traitement du signal sur GPU REActif

Sébastien COUDERT

.

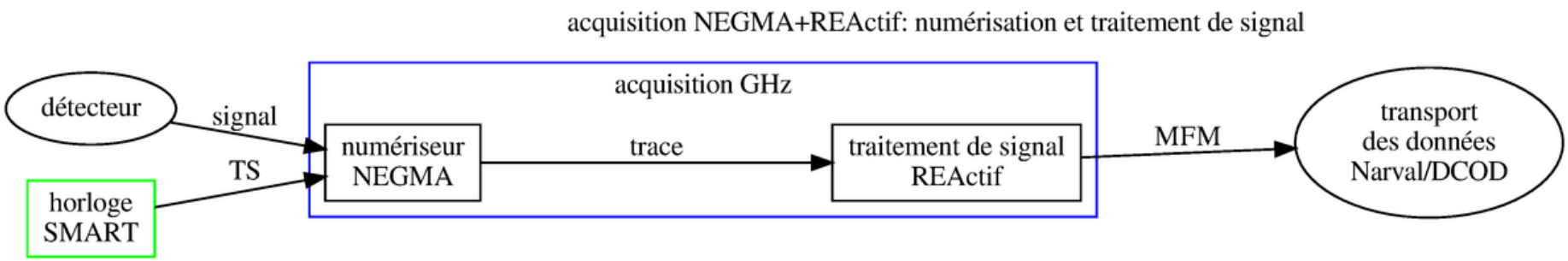
CNRS/IN2P3/GANIL/DPHY/GT**Acquisition**

14 juin 2023

flux d'acquisition

NEGMA + REActif

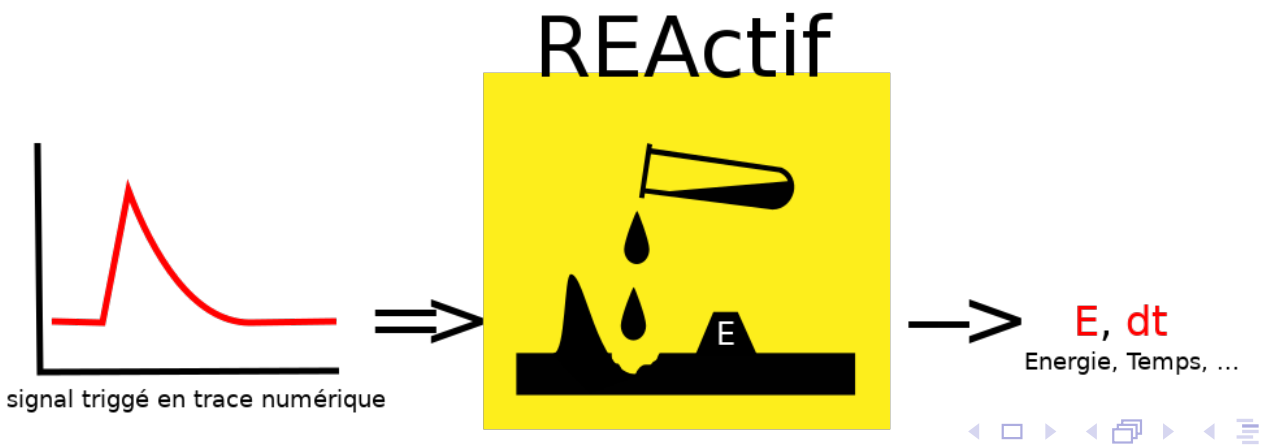
- NEGMA: numérisation + horodatage
- REActif: traitement du signal



REActif: traitement informatique du signal

REActif

- réaliser le traitement du signal de manière +souple et +rapide dans un co-processeur facilement programmable
- 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 NEGMA => REACtif en GEthernet (16kB MFM)

