Distributed Training of Generative Adversarial Networks for fast detector simulation

S. Vallecorsa CERN
F. Carminati CERN
G. Khattak CERN
D. Podareanu SURFsara
V. Codreanu SURFsara
V. Saletore Intel
H. Pabst Intel

***Abstract*—The simulation of the interaction of particles in High Energy Physics detectors is a computing intensive task. Since some level of approximation is acceptable, it is possible to implement *fast simulation* simplified models that have the advantage of being less computationally intensive. Here we present a fast simulation based on Generative Adversarial Networks (GANs). The model is constructed from a generative network describing the detector response and a discriminative network, trained in adversarial manner. The adversarial training process is compute-intensive and the application of a distributed approach becomes particularly important. We present scaling results of a data-parallel approach to distribute GANs training across multiple nodes from TACC's Stampede2. The efficiency achieved was above 94% when going from 1 to 128 Xeon Scalable Processor nodes. We report on the accuracy of the generated samples and on the scaling of time-to-solution. We demonstrate how HPC installations could be utilized to globally optimize this kind of models leading to quicker research cycles and experimentation, thanks to their large computation power and excellent connectivity.**

***Index Terms*—High Performance Computing, High Energy Physics Simulations, Deep Neural Networks, Generative Adversarial Networks, detector simulation, high energy physics, radiation transport, Intel Xeon Phi, Intel Xeon.**

I. Introduction

High Energy Physics (HEP) simulations rely on detailed Monte Carlo evaluations requiring complex and time-consuming computations. Experiments at the Large Hadron Collider (LHC) devote today a large fraction (50%) of their worldwide distributed computing power (WLCG, the LHC Grid [1]) to simulation tasks. In 10 years from now, with the next High Luminosity LHC phase, the need for simulated data is expected to increase by a factor of 100 [2].

Several initiatives are on-going to modernize Monte Carlo simulation software and make it better suited to leverage the latest hardware advancement. [3], [4] The HEP community is exploring alternative strategies to reduce the amount of data that needs to be simulated or trade some accuracy in order to reduce computation time (*fast simulation*) [5].

This article describes a new fast simulation algorithm based on Deep Learning methods, typically used in the field of image recognition: Generative Adversarial Networks (GANs) [6]. These techniques seem well suited to replace Monte Carlo since it is straightforward to use them to produce realistic samples. From the computing perspective, once the model is trained, the inference step is orders of magnitude faster than traditional Monte Carlo simulation. In this work, we describe how we can use GANs to simulate the response of detectors such as highly segmented calorimeters, whose Monte Carlo-based simulation is particularly time-consuming. This study represents the initial step of a wider plan aimed at providing a generic detector simulation tool based on Deep Learning, fully configurable for different detector use cases. Our final goal is to prove that, by using meta-optimization and hyper-parameters scans, it is possible to tune the network architectures to simulate different detectors.

In this perspective, an efficient training process becomes essential and the accent should therefore be on optimizing the computing resources needed to train the networks, studying parallelization on clusters and cross-platform development. Technological trends seem to be bringing the Deep Learning (and AI) and HPC worlds closer together with impressive development in terms of performance capabilities related to storage, networks and computing architectures. It seems only natural to see an equivalent effort spent on the development of HPC friendly Deep Neural Networks (DNN).

After a brief review on previous related work, we introduce our 3D convolution GANs models and describe how we validate physics results. We then discuss the implementation of a distributed approach together with some preliminary results on scaling. We will then conclude with a brief outlook on our plans for future development.

II. Previous Work

Machine Learning techniques have been applied to different tasks within the HEP community for quite many years already [7], [8]. One example is the search for the Higgs boson employing Artificial Neural Networks [9]. Recently, the advances on Deep Learning and its application to many domains, have generated a renewed interest for possible applications in HEP [10], [11].

GANs are among the most interesting Deep Neural Network recently developed. They implement the idea of adversarial training as introduced by Goodfellow in [6]. In spite of being relatively recent, GANs have inspired research in many fields and several variations and applications exist [12]–[14]. Since, HEP detectors can be regarded as cameras that take pictures of the decay products of particles colliding in modern accelerators, such as the Large Hadron Collider, researchers have been experimenting with GANs for generating images belonging to particularly complicated distributions. LAGAN, for example, or Location Aware GAN used locally connected

layers to generate images corresponding to transverse section of a jet of particles traversing a simplified calorimeter detector [15]. Soon after, CaloGAN extended LAGAN results to a more realistic example, inspired by the ATLAS electromagnetic calorimeter [16].

As Deep Learning models increase in complexity, model size and training time, the role played by distributed training becomes of primary importance. Several different algorithms have been developed in recent years. Generally, these algorithms work by splitting the training load across multiple concurrent processes, either threads on a single machine or jobs spread across separate nodes [17].

Two main approaches exist to distribute the training workload: model parallelism and data parallelism. Model parallelism distributes the training across the model dimension, by having multiple processes train distinct parts of the model, whereas data parallelism employs a distribution of the dataset, each process training the whole model, but on different parts of the data. Out of the two methods, data parallelism is the most popular, being easier to implement. The DL frameworks employ two main methods to maintain the model consistent across all processes, based on where the neural network parameters are stored: parallel (also called replicated) and parameter server. In the parallel case, all nodes maintain a local copy of the weights (a local *replica* of the model), whereas in the parameter server mode, the nodes are assigned separate roles: workers (nodes that run training) and parameter servers (nodes that store the parameters of the model). In both cases, nodes need to exchange data (gradients) to keep the model consistent. With parallel training, nodes need to apply the global (averaged) gradient, an operation that is equivalent to calling a classical reduce operation (e.g. implemented in MPI via `MPI_Allreduce` [18]) to get the average gradient across all nodes. In the parameter server approach, workers send the gradient data to the parameter server nodes that hold the global state of the model, and then read the current state (model parameters) before moving on the next training step. Typically, the size of the data that is sent across the network is considerably large, directly relating to the size of the neural networks in terms of number of parameters (weights and biases).

