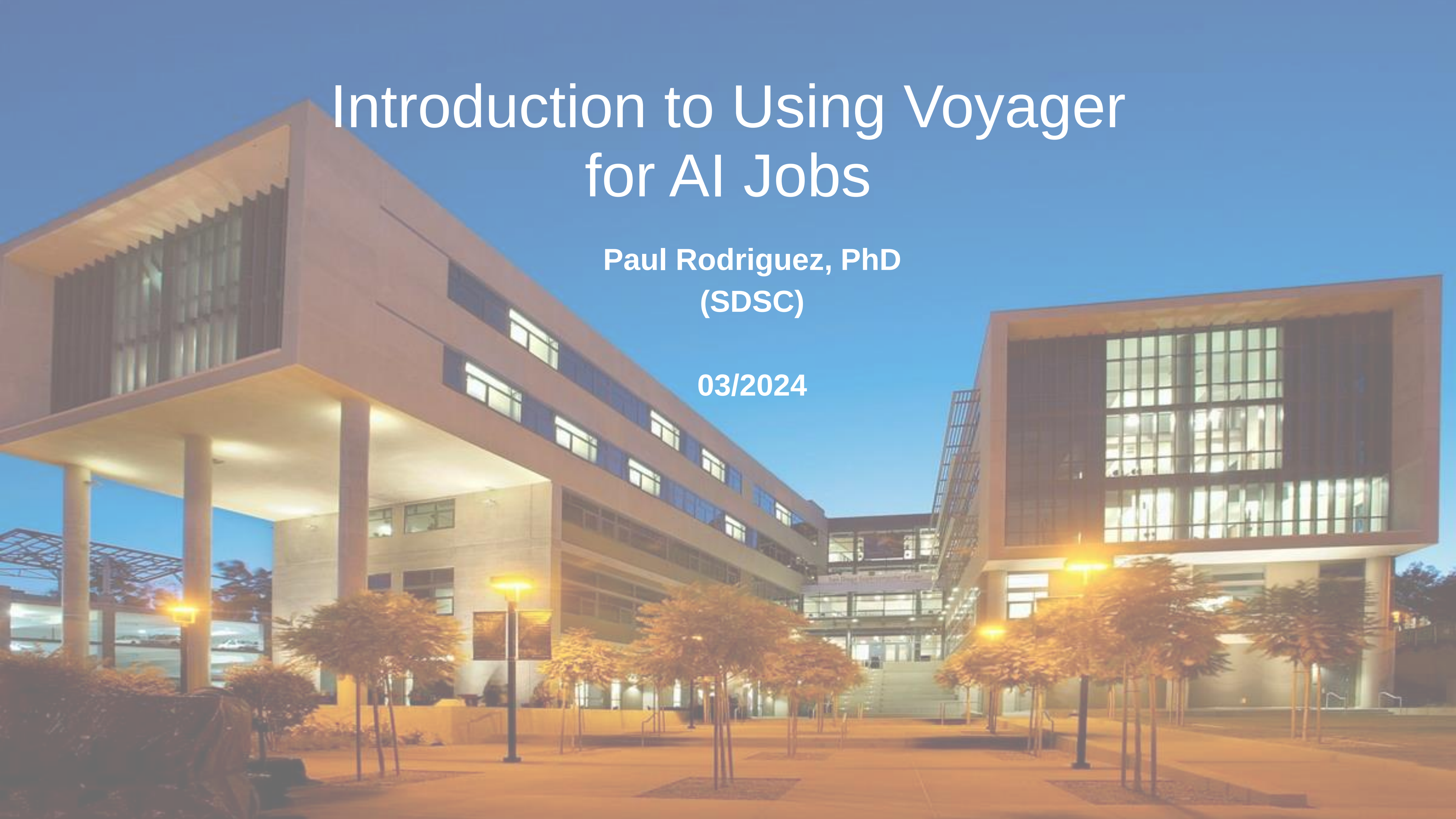
Introduction to Using Voyager for AI Jobs

Paul Rodriguez, PhD
(SDSC)

03/2024



Outline

- Getting Started
  - Habana model references
  - Porting code
  - Parallelization
  - Deepspeed demo
  - Hugging Face

Voyager Applications

- Gaudi supports standard Pytorch deep learning applications and is particularly good for scaling out training
- Voyager is readily available for testing and development
- Many applications have been tried out and evaluated
- Habana has done extensive benchmarking as well

Some of the Applications on Voyager

Project	Model
Data Driven Weather Prediction	U-Net
High energy physics	GNN
Cardiac image analysis	U-Net
Biomedical text analytics	BERT DL models
Ultrasound computed tomography	U-Net
Dose prediction in cervical brachytherapy	U-New
Systems biology	Dense neural network
Atmospheric sciences	VAE model
Human microbiome research	Categorical VAE
Astronomy	NN
Cognitive Neuroscience	CNN

Some of the Applications on Voyager

Project	Model
Natural language processing	Transformers
Biochemistry – Molecular Dynamics	VAE, AAE, ANCA-AE
Camera trap animal detection	Context R-CNN
Hyperdimensional computing	Graph architecture
Computer vision	VGGnet
E2E ML Pipeline for complex dataset	Hugging Face
Application of DL in Radiology	3D DL models (VGG, ResNet)
MVAPICH MPI implementation on Voyager	Not applicable
Analyzing EEG data with DL	CNN
Research accessibility via visual representation	Diffusion model
Hugging Face GPT2-XL model with 1.5 Billion parameters and GPT3-XL with 1.3 Billion parameters	Large language models

Training of the Hugging Face GPT-2 XL and GPT3-XL model with DeepSpeed ZeRO on Voyager

- Hugging Face GPT2-XL with 1.5 Billion parameters
- GPT2-XL numbers are from Synapse version 1.7.0 and with a Global BS of 512
- GPT3-XL with 1.3 Billion parameters
- GPT3-XL numbers are from Synapse version 1.8.0 and with a Global BS of 2048
- DeepSpeed includes ZeRO (Zero Redundancy Optimizer), a memory-efficient training tool

GPT2-XL pretraining throughput.

# Devices	Samples Per Second	Tokens Per Second	Ideal Throughput (calculated assuming ideal linear scaling of 100%)	Scaling efficiency	Grad Accumulation Steps
8	19.17	19630	19.17	100%	8
16	37.50	38404	38.34	98%	4
32	72.63	74370	76.68	95%	2
64	119.00	121856	153.36	78%	1
128	233.42	239022	306.72	76%	1

GPT3-XL pretraining throughput.

