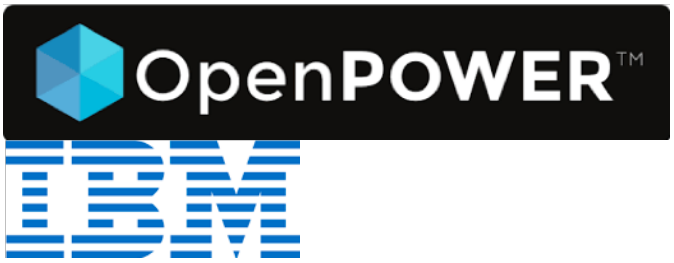


MULTI-GPU ACCELERATION OF PHYSICS-INFORMED GANS IN COMPUTATIONAL FLUID DYNAMICS
OPENPOWER VT HACKATHON PROJECT REPORT
MAY 3, 2019

VERSION 0.1

AUTHORS: WEICHENG XUE, YANG ZENG AND ZHIYI LI

1. EXECUTIVE SUMMARY AND INTRODUCTION

1.1. INTRODUCTION

Traditional Computational Fluid Dynamics (CFD) methods are commonly computational expensive in solving a system of Partial Differential Equations (PDEs) on well refined meshes, especially for turbulent and high Reynolds number flows. Fortunately, the development of machine learning (ML) or deep learning (DL) technique provides us with a completely innovated way to get solutions for some fluid dynamics problems in hours, or even in minutes. This is a multi-disciplinary field crossed by Aerospace Engineering and Data Sciences and has already attracted a lot of attention. Wang [1, 2] and Wu [3] proposed a basic idea of physics-informed machine learning (PIML) and applied a random forest and neural network model to a turbulent flow case. Raissi et al. [4] developed a Gaussian Process (GP) based machine learning method to solve both linear and non-linear systems of differential equations, and also imported some physical constraints in their follow-up work [5]. King [6] applied a data-driven Gaussian process (GP) regression to the wind plant flow model, and utilized a gradient-based optimization for the high-dimensional PDEs to improve accuracy. Rohit [7] used the deep neural network as surrogate models for solving high-dimensional uncertainty quantification.

Having the capability of generating false data, the generative adversarial networks (GANs) [8] have been regarded as one of the most promising deep learning methods. GANs is composed of two neural networks, one of which is generative neural network and the other is discriminative neural network so that the two neural networks can compete with each other to generate some false data which mimic the true data. However, the training process of GANs may be unstable and expensive, so a lot of variants were proposed to improve the accuracy and reduce the training time. In order to address the issue of vanishing gradients of original loss function (sigmoid cross entropy), Mao [9] proposed Least Square GANs. Nowozin [10] replaced the Jensen-Shannon (JS) divergence of original GANs with f-divergence function and proposed f-GAN, in which various choices of divergence of GANs were suggested to improve the training quality and complexity. Wasserstein GAN (WGAN) [11] and WGAN-GP [12] were proposed to address the vanishing gradients of the JS divergence issue. The Earth-Mover distance was used to ensure gradient exist everywhere in WGAN and WGAN-GP.

When applying GANs to physical problems, there may be some serious issues. The first issue is that the generated data may not satisfy physical conservation laws or constraints due to its poor ability to extract complex physical features correctly. The second issue is that the training process may become more difficult and time-consuming after importing some physical constraints to the model. Some researchers have already proposed some insightful methods to address the two issues. Stinis [13] integrated the residual errors which stemmed from substituting the generated data into the constrained equation to the inputs of discriminator, thus physical constraints can be satisfied. Also, some noise contents were added to the constrained residuals to address the instability issue caused by the inconsistent training speed between generator and discriminator. Yang [14] proposed a physics-informed GANs (PI-GANs) to solve stochastic differential equations (SDEs) more accurately. The coefficients, solution, force term and boundary conditions of the stochastic differential equations (SDEs) were all integrated as the inputs of discriminator, thus the stochastic physical information described by SDEs can be included when training. Yang [15] proposed a physics-informed generative model which uses an entropy regularization for data generation to include physical constraints.

1.2. PROJECT EXECUTIVE SUMMARY

Our project is focused on two aspects. The first aspect is about using a physics-informed GANs model (PI-GANs) to simulate a family of potential flows (uniform flow + source flow). In PI-GANs, physical information such as the mass conservation law (for an incompressible flow) is

integrated to GANs as a penalty term. This penalty term embedded to the generator to enforce the generated data inform the physical information. Although the flow itself studied in this project is elementary, the goal of this project is to see whether this PI-GANs can generate some “true” flows that mimic the real flows satisfying physical constraints, and it is a good start for future training on more complicated flows. The second aspect is to accelerate the serial PI-GANs code developed and scale this code to multiple GPUs, both intra-node and inter-node.
<fill in a summary on your specifc project>

2. HACKATHON REQUIREMENTS

The **VT OpenPower Hackathon** was a first-of-its-kind AI Hackathon at Virginia Tech focusing on accelerating AI model training with the ambition to reap and share the power of AI technologies with the VT student community.

The OpenPOWER Foundation Academic Workgroup, in collaboration with the Faculty of Virginia Tech and Advanced Research Computing with support from the IBM Systems Client Experience Center organized an AI Hackathon held April 19th, 2019 to May 3rd, 2019. The initiation and problem statement were given Monday, April 19st on Hackathon Launch Day. The Hackathon ended with power hackers submitting projects presented and judged on the Hackathon Awards Day Friday, May 3rd, 2019.

The theme of the Hackathon was **"Accelerating AI at Scale"**. The goal of the hackathon was to demonstrate that optimizing the time it takes to train a model can have a significant impact on time to results. Students were encouraged to take an existing model they have trained to a given accuracy on Power Systems and demonstrate how their training time can be accelerated using the latest in state-of-the-art techniques on the same Power Systems to the same accuracy.

