intel ai summit
英特爾AI科技論壇

Bringing AI Everywhere

oneAPI and Intel AI Products: Optimizing Deep Learning Performance on Intel Hardware Platforms

Jing Xu
人工智能軟體科技顧問

March 27th, 2024

Overview

Bring AI Everywhere

intel Ai summit

Bringing AI Everywhere

Large

Unlock the AI Continuum

Novel Applications

Small

Training & Fine-Tuning

Training

Fine-Tuning

Deployment

Inference

Inference & Deployment

Cloud

Simplify AI Infrastructure

Scalable Systems & Solutions

Client

AI Specific

Accelerate the Workload

Foundational Silicon & Software

SynapseAI

intel. Gaudi

intel. Data Center GPU Max Series

intel. Data Center GPU Flex Series

intel. Xeon

intel. Arc Graphics

intel. Core Ultra

intel. Core

oneAPI

General Purpose

intel AI summit

Bringing AI Everywhere

Large

Small

Unlock the AI Continuum

Novel Applications

Training & Fine-Tuning

Training

Fine-Tuning

Deployment

Inference

Inference & Deployment

Cloud

Client

Simplify AI Infrastructure

Scalable Systems & Solutions

AI Specific

General Purpose

Accelerate the Workload

Foundational Silicon & Software

SynapseAI

intel Gaudi

intel Data Center GPU Max Series

intel Data Center GPU Flex Series

intel Xeon

intel Arc Graphics

intel Core Ultra

intel Core

oneAPI

intel AI summit

Intel® Gaudi® 2 AI Accelerator

7nm Process Technology

24 Tensor Processor Cores

96 GB On-Board HBM2

48 MB SRAM

24 Integrated Ethernet ports

Proven Performance

- The ONLY alternative to H100 for training LLMs based on MLPerf
- Trained GPT-3* model in 311 minutes on 384 Intel Gaud2 accelerators

Price Performance

- Intel Gaudi2 accelerators with FP8 estimated to deliver price-performance >H100
- ~2x price-performance to A100

Scalability

- 95% linear scaling on MLPerf GPT-3 training benchmark
- Access large Intel Gaudi2 cluster on the Intel Developer Cloud

Ease of Use

- Software optimized for deep learning training and inference
- PyTorch, Hugging Face, Optimum Library optimizations

intel Ai summit

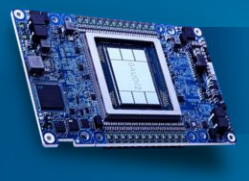
Performance metrics based on MLPerf Training 3.0 benchmark. For configuration details, see the results published by MLPCommons..
GPT-3 model tested on MLPerf Training 3.0 consisted of representative 1% slice of the entire GPT-3 model Performance expectations for Gaudi2 with FP8 based on Intel internal evaluation June 2023
Price-performance claim based on comparable pricing of Intel Gaudi server and Nvidia A100 server and MLPerf Inference 3.1 Results. , Aug 2023. See Supermicro for server pricing. Price-performance claim based on significant pricing differential between Intel Gaudi2 and Nvidia H100 server, MLPerf Training 3.0 Results, May 2023 and internal estimates of performance advancement with FP8. See Supermicro for server pricing. Results may vary

Seamless Code Transitioning

Performant AI Code with Minimal Changes

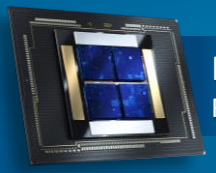
PyTorch deepspeed •••

Across Generations & Architectures



Intel® Gaudi® 2 AI Accelerator

Intel® Gaudi® 3 AI Accelerator



Next Gen GPU (Codenamed Falcon Shores)

SynapseAI®

Gaudi Software Suite

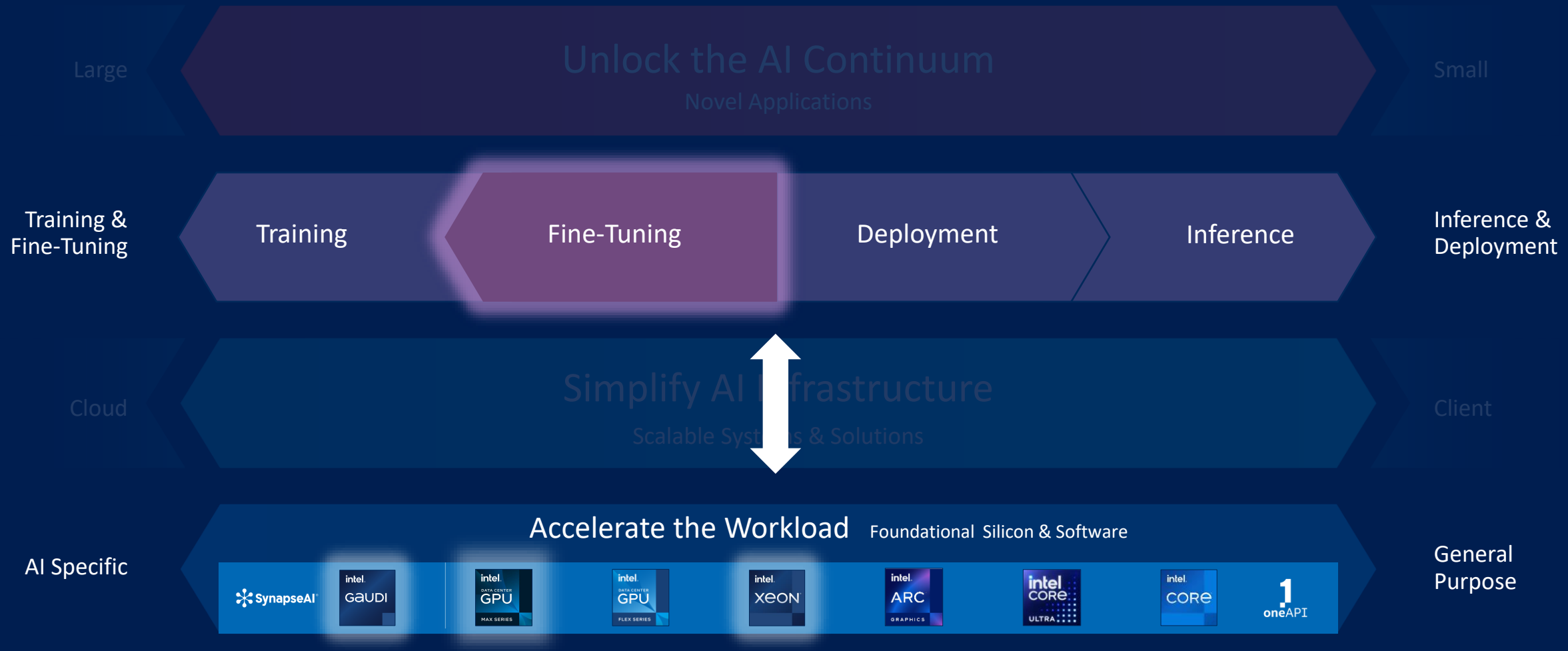
Powered by Transitioning Into a Single Software Environment

oneAPI

Unified Programming Model

intel Ai summit

Fine Tuning



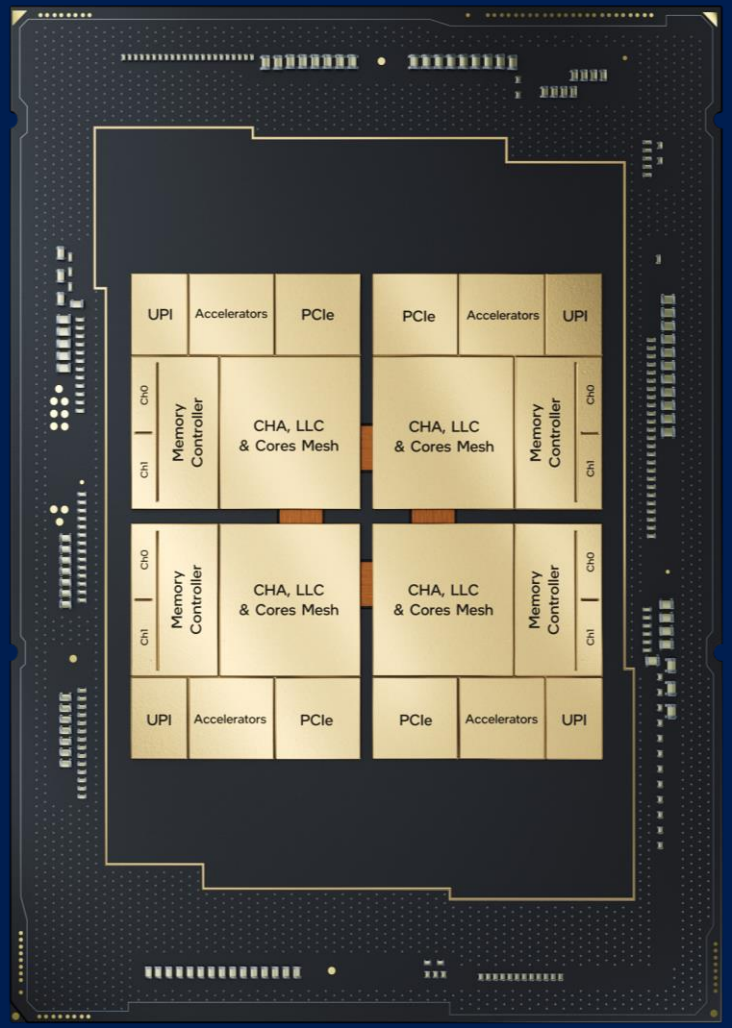
intel AI summit

Fine-tune with
Intel® Gaudi® 2 Processor
When Optimal Speed is Desired

Fine-tune On Intel® Xeon®/Intel® Data Center GPUs,
Exploiting Its Industry-leading
Ubiquity In the Data Center

Intel Provides Solution Options for Fine-tuning Gen AI and LLMs to Fit Workload Needs

4th Generation Intel® Xeon® Scalable Processor



3-10x speedup and 7.7x performance/watt[1]

for INT8/ BF16 models
with Built-In AI Acceleration

4th Gen Intel® Xeon® Scalable Processor with Intel® Advanced Matrix Extensions acceleration vs. 3rd Gen Intel® Xeon® Scalable Processors

Intel® AI software

300+ DL Models
50+ optimize ML and Graph Models
Optimizations up-streamed
Intel® AI Developer Tools

2x PCI Express 5.0 Bandwidth

Compared to 3rd Gen Intel® Xeon® Scalable Processors

OneAPI AI Ecosystem

Use any popular DL, ML, and Data processing library and framework, operating system, and virtual machine manager

1.5x DDR5 Memory Bandwidth and Capacity

Compared to 3rd Gen Intel® Xeon® Scalable Processors

Up to 512 GB/Socket Protected Memory Enclave—Intel® Security Guard Extensions

Confidential AI supported in BigDL and OpenVINO™ toolkit

intel Ai summit

1 See [A16, A17, A33] at intel.com/processorclaims: 4th Gen Intel Xeon Scalable processors. Results may vary.

Intel® Advanced Matrix Extensions (Intel® AMX)

Acceleration Engine

What is Intel® AMX?

