Automated Python-to-RTL Transformation and Optimization for Neural Network Acceleration

Presenter： Chen Yang

School of Electronic Engineering

Beijing University of Posts and Telecommunications

Beijing 100876, China

2024·May·10th

CONTENTS

#01
Introduction

Background

Field-Programmable Gate Array (FPGA)

- low power consumption
- low latency
- parallel computing
- Reconfigurability
- ……

Good carrier → Neural Network （NN）

← *Realized tool* Vitis HLS (High-level Synthesis)

General Method

- Deployment of NN on FPGA
  - NN deployment on FPGA is typically done at the RTL (Register-Transfer Level) hardware development stage.
- Limitation
  - Development at the RTL level is challenging and time-consuming.
  - NN networks are mostly based on architectures like PyTorch and implemented in Python language.
  - The Vitis HLS tool lacks targeted optimization.

Motivation

- Deployment of NN on FPGA
  - Convert Python to C++ code
  - Simplify the development process
  - Optimize the implementation of NN deployment on FPGA

How to directly convert NN from Python to C++ code?

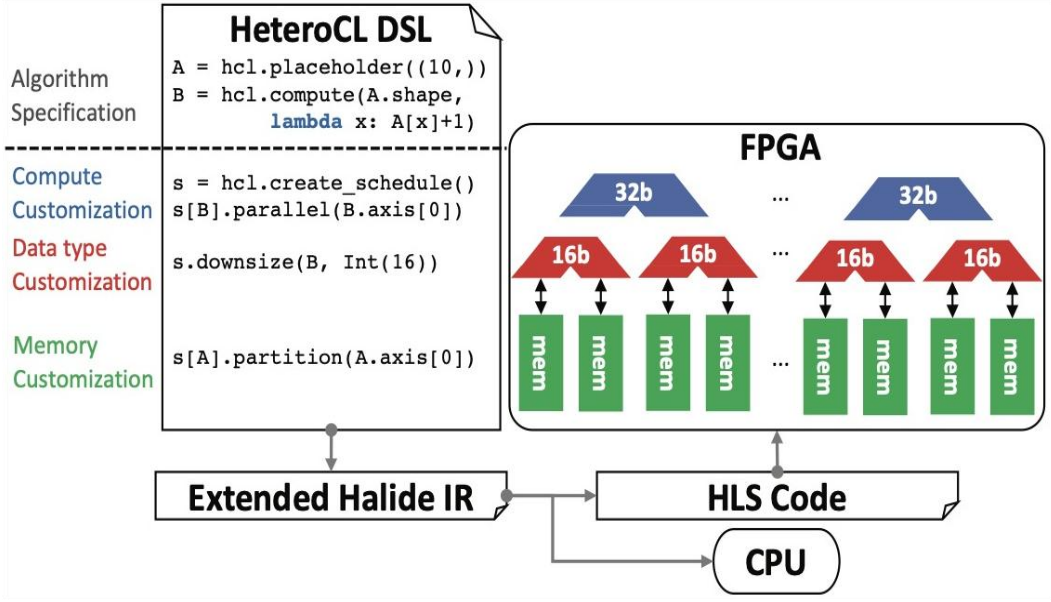
How can specific optimizations be applied to NN during this process?

#02

Related Work

Related Work

- HeteroCL
  - Multi-paradigm programming environment
  - Based on the Python language
  - Provides multiple optimization strategies



- Limitation

  Insufficient specific optimizations for deep learning.

  - *Data quantization*
  - *Memory access optimization*
  - *Computational optimization*

Lai Y H, Chi Y, Hu Y, et al. Heterocl: A multi-paradigm programming infrastructure for software-defined reconfigurable computing[C]//Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. 2019: 242-251.

Related Work

- Dynamic Fixed-Point Quantization

[1] **Method :**

$$l_{int} = 1bmax(x) + 1\ ,\ \ l_{float} = l_{bw} - \ l_{int}$$

x is the number to be quantized; l_{int} represents the length of integer bits, l_{float} denotes the width of fractional bits, and l_{bw} indicates the width after quantization.

Advantage:

- Low computational overhead
- Low storage overhead

[1] J. Gong, S. Zhao, H. He, et al. "Design of Quantized CNN Acceleration System Based on FPGA"[J]. Computer Engineering, 2022(3):170-174..

