# Automatic Compiler Based FPGA Accelerator for CNN Training

Shreyas Venkataramanaiah[1], Yufei Ma[1] ,
Shihui Yin[1], Eriko Nurvithadhi[2], Aravind Dasu[3],
Yu Cao[1], Jae-sun Seo[1]

[1] School of ECEE, Arizona State University, Tempe, AZ, USA
[2] Intel Labs, Intel Corporation, OR, USA
[3] Programmable Solutions Group, Intel Corporation, CA, USA

## Outline

- Introduction
- CNN training algorithm
- RTL compiler
- CNN training accelerator
- Results
- Conclusion

## Introduction

- Challenges in training of neural networks
  - Large storage, memory bandwidth, energy consumption
  - New DNN structures rapidly evolving and developed for diverse applications
- GPU’s are power hungry
- ASIC not good for programmability, cannot predict future DNNs
- FPGA’s are flexible
  - Reconfigurable, scalable training hardware
  - Can support low-precision or sparse matrix computations

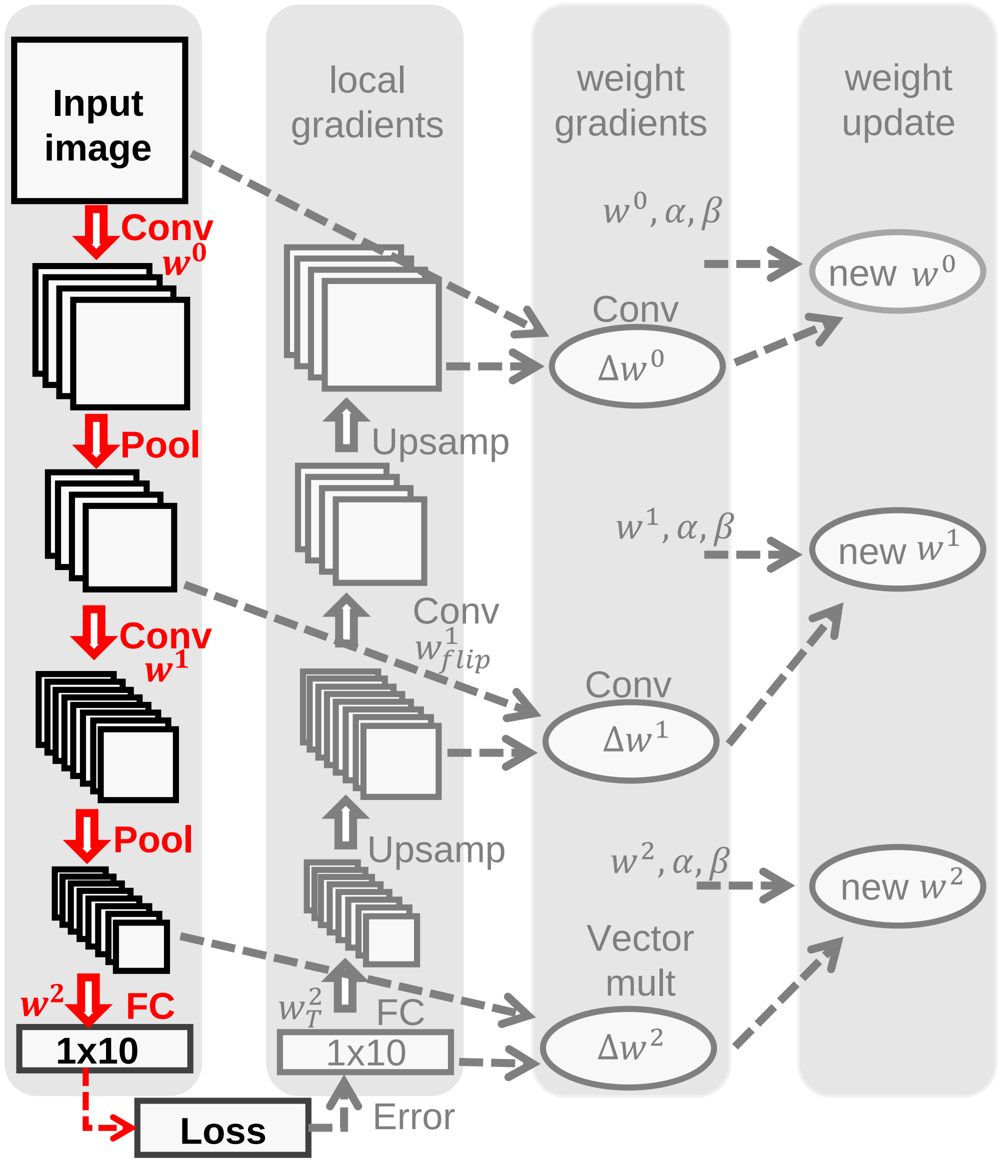
## Outline

- Introduction
- **CNN training algorithm**
- RTL compiler
- CNN training accelerator
- Results
- Conclusion

