

traitement du signal sur GPU REActif

Sébastien COUDERT,
Maria BLAIZOT, Patrice BOURGAULT, Gabriel ROBERTO-LIMA

CNRS/IN2P3/GANIL/DPHY/GT**Acquisition**

19 nov. 2025

1 HACKtif

- carte HACKtif
- Jetson (NVidia)

2 TAsTout

- chaine de test avec tatouage de signal
- tattoo library: stéganographie

3 REAction

- code informatique
- boucle sur le signal
- code de traitement du signal
- mesure de performance

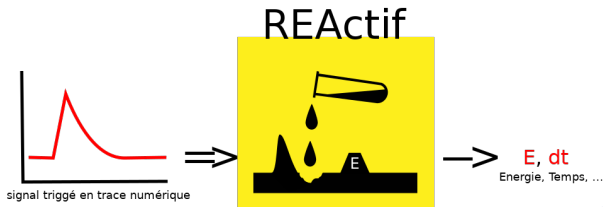
4 performances, conclusion et perspectives

- performance de traitement du signal
- performance de consommation énergétique
- conclusion et perspectives

REActif: traitement informatique du signal

REActif

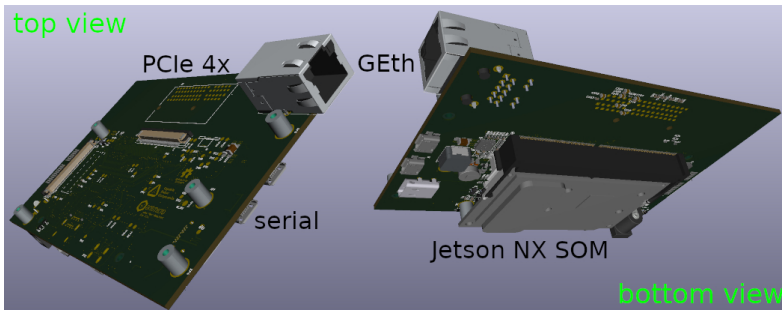
- réaliser le traitement du signal de manière **+souple** et **+rapide** dans un co-processeur facilement programmable (GPU) vs FPGA
- traitement du signal triggé en provenance du numériseur sur co-processeur en langage informatique (C++)
- plateforme embarquée afin d'être **+proche** du numériseur avec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



carte HACKtif (∠ Patrice BOURGAULT)

carte mère open source (GANIL/DPHY/GTA)

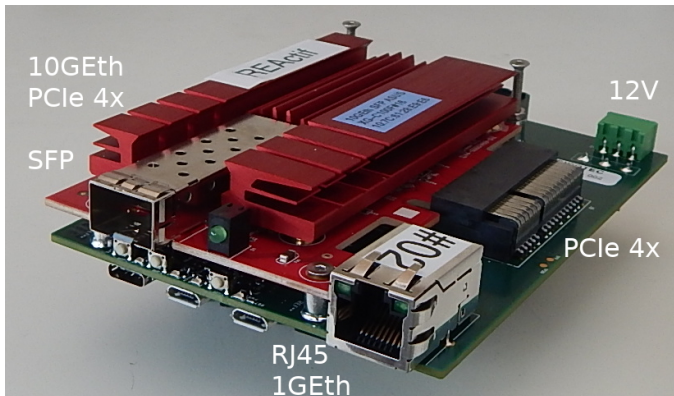
- dim. 120x80mm pour chassis NIM, μ TCA, ..., table.
- dessin sous KiCAD (projet open source)
- connecteur SODIMM pour SOM Jetson NX (Nano, TX2 ou xNX)
- connecteur PCIe 4x (à plat) épaisseur 20mm



