Optimizing FPGA-Based Accelerator Design for Deep Convolutional Neural Network

Chen Zhang, Peng Li, Guangyu Sun, et al

Presented by Zhuangwei Zhuang

South China University of Technology

August 14, 2017

Content

Motivation
Background
Method
Exploration
Implementation
Experiment and Result
Conclusion

1 Motivation

2 Background

3 Method

4 Exploration

5 Implementation

6 Experiment and Result

7 Conclusion

Motivation
Acceleration Platforms

Convolutional neural network (CNN) has been widely employed for image recognition for its ability to achieve high accuracy.

Recently, various FPGA-based accelerators for deep CNN have been proposed.

Platforms to Accelerate CNN

- FPGA - Field Programmable Gate Array
- GPU - Graphics Processing Unit
- ASIC - Application Specific Integrated Circuit

Motivation

Compare FPGA with GPU

Table: Projected ImageNet 1K Performance and Power[1]

Platform	ImageNet 1K (Image/s)	Max Device Power (W)	Efficiency (Image/Sec/W)
Catapult Server + Stratix V D5	134	25	5.36
Catapult Server + Arria 10 GX1150	233	25	9.32
Caffe + cuDNN on Tesla K20	376	235	1.6
Caffe + cuDNN on Tesla K40	500∼824	235	2.13∼3.51

Table: Projected AlexNet Performance and Power[2]

CNN Classification Platform	Performance(Image/s)	Power(W)	Efficiency (Image/Sec/W)
E52699 Dual Xeon CPU (18 core per Xeon)	1320	321	4.11
PCIe w/Dual Arria 10 1150	1200	130	9.27

[1] Kalin et al.Accelerating Deep Convolutional Neural Networks Using Specialized Hardware

[2] Xilinx®.Efficient Implementation of Neural Network Systems Built on FPGAs,Programmed with OpenCL

Motivation

FPGA Structure

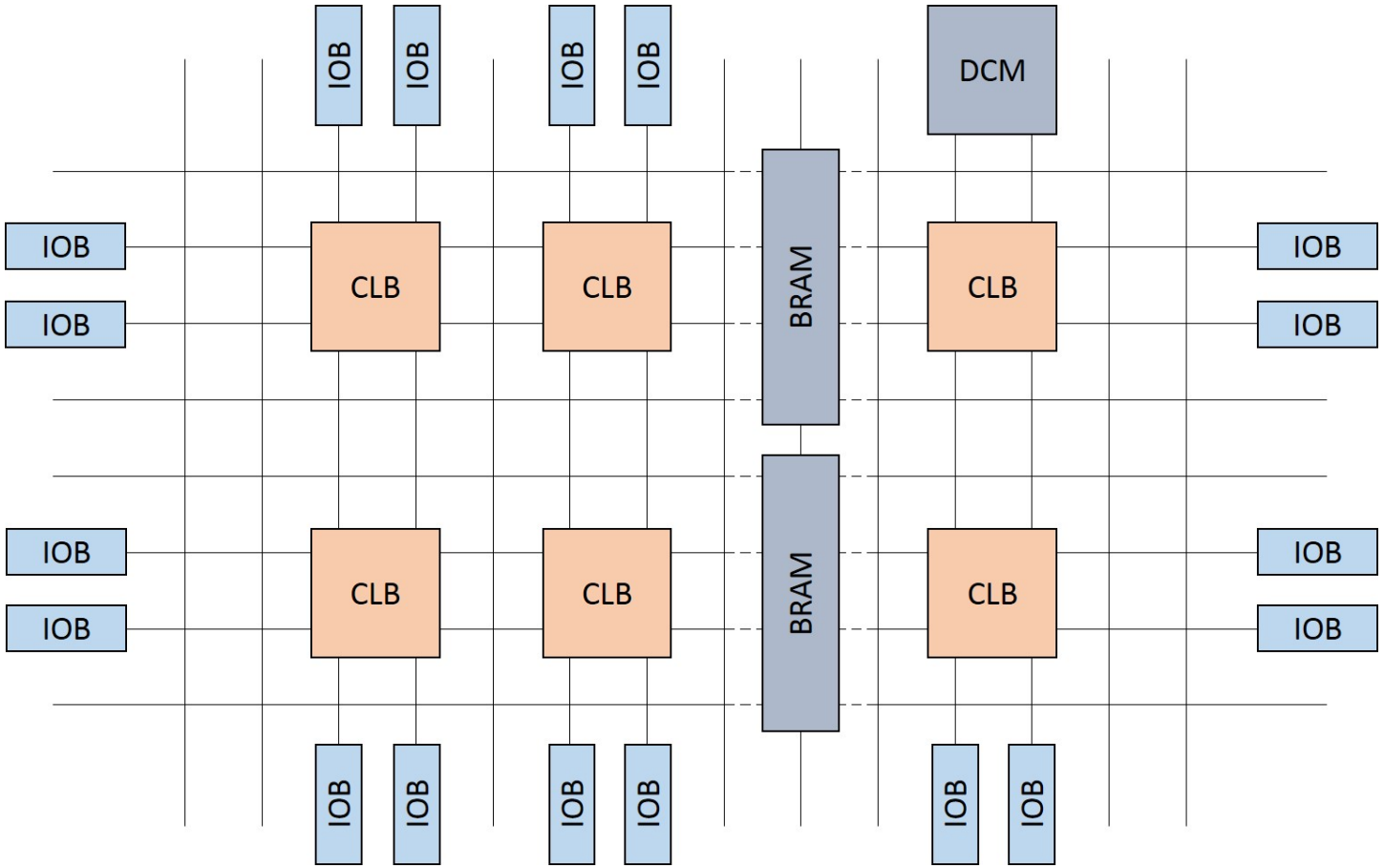


Figure: FPGA Structure

Motivation

FPGA-based Accelerator for Deep CNN

Advantages

- High performance
- High energy efficiency
- Capability of Reconfiguration
- Fast development round

