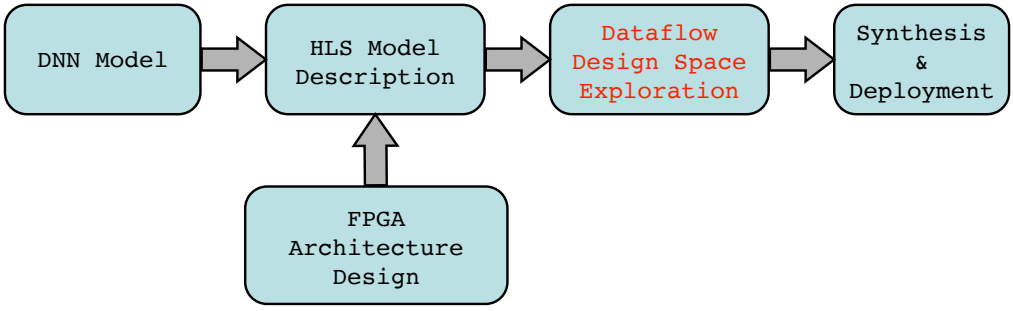
Power-Driven DNN Dataflow Optimization on FPGA

Qi Sun[1], Tinghuan Chen[1], **Jin Miao**[2], Bei Yu[1]

[1]The Chinese University of Hong Kong
[2]Cadence Design Systems, Inc.

cadence

FPGA Targeted DNN Accelerator Design Flow



- Design power efficient dataflow with limited latency loss
- Enabled by proposed power modeling and a hierarchical strategy

Dataflow Optimization

Basic Techniques for Loop Nest Optimization

```
for to in range(0, M):
  for row in range(0, H):
    for col in range(0, W):
      for ti in range(0, N):
        for k1 in range(0, K):
          for k2 in range(0, K):
            OUTto,row,col+=WTto,ti,k1,k2
                        × INti,row+k1,col+k2
```

A 6-level loop
convolutional layer

- Notations:
  N: # input channel
  M: # output channel
  K: kernel size
  H: height of feature
  W: width of feature

Dataflow Optimization

Basic Techniques for Loop Nest Optimization

```
for to in range(0, M, OC):
  for to1 in range(0, OC):
    for row in range(0, H):
      for col in range(0, W):
        for ti in range(0, N):
          for k1 in range(0, K):
            for k2 in range(0, K):
              OUT[to1+to,row,col] += WT[to1+to,ti,k1,k2]
                                   × IN[ti,row+k1,col+k2]
```

Loop Tiling

```
conv_group(BIN[IC],BWT[OC][IC],BOUT[OC]):
  for L1 in range(0, OC):
    for L2 in range(0, IC):
      convolution(BIN_L2, BWT_L1,L2, BOUT_L1)
```

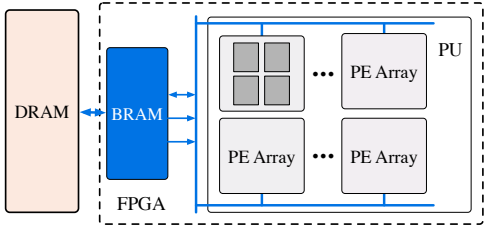
```
conv_group(BIN[IC],BWT[OC][IC],BOUT[OC]):
  for L1 in range(0, OC):
    convolution(BIN_0, BWT_L1,0, BOUT_L1)
    convolution(BIN_1, BWT_L1,1, BOUT_L1)
    ...
    convolution(BIN_IC-1, BWT_L1,IC-1, BOUT_L1)
```

