Intel® oneAPI AI Analytics Toolkit on Data Center GPUs

Yuning Qiel
Orel Yehuda
Louie Tsai
Jeff Rodgers

@Intel AI Customer Engineering Team

Agenda

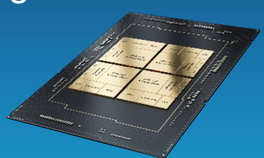
- oneAPI - AI Analytics Toolkit
- Hands-on Environment Setup
- Intel Optimizations for PyTorch on XPU
- Intel Optimizations for TensorFlow on XPU
- Distributed DL on XPU
- Intel Distribution for Python (IDP) on XPU

OneAPI - AI Analytics Toolkit Overview

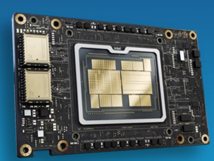
Modern Applications Demand Increased Processing

Diverse accelerators needed to meet today's performance requirements:
48% of developers target heterogeneous systems
that use more than one kind of processor or core[1]

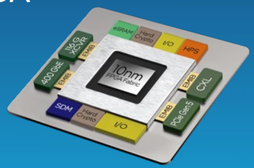
CPU



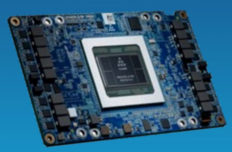
GPU



FPGA



Other Accelerators



Developer Challenges: Multiple Architectures, Vendors, and Programming Models

oneAPI

Open, Standards-based, Multiarchitecture Programming

1. Evans Data Global Development Survey Report 22.1, June 2022

oneAPI Industry Initiative

Break the Chains of Proprietary Lock-in

Freedom to Make Your Best Choice

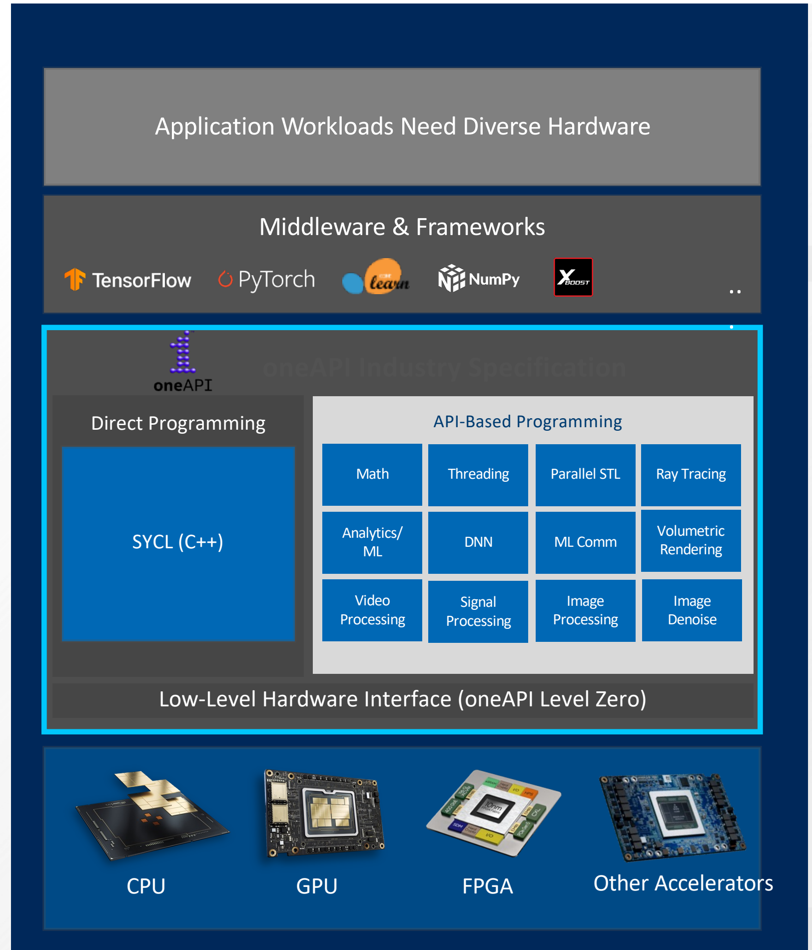
- C++ programming model for multiple architectures and vendors
- Cross-architecture code reuse for freedom from vendor lock-in

Realize all the Hardware Value

- Performance across CPU, GPUs, FPGAs, and other accelerators
- Expose and exploit cutting-edge features of the latest hardware

Develop & Deploy Software with Peace of Mind

- Open industry standards provide a safe, clear path to the future
- Interoperable with familiar languages and programming models including Fortran, Python, OpenMP, and MPI
- Powerful libraries for acceleration of domain-specific functions



oneAPI

The productive, smart path to freedom for accelerated computing from the economic and technical burdens of proprietary programming models

oneAPI Industry Momentum

End Users

National Labs

ISVs & OSVs

OEMs & SIs

Universities & Research Institutes

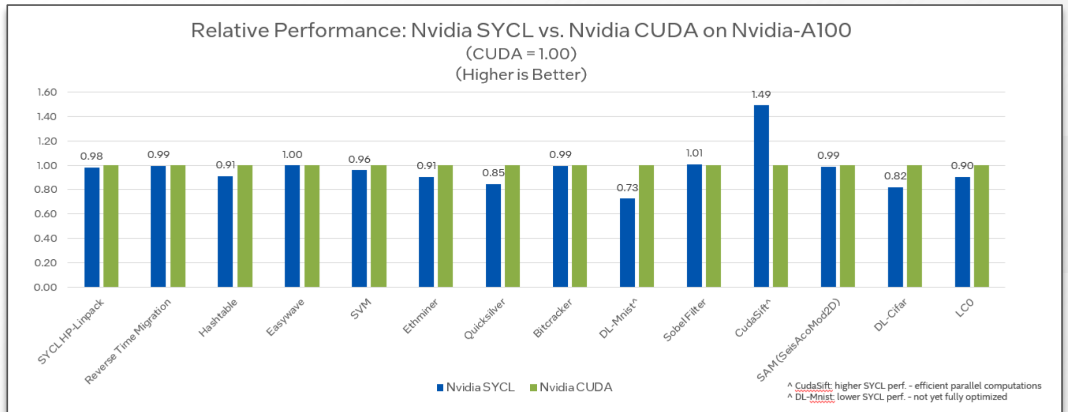
CSPs & Frameworks

These organizations support the oneAPI initiative for a single, unified programming model for cross-architecture development.
It does not indicate any agreement to purchase or use of Intel's products. *Other names and brands may be claimed as the property of others.

Accelerating Choice with SYCL

Khronos Group Standard

- Open, standards-based
- Multiarchitecture performance
- Freedom from vendor lock-in
- Comparable performance to native CUDA on Nvidia GPUs
- Extension of widely used C++ language
- Speed code migration via open source [SYCLomatic]() or Intel® DPC++ Compatibility Tool



Architectures

Intel | Nvidia | AMD CPU/GPU | RISC-V | ARM Mali | PowerVR | Xilinx

Testing Date: Performance results are **based on testing by Intel as of April 15, 2023** and may not reflect all publicly available updates.

Configuration Details and Workload Setup: Intel® Xeon® Platinum 8360Y CPU @ 2.4GHz, 2 socket, Hyper Thread On, Turbo On, 256GB Hynix DDR4-3200, ucode 0xd000363. GPU: Nvidia A100 PCIe 80GB GPU memory. Software: SYCL open source/CLANG 17.0.0, CUDA SDK 12.0 with NVIDIA-NVCC 12.0.76, cuMath 12.0, cuDNN 12.0, Ubuntu 22.04.1. SYCL open source/CLANG compiler switches: -fscycl-targets=nvptx64-nvidia-cuda, NVIDIA NVCC compiler switches: -O3 –gencode arch=compute_80, code=sm_80. Represented workloads with Intel optimizations.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See configuration disclosure for details. No product or component can be absolutely secure.

Performance varies by use, configuration, and other factors. Learn more at www.Intel.com/PerformanceIndex. Your costs and results may vary.

Intel® oneAPI Toolkits

A complete set of proven developer tools expanded from CPU to Accelerators

oneAPI

Intel® oneAPI Base Toolkit

A core set of high-performance libraries and tools for building C++, SYCL and Python applications

Add-on Domain-specific Toolkits

Intel® oneAPI Tools for HPC
Deliver fast Fortran, OpenMP & MPI applications that scale

Intel® oneAPI Tools for IoT
Build efficient, reliable solutions that run at network's edge

Intel® oneAPI Rendering Toolkit
Create performant, high-fidelity visualization applications

Toolkits powered by oneAPI

Intel® AI Analytics Toolkit
Accelerate machine learning & data science pipelines end-to-end with optimized DL frameworks & high-performing Python libraries

OpenVINO

Intel® Distribution of OpenVINO™ Toolkit
Deploy high performance inference & applications from edge to cloud

Latest version available 2022.2

Intel® AI Analytics Toolkit

Accelerate end-to-end AI and data analytics pipelines with libraries optimized for Intel® architectures

Who needs this product?

Data scientists, AI researchers, ML and DL developers,
AI application developers

Top Features/Benefits

- Deep learning performance for training and inference with Intel optimized DL frameworks and tools
- Drop-in acceleration for data analytics and machine learning workflows with compute-intensive Python packages

Intel® AI Analytics Toolkit

Deep Learning
- Intel® Optimization for TensorFlow
- Intel® Optimization for PyTorch
- Intel® Neural Compressor
- Model Zoo for Intel® Architecture

Machine Learning
- Intel® Extension for Scikit-learn
- Intel-optimized XGBoost

Data Analytics
- Intel® Distribution of Modin
- OmniSci Backend

Intel-optimized Python
- NumPy
- SciPy
- Numba
- Pandas
- Data Parallel Python

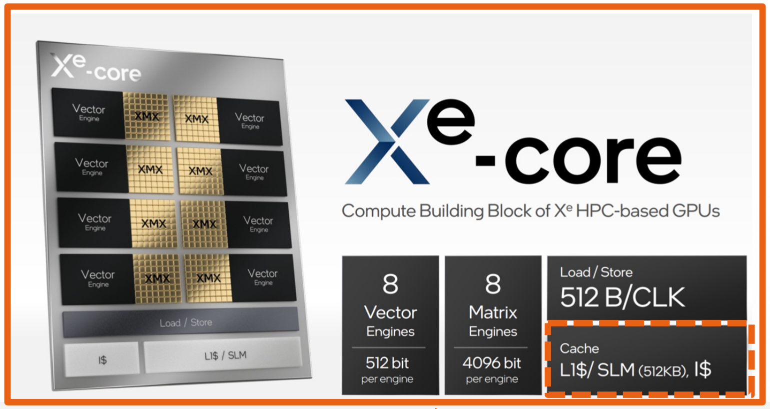
CPU GPU

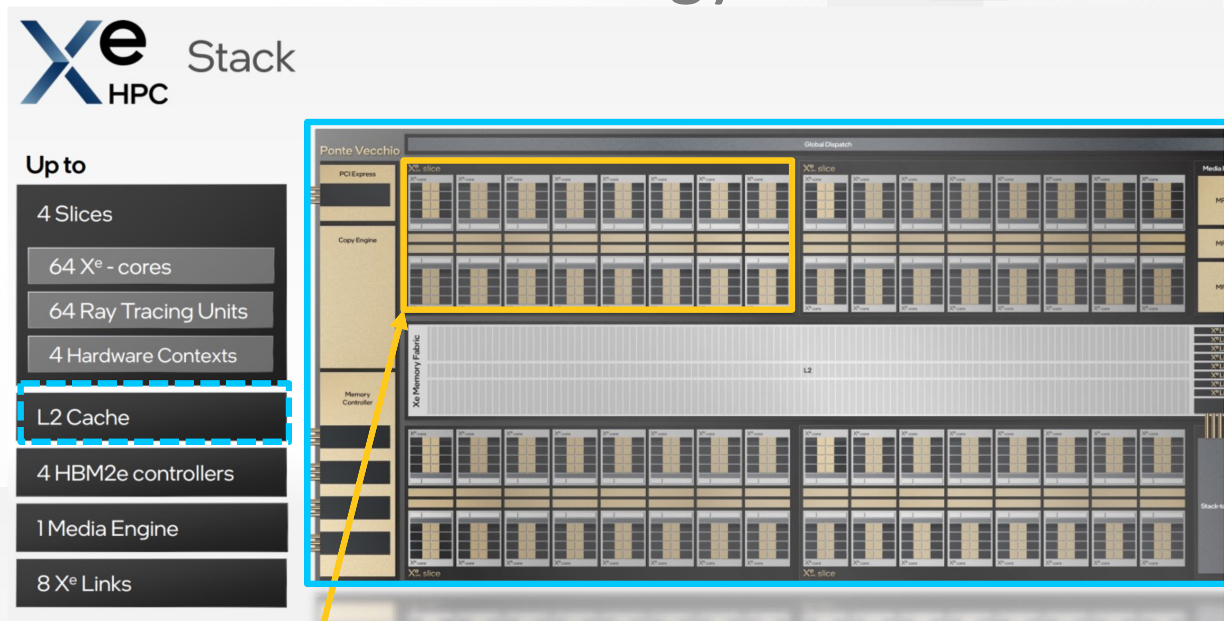
Hardware support varies by individual tool. Architecture support will be expanded over time.

