MLBlocks: FPGA Blocks for Machine Learning Applications

SeyedRamin Rasoulinezhad, David Boland, and Philip H.W. Leong
School of Electrical and Information Engineering, The University of Sydney

THE UNIVERSITY OF SYDNEY

Motivation

› Observation:

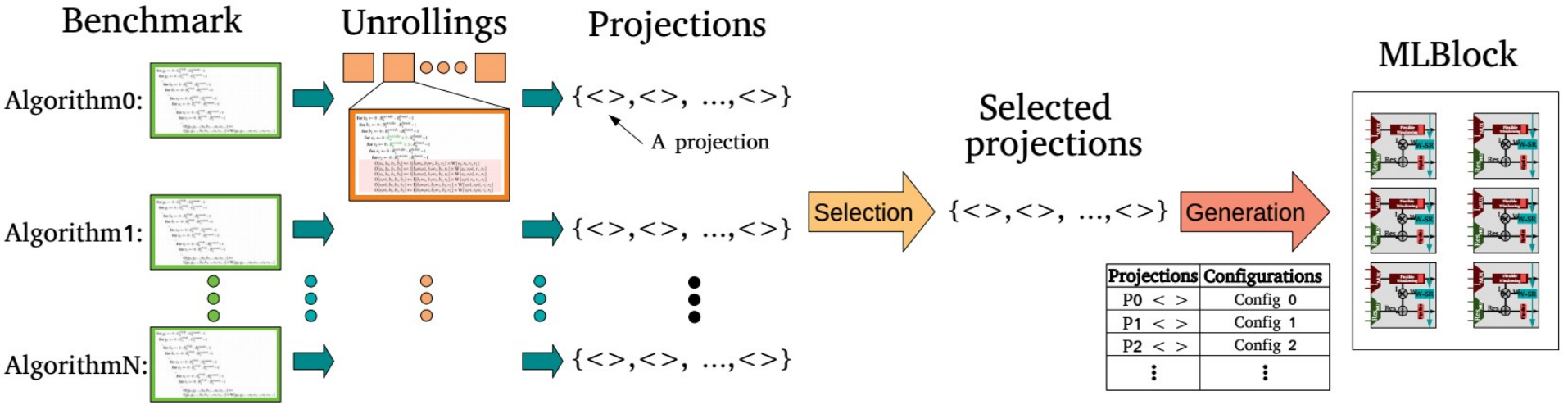
- FPGA architectures are Optimized for networking, signal/image processing using high-precision DSP blocks
- DSP48 → 27×18 multiplier, 48-bit Accumulation or two one input shared 8x8 MACs

› Previous works:

- **Enhancing existing DSP blocks**: PIR-DSP [1], Boutrous et al [2], Intel Agilex Architecture
  - Works on logic elements: LUXOR [3], Boutros et al. [4]
- **Integrating domain-specific engines:** Xilinx Versal AI-engines
  - Fundamental issue: They do not address the shortcomings of current FPGA architectures
- **Designing a new embedded block:** Hamamu [5], Achronix MLP72

› Aim: How to design a new block by a systematic fashion targeting future workloads

Overview and Generalized nested loop model



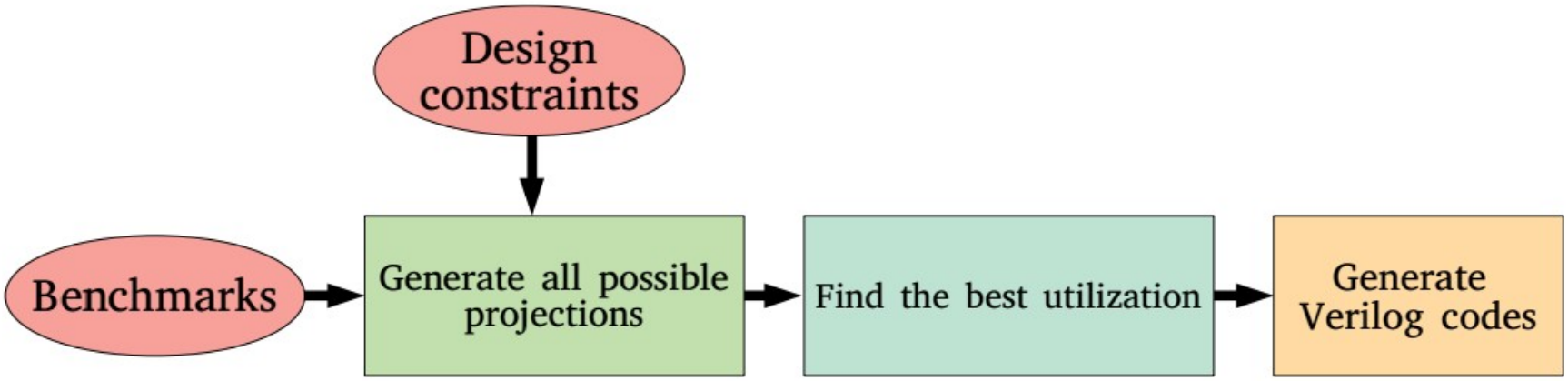
› Four loop variable groups:

- **R**eduction(IW) → reuse of temporary O
  - Like: dot product
- **E**xpansion(WO) → reuse of I
- **B**atching(IO) → reuse of W
- **G**rouping(IWO) → computation replication
  - Used in depth-wise and grouped convolution

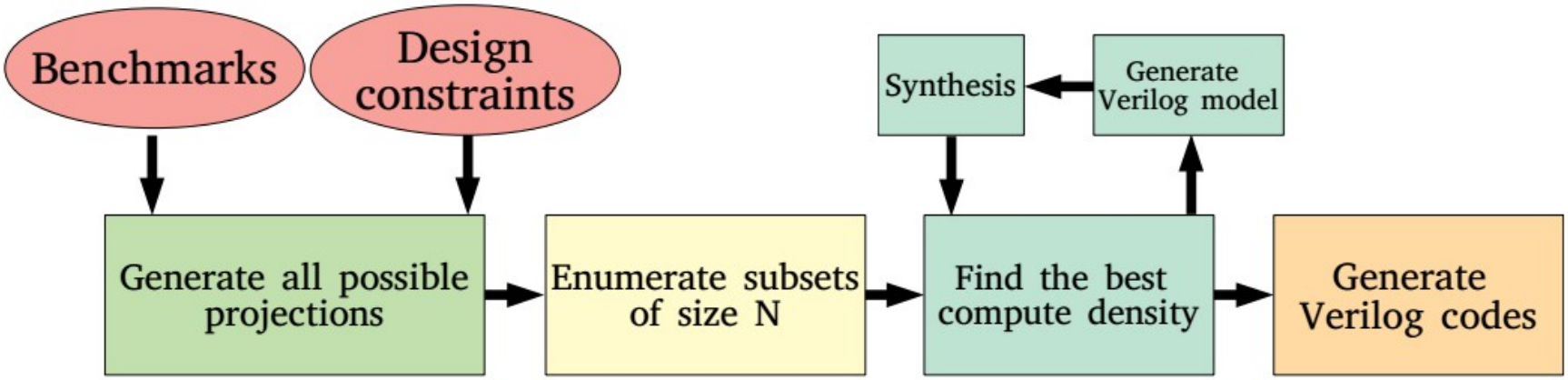
Algorithm 1: Pseudo code of the generalized nested loop model

$$
\begin{aligned}
&\textbf{for } g_0 \leftarrow 0 : G_0^{stride} : G_0^{limit} - 1 \\
&\quad \textbf{for } g_1 \leftarrow 0 : G_1^{stride} : G_1^{limit} - 1 \\
&\quad\quad \ldots \\
&\quad\quad\quad \textbf{for } b_0 \leftarrow 0 : B_0^{stride} : B_0^{limit} - 1 \\
&\quad\quad\quad\quad \textbf{for } b_1 \leftarrow 0 : B_1^{stride} : B_1^{limit} - 1 \\
&\quad\quad\quad\quad\quad \ldots \\
&\quad\quad\quad\quad\quad\quad \textbf{for } e_0 \leftarrow 0 : E_0^{stride} : E_0^{limit} - 1 \\
&\quad\quad\quad\quad\quad\quad\quad \textbf{for } e_1 \leftarrow 0 : E_1^{stride} : E_1^{limit} - 1 \\
&\quad\quad\quad\quad\quad\quad\quad\quad \ldots \\
&\quad\quad\quad\quad\quad\quad\quad\quad\quad \textbf{for } r_0 \leftarrow 0 : R_0^{stride} : R_0^{limit} - 1 \\
&\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \textbf{for } r_1 \leftarrow 0 : R_1^{stride} : R_1^{limit} - 1 \\
&\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \ldots \\
&\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \mathbf{O}[g_0, g_1, .., b_0, b_1, .., e_0, e_1, ..] \mathrel{+}= \\
&\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \mathbf{I}[g_0, g_1, .., b_0, b_1, .., r_0, r_1, ..] \times \mathbf{W}[g_0, g_1, .., e_0, e_1, .., r_0, r_1, ..]
\end{aligned}
$$