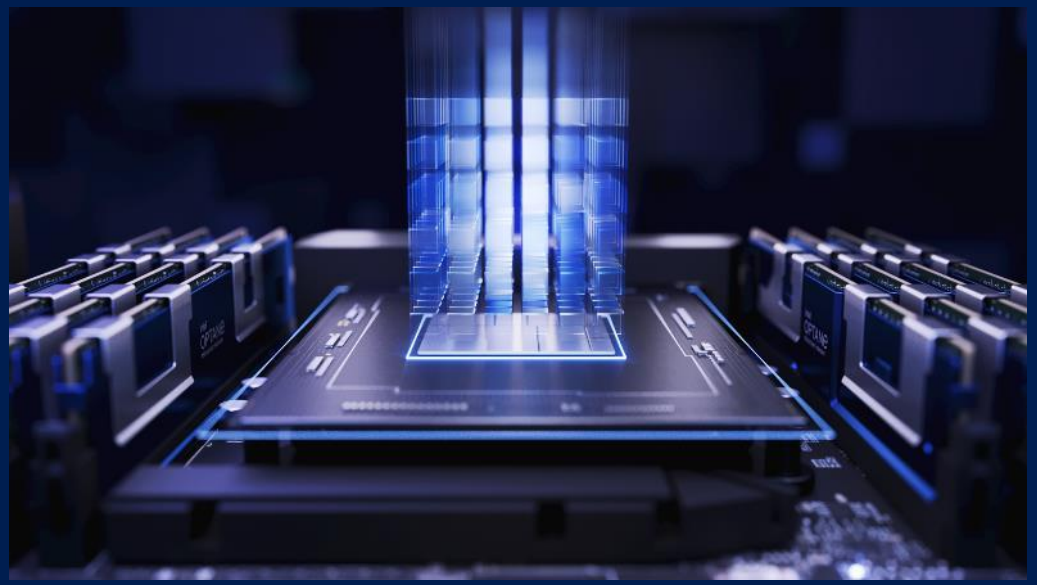
- Intel® AMX is a built-in accelerator that improves the performance of **deep learning** training and inference on 4th Gen Intel® Xeon® processors
- Advanced matrix multipliers are integrated into **EVERY** core

Business Value

- Help to **lower customers' TCO** as it raises the bar for where they can meet AI SLAs without the need for a discrete accelerator

Software Support

- Works **out-of-box** on industry-standard frameworks, toolkits and libraries such as PyTorch, TensorFlow, and OpenVINO
- **vSphere 8 supports Intel AMX**



PyTorch Training and Inference

Up to

10x higher

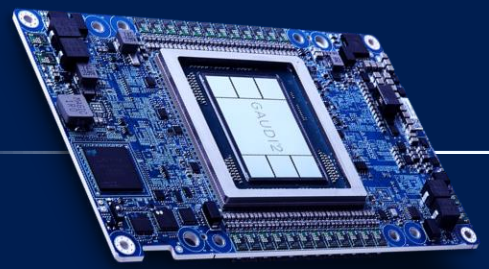
PyTorch for both real-time inference and training performance with built-in Intel AMX (BF16) vs. the prior generation (FP32)

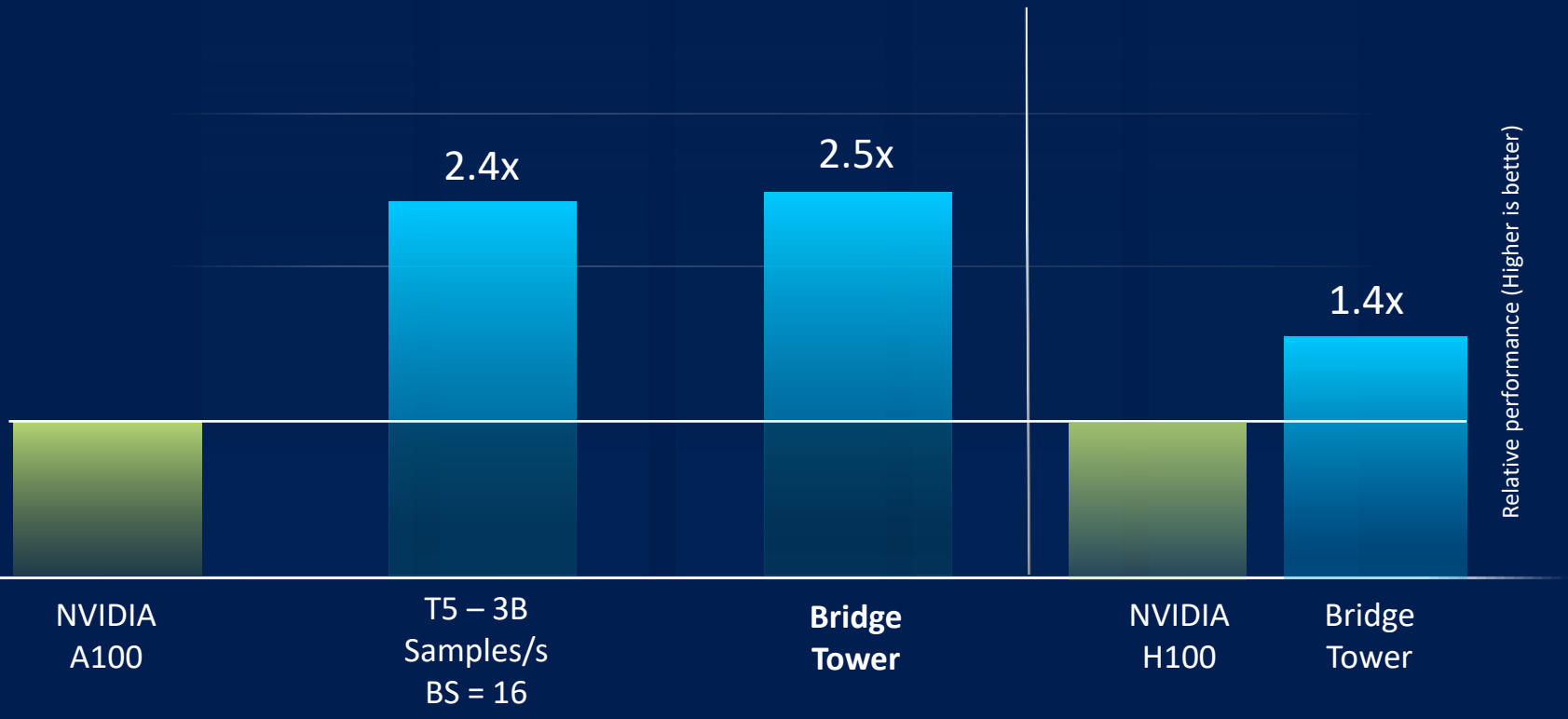
intel Ai summit

Visit here for workloads and configurations. Results may vary.

Fine-tuning Across Numerous LLMs

Hugging Face Evaluations Substantiate Intel® Gaudi® 2 Accelerator LLM Performance vs. Nvidia A100 and H100





Visit https://habana.ai/habana-claims-validation for workloads and configurations. Results may vary.
https://huggingface.co/blog/habana-gaudi-2-benchmark
https://huggingface.co/blog/bridgetower

Out-of-the Box Intel® Xeon® Fine Tuning

Optimized Models & Spaces

Dolly	LLAMA2	MPT	LDM3D	Whisper	*100k's Mode*

Intel Optimized Hugging Face Libraries & Tools

Transformers	Diffusers	Accelerate	PEFT	Optimum
Fine Tuning	Use Cases	Fine Tuning at Scale	Efficient Fine Tuning	Performance Optimization