The aim of the competition was to bring together students from different creative fields to accelerate the training of AI model prototypes using AI tools during a limited amount of time. The competition tested the students' skill of fortitude and their ability to work under pressure.

3. PROJECT DESCRIPTION

3.1. OVERVIEW

3.1.1. GANs

Goodfellow et al. [8] firstly proposed GANs in 2014. The objective function of GANs used in their work was given as:

$$\min_G \max_D V(D,G) = E_{X\sim p_{data}(X)}[\log D(X)] + E_{z\sim p_z(z)}[\log(1-D(G(z))] \quad (1)$$

where *G* and *D* are generator and discriminator, respectively, *z* the latent variables which are sampled from a given distribution $p_z(z)$ such as a uniform and Gaussian distribution, *X* the given training samples, $p_{data}(X)$ the distribution of these training samples. The objective of GANs is to maximize $D(X)$ and minimize $D(G(z))$ at the same time. Since a generator *G* which is a neural network is able to transfer the given distribution p_z to the distribution of training samples $p_{data}(x)$ through training, generator *G* can generate some pseudo-realistic data. In order to make this pseudo-realistic data approach the training samples, another neural network called discriminator *D* is used to discriminate the pseudo-realistic data generated by generator *G* from training samples. The max*D* is to train the discriminator to make the outputs of real data $D(X)$ approach to 1 and make the output of fake data $D(G(z))$ approach to 0.

Due to the possibility of vanishing gradient, the original GANs is difficult to be trained. Arjovsky et al. [11] addressed this issue by using the Earth-Mover (Wasserstein-1) distance W(*q*, *p*) whose gradient existed everywhere as the loss function, and the model is called WGANs. The objective function of Wasserstein-1 distance can be defined as:

$$\min_G \max_{D\in\Omega} V(D,G) = E_{X\sim p_{data}(X)}[D(X)] - E_{z\sim p_z(z)}[D(G(z))] \quad (2)$$

where Ω denotes the set which consists of 1-Lipschitz function. In WGANs, the weights of generator are limited in a compact space with clipping. Gulrajani et al. [12] later pointed out that the poorly designed compact space would result in vanishing or exploding gradient. In order to bypass the weight-clipped method, they proposed WGAN-GP [12]. In this new model, a gradient penalty function is used to enforce the Lipschiz constraint. The new loss function to be optimized is:

$$V(D,G) = E_{z\sim p_z(z)}[D(G(z))] - E_{X\sim p_{data}(X)}[D(X)] - \beta E_{\hat{X}\sim P(\hat{X})}[(\left\|\nabla_{\hat{X}} D(\hat{X})\right\|_2 - 1)^2] \quad (3)$$

where $\hat{X}$ is the sample which is uniformly generated between the generated data $G(z)$ and the training sample *X*, β the weight of gradient penalty.

Until this point, physical constraints have not yet been considered. Physical constraints can be usually denoted as:

$$H(X) \le 0 \quad (4)$$

where $H(\cdot)$ can be a system of differential equations or more complex physical constraints. A General GANs usually cannot extract complex physical features from training samples, and thus some improvements were made. To evaluate whether the generated data of GANS satisfies some physical constraints, we define a term:

$$L_{cons} = E_{z\sim p_z(z)}[\max(H(G(z)),0)] \quad (5)$$

where L_{cons} is always non-negative and a smaller L_{cons} indicates that the model trained satisfies the physical constraints better. This term is integrated into the loss function of GANS through:

$$V(G,D) + \lambda L_{cons} \quad (6)$$

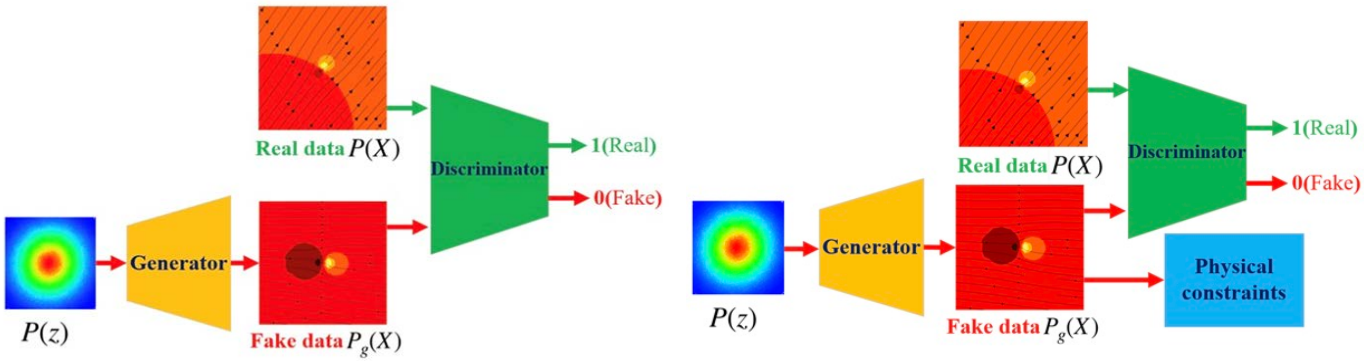
where λ is a tuning parameter. In the beginning of training, the generated data can be extremely diffusing. This would lead to an extremely large or even infinite value of L_{cons}. However, via training, the generated data approach the training samples and this would lead to much smaller order of magnitude of penalty term than that in the beginning. In order to balance the contribution

of original loss function and penalty term, an improved penalty term was added into the loss function:

$$V(G,D)+\lambda\log(L_{cons}+1) \tag{7}$$

with this penalty term, physical constraints can be included in our model without getting the training to be too much unstable, especially in the beginning. This improved model is called Physical-informed GANs (PI-GANs).

A comparison of standard GANs and PI-GANs in our application is shown in Fig. 1.

