calculation. **(b)** Het Grad gradient calculation.

Fig. 1 Gradient hyper-plane. Red arrow represent the gradient direction of the obtained total gradient $\overrightarrow{\nabla}_T$ after sharing replica gradients $\overrightarrow{\nabla}_p$. The weighted gradient for each process is represented as δ_p. Angle direction of the gradients is represented as ϕ_p (color figure online)

Eq. (1) considers the contributions of the replicas at the same level, giving them equal weight in the final calculation of the global gradient as shown in Fig. 1a. As a result, the global gradient has been obtained disregarding the amount of data involved in the calculation of the local gradients. Nevertheless, this is not fair as each replica employs a different partition size. Therefore, replicas with less data contribute the same as replicas with more data, potentially hampering the gradient calculation.

To properly avoid this problem, the proposed method introduces the weighted gradient calculation, which aligns the contribution of each replica to the partition size it receives. This adjustment is performed during the calculation of the global gradient as Eq. (2) indicates:

$$\mathcal{L}^k(\boldsymbol{\theta})_{\text{global}} = \sum_{i=1}^{D} \mathcal{L}^k(\boldsymbol{\theta})_i \cdot c_i \tag{2}$$

Therefore, the gradients of each partition are scaled according to the computational capability of the device, as shown in Fig. 1b. This provides more weight to those gradients calculated with a larger number of samples, which are more robust and accurate than those calculated in smaller partitions, indeed. Consequently, the global gradient will take a more appropriate direction, avoiding local minima and stagnation introduced by small partitions, resulting in a better parameter fit and in an improved classification result. As a consequence, the floating points operations are defined as:

$$\text{FLOPS} = \left(\sum_{l=0}^{\text{Layers}} B \times k^2 \times \text{Dim} \right) + \sum_{g=0}^{G} 1, \tag{3}$$

where Dim defines the (height $\times$ width $\times$ filters) data at the input of each layer l, k is the kernel size of the convolutions and G is the size of the gradient vector $\mathcal{L}^k(\boldsymbol{\theta})_i$ that represents one multiplication for each gradient value.

4 Experimental results

4.1 Benchmark datasets

The experimentation have been conducted on three different datasets, i.e., (1) CIFAR-10, (2) CIFAR-100, and (3) Mini-ImageNet. The former is composed of 60,000 RGB images of size $32 \times 32 \times 3$, with the images belonging to 10 different classes. The second one is similar to CIFAR-10, where the number of classes rises to 100, and hence, the complexity is higher. The last one is composed of 60,000 RGB images of size $84 \times 84 \times 3$ resized to $256 \times 256 \times 3$, and center cropped to $224 \times 224 \times 3$ before the training step. Similar to CIFAR-100, Mini-ImageNet is composed of 100 different classes.

4.2 Platform description

The used platform for the experiments is the modular supercomputer architecture (MSA) that has been developed by the European project Dynamical Exascale Entry Platform-Extreme Scale Technologies (DEEP-EST) [27]. The MSA is an innovative HPC architecture that can integrate an arbitrary number of modules with heterogeneous hardware components. In this work, two modules have been used. The first one is known as Data Analytics Module (DP-DAM), which is composed of 16 nodes communicated with an EXTOLL network of 100 Gb/s. In turn, each DP-DAM node is composed of two Intel Xeon "Cascade Lake" Platinum 8260M CPU running at 2.40 GHz with 24 physical cores per CPU and a Nvidia V100 Tesla GPU of 32 GB HBM2. The second module is the Extreme Scale Booster (DP-ESB) which is composed of 74 nodes. Each DP-ESB node is composed of an Intel Xeon "Cascade Lake" Silver 4215 CPU running at 2.50 GHz with 8 physical cores and a Nvidia V100 Tesla GPU with 32 GB HBM2. Nodes are connected through a 100 Gb/s EDR Infiniband network.

