Hindawi
Complexity
Volume 2018, Article ID 5348265, 9 pages
https://doi.org/10.1155/2018/5348265

WILEY | Hindawi

Research Article

Multi-GPU Development of a Neural Networks Based Reconstructor for Adaptive Optics

Carlos González-Gutiérrez,[1] **María Luisa Sánchez-Rodríguez**,[2] **José Luis Calvo-Rolle**,[3] **and Francisco Javier de Cos Juez**[1]

[1]*Department of Exploitation and Exploration of Mines, University of Oviedo, Oviedo, Spain*
[2]*Department of Physics, University of Oviedo, Oviedo, Spain*
[3]*Department of Industrial Engineering, University of A Coruña, Ferrol, A Coruña, Spain*

Correspondence should be addressed to José Luis Calvo-Rolle; jlcalvo@udc.es

Received 25 November 2017; Accepted 21 February 2018; Published 28 March 2018

Academic Editor: José Manuel Andújar

Aberrations introduced by the atmospheric turbulence in large telescopes are compensated using adaptive optics systems, where the use of deformable mirrors and multiple sensors relies on complex control systems. Recently, the development of larger scales of telescopes as the E-ELT or TMT has created a computational challenge due to the increasing complexity of the new adaptive optics systems. The Complex Atmospheric Reconstructor based on Machine Learning (CARMEN) is an algorithm based on artificial neural networks, designed to compensate the atmospheric turbulence. During recent years, the use of GPUs has been proved to be a great solution to speed up the learning process of neural networks, and different frameworks have been created to ease their development. The implementation of CARMEN in different Multi-GPU frameworks is presented in this paper, along with its development in a language originally developed for GPU, like CUDA. This implementation offers the best response for all the presented cases, although its advantage of using more than one GPU occurs only in large networks.

1. Introduction

Development of large telescopes is one of the biggest challenges in nowadays astronomy and astrophysics. Future construction of the Thirty-Meter Telescope (TMT) [1] and the European Extremely Large Telescope (E-ELT) [2], the two largest telescopes in the world, have originated enormous challenges for engineers and researchers [3]. One of the key elements of these telescopes is the development of the adaptive optics (AO) [4] system that helps to improve the quality of the received image.

There are several tomographic techniques employed in the image reconstruction for AO systems, like Single Conjugate Adaptive Optics (SCAO), Multiconjugate Adaptive Optics (MCAO), or Multiobject Adaptive Optics (MOAO) [5] to be used in the future E-ELT [3]. MOAO uses several reference guide stars to obtain information to reconstruct the atmosphere turbulence profile [6]. To combine this information, it is necessary to use tomographic reconstruction algorithms. Some of the most popular are based on a matrix vector multiplication, with the control matrix being defined by either least squares (LS) [7, 8] or minimum variance techniques [9]. However, during recent years most complex solutions have been developed, like Learn and Apply (L&A) [10], or the intelligent system known as Complex Atmospheric Reconstructor based on Machine Learning (CARMEN) [11, 12]. Due to the increasing complexity and amount of data used by these algorithms [13], some of previous algorithms have been implemented in Graphics Processing Units (GPUs) [14, 15], speeding up substantially their execution and development [16, 17].

CARMEN is a tomographic reconstructor for MOAO systems created at the University of Oviedo. It was initially developed using nonparametric estimation techniques [18] and Multivariate Adaptive Regression Splines (MARS) [19], but its development using Artificial Neural Networks (ANN)

achieves great results at on-sky testing [20, 21]. ANN and deep learning have become very popular in recent years [22], and several frameworks have been developed to help researchers in their projects [23]. Most of these frameworks provide GPU acceleration, and some of them have shown good results speeding up CARMEN training and execution [24–26]. However, only one GPU has been used in previous tests, while there are already some Multi-GPU implementations both for convolutional neural networks [27, 28] and for L&A [29].

The purpose of this paper is to detail the implementation of CARMEN in different Multi-GPU environments and compare their training and execution times. The implementations include some of the most popular neural network frameworks, with their different Multi-GPU proposals and the development of a code in a native GPU language such as CUDA.

In Section 2, a most detailed explanation of CARMEN an AO will be provided. In Section 3, different approaches to Multi-GPU systems will be given, while Section 4 talks about how the selected frameworks implement this Multi-GPU approach. Section 5 describes the proposed experiment, and Section 6 shows obtained results with its analysis. Finally, conclusions are provided along with future lines in the search for improvements.

2. CARMEN

CARMEN is a tomographic reconstructor based on artificial neural networks. In Section 2.1 details about the adaptive optics system are explained, and Section 2.2 is a small summary about the neural network architecture.

2.1. Adaptive Optics. MOAO systems use several guide stars inside the field of view of the astronomical object as a reference. These stars provide information about the wave-front aberrations produced by the atmosphere. There are several techniques to combine this information in order to reconstruct the incoming wave-front.

To characterize the incoming wave-front received in large telescopes, it is common to use the Shack-Hartman Wave-front Sensor (SH-WFS) [30]. This sensor is composed by several lenses (called lenslets) with the same focal length and are focused on different photon sensors. As it can be observed in Figure 1, the incident wave-front can be split in a matrix of tilts, and the deviation from the focal spot can be calculated. Both the x and y deviations are computed, using a centroiding algorithm. This matrix of centroids is used as input to the system and allows characterizing the aberrations introduced by the atmosphere, combining the information of several reference stars.

CANARY [31, 32] is an (AO) on-sky demonstrator, principally intended for developing and testing AO concepts for the future 39 m E-ELT. It is operated on a Nasmyth platform of the 4.2 m William Herschel Telescope, one of the Isaac Newton Group of Telescopes (ING) of the Observatorio del Roque de los Muchachos (ORM), La Palma, Canary Islands, Spain. CANARY has operated in several configurations to demonstrate different AO techniques on-sky. Different sizes and improved SH-WFS have been used through the years, which provides a wide range of configurations. Two configurations that have already been tested on-sky between 2013 and 2015 [13] will be discussed in this paper.

