global AI bootcamp

March 4th 2023, Torino - Italy

Accelerating Pytorch with IPEX

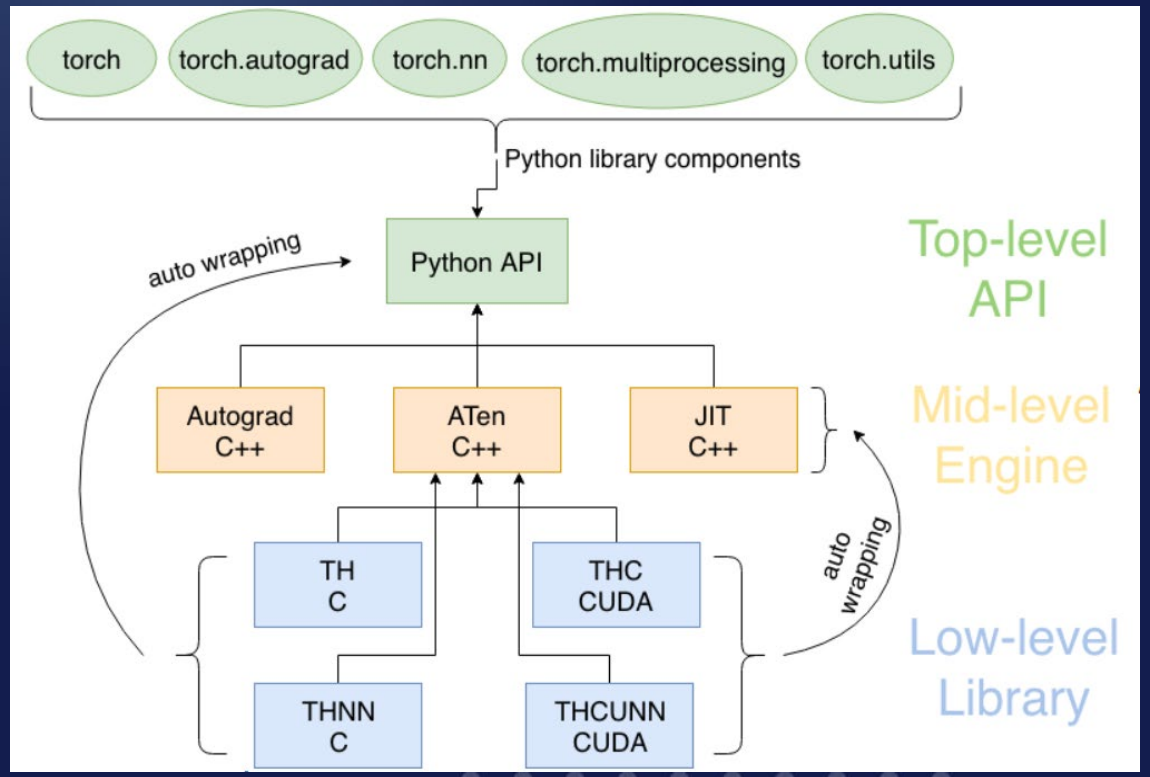
Abhilash Majumder
Intel

global AI community

Deltatre Innovation Lab

#GlobalAIBootcamp

Pytorch


torch
torch.autograd
torch.nn
torch.multiprocessing
torch.utils
Python library components
auto wrapping
Python API
Top-level API
Autograd C++
ATen C++
JIT C++
Mid-level Engine
TH C
THC CUDA
auto wrapping
THNN C
THCUNN CUDA
Low-level Library

