traitement du signal sur GPU REActif

Sébastien COUDERT,
Maria BLAIZOT, Patrice BOURGAULT
.
CNRS/IN2P3/GANIL/DPHY/GT**Acquisition**

25 sept. 2024

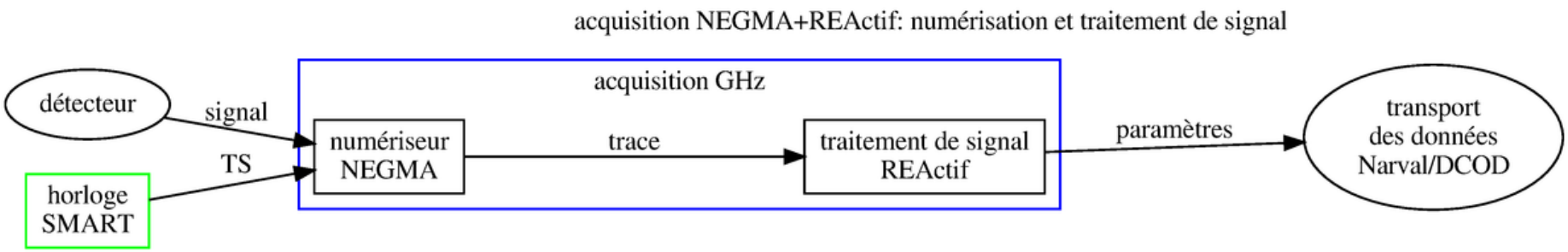
compilations/flux d'acquisition

ordre du temps de "compilation":

- informatique: 30 secondes
- électronique: 3 heures

NEGMA + REActif

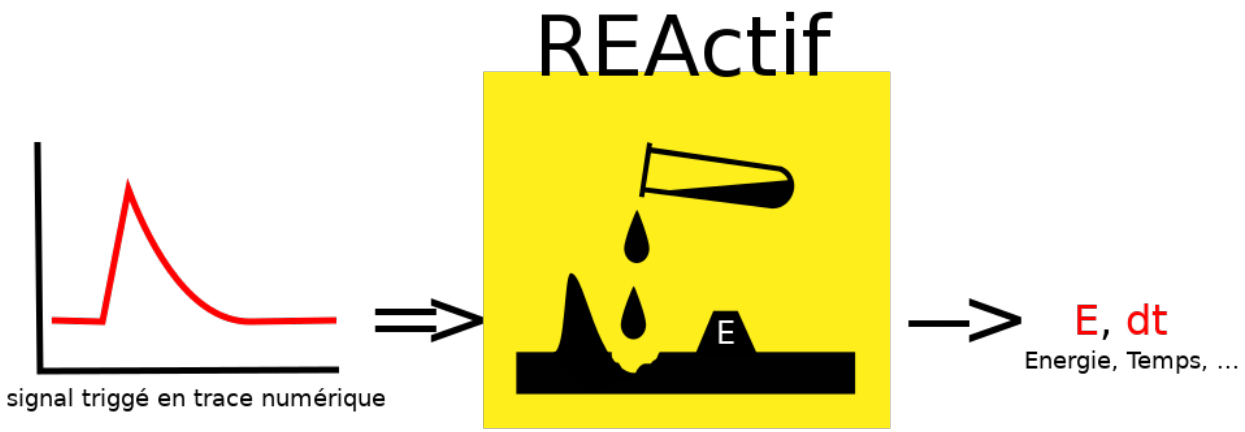
- NEGMA: numérisation + horodatage
- REActif: traitement du signal



REActif: traitement informatique du signal

REActif

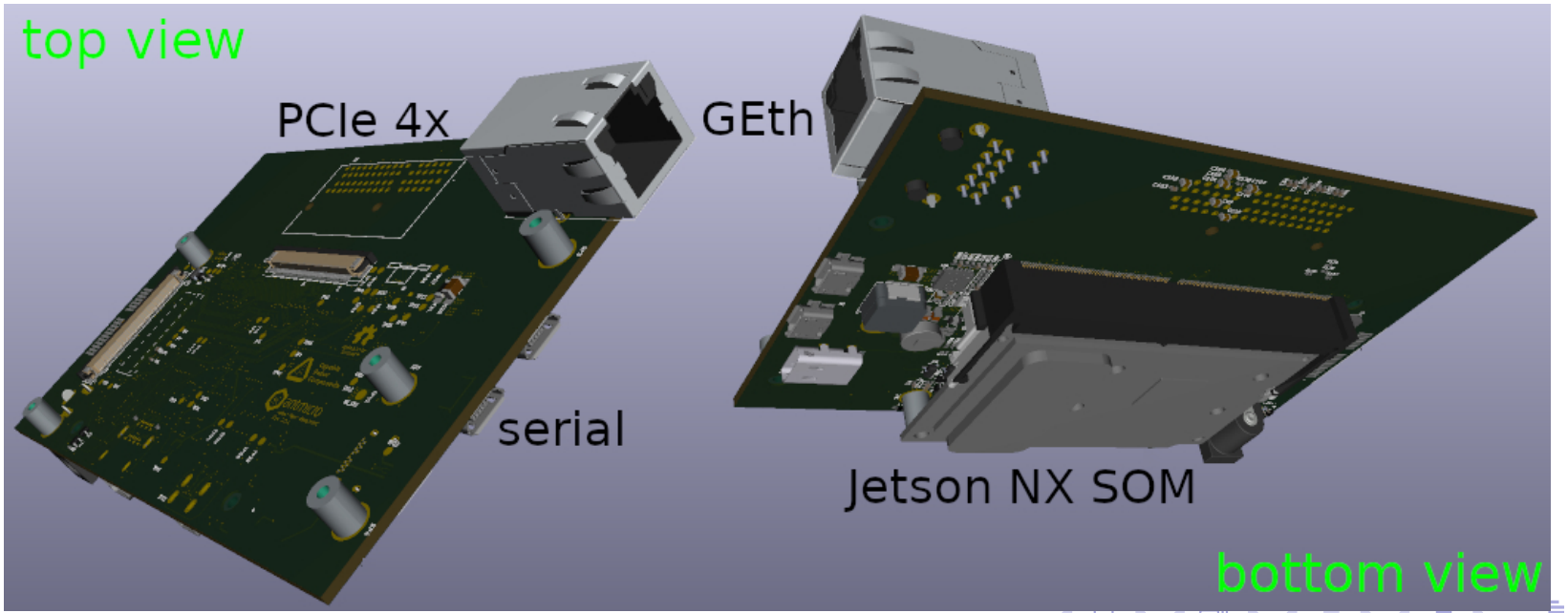
- réaliser le traitement du signal de manière **+souple** et **+rapide** dans un co-processeur facilement programmable (GPU)
- traitement du signal triggé en provenance du numériseur sur co-processeur en language informatique (C++)
- plateforme embarquée afin d'être **+proche** du numériseur avec un encombrement minimal et **-énergivore**
- transfert NEGMA => REActif en GEthernet (16kB MFM)



carte HACKtif (◿ Patrice BOURGAULT)

carte mère open source (GANIL/DPHY/GTA)

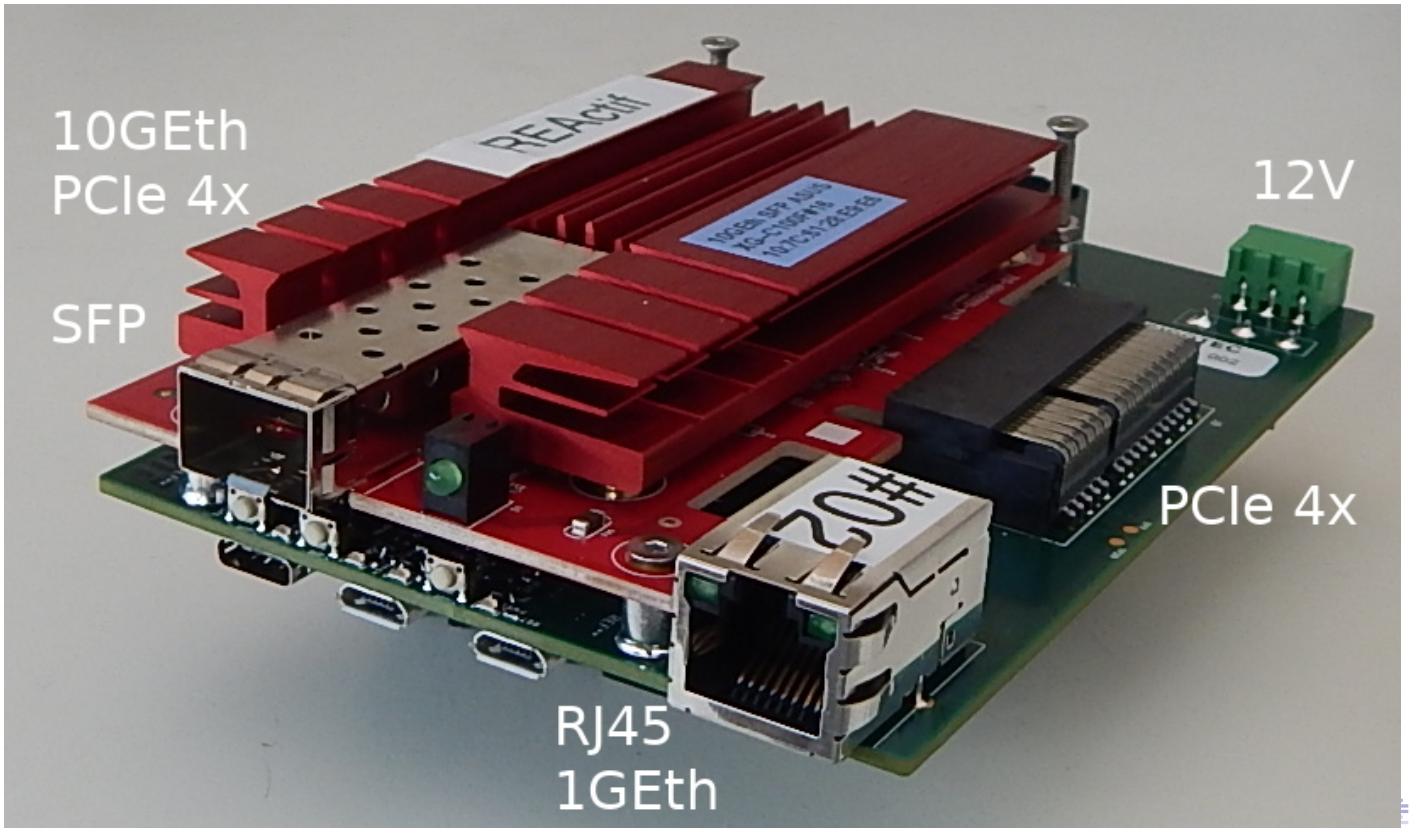
- dim. 120x80mm pour chassis NIM, μTCA, ..., table.
- dessin sous KiCAD (projet open source)
- connecteur SODIMM pour SOM Jetson NX (Nano, TX2 ou xNX)
- connecteur PCIe 4x (à plat) épaisseur 20mm



carte HACKtif

matériels

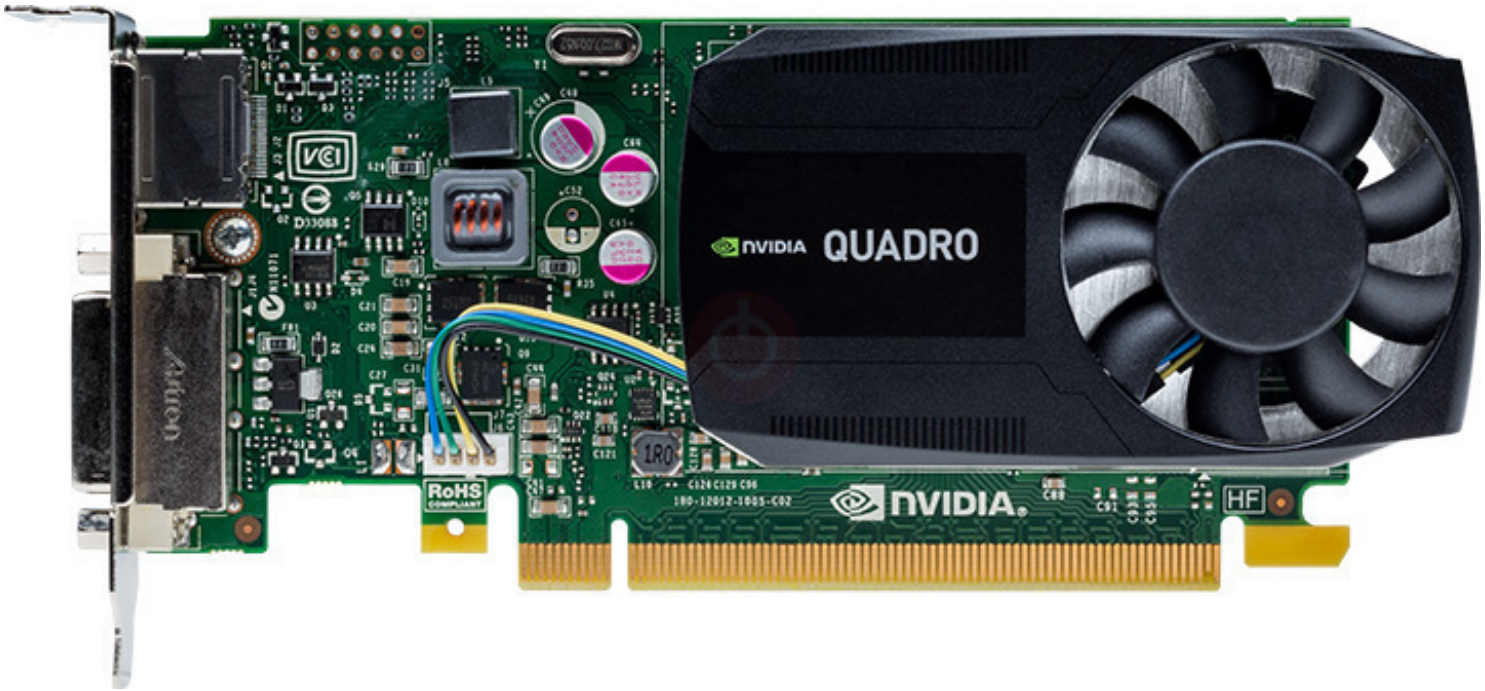
- carte mère HACKtif (en vert) avec SOM Jetson Nano (dessous)
- carte fille 10GEth SFP (en rouge) sur port PCIe 4x, alim. 12V