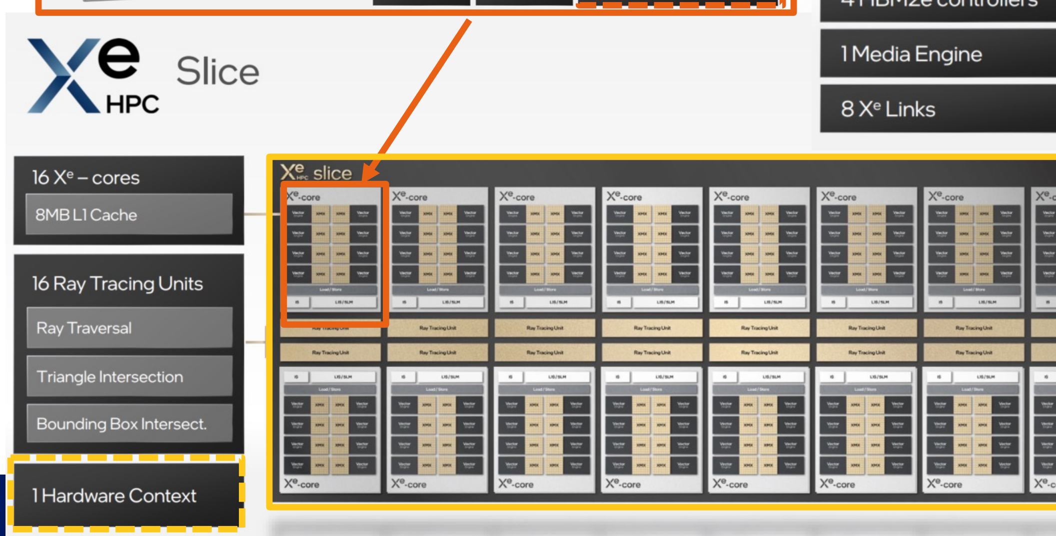
Get the Toolkit HERE or via these locations

- Intel Installer
- Docker
- Apt, Yum
- Conda
- Intel® DevCloud

Intel Data Center GPU Architecture Terminology







Old Term	New Intel Term	Generic Term	New Abbreviation
Execution Unit (EU)	Xe Vector Engine	Vector Engine	XVE
Systolic/"DPAS part of EU"	Xe Matrix eXtension	Matrix Engine	XMX
Subslice (SS) or Dual Subslice (DSS)	Xe-core	NA	XC
Slice	Render Slice / Compute Slice	Slice	SLC
Tile	Stack	Stack	STK

Intel® Data Center GPU Flex Series

Intel® Data Center GPU Max Series

- Up to 408 MB of L2 Cache
- AI-Boosting Intel® X^e Matrix Extensions (XMX)

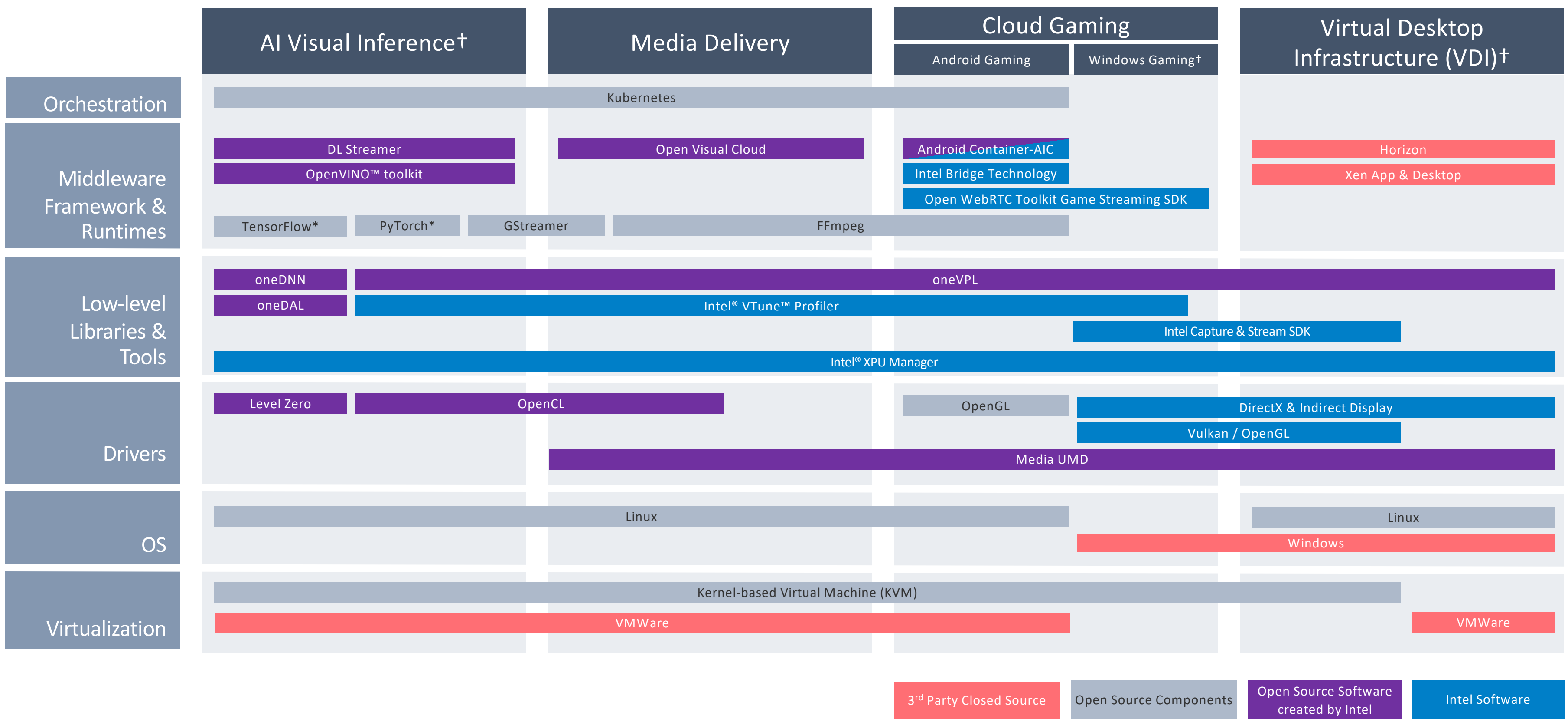
Intel Data Center GPU Max Series Products & Form Factor

	Max 1550 GPU (600W OAM)	Max 1350 GPU (450W OAM)	Max 1100 GPU (300W PCIe)
Architecture	X^e HPC		
X^e Cores	128	112	56
Memory	HBM2E 128 GB	HBM2E 96 GB	HBM2E 48 GB
Cache	L1 64 MB L2 408 MB	L1 48 MB L2 216 MB	L1 28 MB L2 108 MB
Max TDP	600W	450W	300W
Form Factor	OAM		PCIe AIC
Host Interconnect	PCIe Gen5		
Physical Ports	X^e Link 53 GB/s 16 ports		X^e Link 53 GB/s 6 ports

Xe configurations

Architecture	X^e-LP (TGL)	X^e-HPG (Arc A770)	X^e-HPG (Data Center GPU Flex 170)	X^e-HPC (Ponte Vecchio 1 Stack)
Slice count	1	8	4	4
XC (DSS/SS) count	6	32	16	64
XVE (EU) / XC	16	16	16	8
XVE count	96	512	256	512
Threads / XVE	7	8	8	8
Thread count	672	4096	4096	4096
FLOPs / clk - single precision, MAD	1536	8192	8192	16384
FLOPs / clk - double precision, MAD	NA	NA	NA	16384
FLOPs / clk - FP16 DP4AS	NA	65536	65536	262144
GTI bandwidth bytes / unslice-clk	r:128, w:128	r:512, w:512	r:256, w:256	r:1024, w:1024
LL cache size	3.84MB	16MB	8MB	up to 204MB
SLM size	$6 \times 128KB$	$32 \times 128KB$	$16 \times 128KB$	$64 \times 128KB$
FMAD, SP (ops / XVE / clk)	8	8	8	16
SQRT, SP (ops / XVE / clk)	2	2	2	4

Software Stack for Intel® Data Center Flex Series GPU



Note: oneDNN is the oneAPI Deep Neural Network Library. oneDAL is oneAPI Data Analytics Library. oneVPL is the oneAPI Video Processing Library. oneVPL, oneDNN, oneDAL, and Intel VTune Profiler are in the Intel® oneAPI Base Toolkit (individual tools can be downloaded separately). Intel-optimized TensorFlow & PyTorch are in Intel® AI Analytics Toolkit. †Reflects capabilities of Intel Data Center GPU Flex Series that will be available when product is fully mature.

Intel ® XPU Manager Product Suite

- A free and open -source suite of solutions built on top of the oneAPI Level Zero interface for monitoring and managing Intel data center XPUs.
- Intel XPU System Management Interface (SMI)
  - A command line utility for local XPU management
- Intel XPU Manager
  - A full -fledged solution with a daemon for aggregate telemetry collection, RESTful APIs for remote XPU management, and a local library for 3rd party solutions integration, and more.

https://github.com/intel/xpumanager

Environment Setup

Intel® Developer Cloud

a service platform for developing and running workloads in Intel®-optimized deployment environments with the latest Intel® processors

- Landing page :
  - https://www.intel.com/content/www/us/en/developer/tools/devcloud/services.html
- **Please follow the instructions to get started : http://tinyurl.com/ReadmeIDC**
  - https://github.com/bjodom/idc

TensorFlow and PyTorch Environments

- **TensorFlow Environment**

  Activate the prepared tensorflow env
  - $conda activate tensorflow_xpu

  Launch an Interactive session to pvc node
  - $srun -p pvc-shared --pty bash
  - $source /opt/intel/oneapi/setvars.sh

- **PyTorch Environment**

  Activate the prepared pytorch env
  - $conda activate pytorch_xpu

  Launch an Interactive session to pvc node
  - $srun -p pvc-shared --pty bash
  - $source /opt/intel/oneapi/setvars.sh

  Pip install necessary python packages
  - $python -m pip install oneccl_bind_pt -f https://developer.intel.com/ipex-whl-stable-xpu
  - $pip install torchvision –no-deps
  - $pip install pillow –no-deps