# CNN Training Algorithm

## Forward Pass

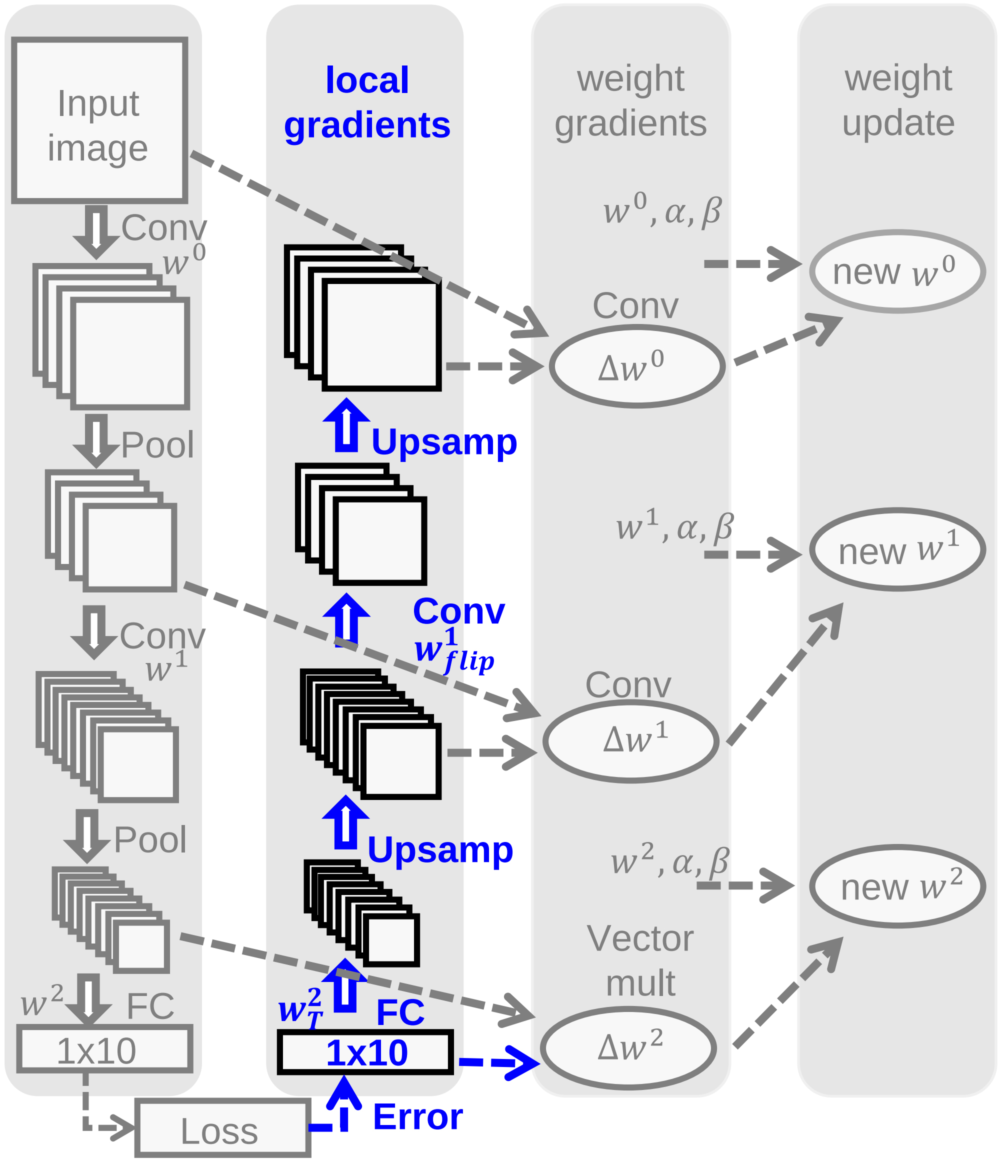
- Each training image is associated with a label
- Loss function estimates the network performance and provides error value
- ReLU: Store the activation gradients
- Maxpool: Store the selected pixel position



