3D convolutional GAN for fast simulation

F. Carminati, G. Khattak, S. Vallecorsa

IXPUG

INTEL EXTREME PERFORMANCE USERS GROUP

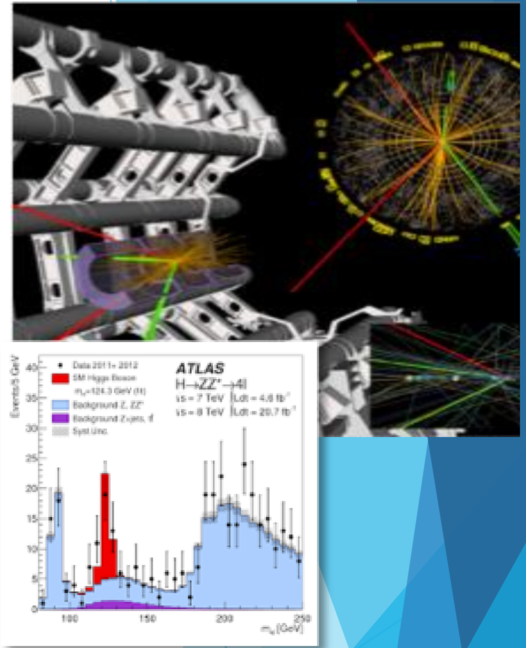
IXPUG Europe Spring 2018

Outline

- Introduction
  - The need for fast simulation
- Status
  - Generative Adversarial Networks for calorimeter simulation
  - Benchmarking on Intel Skylake
  - Testing Intel Nervana Neon
- Plan for 2018
  - Generalisation
  - Optimisation of computing resources
- Summary

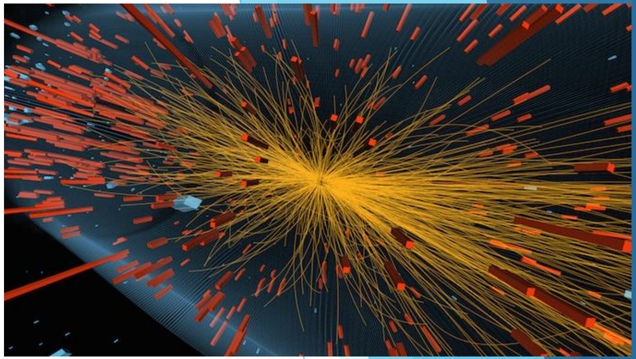
Monte Carlo Simulation: Why

- **Detailed simulation of subatomic particles is essential for data analysis, detector design**
  - Understand how detector design affect measurements and physics
  - Use simulation to correct for inefficiencies, inaccuracies, unknowns.
  - The theory models to compare data against.



A good simulation demonstrates that we understand the detectors and the physics we are studying

The problem



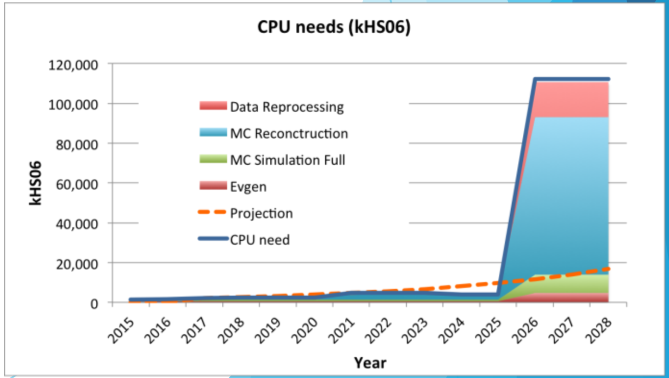
- Complex physics and geometry modeling
- Heavy computation requirements, massively CPU-bound
- Today more than 50% of WLCG power is used for simulations
- By 2025 with the High Luminosity LHC run we will simulate:
    - Much more data!
    - More complex events!
    - Faster!

WLCG
Worldwide LHC Computing Grid

200 Computing centers in 20 countries:
> 600k cores

@CERN (20% WLCG): 65k cores;
30PB disk + >35PB tape storage

ATLAS experiment:
Campana, CHEP 2016

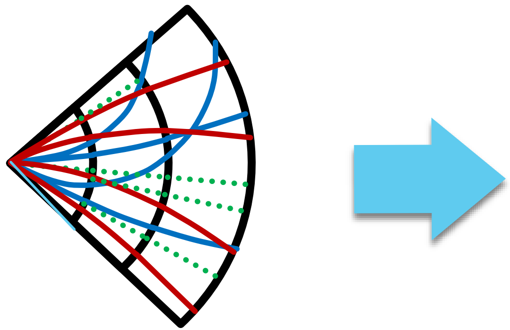


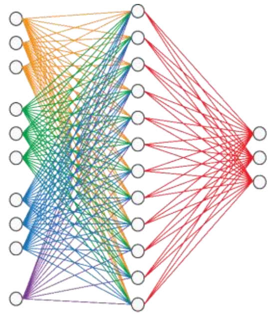
Fast simulation

- Activities on-going to speedup Monte Carlo techniques (new vectorized geometry library VecGeom)
  - Current code cannot cope with HL-LHC expected needs
- Improved, efficient and accurate fast simulation
  - Currently available solutions are detector dependent
- A general fast simulation tool based on Machine Learning/Deep Learning
  - ML techniques are more and more performant in different HEP fields
  - Optimizing training time becomes crucial