Existing Deep Learning tools and frameworks implement the communication layer with sockets or MPI. TensorFlow's distributed layer [19] uses by default gRPC [20] for the data transfers between workers and parameter servers, while also providing the alternative of using MPI or Remote Direct Memory Access (RDMA) (but still following the parameter server approach). For the purely parallel mode, MPI is undoubtedly the most popular choice for the communication layer. The different existing libraries implement various communication patterns in order to allow for better scalability: the MaTEx project [21] using `MPI_Allreduce`; a more scalable communication pattern was proposed by Baidu via the ring allreduce [22] where workers are organized as a ring and data is exchanged only between neighbors; based on the ring allreduce pattern, Uber developed their own communication library that uses NVIDIA's Collective Communication Libraries (NCCL and NCCL2), called Horovod [23].

Distributed training of GANs is a new research subject. We believe our work to be one of the first detailed contributions in this direction.

III. Three-dimensional GANs for calorimeter simulation

Calorimeters are important components of HEP detectors. They are intended to measure the energy of particles traversing their dense material. Calorimeters are segmented in a large number of cells. By recording the energy deposited in each cell, it is possible to reconstruct the energy pattern produced by showers of secondary particles inside the detector volume (commonly "energy shower"). These showers can be treated as 3-dimensional images, interpreting each cell as a pixel in an image and the amount of deposited energy per cell as the pixel grey-scale intensity.

A. Calorimeter data

The dataset for the present work was simulated using Geant4 [24]. As described in [25], this dataset is produced in the context of the current design studies of the Linear Collider Detector (LCD) for the Compact Linear Collider (CLIC) [26]. The LCD electromagnetic calorimeter (ECAL) is a sampling calorimeter consisting of 25 layers of tungsten absorbers with silicon sensors between them. It consists of a regular grid of 3D cells of 5.1 mm^3. Individual electrons are shot into the central part of the detector, in a region with regular sensor geometry and orthogonally to the calorimeter surface. Each entry in the dataset represents the energy depositions in individual calorimeter cells produced by one incoming electron (called "events"). These "events" are stored as 25x25x25 cell slices of the electromagnetic calorimeter in 3D arrays (*ECAL arrays*) and they are stored in compressed HDF5 files. The present work trains on $200,000$ electrons with energies ranging from 10 to 510 GeV.

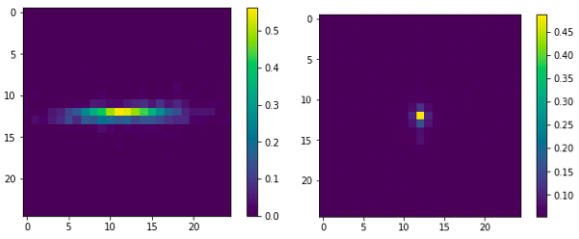

Fig. 1. (right) Longitudinal section of a single electron energy shower. (left) Corresponding transverse section.

B. Networks architecture

Our 3DGAN model is based on three-dimensional convolutions to capture the whole shower development along the three space dimensions. As shown in Figure 2, the generator and discriminator networks contain, respectively, three and four 3-dimensional convolution layers. Details about the networks architectures can be found in [27].