Conducted experiments use four nodes from the DP-DAM module for Experiments 1 and 3. Meanwhile, for Experiment 2, eight nodes have been used from the DP-ESB module.

4.3 Experimental settings

A data-parallel scheme is assumed and a heterogeneous workload partitioning is performed. In order to perform the data partitioning step, each replica is defined by the computing speed based on its computational capabilities. As a consequence, speeds are different for each replica. Thus, the speed is used to assign an input data portion to the replicas. This means that acceleration speedups are determined by the differences between resource speeds. In the experimentation, five repetitions of three experiments have been conducted.

Table 1 Average accuracy (%) after five Monte Carlo runs for CIFAR-10 and CIFAR-100

	VGG16	ResNet18	ResNet50	DenseNet121
CIFAR-10				
HetGrad	**91.77±0.14**	**92.69±0.18**	**93.01±0.10**	**93.46±0.16**
Baseline	89.87±0.32	90.98±0.20	90.63±0.26	92.25±0.24
CIFAR-100				
HetGrad	**67.02±0.31**	**71.28±0.30**	**73.08±0.32**	**73.39±0.19**
Baseline	56.24±0.16	68.58±1.32	68.70±1.83	71.81±1.92

Bold text indicates the best result obtained between both methods

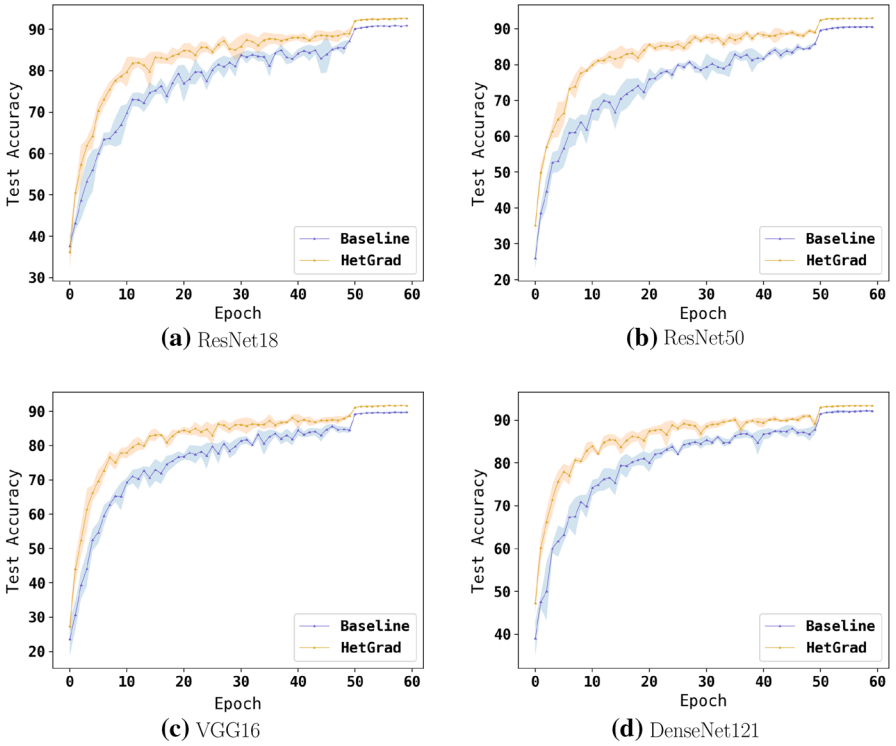


Fig. 2 Accuracy per epoch using CIFAR-10

4.4 Experimental discussion

4.4.1 Experiment 1: Initial performance insights

The first experiment compares the obtained accuracy from the baseline unweighted gradient computation with the HetGrad proposal, using the DP-DAM module with CIFAR-10 and CIFAR-100. Experiments are conducted for the VGG16, ResNet18, ResNet50 and DenseNet121 models. The objective of this experiment is to obtain initial insights on the HetGrad accuracy. Furthermore, an evaluation of the speedup