# CNN Training Algorithm

## Backward Pass

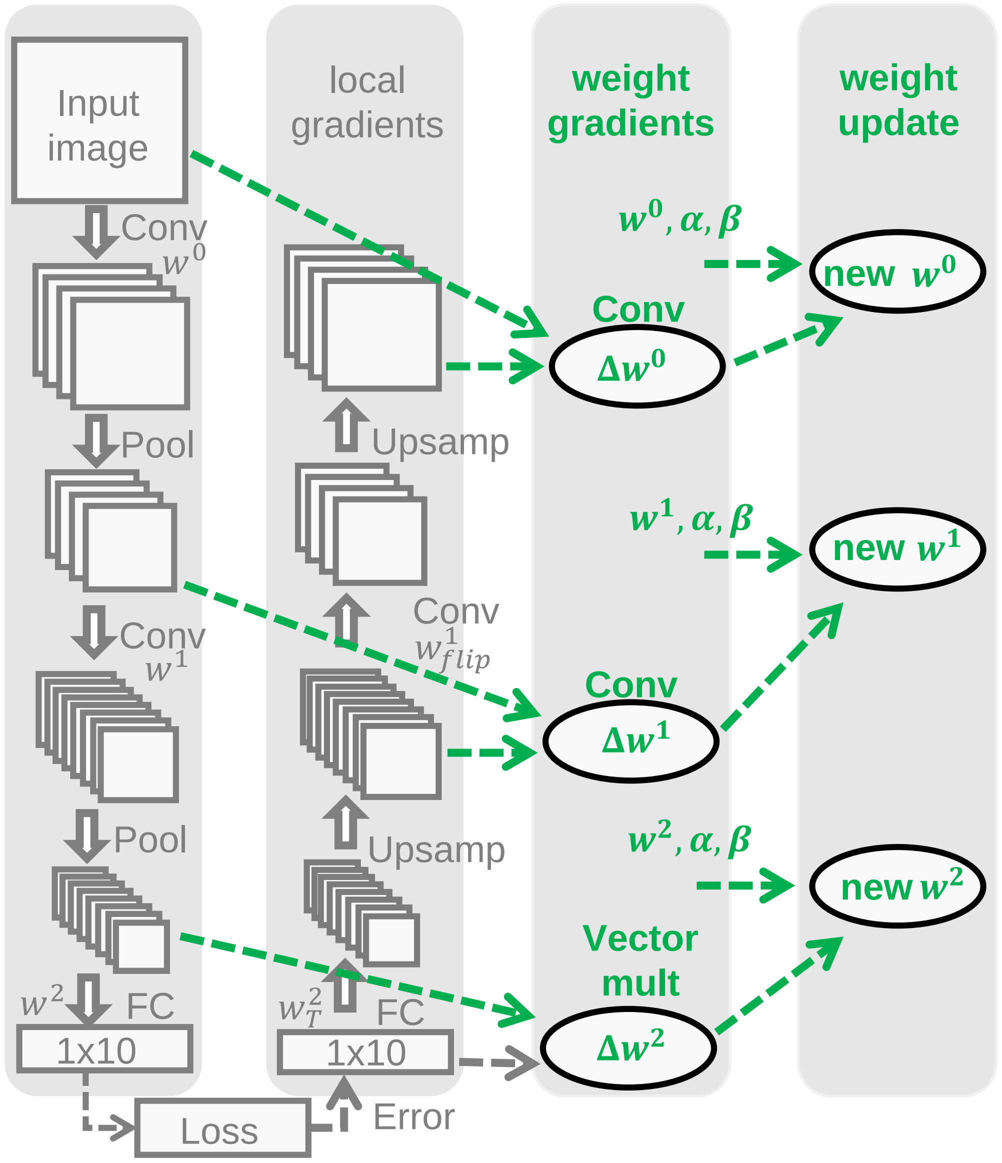
- Error values are propagated back in the network
- Flipped kernels are used in convolutions
- ReLU : gradients are scaled by activation gradients
- Maxpool: Upsample the image using pooling indices



# CNN Training Algorithm

## Weight Update

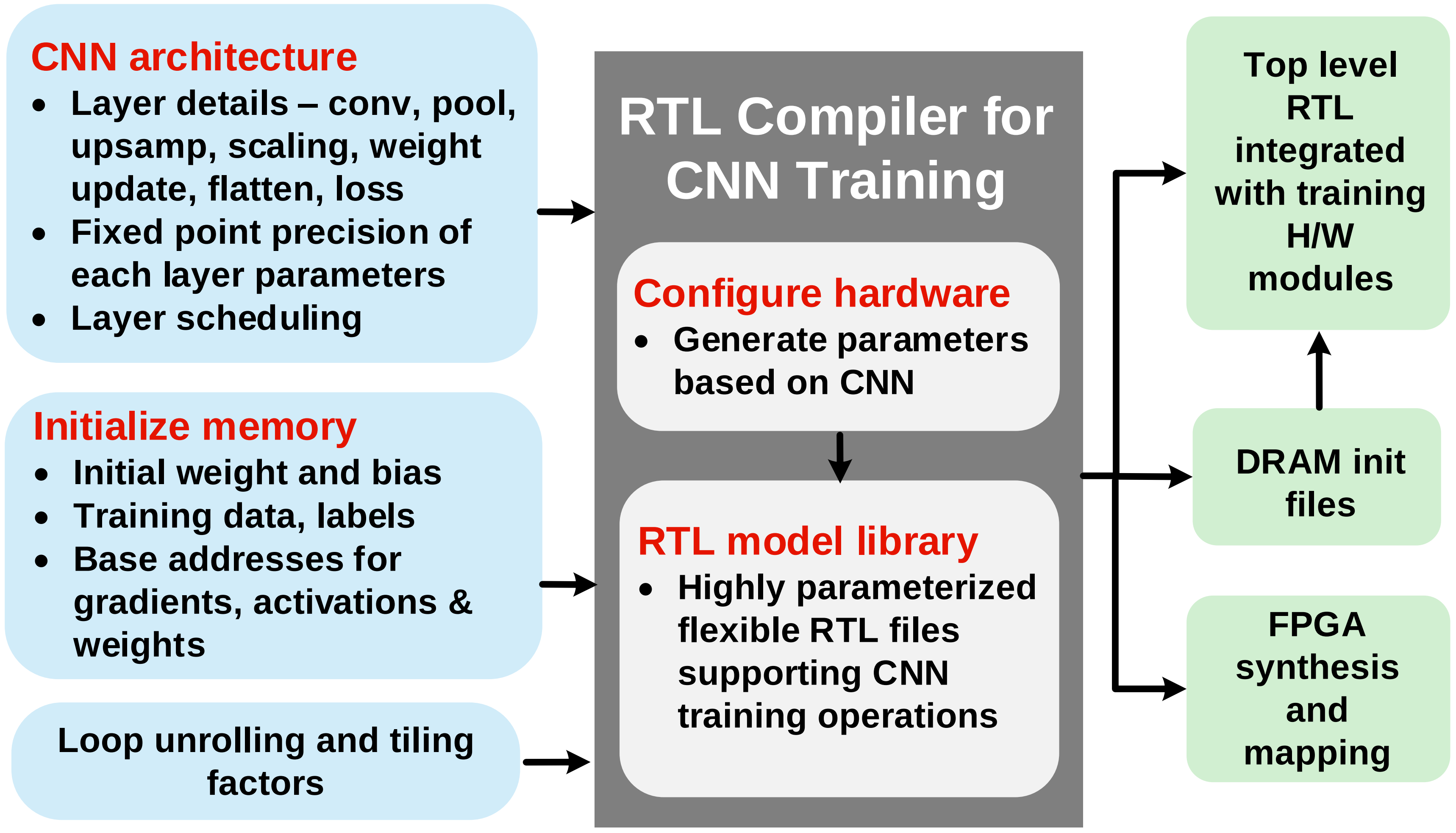
- Weight gradients are computed and accumulated
- Convolutions involve intra tile accumulations
- New weights are computed at the end of batch
- Learning rate ($\alpha$) and momentum ($\beta$) parameters are used