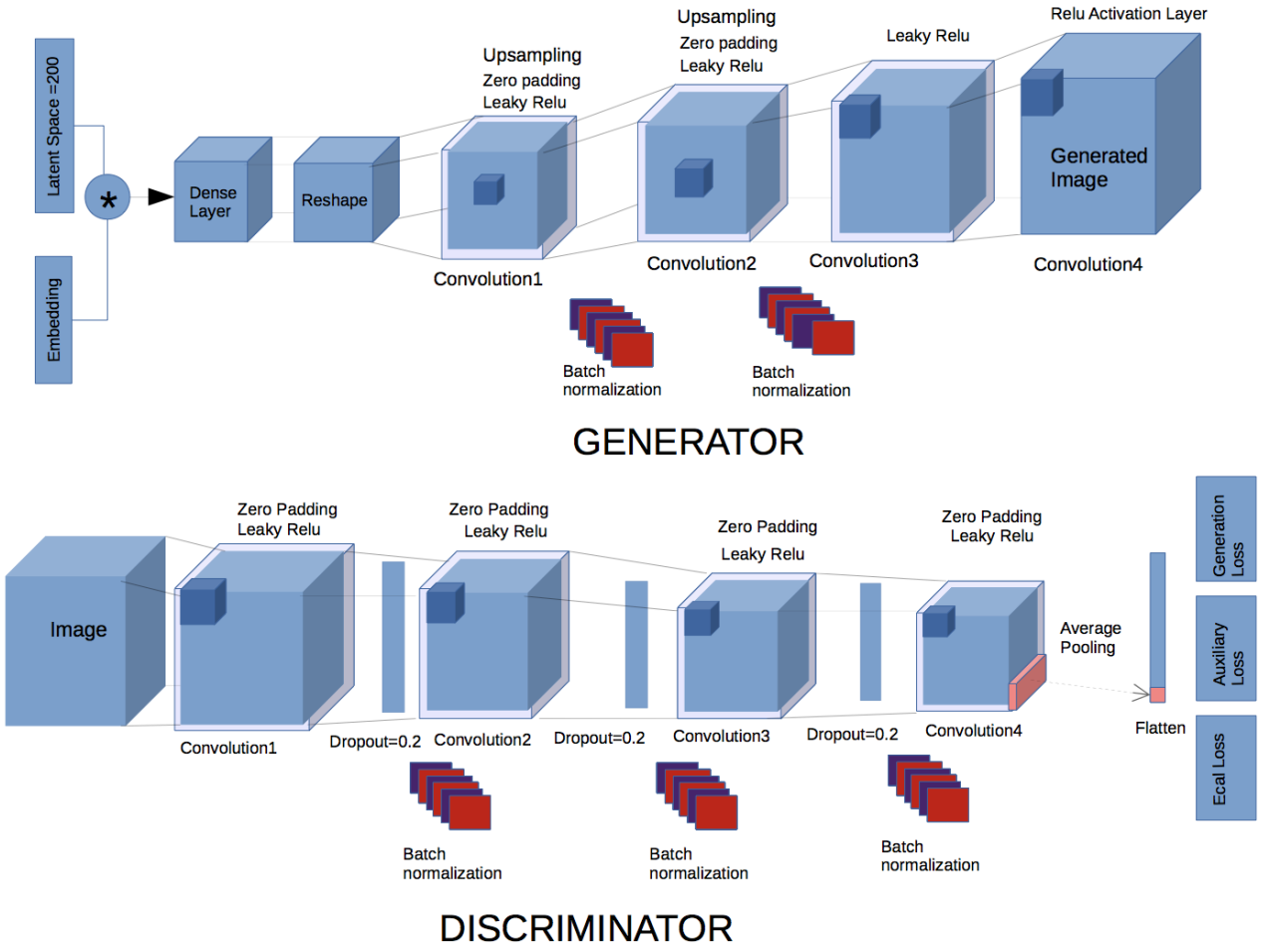

Fig. 2. The 3D GAN Architecture

The generator uses as input a vector of random numbers (the latent vector) generated according to normal distributions, and outputs 25x25x25 images corresponding to the ECAL energy shower.

The discriminator takes as input the generated image. Its structure is very similar to the generator. In addition, a set of three dense layers maps the flattened discriminator network response to three different outputs.

Loosely following the strategy of auxiliary classifier GANs [14], indicating that the introduction of labels provides faster convergence and stability, we assign to the discriminator two additional regression tasks: an estimation of the incoming particle energy (E_p) and of the total energy measured by the calorimeter (E_{cal}), corresponding to the sum of all the energy depositions in the cells of the image.

The adversarial training uses a loss function based on the probability of the discriminator correctly differentiating between real and fake images. In our case we use a binary cross entropy function. The generator in turn uses the inverse of the discriminator loss as the cost to minimize. The absolute mean percentage error is used to calculate the loss terms corresponding to the additional regression tasks performed by the discriminator: the incoming particle energy and total energy deposited in the detector. Weights are employed to balance the contributions of the three individual terms to the total loss.

The RMSProp [28] optimizer is used, with a learning rate of 10^{-3}. The architecture is implemented using Keras [29] with Tensorflow [30] as a backend.

C. Physics validation

We used the proposed 3DGAN to reproduce energy showers deposited in the detector by electrons with energies sampled from a uniform spectrum between 10 and 500 GeV. To fully validate GANs results in terms of physics we compare to standard Monte Carlo simulation based on Geant4. We have

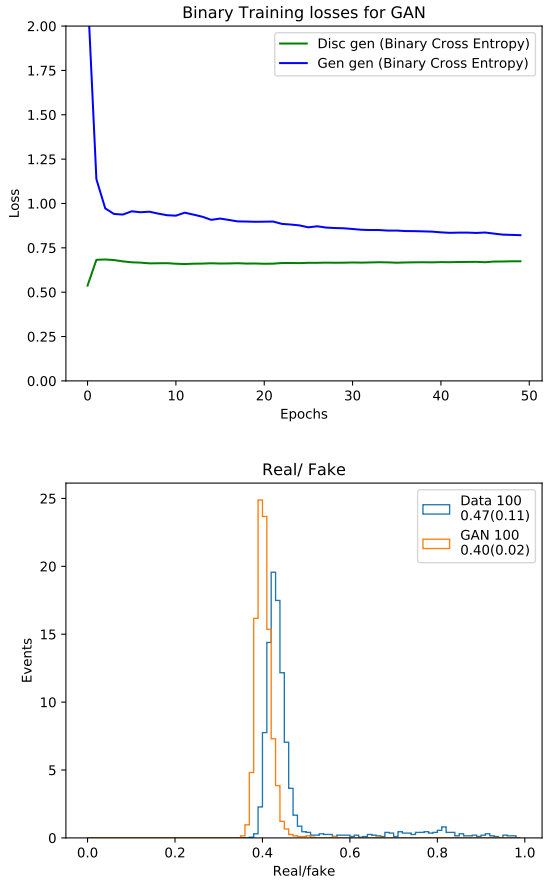

Fig. 3. (top) Binary cross-entropy loss function for the generator and discriminator networks. (bottom) The discriminator real/fake probability for a real image (in blue) and a generated one (orange). The optimal result should peak around 0.5 showing that the discriminator can no longer distinguish the generated images from the original ones.