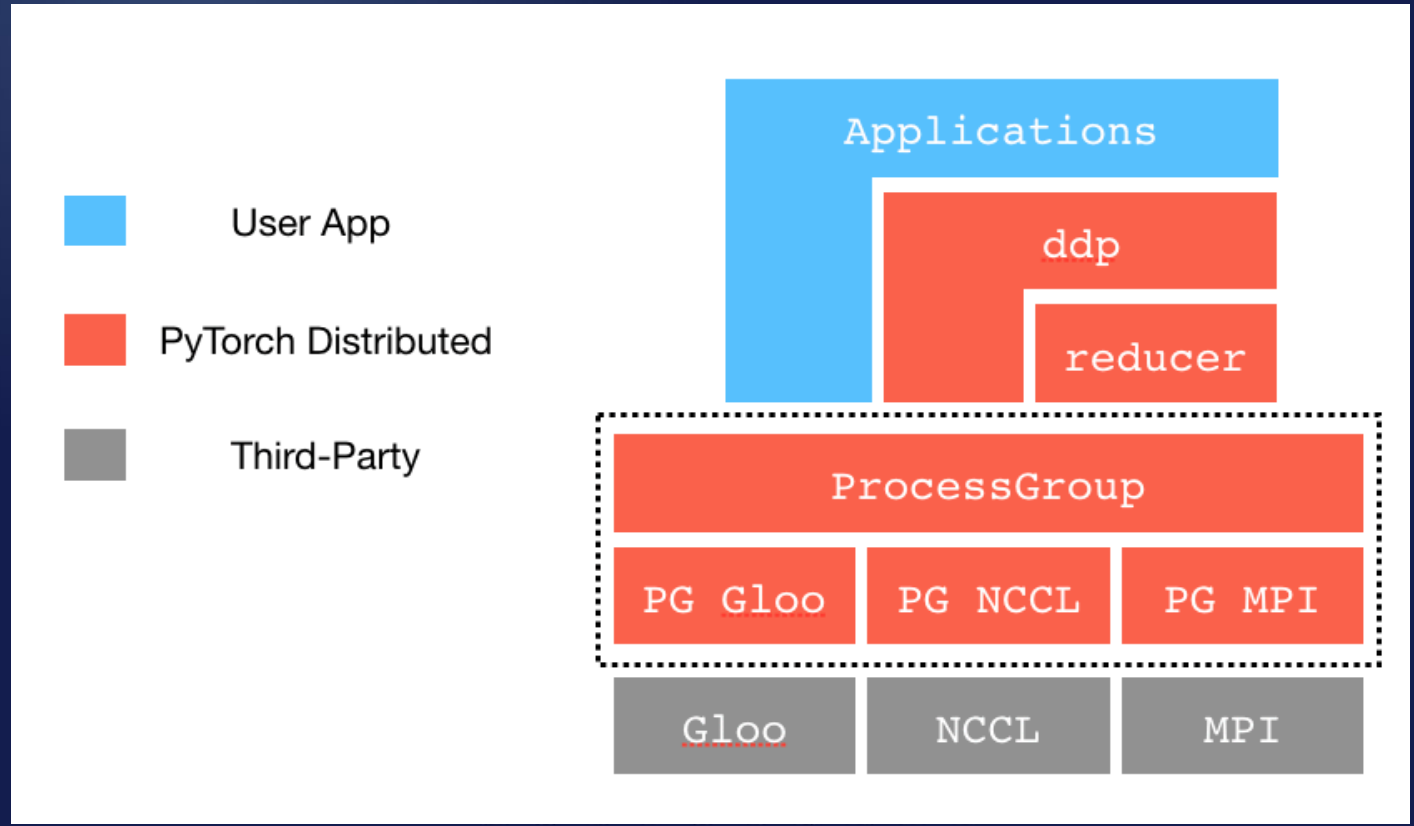
Introduction to Pytorch

- Pytorch is a framework created for accelerating deep learning workloads across hardwares.
- Pytorch leverages several compute backend and compilers (consistent with LLVM) for creating model graphs .
- The most significant kernels used for compute fall under ATen, Autograd and JIT.
- Pytorch uses different backends for different devices such as binding to NCCL for Cuda /Nvidia or CCL for XPU/Intel backends for GPU training & inference

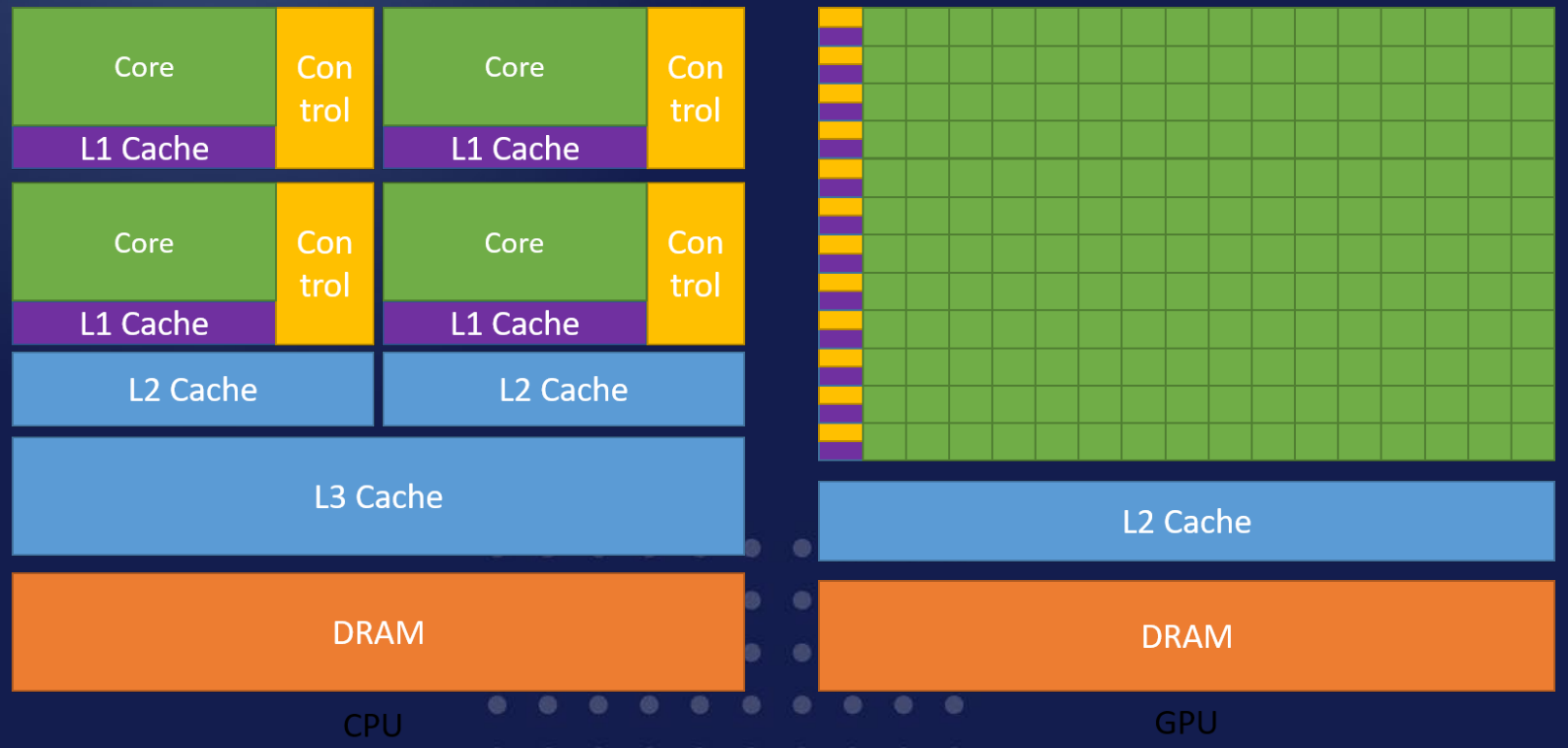
Introduction to Pytorch

- Pytorch has different higher level wrappers to ensure distributed training and inference across hardwares
- Distributed training enables pytorch to shard data and model across GPUs/TPUs or CPUs (multi). In case of CUDA systems this is done using distributed NCCL backend which inturn calls Level 0.
- Level 0 is where optimized atomic kernels are run for each independent GPU (where part of the training is offloaded) and then send the results of compute to CPU (synchronization)

