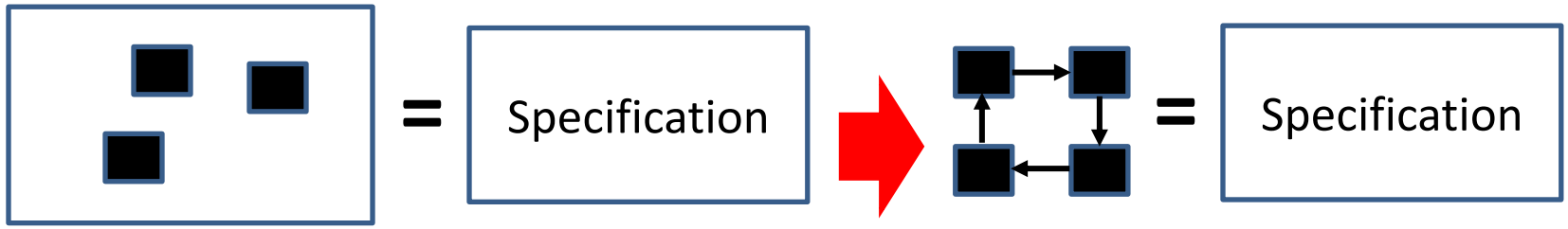
Automatic Synthesis of Distributed/Parallel Computing through Templates and Inductive Reasoning

Masahiro Fujita
VLSI Design and Education Center
University of Tokyo

Basic ideas

- Use of our partial logic synthesis based on QBF algorithms
  - Automatically synthesizing only missing portions in the circuit
- Formulation for automatic synthesis of parallel/ distributed computing as partial logic synthesis
  - Solved by QBF/SAT solvers with implicit and exhaustive search
  - Works only for small instances of the problems
- Use human induction to generalized the solutions
  - Generalized solution can be formally verified
- With templates, induction can be automated?



: Missing portion and synthesis target

: Chip/Core

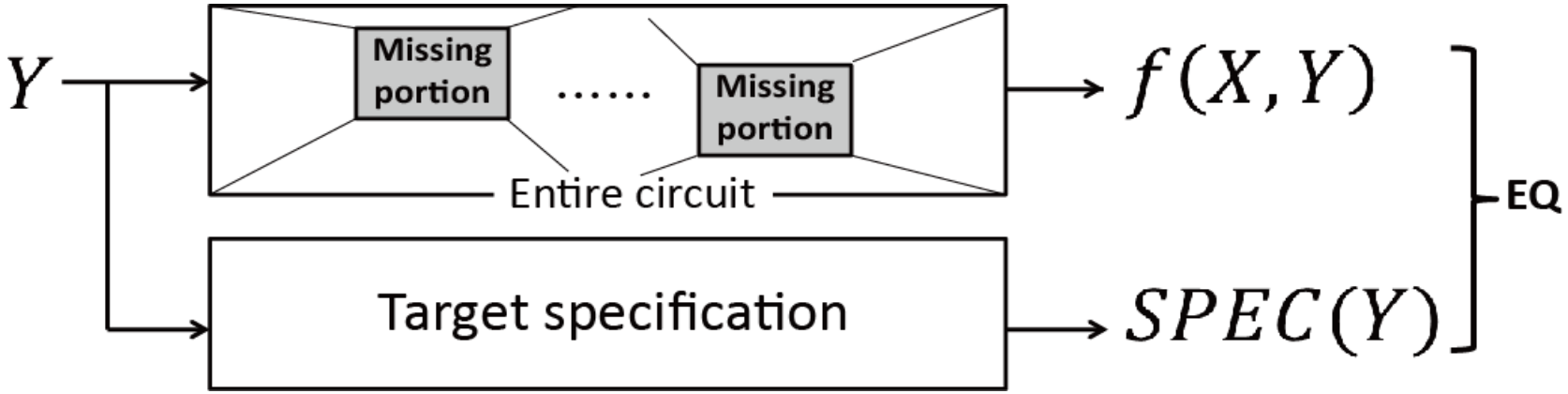
Partial logic synthesis

"Under appropriate programs for LUTs (existentially quantified), circuit behaves correctly for all possible input values (universally quantified)"

$\exists X \forall Y . f(X,Y) = SPEC(Y)$ Quantified Boolean Formula

X : configurations of LUTs, Y : inputs value of the circuit

f : output value of target circuit, $SPEC$: output value of specification



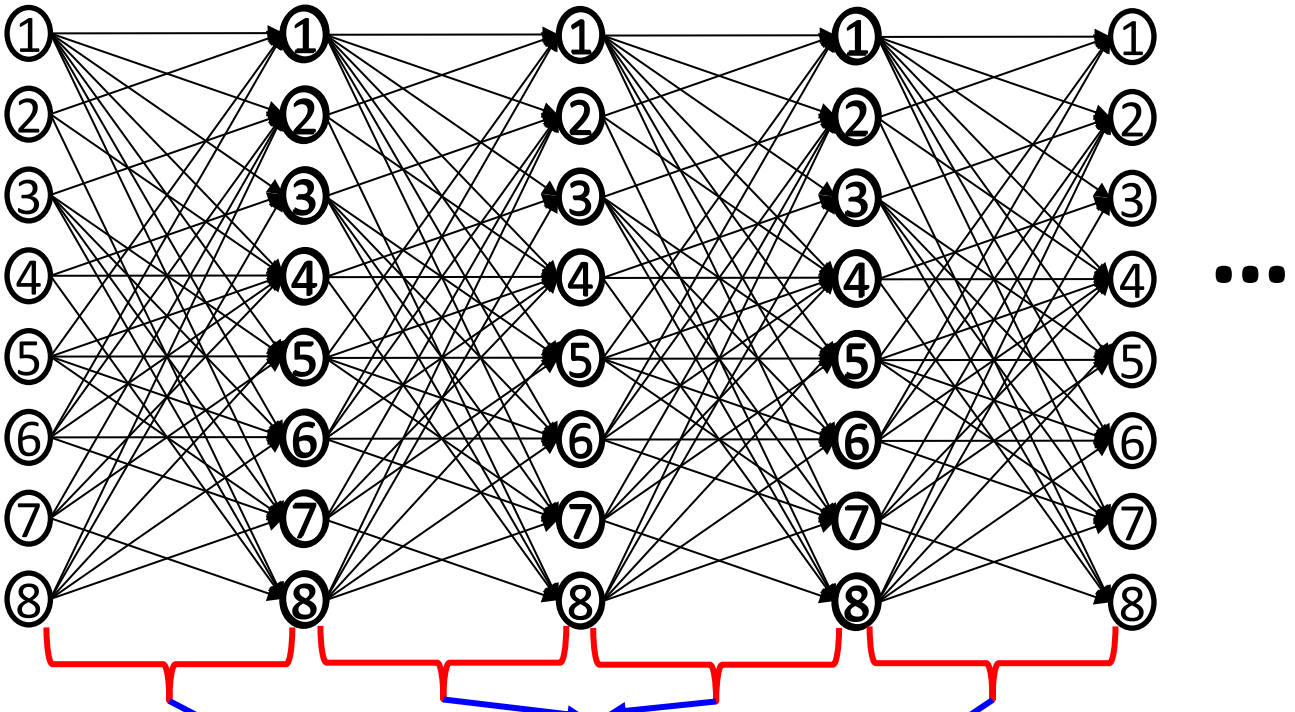
- By concentrating on data transfers, the problem can be formulated as pure SAT problem instead of QBF

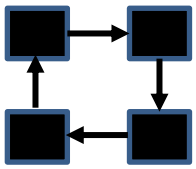
Application : Deep learning

- Each layer is processed one by one with M cores connected through one way ring communication

Connections are sparse and not like to perform multiplication by 0

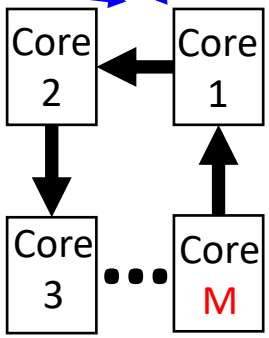
Overall computation should be accelerated by M times





= Specification

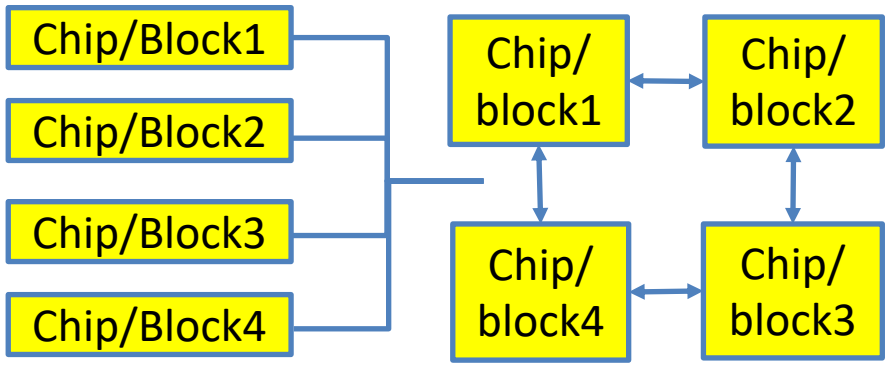
: Chip/Core

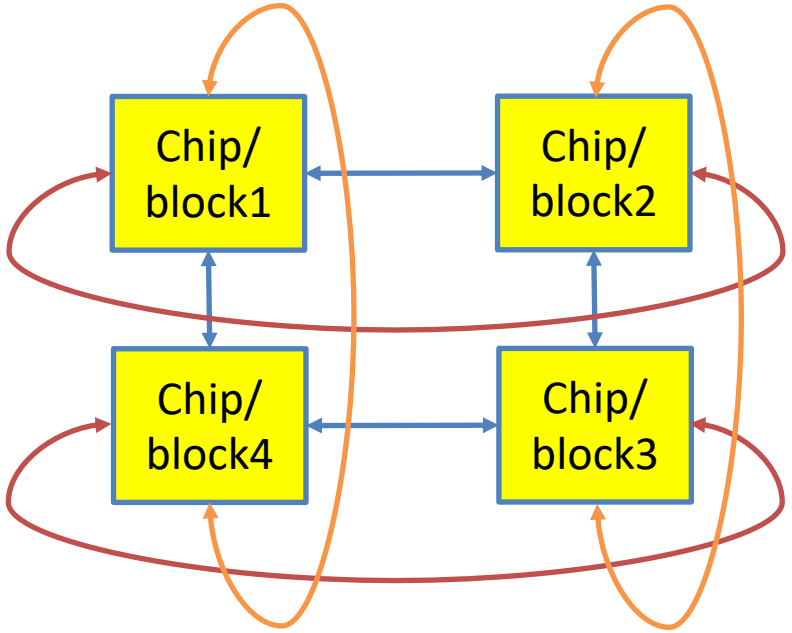


Target: Matrix-vector products (Lots of MAC)

Computations with multiple blocks/chips

- Matrix-vector products or MAC operation is memory and computation resource consuming
- Communication latency can easily become bottleneck
- Easily implementable network topology for multiple chips
  - Common Bus
    - One pair of communication
  - Ring
    - Only with neighbors but all pairs at the same time
  - Mesh (2D, 3D, 4D, 5D, 6Dtorus)