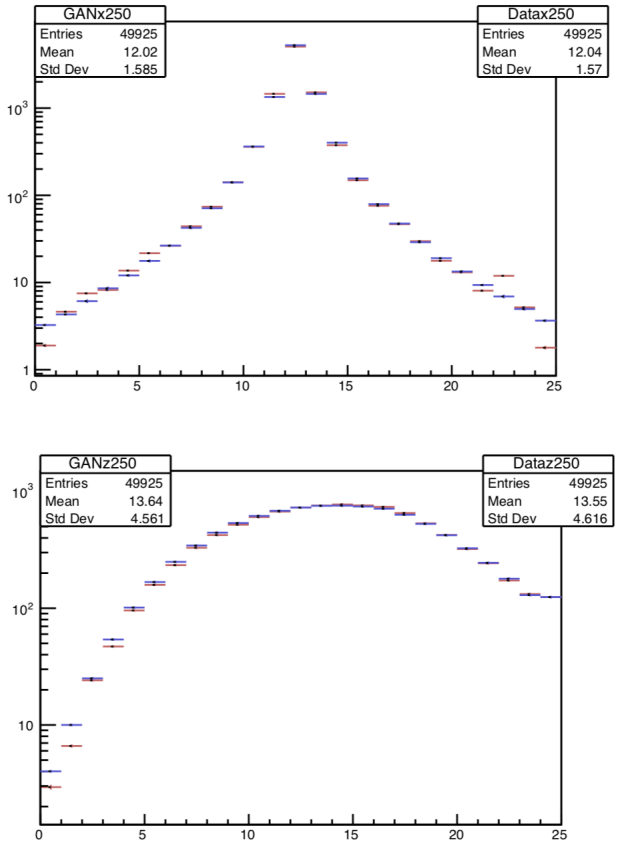

Fig. 4. Transverse energy profile along x (top) and longitudinal energy profile along z (bottom) for 250 GeV electrons. Geant4 prediction is shown in blue and the GANs's in red. Units are GeV.

studied both low level variables describing the calorimeter energy response, i.e. the single cell energy distributions, and more complicated features such as average profiles of the energy showers along the three calorimeter axes. The z axis is chosen along the direction of the incoming electron, while x and y are on the transverse plane, perpendicular to z. These profiles, usually called the energy shower shapes, are relevant quantities for identifying, for example, the type (electron, photon, pion, etc..) of the particle that generated them. Figure 7 shows the shower profile along the x and z axes for 300 GeV electrons: the distributions in x and y are quite similar so only the x profile is presented. The GANs prediction (red) is compared to the corresponding Geant4 simulation (blue). Figure 5 shows the performance of the auxiliary regression task on the primary particle energy assigned to the discriminator. It shows the excellent performance achieved by the discriminator network across the whole energy spectrum and shows the network correctly identifies the main shower images features. Similar agreement can be seen in Figure 6: it represents the calorimeter response in terms of deposited energy (calorimeter sampling fraction). The network performance over this large energy range is remarkably good considered that no effort was made to optimize the network architecture for different energy values. Convolutional layers are sensitive to the spatial shape of the energy showers, which changes according to the primary particle energy, while the architecture parameters were optimized to reproduce at best particles with energies around the central region of the spectrum. In the same way the network is capable of correctly reproduce energy showers produced by different particles, such as pions. Pions are hadronic particles that tend to deposit a smaller amount of energy with respect to electrons in an electromagnetic calorimeter such as the ECAL. Figure 5 represent the network response for pions with energies between 10 and 500 GeV. Once again it is remarkable the agreement with Geant4 simulation although the network and training hyper-parameters were optimized on electrons.

D. Computing performance and training time

Once trained, the GANs simulation tool is a relatively lightweight application: a few MB are enough to describe the layers configuration for the two networks and the inference step is orders of magnitude faster than a standard Monte Carlo. We have run a test on an Intel Xeon 8180 processor (codenamed "Skylake") measuring the time it takes to generate

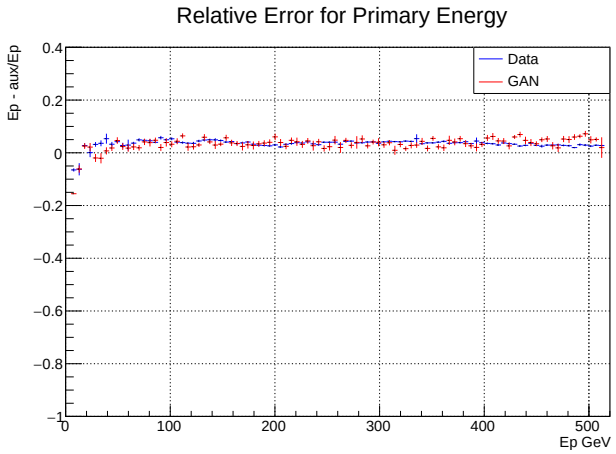

Fig. 5. Longitudinal energy profiles along z for different electron energies (50, 100, 400, 500 GeV). Geant4 prediction is shown in blue and the GANs's in red.

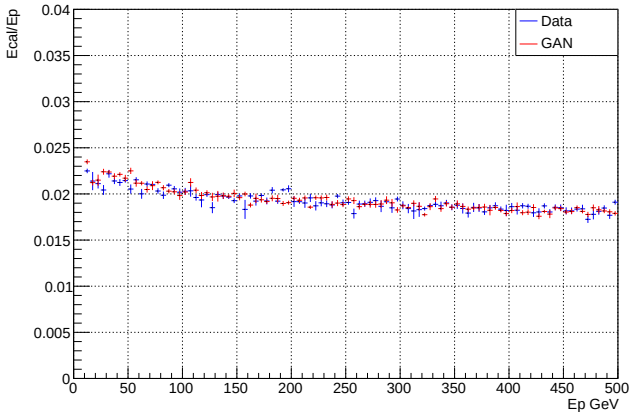

Fig. 6. Calorimeter sampling fraction as a function of the incoming electron energy as predicted by Geant4 simulation (in blue) and GANs (in red)

one electromagnetic shower for our benchmark detector. We obtained 17 s/shower using Geant4 and 4 ms/shower using our trained GANs model, yielding a speedup factor larger than 2500. Using dedicated hardware, such as GPGPUs, the generation time reduces further (we choose not to quote comparison results since the Geant4 application cannot run on GPGPUs).