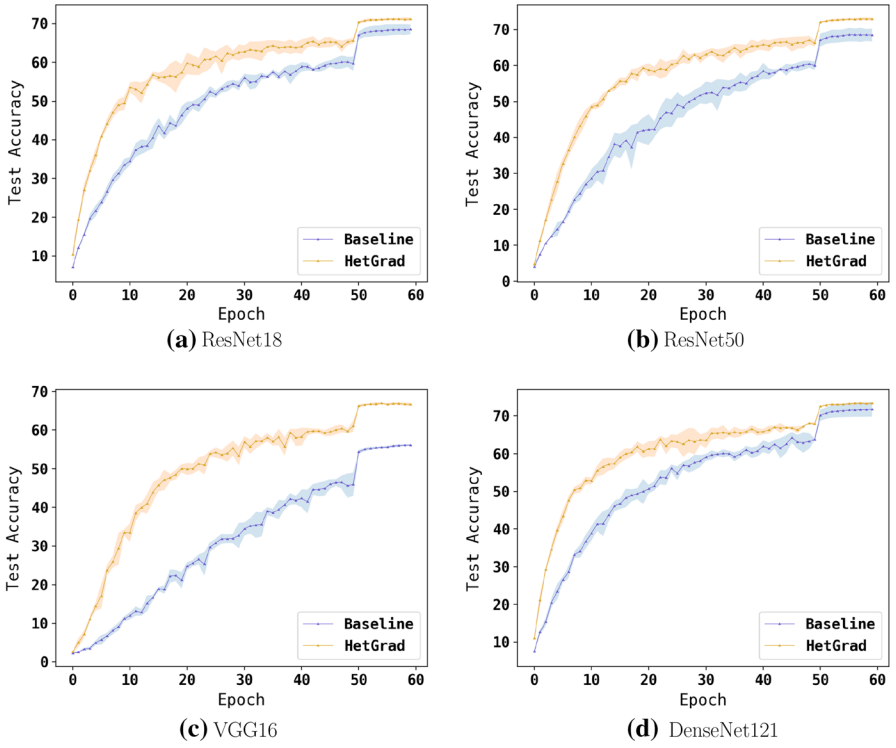


Fig. 3 Accuracy per epoch using CIFAR-100

obtained by the heterogeneous partitioning of the workload according to the speeds of the processes has been conducted and compared with a homogeneous partitioning.

Obtained results from the first experiment are detailed by Table 1 and depicted by Figs. 2 and 3 considering CIFAR-10 and CIFAR-100, respectively. As it is appreciated, positive results have been obtained for both datasets. Attending to CIFAR-10, the accuracy improvement is similar for all models, where DenseNet121 presents the higher accuracy gain, from 92.25%±0.24 to 93.46%±0.16. Regarding CIFAR-100, accuracy improvements are more significant since the complexity of the dataset is higher. The model with the highest accuracy gain is VGG16 with an accuracy improvement from 56.24%±0.16 to 67.02%±0.31. Meanwhile, DenseNet121 is the

Table 2 Speedup and epoch time (in seconds) using CIFAR-100

	VGG16	ResNet18	ResNet50	DenseNet121
CIFAR-100				
Balanced	**25.87s**	**32.10s**	**86.45s**	**80.01s**
Unbalanced	194.67s	314.56s	1014.25s	952.34s
Speedup	7.82	9.79	11.73	11.90

Bold text indicates the best result obtained between both methods

most accurate model, with an improvement from 71.81%±1.92 to 73.39%±0.19. As we can observe, HetGrad proposal obtains a higher final accuracy for all models and datasets, through boosting the accuracy in early epochs.

Finally, obtained speedups with HetGrad are provided by Table 2, where the heterogeneous workload balance provides a notable improvement in terms of performance for the training step of DNNs.

