Efficient simulations of ML and LQCD

Denis Boyda, ALCF

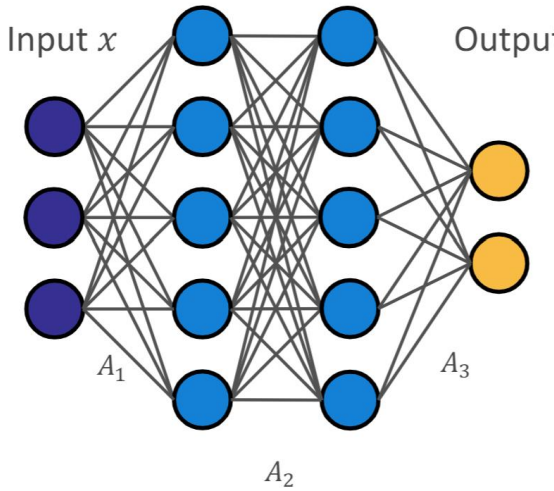
Efficient Simulations on GPU hardware, October 24-27, 2022

Outline

- One slide intro to ML
- Strategy for ML for LQCD software development
- Scaling the number of ML experiments / MLOPs
- Weak Scaling / Increasing Batch Size
- Towards larger memory limits:
  - ❑ Checkpointing
  - ❑ Model (pipeline/tensor) parallelism or strong scaling
- Efficient Simulations on one GPU
  - ❑ Profiling PyTorch
  - ❑ Move functions to GPU / Autograd function
  - ❑ JIT compiler
  - ❑Custom kernels

One slide intro to ML

CRASH COURSE: DEEP LEARNING



"neural network"

$$y = \sigma(A_3\sigma(A_2\sigma(A_1x + b_1) + b_2) + b_3)$$

Forward path
- a layer get output from previous layer,
- does matmul with weights,
- applies activation function,
- sends data to the next layer.
- Computations repeated for all layers

Backward path
- Gradients are computed using chain rule
- Weights are updated using gradients

Training is done in batches.
Stochastic gradient descent is used

Slide credit: Bethany Lusch, ALCF

Strategy for ML for LQCD software development

Strategy for ML for LQCD software development

Measurements

observable regression

B. Yoon, T. Bhattacharya, R. Gupta, Phys. Rev. D 100, 014504 (2019)

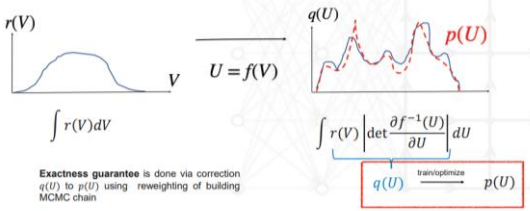
- Observables calculation takes similar or larger than generation resources
- Use ML regression to compute them faster
  - N configurations, $N_{train} + N_{bs}$ measurements of O, N_{train} measurements are used for training
  - effective samples size is increased from $N_{train} + N_{bs}$ to N
- Correct bias

$$\bar{O} = \frac{1}{N - N_{train} - N_{bc}} \sum O_i^{ML} + \frac{1}{N_{bc}} \sum (O_j - O_j^{ML})$$

Can we decrease time for observable measurements and get higher statistics?

Configuration generation with normalizing flows

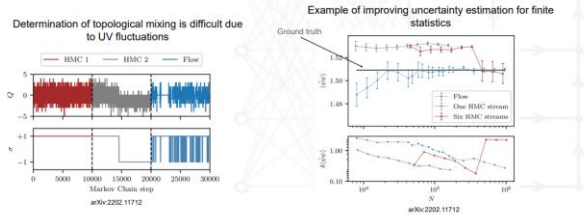
Flow-based models learn a change-of-variables that transforms a known distribution to the desired one



Exactness guarantee is done via correction $q(U)$ to $p(U)$ using reweighting of building MCMC chain

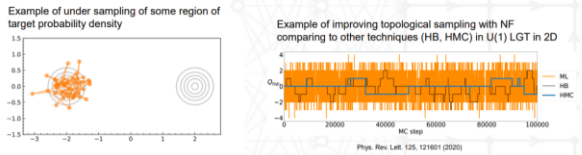
Normalizing flows: better uncertainty qualification at finite statistics

Lattice Schwinger model near criticality



Normalizing flows: better mixing

U(1) lattice gauge theory in 2D



Increase signal to noise ratio via contour deformation

Variable transformation does not change integral

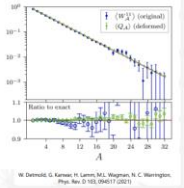
$$\langle O \rangle = \int D\,\bar{U} e^{-S(\bar{U})} O(\bar{U}) = \int DU\, J(U) e^{-S(\bar{U}(U))} O(\bar{U}(U))$$

$$\langle O \rangle = \langle Q \rangle = \left\langle J(U) e^{-S(\bar{U}(U)) + S(U)} \right\rangle$$

but changes uncertainties

transformation $\bar{U} = f(U)$ is optimized such that $var(Q) \ll var(O)$

Demonstration in SU(3) gauge theory in 2D



Can we apply it for viscosity computations in full QCD?

Compute QCD phase diagram in (T, mu) with normalizing flows

Direct MCMC simulations of QCD at nonzero chemical potential is not tractable due to Sign Problem

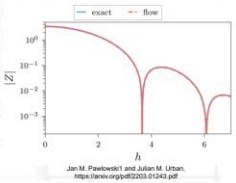
Several approaches use MCMC simulations at zero and/or imaginary chemical potential

Simulations at several values of imaginary chemical potential required in order to do extrapolation to real region

After training Normalizing flow model gives access to "all" values of imaginary chemical potential

Demonstration of flow-based Density of State

$$S(\phi) = \frac{m^2}{2}(\phi_1^2 + \phi_2^2) + \lambda(\phi_1^2 + \phi_2^2)^2 + ih\phi$$



Can we get larger chemical potential?

Thermodynamical properties of QGP and EoS with normalizing flows

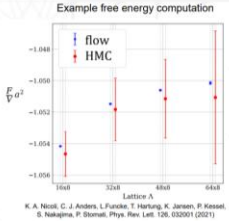
Lattice scalar field theory

The fundamental difficulty is that MCMC is not able to directly estimate the partition function of the lattice field theory.

Normalizing flows have direct access to partition function

$$Z = \int D\phi q_\theta(\phi) \frac{e^{-S(\phi)}}{q_\theta(\phi)} = \left\langle \frac{e^{-S(\phi)}}{q_\theta(\phi)} \right\rangle_{q_\theta(\phi)}$$

Example free energy computation



Can we compute QCD EoS with higher precision?

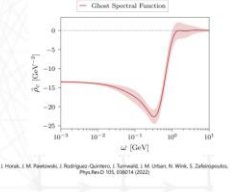
Reconstructing QCD Spectral Functions with Gaussian Processes

Spectral functions are extracted from lattice QCD correlator using inverse integral transformation which is ill-defined problem

Reconstruction using Gaussian Process Regression

what is most probably value and uncertainty of $\rho(\omega)$ given some observations $G(t_i)$ with uncertainties

Demonstration of extraction Ghost spectral function in 2 + 1 LQCD; band shows uncertainty.



Did we improve a solution of inverse problem?

Slides credit: Denis Boyda, Machine Learning techniques in lattice QCD, 2022 RHIC/AGS Annual Users' Meeting, June 7, 2022

Strategy for ML for LQCD software development

Machine Learning in Nuclear Physics

Amber Boehnlein, Markus Diefenthaler, Nobuo Sato, Malachi Schram, and Veronique Ziegler

Thomas Jefferson National Accelerator Facility,
12000 Jefferson Avenue,
Newport News, Virginia,
USA

Cristiano Fanelli

Laboratory for Nuclear Science,
Massachusetts Institute of Technology,
Cambridge, MA 02139,
USA
The NSF AI Institute for Artificial Intelligence and Fundamental Interactions

Morten Hjorth-Jensen

Facility for Rare Isotope Beams and Department of Physics and Astronomy,
Michigan State University, MI 48824,
USA
Department of Physics and Center for Computing in Science Education,
University of Oslo, N-0316 Oslo,
Norway

Tanja Horn

Department of Physics,
The Catholic University of America,
N. E. Washington, DC 20064,
USA
Thomas Jefferson National Accelerator Facility,
12000 Jefferson Avenue,
Newport News, Virginia,
USA

Michelle P. Kuchera

Department of Physics and Department of Mathematics & Computer Science,
Davidson College,
North Carolina 28035,
USA

Dean Lee, Witold Nazarewicz, and Peter Ostroumov

Facility for Rare Isotope Beams and Department of Physics and Astronomy,
Michigan State University, MI 48824,
USA

Kostas Orginos

Department of Physics,
William & Mary,
Williamsburg, Virginia,
USA
Thomas Jefferson National Accelerator Facility,
12000 Jefferson Avenue,
Newport News, Virginia,
USA

Alan Poon and Xin-Nian Wang

Nuclear Science Division,
Lawrence Berkeley National Laboratory,
1 Cyclotron Road,
Berkeley, California 94720,
USA

Alexander Scheinker

Accelerator Operations and Technology Division Applied Electrodynamics Group,
Los Alamos National Laboratory,
Los Alamos, New Mexico 87544,
USA

Michael S. Smith

Physics Division,
Oak Ridge National Laboratory,
Oak Ridge, Tennessee, 37831-6354,
USA

Long-Gang Pang

Key Laboratory of Quark and Lepton Physics,
Institute of Particle Physics,
Central China Normal University, Wuhan 430079,
China

Advances in machine learning methods provide tools that have broad applicability in scientific research. These techniques are being applied across the diversity of nuclear physics research topics, leading to advances that will facilitate scientific discoveries and societal applications. This Colloquium provides a snapshot of nuclear physics research which has been transformed by machine learning techniques.

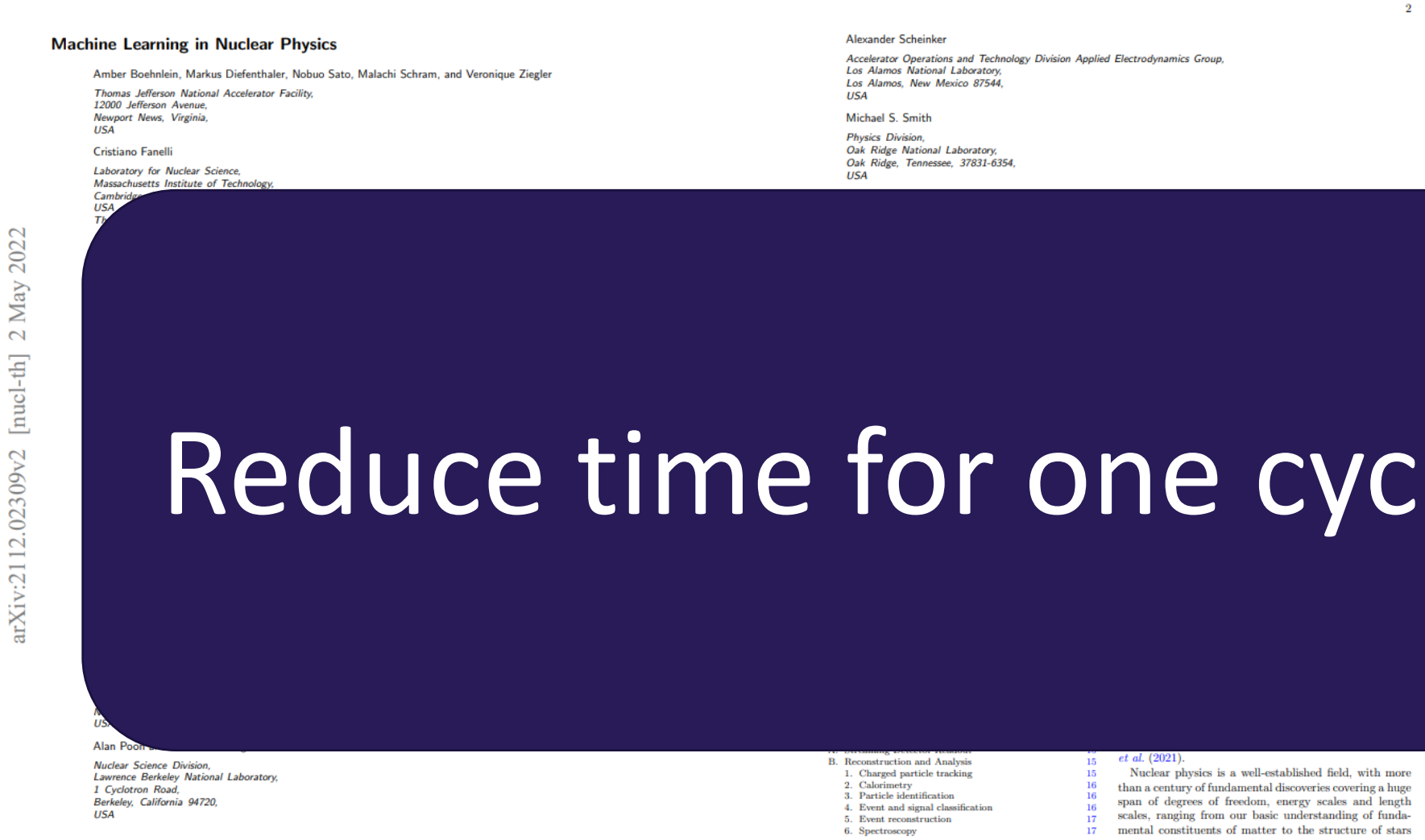
CONTENTS

I. INTRODUCTION

This Colloquium represents an up-to-date summary of work in the application of machine learning (ML) in nuclear science, covering topics in nuclear theory, experimental methods, accelerator technology, and nuclear data. An overview of the use of artificial intelligence (AI) techniques in nuclear physics which aimed at identifying commonalities and needs has been provided in Bedaque *et al.* (2021).