Will show that ring is sufficient for maximum speed up

Normal way to compute weighted sum

- 4X4 weighted sum

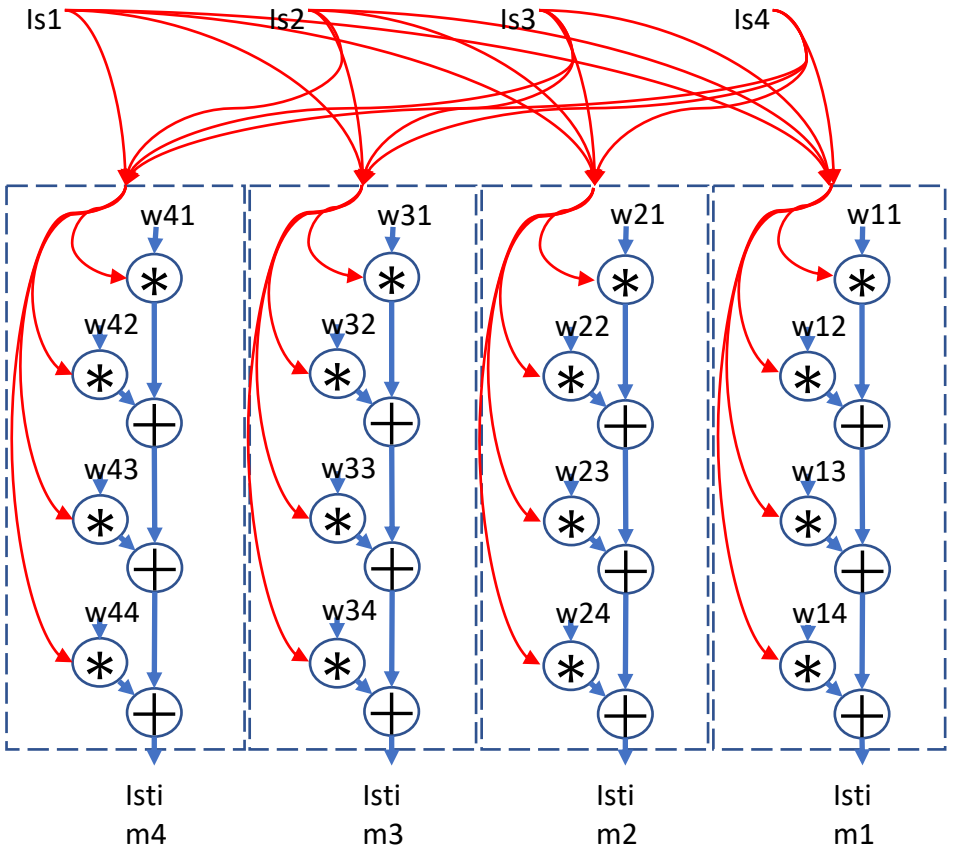
$$\begin{pmatrix} I_{stim}1 \\ I_{stim}2 \\ I_{stim}3 \\ I_{stim}4 \end{pmatrix} = \begin{pmatrix} w11 & w12 & w13 & w14 \\ w21 & w22 & w23 & w24 \\ w31 & w32 & w33 & w34 \\ w41 & w42 & w43 & w44 \end{pmatrix} \cdot \begin{pmatrix} I_s1 \\ I_s2 \\ I_s3 \\ I_s4 \end{pmatrix}$$

- Simple partitioning

$$\begin{pmatrix} w11 & w12 & w13 & w14 \\ w21 & w22 & w23 & w24 \\ w31 & w32 & w33 & w34 \\ w41 & w42 & w43 & w44 \end{pmatrix}$$

  - Many global communications
  - High level synthesis generates lots of communications

- Need to change DFG for better high level synthesis
  - Need to change the order of summation



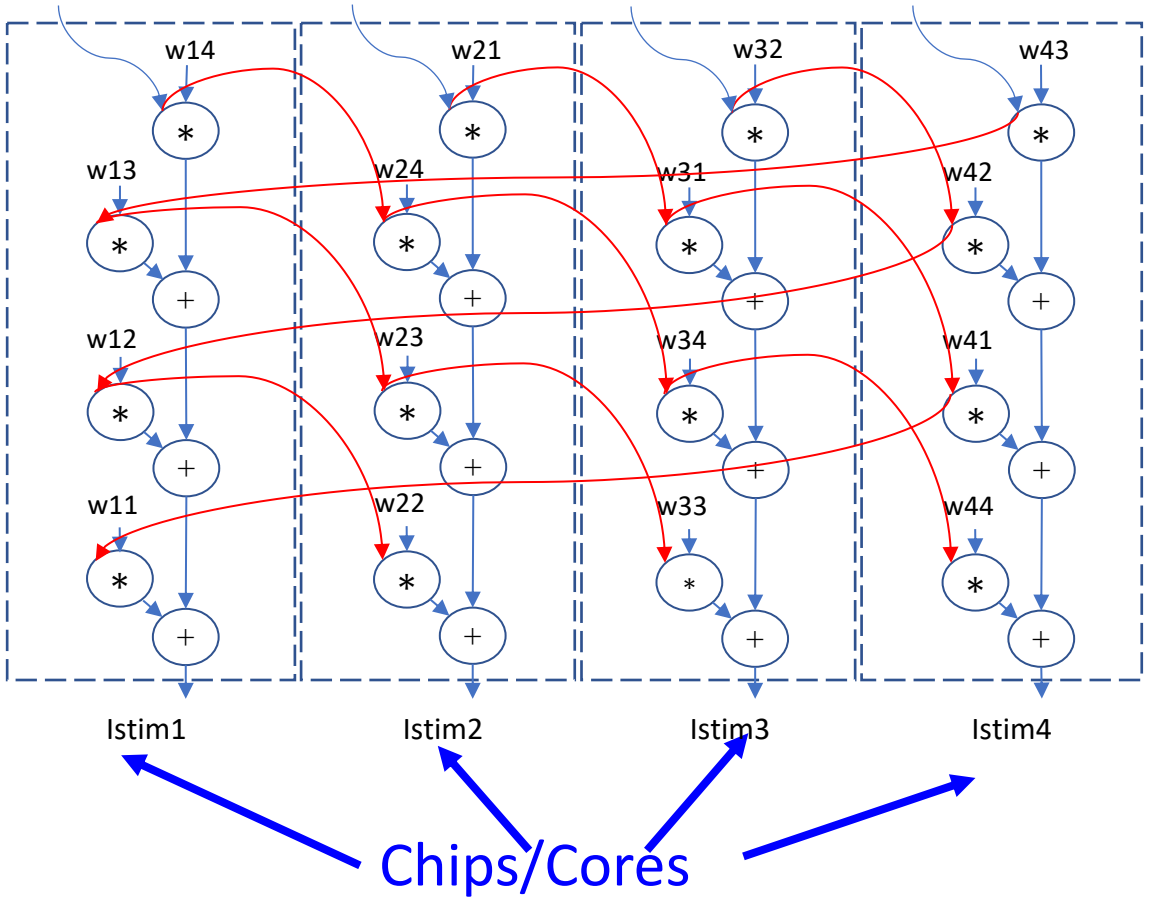
Change of order of computations

- 4X4 weighted sum

$$\begin{pmatrix} I_{stim}1 \\ I_{stim}2 \\ I_{stim}3 \\ I_{stim}4 \end{pmatrix} = \begin{pmatrix} w11 & w12 & w13 & w14 \\ w21 & w22 & w23 & w24 \\ w31 & w32 & w33 & w34 \\ w41 & w42 & w43 & w44 \end{pmatrix} \cdot \begin{pmatrix} I_s1 \\ I_s2 \\ I_s3 \\ I_s4 \end{pmatrix}$$

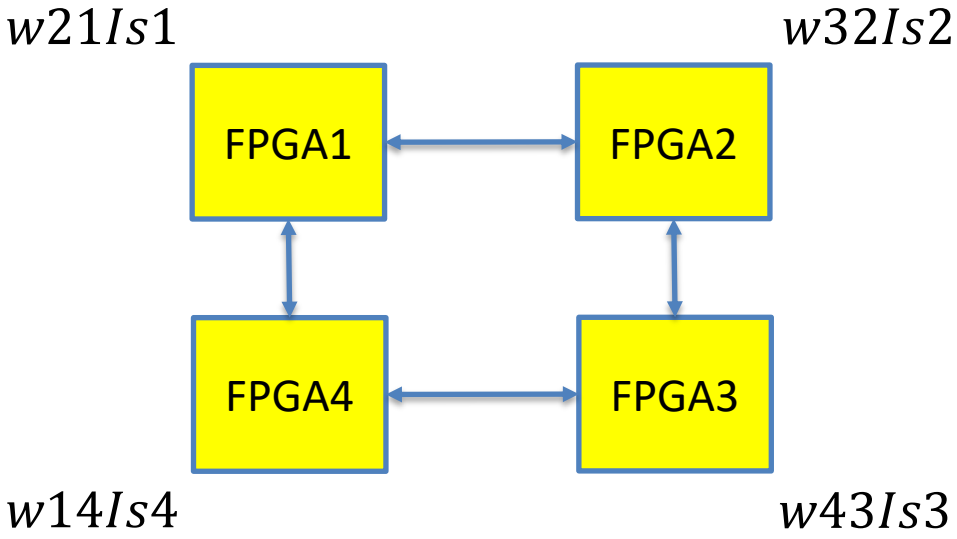
Is4 Is1 Is2 Is3

- Only communicate with neighbors
- One-way ring connection is sufficient



Communication algorithm (1)

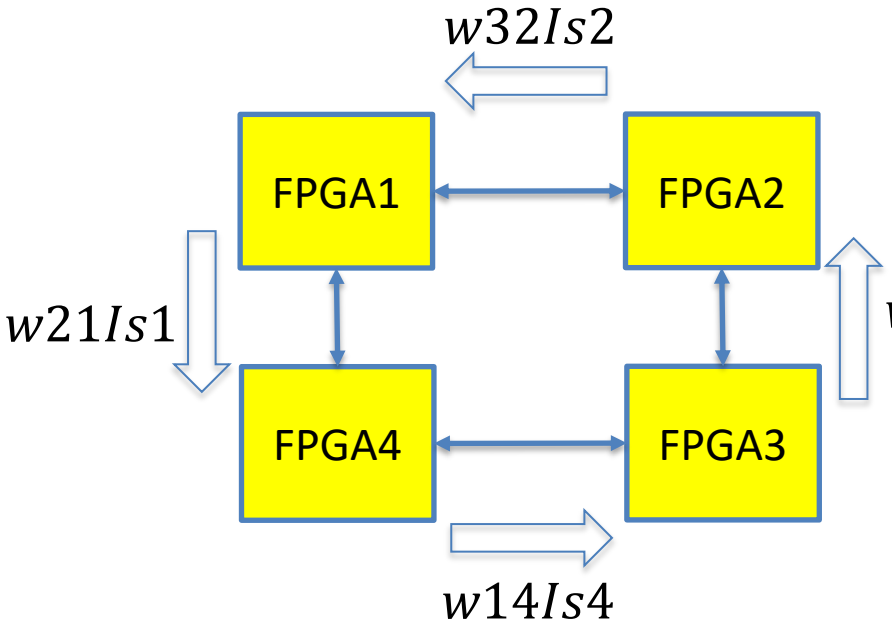
- N=4 (step1)
- All communications are in the same direction



$$\begin{pmatrix} w11 & w12 & w13 & w14 \\ w21 & w22 & w23 & w24 \\ w31 & w32 & w33 & w34 \\ w41 & w42 & w43 & w44 \end{pmatrix}$$

Communication algorithm (2)

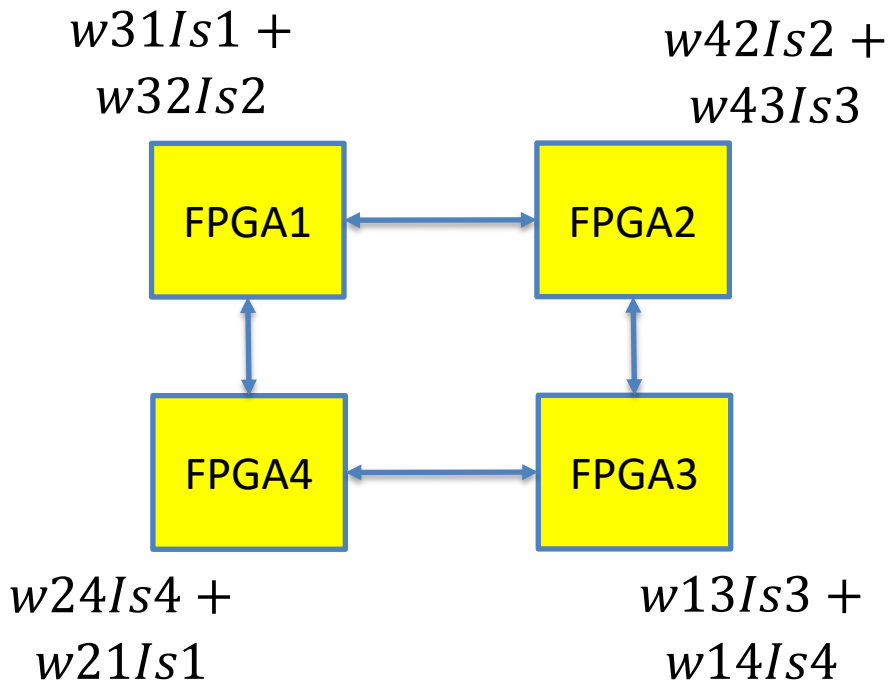
- N=4 (step2)
- All communications are in the same direction



$$\begin{pmatrix} w11 & w12 & w13 & w14 \\ w21 & w22 & w23 & w24 \\ w31 & w32 & w33 & w34 \\ w41 & w42 & w43 & w44 \end{pmatrix}$$

Communication algorithm (3)