Loop Unrolling

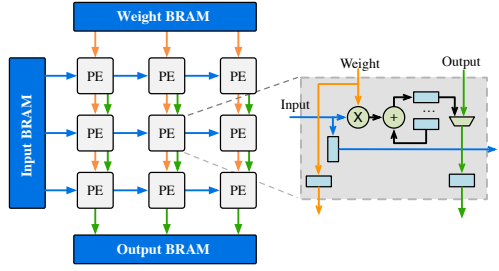
```
for to in range(0, M, OC):
  for row in range(0, H, PH):
    for col in range(0, W, PW):
      load_output(BOUT[OC][PH][PW],DOUT)
      for ti in range(0, N, IC):
        load_input(BIN[IC][PH][PW],DIN)
        load_weight(BWT[OC][IC][K][K],DWT)
        conv_group(BIN[IC],BWT[OC][IC],
                   BOUT[OC])
      store_output(BOUT[OC][PH][PW],DOUT)
```

Output Data Reuse

Typical FPGA Architecture for DNN



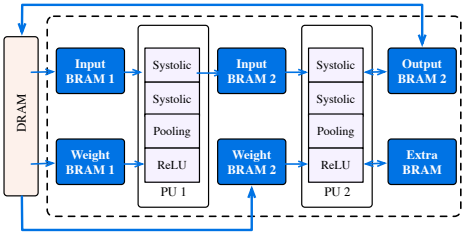
Architecture Design



Systolic Array

To Combine: Layer Fusion

Reduce Data Transfer



Hardware arch. of fusing 2 layers. Outputs of the first layer are the inputs of the second layer.

```
for to1 in range(0, M1, OC1):
  for row1 in range(0, H1, PH1):
    for col1 in range(0, W1, PW1):
      for ti1 in range(0, N1, IC1):
        load_input(BIN1[IC1][PH1][PW1],DIN1)
        load_weight(BWT1[OC1][IC1][K1][K1],DWT1)
        systolic_group_1(BIN1[IC1],BWT1[OC1][IC1],
                         BOUT1[OC1],U1)
      prepare_for_layer2(BOUT1[OC1][PH1][PW1],
                         extra_buffer)
      for to2 in range(0, M2, OC2):
        load_output(BOUT2[OC2][PH2][PW2],DOUT2)
        load_weight(BWT2[OC2][OC1][K2][K2],DWT2)
        systolic_group_2(BOUT1[OC1],BWT2[OC2][OC1],
                         BOUT2[OC2],U2)
        store_output(BOUT2[OC2][PH2][PW2],DOUT2)
```

Pseudocode of fusing 2 layers.

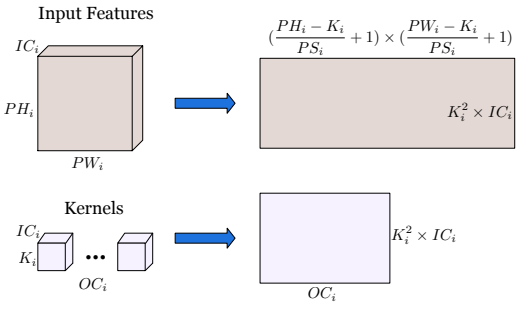
How To Configure The Variables?

Power Minimization

TABLE I List of Parameters

Name	Definition
OC_i	# of output channels per feature block in fused layer i
IC_i	# of input channels per feature block in fused layer i
PH_i	Feature height of feature block in fused layer i
PW_i	Feature width of feature block in fused layer i
Th_i	row # of PEs in one systolic array in fused layer i
Tw_i	column # of PEs in one systolic array in fused layer i
U_i	# of instantiated systolic arrays in fused layer i

- Notations:
  N_i: # of input channels in layer i
  M_i: # of output channels in layer i
  H_i: height of features in layer i
  W_i: width of features in layer i
  K_i: kernel size in layer i
  PS_i: block stride in layer i



From convolution blocks to matrices (systolic array). Matrices will be further decomposed to feed into systolic array.

