Weekly Progress

崔烁为, 麦景
04/10/2020

previous issue: pattern in code

- The routine of a specificed ML algorithm is described by customized functions f1-f6.

```python
#!/usr/bin/env python
# coding: utf-8

import numpy as np
from functools import reduce


m = 10 # number of samples
n = 3 # number of features

# input

x = np.arange(m*n).reshape(m, n)
y = np.arange(m)
z = np.arange(m*n).reshape(m, n)

# parameter

w = np.arange(n)
```

```python
# customized functions

f1 = lambda x, y: x * y
f2 = lambda acc, x: acc + x
f3 = lambda x: x
f4 = lambda x, y: x - y
f5 = lambda x, y: x * y
f6 = lambda acc, x: acc + x


# Pattern(implictly leverage "broadcast")


s = np.array(list(map(lambda t: reduce(f2, f1(t, w)), x)))
s = f4(f3(s), y)
s = f5(z, s[:,np.newaxis])
s = reduce(f6, s)
```

Issue: Paper citing TABLA

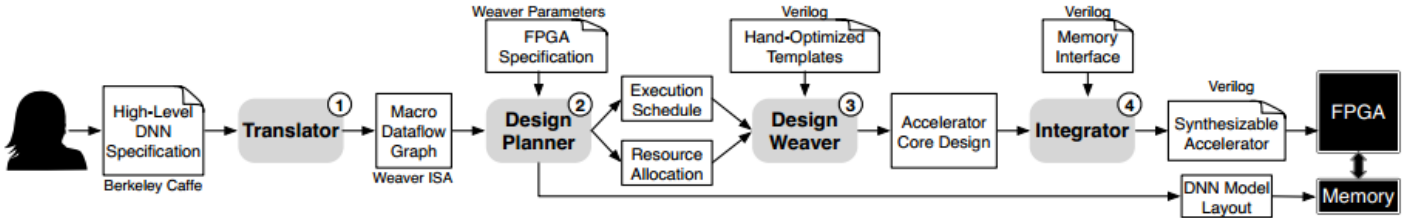
Title	Author	Tag	Conference	Year	Citing
Cambricon-F		ISA	ISCA	2019	
TABLA			HPCA	2016	
PipeLayer: A Pipelined ReRAM-Based Accelerator for Deep Learning		ReRAM	HPCA	2017	TABLA, PRIME, ISAAC
PUMA: A Programmable Ultra-efficient Memristor-based Accelerator for Machine Learning Inference		ReRAM	ASPLOS	2019	TABLA
PyRTLMatrix: an Object-Oriented Hardware Design Pattern for Prototyping ML Accelerators			EMAC2(works	2019	TABLA
FPSA: A Full System Stack Solution for Reconfigurable ReRAM-based NN Accelerator Architecture.	Yu Ji, Youhui Zhang, Yuan Xie	ReRAM	ASPLOS	2019	
Bridge the Gap between Neural Networks and Neuromorphic Hardware with a Neural Network Compiler	Yu Ji, Youhui Zhang, WenGuang Chen, Yuan Xie	ReRAM	ASPLOS	2018	
DNNWeaver: From High-Level Deep Neural Models to FPGAs	ACT lab	ISA	MICRO	2016	TABLA
Bit Fusion: Bit-Level Dynamically Composable Architecture for Accelerating Deep Neural Network	ACT lab		ISCA	2018	
An Instruction Set Architecture for Machine Learning	Yunji Chen	ISA			TABLA
CoSMIC: Scale-Out Acceleration for Machine Learning	ACT lab		MICRO	2017	TABLA
FlexiGAN: An End-to-End Solution for FPGA Acceleration of Generative Adversarial Networks	ACT lab		FCCM	2018	TABLA

See more info in https://github.com/magic3007/ML-on-Silicon

From High–Level Deep Neural Models to FPGAs

· Design DNNWEAVER: a framework that automatically generates a synthesizable accelerator for a given (DNN, FPGA) pair from a high-level specification in Caffe

· work flow



From High–Level Deep Neural Models to FPGAs

· Translator: translate Caffe code to state machine and microcodes

· Hand-Optimized Templates:

The PEs and the buffers in the template PU architecture provide compute capabilitiesfor convolution and inner product layers. The customizablenormalization, pooling, and activation modules provide supportfor the other possible layers in DNNs