Foundational Stack

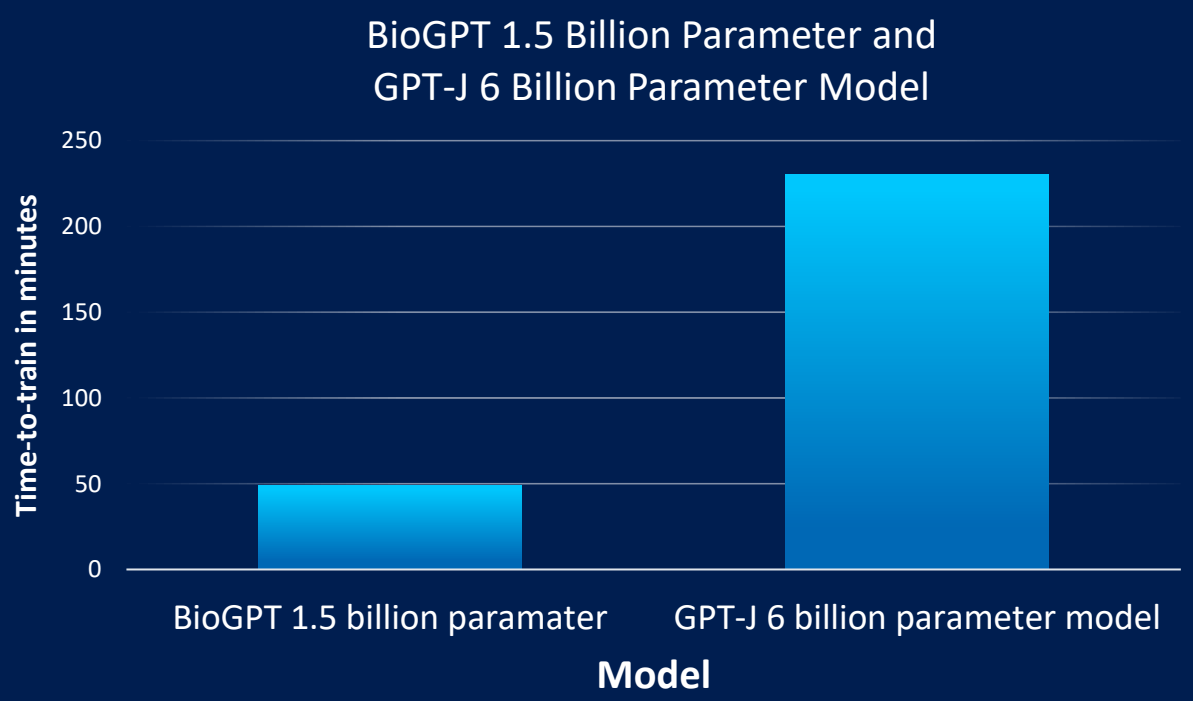
PyTorch + Intel® Extension for PyTorch*

deepspeed

ONNX RUNTIME

OpenVINO

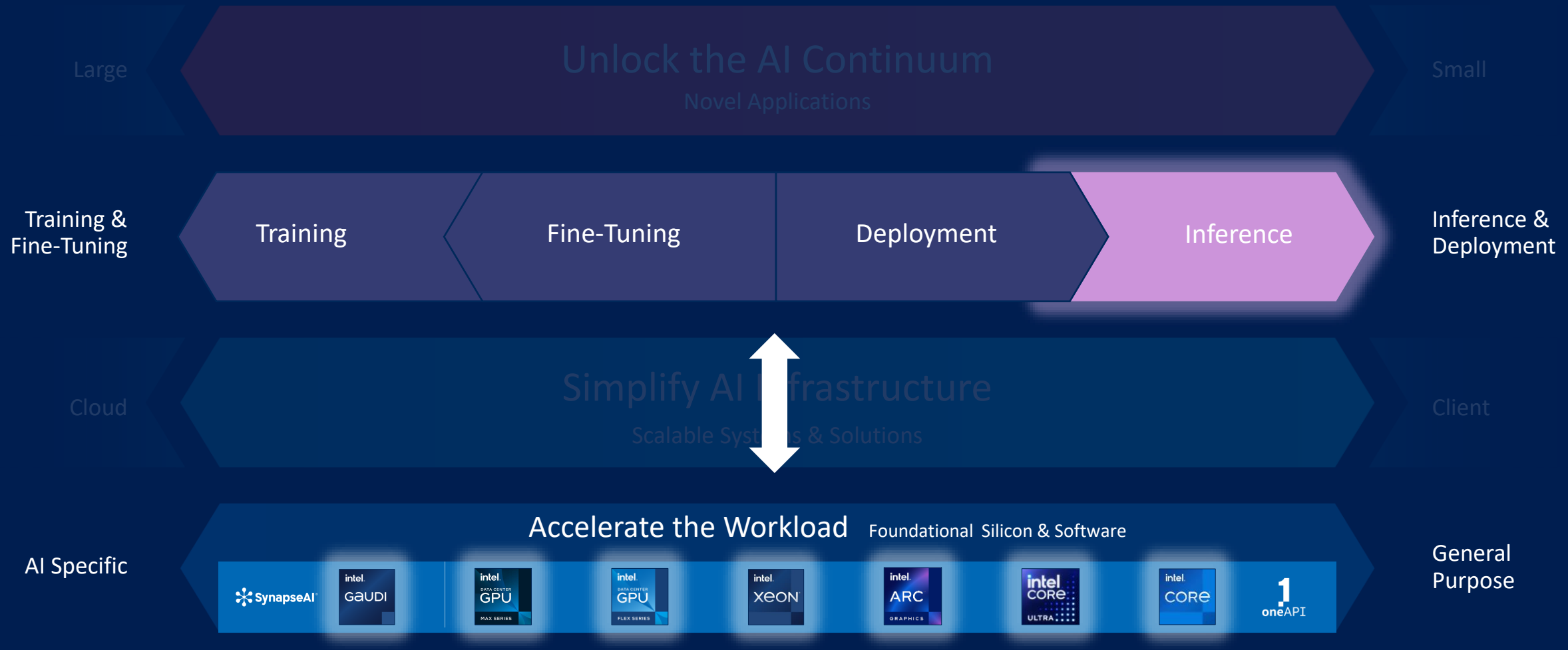
Fine Tuning Open-source Commercial Large Foundational Models In Minutes To Hours



Visit here for workloads and configurations. Results may vary.

intel Ai summit

Inference



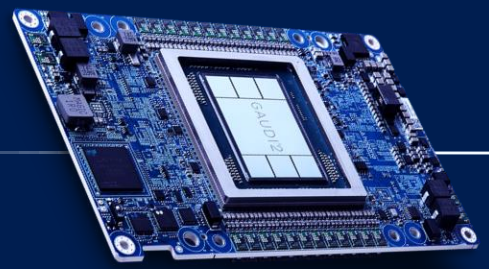
intel AI summit

intel GAUDI

Energy Efficiency

Throughput-per-Watt on BLOOMZ 176B Inference is 1.79x better than H100; 1.61x better than A100

Inference Advantage Across Multiple LLM Performance Metrics

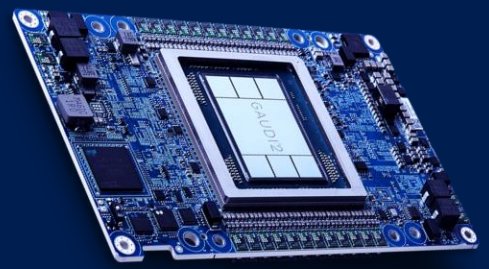


intel Ai summit

https://huggingface.co/blog/habana-gaudi-2-benchmark
https://huggingface.co/blog/habana-gaudi-2-bloom
Visit https://habana.ai/habana-claims-validation/for workloads and configuration regarding power consumption claims. Results may vary.

Intel® Gaudi® 2 AI Accelerator:

Solving LLM Challenges



Inference on GPT-J

Intel Gaudi 2 Accelerator with FP8

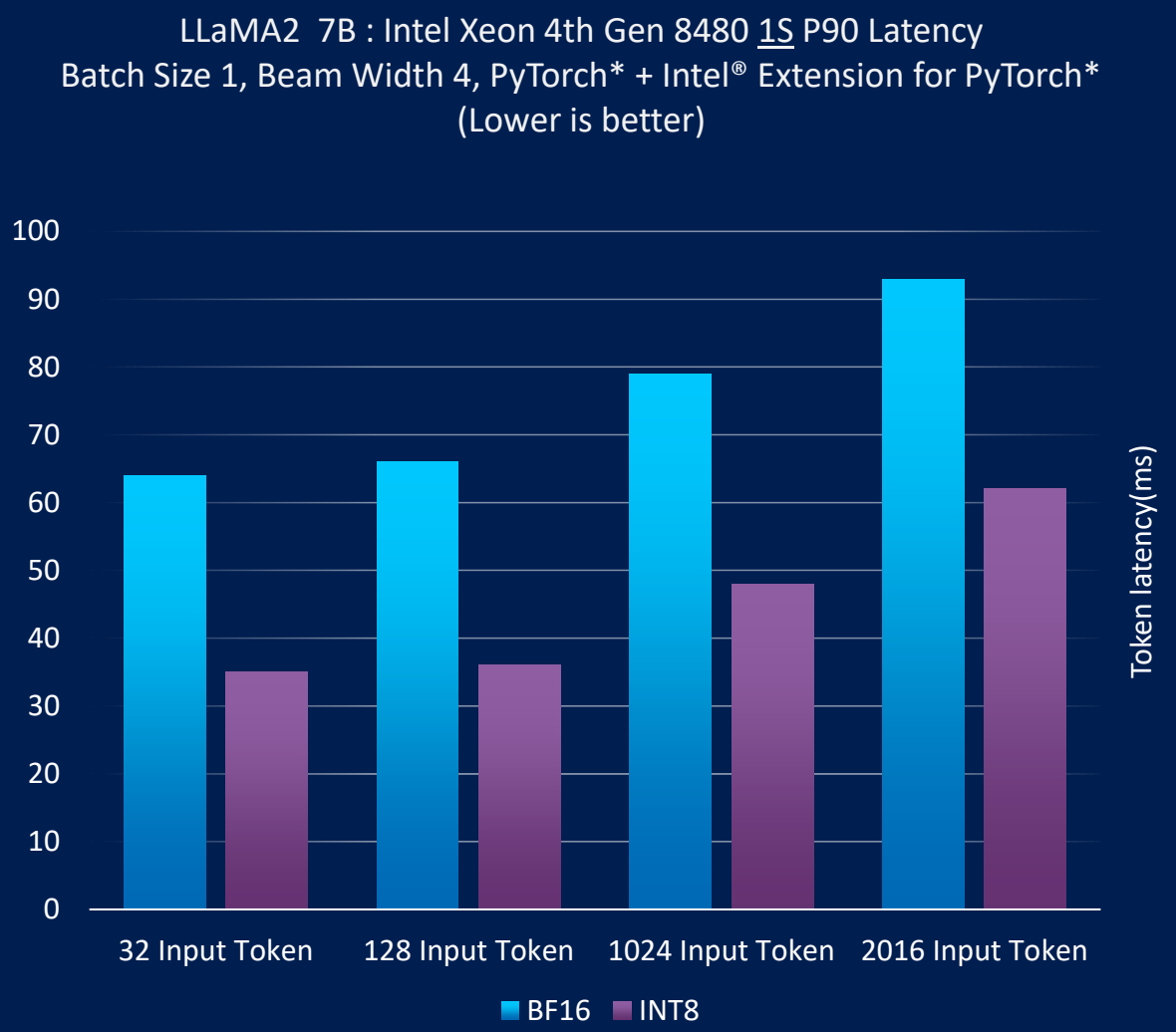
- Near-parity* on GPT-J with H100
- Outperformed A100 by 2.4x (Server) and 2x (Offline)
- Achieved 99.9% accuracy with FP8

GPT-J On MLPerf Inference Benchmark

intel Ai summit

Source: MLPerf Inference 3.1 Data Center Benchmark Results: https://mlcommons.org/en/inference-datacenter-31/
Intel® Gaudi® 2 AI accelerator on GPT-J Vs H100 with 1.09x (Server) and 1.28 (Offline). Results may vary

LLaMA2 (7B) Inference with 4th Gen Intel® Xeon® Processors



Visit here for workloads and configurations. Results may vary.