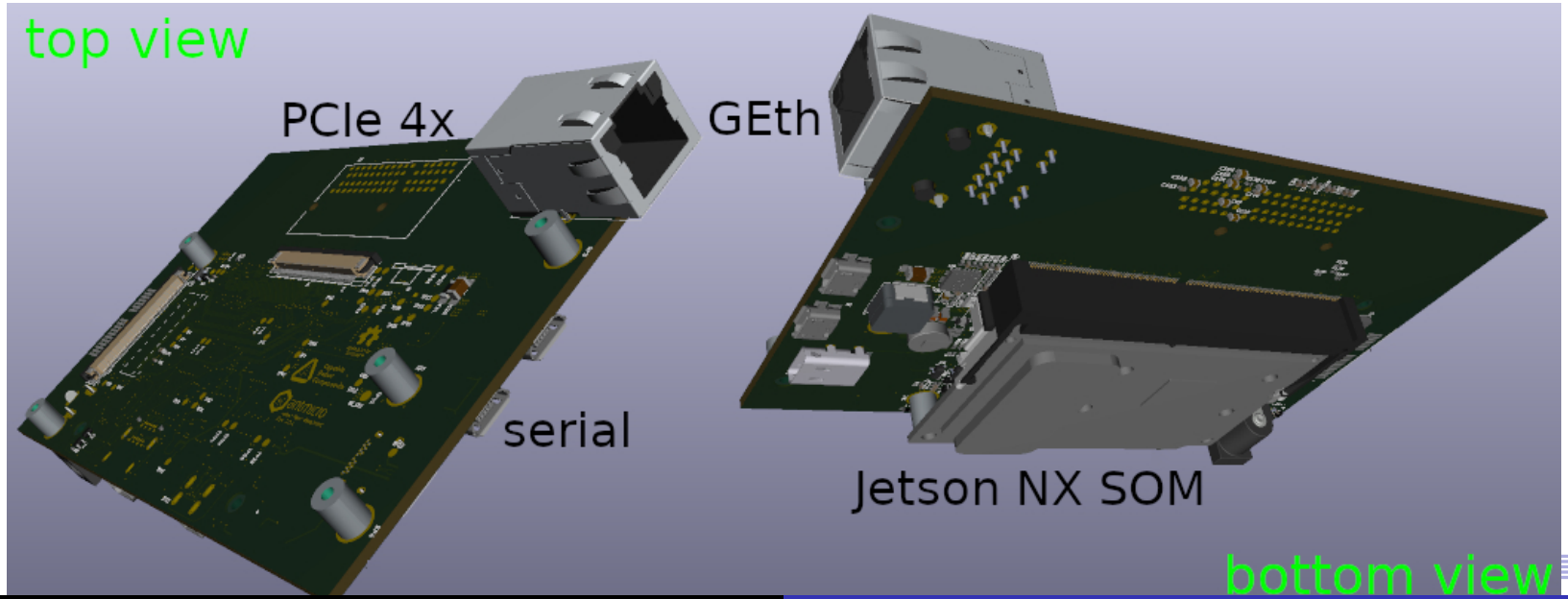


carte HACKtif (◿ Patrice BOURGAULT)

dim. 120x85mm pour les chassis NIM, μTCA, ... ou sur table.