Nuclear physics is a well-established field, with more than a century of fundamental discoveries covering a huge span of degrees of freedom, energy scales and length scales, ranging from our basic understanding of fundamental constituents of matter to the structure of stars

Strategy for ML for LQCD software development



Machine Learning in Nuclear Physics

Amber Boehnlein, Markus Diefenthaler, Nobuo Sato, Malachi Schram, and Veronique Ziegler

Thomas Jefferson National Accelerator Facility,
12000 Jefferson Avenue,
Newport News, Virginia,
USA

Cristiano Fanelli

Laboratory for Nuclear Science,
Massachusetts Institute of Technology,
Cambridge
USA

Alan Poon

Nuclear Science Division,
Lawrence Berkeley National Laboratory,
1 Cyclotron Road,
Berkeley, California 94720,
USA

Alexander Scheinker

Accelerator Operations and Technology Division Applied Electrodynamics Group,
Los Alamos National Laboratory,
Los Alamos, New Mexico 87544,
USA

Michael S. Smith

Physics Division,
Oak Ridge National Laboratory,
Oak Ridge, Tennessee, 37831-6354,
USA

et al. (2021).

Nuclear physics is a well-established field, with more than a century of fundamental discoveries covering a huge span of degrees of freedom, energy scales and length scales, ranging from our basic understanding of fundamental constituents of matter to the structure of stars

Reduce time for one cycle

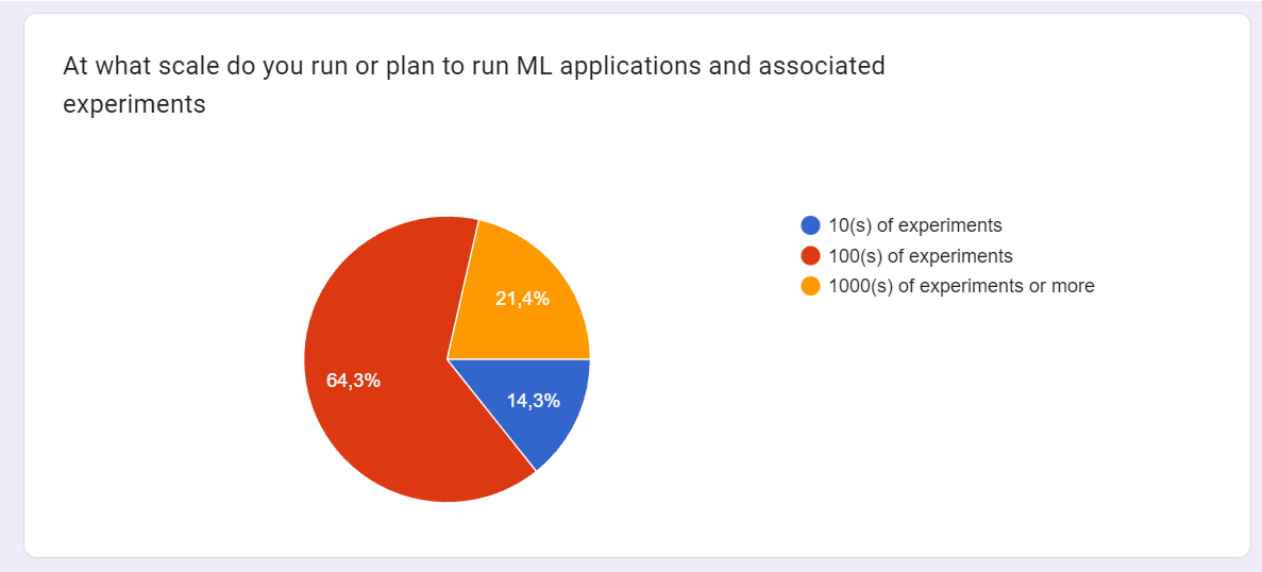
Scaling the number of ML experiments / MLOPs

Scaling the number of ML experiments / MLOPs

Number of ML experiments in our data bases. Development of flow-base models for

u1 (ml_generation_u1)	phi4 (ml_generation_phi4)	su2 (ml_generation_su2)	su3 (ml_generation_su3)	ferm (ml_generation_ferm)	QCD (ml_generation_qcd)
Id	Id	Id	Id	Id	Id
52968	1386	1844	1527	1598	244

Recent review of users of Argonne Leadership Computer Facility



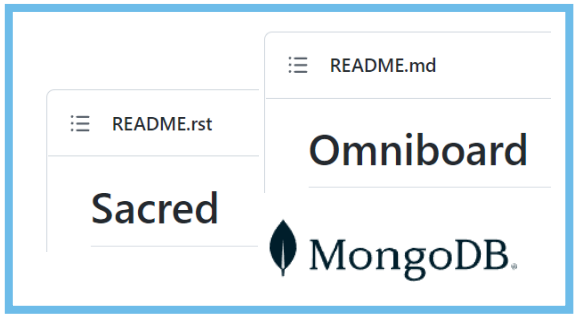
Scaling the number of ML experiments / MLOPs - 2

MLOps (Machine Learning Model Operationalization Management) provides

- Hyperparameters/configuration tracking
- Live information (stdout, stderr, results)
- Artifacts (models, datasets) control and versioning
- Code control and versioning
- Environment configuration
- Fail trace

and an efficient way of analyzing experiments though

- Dashboard
- API to database



Scaling the number of ML experiments / MLOPs - 3

Omniboard | u1 (ml_generation_u1) | su2 (ml_generation_su2) | su3 (ml_generation_su3) | phi4 (ml_generation_phi4) | ferm (ml_generation_ferm) | QCD (ml_generation_qcd)

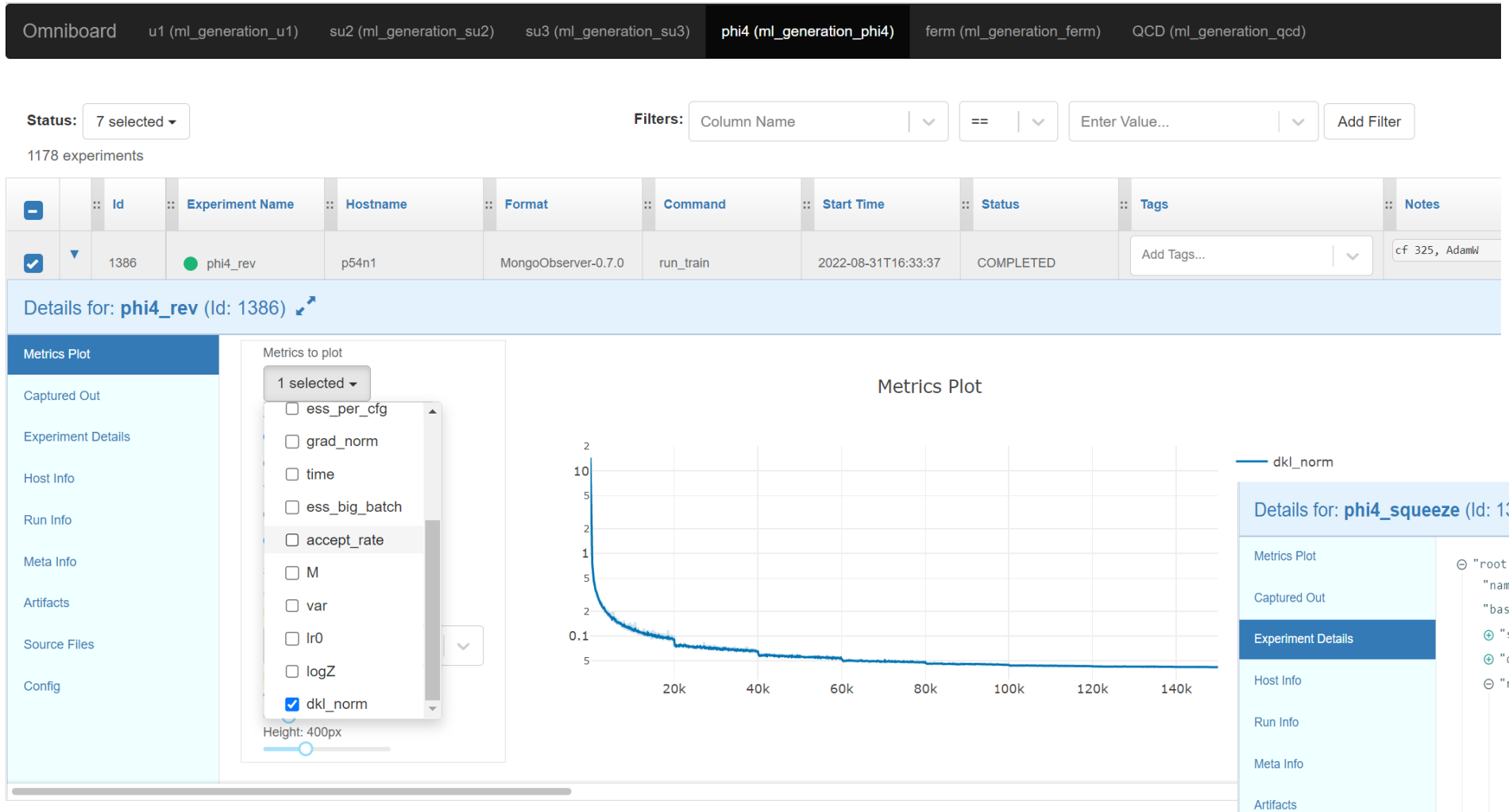
Status: 7 selected

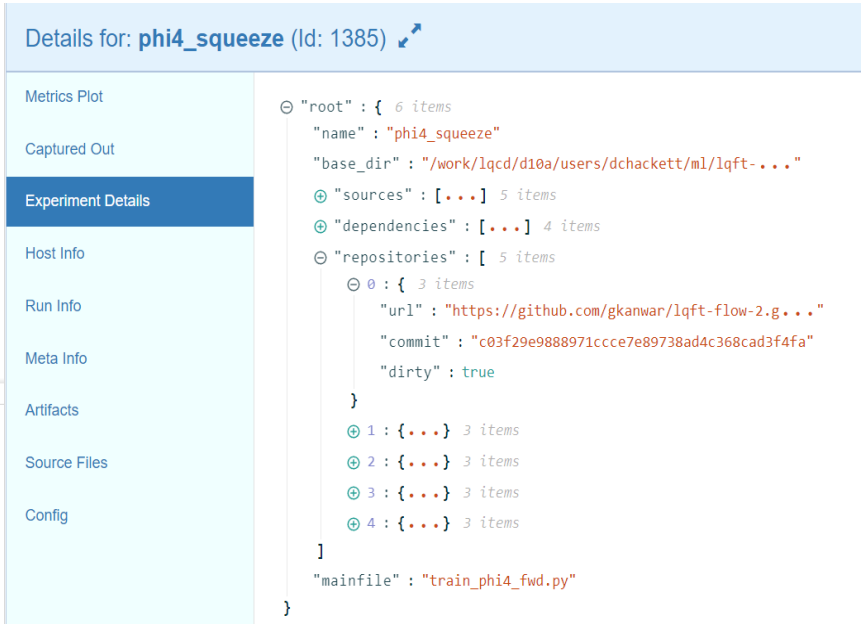
Filters: Column Name | == | Enter Value... | Add Filter