- Use any popular industry standard AI libraries
- Intel AI Platform validated with over 300 inference models
- One socket of 4th Gen Intel® Xeon® processors can run LLaMa2 chatbots in under 100ms 2nd token latency

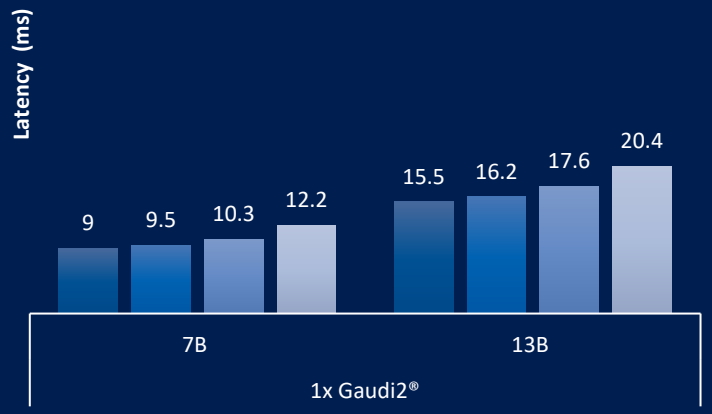
intel Ai summit

Inference Across Multiple Products

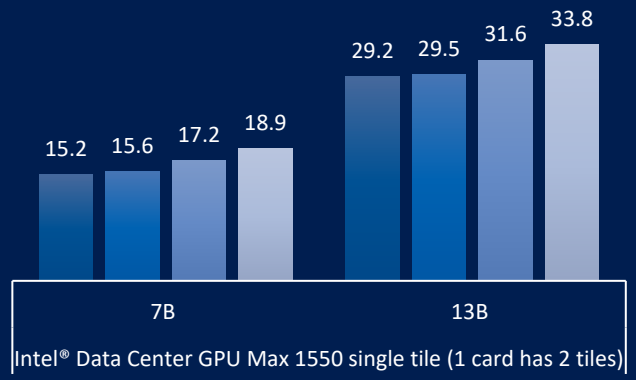
LLaMA2 7B & 13B Inference

Llama 2 Next Token Latency (Lower is Better)

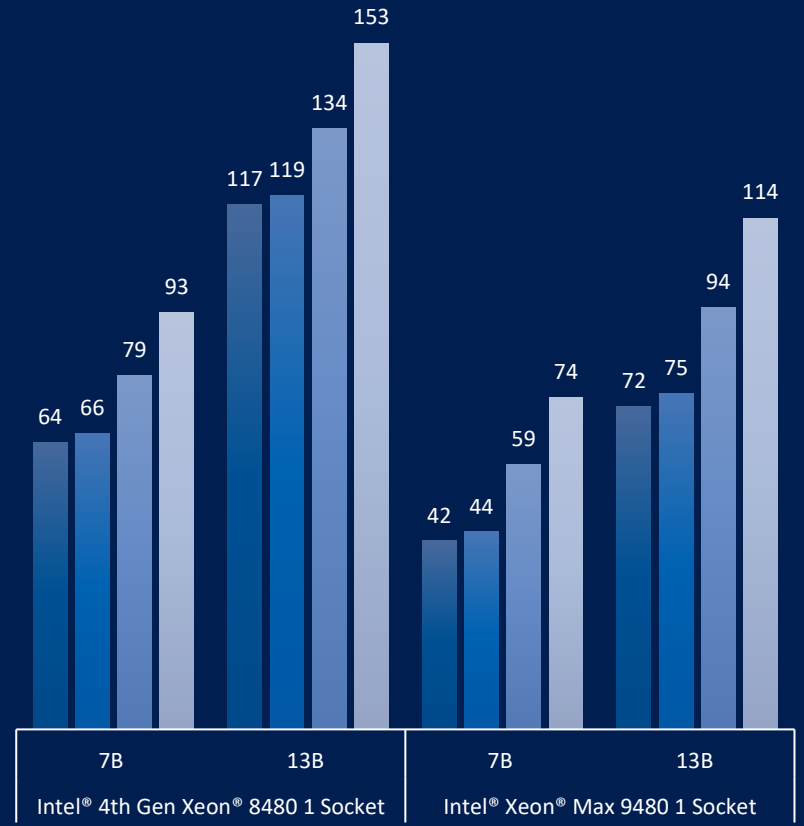
Greedy Mode, Mixed Precision (bfloat16), BS = 1, 256 Output Tokens



On 1 Tile (out of 2 tiles per card) Intel® Data Center GPU Max 1550



On 1 Socket Intel® Xeon® Scalable Processor



Intel® Gaudi2 ®AI Accelerator

Intel Data Center GPU Max 1550

Intel Xeon Scalable Processor

intel Ai summit

For configuration details see: https://habana.ai/habana-claims-validation/ and www.intel.com/PerformanceIndex

Software Optimizations

Optimization Methodologies

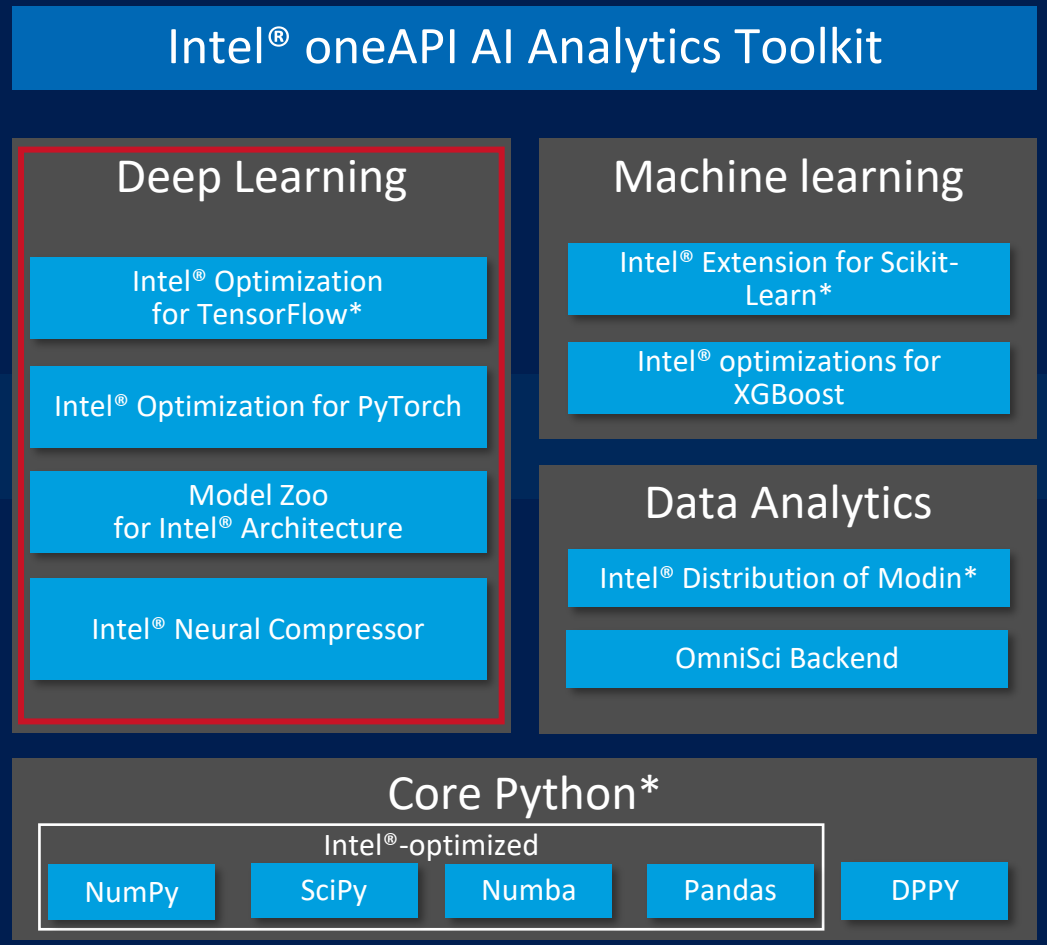
intel Ai summit

Intel® oneAPI AI Analytics Toolkit

Accelerates end-to-end Machine Learning and Data Analytics pipelines with frameworks and libraries optimized for Intel® architectures

Who Uses It?

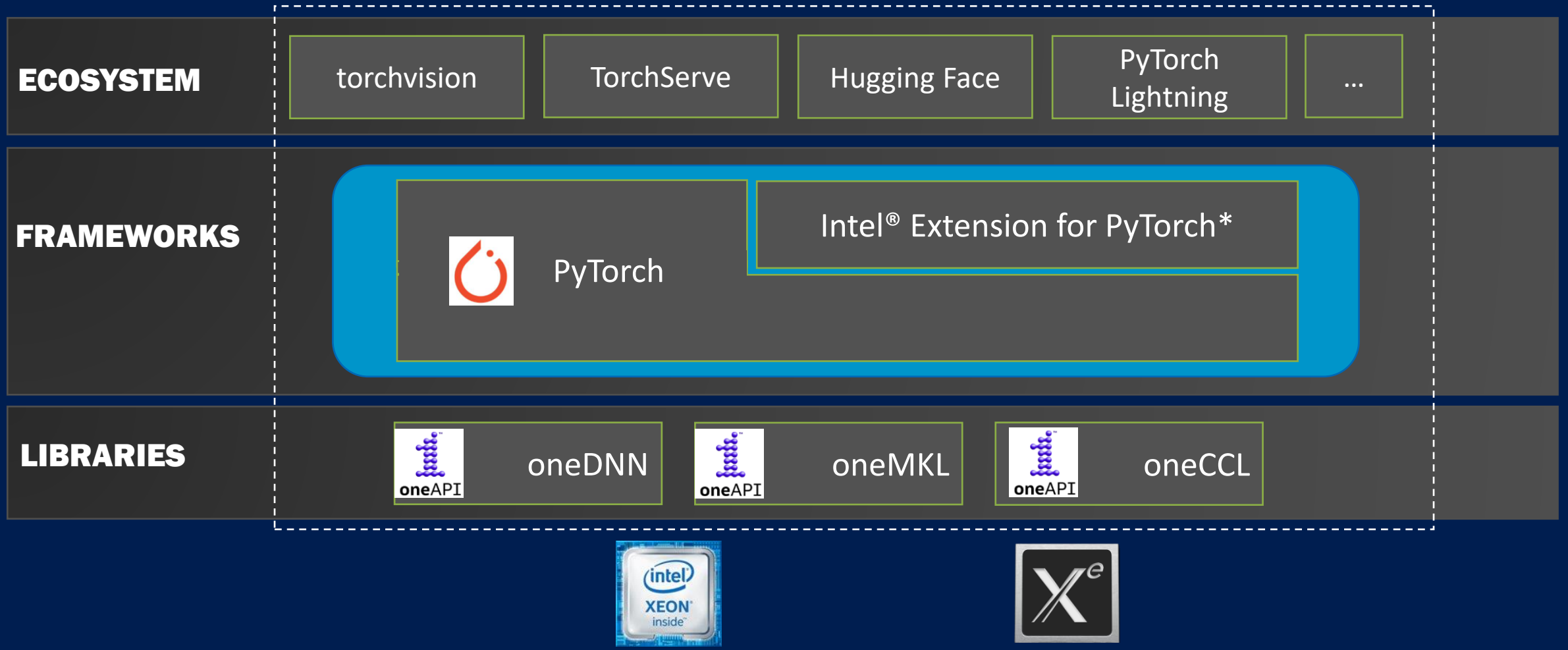
Data scientists, AI Researchers, Machine and Deep Learning developers, AI application developers