matériel

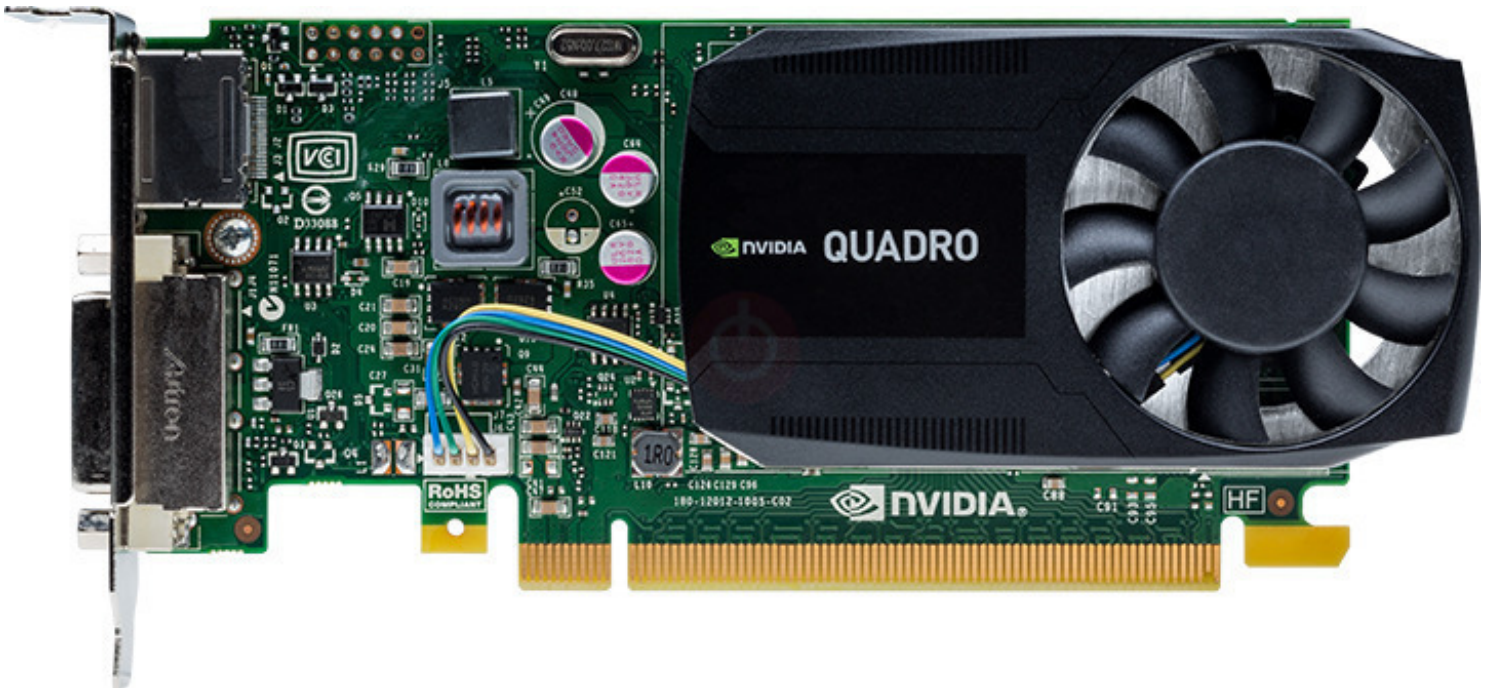
- carte mère open source (GANIL/DPHY/GTA)
  - dessin sous KiCAD (projet open source)
  - SOM Jetson NX sur connecteur SODIMM: Nano, TX2 ou xNX
- carte fille GEthernet PCIe 4x