(i) *CANARY Phase B1* is designed to perform observations with one Rayleigh Laser Guide Star (LGS) and up to four Natural Guide Stars (NGSs). It has a SH-WFS with 7×7 subapertures, although only 36 of them are activated due to the circular telescope pupil and secondary obscuration.

(ii) *CANARY Phase C2* is designed for the study of Laser Tomography AO (LTAO) MOAO. There are four Rayleigh Laser Guide Stars, and the corresponding WFS have 14×14 subapertures (144 active).

Results in DRAGON [33], the successor of CANARY, will also be presented, although it is still under development at Durham University. Since it has not been fully finished and has never been tested on-sky, DRAGON have been simulated in Durham AO Simulation Platform (DASP) [34].

(i) *DRAGON* provides a single channel MOAO system with a woofer-tweeter DM configuration, four NGSs, and four LGSs each with 30×30 subapertures, where only 704 of them are operative.

2.2. Architecture. CARMEN is a multilayer perceptron, which contains a single hidden layer. This means that it has two fully connected layers, where every neuron is connected to all the neurons in the previous layer. The output of each neuron is computed following (1), where w is the weight of each connection, x is the value of the neurons in the previous layer, b is a constant value called *bias*, and f is an activation function.

$$Y = f\left(\sum_{i=0}^{n} (w_i \cdot x_i) + b\right). \tag{1}$$

The number and size of the hidden layers have been the purpose of previous studies [35, 36], so the architecture of the neural network will not be analyzed in this paper, and it will follow the patterns previously defined. In the input layer, the number of neurons depends on the numbers of guide stars and the size of SH-WFS. The number of input neurons will be the numbers of subapertures of the SH-WFS, multiplied by 2 (the x and y coordinates of the centroid), also multiplied by the amount of reference stars. Following the architecture of the previous studies, the number of neurons in the hidden layer is equal to the number of input neurons. Finally, the output of the ANN is the expected wave-front slope measurements as seen by the on-axis wave-front sensor, so the number of output neurons will be the number of subapertures of the SH-WFS multiplied by two. The neural network architecture is shown in Figure 2.

3. Multi-GPU Implementation

There are different approaches about how to parallelize an artificial neural network. Although using multiple CPUs has

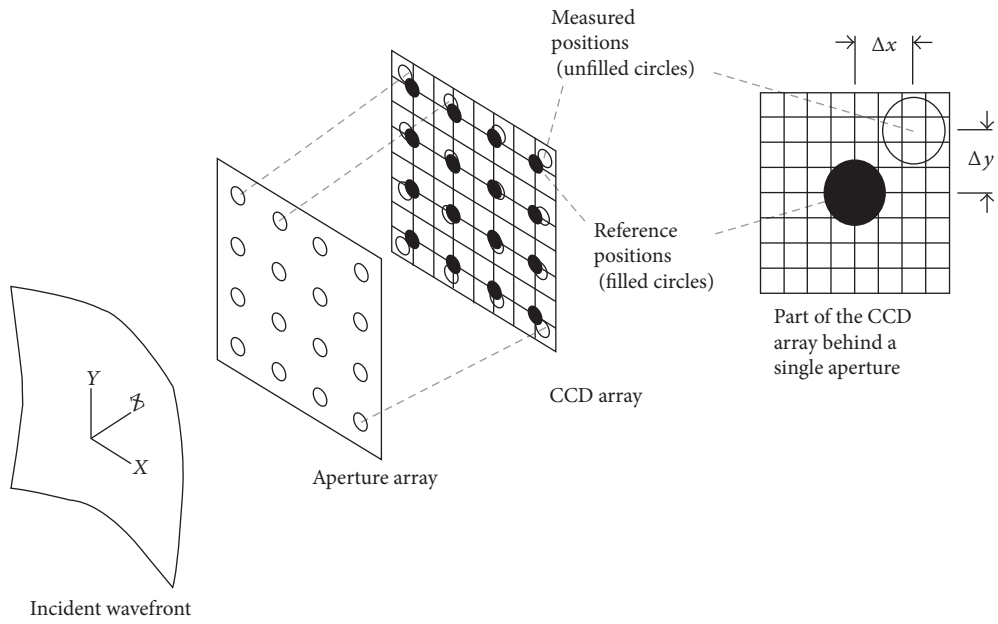

Figure 1: Measurement of Wave Front Tilts using a SH-WFS.

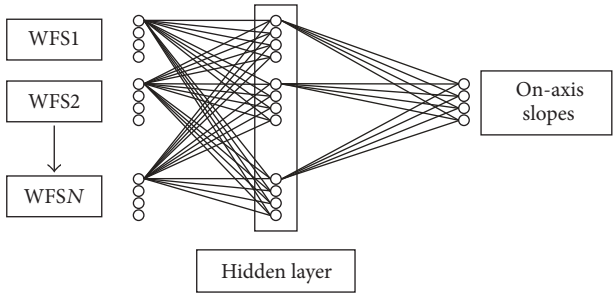

Figure 2: CARMEN architecture.

been proven as a good solution [37], in this paper we are going to focus on parallelizing the network by using GPUs, since it has been one of the common trends during the last years [25, 38]. However, splitting the work across different GPUs has been one of the biggest challenges that researchers have had to deal with. Two different solutions have been used to solve this problem, data parallelism and model parallelism, which will be detailed beneath. There are also new proposals, which can combine both data and model parallelism [28] or take advantage from asynchronous synchronization to train the neural networks [39, 40].

3.1. Data Parallelism. The idea behind data parallelism is to split the input data of each iteration in different GPUs. By using the stochastic gradient descent based on minibatches, it is easy to divide each minibatch, into smaller pieces, and use them as a smaller input to the network. Each GPU holds a copy of the complete model of the neural network and take the forward pass with the respective chunk of the minibatch. Once the output is obtained, gradients are calculated in the backward pass for each chunk. The last step is to collect all the gradients, update the model values, and send the new weights back to each GPU.

