DAQ Intelligent

➔ Introduction – Dominant Design
➔ Timing & Synchronisation
➔ THINK: IA embarqué
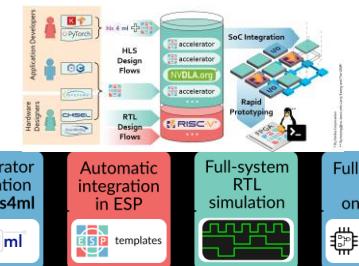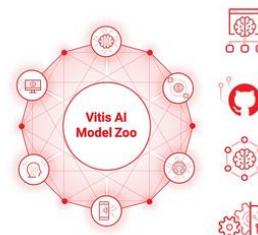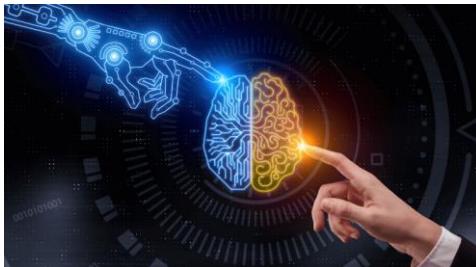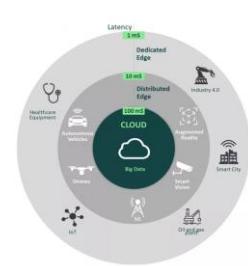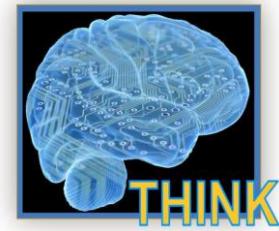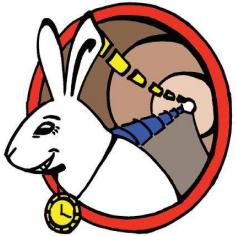
-TIMED:
Evolution du White Rabbit

-THINK phase 1:
IA embarqué
Résultats
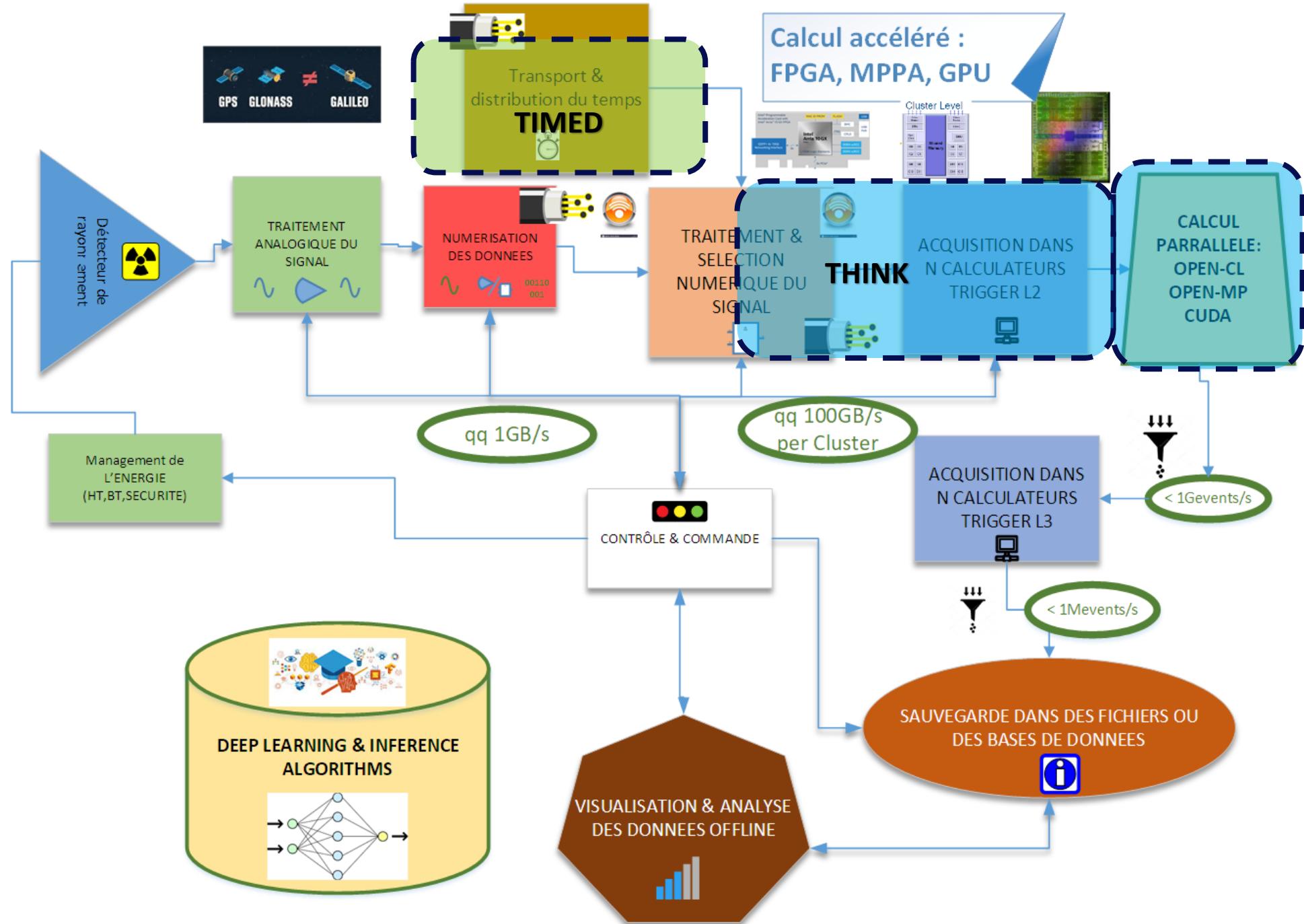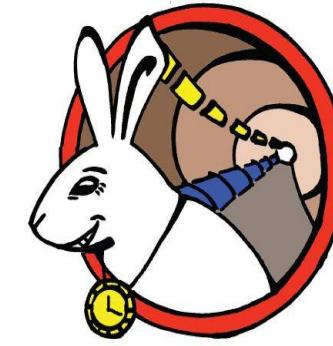
Synchronistion with WhiteRabbit

Antoine Back, Olivier Bourrion, Chafik Cheikali, **Daniel Charlet**, Eric Plaige, Paul-Eric Pottie, Monique Taurigna, Cédric Viou

- Synchronization of multi-nodes through an Ethernet network with no distance limitation

- Delivering a clock and time tagging with a precision and accuracy better than the current system

- Auto calibration system & automatic system adjustment (phase and delay)

- Accuracy & precision better of one order of magnitude compared to current system (GPSIPPP ~3ns)
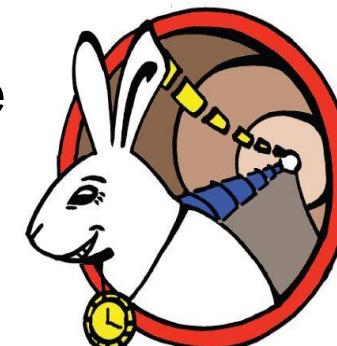
-

- **The issue :**
  - Clock phase stability
  - Real-timed communication system : tight time constraints
  - Long distances between nodes : long transmission delays
  - Numbers of nodes : several hundred

- Current system require thermostatically controlled fibers for clock distribution
- 2 networks : Synchro, data
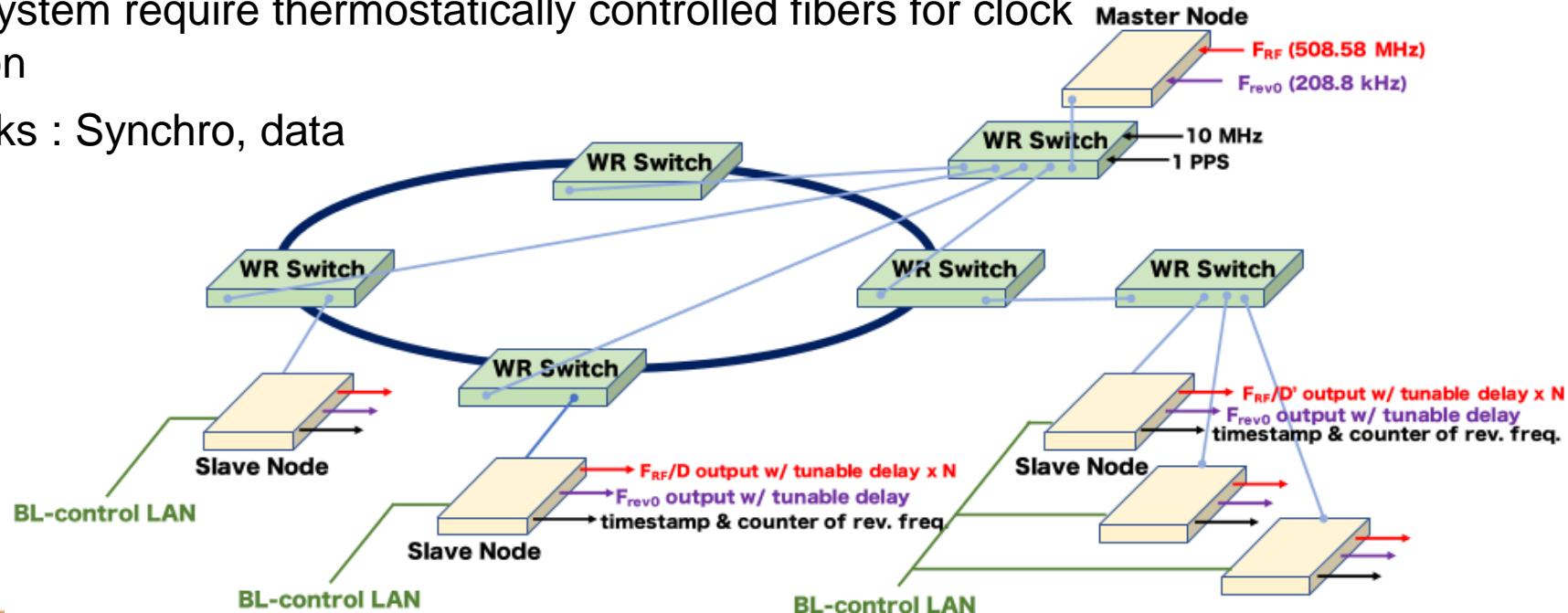
o **Real-time communication systems**: tight **time constraints**