Jetson SOM: CPU+GPU+RAM

NVidia PCIe

- carte graphique=GPU NVidia + RAM



Jetson SOM

- Tegra=CPU+GPU=ARM+NVidia

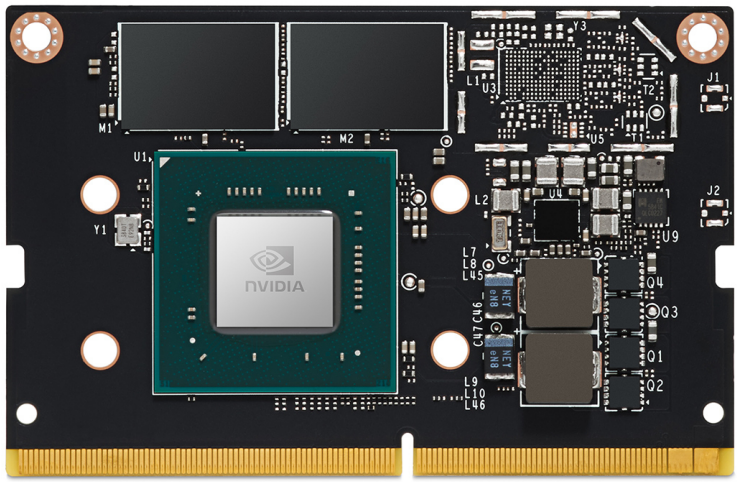
Jetson SOM: CPU+GPU+RAM

NVidia PCIe

- carte graphique=GPU NVidia + RAM

Jetson SOM