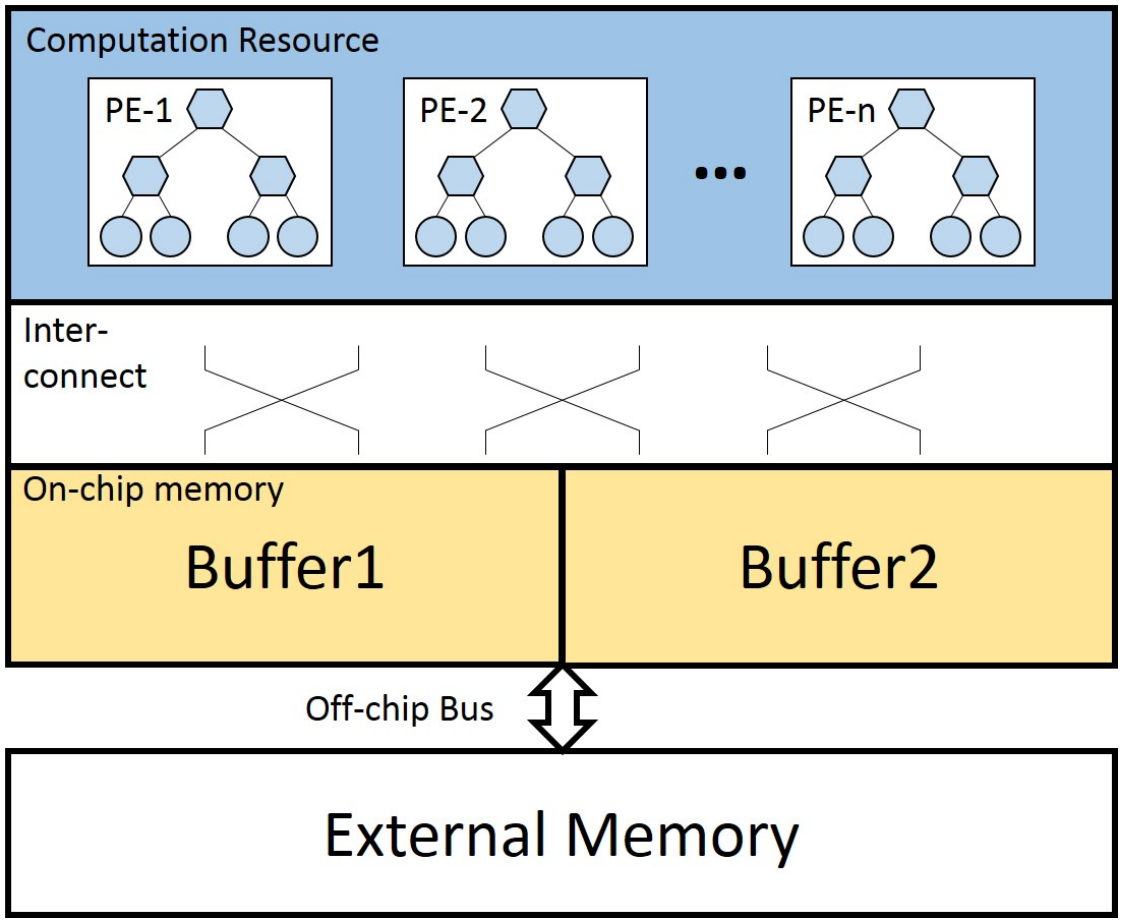
Disadvantages

- Limited computation resource
- Limited memory bandwidth

Motivation
Problem

There would be as much as **90%** performance difference between two different solutions with the same logic resource utilization.

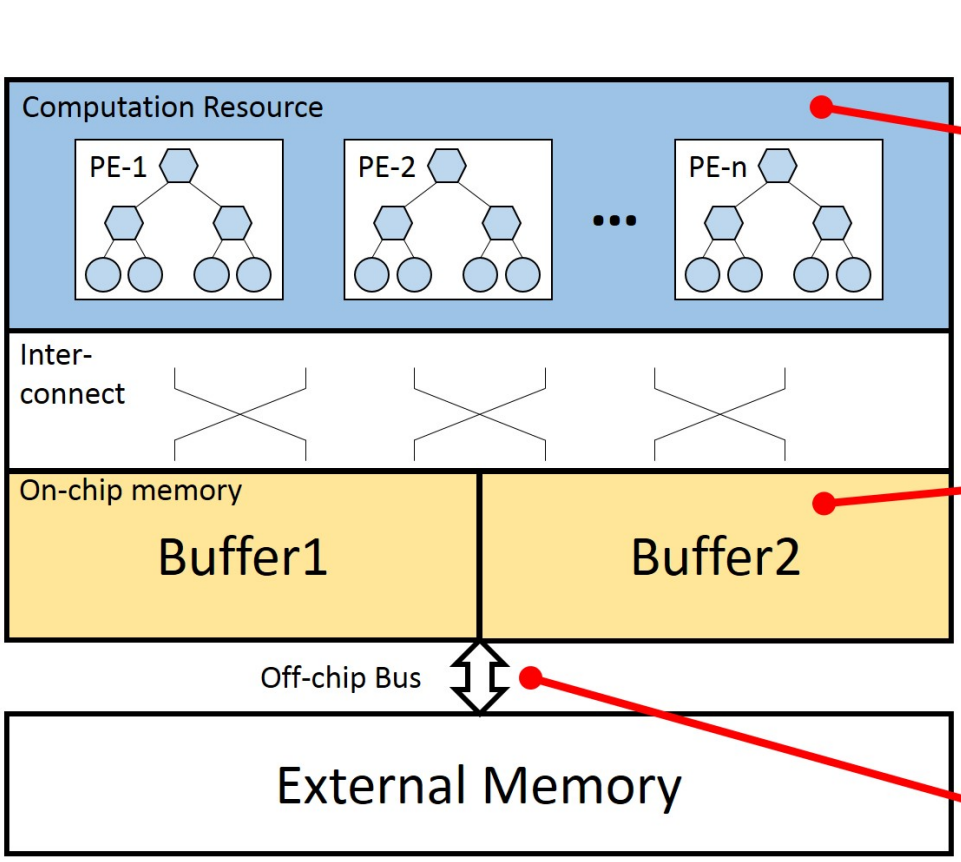
FPGA
Chen Zhang,et al
Motivation
Background
Method
Exploration
Implementation
Experiment and Result
Conclusion



FPGA-based Accelerator Structure

Background

FPGA Design in Roofline Model



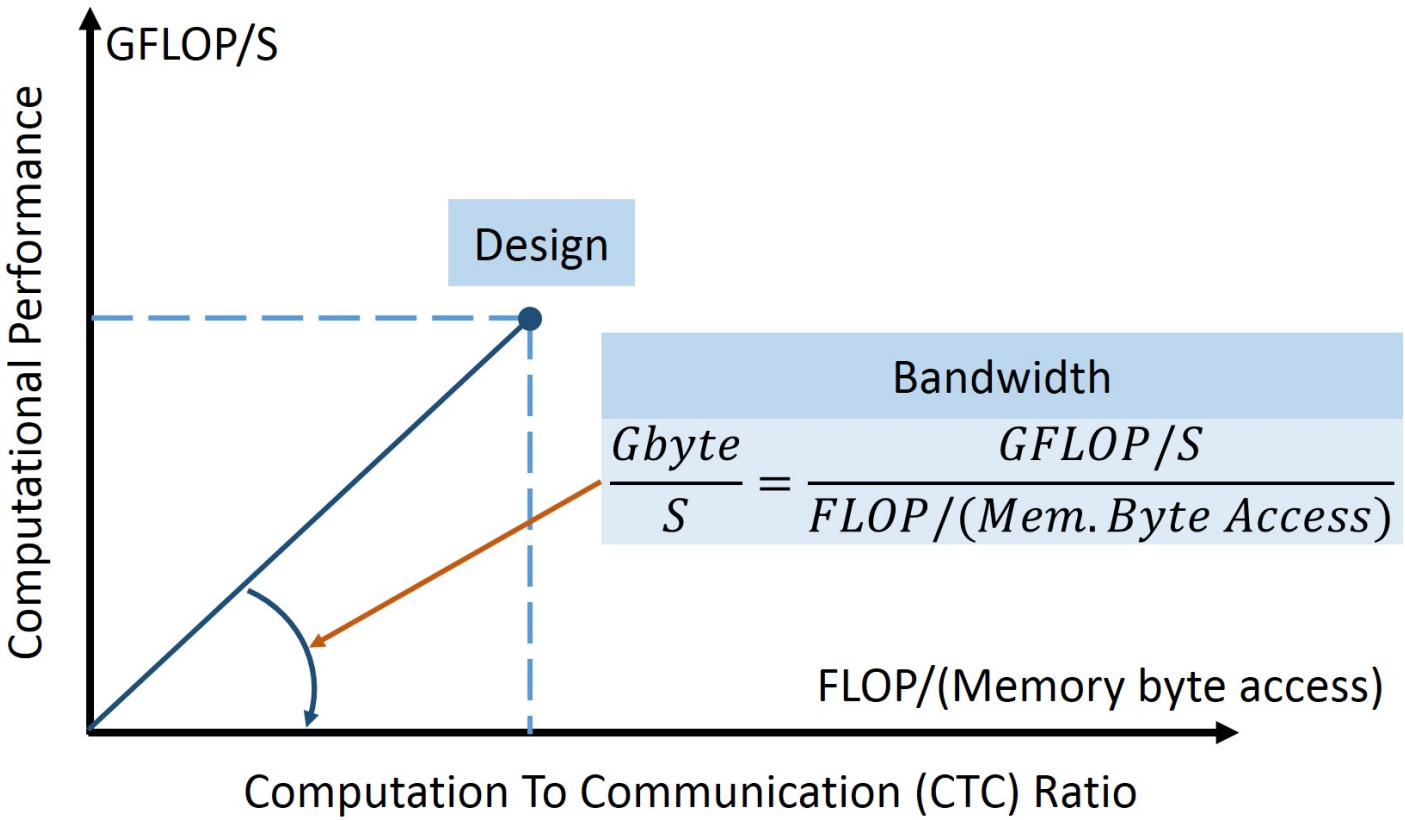
Computational Perf.
$= \frac{total\ number\ of\ operations}{execution\ cycles}$
GFLOP/S