Unfortunately, training performance is not as good in terms of time. As mentioned above, the core structure of our generator and discriminator models is based on four 3-dimensional convolutional layers, so the models are not very deep, compared to other existing convolutional networks. Even so, our generator network sums up to more than 1M parameters when a relatively small latent space is chosen (latent space dimension=200). Moreover, the adversarial training itself, relying on continuous feedback between the discriminator network and the generator network is particularly time-consuming. In fact, we do train each network twice following a process

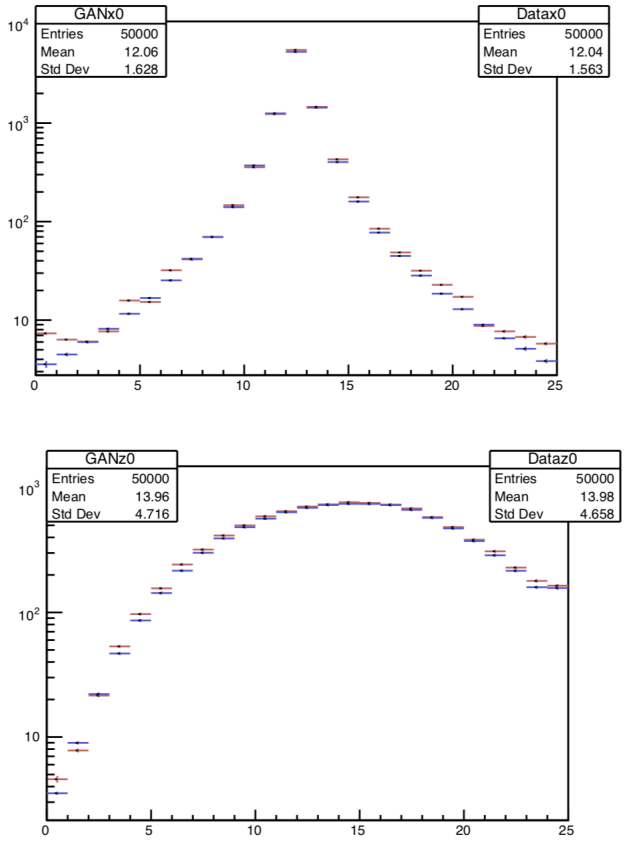

Fig. 7. Transverse energy profile along x (top) and longitudinal energy profile along z (bottom) for pions. Geant4 prediction is shown in blue and the GANs's in red. Units are GeV.

that is sketched in Figure 8. The two-steps discriminator training helps to insure fast convergence of the network. It has become frequent practice while training GANs to build separate batches of real and fake images [31]. This results in a relatively large computing time: training the 3DGAN on $200,000$ electrons for 30 epochs takes about 1 day on a NVidia GTX-1080 card. While this does not represent, per se, a critically long amount of time, it critically impacts any attempt to run large hyper-parameter scans or implement meta-optimization algorithms, which are key to our model generalization effort (optimizing the network architecture for the type of detector that is being simulated). Moreover, the simulation of more realistic detectors and more complicated physics processes could increase the complexity of the networks and the subsequent training time.

IV. Distributed Training

Training of large neural networks on large datasets, of convolution networks, is a computationally intensive task, taking a substantial amount of time, from several hours to days or even weeks. Parallelizing the training workload is thus critical and key to obtaining meaningful results in a timely manner.

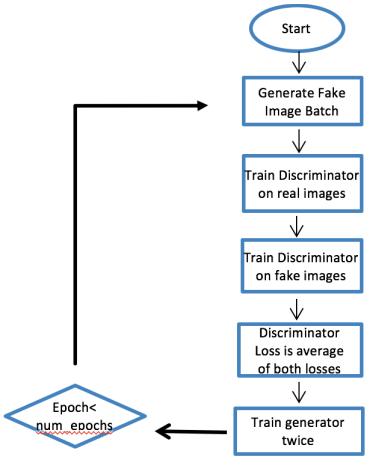

Fig. 8. 3DGAN training process

In this work, we focus on distributing the training of the 3DGAN via data parallelism and *synchronous* and RMSprop optimization. This optimizer divides the learning rate for a weight by a running average of the magnitudes of recent gradients for that weight. Moreover, we use more than one worker per node to maximize the hardware efficiency as described in the sections below.

A. Distributing the training of the 3DGAN with Keras/Tensorflow and Horovod

Training 3D convolutional networks is an even more time-consuming task compared to their traditional 2D counterpart, therefore training these generative models can take weeks. To speed up this research cycle, we leverage the power of modern supercomputers to decrease the time necessary for training these models.

Due to the more controllable nature of *synchronous* stochastic gradient descent and relatively limited straggling effects, we chose a synchronous instead of an asynchronous approach for the 3D GANs optimization. Also, due to the limited communication pressure imposed by our 3D GANs architecture, we chose to use a data parallel approach.