# Nodes (# HPUs)	Samples Per Second	Tokens Per Second	Ideal Throughput (calculated assuming ideal linear scaling of 100%)	Scaling efficiency	Grad Accumulation Steps
1(8)	12.59	25793.93	12.59	100%	32
2(16)	25.02	51235.20	25.19	99%	16
4(32)	49.95	102291.87	50.38	99%	8
8(64)	102.38	209680.51	100.76	102%	4
16(128)	220.16	450892.70	201.52	109%	2

Training of Stable Diffusion Model on Voyager

- Stable Diffusion model is based on latent text-to-image diffusion model
- Used SynapseAI SW Stack is 1.9.0 pre-release.
- The result shows a good scaling rate: with 256 Gaudis, it reached 91% scaling efficiency versus 8 cards

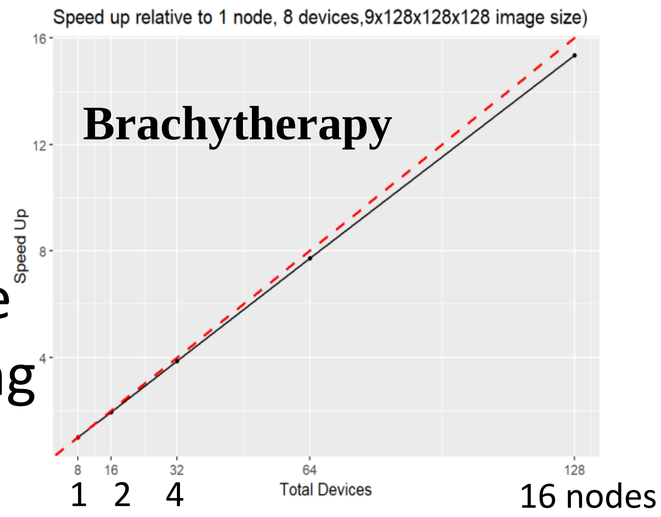
Stable Diffusion Model Scaling.

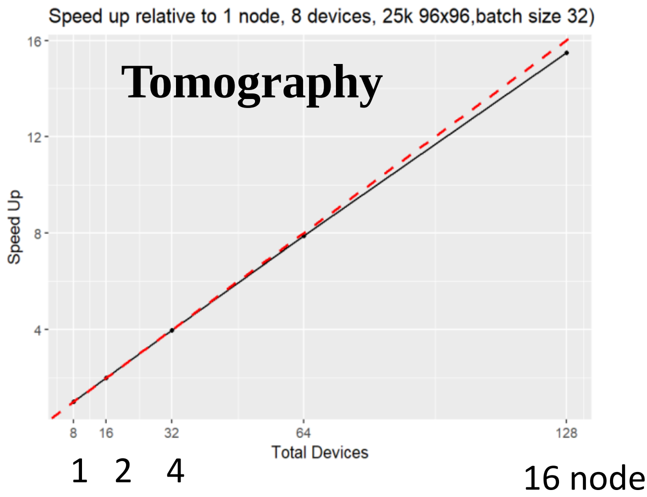
# Nodes (# HPUs)	Avg it/s	Average throughput	Scaling rate
1(8)	4.92	235.99	1.0
2(16)	4.83	464.53	0.98
4(32)	4.81	924.16	0.98
8(64)	4.80	1841.92	0.98
16(128)	4.72	3623.25	0.96
32(256)	4.48	6884.63	0.91

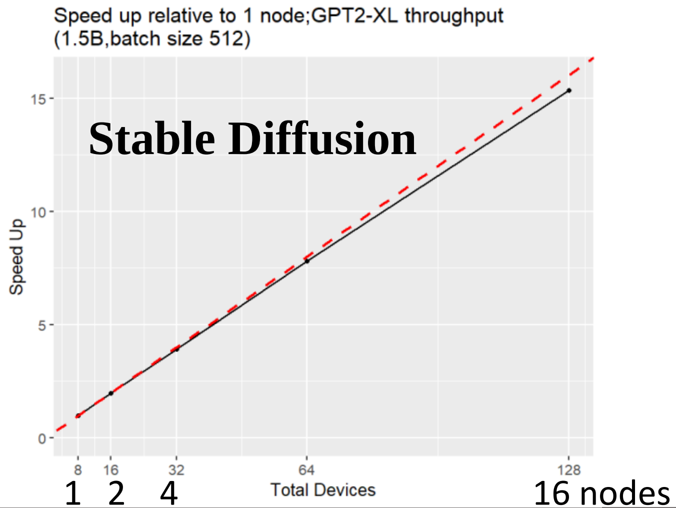
Benchmarking User Applications

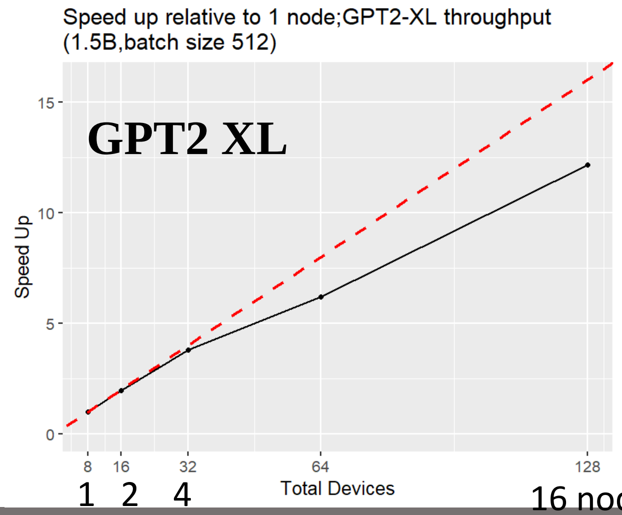
- The two UNET applications (<50M parameters) have similar scaling to Habana’s test with ~1B parameter Stable Diffusion – between 8 to 16 nodes scaling drop off relative to linear