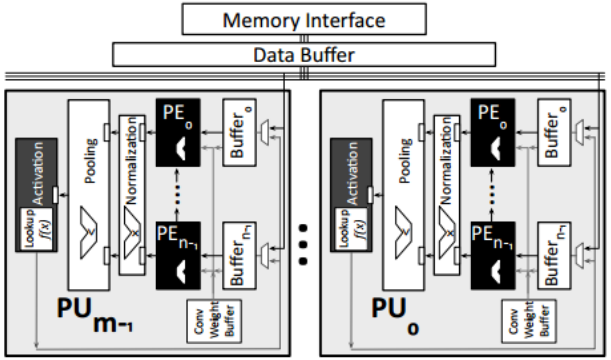


Figure 5: Overview of a clustered hierarchical template design. The template accelerator is divided into Processing Units (PUs) that are comprised of multiple smaller Processing Engines (PEs).

From High-Level Deep Neural Models to FPGAs

- Optimizations in different layers:
  - conv: dedicated buffer in PUs for weights, parallelism across output elements saving partial results, data forwarding across PEs, reusing data across convlutions kernels
  - pooling: the pooling module overlaps its operations with the convolution operation
  - IP: parallelism across output elements
  - Normalization and Activation

An End–to–End Solution for FPGA Acceleration of Generative Adversarial Networks

· Similiar to previous papers:

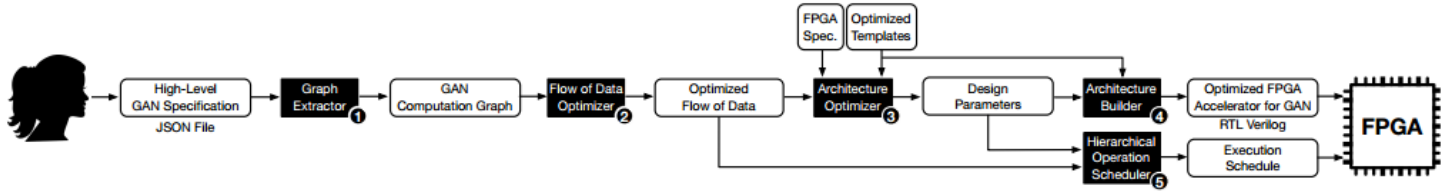


Figure 2: Overview of FlexiGAN end-to-end solution. FlexiGAN receives a high-level description of GAN and the target FPGA specification. At the end, it generates an optimized FPGA accelerator and the instruction schedules.

· Chanllenge for GAN accelerations: Inefficiency of using convolution hardware to performtransposed convolution

An End–to–End Solution for FPGA Acceleration of Generative Adversarial Networks

- To overcome this challenge
  - reorganize output rows: the even-indexed output rows (2 and 4) become adjacent. Similarly, the odd-indexed rows (3 and 5) are placedadjacent to each other
  - reorganizes thefilter rows

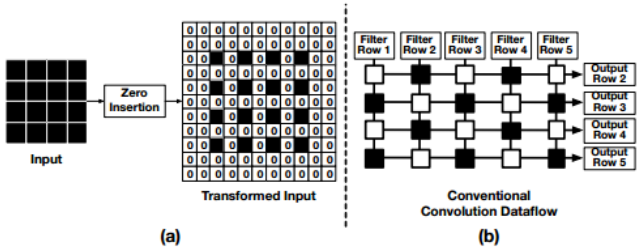


Figure 3: (a) Zero-insertion step in TranConv **operation for a 4×4 input and the transformed input. The white-colored squares represent zero values in the transformed input. (b) Using convolution dataflow for performing** TranConv **operations.**

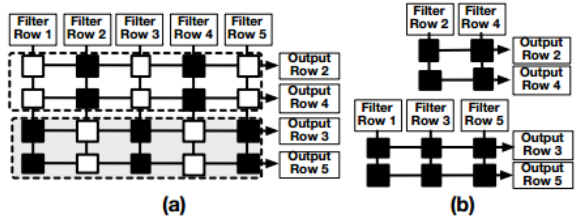


Figure 4: The flow of data after applying (a) output row reorganizaton and (b) filter row reorganization. The combination of these flow optimizations reduces the idle (white) operations and improves the resource utilization.