1178 experiments

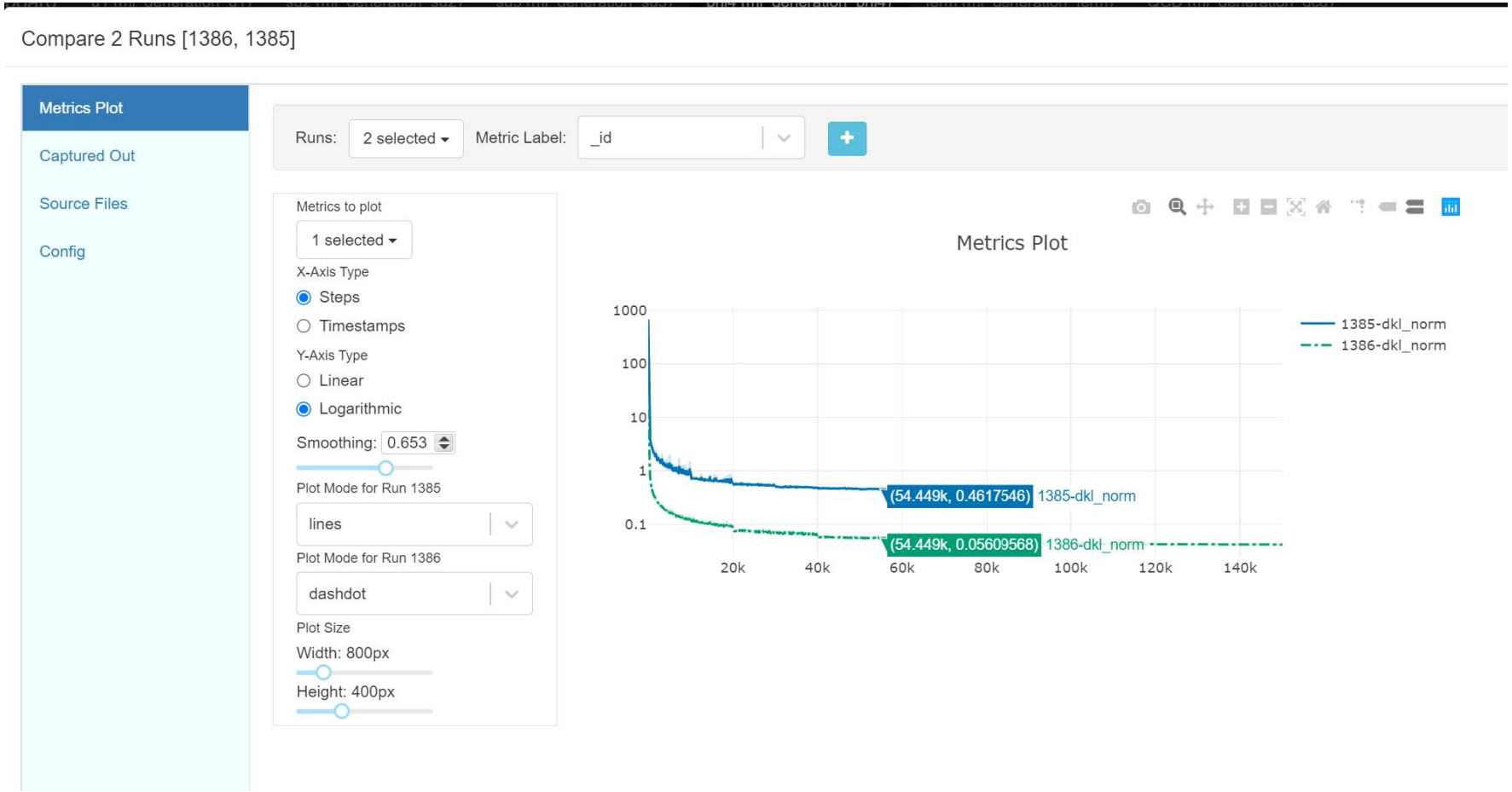
Id	Experiment Name	Hostname	Format	Command	Start Time	Status	Tags	Notes
1386	phi4_rev	p54n1	MongoObserver-0.7.0	run_train	2022-08-31T16:33:37	COMPLETED	Add Tags...	cf 325, AdamW
1385	phi4_squeeze	p56n1	MongoObserver-0.7.0	run_train	2022-08-30T05:05:11	PROBABLY_DEAD	Add Tags...	Enter Notes
1384	phi4_squeeze	p56n4	MongoObserver-0.7.0	run_train	2022-08-30T02:29:37	PROBABLY_DEAD	Add Tags...	Enter Notes
1383	phi4_squeeze	p56n2	MongoObserver-0.7.0	run_train	2022-08-29T11:53:13	COMPLETED	Add Tags...	Enter Notes
1382	phi4_squeeze	p55n4	MongoObserver-0.7.0	run_train	2022-08-29T11:53:02	COMPLETED	Add Tags...	Enter Notes
1381	phi4_squeeze	p55n1	MongoObserver-0.7.0	run_train	2022-08-29T11:53:02	PROBABLY_DEAD	Add Tags...	Enter Notes
1380	phi4_squeeze	p54n3	MongoObserver-0.7.0	run_train	2022-08-29T11:52:53	PROBABLY_DEAD	Add Tags...	Enter Notes
1379	phi4_squeeze	p56n3	MongoObserver-0.7.0	run_train	2022-08-29T11:51:46	PROBABLY_DEAD	Add Tags...	Enter Notes
1378	phi4_squeeze	p51n4	MongoObserver-0.7.0	run_train	2022-08-29T11:51:46	PROBABLY_DEAD	Add Tags...	Enter Notes
1377	phi4_squeeze	p54n3	MongoObserver-0.7.0	run_train	2022-08-29T11:51:25	PROBABLY_DEAD	Add Tags...	Enter Notes
1376	phi4_squeeze	p55n4	MongoObserver-0.7.0	run_train	2022-08-29T11:51:24	PROBABLY_DEAD	Add Tags...	Enter Notes

Scaling the number of ML experiments / MLOPs - 4

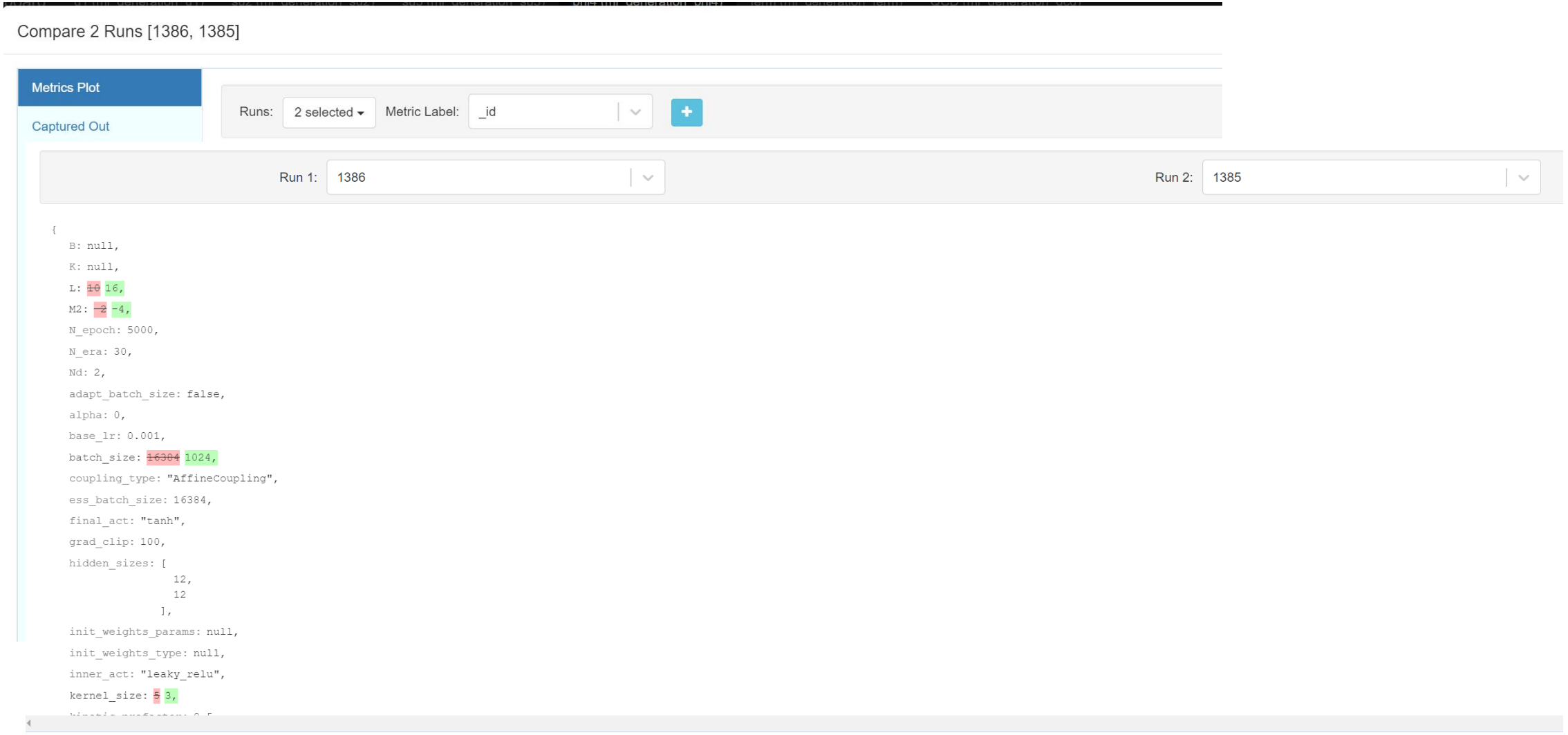




Scaling the number of ML experiments / MLOPs - 5



Scaling the number of ML experiments / MLOPs - 6



Weak Scaling Increasing Batch Size

Introducing of Batch Size

In LQCD one deals with one configuration U_μ at every moment

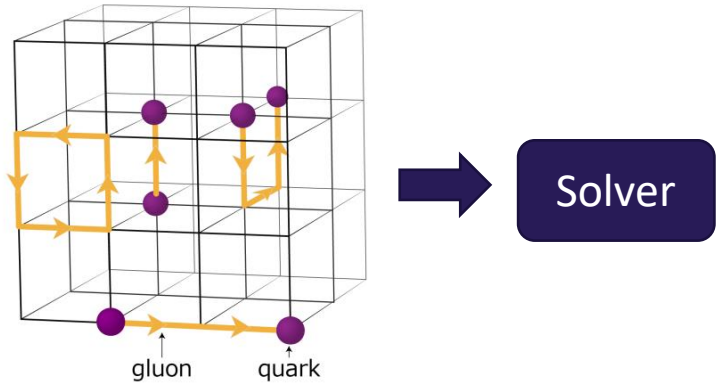


Image credit to EPJ Web of Conferences 245, 09008 (2020), CC

In ML one work with **batch** of configurations at every moment due to Stochastic Gradient Descent

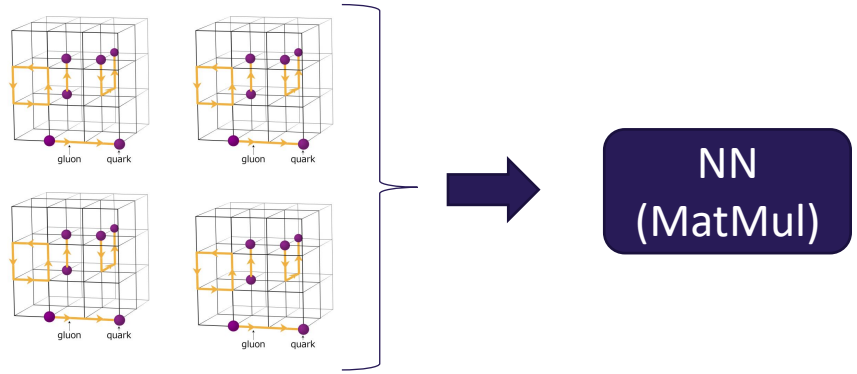


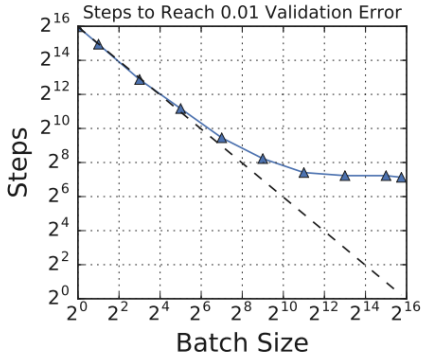
Image credit to EPJ Web of Conferences 245, 09008 (2020), CC