As deep learning framework we have chosen Keras for its simplicity in use and high productivity. As for Keras's backend we have used Tensorflow. Tensorflow was configured and compiled with MKL-DNN support, so that its ops can be well mapped on CPU-based HW. The binaries were built with architecture specific flags (AVX512 FMA) and XLA support. The distribution framework that is used for aggregating Tensorflow tensors on CPUs based systems is Horovod, that internally uses an MPI-based all-reduce primitive to implement the essential allreduce operations performed on gradients in the backward pass. Depending on the cluster topology, MPI_AllReduce collectives are optimized for shared-memory communication within a single node and interconnection fabric across multiple nodes.

Adding Horovod support to a single-device Keras/Tensorflow training program is trivial, the only changes needed are initializing the library, broadcasting the initial weights, and performing all-reduce on the gradients, obtained by wrapping the Tensorflow/Keras optimizer inside of the HorovodDistributedOptimizer class. Another concern is to ensure data loading is performed properly for the data parallel environment. This is also as simple as adapting existing code to consider the Horovod world size. For the optimizer we compared Adam with RMSprop, resulting in quite similar scaling and timings behavior for both.

B. Execution environment

As execution environment we performed our experiments on TACC's Stampede2 system [1], predominantly on the Xeon Skylake partition. This partition is composed of dual-socket Xeon Scalable Processors Platinum 8160 CPUs, each with 24 cores for a total of 48 execution cores per node. Each node also features 192GB of RAM, allowing us to train much larger models than possible on commodity GPUs. This is particularly important for 3D models.

To increase the efficiency, we schedule 2, 4, or 8 MPI processes on each node, with each process utilizing 24, 12, or 6 cores respectively. We also carefully set the $inter\_op\_parallelism\_threads$ and $intra\_op\_parallelism\_threads$ for these hybrid configurations. This is effectively handled by Horovod, leading to intra-node distributed training, the aggregation operation being performed in shared-memory. Results comparing these execution modes are presented in the section below.

We did not notice severe bottlenecks due to communication when scaling to 64 nodes and 256 workers on the current GANs architecture.

Since dwelling into the distributed training revealed performance implications of having a multiple of 16 number of filters, we re-checked the efficiency of the algorithm for a single node with various combinations of underlying libraries. The comparison in timings between Eigen and MKL-DNN and the importance of number of filters can be found in Figure 9. For this test we are using Eigen commit cc79f86b2ace23b92b0760ca798595a0c84fcbad, MKL version 0.14, MKL-DNN commit a29d8487a63afca3d5b8c5bbdbb473cf8ccc6e51, and Tensorflow v1.9.

The results show a performance improvement when using Tensorflow in combination with MKL-DNN and four workers per node compared to just Tensorflow and Eigen. This difference comes from a large number of reorder ops. MKL needs the tensors in an internal data format that is suitable for vectorization, which is different from the TensorFlow format. MKL Reorders happen when a certain op (layer) is not supported in MKL-DNN and so the tensors have to converted between TF and MKL. The fewer the reorder ops, the better

[1] https://www.tacc.utexas.edu/

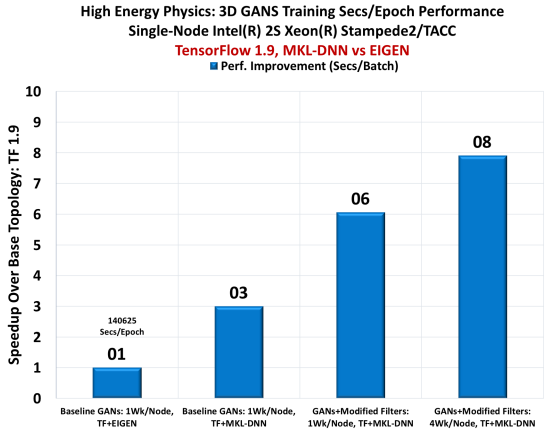

Fig. 9. Comparison of Eigen with MKL-DNN for Tensorflow

the performance. We could also observe that the forward pass has fewer reorders compared to backward pass. When the number of output channels is not a multiple of 16, MKL-DNN uses GEMM-based convolution (from Eigen) which is slower compared to native JIT-based convolution. Both GEMM and native JIT-based convolutions use AVX512 instructions, but GEMM case has additional data transformation overheads.

In order to efficiently use the multi-core CPUs in Tensorflow, we ensured that the $inter\_op\_parallelism\_threads$ and $intra\_op\_parallelism\_threads$ parallelism flags are correctly passed to the underlying Tensorflow session. Tensorflow makes use of two separate thread pools, one that controls the parallelism for operations that can be parallelized internally, such as matrix-matrix multiplications ($intra\_op\_parallelism\_threads$), and one that controls the parallelism for independent nodes in the Tensorflow graph ($inter\_op\_parallelism\_threads$).

C. Scaling results

When performing scale-out experiments, there are two components to keep in focus: the scaling efficiency and the machine efficiency. Empirical evidence lead us to determine that the highest machine efficiency can be obtained if each node hosts four workers. This is the setup used for most of the results. To illustrate this point, we can present the one node - one worker setup compared to the one node - four workers. The time per batch in the first case is 16.5s and in the second is it is 12.8s.

In subsection IV-D we will discuss the impact of large batches on the final accuracy, while in this subsection we are concerned with weak scaling results. This means that the problem size (workload) assigned to each processing element stays constant and additional elements are used to solve a larger total problem. We will present here a weak scaling case with a starting batch size of 8. The efficiency going from one node to 128 nodes is above 94%. In future experiments we will re-implement our solution in Tensorflow directly and check the framework implications on scaling. Figure 10 shows this behavior for our application.

