AVANTIX

Avancement Simulateur IQ

AVANTIX

Temps contraint : SXX

10/01/2024

Expérimentations

Paramètres fixes :

- fs = 4GSps pour 3GHz de bande utile
- NPERSEGS : 128, 256, 512, 1024, 2048, 8192, 32768, 131072
- IMG_WIDTH = 256
- IMG_HEIGHT = 128
- 16 842 752 échantillons (4,211ms) → Donne SXX à IMG_WIDTH colonnes pour nperseg=131075

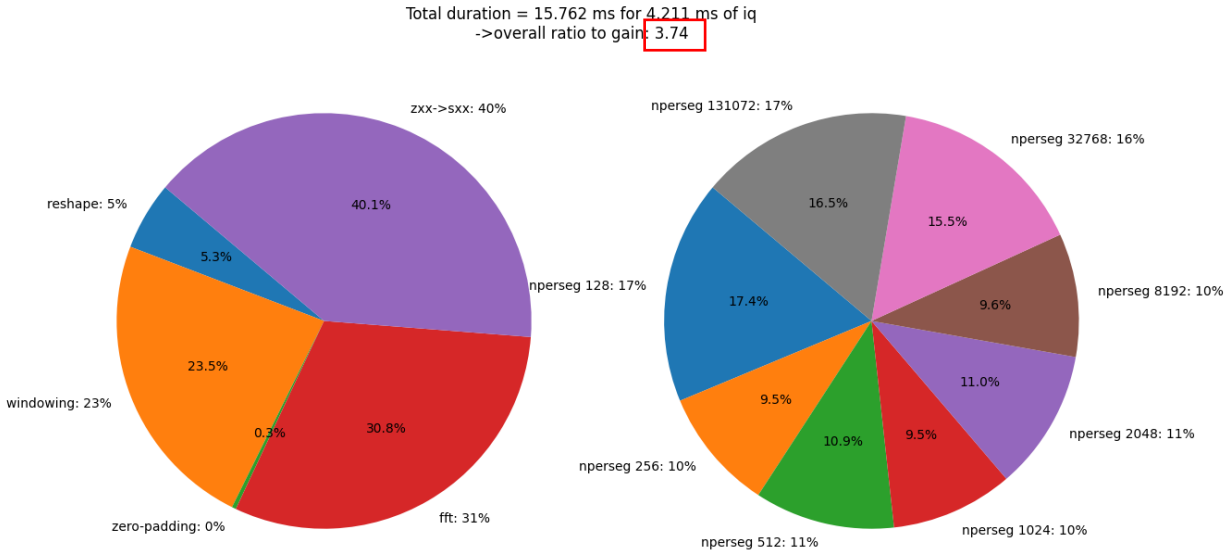
Script : CBE/_temps_contraint/sxx/

Rappel du calcul des spectrogrammes

```
def sxx(iq, window):
     # Reshape en temps
     iq = iq.unfold(0, nperseg, nperseg – noverlap).T
     # Windowing par tukey (alpha = 0,25)
     iq = iq * window[:, None]
     # FFT
     zxx = torch.fft.fft(iq)
     # SXX
     sxx = abs(zxx)
     return sxx

def compute_all_sxxs(NPERSEGS):
    sxxs_lin = {}
    for nperseg in NPERSEGS:
           sxxs_lin[nperseg] = sxx(iq, window)
    return sxxs_lin
```