Projection selections

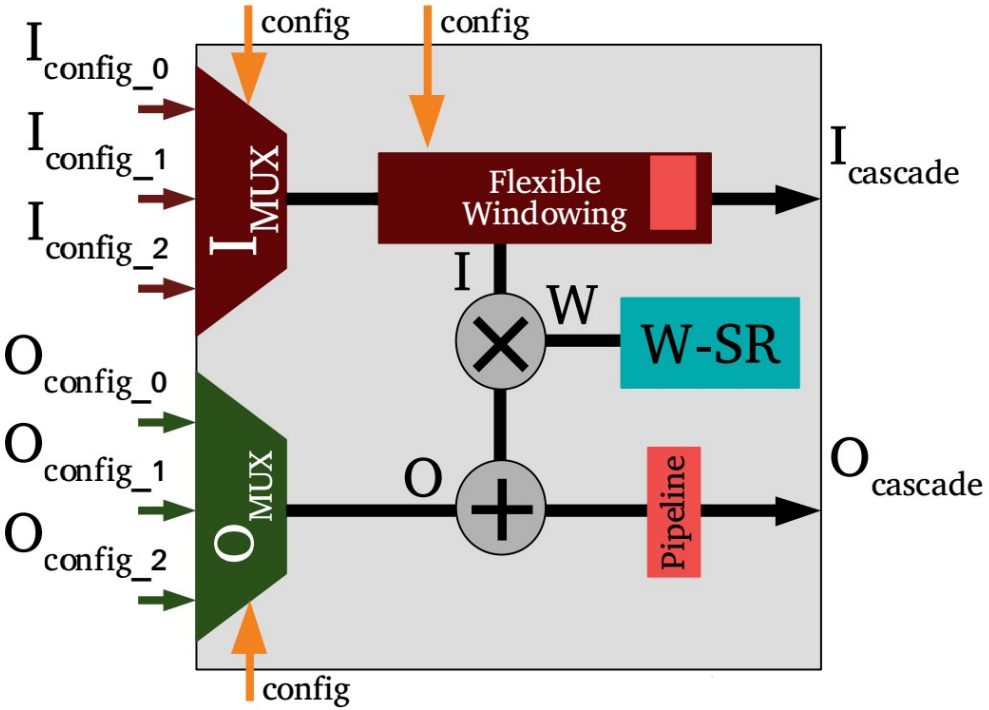


(a) Greedy search

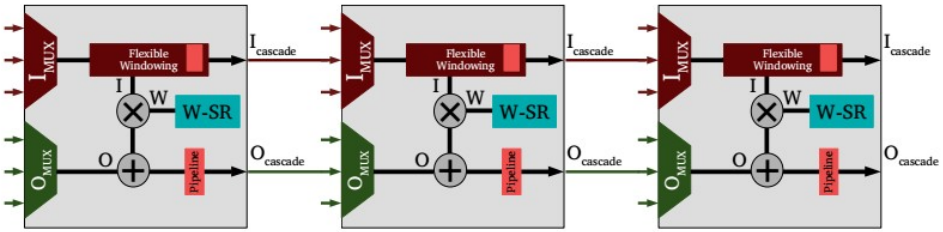


(b) N-Config search

Micro Architecture



