A Study on the Performance of Distributed Training of Data-driven CFD Simulations

Article *in* International Journal of High Performance Computing Applications · May 2023

DOI: 10.1177/10943420231160557

Sergio Iserte
Barcelona Supercomputing Center
41 PUBLICATIONS **213** CITATIONS
SEE PROFILE

Paloma Barreda
Universitat Jaume I
4 PUBLICATIONS **5** CITATIONS
SEE PROFILE

Alejandro González-Barberá
Universitat Jaume I
1 PUBLICATION **1** CITATION
SEE PROFILE

Krzysztof Rojek
Czestochowa University of Technology
26 PUBLICATIONS **278** CITATIONS
SEE PROFILE

Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

ARTICLE TYPE

A Study on the Performance of Distributed Training of Data-driven CFD Simulations

S. Iserte*[1] | A. González-Barberá[1] | P. Barreda[1] | K. Rojek[2]

[1]Universitat Jaume I, Castelló, Spain
[2]Czestochowa University of Technology, Poland

Correspondence
*siserte@uji.es

Summary

In this study, we propose a method, based on deep learning (DL), for accelerating computational fluid dynamics (CFD) simulations. For this purpose, we focus on the phenomenon of water flow thru a biological reactor used for wastewater recovery. Based on this study case, we develop a predictive model capable of inferring future states from the learned time series.
The traditional approach to simulating the physics of fluids relies on solving partial differential equations (PDE). Since calculating this iterative equations are highly computational demanding and time consuming, our method uses a DL model to predict any time step of a simulation until a steady-state in a fraction of time.
The proposed solution is evaluated in clusters equipped with multicore CPUs and GPUs. We analyze the behavior of two methods of distributed training including parallelization with Horovod and Tensorflow `mirrored` & `multiworker strategy`. Moreover, we examine the accuracy of the proposed method and compare it with the original CFD solver. Finally, we compare the performance speedup of training in multi-GPU computational clusters.

KEYWORDS:
Deep Learning, CFD Simulation, CPU/GPU Computing, Distributed Training

1 | INTRODUCTION

Artificial intelligence (AI) methods have become pervasive in recent years due to numerous algorithmic advances and the accessibility of computational power[1,2]. In computational fluid dynamics (CFD), AI has been used to assist, accelerate, enhance, or even replace existing physical solvers based on partial differential equations (PDE)[3]. Especially, for the last few years, deep learning (DL) methods have been widely used in the field of CFD.

In this work, we propose a domain-specific AI model that can be easily integrated with CFD simulations. The model training is thoroughly evaluated with distributed techniques on CPU and GPU devices.

Our research is conducted using a geometry of a full-scale biological reactor in production built in a water resource recovery facility (WRRF). Wastewater treatment is a critical process to effectively recover water including indirect potable reuse, direct potable reuse, cooling tower, and boiler make-up, food processing wash water, and/or direct discharge to the environment.

Figure 1 depicts the geometry of the reactor and the location of its areas of interest which are not simple walls and can be configured. As it is shown, the reactor is composed of a tank of $15m$ length, $6m$ width, and $5m$ depth. The flow enters the tank through two differentiate locations *inflow #1* and *inflow #2* and it leaves the reactor through the *outflow*. Inside the tank there is also a stirrer which is responsible for enhancing the mixing within the flow.

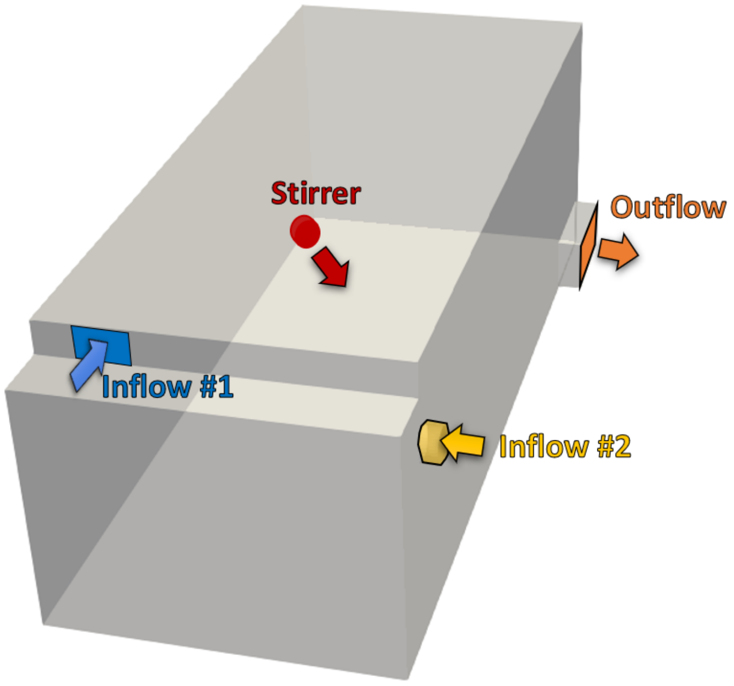

FIGURE 1 Geometry of the reactor under study. Arrows represent the intended direction of the flow in the different areas of interest.

Notice that this study case is limited to evaluating a predictive model and its performance. Thus, specific details about the reactor are beyond the scope of this paper. Nevertheless, it has been widely studied in other works where further information can be found[4,5].

Our goal is to investigate the performance and different strategies of learning the AI model based on a data-driven approach[6]. Then we provide interaction between AI and CFD solver for much faster analysis and reduced cost of trial & error experiments. The scope of our research includes steady-state simulations, which use an iterative scheme to progress to convergence. Steady-state models perform a mass and energy balance of a process in an equilibrium state, independent of time[7]. In other words, we assume that a solver calculates a set of time steps to achieve the convergence state of the simulated phenomenon. Whence, our method is responsible for predicting the subsequent state with the AI model based on former time steps generated by the CFD solver.

The most relevant contributions of this work are:

- an AI-based method that can be integrated with a CFD solver and allows users to predict the evolution of simulations based on initial time steps generated by the CFD solver;
- a recursive neural network (RNN) that predicts future states in a flow time-series;
- a performance analysis of different distributed training methods considering Horovod and Tensorflow `multiworker strategy`; and
- a performance comparison between CPU and GPU clusters for training and evaluating the devised predictive model.

