Update on Ara

18/05/2022

Matteo Perotti
Matheus Cavalcante
Nils Wistoff
Gianmarco Ottavi

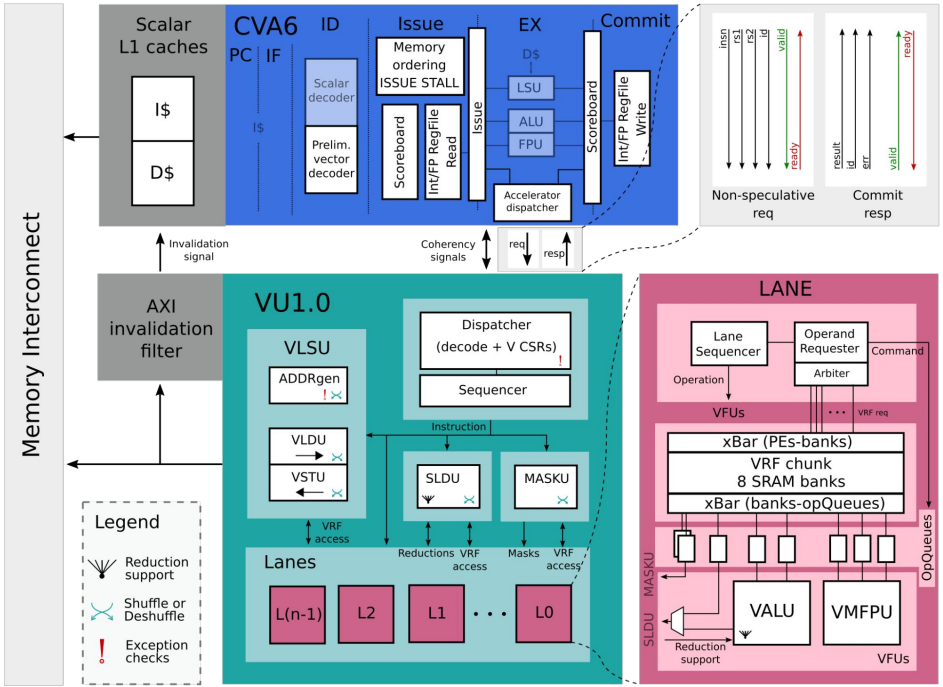
Professor Luca Benini
Integrated Systems Laboratory
ETH Zürich

Summary

- **Conference paper**
- **WIP: benchmarks**
  - FFT
- **Kernel:**
  - **3D 3x3 Fconv**
- **Student projects**
  - FP reductions
- **X-interface, future projects**

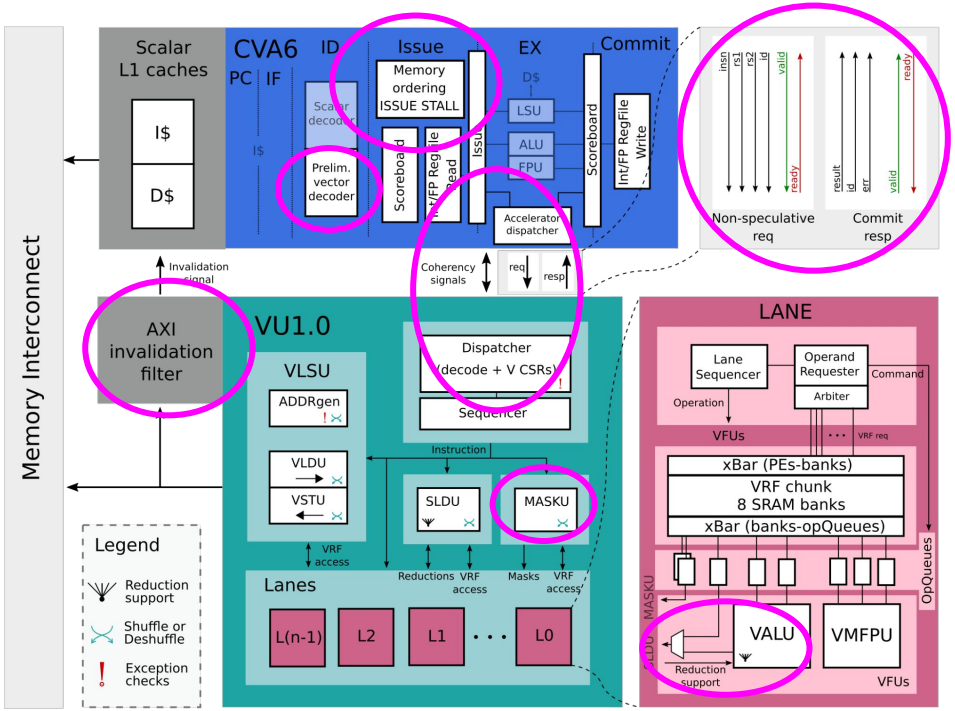
Conference submission

- Paper accepted!
- Add arch diagram:



Conference submission

- Paper accepted!
- Add arch diagram:



Main changes

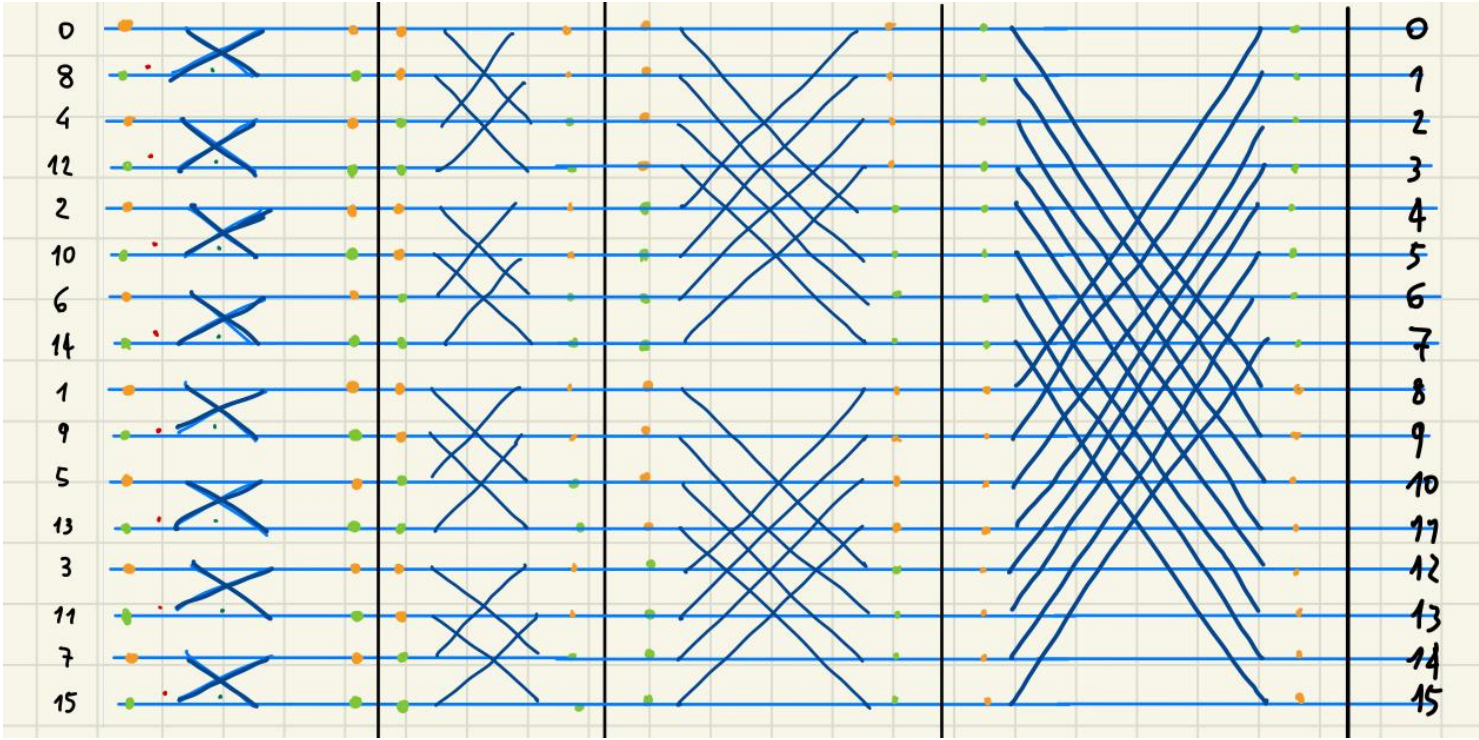
FFT

- Buffer all the samples in the VRF
  - Avoid unnecessary memory accesses
  - VRF_size = LANES * 4 KiB

