How to compile model and run inference on Coral Edge TPU using STM32MP1

Stable: 08.02.2021 - 12:42 / Revision: 04.02.2021 - 08:08

A quality version of this page, approved on *8 February 2021*, was based off this revision.

Contents

1 Article purpose

This article aims at describing how to run an inference on Coral Edge TPU using STM32MP1 microprocessor devices.

Information

This article provides a simple example. Other methods exist that might be better adapted to your development constraints. Feel free to explore them.

2 Prerequisites

2.1 Installing the Edge TPU compiler

To perform an inference on the Coral Edge TPU hardware, we need to convert our TensorFlow Lite model into an Edge TPU model. This can be achieved by using the Edge TPU compiler. To do this, install the Edge TPU compiler on your host computer.

Information

The Edge TPU compiler is provided only for 64-bit architectures. Thus, it is not possible to install it on the STM32MP1 target board. Besides, you must ensure that your host computer uses a Debian-based Linux system.

Install the Coral Edge TPU compiler by running the following commands on your host computer:

```
 PC $> curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
 PC $> echo "deb https://packages.cloud.google.com/apt coral-edgetpu-stable main" | sudo
tee /etc/apt/sources.list.d/coral-edgetpu.list
 PC $> sudo apt-get update
 PC $> sudo apt-get install edgetpu-compiler
```

Enter the following command to check that the compiler has been installed:

```
 PC $> edgetpu_compiler -v
 Edge TPU Compiler version 2.x.y
```

3 Fetching a Coral Edge TPU compiled model

3.1 Coral compiled models

Coral [1] offers a wide set of quantized and compiled models for demonstration purposes. Ready-to-use models are available from https://coral.ai/models/.

3.2 Compiling your own model

The Edge TPU compiler role is to convert one or several TensorFlow Lite models into Edge TPU compatible models. It takes as an argument your *.tflite* model and returns a *.tflite* Edge TPU compatible model. You can pass multiple models as arguments (each separated with a space). They are then co-compiled and share the Edge TPU 8 Mbytes of RAM for parameter data caching.

Be aware that not all the operations supported on TensorFlow Lite are supported on Edge TPU. While building your own model architecture, check the operations and the layers supported by the Edge TPU compiler on https://coral.ai/docs/edgetpu/models-intro/#supported-operations.

If your model architecture uses unsupported operations and does not meet all the requirements, then only the first portion of the model will be executed on the Edge TPU compiler. Starting from the first node in your model graph where an unsupported operation occurs, all operations are run on the CPU of the target board even if an Edge TPU supported operation occurs. The Edge TPU compiler cannot partition the model more than once.

Information

If an important percentage of your model is executed on the CPU, you have to expect a significant degradation of the inference speed compared to a model that is entirely executed on the Edge TPU compiler. To achieve a maximal speed, use only Edge TPU supported operations in your model

To compile your *.tflite* model, execute the following command:

```
PC $> edgetpu_compiler your_model_1.tflite your_model_1.tflite ...
```

If you do not specify the *-out_dir* option, the compiled model is saved in the current directory with the *input_filename_edgetpu.tflite* name. Another *.log* file that provides information about data caching and memory consumption is stored in the Edge TPU RAM.

3.3 Example: compiling an object detection model

The object detection model used is the *ssd_mobilenet_v1_coco_quant.tflite* downloaded from the **Coral** [1] website and compiled for the Coral Edge TPU using the steps detailed below:

```
PC $> wget http://storage.googleapis.com/download.tensorflow.org/models/tflite
/coco_ssd_mobilenet_v1_1.0_quant_2018_06_29.zip
PC $> unzip ./coco_ssd_mobilenet_v1_1.0_quant_2018_06_29.zip
 Archive:  ./coco_ssd_mobilenet_v1_1.0_quant_2018_06_29.zip
           inflating: detect.tflite
           inflating: labelmap.txt
```

Now that the model has been downloaded and extracted successfully, it is time to compile it using the Edge TPU compiler:

```
PC $> edgetpu_compiler detect.tflite
```

3.4 Sending your model to the target

In your board workspace directory, create two main directories to organize our workflow:

```
Board $> cd /usr/local && mkdir -p workspace
Board $> cd /usr/local/workspace && mkdir -p models
```

Then transfer your compiled model from the **host computer** to the *models* directory in the **board** workspace:

```
PC $> scp path/to/your/compiled_model_edgetpu.tflite root@<board_ip_address>:/usr/local
/workspace/models/
```

Now that your workspace is ready with the compiled model file, it is time to see how to run an inference using the C++ benchmark application.

4 Running the inference

4.1 Installing the benchmark application

Configure the AI OpenSTLinux package, then install X-LINUX-AI components for this application:

Warning

The software package is provided AS IS, and by downloading it, you agree to be bound to the terms of the software license agreement (SLA). The detailed content licenses can be found here.

```
Board $> apt-get install tflite-edgetpu-benchmark
```

4.2 Execute the benchmark on the model

Now that our compiled model is loaded on the board, it is time to run an inference using the benchmark example. This example aims at measuring your model average inference time on a predefined number of 25 loops. To do this, execute the following command:

```
Board $> cd /usr/local/demo-ai/benchmark/tflite-edgetpu/
Board $> ./tflite_edgetpu_benchmark -m /usr/local/workspace/models/your_compiled_model.
tflite -l 25
```

The first inference may take longer since the model is being loaded on the Coral Edge TPU RAM. This time is not taken into account in the average inference time.

4.3 Customizing your application

You can adapt your application to your development constraints and requirements.

To build a prototype of your application using Python, go through Image classification Python example or Object detection Python example.

To run your application using the C++ API, refer to the Image classification C++ example or Object detection C++ example.

5 References

- [1.01.1] Coral AI

Linux® is a registered trademark of Linus Torvalds.

Random Access Memory (Early computer memories generally hadserial access. Memories where any given address can be accessed when desired were then called "random access" to distinguish them from the memories where contents can only be accessed in a fixed order. The term is used today for volatile random-acces ssemiconductor memories.)

Central processing unit

Artificial Intelligence

Application programming interface