PYTORCH
AMPERE® OPTIMIZED
FRAMEWORK
Documentation

V.1.3.0

AMPERE COMPUTING

Table of Contents

RELEASE NOTES

V1.3.0:

- Binary integer operations support.
- New operators supported: Reshape, Squeeze, Unsqueeze, Flatten, PixelShuffle, GroupNorm, InstanceNorm.
- Using custom compiled OpenBLAS, as Pytorch BLAS backend.
- Bug fixes

V1.2.0:

- Ampere Optimized PyTorch updated to 0.3.0
- New optimized operators: Gelu, Silu, Softmax, Div, Binary ops between Tensor and Scalar, Permute, View, Layer Norm, Size, Pow, Tanh, Sigmoid
- Improved Concat support
- Graph optimizations
- Various bugfixes

V1.1.0:

- Ampere Optimized PyTorch updated to 0.2.1
- Batch Matmul supported (enhancing DLRM performance)
- Adaptive Avg Pool supported
- LeakyRelu supported
- AIO_NUM_THREADS no longer needed to set Ampere Optimized PyTorch threads, inherits Pytorch intra-op thread count.

OVERVIEW

Ampere Optimized PyTorch inference acceleration engine is fully integrated with the PyTorch framework. PyTorch models and software written with the PyTorch API can run as-is, without modifications.

PYTORCH FRAMEWORK

Python is installed with Ampere Optimized PyTorch and all dependencies. No additional installation steps are needed.

Versions Compatibility

This release is based on Pytorch 1.11.0 and comes with the compatible Torchvision 0.12.0 installed.

PYTHON

Pytorch 1.11.0 is built for Python 3.8. Regarding other Python versions, please contact your Ampere sales representative. If you are using the software through a third party, contact their customer support team for help. You can also contact the AI team at ai-support@amperecomputing.com.

CONFIGURATIONS

Ampere Optimized PyTorch inference engine can be configured by a set of environment variables for performance and debugging purposes. They can be set in the command line when running Pytorch models (e.g., AIO_NUM_THREADS=16 python run.py -p fp32) or set in the shell initialization script.

AIO_PROCESS_MODE

This variable controls whether the Ampere Optimized PyTorch inference engine is used to run the Pytorch model:

- 0: disabled.
- 1: enabled (Default).

AIO_CPU_BIND

Enables core binding. If enabled, each Ampere Optimized PyTorch thread will bind itself to a single core:

- 0: Core binding disabled.
- 1: Core binding enabled (Default).

AIO_MEM_BIND

Binds memory to NUMA (Non-uniform memory access) node 0. For optimal performance, numactl (https://linux.die.net/man/8/numactl) is preferred. numactl bind will affect both the Pytorch framework and the optimized framework buffers, while the optimized framework is unable to affect buffers allocated by the Pytorch framework:

- 0: Membind disabled.
- 1: Membind to node 0 (Default).

AIO_NUMA_CPUS

Select the cores that Ampere Optimized PyTorch should bind to (if CPU_BIND is enabled):

- Not set: use the first N cores of the machine, excluding hyper-threaded (Default).
- Set: use N first cores from the list of cores for N threads. The list is in space separated, 0-based number format. For example, selecting cores 0 to 1: AIO_NUMA_CPUS="0 1".

AIO_DEBUG_MODE

Control the verbosity of debug messages:

- 0: No messages
- 1: Errors only
- 2: Basic information, warnings, and errors (Default)
- 3: Most messages
- 4: All messages

QUICKSTART

The following instructions run on Altra/Altra Max Linux machines installed **with Docker**. To initialize Ampere Optimized PyTorch environment run:

```
$ wget -O aio-pytorch.tar.gz "<your_unique_url>"
$ docker load < aio-pytorch.tar.gz
$ docker run --privileged=true --rm --name pytorch-aio --network host -it aio-pytorch-1.11.0:1.3.0
```

Skip the above steps if running **without a Docker** container.

You can try Ampere Optimized PyTorch by either running the Jupyter Notebook examples or Python scripts on the CLI level.

To run the Jupyter Notebook QuickStart examples follow the instructions below:

Set AIO_NUM_THREADS to the requested value first.

```
$ export AIO_NUM_THREADS=16; export OMP_NUM_THREADS=16
$ cd /workspace/aio-examples/
$ bash start_notebook.sh
```

If you run the Jupyter Notebook Quickstart on a cloud instance, make sure your machine has port 8080 open and on your local device run:

```
$ ssh -N -L 8080:localhost:8080 -I <ssh_key> your_user@xxx.xxx.xxx.xxx
```

Use a browser to point to the URL printed out by the Jupyter Notebook launcher.

You will find Jupyter Notebook examples (examples.ipynb) under the /classification and /object detection folders.

The examples run through several inference models, visualize results they produce, and present the performance numbers.

To use CLI-level scripts:

Set AIO_NUM_THREADS to the requested value first.

```
$ export AIO_NUM_THREADS=16; export OMP_NUM_TREADS=16
$ cd /workspace/aio-examples/
```

