Neural Processor Unit Compiler

Release 0.1

Chen Chung-Shu

May 24, 2020

CONTENTS

CHAPTER

ONE

DEEP LEARNING MODEL

1.1 AI

Hung-Yi Lee's video [1].

1.2 CNN

CNN: They have applications in image and video recognition, recommender systems, image classification, medical image analysis, natural language processing, and financial time series [4].

Concept about how to apply Convolution and MaxPool to getting features from image [2]. Conv+MaxPool -> get features map and downsize image, more Conv+MaxPool can filter image and higher level of features and downsize more image. CNN model used in image recognition.

Concept and data applying in Deap Learning for different models of CNN [3].

[1] https://www.youtube.com/watch?v=CXgbekl66jc&list=PLJV_el3uVTsPy9oCRY30oBPNLCo89yu49
[4] https://en.wikipedia.org/wiki/Convolutional_neural_network
[2] http://violin-tao.blogspot.com/2017/07/ml-convolutional-neural-network-cnn.html
[3] https://github.com/onnx/models

CHAPTER

TWO

NPU COMPILER

2.1 NPU compiler reference

https://arxiv.org/pdf/2002.03794.pdf

Tensorflow support unknown shape [2]. Though our npu support kernel call where kernel call is a set of commands to npu to deal shape at run time, it is unefficiency. As I remember mlit supports binding shape for unknown at compile-time but not always work. Lukily, we can customilze by redefining model to binding shape staticlly [20200412].

2.2 MLIR and IREE

IREE (Intermediate Representation Execution Environment, pronounced as "eerie") is an MLIR-based end-to-end compiler that lowers ML models to a unified IR optimized for real-time mobile/edge inference against heterogeneous hardware accelerators. IREE also provides flexible deployment solutions for the compiled ML models [1] as the following figure.

- HAL IR: Vulkan-like allocation and execution model encoding -> on-line first-time compilation and save in cache. Executable compilation via architecture specific backend compiler plugins.
- VM IR: Dynamic module linkage definitions (imports, exports, globals, etc) [3].

The purpose of mlir is:

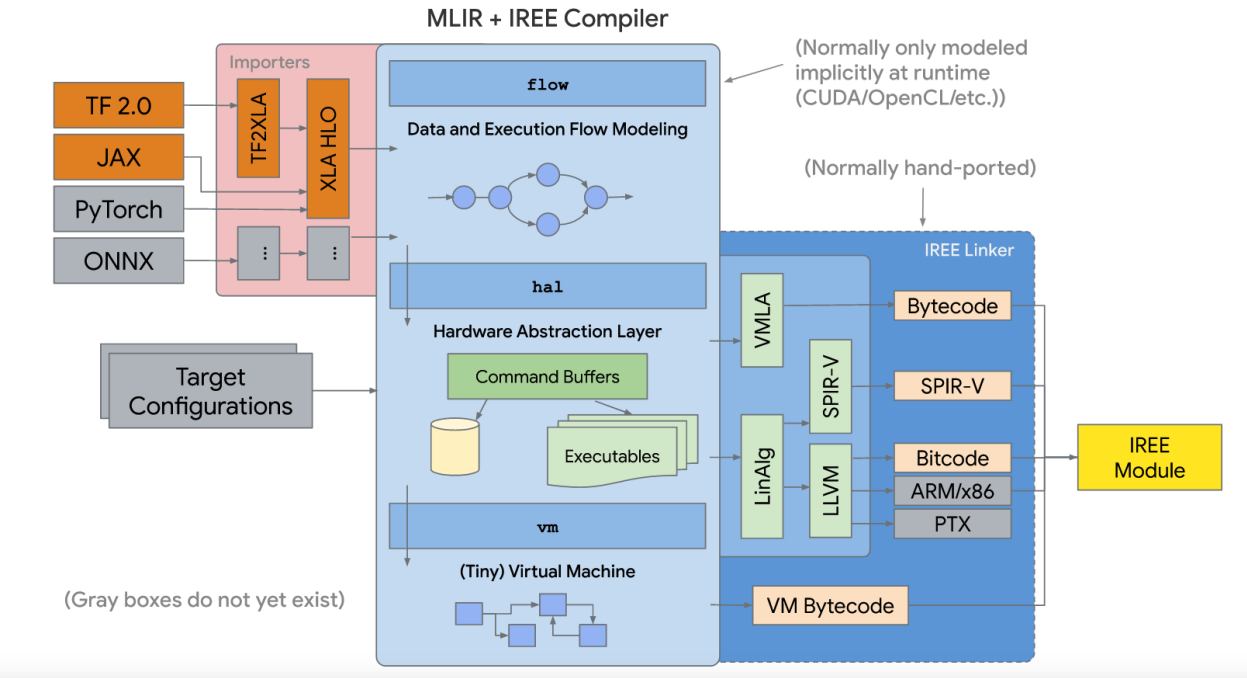
- Connect cpu with mlir-to-llvm-ir.