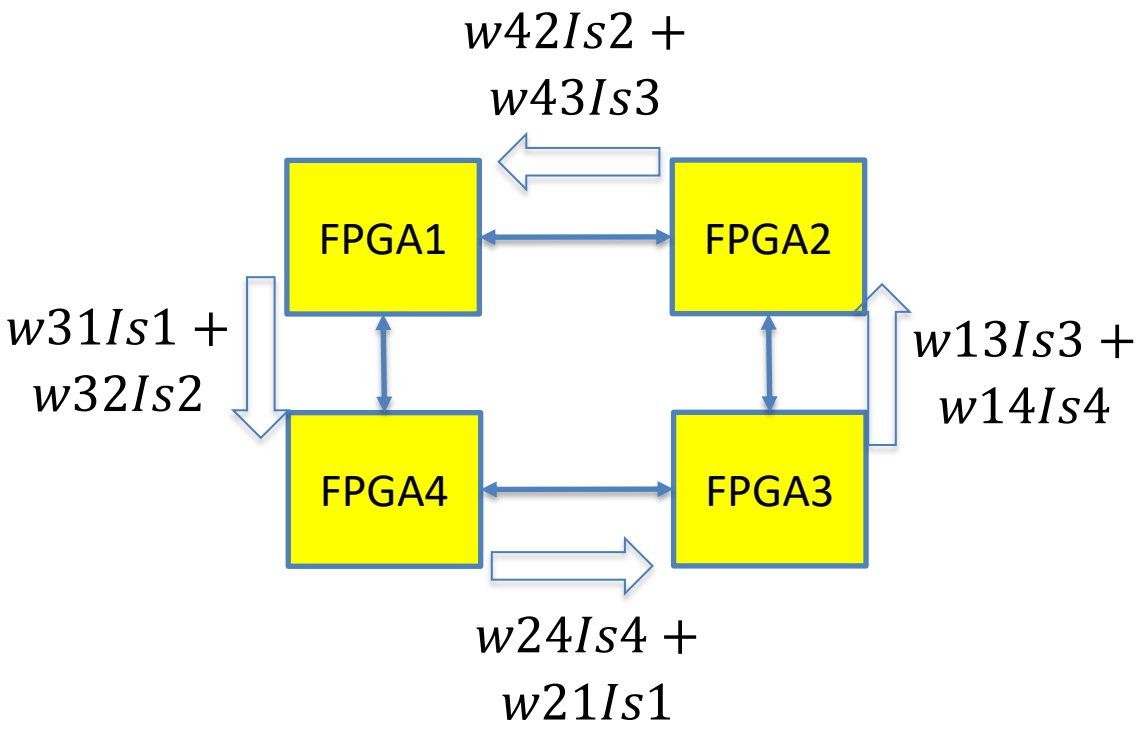
- N=4 (step3)
- All communications are in the same direction



$$\begin{pmatrix} w11 & w12 & w13 & w14 \\ w21 & w22 & w23 & w24 \\ w31 & w32 & w33 & w34 \\ w41 & w42 & w43 & w44 \end{pmatrix}$$

Communication algorithm (4)

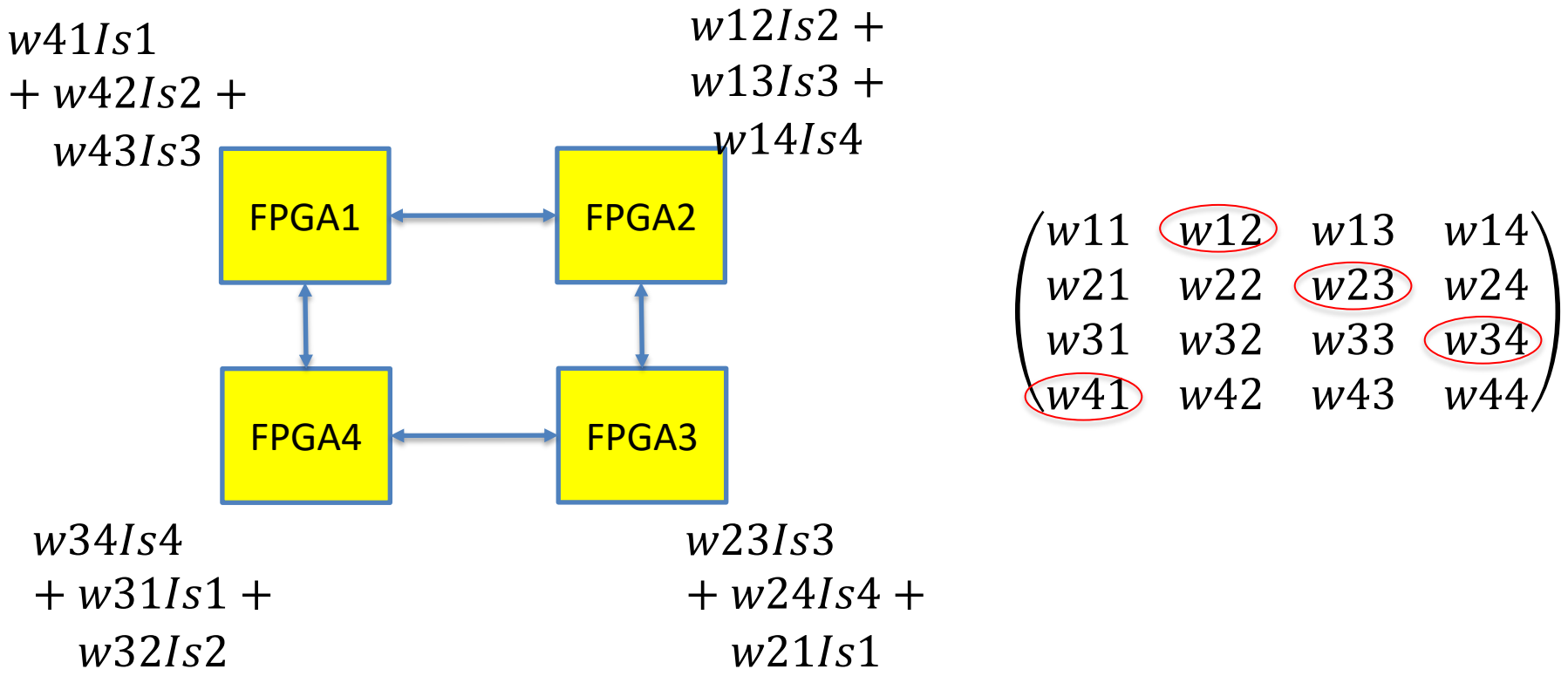
- N=4 (step4)
- All communications are in the same direction



$$\begin{pmatrix} w11 & w12 & w13 & w14 \\ w21 & w22 & w23 & w24 \\ w31 & w32 & w33 & w34 \\ w41 & w42 & w43 & w44 \end{pmatrix}$$

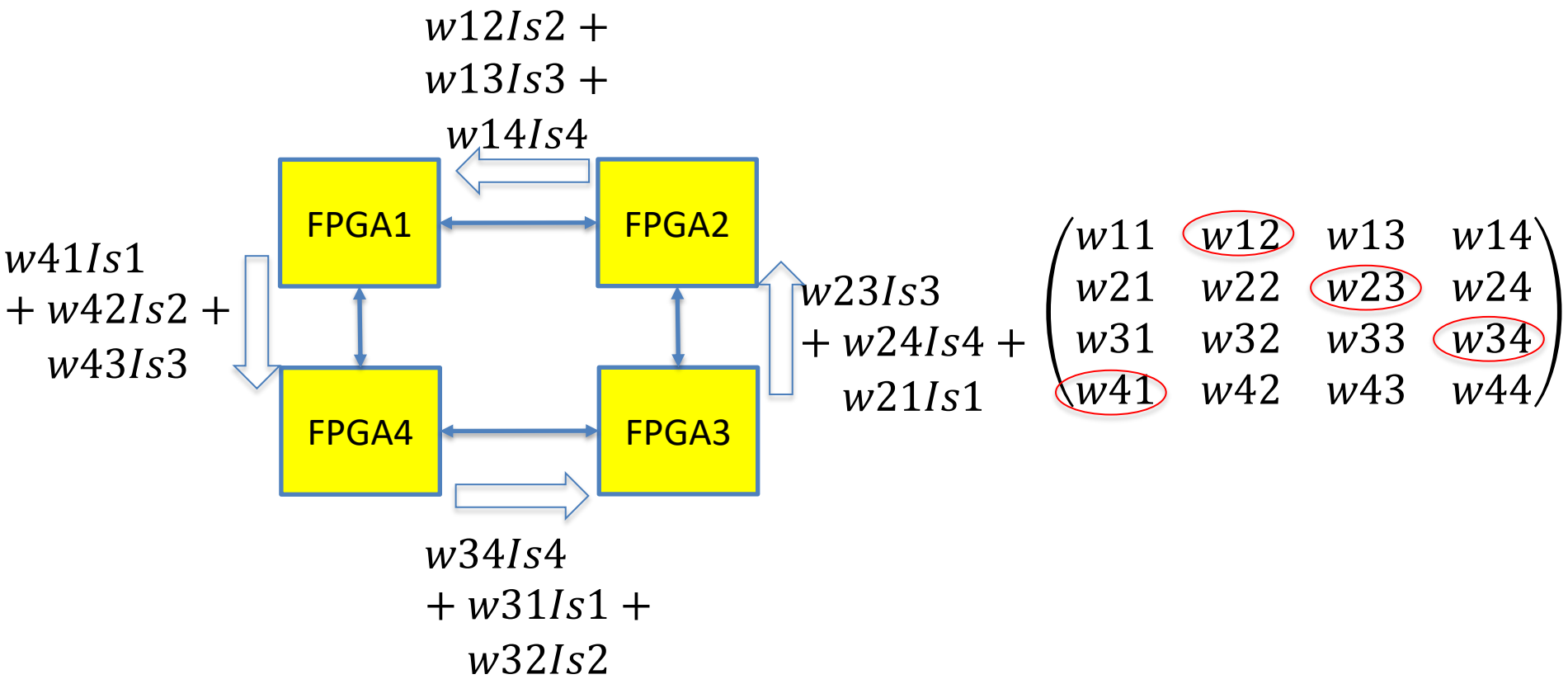
Communication algorithm (5)

- N=4 (step5)
- All communications are in the same direction



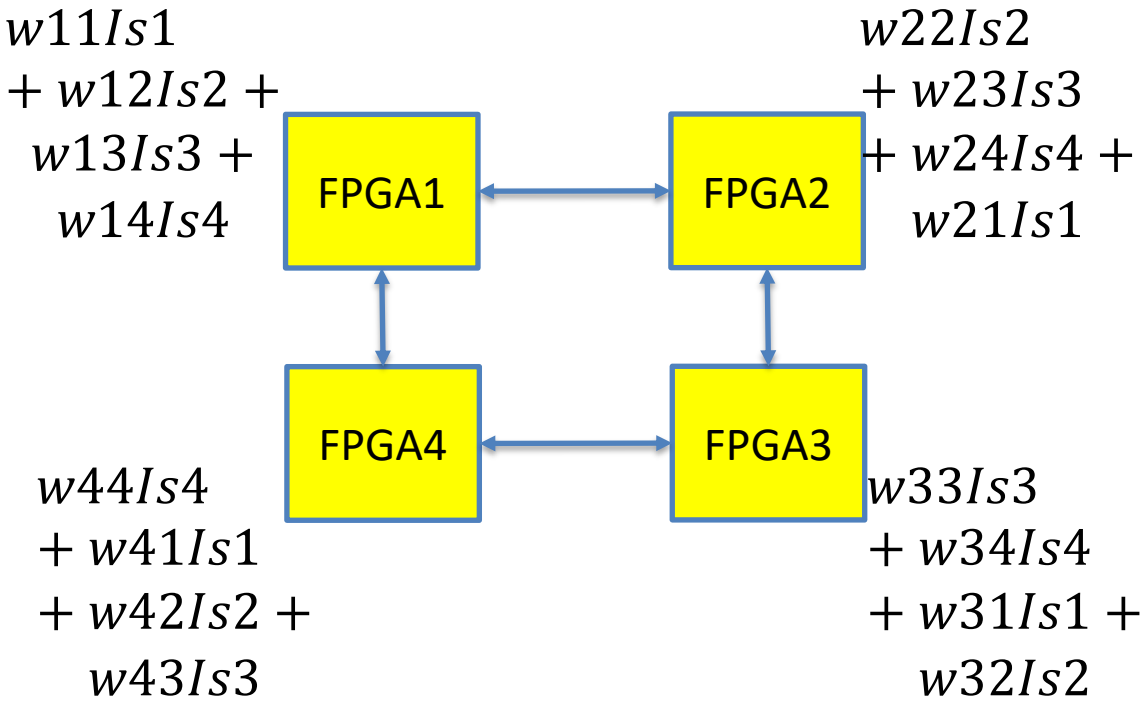
Communication algorithm (6)

- N=4 (step6)
- All communications are in the same direction



Communication algorithm (7)

- N=4 (step7)
- All communications are in the same direction



$$\begin{pmatrix} w11 & w12 & w13 & w14 \\ w21 & w22 & w23 & w24 \\ w31 & w32 & w33 & w34 \\ w41 & w42 & w43 & w44 \end{pmatrix}$$

Can we automatically transform the computations?