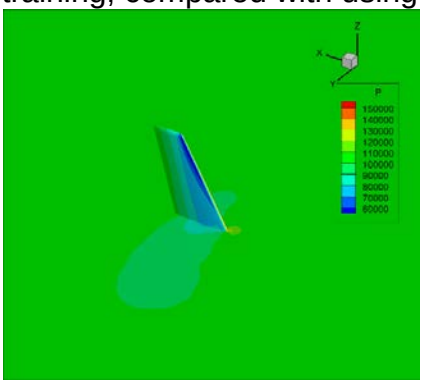


a) Standard GANs b) PI-GANs

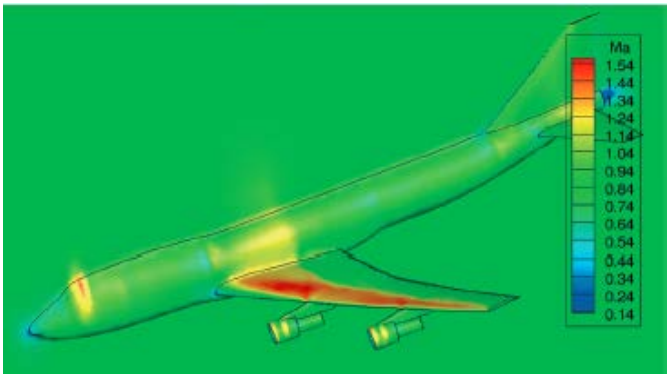
Figure 1. A comparison between standard GANs and PI-GANs

3.1.2. CFD Applications

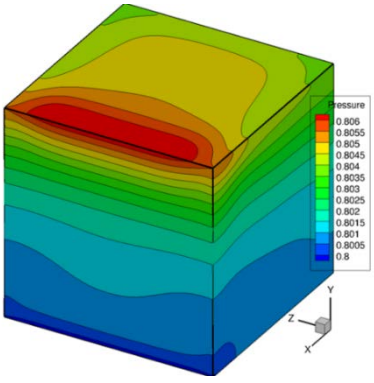
Fig. 2 shows some applications of using CFD. To get the numerical solutions, we usually need to iteratively solve a system of nonlinear partial differential equations. Iteratively solving these equations on well refined meshes can cost a very long time, therefore researchers in the CFD area have been thinking about new ways to get numerical solutions with high accuracy faster. High Performance Computing (HPC) has been applied for a lot of fluid problems in the CFD area and has drawn a lot of attentions. With the development of GANs, specifically PI-GANS, researchers in the traditional CFD area can solve problems numerically much faster through GANs modeling training, compared with using traditional CFD methods.



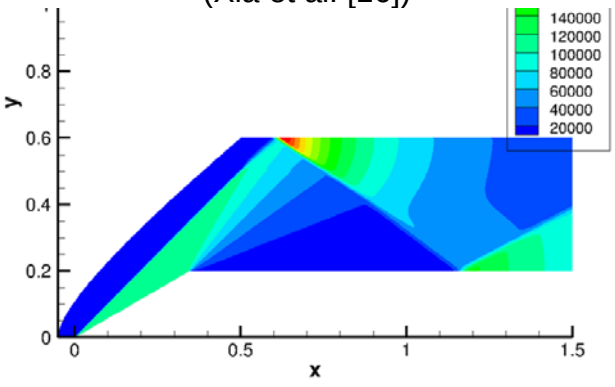
a) Transonic wing flow



b) Boeing 747 aircraft flow (Xia et al. [16])



c) 3D Bouyancy driven cavity flow



d) 2D Supersonic front step flow

Figure. 2 Some applications in the CFD area

3.1.3. PARALLELISMS

Even though PI-GANS provide Aerospace Engineering researchers with a good way to solve PDEs faster than before, it is still computational expensive for complicated case having millions of cells or elements to be converged. We are seeking ways to accelerate the code on heterogeneous systems (with CPU/GPU the most popular). To accelerate a ML/DL code, there are mainly two kinds of parallelisms, model parallelism and data parallelism. At first glance, model parallelism is not our favorite and focus as its design is very likely to be tailored to model selected and its scaling to a large number of nodes may be poor due to a lack of generality. For model parallelism, different devices handle different parts in the graph and all batches of data pass through all devices. Adding more devices may easily cause the efficiency of previous parallelism design to drop and also how to add more devices remain a big question. While for data parallelism, it is a more general method and it can have different forms: in-graph replication, between-graph replication, synchronous and asynchronous training. For data parallelism, multiple GPUs run the same code, and each GPU is feed with different data. To be more specific, both in-graph replication and between-graph replication can be either synchronous or asynchronous, so generally there are four combinations in total. The differences between in-graph and between-graph replication can be divided into three aspects. First, for in-graph replication, all instances replicate the whole graph while between-graph just replicate part of the whole graph so that different instances have different graphs. Second, between-graph usually has multiple sessions and multiple clients while in-graph replication only has one. This brings a lot of advantages when we want to distribute the total task to multiple clients, and clients can be easily added to or removed from the task queue. The difference between synchronous and asynchronous training is whether there is a synchronization between instances to compute the averaged gradients on the controller. In synchronous training, every device shares the same weight so that synchronization should be placed. While for asynchronous training, weights on different devices can be slightly different and different devices can update weights without locking to each other. Either CPU or GPU can be used as controller, depending on the model, network, etc. Fig. 2 shows the comparison between different forms of parallelisms mentioned above.