Jetson SOM: CPU+GPU+RAM

NVidia PCIe

- carte graphique=GPU NVidia + RAM



Jetson SOM

- Tegra=CPU+GPU=ARM+NVidia

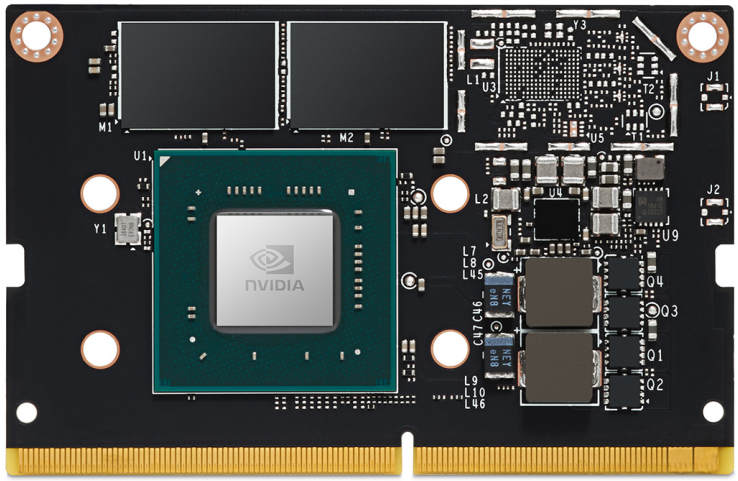
Jetson SOM: CPU+GPU+RAM

NVidia PCIe

- carte graphique=GPU NVidia + RAM

Jetson SOM

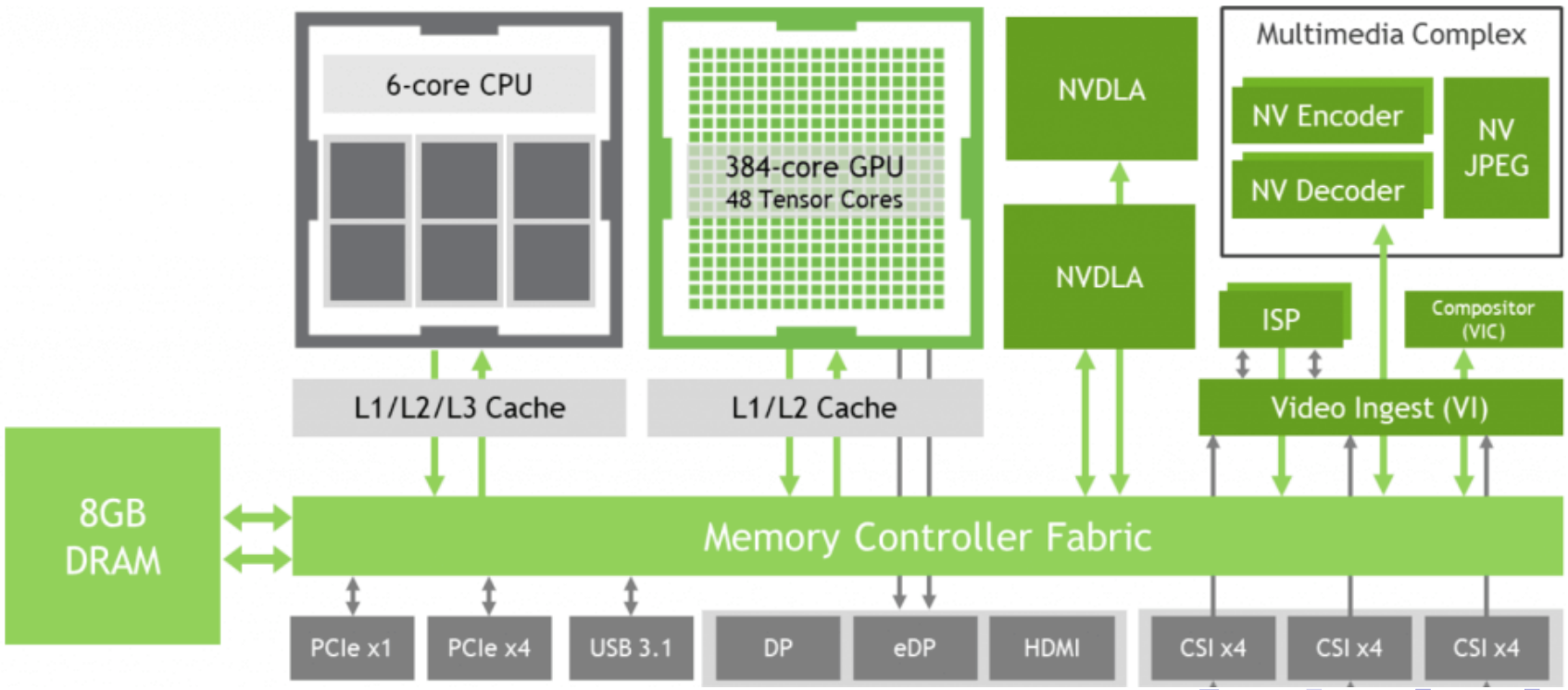
- Tegra=CPU+GPU=ARM+NVidia
- DDR RAM
- NX SOM: soDIMM Nano/TX2NX/XavierNX



Jetson SOM: CPU+GPU+RAM

Tegra ARM (Nano/xNX)

4/6 cores CPU, ARM A57/V8, 1.4/1.9 GHz, 4/8 GB lpDDR4, 64/128bit memory bus



Jetson SOM: CPU+GPU+RAM

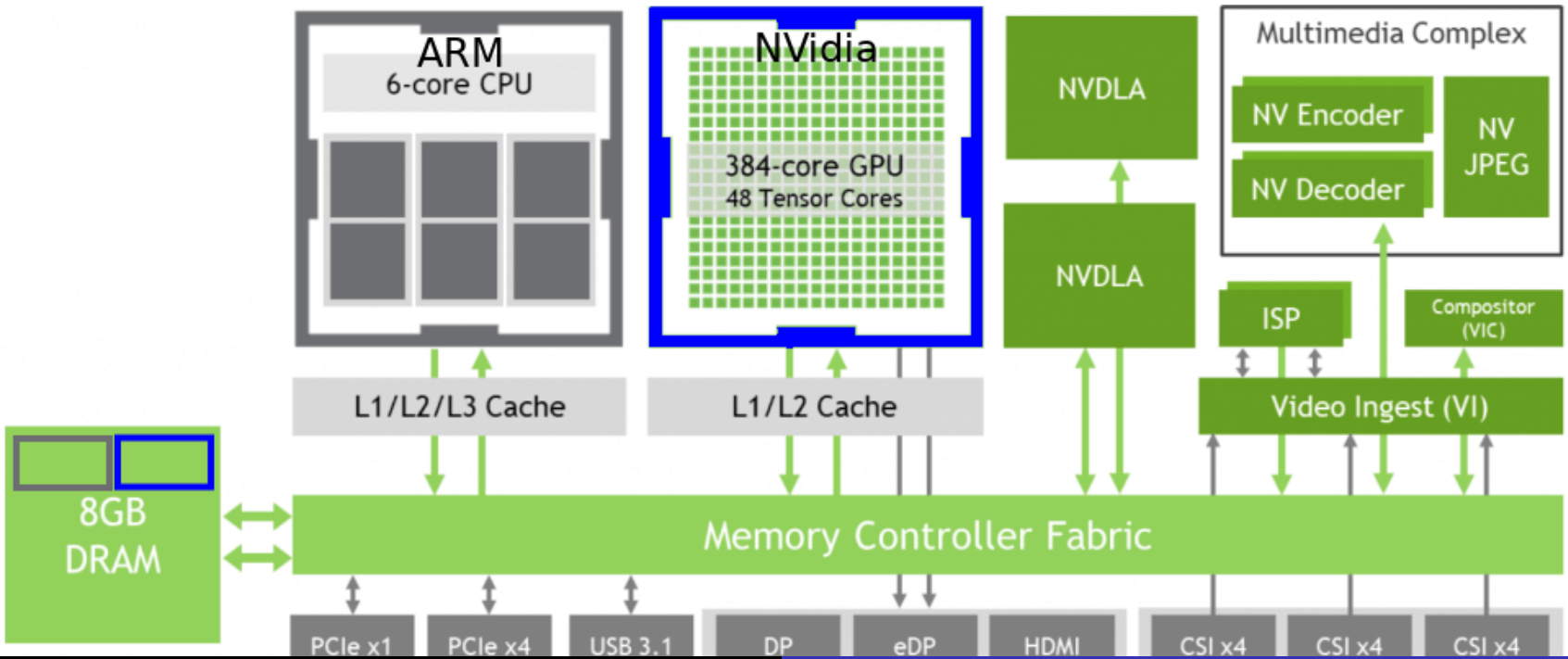
Tegra GPU (Nano/xNX)

128/384 cores GPU, NVidia Maxwell/Volta, 900/1100 MHz^{boost}, DDR partagée

Tegra CPU+GPU

données entre CPU/GPU

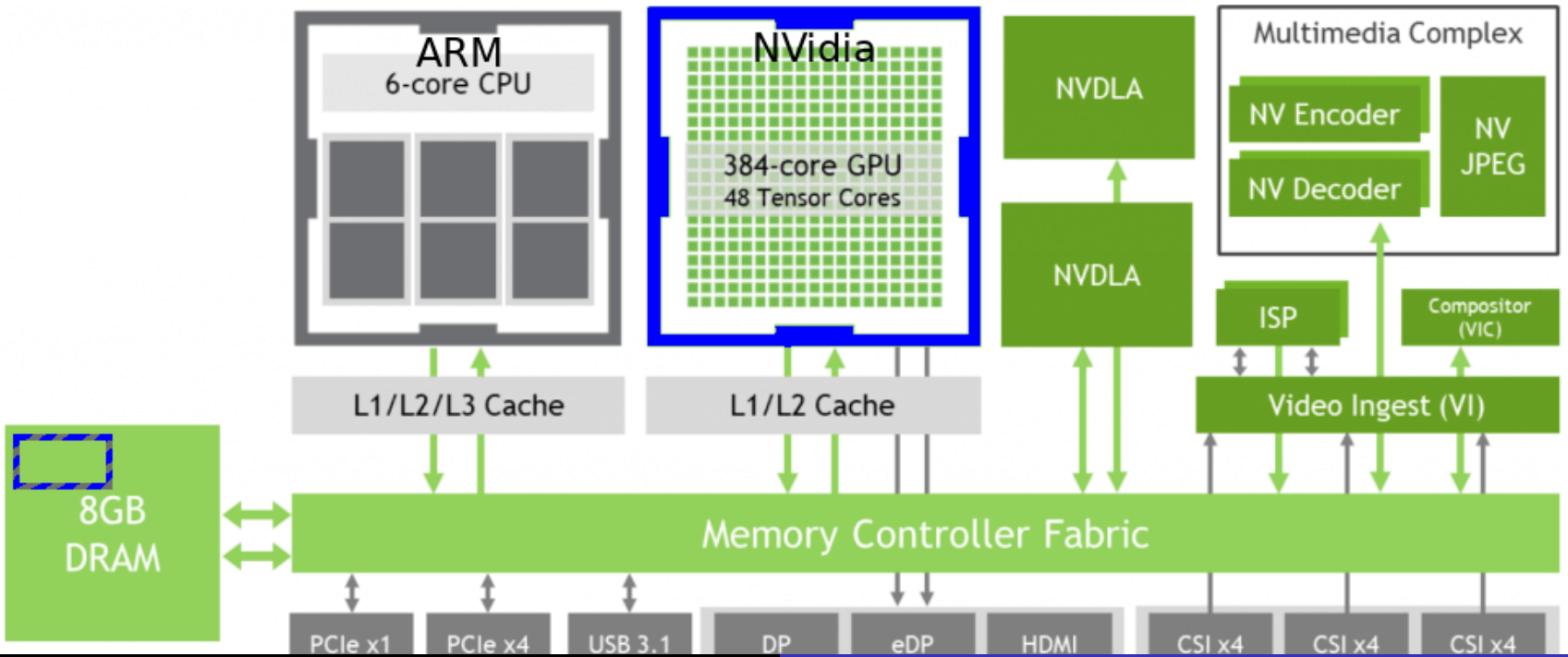
- mémoire partagée, mais zones mémoires réservées
- procédure **MemCopy**: copie des données d'entrée, traitement du signal, copie des données de sortie