(a) A MAC unit



(c) Cascading MAC units using I_{out} and Res_{out} signals

+ Serial Multiplier

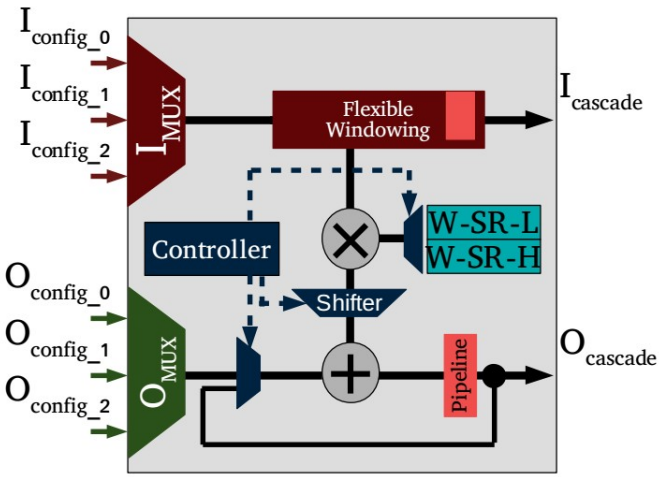
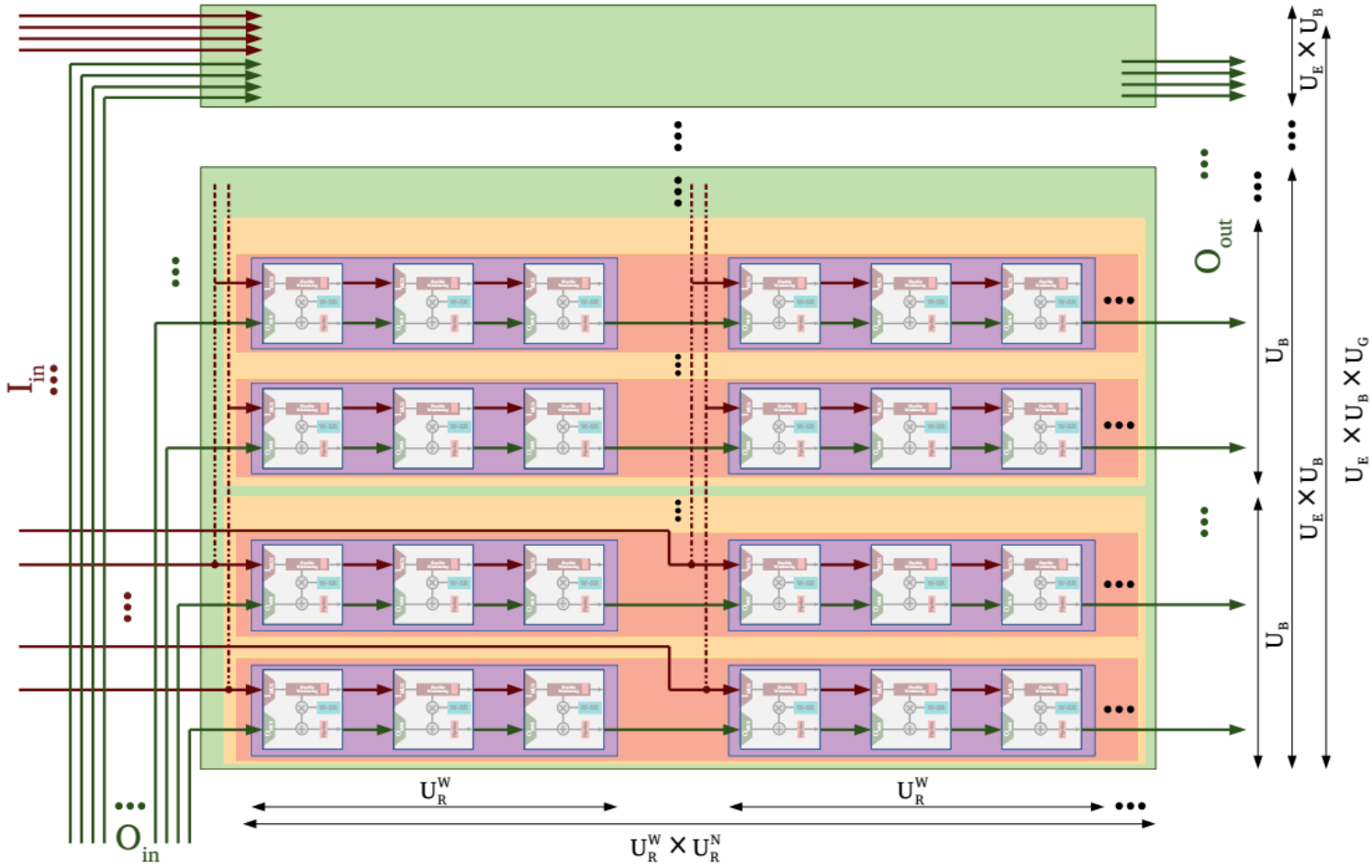