- The ~1.5B GPT2 XL tests show drop off after 4 to 8 nodes









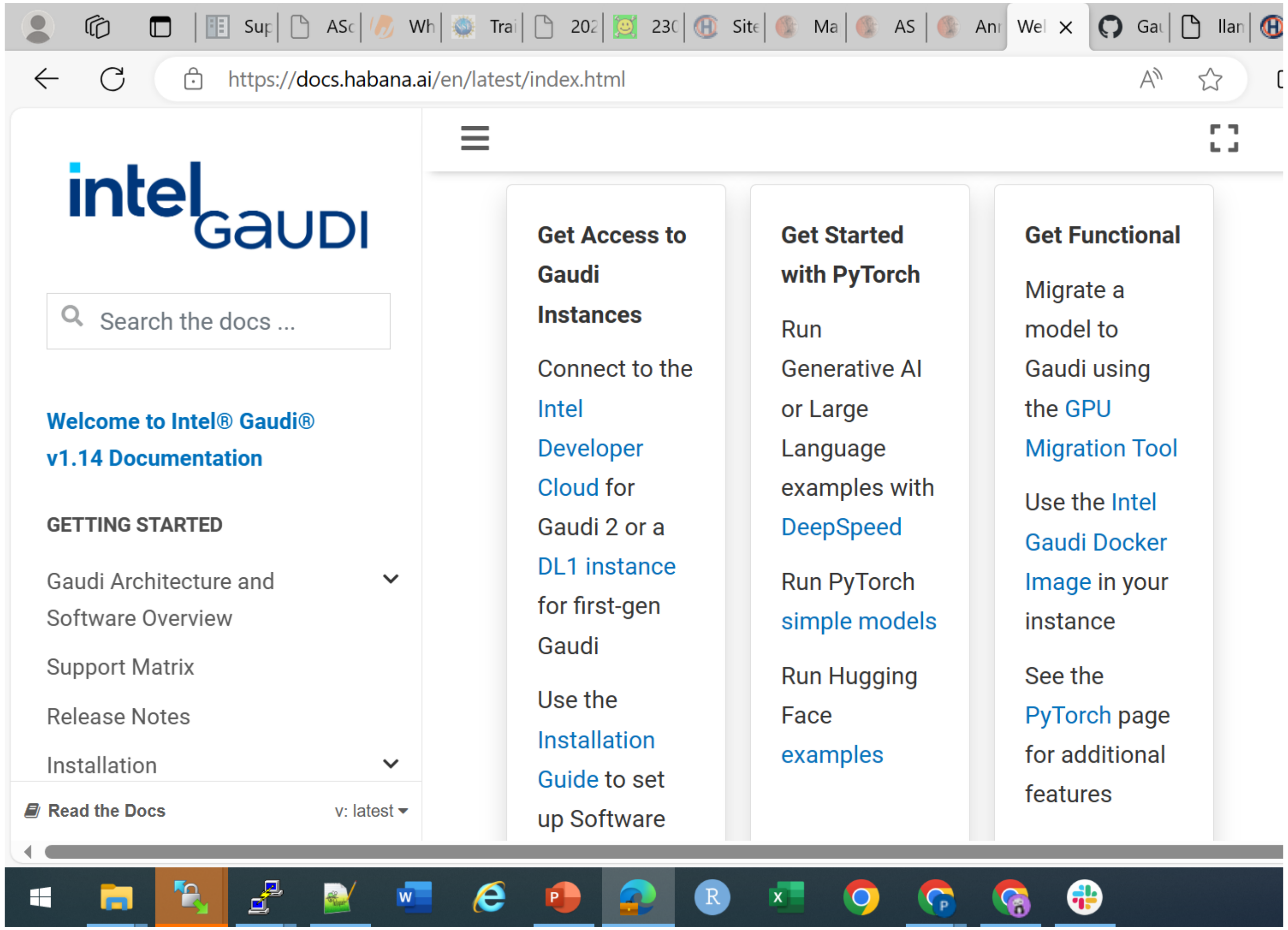
Getting started

- Where to start:
  - Own code or prior examples/tutorials/documentation

- Setting up your environment
  - Versions, pip install and docker images

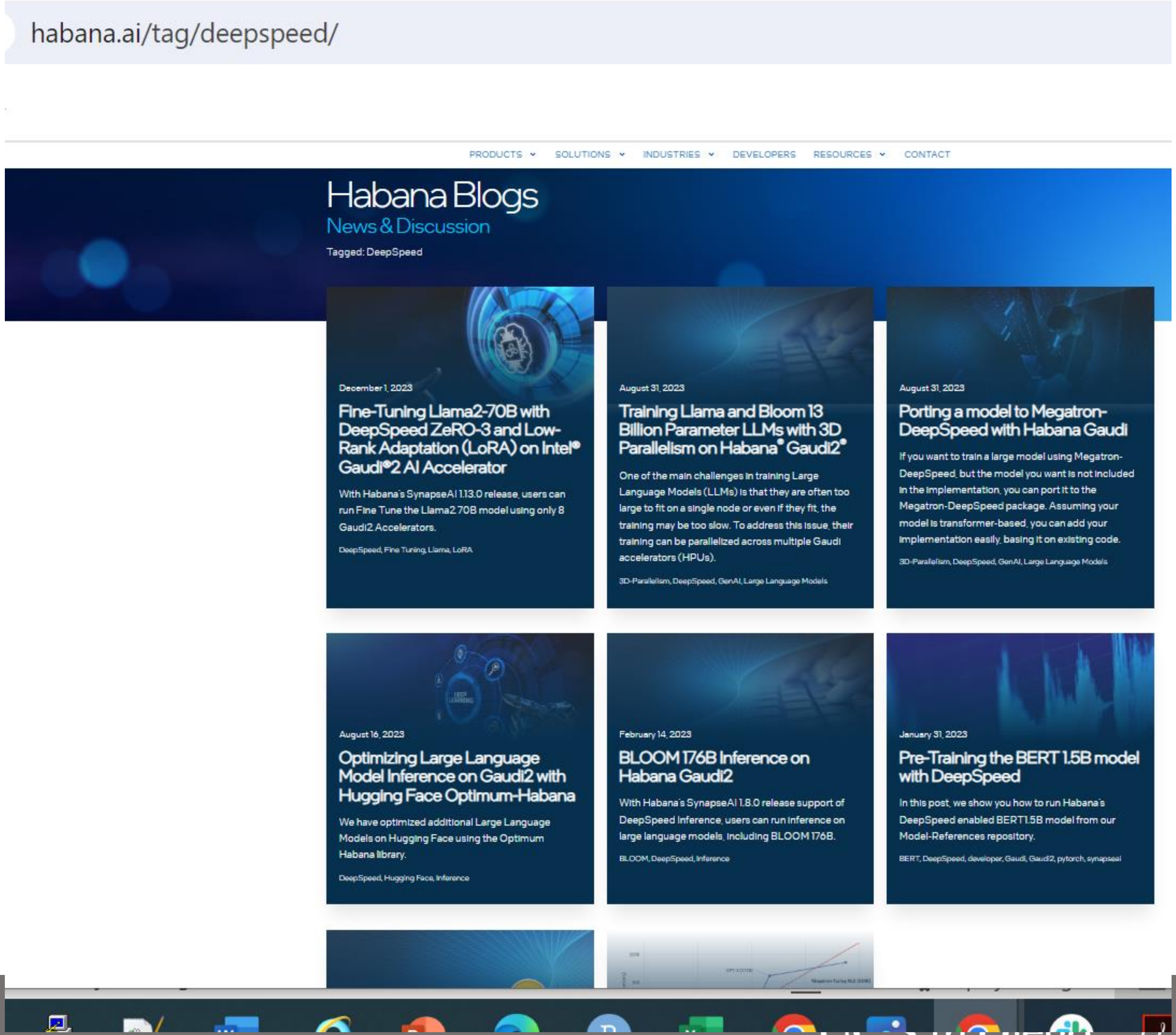
Where to start: Habana docs

- https://docs.Habana.ai/



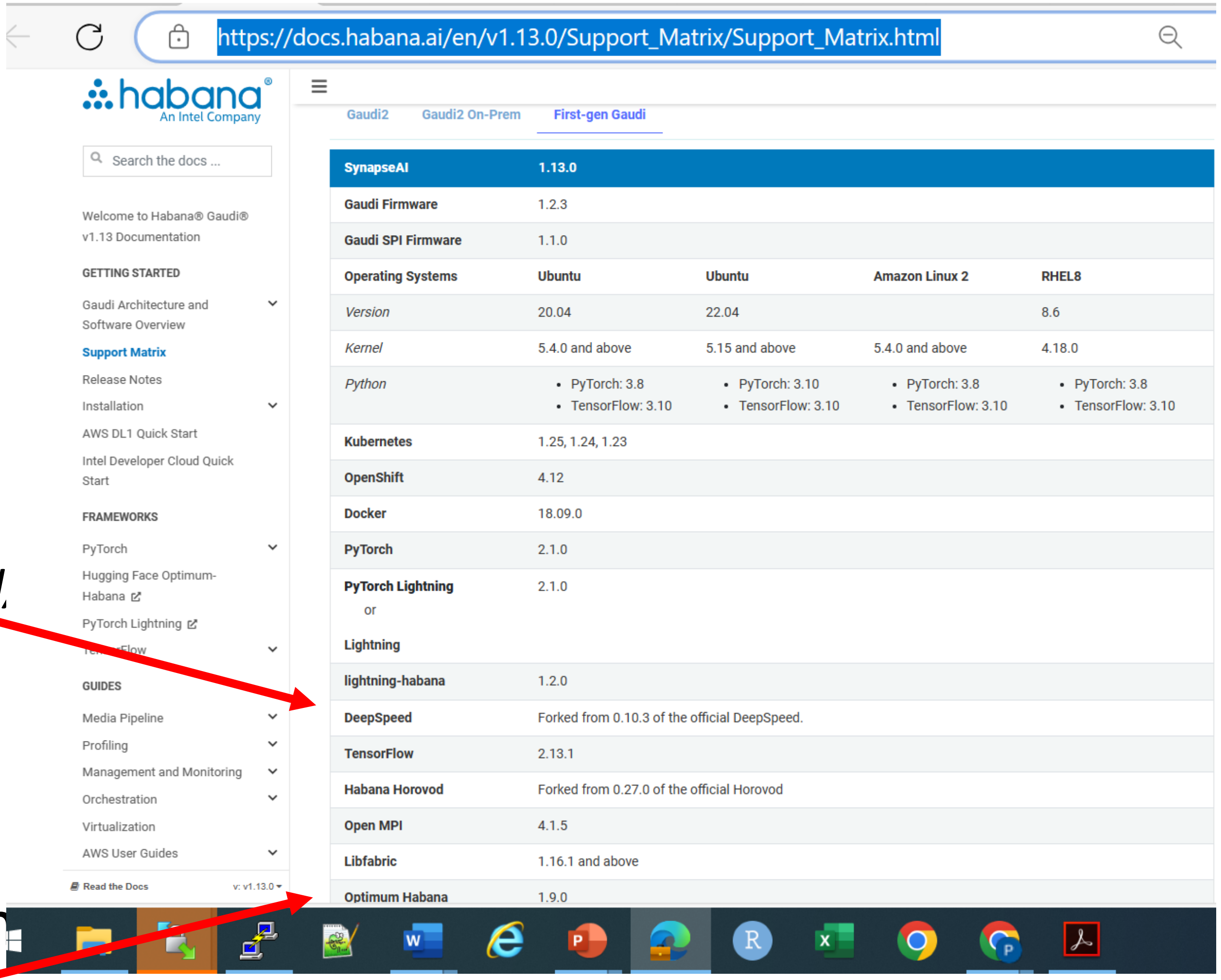
Where to start: Habana blog

- https://habana.ai/tag/deepspeed/



Where to start: Habana support matrix

- Some libraries, eg 'deepspeed', are forked - you might need to:
  - ➢ *pip install git+https://github.com/HabanaAI/DeepSpeed.git@1.13.0*


https://docs.habana.ai/en/v1.13.0/Support_Matrix/Support_Matrix.html

habana
An Intel Company

Search the docs ...

Welcome to Habana® Gaudi® v1.13 Documentation

GETTING STARTED

Gaudi Architecture and Software Overview

Support Matrix

Release Notes

Installation

AWS DL1 Quick Start

Intel Developer Cloud Quick Start

FRAMEWORKS

PyTorch