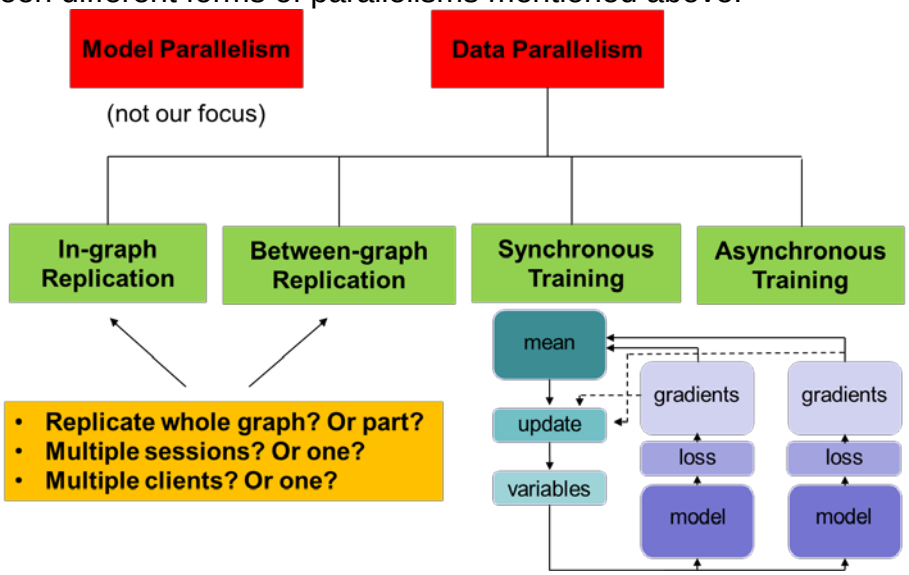


Figure 2. Comparison between different forms of parallelisms

<your project overview: the science behind it and goals of the model>

3.2. DATASETS

The data are generated by analytically solving the velocity field which is a superposition of two kinds of potential flows: uniform and source flow, and then solving the Bournoulli's equation to get the pressure filed. In our project, the PI-GANS model needs to be trained on the velocity field and pressure field at the same time. The velocity field of the flow is given by a complex potential function:

$$F(z) = ce^{-i\alpha z} + \frac{m}{2\pi}\log(z - z_0) \tag{8}$$

where *z=x+iy*, *c* determining the magnitude of velocity and α the flow direction angle of the uniform flow, *m* denoting the strength of the source flow, z_0 the position of the source. Using Bournoulli's equation, we can also get the pressure field:

$$p = p_{\text{inf}} + \rho\frac{V_{\text{inf}}^2}{2} - \rho\frac{V^2}{2} \tag{9}$$

where p_{inf} and V_{inf} are the pressure and velocity magnitude at infinity, and *p* and V representing the pressure and velocity magnitude at any locations. Since this family of flow is an incompressible flow, density ρ is a constant. We set ρ=1 kg/m^3 in this project.

The physical constraint used in this project is that the divergence of the velocity field should be 0 for an impressible flow, that is,

$$\nabla \cdot \vec{V} = 0 \tag{10}$$

where $\vec{V}$ is the velocity vector.

There are some parameters which need to be defined for the training samples. The parameters *c*, α and *m* are sampled from independent Gaussian distribution *c*~*N*(4, 0.4), α ~*N*(0, π/4) and *m*~*N*(1, 0.2) to generate the velocity field and pressure field for 20000 samples. Cartesian grids with mesh sizes of 32x32, 64x64 and 128x128 are all generated for this project. Input samples are normalized and compressed into [-1, 1] for all inputs.
<the datasets you used, their description, sources, limitations, transformations you made, etc>

3.3. MODELS

The configurations of the generator and discriminator used in our project are given in Table 1. and Table. 2. We used physics-informed WGAN-GP to stabilize the training.

Table 1. Discriminator Configuration

Input layer	32x32, 64x64 or 128x128
1st convolutional layer	5x5, strides of 2, 64 feature maps, BN, ReLU
2nd convolutional layer	5x5, strides of 2, 64 feature maps, BN, ReLU
Additional layer if sample size ≥64	5x5, strides of 2, 64 feature maps, BN, ReLU
Additional layer if sample size ≥128	5x5, strides of 2, 64 feature maps, BN, ReLU
Last convolutional layer	4x4, strides of 2, 1 feature map, softmax

Table 2. Generator Configuration

Input layer	100
1st convolutional layer	4x4, strides of 2, 64 feature maps, BN, ReLU
2nd convolutional layer	5x5, strides of 2, 64 feature maps, BN, ReLU
Additional layer if sample size ≥64	5x5, strides of 2, 64 feature maps, BN, ReLU
Additional layer if sample size ≥128	5x5, strides of 2, 64 feature maps, BN, ReLU
Last convolutional layer	5x5, strides of 2, 2 feature maps, tanh

<the models you used, why those were picked, code (can also add to appendix), etc >

3.4. METRICS

The divergence of velocity is used as a metric to determine whether a training is converged or not. As mentioned earlier, theoretically divergence should be 0 for an ideal potential flow. Considering that there are many data points for every sample, we used a *L2* norm of velocity divergence as the convergence metric in our project. To prove that using velocity divergence is

better for a physical problem like ours, a comparison between standard GANs and PI-GANs convergence history is given in Fig. 3. As can be seen, standard GANs has a difficulty to extract the physical feature correctly and the generated samples cannot satisfy the mass conservation law very well. In comparison, PI-GANs ensures that the divergence of velocity to be close to 0, which means that the samples generated are more “real”. In addition, it can be found that the divergence curve of PI-GANs goes down more smoothly than that of the standard GANs, also indicating that the PI-GANs may help to stabilize the training in problems like ours, with the import of some physical constraints.

