Application Note 001

ONNC Utilities

This application note lists current ONNC utilities. This document is compliant to the ONNC Community Docker image v1.0. You may download the Docker image from Docker Hub[1].

[1] https://hub.docker.com/r/onnc/onnc-community/

Table of content

1. Overview

An ONNC Docker image is available in Docker Hub for fast deployment. It includes a pre-built ONNC source tree cloned from the ONNC/onnc[2] GitHub repository, pre-installed dependent libraries, and ready-to-run working environment. Users may run benchmarks from the ONNX Model Zoo[3] easily with this Docker image. For those who like to modify ONNC source code for research or product development, this document describes a couple of available utilities and scripts that are helpful in a common workflow using ONNC. The covered topics include the Docker image, built-in unit tests, benchmarking and the detailed steps to rebuild modified source code.

2. Prerequisite

If Docker is not installed in your system, please download Docker[4] and install it first.

3. Get Docker Image

Pull the Docker image from Docker Hub using the following shell command:

```
$ docker pull onnc/onnc-community
```

4. Build ONNC with the Docker Image

Although Docker image includes a source code tree, it might not be the latest release version of ONNC. We strongly suggest you clone the latest version of ONNC from the GitHub repository, mount the source code directory to the Docker image, and modify the source code with your favorite editor on your host machine. You may clone the source code from the GitHub ONNC repository[5]. To download large model files when cloning the ONNC source, you have to install Git LFS[6] first.

```
$ mkdir -p <source_dir> && cd <source_dir>
$ git clone https://github.com/ONNC/onnc.git
```

Once the latest source code is ready, you may invoke the following command to enter the ONNC build environment:

[2] https://github.com/ONNC/onnc

[3] https://github.com/onnx/models

[4] http://www.docker.com

[5] https://github.com/ONNC/onnc

[6] https://github.com/git-lfs/git-lfs/wiki/Installation

```
$ docker run -ti –rm --cap-add=SYS_PTRACE -v <source_dir>/onnc:/onnc/onnc onnc/onnc-
community
```

<source_dir> is the directory where you cloned the latest ONNC source code. The *-v* option mounts the directory to the Docker image, and the *--cap-add=SYS_PTRACE* option enables debug support (e.g. gdb) in the container. You can make some change to the source code (**<source_dir>/onnc**) and run the following command to build ONNC.

```
// run in the container cli, under build directory '/onnc/onnc-umbrella/build-
normal' by default
$ smake -j8 install
```

The ***smake*** command synchronizes the build directory with ***<source_dir>/onnc*** and invokes the make command to build ONNC. The ***-j8*** option is to parallelize building with 8 cores. This command will automatically install the compiled binary in this container environment.

Configure Build Environment

The ONNC project uses CMake as its building system. There is a pre-defined build directory ***build-normal*** in the container and you may create another build variant (e.g. for debugging ONNC tools) using the following command:

```
// run in the container cli
$ cd /onnc/onnc-umbrella
$ ssync && ./build.cmake.sh <build_mode>
```

<build_mode> supports one of the following modes:

```
normal, dbg, rgn, opt
```

The ***ssync*** command synchronizes the build directory with the ***<source_dir>/onnc*** directory. The ***build.cmake.sh*** script creates a directory named ***build-<build_mode>*** for generated files. The mapping between ***<build_mode>*** and CMake build type variable ***CMAKE_BUILD_TYPE*** is listed as below.

Build Mode	CMAKE_BUILD_TYPE
Normal	Release
dbg	Debug
rgn	Regression
opt	Optimized

Table 1. CMAKE_BUILD_TYPE and build mode mapping

5. Run Unit Tests

There are 18 unit tests available in the ONNC repository. Those tests are written in C language and you may run all of them using the following shell command:

```
// run in the container cli
$ ctest
```

If all tests pass, you will see the same output as shown below.

```
Test project /onnc/onnc-umbrella/build-normal
      Start  1: Digraph
 1/18 Test  #1: Digraph ..........................   Passed    0.02 sec
      Start  2: FileHandle
 2/18 Test  #2: FileHandle .......................   Passed    0.01 sec
      Start  3: PassManager
 3/18 Test  #3: PassManager ......................   Passed    0.00 sec
      Start  4: Quadruple
 4/18 Test  #4: Quadruple ........................   Passed    0.00 sec
      Start  5: StringRef
 5/18 Test  #5: StringRef ........................   Passed    0.00 sec
      Start  6: Any
 6/18 Test  #6: Any ..............................   Passed    0.00 sec
```

skymizer build software better and faster

```
      Start  7: BinaryTree
 7/18 Test  #7: BinaryTree .......................   Passed    0.00 sec
      Start  8: StringSwitch
 8/18 Test  #8: StringSwitch .....................   Passed    0.00 sec
      Start  9: StringMap
 9/18 Test  #9: StringMap ........................   Passed    0.45 sec
      Start 10: Json
10/18 Test #10: Json .............................   Passed    0.00 sec
      Start 11: ComputeIR
11/18 Test #11: ComputeIR ........................   Passed    0.00 sec
      Start 12: TensorSel
12/18 Test #12: TensorSel ........................   Passed    1.24 sec
      Start 13: StatisticsTest
13/18 Test #13: StatisticsTest ...................   Passed    0.13 sec
      Start 14: MemAllocTest
14/18 Test #14: MemAllocTest .....................   Passed    0.01 sec
      Start 15: CounterTest
15/18 Test #15: CounterTest ......................   Passed    0.00 sec
      Start 16: Runtime_Abs
16/18 Test #16: Runtime_Abs ......................   Passed    0.14 sec
      Start 17: Runtime_Transpose
17/18 Test #17: Runtime_Transpose ................   Passed    0.00 sec
      Start 18: onnx2tg
18/18 Test #18: onnx2tg ..........................   Passed    0.00 sec
100% tests passed, 0 tests failed out of 18
Total Test time (real) =   2.03 sec
```