o **Large distances** between nodes: long transmission delays

o **Dynamic changes** to the number of nodes: not easy

EISCAT: The most sophisticated Imaging Radar ever

With White Rabbit a farm of sensors extended over **several kms** is possible
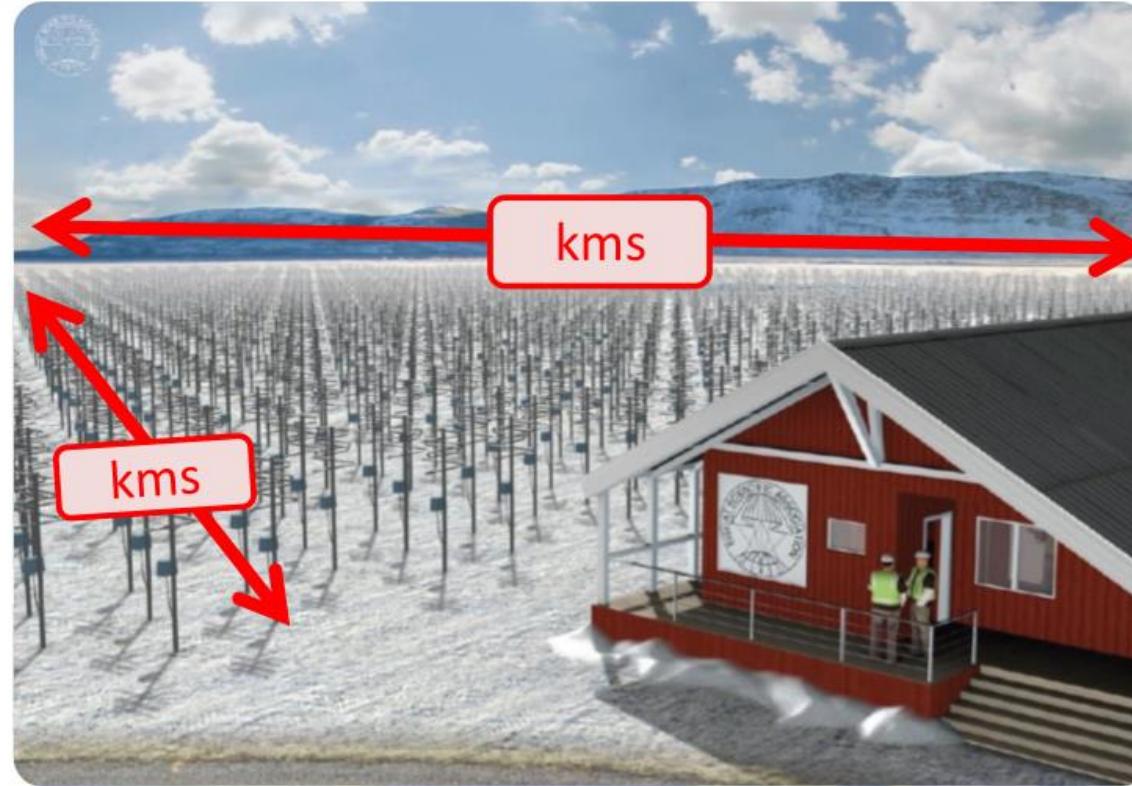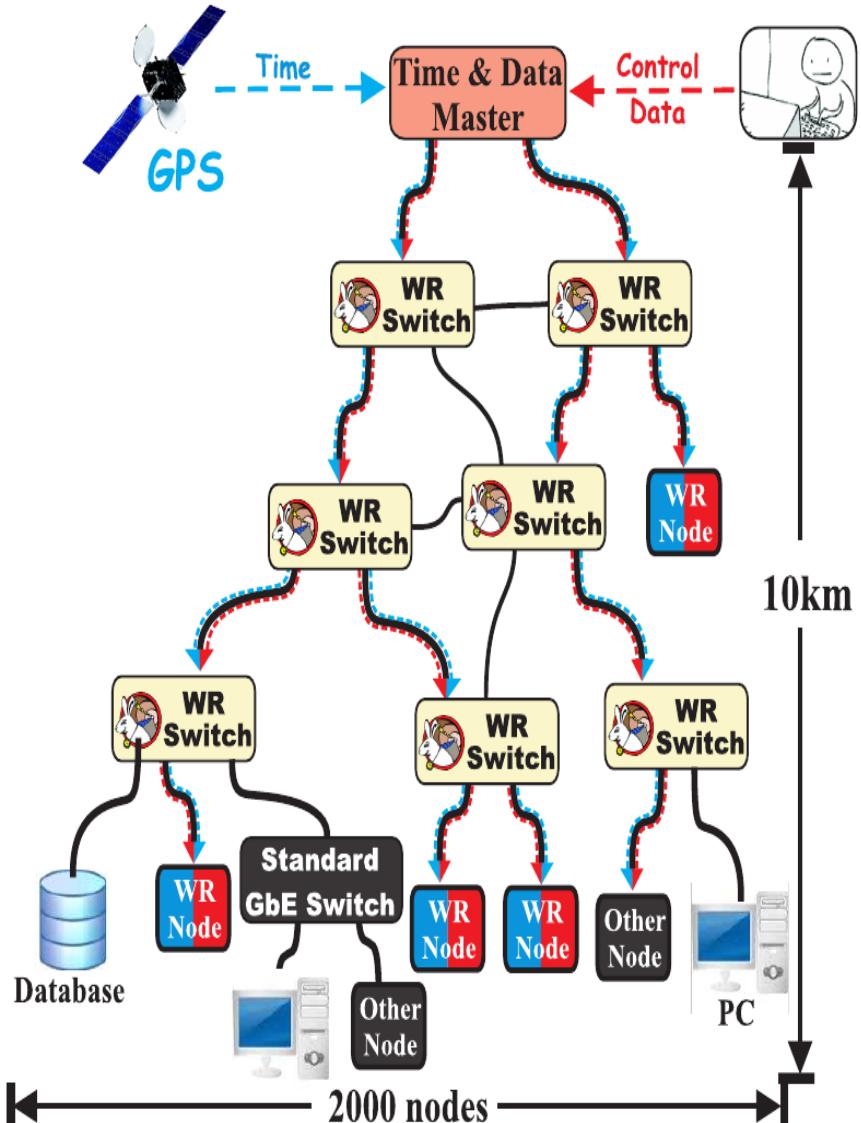
- An extension of Ethernet which provides :
  - Synchronous mode (Syn-E) – common clock for physical layer in entire network, allowing for precise time and frequency transfer
  - Deterministic routing latency – a guarantee that packet transmission delay between two stations will never exceed a certain boundary.

- Technology overview
  - Precision Time Protocole (IEEE1588)
  - Synchronous Ethernet
  - DDMTD Phase tracking (Digital Dual Mixer Domain) …

- Off the shelf WR system performance
  - Accuracy : 200ps
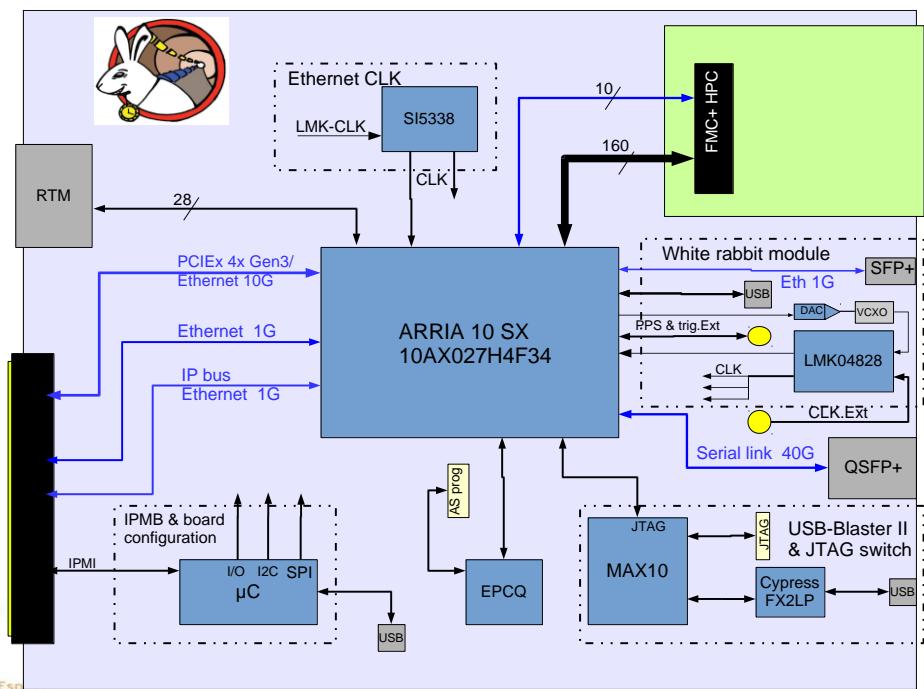
- High performance WR low jitter
  - Expertise from SYRTE for clock & qualification
- Design & realization by IJCLAB
- Firmware by Nancay Observatory

- TCA 4.0 standard, double width full-size
- Stand-alone mode
- VITA57.1 (FMC slot)
- 160 single-ended I/Os (80 LVDS) and/or up to 10 serial transceivers in a 40 x 10 configuration
- Full WhiteRabbit compliant enhance node