The purpose of iree is:

[2] https://pgaleone.eu/tensorflow/2018/07/28/understanding-tensorflow-tensors-shape-static-dynamic/

[1] https://github.com/google/iree

[3] Page 15 of https://docs.google.com/presentation/d/1RCQ4ZPQFK9cVgu3IH1e5xbrBcqy7d_cEZ578j84OvYI/edit#slide=id.g6e31674683_0_23101



- Connect to gpu with iree-to-spirv.

Both purpose of mlir and iree is:

- Reduce bug and problem between heterogeneous hardware accelerators [4].

2.3 Tensorflow

The mechansim of Mlir and iree applied on tensorflow as the figure above section is not fitted for off-line edge npu that stand alone without server-connection for tunning weight of face detection's purpose. It is designed for on-line server-connected npu. The gpu of supporting spirv is best candidate until this date 2020/5/12.

At beginning, tensorflow rely on api without fixed format such as ONNX [5]. As a result ONNX emerged and adopted for most of npu in their private backend compiler. Google does not like to hire onnx as the format for npu backend compiler onnx-mlir project [6] which convert onnx to mlir dialect is sponsored by Google I guess [7] for encourging new npu compiler development hiring mlir as their compiler input (convert onnx to mlir then handling mlir input).

With mlir and iree appear on tensorflow as a series of fixed formats in tensorflow as section above. The hardware vendors for cloud server AI machine with heterogeneous hardware accelerators will run tensorflow system by supporting mlir/iree input format in their compilers more and more. So, it is unavoidable that tensorflow system's npu vendors have to support mlir/iree input format beyond onnx. Or open source software or vendor software appear to do transfer from mlir/iree to onnx. (python in tensorflow api allow unknown type and shape size, so it cannot transer python api to onnx fully).

If lucky, google may hire onnx. Because onnx format is older than mlir in history. In addition in aspect of format, mlir has mult-level mult-dialect and more complicate while onnx is easy and better to understand (P.S. I don't dig into mlir

[4] https://kknews.cc/zh-tw/tech/klkombr.html
[5] Actually onnx format based on IO api with protobuffer. It has real binary format but may change from version to version. Tensorflow api has no real binary format.
[6] https://github.com/onnx/onnx-mlir
[7] https://groups.google.com/a/tensorflow.org/forum/#!topic/mlir/2FT4sD8kqTY

yet). Many AI models has supported onnx file format. For some AI model's formats that run on tensorflow without supporting onnx, aplly tensorflow-onnx open source project [8] can convert tensorflow to onnx partly.

Onnx alliance may release some programs for transfering mlir to onnx for fighting agiant mlir-iree growing in npu compiler but not at this moment.

For off-line edge npu that stand alone without server-connection for tunning weight of face detection's purpose, supprting mlir-iree compiler may not necessary.

2.4 mlir to onnx

https://www.tensorflow.org/mlir

https://mlir.llvm.org/talks/

https://llvm.org/devmtg/2019-04/talks.html#Tutorial_1

- 3 ppt in llvm tutorials

https://llvm.org/devmtg/2019-04/slides/Tutorial-AminiVasilacheZinenko-MLIR.pdf

build mlir: https://mlir.llvm.org/getting_started/

```
~/llvm/1/llvm-project/build$ cmake -G Ninja ../llvm \
>     -DLLVM_ENABLE_PROJECTS=mlir \
>     -DLLVM_BUILD_EXAMPLES=ON \
>     -DLLVM_TARGETS_TO_BUILD="X86;NVPTX;AMDGPU" \
>     -DCMAKE_BUILD_TYPE=Release \
>     -DLLVM_ENABLE_ASSERTIONS=ON

~/llvm/1/llvm-project/build$ cmake --build . --target check-mlir
[200/1919] Generating VCSRevision.h
-- Found Git: /usr/bin/git (found version "2.17.1")
[1604/1919] Building CXX object tools/mlir/tools/mlir-linalg-ods-gen/CMakeFiles/mlir-
 ↪linalg-ods-gen.dir/mlir-linalg-ods-gen.cpp.o
/home/cschen/llvm/1/llvm-project/mlir/tools/mlir-linalg-ods-gen/mlir-linalg-ods-gen.
 ↪cpp:935:6: warning: 'bool {anonymous}::Expression::operator==(const {anonymous}::
 ↪Expression&) const' defined but not used [-Wunused-function]
 bool Expression::operator==(const Expression &e) const {
      ^~~~~~~~~~
[1918/1919] Running the MLIR regression tests

Testing Time: 9.88s
  Unsupported Tests:  16
  Expected Passes   : 465
```

run: https://mlir.llvm.org/docs/Tutorials/Toy/

```
~/llvm/1/llvm-project/mlir/test/Examples/Toy/Ch1$ ~/llvm/1/llvm-project/build/bin/
 ↪toyc-ch1 ast.toy -emit=ast
...
~/llvm/1/llvm-project/mlir/test/Examples/Toy/Ch1$ ~/llvm/1/llvm-project/build/bin/
 ↪toyc-ch1 ast.toy -emit=ast 2>&1 | ~/llvm/1/llvm-project/build/bin/FileCheck ast.toy
~/llvm/1/llvm-project/mlir/test/Examples/Toy/Ch1$ ~/llvm/1/llvm-project/build/bin/
 ↪llvm-lit ast.toy
-- Testing: 1 tests, 1 workers --
PASS: MLIR :: Examples/Toy/Ch1/ast.toy (1 of 1)
```