Learn More: Intel®.com/oneAPI-AIKit

intel AI summit

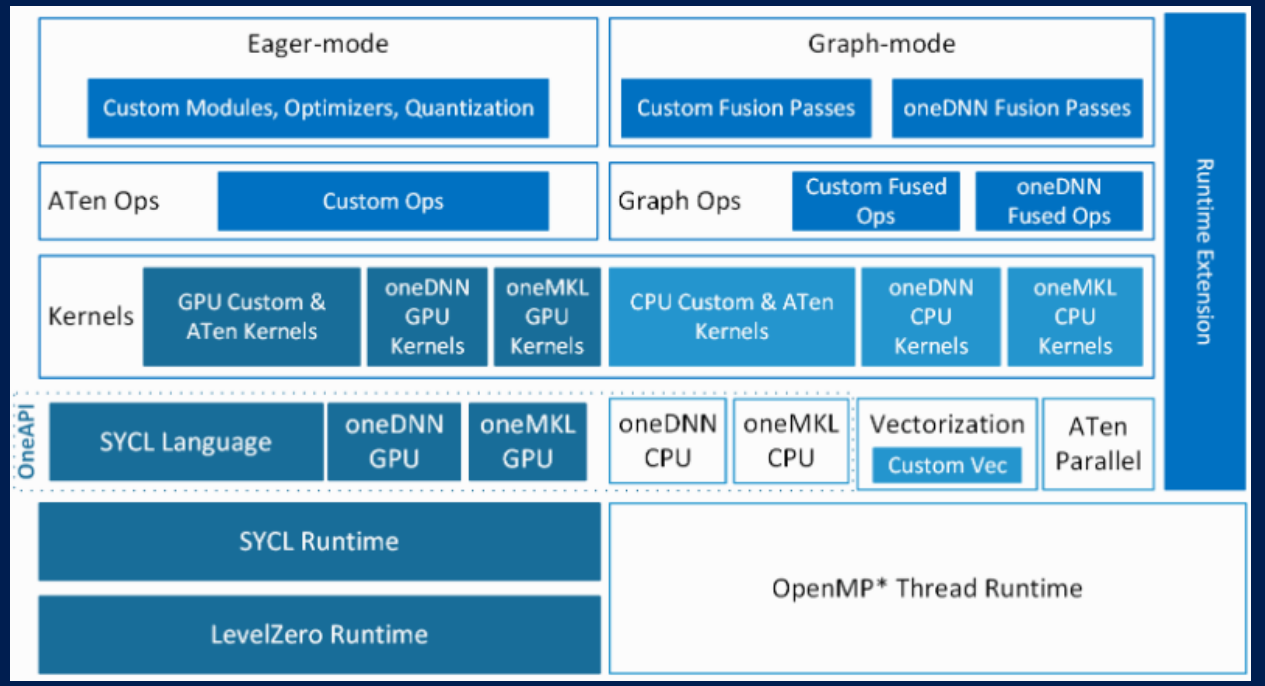
Intel® Optimization for PyTorch*



Overview

- 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**
- oneDNN is available.
- AMX automatically enabled with oneDNN.
- Dynamically linked in CPP executables.

Intel® Extension for PyTorch*



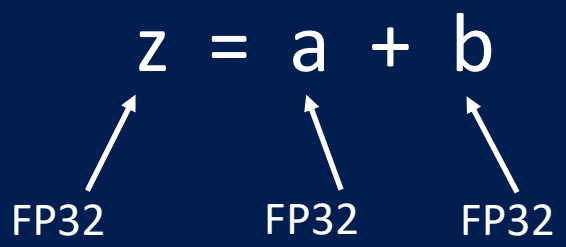
CPU: Code and Documentation
GPU: Code and Documentation

Major Optimization Methodologies

- General performance optimization and Intel new feature enabling in PyTorch upstream
- Additional performance boost and early adoption of aggressive optimizations through Intel® Extension for PyTorch*

Operator Optimization	Graph Optimization	Runtime Extension
• Vectorization • Parallelization • Memory Layout • Low Precision	• Operator fusion • Constant folding	• Thread affinity • Memory allocation • Customized execution

Vectorization

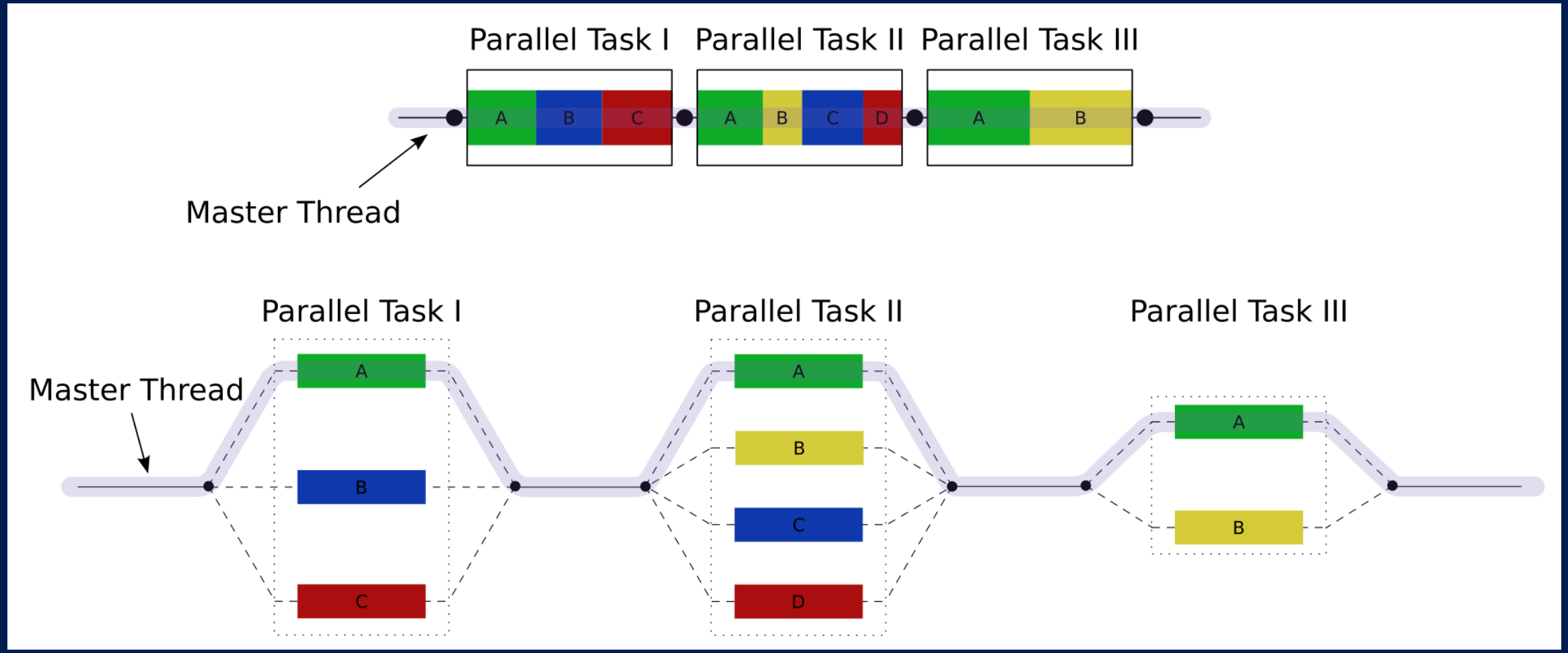


ISA	Length	Num of FP32
AVX	128 bits	4
AVX2	256 bits	8
AVX512	512 bits	16

zmm2	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
				+				
zmm3	b[0]	b[1]	b[2]	b[3]	b[4]	b[5]	b[6]	b[7]
				=				
zmm1	a[0]+b[0]	a[1]+b[1]	a[2]+b[2]	a[3]+b[3]	a[4]+b[4]	a[5]+b[5]	a[6]+b[6]	a[7]+b[7]

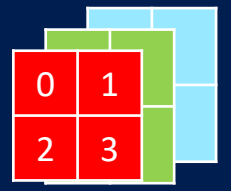
https://www.Intel®.com/content/www/us/en/developer/articles/technical/improve-performance-with-vectorization.html