The rest of the paper is structured as follows: Section 2 details the methods for simulating the flow, the RNN design, and validation, and introduces the distributed training techniques. Section 3 presents the performance evaluation of the training techniques. Section 4 overviews the related work and research efforts where the presented study could be valuable. The manuscript concludes in Section 5 where future work is also included.

2 | MATERIALS AND METHODS

This section describes the CFD model that resolves the hydrodynamic inside the tank, as well as the predictive model, and how data is generated and processed. Distributed training techniques are also presented in this section.

2.1 | CFD Simulation

This point describes the CFD tools leveraged in this work and a brief explanation of how they have been configured. All the CFD simulations have been performed using the software OpenFOAM[†].

In order to simulate the hydrodynamic in the reactor, its geometry needs to be discretized into a mesh. For this purpose, OpenFOAM tools `blockMesh`[‡] and `snappyHexMesh`[§] were leveraged. The generated mesh corresponds to an octree with a maximum cell length of 0.213 cm, and refinement up to two levels for each boundary. Furthermore, the *inflow #2* and *outflow* surfaces are inflated with three layers. As a result, the mesh domain is composed of $125,565$ cells.

The hydrodynamic numerical resolution inside the domain is performed with the OpenFOAM solver `pimpleFoam`[¶], a large time step transient solver for incompressible flow. The flow equations are modeled using the Unsteady Reynolds-averaged Navier-Stokes (URANS) equations.

In a nutshell, the fluid dynamic model is configured as follows: The kinematic viscosity is set to 0.000001 m^2/s, which is associated with the water properties. For the boundary conditions, both, *inflow #1* and *inflow #2* surfaces are defined as a `flowRateInletVelocity` type condition with a constant volumetric flow rate. The *outflow* surface is configured as `zeroGradient` type. The walls and top surfaces of the reactor count with the free-slip condition, while the remaining surfaces are specified as no-slip.

The time step of the simulation is adjusted to the write interval, which corresponds to 10 seconds of simulated time. The flow is evolved until the second $4,201$ of the simulated time. What it is translated into 420 stored time steps per executed simulation.

2.2 | Predictive Model

This section describes the dataset generation, the predictive model design, its evaluation and validation, and the distributed training methods leveraged in the performance evaluation.

2.2.1 | Dataset

The dataset is composed of the stored time steps coming from the performed simulations. For this study, the execution have been limited to flow variations on the *inflow #1* and *inflow #2* boundaries. Table1 contains the utilized values.

Boundary	Values (m^3/s)
Inflow #1	87 values in the range from 0.1666 to 0.3389
Inflow #2	0.3333, 0.3611, 0.3889, 0.4167, and 0.4443

TABLE 1 Range of values for the simulations in the dataset

The combination of the values in Table 1 returns 131 different configurations to simulate. In this regard, the dataset is composed of 131 cases of 420 steps each. In turn, every single step contains the information of 125,565 cells. Although each cell in the domain can host several calculated metrics such as pressure or turbulence, the predictive model is focused on the three-dimensional metric of velocity. The eventual dataset with shape $131 \times 420 \times 125,565 \times 3$ has a size of 38.6 GB in memory using the `float32` data type.

[†] http://www.openfoam.com
[‡] https://www.openfoam.com/documentation/user-guide/4-mesh-generation-and-conversion/4.3-mesh-generation-with-the-blockmesh-utility
[§] https://www.openfoam.com/documentation/guides/latest/doc/guide-meshing-snappyhexmesh.html
[¶] https://www.openfoam.com/documentation/guides/latest/doc/guide-applications-solvers-incompressible-pimpleFoam.html

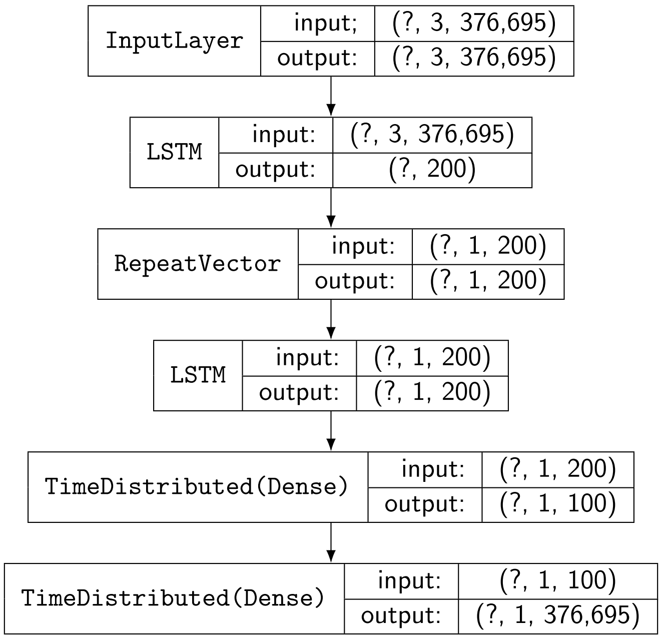

FIGURE 2 RNN architecture. Next to the type of layer, the input and output data shapes of each layer are indicated.

Before feeding the predictive model, each velocity dimension is normalized to have a distribution of mean zero and standard deviation of one. Moreover, the dataset is split into train and test subsets. For this purpose, cases are shuffled and 80% of them (104 cases) are assigned to the training dataset, while the remaining (27 cases) to the testing dataset. Notice that 20% of training cases (20 cases) are used for cross-validating the learning.

2.2.2 | Neural Network

The objective of the predictive model is to be capable of predicting time series given a short sequence of steps. For this reason, an RNN able to retain temporal knowledge has been devised. Figure 2 depicts the layers architecture of the designed RNN. Each layer in the figure describes the data shape of their input and output. Notice that "?" symbol in the figure refers to the mini-batch size, which is not defined during the model design.

To begin with, the input layer expects three time steps and the velocity values in each cell. Since velocity is three-dimensional, the model expects to receive the velocity in a flat array of 376,695 values corresponding to $mesh\ cells \times velocity\ dimensions$.