- Pulse Per Second 2 IDROGEN board
- 25m & 125m of optic fiber
  - ~50ps of dispersion of the PPS with calibration
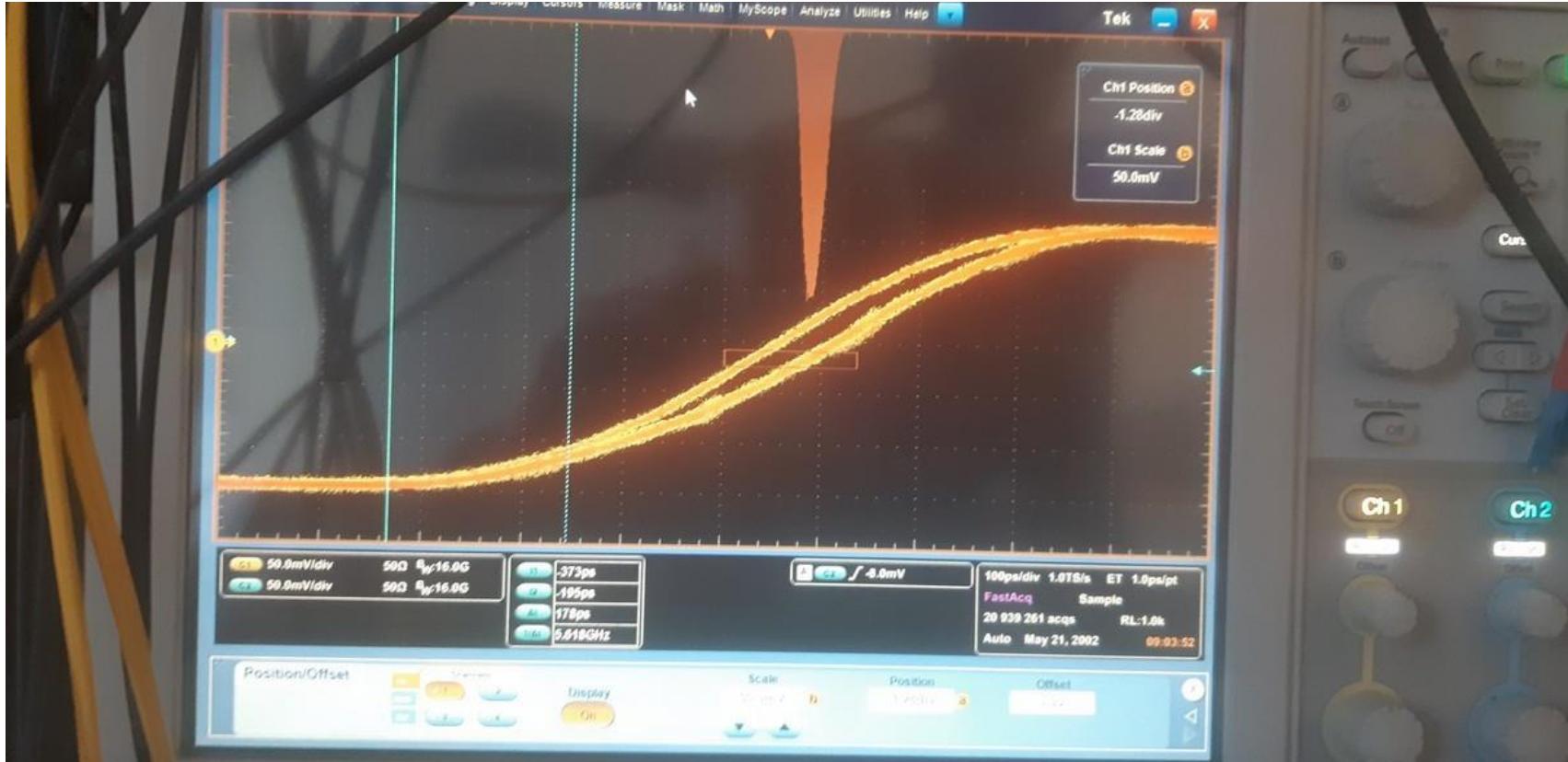
R & T TIMED

- **D.Etasse**

- **D.Toures**

- **F.Druillole**

- **D.Charlet**
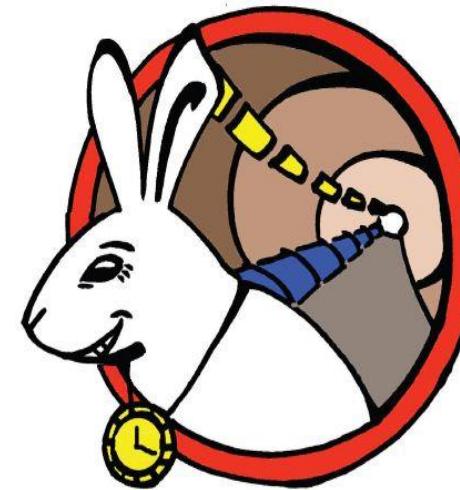
- WR integration in a crate (µTCA , ..)

- Improvement of the WR performance

- Hardware simplification

- Frequency distribution
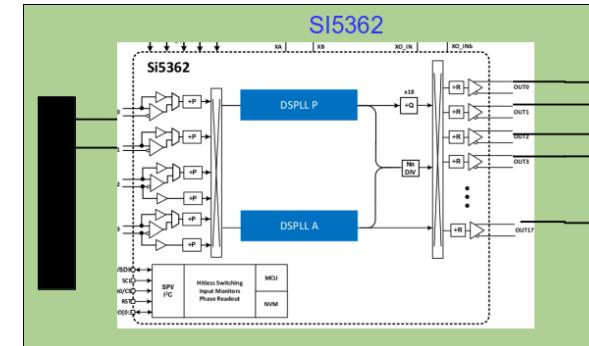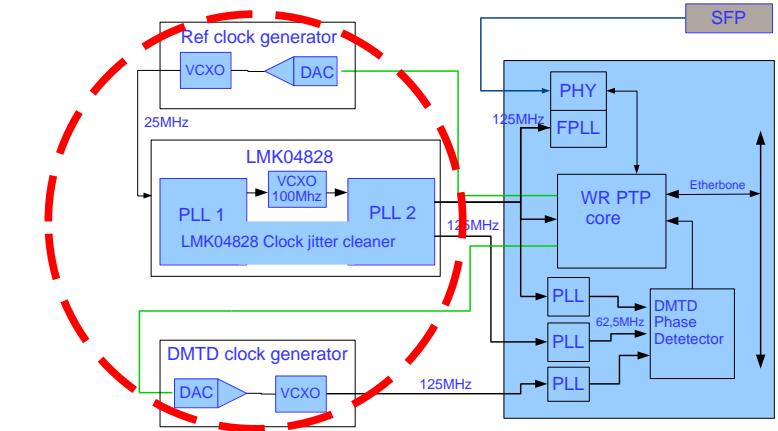
- The White Rabbit is a very powerful protocol
- It's real technological breakthrough
- But not very simple to uses and to implemented
- To go below the ns it's mandatory to began expert.
  - The more precision increases the more experience is required
- The R & T timed will offer simpler use solution for current accuracy
- A new promising development expected to achieve fento second of accuracy

- More technical details will be deliver during DAQ session
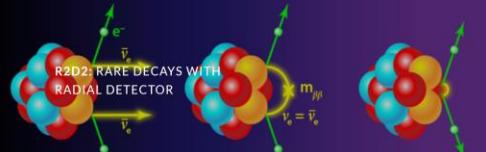
R&T IN2P3 THINK
Testing Hardware Instantiations of Neural Kernels

Bilan: AMD-Xilinx et IA embarqué

Jean-Pierre Cachemiche, Monnier Emmanuel, George Aad, Thomas Calvet, Arthur Ducheix, Etienne Fortin, **CPPM**,
Frédéric Magniette, **LLR**
Joana Fronteras-Pons, **IRFU/AIM**
Frédéric Druillole, Abdelkader Rebii, Raphael Bouet, **LP2IB**
David Etasse, **LPC**
Vladimir Gligorov, Le Dortz Olivier, **LPNHE**
Fatih Bellachia, Lafrasse Sylvain, **LAPP**
Claude Girerd, **LP2IL**

2020 - 2023

Challenges in the field

LHCb – 2032 ~2000 Exabytes/year
ATLAS+CMS 2027 ~ 260 Exabytes/year
Square Kilometers Array – 2030 ~ 30000 EB/year

2021 global Ethernet Dataflow ~2800 EB/year

DataStream before storage
LHCb – 2032 ~500TB/s
ATLAS+CMS 2027 ~ 20-40 TB/s

Forecast cost of storing data to disk (Annual)
LHCb – 2032 ~2,5 Billions of €
ATLAS+CMS 2027 ~ 325 Millions of €

Challenge: Real-time data reduction to avoid disk storage (very expensive):

➔ **Embedded algorithms in decision nodes**

➔ **Optimize processing (classifications, Prediction, Selection)**

➔ **Use a mixed GPU, MPPA, FPGA**
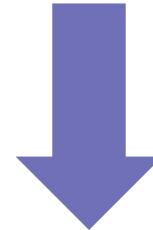
- **Use powerfull hardware component to compute ML Model**
- **And deploy them in ours instruments**