Hugging Face Optimum-Habana

PyTorch Lightning

TensorFlow

GUIDES

Media Pipeline

Profiling

Management and Monitoring

Orchestration

Virtualization

AWS User Guides

Read the Docs v: v1.13.0

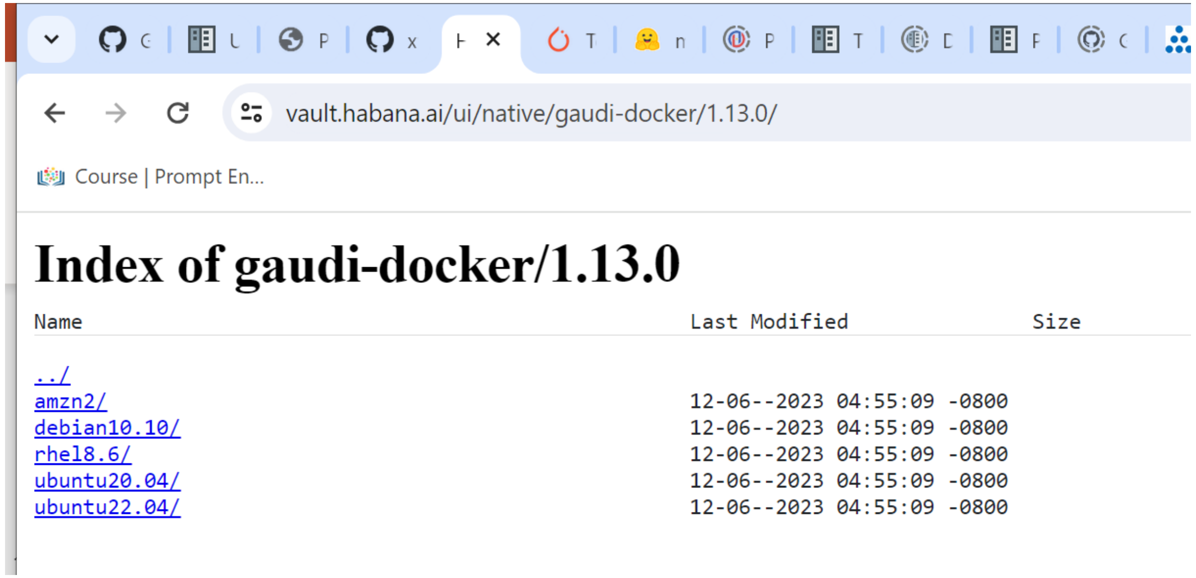
Gaudi2 | Gaudi2 On-Prem | First-gen Gaudi

SynapseAI	1.13.0			
Gaudi Firmware	1.2.3			
Gaudi SPI Firmware	1.1.0			
Operating Systems	Ubuntu	Ubuntu	Amazon Linux 2	RHEL8
Version	20.04	22.04		8.6
Kernel	5.4.0 and above	5.15 and above	5.4.0 and above	4.18.0
Python	• PyTorch: 3.8 • TensorFlow: 3.10	• PyTorch: 3.10 • TensorFlow: 3.10	• PyTorch: 3.8 • TensorFlow: 3.10	• PyTorch: 3.8 • TensorFlow: 3.10
Kubernetes	1.25, 1.24, 1.23			
OpenShift	4.12			
Docker	18.09.0			
PyTorch	2.1.0			
PyTorch Lightning or Lightning	2.1.0			
lightning-habana	1.2.0			
DeepSpeed	Forked from 0.10.3 of the official DeepSpeed.			
TensorFlow	2.13.1			
Habana Horovod	Forked from 0.27.0 of the official Horovod			
Open MPI	4.1.5			
Libfabric	1.16.1 and above			
Optimum Habana	1.9.0			