Modeling the Energy

Assumptions

- Energy is decomposed as Data Transfer and Computation
- Energy per unit data access is considered as a constant for each memory level
- Data transfer energy inside systolic array is considered as computation energy for atomic nature

Modeling the Energy

Data Transfer Energy

- Energy (layer i):

$$ED_i = \frac{\alpha_1}{OC_i} + \frac{\alpha_2}{PH_i \times PW_i} + \alpha_3 PH_i + \alpha_4 PW_i + \frac{\alpha_5}{PH_i} + \frac{\alpha_6}{PW_i} + \frac{\alpha_7}{IC_i}.$$

- $\{\alpha_1, \ldots, \alpha_7\}$ are model-specific pre-characterized constants
- (α_1/OC_i) and (α_7/IC_i) refers to input and output. $(\alpha_2/(PH_i \times PW_i))$ for weights
- Other items are for data preparation process used in layer fusion

Modeling the Energy

Computation Energy

- Systolic array has $(Th_i \times Tw_i)$ PEs which are all active in computation
- Denote $K_t^2 \times IC_i$ as D_i. $(D_i + Th_i + Tw_i - 2)$ cycles are needed to compute each pair of inputs and weights sub-matrices.
- A single call of the systolic array consumes energy:

$$(Th_i \times Tw_i) \times (D_i + Th_i + Tw_i - 2) \times ec,$$

  where ec is the energy constant consumed by each PE per cycle
- Denote the energy of computing each pair of inputs and weights blocks as eb
- Total computation energy of layer i is EC_i:

$$EC_i = \lceil N_i/IC_i \rceil \times \lceil H_i/PH_i \rceil \times \lceil W_i/PW_i \rceil \times \lceil M_i/OC_i \rceil \times eb$$

Modeling the Latency

Data Transfer Latency

- Latency (layer i):

$$LD_i = \frac{\beta_1}{OC_i} + \frac{\beta_2}{PH_i \times PW_i} + \beta_3 PH_i + \beta_4 PW_i + \frac{\beta_5}{PH_i} + \frac{\beta_6}{PW_i} + \frac{\beta_7}{IC_i}.$$

- $\{\beta_1, \ldots, \beta_7\}$ are model-specific pre-characterized constants
- Other terms are following similar notation convention as used in energy formulation

Modeling the Latency

Computation Latency

- A single call to the systolic array needs $(D_i + Th_i + Tw_i - 2)$ cycles
- Denote the latency of computing each pair of inputs and weights blocks as lc which includes multiple calls to systolic arrays
- Total computation latency of layer i is LC_i:

$$LC_i = \lceil \lceil N_i \,/\, IC_i \rceil \times \lceil H_i \,/\, PH_i \rceil \,/\, U_i \rceil \times \lceil W_i \,/\, PW_i \rceil \times \lceil M_i \,/\, OC_i \rceil \times lc$$

Constrained Power Minimization

- Constrained by memory and DSP resources, i.e. $Buffer_{total}$ and DSP_{total}
- Constrained by latency upper bound L_{upper}
- Formulation

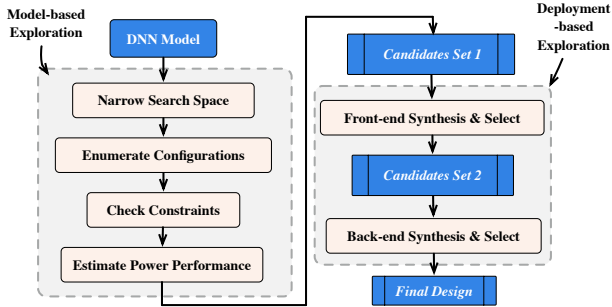
$$\begin{aligned} \min \ & \frac{E_{total}}{L_{total}}, \\ \text{s.t.} \ & Buffer_{used} \leq Buffer_{total}, \\ & DSP_{used} \leq DSP_{total}, \\ & L_{total} \leq L_{upper}, \end{aligned}$$

  where $E_{total} = \sum_{i=1}^{Z}(ED_i + EC_i)$, $L_{total} = \sum_{i=1}^{Z}(LD_i + LC_i)$, Z is the total number of layers in model

