03--TorchServe with Intel Extension for PyTorch

参考链接：

- TorchServe with Intel Extension for PyTorch

目录：

1. Install Intel Extension for PyTorch
2. Serving model with Intel Extension for PyTorch
3. TorchServe with Launcher
4. Creating and Exporting INT8 model for IPEX
5. Benchmarking with Launcher
6. Performance Boost with IPEX and Launcher

1、Install Intel Extension for PyTorch

Refer to the installation documentation.

2、Serving model with Intel Extension for PyTorch

安装后，通过在 `config.properties` 中启用TorchServe来使用IPEX。

```
ipex_enable=true
```

一旦启用IPEX，部署PyTorch模型的程序与 PyTorch用例文档 中所示的相同。带有IPEX的TorchServe可以部署任何模型并进行推理。

3、TorchServe with Launcher

Launcher 是一个脚本，用于在英特尔硬件上自动调整配置设置以提高性能。调整 OMP_NUM_THREADS、线程亲和性和内存分配器等配置会对性能产生显着影响。有关详细信息，请参阅 性能调整指南 和 性能调整启动脚本 文档。

通过在 `config.properties` 中设置配置来启用带有启动器的 TorchServe。

在 config.properties 中添加以下行以使用具有默认配置的启动器：

```
ipex_enable=true
cpu_launcher_enable=true
```

如果安装了 `numactl`，Launcher 会使用 numactl 以确保套接字被固定，从而从本地 numa 节点分配内存。要使用没有 numactl 的启动器，请在 `config.properties` 中添加以下行。

```
ipex_enable=true
cpu_launcher_enable=true
cpu_launcher_args=--disable_numactl
```

Launcher 默认只使用非超线程内核来避免内核计算资源共享。要将启动器与物理和逻辑（超线程）的所有内核一起使用，请在 `config.properties` 中添加以下行：

```
ipex_enable=true
cpu_launcher_enable=true
cpu_launcher_args=--use_logical_core
```

下面是一个将多个参数传递给 cpu_launcher_args 的示例：

```
ipex_enable=true
cpu_launcher_enable=true
cpu_launcher_args=--use_logical_core --disable_numactl
```

需要注意的一些有用的 cpu_launcher_args 是：

1. Memory Allocator: [PTMalloc `--use_default_allocator` | *TCMalloc* `--enable_tcmalloc` | JeMalloc `--enable_jemalloc`]
    - PyTorch 默认使用 PTMalloc。 TCMalloc/JeMalloc 通常提供更好的性能。
2. OpenMP library: [GNU OpenMP `--disable_iomp` | *Intel OpenMP*]
    - PyTorch默认使用GNU OpenMP。Launcher默认使用Intel OpenMP。英特尔OpenMP库通常能提供更好的性能。
3. Node id: [`--node_id`]
    - 启动器默认使用所有物理核心。将内存访问限制在第N个套接字上的本地内存，以避免非统一内存访问（NUMA）。

有关启动器可调配置的完整列表，请参阅性能调整启动脚本。

一些值得注意的启动器配置是：

1. `--ninstances`:

    多实例推理/训练的实例数。

2. `--instance_idx`:

    启动器在运行多个实例时默认运行所有实例。指定 instance_idx 在实例中运行单个实例。这在独立运行每个实例时很有用。

Creating and Exporting INT8 model for IPEX

英特尔® PyTorch* 扩展支持 Eager 和 Torchscript 模式。在本节中，我们将展示如何为 IPEX 部署 INT8 模型。

1、Creating a serialized file

首先使用 IPEX INT8 推理创建 .pt 序列化文件。在这里，我们展示了 BERT 和 ResNet50 的两个示例。

BERT

```
import torch
import intel_extension_for_pytorch as ipex
import transformers
from transformers import AutoModelForSequenceClassification, AutoConfig

# load the model
config = AutoConfig.from_pretrained(
    "bert-base-uncased", return_dict=False, torchscript=True, num_labels=2)
model = AutoModelForSequenceClassification.from_pretrained(
    "bert-base-uncased", config=config)
model = model.eval()

# define dummy input tensor to use for the model's forward call to record operations in
the model for tracing
N, max_length = 1, 384
dummy_tensor = torch.ones((N, max_length), dtype=torch.long)

# calibration
# ipex supports two quantization schemes to be used for activation:
torch.per_tensor_affine and torch.per_tensor_symmetric
# default qscheme is torch.per_tensor_affine
conf = ipex.quantization.QuantConf(qscheme=torch.per_tensor_affine)
n_iter = 100
with torch.no_grad():
    for i in range(n_iter):
        with ipex.quantization.calibrate(conf):
            model(dummy_tensor, dummy_tensor, dummy_tensor)

# optionally save the configuration for later use
# save:
# conf.save("model_conf.json")
# load:
# conf = ipex.quantization.QuantConf("model_conf.json")

# conversion
jit_inputs = (dummy_tensor, dummy_tensor, dummy_tensor)
model = ipex.quantization.convert(model, conf, jit_inputs)

# save to .pt
torch.jit.save(model, 'bert_int8_jit.pt')
```

ResNet50

```python
import torch
import torch.fx.experimental.optimization as optimization
import intel_extension_for_pytorch as ipex
import torchvision.models as models

# load the model
model = models.resnet50(pretrained=True)
model = model.eval()
model = optimization.fuse(model)

# define dummy input tensor to use for the model's forward call to record operations in the model for tracing
N, C, H, W = 1, 3, 224, 224
dummy_tensor = torch.randn(N, C, H, W).contiguous(memory_format=torch.channels_last)

# calibration
# ipex supports two quantization schemes to be used for activation: torch.per_tensor_affine and torch.per_tensor_symmetric
# default qscheme is torch.per_tensor_affine
conf = ipex.quantization.QuantConf(qscheme=torch.per_tensor_symmetric)
n_iter = 100
with torch.no_grad():
    for i in range(n_iter):
        with ipex.quantization.calibrate(conf):
           model(dummy_tensor)

# optionally save the configuration for later use
# save:
# conf.save("model_conf.json")
# load:
# conf = ipex.quantization.QuantConf("model_conf.json")

# conversion
jit_inputs = (dummy_tensor)
model = ipex.quantization.convert(model, conf, jit_inputs)

# save to .pt
torch.jit.save(model, 'rn50_int8_jit.pt')
```

2、Creating a Model Archive

创建序列化文件 (.pt) 后，它可以像往常一样与 torch-model-archiver 一起使用。使用以下命令打包模型。

```
$ torch-model-archiver --model-name rn50_ipex_int8 --version 1.0 --serialized-file rn50_int8_jit.pt --handler image_classifier
```

3、Start TorchServe to serve the model

确保在 config.properties 中设置 ipex_enable=true。使用以下命令以 IPEX 启动 TorchServe。

```
$ torchserve --start --ncs --model-store model_store --ts-config config.properties
```

4、Registering and Deploying model

Registering and deploying the model follows the same steps shown in the PyTorch Use Case documentation.