- $ lspci | grep -i display

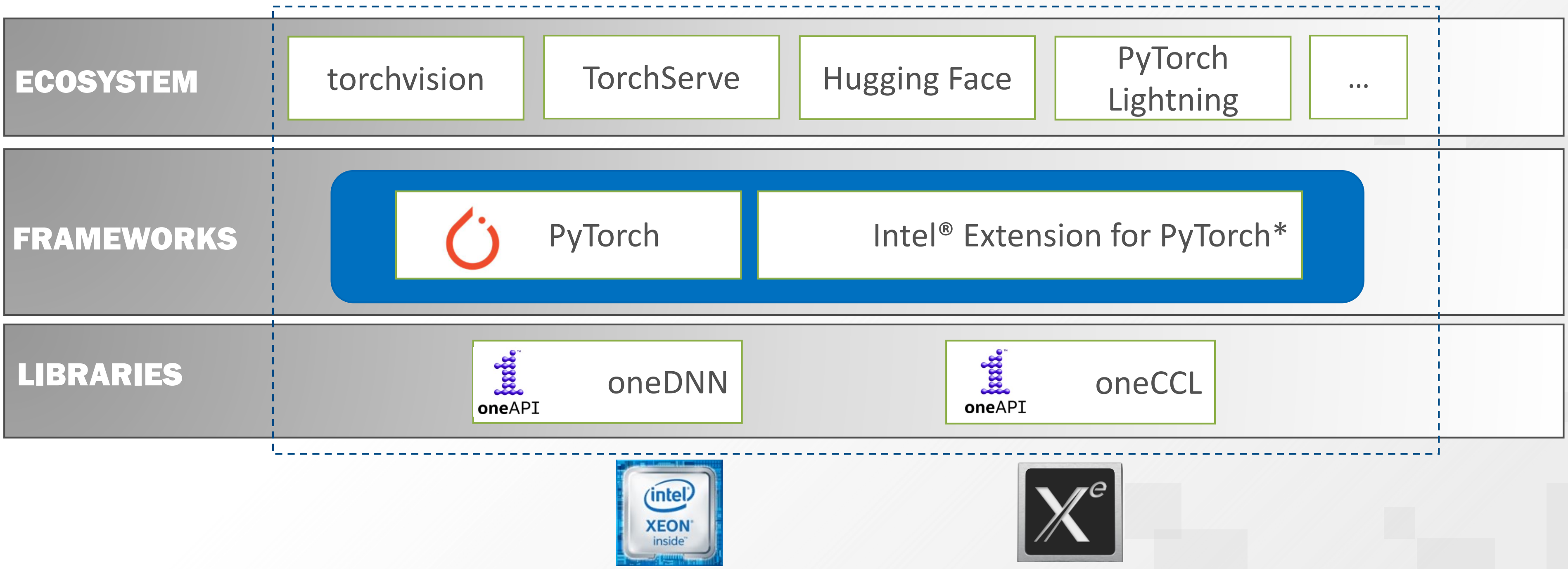
```
u105946@idc-beta-batch-pvc-node-10:~$ lspci | grep -i display
29:00.0 Display controller: Intel Corporation Device 0bda (rev 2f)
3a:00.0 Display controller: Intel Corporation Device 0bda (rev 2f)
9a:00.0 Display controller: Intel Corporation Device 0bda (rev 2f)
ca:00.0 Display controller: Intel Corporation Device 0bda (rev 2f)
```

- Validation for both TensorFlow and PyTorch
  - $wget https://raw.githubusercontent.com/oneapi-src/oneAPI-samples/master/AI-and-Analytics/version_check.py
  - $python version_check.py

Nodes and GPU cards	How to login the node	How to use one of 4 GPU card	User Assigned Index
Node 9. cards 0-3	srun -p pvc-shared -w idc-beta-batch-pvc-node-9 --pty /bin/bash	ZE_AFFINITY_MASK=0	0
		ZE_AFFINITY_MASK=1	1
		ZE_AFFINITY_MASK=2	2
		ZE_AFFINITY_MASK=3	3
Node 10. cards 0-3	srun -p pvc-shared -w idc-beta-batch-pvc-node-10 --pty /bin/bash	ZE_AFFINITY_MASK=0	4
		ZE_AFFINITY_MASK=1	5
		ZE_AFFINITY_MASK=2	6
		ZE_AFFINITY_MASK=3	7
Node 11. cards 0-3	srun -p pvc-shared -w idc-beta-batch-pvc-node-11 --pty /bin/bash	ZE_AFFINITY_MASK=0	8
		ZE_AFFINITY_MASK=1	9
		ZE_AFFINITY_MASK=2	10
		ZE_AFFINITY_MASK=3	11
Node 12. cards 0-3	srun -p pvc-shared -w idc-beta-batch-pvc-node-12 --pty /bin/bash	ZE_AFFINITY_MASK=0	12
		ZE_AFFINITY_MASK=1	13
		ZE_AFFINITY_MASK=2	14
		ZE_AFFINITY_MASK=3	15

Intel Optimizations for PyTorch on XPU

Intel® Optimization for PyTorch



Other names and brands may be claimed as the property of others

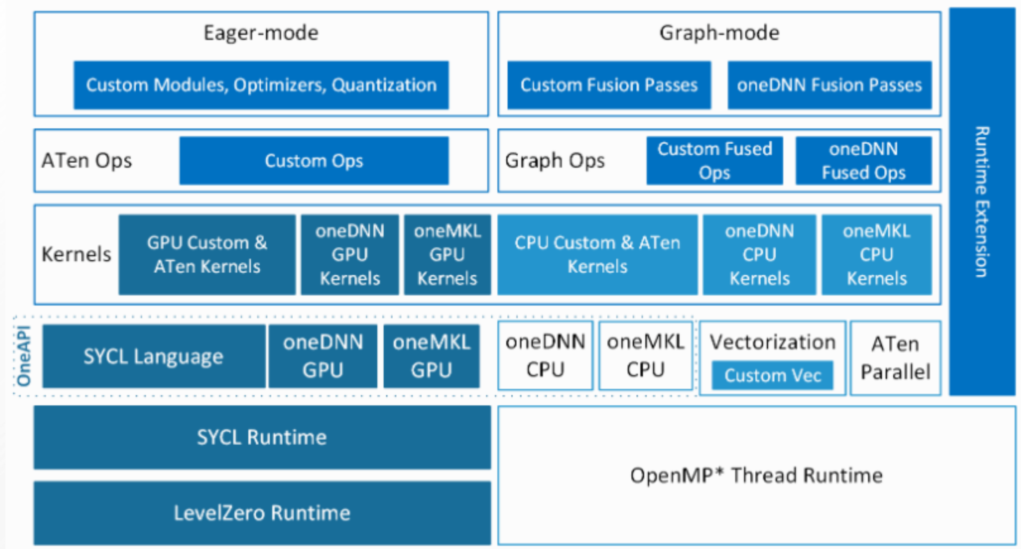
Major Optimization Methodologies

- Enabled functionality and performance optimizations on Intel GPUs.
- Additional performance boost and early adoption of aggressive optimizations through Intel® Extension for PyTorch*

Operator Optimization	Graph Optimization	Runtime Configuration
• Vectorization • Parallelization • Memory Layout • Low Precision	• Operator fusion • Constant folding	• Customize execution

Overview of Intel® Extension for PyTorch*

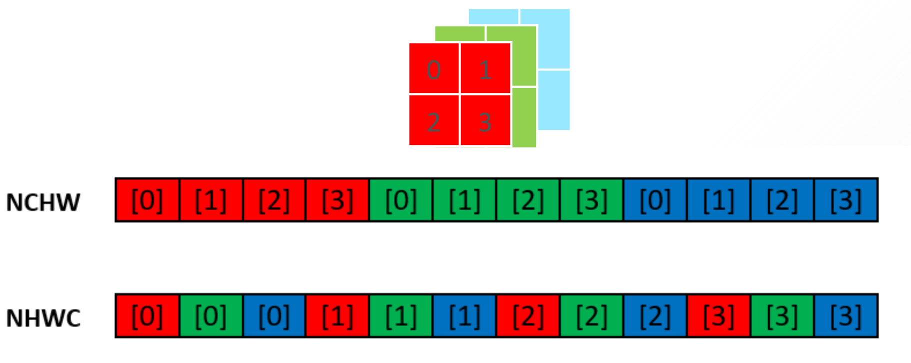
- Eager Mode (Default)
  - Focus on operators
  - For **development** and **debugging**
- Graph Mode (TorchScript)
  - Fuse operators and use constant folding to modify and merge the model structure to reduce time loss on invalid operations
  - For **deployment**
  - To switch to graph mode use TorchScript: torch.jit.trace() or torch.jit.script()
- oneDNN available in both PyTorch and IPEX
- AMX automatically enabled with oneDNN v2.6 and newer.
- Loaded dynamically in Python* script
- Dynamically linked in CPP executables



GitHub
https://github.com/intel/intel-extension-for-pytorch

Documentation (XPU)
https://intel.github.io/intel-extension-for-pytorch/xpu/latest/index.html

Memory Layout



- Used mainly in image workloads
- NCHW (PyTorch default)
  - *torch.contiguous_format*
- **NHWC (IPEX only default 1.13+)**
  - *torch.channels_last*
  - NHWC format yields higher performance on Intel® hardware

a. tensor creation

```
x = torch.empty(N, C, H, W, memory_format=torch.channels_last).to("xpu")
```

b. tensor conversion

```
## .contiguous() transforms NHWC noncontiguous to NHWC contiguous.
## .to() converts NCHW tensor to NHWC one, it is outplace.
x = x.to("xpu")
x = x.contiguous(memory_format=torch.channels_last)
x = x.to(memory_format=torch.channels_last)

## contiguous check
x.is_contiguous(memory_format=torch.channels_last)
```

c. model conversion

```
## NB: tensor.to() is an outplace operation
##   model.to() is inplace. It calls _apply() which is inplace.
model = model.to("xpu").to(memory_format=torch.channels_last)
input = input.to("xpu").to(memory_format=torch.channels_last)
```

PyTorch to IPEX – Getting Started

GPU FP32 Inference Example

```
import torch
import torchvision.models as models
############# code changes ###############
import intel_extension_for_pytorch as ipex
############# code changes ###############

model = models.resnet50(weights='ResNet50_Weights.DEFAULT')
model.eval()
data = torch.rand(1, 3, 224, 224)

######## code changes #######
model = model.to("xpu")
data = data.to("xpu")
model = ipex.optimize(model)
######## code changes #######

with torch.no_grad():
  model(data)
```

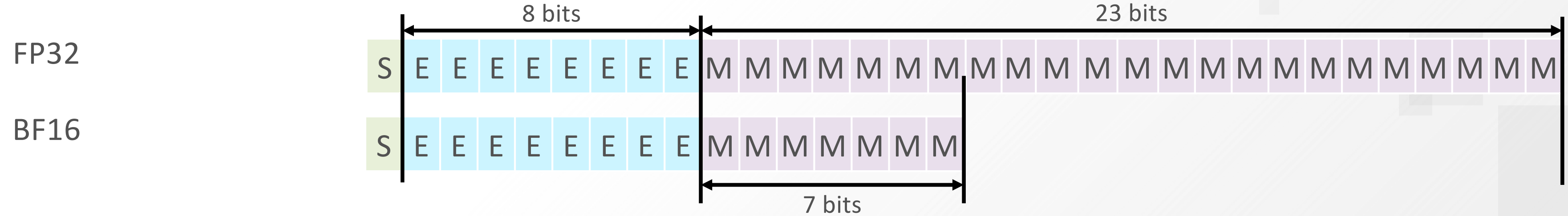
- Import Intel® Extension for PyTorch* package
- Set model and data to xpu (or ***'xpu:id'***)
  - ***model.to('xpu')***
  - ***data.to('xpu')***
- ***torch.xpu.optimize()*** is an alternative of ***ipex.optimize()*** in Intel® Extension for PyTorch*, to provide identical usage for XPU device only
- Code sample can be found here: https://intel.github.io/intel-extension-for-pytorch/xpu/latest/tutorials/examples.html

Enable Float32 using IPEX

- ipex.optimize function applies optimizations against the model object, as well as an optimizer object.
- In function, set `dtype` parameter to customize data type

```
model = model.to("xpu")
model, optimizer = ipex.optimize(model, optimizer=optimizer, dtype=torch.float32)
```

Low-precision Optimization – BF16



BF16 has the <u>same range</u> as FP32 but <u>less precision</u> due to 16 less mantissa bits. Running with 16 bits can give significant performance speedup.

https://www.intel.com/content/dam/develop/external/us/en/documents/bf16-hardware-numerics-definition-white-paper.pdf

Enable BFloat16 using IPEX

- Similar to Float32, the `optimize` function also works for BFloat16
- Set `dtype` parameter to `torch.bfloat16` instead

```
model = model.to("xpu")
model, optimizer = ipex.optimize(model, optimizer=optimizer, dtype=torch.bfloat16)
```

- Auto Mixed Precision (AMP) needed to run in BFloat16

```
with torch.xpu.amp.autocast(enabled=True, dtype=torch.bfloat16):
```

Training with Intel® Extension for PyTorch

```python
import torch
import torchvision
############# code changes ###############
import intel_extension_for_pytorch as ipex
############# code changes ###############

LR = 0.001
DOWNLOAD = True
DATA = 'datasets/cifar10/'

transform = torchvision.transforms.Compose([
    torchvision.transforms.Resize((224, 224)),
    torchvision.transforms.ToTensor(),
    torchvision.transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
train_dataset = torchvision.datasets.CIFAR10(
        root=DATA,
        train=True,
        transform=transform,
        download=DOWNLOAD,
)
train_loader = torch.utils.data.DataLoader(
        dataset=train_dataset,
        batch_size=128
)

model = torchvision.models.resnet50()
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr = LR, momentum=0.9)
model.train()
```