Tegra CPU+GPU

données entre CPU/GPU

- mémoire partagée, mais zones mémoires réservées
- procédure **MemAttach**: réservation des données d'entrée, traitement du signal, réservation des données de sortie



procédure sur plateforme hétérogène

couple CPU+GPU

- CPU: IO (ex. GEth) et commande GPU (co-processeur/accelerateur)
- GPU: traitement du signal

Le CPU ordonne l'enchainement de copies mémoires et de noyaux de calcul (kernel) en langage CUDA=C++

procédure sur plateforme hétérogène

couple CPU+GPU

- CPU: IO (ex. GEth) et commande GPU (co-processeur/accelerateur)
- GPU: traitement du signal

Le CPU ordonne l'enchainement de copies mémoires et de noyaux de calcul (kernel) en langage CUDA=C++

example CUDA code: copy $->$ kernel $->$ copy

float h_**p[size]**, *d_**p** ... //cudaMalloc
cuda**Memcpy**Async(d_p, h_p, size, cudaMemcpy**HostToDevice**, str...
cudaStream**Synchronize**(stream);
kernel0<<<blocksPGrid, threadsPBlock, 0, stream>>>(d_p,**size**);
cuda**Memcpy**Async(h_p, d_p, size, cudaMemcpy**DeviceToHost**, str...
cudaStream**Synchronize**(stream);

note sur la **compilation**: un compilateur pour le CPU et un autre compilateur pour le(s) GPU(s) , le tout embarqué dans **un seul binaire**.

procédure sur plateforme hétérogène

couple CPU+GPU

- CPU: IO (ex. GEth) et commande GPU (co-processeur/accelerateur)
- GPU: traitement du signal

Le CPU ordonne l'enchainement de copies mémoires et de noyaux de calcul (kernel) en langage CUDA=C++

example CUDA code: attach $->$ kernel $->$ attach

```
float h_p[size]; ... //cudaMallocManaged
cudaStreamMemAttachAsync(h_p, size, cudaMemAttachGlobal, str...
cudaStreamSynchronize(stream);
kernel0<<<blocksPGrid, threadsPBlock, 0, stream>>>(h_p,size);
cudaStreamMemAttachAsync(h_p, size, cudaMemAttachHost, str...
cudaStreamSynchronize(stream);
```

notes: lors de la compilation, spécifier l'arch. SMS de la Jetson.
Utilisation de l'Unified Mem. pas possible, car Nano fixé sur CUDA 10 !

boucle sur le signal

GPU processeur graphique: traitement de pixel en parallèle (float p[**size**])

boucle sur CPU: explicite (séquentielle)

```
for(int i=0;i<size;++i) p[i]=i;
```

boucle sur GPU: implicite (parallèle)

boucle sur le signal

GPU processeur graphique: traitement de pixel en parallèle (float p[**size**])

boucle sur CPU: explicite (séquentielle)

```
for(int i=0;i<size;++i) p[i]=i;
```

boucle sur GPU: implicite (parallèle)

CPU: *kernel call (implicite for)*

```
kernel0<<<blocksPGrid, threadsPBlock, 0, stream>>>(d_p,size);
```

GPU kernel: *p[i]=i;*

```
__global__ void kernel0(float *p, int size)
{
const int i = blockDim.x * blockIdx.x + threadIdx.x;
if(i<size) p[i]=i;
}//kernel0
```

REAction: traitement du signal sur co-processeur

code informatique

- traitement du signal sur GPU en langage informatique (CUDA=C++) 1 coeur par trace (ex. calcul d'énergie)
- plusieurs traces en parallèle (ex. 128 traces sur JetsonNano)

calcul de l'énergie d'un signal: code de filtrage trapèzoidal (CUDA)

```
__global__ void trapezoidal_filter(const float *e, *s , int dimX, dimY
, const int ks, const int ms, const float alp, const int decalage)
{
const int si = threadIdx.y*dimX, ei = si+dimX, pi = si+decalage-1;...
//create a filter
for(int n=pi;n<ei;++n)
s[n]=2*s[n-1]-s[n-2] + e[n-1]-alp*e[n-2] -e[n-(ks+1)]
+alp*e[n-(ks+2)]-e[n-(ks+ms+1)]+alp*e[n-(ks+ms+2)]
+e[n-(2*ks+ms+1)]-alp*e[n-(2*ks+ms+2)];
}
```

mesure de performance

mesure de debit Ethernet

PCIe 4x GEth: **10**GEth 700MB/s ,**2.5**GEth 300MB/s , **1**GEth 100MB/s

mesure du temps d'execution

mesure de performance

mesure de debit Ethernet

PCIe 4x GEth: **10**GEth 700MB/s ,**2.5**GEth 300MB/s , **1**GEth 100MB/s

mesure du temps d'execution

- durée moyenne de plusieurs iterations (copy$^{\Rightarrow}$, kernel(s)^{GPU}, copy$^{\Leftarrow}$)
- données en mémoire ^{CPU} ($\lhd$ 32 ensembles de 128 signaux = 65MB)
- durée: $\Delta t = t_{end} - t_{start}$

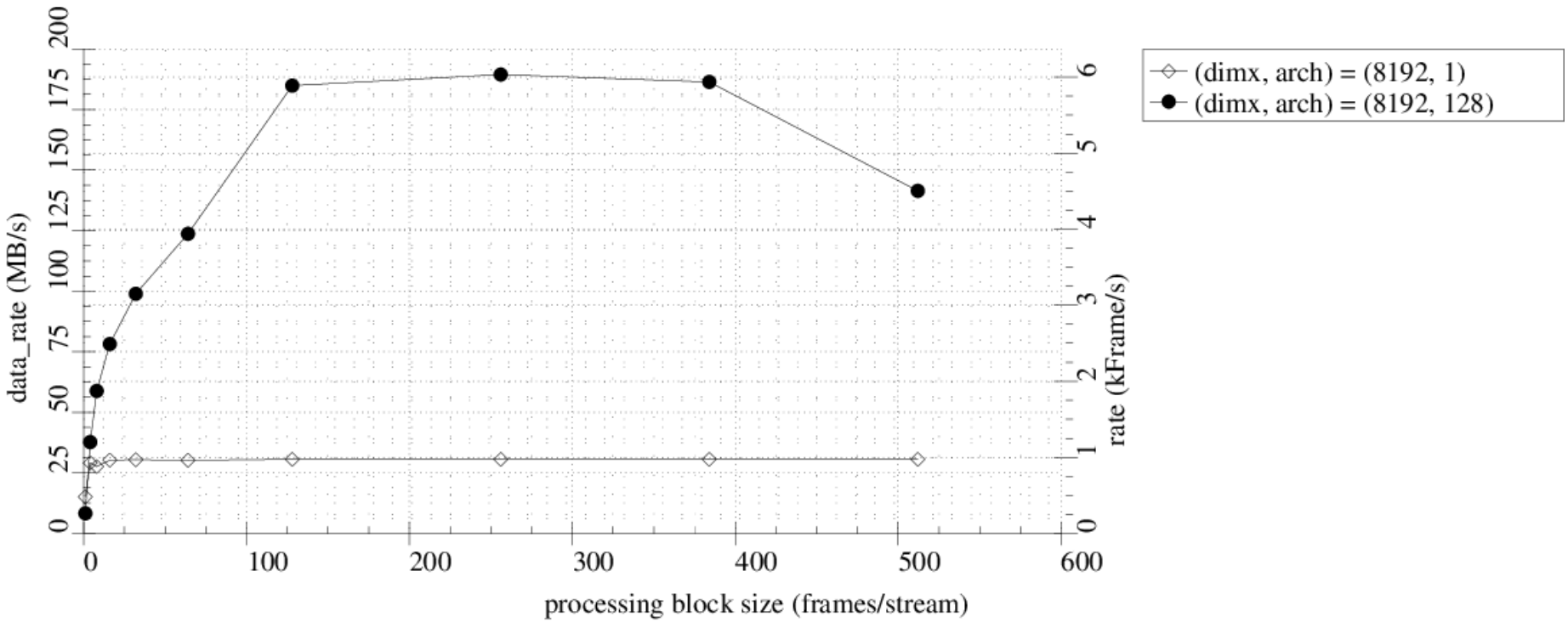
CPU: elapsed time

cudaEvent_t start,stop; cuda**EventCreate**(&start...;float msecTotal=0.0f;
cuda**EventRecord**(**start**, stream);
run_kernel(algo, blocksPerGrid,threadsPerBlock, ...
cuda**EventRecord**(**stop**, stream);
cuda**EventSynchronize**(stop);
cudaEvent**ElapsedTime**(&msecTotal, start, stop);

performance de traitement du signal

calcul d'energie sur 128cores: Jetson Nano

copy perf.: 6k frame/s, 200 MB/s, >1GEth (16kBoF=8kS, 128 GPU cores)

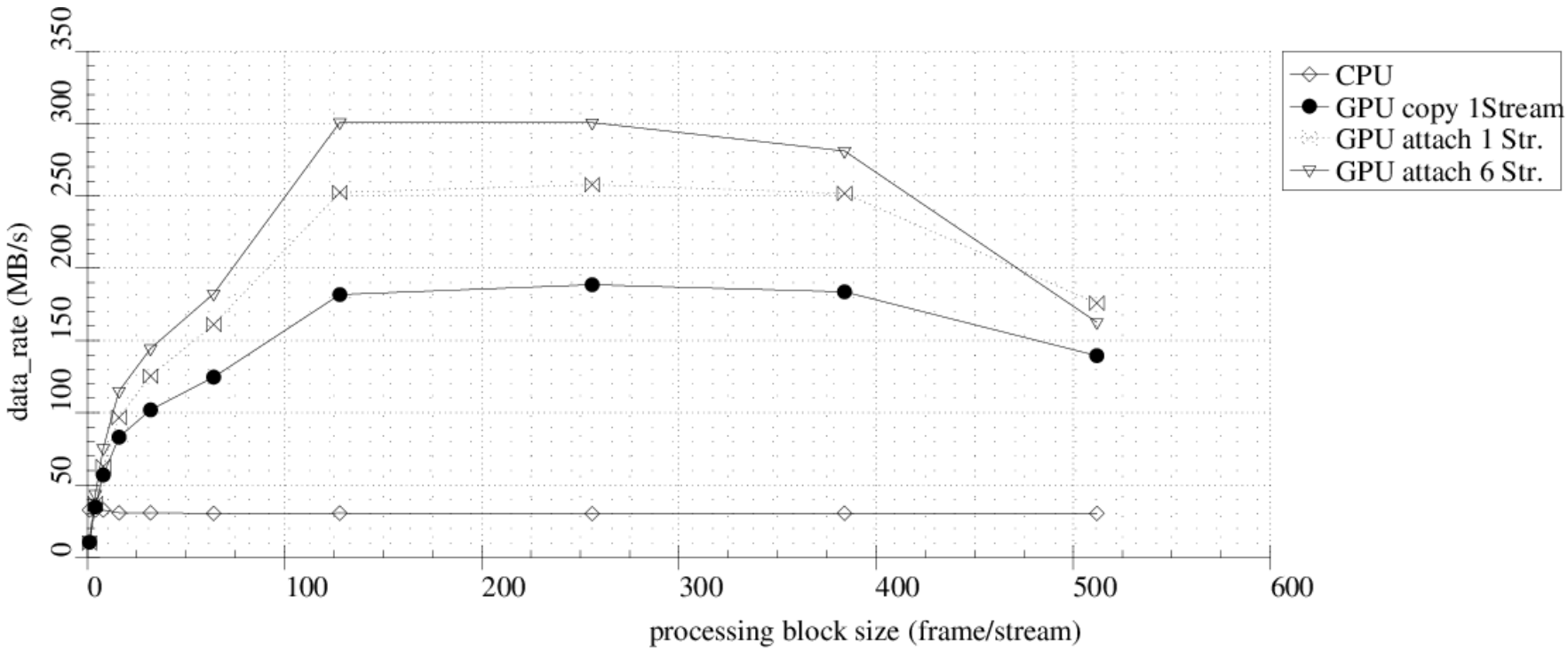


data rate (and frame rate) vs. 8kS frames per stream (1 str. on 128 cores in mode 10W) arch.=Nano

performance de traitement du signal

calcul d'energie sur 128cores: Jetson Nano

attach+stream perf.: 10k frame/s, 300 MB/s, 2.5GEth (16kBoF=8kS, 128 GPU c.)