In Figure 3 it is possible to observe how data flows through the system and the theoretical speeds of exchanging information between graphics cards. Although there is no need of sending data between GPUs in the forward pass, one of the biggest problems of this kind of parallelism is exchanging the gradients in the backward pass. If the number of parameters of the different layers is too high, sending the information through the PCI-Express could consume too much time. Also, it is necessary to wait until the end of the gradient exchange and the weight matrix update, to start the next iteration of the training process, which could be a huge bottleneck.

3.2. Model Parallelism. In this scenario, the neural network model is split across the GPUs. This can be done in two different ways. One easy approach is to put each layer in a separate graphics card and send the outputs of each layer to the next GPU. However, this approach is strongly limited by the number of layers and their size. The other solution is to split the model "horizontally," so each GPU has a portion of the weights matrices from every layer.

As it can be observed in Figure 4, the outputs of each layer have to be shared between GPUs during the forward pass. This requires that each GPU needs to wait for the rest of them to be finished, so it is necessary to start all the computations almost at the same time, so all GPUs end their calculations simultaneously. During the backward pass, the error needs to be exchanged between cards also, so each matrix could be updated.

Although this method seems to exchange much more information between GPUs than the Data Parallelism idea, this is strongly dependent on the network architecture. For networks that contain large layers, it could be faster to share

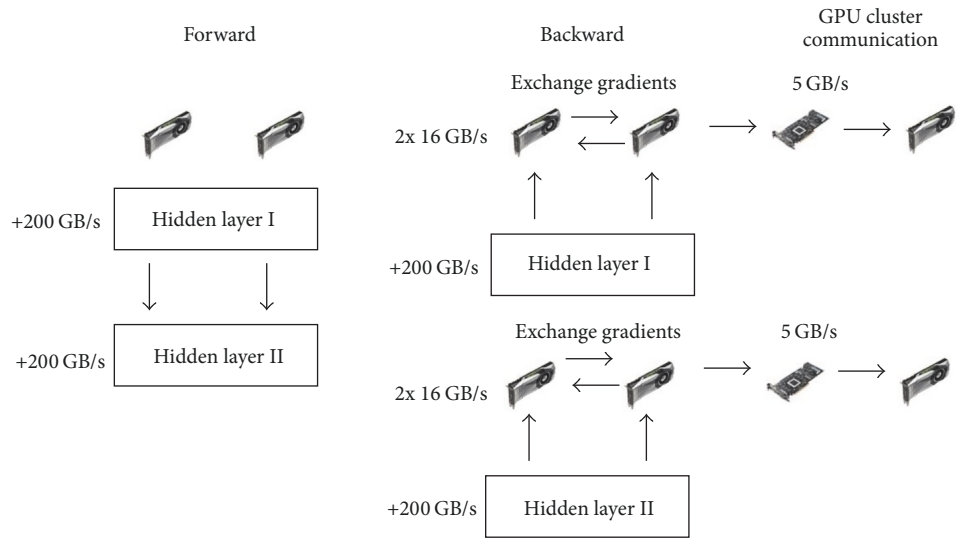

Figure 3: Data parallelism reproduced from [41].

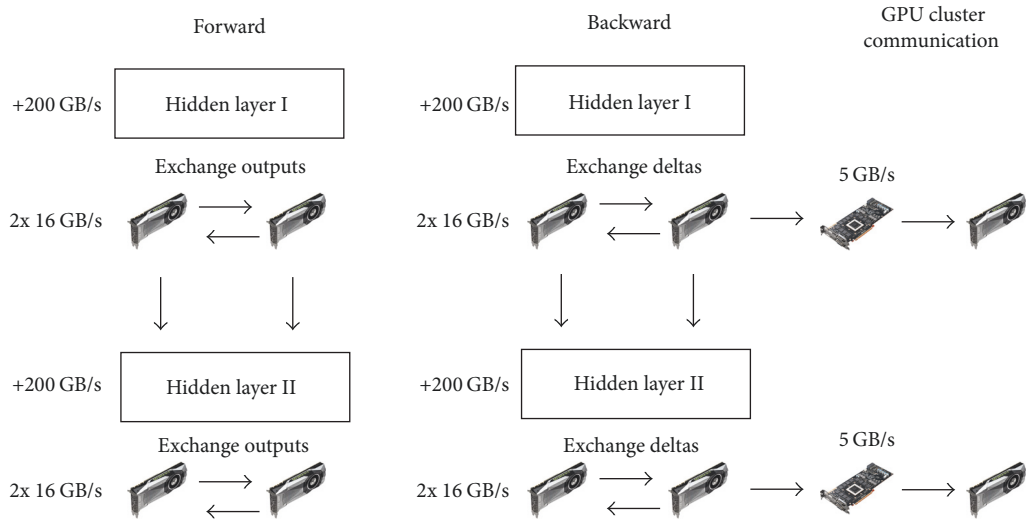

Figure 4: Model parallelism reproduced from [41].

only the output and the errors than to send the whole matrix to all the GPUs. Also splitting the matrices across the GPUs allows the use of bigger layers that could not fit in the memory of a single GPU.

4. Neural Network Frameworks Overview

With the recent success of Deep Learning in university and industry, dozens of neural networks frameworks have been developed [23]. For these analysis, only three of the most popular [42–44] have been selected. Although it was used in our previous papers [24, 26], Theano has been left out of the comparative due to the recent announcement of their creators not to continue its development [45].

4.1. Caffe. Caffe is a deep learning framework developed by the Berkeley Vision and Learning Center and is released under the BSD 2-Clause license [46]. It is mainly written in C/C++ and uses CUDA to speed up the execution on the GPU, with support for Nvidia cuDNN. It provides an interface that supports data parallelism out of the box, just selecting which GPUs of the system are going to be used. However, it does not have an implementation for executing one single input across multiple-GPUs.

