Ming Hsieh Department of Electrical Engineering

Convolutional Neural Network on FPGA

Chi Zhang
FPGA/Parallel Computing Lab

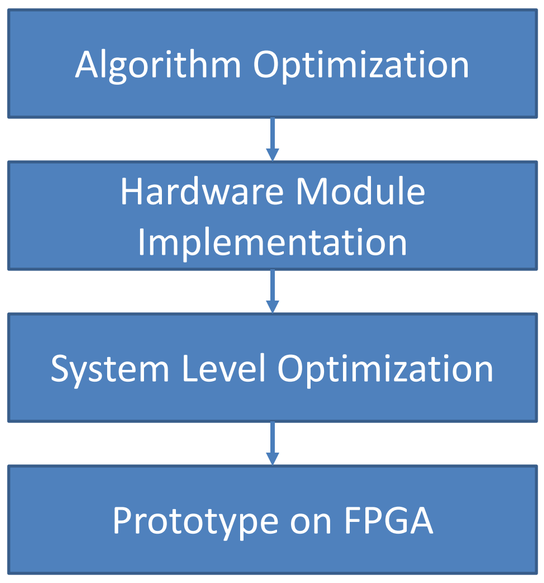
USC Viterbi School of Engineering

Motivation and Problem Definitions

- Convolutional Neural Network (CNN) achieves the state-of-art performance in image recognition, natural language processing and bioinformatics.
- High computation complexity in both inference and training, which needs specific hardware to accelerate.
- FPGA plays an important role due to its re-configurability and massive parallelism.
- Automatic Generation Tool: How to design a system that apply to a wide range of CNN architectures remain challenging.

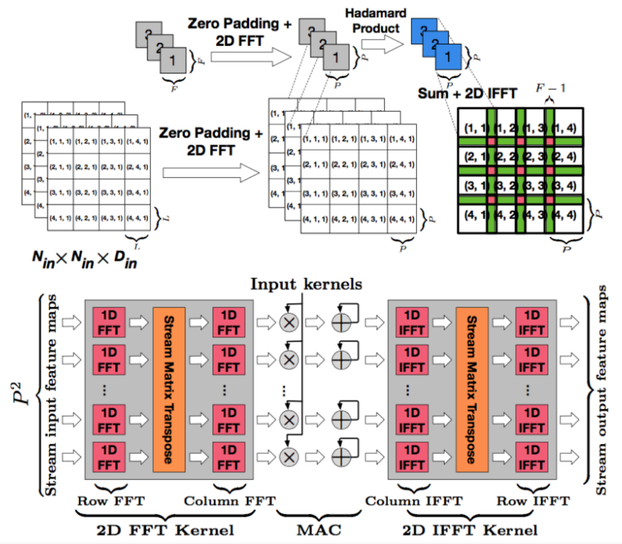
Approach

- Analyze the basic computational structure of CNN and identify fast algorithm for acceleration.
- Build highly-optimized and parameterized hardware accelerator module for specific algorithm including Matrix operations, Fast Fourier Transform, Winograd algorithm, etc.
- Given CNN architecture, utilize the hardware modules from library to build a full system.
- Explore the system level optimization including memory sub-system, resources distribution and scheduling algorithm.
- Prototype on specific FPGA device.



Algorithm and Hardware

- Frequency Domain Acceleration using FFT and Overlap-and-Add.



CPU + FPGA Mapping

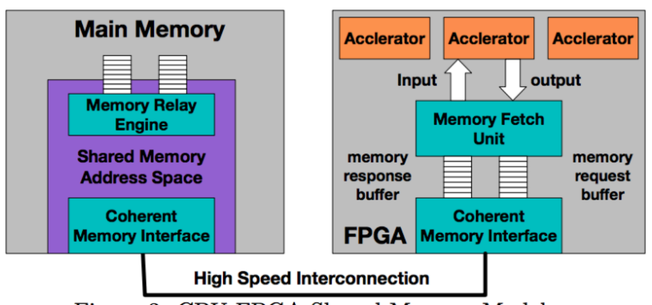


Figure 3: CPU-FPGA Shared Memory Model

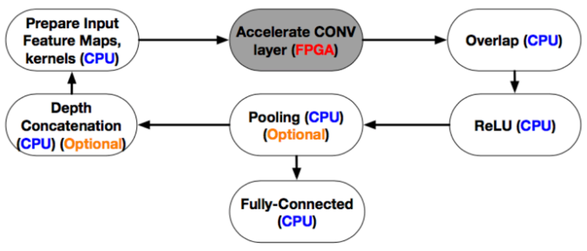
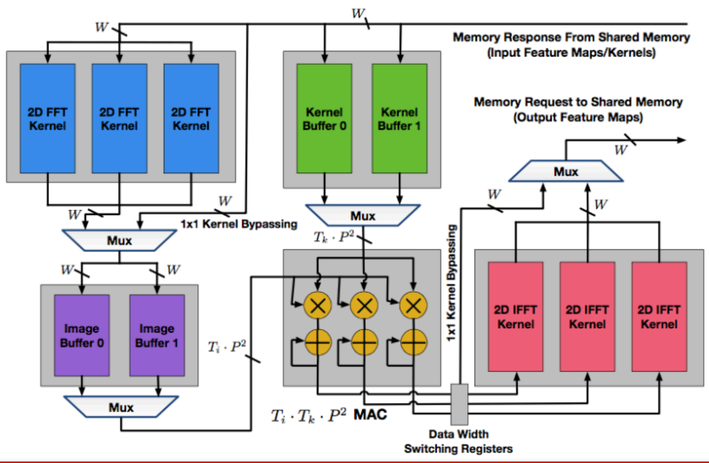