Related Work

- Dynamic Fixed-Point Quantization

[2] **Method :** Quantization Method based on Kullback-Leibler (KL) Divergence

$$\mathrm{KL}(\mathrm{P},\mathrm{Q}) = \sum_{x\in X}\left(P[x] * \log\left(\frac{P[x]}{Q[x]}\right)\right)$$

$$fl\_in = (-1)^s \frac{T}{\sum_{i=0}^{B-2} 2^i * x_i}$$

Advantage:

- Taking into account the influence of input on the quantization bit width
- The resulting data width is more rigorous

[2] X. K. Lei, Z. G. Yin and R. L. Zhao. "FPGA-based convolutional neural network fixed-point acceleration"[J]. Journal of Computer Applications, 2020(10):2811-2816.

Related Work

- Computational Optimization

[3] **Method :**

- Loop unrolling
- Loop pipelining

Table 2: Data sharing relations of CNN code

	input_fm	*weights*	*output_fm*
trr	dependent	irrelevant	independent
tcc	dependent	irrelevant	independent
too	irrelevant	independent	independent
tii	independent	independent	irrelevant
i	dependent	independent	irrelevant
j	dependent	independent	irrelevant

[3] C. Zhang, P. Li, G. Y. Sun, Y. J. Guan, B. J. Xiao, and Jason Cong. Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks[C]//Proceedings of the 2015 ACM/SIGDA international symposium on field programmable gate arrays. 2015: 161-170

