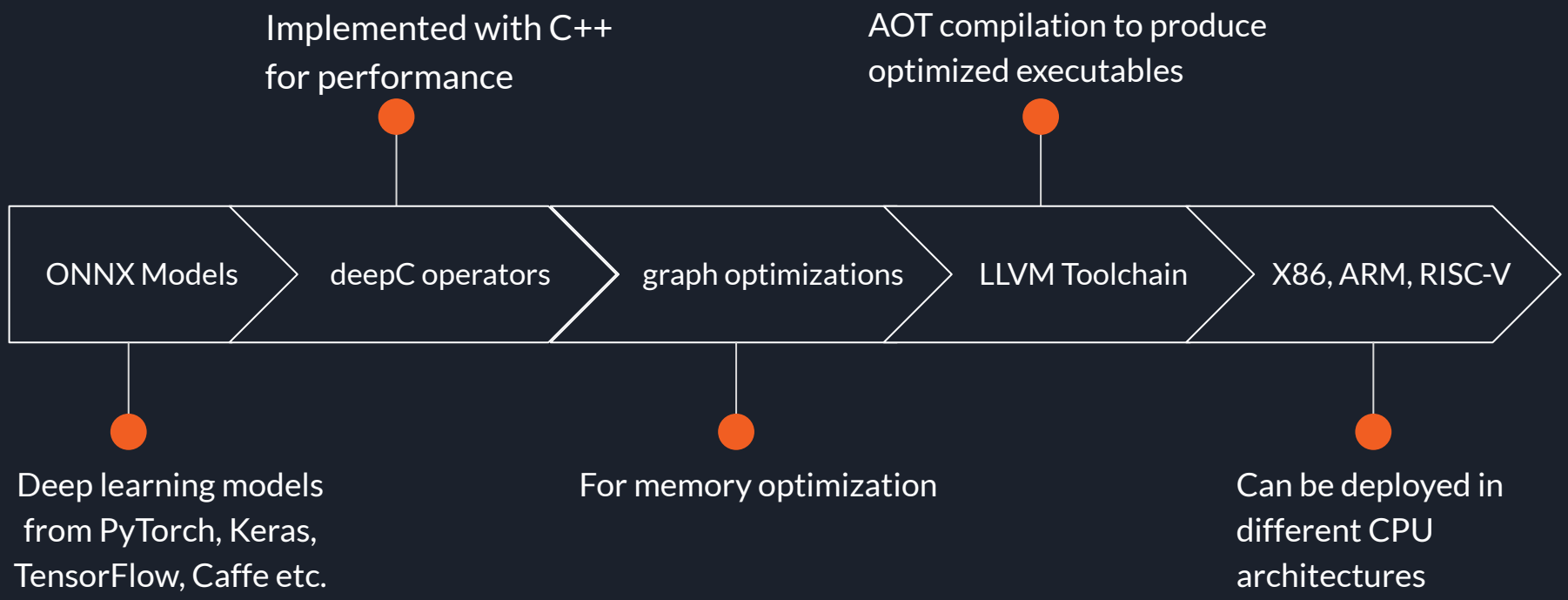
deepC

August 11, 2020

What is deepC

- deepC is a deep learning compiler and inference framework to perform deep learning in edge devices like raspberry pi, arduino, node MCU etc.
- deepC uses ONNX as frontend, which enables compilation of deep learning models from popular frameworks like tensorflow, pytorch, keras.
- In the backend deepC uses LLVM compiler toolchain, which helps to produce optimized executables for resource constrained devices.
- deepC also comes with python interface, with deep learning operators and numpy compatibility.

Design


Implemented with C++
for performance
AOT compilation to produce
optimized executables
ONNX Models
deepC operators
graph optimizations
LLVM Toolchain
X86, ARM, RISC-V
Deep learning models
from PyTorch, Keras,
TensorFlow, Caffe etc.
For memory optimization
Can be deployed in
different CPU
architectures

Features

- **Operator fusion:** deepC is implemented with loosely typed base operator with generic type information in the compute graph. This design makes it easy to fuse two or more operators painlessly with trivial operator type, tensor shape and rank checking.
- **Quantization:** In machine learning models quantization is used to lower power consumption, memory bandwidth, storage, and increase performance. Using C++ template types, deepC can convert any mixed precision data types for compilation .

Features Contd.

- **Memory layout:** Eigen C++ library helps deepC with improved tensorization, reduced memory accesses and cache misses using latency hiding, scheduling and other optimizations.
- **Scalability:** With the help of Swig, deepC leverages the binding of C++ in the backend for performance, and Python in the frontend for quick testing and wider usage.
- **Wide support:** With cross compilation, deepC can build app for edge devices, and flash the binaries directly to that device.

deepC Framework Usage

Perform Pythonic Operations

```
import deepC.dnnc as dc

# Addition
t0 = dc.array([[1,2,3],[4,5,6]])
t1 = dc.array([[10,20,30],[40,50,60]]).asTypeInt()
t2 = t0 + t1

# Broadcasting
a = True
b = dc.array([[1,2,3],[4,5,6]])
y = a + b

# convert to python types
py_list = list(y);   # tensor to python list
py_tuple = tuple(y); # tensor to python tuple
```

Numpy and Matplotlib Compatibility

```
import numpy as np
import matplotlib.pyplot as plt

# numpy to deepC
x = np.arange(5)
y = dc.array(x.tolist())

# deepC to numpy
a = dc.arange(5)
b = a.numpy()

# plot with matplotlib
sinx = dc.sin(a)
plt.plot (x.data(),sinx.data())
```

deepC Compiler Usage

Compile Tensorflow, Keras models

```
# Compile tensorflow models
!deepCC tensorflow_model_path.tf --format=tensorflow


# Compile keras models
!deepCC keras_model_file.h5 --format=keras


# For tensorflow models, we have to pass saved_model
# directory, but for keras models, as deepCC can find
# out it's keras model by looking at model extension,
# there is no need to pass format argument.
# The command below will work too.

!deepCC keras_model_file.h5
```

Compile other deep learning models

```
# PyTorch supports ONNX model export natively
import torch.onnx
torch.onnx.export(model, dummy_input, "model_file.onnx")


# In order to compile deep learning
# models from other frameworks, those
# models needs to be converted to ONNX
# models first, then compile with deepCC

# Compile ONNX models
!deepCC model_file.onnx
```

References

1. Guennebaud, G., Jacob, B., et al.: Eigen v3 (2010). (Eigen)
2. Bai, Junjie and Lu, Fang and Zhang, Ke and others, ONNX 3.0: Open Neural Network Exchange, 2019, (onnx/onnx: Open standard for machine learning interoperability)
3. D. Ulyanov, A. Vedaldi, and V. Lempitsky. Instance normalization: The missing ingredient for fast stylization. arXiv preprint arXiv:1607.08022, 2017.
4. Rohit Sharma et. Al, DNNC: Deep Neural Network Compiler, 2019 (ai-techsystems/deepC: vendor independent deep learning library, compiler and inference framework microcomputers and micro-controllers)
5. Gemm (https:#en.wikipedia.org/wiki/Basic_Linear_Algebra_Subprograms#Level_3)
6. E. Fiesler, A. Choudry, and H. J. Caulfield, “Weight discretization paradigm for optical neural networks,” in Optical Interconnections and Networks, H. Bartelt, Ed. SPIE, Aug 1990: (Weight discretization paradigm for optical neural networks)
7. Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel Emer, Efficient Processing of Deep Neural Networks: A Tutorial and Survey, Proceedings of the IEEE. 2017.

Thank You