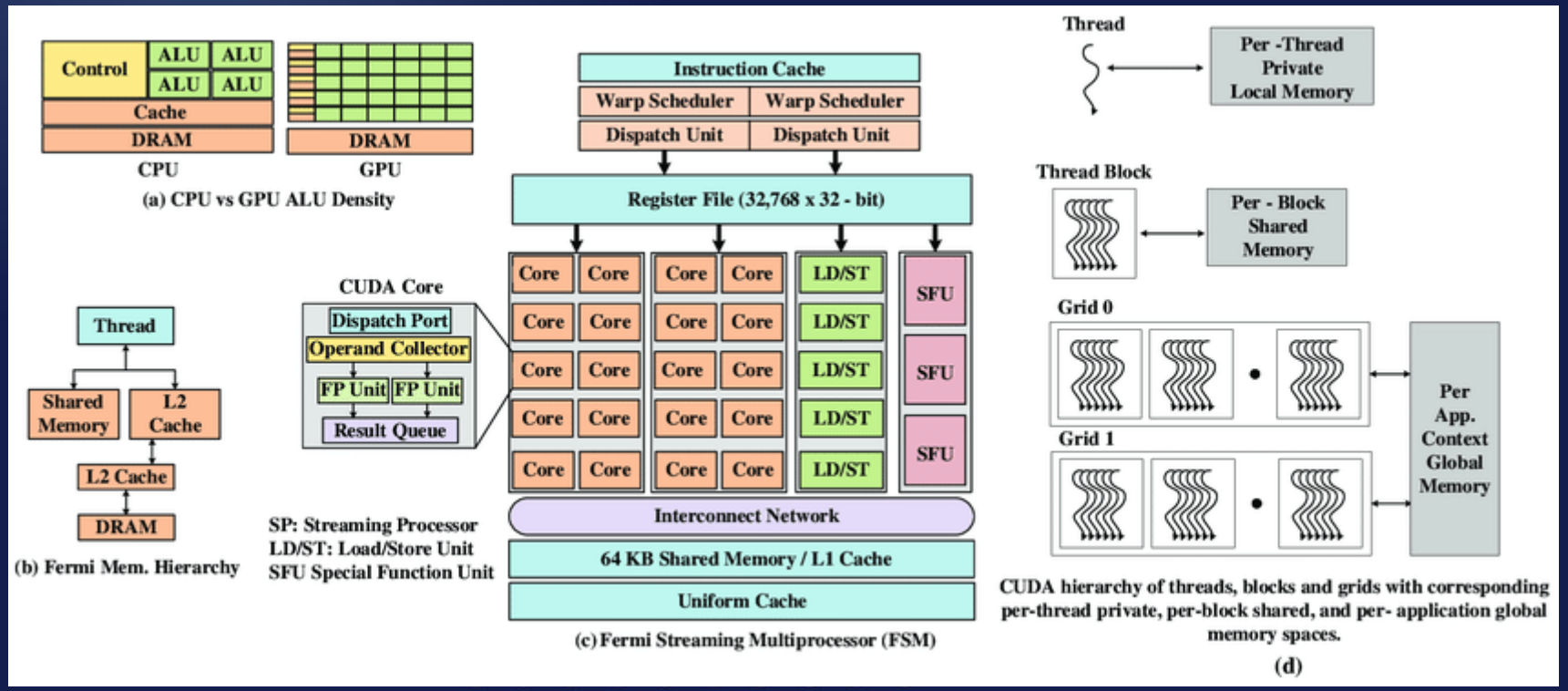
Pytorch Distributed



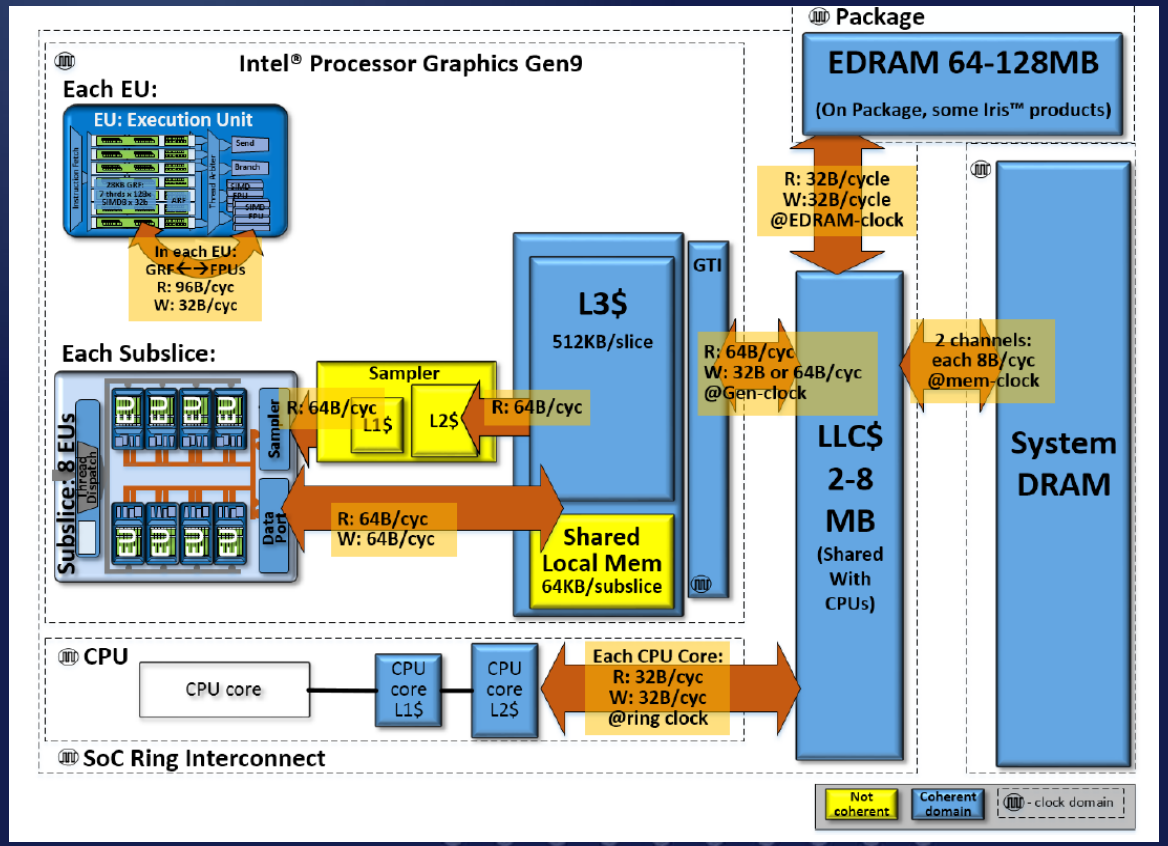
Pytorch GPU -CPU



Pytorch GPU L0 (Cuda)