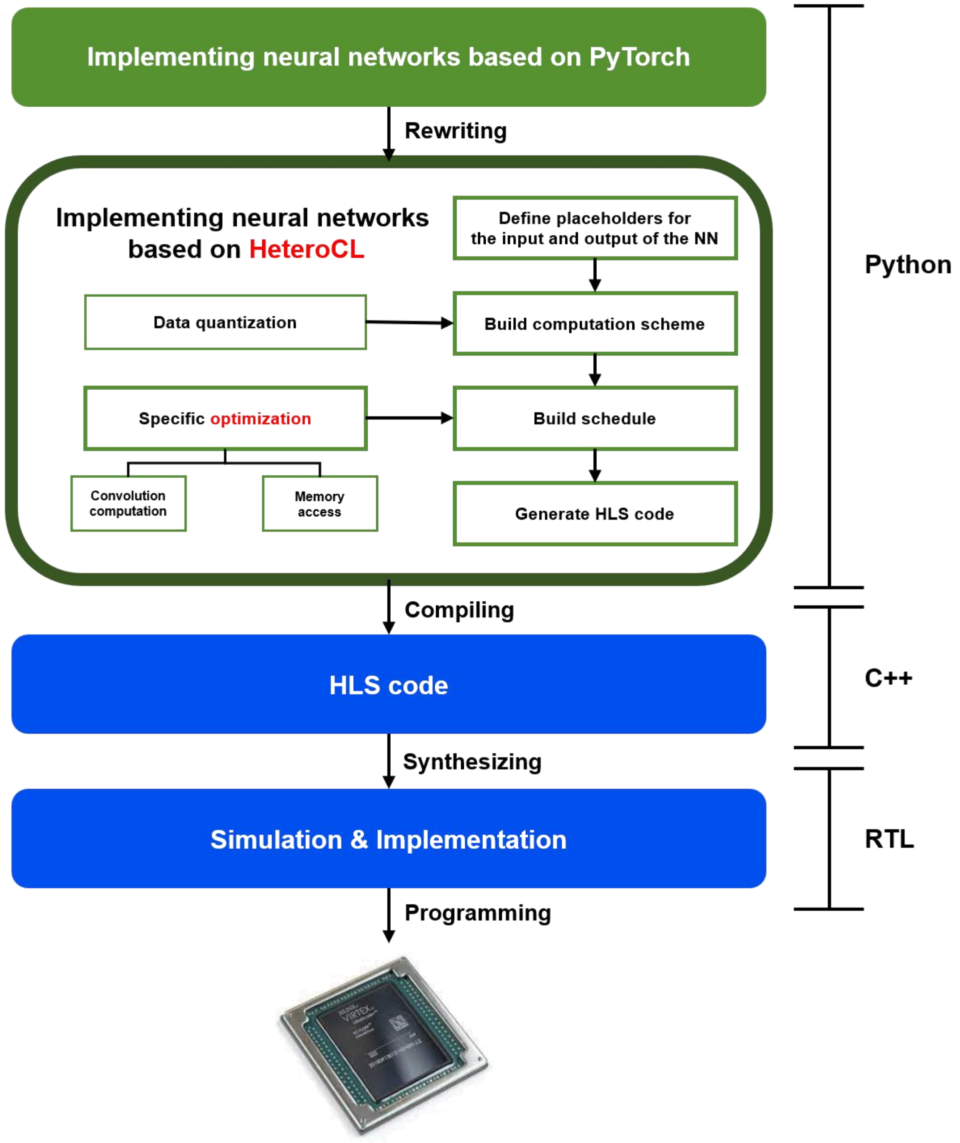
#03
Method

Total Flow

Network Import

- Step:
  - Build placeholders for the input and output of each layer
  - Construct Scheme
  - Construct Schedule
  - Generate code

- Importing Input and Weights:
  - Generating .dat files
  - For inputs, directly reading from the test file
  - For weights, importing into HLS code as constant arrays



1 Support for deep learning interfaces

Deep Learning Interface

- Implementing neural network function layer interface using HeteroCL and conventional library functions.
- Advantages:
  - The interface allows for the rapid and convenient construction of neural networks.
  - Leveraging the computational graph features of HeteroCL facilitates targeted optimization in subsequent steps.

Interface name	Interface description
conv2d_nchw	2D convolution layer without bias in nchw order
conv2d_nchw_bias	2D convolution layer with bias in nchw order
conv2d_nhwc	2D convolution layer without bias in nhwc order
avg_pool2d_nchw	2D average pooling layer in nchw order
avg_pool2d_nhwc	2D average pooling layer in nhwc order
maxpool2d	2D max pooling layer in nchw order
softmax2d	2D softmax layer in nchw order
logsoftmax2d	2D log softmax layer in nchw order
linear	2D linear layer in nchw order
relu	2D ReLU layer
flatten	Convert multi-dimensional array to 2D array
flatten_nchw	Convert 4D array in nchw order to 2D array
dense	2D fully connected layer

2.1 Fixed-point quantization scheme based on data distribution

Key Idea:

Based on statistical analysis of weight distributions, determine bit width according to the ratio between the integer and fractional parts for data with concentrated distributions.

Considering the influence of the input:

- $O_{int} = I_{int} + W_{int} + 1$
- $O_{\mathrm{dec}} = \max(I_{dec}, W_{dec})$

Algorithm 1 Global fixed-point Quantization when the distribution is concentrated

---

Require $:max, min, mean$

Initialize maximum integer bit width max_int_width=0, maximum decimal bit width max_dec_width=0, fixed-point quantization bit width $width$

for $value$ in $[max, min, mean]$ **do**

  Calculate the integer part of the value int_part and the decimal part dec_part

  Integer bit width int_width=$[log_2(abs(int\_part))]$+1

  if $10 \leq \frac{int\_part}{dec\_part} < 100$ **then**

    dec_width=4

  else if $\frac{int\_part}{dec\_part} \geq 100$ **then**

    dec_width=1 or dec_width=0

  else if int_part=0 or $\frac{int\_part}{dec\_part} \leq 10$ **then**

    dec_width=7 or $dec\_width = 10, int\_width$=0

  end if

  $max\_int\_width = max(max\_int\_width, int\_width)$

  $max\_dec\_width = max(max\_dec\_width, dec\_width)$

end for

$width$=max_int_width+max_dec_width

2.2 Memory Access Optimization

- Propagation of intermediate layer computation results
  - Setting up FIFO queues for writing to and reading from data
  - Between layers
  - Support pipelined parallelism for each layer, thereby reducing network latency
- Convolution Buffer
  - Establishing row buffers and window buffers to record data
  - Buffer Reading Queue for retrieving data, Convolution Reading Buffer for accessing buffer data

2.3 Loop computation optimization

- Loop unrolling
  - Combined with array partition
  - Method : Inserting unroll pragma/ calling Schedule's unroll
  - More suitable for loops with low replication overhead

- Loop pipelining
  - Combined with array partition
  - Method : Inserting pipeline pragma / calling Schedule's pipeline
  - More suitable for the outer loop of the convolution operation

- Loop merging
  - Merging layers that have continuous computations and identical outer loops.

