Integrating NVDLA into TVM

CS6280 Project Presentation

Dan Wu | Shivam Aggarwal

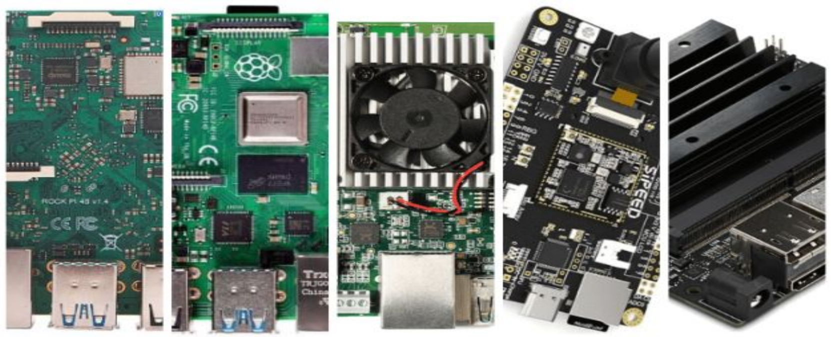
Nov 19, 2020

Outline

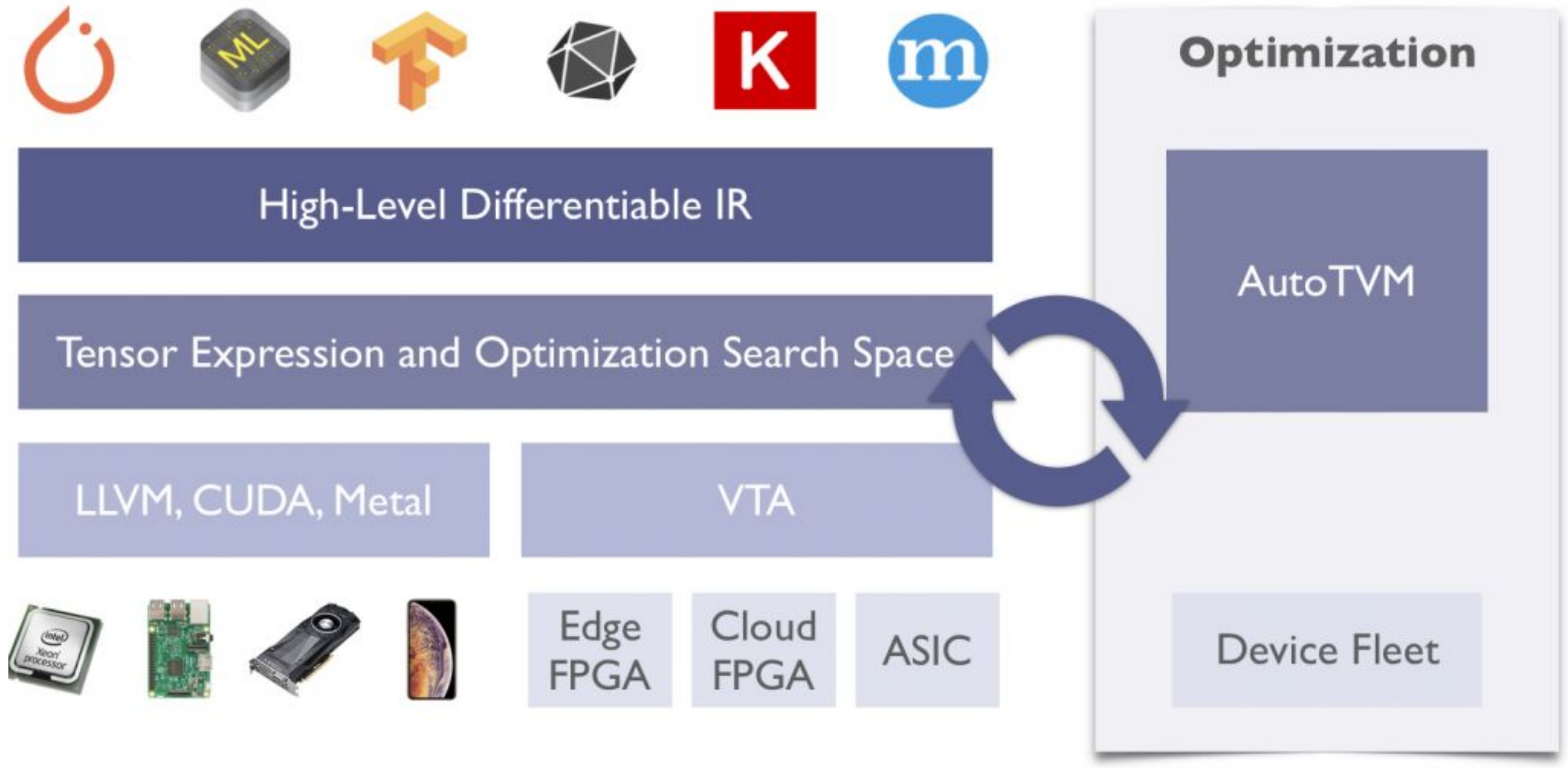
- Background
- Motivation
- Related work
- System design
- Challenges
- Limitations
- Evaluation
- Future work