Deep Learning for fast simulation

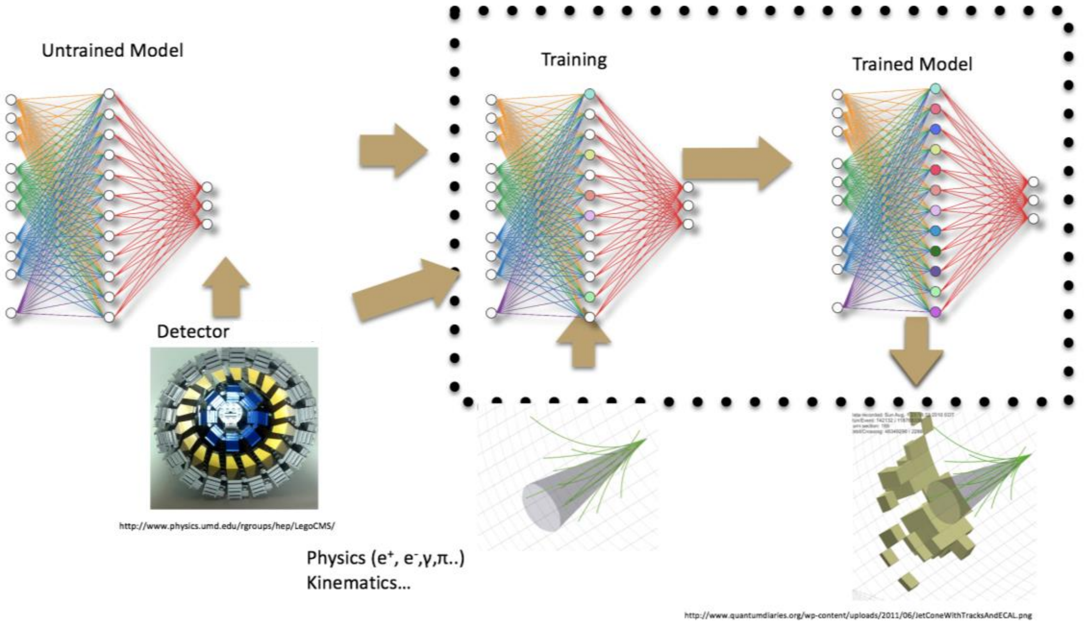
- Generic approach
- Can encapsulate expensive computations
- DNN inference step is faster than algorithmic approach
- Already parallelized and optimized for GPUs/HPCs.
- Industry building highly optimized software, hardware, and cloud services.





A DL engine for fast simulation

- Start with time consuming detectors
  - Reproduce particle showers in calorimeters
- Train on detailed simulation
  - Test training on real data
- Test different models
  - Generative Adversarial Networks, Recurrent Networks
- Embed training-inference cycle in simulation



Requirements

- A fast inference step:
  - It takes ~1 minute to simulate one electromagnetic shower with detailed simulation --> need at least a x100-1000 speedup
- Precise simulation results:
  - Need a detailed validation process
  - Probably cannot go below single precision floating points
- Generic customizable tool
  - Easy-to-use and easily extensible framework
- Large hyper parameters scans and meta-optimisation of the algorithm:
  - Training time under control
  - Scalability
  - Possibility to work across platforms

A plan in two steps

Can image-processing approaches be useful?

- Can we preserve accuracy while increasing speed?
- Can we sustain the increase in detector complexity (future highly-granular calorimeters)?

- A first proof of concept
- Understand performance and validate accuracy

How generic is this approach?

- Can we "adjust" architecture to fit a large class of detectors?

What resources are needed?

- Prove generalisation is possible
- Understand and optimise computing resources

Status

Proof of concept, benchmarking and Validation

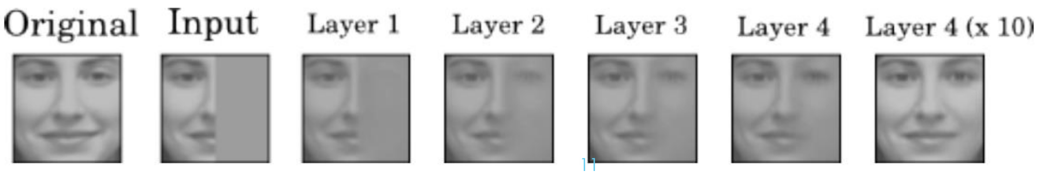
Generative models for simulation

Typically used in computer vision techniques

Many models: Generative Stochastic Networks, Variational Auto-Econders, Generative Adversarial Networks ..

- Realistic generation of samples
- Optimise multiple output for a single input
- Can do interpolation
- Work well with missing data

https://arxiv.org/pdf/1605.05396.pdf



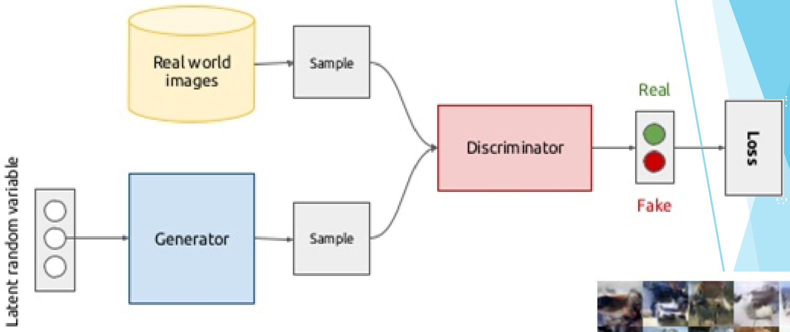
Ranzato, Susskind, Mnih, Hinton, IEEE CVPR 2011