Increasing Batch Size

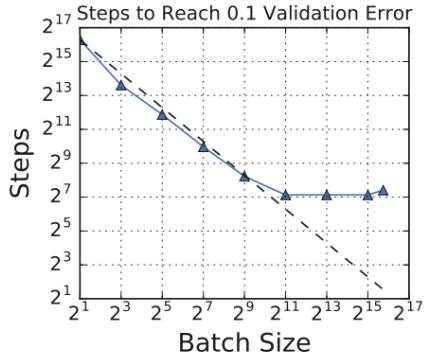
- Speeds up training
- Increases GPU utilization
- Does not increase communications between GPUs

Increasing batch size is the most common way to increase GPU utilization and do distributed training in ML

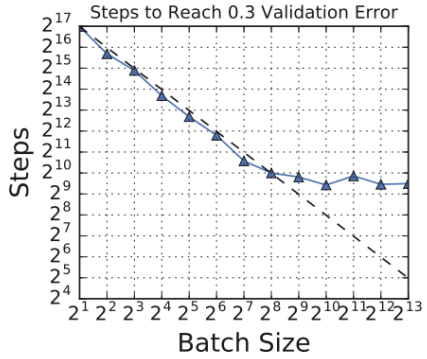
Scaling Batch Size Has Limits



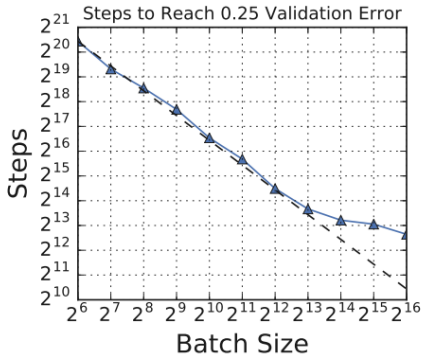
(a) Simple CNN on MNIST



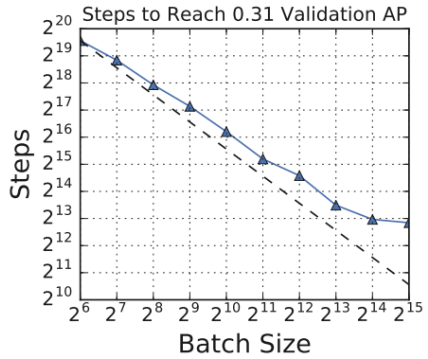
(b) Simple CNN on Fashion MNIST



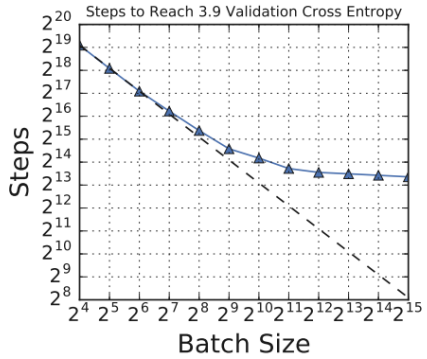
(c) ResNet-8 on CIFAR-10



(d) ResNet-50 on ImageNet



(e) ResNet-50 on Open Images



(f) Transformer on LM1B

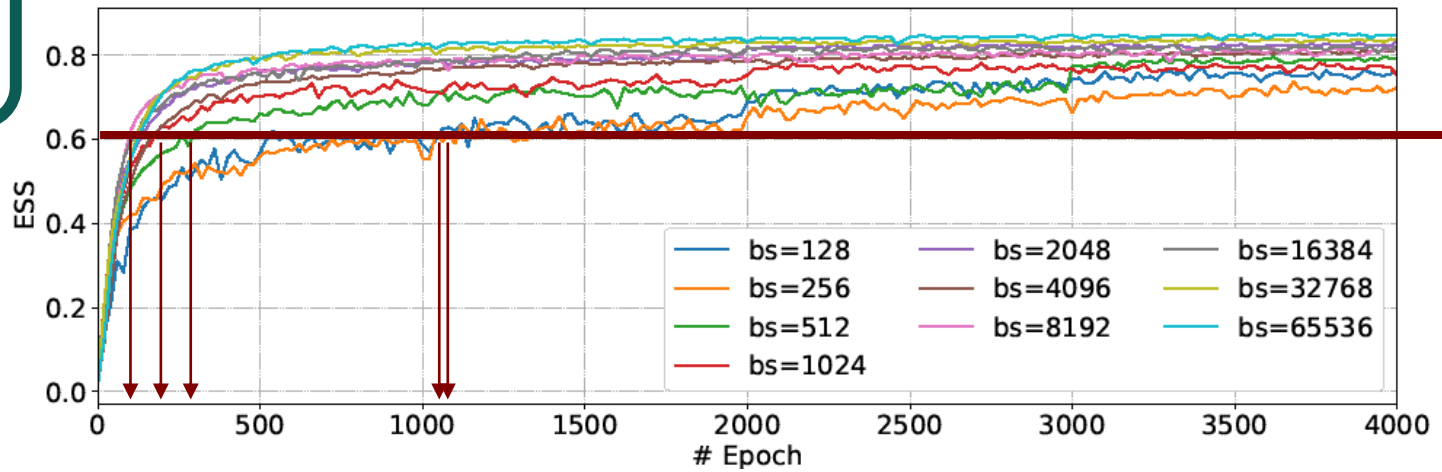
C. J. Shallue et.al. Measuring the Effects of Data Parallelism on Neural Network Training. Journal of Machine Learning Research 20 (2019) 1-49, CC

Three scaling regimes:

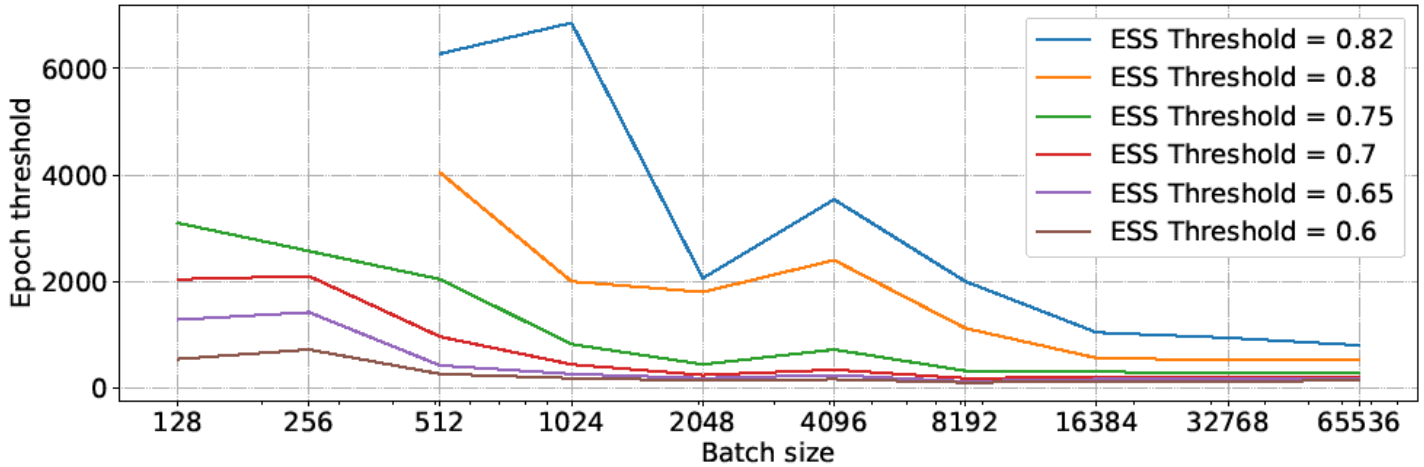
- perfect scaling
- diminishing returns
- maximal data parallelism

Scaling Batch Size in Flow-based ML model for U(1) in 2D

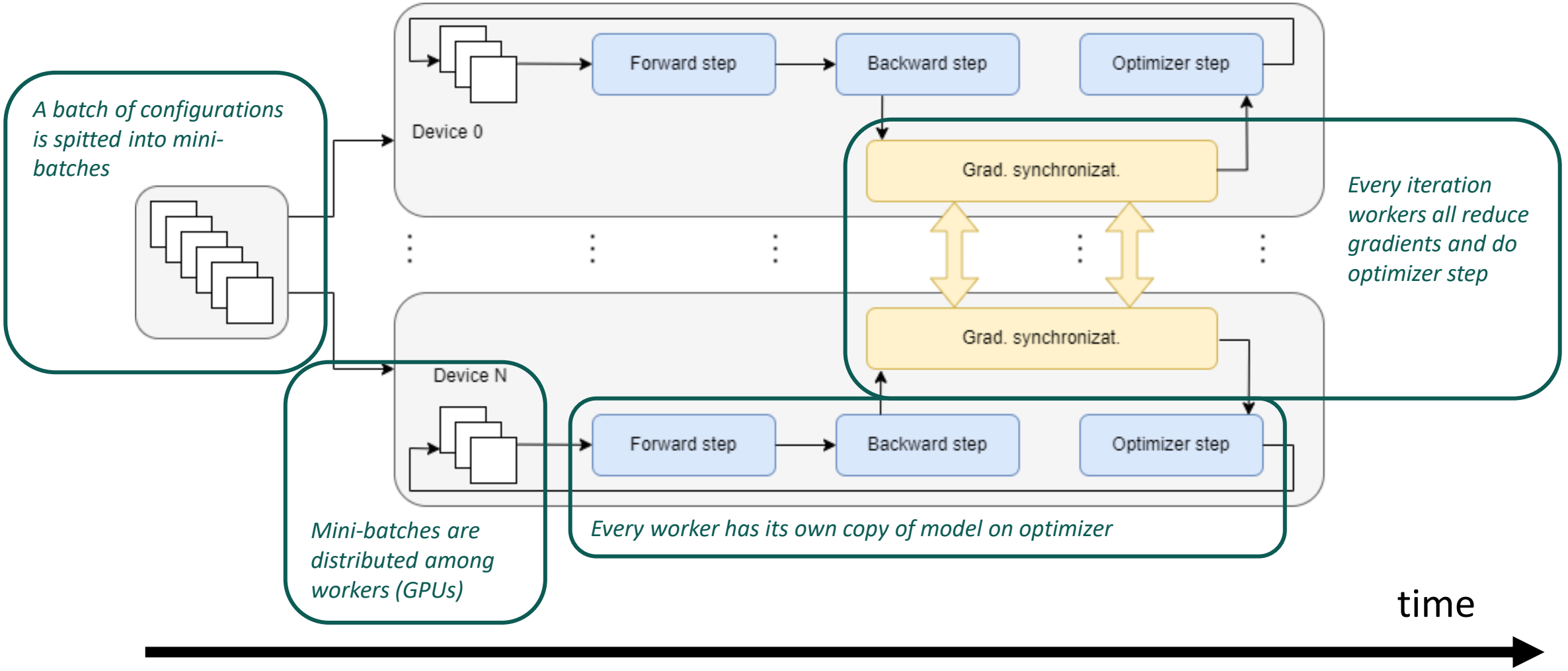
ESS [0, 1] (effective sample size) is metric of quality of generative model



Horizontal slices of training curves for different target ESS determine period of training (# Epochs)



Distributed Training: Data parallelism



Scaling Data Parallelism

Horovod was scaled on Summit up to 4600 nodes (1 Summit node = 6 Nvidia V100) with 93% efficiency

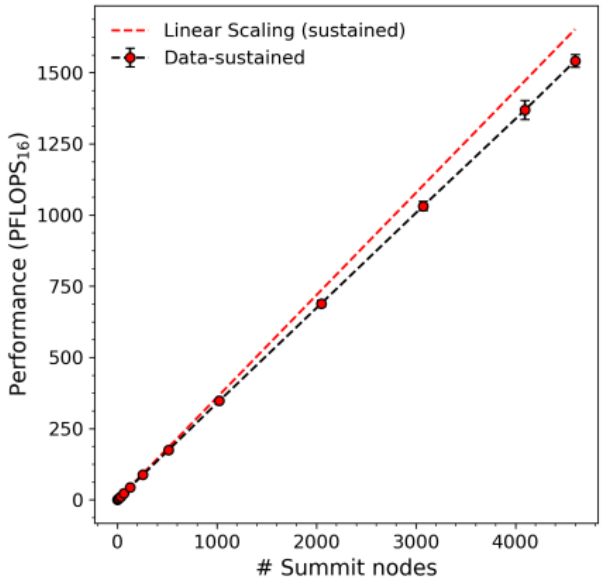


Figure 6. Scaling efficiency and Sustained Performance of distributed Deep Learning using the improved gradient reduction strategies up to 27,600 V100 GPUs.