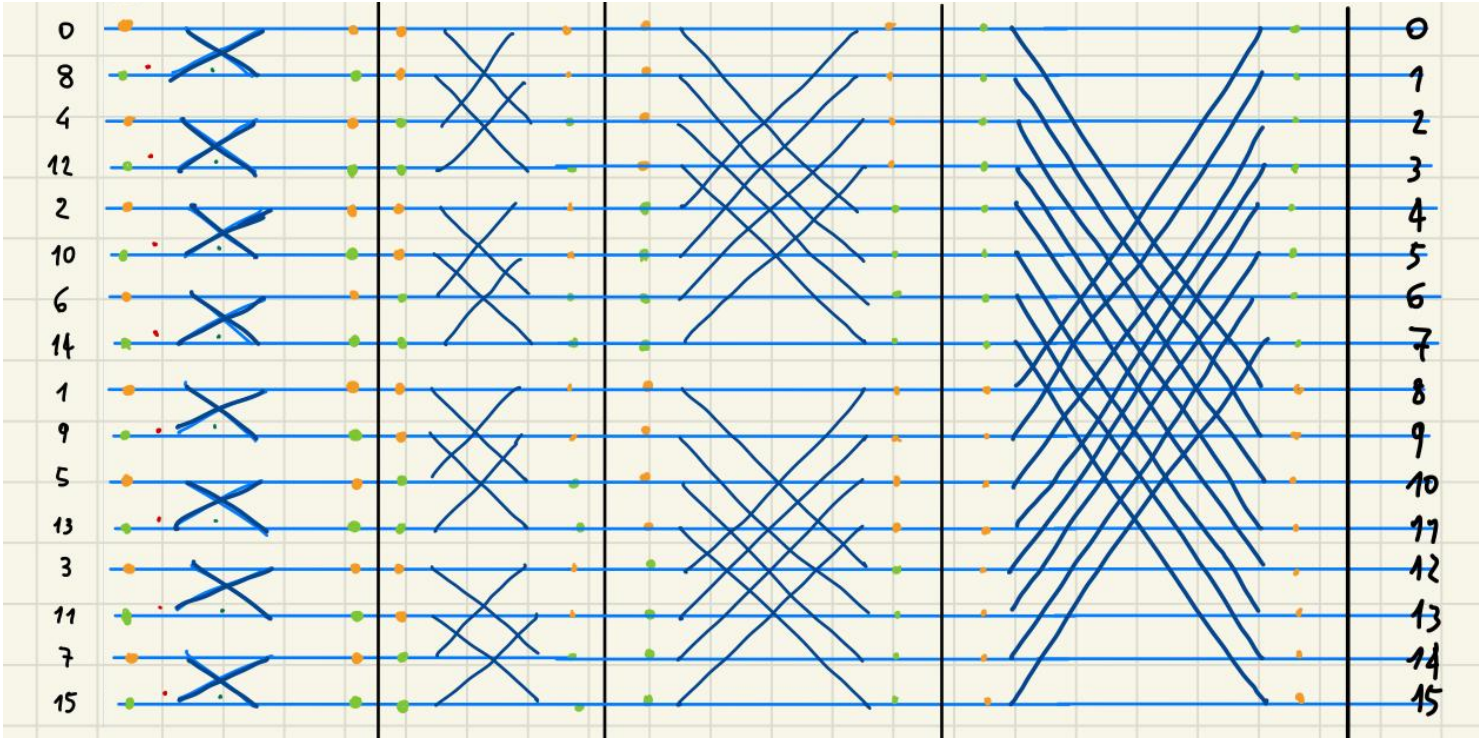
	2 Lanes	4 Lanes	8 Lanes	16 Lanes
Single vreg size (LMUL == 1)	256 B -> 1 KiB	512 -> 2 KiB	1 KiB -> 4 KiB	2 KiB -> 8 KiB
64-bit samples	64 -> 256	128 -> 512	256 -> 1 Ki	512 -> 2 Ki
32-bit samples	128 -> 512	256 -> 1 Ki	512 -> 2 Ki	1 Ki -> 4 Ki
16-bit samples	256 -> 1 Ki	512 -> 2 Ki	1 Ki -> 4 Ki	2 Ki -> 8 Ki
8-bit samples	512 -> 2 Ki	1 Ki -> 4 Ki	2 Ki -> 8 Ki	4 Ki -> 16 Ki

  - The max number of samples is twice what fits in a vreg
  - LMUL limited to 4 (8 would not allow any algorithm)