*The .to("xpu") is needed for GPU only
**Use torch.cpu.amp.autocast() for CPU
***Channels last format is automatic

```python
##################################### code changes ################################
model = model.to("xpu")
criterion = criterion.to("xpu")
model, optimizer = ipex.optimize(model, optimizer=optimizer, dtype=torch.bfloat16)
##################################### code changes ################################

for batch_idx, (data, target) in enumerate(train_loader):
    optimizer.zero_grad()
    ######################### code changes #########################
    data = data.to("xpu")
    target = target.to("xpu")
    with torch.xpu.amp.autocast(enabled=True, dtype=torch.bfloat16):
    ######################### code changes #########################
        output = model(data)
        loss = criterion(output, target)
    loss.backward()
    optimizer.step()
    print(batch_idx)
torch.save({
     'model_state_dict': model.state_dict(),
     'optimizer_state_dict': optimizer.state_dict(),
     }, 'checkpoint.pth')
```

Inference with Intel® Extension for PyTorch

*The .to("xpu") is needed for GPU only
**Use torch.cpu.amp.autocast() for CPU
***Channels last format is automatic

Resnet50

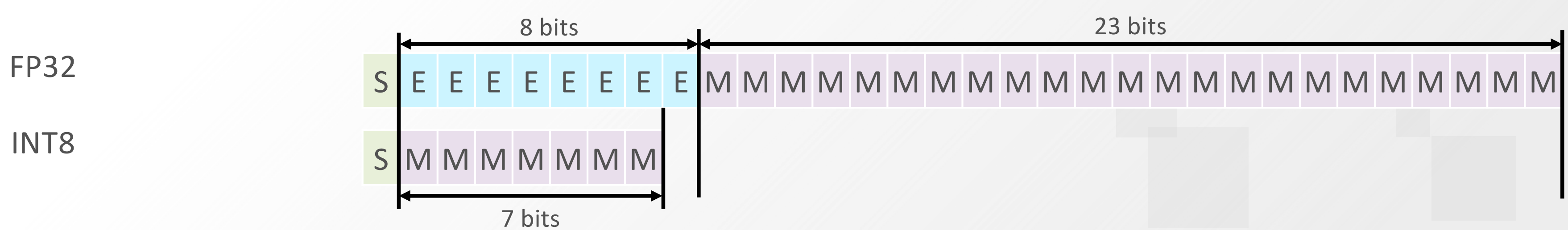
```
import torch
import torchvision.models as models
############# code changes ###############
import intel_extension_for_pytorch as ipex
############# code changes ###############

model = models.resnet50(pretrained=True)
model.eval()
data = torch.rand(1, 3, 224, 224)

#################### code changes #################
model = model.to("xpu")
data = data.to("xpu")
model = ipex.optimize(model, dtype=torch.bfloat16)
#################### code changes #################

with torch.no_grad():
    d = torch.rand(1, 3, 224, 224)
    ############################# code changes #####################
    d = d.to("xpu")
    with torch.xpu.amp.autocast(enabled=True, dtype=torch.bfloat16):
    ############################# code changes #####################
        model = torch.jit.trace(model, d)
        model = torch.jit.freeze(model)
        model(data)
```

BERT

```
import torch
from transformers import BertModel
############# code changes ###############
import intel_extension_for_pytorch as ipex
############# code changes ###############

model = BertModel.from_pretrained(args.model_name)
model.eval()

vocab_size = model.config.vocab_size
batch_size = 1
seq_length = 512
data = torch.randint(vocab_size, size=[batch_size, seq_length])

#################### code changes #################
model = model.to("xpu")
data = data.to("xpu")
model = ipex.optimize(model, dtype=torch.bfloat16)
#################### code changes #################

with torch.no_grad():
    d = torch.randint(vocab_size, size=[batch_size, seq_length])
    ############################# code changes #####################
    d = d.to("xpu")
    with torch.xpu.amp.autocast(enabled=True, dtype=torch.bfloat16):
    ############################# code changes #####################
        model = torch.jit.trace(model, (d,), strict=False)
        model = torch.jit.freeze(model)

        model(data)
```

Low-precision Optimization – INT8

- What is Quantization?
  - An approximation method
  - The process of mapping values from a large set (e.g. continuous, FP64/FP32) to those with smaller set (e.g. countable, BF16, INT8)
- Why Quantization?
  - Significant performance increase with similar accuracy
- How to Quantize?
  - PyTorch quantization
  - **IPEX quantization** (with or **w/o INC integration**)
  - Inter Neural Compressor (INC)



NOTE: Some models are not traceable, and therefore cannot be statically quantized.

Static vs Dynamic Quantization

- Static (Preferred)
  - Quantizes weights and activations of model
  - Fuses activations into preceding layers
  - **Requires calibration dataset** to determine optimal quantization parameters for activations
  - Used when both memory bandwidth and compute savings are important
  - Only works on inputs with fixed sizes; typically used for CNNs
- Dynamic
  - Weights are quantized ahead of time, but activations are quantized during inference
  - Used when model execution time is dominated by memory bandwidth
  - Can work on inputs with variable sizes; typically used for LSTM and Transformer models with small batch size

Quantization Workflow and API

Static Quantization:

1. Import `intel_extension_for_pytorch` as `ipex`.
2. Import `prepare` and `convert` from `intel_extension_for_pytorch.quantization`.
3. Instantiate a config object from `torch.ao.quantization.QConfig` to save configuration data during calibration.
4. Prepare model for calibration.
5. Perform calibration against dataset.
6. Invoke `ipex.quantization.convert` function to apply the calibration configure object to the fp32 model object to get an INT8 model.
7. Save the INT8 model into a `pt` file.

```
import os
import torch
#################### code changes ####################
import intel_extension_for_pytorch as ipex
from intel_extension_for_pytorch.quantization import prepare, convert
######################################################

model = Model()
model.eval()
data = torch.rand(<shape>)

qconfig = ipex.quantization.default_static_qconfig
# Alternatively, define your own qconfig:
#from torch.ao.quantization import MinMaxObserver, PerChannelMinMaxObserver, QConfig
#qconfig = QConfig(activation=MinMaxObserver.with_args(qscheme=torch.per_tensor_affine, dtype=torch.quint8),
#        weight=PerChannelMinMaxObserver.with_args(dtype=torch.qint8, qscheme=torch.per_channel_symmetric))
prepared_model = prepare(model, qconfig, example_inputs=data, inplace=False)

for d in calibration_data_loader():
  prepared_model(d)

converted_model = convert(prepared_model)
with torch.no_grad():
  traced_model = torch.jit.trace(converted_model, data)
  traced_model = torch.jit.freeze(traced_model)

traced_model.save("quantized_model.pt")
```

Dynamic Quantization:

1. Import `intel_extension_for_pytorch` as `ipex`.
2. Import `prepare` and `convert` from `intel_extension_for_pytorch.quantization`.
3. Instantiate a config object from `torch.ao.quantization.QConfig` to save configuration data during calibration.
4. Prepare model for quantization.
5. Convert the model.
6. Run inference to perform dynamic quantization.
7. Save the INT8 model into a `pt` file.

```
import os
import torch
#################### code changes ####################
import intel_extension_for_pytorch as ipex
from intel_extension_for_pytorch.quantization import prepare, convert
######################################################

model = Model()
model.eval()
data = torch.rand(<shape>)

dynamic_qconfig = ipex.quantization.default_dynamic_qconfig
# Alternatively, define your own qconfig:
#from torch.ao.quantization import MinMaxObserver, PlaceholderObserver, QConfig
#qconfig = QConfig(
#        activation = PlaceholderObserver.with_args(dtype=torch.float, compute_dtype=torch.quint8),
#        weight = PerChannelMinMaxObserver.with_args(dtype=torch.qint8, qscheme=torch.per_channel_symmetric))
prepared_model = prepare(model, qconfig, example_inputs=data)

converted_model = convert(prepared_model)
with torch.no_grad():
  traced_model = torch.jit.trace(converted_model, data)
  traced_model = torch.jit.freeze(traced_model)

traced_model.save("quantized_model.pt")
```

Quantization Workflow and API (GPU)

Static Quantization:

```
import Intel®_extension_for_pytorch

# define the model
def MyModel(torch.nn.Module):
  …

# construct the model
model = MyModel(…)
model = model.to(‘xpu’)
model = torch.jit.trace(model, input)
model.qconfig = torch.quantization.QConfig(…)
model = torch.quantization.quantize_jit.prepare_jit(model, {‘’: qconfig}, True)

for images in calibration_data_loader():
   images = images.to(‘xpu’)
   model(images)
model = torch.quantization.quantize_jit.convert_jit(model, True)

# run the model
with torch.inference_mode():
   input = input. to(‘xpu’)
   output = model(input)
```

TorchScript and torch.compile()

- TorchScript
  - Converts PyTorch **model** into a graph for faster execution
  - torch.jit.trace() traces and records all operations in the computational graph; requires a sample input
  - torch.jit.script() parses the Python source code of the model and compiles the code into a graph; sample input not required
- torch.compile() – in BETA
  - Makes PyTorch **code** run faster by just-in-time (JIT)-compiling PyTorch code into optimized kernels

TorchScript

- A method to run PyTorch in graph mode
- Invoke script mode with `torch.jit.trace` (requires sample input) or `torch.jit.script`
- `torch.xpu.amp.autocast` can be used with `torch.jit.trace` to apply graph optimizations

```
model = torch.jit.trace(model, d)
model = torch.jit.freeze(model)
```

Hands-on Demo

- Examples: https://github.com/intel/intel-extension-for-pytorch/tree/xpu-master/examples/gpu
- Steps
  - *git clone https://github.com/intel/intel-extension-for-pytorch.git*
  - *cd intel-extension-for-pytorch*
  - *git checkout xpu-master* # must use this branch for GPUs
  - *cd examples/gpu* # contains inference and training samples
  - Navigate into inference or training
  - Create a jupyter notebook (.ipynb) file and open it
  - Run different cases by copy/pasting the code and note the runtime differences from the code changes. Be sure to call **torch.xpu.synchronize()** before measuring time. This waits for all kernels to finish before proceeding

Runtime Configuration for GPU

Launch Option	Default Value	Description
IPEX_VERBOSE	0	Verbose level in integer. Set to 1 to print verbose output for Intel® Extension for PyTorch* GPU customized kernel. Set to other value is not supported so far.
IPEX_SIMPLE_TRACE	OFF	Simple trace functionality. If set to ON, enable simple trace for all operators. Set to other value is not supported.
IPEX_TILE_AS_DEVICE	ON	Device partition. If set to OFF, tile partition will be disabled and map device to physical device. Set to other value is not supported.
IPEX_XPU_SYNC_MODE	OFF	Kernel Execution mode. If set to ON, use synchronized execution mode and perform blocking wait for the completion of submitted kernel. Set to other value is not supported.
IPEX_FP32_MATH_MODE	FP32	Floating-point math mode. Set to TF32 for using TF32 math mode, BF32 for using BF32 math mode. Set to other value is not supported. Refer to https://github.com/oneapi-src/oneDNN/tree/rfcs/rfcs/20210301-computation-datatype for the definition of TF32 and BF32 math mode.