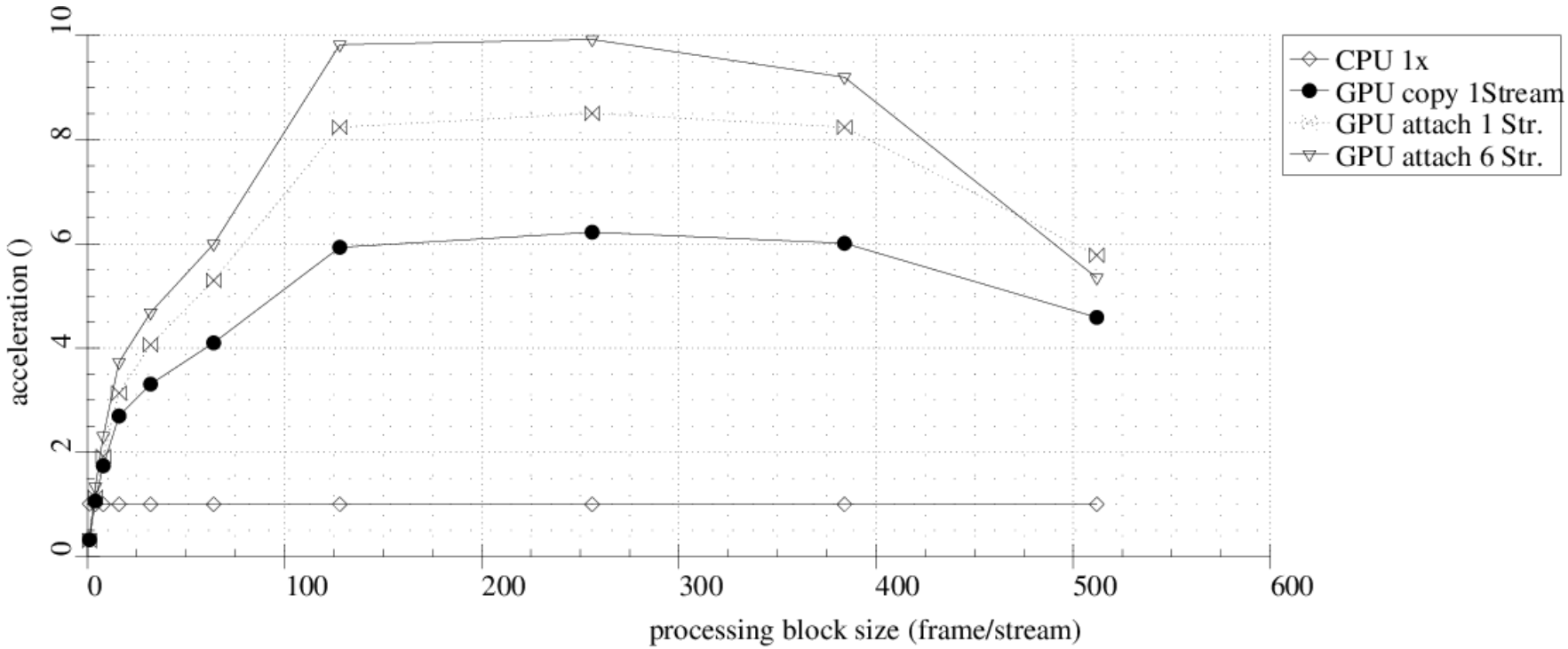


data rate vs. number of frames per Stream (8kS frame) for different run scheme: MemCopy, MemAttach and MemAttach on 6 streams

performance de traitement du signal

accélération du calcul d'energie sur 128cores: Jetson Nano

attach+stream perf.: 10x , max.: 6x⇒8x⇒10x (16kBoF=8kS, 128 GPU cores)

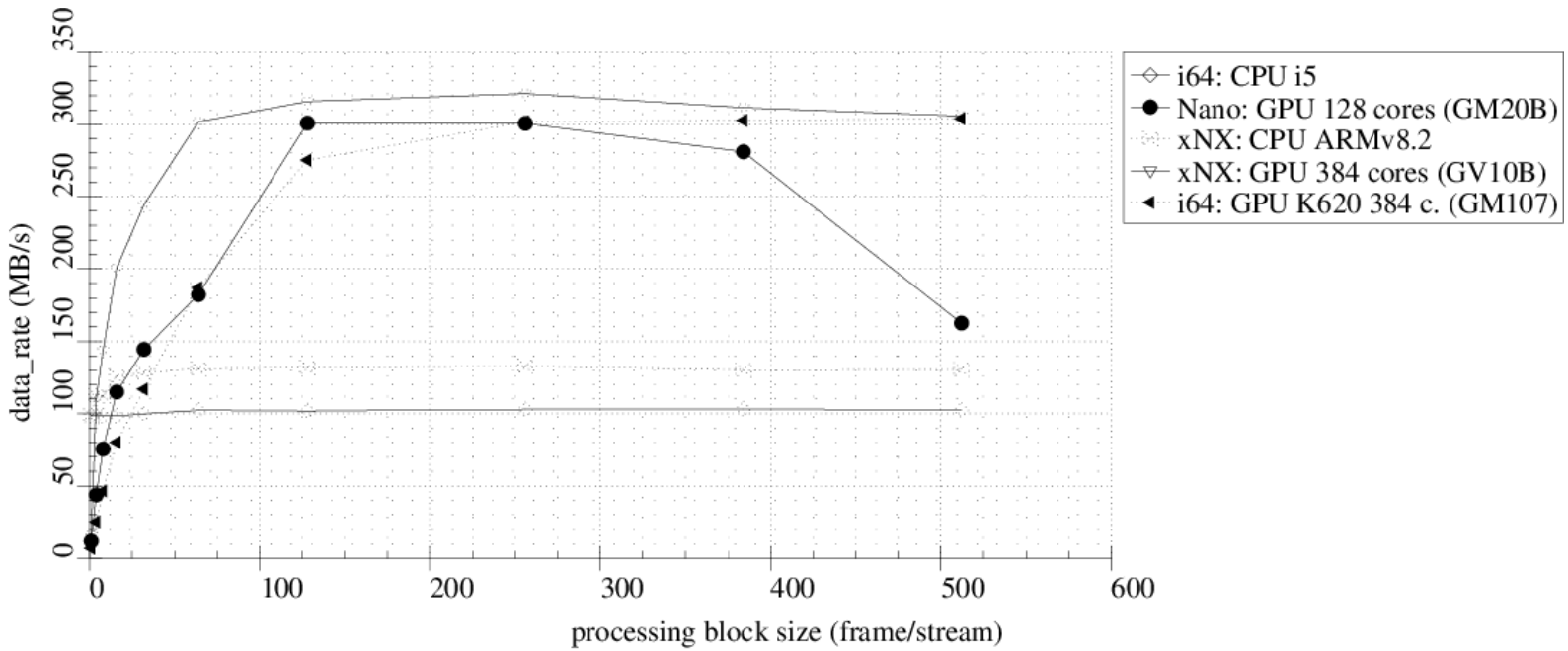


acceleration of GPU regarding to CPU vs. number of frames per Stream (8kS frame)

performance énergétique et encombrement

calcul d'énergie sur 384cores: Jetson xNX vs i5+K620

- performance identique entre embarqué/PCIe (xNX ↑ en petits quantités de données)
- 10 fois moins de consomation électrique (10W vs 100W)
- 10 fois moins d'encombrement (1L vs 10L)

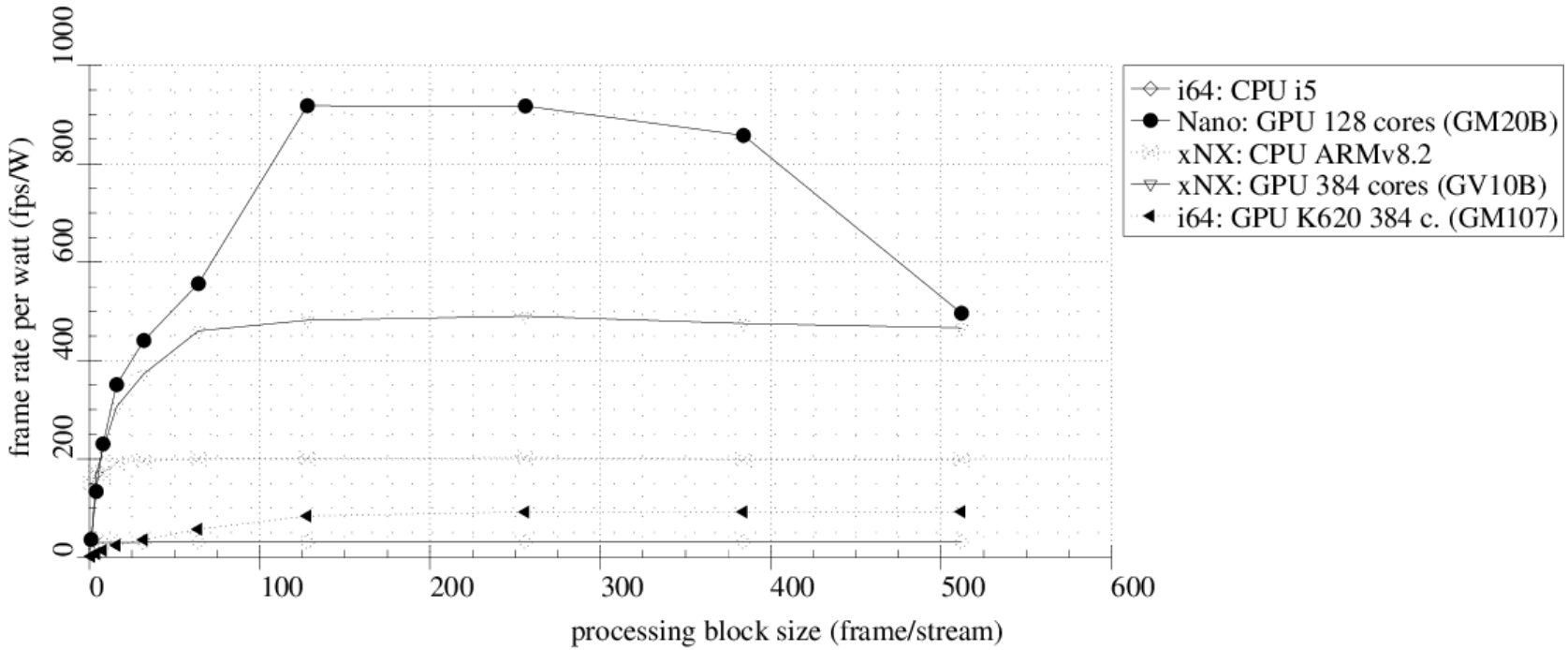


data rate vs. 8kS frames per stream (Nano@10W, xNX@20W, K620@100W)

performance énergétique et encombrement

calcul d'énergie: Jetson Nano et xNX vs i5+K620 (10W, 20W, 100W)

- efficiency in frame/J, i.e. fps/W
- For 1W, get 1k frame/s for Nano and only 100 fps for K620 dGPU.

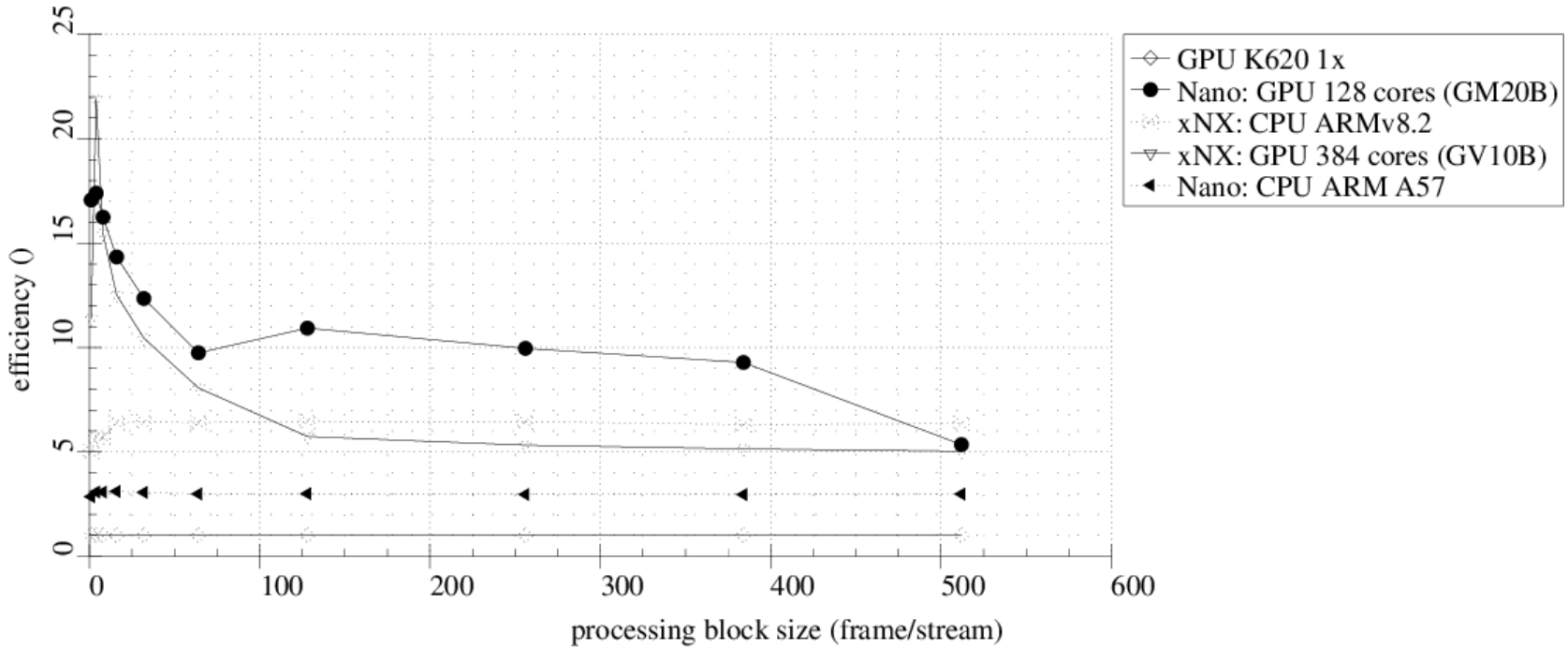


frame rate per watt vs. 8kS frames per stream (Nano@10W, xNX@20W, K620@100W), fps=frame per second

performance énergétique et encombrement

calcul d'énergie: Jetson Nano et xNX vs i5+K620 (10W, 20W, 100W)