TensorFlow
Keras
mxnet
PyTorch
Caffe
Microsoft Cognitive Toolkit
DL4J
ONNX

tvm

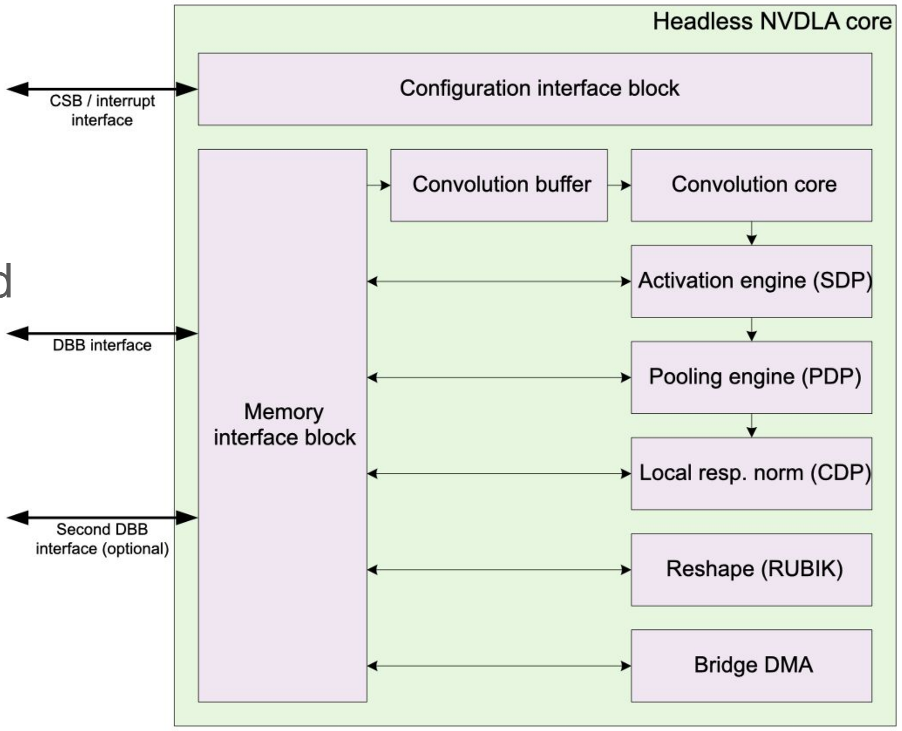


TVM: High level overview

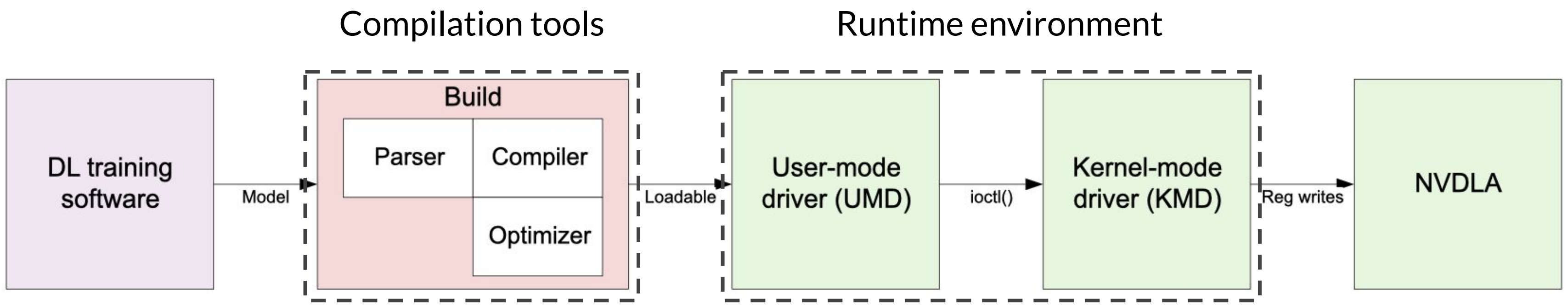

Optimization
High-Level Differentiable IR
Tensor Expression and Optimization Search Space
AutoTVM
LLVM, CUDA, Metal
VTA
Edge FPGA
Cloud FPGA
ASIC
Device Fleet

NVDLA: Hardware Architecture