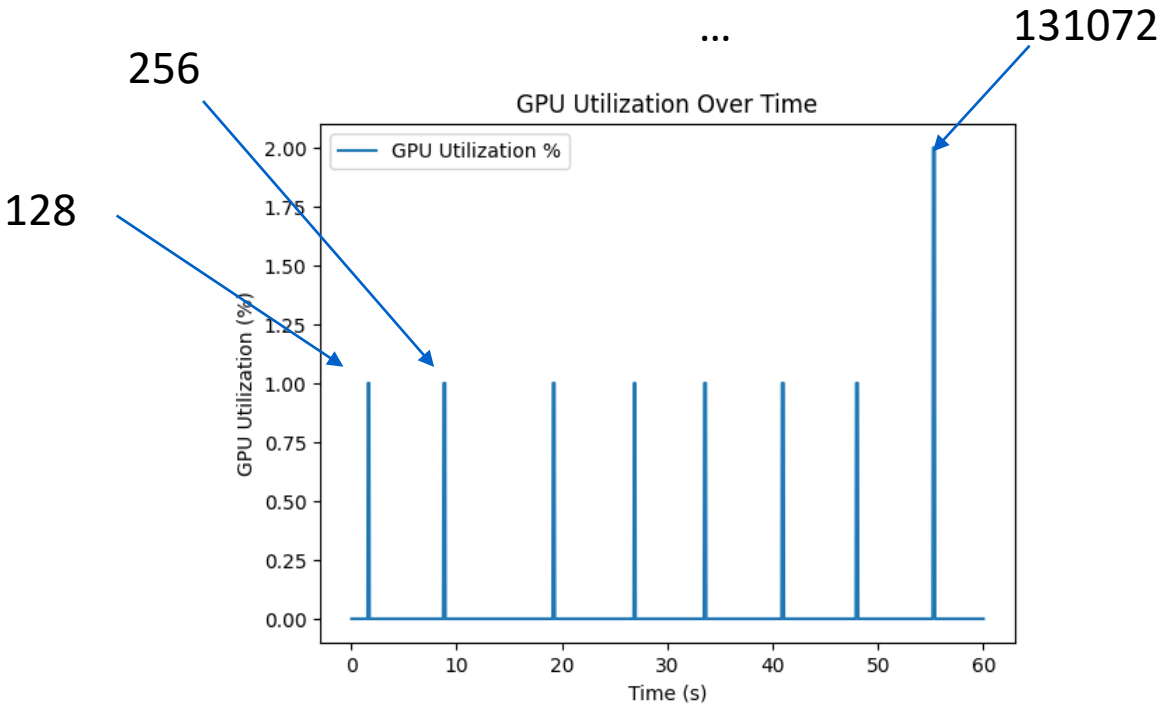
Durée SXX



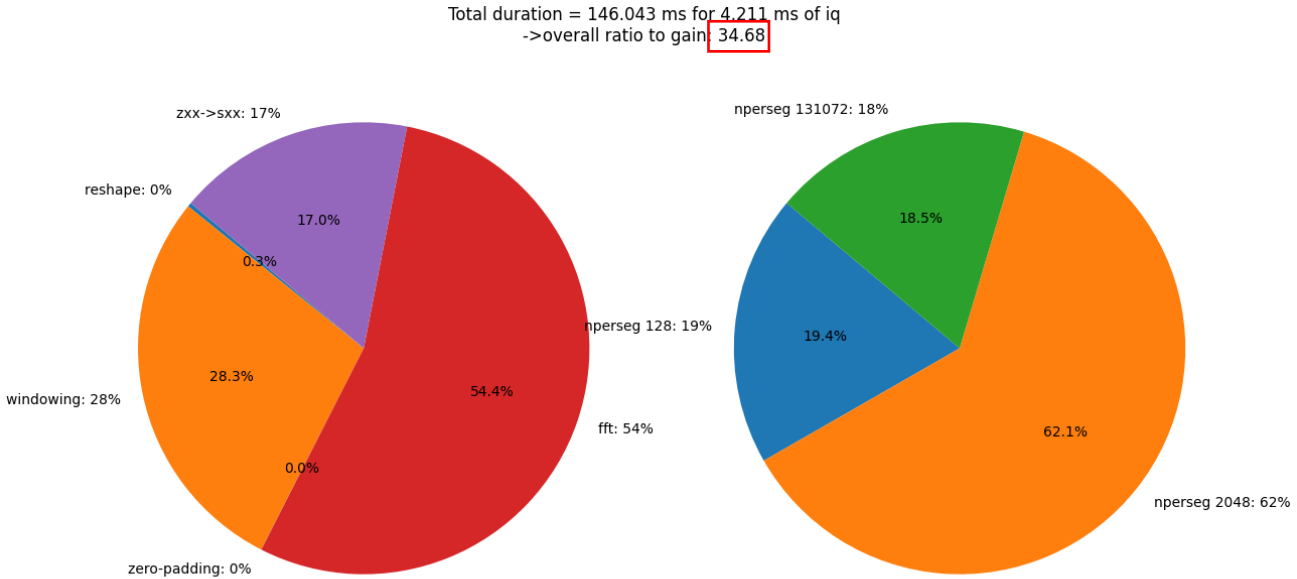
Temps réel atteignable en passant à la H100 ?