- efficiency factor based on Intel64 reference
- Nano is 10 time more efficient than K620 dGPU.



efficiency vs. 8kS frames per stream (Nano@10W, xNX@20W, K620@100W)

conclusion et perspectives

REActif

- traitement du signal en langage simple, souple et REActif
- tests préliminaires convainquants en parallèlisation naive
- gains en prix, énergie et encombrement (carte HACKtif)

REActif

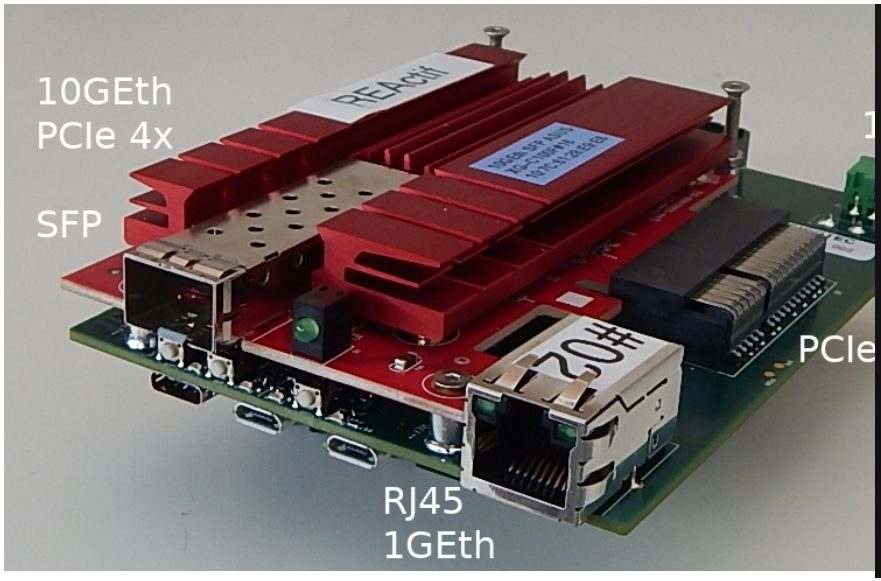
- optimisations , par ex. utilisation de la mémoire ^{GPU} partagee , mais garder la simplicité de codage
- code de production (REAction) traitement du signal + transferts Ethernet: réception/envoi des données
- tests de la carte HACKtif (matériel,logiciel,performance)

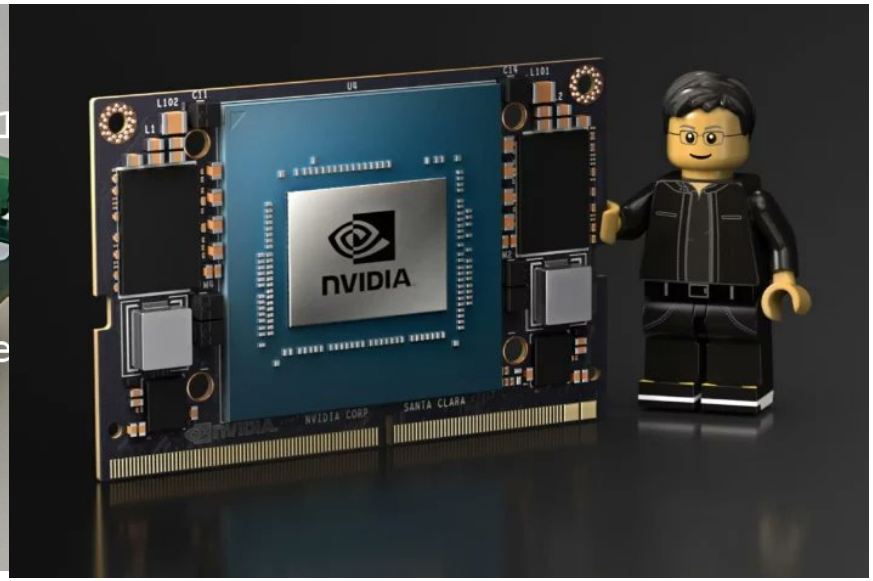
NEGMA+REActif

- rassemblement de RF-SOC + HACKtif via lien SFP GEth

REActif/HACKtif (GANIL)

- 10GEth
- Tegra
- Linux



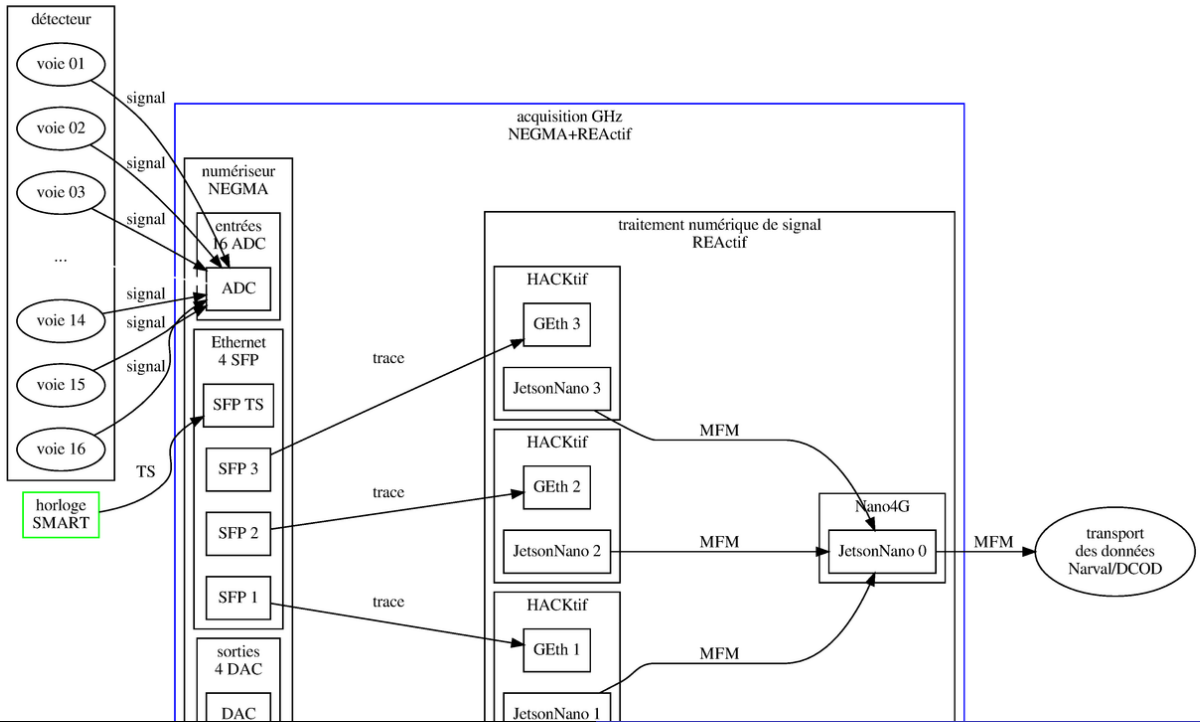


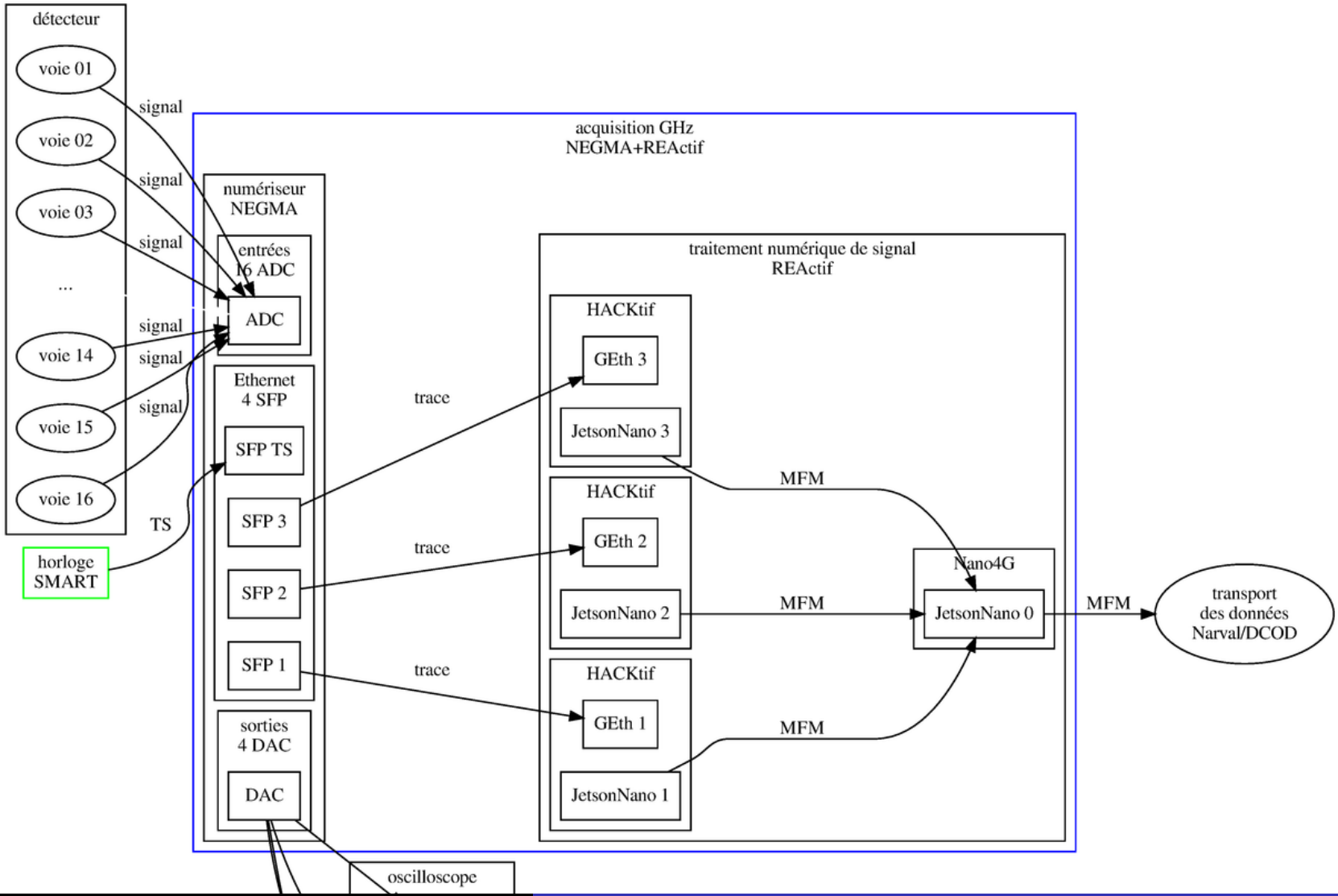
annexes

matériels d'acquisition

NEGMA + REActif

- NEGMA: 1 kit de développement RF-SOC ZCU-216 (Xilinx)
- REActif: 3 cartes HACKtif (GTA -open source-), 1 kit de dév.




détecteur
voie 01
voie 02
voie 03
...
voie 14
voie 15
voie 16
signal
signal
signal
signal
signal
signal
TS
horloge
SMART
acquisition GHz
NEGMA+REActif
numériseur
NEGMA
entrées
16 ADC
ADC
Ethernet
4 SFP
SFP TS
SFP 3
SFP 2
SFP 1
sorties
4 DAC
DAC
trace
trace
trace
traitement numérique de signal
REActif
HACKtif
GEth 3
JetsonNano 3
HACKtif
GEth 2
JetsonNano 2
HACKtif
GEth 1
JetsonNano 1
MFM
MFM
MFM
Nano4G
JetsonNano 0
MFM
transport
des données
Narval/DCOD
oscilloscope