Generative Adversarial Networks

arXiv:1406.2661v1

Simultaneously train two networks that compete and cooperate with each other:

- Generator learns to generate data starting from random noise
- Discriminator learns how to distinguish real data from generated data

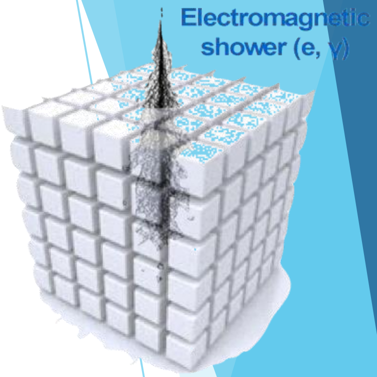


The counterfeiter/police case

- Counterfeiter shows police the fake money
- Police says it is fake and gives feedback
- Counterfeiter makes new money based on feedback
- Iterate until police is fooled

GAN samples for CIFAR-10

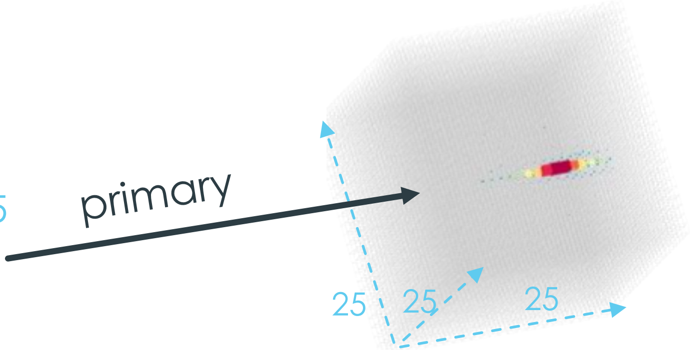
CLIC calorimeter simulation

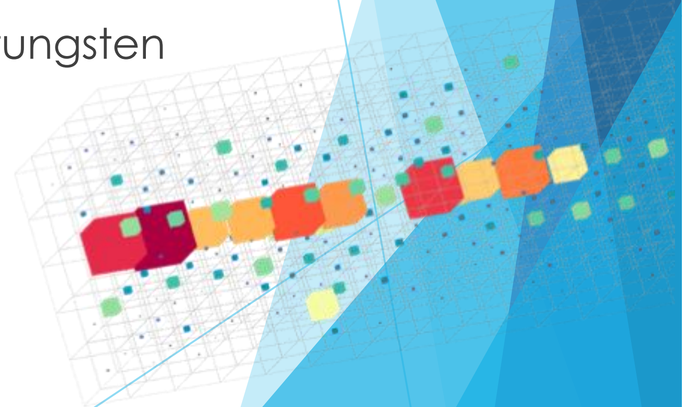


- CLIC is a CERN project for a linear accelerator of electrons and positrons to TeV energies
- Associated electromagnetic calorimeter detector design(*)
- A highly segmented array of absorber material and silicon sensors
  - 1.5 m inner radius, 5 mm×5 mm segmentation: 25 tungsten absorber layers + silicon sensors

Data is essentially a 3D image

Stored as a 25x25x25 HDF5 dataset

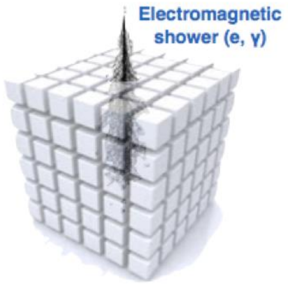


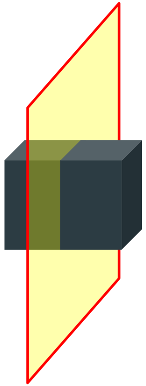


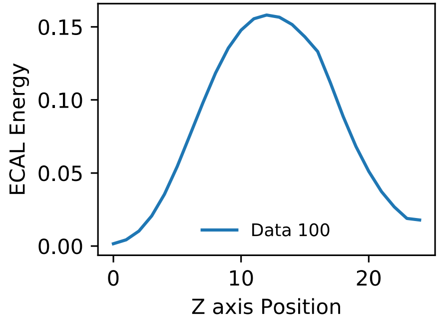
(*) http://cds.cern.ch/record/2254048#

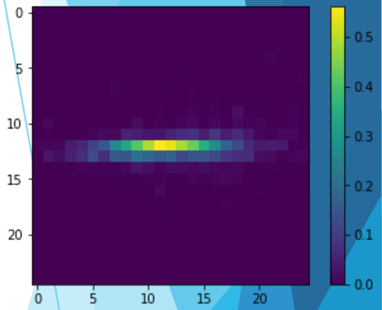
CLIC calorimeter data

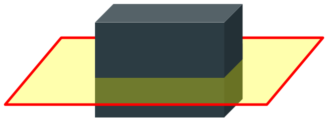
- Highly segmented (pixelized)
  - Segmentation is critical for particle identification and energy calibration.
- Sparse.
- Non-linear location-dependency