FFT - DIT



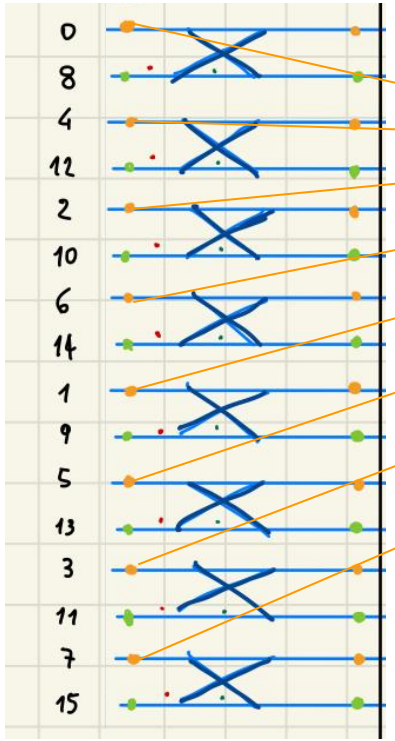
FFT - DIT



Upper wing vector

Lower wing vector

FFT - DIT

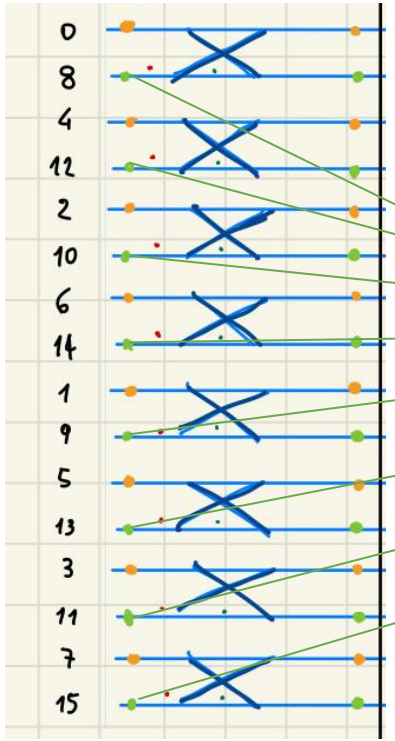


Upper wing vector

0	4	2	6	1	5	3	7

Lower wing vector

FFT - DIT



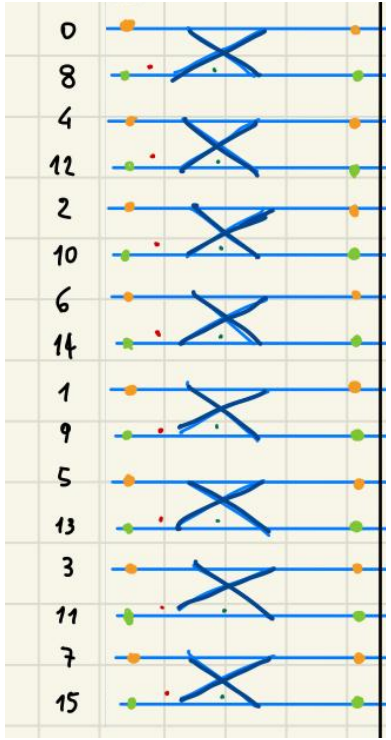
Upper wing vector

0	4	2	6	1	5	3	7

Lower wing vector

8	12	10	14	9	13	11	15

FFT - DIT



Upper wing vector

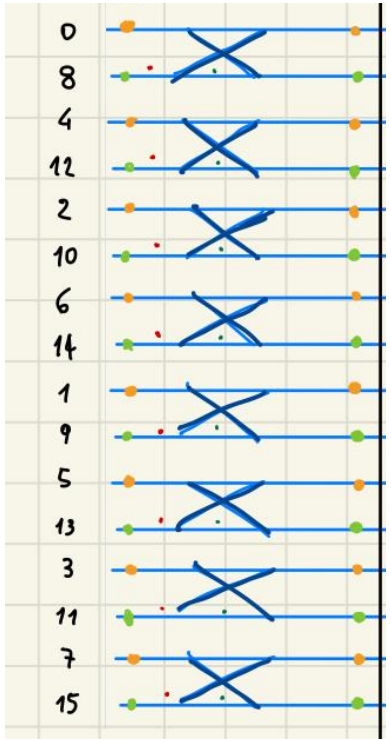
0	4	2	6	1	5	3	7

Lower wing vector

8	12	10	14	9	13	11	15

Every vector is actually two vectors: real and img parts
They are kept in two separated vector registers
(segmented memory operations!)

FFT - DIT



Upper wing vector

0	4	2	6	1	5	3	7

Lower wing vector

8	12	10	14	9	13	11	15

1) Gather?

Slow, but we need it only in the beginning

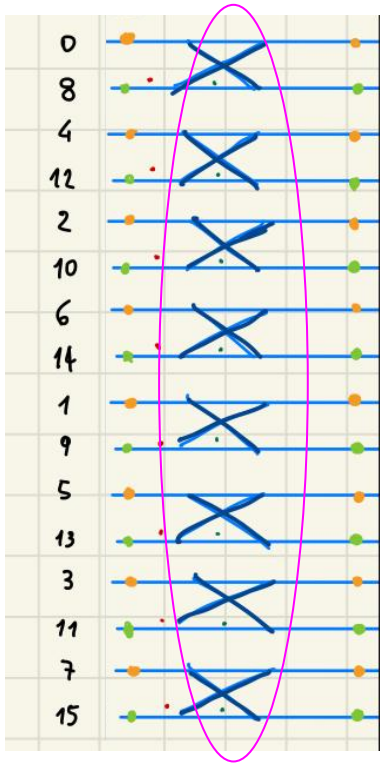
FFT

Twiddle factors

Tw0	Tw0	Tw0	Tw0	Tw0	Tw0	Tw0	Tw0

2) Load Twiddles with normal Load
(or scalar move + replication, only during this first step)