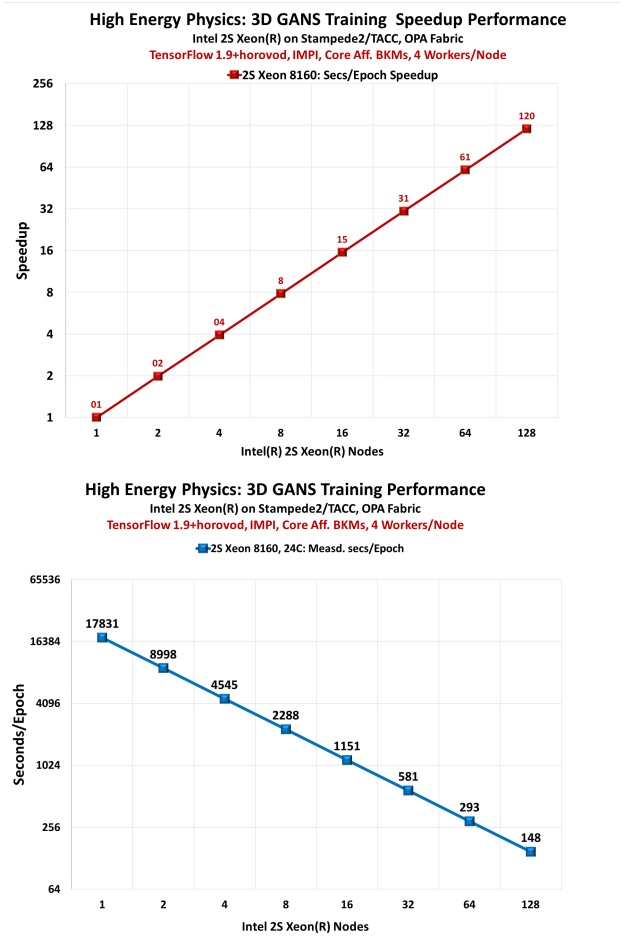

Fig. 10. Training and speedup performance going from 1 node to 128 nodes

D. Validation at scale

At larger scales, the effective batch size (the amount of data that is globally used during one stochastic step) can have a negative impact on accuracy. The degradation of accuracy results while scaling training over a large number of processes is a known issue and several studies are proposing strategies to regularize the SGD multi-process evolution in order to recover most of the single process accuracy [32]–[36].

Therefore, we have studied the consistency of parallel training results in terms of several physics quantities. Figure 11, for example, shows the calorimeter sampling fraction E_{cal}/E_p (E_{cal} being the deposited energy and E_p the energy of the incoming particle) as a function of E_p. It compares the result of scalar training to multi-process training.

While the overall performance stays stable and the different training configurations manage to reproduce very well the energy dependency, as we increase the batch size, the performance clearly degrades at the two edges of the energy spectrum. Those two cases generate energy shower images that can look very different than the images belonging to the center

of the energy spectrum (with the maximum of the energy distribution that shifts along the z axis). This could suggest that the parallel training reduces the capability of the network to generalize to cases that look "different" than the standard one. In fact, as previously mentioned, our network architecture was originally optimized for 100-200 GeV energy showers. This area corresponds exactly to the part of the spectrum were the multi-process curves seem to agree better and results stay consistent.

At the time of this writing, we only applied warmup (techniques established by Goyal et al. 2017) and scaling of initial learning rate (according to Horovod best practices) to stabilize the training procedure. Further studies are currently on-going in order to better understand the origin of this accuracy loss and implement other possible solutions, including SGD with momentum correction and exploration of capsule-based network architectures.

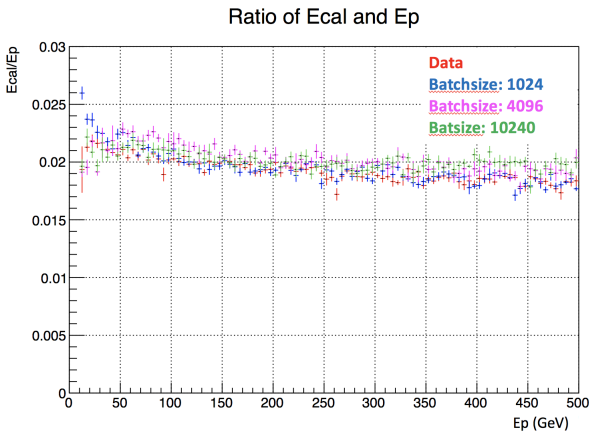

Fig. 11. Calorimeter sampling fraction as a function of the incoming electron energy for batch sizes up to 10240

Figure 11 shows a pronounced degradation above a global batch size of 4096 so we have capped our scaling experiments to this value.

V. Conclusions and Future goals

Monte Carlo production has been so far a major fraction of WLCG computing workload and the High Luminosity LHC experiments needs will increase by orders of magnitude, in the next ten years. In this context, generative models relying on the possibility to treat detector response as images, seem natural candidates to speedup simulation. Generative Adversarial Networks require relatively small amounts of data to train and are the subject of many ongoing studies. Their performance as imaging tools for calorimeter simulation is very promising, and from a computing resources perspective, the gain in the time needed to generate a shower is huge. We have presented an application of 3D convolutional GANs to the simulation of high-granularity detectors, typically foreseen for the next generation accelerators. We have successfully generated images of shower energy deposition in three dimensions, including energy-related information. A detailed validation of the physics results based on the comparison to state-of-the-art Monte Carlo simulation software shows very good results. We plan to work on the generalization of our model to the simulation of different detectors and therefore implementing large hyper-parameter scans to optimize the network architecture to the specific detector to simulate. We have also presented the first results on distributed GANs training.