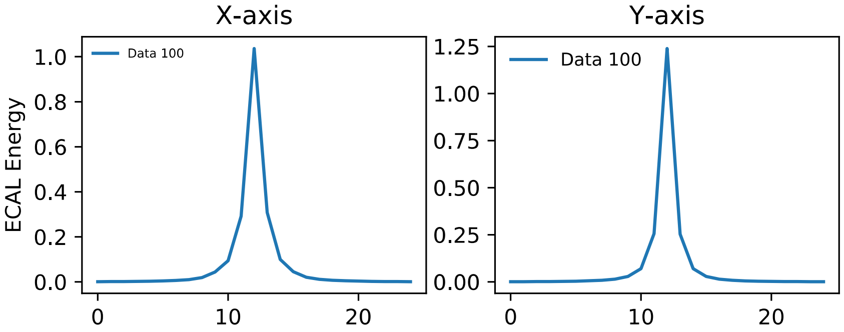


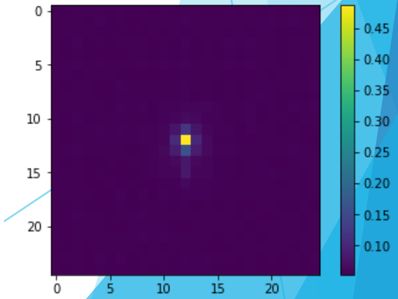






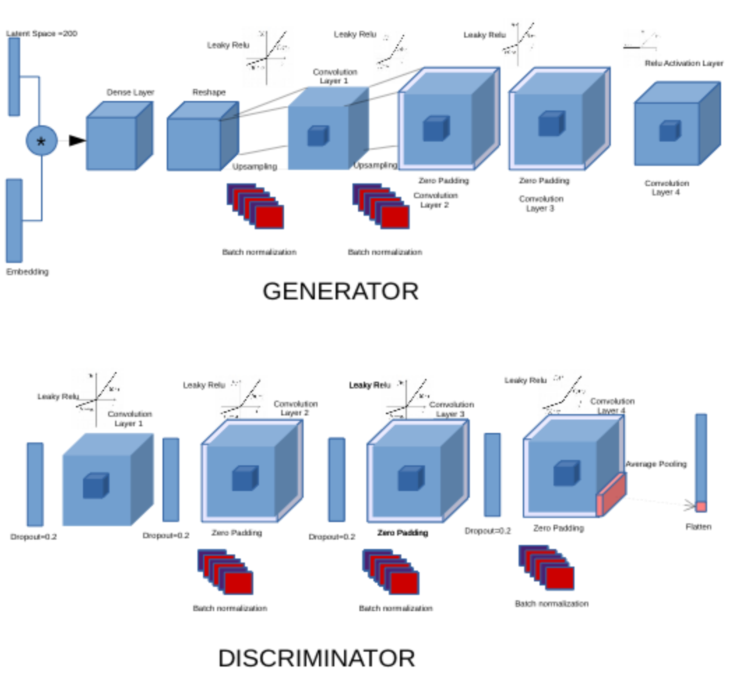






3D GAN

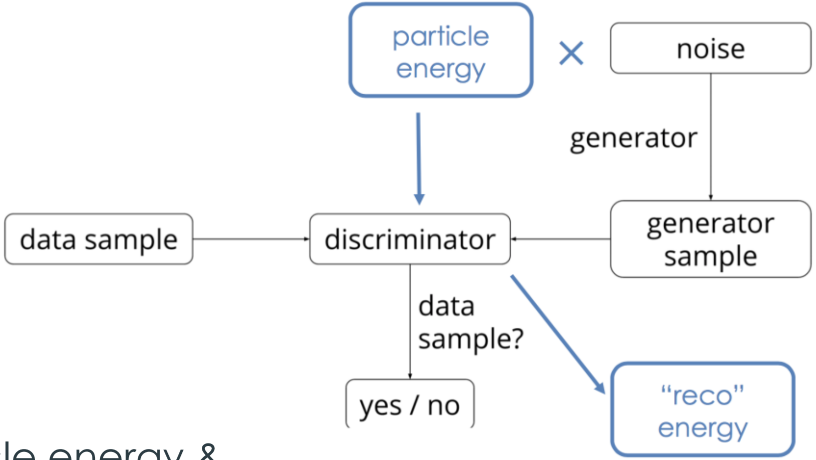
- Similar discriminator and generator models
  - 3d convolutions (keep X,Y symmetry)
- Tested several tips&tricks found in literature*
  - Some helpful (no batch normalisation in the last step, LeakyRelu, no hidden dense layers, no pooling layers)
- Batch training
- Loss is combined cross entropy



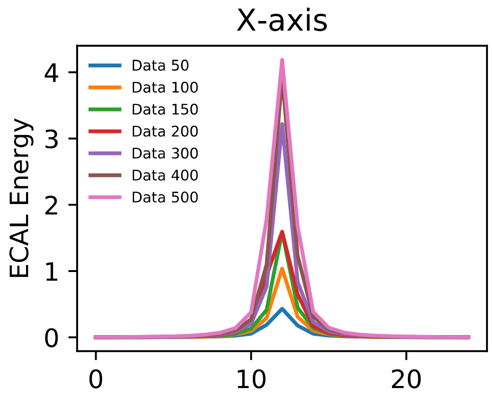
*https://github.com/soumith/ganhacks

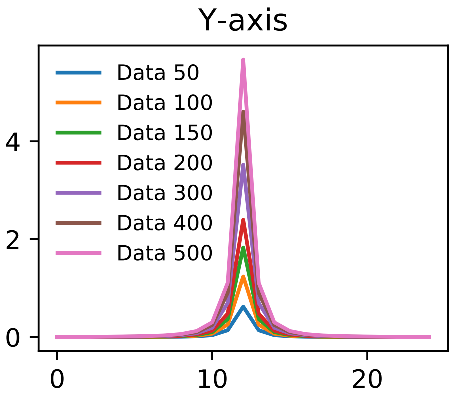
Conditioning on additional variables

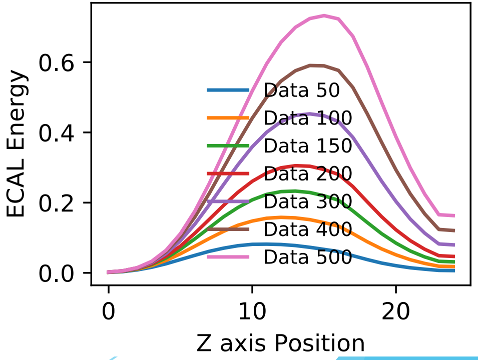
Training generator and discriminator using initial particle energy