Horovod has knobs to tune for reaching high efficiency!

PyTorch

- Scales out of the box without proper knobs tuning
- Up to my best knowledge there are no numerical demonstrations of scaling up to 4-5k nodes, but there is no reason not to scale

PyTorch DDP should scale out of the box

Towards larger memory limits: ❑ Checkpointing

Memory Allocation sources in ML

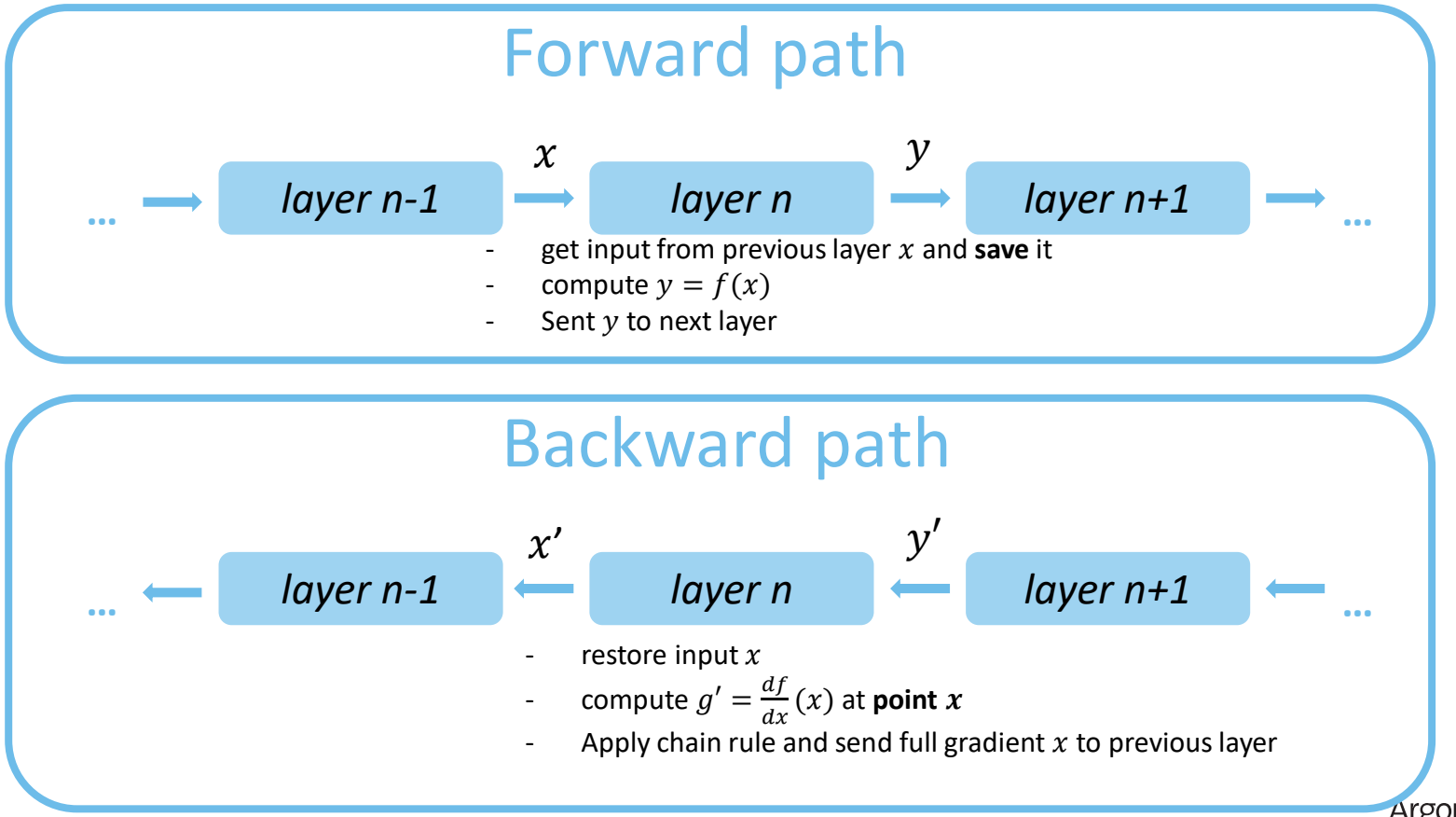
There are two main sources of memory allocations in ML

- Storing weights and gradients for **model** as well as optimizer states — bottleneck for large models
- Storing **activations** — bottleneck for large configurations, LQCD!

Storing activations allows to save compute.

Without storing activations compute $\sim O(n^2)$

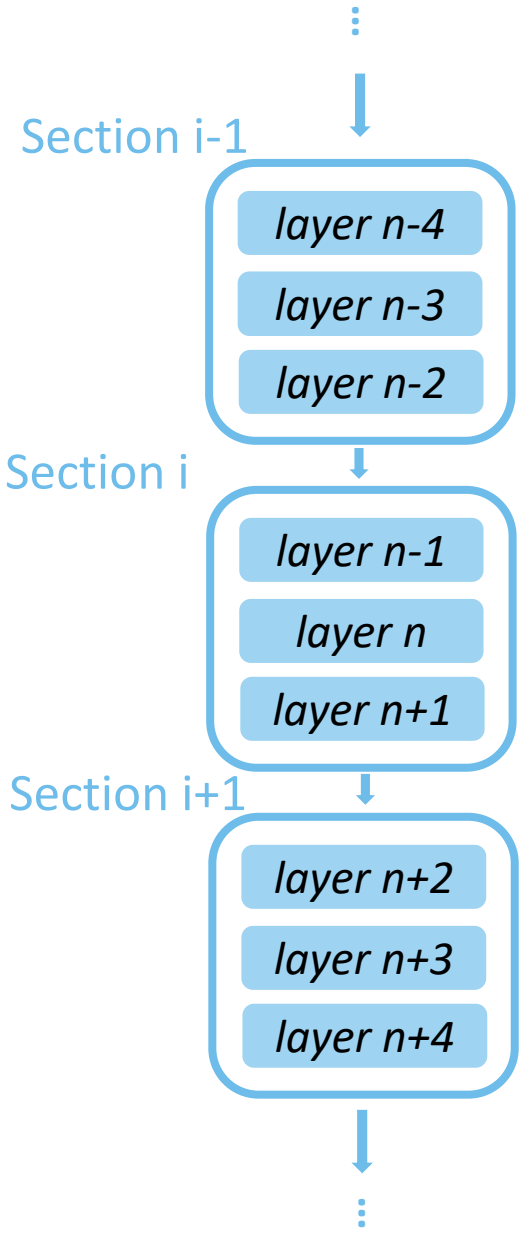
With storing activations compute $\sim O(n)$



Gradients checkpointing - 1

Through recomputation of activations during backward path memory cost can be reduced to $\boldsymbol{O(\sqrt{n})}$, where n is number of layers of NN with only **extra forward path** (30% of computation time)

Tianqi Chen, Bing Xu, Chiyuan Zhang, Carlos Guestrin, Training Deep Nets with Sublinear Memory Cost, arXiv:1604.06174



Forward Path

- Input to every section is saved in memory
- Activations inside sections are not saved

Backward path

- Recompute forward path inside a section with storing intermediate activations
- Do backward path as usual inside the section

Gradients checkpointing - 2

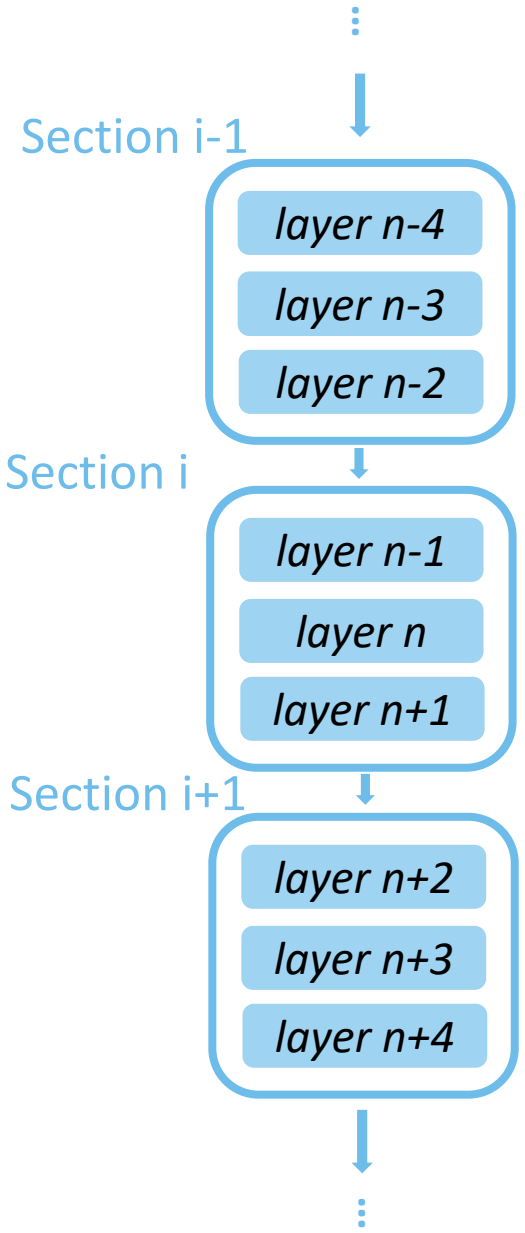
```python
class CheckpointFunction(torch.autograd.Function):
    @staticmethod
    def forward(ctx, run_function, preserve_rng_state, inputs, *args):
        ctx.run_function = run_function
        ctx.save_for_backward(*inputs)
        with torch.no_grad():
            outputs = run_function(*args)
        return outputs

    @staticmethod
    def backward(ctx, *args):
        inputs = list(ctx.inputs)
        with torch.enable_grad():
            outputs = ctx.run_function(*inputs)
        torch.autograd.backward(outputs, inputs)
        grads = tuple(inp.grad for inp in inputs)
        return (None, None) + grads
```

Forward Path

- Input to every section is saved in memory
- Activations inside sections are not saved

Backward path

- Recompute forward path inside a section with storing intermediate activations
- Do backward path as usual inside the section



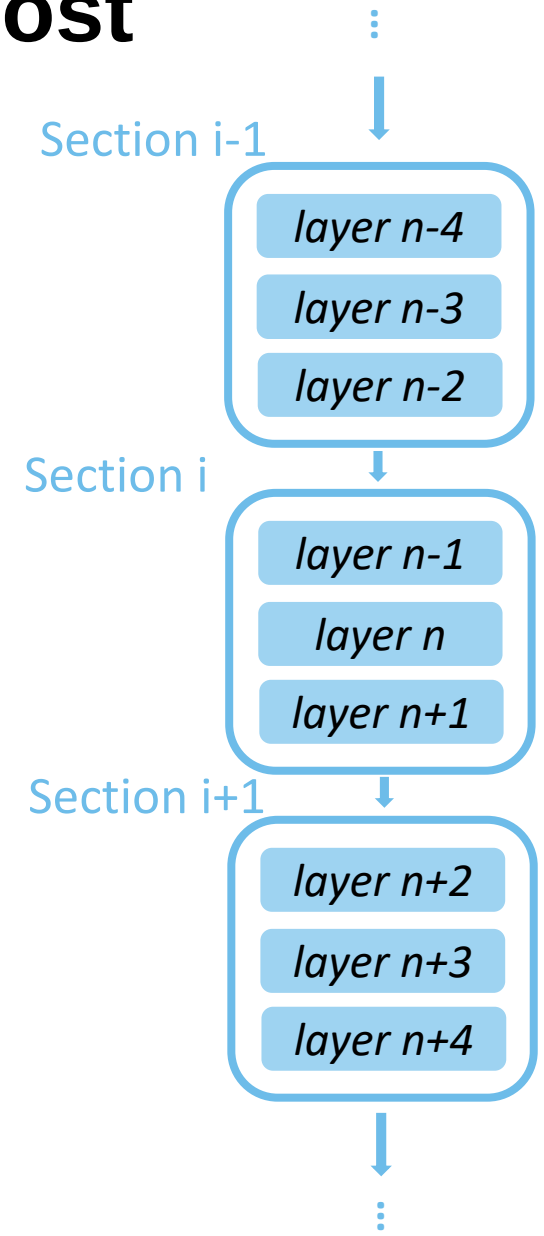
Gradients checkpointing – memory cost

Memory cost

Every section stores input activations

At every moment we need to store activations for one entire section, i.e. all layers in it