FFT

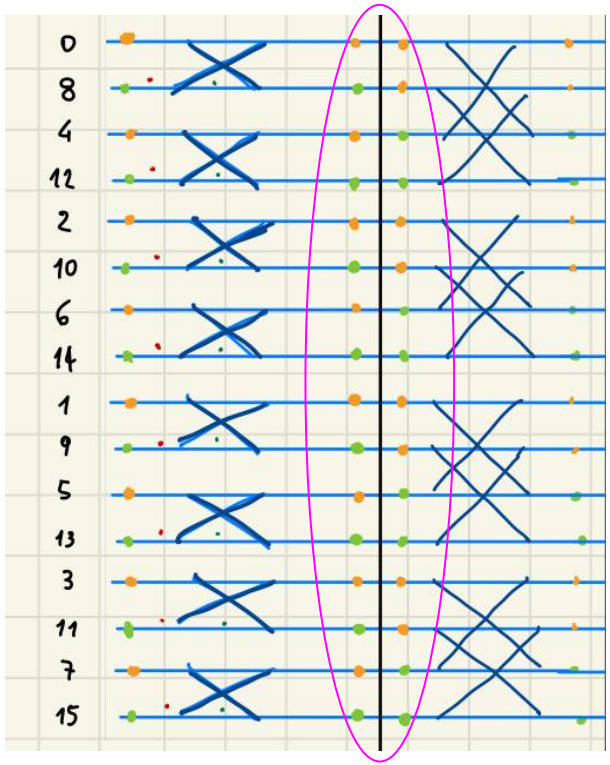


Twiddle factors

Tw0	Tw0	Tw0	Tw0	Tw0	Tw0	Tw0	Tw0

Butterfly operation:
One vector is added, the other is subtracted

FFT



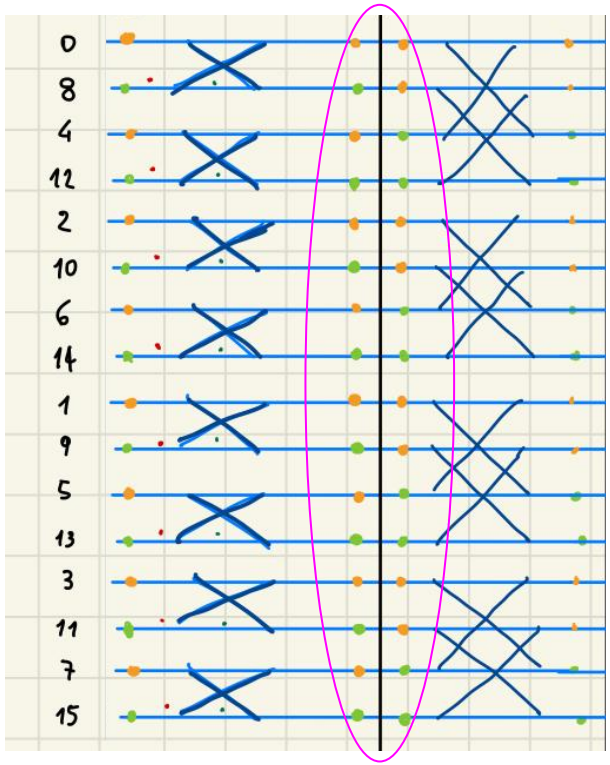
Upper wing vector

old	0	4	2	6	1	5	3	7
new	0	8	2	10	1	9	3	11
old	8	12	10	14	9	13	11	15

Lower wing vector

Permutation of the elements to get the "next butterfly" layout

FFT



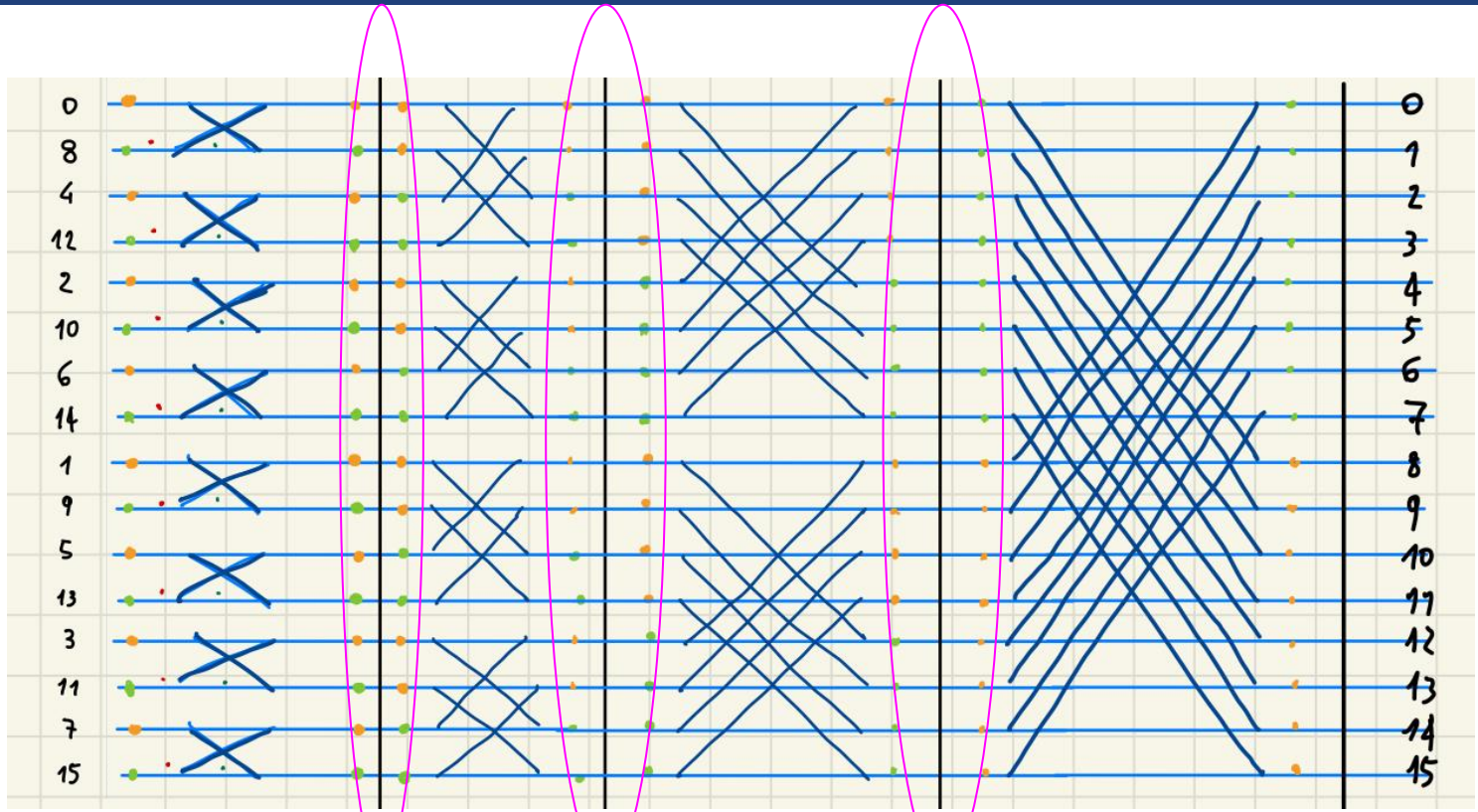
Upper wing vector

old	0	4	2	6	1	5	3	7
new	0	8	2	10	1	9	3	11
old	8	12	10	14	9	13	11	15

Masked slide!