- Auxiliary discriminator output
  - Multi-objective optimisation: primary particle energy & reconstructed energy
- Train the generator to reproduce correct shapes



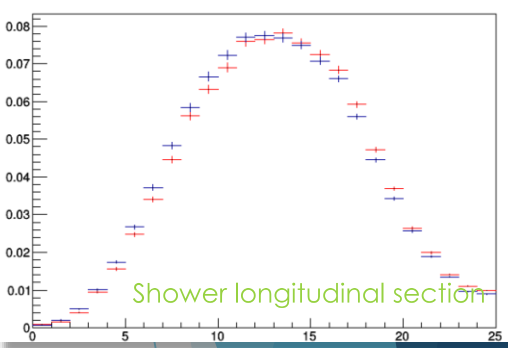


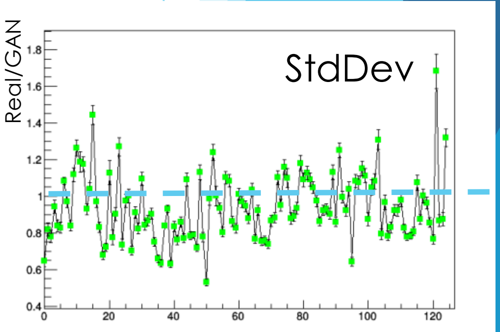


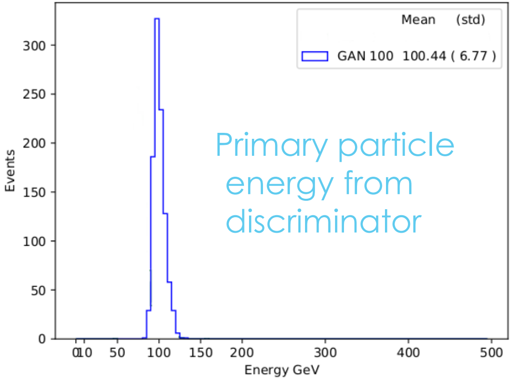
Measuring physics performance

We do not rely on typical image quality assessments

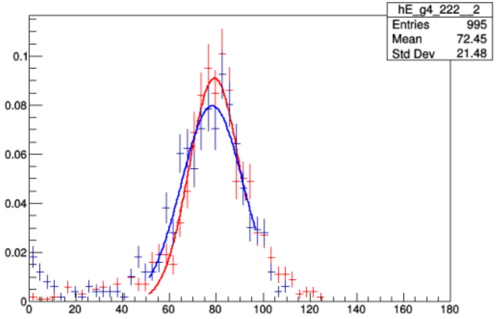
- Comparison to Monte Carlo
- High level quantities (energy shower shapes)
- Detailed calorimeter response (single cell response)
- Particle properties (primary particle energy)







Primary particle energy from discriminator



Physics results are very promising

Need Hyperparameter scans for further optimisation

Computing resources

- All tests run with Intel optimised Tensorflow 1.4.1. + keras 2.1.2
  - Compiled TF sources (-O3 –march=broadwell –config=mkl) (AVX2)*
  - TF linked to MKL-DNN
- Use NCHW data format
- OpenMP setup (for Skylake)
  - KMP_BLOCKTIME = 1
  - KMP_HW_SUBSET=1T
  - OMP_NUM_THREADS=28 (physical cores)
  - KMP_AFFINITY=balanced
- Systems:
  - Intel Xeon Platinum 8180 @2.50 GHz (28 physical cores)
  - NVIDIA GeForce GTX 1080

* Currently AVX512 TF build is broken

Computing resources: inference

- Using a trained model is very fast
  - Orders of magnitude faster than detailed simulation (👍)
  - Next step: test inference on FPGA and integrated accelerators

Time to create an electron shower

Problem	**Machine**	**Time/Shower (msec)**
Full Simulation (geant4)	Intel Xeon Platinum 8180	17000
3d GAN (batch size 128)	**Intel Xeon Platinum 8180**	**7**
3d GAN (batchsize 128)	GeForce GTX 1080	0.04
3d GAN (batchsize 128)	Intel i7 @2.8GHz (MacBookPro)	66

Computing resources: training

Using AVX512 might bring the ratio down to ~15

- Training time (30 epochs, 200k particles)
  - 1d on an NVIDIA GTX-1080
  - ~30 days on Intel Xeon 8180

Time to train for 30 epochs

Problem	Machine	Training time (days)
3d GAN (batchsize 128)	Intel Xeon Platinum 8180 (Intel optimised TF)	30
3d GAN (batchsize 128)	GeForce GTX 1080	1