Algorithm 2 Example of 4D convolutions with FIFO queues and row and window buffering

```
for (int nn=0;nn<2;nn++) do
  ...//Omitted outer loops
  for (int v131=0;v131<3;v131++) do
    //update the row buffer
  end for
  if (yy=2)>=0 then
    for (int v135=0;v135<3;v135++) do
      for (int v136=0;v136<3;v136++) do
        //update the window buffer
      end for
    end for
    if (xx-2)>=0 then
      float sum=0;
      for (int rc=0;rc<3;rc++) do
        for (int ry=0;ry<3;ry++) do
          for (int rx=0;rx<3;rx++) do
            //convolution computation
          end for
        end for
      end for
      ap_fixed<10,4> v155=sum;
      conv1_x_0_conv1.write(v155);
      //HLS::Stream write
    end if
  end if
end for
```

#04
Experiment

Experimental Setup

Setup

- Implemented on Xilinx Virtex7 with a clock cycle of 10ns
- Linux ubuntu 4.4.0-210-generic platform
- Vitis HLS - High-Level Synthesis from C, C++ and OpenCL v2021.2 (64-bit)
- Vivado v2021.2 (64-bit)
- HeteroCL v0.5 with MLIR

Neural Network and Dataset Selection

- LeNet-5, MNIST dataset for handwritten digit classification task (10 classifications)
- MobileNet-v1, Cifar-100 dataset for image classification task (100 classifications)
- ResNet-18, Cifar-100 dataset for image classification task (100 classifications)

Fixed-Point Quantization Experimental Results

Conditions:

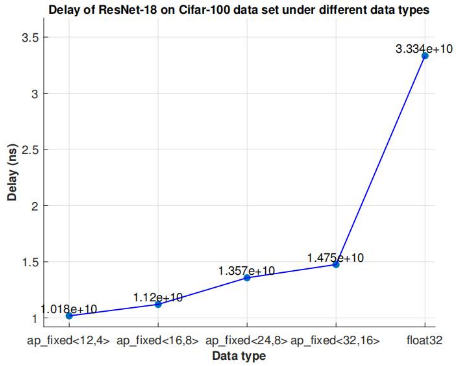
- The same network, weights, and inputs

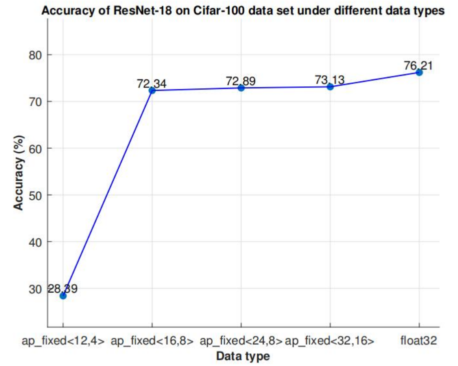
Metrics:

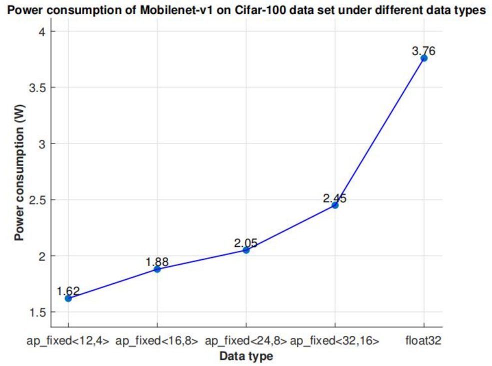
- The network's accuracy, latency, and power consumption

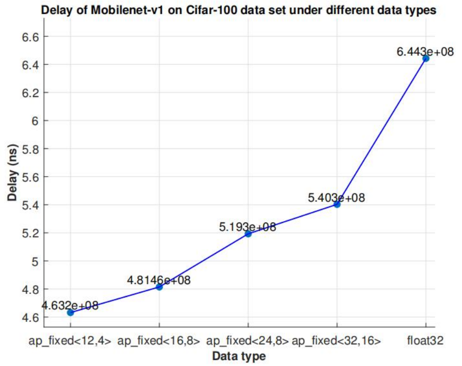
Results：

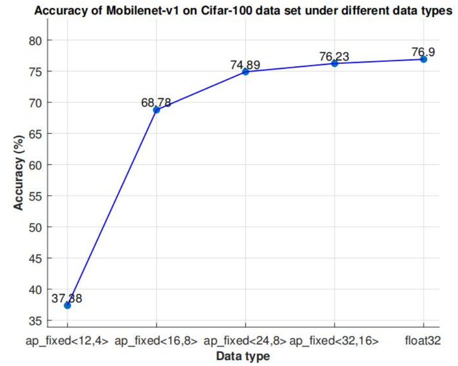
- Under the premise of similar final accuracy, there is a significant reduction in power consumption and latency