$$C_{total} \sim s\,A + \frac{n}{s}A$$



when we identical layers and equal splitting of NN to integer number of sections
s - number of sections (checkpoints)
n – number of layers in NN
A – activation size of one layer

with $\mathbf{s} = \sqrt{\mathbf{n}}$ total memory cost

$$C_{total} \sim \sqrt{n}$$

when sections contain different layers optimal number of checkpoints

$$s = \sqrt{C_{intermediate}/C_{input}}$$

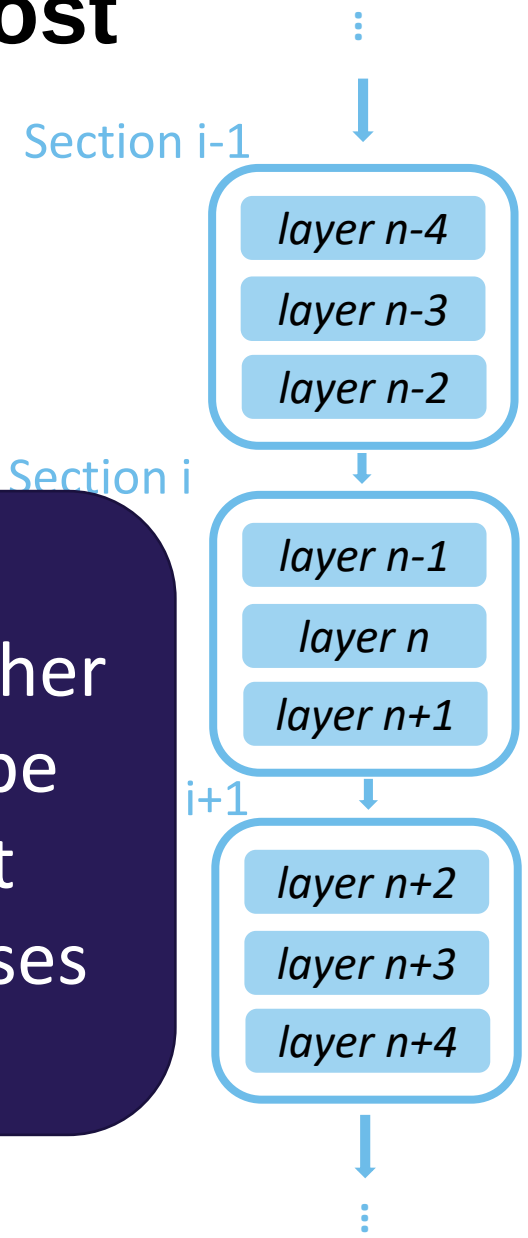
Gradients checkpointing – memory cost

Memory cost

Every section stores input activations

At every moment we need to store activations for one entire section, i.e. all layers in it

$$C_{total} \sim \boxed{s\,A} + \boxed{\frac{n}{s}A}$$



when we ide

s - number

n – number

A – activati

with $\mathbf{s} =$

Spoiler: with normalizing flows or any other invertible architecture no inputs must be saved which makes total memory cost constant as the number of layers increases

when sections contain different layers optimal number of checkpoints

$$s = \sqrt{C_{intermediate}/C_{input}}$$

Towards larger memory limits:

- Model (pipeline/tensor) parallelism or strong scaling

Distributed training: Model vs Data parallelism

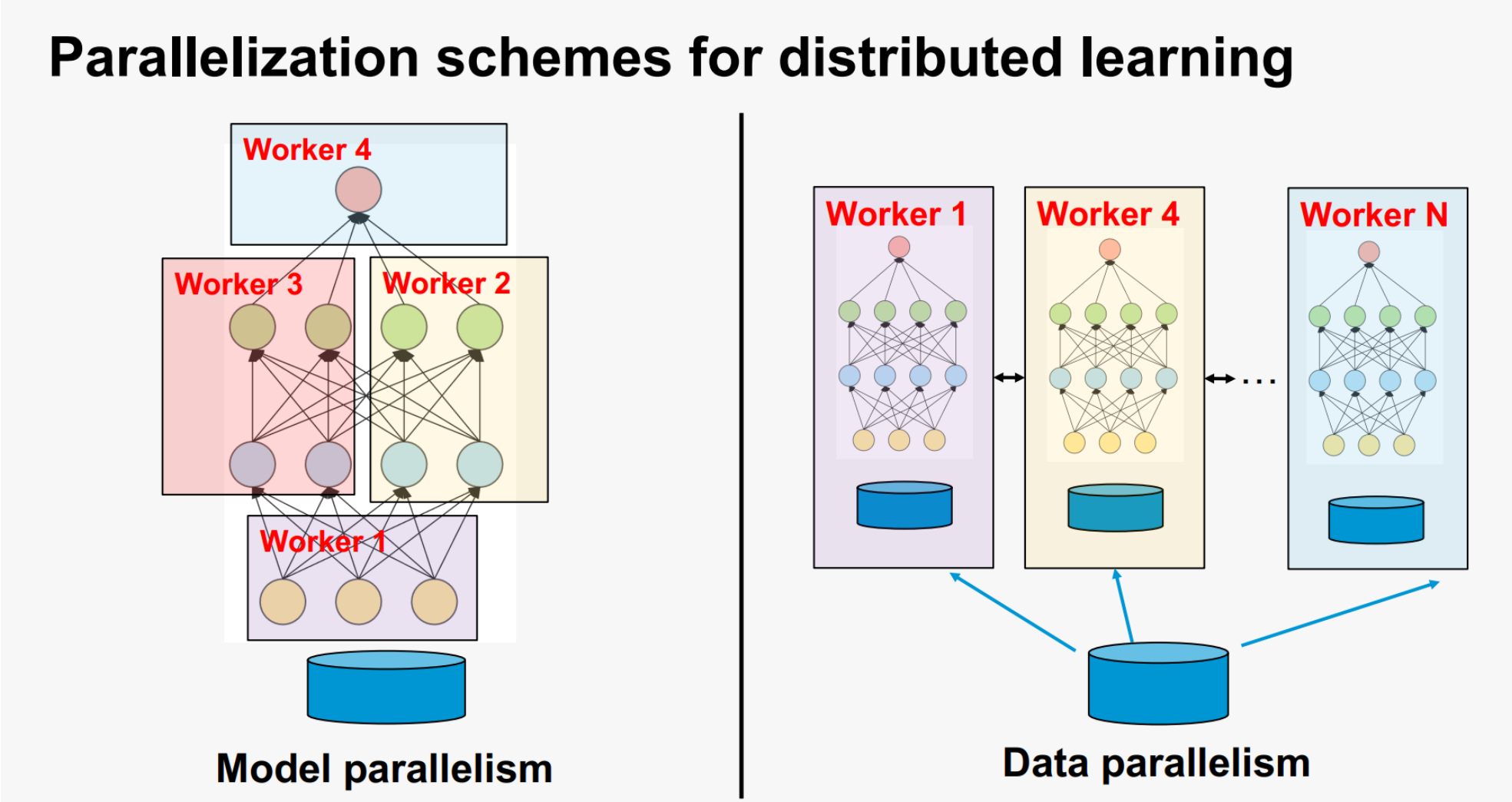
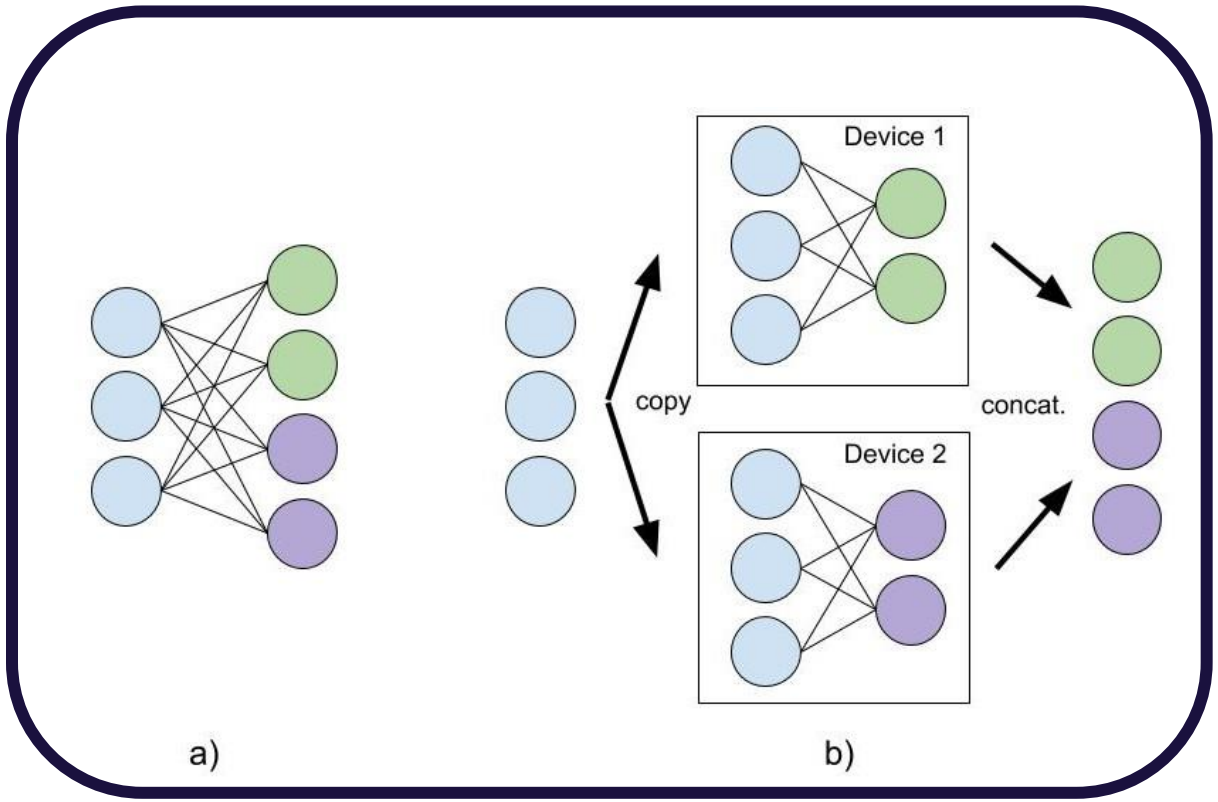
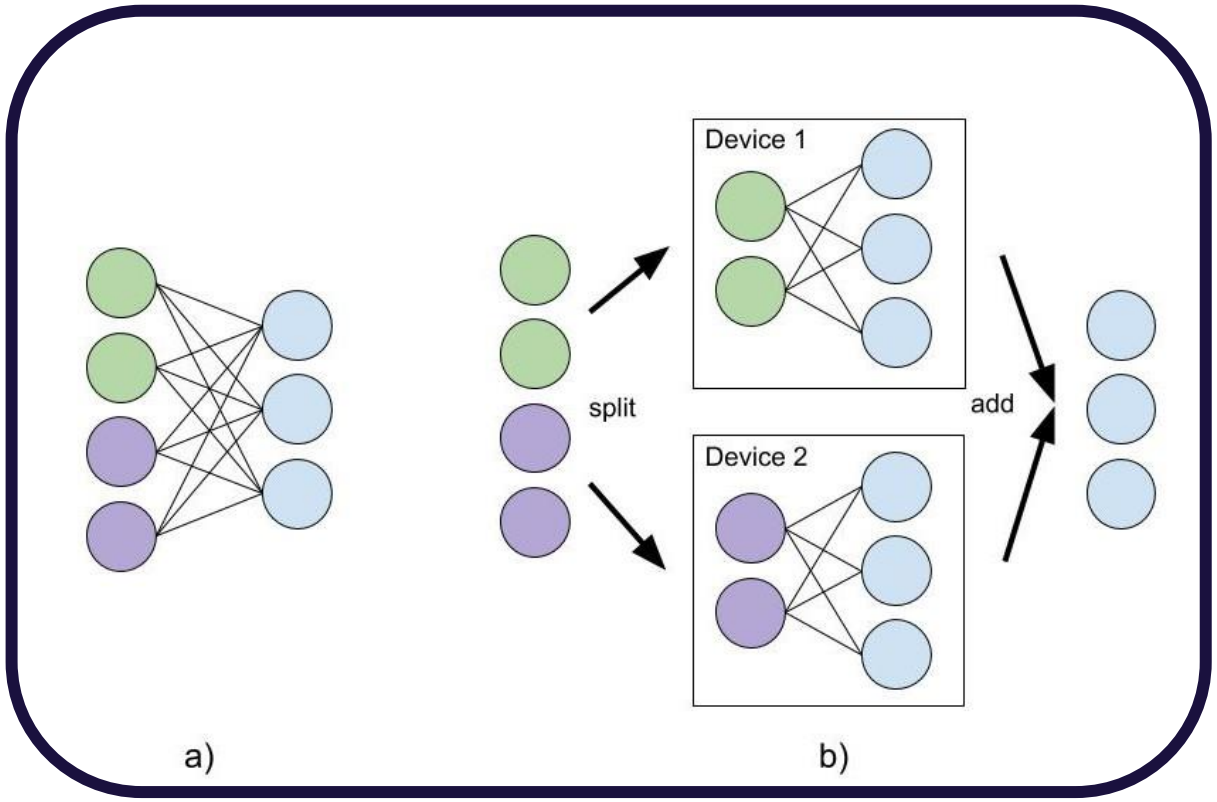