Recurrence is implemented using Long Short-Term Memory (LSTM) layers. The proposed model is composed of two 200-unit LSTM layers. The first one receives the input and transforms it accordingly to the given units to conform to the output space. That output is repeated as many times as the desired predicted steps. In this study, the predicted steps are limited to one. At this point, each repeated vector follows its path composed of another LSTM, a 100-neurons fully-connected layer, and finally, another dense layer that outputs the initial velocity flat array.

The neurons within the hidden layers are activated with the rectified linear unit (ReLU) non-linear function which allows small positive gradients when the unit is not active[8].

The model is compiled with the *Adam* optimizer[9] to update weights and biases within the network. The optimizer is configured with a learning rate of 0.00025. The chosen loss function computes the mean of the absolute difference between labels and predictions (MAE):

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y_i - y_i'|. \tag{1}$$

Because of the hardware limitations (described in Section 3), the mini-batch size is set to 14. The model has been trained until validation loss no longer decreased during a period of ten epochs. During this process, validation loss kept above training loss showing good signs of overfitting prevention. Eventually, the training was considered completed after 20 epochs, yielding a training loss in the magnitude of 10^{-2}.

The model is fed with the help of a generator which not only arranges the mini-batches but also reshapes the data and the target accordingly to the network characteristics previously described.

The model is fed with the help of a base object from the keras library used as a generator for fitting to a sequence of data, namely a dataset. Our implementation contains five different methods which are described below.

The initialization of the generator contains relevant information for our problem such as the dimensions, batch-size or number of channels. Moreover, a method which denotes the number of batches per epoch and another that is in charge of generating one batch of data through the entire dataset are required. Besides, a method is needed so as to update indexes after each epoch. Finally, the one that achieves the production of each batch from the data. Review

2.2.3 | Evaluation

In this section, the accuracy of the model is assessed. For this purpose, we have analyzed a set of measurements to validate the results. The analysis is based on plots of the velocity field over the XZ cutting plane defined in the center point of the reactor.

The results are shown in Figs.3-5, where we see the $10-th$, $20-th$, and the steady-state calculated by the traditional CFD solver (left side) and AI-accelerated results (right side).

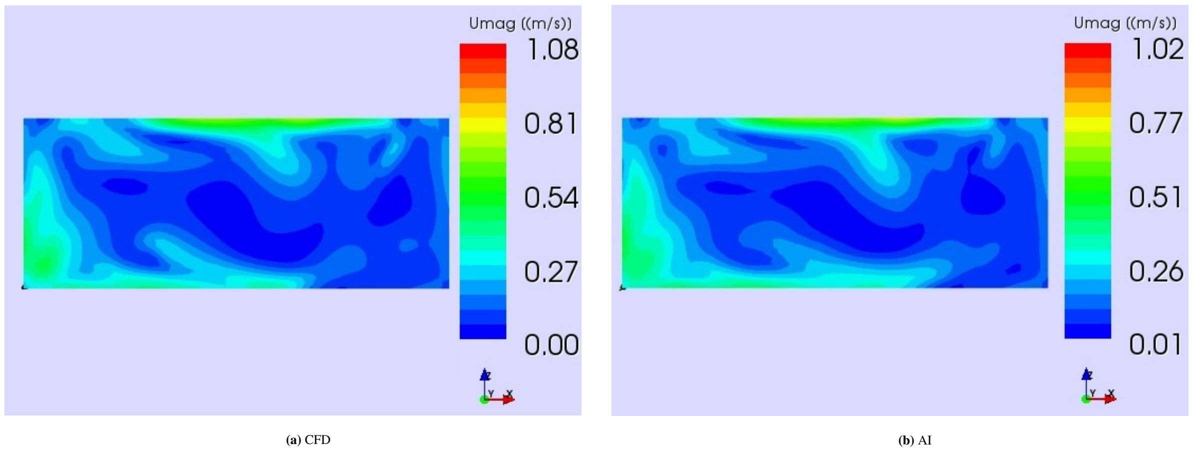

FIGURE 3 Contour plot of the velocity magnitude vector field ($U[m/s]$) using either the conventional CFD solver (a) and AI-accelerated approach (b) after 10 time steps

The accuracy is also verified with statistical metrics. The first two are correlation coefficients that measure the extent to which two variables tend to change together. These coefficients express the strength and the direction of the relationship. The Pearson correlation estimates the linear relationship between two continuous variables. The Spearman correlation evaluates the monotonic relationship between true and predicted variables. The Pearson correlation ranges from 0.98 for the $10-th$ time step to 1.0 for the converged state. The average Pearson correlation for all the time steps is 0.99. The results are convergent, mainly for the values from the upper bound of the range. The Spearman correlation ranges from 0.96 to 1.0 with an average value of 0.98. It shows a strong monotonic association between the CFD and AI results. The next statistical metric is the root mean square error (RMSE). It is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are. RMSE ranges from 0.041 for the $10-th$ time step to 0.005 for the steady state. The average RMSE for all the validated time steps is 0.023. Since RMSE is an absolute metric, we would like to underline that the velocity vector field is within a range from 0.0 to 1.1. It shows that RMSE is below 5% of the maximum value of the velocity field.

In the next method, we generate histograms for the true and predicted values and calculate the coefficient of determination between them. This method is called histogram equalization. The obtained results range from 87

Finally, we plot a function $y(x)$, where x represents the results received from the CFD solver, and y is the prediction. The results are shown in Fig. 6. The blue dots show the prediction uncertainty. A straight line represents ideal results, where $y(x) = x$. The results confirm our previous conclusion that higher values (the most significant) are predicted with the highest accuracy.