RF-SOC révolution électronique

ADC intégré au SOC

- ADC+DAC intégrés au FPGA

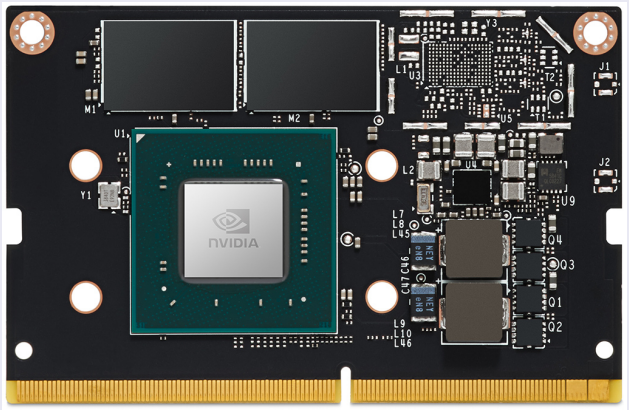
glossaire

- **SoC**: system on a chip, i.e. CPU+FPGA+ADC on single chip
- **FPGA**: Field-Programmable Gate Array , **ADC**: Analog-to-Digital Converter , **DAC**: Digital-to-Analog Converter

Jetson révolution informatique

SoC intégré au SOM

- [CPU+GPU] intégré avec la mémoire vive (RAM)

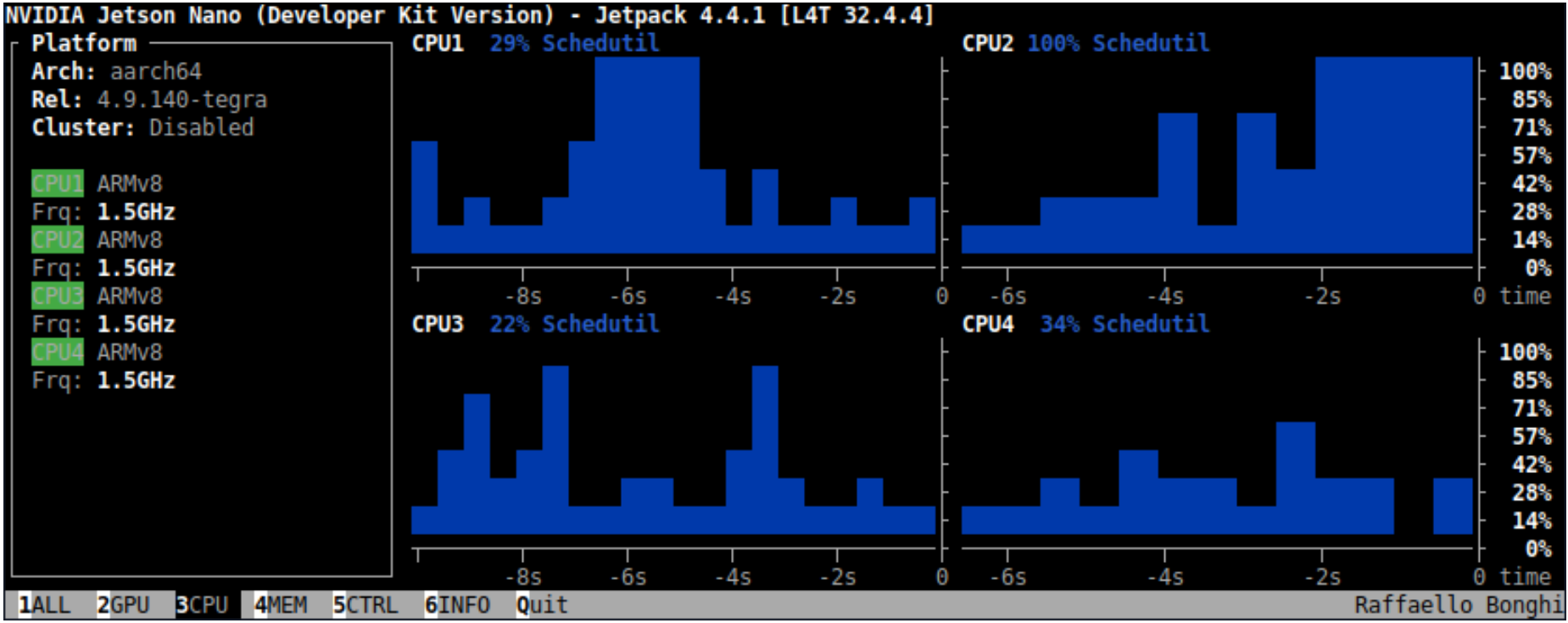


glossaire

- **SOM**: system on a module, i.e. RAM and SoC chips on a board
- **SoC**: system on a chip, i.e. CPU+GPU on single chip
- **RAM**: random-access memory , **GPU**: graphics processing unit , **CPU**: central processing unit

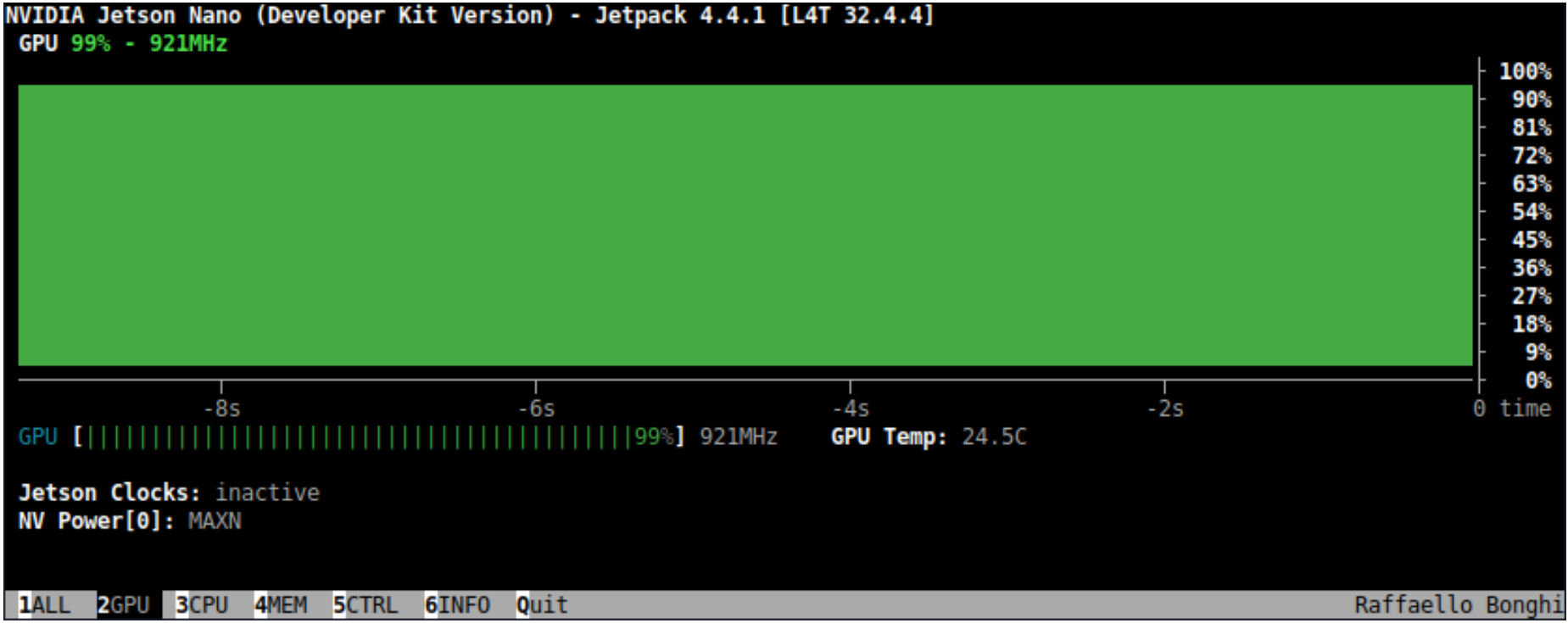
charge pendant le traitement du signal

CPU



charge pendant le traitement du signal

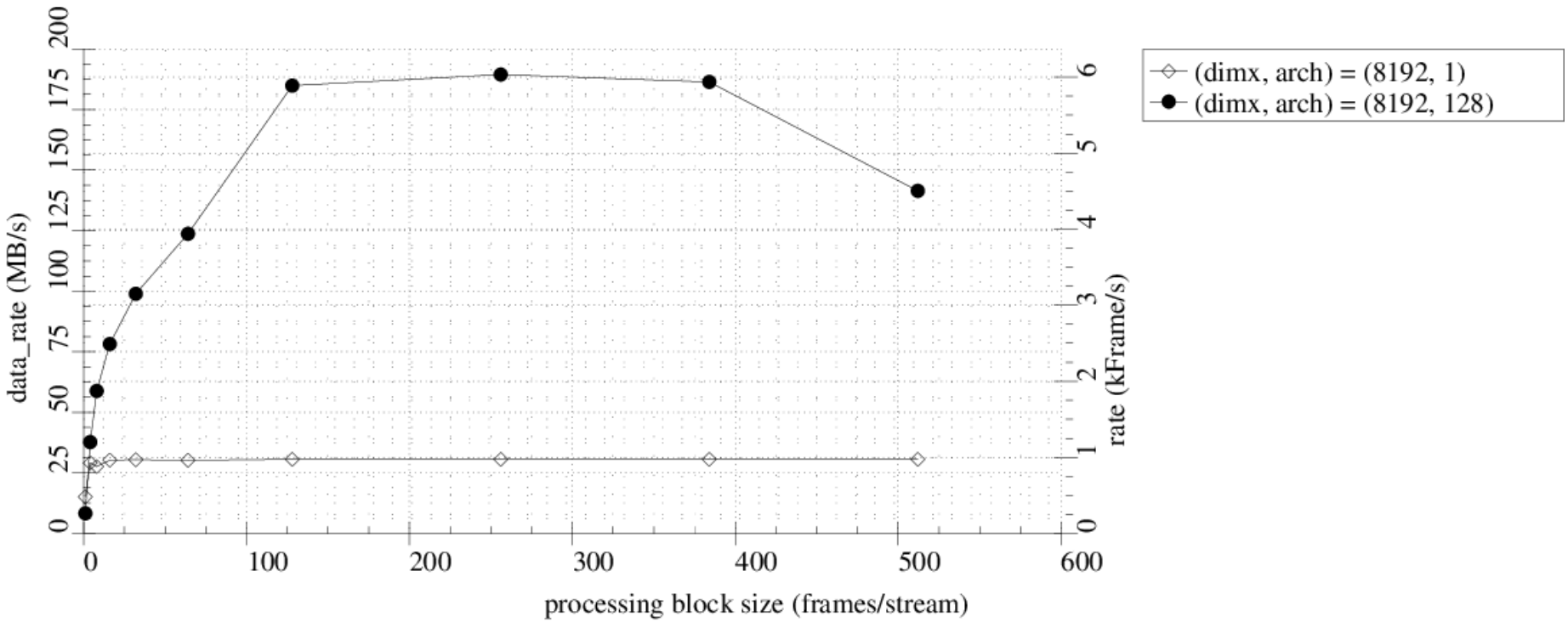
GPU



performance de traitement du signal

calcul d'energie sur 128cores: Jetson Nano

copy perf.: 6k frame/s, 200 MB/s, >1GEth (16kBoF=8kS, 128 GPU cores)

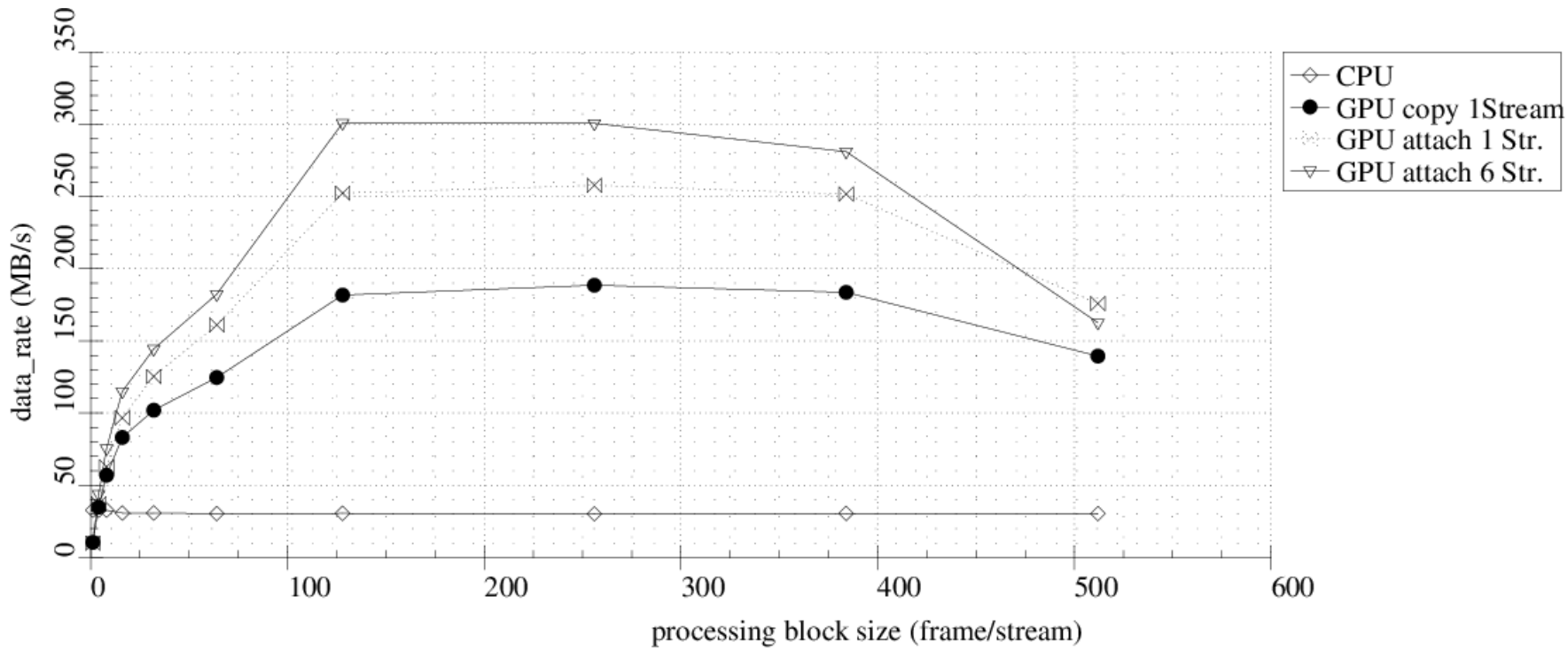


data rate (and frame rate) vs. 8kS frames per stream (1 str. on 128 cores in mode 10W) arch.=Nano

performance de traitement du signal

calcul d'energie sur 128cores: Jetson Nano

attach+stream perf.: 10k frame/s, 300 MB/s, 2.5GEth (16kBoF=8kS, 128 GPU c.)