Pytorch GPU L0 (Intel)



global AI community

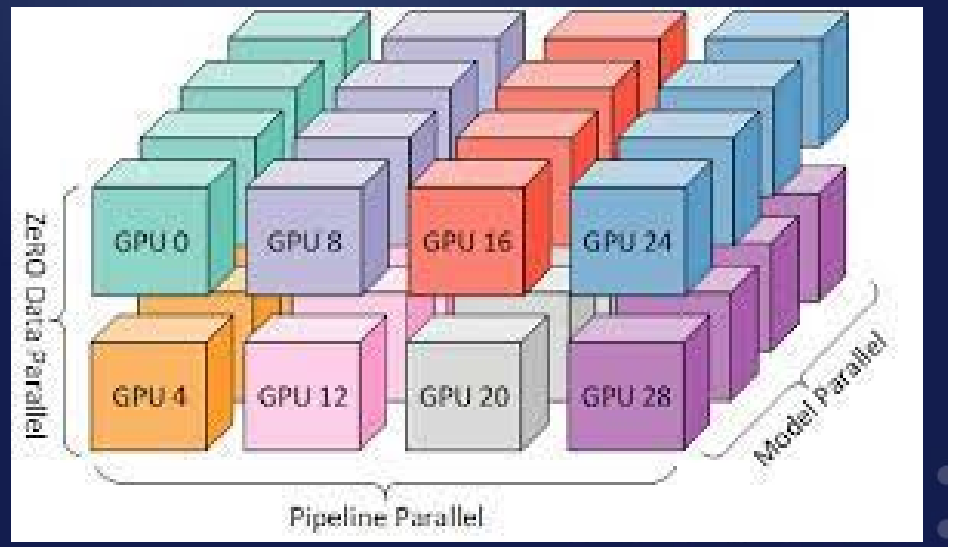
Deltatre Innovation Lab

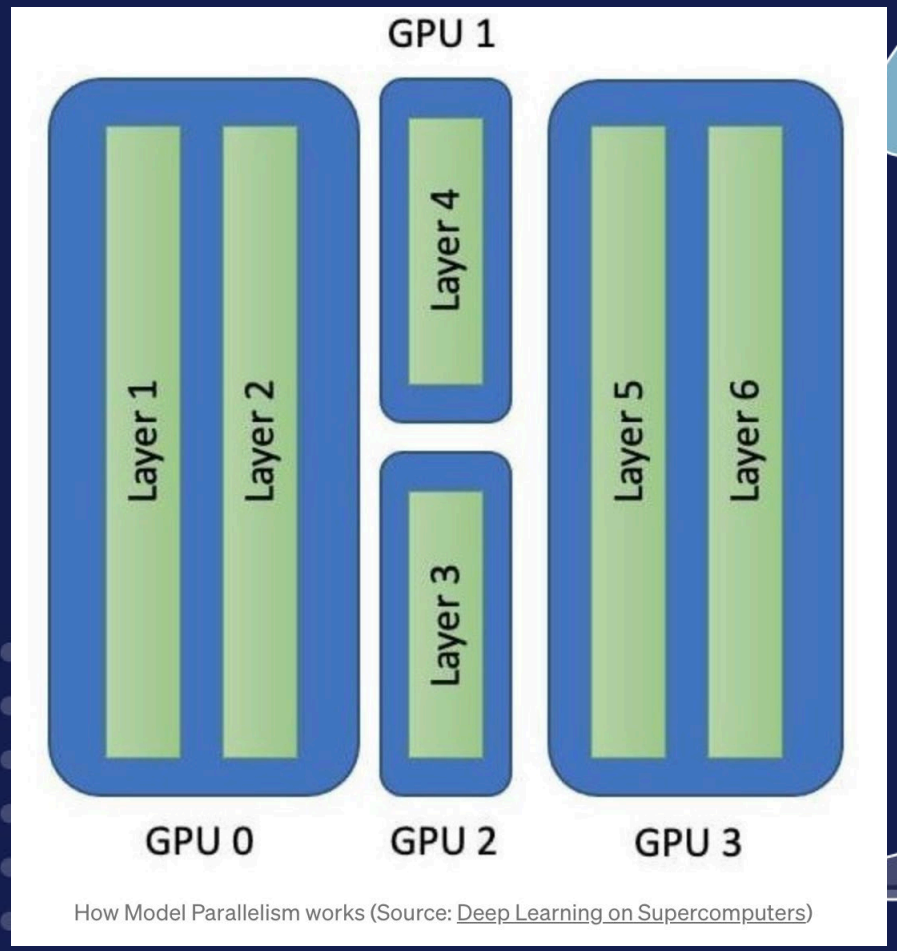
#GlobalAIBootcamp

Introduction to Pytorch

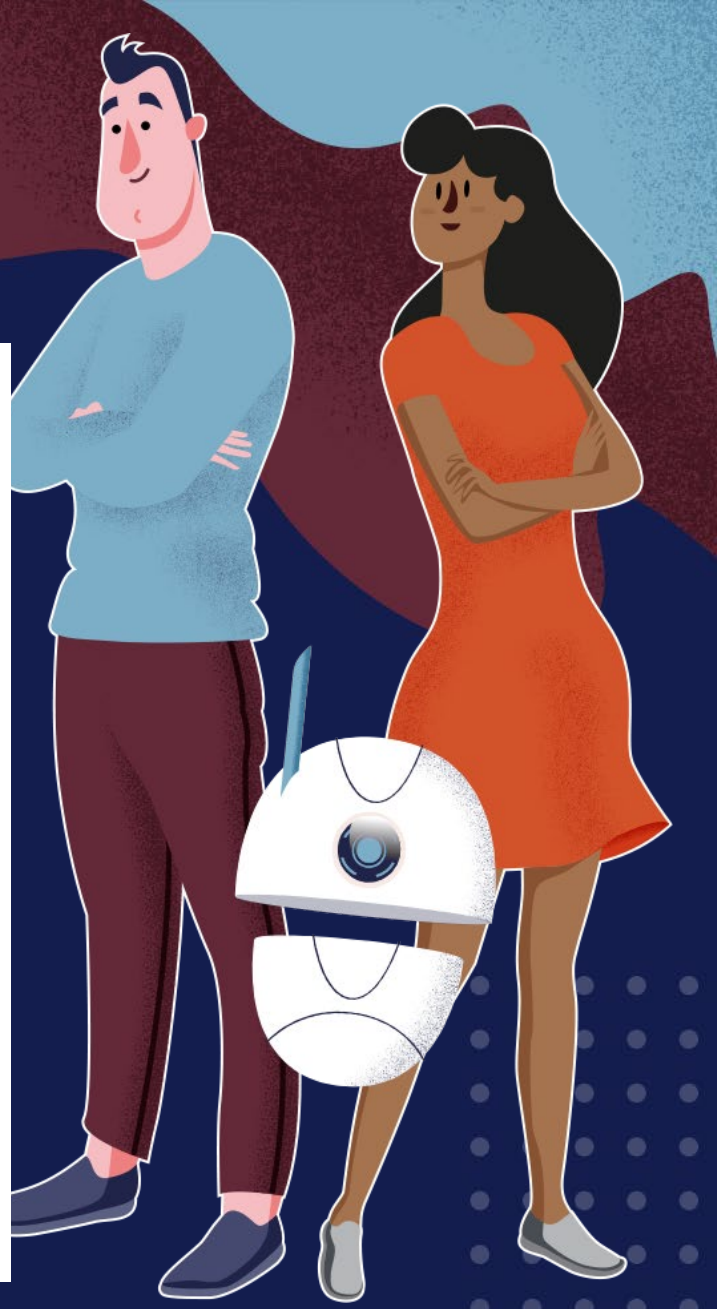
- Pytorch enables training for billion/trillion parameter models through 2 major ways – Data Parallel and Model Parallel.
- Data Parallel involves sharding data across accelerators(GPUs) . Model Parallel involves sharding the model across activations across accelerators(GPUs)
- When distributed model training or inference is in progress, there is always a need for sending data to and from CPUs and GPUs. For this there are different collective algorithms which control the flow of data across accelerators.