4.2. Torch. Torch is a scientific computing framework with wide support for machine learning algorithms. It is written in Lua and C/CUDA to provide fast execution and includes the possibility of importing modules to complete and accelerate the system.

For the multi-GPU implementation, it provides the *DataParallelTable* module, which allows parallelizing the training with the implementation of data parallelism model. By using this module, it is possible to replicate the model across

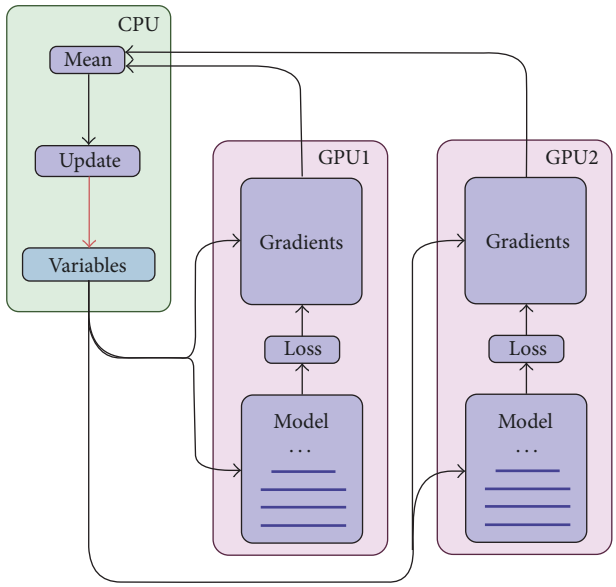

Figure 5: Data parallelism in TensorFlow.

different GPUs and split the minibatch in smaller chunks, sending each piece to a separate graphics card. Once the inputs are split, the framework automatically shares gradients between GPUs and updates the model after each iteration. As it happens with Caffe, a Multi-GPU implementation of the execution process is not possible without big changes in the source code.

4.3. TensorFlow. TensorFlow is an open source software library for numerical computation. It was originally developed by the Google Brain Team. It is written in C++ and CUDA and provides a Python API for an easy implementation.

Unlike the rest of the cases, two different implementations will be used. In the case of using only one GPU, all computations will take place in it. However, the multi-GPU implementation is a little more complex. A replica of the model is created on every GPU, as it is suggested in [47]. Then, each one of them computes the forward and backward pass and sends the gradients back to the CPU. The main processor calculates the means of the gradients, updates the model, and sends the information back to each GPU. Figure 5 is a schematic representation of how data parallelism is implemented in TensorFlow.

4.4. CUDA. CUDA is a programming language developed by Nvidia, which allows parallelizing different computations by using GPUs. It provides libraries such as CUBLAS, cuFFT, or cuDNN to facilitate the work of developers. Their use in all the previous presented frameworks and some of the tools and libraries provided, making CUDA a most suitable choice than other GPGPU languages like OpenCL.

In this case, the code has been developed to use model parallelism instead of data parallelism. Each weight matrix is divided among the GPUs, and all graphics cards received a full copy of the minibatch. After computing the outputs of each layer, by using CUBLAS and cuDNN libraries, the results are shared between GPUs. In order to avoid overloading the PCI, the sharing process is asynchronous, and it starts once one of the GPUs has finished the computation of the information. However, it is not possible to compute the next layer until all the GPUs have all the outputs.

The backward process is very similar. Each GPU computes the gradients of their portion of the matrix. After that, errors are shared and every GPU updates the corresponding weights. Since it is possible to control when the data are copied from RAM, the next minibatch is copied to every GPU while updating the matrices, so it is possible to speed up even more the training process.

5. Experiment Description

In the present paper, two different measurements will be assessed. First of all, training times will be compared, evaluating changes in the time needed by the different frameworks, when the number of GPUs is changed. The second measurement is the performance of the CUDA code for execution. The other three frameworks are left out of this comparison, because data parallelism is not suitable for executing only one input, since it is not possible to split that input among several GPUs.

5.1. Training. The training benchmark is the same method used in [24–26] and will be summarized. In this case, not only learning rate, momentum, and backpropagation algorithm, Stochastic Gradient Descent (SGD) [48], are fixed, but also the minibatch size will always be 256. A comparison will be made by changing the number of GPUs used for training, from 1 to 4. Size of the dataset will increase according to the size of the neural network, although it is not relevant since increasing the size of the dataset will increase training times linearly.

To measure training times, a timer is started once the whole dataset is copied to RAM and all weights matrixes are initialized and copied to VRAM (VRAM: GPU Video RAM, RAM: CPU RAM). Timer is stopped when the whole dataset is executed and backpropagated, which is denominated as an epoch. The operation is repeated 20 times and averaged, to ensure more reliable results, which allows checking that there is no substantial difference between each epoch, less than 1%.

5.2. Execution. In execution, there will be two different measurements for CUDA code. One of them will analyze the ideal scenario, where the input is already copied into RAM and a second where the system must load the input from SSD. This differentiation is needed because it is not possible to ensure how the future integration of the reconstructor with a real telescope will be.

For this benchmark, data is in separated h5 files [49], and the output was written to a separate file. System is fed with 10,000 inputs one at a time and averaged, which allows measuring execution time. All weight matrixes are copied to VRAM before execution and they remain constant during all iterations. In one case, the copy from SSD to RAM and vice versa is considered for the time measurement. However,

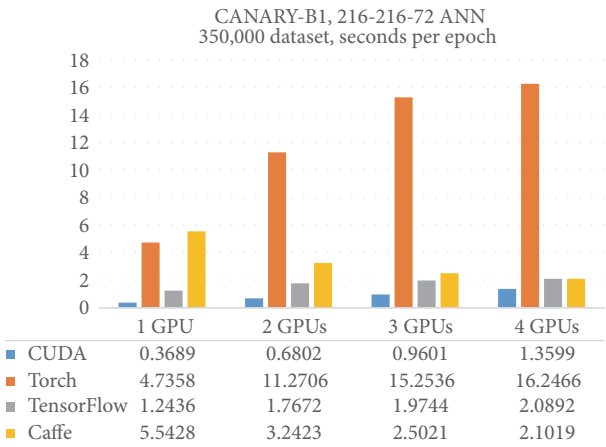

Figure 6: Training times per epoch for CANARY-B1.