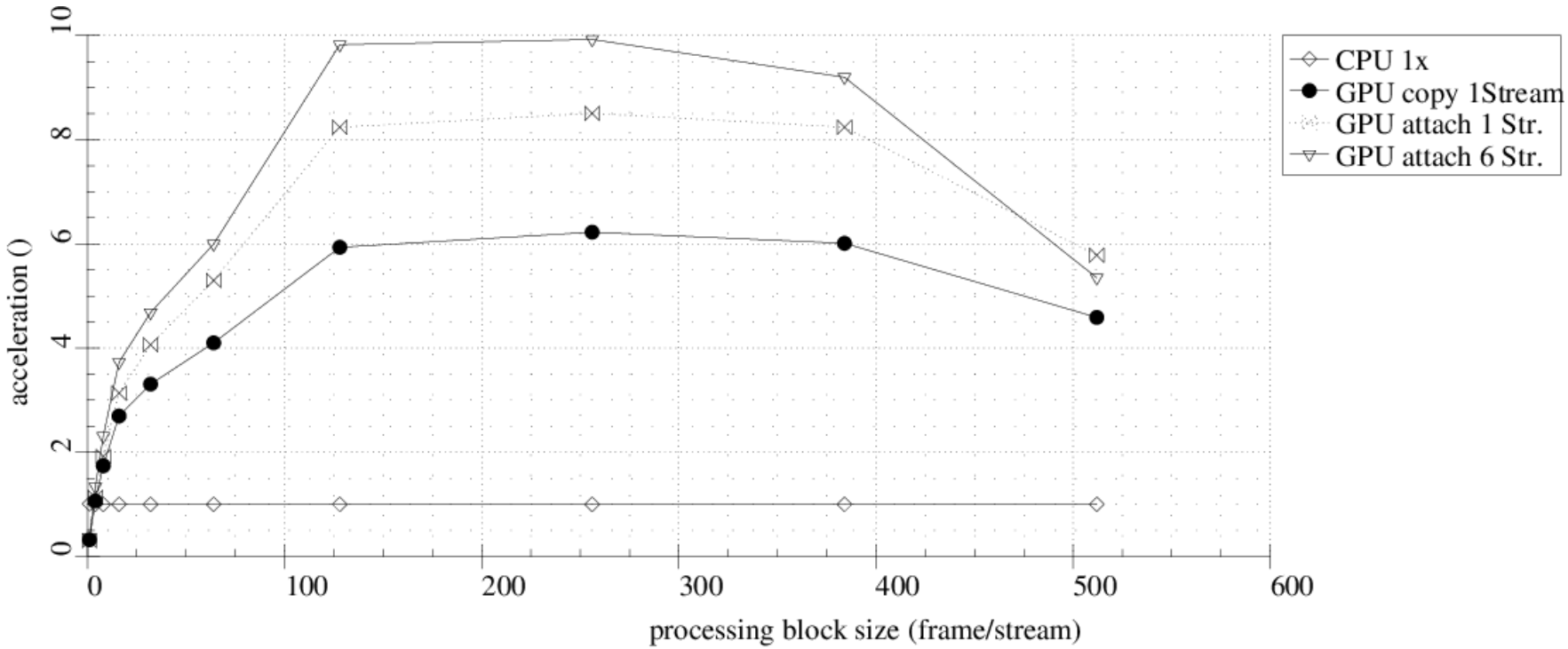


data rate vs. number of frames per Stream (8kS frame) for different run scheme: MemCopy, MemAttach and MemAttach on 6 streams

performance de traitement du signal

accélération du calcul d'energie sur 128cores: Jetson Nano

attach+stream perf.: 10x , max.: 6x⇒8x⇒10x (16kBoF=8kS, 128 GPU cores)

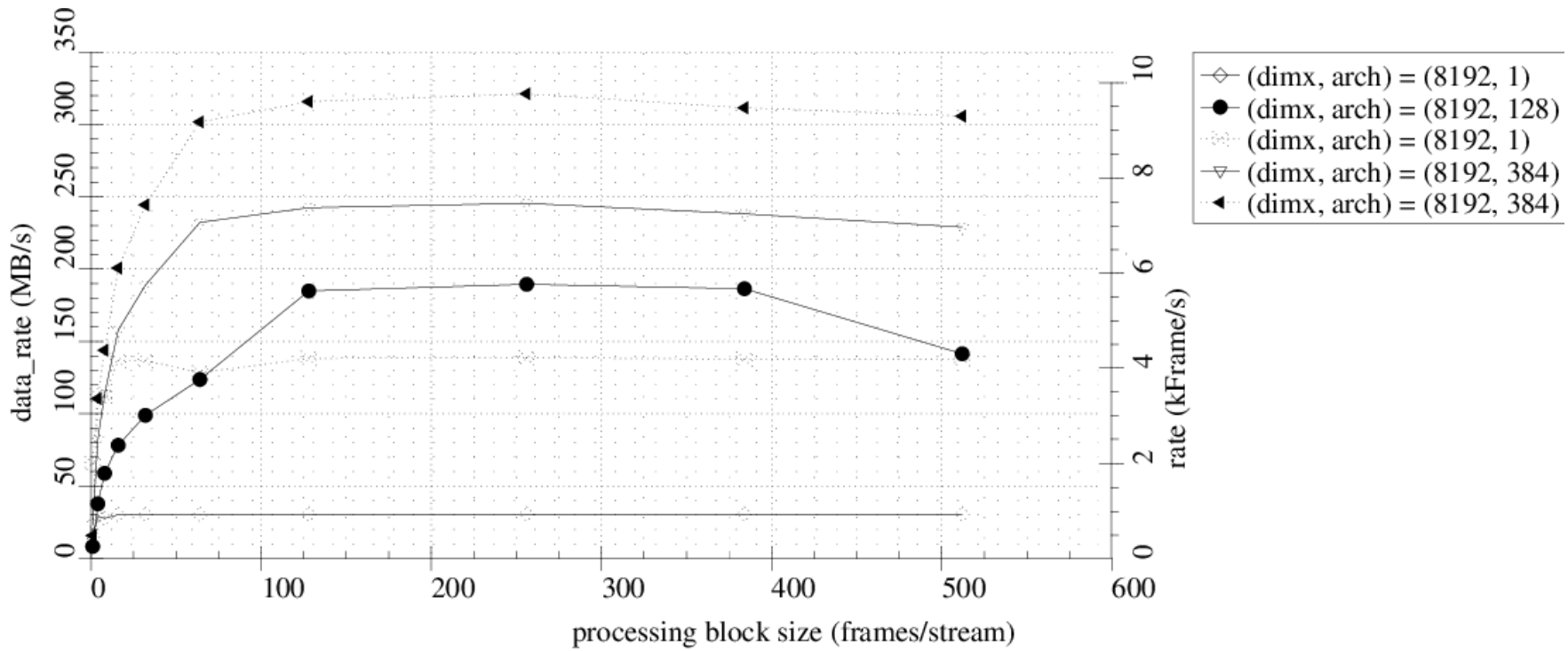


acceleration of GPU regarding to CPU vs. number of frames per Stream (8kS frame)

performance JetsonNano/XavierNX

calcul d'energie: Jetson xNX

perf.: 10k frame/s, 300 MB/s, 2.5 GEth (power model: 10W/15W,20W)

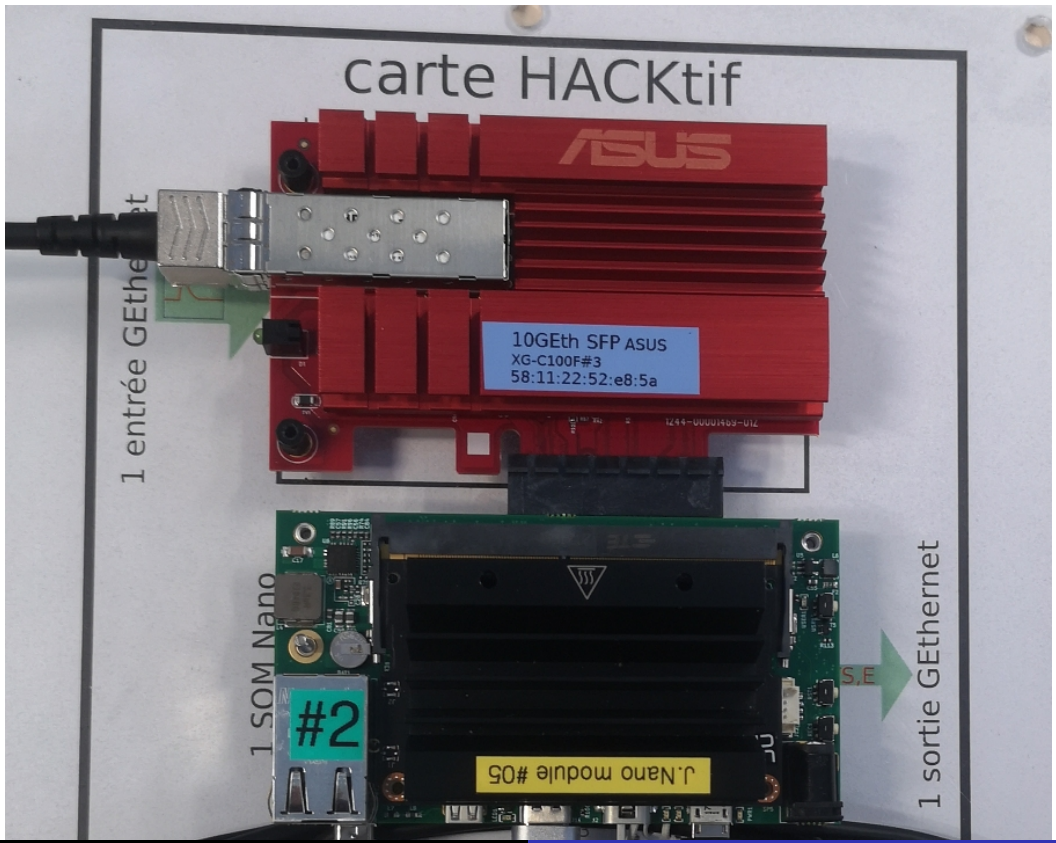


data rate vs. 8kS frames per stream (1 str. on 128/384 cores in mode 10W/15W/20W) arch.=Nano/xNX

carte HACKtif assemblée (v0)