Pytorch MP





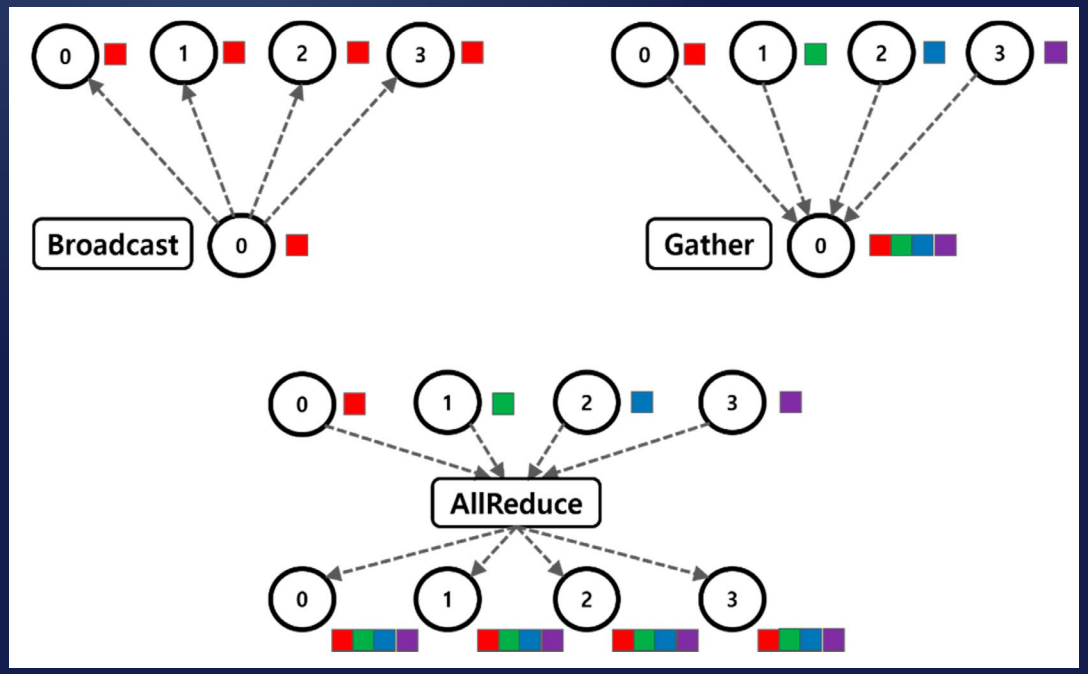
How Model Parallelism works (Source: Deep Learning on Supercomputers)



Introduction to Pytorch

- Acceleration on collective calls is done better through either NCCL backend or CCL backend depending on Nvidia or Intel GPUs using DP/MP approaches.
- Collective calls mainly include broadcast, reduce,reduce-scatter,all-reduce,all-to-all, all gather. These are different algorithms which facilitate transfer data to and fro devices.
- Advanced Pytorch samples for DP/MP : Link
- Pytorch Documentation on torch distributed: Link
- Pytorch Core: Link