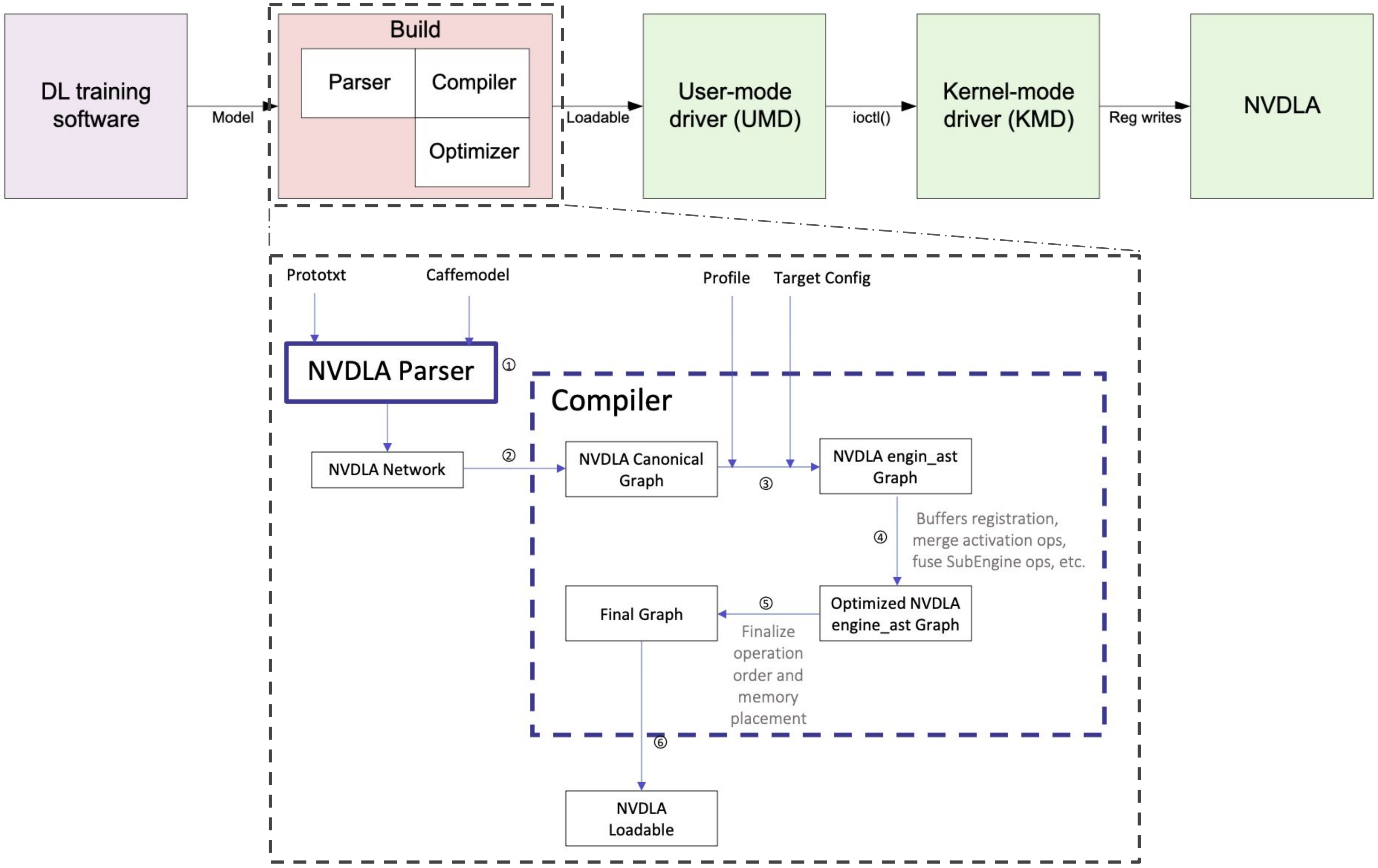
- **Open-source DNN accelerator**: can achieve industry-level DNN inference throughput.
- **Configurable:** hardware configuration, the system's CPU and memory controller configurations, and the application's custom neural network.
- **Scalable:** nv_small, nv_large
- Convolutional core: matrix-matrix multiplication
- Post-processing: activation function, pooling, etc.