Benchmarking on Skylake

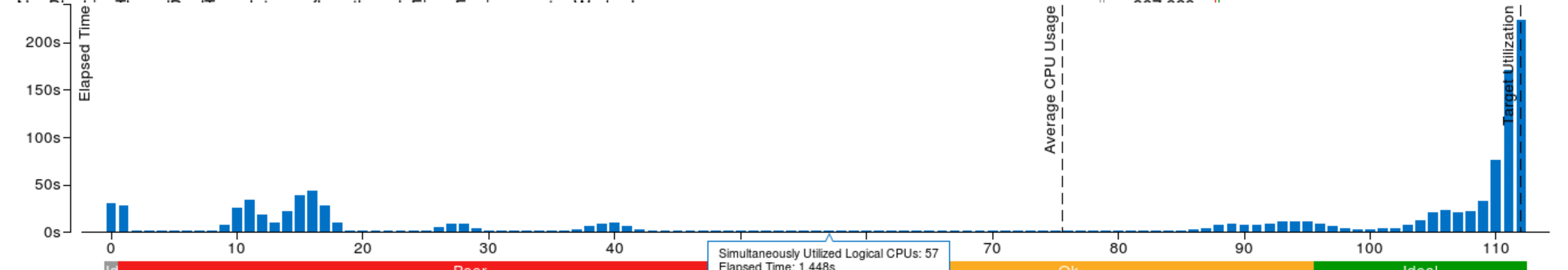
PRELIMINARY

- Major hotspot related to Data Layout optimization: tensor elements copy operation
- Cores are filled

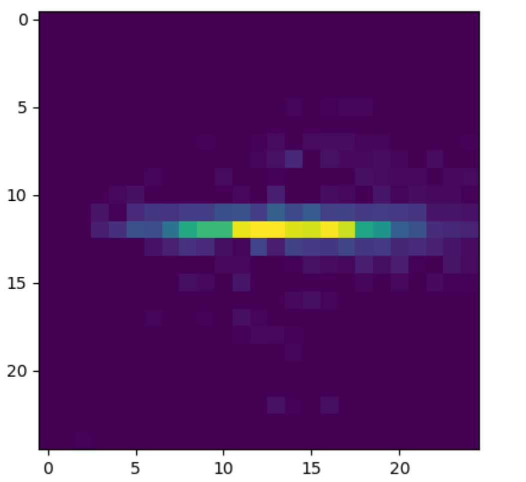
MULTIPLICATION

COPY TENSOR ELEMENTS

Function / Call Stack	Effective Time by Utilization ▼ (Idle, Poor, Ok, Ideal, Over)
Eigen::CustomTensorEvaluator<(long)-1, (long)-1, (long)-1, Eigen::TensorMap<Eigen::Tensor<float const, (int)5, (int)1, long>, (int)16, Eigen::MakeP	60945.735s
Eigen::internal::gebp_kernel<float, float, long, Eigen::internal::blas_data_mapper<float, long, (int)0, (int)0>, (int)8, (int)4, (bool)0, (bool)0>::operator()	11233.760s
Eigen::internal::EvalRange<Eigen::TensorEvaluator<Eigen::TensorAssignOp<Eigen::TensorMap<Eigen::Tensor<float, (int)5, (int)1, long>, (int)16, E	4182.117s
Eigen::internal::gemm_pack_rhs<float, long, Eigen::internal::TensorContractionSubMapper<float, long, (int)0, Eigen::TensorEvaluator<Eigen::Tensor	3550.852s
std::_Function_handler<void (long, long), Eigen::ThreadPoolDevice::parallelFor(long, Eigen::TensorOpCost const&, std::function<long (long)>, std::fu	2245.278s
Eigen::CustomTensorEvaluator<(long)-1, (long)-1, (long)-1, Eigen::TensorMap<Eigen::Tensor<float const, (int)5, (int)1, long>, (int)16, Eigen::MakeP	1004.295s
Eigen::internal::EvalRange<Eigen::TensorEvaluator<Eigen::TensorAssignOp<Eigen::TensorMap<Eigen::Tensor<float, (int)5, (int)1, long>, (int)16, E	980.972s
Eigen::internal::EvalRange<Eigen::TensorEvaluator<Eigen::TensorAssignOp<Eigen::TensorMap<Eigen::Tensor<float, (int)5, (int)1, long>, (int)16, E	632.088s
Eigen::NonBlockingThreadPoolTempl<tensorflow::thread::EigenEnvironment>::Schedule	528.436s
Eigen::internal::EvalRange<Eigen::TensorEvaluator<Eigen::TensorAssignOp<Eigen::TensorMap<Eigen::Tensor<float, (int)2, (int)1, long>, (int)16, E	480.056s
Eigen::NonBlockingThreadPoolTempl<tensorflow::thread::EigenEnvironment>::WaitForWork	408.586s



Benchmarking on Skylake: reducing dimensions



- Simplify network by reducing the number of axis: 2D longitudinal shower shape (typically used to identify particles)
  - Simple 2D convolutions
  - Network parameters reduced by a factor x6