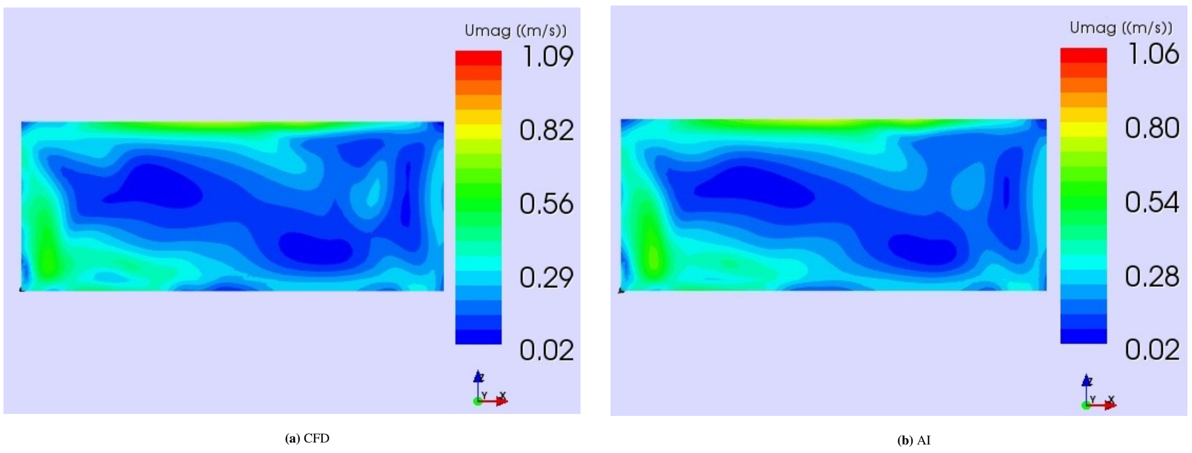

FIGURE 4 Contour plot of the velocity magnitude vector field ($U[m/s]$) using either the conventional CFD solver (a) and AI-accelerated approach (b) after 20 time steps

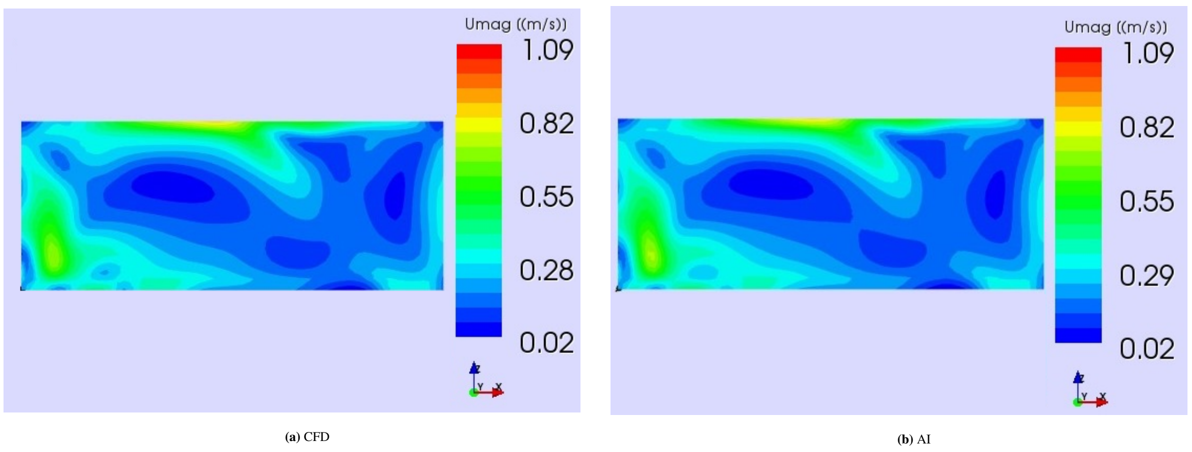

FIGURE 5 Contour plot of the velocity magnitude vector field ($U[m/s]$) using either the conventional CFD solver (a) and AI-accelerated approach (b) in the converged state

Time step	Pearson's coeff.	Spearman's coeff.	RMSE	Histogram equaliz.
10th	0.984	0.969	0.041	97.3%
20th	0.995	0.993	0.023	98.0%
Converged	1.000	1.000	0.005	99.5%

TABLE 2 Accuracy results with statistical metrics

2.3 | Distributed Learning

One of our goals is to evaluate the performance of training the model using many accelerators distributed in several nodes. For this purpose, this study leverages two different strategies which provide support for distributed training. While the first strategy presented is implicitly supported by Tensorflow, the second is a framework for several deep learning libraries, among them, Tensorflow.

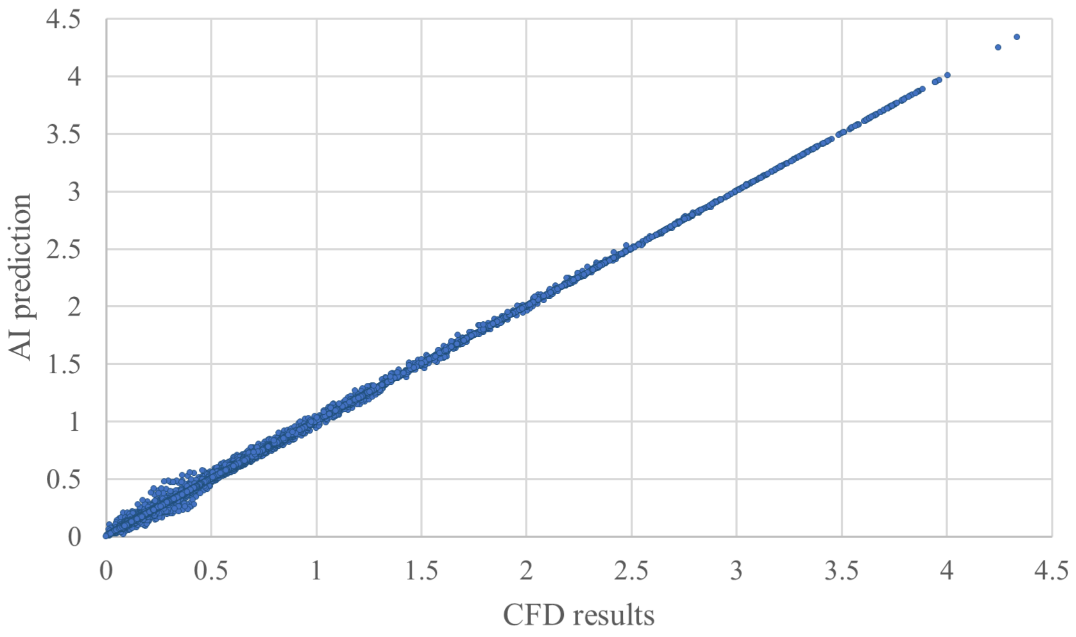

FIGURE 6 Comparison of simulation results for the conventional CFD solver and AI-accelerated approach.