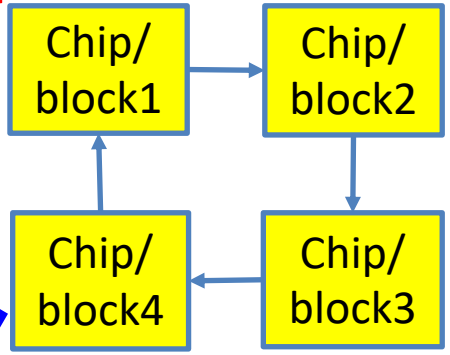
Template based synthesis

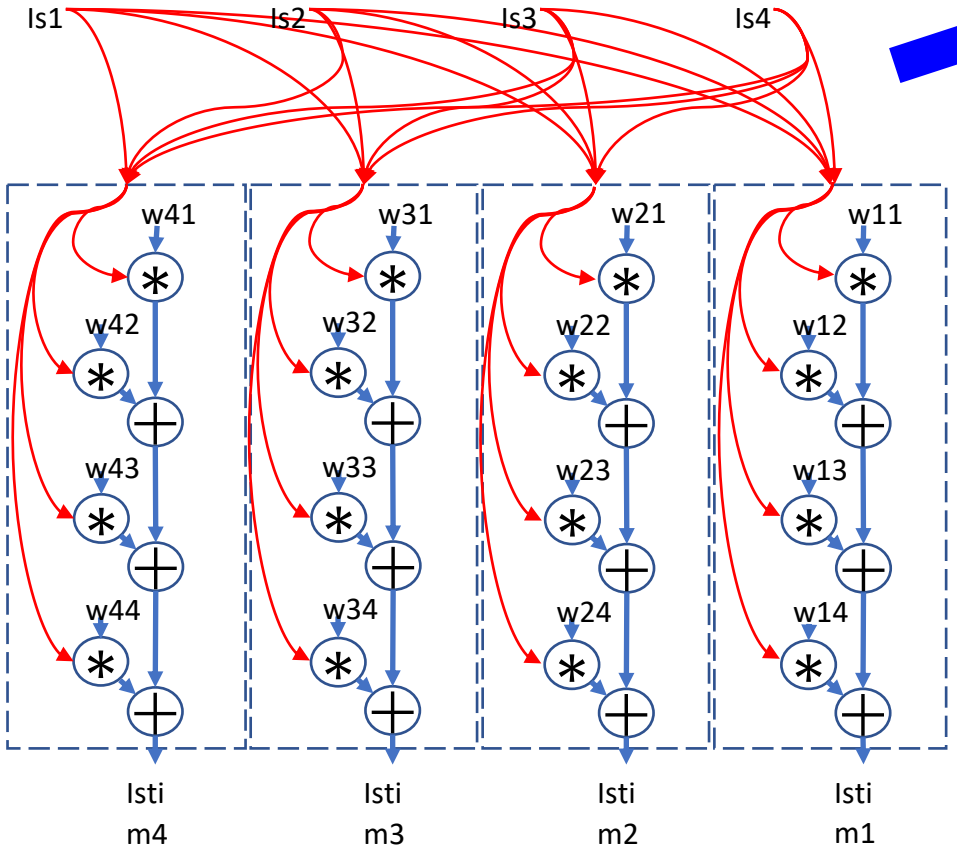
4X4 weighted sum

$$\begin{pmatrix} I_{stim}1 \\ I_{stim}2 \\ I_{stim}3 \\ I_{stim}4 \end{pmatrix} = \begin{pmatrix} w11 & w12 & w13 & w14 \\ w21 & w22 & w23 & w24 \\ w31 & w32 & w33 & w34 \\ w41 & w42 & w43 & w44 \end{pmatrix} \cdot \begin{pmatrix} I_s1 \\ I_s2 \\ I_s3 \\ I_s4 \end{pmatrix}$$

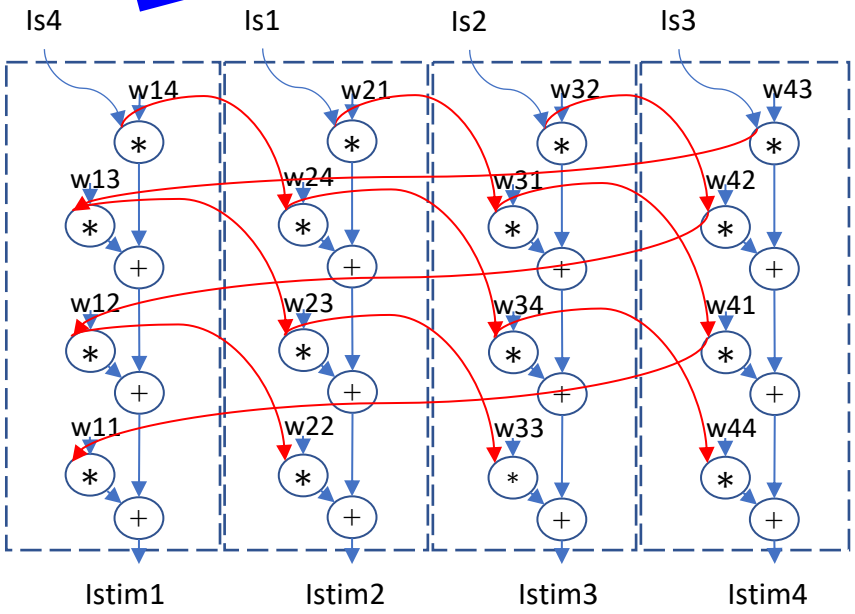
Automatic identification of portions to be transformed

Template for one input stream and one output stream from library

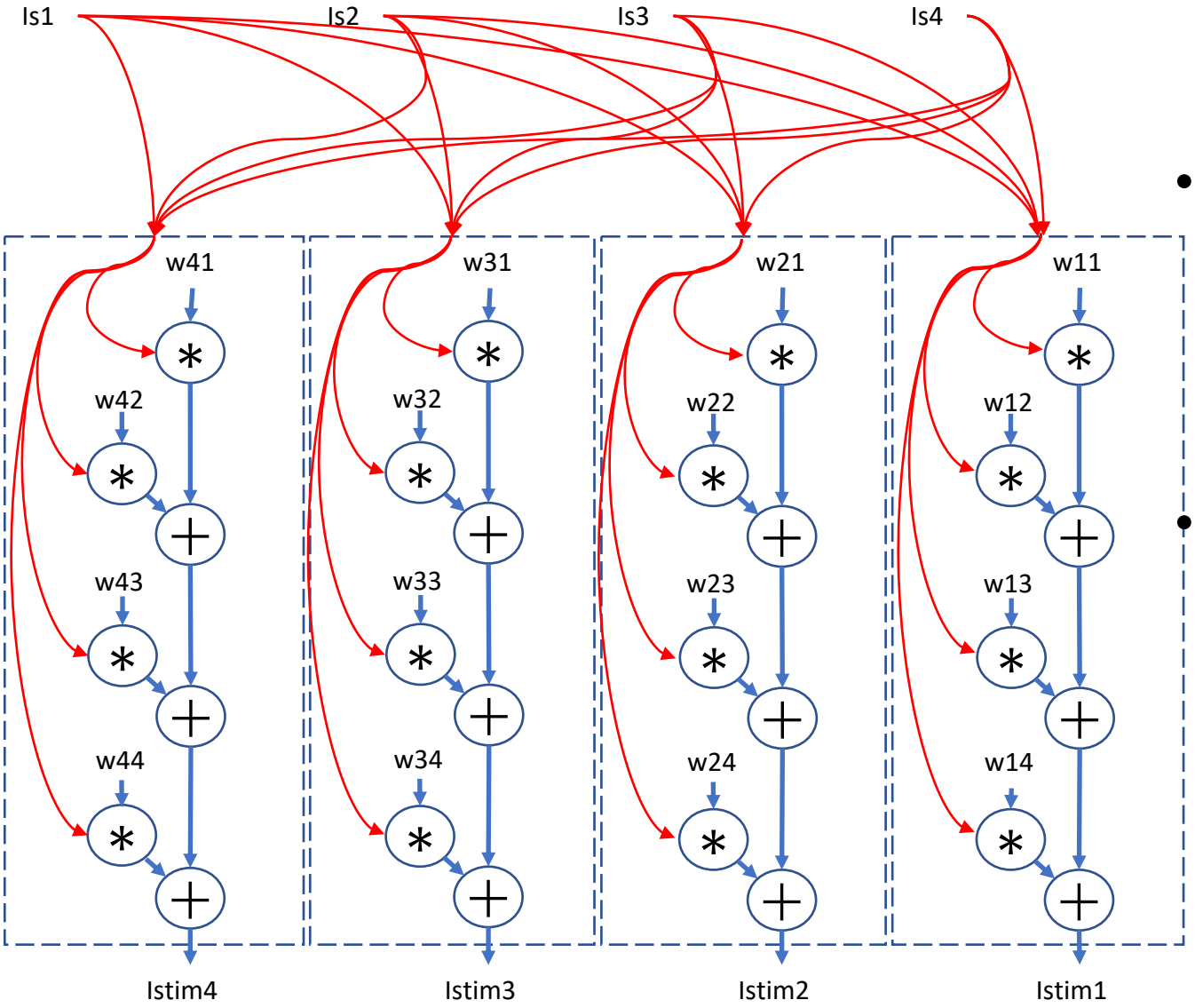




After automatic refinement



Method 1



- Send all vector elements to every node initially
- The communication may become overhead of calculation
- Need lots of storage

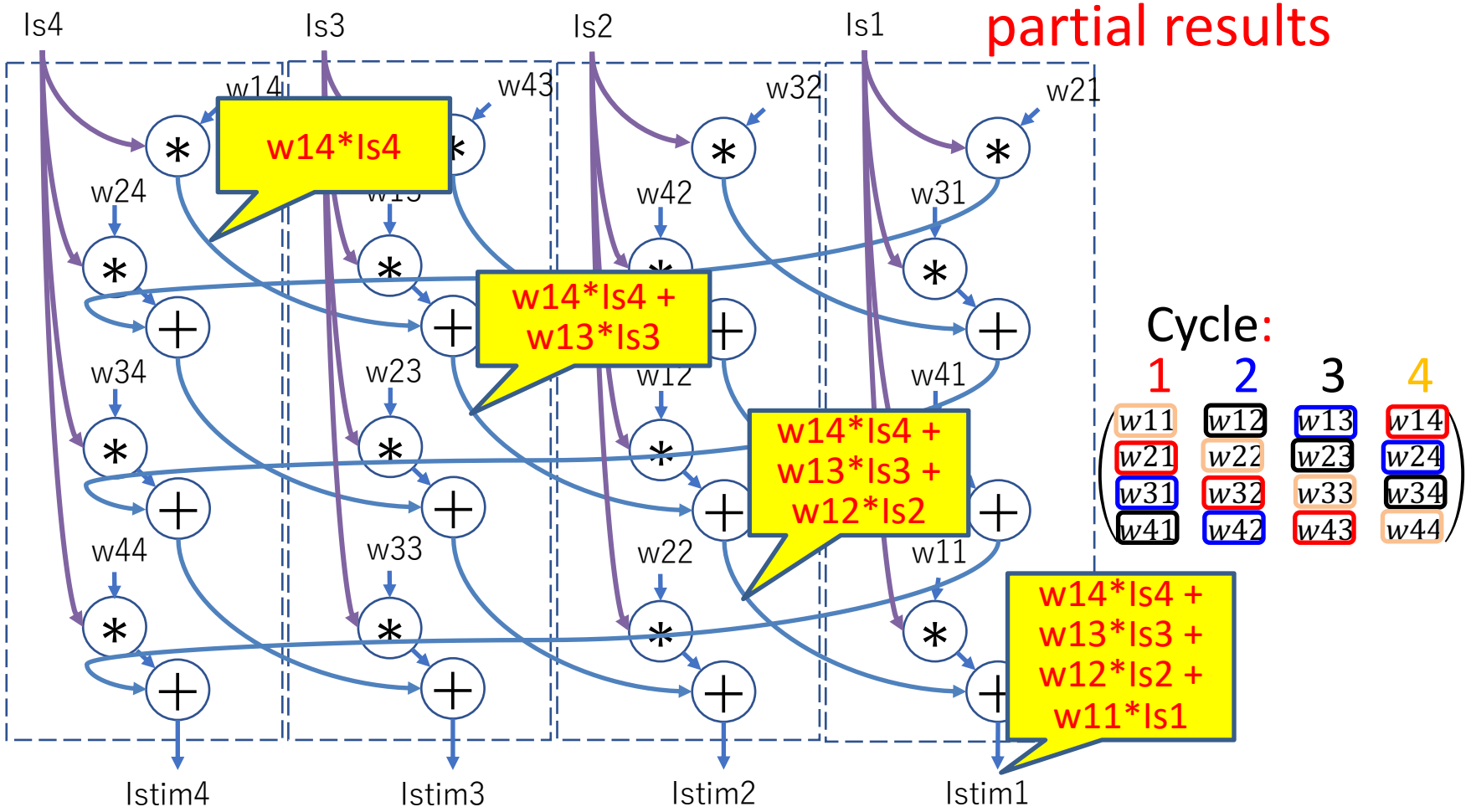
Cycle:

1	2	3	4
$w11$	$w12$	$w13$	$w14$
$w21$	$w22$	$w23$	$w24$
$w31$	$w32$	$w33$	$w34$
$w41$	$w42$	$w43$	$w44$

Method 2 (original manual one)

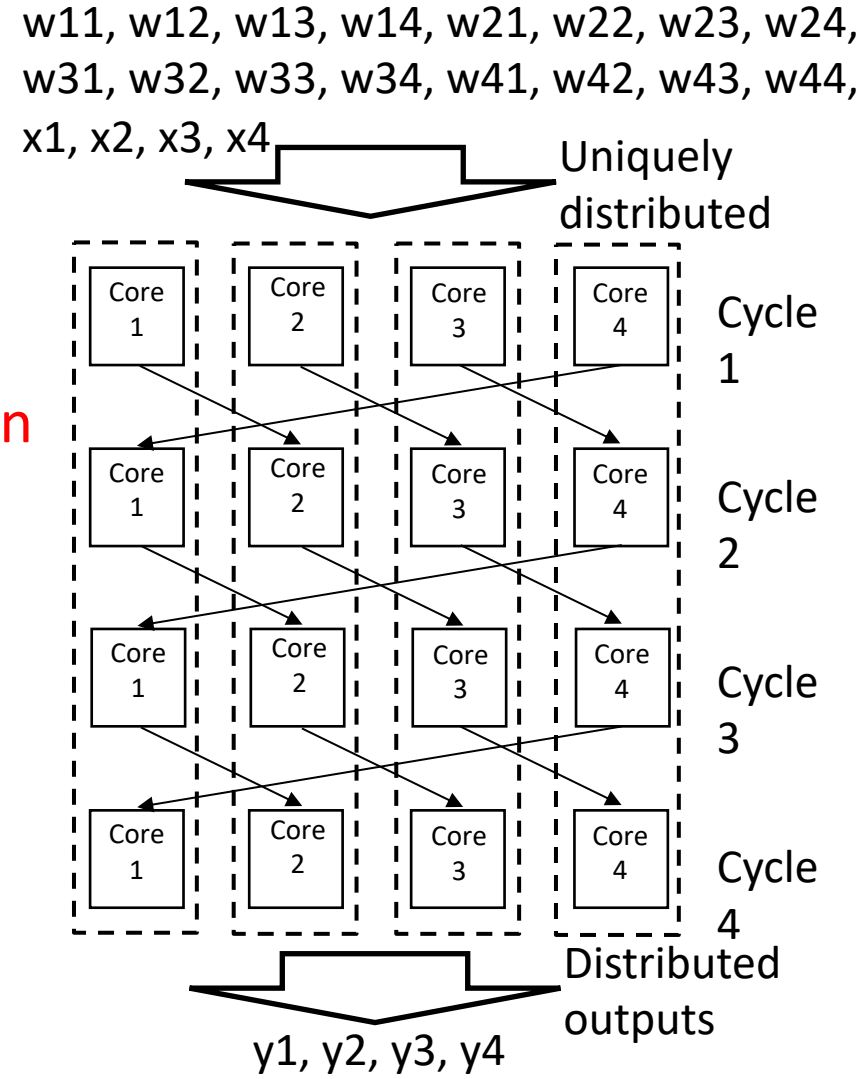
- Communicate partial products among nodes by cycle
- No communication overhead!

Always sending partial results



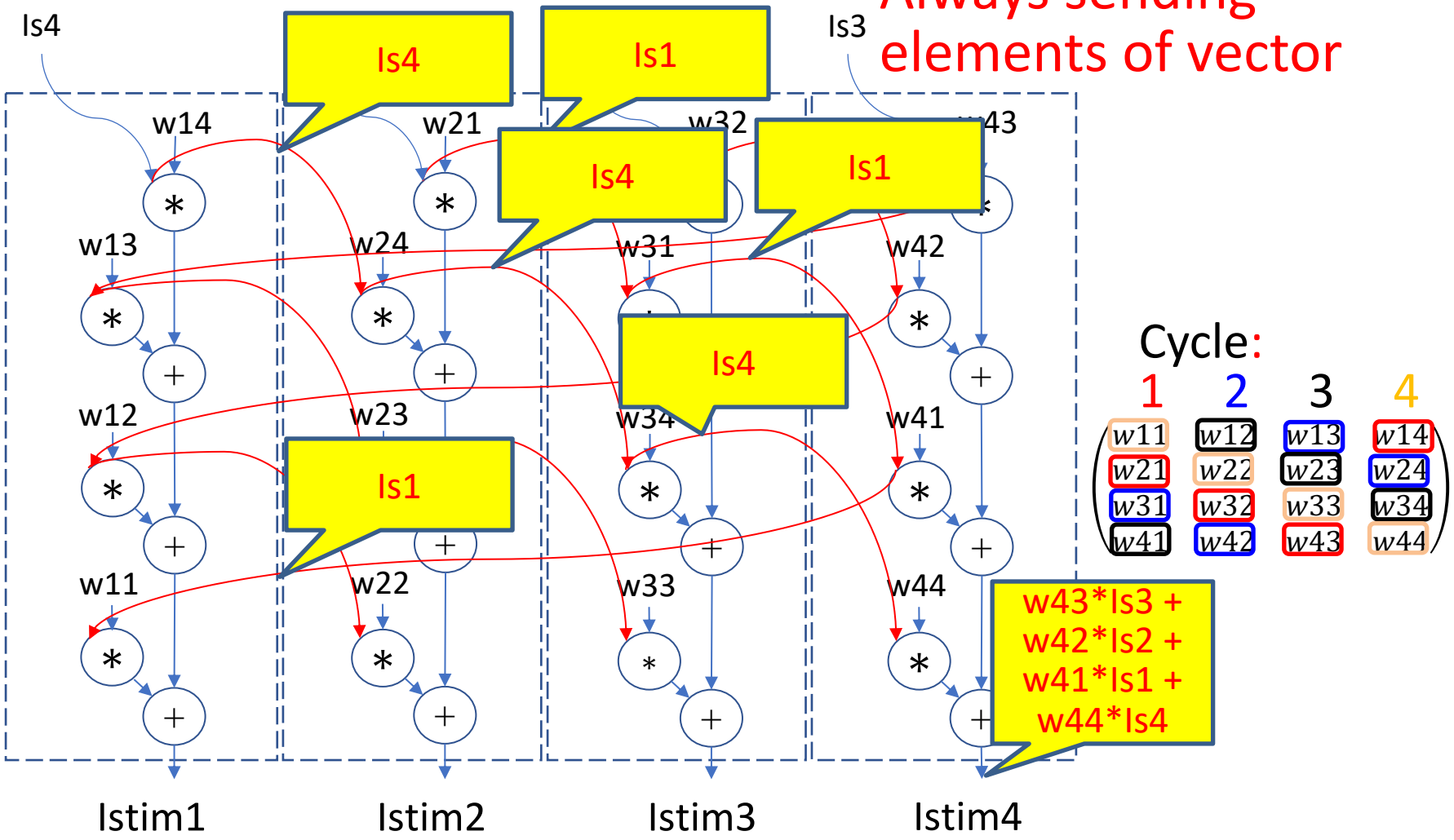
The synthesis problem: 4x4 matrix-vector products on 4 cores

Note: Ring connection



Method 3

- Communicate vector elements among nodes by cycle
- No communication overhead!

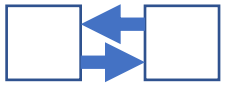


Synthesis example with QBF formulation

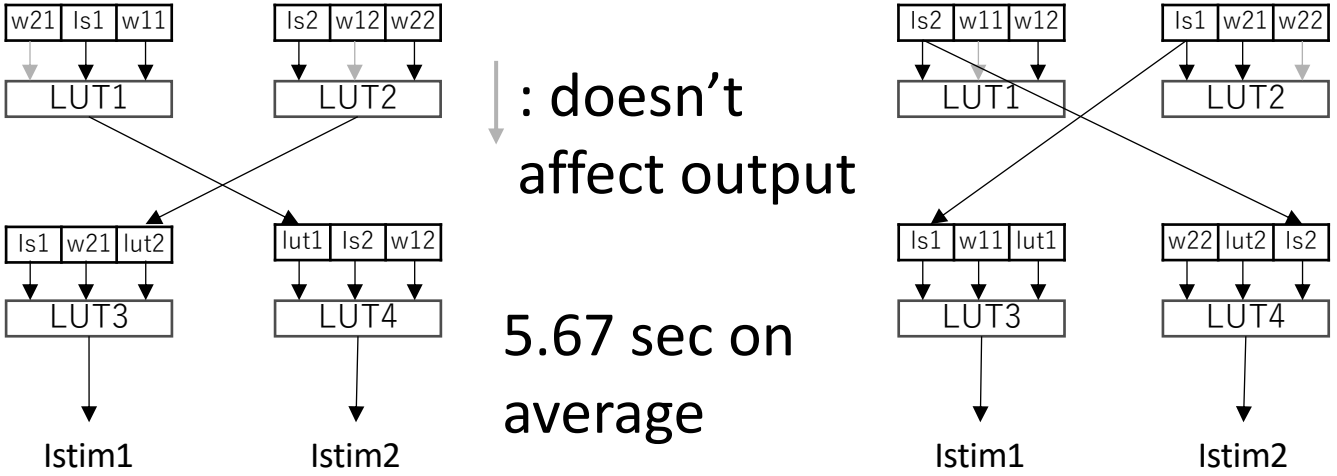
$$\begin{pmatrix} I_{stim}1 \\ I_{stim}2 \end{pmatrix} = \begin{pmatrix} w11 & w12 \\ w21 & w22 \end{pmatrix} \cdot \begin{pmatrix} I_s1 \\ I_s2 \end{pmatrix}$$

2 × 2 Matrix Vector Product

with



2 cores connected mutually



- Correct dataflow was derived with 1 bit variables
- May get different types of solution as shown before

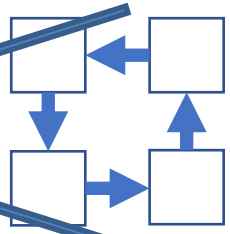
Learning additional constraints for larger problems

$$\begin{pmatrix} I_{stim}1 \\ I_{stim}2 \\ I_{stim}3 \\ I_{stim}4 \end{pmatrix} = \begin{pmatrix} w11 & w12 & w13 & w14 \\ w21 & w22 & w23 & w24 \\ w31 & w32 & w33 & w34 \\ w41 & w42 & w43 & w44 \end{pmatrix} \begin{pmatrix} I_s1 \\ I_s2 \\ I_s3 \\ I_s4 \end{pmatrix}$$

4 × 4 Matrix Vector Product

Cannot solve

with



4 cores connected by ring

Make Small Instance

$$\begin{pmatrix} I_{stim}1 \\ I_{stim}2 \end{pmatrix} = \begin{pmatrix} w11 & w12 \\ w21 & w22 \end{pmatrix} \cdot \begin{pmatrix} I_s1 \\ I_s2 \end{pmatrix}$$

2 × 2 Matrix Vector Product

with

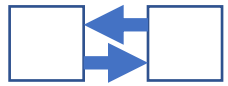
2 cores connected mutually

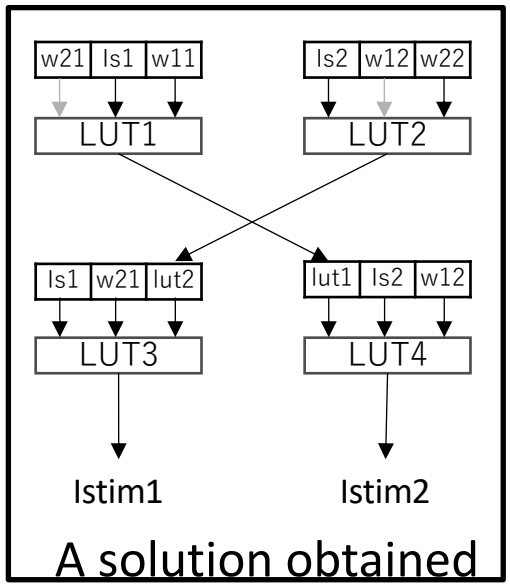
Easy to solve

Add Constraints on MUXs and LUTs

Analysis of solution obtained (1)