Figure 6: A serial multiplier-armed MAC unit

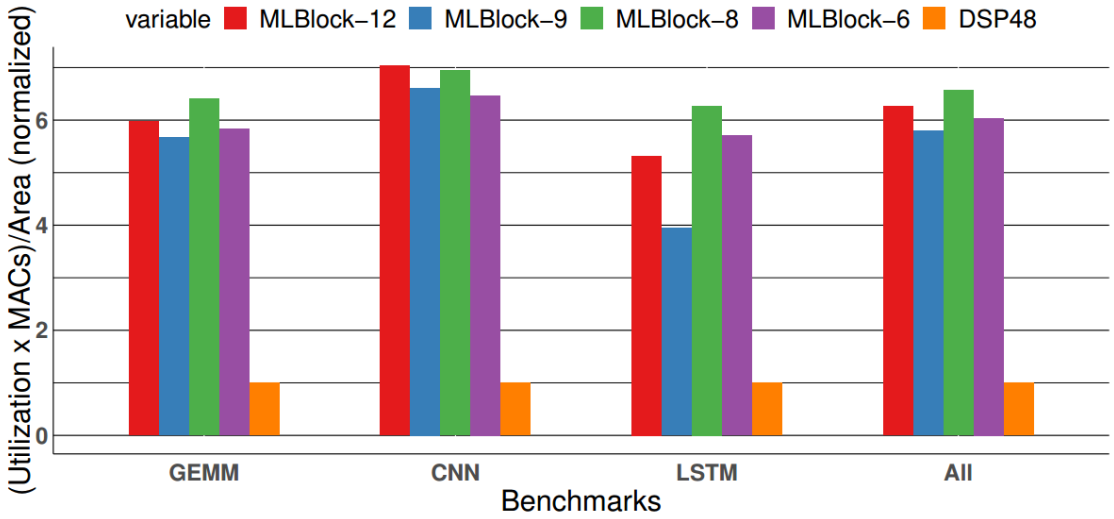
Macro Architecture

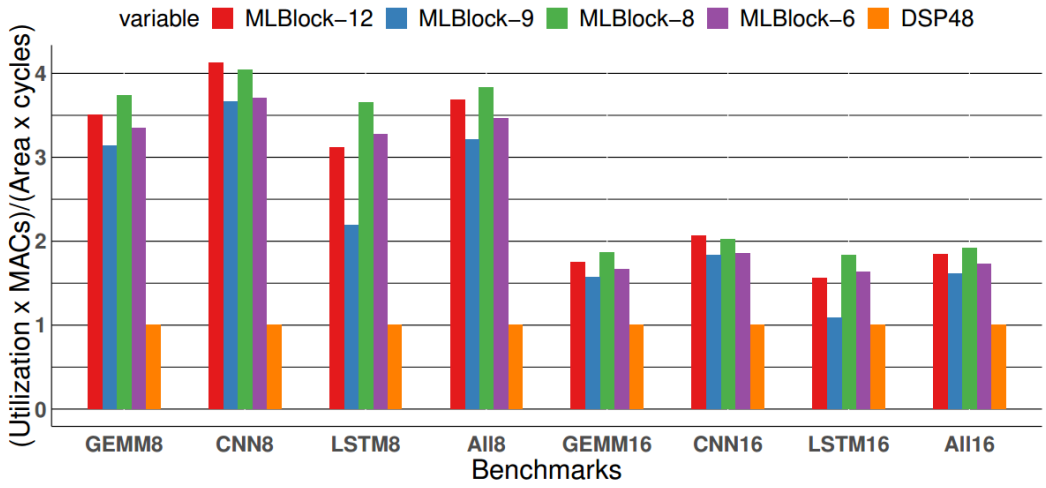
- Each selected projection maps to an MLBlock configuration
- A projection routing:



Results – DSP48-like MLBlocks

- Assumptions:
  - Greedy projection selection
  - DSP48 IO and Area constraints
- Compute density improvement:
  - Only 8x8 (top)
  - 8x8, 8x16, 16x8, 16x16 (down)