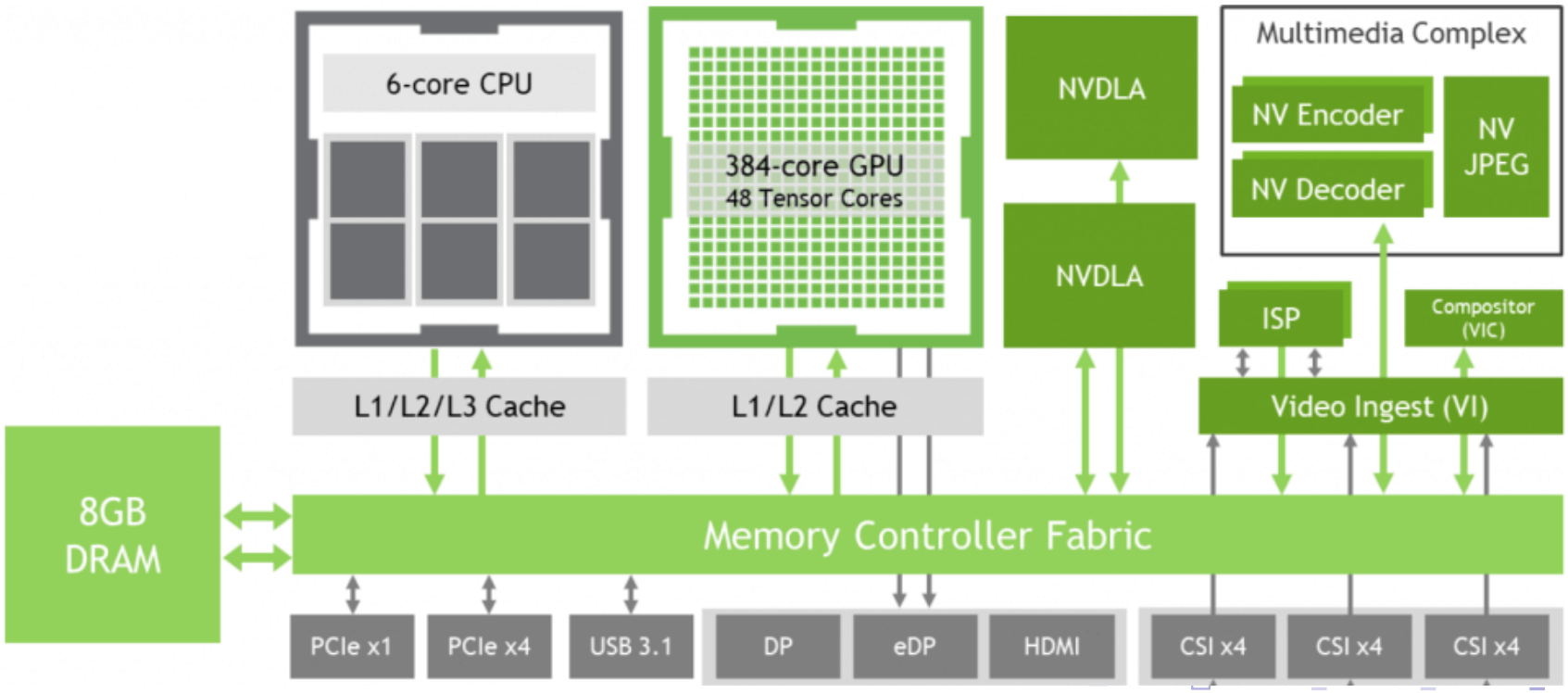
- Tegra=CPU+GPU=ARM+NVidia
- DDR RAM
- NX SOM: soDIMM Nano/TX2NX/XavierNX



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

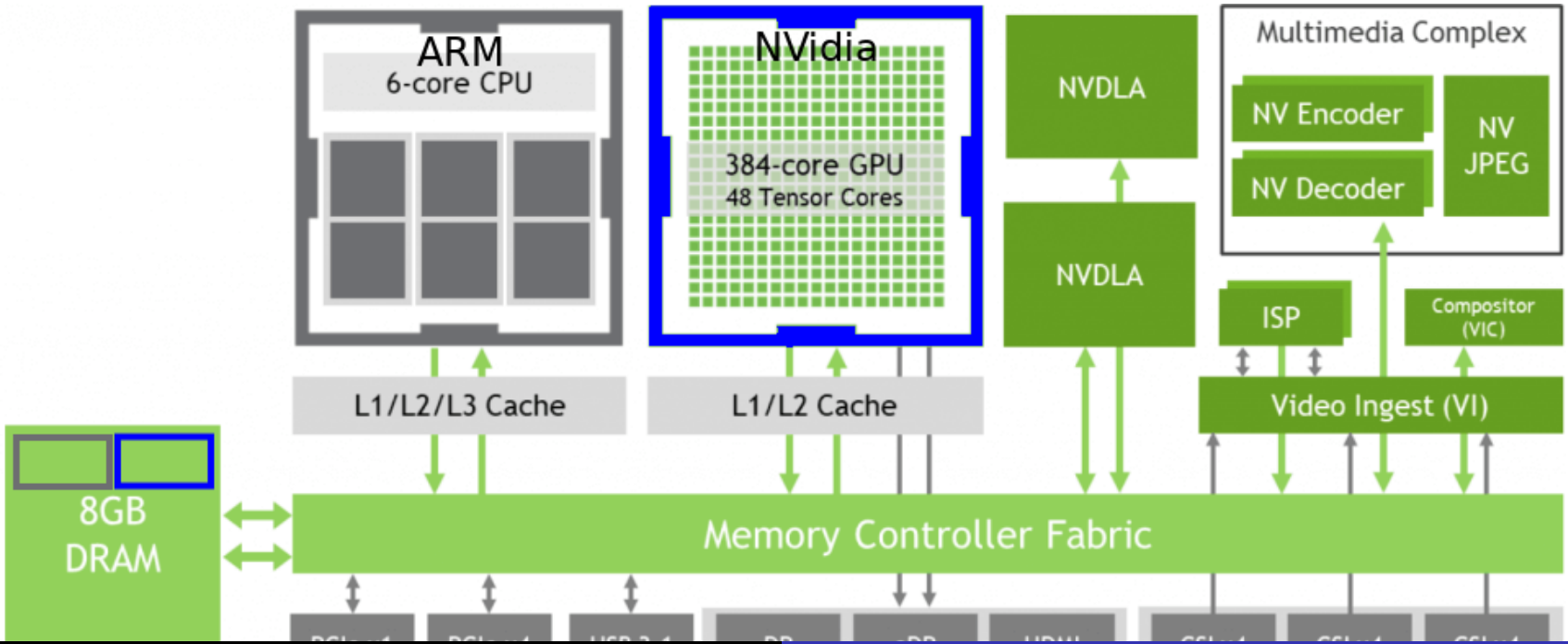
Tegra GPU (Nano/xNX)