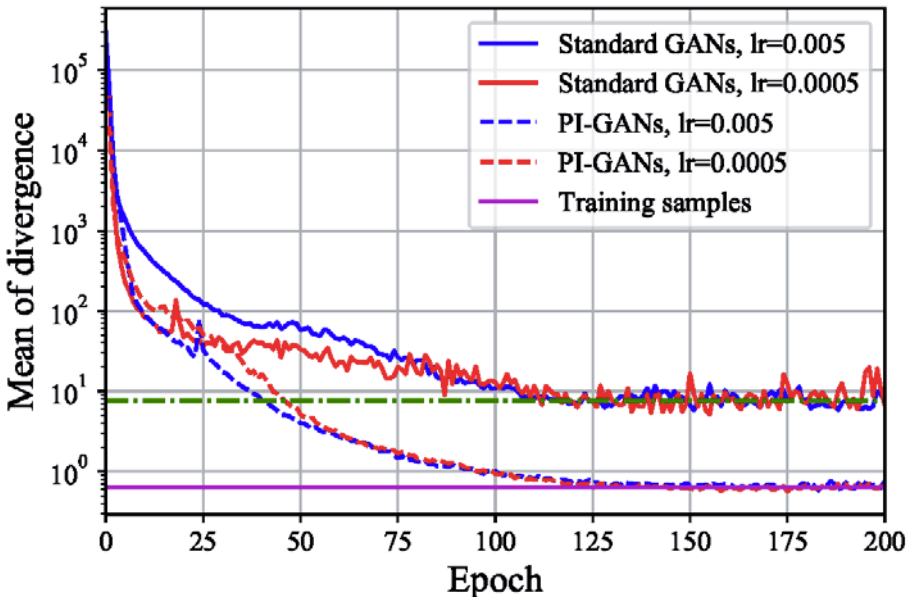


Figure 3. Divergence history comparison between using standard GANs and PI-GANs

< there is a separate section on results so this section is just to describe what the model outputs>

4. HACKATHON SYSTEMS USED

DESCRIPTION OF SYSTEMS USED FOR PROJECT ACTIVITIES

Huckleberry is a high performance computing system at Virginia Tech, which is targeted at deep learning applications. Our tests are mainly done on huckleberry GPU nodes.
<can update this section with architecture diagrams, etc VT ARC typically uses to describe environment>

4.1. IBM SYSTEMS

The following systems were used to perform the benchmark measurements and tests:

Table 3: IBM benchmark systems used

Name	Location	Technical characteristics
Huckleberry	Blacksburg, VA	8 nodes IBM System SL822 for HPC 2 x IBM POWER8 (20 cores)) 8 x 32 GB (256 GB) DDR4 RAM 4xNVIDIA V100 GPU Infiniband Mellanox Adapter 100Gb EDR

4.2. SOFTWARE

Table 4 : IBM benchmark software used

Name	Location	Technical characteristics
Huckleberry	Blacksburg, VA	Operating System: Compiler: C/C++: Fortran: PowerAI: 1.6.0 Other Software:

4.3. OTHER SYSTEM NOTES

- Clock frequency policy: There were no special adjustments made to clock frequency.
- Simultaneous Multi Threading policy: All the systems have SMT technology activated: SMT=1 is used unless noted differently. With SMT=2, the operating system sees twice the number of physical cores as available processors: for a 20 physical cores node, Linux reports 40 cores.

<can add anything specific on HW/SW nodes from your experiements>

5. HACKATHON RESULTS

FOR EACH OF THE TESTED SCENARIOS, THE RESULTS OBTAINED ON THE HACKATHON SYSTEMS IS PRESENTED.

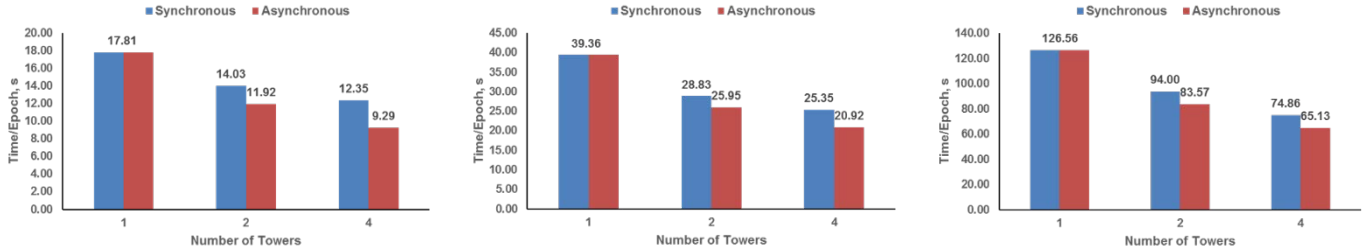
5.1. SOME CONSIDERATIONS

There are a lot of considerations for this project. The most important factor considered is scaling. As mentioned earlier, data parallelism is a more general way of accelerating a code, and should outperform model parallelism in terms of scaling. Also, data parallelism is much easier in terms of the ease of programming. Latency should be an important factor to be considered, especially in compute-intensive but latency-bounded problems like ours. There are some ways to hide latency such as pipelining the input of data through prefetching. We could also train large problems to compensate for the latency overhead. In the multi-tower fashion, we need to consider whether to use CPU or GPU as a controller. Finally, we want to write our code in a distributed training way and then scale it to more GPUs or across nodes, which requires interconnection and communications. Another idea for the distributed is to use a MPI implementation (such as horovod and IBM ddl) to deal with send/recv, broadcast, etc., which should be much easier.

5.2. MULTI-TOWER REPLICATION

5.2.1. CPU # 0 AS THE CONTROLLER

One tower is placed on one GPU and it is one copy of the model, so it is in-graph replication training. Both synchronous and asynchronous training were used for the multi-tower fashion. In this part, CPU # 0 is set to be the controller. A controller can track all variables in the graph. The results are given in Fig. 4. It can be concluded that asynchronous training is always faster than synchronous training. If using 2 GPUs on a 128x128 size problem, the efficiency of synchronous training is 67.32%, while the asynchronous efficiency is 75.72%. However, the scaling efficiency on 4 GPUs is poor, with the synchronous training efficiency to be 42.47%, and the asynchronous training efficiency to be 48.58%. The low efficiency is very likely caused by latency, which can be justified by checking the GPU utilization (just a little above 50%). This also indicates that multi-tower replication cannot scale to many GPUs, and it also cannot scale across nodes. This method requires users to split the minimize step of the minimizer into compute gradients (performed on the device) and apply gradients (running on the controller).