Image source is Huihuo Zheng Talk at Argonne Training Program on Extreme-Scale Computing 2019
https://extremecomputingtraining.anl.gov/files/2019/08/ATPESC_2019_Track-8_6_8-9_130pm_Zheng-Data_Parallel_DL.pdf

Tensor parallelism

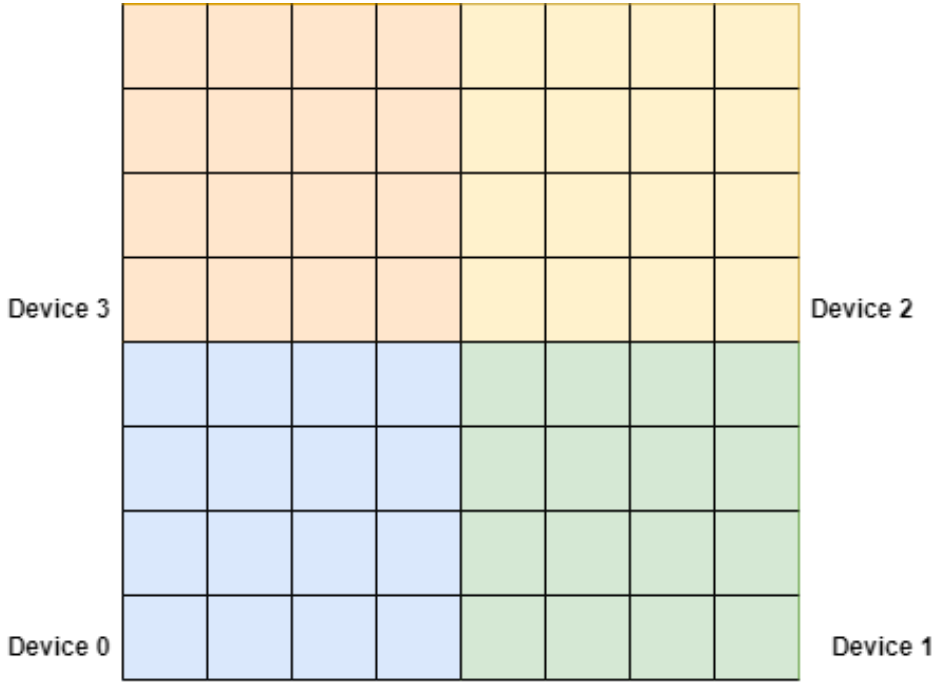


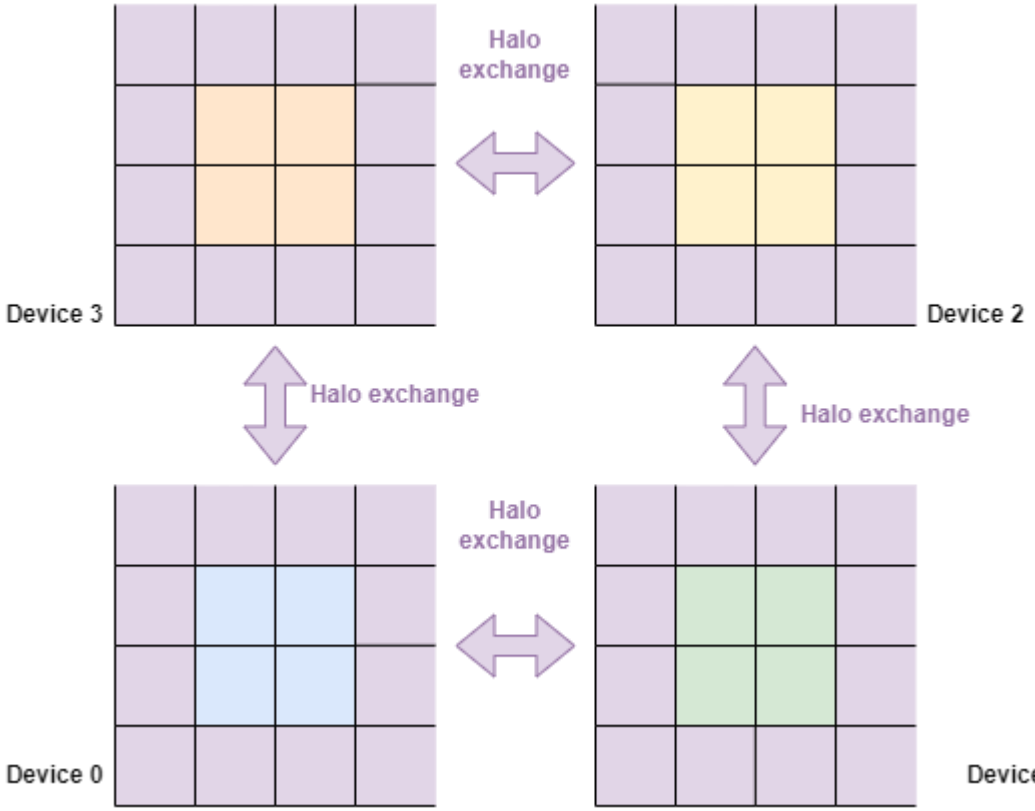


- For fully-connected layers and attention heads **amount of communications** is proportional to **number of layers and layer size**
- Collective required

Tensor parallelism for LQCD – halo exchange

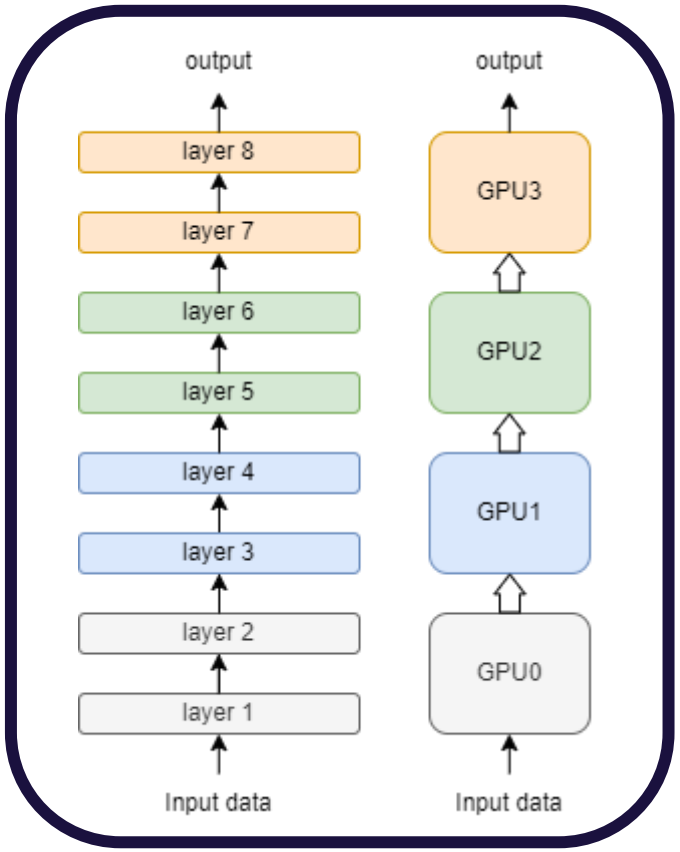
In LQCD just split configuration and do halo exchange

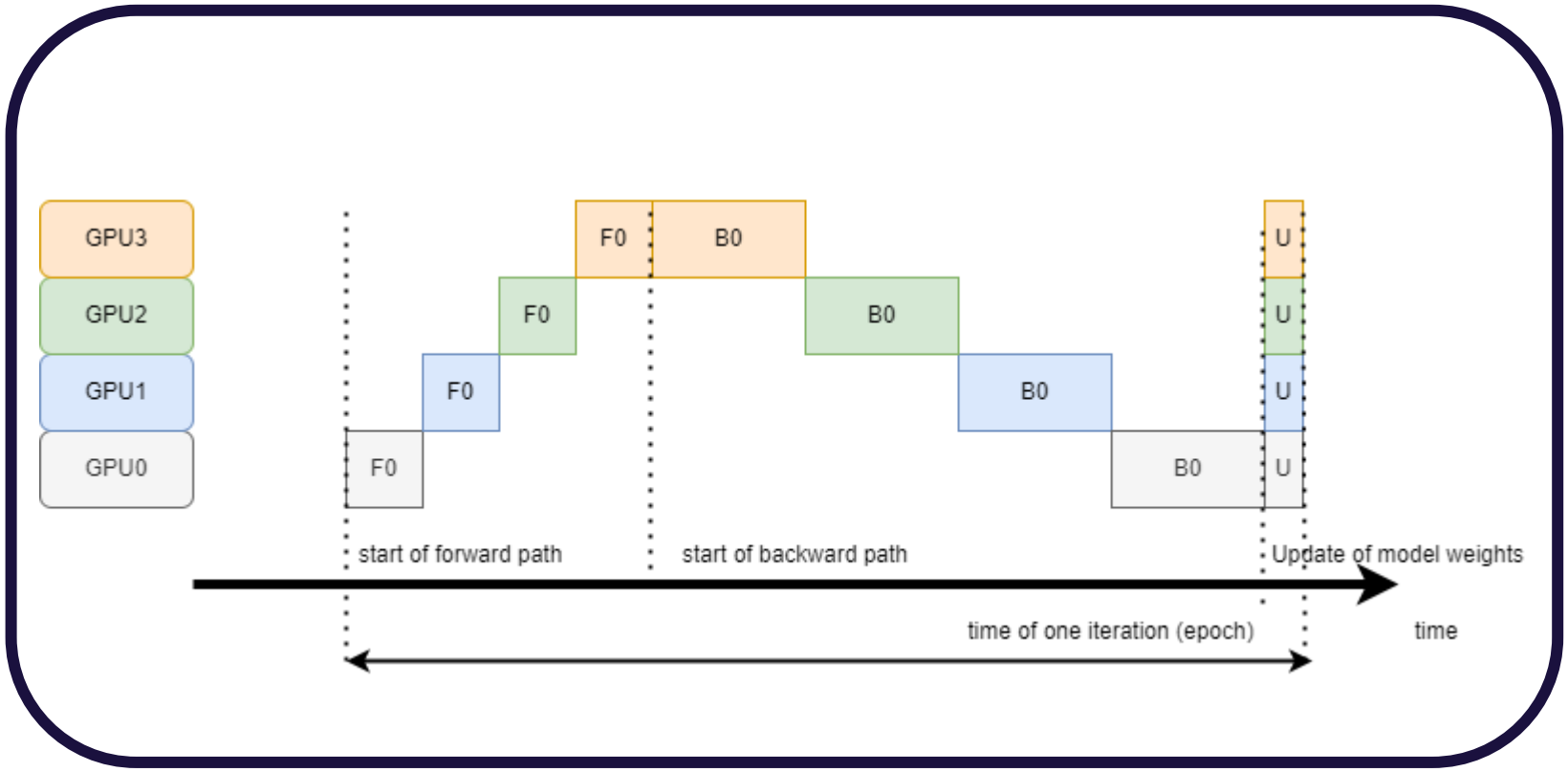




- Only halo needs to be exchanged
- Point-to-point communications

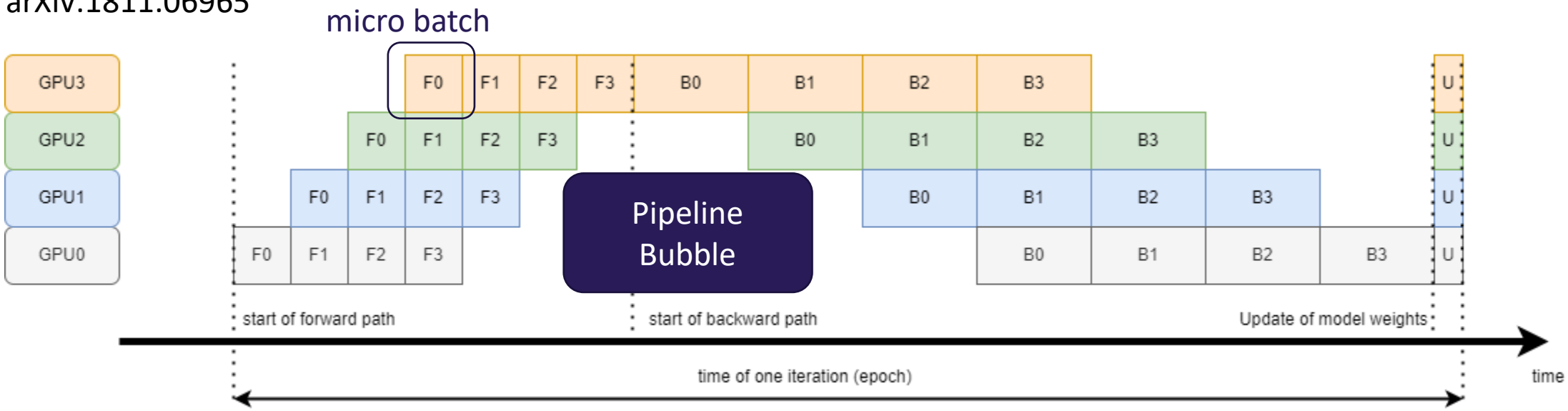
Pipeline parallelism





Pipeline parallelism: efficient training

Efficient implementation is possible and requires advanced pipeline **schedulers**
Yanping Huang et. al. *GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism.* 2018. arXiv:1811.06965