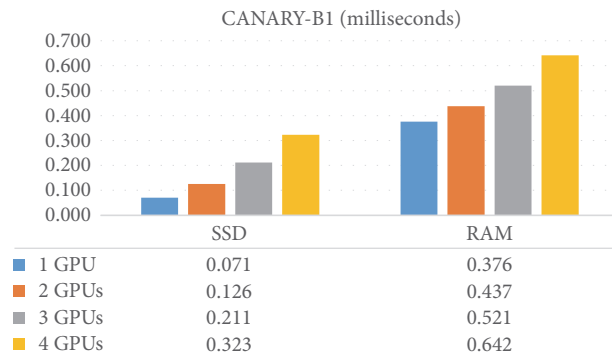

Figure 7: Execution times for CANARY-B1.

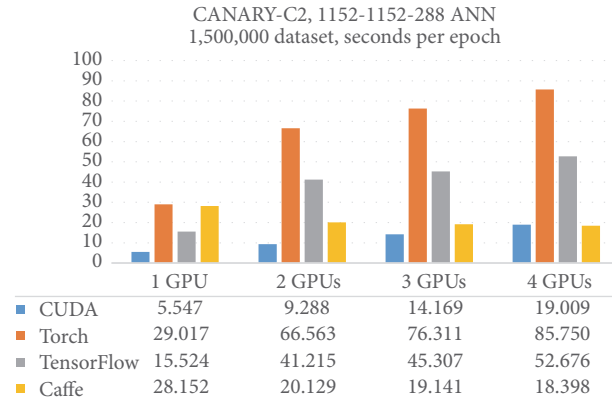

Figure 8: Training times per epoch for CANARY-C2.

in the other scenario, the timer is only started when data is copied to RAM.

5.3. Benchmark Equipment. The experiments were performed on a computer running on Ubuntu LTS 14.04.3, with an Intel Xeon CPU E5-1650 v3 @ 3.50 GHz, 128 Gb DDR4 memory, 4 Nvidia GeForce GTX TitanX, with 12 Gb DDR5 VRAM, and 1 Tb SSD hard drive.

6. Results

In this section, results obtained with the different frameworks and varying the number of GPUs will be presented. Also, an analysis of the results will be provided, explaining the behavior of the different frameworks.

6.1. CANARY-B1. CANARY-B1 is the smallest system analyzed. In Figure 6, it is possible to observe some interesting behavior. For instance, Torch is the slowest framework by far, and increasing the number of GPUs only provokes worst training times. Something similar could be monitored for CUDA and TensorFlow. Although both are the faster options with only one GPU, increasing the number of cards harms their performance. In this scenario Caffe, despite having good performance results with several GPUs, has the worst results with a single GPU. This behavior contrasts with the performance of the other frameworks considered. In particular when the number of GPUs increases, it is able to reduce time until matching times with TensorFlow with 4 GPUs.

In the execution with CUDA, similar results can be observed in Figure 7. Increasing the number of GPUs not only does not provide any benefit but also increases execution times substantially. For RAM execution has almost a linear relation, where doubling or tripling the number of GPUs has the same impact in times. Also, as it was analyzed in previous works [24, 25], loading data from SSD instead of directly from RAM has an important impact on performance, although in this case it could be observed how this difference is almost fixed (about 0.3 milliseconds) for every number of GPUs.

6.2. CANARY-C2. In CANARY-C2 results from Figure 8 are quite similar to those of the previous system. Torch is the slowest framework again and increasing the number of GPUs is very harmful for training times. In this case, TensorFlow has a similar behavior, having a huge increase in their times when escalating the number of GPUs. Again, CUDA is the fastest solution but is not able to take advantage of the increasing number of GPUs to speed up training. At last, Caffe has similar training times than torch for single GPU. However, with 4 GPUs it is able not only to decrease their training times, but also to improve the results obtained by CUDA.

As it happens in training times, CANARY-C2 shows almost the same results (Figure 9) as CANARY-B1. It is interesting to notice that even having a neural network with about 28x more weights, the increment in execution times is low, especially in the case of 4 GPUs.

6.3. DRAGON. DRAGON is the biggest network employed in this paper and has about 100x more connections than CANARY-C2. In this scenario there is a shift in some trends observed in previous networks. In Figure 10, TensorFlow is the slowest framework for every number of GPUs. Torch is the second worst option, although training times do not increase as much as in the other cases. For a network of this size, Caffe has better results than the other two frameworks, but, unlike the previous cases, it is not able to reduce training times when increasing the number of GPUs. At last, CUDA is again the fastest solution. However, in this case when the number of GPUs is increased, training times are reduced.

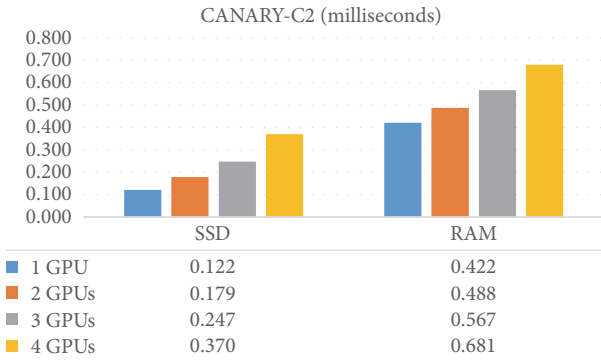

Figure 9: Execution times for CANARY-C2.