WHAT WE COULD DO WITH ML

DEEP NEURAL NETWORK
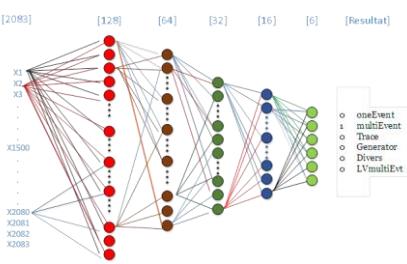
CONVOLUTIONAL NEURAL NETWORK

Conv + ReLU   Dropout   Pooling   Upsampling   Conv + Sigmoid
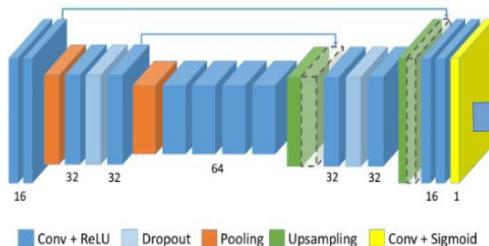
RECURRENT NEURAL NETWORK
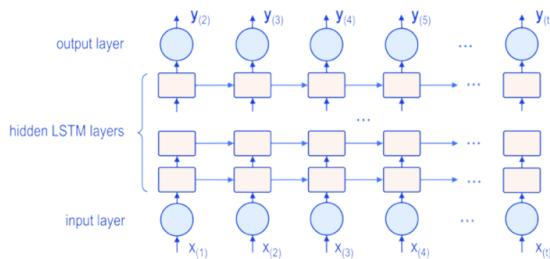
DECISION TREE
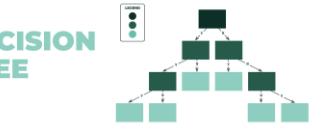
RANDOM FOREST
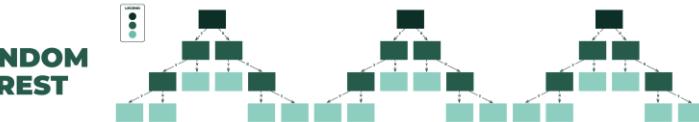
Off-line

Signal generation

Design Optimisation

On-line

Instrument Optimization
  Signal recognition
  Pile-Up recovery
  Signal deconvolution

Selection/ Classification/ Decision
  Data selection
  Data parameters prediction
  Denoizing

Data Compression
  Reduced data format

L1

L2

L3

CPU ↔ Additional computing architecture
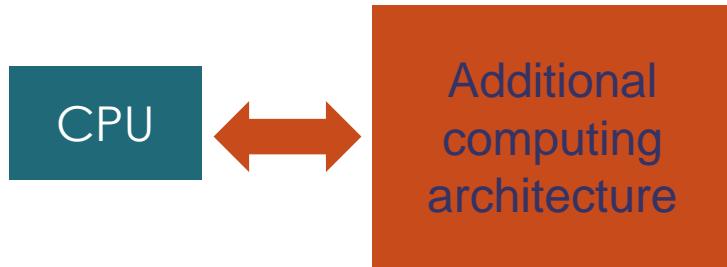
Edge AI Responsive
(~100μs to several second)

-- Based on software programmation

MMPA
GPU
FPGA – SOM-
PU designed for AI(TPU, KPU…)

1 Pilot current industrial developments

CPU ↔ Calculation implemented in a dedicated hardware solution
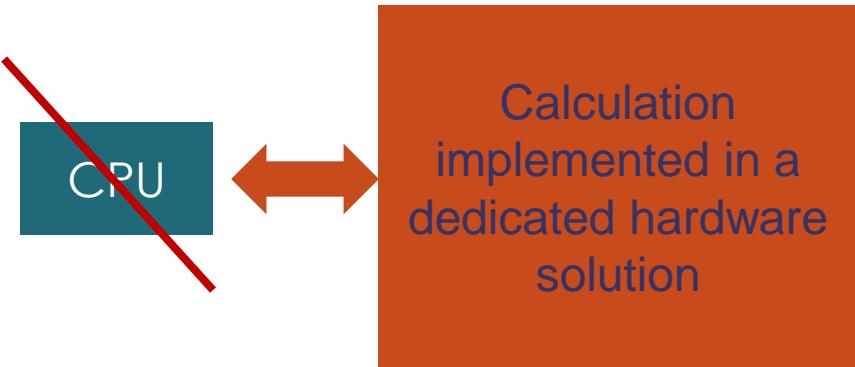
Spatial Accelerators = Fully Firmware
(~10ns to several 10μs)

-- based on hardware functions dedicated to calculations without software (matrix calculation)

ASIC
Neuromorphic Circuit
FPGA

2 In Progress

CPUs

GPUs

FPGAs

NMC
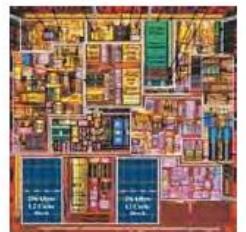
MPPA
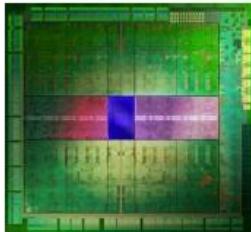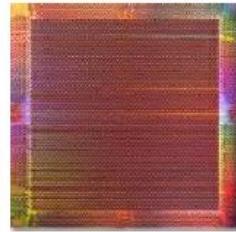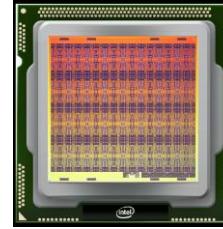
nVidia
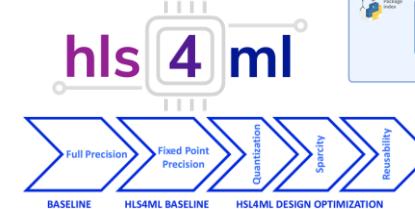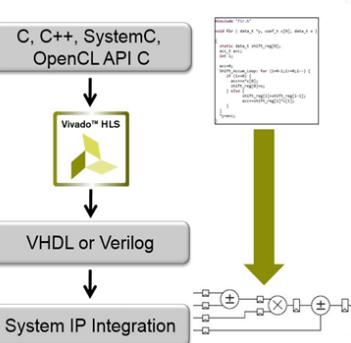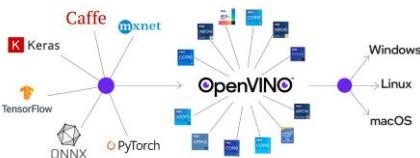
DE10-Agilex

ZCU102, ZCU104

https://think.in2p3.fr/

VCK190

MPPA® platform

Video Sources → Output /Display

INPUT DATA
- Camera
- Images
- Lidar

RESULTS
- Classification
- Segmentation

C, C++, SystemC, OpenCL API C

Vivado™ HLS

VHDL or Verilog

System IP Integration

INPUTS

NN : model with a huge number of parameters

Hardware:
Create a model which fit inside FPGA
FPGA ressources Management

GOALS

Conversion computer model inside hardware with low latency and minimum of resources

To optimize the network in term of performances (precision) and hardware use ratio.

Methods

Pruning:
To reduce the number of parameters

Quantification:
To optimize parameters size (number of bit) in function of performances

Methodology of design

```
┌──────────────┐      ┌──────────────┐      ┌──────────────────┐      ┌──────────────────────┐      ┌──────────────┐
│ INPUT DATA   │ ───▶ │ DATA ANALYSIS│ ───▶ │ MACHINE LEARNING │ ───▶ │ LEARNING PHASE:      │      │   ML Tools   │
│ LIBRARY      │      │ (Preparatory │      │ – MODEL CREATION –│     │ TensorFlow-Keras,    │      │              │
│              │      │  phase)      │      │                  │      │ PyTorch              │      │              │
└──────────────┘      └──────────────┘      └──────────────────┘      └──────────────────────┘      └──────────────┘
```

- INPUT DATA LIBRARY
- DATA ANALYSIS (Preparatory phase)
- MACHINE LEARNING – MODEL CREATION –
- LEARNING PHASE: TensorFlow-Keras, PyTorch
- **ML Tools**

- WEIGHT
- MODEL

- VALIDATION WITH R2D2 DETECTOR
- *Example*
- OWEN DAQ SYSTEM
- WEIGHT FORMAT CONVERSION
- **PRUNING**
- **QKERAS TF/PyTorch pruning tools**

- SYSTEM INTEGRATION
- WEIGHT COMPRESSION

- SYNTHESIS & VALIDATION in real environment
- SCORING WITH DATA SET
- FPGA CODE GENERATION
- **HLS4ML FiNN/Brevitas Conifer HLS …**

- VHDL description
- **High Level Synthesis**

Nombre de paramètres en fonction du réseau étudié

- **Zynq + HLS4ML**: io_parallel / io_stream /Reusefactor / Resource / Latency

	ARTY-Z7 CH1	ARTY Z7 CH3	ARTY Z7 CH3 Optimisé	ZCU102 CH3	CH4	ZCU102 CH4 Qbit<16,2>	ZCU102 CH5	ZCU102 CH7 Optim Ressource	ZCU102 CH7 Optim Latence	ARTY Z7 CH7 Optim HLS Stream
Nombre de cellules	16	25801	25801	25801	658951	658951	156951	156951	156951	156951
Horloge de reference	4,166ns	9,408ns	9,408ns	4,396ns		4,369ns	4,369ns	4,369ns	4,028ns	9,410ns
Temps de latence	70ns	19,804us	19,804us	2,510us		5,510us	5,510us	10,010us	10,015us	141us
BRAM	0%	41%	8%	6%		36%	18%	9%	15%	72%
DSP48E	21%	115%	11%	10%		59%	59%	12%	24%	6%
FF	2%	85%	28%	10%		28%	22%	32%	53%	167%
LUT	1%	280%	68%	46%		103%	113%	81%	104%	423%
URAM	0%	0%	0%	0%		0%	0%	0%	0%	0%

ARTY Z7
ZCU104

Tensorflow/Keras
+ HLS4ML
VITIS HLS

depending on VITIS-HLS #pragma
The way HLS handles vector/matrix before DSP

Unroll

Iteration 1   Stage 1   Stage 2   Stage 3
Iteration 2   Stage 1   Stage 2   Stage 3
Iteration 3   Stage 1   Stage 2   Stage 3
Time
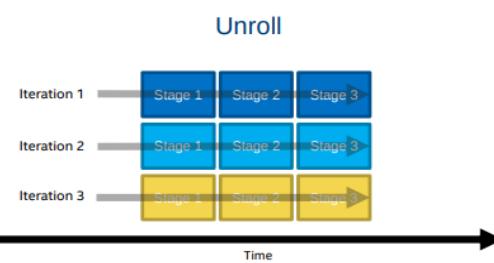
Pipeline

Question of HLS optimisation
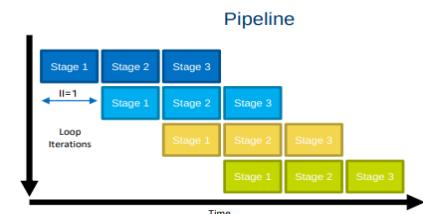
What if directly written in VHDL ?