$$\text{Buble time ration} = \frac{\text{bubble time}}{\text{computation time}} = \frac{\text{number of devices (pipeline stages)} - 1}{\text{number of micro batches}} = \frac{p-1}{m}$$

$$\text{Memory cost per device} \sim \text{number of micro batches} \sim m$$

Pipeline parallelism: advanced scheduling

More efficient pipeline scheduling allows to reduce bubble time ration and memory allocation

	Bubble time ration	Activations memory cost
GPipe *Y. Huang et. al. GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism. 2018. arXiv:1811.06965*	$\frac{p-1}{m}$	$O(m)$
PipeDream-Flush *D. Narayanan et.al. Memory-Efficient Pipeline-Parallel DNN Training. arXiv:2006.09503*	$\frac{p-1}{m}$	$O(p)$
Interleaved Stages *D. Narayanan et.al. Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM. arXiv:2104.04473*	$\frac{1}{v}\frac{p-1}{m}$	$O(p)$

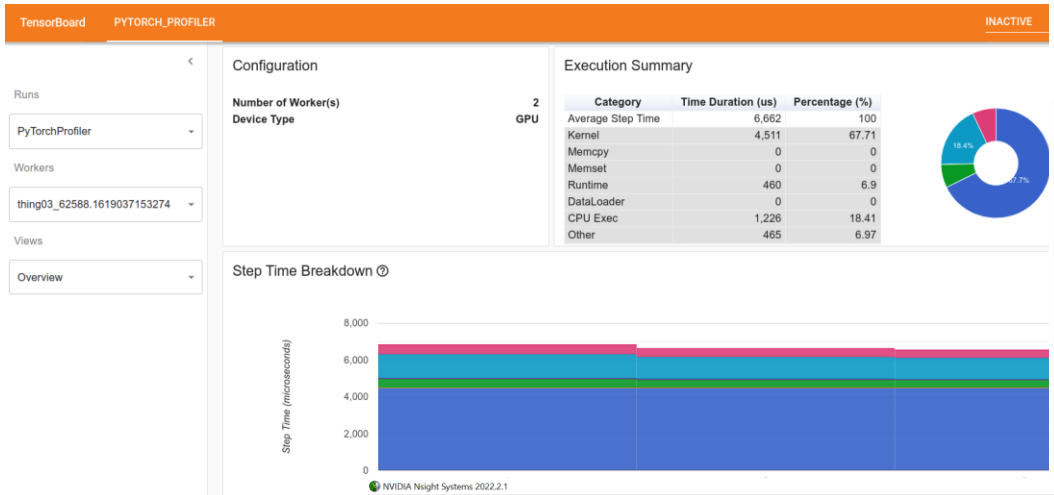
p - number of pipeline stages (devices)
m - number of micro-batches
v - number of model chuncs

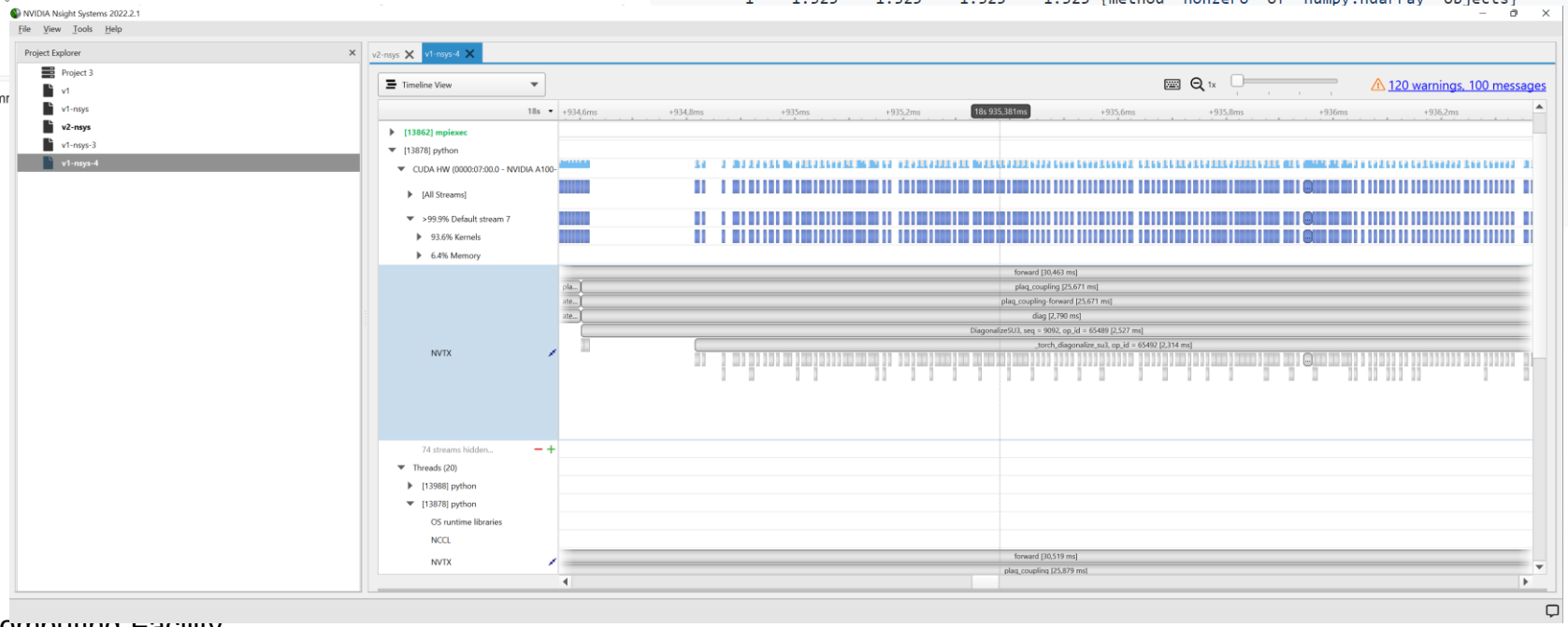
Argonne NATIONAL LABORATORY

Efficient Simulations on one GPU

- ❑ Profiling PyTorch
- ❑ Move functions to GPU / Autograd function
- ❑ JIT compiler
- ❑ Custom kernels, Sycl/cupy/Numba, DLpack

Profiling PyTorch





Python cProfile

PyTorch profile analyses only PyTorch operations which makes understanding of hotspots confusing. To profile all operations, one may use python profiler - example2/v2.py:

```
         101 function calls (89 primitive calls) in 2.422 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     10/1    0.000    0.000    2.422    2.422 module.py:1096(_call_impl)
        1    0.077    0.077    2.422    2.422 v2.py:24(forward)
        1    0.000    0.000    1.937    1.937 <__array_function__ internals>:2(argwhere)
      4/1    0.000    0.000    1.937    1.937 {built-in method numpy.core._multiarray_umath.implement_array_function}
        1    0.004    0.004    1.937    1.937 numeric.py:570(argwhere)
        2    0.000    0.000    1.933    0.967 fromnumeric.py:52(_wrapfunc)
        1    0.000    0.000    1.325    1.325 <__array_function__ internals>:2(nonzero)
        1    0.000    0.000    1.325    1.325 fromnumeric.py:1827(nonzero)
        1    1.325    1.325    1.325    1.325 {method 'nonzero' of 'numpy.ndarray' objects}
```

Move all functions to GPU / Autograd function

`torch.linalg.eig(A, *, out=None)`

• NOTE

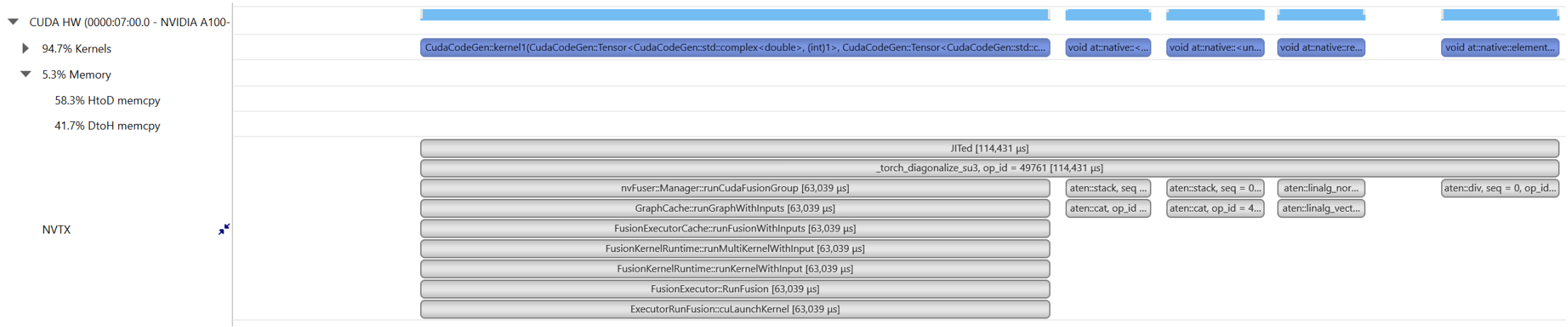
When inputs are on a CUDA device, this function synchronizes that device with the CPU.

```
class DiagonalizeSUN(torch.autograd.Function):
    @staticmethod
    def forward(ctx, U_re, U_im, *, diagonalize=_simple_np_diagonalize):
        U = ComplexTensor(U_re, U_im)
        d, P = diagonalize(U)
        ctx.save_for_backward(d.real, d.imag, P.real, P.imag, U.real, U.imag)
        return d.real, d.imag, P.real, P.imag
    @staticmethod
    def backward(ctx, gd_re, gd_im, gP_re, gP_im):
        d_re, d_im, P_re, P_im, U_re, U_im = ctx.saved_tensors
        gU_re, gU_im = grad_diagonalize(d_re, d_im, P_re, P_im, U_re, U_im, gd_re, gd_im, gP_re, gP_im)
        return gU_re, gU_im


def diagonalize_suN(U):
    d_re, d_im, P_re, P_im = DiagonalizeSUN.apply(U.real, U.imag)
    return ComplexTensor(d_re, d_im), ComplexTensor(P_re, P_im)
```

Just in time (JIT) Pytorch compiler

Fuse a lot of small cuda kernels to one kernel
PYTORCH_NVFUSER_ENABLE=complex

```
@torch.jit.script
def _torch_diagonalize_su3(U):
```



Custom Kernels

Write custom CUDA or SYCL kernels
https://pytorch.org/tutorials/advanced/cpp_extension.html

or bind to existing LQCD codes

or use NUMBA


```
diag (torch): 21.35-23.43ms
diag (jit)  : 18.01-20.22ms
diag (sycl)  : 3.48-4.00ms
diag (numba)  : 2.77-3.07ms
```

Summary:

Invest time in development only when necessary

Acknowledgment

This work is supported by the Argonne Leadership Computing Facility, which is a U.S. Department of Energy Office of Science User Facility operated under contract DE-AC02-06CH11357