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



carte HACKtif avec 1 port PCIe 4x

exemples de cartes filles Ethernet commerciales
10GEth SFP/RJ45, 2.5GEth RJ45, 4x 1GEth RJ45



mesure de débit Ethernet

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

Jetson SOM: CPU+GPU+RAM

NVidia PCIe

- carte graphique=GPU NVidia + RAM



Jetson SOM

- Tegra=CPU+GPU=ARM+NVidia

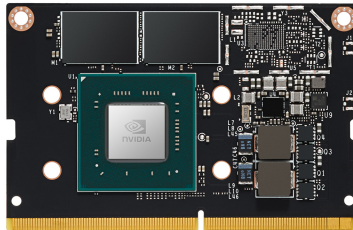
Jetson SOM: CPU+GPU+RAM

NVidia PCIe

- carte graphique=GPU NVidia + RAM

Jetson SOM

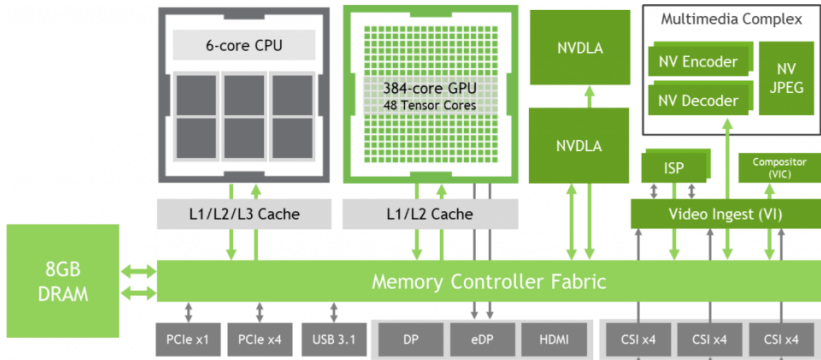
- Tegra=CPU+GPU=ARM+NVidia
- DDR RAM
- NX SOM: soDIMM Nano/TX2NX/XavierNX



Jetson SOM: CPU+GPU+RAM

Tegra ARM (Nano/xNX)

4/6 cores CPU, ARM A57/V8, 1.4/1.9 GHz, 4/8 GB IpDDR4, 64/128bit memory bus



Jetson SOM: CPU+GPU+RAM

Tegra GPU (Nano/xNX)

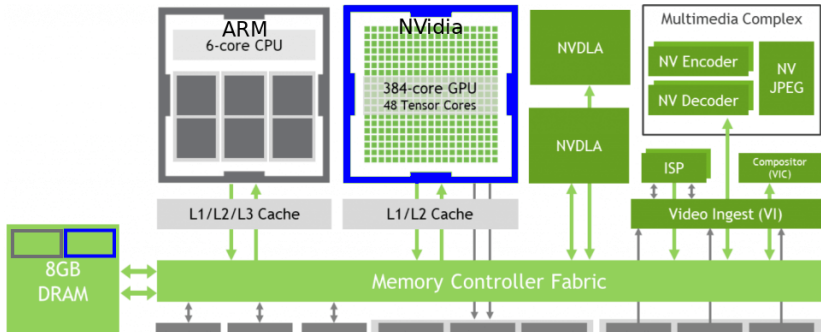
128/384 cores GPU, NVidia Maxwell/Volta, 900/1100 MHz^{boost}, DDR partagée



Tegra CPU+GPU

données entre CPU/GPU

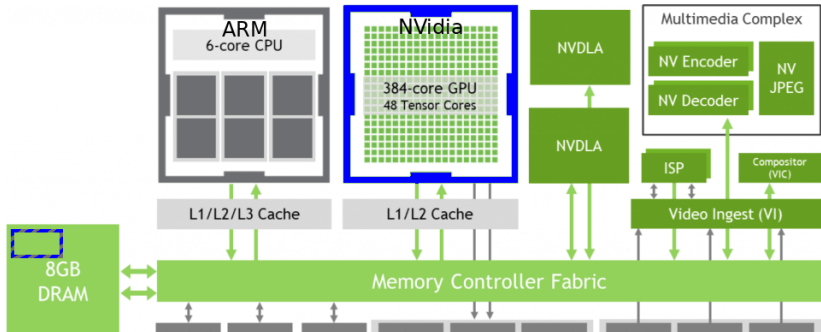
- mémoire partagée, mais zones mémoires réservées
- procédure **MemCopy**: copie des données d'entrée, traitement du signal, copie des données de sortie