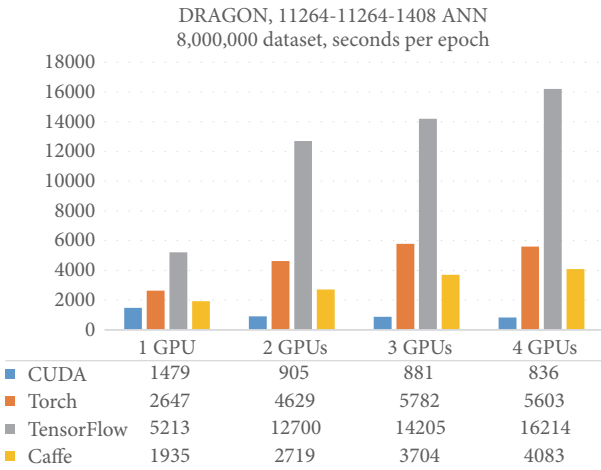

Figure 10: Training times per epoch for DRAGON.

A similar situation can be observed for the execution times. As it can be observed in Figure 11, increasing the number of GPUs can decrease execution times. The difference between using one and two GPUs is especially significant, although even in the case of four, execution times keep improving.

6.4. Discussion. There are lots of ideas that could be extracted from the previous results. One of them is that data parallelism implementations of Torch and TensorFlow are not suitable solutions for the present problem. Keeping this in mind, it is interesting to notice how the "full-GPU" backpropagation in Torch is quite slow for smaller systems but improves when the neural network size increases. However, for the hybrid CPU-GPU solution (Figure 5) used in TensorFlow, better results are obtained when the size is smaller. This behavior could be explained by how backpropagation in a small network cannot take advantage of the high level of parallelism provided by a GPU and consequently is better performed in a CPU. On the opposite side, when the neural network grows, not only this parallelism is better used, but also the time consumed by sending all the weight matrixes from RAM to VRAM is a high time-consuming operation.

Regarding Caffe, some differences are observed with respect to the other solutions. For small networks it is able to

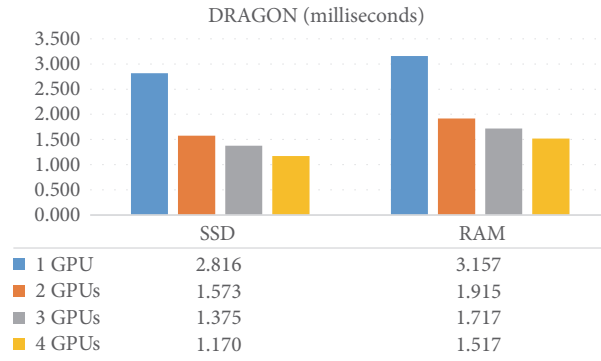

Figure 11: Execution times for DRAGON.

reduce training times when increasing the number of GPUs. However, this is not enough to improve results provided by the CUDA code. Since the multi-GPU mode is a built-in solution, it is not possible to explain why these results take place.

At last, the model parallelism solution implemented in CUDA provides the best results for all cases. However, the inability to improve training and execution times for smaller networks, when more GPUs are available, should be analyzed. For CANARY systems, the workload is too low even for one GPU, and increasing the number only provokes time overload because of the movement of data between cards. However, when the size of the network increases, and the GPUs are fully loaded, both training and execution times start to reduce. Although DRAGON is currently the largest system available, it will keep growing in the next years [13], and bigger systems will take much more advantages of this multi-GPU implementation.

In the case of execution, an interesting behavior can be observed. Including data transfer between SSD and RAM in the experiment triggers a huge overload for all cases. There is a 0.3 milliseconds difference for every case, independently of the size of the data transfer. However, it is expected that, in a real telescope, it will not be necessary to read and copy data to the SSD, because of a better integration with the system. Also, as it happens with training, increasing the size of the network takes advantages of the number of GPUs. As it was mentioned before, the smaller networks do not provide enough workload to fill a GPU, but due to the increasing size of the sensors, it will not be a problem in the near future.

7. Conclusions and Future Lines

In this paper, some of the most popular neural network frameworks [42] and a code directly developed in CUDA have been analyzed. It is possible to extract from the exposed results that both TensorFlow and Torch are not suitable for Multi-GPU execution when the neural network used is a multilayer perceptron. In the case of Caffe, it has shown that increasing the number of GPUs in smaller networks offers some benefits, although it is not able to keep those benefits when the size increases. For the three SH configurations presented, the best solution is the code directly developed in CUDA, which has also the potential of taking better

advantages of the number of GPUs, when the size of the network increases.

However, developing these solutions in different frameworks can provide some advantages in the future. Convolutional neural networks (CNN) are very popular for image processing [50, 51], and their use could be an interesting upgrade for CARMEN architecture. Previous studies show that model parallelism could be a better solution for CNN than data parallelism [27, 28], so any of the frameworks could be faster for CNN than the CUDA solution. Also, Recurrent Neural Networks (RNN) [52, 53] should be easier to implement by using a neural network framework, since they are much more prepared for that use [42].

One last idea is the use of online learning, which means that CARMEN could be constantly training when the reconstructor is executed on-sky. Recent works show that this technique could provide an interesting boost in the performance of the reconstructor [54, 55]. By using more than one GPU, it will be possible to adapt the neural network to work faster and be much more precise correcting what is happening on sky.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

The authors appreciate support from the Spanish Economics and Competitiveness Ministry, through Grant AYA2014-57648-P, and the Government of the Principality of Asturias (Consejería de Economía y Empleo), through Grant FC-15-GRUPIN14-017.

References

[1] J. Nelson and G. H. Sanders, "The status of the thirty meter telescope project," in *Proceedings of the Ground-based and Airborne Telescopes II*, France, June 2008.

[2] R. Gilmozzi and J. Spyromilio, "The European extremely large telescope (E-ELT)," *The Messenger*, vol. 127, no. 3, 2007.

[3] S. K. Ramsay, M. M. Casali, J. C. González, and N. Hubin, "The E-ELT instrument roadmap: a status report," in *Ground-based and Airborne Instrumentation for Astronomy V, 91471Z*, Proceedings of SPIE, 2014.