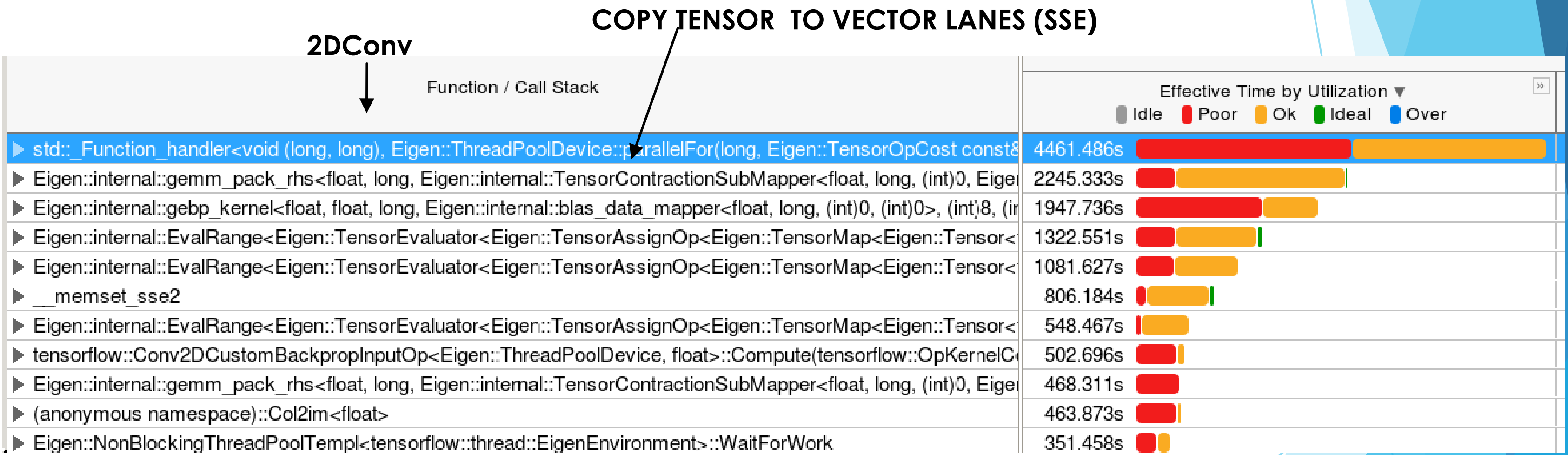
Time to train for 30 epochs

Problem	Machine	Training time (hours)
2d GAN (batchsize 128)	Intel Xeon Platinum 8180 (Intel optimised TF)	14
2d GAN (batchsize 128)	GeForce GTX 1080	1.5

Difference is down to a factor x10 (AVX2) !

Benchmarking on Skylake: 2D profiling

- Call stack is as “expected”: firs hotspot is tensor multiplication



- Problem is related 3D convolutions!
  - Work ongoing with Intel experts to find a solution

Implementation in Neon

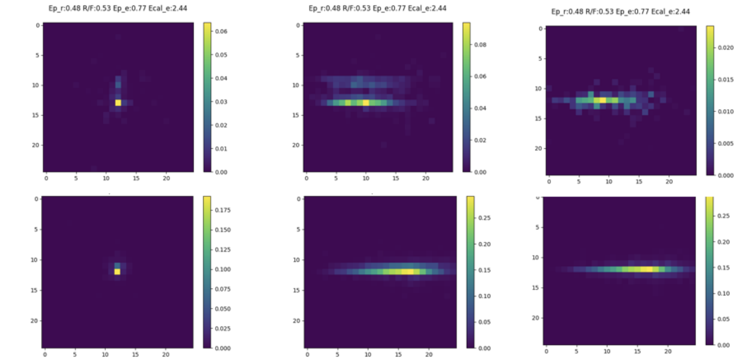
- Intel Nervana's deep learning framework
- Optimised for Intel hardware
- Also available GPU kernel library
- Integration in NervanaCloud and NervanaGraph: upcoming multinode scaling
- Extensive development work on Neon itself needed to implement our 3D GAN architecture
  - Unfortunately performance does not compare to Tensorflow

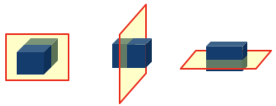
Thanks to Intel support (A. Zanetti)!

A.Zanetti, Intel

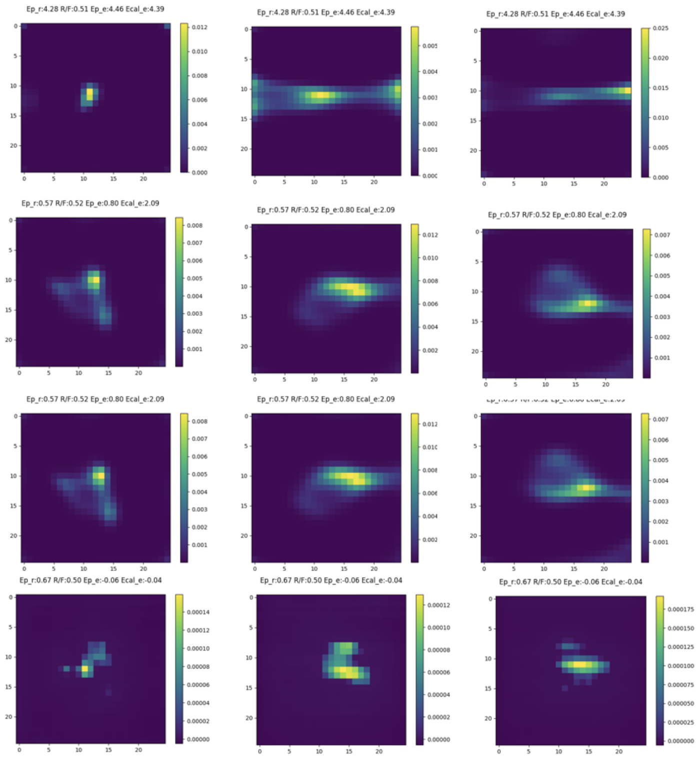
Results so far: Samples of generated vs real images