- The formulation is of non-convex, non-linear, integer constrained

Solution Exploration

A Two-Step Hierarchical Strategy



- Model-based exploration
- Deployment-based exploration

Solution Exploration

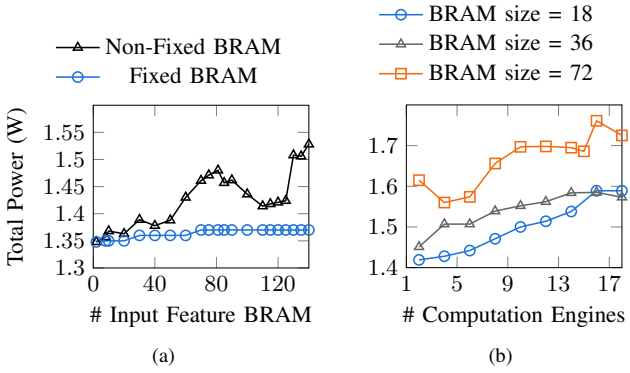
Model-based exploration

- Narrow configuration search space with empirical constraints
- Enumerate solutions
- Exclude invalid solutions violating resource and latency constraints
- **Estimate power-performance using the proposed power model and prune suboptimal ones**

Deployment-based exploration

- Verify surviving configurations by HLS further excluding FPGA incompatible solutions
- Synthesize further in back-end for real power-performance metrics
- Choose final configuration

Experiments



- (a) Non monotonic relation on BRAM size vs power consumption
- (b) Deploying 2×2 convolutions on PYNQ-Z1. Power increases with the number of computation engines and BRAMs.

Experiments

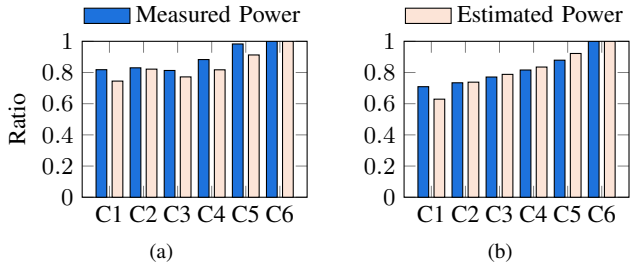
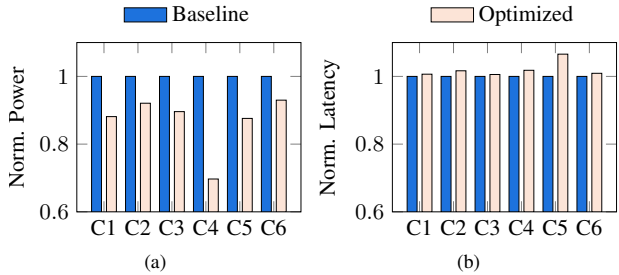


Fig. 7 Model fidelity analysis: (a) AlexNet; (b) VGG16.

- Model fidelity analysis on AlexNet and VGG16

Experiments



- Corresponding performance analysis.
- By sacrificing little latency performance, at most 6.5%, we can achieve more than 10% power benefits, with best up to 31%.

Further Discussions – Design Space Exploration

Current Solutions

- Most existing efforts explore the design space by defining white-box models to approximate HLS results
- Still need lots of time to verify the results
- May lose good designs because of low model accuracy

Possible Directions

- Some machine learning models can be used, e.g. Bayesian Optimization in analog circuit design
- Multi-fidelity algorithms, use cheap but inaccurate HLS results to help build the more accurate back-end implementation models

Further Discussions – Timing Closure

Current Solutions

- More and more complicated designs would impose difficulties on timing closure
- Some arts were proposed by tuning parameters in logical and physical synthesis
- Performance benefits are limited, by only considering back-end parameters tuning or P&R methods selection

Possible Directions

- Take HLS code revision into consideration
- Revising HLS code, tuning back-end parameters and customizing FPGA placement/routing methods can be considered in a uniform framework

Thank You


2019 INTERNATIONAL
CONFERENCE ON
COMPUTER-AIDED
DESIGN
30th Edition