2.3.1 | Tensorflow

Tensorflow[10] is an open-source library for artificial intelligence. It is intended for the training and inference of deep neural networks.

Tensorflow provides native mechanisms of synchronous training across multiple replicas on one or more machines, particularly:

- *MirroredStategy*[#]*:* a variable created with this strategy is a *MirroredVariable* replicated in all the involved GPUs, or CPUs if no GPUs are found. All the CPUs of a machine are treated as a single device (such as a GPU) and a thread per core is spawned for parallelism.
- *Multiworker*[‖]*:* this strategy complements the *MirroredStrategy* with the capability of communicating multiple nodes (*workers*), each with its GPUs. For this purpose, it replicates all variables and computations to each local device. Multiple workers can work together using an all-reduce operation.

2.3.2 | Horovod

Horovod[11] is a distributed learning framework implemented using the Message Passing Interface (MPI) model (although it supports other communication libraries such as Gloo) in an effort to reduce the configuration complexity introduced by the *MultiworkerStrategy* of Tensorflow to operate in computational clusters. Horovod provides support for AI libraries such as PyTorch, MXNet, and Tensorflow. Once a training script has been written for scaling, it can run on any number of GPUs and/or nodes without any further code changes.

Horovod relies on the NVIDIA Collective Communication Library[**] (NCCL) to implement optimized multi-GPU and multi-node communications. Horovod core is based on MPI primitives such as *rank*, or *all-reduce*. In this regard, Horovod documentation informs that when using NCCL, performance will be similar to the original library. However, thanks to the high degree of specialization of MPI in HPC environments, especially in communications, users can expect that their only-CPU training runs faster.

[#]https://www.tensorflow.org/api_docs/python/tf/distribute/MirroredStrategy
[‖]https://www.tensorflow.org/api_docs/python/tf/distribute/MultiWorkerMirroredStrategy
[**]https://developer.nvidia.com/nccl

3 | PERFORMANCE ANALYSIS

In this section, a thorough performance evaluation of the training, as well as the validation of the model, in different scenarios is performed. Results presented in this section have been obtained with the following hardware:

- CFD simulations are executed on Tirant III supercomputer. Tirant III servers are composed each of two sockets Intel Xeon SandyBridge E5-2670 @ 2.6GHz with a total of 16 threads, and 32GB of main memory.
- The predictive model training and inference are performed on the cluster CTE-Power. CTE-Power nodes are equipped each with two processors IBM Power9 8335-GTH @ 2.4GHz with a total of 160 threads, 512GB of main memory, and four GPU NVIDIA V100 with 16GB HBM2. Nodes are interconnected via single Port Mellanox EDR (25Gb/s).

Notice that with CTE-Power not only the neural network training but also the CFD simulation could have been performed. Nevertheless, we opted for using Tirant III for the CFD simulations since the solvers are not implemented for GPU architectures. It would have been translated into a lot of wasted GPU time during their executions, and many users could have seen their research impaired.

Regarding the software stack, CFD simulations have been performed in Tirant III with OpenFOAM v2006 and Intel MPI 2018 Update 3. While the predictive model has been developed on CTE-Power with Python 3.7.4, CUDA 10.2.89, Keras 2.4, Tensorflow-gpu 2.3, Horovod 0.20.3, and OpenMPI 4.0.1.

CFD simulations take an average of 9,000 seconds running on a 16-core node of Tirant III. Moreover, for each of the 131 simulated cases, results have to be post-processed in order to extract and adapt data into a format readable by the predictive model. This postprocessing takes around 2,700 seconds in a single core.

The following showcased times correspond to exclusively the training time, not to the program initialization or the data loading.

3.1 | Only CPU training

As a baseline, an initial study on training only with CTE-Power CPUs is performed on up to four nodes. By default, Tensorflow and Horovod use all the cores with their threads, so the number of spawned processes is set to one per node.

To begin with, the model is fit with Tensorflow. Using a single node, the training time is 76,674 seconds. When enabling distributed training with *multiworker strategy* a series of core dumps aroused because the Tensorflow-GPU version runs without GPUs.

On the contrary, execution times of the Horovod-based model fitting are compiled in Table 3. The table also shows the speedup of Horovod compared to the Tensorflow baseline training time introduced before.

# Nodes	1	2	3	4
Time (s.)	75,346 s.	66,573 s.	58,678 s.	52,908 s.
Speedup	1.02x	1.15x	1.31x	1.45x

TABLE 3 Horovod CPU training time and speedup compared with the single-node Tensorflow training.

Although the speedup increases with the number of nodes, it is far from being linear and too high compared with the GPU-enabled configuration showcased in the next section.

3.2 | GPU-enabled training

In this subsection, the RNN distributed training on GPUs is studied. As it was described in Section 2.3, the distributed training of the model in Tensorflow could be performed either with its native tools (*multiworker strategy*) or with the framework Horovod. For this reason, the training was implemented in both alternatives.

Firstly, distributed training in Tensorflow has been evaluated. Table 4 contains the training results of *multiworker strategy* for different number of nodes involved. In this evaluation, only a GPU per node has been leveraged.

# Nodes	1	2	3	4
Time (s.)	10,803 s.	43,288 s.	38,420 s.	32,928 s.
Speedup	-	0.25x	0.28x	0.33x

TABLE 4 *Multiworker strategy* GPU training time and speedup

However, distributing the workload in several nodes is translated into performance degradation. This is a known issue in CTE-Power for the TCP/socket-based networking communications which are reported to have half the bandwidth. Apart from these issues, the model training still scales along with the number of nodes. The study with *multiworker strategy* is not continued with more GPUs per node due to the high times obtained in this stage. In this regard, henceforth, the results of distributed configurations correspond to Horovod with MPI, compiled to use Infiniband over the Mellanox EDR ports installed in the nodes.