DE10-Agilex

2x8x1

-niter	Nombre d'entrées à traiter
-nireq	Nombre d'exécutions à réaliser en parallèle (sous-multiple de niter)

Mode	
Just-In-Time	Compilation au vol du modèle en ligne
Ahead-In-Time	Compilation du modèle avant son execution

Tableau 1: Exécution du réseau CH1 avec **sigmoid** et inférence **JIT**

-api	-niter	-nireq	FPS	Précision
async	12	6	8053	100 %
async	12	4	10535	100 %
async	12	3	7940	100 %
async	12	2	7403	100 %
async	12	1	3482	100 %
async	12	X	10504	100 %

Tableau 2: Exécution du réseau CH1 avec **sigmoid** et inférence **AOT**

| -api	-niter	-nireq	FPS	Précision
async	12	6	11128	100 %
async	12	4	10956	100 %
async	12	3	8839	100 %
async	12	2	6809	100 %
async	12	1	4216	100 %
async	12	X	8216	100 %

Tableau 3: Exécution du réseau CH1 avec **softmax** et inférence **JIT**

-api	-niter	-nireq	FPS	Précision
async	12	6	14046	100 %
async	12	4	13982	100 %
async	12	3	11505	100 %
async	12	2	9752	100 %
async	12	1	6484	100 %
sync	12	X	13056	100 %

Tableau 4: Exécution du réseau CH1/ avec **softmax** et inférence **AOT**

-api	-niter	-nireq	FPS	Précision
async	12	6	6650	100 %
async	12	4	7252	100 %
async	12	3	6477	100 %
async	12	2	6073	100 %
async	12	1	4391	100 %
sync	12	X	8704	100 %

Les performances en mode **JIT** semblent s'être améliorées tandis que les performances en mode **AOT** semblent s'être détériorées

Result with Intel FPGA: Edge computing

CH3
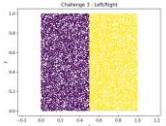
2x200x100x50x1

-api	-niter	-nireq	FPS JIT	FPS AOT
async	12	6	18489	7245
async	12	4	15003	7503
async	12	3	14003	6525
async	12	2	13514	5747
async	12	1	6178	4198
sync	12	X	16316	10294

CH4
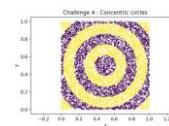
-api	-niter	-nireq	FPS JIT	FPS AOT
async	12	6	14053	8400
async	12	4	11848	7643
async	12	3	10987	7576
async	12	2	8225	5925
async	12	1	6178	5421
sync	12	X	9224	7711

Not expected: JIT more performant than AOT !

CH5
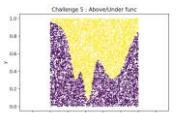
-api	-niter	-nireq	FPS JIT	FPS AOT
async	12	6	18662	7498
async	12	4	13506	6903
async	12	3	13450	6281
async	12	2	11075	5852
async	12	1	6317	4116
sync	12	X	14023	9596

CH6
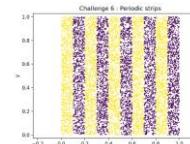
-api	-niter	-nireq	FPS JIT	FPS AOT
async	12	6	17606	8619
async	12	4	13714	6364
async	12	3	13350	6944
async	12	2	12521	6019
async	12	1	6540	3741
sync	12	X	14456	8036

Algorithms

Functional Iterations

C/C++ ← HLS Libraries (planned)

Architectural Iterations

Intel® HLS Compiler

Qsys ← Intel® FPGA IP

Intel® Quartus® Prime
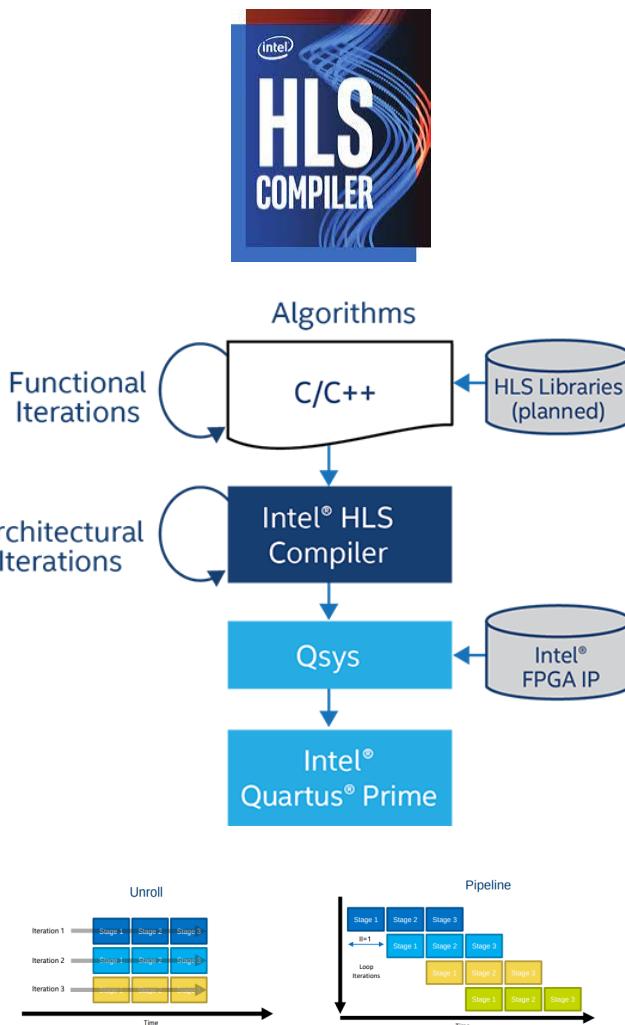
Unroll

Pipeline

	ALUTs	FFs	RAMs	MLABs	DSPs
Ch1 vanilla	602 (0%)	547 (0%)	4 (0%)	2 (0%)	1.5 (0%)
Ch1 pipeline	610 (0%)	624 (0%)	4 (0%)	5 (0%)	1.5 (0%)
Ch1 unroll	515 (0%)	245 (0%)	4 (0%)	1 (0%)	0 (0%)
Ch1 u+p	515 (0%)	245 (0%)	4 (0%)	1 (0%)	0 (0%)

	Latence	Débit	Nbr d'instances	Débit suffisant ?
Ch1 vanilla	44	46	678	Oui
Ch1 pipeline	43	9	678	Oui
Ch1 unroll	17	1	678 / 250	Non
Ch1 u+p	17	1	678 / 250	Non

	ALUTs	FFs	RAMs	MLABs	DSPs
Ch3 vanilla	1 408 (0%)	1 809 (0%)	30 (1%)	12 (0%)	2.5 (0%)
Ch3 pipeline	2 093 (0%)	4 460 (0%)	32 (1%)	48 (0%)	2.5 (0%)
Ch3 unroll	118 041 (14%)	36 737 (2%)	5 (0%)	37 (0%)	0 (0%)
Ch3 u+p	27 524 (3%)	42 546 (2%)	1 855 (68%)	298 (1%)	75 (5%)

	Latence	Débit	Nbr d'instance	Débit suffisant ?
Ch3 vanilla	25 575	25 578	~ 100	Oui
Ch3 pipeline	25 320	20 020	~ 100	Oui
Ch3 unroll	50	1	7	Non
Ch3 u+p	232	101	1	Non

	ALUTs	FFs	RAMs	MLABs	DSPs
Ch4 vanilla	4 442 (0%)	6 415 (0%)	355 (1%)	20 (0%)	3.5 (0%)
Ch4 pipeline	5 555 (0%)	10 899 (0%)	362 (1%)	113 (0%)	3.5 (0%)
Ch4 unroll	Problème d'implémentation				
Ch4 u+p					

	Latence	Débit	Nbr d'instance	Débit suffisant ?
Ch4 vanilla	657 926	657 928	7	Oui
Ch4 pipeline	656 168	500 002	7	Oui
Ch4 unroll	Problème d'implémentation			
Ch4 u+p				