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



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€

TAsTout pour la chaîne de test: simulation, transfert, traitement du signal, vérification

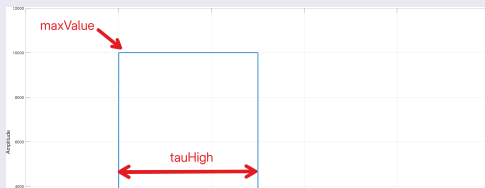
chaîne de production



chaîne de test



signal simulé: ex. impulsion à 4 paramètres

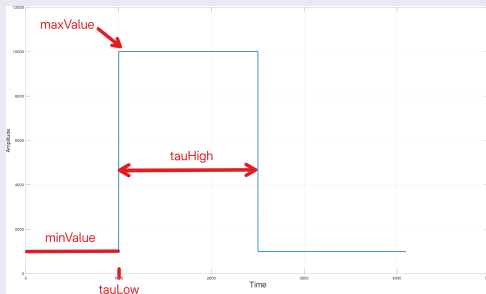


TAsTout pour la chaîne de test: simulation, transfert, traitement du signal, vérification

chaîne de test



signal simulé: ex. impulsion à 4 paramètres



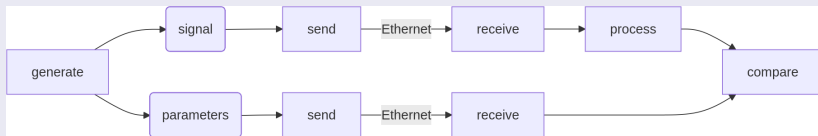
TAsTout pour la chaîne de test: simulation, transfert, traitement du signal, vérification

chaîne de test



signal simulé: ex. impulsion à 4 paramètres

banc de test: éventualité d'ajout d'un lien ethernet



banc de test avec TAsTout = tout intégré sans modification

TAsTout pour la chaîne de test: simulation, transfert, traitement du signal, vérification

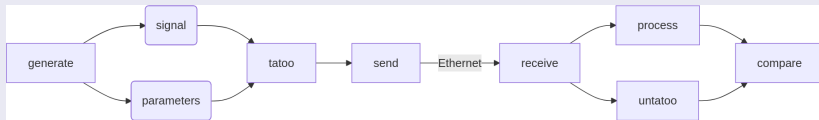
chaîne de test



signal simulé: ex. impulsion à 4 paramètres

banc de test: éventualité d'ajout d'un lien ethernet

banc de test avec TAsTout = tout intégré sans modification



lib. TAsTout: tatouage de paramètres dans un signal bruité

tatouage

- paramètres dans le bit de poids faible des échantillons du signal
- bruit extrêmement faible ± 1 digit

format: TAsTout V1 16b 4i 000A 000B 000C 000D

TAsTout: stéganographie de signal bruité

stéganographie^{@WikiPedia}

- faire passer un message caché
- message = "TAsTout V1 16b 4i 000A 000B 000C 000D"

file signature: TAsTout

version et échantillonnage: V1 16b

lib. TAsTout: tatouage de paramètres dans un signal bruité

tatouage

format: TAsTout V1 16b 4i 000A 000B 000C 000D

TAsTout: stéganographie de signal bruité

file signature: TAsTout

file signatures^{@WikiPedia}

- %00PNG
- %00ELF

image: PNG format (Portable Network Graphics)

```
head -c 4 TAsTout.png;echo
```

```
%00PNG
```

```
file TAsTout.png
```

```
TAsTout.png: PNG image data, 672 x 598, 8-bit/color RGBA,
```

lib. TAsTout: tatouage de paramètres dans un signal bruité

tatouage

format: TAsTout V1 16b 4i 000A 000B 000C 000D

TAsTout: stéganographie de signal bruité

file signature: TAsTout

file signatures^{@WikiPedia}

- %00PNG
- %00ELF

linux binary: ELF format (Executable and Linkable Format)

```
head -c 4 tastout;echo
```