Table 5 contains the training times with different execution configurations concerning the number of nodes and GPUs. The results of *mirrored strategy* for several GPUs in a single node are placed in the second row of the table. The rest of the rows contains the results using Horovod with a different number of nodes and GPUs.

	1 GPU	2 GPUs	3 GPUs	4 GPUs
Tensorflow `mirrored strategy`	10,803 s.	6,703 s.	5,259 s.	4,490 s.
Horovod 1 node	10,931 s.	6,764 s.	5,205 s.	4,492 s.
Horovod 2 nodes	6,825 s.	4,793 s.	4,140 s.	3,585 s.
Horovod 3 nodes	5,453 s.	3,975 s.	3,649 s.	3,652 s.
Horovod 4 nodes	4,917 s.	3,803 s.	3,131 s.	3,261 s.

TABLE 5 GPU training time

Correspondingly, Table 6 shows the speedups of the different configurations compared with Tensorflow on a single GPU (second row and column).

	1 GPU	2 GPUs	3 GPUs	4 GPUs
Tensorflow `mirrored strategy`	1.00x	1.61x	2.05x	2.41x
Horovod 1 node	0.99x	1.60x	2.08x	2.40x
Horovod 2 nodes	1.58x	2.25x	2.61x	3.01x
Horovod 3 nodes	1.98x	2.72x	2.96x	2.96x
Horovod 4 nodes	2.20x	2.84x	3.45x	3.31x

TABLE 6 GPU training speedup

In the previous tables, we can also appreciate that for the same number of processes depending on how they are distributed among the underlying hardware, times vary. The tables reveal that the training operation scales with the amount of leveraged resources. From those results, we can calculate the negative effect of the communication network in different processes layout. Notice that the amount of spawned processes corresponds to the expression $\#Processes = \#Nodes \times \#GPUs$.

Table 7 compiles the communication overhead for the different layouts for the same number of processes. Each cell contains the variation in the percentage of time compared with the symmetric cell on the other side of the diagonal. For instance, two processes in a node using two GPUs run 0.89% faster than the configuration of two nodes using a GPU in each one, which needs 0.9% more time to complete.

Ideally, configurations on the upper triangle should run faster (positive percentages) than in the lower triangle (negative percentages), since fewer nodes involve less communication. However, we can find two cases that run faster than their fewer-nodes counterpart: three nodes and two GPUs in each one; and four nodes with three GPUs each.

	1 GPU	2 GPUs	3 GPUs	4 GPUs
1 node	-	0.89%	4.55%	8.64%
2 nodes	-0.90%	-	-4.15%	5.73%
3 nodes	-4.76%	3.99%	-	-16.64%
4 nodes	-9.46%	-6.08%	14.27%	-

TABLE 7 Communication network effect in Horovod on execution times comparing the same number of processes distributed in different layouts.

In more detail, Figure 7 represents the execution time of layouts for a different number of processes. In the y-axis, the number multiplying on the left is the number of nodes and the number of GPUs on the right.

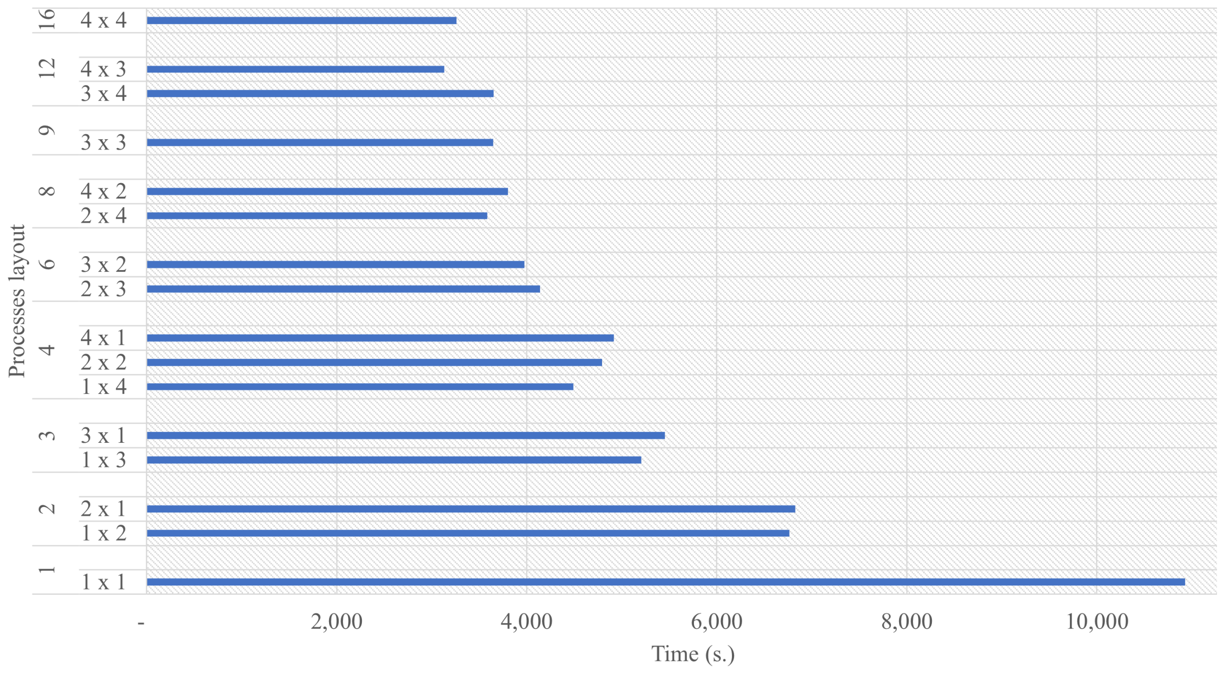

FIGURE 7 Horovod deep neural network training time for different processes configurations and layouts (nodes x processes).

The theoretical trend of reducing execution time with a lower number of nodes for the same amount of processes is followed, except for the previously named cases of six and twelve processes.