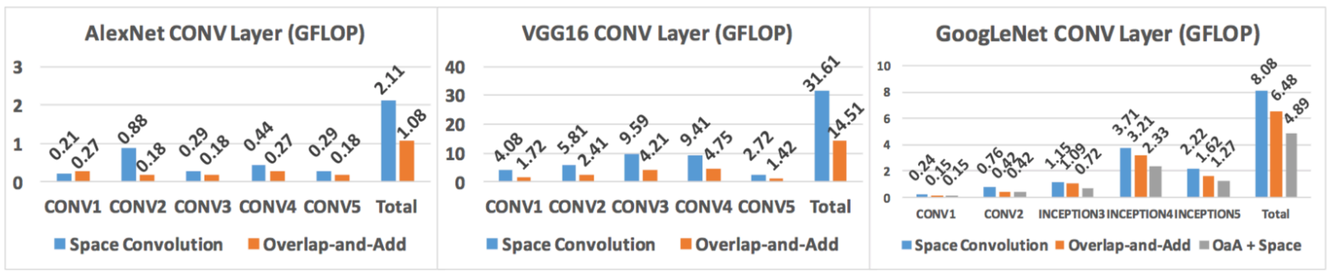
Figure 4: Proposed Mapping

System Level Optimization

- Increase on-chip data reuse to reduce FPGA-memory bandwidth requirement.
- Exploit task parallelism if there are available resources and bandwidth.



Experiments and Results



(a) AlexNet (2012) (b) VGG16 (2014) (c) GoogLeNet (2014)

Figure 7: Floating Point Operations required in various state-of-art CNNs

Asymptotic Analysis

- Space convolution: $O(N^2F^2)$
- OaA convolution: $O(N^2 \log F)$

GFLOP Reduction

- CaffeNet (2012): 48.82%
- VGG16 (2014): 54.10%
- GoogLeNet (2014): 39.43%

GoogLeNet (convolutional layer only) Execution Time (ms)		
Layer	**CPU (16 thds)**	**CPU (1 thd) + FPGA**
CONV1	38.53	12.70
CONV2	117.13	17.14
Inception3	91.05	16.16
Inception4	173.21	29.16
Inception5	45.45	8.48
Total	**465.37**	**83.64**
Throughput (GFLOPS)	**17.36**	**96.60**

Table 9: GoogLeNet Convolutional Layer Performance

	[21]	[14]	Our Work	
Platform	Virtex7 VX485t	Zynq XC7Z045	Intel QuickAssist QPI FPGA	
Clock (MHz)	100	150	200	
Data Precision	32-bit float	16-bit fixed	32-bit float	
Bandwidth (GB/s)	12.8	4.2	5.0	
CNN Model	AlexNet	VGG16-SVD	AlexNet	VGG16
Memory	4.5 MB	2.13 MB	4.0 MB	4.0 MB
Multipliers	747	780	224	224
Throughput (CONV) (GFLOPS)	61.62	187.80	83.00	123.48
Delay (CONV) (ms)	21.61	163.42	CPU:17.17 FPGA:23.64	263.27
Delay×Multipliers	16142	127467	9141/5295	58972
Classification Accuracy	Lossless	Lossy	Lossless	
Flexibility	Any CNN	Limited	Any CNN	

Table 7: Performance Comparison with the State-of-Art CNN Implementations on FPGA

Experimental Setup

- Intel QuickAssist QPI FPGA Platform (Xeon + Altera Stratix V)
- Shared memory between CPU and FPGA
- 6 GB/s FPGA-memory bandwidth + 6.25 MB on-chip BRAM
- Experimented CNN architectures: AlexNet, VGG16, GoogLeNet

Performance Comparison

- 1.1x delay with 3x less resources consumption
- Without loss of accuracy for any modern CNN architectures.

Layer (Group)	VGG16 Execution Time (ms)					AlexNet Execution Time (ms)			
	FPGA (Theoretical)	**FPGA (Actual)**	**CPU**	**Sequential**	**Concurrent**	**FPGA (Actual)**	**CPU**	**Sequential**	**Concurrent**
CONV1	30.96	31.53	7.76	39.29	32.74	-	17.17	17.17	17.17
CONV2	44.36	46.01	4.18	50.19	46.48	7.86	0.09	7.95	7.94
CONV3	81.92	82.27	3.54	85.81	82.75	4.42	0.12	4.54	4.50
CONV4	81.92	82.77	1.37	84.14	82.90	6.64	0.15	6.79	6.71
CONV5	17.69	18.36	0.27	18.63	18.40	4.42	0.11	4.53	4.49
CONV Total	256.85	262.94	17.12	280.06	**263.27**	**23.34**	17.64	40.98	**40.81**

Table 6: Execution Time for VGG16 and AlexNet

Discussions and Future Work

Energy Efficiency. A CPU-FPGA based design will consume more power than FPGA-only based design. However, the CPU adds more flexibility to the design. Moreover, since most of the computations are inside convolutional layer, the CPU simply performs adding and data rearrangement and the energy consumption will not scale up quickly if we increase the CNN size.
Automatic Code Generation. Our framework provides complete solution to accelerate CNN on FPGA including inter-layer data rearranging. Modern CNNs' convolutional layers are mainly consist of small kernels. Thus, by zero-padding various kernel sizes to fit a chosen FFT size, and using FPGA to accelerate it by exploiting massive parallelism, we can achieve considerable performance improvement for various CNN models. We can use our framework to develop an automatic code generation tool so high-level users can specify CNN models and generate the design automatically.
Fixed Point vs. Floating Point. Many previous approaches use fixed point instead of floating point for computations. The advantage is less resources consumption and the power efficiency. However, it penalizes the classification accuracy. Some approaches claim that the classification accuracy is tolerable according to experiments. However, it is hard to generalize to an arbitrary CNN model.