## Outline

- Introduction
- CNN training algorithm
- **RTL compiler**
- CNN training accelerator
- Results
- Conclusion

# Proposed RTL Compiler

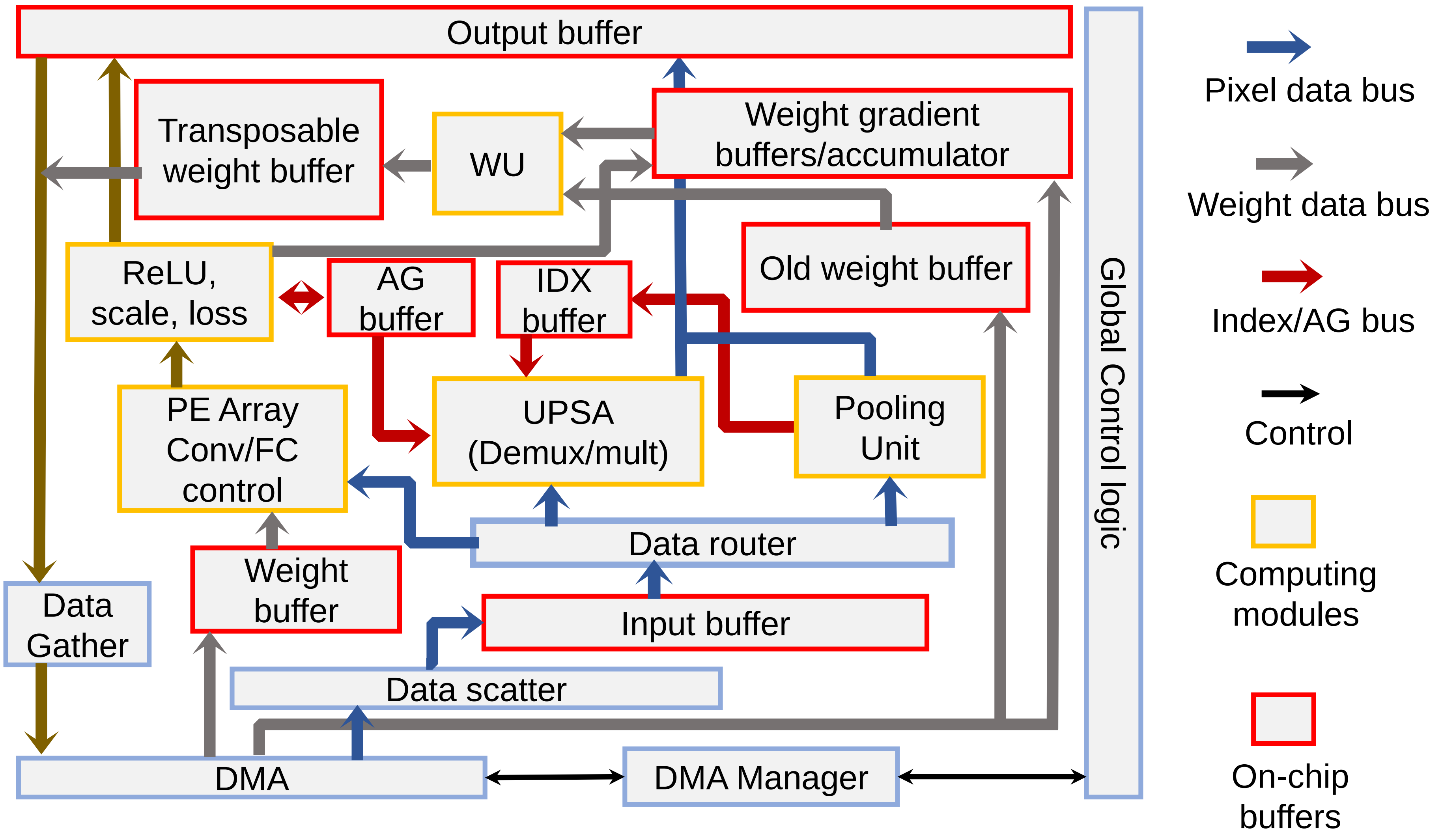

CNN architecture
• Layer details – conv, pool, upsamp, scaling, weight update, flatten, loss
• Fixed point precision of each layer parameters
• Layer scheduling
Initialize memory
• Initial weight and bias
• Training data, labels
• Base addresses for gradients, activations & weights
Loop unrolling and tiling factors