Go to the directory of choice, e.g.

```
$ cd classification/resnet_50_v1
```

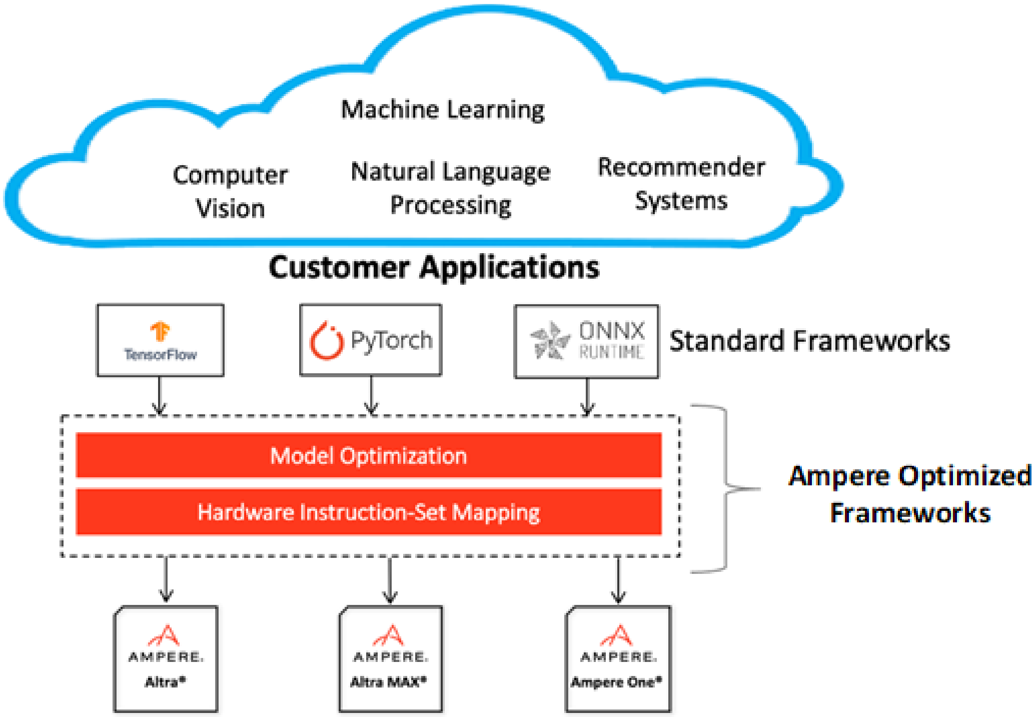
Evaluate the model.

```
$ numactl --physcpubind=0-15 python3 run.py -p fp32
```

AMPERE OPTIMIZED PYTORCH PROGRAMMING GUIDE

Overview

Ampere Optimized PyTorch is powered by Ampere® AI backend that accelerates Deep Learning (DL) operations on the Ampere® Altra family of processors. Ampere Optimized PyTorch accelerates DL operations through model optimization, highly vectorized compute kernels and multi-thread operations that are automatically tuned to deliver the best latency and throughput on Ampere Altra processors. It delivers 2-5x gains over alternative backend solutions.



Supported Inference Ops

Ampere Optimized Pytorch accelerates most common Pytorch ops that are used in various types of models. Here is a list of accelerated ops and formats (Note: non-accelerated ops will still run without a problem, at the original framework operator speed):

Layer	FP32	Notes
Conv2d	Y	
Linear	Y	
MaxPool2d	Y	
AvgPool2d	Y	
AdaptiveAvgPool2d	Y	
Relu	Y	
Relu6	Y	
LeakyRelu	Y	
Softmax	Y	
Gelu	Y	
Silu	Y	
Sigmoid	Y	
Tanh	Y	
Transpose	Y	
Permute	Y	
BatchNorm	Y	
LayerNorm	Y	
GroupNorm	Y	
InstanceNorm	Y	
Add	Y	Int version not optimized
Mul	Y	Int version not optimized
Div	Y	Int version not optimized
Pow	Y	Int version not optimized
Matmul	Y	
MM	Y	

BMM	Y	
PixelShuffle	Y	
View	Y	
Reshape	Y	
Squeeze	Y	
Unsqueeze	Y	
Flatten	Y	
Contiguous	Y	
Size	Y	One dimension case

PyTorch JIT Trace

While Pytorch Eager Execution provides excellent model building, programming, and debugging experience, it is slower than graph execution. So, Torchscript is typically used for inference deployment. In the current version of Ampere Optimized Pytorch, only Torchscript mode is accelerated.

To use Ampere Optimized Pytorch, conversion of Pytorch module to Torchscript is needed. There are two ways to convert: torch.jit.script() or torch.jit.trace(input) API calls. See https://pytorch.org/docs/stable/jit.html for more details. After converting to Torchscript user should call torch.jit.freeze() to freeze the models and enable model optimizations for inference.

Threading

Ampere Optimized PyTorch controls the number of Ampere Optimized Pytorch intra_op threads with torch.set_num_threads(). This controls both the number of threads used for ops delegated to Ampere Optimized Pytorch as well as the ops running on default CPU backend.

Some default CPU backend ops (non-AIO) also need to set OMP_NUM_THREADS environment variable to control the intra_op threads.

To correctly switch between Ampere Optimized Pytorch and Pytorch thread pools we recommend setting following environmental variables to ensure best performance:

```
OMP_WAIT_POLICY=ACTIVE
GOMP_SPINCOUNT=10000
KMP_BLOCKTIME=1
```

Programming Tips

- In the first two inference passes, Ampere Optimized Pytorch performs runtime compilation of PyTorch script and prepares Ampere Optimized Pytorch network. So, the latency of the first two pass is expected to be longer. Subsequent passes will be accelerated.
- Ampere Optimized PyTorch provides much better latency scaling as core count increase, comparing to other platforms. You can easily try the optimal number of cores with the above

set_num_threads() function that can give you the best price / performance, while meeting your latency requirements.

- If any issues occur, Ampere AI team is ready to help. Typically, the first step is to get more debug logs and send it to ai-support@amperecomputing.com. Please set environment variable AIO_DEBUG_MODE=5 to capture low level logs.

We can also provide more in-depth profiling of your model to help enhancing performance to meet your needs.

Ampere Computing® / 4655 Great America Parkway, Suite 601 / Santa Clara, CA 95054 /
www.amperecomputing.com