%00ELF

file tastout

tastout: ELF 64-bit LSB pie executable, x86-64, version 1 (GNU/Linux),

lib. TAsTout: tatouage de paramètres dans un signal bruité

tatouage

format: TAsTout V1 16b 4i 000A 000B 000C 000D

TAsTout: stéganographie de signal bruité

file signature: TAsTout

file signatures^{@WikiPedia}

- %00PNG
- %00ELF
- TAsTout

version et echantillonnage: V1 16b

parametres: 4i 000A 000B 000C 000D

lib. TAsTout: tatouage de paramètres dans un signal bruité

tatouage

format: TAsTout V1 16b 4i 000A 000B 000C 000D

TAsTout: stéganographie de signal bruité

file signature: TAsTout

version et échantillonnage: V1 16b

- version du format de la librairie TAsTout
- échantillonnage du signal, ici 16bit

parametres: 4i 000A 000B 000C 000D

lib. TAsTout: tatouage de paramètres dans un signal bruité

tatouage

format: TAsTout V1 16b 4i 000A 000B 000C 000D

TAsTout: stéganographie de signal bruité

file signature: TAsTout

version et échantillonnage: V1 16b

parametres: 4i 000A 000B 000C 000D

- nombre et type: 4 parametres de type entier
- valeurs des parametres en hexadecimal: min, time low, max, time high

procédure sur plateforme hétérogène

couple CPU+GPU

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

Le CPU ordonne l'enchaînement 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/accélérateur)
- GPU: traitement du signal

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

exemple CUDA code: copy → kernel → copy

```
float h_p[size], *d_p ... //cudaMalloc  
cudaMemcpyAsync(d_p, h_p, size, cudaMemcpyHostToDevice, str...  
cudaStreamSynchronize(stream);  
kernel0<<<blocksPGrid, threadsPBlock, 0, stream>>>(d_p,size);  
cudaMemcpyAsync(h_p, d_p, size, cudaMemcpyDeviceToHost, str...  
cudaStreamSynchronize(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/accélérateur)
- GPU: traitement du signal

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

exemple 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 du temps d'exécution

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

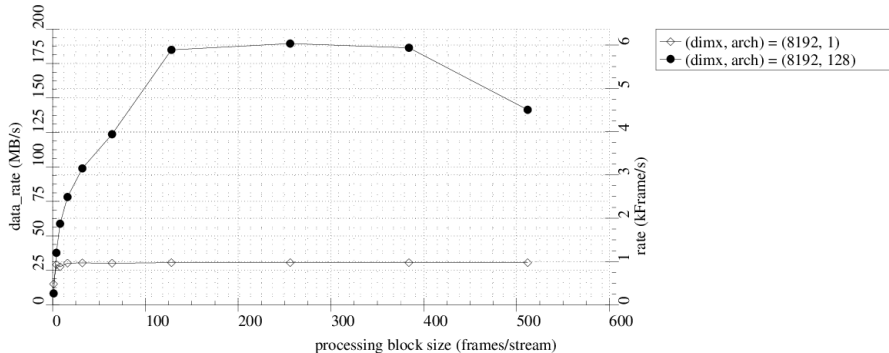
CPU: elapsed time