Pytorch Collective Calls

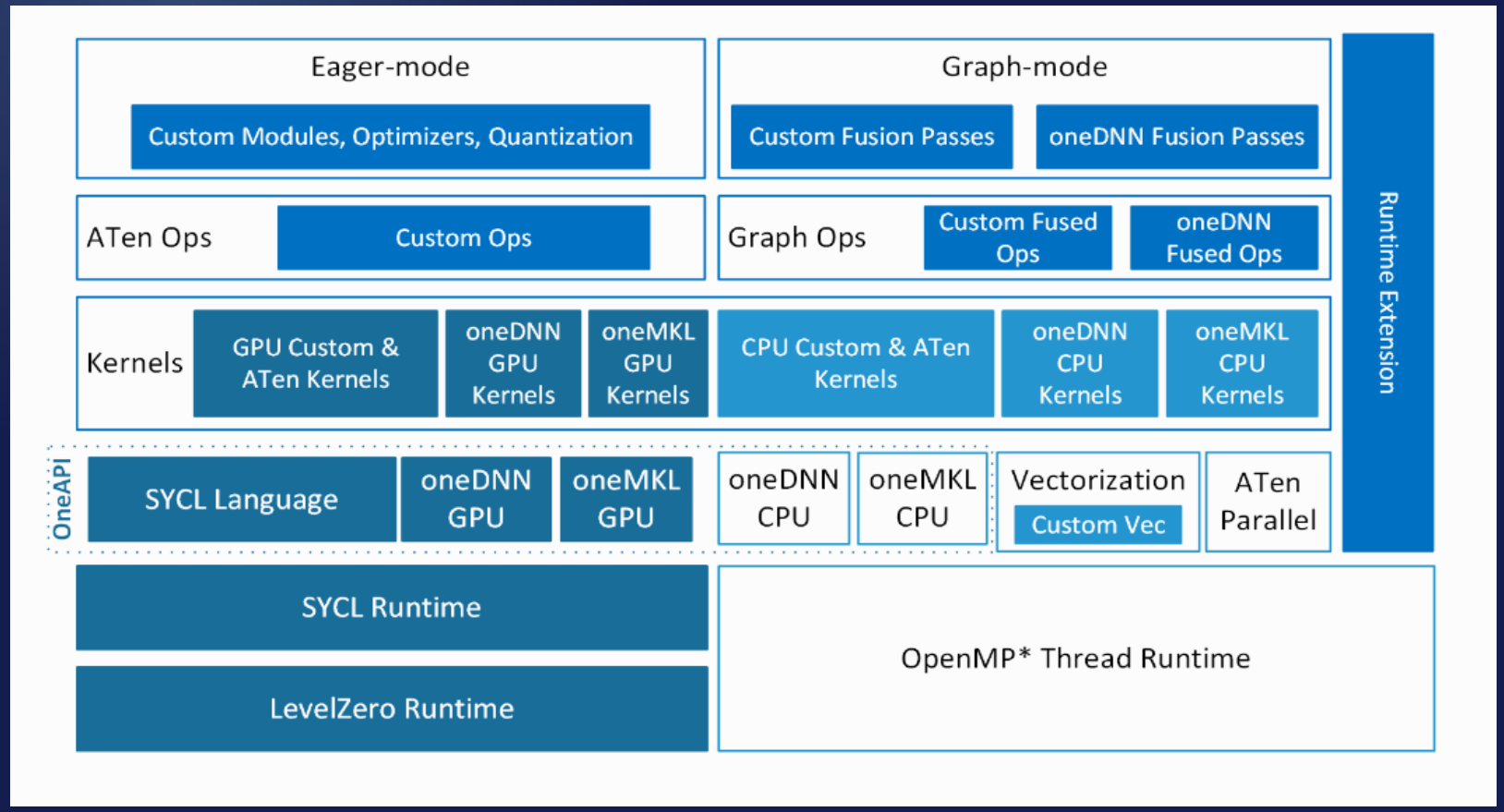


global AI community

Deltatre Innovation Lab

#GlobalAIBootcamp

Intel Extension for Pytorch



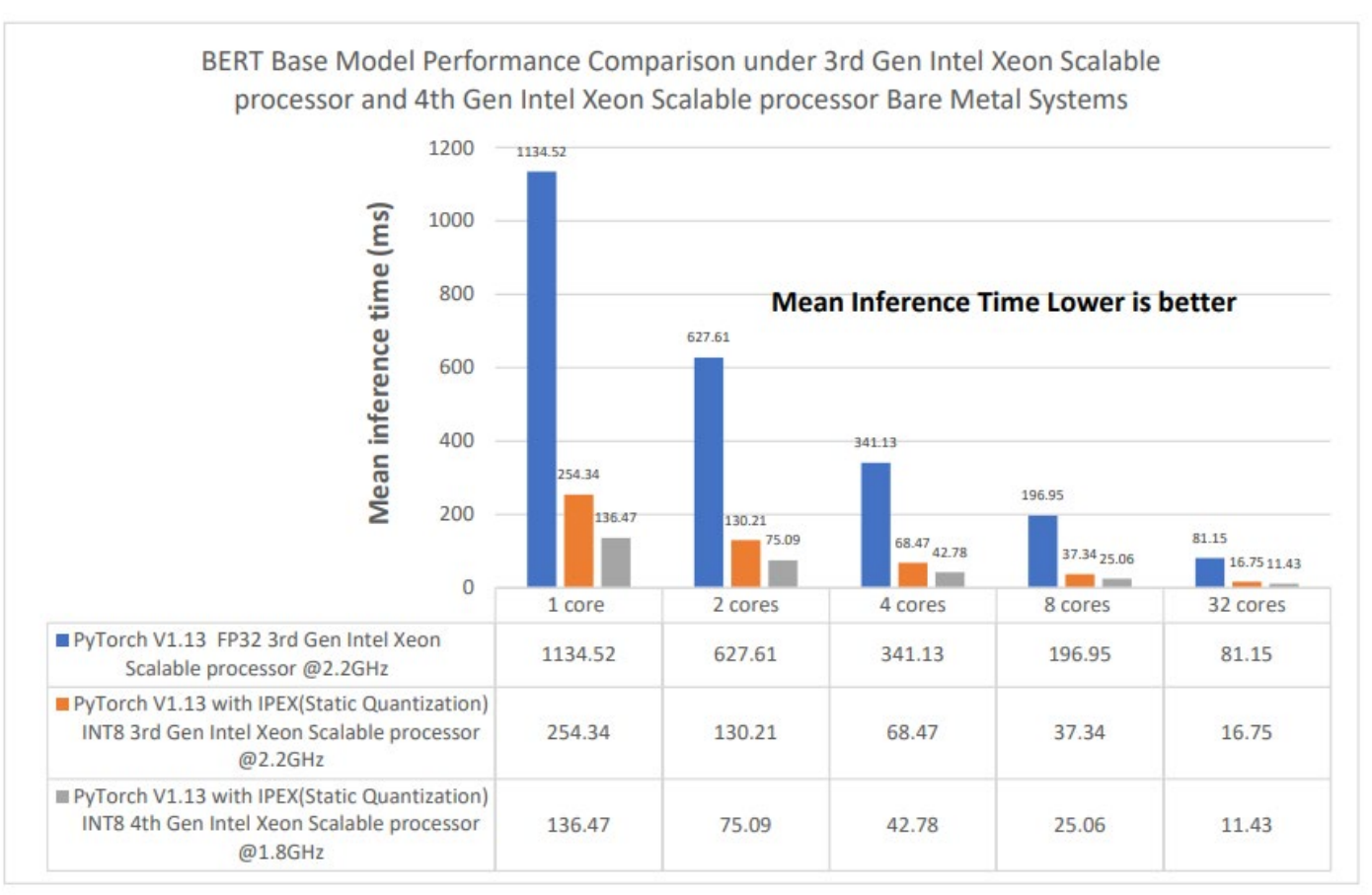
IPEX

- IPEX is a framework extension for pytorch dedicated for accelerating training and inference across Intel Xeon CPUs and GPUs.
- IPEX offers extra performance boost on CPUs owing to dedicated AVX-512 Vector Neural Network Instructions(VNNI) and Intel Advanced Matrix Extensions(AMX) .
- For eager execution modes including fusion operations, IPEX optimizes pytorch graph although using the same methods as standard pytorch (such as jit).
- It extends custom compute kernels for faster execution on baseline pytorch