Parallelization

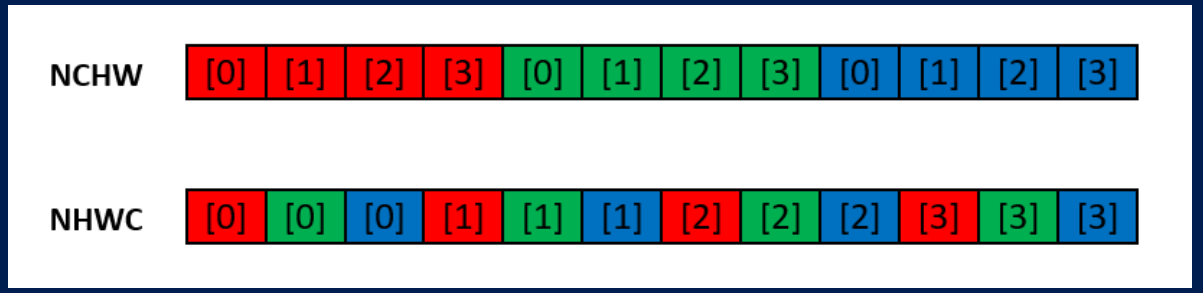


https://en.wikipedia.org/wiki/OpenMP

Memory Layout



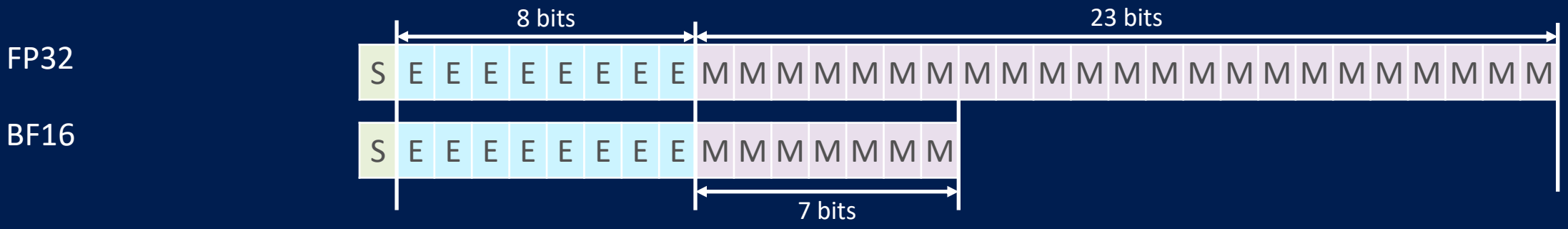
- Used mainly in image workloads
- NCHW (PyTorch default)
  - *torch.contiguous_format*
- **NHWC**
  - *torch.channels_last*
  - NHWC format yields higher performance on Intel® hardware
- For GPU, move the input and model to "xpu" before converting to channels last
  - *input = input.to("xpu")*
  - *model = model.to("xpu")*




```
## NB: internally blocked format will still be used.
##   aka. we do 'reorder' for 'input', 'weight' and 'output',
##   and believe me this is expensive, roughly 50% perf loss...
input = torch.randn(1, 10, 32, 32)
model = torch.nn.Conv2d(10, 20, 1, 1)
output = model(input)
```

```
input = torch.randn(1, 10, 32, 32)
model = torch.nn.Conv2d(10, 20, 1, 1)
## NB: convert to Channels Last memory format.
##   oneDNN supports NHWC for feature maps (input, output),
##   but weight still needs to be of blocked format.
##   Still we can save reorders for feature maps.
input = input.to(memory_format=torch.channels_last)
model = model.to(memory_format=torch.channels_last)
output = model(input)
```

Low-precision Optimization – BF16



BF16 has the same range as FP32 but less precision 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

Training w/AMX BF16 on CPU

```python
import torch
import torchvision
import intel_extension_for_pytorch as ipex

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
)
```

```python
model = torchvision.models.resnet50()
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr = LR, momentum=0.9)
model.train()
model, optimizer = ipex.optimize(model, optimizer=optimizer, dtype=torch.bfloat16)

for batch_idx, (data, target) in enumerate(train_loader):
  optimizer.zero_grad()
  with torch.cpu.amp.autocast():
    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 w/AMX BF16 on CPU

Resnet50

```
import torch
import torchvision.models as models

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

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

with torch.no_grad(), torch.cpu.amp.autocast():
  model = torch.jit.trace(model, torch.rand(1, 3, 224, 224))
  model = torch.jit.freeze(model)

  model(data)
```

BERT

```
import torch
from transformers import BertModel

model = BertModel.from_pretrained("bert-base-uncased")
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 ####################
import intel_extension_for_pytorch as ipex
model = ipex.optimize(model, dtype=torch.bfloat16)
########################################################

with torch.no_grad(), torch.cpu.amp.autocast():
  d = torch.randint(vocab_size, size=[batch_size, seq_length])
  model = torch.jit.trace(model, (d,), check_trace=False, strict=False)
  model = torch.jit.freeze(model)

  model(data)
```

Training with Intel® Extension for PyTorch* (GPU)

```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

```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* (GPU)

*The .to(“xpu”) is needed for GPU

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)
```

intel AI summit

Low-precision Optimization – INT8

What is Quantization?

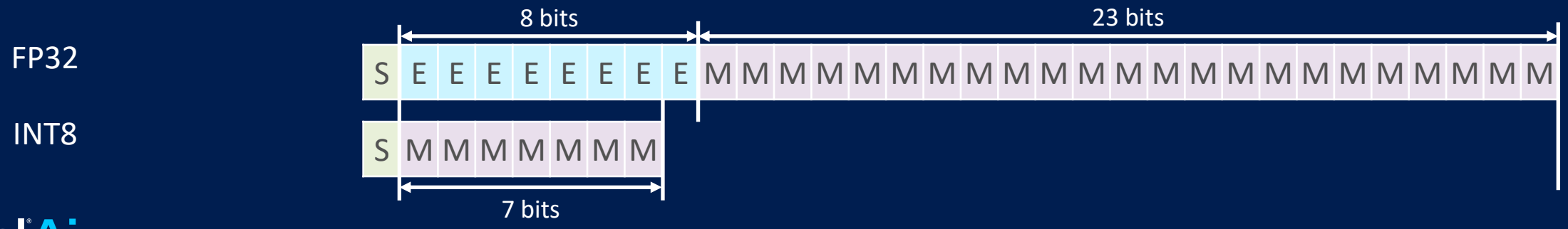
- An approximation method
- The process of mapping values from a large set (e.g., continuous, FP64/FP32) to those with smaller set (e.g., countable, BF16, INT8)

How to Quantize?

- PyTorch quantization
- https://pytorch.org/docs/stable/quantization.html

Why Quantization?

- Significant performance increase with similar accuracy



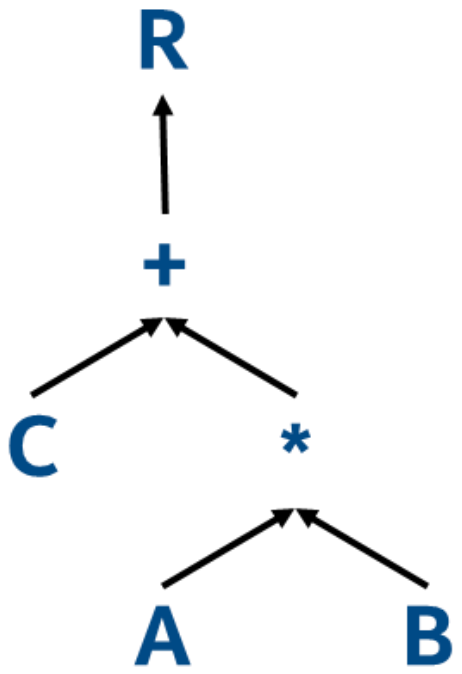
intel AI summit

Quantization Types

Quantization Mode			Dataset Requirement	Works Best For	Accuracy	Notes
Post Training Quantization	Dynamic/Weight Only Quantization	activation dynamically quantized (fp16, int8) or not quantized, weight statically quantized (fp16, int8, in4)	None	LSTM, MLP, Embedding, Transformer	good	Easy to use, close to static quantization when performance is compute or memory bound due to weights
	Static Quantization	activation and weights statically quantized (int8)	calibration dataset	CNN	good	Provides best perf, may have big impact on accuracy, good for hardwares that only support int8 computation
Quantization Aware Training	Dynamic Quantization	activation and weight are fake quantized	fine-tuning dataset	MLP, Embedding	best	Limited support for now
	Static Quantization	activation and weight are fake quantized	fine-tuning dataset	CNN, MLP, Embedding	best	Typically used when static quantization leads to bad accuracy, and used to close the accuracy gap

https://pytorch.org/docs/stable/quantization.html

Operator Fusion




```
for (i in 1:n)
    temp[i, 1] = A[i, 1] * B[i, 1]

for (i in 1:n)
    R[i, 1] = C[i, 1] + temp[i, 1]
```

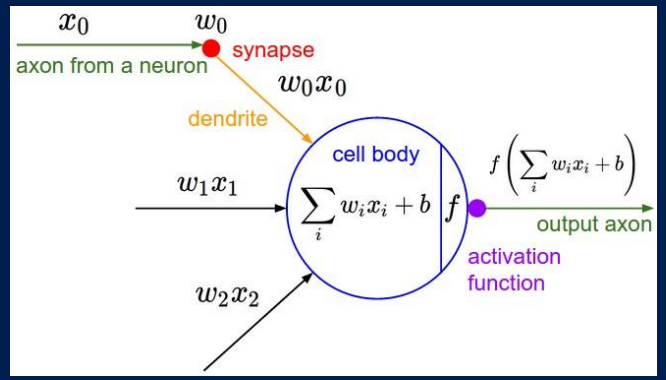
```
for (i in 1:n)
    R[i, 1] = C[i, 1] + (A[i, 1] * B[i, 1])
```

FP32 & BF16 Fusion Patterns

- Conv(2, 3)D + ReLU
- Conv(2, 3)D + SUM
- Conv(2, 3)D + SUM + ReLU
- Conv(2, 3)D + Sigmoid
- Conv(2, 3)D + Sigmoid + MUL
- Conv(2, 3)D + HardTanh
- Conv(2, 3)D + SiLU
- Conv(2, 3)D + ELU
- Linear + ReLU
- Linear + GELU
- …

Constant Folding

Binary Folding (ADD/SUB/MUL/DIV)



$y = W \times x + b$

+

$y' = y + \beta$

OR

$y' = y \times \beta$

$y' = y + \beta$ | $y' = y \times \beta$

$y' = W \times x + b + \beta$ OR $y' = (W \times x + b) \times \beta$

$y' = W \times x + (b + \beta)$ | $y' = (W \times \beta) \times x + (b \times \beta)$

$y' = W' \times x + b'$ | $y' = W' \times x + b'$

$W' = W$ OR $W' = W \times \beta$

$b' = b + \beta$ | $b' = b \times \beta$

TorchScript and TorchDynamo

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

TorchDynamo – in BETA

- Makes PyTorch **code** run faster by just-in-time (JIT)-compiling PyTorch code into optimized kernels