Samples of Real Images (central slices of the "cube")





Samples of Generated Images (central slices of the "cube")



2018 PLAN

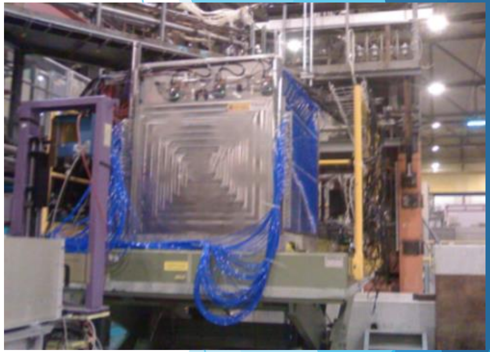
Some work on validation is still ongoing

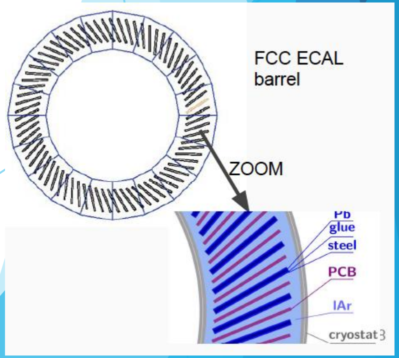
Focus on generalisation and computing resources optimisation

GENERALISATION

- Our baseline is an example of next generation highly granular detector
- Extend to other calorimeters (FCC LAr calorimeter, CALICE SDHCAL)
- Explore optimal network topology according to the problem to solve
    - Hyper-parameters tuning and meta-optimization
        - Sklearn/skopt, Spearmint, …

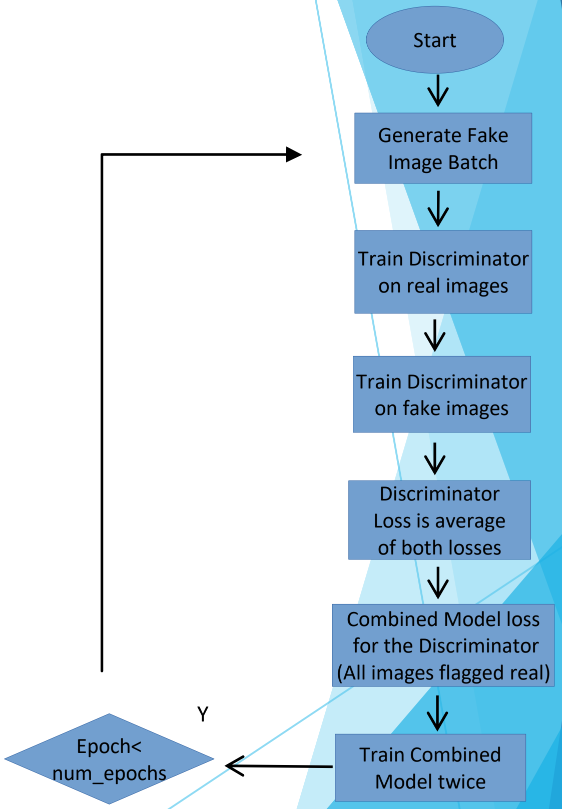
SDHCAL prototype during SPS test beam





Parallel Training

- Test different hardware/environments
  - Intel® Xeon Phi™, DL-100
  - Cloud
  - Try NervanaGraph as soon as available
- Parallelization on distributed systems
  - Implement data parallelism and study scaling on clusters
    - Horovod, mpi-learn, …
- Optimise training data management
  - Test "Big Data" frameworks (e.g. Spark/SparkML, ..)



Summary

- Generative models seem natural candidates for fast simulation
  - Rely on the possibility to interpret “events” as “images”
  - First GANs applications to calorimeter simulations look very promising
  - Many studies ongoing in the different experiments
- 3d GAN is the initial step of a wider plan to investigate simulation with DL

- Eager to see good performances on CPUs
- Need to solve the 3D convolutions issues in TF and/or MKL/MKL-DNN

Spinoffs?

- Direct
  - Radiation treatment planning
  - Medical instrument design / optimization
  - Radiation safety
- Indirect
  - Complex / multidimensional DL applications for other sciences
  - Combination DL / Big Data
  - Combination DL / HPC (c.f. ACAT 2019)

Thanks !

Questions?

Some references

- GANs:
  - Just google “Generative Adversarial Networks”!
  - I. Goodfellow recent seminar: https://indico.cern.ch/event/673989/
  - A. Radford, L. Metz and S. Chintala, Unsupervised representation learning with deep convolutional generative adversarial networks. 2015.
  - Mirza, Mehdi and Osindero, Simon. Conditional generative adversarial nets. 2014.
  - Augustus Odena, Christopher Olah, Jonathon Shlens, Conditional Image Synthesis with Auxiliary Classifier GANs. ICML, 2017.
  - Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, Pieter Abbeel. InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets. 2016.
  - Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen. Improved Techniques for Training GANs. NIPS, 2016.
- Advanced GANs:
  - https://indico.cern.ch/event/655447/contributions/2742180/attachments/1552018/2438676/advanced_gans_iml.pdf (see refs on page 16)
- Physics and ML:
  - DS@HEP : (2017 workshop) https://indico.fnal.gov/event/13497/timetable/#20170508
  - Connecting the dots:
  - https://indico.hephy.oeaw.ac.at/event/86/timetable/#20160222 (2016 workshop)
  - IML workshops: https://indico.cern.ch/event/595059/ and https://indico.cern.ch/event/655447/