Computation To Communication Ratio
$= \frac{total\ number\ of\ operations}{Amount\ of\ external\ data\ access}$
FLOP/(Mem. Byte Access)

Bandwidth
Gbyte/S

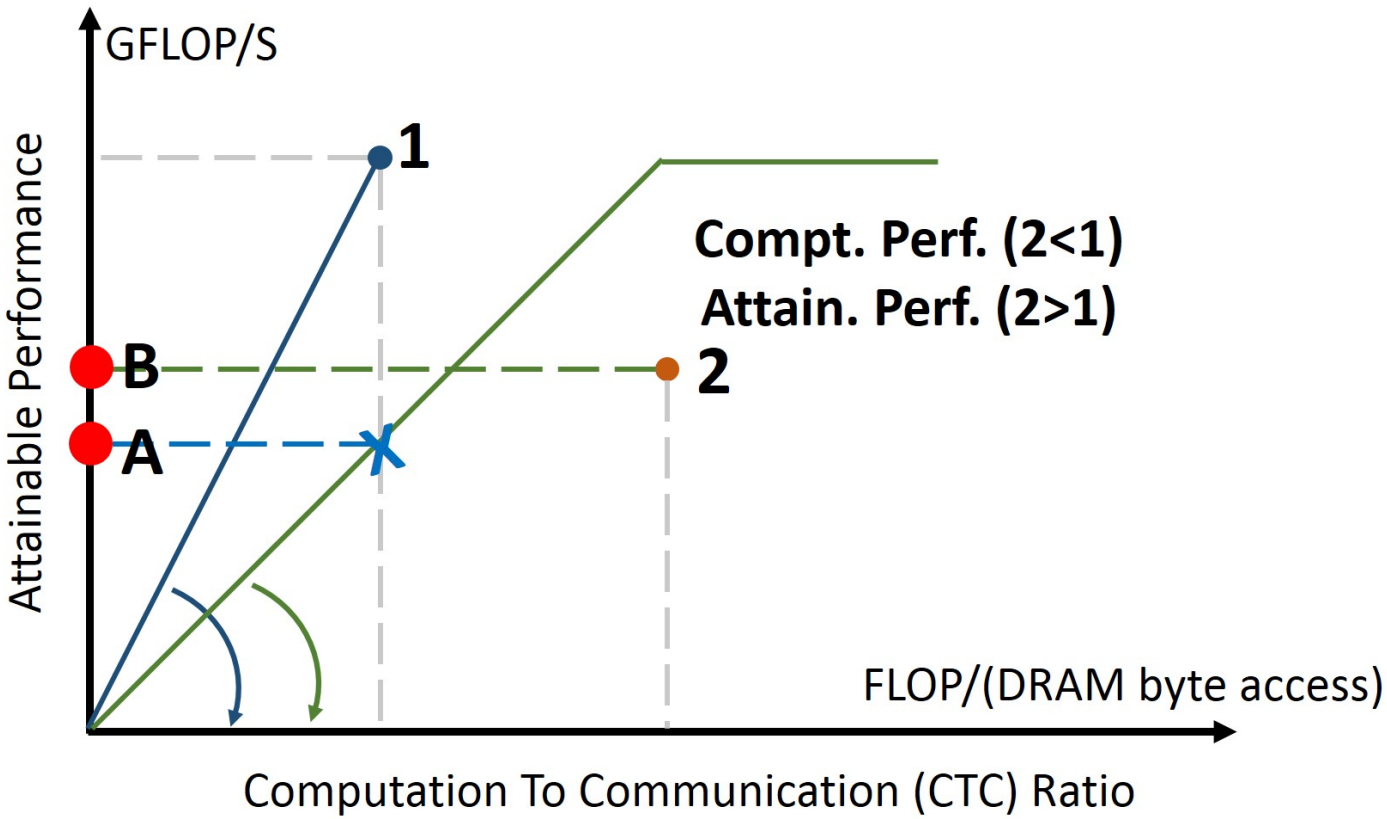
Background

FPGA Design in Roofline Model



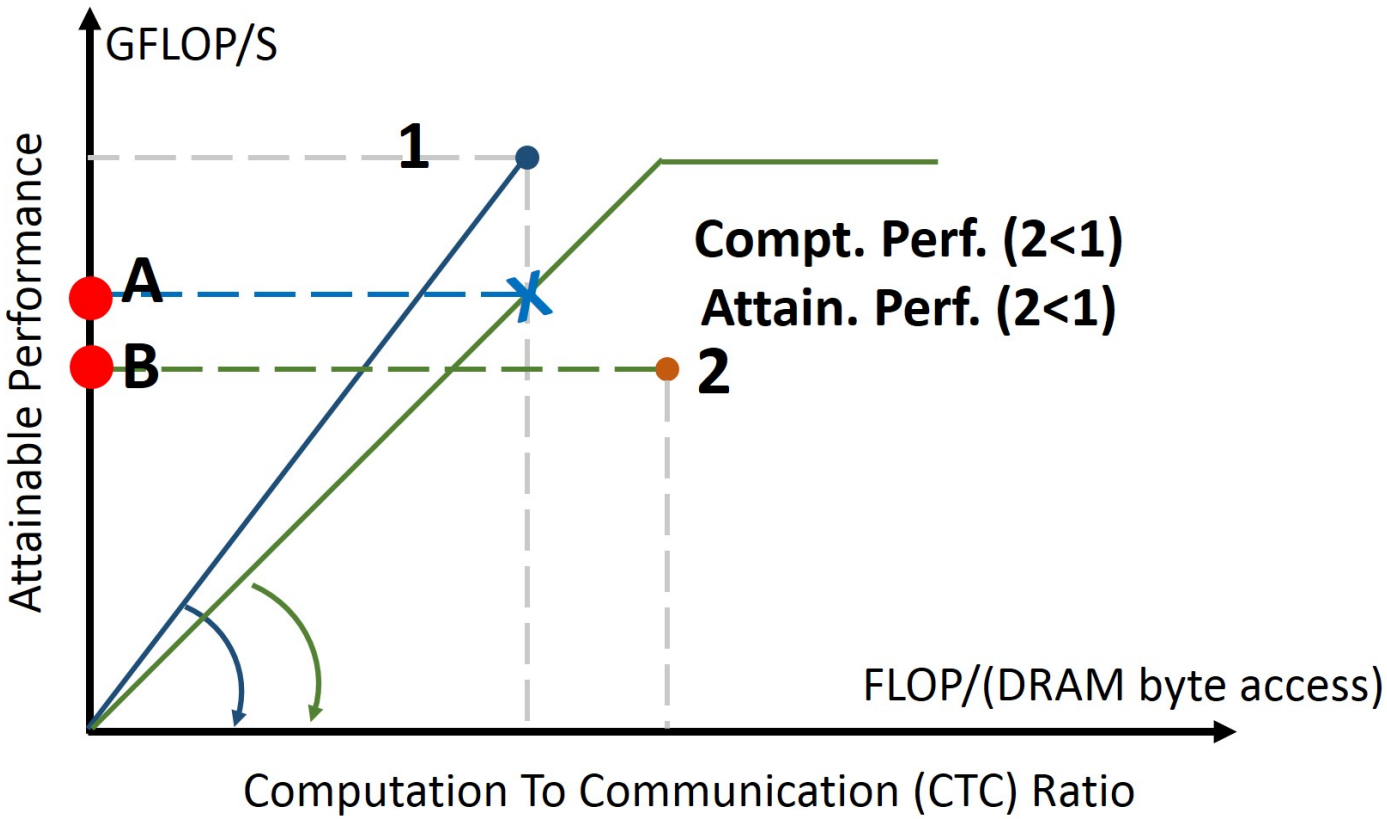
Background

Attainable Performance vs. Computation performance



Background

Attainable Performance vs. Computation performance



FPGA

Chen Zhang,et al

Motivation

Background

Method

Exploration

Implementation

Experiment and Result

Conclusion

Optimizing Target:

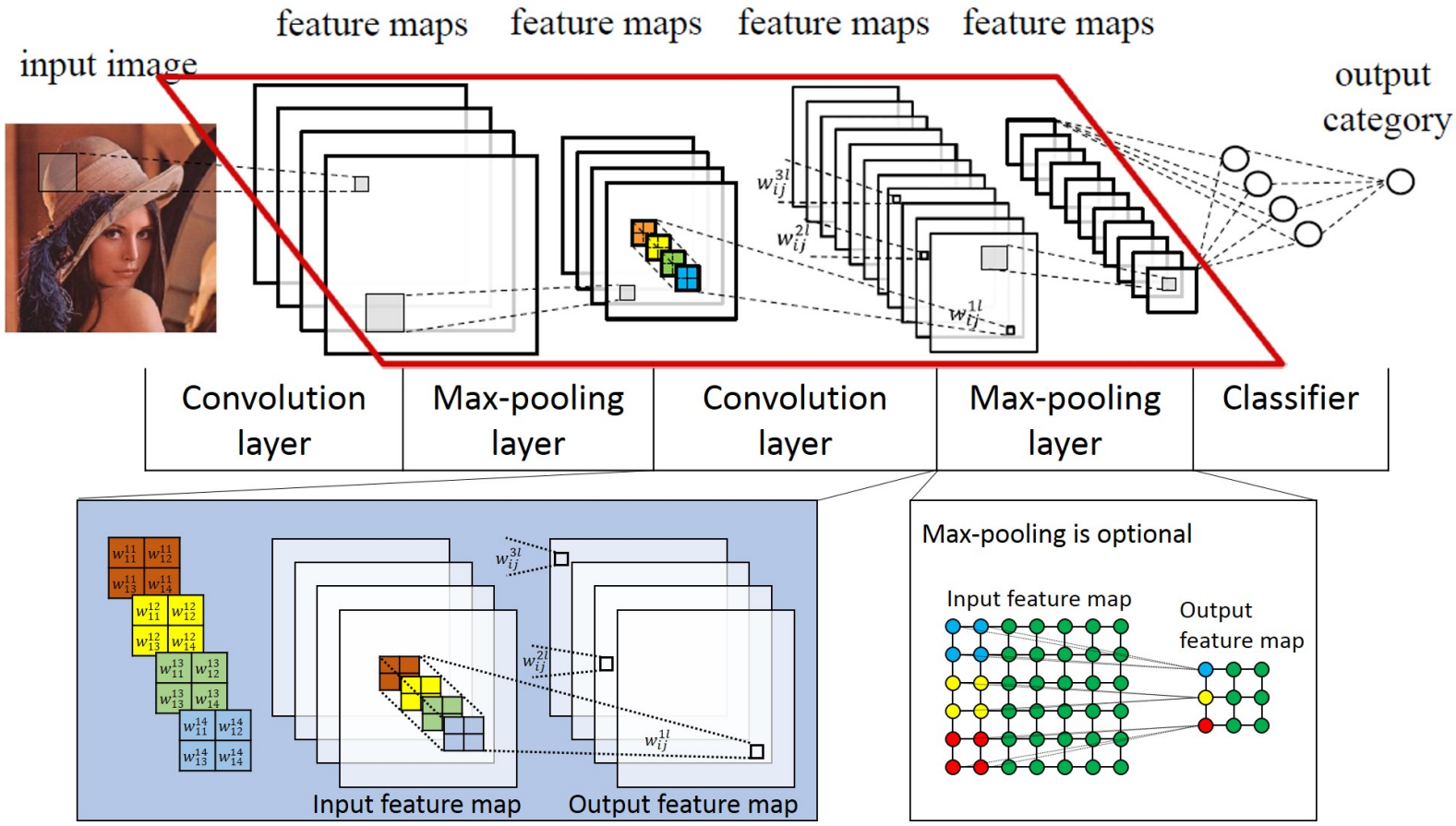
Find a design with maximized attainable performance

Optimizing Methods

- Loop Unroll
- Pipeline
- Loop Tiling
- Data Reuse

Background

Convolutional Neural Network



Background
Convolutional Neural Network

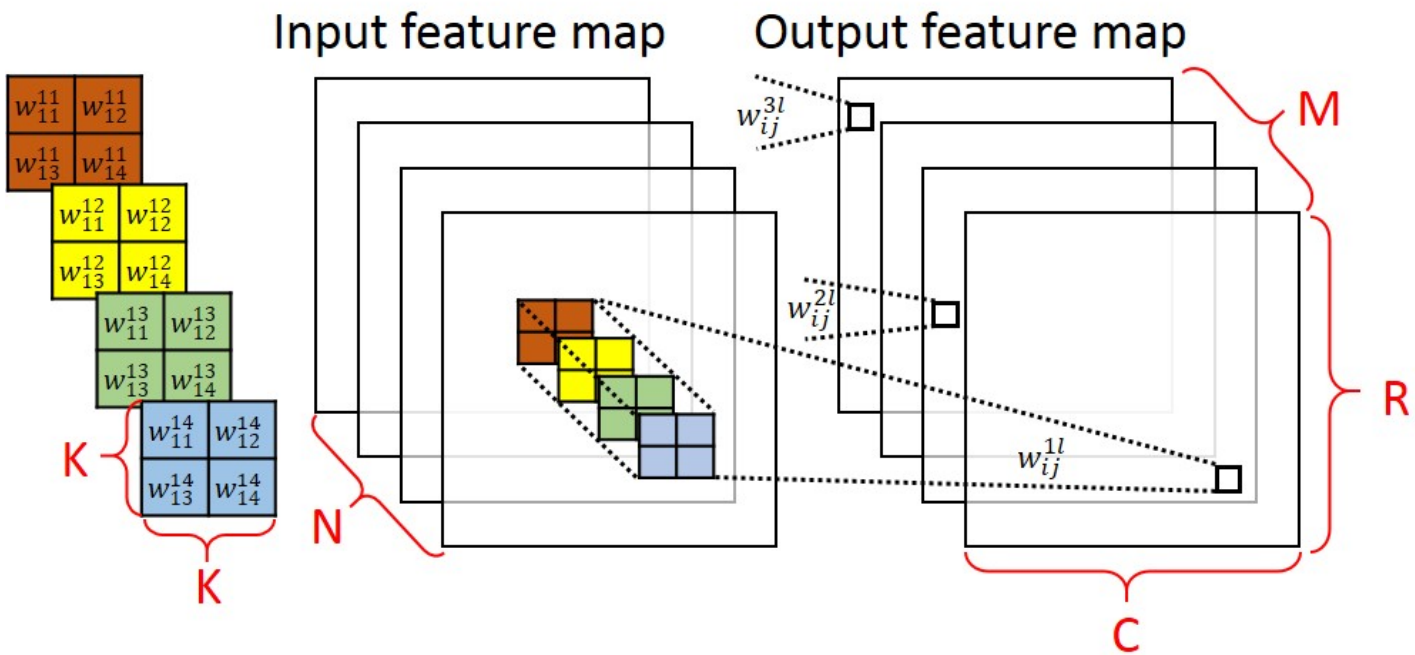
Convolutional layers account for over **90%** computation[3,4]

[3] A. Krizhevsky, etc. Imagenet classification with deep convolutional neural networks. NIPS 2012.