[4] J. M. Beckers, "Adaptive Optics for Astronomy: Principles, Performance, and Applications," *Annual Review of Astronomy and Astrophysics*, vol. 31, no. 1, pp. 13–62, 1993.

[5] D. R. Andersen, K. J. Jackson, C. Blain et al., "Performance modeling for the RAVEN Multi-Object Adaptive Optics demonstrator," *Publications of the Astronomical Society of the Pacific*, vol. 124, no. 915, pp. 469–484, 2012.

[6] O. Lardière, D. Andersen, C. Blain et al., *Multi-object adaptive optics on-sky results with Raven, 91481G*, E. Marchetti, L. M. Close, and J.-P. Véran, Eds., International Society for Optics and Photonics, 2014.

[7] B. L. Ellerbroek, "First-order performance evaluation of adaptive-optics systems for atmospheric-turbulence compensation in extended-field-of-view astronomical telescopes," *Journal of the Optical Society of America A: Optics and Image Science, and Vision*, vol. 11, no. 2, pp. 783–805, 1994.

[8] T. Fusco, J.-M. Conan, G. Rousset, L. M. Mugnier, and V. Michau, "Optimal wave-front reconstruction strategies for multiconjugate adaptive optics," *Journal of the Optical Society of America A: Optics and Image Science, and Vision*, vol. 18, no. 10, pp. 2527–2538, 2001.

[9] M. C. Roggemann, "Optical performance of fully and partially compensated adaptive optics systems using least-squares and minimum variance phase reconstructors," *Computers and Electrical Engineering*, vol. 18, no. 6, pp. 451–466, 1992.

[10] F. Vidal, E. Gendron, and G. Rousset, "Tomography approach for multi-object adaptive optics," *Journal of the Optical Society of America A: Optics and Image Science, and Vision*, vol. 27, no. 11, pp. A253–A264, 2010.

[11] F. J. de Cos Juez, F. S. Lasheras, N. Roqueñí, and J. Osborn, "An ANN-based smart tomographic reconstructor in a dynamic environment," *Sensors*, vol. 12, no. 7, pp. 8895–8911, 2012.

[12] J. Osborn, F. J. De Cos Juez, D. Guzman et al., "Using artificial neural networks for open-loop tomography," *Optics Express*, vol. 20, no. 3, pp. 2420–2434, 2012.

[13] A. G. Basden, D. Atkinson, N. A. Bharmal et al., "Experience with wavefront sensor and deformable mirror interfaces for wide-field adaptive optics systems," *Monthly Notices of the Royal Astronomical Society*, vol. 459, no. 2, pp. 1350–1359, 2016.

[14] D. Romero-Laorden, J. Villazon-Terrazas, O. Martinez-Graullera, A. Ibanez, M. Parrilla, and M. S. Penas, "Analysis of Parallel Computing Strategies to Accelerate Ultrasound Imaging Processes," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 12, pp. 3429–3440, 2016.

[15] D. Romero-Laorden, J. Villazón-Terrazas, M. Santos Peñas, M. A. García-Izquierdo, and O. Martínez-Graullera, "Analysis of a software implementation of an ultrasonic signal beamformer in real-time," *RIAI - Revista Iberoamericana de Automatica e Informatica Industrial*, vol. 13, no. 4, pp. 462–471, 2016.

[16] H. Ltaief and D. Gratadour, "Shooting for the Stars with GPUs," 2016, http://on-demand.gputechconf.com/gtc/2015/video/S5122.html.

[17] J. Marichal-Hernández, L. F. Rodríguez-Ramos, F. Rosa, and J. M. Rodríguez-Ramos, "Atmospheric wavefront phase recovery by use of specialized hardware: Graphical processing units and field-programmable gate arrays," *Applied Optics*, vol. 44, no. 35, pp. 7587–7594, 2005.

[18] D. Guzmán, F. J. De Cos Juez, R. Myers, A. Guesalaga, and F. S. Lasheras, "Modeling a MEMS deformable mirror using non-parametric estimation techniques," *Optics Express*, vol. 18, no. 20, pp. 21356–21369, 2010.

[19] D. Guzmán, F. J. C. De Juez, F. S. Lasheras, R. Myers, and L. Young, "Deformable mirror model for open-loop adaptive optics using multivariate adaptive regression splines," *Optics Express*, vol. 18, no. 7, pp. 6492–6505, 2010.

[20] J. Osborn, D. Guzman, F. J. de Cos Juez et al., "First on-sky results of a neural network based tomographic reconstructor: Carmen on Canary," in *SPIE Astronomical Telescopes + Instrumentation, 91484M*, E. Marchetti, L. M. Close, and J.-P. Véran, Eds., International Society for Optics and Photonics, 2014.

[21] J. Osborn, D. Guzman, F. J. D. C. Juez et al., "Open-loop tomography with artificial neural networks on CANARY: on-sky results," *Monthly Notices of the Royal Astronomical Society*, vol. 441, no. 3, pp. 2508–2514, 2014.

[22] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[23] C. Gulcehre, "Deep Learning - Software Links," 2017, http://deeplearning.net/software_links/.

[24] C. González-Gutiérrez, J. D. Santos, M. Martínez-Zarzuela et al., "Comparative study of neural network frameworks for the next generation of adaptive optics systems," *Sensors*, vol. 17, no. 6, article no. 1263, 2017.

[25] C. González-Gutiérrez, J. D. Santos-Rodríguez, R. Á. F. Díaz, J. L. C. Rolle, N. R. Gutiérrez, and F. J. de Cos Juez, "Using GPUS to speed up a tomographic reconstructor based on machine learning," *Advances in Intelligent Systems and Computing*, vol. 527, pp. 279–289, 2017.