128/384 cores GPU, NVidia Maxwell/Volta, 600/900 MHz, DDR partagée

Tegra CPU+GPU

données entre CPU/GPU

- mémoire partagée, mais zones mémoires réservées
- procédure: copie des données d'entrée, traitement du signal, copie 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]**; ...
cuda**Memcpy**Async(d_p, h_p, size, cudaMemcpy**HostToDevice**, str...
cudaStream**Synchronize**(stream);
kernel0<<<blocksPGrid, threadsPBlock, 0, stream>>>(d_Pixel,**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**.

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_Pixel,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: $\Delta t = t_{end} - t_{start}$

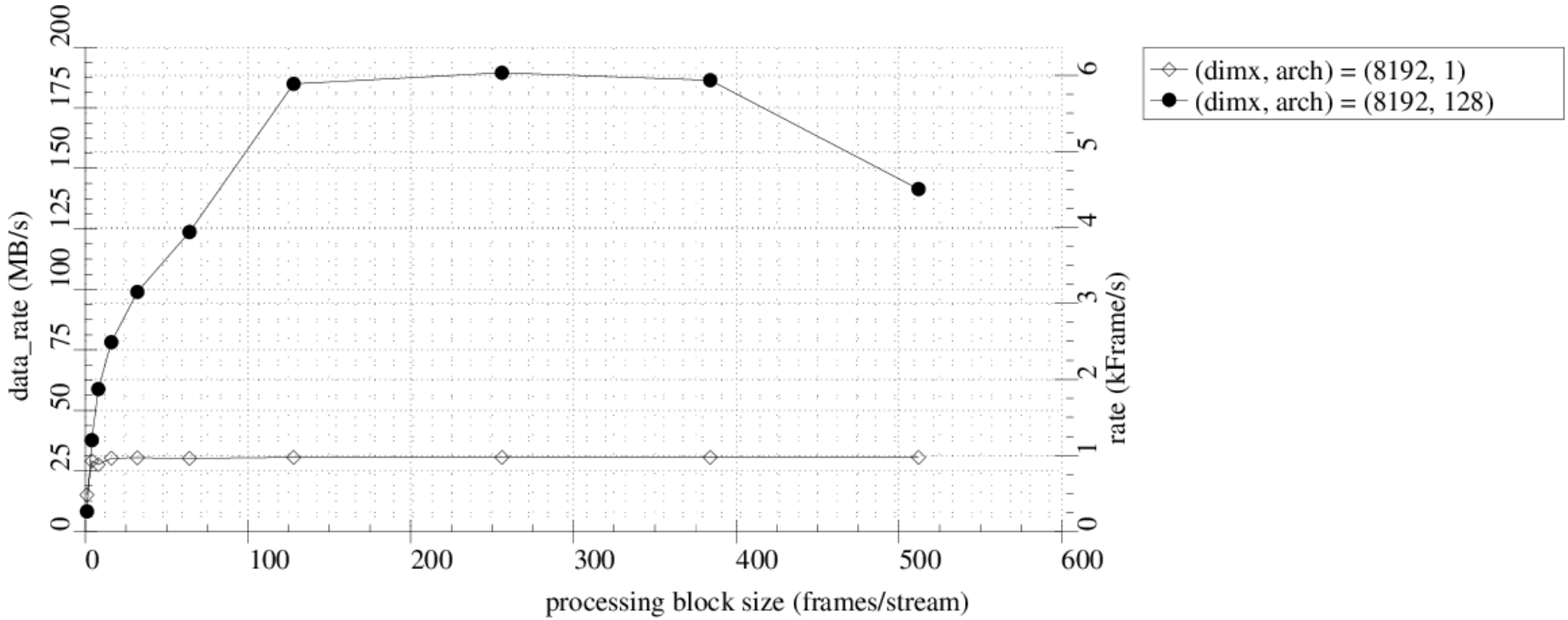
CPU: elapsed time

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

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

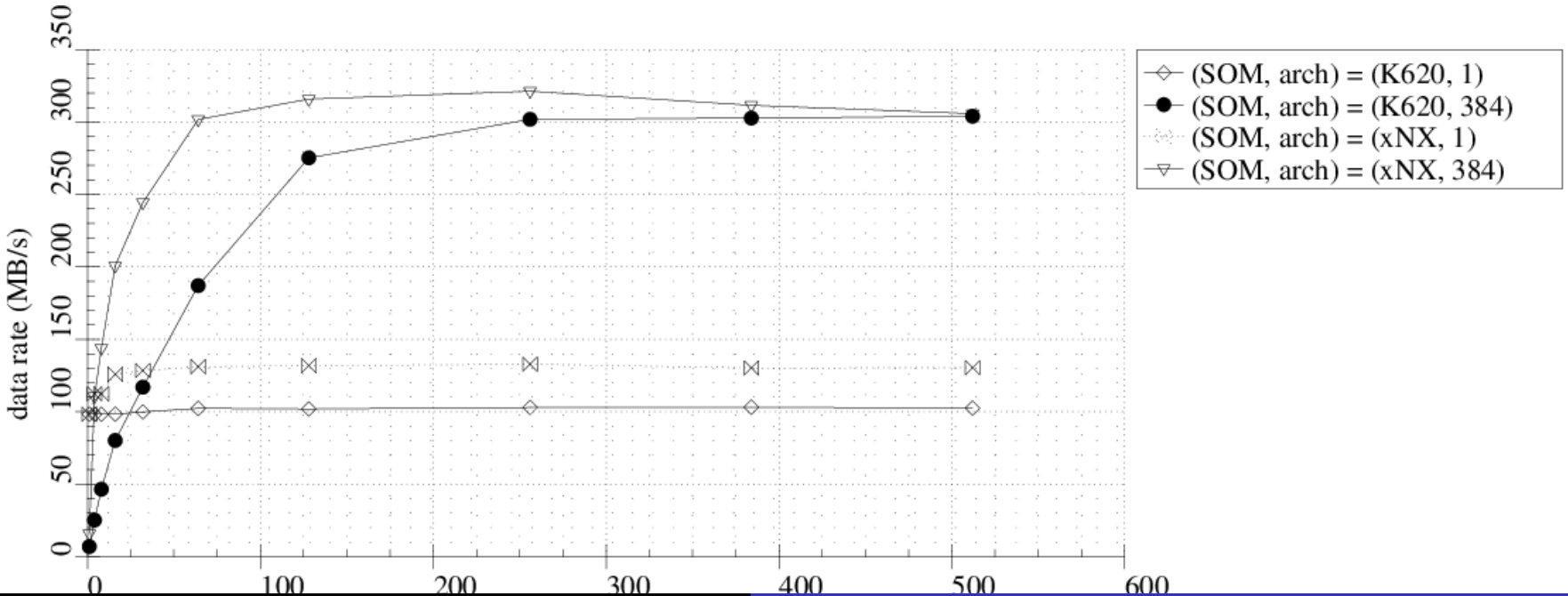


data rate (and frame rate) vs. 8kS frames per stream (1 str. on 128 cores in mode 10W) arch.=Nano

performance 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)



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