Scale–Out Acceleration for Machine Learning

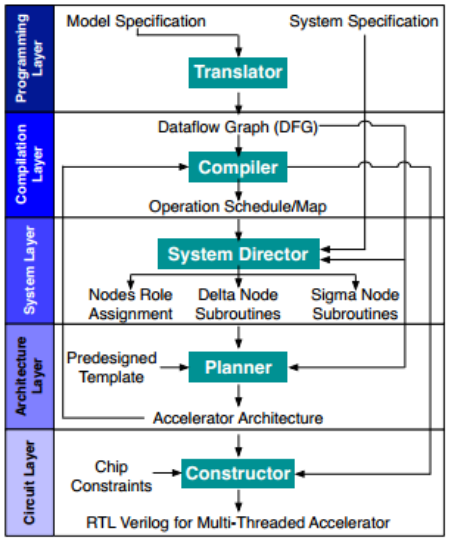
· Design CoSMIC: an entire stack of layers to execute a wide range of learning algorithms on accelerator-augmented scale-out systems

· Theoretical foundation of disturbed learning——Parallelizing Stochastic Optimization

$$\underset{j:1\rightarrow n}{\textbf{Parallel}}\ \langle \theta_j^{(t+1)} = \textbf{SGD}\left(\{XY_1,\ldots,XY_b\},\theta^{(t)},f\right)\rangle$$

$$\theta^{(t+1)} = \frac{\sum_j \theta_j^{(t+1)}}{n}$$

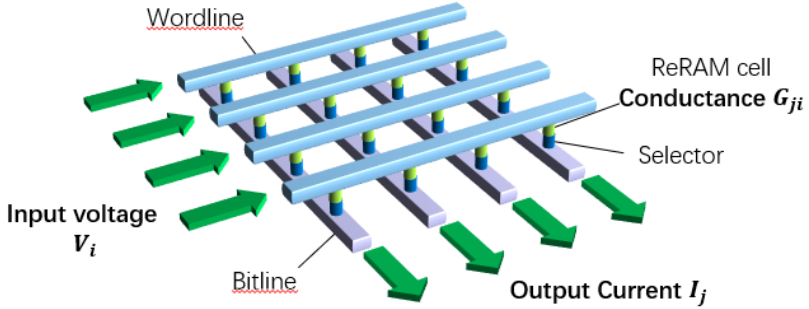
Scale–Out Acceleration for Machine Learning



Programming Layer	Algorithmic Specification	Partial Gradient
		Aggregation Operator
		Mini-Batch Size
	System Specification	Number of Nodes
		Number of Groups
		Accelerator Type
Compilation Layer	Translator	Dataflow Graph (DFG)
	Compiler	Operation Schedule/Map
System Layer	System Director	Node Roles
	System Subroutines: Delta Nodes	Accelerator Invocation Module
		Module for Communication with Sigma Node
	System Subroutines: Sigma Nodes	Accelerator Invocation Module
		Networking Thread Pool for Communication with Delta Nodes
		Circular Buffer for Consumer-Producer Networking & Aggregation
		Aggregation Thread Pool
		Module for Communication with Next Level of Hierarchy Node
Architecture Layer	Hand-Optimized Template Design	RTL Verilog
	Performance Estimation Tool	Design Space of Possible Architectures
	Planner	Number of Threads
		Resources per Thread
		Accelerator Datapath
Circuit Layer	Constructor	RTL Verilog of the Multi-Threaded Accelerator

Figure 3: The full CoSMIC stack.

Yu Ji



$$I_j = \sum_i G_{ji} V_i \quad \Rightarrow \quad I = GV$$

Latency	$10ps$ for 100×100 crossbar (RC delay)
Area	$4F^2$ for each cell

- PhD, Computer Architecture
- Tsinghua University, 2015-now
- Advisor: Youhui Zhang, Yuan Xie
- Research Interests
  - Neuromorphic Hardware, ReRAM
    - *NEUTRAMS: Neural network transformation and co-design under neuromorphic hardware constraints(MICRO 2016)*
    - *Bridge the Gap between Neural Networks and Neuromorphic Hardware with a Neural Network Compiler(ASPLOS 2018)*
    - *FPSA: A Full System Stack Solution for Reconfigurable ReRAM-based NN Accelerator Architecture(ASPLOS 2019)*

- NN Attack from the perspective of computer Architecture(not the first author)
  - *Memory Trojan Attack on Neural Network Accelerators(DATE 2019)*
  - *DeepSniffer: A DNN Model Extraction Framework Based on Learning Architectural Hints(ASPLOS 2020)*
  - *Programmable Neural Network Trojan for Pre-Trained Feature Extractor*

See More Info in attached materials.