Akida™ Development Kit - Shuttle PC

Akida™ Development Kit - Raspberry Pi

Akida™ PCIe Board
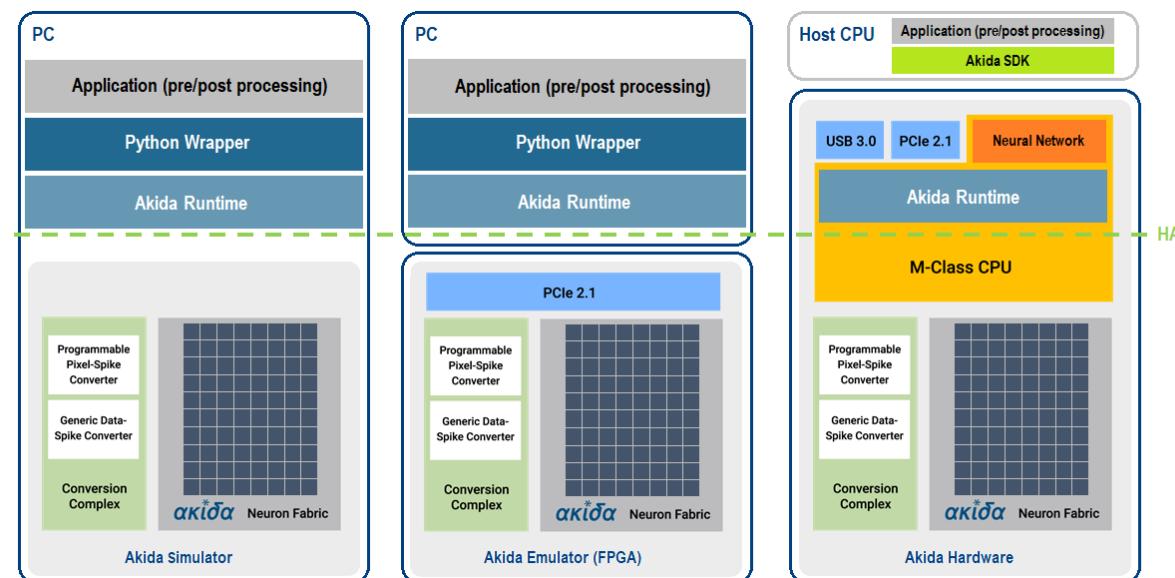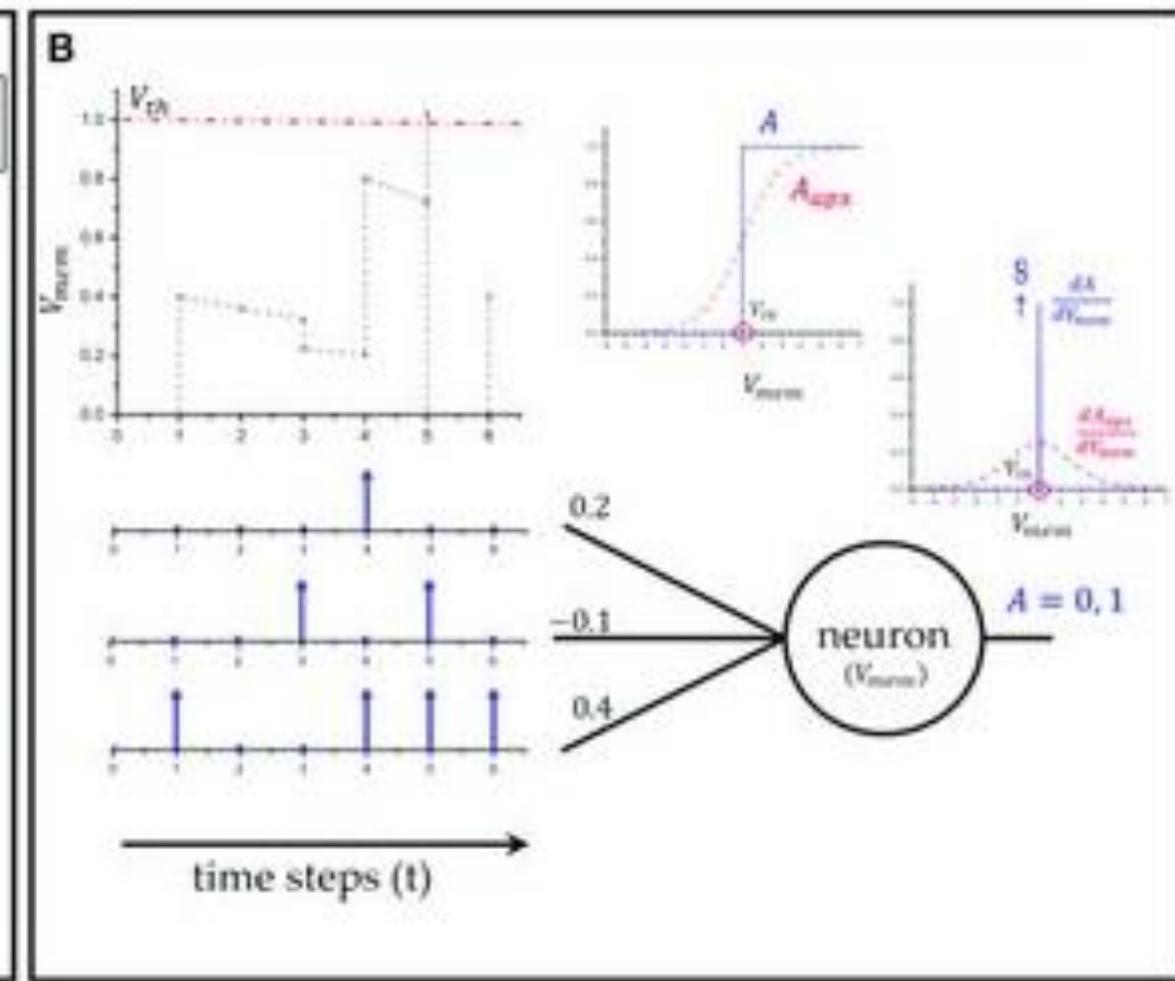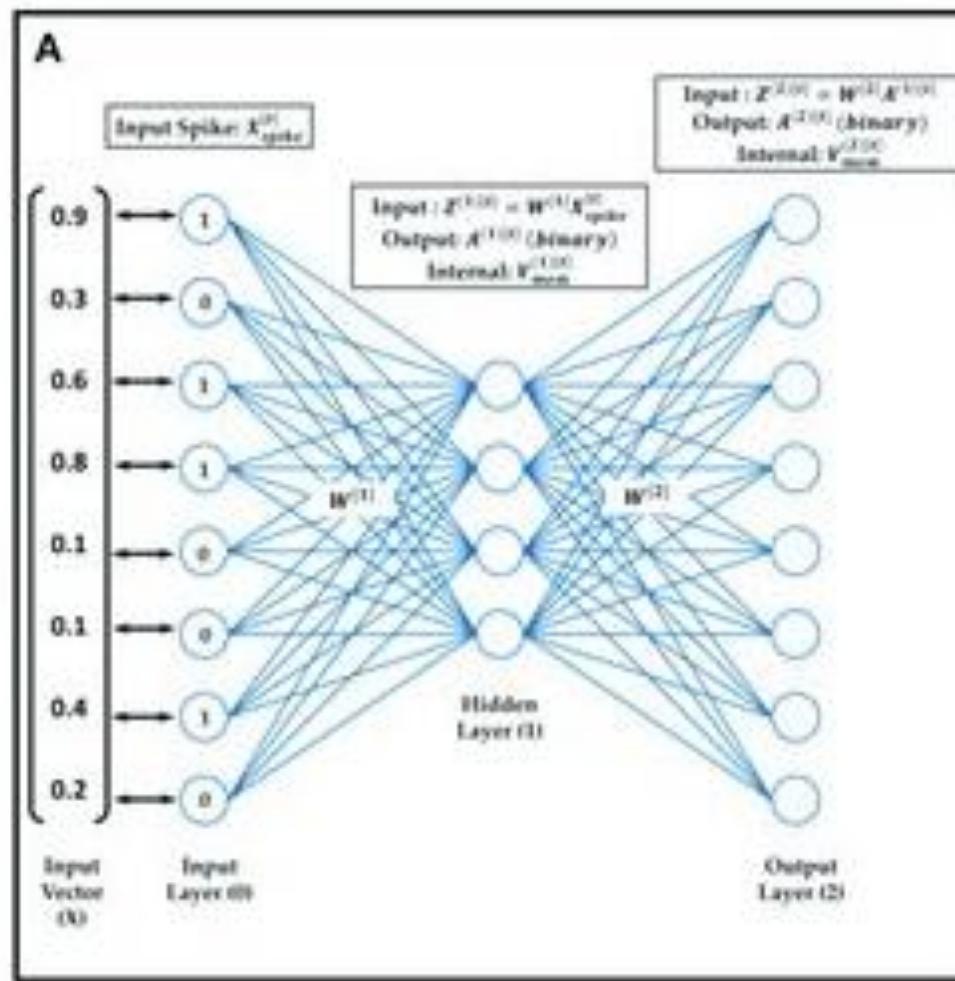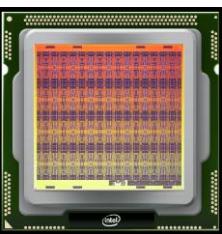
Spikes Neural Network: Akida™ PCIe Board.

➡ Spiking Neural Network

* ARM Cortex-M4 32-bit @ 300MHz (Subblock of Akida)
* RAM: 256M x 16 bytes LPDDR4 SDRAM @ 2400MT/s
* FLASH: Quad SPI 128Mb NOR @ 12.5MHz
* Onboard Akida core current monitor
* Operates under Linux on arch or x86-64 architectures

* GPIO: 2 LED's
* Interfaces: 5GT/s PCI Express 2.0 x1-lane
* 40mm x 76mm x 5.3mm (exc. PCIe rear panel bracket)
* Weight: 15g (exc. PCIe rear panel bracket)
* Small form factor PCIe 1.6"x3 – rear panel PC bracket included

➡ NO NEED EMBEDDED CPU OR DRAM ➜ Simple architecture , less power consumption

DRIVERS : Communication via PCIe Gen2 with HOTFIX
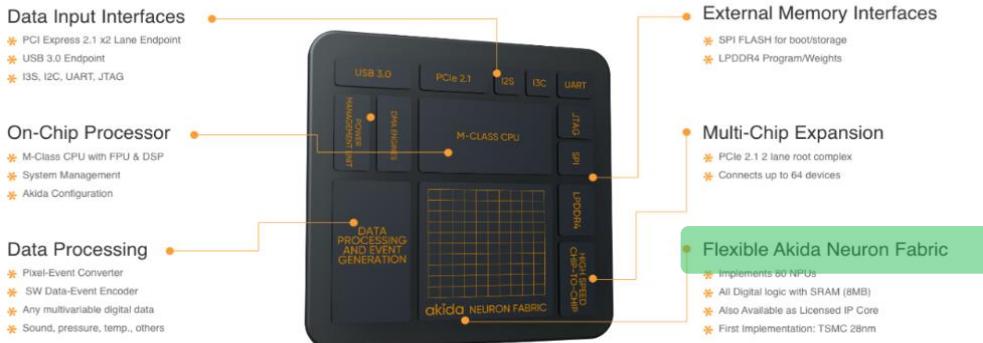
80 NPUs (Neural Processor Units)