```
cudaEvent_t start, stop; cudaEventCreate(&start...; float msecTotal=0.0f;  
cudaEventRecord(start, stream);  
run_kernel(algo, blocksPerGrid, threadsPerBlock, ...  
cudaEventRecord(stop, stream);  
cudaEventSynchronize(stop);  
cudaEventElapsedTime(&msecTotal, start, stop);
```

performance de traitement du signal

calcul d'énergie sur 128cores: Jetson Nano

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

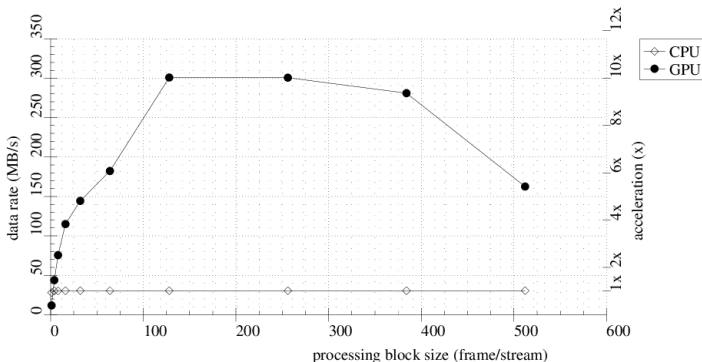


data rate (and frame rate) vs. 8kS frames per stream (1 str. on 128 cores in mode 10W) arch.=Nano

performance de traitement du signal

calcul d'énergie sur 128cores: Jetson Nano

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

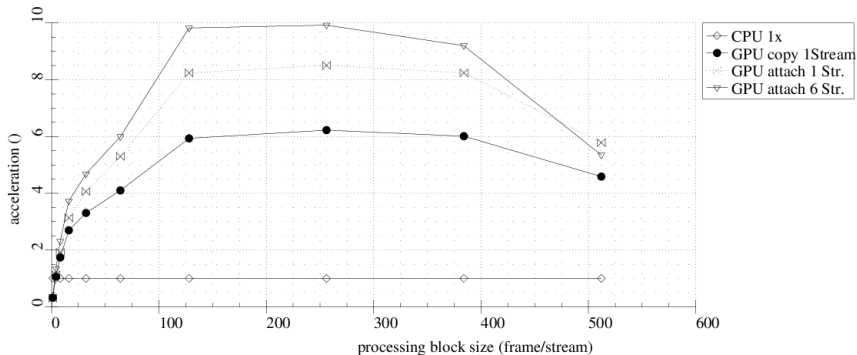


data rate vs. number of frames per Stream and acceleration of GPU regarding to CPU (8kS frame)

performance de traitement du signal

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

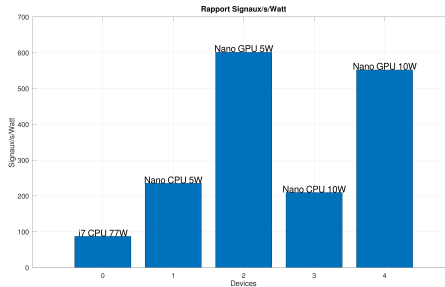
attach+stream perf.: 10x , max.: $6x \Rightarrow 8x \Rightarrow 10x$ (16kBoF=8kS, 128 GPU cores)



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

performance énergétique

cREAtion — buffer=256 — signal=32kB — exécution parallèle



performances

- **CPU+GPU(5W) :**
600 signaux/s/Watt
- **CPU+GPU(10W) :**