RTL Compiler for CNN Training
Configure hardware
• Generate parameters based on CNN
RTL model library
• Highly parameterized flexible RTL files supporting CNN training operations
Top level RTL integrated with training H/W modules
DRAM init files
FPGA synthesis and mapping

RTL compiler generates the training accelerator using high level CNN description

## Outline

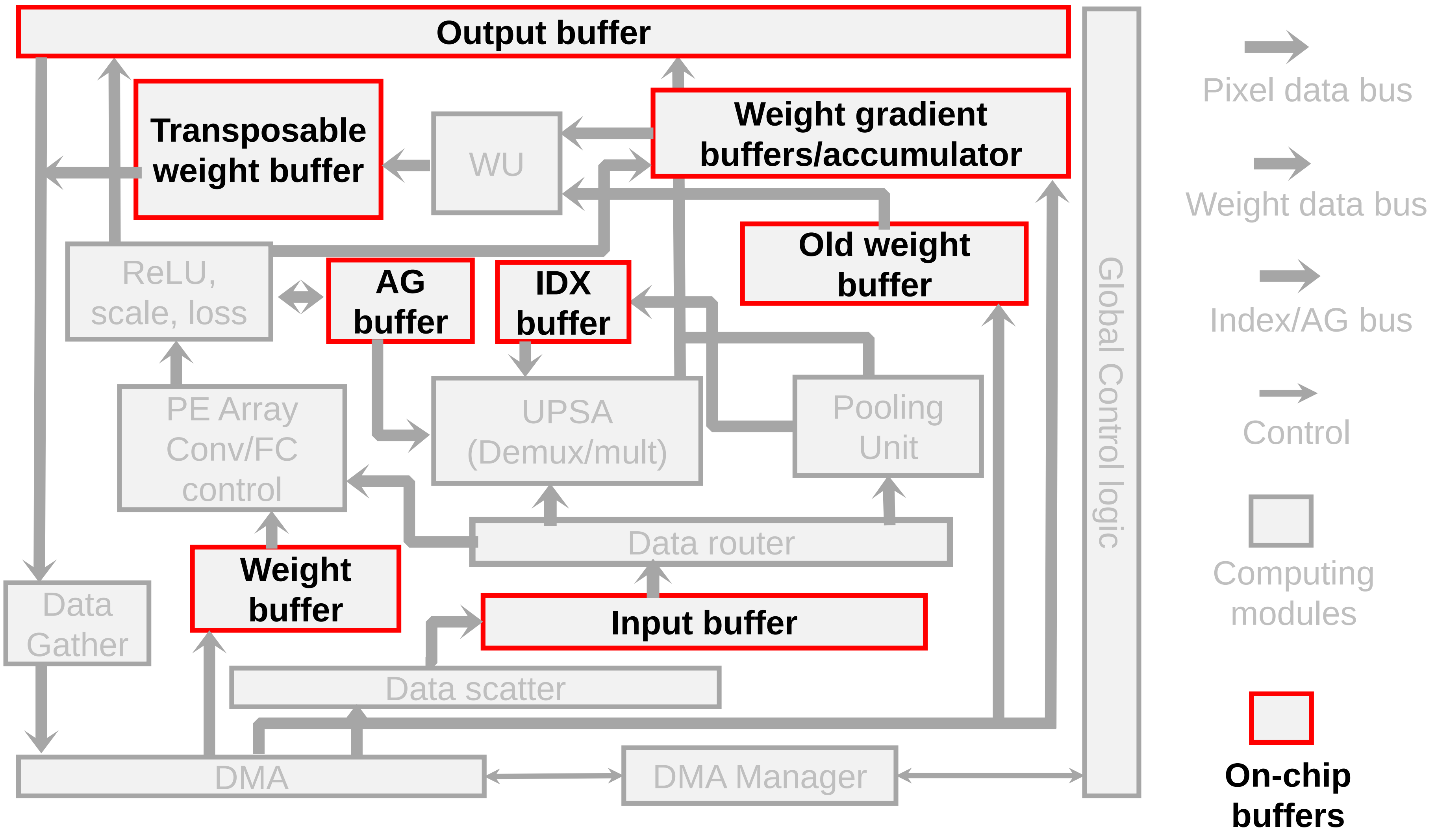
- Introduction
- CNN training algorithm
- RTL compiler
- **CNN training accelerator**
- Results
- Conclusion