4.4.2 Experiment 2: Deep models and scalability

In this experiment, both baseline and HetGrad models are also compared using DP-ESB module on the Mini-ImageNet dataset. Other evaluated models are ResNet18, ResNet50 and ResNet101. Since DP-ESB increments the number of nodes in the platform to eight with respect to the former experiment, this experiment includes a scaling factor to support deeper ResNet models.

Table 3 details the obtained results. As shown, the accuracy improves for all evaluated ResNet models. Focusing on deep complex models as ResNet101, the accuracy increases from 60.94±4.01 to 69.37±0.13. Note that a complex dataset as Mini-Imagenet does not improves the accuracy for deeper models in the baseline proposal, which stagnates in around $\simeq$ 60%. Meanwhile, it is demonstrated that the proposed HetGrad obtains notable improvements in terms of scaling for complex datasets and models. Such complex datasets and models require more processes to improve the performance of the training, which includes heterogeneity in the computational resources that are used. This fact highly benefits our proposal, since heterogeneity is a main factor impacting the parameter updating.

4.4.3 Experiment 3: Optimizers performance analysis

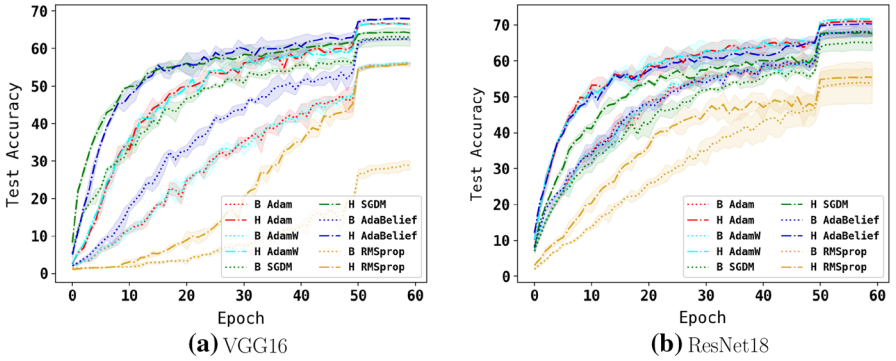
Finally, different optimizers have been evaluated to complete the evaluation of the proposal using CIFAR-100. In particular, these optimizers are SGDM, RMSprop, Adam, AdamW and AdaBelief. Two models have been selected to conduct this experiment, i.e., VGG16 and ResNet18.

Results from Fig. 4 report that every optimizer works better using the proposed HetGrad. Focusing on ResNet18 model, the best accuracy is obtained using AdamW, with a 68.52%±1.66 for the baseline and 71.69%±0.40 for the HetGrad proposal. In VGG16, AdaBelief obtains the best accuracy for the HetGrad proposal with 68.01%±0.22 and 62.59%±0.24 for the baseline. Regarding optimizers, the overall conclusion is that optimizers with regularization methods, as decoupled weight decay in AdamW, work better for complex models. In addition, the HetGrad

Table 3 Accuracy (in %) for mini-ImageNet using ResNet models

	ResNet18	ResNet50	ResNet101
Mini-ImageNet			
HetGrad	**65.02±1.89**	**66.59±2.42**	**69.37±0.13**
Baseline	60.47±4.33	60.16±5.75	60.94±4.01

Bold text indicates the best result obtained between both methods



(a) VGG16 **(b)** ResNet18

Fig. 4 Accuracy for CIFAR-100 using different optimizers. *B* denotes the baseline, whilst *H* the HetGrad

proposal obtains significantly better accuracy results than the baseline. These results get minimized at the convergence of the model but a significant improvement in accuracy is maintained.