OneDNN Verbose

- Generate oneDNN Verbose logs using guide and parser
- To enable verbosity, set environment variables:
  - export DNNL_VERBOSE=1
  - export DNNL_VERBOSE_TIMESTAMP=1
- Set a Python breakpoint RIGHT AFTER one iteration of training/inference

PyTorch to IPEX – Validating Output

```
onednn_verbose,info,oneDNN v3.1.0 (commit 928065bd62372d2824900a86c5438bba407c98d1)
onednn_verbose,info,cpu,runtime:threadpool,nthr:16
onednn_verbose,info,cpu,isa:Intel AVX-512 with Intel DL Boost
onednn_verbose,info,gpu,runtime:DPC++
onednn_verbose,info,gpu,engine,0,backend:Level Zero,name:Intel(R) Data Center GPU Flex 170,driver_version:1.3.25593,binary_kernels:enabled
onednn_verbose,info,gpu,engine,1,backend:Level Zero,name:Intel(R) Data Center GPU Flex 170,driver_version:1.3.25593,binary_kernels:enabled
onednn_verbose,info,experimental features are enabled
onednn_verbose,info,use batch_normalization stats one pass is enabled
onednn_verbose,info,prim_template:timestamp,operation,engine,primitive,implementation,prop_kind,memory_descriptors,attributes,auxiliary,problem_desc,exec_time
onednn_verbose,1688158533422.138916,exec,gpu:0,reorder,jit:ir,undef,src_bf16::blocked:abcd:f0 dst_bf16:p:blocked:AcdB8a2b:f0,,,64x3x7x7,0.0820312
onednn_verbose,1688158533439.231934,exec,gpu:0,convolution,jit:ir,forward_training,src_bf16::blocked:abcd:f0 wei_bf16:p:blocked:AcdB8a2b:f0 bia_bf16::blocked:a
onednn_verbose,1688158533652.840088,exec,gpu:0,eltwise,ocl:gen9:any,forward_training,data_bf16::blocked:aBcd16b:f0 diff_undef::undef::,attr-scratchpad:user ,al
onednn_verbose,1688158533861.677002,exec,gpu:0,pooling,ocl:ref,forward_training,src_bf16::blocked:aBcd16b:f0 dst_bf16::blocked:aBcd16b:f0 ws_s32::blocked:aBcd1
onednn_verbose,1688158533872.655029,exec,gpu:0,reorder,jit:ir,undef,src_bf16::blocked:abcd:f0 dst_bf16::blocked:ABcd8b8a2b:f0,,,64x64x1x1,0.187012
onednn_verbose,1688158533887.431885,exec,gpu:0,convolution,jit:ir,forward_training,src_bf16::blocked:aBcd16b:f0 wei_bf16::blocked:ABcd8b8a2b:f0 bia_bf16::block
onednn_verbose,1688158534103.305908,exec,gpu:0,eltwise,ocl:gen9:any,forward_training,data_bf16::blocked:aBcd16b:f0 diff_undef::undef::,attr-scratchpad:user ,al
onednn_verbose,1688158534113.702881,exec,gpu:0,reorder,jit:ir,undef,src_bf16::blocked:abcd:f0 dst_bf16::blocked:ABcd8b8a2b:f0,,,64x64x3x3,0.172119
onednn_verbose,1688158534137.868896,exec,gpu:0,convolution,jit:ir,forward_training,src_bf16::blocked:aBcd16b:f0 wei_bf16::blocked:ABcd8b8a2b:f0 bia_bf16::block
onednn_verbose,1688158534144.310059,exec,gpu:0,eltwise,ocl:gen9:any,forward_training,data_bf16::blocked:aBcd16b:f0 diff_undef::undef::,attr-scratchpad:user ,al
onednn_verbose,1688158534150.173096,exec,gpu:0,reorder,jit:ir,undef,src_bf16::blocked:abcd:f0 dst_bf16::blocked:ABcd8b8a2b:f0,,,256x64x1x1,0.0678711
onednn_verbose,1688158534166.833008,exec,gpu:0,convolution,jit:ir,forward_training,src_bf16::blocked:aBcd16b:f0 wei_bf16::blocked:ABcd8b8a2b:f0 bia_bf16::block
onednn_verbose,1688158534173.551025,exec,gpu:0,reorder,jit:ir,undef,src_bf16::blocked:abcd:f0 dst_bf16::blocked:ABcd8b8a2b:f0,,,256x64x1x1,0.0390625
onednn_verbose,1688158534173.620117,exec,gpu:0,convolution,jit:ir,forward_training,src_bf16::blocked:aBcd16b:f0 wei_bf16::blocked:ABcd8b8a2b:f0 bia_bf16::block
onednn_verbose,1688158534356.812988,exec,gpu:0,binary,ocl:ref:any,undef,src_bf16::blocked:aBcd16b:f0 src_bf16::blocked:aBcd16b:f0 dst_bf16::blocked:aBcd16b:f0,
onednn_verbose,1688158534569.726074,exec,gpu:0,eltwise,ocl:gen9:any,forward_training,data_bf16::blocked:aBcd16b:f0 diff_undef::undef::,attr-scratchpad:user ,al
onednn_verbose,1688158534579.447021,exec,gpu:0,reorder,jit:ir,undef,src_bf16::blocked:abcd:f0 dst_bf16::blocked:ABcd8b8a2b:f0,,,64x256x1x1,0.167969
onednn_verbose,1688158534591.875977,exec,gpu:0,convolution,jit:ir,forward_training,src_bf16::blocked:aBcd16b:f0 wei_bf16::blocked:ABcd8b8a2b:f0 bia_bf16::block
onednn_verbose,1688158534598.406982,exec,gpu:0,eltwise,ocl:gen9:any,forward_training,data_bf16::blocked:aBcd16b:f0 diff_undef::undef::,attr-scratchpad:user ,al
```

Getting Started with Model Zoo

- Model Zoo for Intel® Architecture: contains Intel optimizations for running deep learning workloads on Intel® Xeon® Scalable processors
- GitHub: https://github.com/IntelAI/models

ImageNet Data Prep

```python
#!/usr/bin/env python
# encoding: utf-8

import os
import tarfile

PATH = "/gpfs/jlse-fs0/projects/intel_anl_shared/imagenet/train/"

def main(argv):
    directory = os.fsencode(PATH)
    for file in os.listdir(directory):
        filename = os.fsdecode(file)

        if filename.endswith(".tar"):
            print("Processing %s" %filename)
            foldername = filename.split(".")[0]
            folderpath = PATH + foldername
            if not os.path.exists(folderpath):
                os.makedirs(folderpath)

            # Extract into this newly created folder
            filepath = PATH + filename
            tarfile_obj = tarfile.open(filepath,"r")
            tarfile_obj.extractall(folderpath)
            tarfile_obj.close()

if __name__ == "__main__":
    import sys
    sys.exit(main(sys.argv))
```

- Download ImageNet2012 training and validation sets
- Extract files from each tar file
- Place files into respective folder
  - Run the valprep.sh script to organize validation data
  - See script on left to prepare training data
- ***NOTE: no need to run scripts. Pre-processed dataset located here:**
  - /gpfs/jlse-fs0/projects/intel_anl_shared/imagenet

Model Zoo w/IPEX

- [Resnet50v1_5 Inference README]
  - $git clone https://github.com/IntelAI/models.git
- Commands to run Resnet50v1.5 inference (with Dummy dataset)
  - $cd models
  - $mkdir output_resnet50v1_5_inference
  - export OUTPUT_DIR=/home/uXXXXXX/models/output _resnet50v1_5_inference
  - export PRECISION="int8"
  - ./quickstart/image_recognition/pytorch/resnet50v1_5/inference/gpu/inference_block_format.sh
- Alternatively, run this script with input argument "inference"
  - Copy modelzoo_resnet.sh into MODEL_DIR
  - source modelzoo_resnet.sh inference
  - *NOTE: this script will not be maintained

Model Zoo w/IPEX Output

- Inference (Results with ImageNet validation)

```
resnet50 int8 inference block nchw
oneccl_bindings_for_pytorch not available!
Use XPU: 0
=> using pre-trained model 'resnet50'
model to xpu
doing int8 jit calibration
doing int8 inference
Test: [ 1/30]   Time 25.014 (25.014)    Loss 5.2118e-01 (5.2118e-01)    Acc@1  87.30 ( 87.30)   Acc@5  96.68 ( 96.68)
Test: [ 2/30]   Time  0.088 (12.551)    Loss 1.0375e+00 (7.7932e-01)    Acc@1  74.61 ( 80.96)   Acc@5  92.58 ( 94.63)
Test: [ 3/30]   Time  0.088 ( 8.397)    Loss 4.6560e-01 (6.7475e-01)    Acc@1  87.11 ( 83.01)   Acc@5  97.46 ( 95.57)
Test: [ 4/30]   Time  0.089 ( 6.320)    Loss 6.7351e-01 (6.7444e-01)    Acc@1  84.77 ( 83.45)   Acc@5  95.12 ( 95.46)
Test: [ 5/30]   Time  0.088 ( 5.073)    Loss 5.9476e-01 (6.5850e-01)    Acc@1  83.98 ( 83.55)   Acc@5  96.48 ( 95.66)
Test: [ 6/30]   Time  0.090 ( 4.243)    Loss 8.3799e-01 (6.8842e-01)    Acc@1  76.46 ( 82.37)   Acc@5  96.00 ( 95.72)
Test: [ 7/30]   Time  0.088 ( 3.649)    Loss 6.5648e-01 (6.8385e-01)    Acc@1  82.71 ( 82.42)   Acc@5  96.58 ( 95.84)
Test: [ 8/30]   Time  0.088 ( 3.204)    Loss 6.5710e-01 (6.8051e-01)    Acc@1  79.59 ( 82.07)   Acc@5  97.36 ( 96.03)
Test: [ 9/30]   Time  0.090 ( 2.858)    Loss 8.2756e-01 (6.9685e-01)    Acc@1  79.69 ( 81.80)   Acc@5  95.21 ( 95.94)
Test: [10/30]   Time  0.091 ( 2.581)    Loss 6.4543e-01 (6.9171e-01)    Acc@1  83.98 ( 82.02)   Acc@5  96.39 ( 95.99)
Quantization Evalution performance: batch size:1024, throughput:11519.11 image/sec, Acc@1:82.02, Acc@5:95.99
```

Intel Optimizations for TensorFlow on XPU

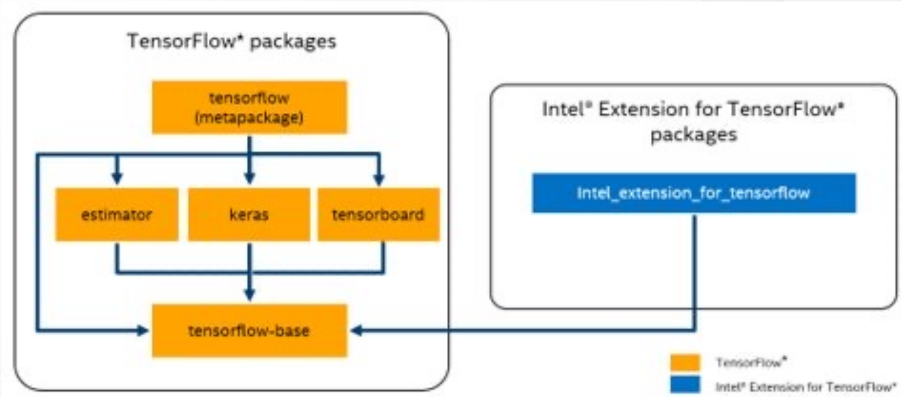
Intel® Extension for TensorFlow*

- Intel® Extension for TensorFlow* is a heterogeneous, high performance deep learning extension plugin based on TensorFlow PluggableDevice interface to bring Intel XPU(GPU, CPU, etc) devices into TensorFlow .

- Good performance using default ITEX setting with no code change

- More performance optimizations with minor code change using simple frontend Python API

- GitHub: https://github.com/intel/intel-extension-for-tensorflow



Intel® Extension for TensorFlow* PyPI packages and dependencies

Intel® Extension for TensorFlow* - GPU Features

- **Features:**
  - Auto Mixed Precision (AMP)
    - support of AMP with BFloat16 and Float16 operations
  - Channels Last
    - support of channels_last (NHWC) memory format
  - DPC++ Extension
    - mechanism to create operators with custom DPC++ kernels running on the XPU device
  - Optimized Fusion
    - support of SGD/AdamW fusion for both FP32 and BF16 precision
    - a set of fusion patterns for inference

Intel® Extension for TensorFlow* - Optimization Methods

- Operator optimizations
- Memory/data layout optimizations
- Graph optimizations
- Mixed Precision

HOWTO: Intel® Extension for TensorFlow*(GPU)

- No code changes, the default backend will be Intel GPU after installing
  - **intel-extension-for-tensorflow[gpu]**

Or

```
import intel_extension_for_tensorflow as itex

#CPU, GPU or AUTO
backend = "GPU"
itex.set_backend(backend)
```

TensorFlow to ITEX – Getting Started

Setting Backend Example

```
import numpy as np
import tensorflow as tf
import intel_extension_for_tensorflow as itex

print(itex.__version__)

backend = "GPU"  #CPU, GPU or AUTO
itex.set_backend(backend)

# Conv + ReLU activation + Bias
N = 1
num_channel = 3
input_width, input_height = (5, 5)
filter_width, filter_height = (2, 2)

x = np.random.rand(N, input_width, input_height, num_channel).astype(np.float32)
weight = np.random.rand(filter_width, filter_height, num_channel, num_channel).astype(np.float32)
bias = np.random.rand(num_channel).astype(np.float32)

conv = tf.nn.conv2d(x, weight, strides=[1, 1, 1, 1], padding='SAME')
activation = tf.nn.relu(conv)
result = tf.nn.bias_add(activation, bias)

print(result)
```

- The backend can be set to CPU, GPU or AUTO using the set_backend API to run the workload on desired hardware.