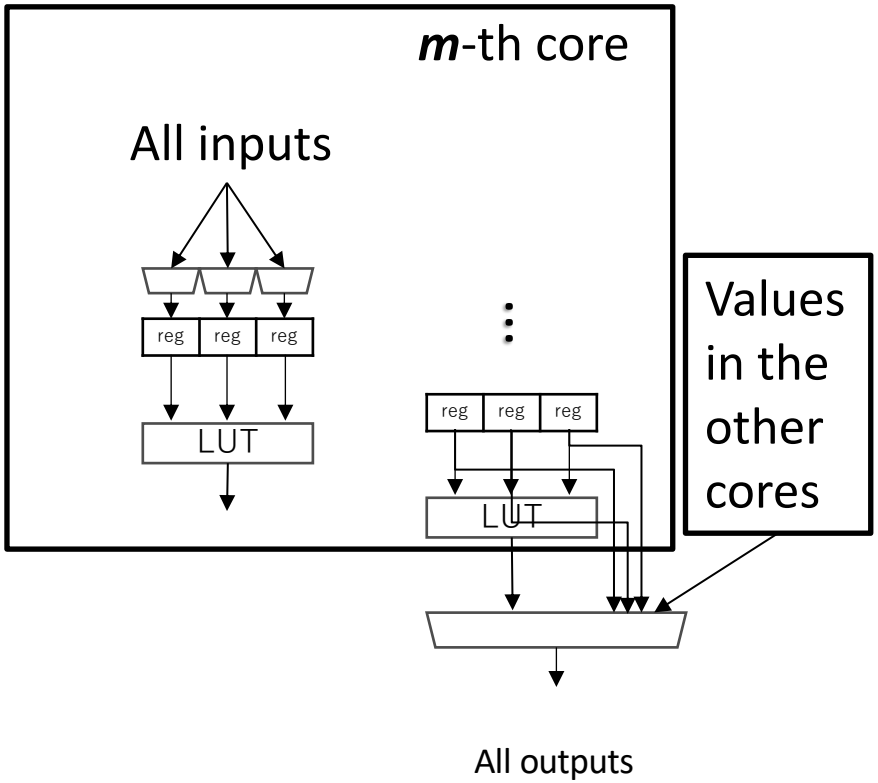
$$\begin{pmatrix} I_{stim}1 \\ I_{stim}2 \end{pmatrix} = \begin{pmatrix} w11 & w12 \\ w21 & w22 \end{pmatrix} \cdot \begin{pmatrix} I_s1 \\ I_s2 \end{pmatrix} \text{ with}$$





Template

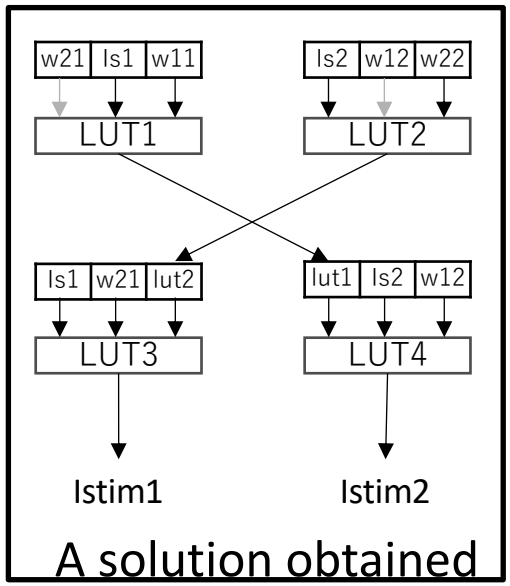
Synthesis



Analysis of solution obtained (1)

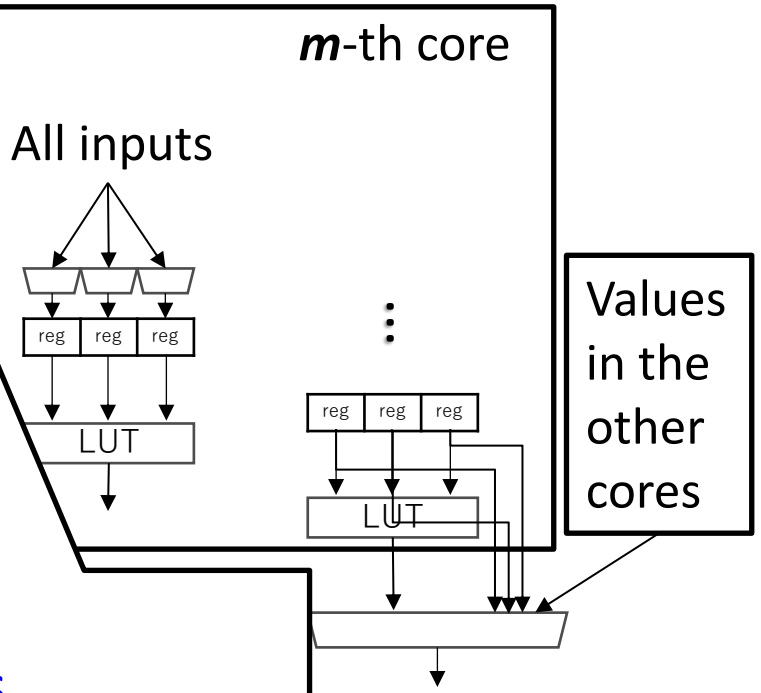
$$\begin{pmatrix} I_{stim}1 \\ I_{stim}2 \end{pmatrix} = \begin{pmatrix} w11 & w12 \\ w21 & w22 \end{pmatrix} \begin{pmatrix} I_s1 \\ I_s2 \end{pmatrix}$$ with

Core1 Core2



Templa e

Synthe



Extracted constraints:

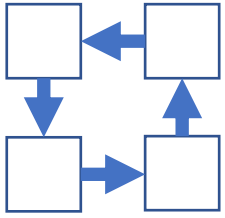
- Column based partitioning into cores
- Opposite order of summation
- Computations are multiplication and addtion

These should be generalized for 4x4 case

Synthesis with Additional Constraints

$$\begin{pmatrix} I_{stim}1 \\ I_{stim}2 \end{pmatrix} = \begin{pmatrix} w11 & w12 \\ w21 & w22 \end{pmatrix} \cdot \begin{pmatrix} I_s1 \\ I_s2 \end{pmatrix}$$ with 

5.67sec → Additional Constraints → 0.16sec

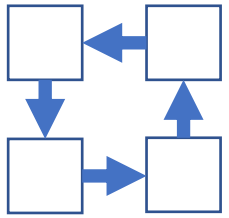
$$\begin{pmatrix} I_{stim}1 \\ I_{stim}2 \\ I_{stim}3 \\ I_{stim}4 \end{pmatrix} = \begin{pmatrix} w11 & w12 & w13 & w14 \\ w21 & w22 & w23 & w24 \\ w31 & w32 & w33 & w34 \\ w41 & w42 & w43 & w44 \end{pmatrix} \cdot \begin{pmatrix} I_s1 \\ I_s2 \\ I_s3 \\ I_s4 \end{pmatrix}$$ with 

Infeasible → Additional Constraints → 53.1sec

Further Example

Synthesis for

$$\begin{pmatrix} I_{stim}1 \\ \dots \\ I_{stim}32 \end{pmatrix} = \begin{pmatrix} w(1,1) & \dots & w(1,32) \\ \dots & \dots & \dots \\ w(32,1) & \dots & w(32,32) \end{pmatrix} \cdot \begin{pmatrix} I_s1 \\ \dots \\ I_s32 \end{pmatrix}$$

with 

32 × 32 Matrix Vector Product

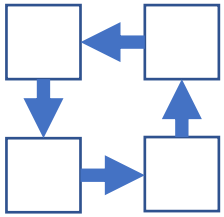
Infeasible

As explained before → Make Small Instance

2 × 2 Matrix Vector Product

with 

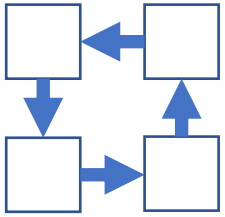
4 × 4 Matrix Vector Product

with 

→ **4 × 4** Matrix Vector Product

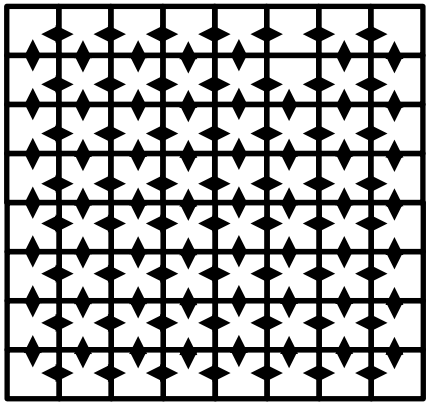
with 

→ **8 × 8** Matrix Vector Product

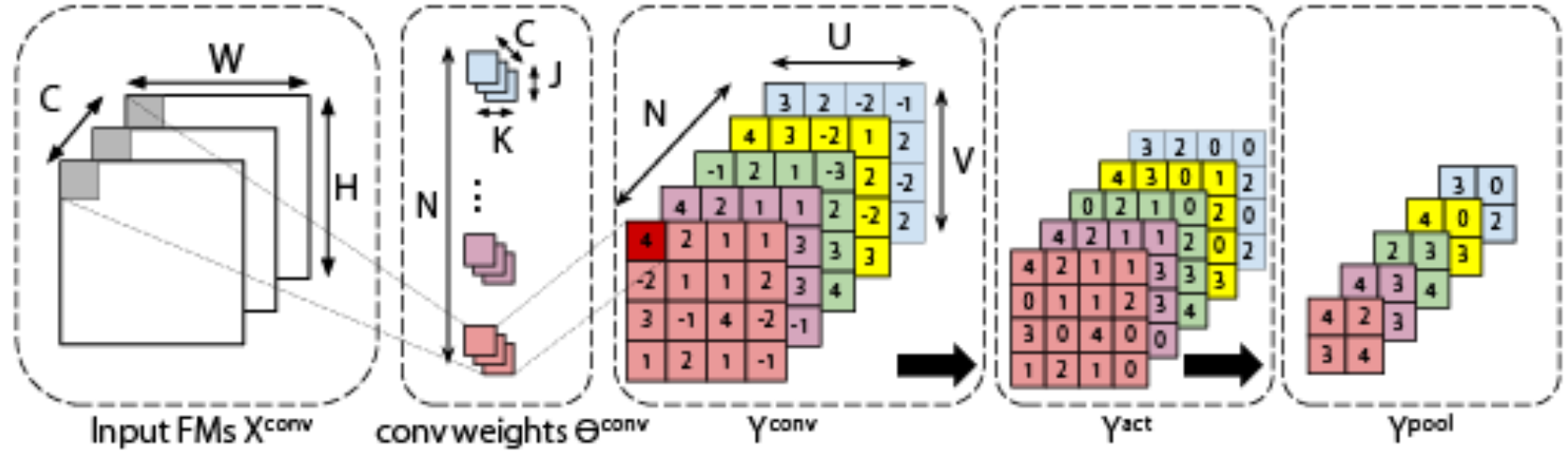
with 

Application example: 1st layer of CNN for image classification

- Implement the 1st layer of the CNN
  - Typically used in image recognition/classification
  - Realize on mesh-architecture "optimally"