THE UNIVERSITY OF SYDNEY

Conclusion

- Briefly our contributions are:
  - A Methodology for designing coarse-grained embedded blocks for machine learning applications.
  - MLBlocks, a parameterized embedded block architecture
  - Two configuration selection techniques, called Greedy and heuristic
  - A python based framework to generate hardware description of an efficient MLBlock instance for a given set of constraints

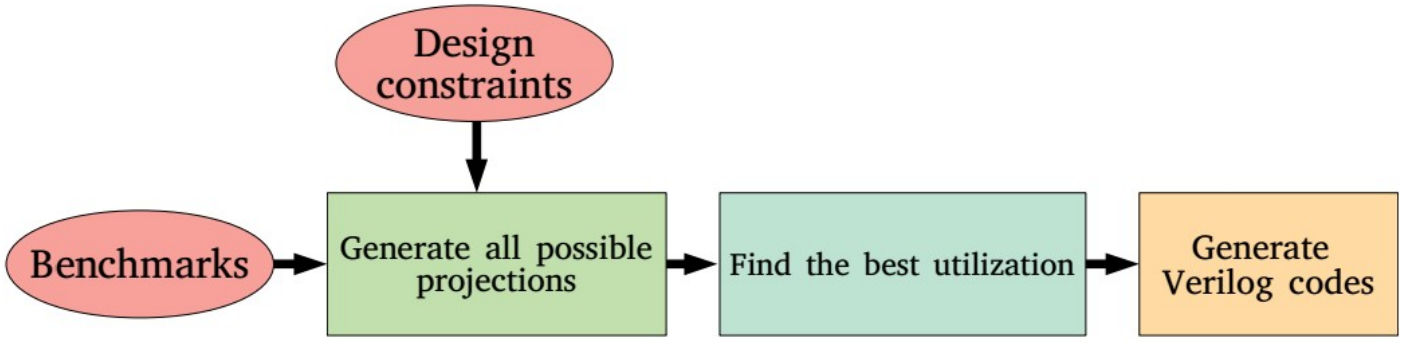
- Conclusions:
  - Using Xilinx DSP48 constraints, our approach results to embedded blocks with 6 times more compute density

References

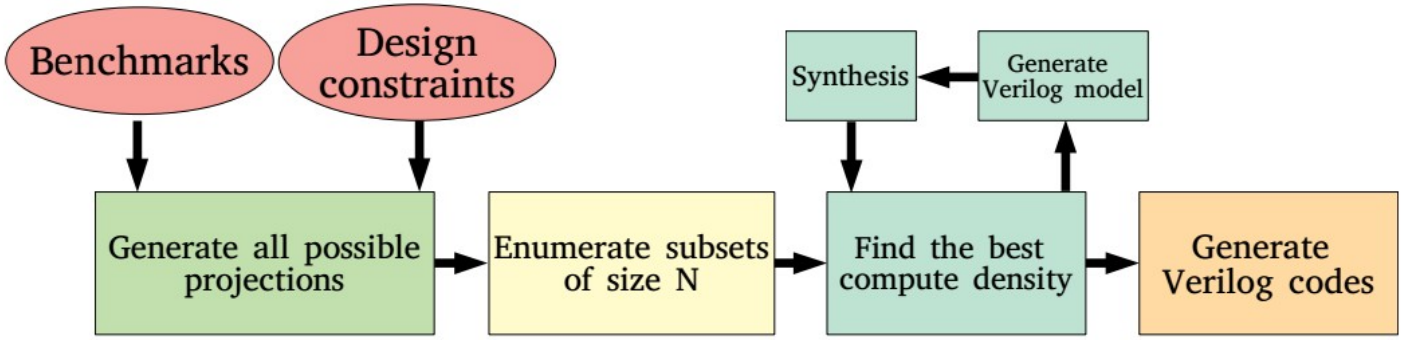
- [1] S. Rasoulinezhad, H. Zhou, L. Wang, and P. H. W. Leong, PIR-DSP: an FPGA DSP block architecture for multi-precision deep neural networks, FCCM 2019,
- [2] A. Boutros, S. Yazdanshenas, and V. Betz, Embracing diversity: Enhanced DSP blocks for low-precision deep learning on FPGAs, FPL 2018
- [3] S. Rasoulinezhad, Siddhartha, H. Zhou, L. Wang, D. Boland, and P. H. W. Leong, LUXOR: an FPGA logic cell architecture for efficient compressor tree implementations, in FPGA '20
- [4] A. Boutros, M. Eldafrawy, S. Yazdanshenas, and V. Betz, Math doesn't have to be hard: Logic block architectures to enhance low-precision multiply-accumulate on FPGAs, FPGA 2019
- [5]A. Aror, Z. Wei2, and L. K. John, Hamamu: Specializing FPGAs for ML Applications by Adding Hard Matrix Multiplier Blocks, ASAP20