5 Conclusions

In this paper, a weighted gradient calculation for data-parallel DNNs in scalable heterogeneous HPC platforms is presented. The proposed approach significantly reduces the impact in the global gradient calculation from replicas with a low amount of data. This is achieved using the speed of each replica in the step of global gradient calculation. Thus, in these calculations, each replica contributes proportionally to its assigned data and speed. The conducted experiments using benchmark datasets, reveal that the proposed HetGrad approach achieves better accuracy results for early epochs, providing a better final accuracy compared with the original gradient calculation while accelerate training. Furthermore, complex datasets and models have been proved to obtain major accuracy improvements in the proposed method under a scaling factor. Additionally, the data-parallelism scheme is used where the amount of data distributed for the replicas is obtained using heterogeneous workload partitioning, providing a significant time reduction. As future works, the implementation of more sophisticated optimization methods based on the heterogeneity of the resources for Federated Learning (FL) or the adaptation of the proposed method for HSI classification could obtain beneficial results.

Acknowledgements This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Grant Agreement No. 754304 DEEP-EST. Moreover, this work was partially supported by Spanish Ministerio de Ciencia e Innovación through project APRISA (Ref. PID2019-110315RB-I00 / AEI / 10.13039/501100011033), the European Regional Development Fund 'A way to achieve Europe' (ERDF) and the Junta de Extremadura (Ref. IB20040). Also, this work was supported by in part Junta de Extremadura FEDER under Grant GR21040 and by in part by 2021 Leonardo Grant for Researchers and Cultural Creators, BBVA Foundation. The BBVA Foundation accepts no responsibility for the opinions, statements and contents included in the project and/or the results thereof, which are entirely the responsibility of the authors. Additionally, this work was supported

in part by Consejería de Economía, Ciencia y Agenda Digital of the Junta de Extremadura and by the European Regional Development Fund of the European Union through the reference GR21040.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature.

References

1. Alistarh D, Grubic D, Li J, Tomioka R, Vojnovic M (2017) QSGD: communication-efficient SGD via gradient quantization and encoding. In: Guyon I, von Luxburg U, Bengio S, Wallach HM, Fergus R, Vishwanathan SVN, Garnett R (eds) Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems, pp 1709–1720
2. Ben-Nun T, Hoefler T (2018) Demystifying parallel and distributed deep learning: an in-depth concurrency analysis. arXiv:1802.09941
3. Byrd J, Lipton Z (2019) What is the effect of importance weighting in deep learning? In: Chaudhuri K, Salakhutdinov R (eds) Proceedings of the 36th International Conference Machine Learning, P. Machine Learning Research, vol. 97. PMLR, pp 872–881
4. Chang HS, Learned-Miller EG, McCallum A (2017) Active bias: training more accurate neural networks by emphasizing high variance samples. In: NIPS
5. Chen C, Weng Q, Wang W, Li B, Li B (2020) Semi-dynamic load balancing. In: Proceedings of the 11th ACM symposium on cloud computing. https://doi.org/10.1145/3419111.3421299
6. Chen CLP, Zhang CY (2014) Data-intensive applications, challenges, techniques and technologies: a survey on big data. Inf Sci 275:314–347
7. Chen J, Monga R, Bengio S, Jozefowicz R (2016) Revisiting distributed synchronous sgd. In: ICLR Workshop Track
8. Clarke D, Zhong Z, Rychkov V, Lastovetsky A (2013) Fupermod: a framework for optimal data partitioning for parallel scientific applications on dedicated heterogeneous hpc platforms. In: Parallel computing technologies. Springer, Berlin, pp 182–196
9. Gupta S, Zhang W, Wang F (2016) Model accuracy and runtime tradeoff in distributed deep learning: a systematic study. In: 2016 IEEE 16th International Conference on Data Mining (ICDM), pp 171–180
10. Gupta S, Zhang W, Wang F (2017) Model accuracy and runtime tradeoff in distributed deep learning: a systematic study. In: IJCAI, pp 4854–4858
11. Haut JM, Paoletti ME, Moreno-Álvarez S, Plaza J, Rico-Gallego JA, Plaza A (2021) Distributed deep learning for remote sensing data interpretation. In: Proceedings of the IEEE
12. Hemanth DJ, Estrela VV (2017) Deep learning for image processing applications, vol 31. IOS Press
13. Hong D, Gao L, Yokoya N, Yao J, Chanussot J, Du Q, Zhang B (2021) More diverse means better: multimodal deep learning meets remote-sensing imagery classification. IEEE Trans Geosci Remote Sens 59(5):4340–4354
14. Hong D, Han Z, Yao J, Gao L, Zhang B, Plaza A, Chanussot J (2021) Spectralformer: rethinking hyperspectral image classification with transformers. IEEE Trans Geosci Remote Sens. https://doi.org/10.1109/TGRS.2021.3130716
15. Huang G, Liu Z, Weinberger KQ (2017) Densely connected convolutional networks. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp 2261–2269
16. Ismayilova N, Ismayilov E (2018) Convergence of hpc and ai: two directions of connection. Azerbaijan J High Perform Comput 1(2):179–184