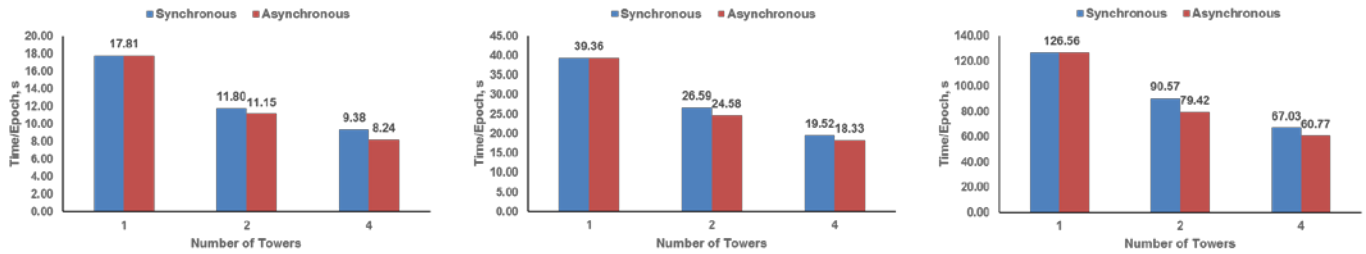


a) 32x32 mesh b) 64x64 mesh c) 128x128 mesh

Figure 4. Multi-Tower Replication Performance (CPU # 0 as the controller)

5.2.2. GPU # 1 AS THE CONTROLLER

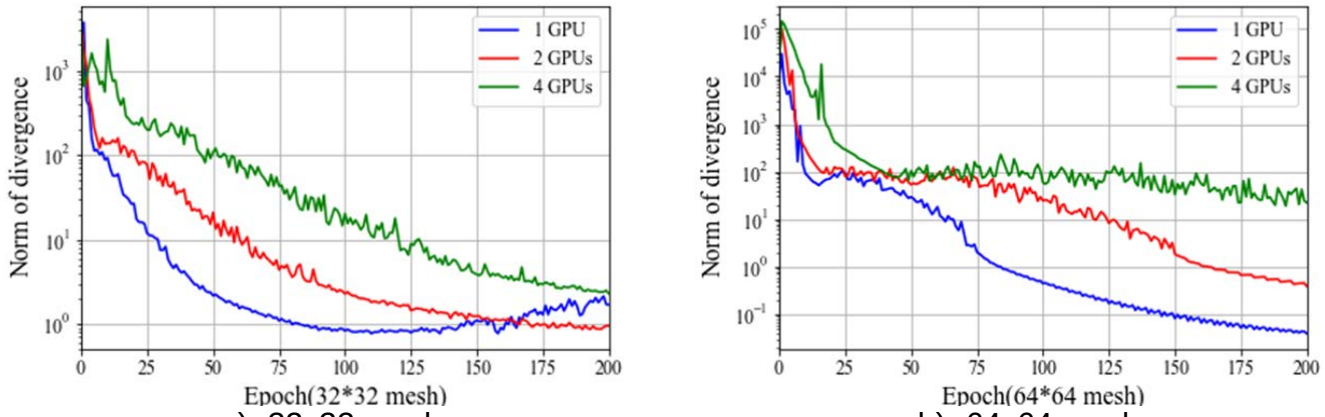
We can also set a GPU to be the controller. In this project, we set GPU # 1 to be the controller. This change requires inter-device transfer as weights are stored on GPUs. Fortunately, huckleberry has NVLink supported so the latency between CPU and GPU can be removed, by using RDMA and GPUDirect P2P transfer. The performance results are shown in Fig. 5. Similarly to Fig. 4, asynchronous training is always faster. We could also see that the efficiency is improved by 3%~12%, compared with setting CPU # 0 to be the controller. However, this may cause some issues for some other applications on a different platform. One potential issue of setting a GPU to be the controller may cause a GPU to be busy and make training to become slower, and it is pretty much centralized. GPU # 1 in our project act as both servers and workers at the same time.



a) 32x32 mesh b) 64x64 mesh c) 128x128 mesh

Figure 5. Multi-Tower Replication Performance (GPU # 1 as the controller)

Fig. 6 shows the norm of divergence history obtained from our results. As mentioned earlier, velocity divergence denotes the velocity's field source strength. For a theoretical potential flow, it should be 0. Therefore, we can use it as a convergence metric to judge how "true" of the pressure field and velocity field are generated by our PI-GANs. In this PI-GANs application, the norm of divergence going down faster means better. On the 32x32 mesh, all the divergence norms declines at an equivalent pace, so that speedup translates easily to faster training. If using 2GPUs, then the number of epochs for convergence is about half as much as that with 1 GPU. On the 64x64 mesh, 4 GPUs converges more slowly compared with using 2 GPUs and 1 GPU, which means adding too many GPUs causes the training slower. The reason for this may be that the traditional averaging of gradients is not good enough. We may need to increase the batch size to get more accurate estimate of the gradients or seek a better way to get more accurate gradients.



a) 32x32 mesh b) 64x64 mesh

Figure 6. Norm of divergence history

Fig. 7 shows the effect of asynchronous training on convergence. It is found that asynchronous training can be more unstable initially, but then goes down well later. This indicates that asynchronous training achieves comparable convergence drop rate, compared with synchronous training.