If you like to run a single unit test, you may run it interactively in the Docker prompt.

```
// In the container cli, run a single unit test.
$ ./tools/unittests/unittest_<test_name>
// e.g.
$ ./tools/unittests/unittest_Json
// You may leave the Docker container by typing ‘exit’ in the Docker shell prompt.
$ exit
```

Table 2 lists the 18 available unit tests in the ONNC repository.

Digraph	Digraph	PassManager
Quadruple	StringRef	Any
BinaryTree	StringSwitch	StringMap
Json	ComputeIR	TensorSel
StatisticsTest	MemAllocTest	CounterTest
Runtime_Abs	Runtime_Transpose	onnx2tg

Table 2. A list of unit tests in ONNC

6. Benchmarking Using Models from the ONNX Model Zoo

The Docker image includes a set of 12 pre-trained models from the ONNX Model Zoo[7] listed in Table 3. You may access the model files and associated input files in the /models directory. Or you may create your own model in the ONNX format, with input data in the ONNX TensorProto format[8].

bvlc_alexnet	bvlc_googlenet	bvlc_reference_caffenet
bvlc_reference_rcnn_ilsvrc13	densenet121	inception_v1
inception_v2	resnet50	shufflenet
squeezenet	vgg19	zfnet512

Table 3. A list of ONNX models from the model zoo

Running a Single Benchmark

You may run a single model for benchmarking using the following shell command:

[7] https://onnx.ai/

[8] https://github.com/onnx/onnx/blob/master/docs/IR.md

```
// run in the container cli
$ onni <model_file_path>/model.onnx <input_file_path>/input_0.pb -verbose=<level>
```

<model_file_path> is the path to the model file for the pre-trained ONNX model and ***<input_file_path>*** is the path to the corresponding input file. In the ONNC Docker container, the model file path is ***/models/<model_name>*** and the input file path is ***/models/<model_name>/test_data_set_<0~6>***. ***<level>*** indicates different levels of verbose information. Higher-level information is a superset of all lower-level information. For example, level 4 will include all information from level 1 to level 4.

Information for each verbose level:

- Level 1: Inference time & memory usage
- Level 2: ONNX operator statistics
- Level 3: Inference time & ONNX operator statistics per layer
- Level 4: Memory allocation log

Here is an example of running AlexNet and printing out all information.

```
// run in the container cli
$ onni /models/bvlc_alexnet/model.onnx
/models/bvlc_alexnet/test_data_set_0/input_0.pb -verbose=4
```

Running All Benchmarks Using Script

If you like to run all benchmarks in the model zoo, a `run-benchmark.sh` script is included in the Docker image to simplify your job. The script will compile a model and run inference for all models one by one. You may type 'run-benchmark.sh -h' to get the following usage description.

```
Usage: run-benchmark.sh [options]... MODEL [ARGUMENTS...]

    --rebuild           Rebuild the source code

                       (Will build the source in the /onnc/onnc directory)

    MODEL               Any model from the ONNX model zoo
    ARGUMENTS          Arguments be passed to target

========================================

ONNX model list (from the ONNX model zoo):

bvlc_alexnet  bvlc_googlenet  bvlc_reference_caffenet  bvlc_reference_rcnn_ilsvrc13
densenet121  inception_v1  inception_v2  resnet50  shufflenet  squeezenet  vgg19
zfnet512
```

run-benchmark.sh has an optional flag to rebuild the ONNC source code before benchmarking. More details will be covered in the next section.

You may run the script from the Docker prompt as simple as the following example:

```
// run in the container cli
$ run-benchmark.sh --rebuild bvlc_alexnet
```

You may also run the script from the shell prompt on your host machine using the following shell command:

```
// run on the host machine
$ docker run -it --rm -v <source_dir>/onnc:/onnc/onnc onnc/onnc-community run-
benchmark.sh --rebuild <model name>
```

Running Customized ONNX Models

Users may create their own ONNX model. In that case, they can run the benchmark using the following command:

```
// run in the container cli
$ onni <model_directory>/<model_file> <model_directory>/<input_file>
```

7. Collecting statistics data via ONNC Statistics API

ONNC provides a set of Statistics classes and APIs in its framework for users to collect and share statistics data across ONNC source code.

For more information, please refer to the ONNC Statistics API[9] document. It will contain up-to-date information as more statistics types and APIs are defined and implemented.

[9] https://github.com/ONNC/onnc/blob/docs/docs/api/statistics.md