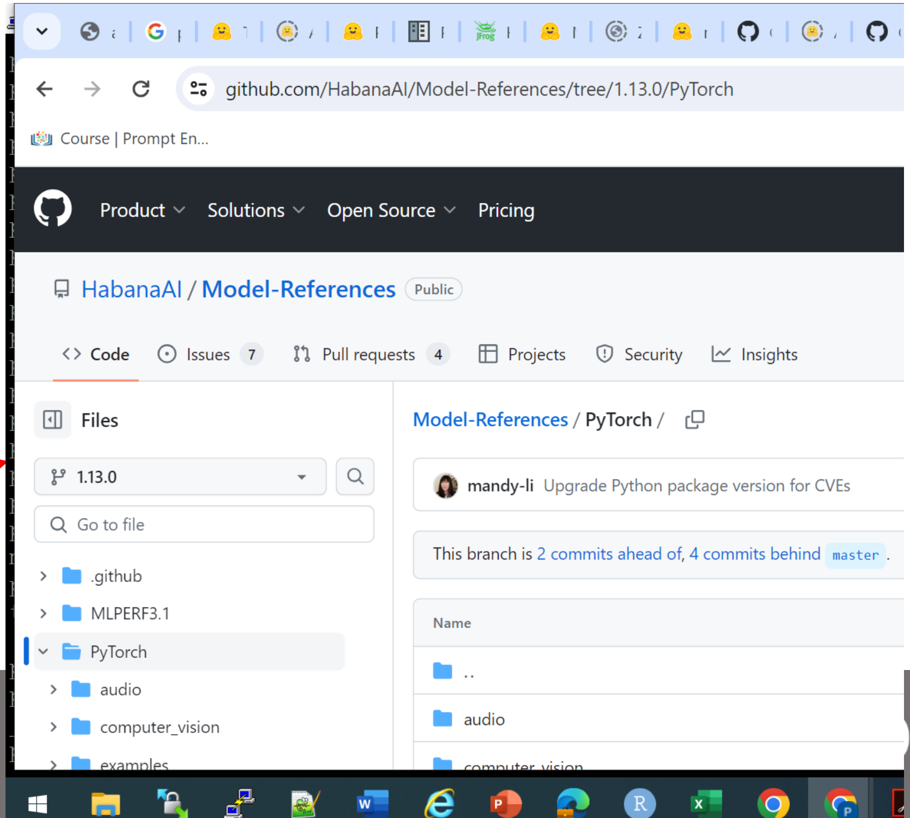
- Some libraries are developed/modified to work on Gaudi HPUs, eg hugging face's optimum-habana

Habana Docker images and Reference Models

- The docker image to use is in your Kubernetes file



- Model-References github repo has example scripts for many models



Where to start: Kubernetes scripts

- First, find example yaml files from SDSC

(https://github.com/javierhndev/Voyager-Reference-Models)

This repo has example scripts and code from Habana that were verified to work on Voyager by SDSC (JHN), and includes a variety of well-known models.

github.com/javierhndev/Voyager-Reference-Models

pt En...

README

Reference Models for Voyager

This repository contains the necessary files and links to run a collection of models on Voyager at the San Diego Supercomputer Center (SDSC). The majority of those models are the ones supported by Intel Habana link.

The last column on the list (*Verified*) indicates the last version of Synapse AI the model was tested on Voyager.

The list of models is being improved. Your feedback is greatly appreciated. Feel free to open an issue or contact m (Javier Hernandez-Nicolau) at `javierhn *at* ucsd.edu`.

Model list

Simple examples

- 'Hello World!': simplest example to run on Voyager.
- Fashion-MNIST model. It shows how to run a python script with TensorFlow.
- MPIJob Learn how to run a MNIST model in multiple HPUs.

Computer Vision

Models	Framewo
ResNet50, ResNet152, ResNeXt101	Pytorch
ResNet50 (Pytorch Lightning)	Pytorch Ligh

ResNet50 (Keras)	TensorFlow	Yes	1.13
ResNeXt101	TensorFlow	Yes	1.13

Natural Language processing

Models	Framework	Multi-node	Verified
BERT	Pytorch	Yes	1.13
BART (fine-tuning,simpletransformers)	Pytorch	Yes	1.13
Hugginface BLOOM (inference)	Pytorch		1.13
LLaMA (Megatron-DeepSpeed)	Pytorch	Yes	1.13
BERT	TensorFlow		1.13

Generative models

Models	Framework	Multi-node	Verified

Where to start: 3 development cases

- Develop your own code (we'll focus on Pytorch)
  - Start with single card execution on HPU, then parallelize code for scaling

- Your code is running on different machine (ie Expanse)
  - Migrate to HPU, parallelize code for scaling if needed

- Start with examples in Voyager Reference Models or Hugging Face
  - There are example scripts and code for a variety of known models, for pretraining, finetuning, etc…, using Bert, GPT2, stable diffusion, etc..

Coding for HPU, single device

- Running on HPU requires only adding the following

  (and remove CUDA references as needed)

```
import habana_frameworks.torch as ht

import habana_frameworks.torch.core as htcore


os.environ["PT_HPU_LAZY_MODE"]='1'            ← or use 2 for 'eager' mode
device = torch.device("hpu")
```

device name is "hpu"

Coding for HPU

- Some useful functions

```
ht.hpu.is_available()       #T or F
ht.hpu.device_count()       #an integer
ht.hpu.get_device_name()    # returns  'hpu'  on Voyager
ht.hpu.current_device()   0     #Note, all hpu devices are 0 on Voyager  (unlike gpus 0 to 7 on NVIDIA)
```

Coding for Pytorch data parallelization

- For Pytorch scripts in general: change data loader for multiple devices

```
train_sampler = torch.utils.data.distributed.DistributedSampler(train_dataset)
train_loader = data.DataLoader(dataset=___ , …. sampler=train_sampler )
```

- And wrap the model for distributed execution

```
torch.nn.parallel.DistributedDataParallel(model,  …
```

Adding code for HPU

- On Voyager, Pytorch scripts are (usually) launched in parallel with ‘mpirun’ command

```
from mpi4py import MPI
mpi_comm = MPI.COMM_WORLD
size = mpi_comm.Get_size()
rank = mpi_comm.Get_rank()
```

- Initialize the backend for handling communication between hpu devices across nodes and within a node

```
import habana_frameworks.torch.distributed.hccl
dist.init_process_group(backend=‘hccl’,  rank=rank, world_size=size)
```

Adding code for HPU

- After the optimizer.step() the parallel optimization needs to be triggered by a Habana function, ‘mark.step’

```
Training loop….

    output = model(data)
    loss = loss_function(output, target)
    loss.backward()
   optimizer.step()
   htcore.mark_step()            ← Trigger the all reduce computations
```