550 signaux/s/Watt
- **3x moins d'énergie**
Nano/GPU vs Nano/CPU
- **6x moins d'énergie**
Nano/GPU vs i7

conclusion et perspectives

REActif

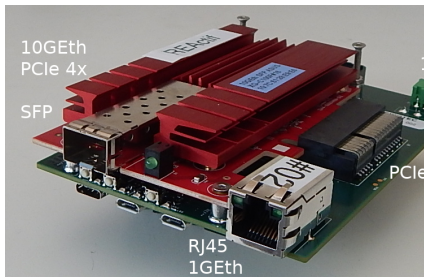
- traitement du signal en langage simple, souple et REActif
- création d'un système complet:
 - matériel: HACKtif
 - logiciel: REAction
- performances convaincantes en parallélisation naturelle
- gains en prix, énergie et encombrement
- en test, vérification de l'algorithme en ligne (tatouage)

REActif

- optimisations , par ex. structure des données vs répartition par coeurs de calcul , mais garder la simplicité de codage
- implémentation d'autres traitements de signal

REActif/HACKtif (GANIL)

- 10GEth
- Tegra
- Linux



annexes

5 annexes

- révolutions électronique/informatique
 - NEGMA
 - REActif
- REAction
- performances REActif
 - JetsonNano
 - JetsonNano/XavierNX
- HACKtif
 - carte HACKtif assemblée (v0)
 - carte HACKtif
- divers
 - Jetson Nano
 - NVidia Converged Accelerator
 - Jetson Nano spec.
 - glossaire

RF-SOC révolution électronique

ADC intégré au SOC

- ADC+DAC intégrés au FPGA



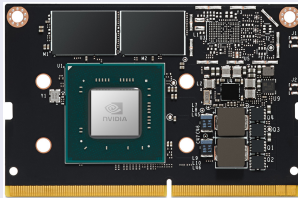
glossaire

- **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

Logiciel REAction

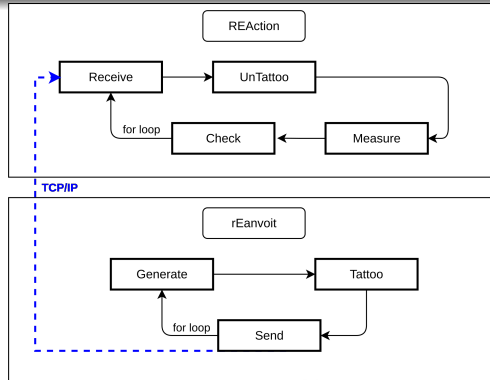
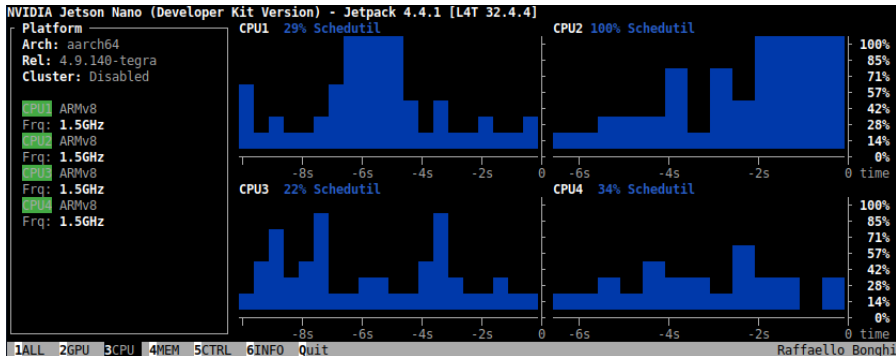