[4] J. Cong and B. Xiao. Minimizing computation in convolutional neural networks. ICANN 2014.

Method

Loop Unroll



$$Y[m][r][c] = \sum_{n=0}^{N-1} \sum_{i=0}^{K-1} \sum_{j=0}^{K-1} W[m][n][i][j] \times X[n][S \times r + i][S \times c + j] \quad (1)$$

FPGA

Chen Zhang,et al

Motivation

Background

Method

Exploration

Implementation

Experiment and Result

Conclusion

Pseudo code of a convolutional layer

```
1 for(row=0;row<R;row++){
2   for(col=0;col<C;col++){
3     for(to=0;to<M;to++){
4       for(ti=0;ti<N;ti++){
5         for(i=0;i<K;i++){
6           for(j=0;j<K;j++){
              output_fm[to][row][col] +=
              weights[to][i][j]*input_fm[ti][S*row+i][S*col+j];
}}}}}}
```

FPGA

Chen Zhang,et al

Motivation

Background

Method

Exploration

Implementation

Experiment and Result

Conclusion

Pseudo code of a tiled convolutional layer

External data transfer (Tile Loop)

```
1 for(row=0;row<R;row+=Tr){
2    for(col=0;col<C;col+=Tc){
3       for(to=0;to<M;to+=Tm){
4          for(ti=0;ti<N;ti+=Tn){
5          //load output feature map
6          //load weight
7          //load input feature map
```

On-chip data computation (Point Loop)

```
8          for(trr=row;trr<min(row+Tr,R);trr++){
9             for(tcc=col;tcc<min(col+Tc,C);tcc++){
10               for(too=to;too<min(To+Tm,M);too++){
11                  for(tii=ti;ti<min(ti+Tn,N);tii++){
12                     for(i=0;i<K;i++){
13                        for(j=0;j<K;j++){
14                        L:  output_fm[too][trr][tcc] += weight[too][tii][i][j]*
                                 input_fm[tii][S*trr+i][S*tcc+j];
         }}}}}}//store output feature maps
   }}}}
```

Method

Computation Optimization

Different unrolling way will affect the complexity of generated hardware, and even affect the hardware operation frequency.

On-chip data computation (Point Loop)

```
8  for(trr=row;trr<min(row+Tr,R);trr++){
9    for(tcc=col;tcc<min(col+Tc,C);tcc++){
10     for(too=to;too<min(To+Tm,M);too++){
11       for(tii=ti;ti<min(ti+Tn,N);tii++){
12         for(i=0;i<K;i++){
13           for(j=0;j<K;j++){
14             L: output_fm[too][trr][tcc] +=
                   weight[too][tii][i][j]*
                   input_fm[tii][S*trr+i][S*tcc+j];
   }}}}}}
```

Before optimized

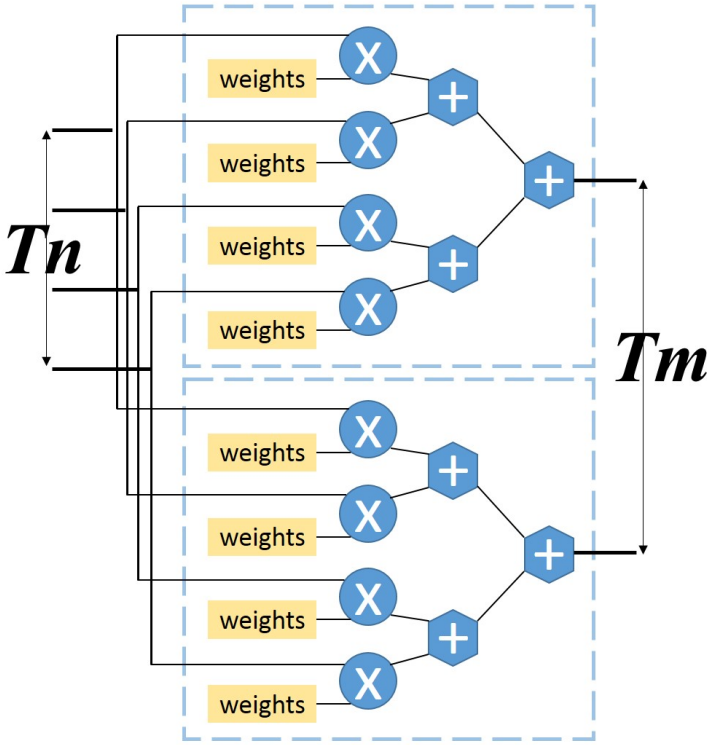
On-chip data computation (Point Loop)

```
8  for(i=0;i<K;i++){
9    for(j=0;j<K;j++){
10     for(trr=row;trr<min(row+Tr,R);trr++){
11       for(tcc=col;tcc<min(col+Tc,C);tcc++){
   #pragma HLS pipeline
12         for(too=to;too<min(To+Tm,M);too++){
   #pragma HLS UNROLL
13           for(tii=ti;ti<min(ti+Tn,N);tii++){
   #pragma HLS UNROLL
14             L: output_fm[too][trr][tcc] +=
                   weight[too][tii][i][j]*
                   input_fm[tii][S*trr+i][S*tcc+j];
   }}}}}}
```

After optimized

Method

Computation Optimization



On-chip data computation (Point Loop)

```
8  for(i=0;i<K;i++){
9    for(j=0;j<K;j++){
10     for(trr=row;trr<min(row+Tr,R);trr++){
11       for(tcc=col;tcc<min(col+Tc,C);tcc++){
   #pragma HLS pipeline
12         for(too=to;too<min(To+Tm,M);too++){
   #pragma HLS UNROLL
13           for(tii=ti;ti<min(ti+Tn,N);tii++){
   #pragma HLS UNROLL
14           L: output_fm[too][trr][tcc] +=
                weight[too][tii][i][j]*
                input_fm[tii][S*trr+i][S*tcc+j];
    }}}}}}
```

Method

Computational Roof

$$
\begin{aligned}
& Computational\ roof \\
= & \frac{total\ number\ of\ operations}{number\ of\ execution\ cycles} \\
= & \frac{2 \times R \times C \times M \times N \times K \times K}{[\frac{M}{T_m}] \times [\frac{N}{T_n}] \times [\frac{R}{T_r}] \times [\frac{C}{T_c}] \times (T_r \times T_c \times K \times K + P)} \\
\approx & \frac{2 \times R \times C \times M \times N \times K \times K}{[\frac{M}{T_m}] \times [\frac{N}{T_n}] \times R \times C \times K \times K} \\
\approx & \ 2 \times T_m \times T_n
\end{aligned}
\tag{2}
$$

where

$$
\begin{cases}
P = pipeline\ depth - 1 \\
0 < T_m \leq M \\
0 < T_n \leq N \\
0 < T_r \leq R \\
0 < T_c \leq C
\end{cases}
$$

Method

Memory Access Optimization

External data transfer (Tile Loop)

```
1 for(row=0;row<R;row+=Tr){
2   for(col=0;col<C;col+=Tc){
3     for(to=0;to<M;to+=Tm){
4       for(ti=0;ti<N;ti+=Tn){
5       //load output feature map
6       //load weight
7       //load input feature map
8       L:  foo(output_fm(to,row,col),
                weights(to,ti),
                input_fm(ti,to,col));
        //store output feature maps
  }}}}
```

Before local memory promotion

$'output_fm'$ memory access

$= 2 \times \frac{R}{T_r} \times \frac{C}{T_c} \times \frac{\mathbf{M}}{\mathbf{T_m}} \times \frac{\mathbf{N}}{\mathbf{T_n}} \times Size_{array}$