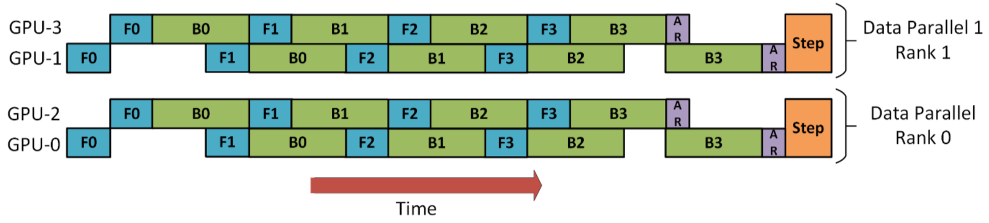
Parallel Execution

Parallelism strategies

- Data Parallelism: partition data and copy the model across devices,

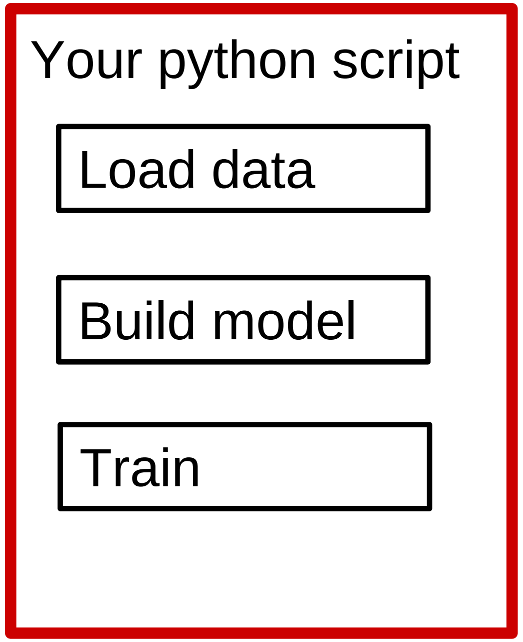
- Pipeline Parallelism: split up the model so that sets of layers are on different devices, ie inter-layer partitions

Use 'microbatches' and gradient accumulation to overlap forward/backward processing



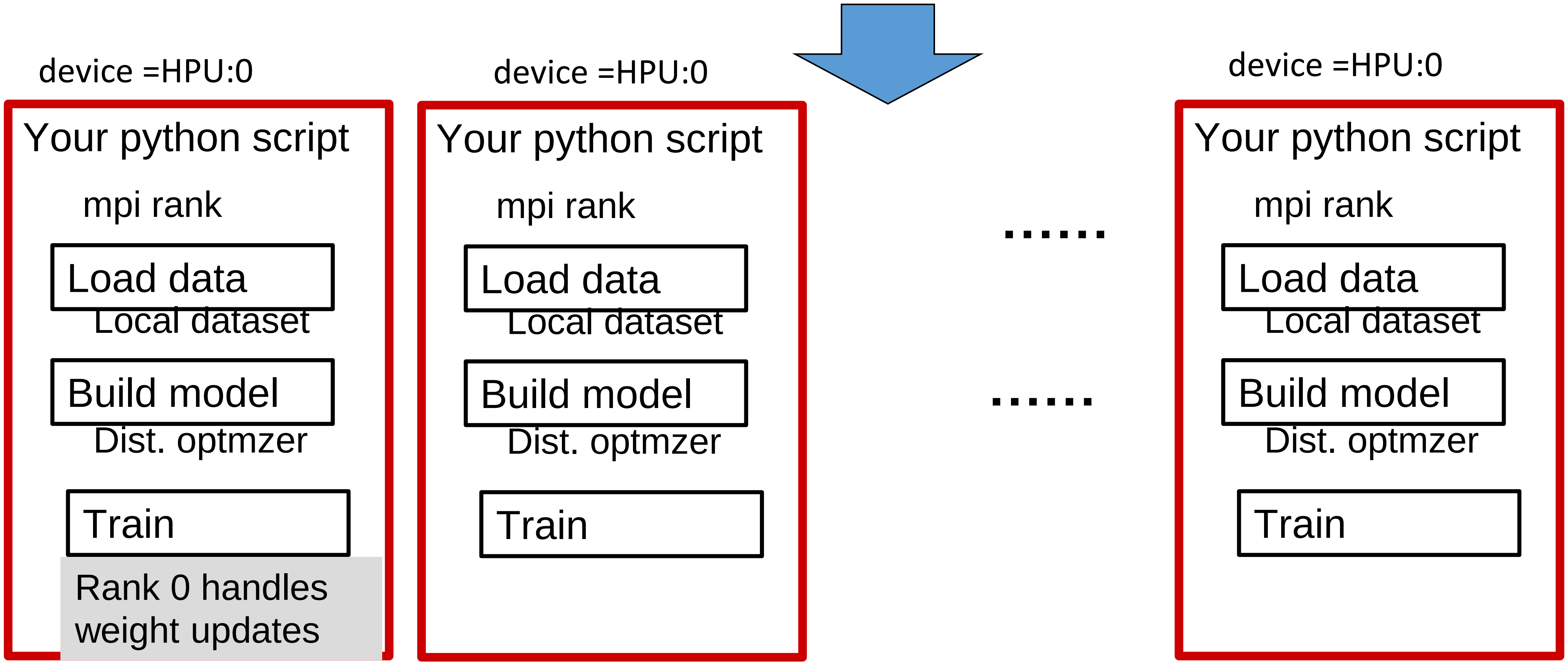
- Tensor Parallelism: split layer to different devices, ie intra-layer partitions