In order to study the performance scalability, Figure 8 represents the training time for the lowest time layout of each process configuration. Furthermore, times are associated with the incremental speedup (the time of a configuration divided by the time of the immediately previous one), and the parallel speedup (the time of a configuration divided by the 1-process configuration).

From the chart, we easily can spot the best performance configuration with the lowest time and the highest parallel speedup. Moreover, depending on our needs, we can determine a “sweet spot” configuration that achieves a balance between execution time and resources utilized. The “sweet spot” for this study may be set to the 2-process configuration, where the speedup increment is above 1.5x. And it would probably be in shared environments, however, since all the experiments have been performed using exclusive nodes, the best balance between time and resources can be defined as four processes running in the same node. This configuration shows an incremental speedup of 1.16x and a parallel speedup of 2.43x.

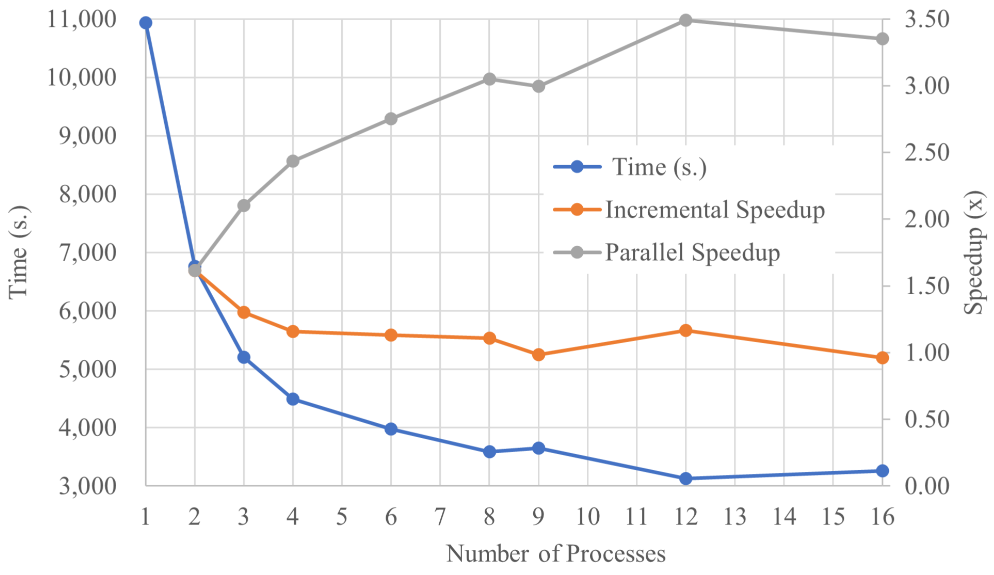

FIGURE 8 Training time and speedups for different processes configurations in Horovod.

3.3 | Inference

The inference time of the evaluation dataset has also been measured. Particularly, Table 8 showcases the prediction times for all the samples in the test dataset for one node, and in the case of the GPU-enabled inference, one GPU.

CPU time	GPU time	GPU Speedup
3,136 s.	1,014 s.	3.09x

TABLE 8 Comparison of inference results.

4 | RELATED WORK

The adaptation of CFD codes to hardware architectures exploring modern compute accelerators is an important issue in the environment of constantly growing needs for computing power. The acceleration of HPC simulations with AI becomes a popular topic these days since it offers much more efficiency than traditional hand-tuning. In other words, AI methods in CFD simulations can reduce the time-to-solution while providing acceptable accuracy[12].

In[6], the authors proposed a method for accelerating a chemical-mixing simulation with an AI model. This work shows that AI generates results for a MixIT tool with 92% accuracy and speedup to 10x compared with the MixIT tool. The performance results are achieved on a single CPU/GPU node and the solution is based on a convolutional neural network (CNN).

The authors in[13] devised a Non-Intrusive Reduced Order Model (NIROM) combining Proper Orthogonal Decomposition (POD) and machine learning techniques. Their results show that the model is capable of predicting the evolution of a turbulent flow in a quasi-steady state. Particularly, their method is trained with 2,000 time steps to provide predictions for the following 2,000 time steps.

A deep learning framework coupled with a physical simulator for RANS models, named CFDNet, is presented in[3]. CFDNet is intended for more generalist simulations, and it always applies Proper Orthogonal Decomposition (POD) methods[14] to reduce the data dimensionality before training the predictive model. In their work, the authors compare several CFD simulations with their coupled model results obtaining similar errors in a reduced time.

A hybrid CFD-AI solver is proposed in[15]. The solver alternates stages of CFD simulation with predictions made by a neural network previously trained with a given model. In this case, the neural network learns spatial-temporal features of the data leveraging convolutional kernels. The novel technique can significantly reduce the simulation time of a transient flow.

In[16], the authors present a novel generative model to synthesize CFD simulations from a set of reduced parameters. The authors show that linear functions are less efficient than their non-linear counterparts. For this reason, deep learning models are promising for representing data in reduced dimensions due to their capability to tailor non-linear functions to input data.

J. Thompson et al.[17] propose a data-driven approach that utilizes deep learning to obtain fast and highly realistic simulations. They use a CNN with a highly tailored architecture to solve the linear system. The key contribution is to incorporate loss training information from multiple time steps and perform various forms of data augmentation.

The authors in[18] introduce an ML framework for the acceleration of RANS to predict steady-state turbulent eddy viscosities, given the initial conditions. As a result, they proposed a framework that is hybridized with ML.

In[19], the authors present a general framework for learning simulation and give a single model implementation that results in state-of-the-art performance across a variety of challenging physical domains, involving fluids, rigid solids, and deformable materials interacting with one another.

5 | CONCLUSION AND FUTURE WORK

In this work, we investigated the distributed training of the RNN model for data-driven CFD simulation. We applied and analyzed two methods including Horovod distribution and the Tensorflow *mirrored strategy* and *multiworker strategy*.

The proposed DL model is validated with an OpenFOAM solver and the results allow us to achieve an accuracy of around 99% based on the histogram equalization of the converged state when predicting single time steps.