- Akida python package
- Akida Model Zoo
- CNN2SNN tool (DNN, CNN)

• Layer: InputConvolutional
• Layer: Convolutional
• Layer: FullyConnected
• Layer: Activation ReLU

```python
def generation_model():
    model = Sequential()
    model.add(Dense(8, input_shape=(2,),
name="FC1"))
    model.add(ReLU(name="relu"))
    model.add(Dense(1, name="FC2"))
    model.add(Activation("sigmoid",
name="sigmoid"))
    model = cnn2snn.quantize(model,
weight_quantization=4,
 activ_quantization=4)
    return model
```
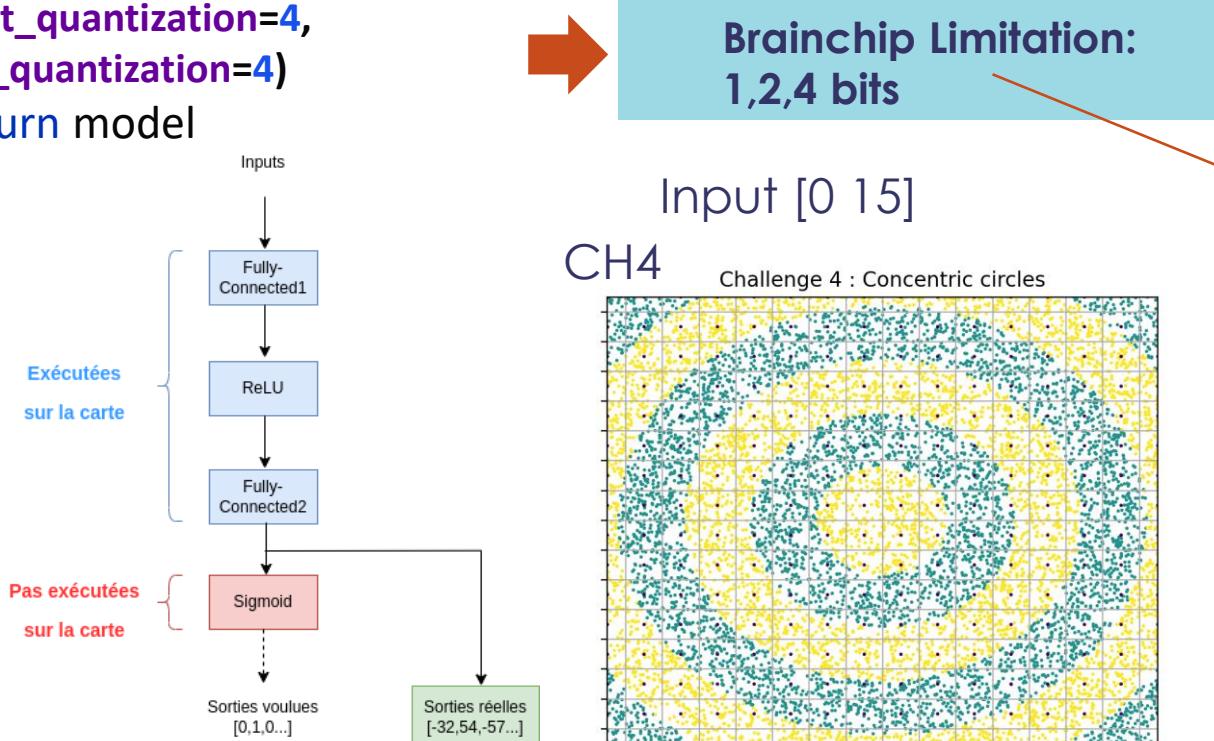
➡ **Brainchip Limitation: 1,2,4 bits**
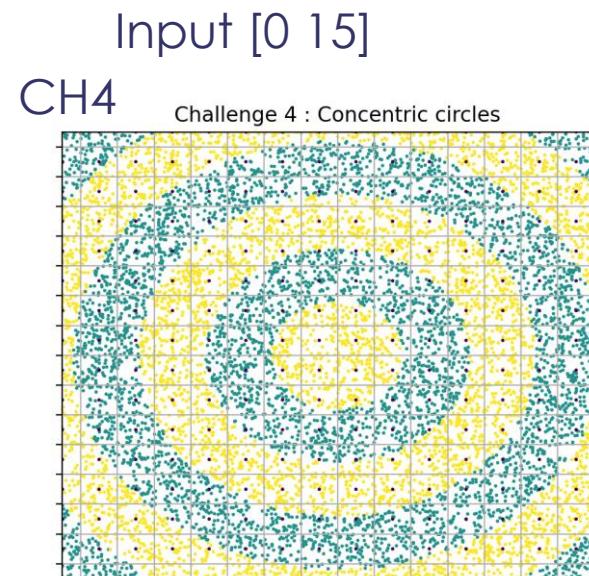
FPS

[weight_q ; activ_q]	ch1	ch3	ch4	ch5
[4;4]	9860	9179	1427	5305
[4;2]	9828	9167	1427	5298
[4;1]	9844	9240	1427	5300
[2;4]	9845	9417	3309	5984
[2;2]	9793	9494	3310	5761
[2;1]	9875	9527	3310	5838

Power Consomption

[weight_q ; activ_q]	ch1	ch3	ch4	ch5
[4;4]	0,09	0,1	0,68	0,18
[4;2]	0,09	0,11	0,68	0,18
[4;1]	0,09	0,1	0,68	0,18
[2;4]	0,09	0,09	0,29	0,17
[2;2]	0,09	0,1	0,29	0,17
[2;1]	0,1	0,1	0,29	0,17

Performances

[weight_q ; activ_q]	ch1	ch3	ch4	ch5
[4;4]	99,84	100	64,99	94,04
[4;2]	99,98	99,45	57,83	93,59
[4;1]	99,98	100	54,8	90,25
[2;4]	99,98	100	62,27	91,84
[2;2]	99,03	93,05	64,78	92,14
[2;1]	99,03	88,12	59,04	79,22

Inputs

Fully-Connected1

ReLU

Fully-Connected2

Exécutées sur la carte

Pas exécutées sur la carte

Sigmoid

Sorties voulues [0,1,0...]

Sorties réelles [-32,54,-57...]

Input [0 15]

CH4

Challenge 4 : Concentric circles

Tx2i

Tx2 4GB

Tx2

Tx2 Nx

Jetson AGX Xavier

Jetson Xavier NX

Jetson Nano

THINK Challenge benchmark

ch1.py ch8.py
Generation.py   --> chx.h5
Train.py   --> chx.pth
Present_res.py

chx.pth

PyTorch
(Inference mode)

Torch2Onnx.py

Torch2Tensorrt.py

Torch-Tensorrt ??

chx.onnx

chx.torch2tensorrt

ONNX RUNTIME

TensorRT

TensorRT

chx.trt

TensorRT

JETSON NANO

JETSON XAVIER NX

JETSON AGX XAVIER

JETSON AGX ORIN
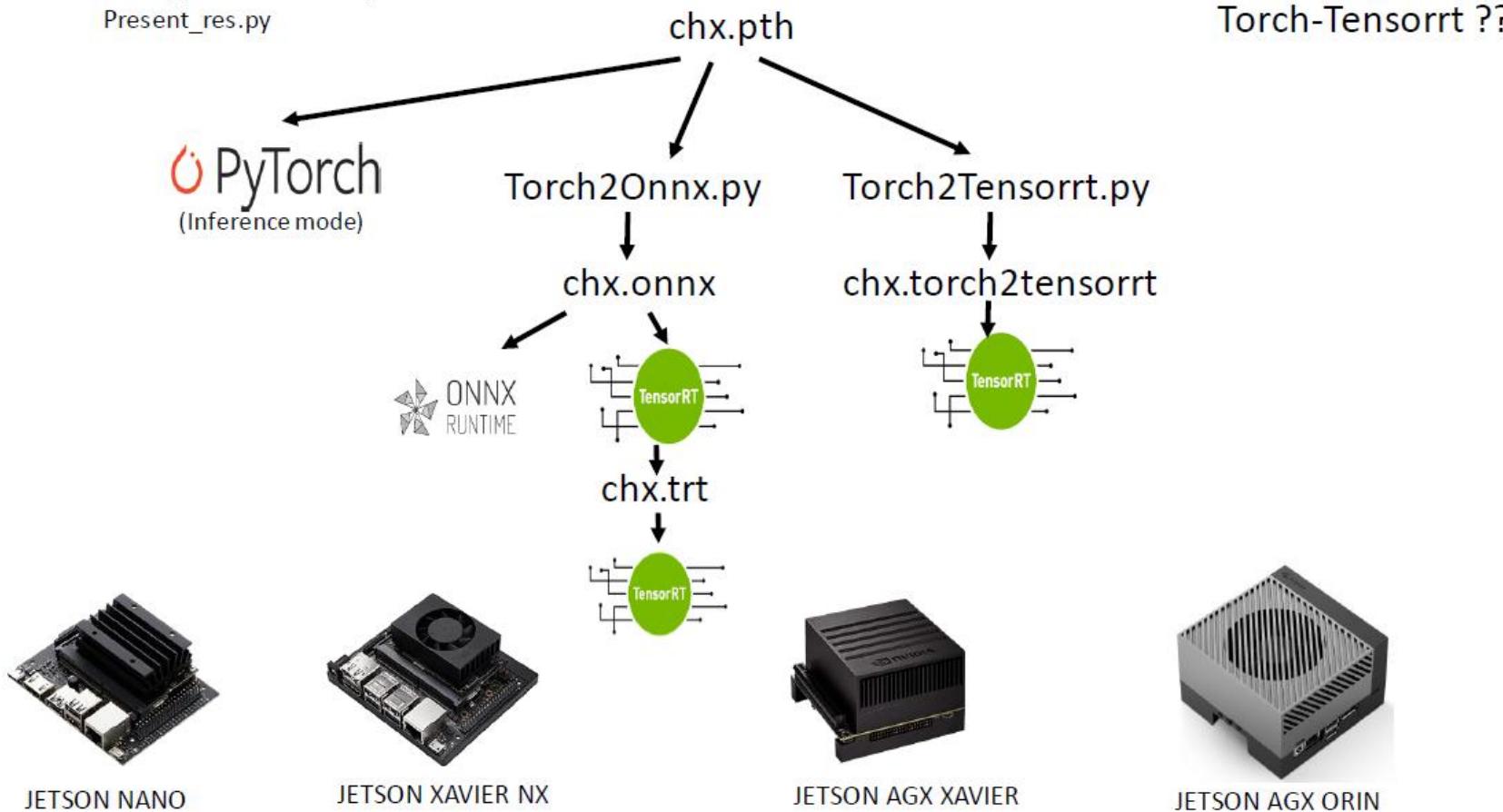
Result GPU+CPU nVidia Jetson: Edge computing