Spare Slides

Projection selections



(a) Greedy search



(b) N-Config search

$$utilization = \prod_{\hat{v} \in \mathbb{V}} \frac{V^{count}}{\lceil \frac{V^{count}}{\hat{V}_i^{unroll}} \rceil \hat{V}^{count}}$$

$$\text{compute density} = M \frac{set_{utilization}}{set_{area}}$$

Unrolling

Algorithm 2: Pseudo code of standard convolution layer

```
for b0 ← 0 : B0^stride : B0^limit − 1
  for b1 ← 0 : B1^stride : B1^limit − 1
    for b2 ← 0 : B2^stride : B2^limit − 1
      for e0 ← 0 : E0^stride : E0^limit − 1
        for r0 ← 0 : R0^stride : R0^limit − 1
          for r1 ← 0 : R1^stride : R1^limit − 1
            for r2 ← 0 : R2^stride : R2^limit − 1
              O[e0, b0, b1, b2] += I[b0 + r0, b1 + r1, b2, r2] × W[e0, r0, r1, r2]
```

Unroll

Algorithm 3: Pseudo code of an unrolled standard convolution layer ($E_0^{unroll} = 2, R_0^{unroll} = 3$)

```
for b0 ← 0 : B0^stride : B0^limit − 1
  for b1 ← 0 : B1^stride : B1^limit − 1
    for b2 ← 0 : B2^stride : B2^limit − 1
      for e0 ← 0 : E0^stride × E0^unroll : E0^limit − 1
        for r0 ← 0 : R0^stride × R0^unroll : R0^limit − 1
          for r1 ← 0 : R1^stride : R1^limit − 1
            for r2 ← 0 : R2^stride : R2^limit − 1
              O[e0, b0, b1, b2] += I[b0+r0, b1+r1, b2, r2] × W[e0, r0, r1, r2]
              O[e0, b0, b1, b2] += I[b0+r0+1, b1+r1, b2, r2] × W[e0, r0+1, r1, r2]
              O[e0, b0, b1, b2] += I[b0+r0+2, b1+r1, b2, r2] × W[e0, r0+2, r1, r2]
              O[e0+1, b0, b1, b2] += I[b0+r0, b1+r1, b2, r2] × W[e0+1, r0, r1, r2]
              O[e0+1, b0, b1, b2] += I[b0+r0+1, b1+r1, b2, r2] × W[e0+1, r0+1, r1, r2]
              O[e0+1, b0, b1, b2] += I[b0+r0+2, b1+r1, b2, r2] × W[e0+1, r0+2, r1, r2]
```

- Unrolling a variable increases the number of:
  - MACs by the same factor
  - IO (I W O) where the variable accesses
- Variables in the same group
  - Have the same characteristics
  - Describe dimensions of the same access pattern
  - Example: in a Convolution layer x and y are similar when f_x is different (respectively B2, B1, R2)
- An Exception: Some variables in Reduction group can be windowed → we a separate them as two subgroups for Reduction group called, Windowed and NonWindowed.

Projections

› We define unrolling degree for each group:

Specifically, we define unrolling degree for each variable group as $U_{group} = \prod \hat{V}_i^{unroll}$ for the relevant variable group. Using this

- Then total number of MACs and required IOs are as follows:

$$M = \prod_{\mathbb{V}} \hat{V}_i^{unroll} \qquad Bandwidth = \prod_{\mathbb{V}_I} \hat{V}_i^{unroll} + \prod_{\mathbb{V}_W} \hat{V}_i^{unroll} + \prod_{\mathbb{V}_O} \hat{V}_i^{unroll}$$

- Where $\mathbb{V}$, $\mathbb{V}_I$, $\mathbb{V}_W$, $\mathbb{V}_O$ are all variables, and the variables appeared in I, W, and O respectively

› A projection is a circuit abstraction based on unrolling degrees and windowing details:

$$< (U_R^W, W_{buffer}, W_{stride}), U_R^N, U_E, U_B, U_G >$$

- A projection maps to a block configuration
- Finding projection is non-invertible, So, more than one unrolling could be mapped to a projection

BRAM

BRAM

BRAM

Results - Projection Selections

Benchmarks: Selected kernels from DeepBench-Baidu [25], for Training/inference tasks.

