ONNX RUNTIME AMPERE® OPTIMIZED FRAMEWORK Documentation

V.1.4.0

AMPERE COMPUTING

Table of Contents

RELEASE NOTES

v1.4.0:

- Backend updated to 0.5.1
    - Misc models speed up
    - Bug fixes

v1.3.0:

- ONNX Runtime updated to v1.11.1.
- Backend updated to 0.4.0
    - Misc models speed up
    - Bug fixes

v1.2.0:

v1.1.0:

- Updated to use Ampere Optimized ONNX Runtime 0.3.0
    - Misc models speed up

OVERVIEW

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

ONNX RUNTIME FRAMEWORK

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

Versions Compatibility

This release is based on ONNX Runtime 1.10.0. Please refer to ONNX Runtime version compatibility documentation, found at ONNX Runtime and ONNX Versioning Guide, to check the compatibility of models built with older versions of ONNX Runtime.

PYTHON

ONNX Runtime 1.10.0 is built for Python 3.8. Regarding other Python versions, 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 Ampere AI team at ai-support@amperecomputing.com.

CONFIGURATIONS

Ampere Optimized ONNX Runtime 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 ONNX models (e.g., AIO_NUM_THREADS=16 python run.py) or set in the shell initialization script.

AIO_PROCESS_MODE

This variable controls whether Ampere Optimized ONNX Runtime inference engine is used to run the ONNX model:

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

AIO_CPU_BIND

Enables core binding. If enabled, each Ampere Optimized ONNX Runtime 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 ONNX Runtime framework and framework and the optimized framework buffers, while the optimized framework is unable to affect buffers allocated by the ONNX Runtime framework:

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

AIO_NUMA_CPUS

Select cores that Ampere Optimized ONNX Runtime should bind to (if CPU_BIND is enabled):

- Not set: use the first N cores of the machine, excluding hyper-threaded machines (Default).
- Set: try to 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_NUM_THREADS

Specifies the number of cores that Ampere Optimized ONNX Runtime should use:

- Not set: use one core (Default).
- "all": use all cores, as specified by AIO_NUMA_CPUS.
- N: use N cores.

AIO_DEBUG_MODE

Control 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**. When you are already using a virtual machine pre-installed with the version of Ampere Optimized ONNX Runtime (e.g. on a cloud service provider) that you need, you can skip the following step of launching Docker container.

Launching Docker Container

There are two ways you can obtain the docker image. You will either be provided with an URL that you can download the Docker image tarball, or you will be provided with a link to pull down the docker image from a Docker image repository.

Downloading Docker Image Tarball

```
$ wget -O aio-onnxrt.tar.gz "<your_unique_url>"
$ docker load < aio-onnxrt.tar.gz
```

Pulling Docker Image from repository

```
$ docker pull <docker image repository link>
```

Launching Docker Container

```
$docker run --privileged=true --rm --name onnxrt-aio --network host -it ghcr.io/amperecomputingai/release-aio-onnxrt:1.4.0
```

Running Examples

You can try Ampere Optimized ONNX Runtime 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 download_models.sh
$ 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 the 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_THREADS=16
$ cd /workspace/aio-examples/
$ bash download_models.sh
$ pip install torch
```

Go to the directory of choice, e.g.,