THINK Challenge
NANO 2GB

Software Environnement
- Ubuntu 18.04
- ONNX 1.8.1
- Cuda 10.2.89
- PyTorch 1.8.0
- TensorRT 7.1.3.0
- CudNN 8.0.0.180

NANO-2GB-4-CPUs-1.5GHz

	BATCH_SIZE = 1 (Frame per Second FPS)							
	CH1	CH2	CH3	CH4	CH5	CH6	CH7	CH8
TORCH	2 844		1 375	310	792	error	765	765
ONNX	15 710		11 192	1 587	5 264	5 444	5 357	5 479
TENSOR-RT	419		490	319	956	362	643	787

	BATCH_SIZE = 2000 (Frame per Second FPS)							
CPU-FPS	CH1	CH2	CH3	CH4	CH5	CH6	CH7	CH8
TORCH	1 193 322		71 380	12 036	33 797		34 324	30 937
ONNX	3 655 665		177 230	11 056	37 788	36 340	37 040	36 947
TENSOR-RT	2 182		2 124	3 531	1 733	930	1 872	1 190

NANO-2GB-GPU-128-COREs-921MHz

	BATCH_SIZE = 1 (Frame per Second FPS)							
	CH1	CH2	CH3	CH4	CH5	CH6	CH7	CH8
TORCH	123		108	101	101		96	139
ONNX	9 275		5 366	1 449	2 716	3 544	2 910	2 346
TENSOR-RT	ERROR	ERROR	ERROR	ERROR	ERROR	ERROR	ERROR	ERROR

	BATCH_SIZE = 2000 (Frame per Second FPS)							
	CH1	CH2	CH3	CH4	CH5	CH6	CH7	CH8
TORCH	115		132	107	119		110	125
ONNX	35 133		14 862	3 780	5 633	8 208	16 653	16 364
TENSOR-RT	ERROR	ERROR	ERROR	ERROR	ERROR	ERROR	ERROR	ERROR

THINK Challenge
XAVIER NX 8GB

Software Environnement
- Ubuntu 18.04
- ONNX 1.6.0
- Cuda 10.2.89
- PyTorch 1.10.0
- TensorRT 7.1.3.0
- CudNN 8.0.0.180

XAVIER_NX-8GB-8-CPU-1.4GHz

	BATCH_SIZE=1 (Frame per Second FPS)							
	CH1	CH2	CH3	CH4	CH5	CH6	CH7	CH8
TORCH	3 380		1 850	543	1 179		1 223	1 244
ONNX	10 890		8 751	3 898	6 278	6 431	6 769	6 666
TENSOR-RT	4 189		4 006	3 482	3 534	3 888	3 765	3 930

	BATCH_SIZE=2000 (Frame per Second FPS)							
	CH1	CH2	CH3	CH4	CH5	CH6	CH7	CH8
TORCH	940 542		70 016	13 083	25 937		14 108	35 814
ONNX	2 431 717		175 084	21 810	73 623	74 872	78 427	85 473
TENSOR-RT	1 582 478		1 523 387	1 237 893	1 515 188	1 243 781	1 395 681	1 189 887

XAVIER_NX-8GB-GPU-384-CORES-48-TENSORS-1.1GHz

	BATCH_SIZE=1 (Frame per Second FPS)							
	CH1	CH2	CH3	CH4	CH5	CH6	CH7	CH8
TORCH	1 070		763	579	670		689	706
ONNX	9 623		7 535	3 545	616	5 985	6 152	5 981
TENSOR-RT	ERROR	ERROR	ERROR	ERROR	ERROR	ERROR	ERROR	ERROR

	BATCH_SIZE=2000 (Frame per Second FPS)							
	CH1	CH2	CH3	CH4	CH5	CH6	CH7	CH8
TORCH	2 001		2 422	2 355	2 416		2411	2 410
ONNX	2 380 501		148 964	23 366	83 828	62 273	82 945	68 459
TENSOR-RT	ERROR	ERROR	ERROR	ERROR	ERROR	ERROR	ERROR	ERROR

THINK Challenge
ORIN AGX 32GB

Software Environnement
- Ubuntu 20.04
- ONNX 1.13.1
- Cuda 11.4.14
- PyTorch 1.14.0
- TensorRT 8.4.0.11
- CudNN 8.3.2.49

ORIN-AGX-32GB-12-CPU-2.2GHz

	BATCH_SIZE = 1 (Frame per Second FPS)							
	CH1	CH2	CH3	CH4	CH5	CH6	CH7	CH8
TORCH	8 069		3 880	1 859	2 577		2 663	2 489
ONNX	31 454		24 882	4 041	17 082	14240	15 438	16 085
TENSOR-RT	10 586		10 603	9 722	9 606	10 335	9 882	10 160
	BATCH_SIZE = 2000 (Frame per Second FPS)							
	CH1	CH2	CH3	CH4	CH5	CH6	CH7	CH8
TORCH	2 536 423		89 878	41 122	61 039		63 404	65 885
ONNX	8 561 643		119 971	24 672	99 470	45 403	100 525	159 620
TENSOR-RT	4 910 047		4 807 692	4 806 213	4 991 614	3 597 329	4 488 652	4 786 704

ORIN-AGX-32GB-GPU-1792-CORES-64-TENSORS-930MHz

	BATCH_SIZE = 1 (Frame per Second FPS)							
	CH1	CH2	CH3	CH4	CH5	CH6	CH7	CH8
TORCH	1 630		1 168	923	1 017		1 042	993
ONNX	25 415		20 697	1 614	14 754	14 493	14 573	10 581
TENSOR-RT	9 854		9 070	9 479	8 481	9 903	9 483	10 027
	BATCH_SIZE = 2000 (Frame per Second FPS)							
	CH1	CH2	CH3	CH4	CH5	CH6	CH7	CH8
TORCH	2 798		2 788	2 840	2 832		2 843	2 776
ONNX	8 110 563		89 232	40 631	98 357	98 307	98 580	98 172
TENSOR-RT	5 726 589		5 797 773	5 426 289	5 706 720	5 506 607	4 930 188	5 610 412

Fully Firmware = spatial accelerator

▌ FPGA/SOM OK (optimisation Pb)

▌ SNN: not mature ➔ to continue to investigate (ASIC)

▌ HLS vs VHDL ➔ VHDL fine optimization to reduce resource and latency

Edge Computing

▌ Hardware OK (CPU, GPU, FPGA (SoC))

▌ GPU jetson: Need memory optimisation usage

▌ A lot of hardware with their own tools

  ▌ Time to learn

  ▌ Time to optimize

  ▌ Become an expert in few technologies (Xilinx, Intel, nVidia etc…)

			développement		
AI Edge Computing / Spatial acceleration	AMD Xilinx	Zynq	ARTY Z7 PYNQ ZCU104 SCU102		VITIS-AI / Tensorflow/Keras + HLS4ML PyTorch + FiNN+Brevitas
AI Edge Computing	AMD Xilinx	VERSAL	VK290		HLS+VITIS-AI
AI Edge Computing	nVidia	Jetson	Nano TX2 Xavier NX AGX Xavier Orin		Jetpack SDK + DeepStream SDK + TensorRT
Spatial acceleration	Intel	AGILEX ARRIA	DE10-AGILEX ARRIA 10 GX		Intel HLS
AI Edge Computing					FPGA AI Suite OpenVino
Spatial acceleration	Brainchip	AKD1000	Akida PCIe		Akida MetaTF ML framework
AI Edge Computing	SiPEED	Kendryte210 RISC-V	MAIXDUiNO		SiPEED SDK + micropython
AI Edge Computing	ST Microelectronic	STM32	Nucleo64F411		CubeMX + Cube.AI
AI Edge Computing	Analog Devices	MAX78000	MAX78000E VKIT		Maxim Micros SDK
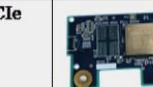
Fully Firmware = spatial accelerator

▌ Test In development framework (N2D2, FiNN…)

▌ SNN: not mature → to continue to investigate (ASIC)

▌ Build a VHDL Librairies without HLS (FPGA Optimisation)

▌ Test real use case

Edge Computing

▌ Hardware OK (CPU, GPU, FPGA (SoC))

▌ To test versatil NN model

▌ New architecture to test like VERSAL/STRATIX

▌ Test real use case

Moke-up of an optimized instrument

▌ Teaching, people trainee

▌ Test new hardware, digital twin model

▌ Mixed model (1DRNN+1DCNN)

▌ Application to HEP (LHC, GW)

▌ Application to select rare events

▌ Develop/Select Embedded AI Framework

Embedded ML Technologies depends on Data Quality (simulation/emulation/data mainframe)
THINK phase 2 ➔ Reseau DAQ