# Overall Architecture



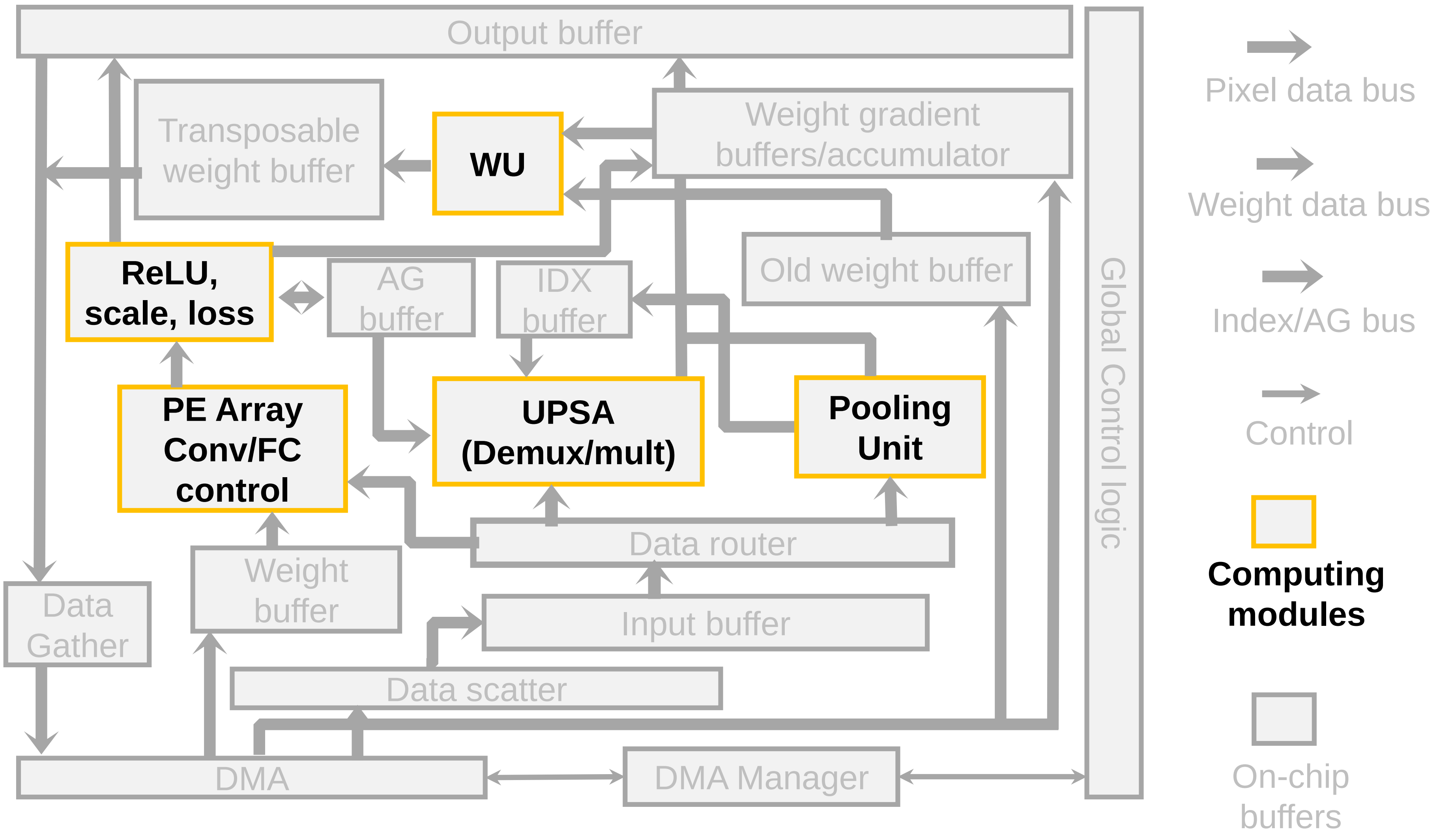
AG – Activation gradients, IDX – Maxpool Indices, UPSA – Upsampling