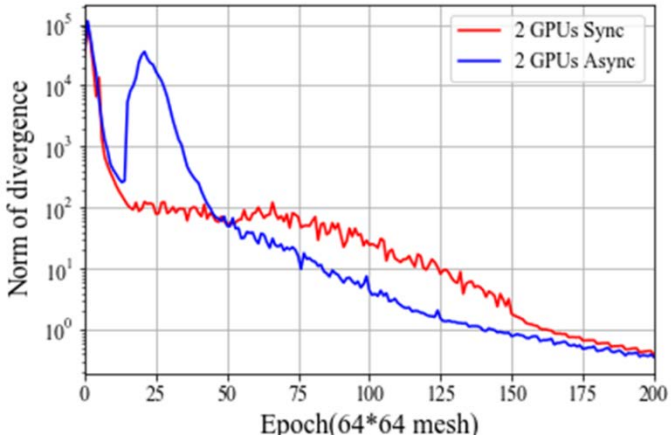


Figure 7. Synchronous training and asynchronous training history

Fig. 8 shows velocity field contours and streamlines generated by our PI-GANs. The flow matches the real flow very well.

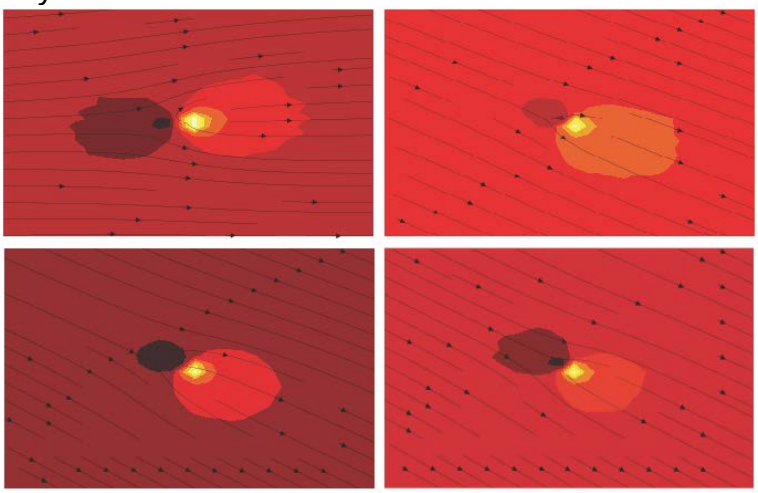


Figure 8. Four samples generated by PI-GANs

5.3. TENSORFLOW DISTRIBUTED TRAINING

5.3.1. EXECUTION

Take the most complicated execution in our project for an example. We used 2 parameter servers and 4 worker servers. On different servers, we used the commands (Supposing the node we used were hu007 and hu009):

On server 1,
python Parallel_PIGANs_PF_dataset.py –ps_hosts=hu007:2222,hu009:2222 --worker_hosts=hu007:2223, hu007:2224,hu009:2223,hu009:2224 –job_name=ps –task_index=0

On server 2,
python Parallel_PIGANs_PF_dataset.py –ps_hosts=hu009:2222,hu009:2222 --worker_hosts=hu009:2223, hu009:2224,hu009:2223,hu009:2224 –job_name=ps –task_index=1

On worker 1,
python Parallel_PIGANs_PF_dataset.py –ps_hosts=hu007:2222,hu009:2222 --worker_hosts=hu007:2223, hu007:2224,hu009:2223,hu009:2224 –job_name=worker –task_index=0

On worker 2,
python Parallel_PIGANs_PF_dataset.py –ps_hosts=hu007:2222,hu009:2222 --worker_hosts=hu007:2223, hu007:2224,hu009:2223,hu009:2224 –job_name=worker –task_index=1

On worker 3,
python Parallel_PIGANs_PF_dataset.py –ps_hosts=hu007:2222,hu009:2222 --worker_hosts=hu007:2223, hu007:2224,hu009:2223,hu009:2224 –job_name=worker –task_index=2

On worker 4,
python Parallel_PIGANs_PF_dataset.py –ps_hosts=hu007:2222,hu009:2222 --worker_hosts=hu007:2223, hu007:2224,hu009:2223,hu009:2224 –job_name=worker –task_index=3

The ClusterSpec was set as:

```
tf.train.CluseterSpec({
   “worker”: [
        “hu007:2223”,
        “hu007:2224”,
        “hu009:2223”,
        “hu009:2224”,
   ]
   “ps”:[
        “hu007:2222”,
        “hu009:2222”
   ]
})
```

5.3.2. RESULTS

After scaling our code within a node using the multi-tower fashion, we moved forward to scale our code across nodes. Tensorflow offers users a standard graph-way to realize it. Fig. 8 shows our understanding of the standard Tensorflow distributed training. We need to setup the ClusterSpec objects, which contains the real sever instance’s addresses and ports.

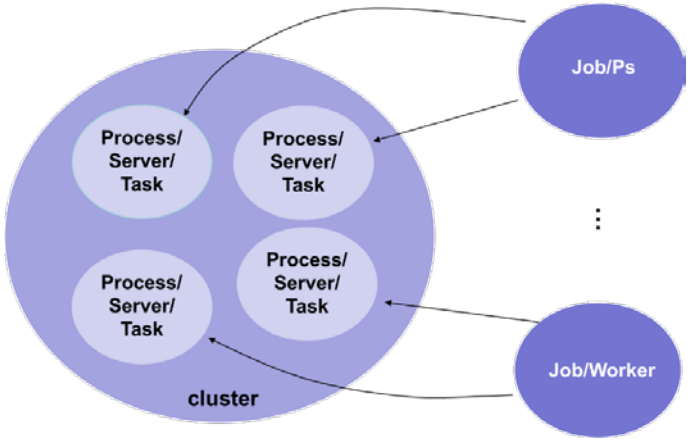


Figure 8. Standard Tensorflow distributed training model

Fig. 9 shows the in-graph/between-graph & asynchronous training time of our code. Some details about running different numbers of GPUs for Fig. 9 is given in Table. 5. If using more GPUs, then the performance results get more unstable, possibly due to the network connections, so an averaged result is given. From Fig. 9, the parallel efficiency is much higher than that in the multi-tower implementation. If using 4 GPUs, the efficiency can reach 85.61% for intra-node scaling and 57,40%, respectively, and if only using 2 GPUs, then the efficiency is 87.90 and 85.82%, respectively. There should be a note that users interested in using standard tensorflow distributed training need to very careful in setting their GPUs as each instance tries to claim all memory on all visible GPUs.