External data transfer (Tile Loop)

```
1 for(row=0;row<R;row+=Tr){
2   for(col=0;col<C;col+=Tc){
3     for(to=0;to<M;to+=Tm){
4       //load output feature map
4       for(ti=0;ti<N;ti+=Tn){
6       //load weight
7       //load input feature map
8       L:  foo(output_fm(to,row,col),
                weights(to,ti),
                input_fm(ti,to,col));
        }//store output feature maps
  }}}
```

After local memory promotion

$'output_fm'$ memory access

$= \frac{R}{T_r} \times \frac{C}{T_c} \times \frac{\mathbf{M}}{\mathbf{T_m}} \times Size_{array}$

FPGA
Chen Zhang,et al
Motivation
Background
Method
Exploration
Implementation
Experiment and Result
Conclusion

$$
\begin{aligned}
Computation\ to\ Communication\ Ratio &= \frac{total\ number\ of operations}{total\ amount\ of\ external\ data\ access} \\
&= \frac{2 \times R \times C \times M \times N \times K \times K}{\alpha_{in} \times B_{in} + \alpha_{weight} \times B_{weight} + \alpha_{out} \times B_{out}}
\end{aligned} \tag{3}
$$

where

$$B_{in} = T_n(ST_r + K - S)(ST_c + K - S) \tag{4}$$

$$B_{wight} = T_m T_n K^2 \tag{5}$$

$$B_{out} = T_m T_r T_c \tag{6}$$

$$0 < B_{in} + B_{wight} + B_{out} \leq BRAM_{capacity} \tag{7}$$

$$\alpha_{in} = \alpha_{weight} = \frac{M}{T_m} \times \frac{N}{T_n} \times \frac{R}{T_r} \times \frac{C}{T_c} \tag{8}$$

$$\alpha_{out} = \frac{M}{T_m} \times \frac{R}{T_r} \times \frac{C}{T_c} \tag{9}$$

Exploration

Design Space

Computation Engine:

Constrains for CNN configurations:

$T_m \in (Integer, 1 < T_m < M)\ M = 128$

$T_n \times T_m \in (Integer, 1 < Tn < N) N = 129$

Constrains for FPGA resource:

$$T_n \times T_m \in (Integer, 1 < T_m \times T_n < \#of\ PE)$$
$$\#of\ PE = 450$$

Communication:

of memory access methods

Legal Solutions of Tm&Tn:

2097

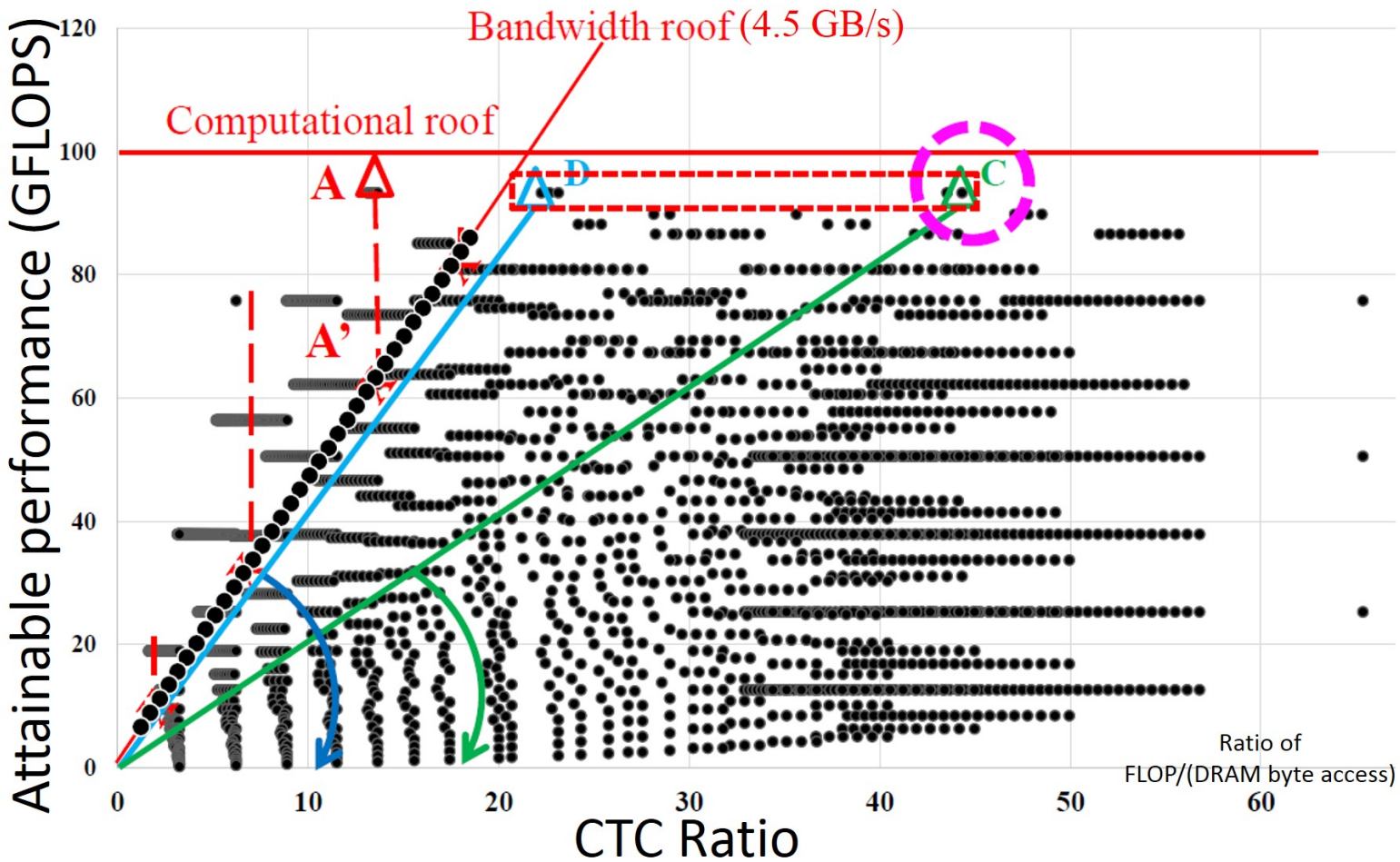
Legal Solutions:

3

Total Legal Solutions: 6291

Exploration

Design Space Exploration



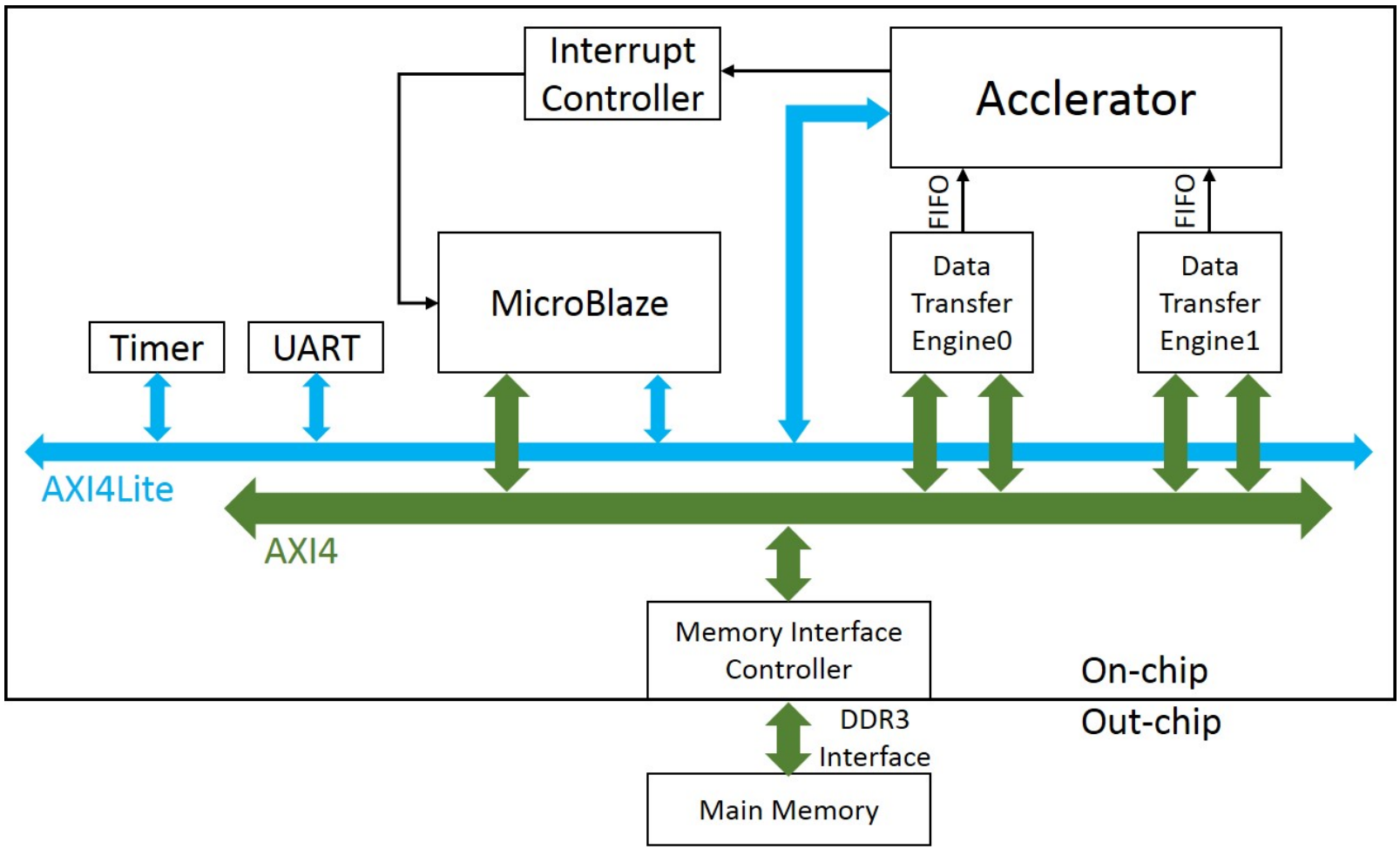
FPGA
Chen Zhang,et al
Motivation
Background
Method
Exploration
Implementation
Experiment and Result
Conclusion

	Optimal Unroll Factor$< T_m, T_n >$	Execution Cycles
Layer 1	$< 48, 3 >$	366025
Layer 2	$< 20, 24 >$	237185
Layer 3	$< 96, 5 >$	160264
Layer 4	$< 95, 5 >$	120198
Layer 5	$< 32, 15 >$	80132
Total	-	963804
Cross-Layer Optimization	$< 64, 7 >$	1008246

With unified unroll factors, the degradation is within **5%** compared to the total execution cycles of each optimized convolutional layer.