How to Check AMX is Actually Used

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

How to Check if Your Hardware Supports AMX

- On bash terminal, enter the following command:
  - *cat /proc/cpuinfo*
- Check the “flags” section for amx_bf16, amx_int8
- Alternatively, you can use:
  - *lscpu | grep amx*
- If you do not see them, consider upgrading to Linux kernel 5.17 and above

```
Flags:                fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse s
se2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpu
id aperfmperf tsc_known_freq pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 s
se4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault epb cat_l3 cat_
l2 cdp_l3 invpcid_single intel_ppin cdp_l2 ssbd mba ibrs ibpb stibp ibrs_enhanced tpr_shadow vnmi flexpriority ept vpid ept_
ad fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms invpcid rtm cqm rdt_a avx512f avx512dq rdseed adx smap avx512ifma clflus
hopt clwb intel_pt avx512cd sha_ni avx512bw avx512vl xsaveopt xsavec xgetbv1 xsaves cqm_llc cqm_occup_llc cqm_mbm_total cqm_
mbm_local split_lock_detect avx_vnni avx512_bf16 wbnoinvd dtherm ida arat pln pts hwp hwp_act_window hwp_epp hwp_pkg_req hfi
 avx512vbmi umip pku ospke waitpkg avx512_vbmi2 gfni vaes vpclmulqdq avx512_vnni avx512_bitalg tme avx512_vpopcntdq la57 rdp
id bus_lock_detect cldemote movdiri movdir64b enqcmd fsrm uintr avx512_vp2intersect md_clear serialize tsxldtrk pconfig arch
_lbr amx_bf16 avx512_fp16 amx_tile amx_int8 flush_l1d arch_capabilities
```

oneDNN Verbose Sample Output (CPU)

Sample oneDNN Verbose Output

```
onednn_verbose,info,oneDNN v2.6.0 (commit 52b5f107dd9cf10910aaa19cb47f3abf9b349815)
onednn_verbose,info,cpu,runtime:OpenMP,nthr:32
onednn_verbose,info,cpu,isa:Intel AVX-512 with Intel DL Boost
onednn_verbose,info,gpu,runtime:none
onednn_verbose,info,prim_template:timestamp,operation,engine,primitive,implementation,prop_kind,memory_descriptors,attributes,auxiliary,problem_desc,exec_time
onednn_verbose,1678917979730.501953,exec,cpu,reorder,jit:uni,undef,src_f32::blocked:abcd:f0 dst_f32:p:blocked:Acdb16a:f0,attr-scratchpad:user ,,1x1x1x37,0.00292969
onednn_verbose,1678917979730.888916,exec,cpu,convolution,jit:avx512_core,forward_training,src_f32::blocked:abcd:f0 wei_f32:p:blocked:Acdb16a:f0 bia_undef::undef::f0 dst_f3
onednn_verbose,1678917979732.105957,exec,cpu,reorder,jit:uni,undef,src_f32:p:blocked:aBcd16b:f0 dst_f32::blocked:abcd:f0,attr-scratchpad:user ,,1x1x1x48000,0.0649414
onednn_verbose,1678917980009.694092,exec,cpu,reorder,jit:uni,undef,src_f32::blocked:abc:f0 dst_f32::blocked:acb:f0,attr-scratchpad:user ,,1x60x305,0.00878906
onednn_verbose,1678917980011.387939,exec,cpu,convolution,brgconv:avx512_core,forward_training,src_f32::blocked:acb:f0 wei_f32::blocked:Acb32a:f0 bia_f32::blocked:a:f0 dst_
onednn_verbose,1678917980012.134033,exec,cpu,reorder,jit:uni,undef,src_f32::blocked:abc:f0 dst_f32::blocked:acb:f0,attr-scratchpad:user ,,1x1024x301,0.278076
onednn_verbose,1678917980012.912109,exec,cpu,reorder,simple:any,undef,src_f32:p:blocked:Acb48a:f0 dst_f32::blocked:Acb64a:f0,attr-scratchpad:user ,,1024x1024x1,3.31201
```

- Note the ISA. For AMX, you should see the following:
  - *Intel AMX with bfloat16 and 8-bit integer support*
- Check for AMX in the primitive implementation:

```
onednn_verbose,1673049613345.454102,exec,cpu,convolution,brgconv:avx512_core_amx_bf16,forward_training,src_bf16::blocked:acdb:f0 wei_
onednn_verbose,1673049613348.691895,exec,cpu,convolution,brgconv_1x1:avx512_core_amx_bf16,forward_training,src_bf16::blocked:acdb:f0
onednn_verbose,1673049613353.259033,exec,cpu,convolution,brgconv_1x1:avx512_core_amx_bf16,forward_training,src_bf16::blocked:acdb:f0
onednn_verbose,1673049613364.104980,exec,cpu,convolution,brgconv_1x1:avx512_core_amx_bf16,forward_training,src_bf16::blocked:acdb:f0
```

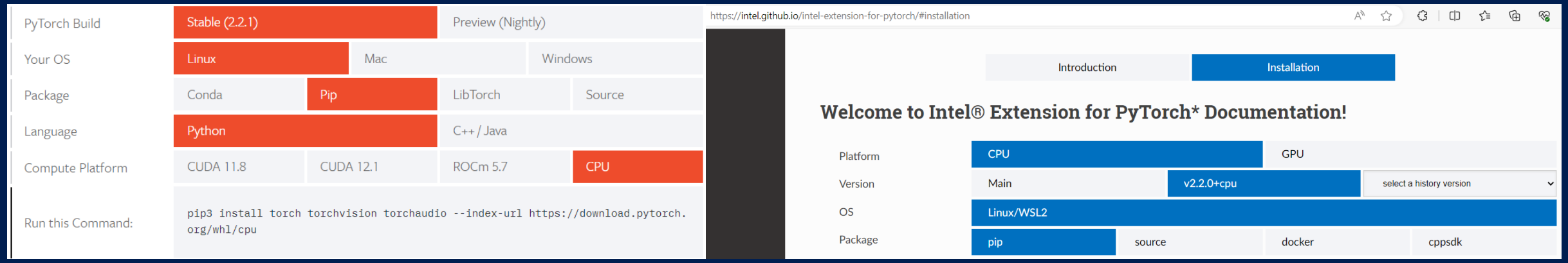
oneDNN Verbose Sample Output (GPU)

```
onednn_verbose,info,oneDNN v3.2.0 (commit 67bc621a2da4aefc51f0a59b2af2398fa1d3e1c8)
onednn_verbose,info,cpu,runtime:threadpool,nthr:56
onednn_verbose,info,cpu,isa:Intel AVX-512 with float16, Intel DL Boost and bfloat16 support and Intel AMX with bfloat16 and 8-bit integer support
onednn_verbose,info,gpu,runtime:DPC++
onednn_verbose,info,gpu,engine,0,backend:Level Zero,name:Intel(R) Data Center GPU Max 1100,driver_version:1.3.26516,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,1692211521687.472900,exec,gpu,convolution,jit:ir,forward_training,src_bf16::blocked:acdb::f0 wei_bf16::blocked:acdb::f0 bia_bf16::blocked:a::f0
onednn_verbose,1692211521710.104004,exec,gpu,convolution,jit:ir,forward_training,src_bf16::blocked:acdb::f0 wei_bf16::blocked:acdb::f0 bia_bf16::blocked:a::f0
onednn_verbose,1692211521724.636963,exec,gpu,convolution,jit:ir,forward_training,src_bf16::blocked:acdb::f0 wei_bf16::blocked:acdb::f0 bia_bf16::blocked:a::f0
onednn_verbose,1692211521738.939941,exec,gpu,convolution,jit:ir,forward_training,src_bf16::blocked:acdb::f0 wei_bf16::blocked:acdb::f0 bia_bf16::blocked:a::f0
onednn_verbose,1692211521743.134033,exec,gpu,convolution,jit:ir,forward_training,src_bf16::blocked:acdb::f0 wei_bf16::blocked:acdb::f0 bia_bf16::blocked:a::f0
onednn_verbose,1692211521750.906982,exec,gpu,convolution,jit:ir,forward_training,src_bf16::blocked:acdb::f0 wei_bf16::blocked:acdb::f0 bia_bf16::blocked:a::f0
onednn_verbose,1692211521755.149902,exec,gpu,convolution,jit:ir,forward_training,src_bf16::blocked:acdb::f0 wei_bf16::blocked:acdb::f0 bia_bf16::blocked:a::f0
onednn_verbose,1692211521755.489990,exec,gpu,convolution,jit:ir,forward_training,src_bf16::blocked:acdb::f0 wei_bf16::blocked:acdb::f0 bia_bf16::blocked:a::f0
onednn_verbose,1692211521755.853027,exec,gpu,convolution,jit:ir,forward_training,src_bf16::blocked:acdb::f0 wei_bf16::blocked:acdb::f0 bia_bf16::blocked:a::f0
onednn_verbose,1692211521756.153076,exec,gpu,convolution,jit:ir,forward_training,src_bf16::blocked:acdb::f0 wei_bf16::blocked:acdb::f0 bia_bf16::blocked:a::f0
onednn_verbose,1692211521756.447021,exec,gpu,convolution,jit:ir,forward_training,src_bf16::blocked:acdb::f0 wei_bf16::blocked:acdb::f0 bia_bf16::blocked:a::f0
onednn_verbose,1692211521765.226074,exec,gpu,convolution,jit:ir,forward_training,src_bf16::blocked:acdb::f0 wei_bf16::blocked:acdb::f0 bia_bf16::blocked:a::f0
onednn_verbose,1692211521779.264893,exec,gpu,convolution,jit:ir,forward_training,src_bf16::blocked:acdb::f0 wei_bf16::blocked:acdb::f0 bia_bf16::blocked:a::f0
onednn_verbose,1692211521792.895996,exec,gpu,convolution,jit:ir,forward_training,src_bf16::blocked:acdb::f0 wei_bf16::blocked:acdb::f0 bia_bf16::blocked:a::f0
onednn_verbose,1692211521804.466064,exec,gpu,convolution,jit:ir,forward_training,src_bf16::blocked:acdb::f0 wei_bf16::blocked:acdb::f0 bia_bf16::blocked:a::f0
onednn_verbose,1692211521821.544922,exec,gpu,convolution,jit:ir,forward_training,src_bf16::blocked:acdb::f0 wei_bf16::blocked:acdb::f0 bia_bf16::blocked:a::f0
onednn_verbose,1692211521835.277100,exec,gpu,convolution,jit:ir,forward_training,src_bf16::blocked:acdb::f0 wei_bf16::blocked:acdb::f0 bia_bf16::blocked:a::f0
onednn_verbose,1692211521839.224121,exec,gpu,convolution,jit:ir,forward_training,src_bf16::blocked:acdb::f0 wei_bf16::blocked:acdb::f0 bia_bf16::blocked:a::f0
onednn_verbose,1692211521839.625000,exec,gpu,convolution,jit:ir,forward_training,src_bf16::blocked:acdb::f0 wei_bf16::blocked:acdb::f0 bia_bf16::blocked:a::f0
onednn_verbose,1692211521839.928955,exec,gpu,convolution,jit:ir,forward_training,src_bf16::blocked:acdb::f0 wei_bf16::blocked:acdb::f0 bia_bf16::blocked:a::f0
onednn_verbose,1692211521840.221924,exec,gpu,convolution,jit:ir,forward_training,src_bf16::blocked:acdb::f0 wei_bf16::blocked:acdb::f0 bia_bf16::blocked:a::f0
onednn_verbose,1692211521840.562012,exec,gpu,convolution,jit:ir,forward_training,src_bf16::blocked:acdb::f0 wei_bf16::blocked:acdb::f0 bia_bf16::blocked:a::f0
```