We have also examined the performance of the training and inference stages between the CPU and GPU configurations considering different layouts of processes. Our research shows that when comparing performances between no-GPU and GPU scenarios, the training time with a GPU reduces the training time by 7.1x. If the CPU time is compared to the maximum performance configuration time of four nodes and three GPUs per node, the speedup is 24.49x. On the contrary, in the inference stage that is expected to be less concurrent, with a GPU the time decreases by 3.09x.

Besides the performance analysis, another interesting insight about the comparison of native distributed training methods of Tensorflow with Horovod is the user experience. Horovod MPI-enabled leverages the network configuration of the cluster deployed for MPI. In this regard, users are able to exploit all the network potential without further tuning. Nevertheless, irrespective of the network communication issues, while with Horovod the environment is simply set up with a call to `init`, the *multiworker strategy* needs to carefully define the cluster configuration of the nodes and GPUs involved in the execution.

The model presented in this paper and the subsequent performance analysis can be really helpful in hybrid AI-CFD strategies of simulation based on surrogate models or co-simulation such as the presented in[20] or[21] where CFD models are complemented with deep learning surrogate models of the high computational demanding subdomains of the case.

All the codes developed for this study are available in https://github.com/AlejandroGB13/CFD_AI. The datasets can be provided on-demand.

ACKNOWLEDGEMENTS

Researcher Sergio Iserte was supported by postdoctoral fellowship APOSTD/2020/026 from GVA-ESF. CTE-Power cluster of the Barcelona Supercomputing Center (BSC-CNS), and Tirant III cluster of the *Servei d'Informàtica* of the University of Valencia (UV) were leveraged in this research.

References

1. Archibald R, Chow E, D'Azevedo E, et al. Integrating Deep Learning in Domain Sciences at Exascale. In: Nichols J, Verastegui B, Maccabe AB, Hernandez O, Parete-Koon S, Ahearn T., eds. *Driving Scientific and Engineering Discoveries Through the Convergence of HPC, Big Data and AI*Springer International Publishing; 2020; Cham: 35–50.

2. Jouppi NP, Young C, Patil N, Patterson D. A Domain-Specific Architecture for Deep Neural Networks. *Commun. ACM* 2018; 61(9): 50–59. doi: 10.1145/3154484

3. Obiols-Sales O, Vishnu A, Malaya N, Chandramowliswharan A. CFDNet: A Deep Learning-based Accelerator for Fluid Simulations. *Proceedings of the International Conference on Supercomputing* 2020. doi: 10.1145/3392717.3392772

4. Climent J, Basiero L, Martínez-Cuenca R, Berlanga JG, Julián-López B, Chiva S. Biological Reactor Retrofitting Using CFD-ASM Modelling. *Chemical Engineering Journal* 2018; 348.

5. Iserte S, Carratala P, Arnau R, et al. Modeling of Wastewater Treatment Processes with HydroSludge. *Water Environment Research* 2021: 1–38.

6. Rojek K, Wyrzykowski R, Gepner P. AI-Accelerated CFD Simulation Based on OpenFOAM and CPU/GPU Computing. In: Springer International Publishing; 2021: 373–385.

7. Bhatt D, Zhang B, Zuckerman D. Steady-state Simulations Using Weighted Ensemble Path Sampling. *The Journal of Chemical Physics* 2010; 133(1).

8. Maas AL, Hannun AY, Ng AY. Rectifier Nonlinearities Improve Neural Network Acoustic Models. *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing* 2013; 28.

9. Kingma DP, Ba JL. Adam: A Method for Stochastic Optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings* 2015: 1–15.

10. Abadi M, Barham P, Chen J, et al. TensorFlow: A System for Large-Scale Machine Learning. In: OSDI'16. USENIX Association; 2016; USA: 265–283.

11. Sergeev A, Balso MD. Horovod: Fast and Easy Distributed Deep Learning in TensorFlow. *CoRR* 2018; abs/1802.05799.

12. Vinuesa R, Brunton SL. The Potential of Machine Learning to Enhance Computational Fluid Dynamics. *arXiv preprint arXiv:2110.02085* 2021.

13. Xiao D, Heaney CE, Mottet L, et al. A Reduced Order Model for Turbulent Flows in the Urban Environment Using Machine Learning. *Building and Environment* 2019; 148: 323–337.

14. Berkooz G, Holmes P, Lumley JL. The Proper Orthogonal Decomposition in the Analysis of Turbulent Flows. *Annual Review of Fluid Mechanics* 1993; 25(1): 539-575. doi: 10.1146/annurev.fl.25.010193.002543

15. Iserte S, Macías A, Martínez-Cuenca R, Chiva S, Paredes R, Quintana-Ortí ES. Accelerating Urban Scale Simulations Leveraging Local Spatial 3D Structure. *Journal of Computational Science* 2022; 62: 101741. doi: https://doi.org/10.1016/j.jocs.2022.101741

16. Kim B, Azevedo VC, Thuerey N, Kim T, Gross M, Solenthaler B. Deep Fluids: A Generative Network for Parameterized Fluid Simulations. *Computer Graphics Forum* 2019; 38(2): 59-70.

17. Tompson J, Schlachter K, Sprechmann P, Perlin K. Accelerating Eulerian Fluid Simulation with Convolutional Networks. In: ICML'17. JMLR.org; 2017: 3424–3433.

18. Maulik R, Sharma H, Patel S, Lusch B, Jennings E. Accelerating RANS Turbulence Modeling Using Potential Flow and Machine Learning. *arXiv: Fluid Dynamics* 2019.

19. Sanchez-Gonzalez A, Godwin J, Pfaff T, Ying R, Leskovec J, Battaglia PW. Learning to Simulate Complex Physics with Graph Networks. In: ICML'20. ; 2020.

20. Srivastava S, Damodaran M, Khoo BC. Machine Learning Surrogates for Predicting Response of an Aero-structural-sloshing System. *arXiv* 2019.

21. Paul A, Mozaffar M, Yang Z, et al. A Real-time Iterative Machine Learning Approach for Temperature Profile Prediction in Additive Manufacturing Processes. *Proceedings - 2019 IEEE International Conference on Data Science and Advanced Analytics, DSAA 2019* 2019: 541–550.

□