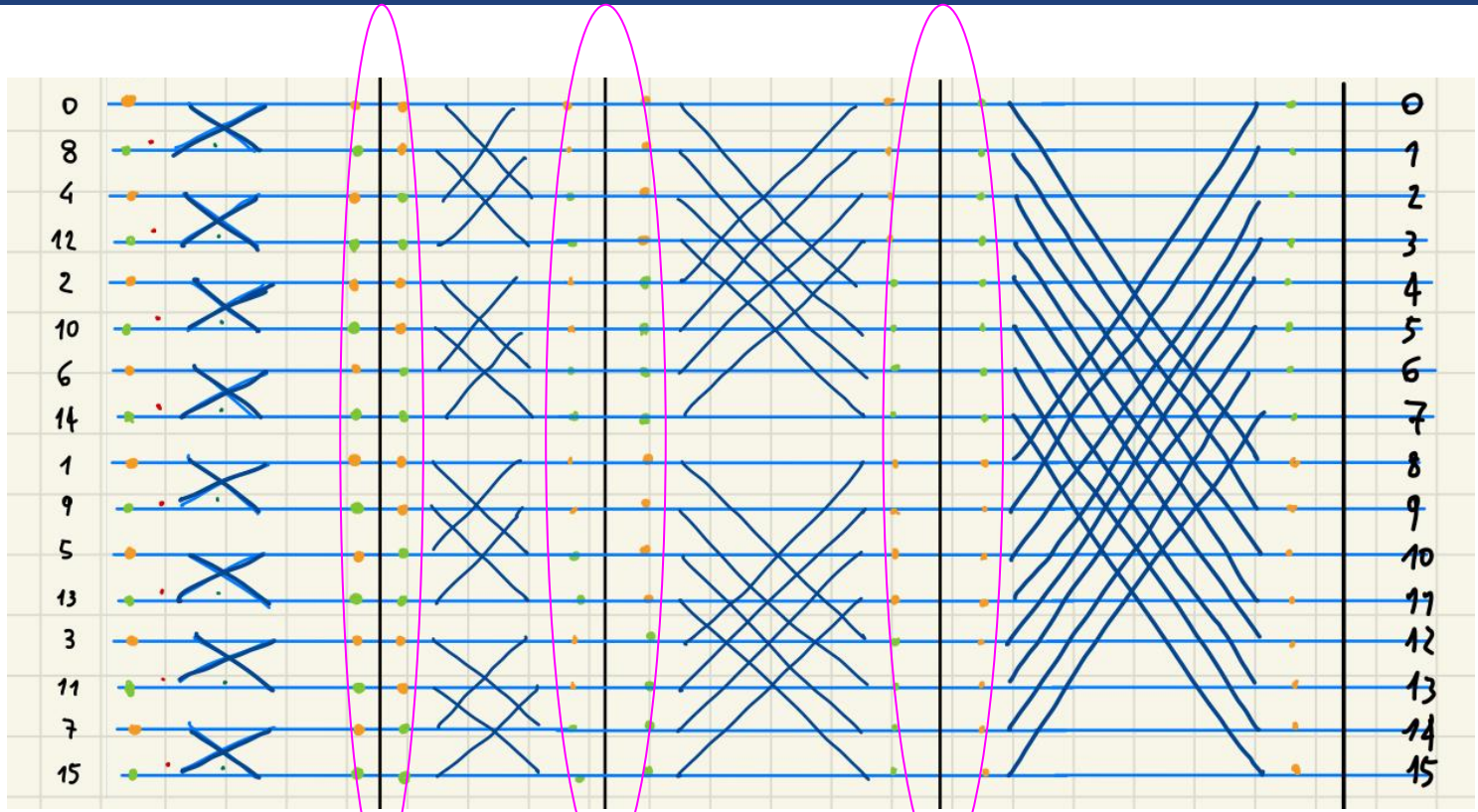
Lower wing vector

FFT



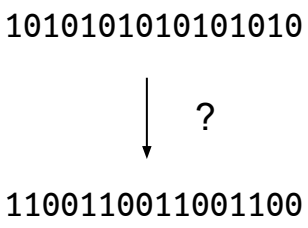
Mask layout for permutation is regular

FFT



1010101010101010 - 1100110011001100 - 1111000011110000 - 1111111110000000 - ….