```
$ cd classification/resnet_50_v1.5
```

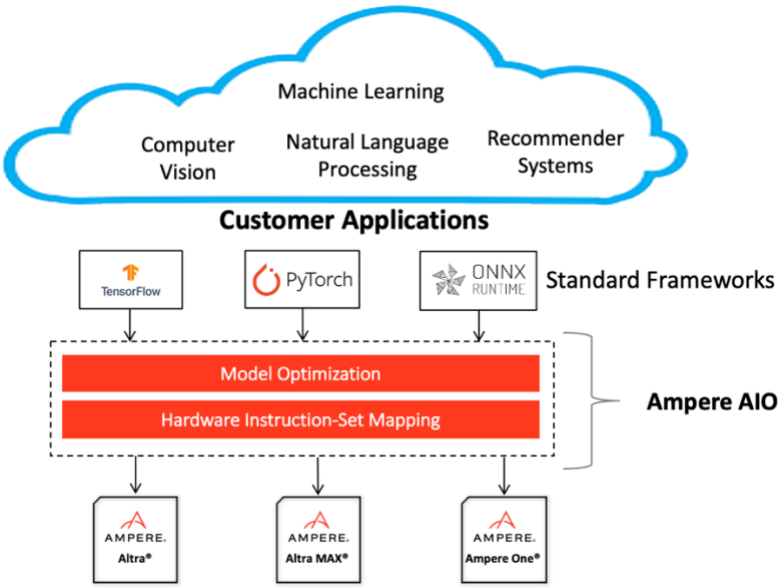
Evaluate the model.

```
$ python3 run.py -m resnet_50_v1.5_fp32.onnx -p fp32
```

AMPERE OPTIMIZED FRAMEWORKS PROGRAMMING GUIDE

Overview

Ampere Optimized ONNX Runtime is powered by Ampere AIO backend that accelerates Deep Learning (DL) operations on Ampere Altra family processors. AIO 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 ONNX Runtime accelerates most common ONNX Runtime 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 problem, at the original framework operator speed):

	FP32	FP16	Remarks
Conv	Y	Y	
MatMul	Y	Y	Input B constant only
MaxPool	Y	Y	
GlobalMaxPool	Y	Y	
AveragePool	Y	Y	
GlobalAveragePool	Y	Y	
Relu	Y	Y	
LeakyRelu	Y	Y	
BatchNormalization	Y	Y	
Softmax	Y	Y	axis == the last dimension only
Concat	Y	Y	
Add	Y	Y	
Sub	Y	Y	
Pow	Y	Y	
Mul	Y	Y	
Div	Y	Y	
Sqrt	Y	Y	
Exp	Y	Y	
Tanh	Y	Y	
Log	Y	Y	
Clip	Y	Y	Min == 0.0 && max == 6.0 only
Squeeze	Y	N	FP32 or INT32 input only
Unsqueeze	Y	N	FP32 or INT32 input only
Sum			
Transpose	Y	N	FP32 or INT32 input only Perm necessary
Scatter	Y	N	FP32 or INT32 input only
LRN			
Gemm	Y	Y	Input C num dims < 2 only Input C constant only transA == 0 only
Cast	Y	Y	
Reshape	Y	N	Shape input constant only FP32 or INT32 only
Shape	Y	Y	
Gather	Y	N	FP32 or INT32 input only

Ampere AI continues to expand the coverage of ONNX Runtime ops. If your model has any op that is not listed in the table or custom ops that need acceleration, please contact ai-support@amperecomputing.com.

Threading

Ampere Optimized ONNX Runtime controls the number of intra_op threads of AIO with onnxruntime.SessionOptions().intra_op_num_threads. This controls both the number of threads used for ops delegated to AIO 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 ONNX Runtime and ONNX Runtime 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 inference pass, AIO performs runtime compilation of ONNX graphs. So, the latency of the first pass is expected to be longer. Subsequent passes will be accelerated.
- Ampere Optimized ONNX Runtime 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 intra_op_num_threads configurations that can give you the best price / performance, while meeting your latency requirements.
- (Experimental): Ampere Optimized ONNX Runtime backend now provides automatic FP16 operator conversion that can boost the performance of your FP32 model on-the-fly. It automatically performs FP16 conversion and computation for certain whitelisted operators through regular expression. To take advantage of that, you can set environment variable.
    - $export AIO_IMPLICIT_FP16_TRANSFORM_FILTER=".*"
    - This activates automatic FP16 conversion for all supported operators.
    - It is estimated that this has very little impact to accuracy of common models. Please contact us if you have any question about this feature.

- 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.

AMP 2019-0039