17. Jiang J, Cui B, Zhang C, Yu L (2017) Heterogeneity-aware distributed parameter servers. In: Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD '17. ACM, New York, pp 463–478
18. Krizhevsky A (2014) One weird trick for parallelizing convolutional neural networks. arXiv:1404.5997
19. Maier A, Syben C, Lasser T, Riess C (2019) A gentle introduction to deep learning in medical image processing. Zeitschrift Medizinische Physik 29(2):86–101
20. Ming Y, Zhao Y, Wu C, Li K, Yin J (2018) Distributed and asynchronous stochastic gradient descent with variance reduction. Neurocomputing 281:27–36
21. Moreno-Alvarez S, Haut JM, Paoletti ME, Rico-Gallego JA (2021) Heterogeneous model parallelism for deep neural networks. Neurocomputing 441:1–12
22. Moreno-Álvarez S, Haut JM, Paoletti ME, Rico-Gallego JA, Diaz-Martin JC, Plaza J (2020) Training deep neural networks: a static load balancing approach. J Supercomput 76(12):9739–9754
23. Nguyen TD, Park JH, Hossain MI, Hossain MD, Lee SJ, Jang JW, Jo SH, Huynh LN, Tran TK, Huh EN (2018) Performance analysis of data parallelism technique in machine learning for human activity recognition using lstm. In: IEEE International Conference on Cloud Computing Technology and Science, pp 387–391 (2019)
24. Otter DW, Medina JR, Kalita JK (2020) A survey of the usages of deep learning for natural language processing. IEEE Trans Neural Netw Learn Syst 32(2):604–624
25. Sergeev A, Balso MD (2018) Horovod: fast and easy distributed deep learning in tensorflow. arXiv:1802.05799
26. Shallue CJ, Lee J, Antognini J, Sohl-Dickstein J, Frostig R, Dahl GE (2018) Measuring the effects of data parallelism on neural network training. arXiv:1811.03600
27. Suarez E, Eicker N, Lippert T (2019) Modular supercomputing architecture: from idea to production. In: Contemporary high performance computing
28. Suresh AT, Yu F, Kumar S, McMahan HB (2017) Distributed mean estimation with limited communication. arXiv:1611.00429
29. Wang F, Jiang M, Qian C, Yang S, Li C, Zhang H, Wang X, Tang X (2017) Residual attention network for image classification. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp 6450–6458
30. Wen W, Xu C, Yan F, Wu C, Wang Y, Chen Y, Li H (2017) Terngrad: ternary gradients to reduce communication in distributed deep learning. In: 31st International Conference on Neural Information Processing Systems (NIPS 2017)
31. Yang X, Ye Y, Li X, Lau RY, Zhang X, Huang X (2018) Hyperspectral image classification with deep learning models. IEEE Trans Geosci Remote Sens 56(9):5408–5423
32. Yoginath S, Alam M, Ramanathan A, Bhowmik D, Laanait N, Perumalla KS (2019) Towards native execution of deep learning on a leadership-class hpc system. In: 2019 IEEE international parallel and distributed processing symposium workshops (IPDPSW). IEEE, pp 941–950 (2019)