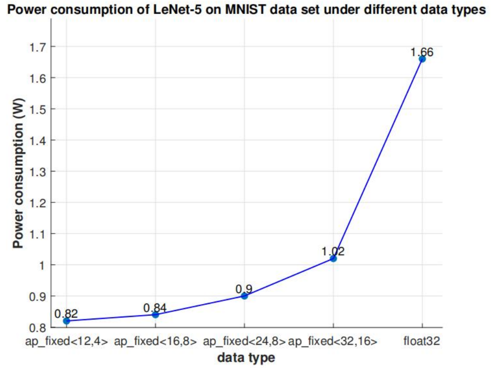


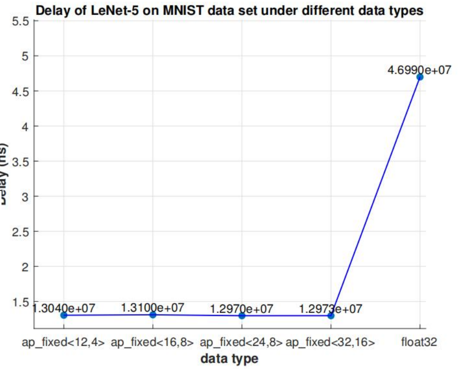


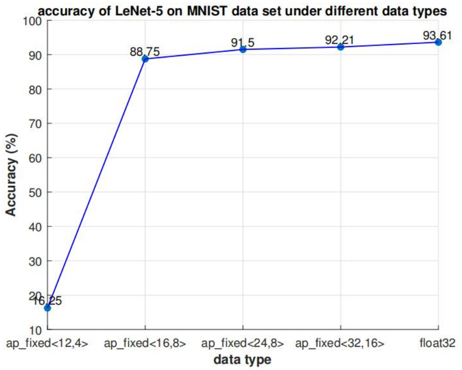


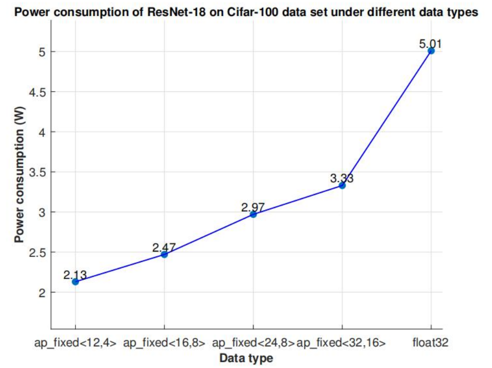












Inference Optimization Scheme

- Select fixed-point bit width based on quantization experimental results.
- Choose optimization schemes according to network characteristics.

Fixed-point quantization bit-width of each network

Network	Fixed point bit width	
	Integer bit width	fractional bit width
LeNet-5	12	12
MobileNet-v1	8	16
ResNet-18	8	16

Composition of comprehensive optimization schemes for the three networks

network	fixed point quantization	stream transfer+buffer	loop pipelining	loop unrolling	loop merging
Lenet-5	✓	✓	✗	✓	✗
Mobilenet-v1	✓	✓	✓	✗	✓
ResNet-18	✓	✓	✓	✗	✓

Inference Optimization Results

Network	Fixed point width		stream transfer+buffer		loop pipelining		loop unrolling	
	Integer bit width	fractional bit width	delay(ns)	power consumption (W)	delay(ns)	power consumption (W)	delay(ns)	power consumption (W)
LeNet-5	12	12	2.106E+07	1.05	5.342E+06	0.98	4.974E+06	1.12
MobileNet-v1	8	16	5.711E+08	2.34	4.062E+07	2.08	3.842E+07	2.47
ResNet-18	8	16	1.878E+09	3.63	8.431E+08	3.11	8.032E+08	3.58
Network	**Fixed point width**		**loop merging**		**Comprehensive optimization solutions**		**Baseline scenario**	
	Integer bit width	fractional bit width	delay(ns)	power consumption (W)	delay(ns)	power consumption (W)	delay(ns)	power consumption (W)
LeNet-5	12	12			**5.548E+06**	**1.14**	4.699E+07	1.66
MobileNet-v1	8	16	6.383E+08	1.89	**4.437E+07**	**2.12**	6.443E+08	3.76
ResNet-18	8	16	1.330E+10	2.63	**8.856E+08**	**2.98**	3.334E+10	5.01

THANKS

For Your Attention