FFT

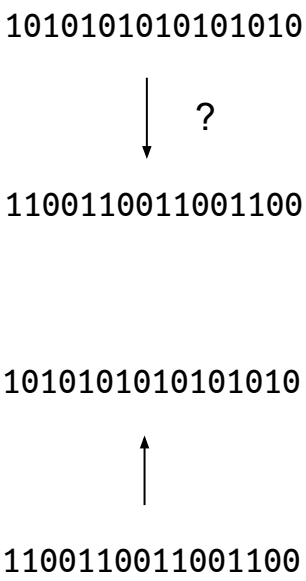


How to get the next mask?

Problem!

It's easier to go in the opposite direction!

FFT



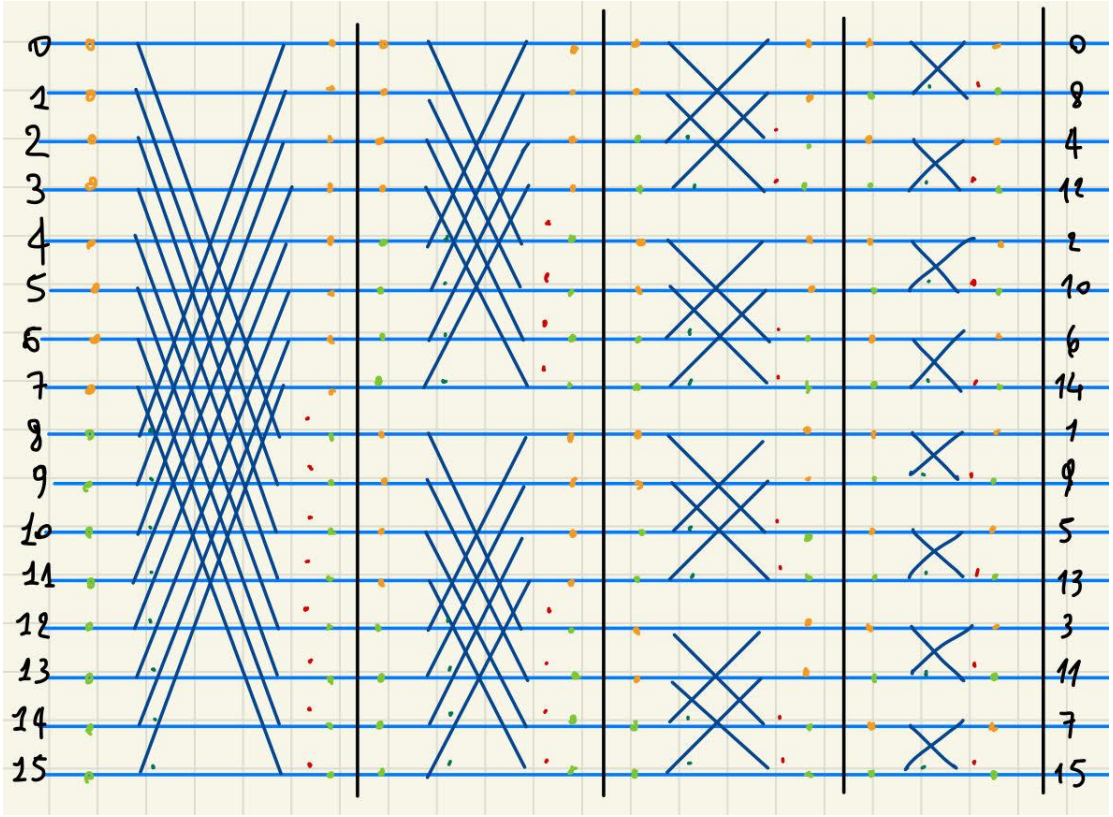
How to get the “next butterfly” mask?