- Code sample can be found here: https://github.com/intel/intel-extension-for-tensorflow/blob/main/examples/quick_example.md

ITEX – Verbose mode

- The output shows the oneDNN verbose output process running on GPU.
- `Export ONEDNN_VERBOSE=1`

```
onednn_verbose,info,oneDNN v3.1 (commit e008c47c7f2e839ff64c206a21c82059a227717c)
onednn_verbose,info,cpu,runtime:DPC++e
onednn_verbose,info,cpu,isa:Intel 64
onednn_verbose,info,gpu,runtime:DPC++
onednn_verbose,info,cpu,engine,0,backend:OpenCL,name:Genuine Intel(R) CPU $0000%@,driver_version:2021.13.11
onednn_verbose,info,gpu,engine,0,backend:Level Zero,name:Intel(R) Graphics [0x020a],driver_version:1.1.20495
onednn_verbose,info,gpu,engine,1,backend:Level Zero,name:Intel(R) Graphics [0x020a],driver_version:1.1.20495
onednn_verbose,info,prim_template:operation,engine,primitive,implementation,prop_kind,memory_descriptors,attributes,auxiliary,p
roblem_desc,exec_time
onednn_verbose,info,gpu,binary_kernels:enabled
onednn_verbose,exec,gpu:18446744073709551615,reorder,ocl:ref:any,undef,src_f32::blocked:cdba:f0
dst_f32:p:blocked:Acdb16a:f0,,,3x3x2x2,1.42993
onednn_verbose,exec,gpu:18446744073709551615,convolution,ocl:gen9:blocked,forward_training,src_f32::blocked:acdb:f0
wei_f32:p:blocked:Acdb16a:f0 bia_undef::undef::f0 dst_f32::blocked:acdb:f0,attr-scratchpad:user
,alg:convolution_direct,mb1_ic3oc3_ih5oh5kh2sh1dh0ph0_iw5ow5kw2sw1dw0pw0,0.352051
onednn_verbose,exec,gpu:18446744073709551615,eltwise,ocl:ref:any,forward_training,data_f32::blocked:abcd:f0
diff_undef::undef::f0,attr-scratchpad:user ,alg:eltwise_relu alpha:0 beta:0,1x5x5x3,0.297852
```

Mixed precision (BF16 & FP16)

- Use Keras mixed precision API in **Stock TensorFlow**
  - ITEX is compatible

- mixed_precision.set_global_policy('mixed_float16')

OR

- mixed_precision.set_global_policy('mixed_bfloat16')

Use Advanced Auto Mixed Precision provided by ITEX for better performance

- 2 modes of activation
- Can be run from frozen graph
- Support for fused operations

	FP16	**BF16**
Intel CPU	No	Yes
Intel GPU	Yes	Yes

ITEX – Advanced Auto Mixed Precision : Python API

- ITEX advanced AMP can be set from code:

```
import intel_extension_for_tensorflow as itex

auto_mixed_precision_options = itex.AutoMixedPrecisionOptions()
auto_mixed_precision_options.data_type = itex.BFLOAT16 (or itex.FLOAT16)

graph_options = itex.GraphOptions()
graph_options.auto_mixed_precision_options=auto_mixed_precision_options
graph_options.auto_mixed_precision = itex.ON

config = itex.ConfigProto(graph_options=graph_options)
itex.set_backend("gpu", config) [in ITEX v1.0.0 and ITEX v1.1.0]
    (NOTE) --> itex.set_config(config) [latest master branch]
```

ITEX – Advanced Auto Mixed Precision : Environment Variable

- ITEX advanced AMP can also be set via env variables:

export ITEX_AUTO_MIXED_PRECISION=1
export ITEX_AUTO_MIXED_PRECISION_DATA_TYPE="BFLOAT16" (or "FLOAT16“)

```
(anl_itex)ac.srikanthan@arcticus06:/gpfs/jlse-fs0/users/ac.srikanthan> export ITEX_AUTO_MIXED_PRECISION_ALLOWLIST_ADD="AvgPool3D,AvgPool"
(anl_itex)ac.srikanthan@arcticus06:/gpfs/jlse-fs0/users/ac.srikanthan> export ITEX_AUTO_MIXED_PRECISION_INFERLIST_REMOVE="AvgPool3D,AvgPool"
```

```
onednn_verbose,exec,gpu:0,convolution,jit:ir,forward_training,src_bf16::blocked:ABcd32a16b:f0 wei_bf16::blocked:ABcd8b8a2b:f0 bia_bf16::blocked:a:f0 dst_bf16::blocked:ABcd32a16b:f0,attr-scr
atchpad:user attr-post-ops:eltwise_relu ,alg:convolution_direct,mb128_ic384oc256_ih8oh8kh1sh1dh0ph0_iw8ow8kw3sw1dw0pw1,0.118896
onednn_verbose,exec,gpu:0,convolution,jit:ir,forward_training,src_bf16::blocked:ABcd32a16b:f0 wei_bf16::blocked:ABcd8b8a2b:f0 bia_bf16::blocked:a:f0 dst_bf16::blocked:ABcd32a16b:f0,attr-scr
atchpad:user attr-post-ops:eltwise_relu ,alg:convolution_direct,mb128_ic384oc256_ih8oh8kh3sh1dh0ph1_iw8ow8kw1sw1dw0pw0,0.10791
onednn_verbose,exec,gpu:0,convolution,jit:ir,forward_training,src_bf16::blocked:ABcd32a16b:f0 wei_bf16::blocked:ABcd8b8a2b:f0 bia_bf16::blocked:a:f0 dst_bf16::blocked:ABcd32a16b:f0,attr-scr
atchpad:user attr-post-ops:eltwise_relu ,alg:convolution_direct,mb128_ic384oc448_ih8oh8kh3sh1dh0ph1_iw8ow8kw1sw1dw0pw0,0.161133
onednn_verbose,exec,gpu:0,convolution,jit:ir,forward_training,src_bf16::blocked:ABcd32a16b:f0 wei_bf16::blocked:ABcd8b8a2b:f0 bia_bf16::blocked:a:f0 dst_bf16::blocked:ABcd32a16b:f0,attr-scr
atchpad:user attr-post-ops:eltwise_relu ,alg:convolution_direct,mb128_ic1536oc256_ih8oh8kh1sh1dh0ph0_iw8ow8kw1sw1dw0pw0,0.150146
onednn_verbose,exec,gpu:0,convolution,jit:ir,forward_training,src_bf16::blocked:ABcd32a16b:f0 wei_bf16::blocked:ABcd8b8a2b:f0 bia_bf16::blocked:a:f0 dst_bf16::blocked:ABcd32a16b:f0,attr-scr
atchpad:user attr-post-ops:eltwise_relu ,alg:convolution_direct,mb128_ic448oc512_ih8oh8kh1sh1dh0ph0_iw8ow8kw3sw1dw0pw1,0.175049
onednn_verbose,exec,gpu:0,convolution,jit:ir,forward_training,src_bf16::blocked:ABcd32a16b:f0 wei_bf16::blocked:ABcd8b8a2b:f0 bia_bf16::blocked:a:f0 dst_bf16::blocked:ABcd32a16b:f0,attr-scr
atchpad:user attr-post-ops:eltwise_relu ,alg:convolution_direct,mb128_ic512oc256_ih8oh8kh1sh1dh0ph0_iw8ow8kw3sw1dw0pw1,0.11792
```

Enable BF 16 capabilities using ITEX

```
y NUMA node of platform XPU ID 1, defaulting to 0. Your kernel may not have been built with NUMA support.
2022-12-05 06:06:26.449994: I tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:272] Created TensorFlo
w device (/job:localhost/replica:0/task:0/device:XPU:0 with 0 MB memory) -> physical PluggableDevice (device: 0, name: XPU, pci
bus id: <undefined>)
2022-12-05 06:06:26.450341: I tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:272] Created TensorFlo
w device (/job:localhost/replica:0/task:0/device:XPU:1 with 0 MB memory) -> physical PluggableDevice (device: 1, name: XPU, pci
bus id: <undefined>)
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
Epoch 1/5
2022-12-05 06:06:52.317983: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugin optimizer for d
evice_type XPU is enabled.
2022-12-05 06:06:52.355039: I itex/core/graph/auto_mixed_precision/auto_mixed_precision.cc:109] Run advanced auto mixed precisio
n datatype BFLOAT16 on XPU
2022-12-05 06:06:52.379360: I itex/core/graph/auto_mixed_precision/auto_mixed_precision.cc:1723] Converted 16/146 nodes to bfloa
t16 precision using 2 cast(s) to bfloat16 (excluding Const and Variable casts)
2022-12-05 06:06:52.406383: I itex/core/graph/auto_mixed_precision/auto_mixed_precision.cc:109] Run advanced auto mixed precisio
n datatype BFLOAT16 on XPU
2022-12-05 06:06:52.411292: I itex/core/graph/auto_mixed_precision/auto_mixed_precision.cc:1723] Converted 0/171 nodes to bfloat
16 precision using 0 cast(s) to bfloat16 (excluding Const and Variable casts)
onednn_verbose,info,oneDNN v2.7.0 (commit 650085b2f3643aad05c629425983491d63b5c289)
onednn_verbose,info,cpu,runtime:DPC++,nthr:1
onednn_verbose,info,cpu,isa:Intel 64
onednn_verbose,info,gpu,runtime:DPC++
onednn_verbose,info,gpu,engine,0,backend:Level Zero,name:Intel(R) Graphics [0x020a],driver_version:1.3.23527
onednn_verbose,info,gpu,engine,1,backend:Level Zero,name:Intel(R) Graphics [0x020a],driver_version:1.3.23527
onednn_verbose,info,experimental features are enabled
onednn_verbose,info,use batch normalization stats one pass is enabled
onednn_verbose,info,prim_template:operation,engine,primitive,implementation,prop_kind,memory_descriptors,attributes,auxiliary,pr
oblem_desc,exec_time
onednn_verbose,info,gpu,binary_kernels:enabled
onednn_verbose,exec,gpu:0,matmul,jit:xe_hp:gemm:any,undef,src_bf16::blocked:ab:f0 wei_bf16::blocked:ab:f0 bia_bf16::blocked:ab:f
0_mask2 dst_bf16::blocked:ab:f0,attr-scratchpad:user attr-post-ops:eltwise_relu ,,8192x784:784x4096:8192x4096,9.68188
onednn_verbose,exec,gpu:0,matmul,jit:xe_hp:gemm:any,undef,src_bf16::blocked:ab:f0 wei_bf16::blocked:ab:f0 bia_bf16::blocked:ab:f
0_mask2 dst_bf16::blocked:ab:f0,attr-scratchpad:user attr-post-ops:eltwise_relu ,,8192x4096:4096x4096:8192x4096,112.995
onednn_verbose,exec,gpu:0,matmul,jit:xe_hp:gemm:any,undef,src_bf16::blocked:ab:f0 wei_bf16::blocked:ab:f0 bia_bf16::blocked:ab:f
0_mask2 dst_bf16::blocked:ab:f0,attr-scratchpad:user ,,8192x4096:4096x10:8192x10,25.5342
onednn_verbose,exec,gpu:0,softmax_v2,ref:any,forward_training,src_bf16::blocked:ab:f0 dst_bf16::blocked:ab:f0,attr-scratchpad:us
er ,alg:softmax_accurate axis:1,8192x10,1.14795
```

- `./infer_fp32_vs_amp.sh gpu bf16`
- The output of enabling auto mixed precision is shown.
- Code sample can be found here: https://github.com/intel/intel-extension-for-tensorflow/tree/main/examples/infer_inception_v4_amp

Customized ITEX operators

- Itex.ops.ItexLSTM
  - Has same semantic with tf.keras.layers.LSTM.
  - Based on available runtime / hardware, layer will choose ITEX or TF implementation to maximize performance.