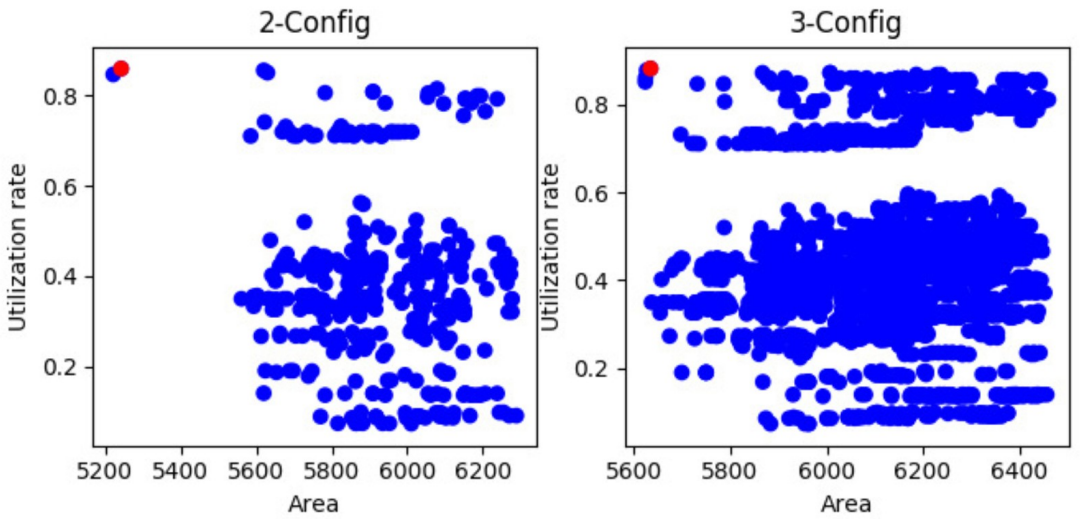
name	# of Cases	b_0	b_1	b_2	e_0	r_0	r_1	r_2
GEMM	19	(35-10752,1)	-	-	(1-1500,1)	-	-	(1024-3584,1)
CNN	12	(7-350,1-2)	(7-224,1-2)	(1-32,1)	(32-2048,1)	(1-7,1)	(1-20,1)	(1-512,1)
LSTM	4	(1024-8448,1)	-	-	(1-4,1)	-	-	(512-5632,1)

Greedy vs N-Config

Table 2: Selection methods for MLBlock-12

Method	utilization	Area†	Obj‡	# Synth.	time
Greedy	88.241	6093	1	1	1–2 mins
1-Config	72.000	5243	0.94	28	3–4 mins
2-Config	86.019	5245	1.13	378	1–2 hours
3-Config	88.192	5634	1.08	3276	≈ a day
4-Config	88.241	-	-	20475	≈ a week
5-Config	88.241	-	-	98280	≈ a month

†um^2, ‡normalized objective: $\frac{utilization}{Area}$



Results – DSP48-like MLBlocks

- Using greedy approach for projection selection

Table 3: Post-synthesis results for different EB architectures

EB name	Precisions	Utilization	Area*	Power*
DSP48E2	27×18 or two 8×8	≈ 1	1	1
MLBlock-12	8×8	88%	0.85	0.98
MLBlock-9	8×8	86%	0.66	0.81
MLBlock-8	8×8	92%	0.56	0.65
MLBlock-6	8×8	93%	0.46	0.54
MLBlock-12	$(16/8) \times (16/8)$	88%	1.44	1.81
MLBlock-9	$(16/8) \times (16/8)$	86%	1.20	1.57
MLBlock-8	$(16/8) \times (16/8)$	92%	0.96	1.20
MLBlock-6	$(16/8) \times (16/8)$	93%	0.80	1.02

*normalized