Problem!

It’s easier to go in the opposite direction!

Slide + XOR!

FFT - DIF



FFT - DIF

- DIF FFT needs the opposite mask layout
- Ongoing: vector implementation of the DIF FFT
- Where are the bottlenecks?

- ISA extensions to speed up the execution
  - Automatic permutation
  - In vector permutation to avoid the initial idx mem op

FFT - DIF

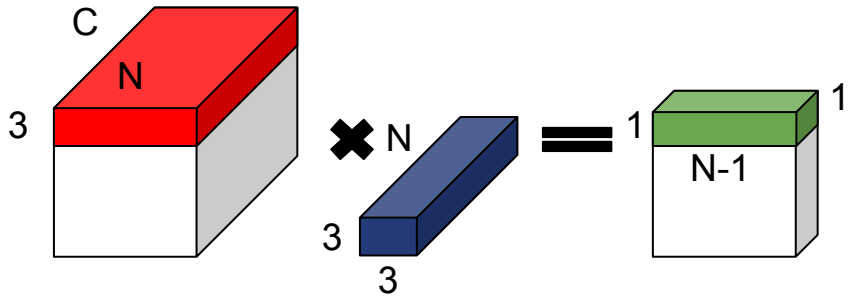
- First implementation:
  - Twiddle factors for each step are in memory
  - This can be optimized thanks to the intrinsic re-use

- ISA extension:
  - Keep the Twiddles in internal buffers and use them appropriately when needed
  - Keep the permutation patterns internally

FConv 3D 3x3 Kernel

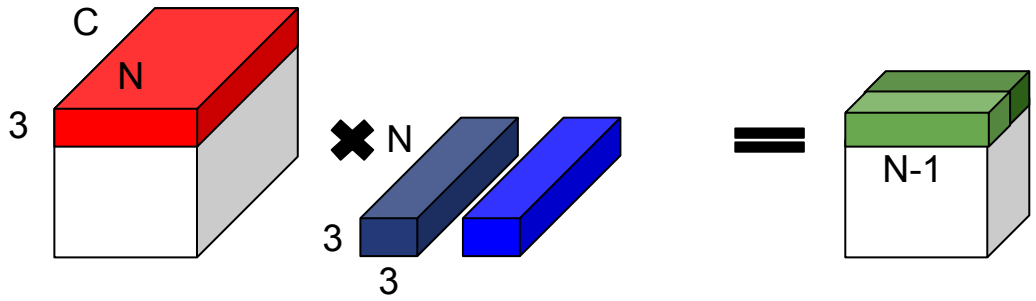
- Added a FConv 3x3 3D Kernel (previously only 7x7) with CHW data layout

- Difficult to reach same utilization as 7x7: each vector reutilization is 3 instead of 7



FConv 3D 3x3 Kernel

- This implementation underutilize the VRF:
  - 8 VReg for inputs (2x LMUL= 2) = 16
  - 4 VReg for outputs (2x LMUL = 2) = 8
  - There are still 4 possible Vreg to leverage
- 2nd Implementation: Compute two output channel per loop iteration



FConv 3D 3x3 Kernel

- Second implementation allow to use reuse twice the input register
- Utilization:
  - Compute 1 Channel per iteration = 70%
  - Compute 2 Channels per iteration = 82%
- We increased reuse but have to store two vectors per iteration

FConv 3D 3x3 Kernel

- Tried also a third implementation:
  - LMUL = 1 (double VRegs)
  - Compute 4 Channels per iteration
  - Utilization ~85% (diminishing return)

- There is probably some margin of improvement by better hiding latency of vmv instructions

Future works on Convolutional Kernel

- Kernel acting on CHW data layout are good for input layers
  - The deeper you go into a Network layers become thin and deep

- A student from university Poli Montreal is working on kernel for HWC data layout on integer format
  - They require reduction instruction (WIP for FP)

FP reductions

- WIP: code review
- Backend trials to check critical paths
- Benchmark for IPC and performance

OpenHW Group X-Interface

- **X-interface**: generic co-processor interface developed by OpenHW + ETHZ
- Integrated in CVA6 (among other processors)
- Shares several modifications of CVA6
  - Pre-decoder
  - Dispatcher
- Does **not** handle
  - Split load-store unit
  - X2X register transfers?