References

[1] The WorldWide LHC Grid, "http://wlcg.web.cern.ch," .

[2] Ian Bird, "Workshop introduction, context of the workshop: Half-way through run2; preparing for run3, run4," WLCG Workshop, 2016.

[3] G Amadio et al, "Geantv: from cpu to accelerators," *J. Phys.: Conf. Ser. 762 012019*.

[4] Gheata A. et al., "GeantV apha-release preview," in *ACAT 2017 conference proceedings to be published in Journal Of Physics: Conference Series*.

[5] S. Vallecorsa, "Generative models for fast simulation," in *ACAT 2017 conference proceedings to be published in Journal Of Physics: Conference Series*.

[6] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Networks," *ArXiv e-prints*, June 2014.

[7] P. Vannerem, K. . Mueller, B. Schoelkopf, A. Smola, and S. Soldner-Rembold, "Classifying LEP Data with Support Vector Algorithms," *ArXiv High Energy Physics - Experiment e-prints*, May 1999.

[8] R.K. Bock, A. Chilingarian, M. Gaug, F. Hakl, T. Hengstebeck, M. Jiina, J. Klaschka, E. Kotr, P. Savick, S. Towers, A. Vaiciulis, and W. Wittek, "Methods for multidimensional event classification: a case study using images from a cherenkov gamma-ray telescope," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 516, no. 2, pp. 511 – 528, 2004.

[9] Shimon Whiteson and Daniel Whiteson, "Machine learning for event selection in high energy physics," *Engineering Applications of Artificial Intelligence*, vol. 22, no. 8, pp. 1203 – 1217, 2009.

[10] A. Vitek, M. Stachon, P. Krmer, and V. Snel, "Towards the modeling of atomic and molecular clusters energy by support vector regression," in *2013 5th International Conference on Intelligent Networking and Collaborative Systems*, Sept 2013, pp. 121–126.

[11] V V Gligorov and M Williams, "Efficient, reliable and fast high-level triggering using a bonsai boosted decision tree," *Journal of Instrumentation*, vol. 8, no. 02, pp. P02013, 2013.

[12] I. J. Goodfellow, "On distinguishability criteria for estimating generative models," *ArXiv e-prints*, Dec. 2014.

[13] Alec Radford, Luke Metz, and Soumith Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *CoRR*, vol. abs/1511.06434, 2015.

[14] A. Odena, C. Olah, and J. Shlens, "Conditional Image Synthesis With Auxiliary Classifier GANs," *ArXiv e-prints*, Oct. 2016.

[15] Luke de Oliveira, Michela Paganini, and Benjamin Nachman, "Learning particle physics by example: Location-aware generative adversarial networks for physics synthesis," *arXiv preprint arXiv:1701.05927*, 2017.

[16] Michela Paganini, Luke de Oliveira, and Benjamin Nachman, "Calogan: Simulating 3d high energy particle showers in multi-layer electromagnetic calorimeters with generative adversarial networks," *arXiv preprint arXiv:1705.02355*, 2017.

[17] Spiropulu M. Anderson D., Vlimant J., "A MPI-based Python Framework for Distributed Training with Keras," .

[18] open MPI Team, "Message Passing Interface," .

[19] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng, "TensorFlow: A System for Large-scale Machine Learning," in *Proceedings of the 12th USENIX Conference on Operating Systems Design and*

Implementation, Berkeley, CA, USA, 2016, OSDI'16, pp. 265–283, USENIX Association.
[20] Google Inc., "GRPC: A high performance, open-source universal RPC framework," .
[21] "Machine learning toolkit for extreme scale (matex)," https://github.com/matex-org/matex.
[22] "Baidu allreduce," https://github.com/baidu-research/baidu-allreduce.
[23] "Horovod: distributed training framework for tensorflow," https://github.com/uber/horovod.
[24] CERN, *Geant 4*, July 2017 (accessed July 31, 2017).
[25] Federico Carminati, Gulrukh Khattak, Maurizio Pierini, Sofia Vallecorsafa, Amir Farbin, Benjamin Hooberman, Wei Wei, Matt Zhang, Barin Pacela, Vitorial, Maria Spiropulu, and Jean-roch Vlimant, "Calorimetry with Deep Learning : Particle Classification , Energy Regression , and Simulation for High-Energy Physics," in *NIPS*, 2017.
[26] The CLIC collaboration, "Conceptual Design Report," .
[27] Vallecorsa S. et al. Carminati F., Khattak G., "Three dimensional Generative Adversarial Networks for fast simulation," in *ACAT 2017 conference proceedings to be published in Journal Of Physics: Conference Series*.
[28] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky, "Lecture 6a overview of minibatch gradi-ent descent.," 2012.
[29] Francois Chollet et al., "Keras," https://github.com/keras-team/keras, 2015.
[30] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, Software available from tensorflow.org.
[31] Chintala S. et al., "How to Train a GAN? Tips and tricks to make GANs work," .
[32] et al. Keskar, Nitish Shirish, "On large-batch training for deep learning: Generalization gap and sharp minima," .
[33] P. et al. Chaudhari, "Entropy-sgd:Biasing gradient descent into wide valleys.," .
[34] L. et al. Dinh, "Sharp minima can generalize for deep nets," .
[35] G. Huang, "Snapshot ensembles: Train 1, get M for free," .
[36] Alex Krizhevsky, "One weird trick for parallelizing convolutional neural networks," *CoRR*, vol. abs/1404.5997, 2014.