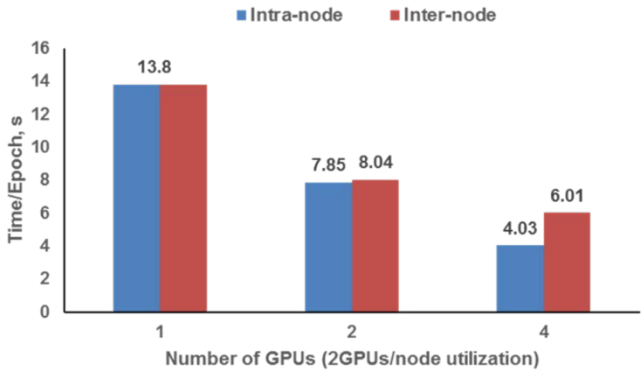


Figure 9. In-graph/Between-graph & Asynchronous Training Time

Table 5. Details of resources used for Figure. 9

Number of GPUs	Number of servers	Intra/Inter node	Node utilization
1 GPU	1 ps + 1 worker	Intra-node	(1 CPU + 1 GPU) x 1 node
2 GPUs	1 ps + 2 workers	Intra-node	(1 CPU + 2 GPUs) x 1 node
2 GPUs	1 ps + 2 workers	Inter-ndoe	1 CPU + 1 GPU x 2 node
4 GPUs	2 ps + 4 workers	Intra-node	(1 CPU + 4 GPUs) x 1 node
4 GPUs	2 ps + 4 workers	Inter-node	(1 CPU + 2 GPUs) x 2 nodes

5.4. More Thoughts and Future Work

We also tried horovod and IBM Power machine built-in ddl package. In comparison with standard TensorFlow distributed training, horovod and ddlrun are much easier to be used, and they are very high-level and efficient tools for distributed training. They are both implemented using a lot of MPI-similar concepts, and are more straightforward and friendly. For horovod, we tried our best with the help from ARC staffs, but could still not install it correctly. We also think that Tensorflow should make distributed training much easier, as users may need to manually specify cluster specifications, which is tedious. However, it is good to scale a code to more nodes and worth using. We also wrote our code by using IBM ddl but the code always hang without moving forward. We are waiting for response from IBM as we requested a help from them.

There are other aspects in which we can think of to further optimize our code. We could try to use NVVP to trace our code and make more optimizations. We could also apply a more heterogeneous CPU/GPU computing method to our code as most CPUs idle when using GPUs to accelerate our code. In this way, very node can contribute their compute potential up to its limit.

REFERENCES

[1] J.-X. Wang, J. Huang, L. Duan, H. Xiao, Prediction of Reynolds stresses in high-Mach-number turbulent boundary layers using physics-informed machine learning, Theoretical and Computational Fluid Dynamics 33 (1) (2019) 1-19.

[2] J.-X. Wang, J.-L. Wu, H. Xiao, Physics-informed machine learning approach for reconstructing reynolds stress modeling discrepancies based on dns data, Physical Review Fluids 2 (3) (2017) 034603.

[3] J.-L. Wu, H. Xiao, E. Paterson, Physics-informed machine learning approach for augmenting turbulence models: A comprehensive framework, Physical Review Fluids 3 (7) (2018) 074602.

[4] M. Raissi, P. Perdikaris, G. E. Karniadakis, Machine learning of linear differential equations using gaussian processes, Journal of Computational Physics 348 (2017) 683-693.

[5] M. Raissi, G. E. Karniadakis, Hidden physics models: Machine learning of nonlinear partial di_erential equations, Journal of Computational Physics 357 (2018) 125-141.

[6] R. N. King, C. Adcock, J. Annoni, K. Dykes, Data-driven machine learning for wind plant ow modeling, in: Journal of Physics: Conference Series, Vol. 1037, IOP Publishing, 2018, p. 072004.

[7] R. Tripathy, I. Bilionis, Deep uq: Learning deep neural network surrogate models for high dimensional uncertainty quanti_cation, arXiv preprint arXiv:1802.00850.

[8] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial nets, in: Advances in neural information processing systems, 2014, pp. 2672-2680.

[9] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang, S. P. Smolley, Least squares generative adversarial networks, in: 2017 IEEE International Conference on Computer Vision (ICCV), IEEE, 2017, pp. 2813-2821.

[10] S. Nowozin, B. Cseke, R. Tomioka, f-gan: Training generative neural samplers using variational divergence minimization, in: Advances in Neural Information Processing Systems, 2016, pp. 271-279.

[11] M. Arjovsky, S. Chintala, L. Bottou, Wasserstein gan, arXiv preprint arXiv:1701.07875.

[12] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, A. C. Courville, Improved training of wasserstein gans, in: Advances in Neural Information Processing Systems, 2017, pp. 5767-5777.

[13] P. Stinis, T. Hagge, A. M. Tartakovsky, E. Yeung, Enforcing constraints for interpolation and extrapolation in generative adversarial networks, arXiv preprint arXiv:1803.08182.

[14] L. Yang, D. Zhang, G. E. Karniadakis, Physics-informed generative adversarial networks for stochastic di_erential equations, arXiv preprint arXiv:1811.02033.

[15] Y. Yang, P. Perdikaris, Physics-informed deep generative models, arXiv preprint arXiv:1812.03511.

[16] Y. D. Xia, J. L. Lou, H. Luo, J. Edwards, F. Mueller, OpenACC acceleration of an unstructured CFD solver based on a reconstructed discontinuous Galerkin method for compressible flows, International Journal for Numerical Methods in Fluids.

APPENDIX

<<additional details can go here – dataset samples. Model code, etc>