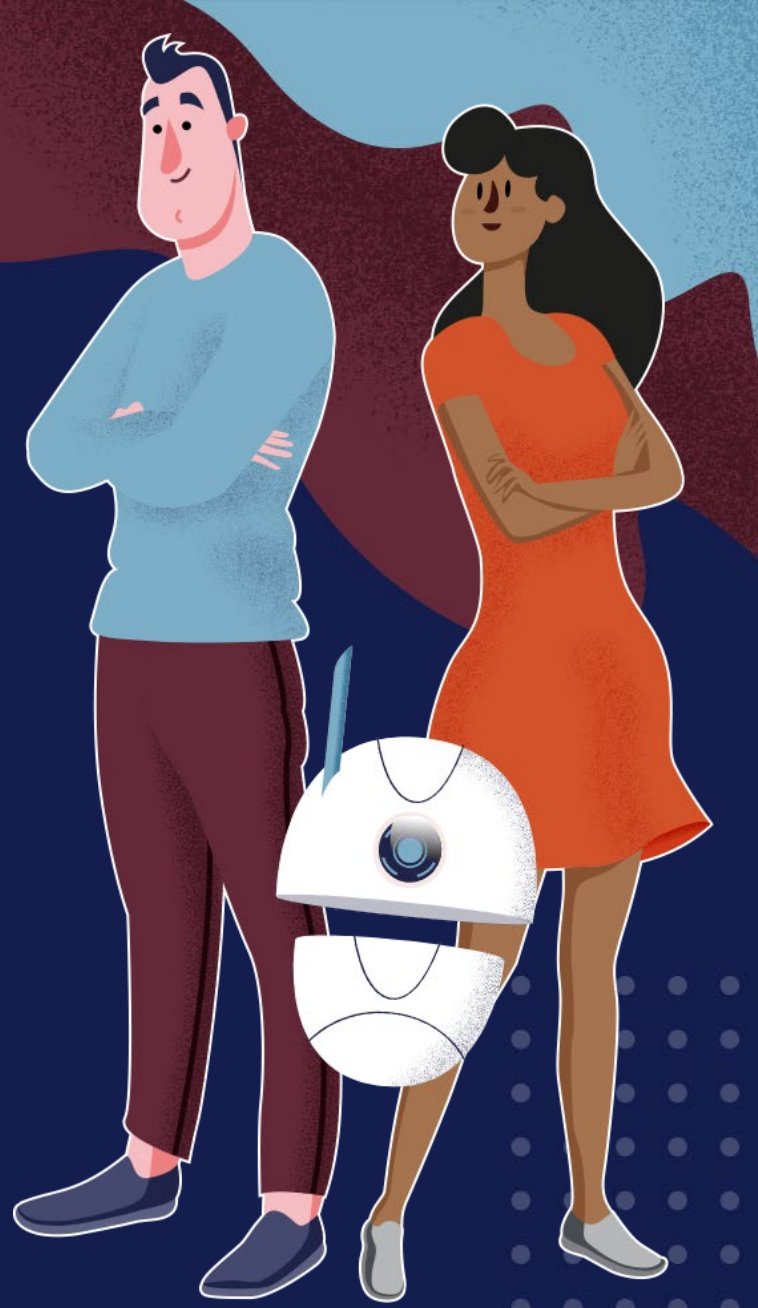
IPEX

- AMX inside Intel CPUs- Sapphire Rapids accelerate GEMM kernels in different dataformats including BF16 and INT8 .
- It uses 2 dimensional CPU register with tile architecture,which need to be saved and restored during context switches.
- As in the case of TorchDDP, for multicluster training and inference, IPEX provides greater support owing to custom compute kernels to achieve high throughput.
- Intel AMX combines new instruction set which turns large matrices into a single operation thereby speeding compute across tiles .

IPEX Benchmarking BERT



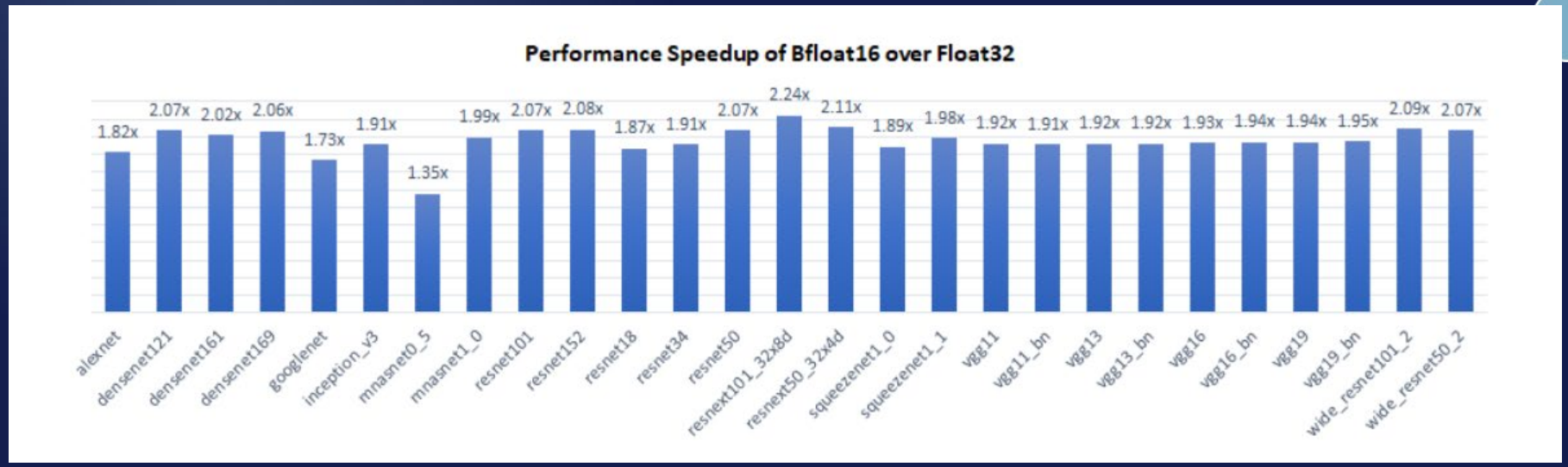
	1 core	2 cores	4 cores	8 cores	32 cores
PyTorch V1.13 FP32 3rd Gen Intel Xeon Scalable processor @2.2GHz	1134.52	627.61	341.13	196.95	81.15
PyTorch V1.13 with IPEX(Static Quantization) INT8 3rd Gen Intel Xeon Scalable processor @2.2GHz	254.34	130.21	68.47	37.34	16.75
PyTorch V1.13 with IPEX(Static Quantization) INT8 4th Gen Intel Xeon Scalable processor @1.8GHz	136.47	75.09	42.78	25.06	11.43