Bechmarking with Launcher

Launcher 可与 TorchServe 官方基准测试一起使用，以在英特尔硬件上以最佳配置启动服务器和基准测试请求。

在本节中，我们提供了使用启动器及其默认配置进行基准测试的示例。

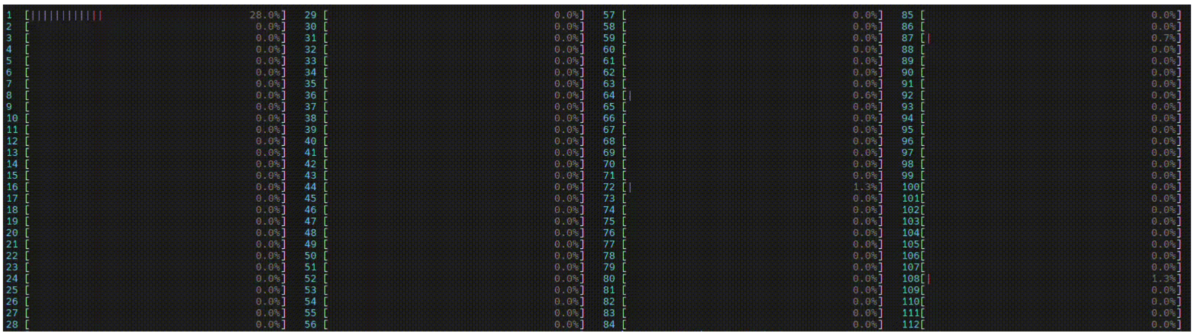
将以下行添加到基准目录中的 config.properties 以使用具有默认设置的启动器。

```
ipex_enable = true
cpu_launcher_enable = true
```

基准测试的其余步骤与 基准测试文档 中显示的步骤相同。

`model_log.log` 包含用于此执行启动的信息和命令。

使用 Intel(R) Xeon(R) Platinum 8180 CPU，2 个插槽，每个插槽 28 个内核，每个内核 2 个线程的机器上的 CPU 使用率如下所示：



```
$ cat logs/model_log.log

2021-12-01 21:22:40,096 - __main__ - WARNING - Both TCMalloc and JeMalloc are not found
in $CONDA_PREFIX/lib or $VIRTUAL_ENV/lib or /.local/lib/ or /usr/local/lib/ or
/usr/local/lib64/ or /usr/lib or /usr/lib64 or /home/<user>/.local/lib/ so the
LD_PRELOAD environment variable will not be set. This may drop the performance
2021-12-01 21:22:40,096 - __main__ - INFO - OMP_NUM_THREADS=56
2021-12-01 21:22:40,096 - __main__ - INFO - Using Intel OpenMP
2021-12-01 21:22:40,096 - __main__ - INFO - KMP_AFFINITY=granularity=fine,compact,1,0
2021-12-01 21:22:40,096 - __main__ - INFO - KMP_BLOCKTIME=1
2021-12-01 21:22:40,096 - __main__ - INFO - LD_PRELOAD=<VIRTUAL_ENV>/lib/libiomp5.so
2021-12-01 21:22:40,096 - __main__ - WARNING - Numa Aware: cores:[0, 1, 2, 3, 4, 5, 6,
7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28,
29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
51, 52, 53, 54, 55] in different NUMA node
```

Performance Boost with IPEX and Launcher

上面显示了 Torchserve 在 ResNet50 和 BERT-base-uncased 上使用 IPEX 和启动器的性能改进。使用 Amazon EC2 m6i.24xlarge 上的 Torchserve 官方 apache-bench 基准测试来收集结果。在 config.properties 中添加以下行以重现结果。请注意，启动器配置为单个实例使用单个套接字上的所有物理内核，以避免跨套接字通信和内核重叠。

```
ipex_enable=true
cpu_launcher_enable=true
cpu_launcher_args=--node_id 0 --ninstance 1 --enable_jemalloc
```

Use the following command to reproduce the results.

```
$ python benchmark-ab.py --url {modelUrl} --input {inputPath} --concurrency 1
```

例如，运行以下命令重现 ResNet50 的延迟性能，数据类型为 IPEX int8，batch size 为 1。

```
$ python benchmark-ab.py --url 'file:///model_store/rn50_ipex_int8.mar' --concurrency 1
```

例如，运行以下命令重现 BERT 的延迟性能，数据类型为 IPEX int8，批量大小为 1。

```
$ python benchmark-ab.py --url 'file:///model_store/bert_ipex_int8.mar' --input
'../examples/Huggingface_Transformers/Seq_classification_artifacts/sample_text_captum_i
nput.txt' --concurrency 1
```