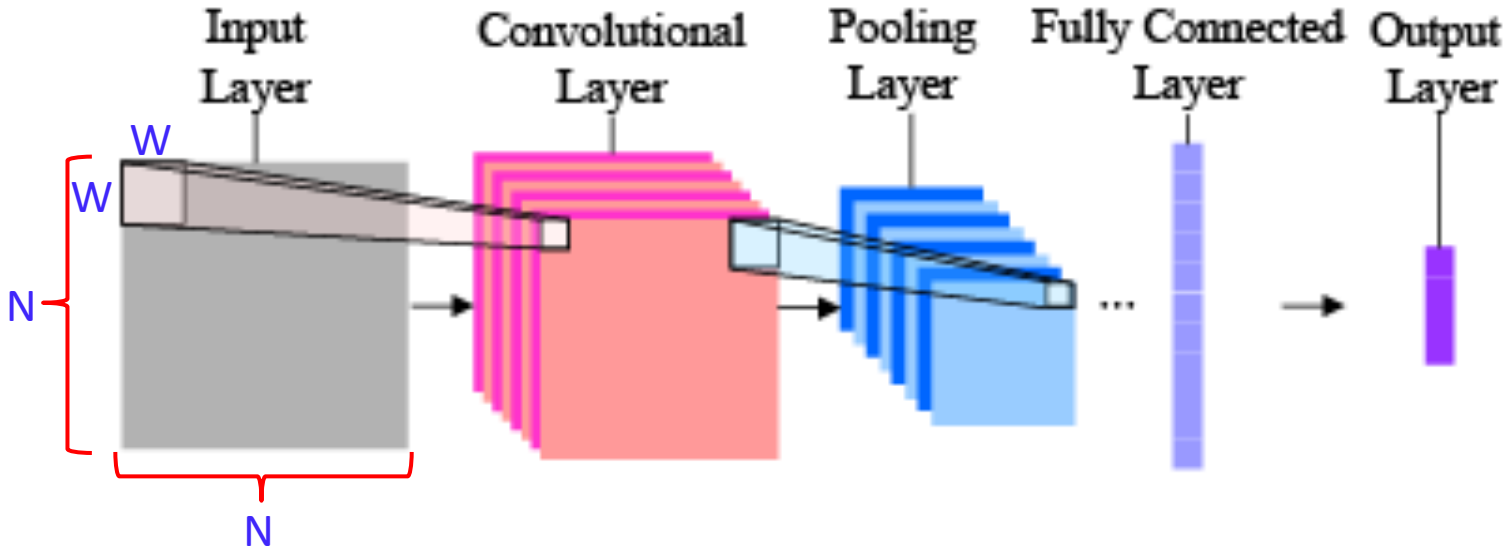


Implement this part on mesh-architectures only with 4 neighbor communication



N^2 parallel computation on NxN mesh

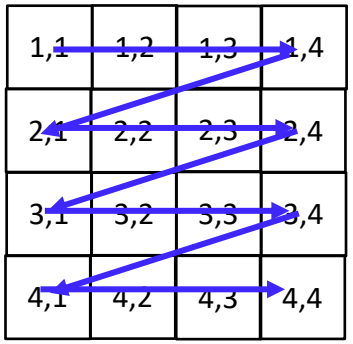
- NxN images on NxN mesh architecture (N^2 MAC units)
- Window size: WxW (N^2 such widows)
- In total $N^2 \cdot W^2$ MAC operations
- Theoretical optimum= $N^2 \cdot W^2 / N^2 = W^2$ cycles for all
- Typical numbers: N=128, W=4, and so all computations should finish in W^2=16 cycles



The key

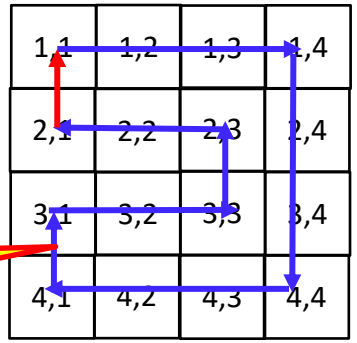
- Change the order of computation in convolution
  - Original:

    (1,1)→(1,2) →(1,3) →(1,4) →(2,1) →(2,2) →(2,3) →(2,4)→(3,1) →(3,2) →(3,3) →(3,4) →(4,1) →(4,2) →(4,3) →(4,4)



  - Proposed, for example:

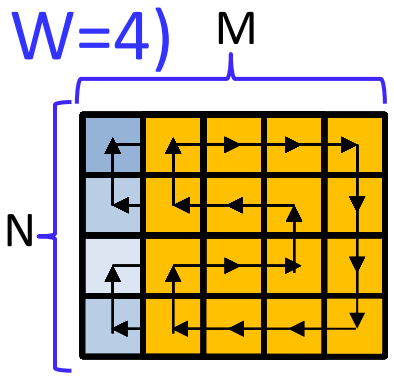
    (1,1)→(1,2) →(1,3) →(1,4) →(2,4) →(3,4) →(4,4) →(4,3)→(4,2) →(4,1) →(3,1) →(3,2) →(3,3) →(2,3) →(2,2) →(2,1)



This is a ring communication

Convolutional NN on mesh architecture

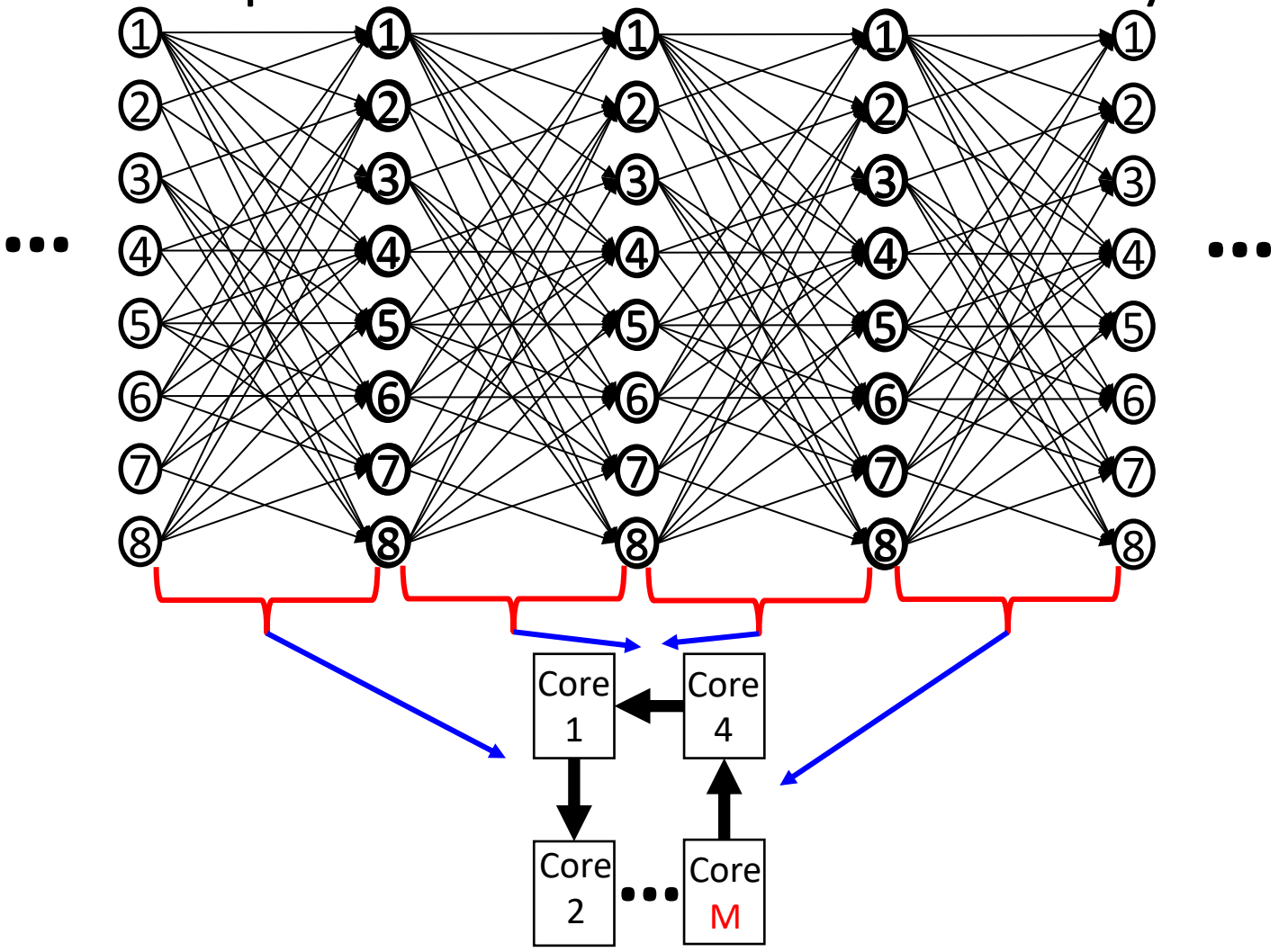
- Realizing theoretical optimum on mesh-architectures
- Mesh has N*M MAC units connected only to 4 neighbors
- There are around N*M windows
- With window size W, takes W^2 cycles for all computations
- Joint work with Dr. Alan Mishchenko of UCB
- We have no plan to apply for patent regarding to this algorithm!
- Our initial implementation on FPGA is 1,000 times faster than GPU (N=128, W=4)



Typical numbers:
N=M=128, W=4
N=M=1024, W=10

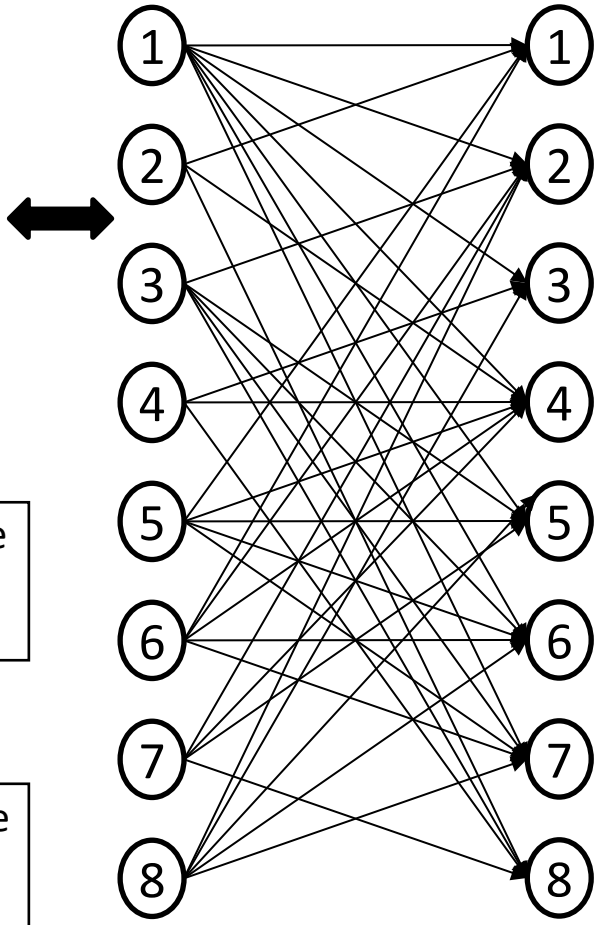
Parallel processing of deep learning on multi cores with ring connection

- Overall computation should be accelerated by M times



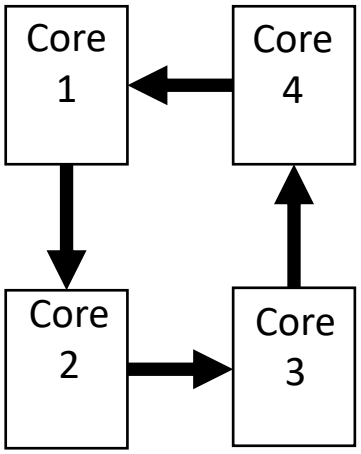
Sparse matrix is also OK to be compiled

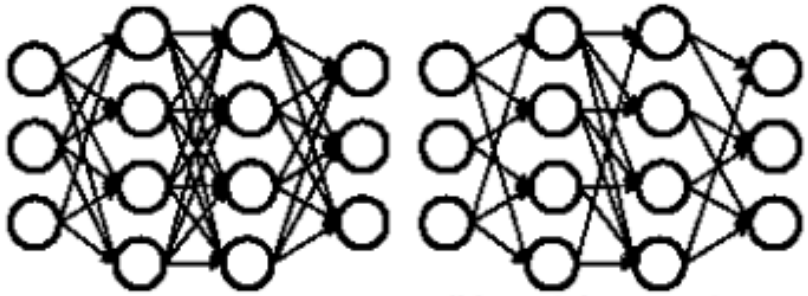
$$\begin{pmatrix} y1 \\ y2 \\ y3 \\ y4 \\ y5 \\ y6 \\ y7 \\ y8 \end{pmatrix} = \begin{pmatrix} w11 & w12 & w13 & w14 & w15 & w16 & w17 & 0 \\ w21 & 0 & 0 & w24 & 0 & 0 & 0 & w28 \\ 0 & w32 & 0 & 0 & w35 & w36 & w37 & w38 \\ 0 & 0 & w43 & w44 & 0 & 0 & 0 & w48 \\ w51 & 0 & 0 & w54 & w55 & w56 & w57 & 0 \\ w61 & w62 & 0 & w64 & 0 & w66 & w67 & 0 \\ 0 & w72 & 0 & w74 & w75 & 0 & 0 & w78 \\ 0 & w82 & w83 & 0 & w85 & w86 & w87 & 0 \end{pmatrix} \cdot \begin{pmatrix} x1 \\ x2 \\ x3 \\ x4 \\ x5 \\ x6 \\ x7 \\ x8 \end{pmatrix}$$



- 8*8-27=37
- 37/4=9.25 => 10 cycles
- Our method generate a scheduling with 10 cycles

on top of

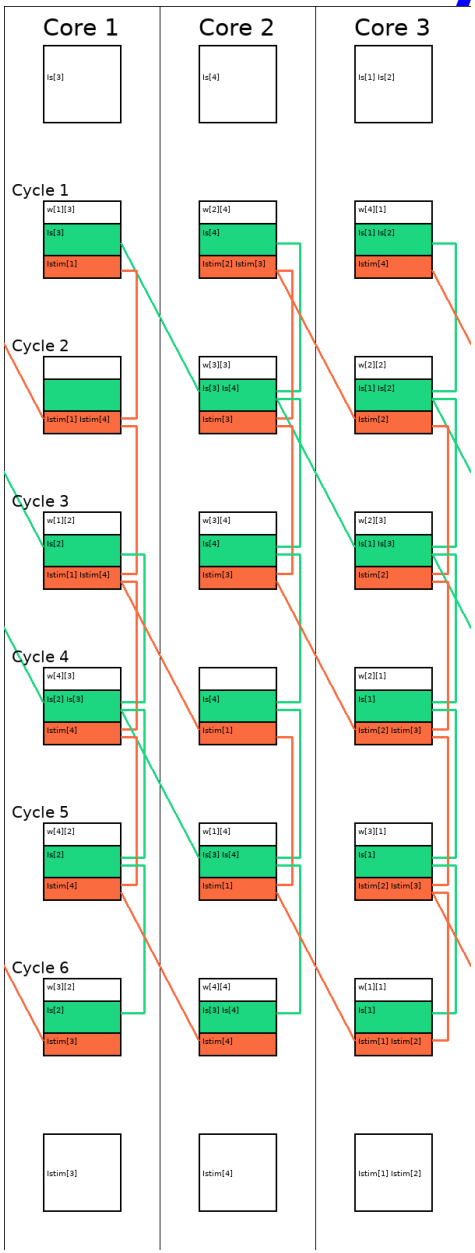




Scheduling is very complicated and hard to understand at all !