# Overall Architecture



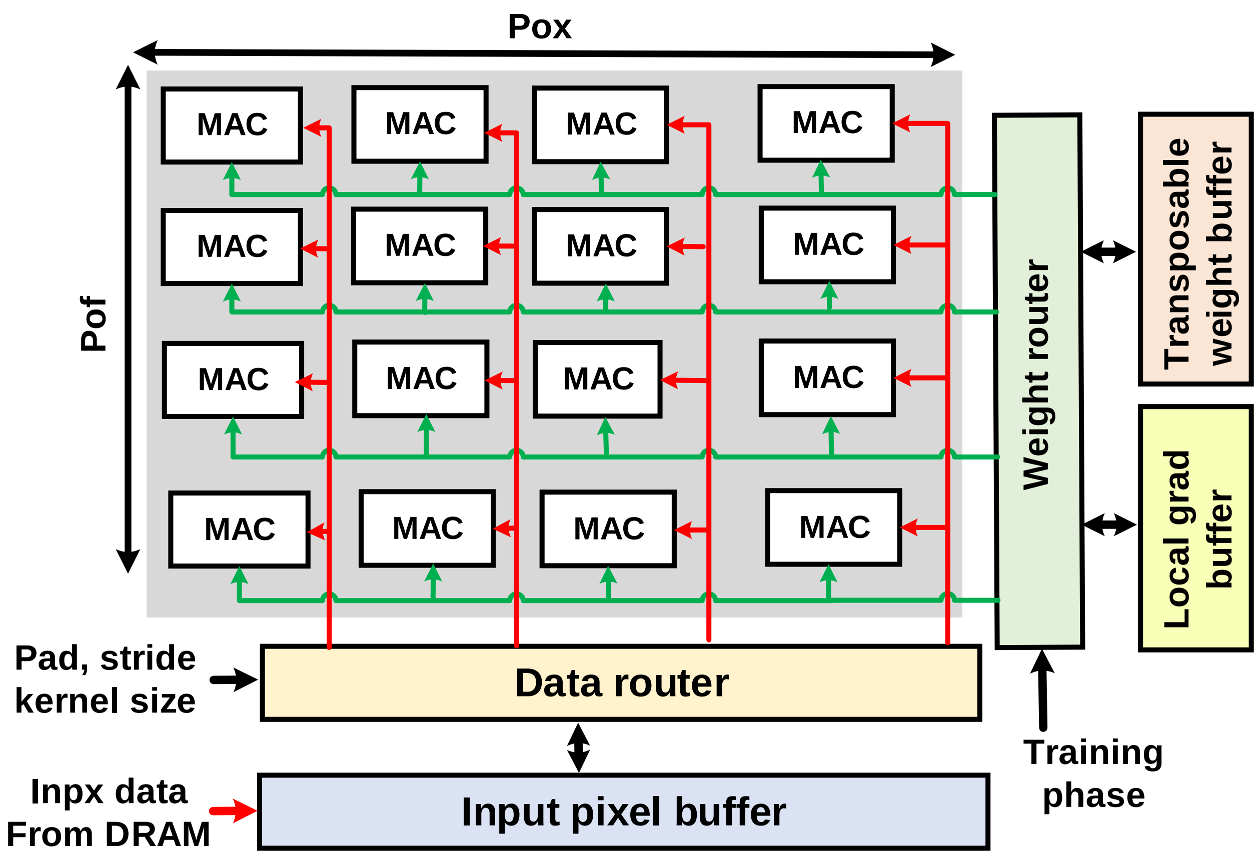
AG – Activation gradients, IDX – Maxpool Indices, UPSA – Upsampling

# Overall Architecture



AG – Activation gradients, IDX – Maxpool Indices, UPSA – Upsampling

# MAC Array



- Output stationary dataflow
- Data/weight re-use to minimize partial sum movement
- Reconfigurable MAC array to support all phases of training
- MAC array size is user determined – loop unroll factors $(P_{of}, P_{ox}, P_{oy})$

| Training Phase | Input px buffer | Weight buffer | Output buffer |
|---|---|---|---|
| FP | Activations | Normal kernels | Activations |
| BP | Local gradients | Flipped kernels | Local gradients |
| WU | Activations | Local gradients | Kernel gradients |

# Transposable Weight Buffers

## FP weight access pattern

Out Feat. Maps (L+1)

Inp Feat. Maps (L)

| | | | |
|---|---|---|---|
| 101 | 102 | 103 | 104 |
| 201 | 202 | 203 | 204 |
| 301 | 302 | 303 | 304 |
| 401 | 402 | 403 | 404 |

Transpose

## BP weight access pattern

Inp Feat. Maps (L)

Out Feat. Maps (L+1)

| | | | |
|---|---|---|---|
| 101 | 201 | 301 | 401 |
| 102 | 202 | 302 | 402 |
| 103 | 203 | 303 | 403 |
| 104 | 204 | 304 | 404 |

## Transposable weight storage

| C0 | C1 | C2 | C3 |
|---|---|---|---|
| 101 | 102 | 103 | 104 |
| 204 | 201 | 202 | 203 |
| 303 | 304 | 301 | 302 |
| 402 | 403 | 404 | 401 |

Independent column buffers

Block circulant matrix

| Training stage | Read address | | | |
|---|---|---|---|---|
| | C0 | C1 | C2 | C3 |
| FP | 0 | 0 | 0 | 0 |
| BP | 0 | 1 | 2 | 3 |

Read controls to transposable buffer during FP, BP, WU

## Outline

- Introduction
- CNN training algorithm
- RTL compiler
- CNN training accelerator
- **Results**
- Conclusion