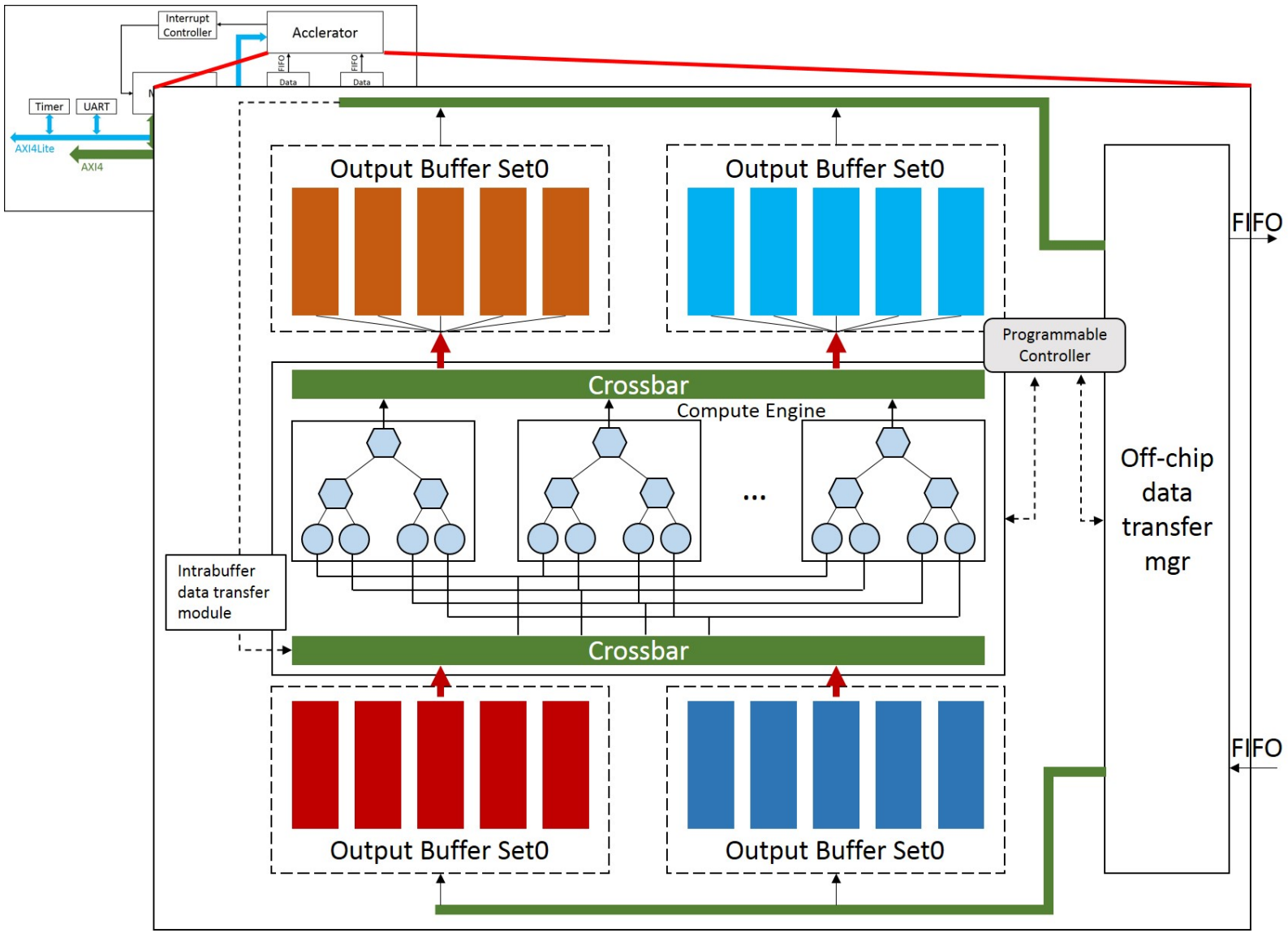
Implementation

System Overview



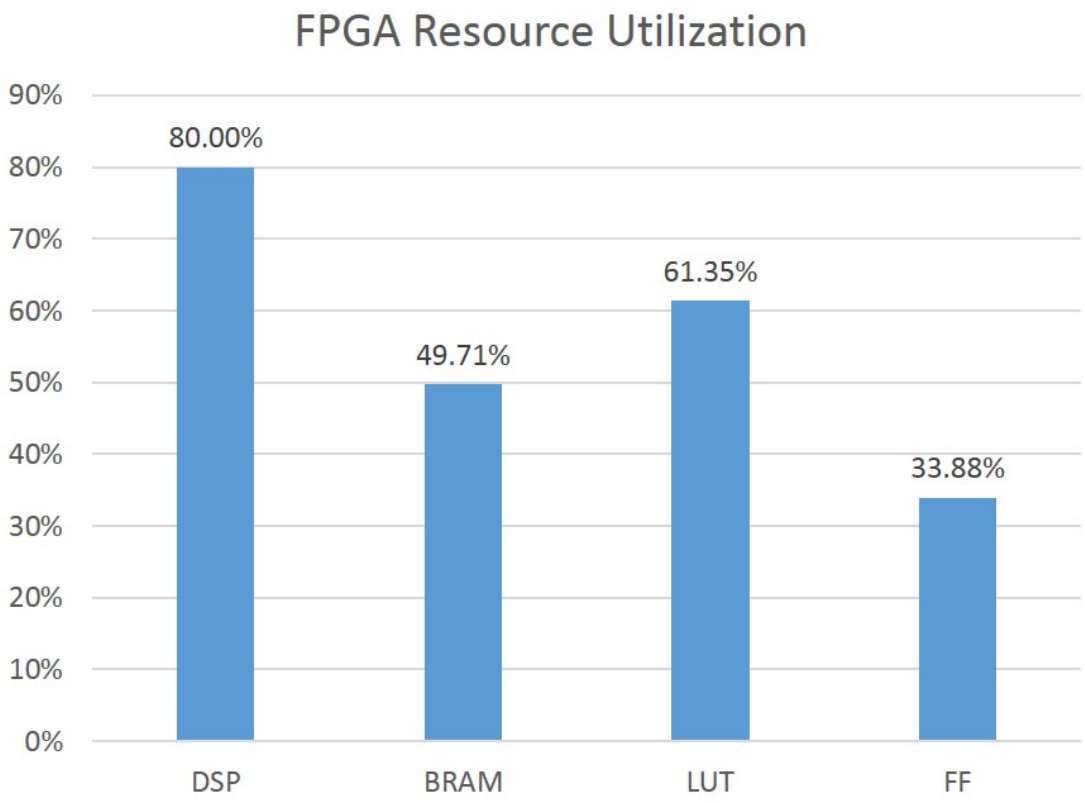
Implementation

Accelerator structure



Implementation

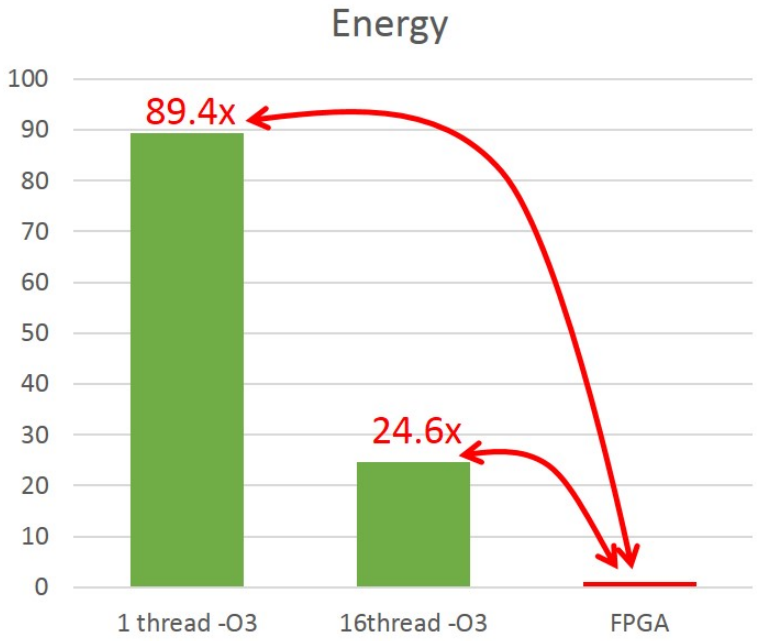
Resource Utilization

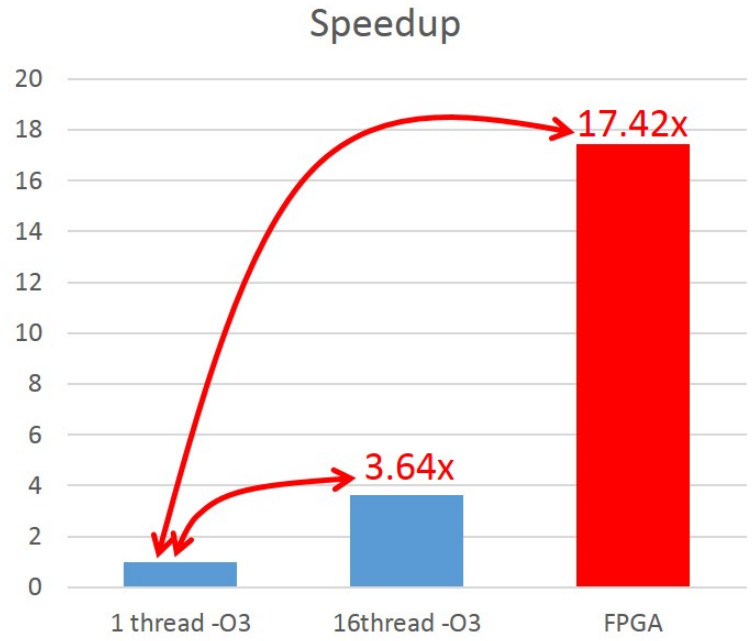


Resource	DSP	BRAM	LUT	FF
Used	240	1024	186251	205704
Available	2800	2060	303600	607200
Utilization	80%	50%	61.3%	33.87%

Experiment and Result

Vs. CPU



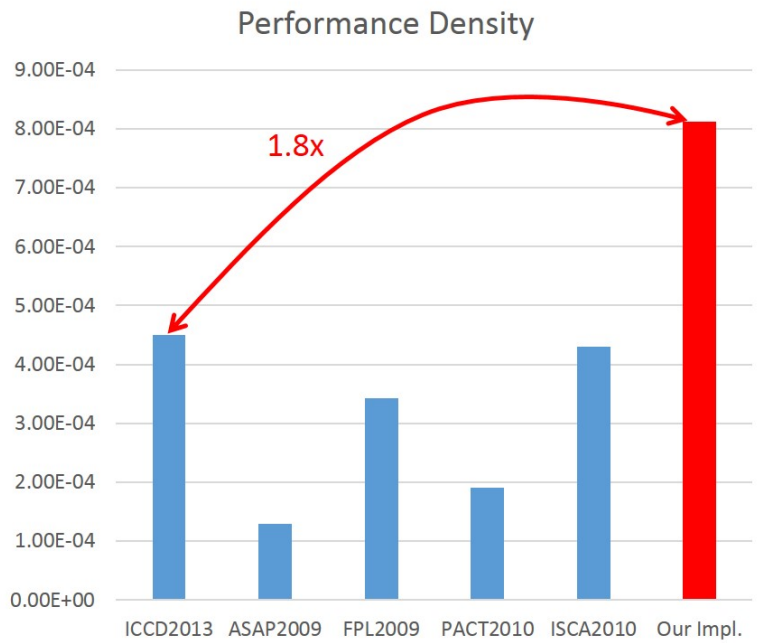


CPU	Xeon E5-2430 (32nm)	16 cores	2.2GHz	gcc 4.7.2 -O3 OpenMP 3.0
FPGA	Virtex7-485t (28nm)	448 PEs	100MHz	Vivado 2013.4 Vivado HLS 2013.4

Experiment and Result

Vs. Other Implementations

	ICCD2013	ASAP2009	FPL2009	PACT2010	ISCA2010	Our Impl.
FPGA Chip	Virtex6	Virtex5	Virtex4	Virtex5	Virtex5	Virtex7
Performance	17GOPS	6.74GOPS	5.25 GOPS	7.0GOPS	16GOPS	61.62GOPS
Performance Density	4.5E-04 GOPS/Slice	1.3E-04 GOPS/Slice	3.42E-04 GOPS/Slice	1.9E-04 GOPS/Slice	4.3E-04 GOPS/Slice	8.12E-04 GOPS/Slice



Conclusion

An accelerator for convolutional neural network

- **Contribution:**
  - Accurate analytical model for computation & communication
  - Find the best solution with roofline model
- **Result:**
  - On-board run implementation
  - **3.5x** better performance over other FPGA implementations
  - **80%** performance/area improvement

THANK YOU

Q & A?