NVDLA Workflow



How NVDLA compiler works?



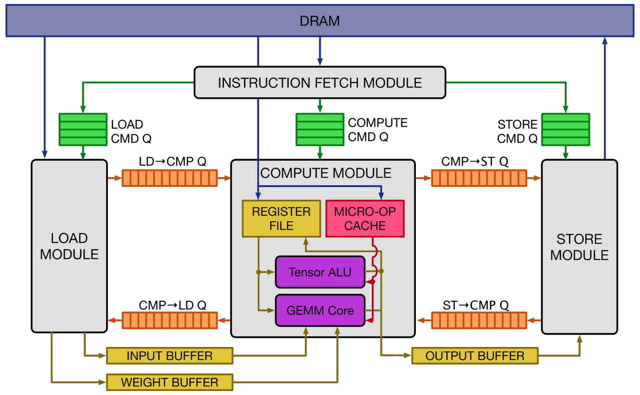
Motivation

NVDLA	TVM
Limited Frontend Support	Diverse Frontend Support
Limited Optimization	Diverse Optimization
No fallback system	Annotation in BYOC

NVDLA.org

tvm

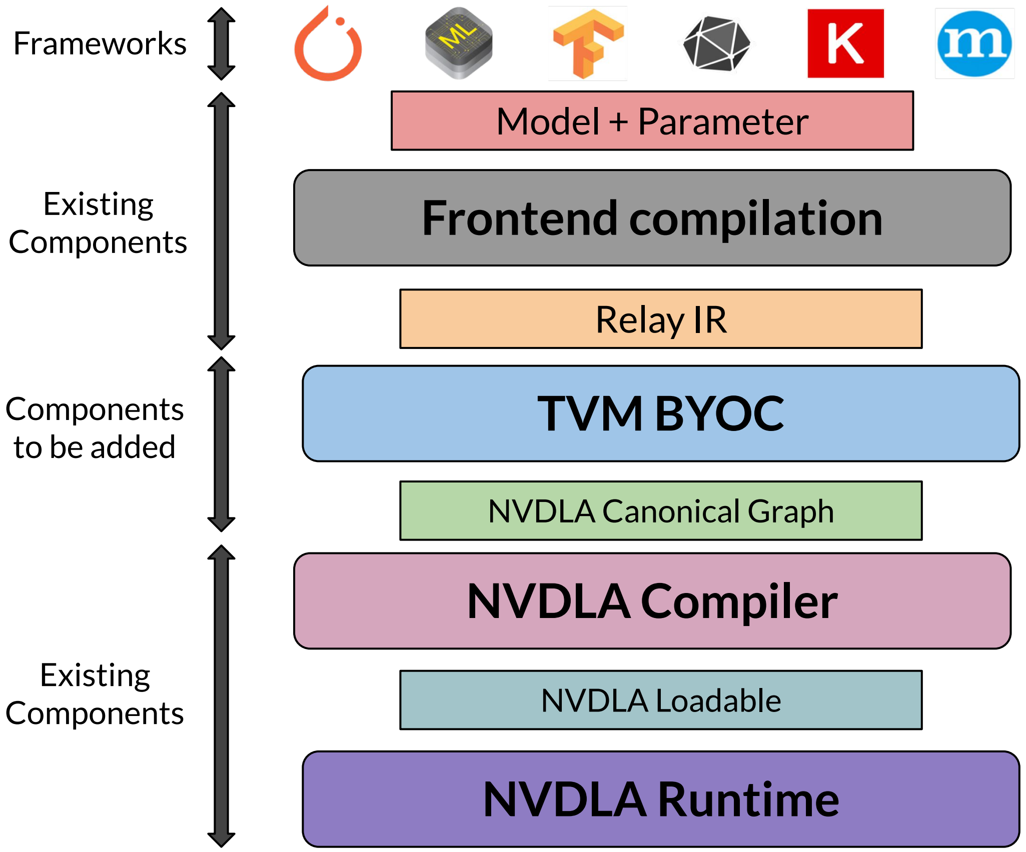
Related work

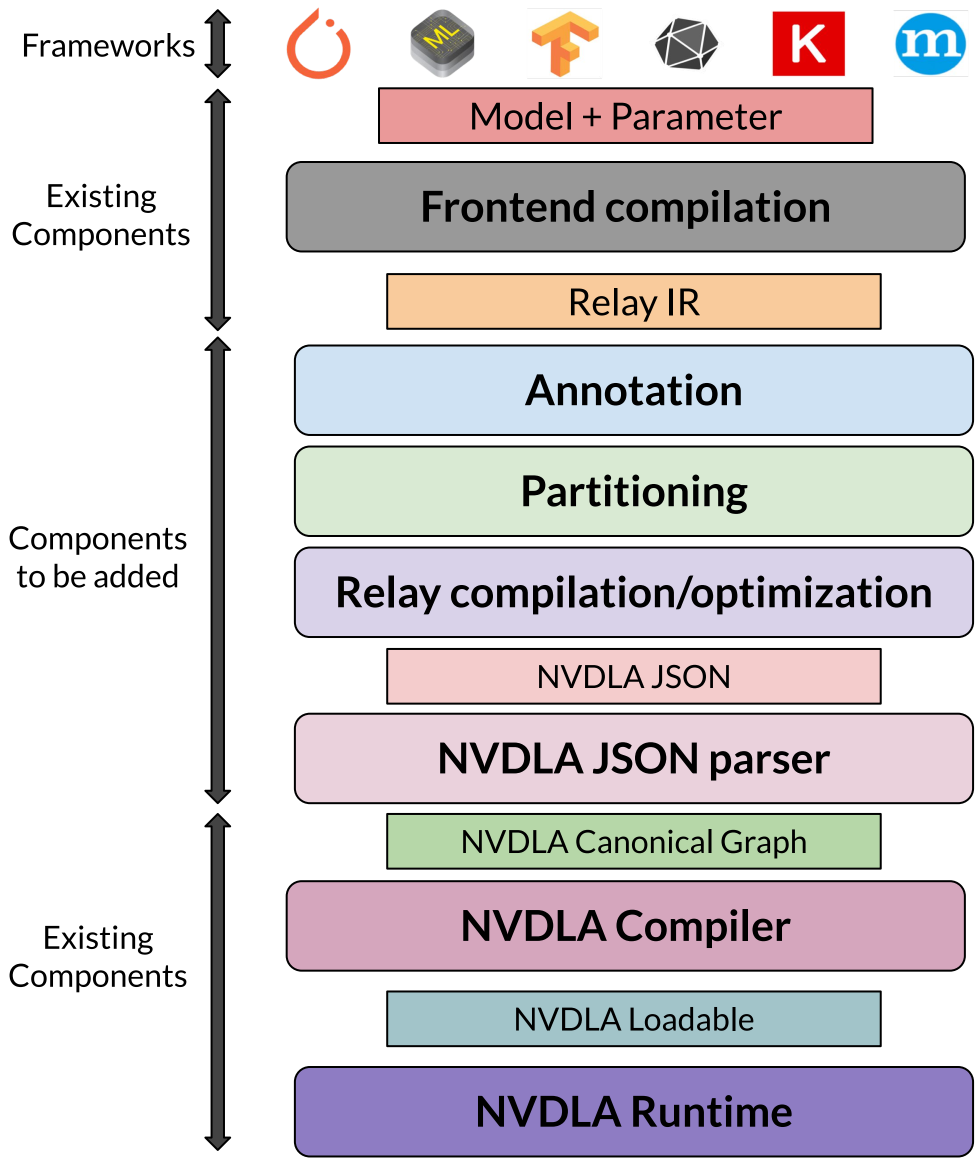
- 

- NVDLA ✖ ONNC

  open source DLA   open source compiler
- XILINX VITIS™ + tvm

How to integrate the two?