For example, single node, single device execution



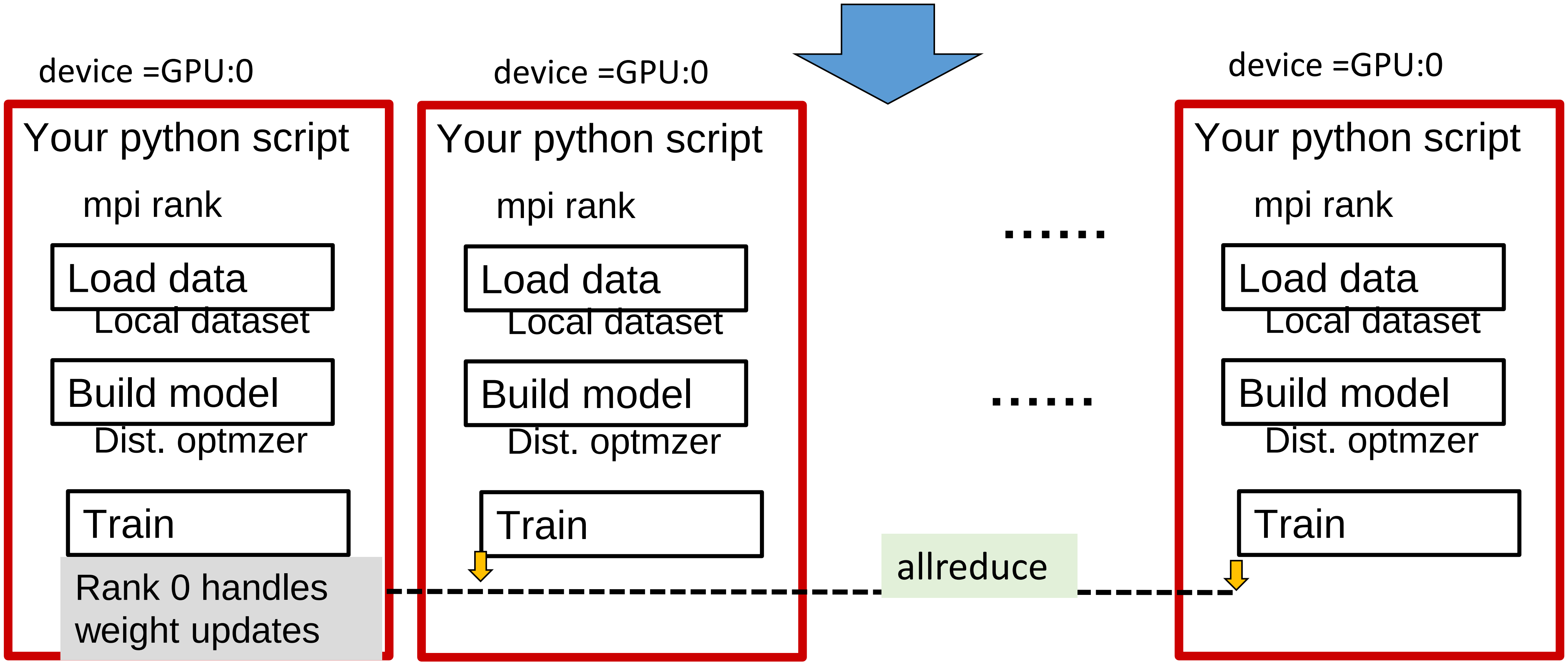
mpi launches one instance per processor

mpirun –n **number of tasks …** python3 myscript.py … arguments



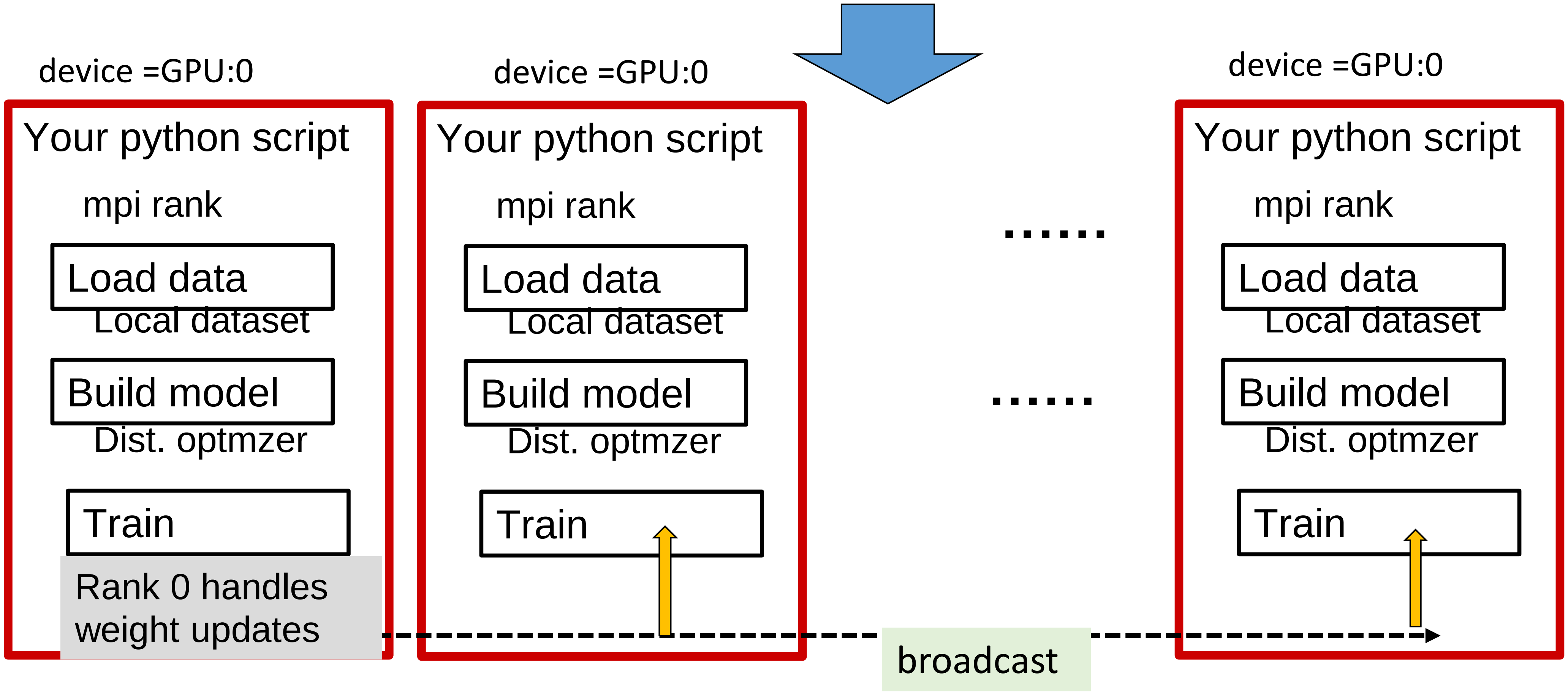
For each batch: aggregate & share weights updates

mpirun –n **number of tasks …** python3 myscript.py … arguments



For each batch: aggregate & share weights updates

mpirun –n **number of tasks …** python3 myscript.py … arguments



Bigger batch size => less comm. but it more memory

Deepspeed Python module

- Optimizers like Adam use a lot of memory b/c it tracks momentum and variances of gradients for each weight parameter update:

$$w^{(t+1)} \leftarrow w^{(t)} - \eta \frac{\hat{m}_w}{\sqrt{\hat{v}_w} + \epsilon}$$

- **Zero Redundancy Optimizer** optimizes memory storage by partitioning these terms for small increased communication

ZeRO: Memory Optimizations Toward Training Trillion Parameter Models 2020, Rajbhandari et al, Microsoft

Deepspeed: 3 stages of incrementally more partitioning

1. Optimizer state partitioning (ZeRO stage 1)
2. Gradient partitioning (ZeRO stage 2)
3. Parameter (weights) partitioning (ZeRO stage 3)

Deepspeed: 3 stages of incrementally more partitioning

1. Optimizer state partitioning (ZeRO stage 1)
2. Gradient partitioning (ZeRO stage 2)
3. Parameter (weights) partitioning (ZeRO stage 3)

All options go in a json file and passed as argument

`--deepspeed_config ds_config.json`

```
1-20:mnist_trialspipe$ more ds_config.json

   "train_batch_size":16,
   "bf16":  { "enabled": true },
   "fp16":  { "enabled": false},
   "gradient_clipping": 1.0,
   "zero_optimization":  { "stage": 0 },
   "zero_allow_untested_optimizer": true
```

Deepspeed code snippets

deepspeed initialization creates a "**model_engine**" to wrap the model

```
model_engine, opt, _, _ = deepspeed.initialize(model=model,
model_parameters=model_params, args=args)
```

training loop now uses **model_engine** for forward,backward processing

```
output = model_engine(data)
    loss = loss_function(output, target)
    model_engine.backward()
    model_engine.step()
   htcore.mark_step()
```

Voyager demo (cheat sheet)

module load Kubernetes

vi k8-mnist-mn13ds.yaml   #try stage 2

kubectl delete -f k8-mnist-mn13ds.yaml

kubectl apply -f k8-mnist-mn13ds.yaml

kubectl get pods |grep ‘p4rod’

kubectl exec –it p4rod-ds-worker-0 –n default   -- hl-smi

kubectl logs p4rod-ds-launcer-XXXX   #use launcher listed

grep ‘MaxMem’ in stdout file

A quick Deepspeed test, using mnist with extra layers (1.356B parameters)

Stage	Python max memory usage (train loop memory usage)	Avg samples per sec
1	~27.1GB (24.6GB)	~79
2	~10.3GB (7.9 GB)	~88
3	~13.3GB (5.0 GB)	~73