[26] S. L. Suárez-Gómez, C. González-Gutiérrez, J. D. Santos-Rodríguez et al., "Analysing the performance of a tomographic reconstructor with different neural networks frameworks," in *Proceedings of the 16th International Conference on Intelligent Systems Design and Applications*, A. M. Madureira, A. Abraham, D. Gamboa, and P. Novais, Eds., pp. 1051–1060, Springer International Publishing, Cham, Switzerland, 2017.

[27] O. Yadan, K. Adams, Y. Taigman, and M. Ranzato, "Multi-GPU Training of ConvNets," https://arxiv.org/abs/1312.5853.

[28] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," https://arxiv.org/abs/1404.5997.

[29] H. Ltaief, D. Gratadour, A. Charara, and E. Gendron, "Adaptive optics simulation for the world's largest telescope on multicore architectures with multiple GPUs," in *Proceedings of the 3rd Conference on Platform for Advanced Scientific Computing, PASC 2016*, Switzerland, June 2016.

[30] B. C. Platt and R. Shack, "History and principles of Shack-Hartmann wavefront sensing," *Journal of Refractive Surgery*, vol. 17, no. 5, pp. S573–S577, 2001.

[31] R. M. Myers, Z. Hubert, T. J. Morris, E. Gendron, and N. A. Dipper, "CANARY: the On-Sky NGS/LGS MOAO Demonstrator for EAGLE," in *Adaptive Optics Systems, 70150E*, Proceedings of SPIE, pp. 1–9, 2010.

[32] E. Gendron, F. Vidal, M. Brangier et al., "MOAO first on-sky demonstration with CANARY," *Astronomy & Astrophysics* , vol. 529, article no. L2, 2011.

[33] A. P. Reeves, R. M. Myers, T. J. Morris et al., "The Durham real-time, tomographic adaptive optics test bench: progress and results," in *SPIE Astronomical Telescopes + Instrumentation, 91485U*, E. Marchetti, L. M. Close, and J.-P. Véran, Eds., International Society for Optics and Photonics, 2014.

[34] A. G. Basden and A. O. Durham, "Simulation Platform," 2017, https://gitlab.com/agb32/dasp.

[35] J. Osborn, F. J. De Cos Juez, D. Guzman et al., "Open-loop tomography using artificial nueral networks," in *Proceedings of the 2nd International Conference on Adaptive Optics for Extremely Large Telescopes, AO for ELT 2011*, September 2011.

[36] M. G. Victoria, *Research of The Tomographic Reconstruction Problem by Means of Data Mining And Artificial Intelligence Technologies*, Universidad de Oviedo, 2014.

[37] J. Dean, G. S. Corrado, R. Monga et al., Large Scale Distributed Deep Networks.

[38] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., pp. 1097–1105, Curran Associates, Inc., 2012.

[39] S. Zhang, C. Zhang, Z. You, R. Zheng, and B. Xu, "Asynchronous stochastic gradient descent for DNN training," in *Proceedings of the 38th IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '13)*, pp. 6660–6663, May 2013.

[40] T. Paine, H. Jin, J. Yang, Z. Lin, and T. Huang, "GPU Asynchronous Stochastic Gradient Descent to Speed Up Neural Network Training," https://arxiv.org/abs/1312.6186.

[41] T. Dettmers, "Making deep learning accessible," 2017, http://timdettmers.com/.

[42] A. Parvat, J. Chavan, S. Kadam, S. Dev, and V. Pathak, "A survey of deep-learning frameworks," in *Proceedings of the 2017 International Conference on Inventive Systems and Control (ICISC)*, pp. 1–7, January 2017.

[43] S. Shi, Q. Wang, P. Xu, and X. Chu, "Benchmarking state-of-the-art deep learning software tools," in *Proceedings of the 7th International Conference on Cloud Computing and Big Data, CCBD 2016*, pp. 99–104, China, November 2016.

[44] S. Bahrampour, N. Ramakrishnan, L. Schott, and M. Shah, "Comparative study of deep learning software frameworks," https://arxiv.org/abs/1511.06435.

[45] P. Lamblin, "Theano 0.9.0 documentation," 2017, http://deeplearning.net/software/theano/.

[46] Y. Jia, E. Shelhamer, J. Donahue et al., "Caffe: convolutional architecture for fast feature embedding," in *Proceedings of the ACM International Conference on Multimedia*, pp. 675–678, ACM, Orlando, Fla, USA, November 2014.

[47] "Google Brain Team Convolutional Neural Networks - Training a Model using Multiple GPU Cards — TensorFlow," 2017, https://www.tensorflow.org/tutorials/deep_cnn#training_a_model_using_multiple_gpu_cards.

[48] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[49] "The HDF Group Introduction to HDF5," 2016, https://www.hdfgroup.org/HDF5/doc/H5.intro.html#Intro-WhatIs.

[50] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. Lecun, "Integrated recognition, localization and detection using convolutional networks," in *Proceedings of the International Conference on Learning Representations (ICLR2014)*, 2014.

[51] C. Szegedy, W. Liu, Y. Jia et al., "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR '15)*, pp. 1–9, Boston, MA, USA, June 2015.

[52] A. Graves, "Generating sequences with recurrent neural networks," Tech. Rep., 2013.

[53] A. Graves, A.-R. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Proceedings of the 38th IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '13)*, pp. 6645–6649, May 2013.

[54] S. L. Suárez-Gómez, M. L. Sánchez, F. Blanco, J. Ayala, and F. J. de Cos Juez, "Successful sulfur recovery in low sulfurate compounds obtained from the zinc industry: Evaporation–condensation method," *Journal of Hazardous Materials*, vol. 336, pp. 168–173, 2017.

[55] S. L. S. Gómez, J. D. S. Rodríguez, F. J. I. Rodríguez, and F. J. D. C. Juez, "Analysis of the temporal structure evolution of physical systems with the self-organising tree algorithm (SOTA): Application for validating neural network systems on adaptive optics data before on-sky implementation," *Entropy*, vol. 19, no. 3, article no. 103, 2017.