Figure: Étapes des programmes de génération et d'envoi (*rEnvoit*) en lien avec le programme de réception et traitement de signal (*REAction*)

⁰le tatouage permet de transporter (de façon transparente) les paramétrés du signal généré dans le programme *rEnvoit* dans l'autre programme *REAction* via Ethernet.

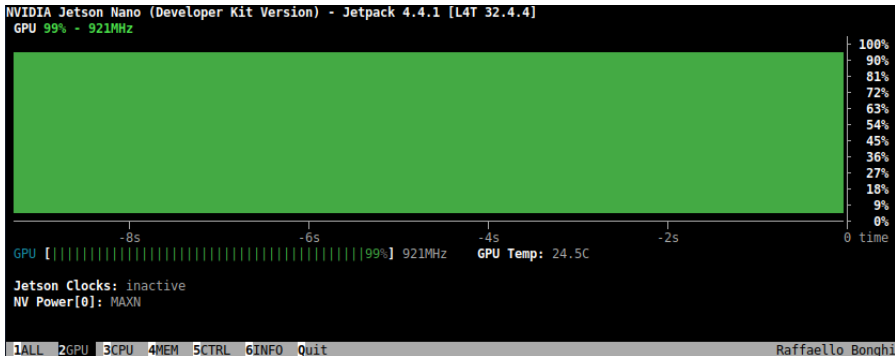
charge pendant le traitement du signal

CPU



charge pendant le traitement du signal

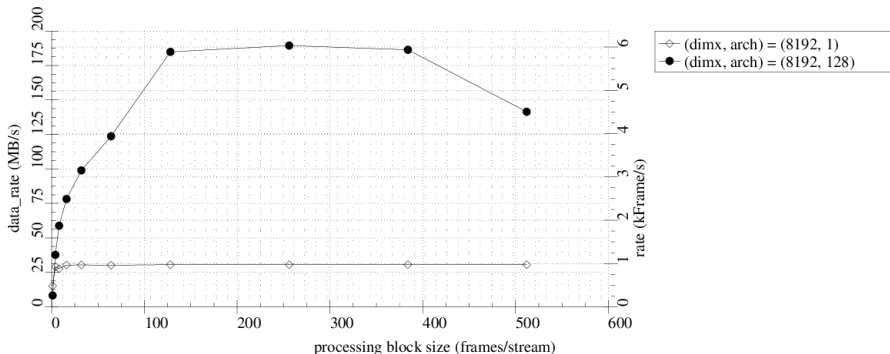
GPU



performance de traitement du signal

calcul d'énergie sur 128cores: Jetson Nano

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

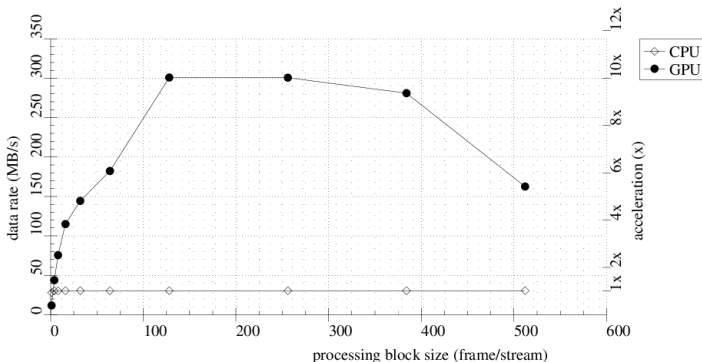


data rate (and frame rate) vs. 8kS frames per stream (1 str. on 128 cores in mode 10W) arch.=Nano

performance de traitement du signal

calcul d'énergie sur 128cores: Jetson Nano

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

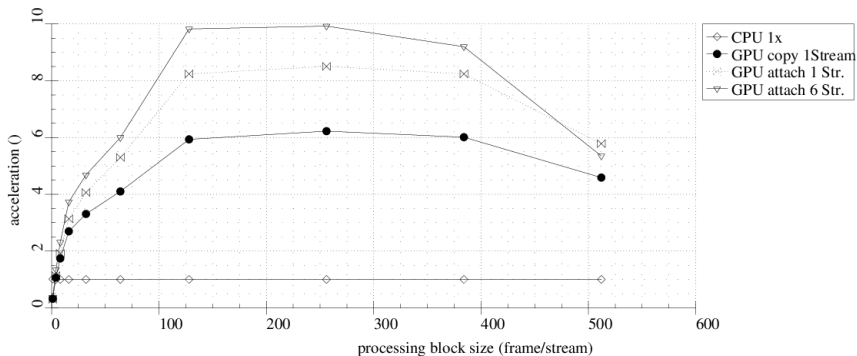


data rate vs. number of frames per Stream and acceleration of GPU regarding to CPU (8kS frame)

performance de traitement du signal

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

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

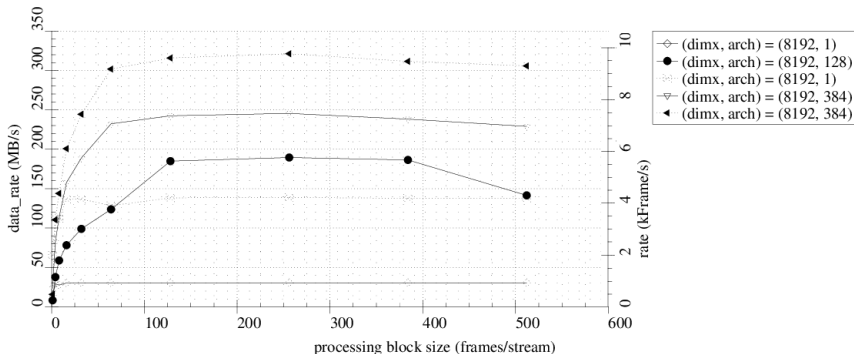