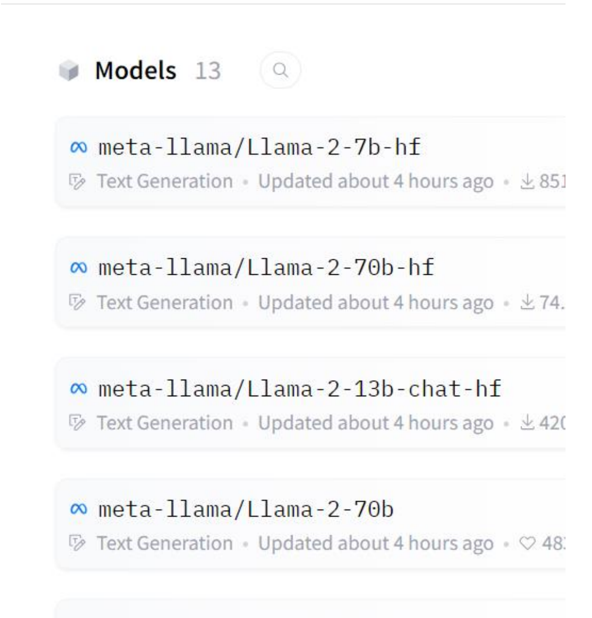
Note: the were short tests with small data, using 1 HPU node (8 devices).
Also, other parameters will impact memory, like model precision

Using Hugging Face

- Hugging Face has repository for large number of models, data and libraries for training, evaluating, fine tuning, tokenizing, image processing, etc..

- Hugging Face supports Gaudi through optimum-habana library that is set up to use ‘hpu’ . E.g. the ‘trainer’ function is now ‘Gaudi-trainer’, etc…

Hugging Face/Habana LLaMa



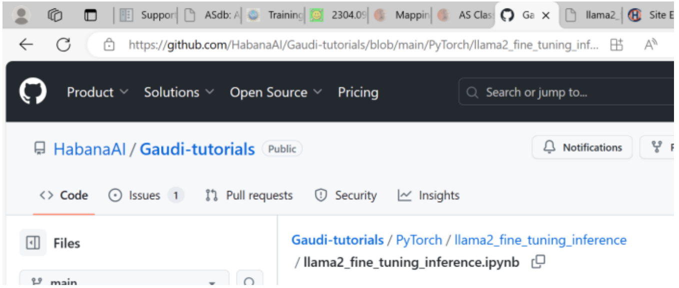
- Collection of pretrained LLMs at 7B,13B,70B parameters

  https:// https://huggingface.co/meta-llama

- Try the LLaMa2 7B model fine tuning example for Gaudi

  https://github.com/HabanaAI/Gaudi-tutorials/blob/main/PyTorch/

  Uses Deepspeed and ISOTA low-rank approximations



- These URLs have more background

https://github.com/huggingface/optimum-habana/tree/main/examples/language-modeling

https://huggingface.co/docs/optimum/en/habana/index

Hugging Face example

- Start with a Kuber. file from Voyager-Model-References

- Find/Review the ‘setup.sh’ script, ‘requirement.txt’ module list from the tutorial. Get interactive access to a node and set up modules and adjust versions if necessary

    e.g. *pip install -q optimum-habana==1.9.0 --prefix=my-local-folder*

Note, you might need to run as root within the docker image

Voyager Kub file with commands from tutorial

List pod resources and docker image info

Note: Often there a paths for model training Python scripts and bash run scripts

1. Declare environment variables, and paths
2. [Run Setup, install modules]

```
apiVersion: kubeflow.org/v2beta1
kind: MPIJob
metadata:
  name: p4rod-llpeft1320
  namespace: default
            . . .
        - image: vault.habana.ai/gaudi-
          command: ["/bin/bash", "-c"]
          args:
            - >-
              declare -xr NUM_NODES=1;
              HOSTSFILE=${HOSTSFILE:-$OM

              export PYTHONPATH=/home/LL

              huggingface-cli login --to

              declare -xr CMD1="python3
                --model_name_or_path met
                --deepspeed llama2_ds_ze
                --dataset_name timdettme
                    . . .

              /home/deepspeed --force_mu
                     --num_nodes ${NUM_N
                     --num_gpus ${NGPU_P
                . . .
                $CMD1 &> stdout1320-13b
    Worker:
      replicas: 1    . . .
```

Voyager Kub file with commands from tutorial

List pod resources and docker image info

Note: Often there a paths for model training Python scripts and bash run scripts

1. Declare environment variables, and paths
2. [Run Setup, install modules]
3. Your hugging face token
4. Declare the command and options
5. Use deepspeed binary to launch the program

Note: For large models there are sets of arguments for training, network config, optimizer, deepspeed, checkpointing, etc.. Sometimes these are in a separate bash run script

Worker resources

Hugging Face Fine-tuning tests

✓ Llama 7B parameter models runs fine-tuning with little change to deepspeed or command options in tutorial

✓ Llama 13B parameter model required some trial and error

To run fine-tuning it needed the Deepspeed option for offloading optimizer and parameter states to cpu ("ZeRO-Infinity").

It was able to run on 1 HPU node (8 devices), also on 4 HPU nodes (32 devices)

? Llama 70B parameter model – testing in progress

Hugging Face Inference Tests

Llama 7B model and 13B model both run with little change to scripts. Here's the 7B example output:

Prompt: I am a dog. Please help me plan a surprise birthday party for my human, including fun activities, games and decorations. And don't forget to order a big bone-shaped cake for me to share with my fur friends!

Response: I'm sorry, but I'm not sure if I can help you with that. I'm a bot, not a dog. However, I can tell you that there are many online resources that can help you plan a surprise birthday party for your human. Here are a few suggestions

Pinterest: This is a great place to find inspiration for party decorations, games, and activities.

Google: Search for "birthday party ideas" and you'll find a wealth of information.

Amazon: You can order decorations, party supplies, and even cakes online.

Local pet stores: Some pet stores may have party supplies or even party planners that can help you with your event.

Remember, the most important part of a party is to have fun, so don't get too hung up on

the details. Just make sure to include some of my favorite things, like a big bone-shaped cake

and lots of playtime with my fur friends. I hope this helps! Happy planning!

Other tools

- Raytune: popular distributed hyperparameter search tool. For now, it can run on 1 Gaudi node and can use all 8 HPUs for different configurations in parallel.
- Creating Docker images: you can use Habana docker images as a base and build your own image
- Jupyter notebooks: run a Kub pod with Jupyter (a URL will be generated), and do Kub port forwarding command
- Profiling, Logging, Checkpointing see Pytorch and Habana docs

Voyager would not be possible without a dedicated team of professionals and experts

Rommie Amaro
Haisong Cai*
Trevor Cooper
Chris Cox*
Javier Duarte
Tom Hutton*
Christopher Irving*
Marty Kandes
Amit Majumdar
Tim McNew*
Dmitry Mishin
Mai Nguyen

Susan Rathbun
Paul Rodriguez
Scott Sakai
Manu Shantharam
Robert Sinkovits
Fernando Silva*
Shawn Strande
Tom Tate*
Mahidhar Tatineni
Mary Thomas
Cindy Wong
Nicole Wolter

Supermicro Team
Habana Team
Arista

*A special thanks to the HPC Systems Group and the Data Center staff