# Results

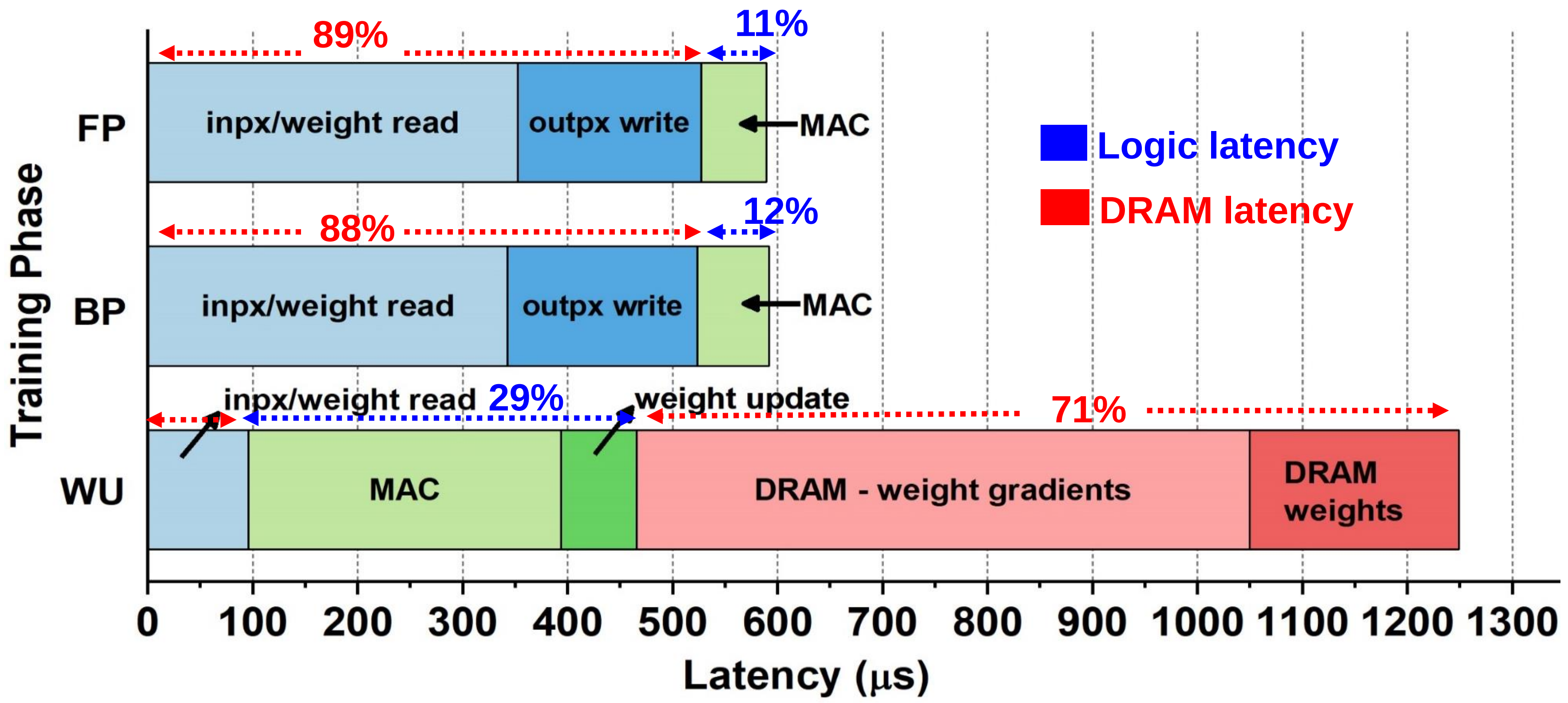
CIFAR-10 1X: 2(16C3)-MP-2(32C3)-MP-2(64C3)-MP-FC

| CNN | Resource | | | Latency per epoch (s) | | |
|---|---|---|---|---|---|---|
| | DSP | ALM | BRAM | BS-10 | BS-20 | BS-40 |
| CIFAR-10 1X | 30% | 19% | 4.4% | 18.2 | 18 | 18.01 |
| CIFAR-10 2X | 58% | 44% | 9.5% | 41.7 | 41.3 | 41 |
| CIFAR-10 4X | 100% | 76.2% | 22.4% | 98.2 | 96.8 | 96.18 |

| | Throughput (GOPs) | | | Efficiency (GOPs/W) | | |
|---|---|---|---|---|---|---|
| Device | Titan XP | | FPGA | Titan XP | | FPGA |
| Batch size | 1 | 40 | (1/40) | 1 | 40 | (1/40) |
| CIFAR-10 1X | 45.6 | 551.8 | 163 | 0.5 | 3.7 | 7.9 |
| CIFAR-10 2X | 128.8 | 1337.9 | 282 | 1.3 | 8.3 | 8.59 |
| CIFAR-10 4X | 331.4 | 2353.7 | 479 | 2.9 | 13.5 | 9.49 |

- Peak throughput of 479 GOPs
- Better energy efficiency than GPU's for smaller batch sizes
- Limited by DRAM B/W
- Images in a batch are processed sequentially

# Latency Breakdown



- Latency of CIFAR-10 4X CNN for one iteration of a batch
- Overall ~20% logic latency and ~80% due to DRAM access
- Weight update phase is memory intense
  - Contributes for ~51% of the overall latency

## Outline

- Introduction
- CNN training algorithm
- RTL compiler
- CNN training accelerator
- Results
- **Conclusion**

## Conclusion

- Automatic RTL compiler-based CNN training accelerator
- Implemented parameterized RTL library to support CNN training operations
- Evaluated performance on Intel Stratix-10 GX FPGA for three CNNs for CIFAR-10 dataset
- Achieved 479 GOPs throughput

## Acknowledgements

***C-BRIC*** (Center for BRain-Inspired Computing)

We thank **Intel Corporation** for supporting and funding this research work. This work was also partially supported by **NSF grant** 1652866 and **C-BRIC**, one of six centers in JUMP, an SRC program sponsored by DARPA