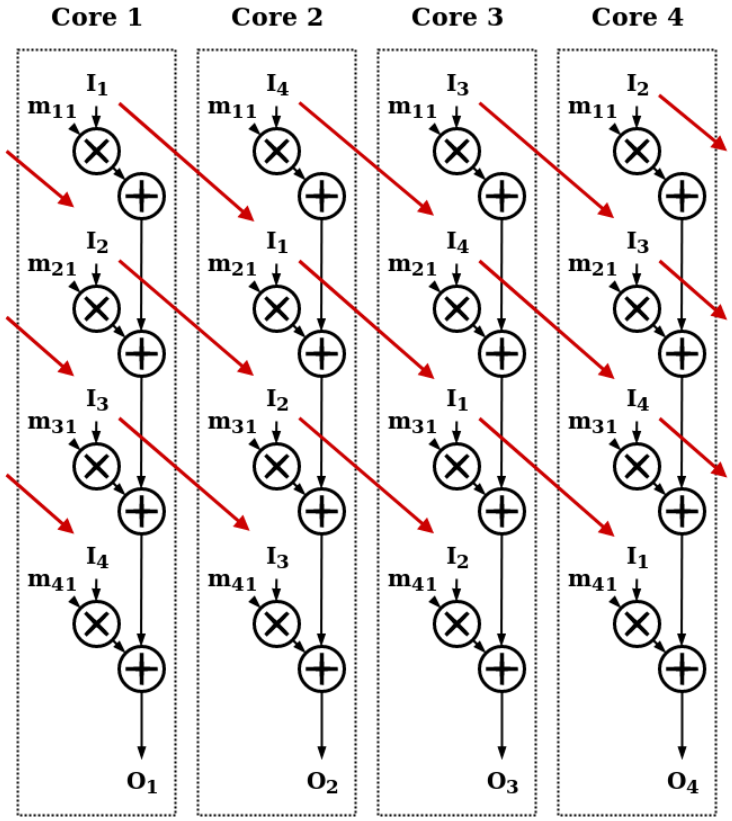
By the way when N is not dividable by M

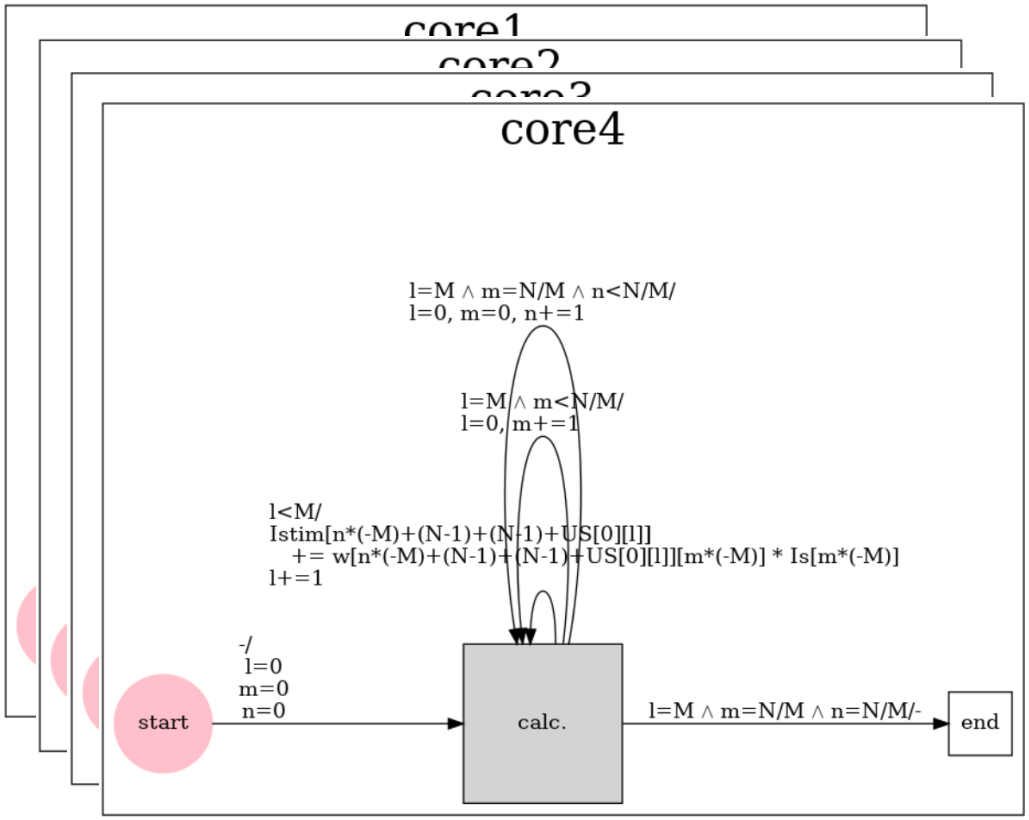
- 4x4 matrix on 3 cores
  - Can be automatically synthesized
- Red: Sending vector element
- Green: Sending partial result
- Sparse connections need similar data transfers
- Very complicated!
  - No way to apply induction!
  - Analysis is on-going



Loop Generalization Tool

- Template for loop based description is assumed
- From the synthesis results for small instances, automatically infer loop parameters



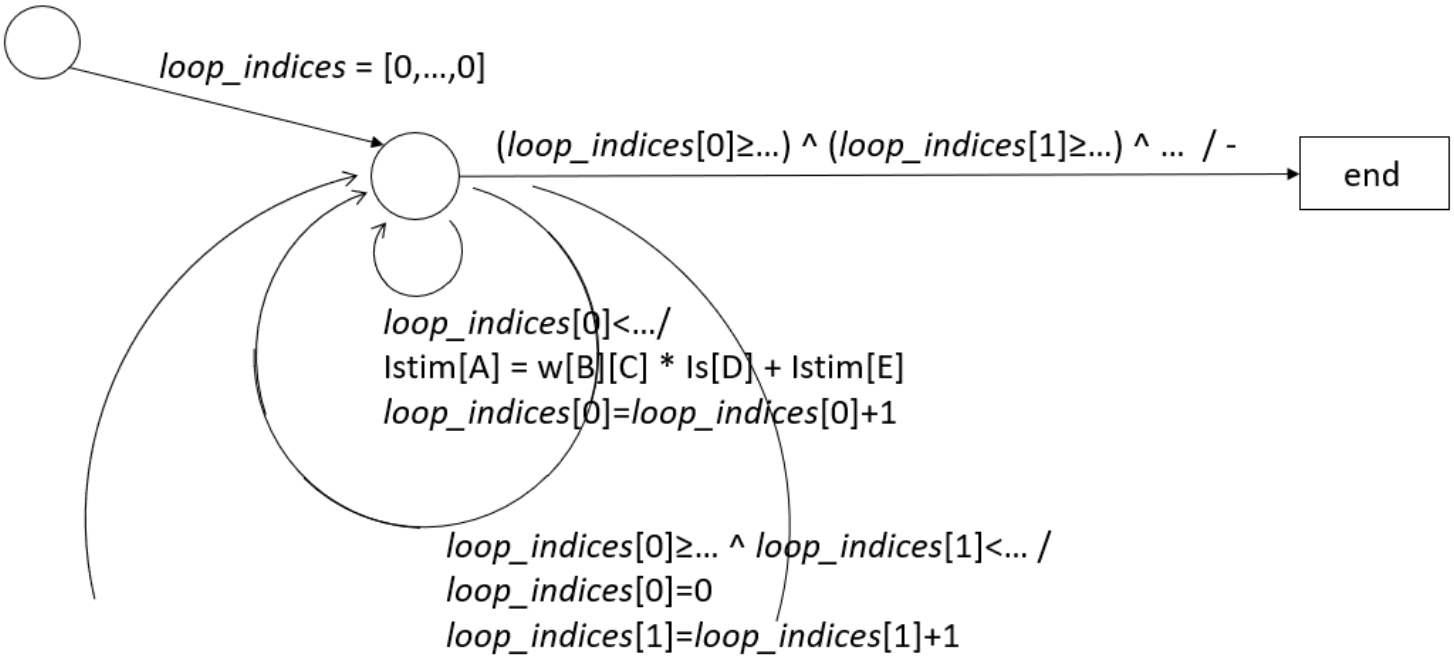


Results

- Same color means same constraints are used

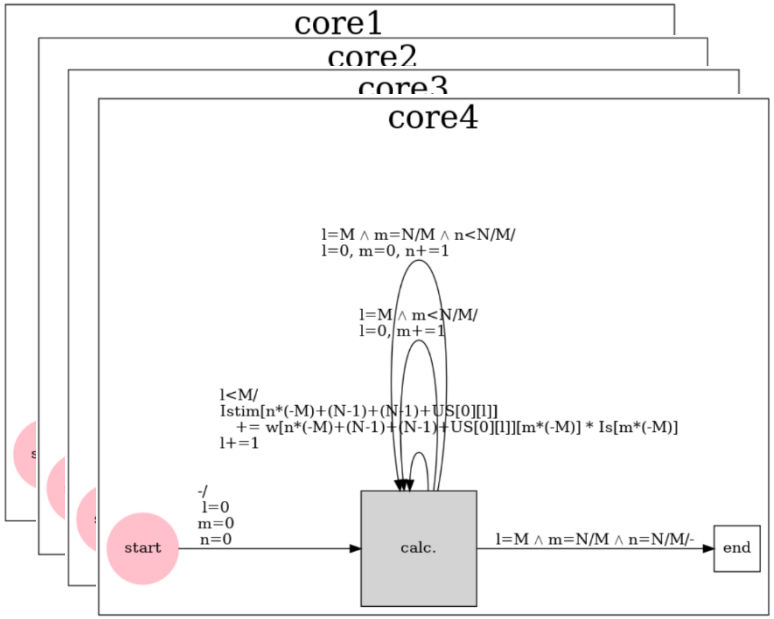
	Synthesis		Mapping	Generalization
2x2 on 2 cores	1.3 s	0.08 s	-	-
4x4 on 4 cores	138 s	0.65 s	0.07 s	1.4 s
4x4 on 2 cores	472 s	0.41 s	-	-
8x8 on 4 cores	4.9 s	0.76 s	0.25 s	12.3 s
16x16 on 4 cores	22.5 s	-	2.26 s	21.5 s
32x32 on 4 cores	3 h	-	40.4 s	265 s

- Generates : Template
- ▪How many nested loops?
- ▪How are these indices represent?
- ▪Wh
- co



Check flow

Matrix size = 8 x 8
cores = 4



How about
Matrix size = 16 x 16
cores = 4

Algorithm for larger matrix is obtained!!

[1]A. M. Gharehbaghi, T. Maruoka and M. Fujita, "A New Reconfigurable Architecture with Applications to IoT and Mobile Computing", IFIP Internet of Things Conference (IoT 2018), Sep. 2018, Springer International Publishing, IFIP Advances in Information and Communication Technology, in press