[8] https://github.com/onnx/tensorflow-onnx

```
Testing Time: 0.11s
  Expected Passes: 1
```

The result I run is based on git commit 455ccde1377b3ec32d321eb7c38808fecdf230a8 Date: Sun May 17 21:00:09 2020 -0400

2.5 llvm IR for NPU compiler

Though npu has no general purpose registers GPR, it is possible to apply llvm ir for npu to do codegen by llvm as follows,

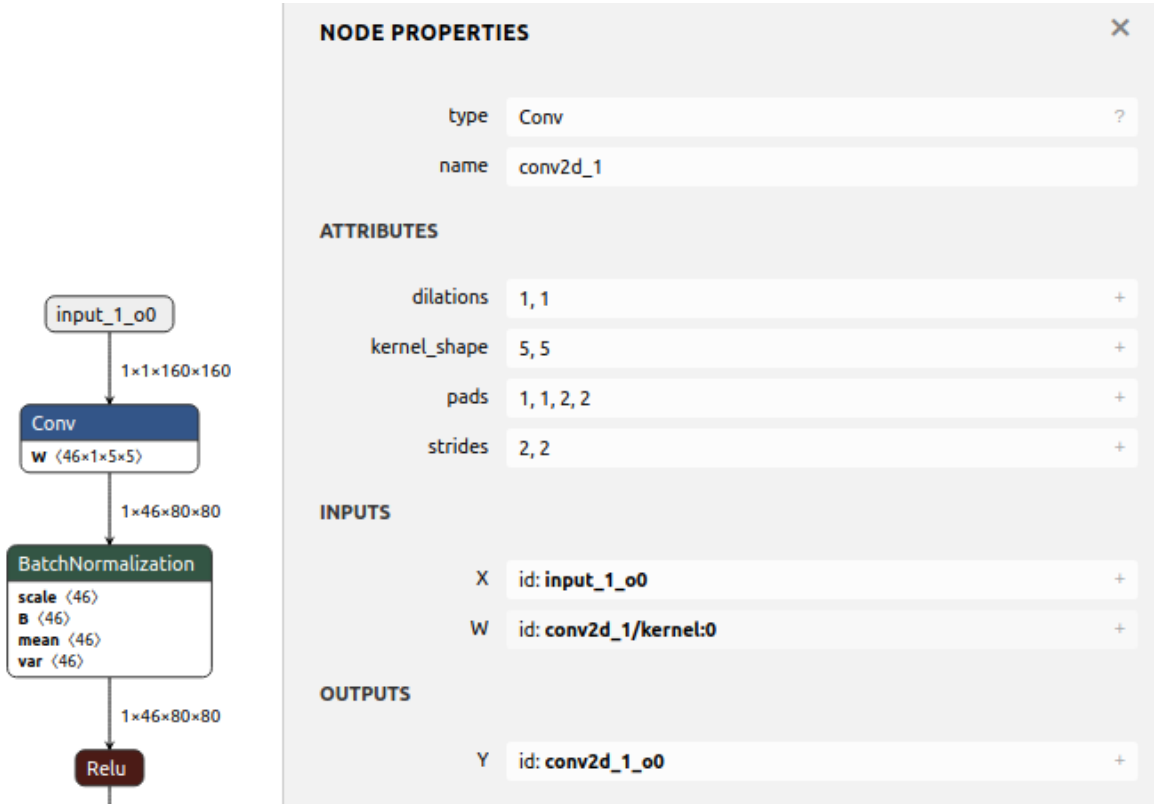


Fig. 2.1: Conv operation in onnx file

```
@weight = global [46 x 1 x 5 [5 x float]] [[[[5 x float] [float 0.05475775524973869, .
↪..], [5 x float] [float ...]], ...]
@conv = @llvm.npu1.conv float* @input, float* @weight, ...
```

Conclusion: Data definition too much and no GPR. Not worth to hire llvm.

2.6 Open source project

- onnx to mlir dialect: https://github.com/onnx/onnx-mlir

- tensorflow to onnx: https://github.com/onnx/tensorflow-onnx
- onnx to tensorflow: https://github.com/onnx/onnx-tensorflow

CHAPTER

THREE

ALTERNATE FORMATS

The book is also available in the following formats:

CHAPTER

FOUR

SEARCH THIS WEBSITE

- search