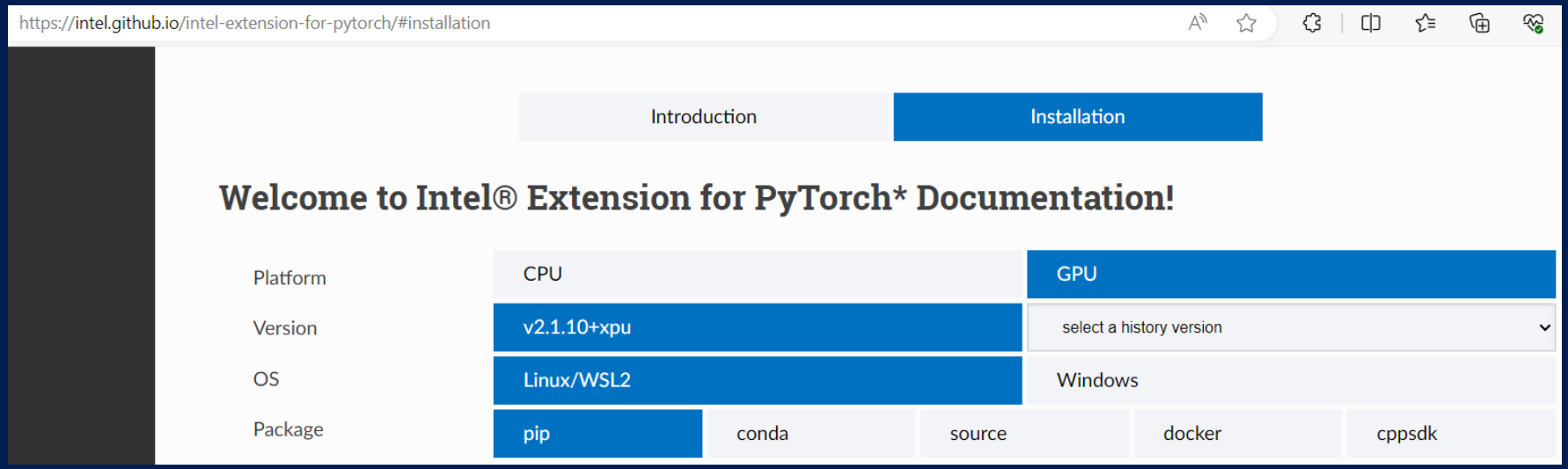
Installation, Code Samples

Getting Started

intel Ai summit



https://pytorch.org/get-started/locally/



https://intel.github.io/intel-extension-for-pytorch/cpu/latest/tutorials/installation.html

*Linux Kernel 5.17 required for AMX

** For TensorFlow, use v2.9.10 or newer for optimizations

intel Ai summit

Model Zoo for Intel® Architecture

Available on GitHub

- https://github.com/intelAI/models/tree/master

Runs out-of-the-box for benchmarking

PyTorch use cases

- Image Recognition, Image Segmentation, Language Modeling/Translation, Object Detection, Recommendation, Text-to-Speech, Shot Boundary Detection, AI Drug Design
- Supported on dGPU Flex and Max Series: training and inference on ResNet50v1.5, SSD-MobileNet, Yolo V4, Bert Large
- Hugging Face: Fine-tuning Stable Diffusion Models on Intel CPUs

Image Recognition

Model	Framework	Mode	Model Documentation	Benchmark/Test Dataset
DenseNet169	TensorFlow	Inference	FP32	ImageNet 2012
Inception V3	TensorFlow	Inference	Int8 FP32	ImageNet 2012
Inception V4	TensorFlow	Inference	Int8 FP32	ImageNet 2012
MobileNet V1*	TensorFlow	Inference	Int8 FP32 BFloat16	ImageNet 2012
ResNet 101	TensorFlow	Inference	Int8 FP32	ImageNet 2012
ResNet 50	TensorFlow	Inference	Int8 FP32	ImageNet 2012
ResNet 50v1.5	TensorFlow	Inference	Int8 FP32 BFloat16 dGPU Int8	ImageNet 2012
ResNet 50v1.5 Sapphire Rapids	TensorFlow	Inference	Int8 FP32 BFloat16	ImageNet 2012
ResNet 50v1.5	TensorFlow	Training	FP32 BFloat16	ImageNet 2012
Inception V3	TensorFlow Serving	Inference	FP32	Synthetic Data
ResNet 50v1.5	TensorFlow Serving	Inference	FP32	Synthetic Data
GoogLeNet	PyTorch	Inference	FP32 BFloat16	ImageNet 2012
Inception v3	PyTorch	Inference	FP32 BFloat16	ImageNet 2012
MNASNet 0.5	PyTorch	Inference	FP32 BFloat16	ImageNet 2012
MNASNet 1.0	PyTorch	Inference	FP32 BFloat16	ImageNet 2012
ResNet 50	PyTorch	Inference	FP32 BFloat16	ImageNet 2012
ResNet 50	PyTorch	Training	FP32 BFloat16	ImageNet 2012
ResNet 101	PyTorch	Inference	FP32 BFloat16	ImageNet 2012
ResNet 152	PyTorch	Inference	FP32 BFloat16	ImageNet 2012
ResNext 32x4d	PyTorch	Inference	FP32 BFloat16	ImageNet 2012
ResNext 32x16d	PyTorch	Inference	FP32 BFloat16	ImageNet 2012
VGG-11	PyTorch	Inference	FP32 BFloat16	ImageNet 2012
VGG-11 with batch normalization	PyTorch	Inference	FP32 BFloat16	ImageNet 2012
Wide ResNet-50-2	PyTorch	Inference	FP32 BFloat16	ImageNet 2012
Wide ResNet-101-2	PyTorch	Inference	FP32 BFloat16	ImageNet 2012
ResNet 50 v1.5	PyTorch	Inference	dGPU Int8	ImageNet 2012

Useful Links

- Intel® Extension for PyTorch GitHub
  - CPU: Code, Documentation, Contributing
  - GPU: Code, Documentation, Contributing
  - Issues
- oneAPI-samples GitHub
- Model Zoo for Intel® Architecture GitHub
- Sapphire Rapids Performance Tuning Guide
- 4th Gen Xeon Info
  - Intel® Xeon® Scalable Processors
  - 4th Gen Intel® Xeon® Scalable Processors
  - 4th Gen Intel® Xeon® Scalable Processor product brief
  - Intel® Accelerator Engines
  - Software for 4th gen Intel Xeon Scalable and Intel® Xeon® Max Series

PyTorch/TensorFlow Benchmarking Configurations

4th Generation Intel® Xeon® Scalable Processors

Hardware and software configuration (measured October 24, 2022):

- **Deep Learning config:**
  - Hardware configuration for Intel® Xeon® Platinum 8480+ processor (formerly code named Sapphire Rapids): 2 sockets, 56 cores, 350 watts, 16 x 64 GB DDR5 4800 memory, BIOS version EGSDCRB1.SYS.8901.P01.2209200243, operating system: CentOS* Stream 8, using Intel® Advanced Matrix Extensions (Intel® AMX) int8 and bf16 with Intel® oneAPI Deep Neural Network Library (oneDNN) v2.7 optimized kernels integrated into Intel® Extension for PyTorch* v1.13, Intel® Extension for TensorFlow* v2.12, and Intel® Distribution of OpenVINO™ toolkit v2022.3. Measurements may vary.
  - Wall power refers to platform power consumption.
  - If the dataset is not listed, a synthetic dataset was used to measure performance. Accuracy (if listed) was validated with the specified dataset.
- **Transfer Learning config:**
  - Hardware configuration for Intel® Xeon® Platinum 8480+ processor (formerly code named Sapphire Rapids): Use DLSA single node fine tuning, Vision Transfer Learning using single node, 56 cores, 350 watts, 16 x 64 GB DDR5 4800 memory, BIOS version EGSDREL1.SYS.8612.P03.2208120629, operating system: Ubuntu 22.04.1 LT, using Intel® Advanced Matrix Extensions (Intel® AMX) int8 and bf16 with Intel® oneAPI Deep Neural Network Library (oneDNN) v2.6 optimized kernels integrated into Intel® Extension for PyTorch* v1.12, and Intel® oneAPI Collective Communications Library v2021.5.2. Measurements and some software configurations may vary.

3rd Generation Intel® Xeon® Scalable Processors

Hardware and software configuration (measured October 24, 2022):

- Hardware configuration for Intel® Xeon® Platinum 8380 processor (formerly code named Ice Lake): 2 sockets, 40 cores, 270 watts, 16 x 64 GB DDR5 3200 memory, BIOS version SE5C620.86B.01.01.0005.2202160810, operating system: Ubuntu 22.04.1 LTS, int8 with Intel® oneAPI Deep Neural Network Library (oneDNN) v2.6.0 optimized kernels integrated into Intel® Extension for PyTorch* v1.12, Intel® Extension for TensorFlow* v2.10, and Intel® oneAPI Data Analytics Library (oneDAL) 2021.2 optimized kernels integrated into Intel® Extension for Scikit-learn* v2021.2. XGBoost v1.6.2, Intel® Distribution of Modin* v0.16.2, Intel oneAPI Math Kernel Library (oneMKL) v2022.2, and Intel® Distribution of OpenVINO™ toolkit v2022.3. Measurements may vary.
- If the dataset is not listed, a synthetic dataset was used to measure performance. Accuracy (if listed) was validated with the specified dataset.

*All performance numbers are acquired running with 1 instance of 4 cores per socket

Notices and Disclaimers

For notices, disclaimers, and details about performance claims, visit www.intel.com/PerformanceIndex or scan the QR code:

Thank You!