matériels principaux du proto. assemblé

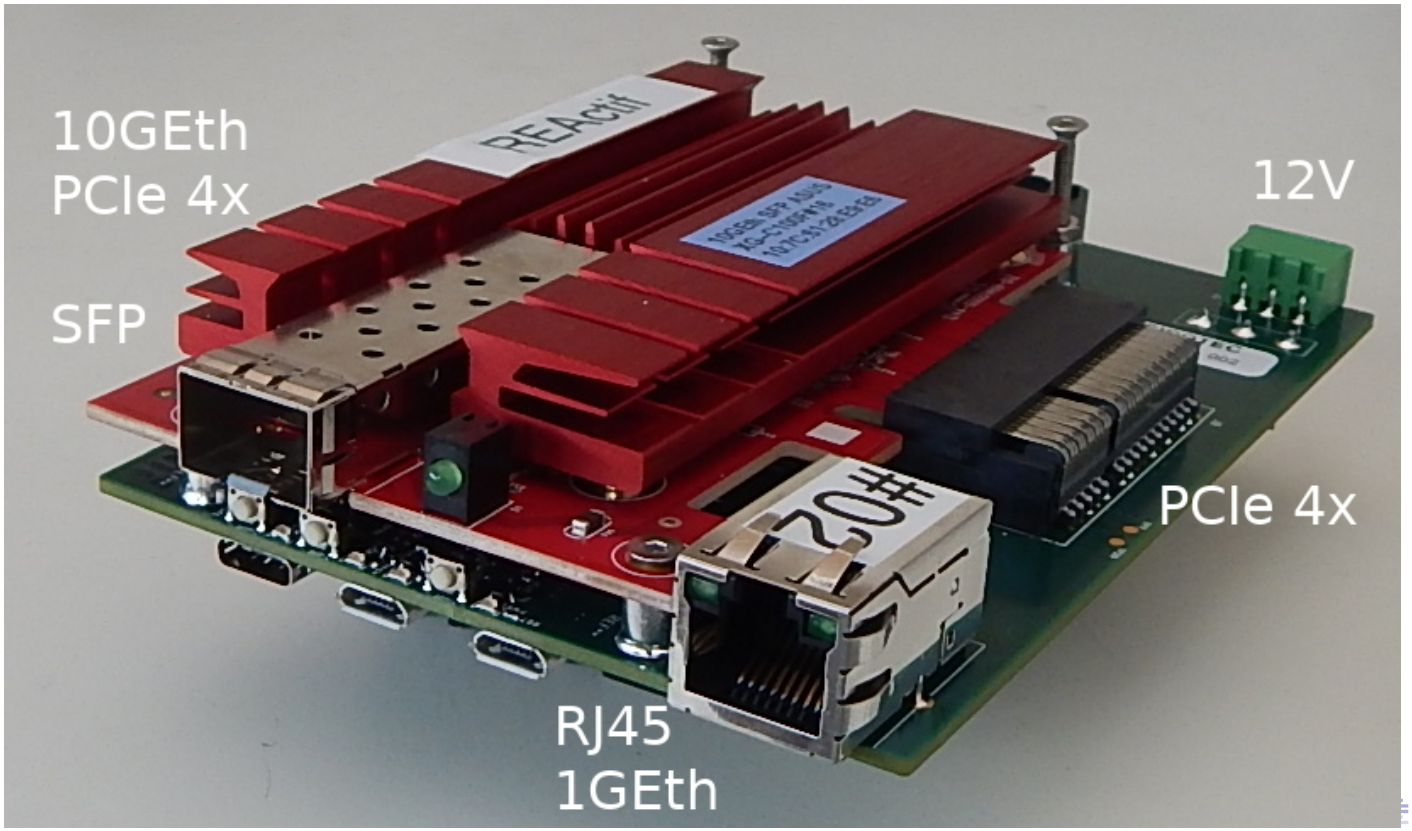
- carte mère open source avec SOM Jetson Nano (en noir)
- carte fille SFP GEthernet PCIe 4x (en rouge)



carte HACKtif

matériels

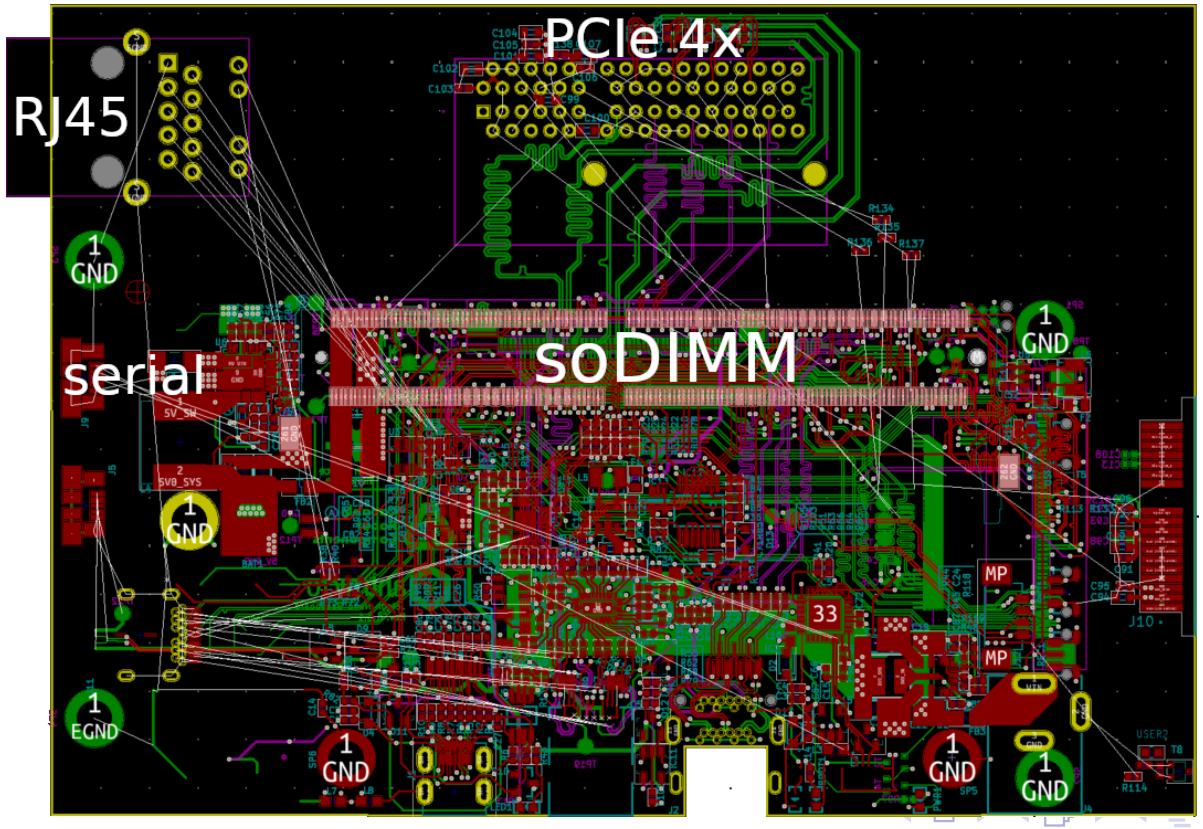
- carte mère HACKtif (en vert) avec SOM Jetson Nano (dessous)
- carte fille 10GEth SFP (en rouge) sur port PCIe 4x, alim. 12V



carte HACKtif developement

HACKtif sous KiCAD

- carte mère embarquant un Jetson NX SOM avec 1GEth RJ45
- carte fille PCIe 4x ($\lhd$ SFP GEthernet)



Jetson Tegra GPU

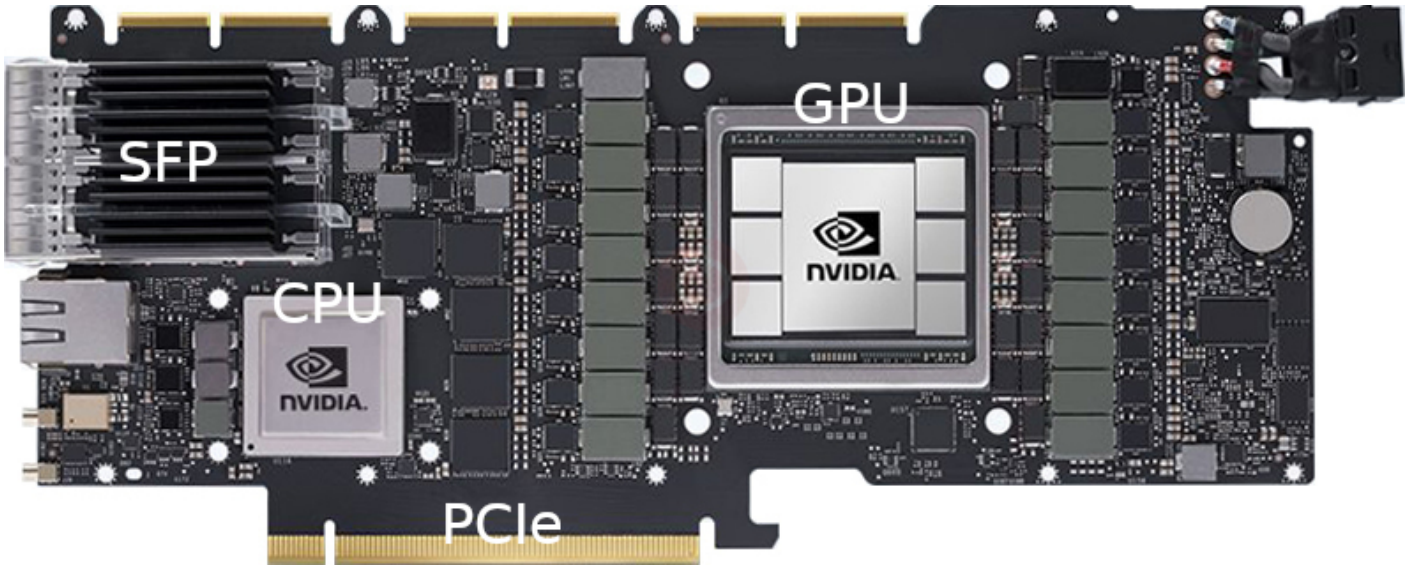
Jetson Nano

- 128 cores Maxwell^2 GPU
- 4x8 cores per block = 32 cores per block
- 4 blocks = 128 cores

NVidia A30X, A100X, AX800

Converged Accelerator

- SFP: Small Form-factor Pluggable (100GEth)
- DPU: CPU+NIC Data Processing Unit (BlueField)
- gpGPU: GeneralPurpose Graphics Processing Unit (Ampere)



Jetson Nano spec. and embedded

Jetson Nano specifications (SOM)

- SoC: Tegra210
- CPU: NVidia ARM Cortex A57, 204MHz to 1.5GHz
- GPU: NVidia GM20B Maxwell 2.0, 76MHz to 921MHz
- RAM: lpDDR4 2/4GB, MemoryBus=64 bit, Bandwidth=25.60 GB/s
- power models: 5W, 10W

Jetson Nano embedded Linux

- L4T: 32.4.4 (JetPack v4.4)
- Linux: Ubuntu18, kernel v4.9, gcc 7.5
- CUDA library v10.2
- CUDA architecture SMS=53 (CUDA v5.3)

Jetson Nano Linux SDK on Ubuntu18

Jetson NX

Tegra — **CPU**	NVidia ARM	cores	freq.
Nano	Cortex A57	4	1.4GHz
TX2 NX	Denver+Cortex A57	6	2.2GHz
xNX	v8.2 Carmel	6	2GHz

Tegra — **GPU**	NVidia	cores	arch.	freq.	CUDA
Nano	GM20B	128	Maxwell 2.0	921MHz	5.3
TX2 NX	GP10B	256	Pascal	1.3GHz	6.2
xNX	GV10B	384	Volta	1.1GHz	7.2

Tegra — **RAM**	size	type	bus	bandwidth
Nano	2/4GB	lpDDR4	64 bit	25 GB/s
TX2 NX	8GB	lpDDR4	128 bit	60 GB/s
xNX	8/16GB	lpDDR4x	128 bit	60 GB/s

Tegra — others	power models	kit price	module price
Nano	5W, 10W	111/166€	/144€
TX2 NX	7.5W, 15W	333€	222€
xNX	10W, 15W, 20W	/555€	/655€

Jetson NX/Orin/AGX

Jetson Nano

- CPU: NVidia ARM Cortex A57, 4cores, $\leq$ 1.4GHz
- GPU: NVidia GM20B Maxwell 2.0, 128 cores, $\leq$ 921MHz, CUDA 5.3
- RAM: lpDDR4 2/4GB, MemoryBus=64 bit, Bandwidth=25 GB/s
- power models: 5W, 10W (kit 111/166€, mod. /144€)

Jetson Orin Nano

- CPU: NVidia ARM v8.2 Cortex A78AE, 6cores, $\leq$ 1.5GHz
- GPU: NVidia GA10B Ampere, 512/1024cores, $\leq$ 625MHz, CUDA 8.6
- RAM: lpDDR5 4/8GB, MemoryBus=128 bit, Bandwidth=70 GB/s
- power models: 7W, 10/15W (kit /456€, mod. 266/388€)

glossaire

generic

- **SOM**: system on a module, i.e. RAM and SoC chips on a board
- **SOC**: system on a chip, i.e. CPU+GPU on single chip
- **FPGA**: Field-Programable Gate Array , **RAM**: Rnd-Access Memory
- **GPU**: Graphics Processing Unit , **CPU**: Central Processing Unit
- **SFP**: Small Form-factor Plug. , **NIC**: Network Interface Controller

NVidia

- **DPU**: Data Processing Unit (NIC+CPU+accelerator)
- *kernel*: processing code for GPU (CUDA: __global__, __device__)
- *thread*: run independent kernel on GPU core
- *thread block*: communication group via shared memory, sync., ...
- *stream*: handle for instruction queue for processing data
- **SM**: Streaming Multiprocessors execute several thread blocks in //

glossaire

NVidia

- **i/dGPU**: **i**ntegrated/**d**iscrete Graphics Processing Unit
- **SMS**: Makefile variable for target arch. (Streaming Multiprocessor**s**)
- **CUDA**: Compute Unified Device Architecture
- **Tegra**: embedded hardware line (i.e. ARM/iGPU vs i64/dGPU line)

NVidia CUDA

- *kernel*: processing code for GPU (CUDA: __global__, __device__)
- *thread*: run independent kernel on GPU core
- *warp*: thread group executing the same instruction
- *thread block*: communication group via shared memory, sync., ...
- *stream*: handle for instruction queue for processing data
- **SM**: Streaming Multiprocessors execute several thread blocks in //