- Other such operators are:
  - itex.ops.gelu
  - itex.ops.LayerNormalization
  - itex.ops.AdamWithWeightDecayOptimizer

For more details refer here

Getting Started with Model Zoo

- Model Zoo for Intel® Architecture: contains Intel optimizations for running deep learning workloads on Intel® Xeon® Scalable processors and Intel Data Center GPU's
- GitHub: https://github.com/IntelAI/models

Model Zoo w/ITEX (Env Setup)

Clone Intel Model Zoo to work directory:

- **$ git clone https://github.com/IntelAI/models.git**

Download the Intel oneAPI sample (ResNet50_Inference):

- **$ wget** https://raw.githubusercontent.com/oneapi-src/oneAPI-samples/release/2023.2/AI-and-Analytics/Features-and-Functionality/IntelTensorFlow_ModelZoo_Inference_with_FP32_Int8/ResNet50_Inference_gpu.ipynb

Activate conda env and run an interactive session with PVC:

- **$ conda activate tensorflow_xpu**
- **$ srun -p pvc-shared -w idc-beta-batch-pvc-node-11 --pty bash**
- **$ source /opt/intel/oneapi/setvars.sh**

Model Zoo w/ITEX (Jupyter setup)

Get the IP of your interactive session:

- **$ echo $(ip a | grep -v -e "127.0.0.1" -e "inet6" | grep "inet" | awk {'print($2)}' | sed 's/\/.*//')**

```
u105946@idc-beta-batch-pvc-node-07:~$ echo $(ip a | grep -v -e "127.0.0.1" -e "inet6" | grep "inet" | awk {'print($2)}' | sed 's/\/.*//')
10.10.10.14
u105946@idc-beta-batch-pvc-node-07:~$
```

Launch jupyter lab from interactive session:

- **$ jupyter-lab --ip 10.10.10.X**
  - Fill in the X with the IP you got from step above (i.e 10.10.10.14)

Take note of the IP and the port that jupyter launches on, it will look something like this:

- http://10.10.10.14:8888/lab?token=9d83e1d8a0eb3ffed84fa3428aae01e592cab170a4119130

From local terminal, port forward IP:

- **$ ssh idc -L PORT:10.10.10.X:PORT**
  - Fill in the IP (X), and the PORT # according to the last step
  - In this example it would look like this: **ssh idc -L 8888:10.10.10.14:8888**

Model Zoo w/ITEX(ResNet50 Inference Demo)

Getting Started with Intel Model Zoo

This code sample will serve as a sample use case to perform TensorFlow ResNet50v1.5 inference on a synthetic data implementing a FP32/FP16 and Int8 pre-trained model. The pre-trained model published as part of Intel Model Zoo will be used in this sample.

Select precision and download model

Select the precision that you would like to run resnet50 model with. `fp32` , `fp16` or `int8`

In [ ]:
```
precision = "fp32"  # or "int8" or "fp16"
batch_size = 1024
output_dir= './''
```

Choose data precision type **[fp32/fp16/int8]**

Choose batch size

In [ ]:
```
import os
initial_cwd = os.getcwd()
model_bucket = 'https://storage.googleapis.com/intel-optimized-tensorflow/models/gpu/'
model_precision = '_int8' if 'int8' == precision else ''
model_file = 'resnet50_v1' + model_precision + '.pb'
model_download_path = os.path.join(model_bucket, model_file)
```

In [ ]:
```
#download Intel's pretrained resnet50 model
if not os.path.exists(os.path.join(os.getcwd(), model_file)):
    ! wget $model_download_path
model_local_path = os.path.join(os.getcwd(), model_file)
if not os.path.exists(model_local_path):
    raise Exception("Failed to download pretrained Model file {}", model_download_path)
```

We will be using a synthetic dataset of size 244x244. It is important to set optimial batch_size, MKL run-time settings, TensorFlow's inter-intra number of threads to enable compute and data layer optimizations. We have identified optimial settings for popular topologies including ResNet50 to maximize GPU utlization. For more details on Run-time settings refer to blogs maximize XPU performance, Intel Model Zoo tutorials.

In [ ]:
```
%cd /opt/intel/oneapi/modelzoo/latest
```

Modify line to direct where Model Zoo is installed

Batch and Online Inference

In [ ]:
```
# Run inference using --batch-size=128 for throughput, or --batch-size=1 for latency
%run models/benchmarks/launch_benchmark.py \
    --in-graph $model_local_path \
    --model-name resnet50v1_5 \
    --framework tensorflow \
    --precision $precision \
    --mode inference \
    --batch-size=$batch_size \
    --benchmark-only \
    --output-dir=$output_dir \
    --gpu
```

Distributed DL on XPU

Multi-cards DL inference via Horovod on TensorFlow

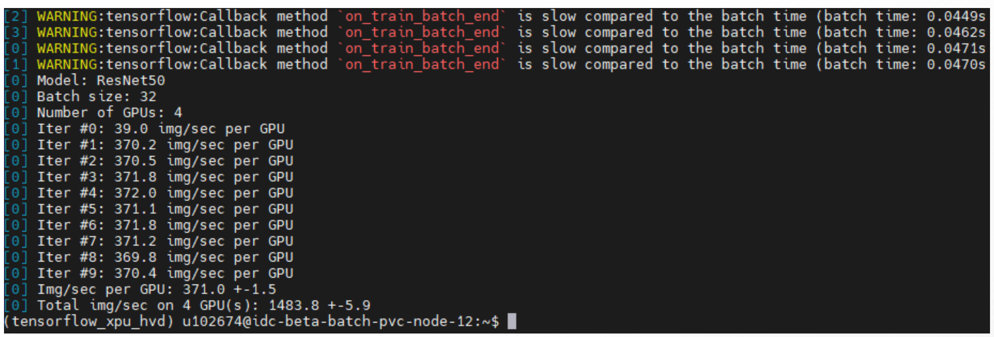
- **Pre-requisite :**
  - $source /opt/intel/oneapi/setvars.sh
  - $source activate tensorflow_xpu
  - $pip install intel-optimization-for-horovod
  - Find out number of root devices (GPU cards) by “sycl-ls”

```
u102674@idc-beta-batch-pvc-node-12:~$ sycl-ls
[opencl:acc:0] Intel(R) FPGA Emulation Platform for OpenCL(TM), Intel(R) FPGA Emulation Device 1.2 [2023.15.3.0.20_160000]
[opencl:cpu:1] Intel(R) OpenCL, Intel(R) Xeon(R) Platinum 8480+ 3.0 [2023.15.3.0.20_160000]
[opencl:gpu:2] Intel(R) OpenCL HD Graphics, Intel(R) Data Center GPU Max 1100 3.0 [23.13.26032.26]
[opencl:gpu:3] Intel(R) OpenCL HD Graphics, Intel(R) Data Center GPU Max 1100 3.0 [23.13.26032.26]
[opencl:gpu:4] Intel(R) OpenCL HD Graphics, Intel(R) Data Center GPU Max 1100 3.0 [23.13.26032.26]
[opencl:gpu:5] Intel(R) OpenCL HD Graphics, Intel(R) Data Center GPU Max 1100 3.0 [23.13.26032.26]
[ext_oneapi_level_zero:gpu:0] Intel(R) Level-Zero, Intel(R) Data Center GPU Max 1100 1.3 [1.3.26032]
[ext_oneapi_level_zero:gpu:1] Intel(R) Level-Zero, Intel(R) Data Center GPU Max 1100 1.3 [1.3.26032]
[ext_oneapi_level_zero:gpu:2] Intel(R) Level-Zero, Intel(R) Data Center GPU Max 1100 1.3 [1.3.26032]
[ext_oneapi_level_zero:gpu:3] Intel(R) Level-Zero, Intel(R) Data Center GPU Max 1100 1.3 [1.3.26032]
u102674@idc-beta-batch-pvc-node-12:~$
```

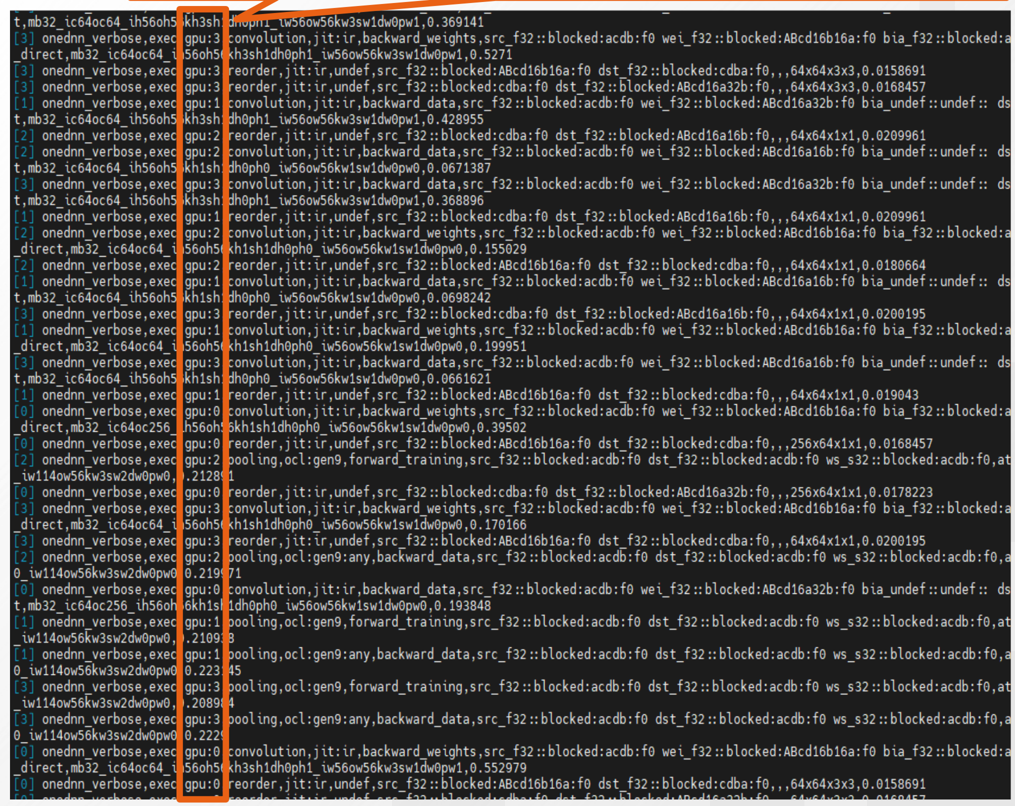
  - $wget https://raw.githubusercontent.com/intel/intel-optimization-for-horovod/main/examples/tensorflow2/tensorflow2_keras_synthetic_benchmark.py
- **Example usage**:
  - Set $NUM_RANKS as the number of root devices
  - **horovodrun -np $NUM_RANKS –p 22 python tensorflow2_keras_synthetic_benchmark.py**
- **Details:**
  - Need to specific port 22 due to firewall settings in Developer cloud
  - The script used SYCL backend to do distributed training and inference
  - Data Parallelism distributes data across GPUs while using the same model
  - The codes are from
    - https://github.com/intel/intel-optimization-for-horovod/blob/main/examples/tensorflow2/tensorflow2_keras_synthetic_benchmark.py

Example output and oneDNN verbose logs

- 4 root devices, 4 GPUs
- 4 ranks and one rank per GPU



Distribute oneDNN computation among gpu 0-3



How to enable Horovod for TF on PVC

- Follow below official guide but replace device name **from GPU to XPU**
  - https://github.com/horovod/horovod/blob/master/docs/tensorflow.rst

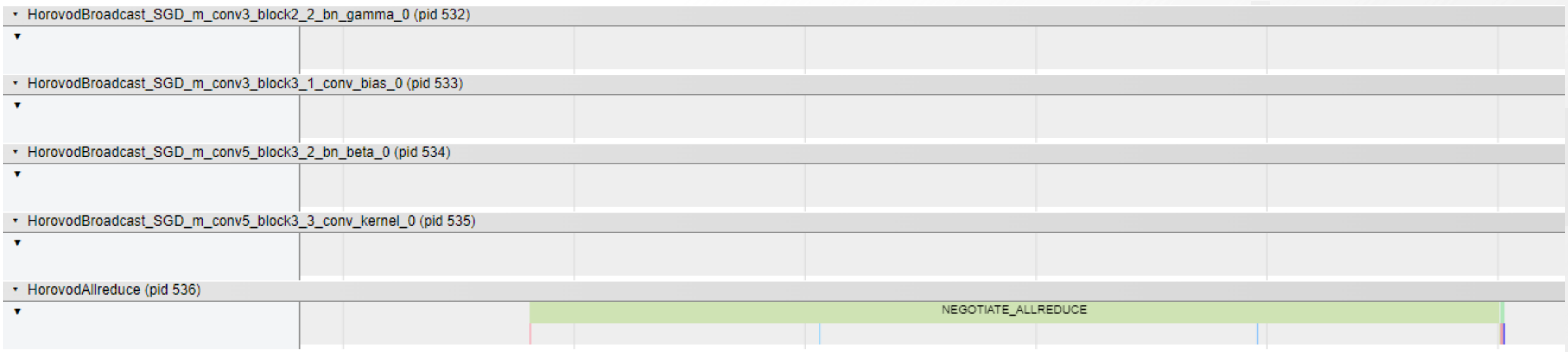
For Tensorflow v2:

```
gpus = tf.config.experimental.list_physical_devices('XPU')
for gpu in gpus:
  tf.config.experimental.set_memory_growth(gpu, True)
if gpus:
  tf.config.experimental.set_visible_devices(gpus[hvd.local_rank()], 'XPU')
```