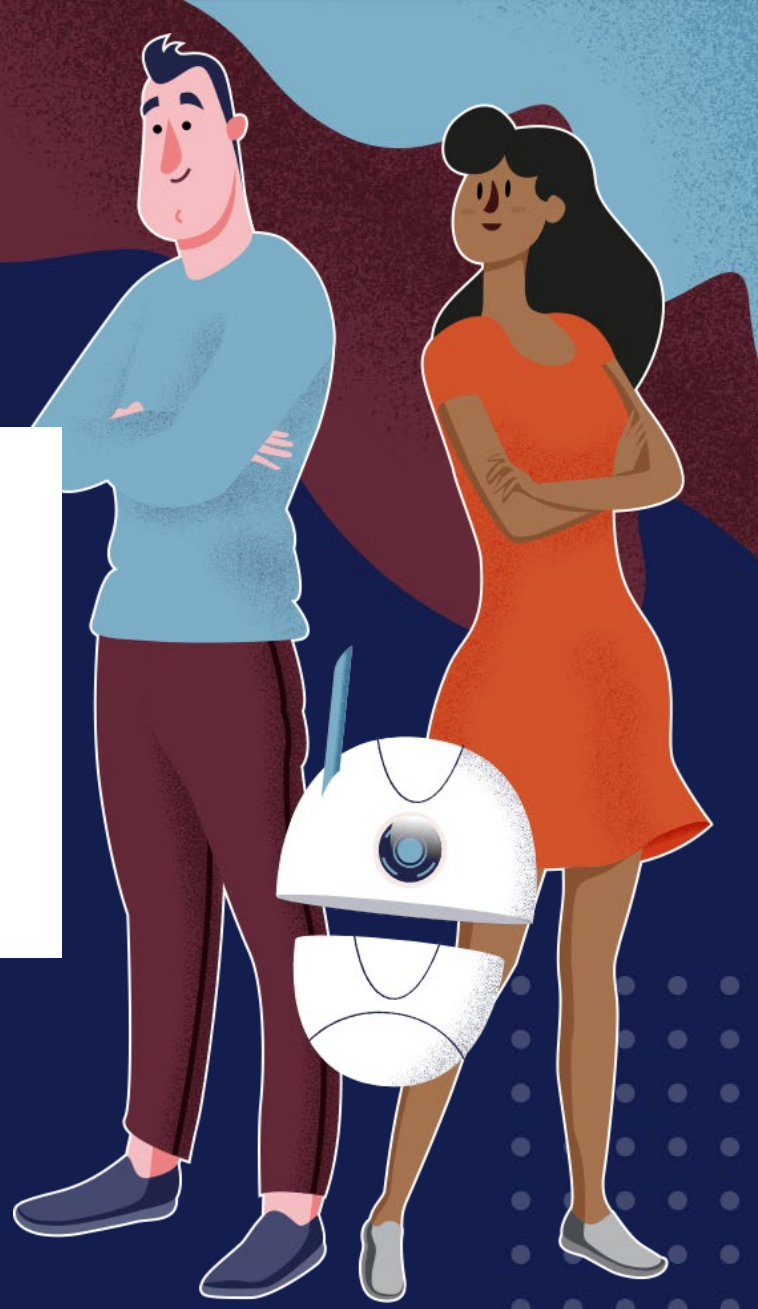


IPEX

- AVX-512 is an instruction set which can dynamically increase performance for demanding workloads and usages like AI inferencing.
- This instruction set combines 3 operations into a single VNNI instruction set which reduces the number of clock cycles. This implies with the same raw power of CPU, there will be a faster execution of compute.
- The BF16 computation is very expensive process as for any compute (add,mul etc bf16 needs to be converted to fp32 and back. AVX set provides additional optimizations for this interconversion.

IPEX Benchmarking BF16/FP32





global AI community

Deltatre Innovation Lab

#GlobalAIBootcamp

IPEX

- VNNI enabled 8bit computes boost not only throughput but also bandwidth which inturn makes communication much more better across devices.
- Acceleration is even more evident across Intel dGPUs which use additional XMX (Matrix Extensions) for deep learning workloads.
- Quantization capacity is optimized in the case of IPEX as some instruction sets are combined to enable fused quantization .
- For devices having larger caches, the bandwidth tolerance involving AVX-VNNI is quite optimized.

IPEX Usage

Intel* Extension for PyTorch* usage

- FP32

```
import torch
import torchvision.models as models

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

import intel_extension_for_pytorch as ipex
model = model.to(memory_format=torch.channels_last)
model = ipex.optimize(model)
data = data.to(memory_format=torch.channels_last)

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

- BFloat16

```
import torch
from transformers import BertModel

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])

import intel_extension_for_pytorch as ipex
model = ipex.optimize(model, dtype=torch.bfloat16)

with torch.no_grad():
  with torch.cpu.amp.autocast():
    model(data)
```

IPEX Optimizations Snapshot

Major Optimization Methodologies

3-Pillar Framework Optimization Techniques

Op

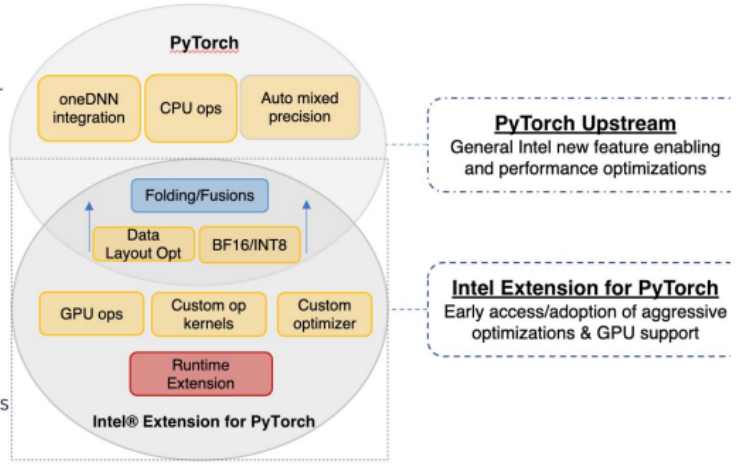
- Vectorization and Multi-threading
- Low-precision BF16/INT8 compute
- Ease-of-use BF16 compute with Auto-Mixed-Precision (AMP)
- Data layout optimization for better cache locality

Graph

- Constant folding to reduce compute
- Op fusion for better cache locality

Runtime

- Thread affinitization and multi-streams
- Memory buffer pooling
- GPU runtime
- Launcher



PyTorch Upstream
General Intel new feature enabling and performance optimizations

Intel Extension for PyTorch
Early access/adoption of aggressive optimizations & GPU support

IPEX

- Ipex is optimized for most kernel modifications which arise from standard Pytorch to be compatible with the Intel dGPUs.
- It follows oneAPI standards for Intel which ensures collective communications are done in an efficient manner.
- IPEX Github Link: Link
- IPEX Blogs: Link
- Model Zoo: Link

global AI bootcamp

March 4th 2023, Torino - Italy

DEMO

Questions and Social

- Github: Link
- Linkedin: Link
- Twitter: Link