acceleration of GPU regarding to CPU vs. number of frames per Stream (8kS frame)

performance JetsonNano/XavierNX

calcul d'énergie: Jetson xNX

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

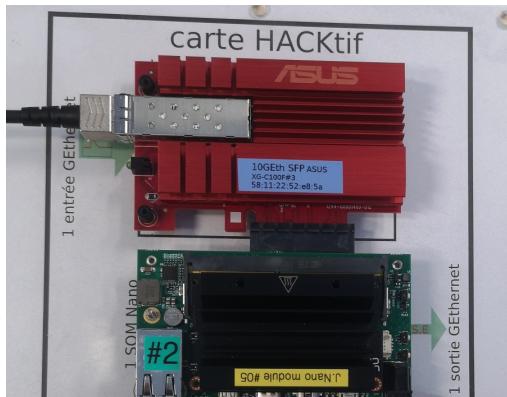


data rate vs. 8kS frames per stream (1 str. on 128/384 cores in mode 10W/15W/20W) arch.=Nano/xNX

carte HACKtif assemblée (v0)

matériels principaux du proto. assemblé

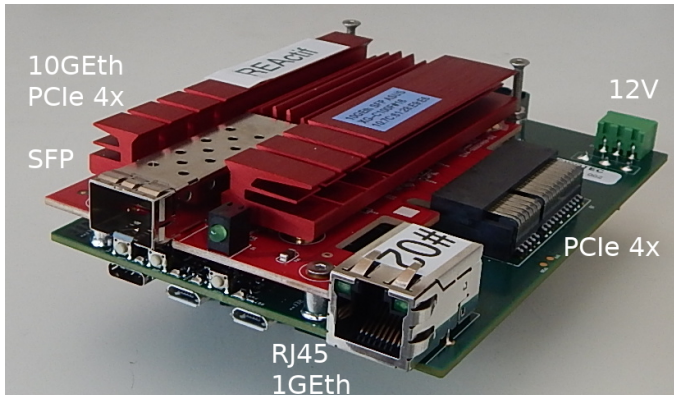
- carte mère open source avec SOM Jetson Nano (en noir)
- carte fille SFP GEthernet PCIe 4x (en rouge)



carte HACKtif

matériels

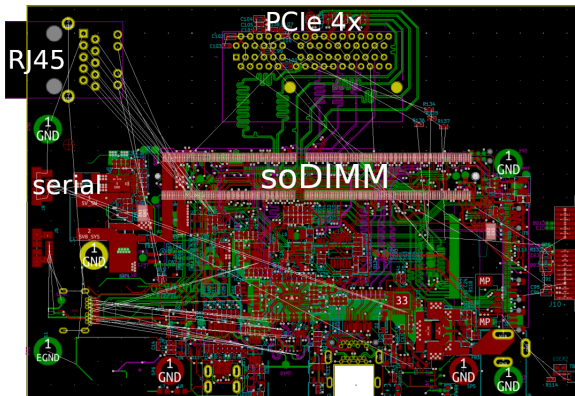
- carte mère HACKtif (en vert) avec SOM Jetson Nano (dessous)
- carte fille 10GEth SFP (en rouge) sur port PCIe 4x, alim. 12V



carte HACKtif developement

HACKtif sous KiCAD

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



Jetson Tegra GPU

Jetson Nano

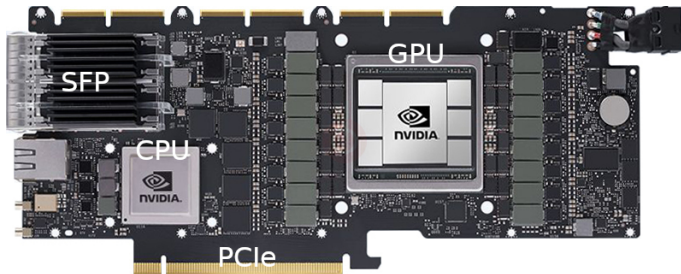
- 128 cores Maxwell² GPU
- 4x8 cores per block = 32 cores per block
- 4 blocks = 128 cores



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€

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**: integrated/**d**iscrete Graphics Processing Unit
- **SMS**: Makefile variable for target arch. (Streaming Multiprocessors)
- **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 //