Frameworks
Existing Components
Components to be added
Existing Components
Model + Parameter
Frontend compilation
Relay IR
Annotation
Partitioning
Relay compilation/optimization
NVDLA JSON
NVDLA JSON parser
NVDLA Canonical Graph
NVDLA Compiler
NVDLA Loadable
NVDLA Runtime

Challenges

- Diverse workflow design
- Different network operator representation
    - In Relay, bias and inputs are stored as network layers;
    - In NVDLA, bias is a parameter of layer and inputs are represented as edges.
- Strict Check in NVDLA compiler
    - Surface check disables single layer testing;
    - Quantization check disallows simple layer without bias item.
- Non-Debuggable NVDLA runtime
    - Older version is more stable but not open-source;
    - Latest version is open-source but have fatal error.

Limitations

- Only high-level optimizations in TVM stack are used.
  - How to involve more lower-level optimizations in TVM?
- No good solution for evaluation.
  - Is it possible to compare the loadable files?
- Not end-to-end process.
  - How to integrate the two code base?

Evaluation

- We can generate JSON file for any trained neural network on diverse ML frameworks using TVM BYOC.
- We can parse a JSON file, rebuild the canonical graph and generate loadable file for all hardware configurations (4 types) in NVDLA.
- We do not have a real NVDLA board and do tests on SystemC-based or AWS FPGA-based simulators.
- Correct output for single convolutional layer.

Future work

- Compare loadable for functional correctness evaluation.
- End-to-end compilation framework for NVDLA.
- Integrating TVM with CGRA based accelerators such as HyCUBE

References

- Chen, Tianqi, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Meghan Cowan, Haichen Shen, et al. “TVM: An Automated End-to-End Optimizing Compiler for Deep Learning.” *ArXiv:1802.04799 [Cs]*, October 5, 2018. http://arxiv.org/abs/1802.04799.
- Chen, Tianqi, Lianmin Zheng, Eddie Yan, Ziheng Jiang, Thierry Moreau, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. “Learning to Optimize Tensor Programs.” *ArXiv:1805.08166 [Cs, Stat]*, January 8, 2019. http://arxiv.org/abs/1805.08166.
- Moreau, Thierry, Tianqi Chen, Luis Vega, Jared Roesch, Eddie Yan, Lianmin Zheng, Josh Fromm, et al. “A Hardware-Software Blueprint for Flexible Deep Learning Specialization.” *ArXiv:1807.04188 [Cs, Stat]*, April 22, 2019. http://arxiv.org/abs/1807.04188.
- Ragan-Kelley, Jonathan, Connelly Barnes, and Andrew Adams. “Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines,” n.d., 12.

Thank you!

Appendix: Relay IR to JSON

```
def @main(%data: Tensor[(1, 14, 14, 512), float32]) -> Tensor[(1, 7, 7, 512), float32] {

  nn.max_pool2d(%data, pool_size=[2, 2], strides=[2, 2], padding=[0, 0, 0, 0],

  layout="NHWC") /* ty=Tensor[(1, 7, 7, 512), float32] */

}
```

```
{
  "input": {
    "dtype": [
      [
        "float32"
      ]
    ],
    "shape": [
      [
        [
          1,
          14,
          14,
         512
        ]
      ]
    ]
  },
  "max_pool2d_0": {
    "ceil_mode": [
      [
        "0"
      ]
    ],
    "dtype": [
      [
        "float32"
      ]
    ],
    "layout": [
      [
        "NHWC"
      ]
    ],
    "num_inputs": "1",
    "num_outputs": "1",
    "padding": [
      [
        "0",
        "0",
        "0",
        "0"
      ]
    ],
    "pool_size": [
      [
        "2",
        "2"
      ]
    ],
    "shape": [
      [
        [
          1,
          7,
          7,
         512
        ]
      ]
    ],
    "strides": [
      [
        "2",
        "2"
      ]
    ]
  }
  }
```