Horovod timeline

- horovodrun -np 4 -p 22 --timeline-filename ./timeline.json python tensorflow2_keras_synthetic_benchmark.py

timeline.json



9 items selected. Slices (9)

Name	Wall Duration	Self time	Average Wall Duration	Occurrences
0	0.000 ms	0.000 ms	0.000 ms	1
1	0.000 ms	0.000 ms	0.000 ms	1
2	0.000 ms	0.000 ms	0.000 ms	1
3	0.000 ms	0.000 ms	0.000 ms	1
NEGOTIATE_ALLREDUCE	20.993 ms	20.993 ms	20.993 ms	1
ALLREDUCE	0.088 ms	0.015 ms	0.088 ms	1
WAIT_FOR_DATA	0.002 ms	0.002 ms	0.002 ms	1
WAIT_FOR_OTHER_TENSOR_DATA	0.056 ms	0.056 ms	0.056 ms	1
MPI_ALLREDUCE	0.015 ms	0.015 ms	0.015 ms	1
Totals	**21.154 ms**	**21.081 ms**	**2.350 ms**	**9**

Multi-cards DL inference via DDP on PyTorch

- **Pre-requisite :**
  - $source /opt/intel/oneapi/setvars.sh
  - $source activate pytorch_xpu
  - $python -m pip install oneccl_bind_pt -f https://developer.intel.com/ipex-whl-stable-xpu
  - Find out number of root devices (GPU cards) by “sycl-ls”

```
u102674@idc-beta-batch-pvc-node-12:~$ sycl-ls
[opencl:acc:0] Intel(R) FPGA Emulation Platform for OpenCL(TM), Intel(R) FPGA Emulation Device 1.2 [2023.15.3.0.20_160000]
[opencl:cpu:1] Intel(R) OpenCL, Intel(R) Xeon(R) Platinum 8480+ 3.0 [2023.15.3.0.20_160000]
[opencl:gpu:2] Intel(R) OpenCL HD Graphics, Intel(R) Data Center GPU Max 1100 3.0 [23.13.26032.26]
[opencl:gpu:3] Intel(R) OpenCL HD Graphics, Intel(R) Data Center GPU Max 1100 3.0 [23.13.26032.26]
[opencl:gpu:4] Intel(R) OpenCL HD Graphics, Intel(R) Data Center GPU Max 1100 3.0 [23.13.26032.26]
[opencl:gpu:5] Intel(R) OpenCL HD Graphics, Intel(R) Data Center GPU Max 1100 3.0 [23.13.26032.26]
[ext_oneapi_level_zero:gpu:0] Intel(R) Level-Zero, Intel(R) Data Center GPU Max 1100 1.3 [1.3.26032]
[ext_oneapi_level_zero:gpu:1] Intel(R) Level-Zero, Intel(R) Data Center GPU Max 1100 1.3 [1.3.26032]
[ext_oneapi_level_zero:gpu:2] Intel(R) Level-Zero, Intel(R) Data Center GPU Max 1100 1.3 [1.3.26032]
[ext_oneapi_level_zero:gpu:3] Intel(R) Level-Zero, Intel(R) Data Center GPU Max 1100 1.3 [1.3.26032]
u102674@idc-beta-batch-pvc-node-12:~$
```

  - $wget https://raw.githubusercontent.com/intel/torch-ccl/master/demo/demo.py
  - $wget https://raw.githubusercontent.com/oneapi-src/oneAPI-samples/master/AI-and-Analytics/Getting-Started-Samples/Intel_oneCCL_Bindings_For_PyTorch_GettingStarted/codes_for_ipynb/gpu.patch
  - $patch < ./gpu.patch
- **Example usage**:
  - Set $NUM_RANKS as the number of root devices
  - **I_MPI_PORT_RANGE=50000:50500 mpirun --launcher ssh -n 4 -l python demo.py**
- **Details:**
  - Need to use ssh launcher due to SLURM limitation and assign MPI port due to firewall settings
  - The script used SYCL backend to do distributed training and inference
  - Data Parallelism distributes data across GPUs while using the same model
  - The codes are from
    - https://github.com/oneapi-src/oneAPI-samples/tree/master/AI-and-Analytics/Getting-Started-Samples/Intel_oneCCL_Bindings_For_PyTorch_GettingStarted

Example output and oneCCL verbose logs

- 4 root devices, 4 GPUs
- 4 ranks and one rank per GPU

CCL_LOG_LEVEL=info I_MPI_PORT_RANGE=50000:50500 mpirun --launcher ssh -n 4 -l python demo.py

```
2023:07:11-13:09:42:(1692534) |CCL_INFO| can not retrieve ze pci properties
2023:07:11-13:09:42:(1692534) |CCL_INFO| can not retrieve ze fabric ports
2023:07:11-13:09:42:(1692531) |CCL_INFO| ze_rank_info_vec:
{
  comm_size: 4
    host: { idx: 0, name: idc-beta-batch-pvc-node-12 }
      rank: { idx: 0, device_uuid: { 134, 128, 218, 11, 47, 0, 0, 0, 41, 0, 0, 0, 0, 0, 0, 0 }, subdev_count: 0, subdev_id: na }
      rank: { idx: 1, device_uuid: { 134, 128, 218, 11, 47, 0, 0, 0, 58, 0, 0, 0, 0, 0, 0, 0 }, subdev_count: 0, subdev_id: na }
      rank: { idx: 2, device_uuid: { 134, 128, 218, 11, 47, 0, 0, 0, 154, 0, 0, 0, 0, 0, 0, 0 }, subdev_count: 0, subdev_id: na }
      rank: { idx: 3, device_uuid: { 134, 128, 218, 11, 47, 0, 0, 0, 202, 0, 0, 0, 0, 0, 0, 0 }, subdev_count: 0, subdev_id: na }
}
2023:07:11-13:09:42:(1692531) |CCL_INFO| host: 0, plane 0 contains ranks: 0
2023:07:11-13:09:42:(1692531) |CCL_INFO| host: 0, plane 1 contains ranks: 1
2023:07:11-13:09:42:(1692531) |CCL_INFO| host: 0, plane 2 contains ranks: 2
2023:07:11-13:09:42:(1692531) |CCL_INFO| host: 0, plane 3 contains ranks: 3
2023:07:11-13:09:42:(1692531) |CCL_INFO| rank_info_vec:
{
  comm_size: 4
    host: { idx: 0, name: idc-beta-batch-pvc-node-12 }
      rank: { idx: 0, local_proc_idx: 0, uuid: 01692531-77401004-1689106182067743 }
      rank: { idx: 1, local_proc_idx: 1, uuid: 01692532-77401004-1689106182067743 }
      rank: { idx: 2, local_proc_idx: 2, uuid: 01692533-12402977-1689106182067745 }
      rank: { idx: 3, local_proc_idx: 3, uuid: 01692534-77401004-1689106182067743 }
}
2023:07:11-13:09:42:(1692531) |CCL_INFO| topo_manager:
{
  comm_size: 4
  single_node: 1
  single_card: 0
  host_rank_counts: 4
  intra_card_colors: 0 0 1 1
  inter_card_colors: 0 1 0 1
  p2p_access: 0
}[0]
```

4 ranks and no subdev

```
3] --------------------------------------------------  ------------  ------------  ------------  --
3]                                               Name    Self CPU %      Self CPU   CPU total %
3] --------------------------------------------------  ------------  ------------  ------------  --
3]                                    aten::ones_like         0.44%       2.530us         4.14%
3]                                    aten::empty_like        0.36%       2.087us         0.84%
3]                                 aten::empty_strided        0.48%       2.786us         0.48%
3]                                          aten::fill_       2.86%      16.494us         2.86%
3] autograd::engine::evaluate_function: MseLossBackward ...   1.60%       9.264us         9.27%
3]                                    MseLossBackward0        0.56%       3.238us         7.66%
3]                              aten::mse_loss_backward       2.80%      16.181us         7.10%
3]                                    aten::zeros_like        0.38%       2.203us         4.30%
3]                                    aten::empty_like        0.34%       1.984us         1.04%
3]                                         aten::empty        0.69%       4.003us         0.69%
3]                                         aten::zero_        2.88%      16.624us         2.88%
3]    autograd::engine::evaluate_function: AddmmBackward0     0.91%       5.253us        18.83%
3]                                      AddmmBackward0        0.62%       3.567us        14.45%
3]                                             aten::t        0.22%       1.280us         0.66%
3]                                     aten::transpose        0.22%       1.245us         0.44%
3]                                    aten::as_strided        0.22%       1.268us         0.22%
3]                                            aten::mm       12.21%      70.495us        12.75%
3]                                         aten::empty        0.14%       0.794us         0.14%
3]                                       aten::resize_        0.40%       2.324us         0.40%
3]                                             aten::t        0.16%       0.918us         0.42%
3]                                     aten::transpose        0.11%       0.629us         0.26%
3]                                    aten::as_strided        0.15%       0.878us         0.15%
3]                                           aten::sum        2.88%      16.615us         3.08%
3]                                         aten::empty        0.20%       1.159us         0.20%
3]                                          aten::view        0.40%       2.288us         0.40%
3] autograd::engine::evaluate_function: torch::autograd ...   1.07%       6.179us         6.25%
3]                      torch::autograd::AccumulateGrad       0.33%       1.900us         3.13%
3]                                          aten::add_        2.80%      16.143us         2.80%
3]                                           aten::mul        2.06%      11.874us         2.06%
3]        autograd::engine::evaluate_function: TBackward0     0.27%       1.587us         0.80%
3]                                          TBackward0        0.13%       0.757us         0.52%
3]                                             aten::t        0.17%       0.963us         0.39%
3]                                     aten::transpose        0.09%       0.539us         0.23%
3]                                    aten::as_strided        0.13%       0.760us         0.13%
3] autograd::engine::evaluate_function: torch::autograd ...   2.16%      12.468us        54.33%
3]                      torch::autograd::AccumulateGrad       0.12%       0.685us         1.50%
3]                                          aten::add_        1.38%       7.949us         1.38%
3]                                           aten::mul        1.50%       8.677us         1.50%
3]                                    c10d::allreduce_        0.24%       1.377us        49.17%
3]                 oneccl_bindings_for_pytorch::allreduce     4.13%      23.845us        48.94%
3]        oneccl_bindings_for_pytorch::xpu::get_ccl_comms     0.11%       0.610us         0.11%
3]                                  aten::record_stream       0.32%       1.838us         0.32%
3]            oneccl_bindings_for_pytorch::xpu::allreduce    44.38%     256.203us        44.38%
3]                                    aten::as_strided        0.49%       2.854us         0.49%
3]                                    aten::as_strided        0.18%       1.066us         0.18%
3]     torch.distributed.ddp.reducer::copy_bucket_to_grad     0.19%       1.097us         3.73%
3]                                         aten::copy_        3.54%      20.443us         3.54%
3]     torch.distributed.ddp.reducer::copy_bucket_to_grad     0.11%       0.617us         1.97%
3]                                         aten::copy_        1.86%      10.742us         1.86%
3] --------------------------------------------------  ------------  ------------  ------------  --
3] Self CPU time total: 577.280us
3] Self XPU time total: 141.760us
3]
3] Runing optim: 2 on device xpu:3
```

Spent 40% of time on XPU allreduce

How to enable DDP for PT on PVC

- Follow below official guide but replace device name **from GPU or CPU to XPU and change backend to ccl**
  - https://pytorch.org/tutorials/intermediate/ddp_tutorial.html

```
--- demo.py     2023-01-22 03:46:30.162956299 +0900
+++ demo-gpu.py 2023-01-22 03:53:57.998938230 +0900
@@ -36,7 +36,7 @@
     # Initialize the process group with ccl backend
     dist.init_process_group(backend='ccl')

-    device = 'cpu' #"xpu:{}".format(dist.get_rank())
+    device = "xpu:{}".format(dist.get_rank())
     model = Model().to(device)
     if dist.get_world_size() > 1:
         model = DDP(model, device_ids=[device] if (device != 'cpu') else None)
```

Back Up