D'après les specs en FP16 il y aurait un gain x7 en passant à la H100

Peut-on paralléliser ?



Méthode 1 parallélisation : torch.multiprocessing



Méthode 2 parallélisation : MPS + shell script

1.1.1. MPS

The Multi-Process Service (MPS) is an alternative, binary-compatible implementation of the CUDA Application Programming Interface (API). The MPS runtime architecture is designed to transparently enable cooperative multi-process CUDA applications, typically MPI jobs, to utilize Hyper-Q capabilities on the latest NVIDIA (Kepler and later) GPUs. Hyper-Q allows CUDA kernels to be processed concurrently on the same GPU; this can benefit performance when the GPU compute capacity is underutilized by a single application process.

```
nperseg=128 -> duration=9.546ms
nperseg=256 -> duration=12.167ms
nperseg=512 -> duration=8.409ms
nperseg=1024 -> duration=12.413ms
nperseg=2048 -> duration=14.101ms
nperseg=8192 -> duration=17.089ms
nperseg=32768 -> duration=9.207ms
nperseg=131072 -> duration=8.739ms
```

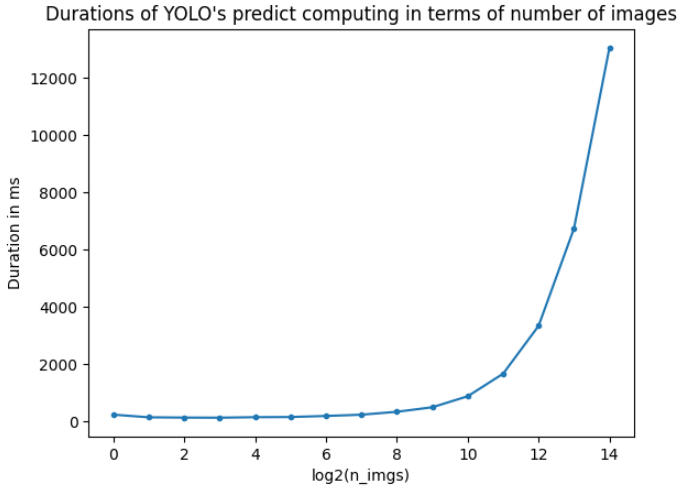
L'étape la plus longue des 8 est plus longue que l'exécution séquentielle au complet (15ms)

AVANTIX

Temps contraint : Inférence + NMS (postprocess)

10/01/2024

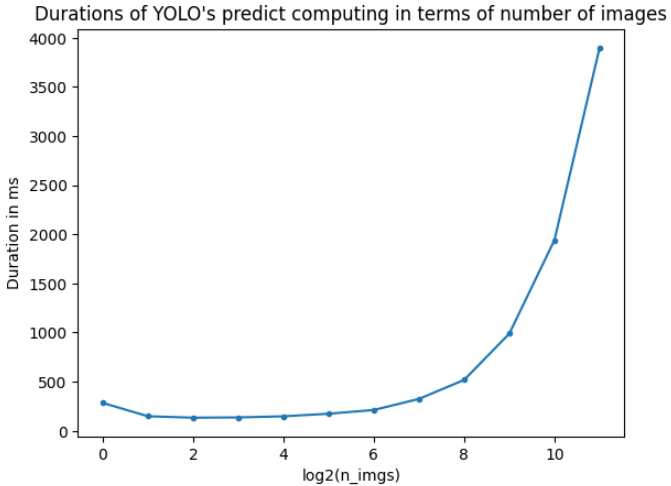
19488 tuiles en 256x128




```
Speed per image at shape (1, 1, 128, 256) :
        0.0µs preprocess
        72.1µs inference
        722.9µs postprocess
```

Ratio 333 à gagner sur l'inférence
Ratio 3345 à gagner sur le NMS

2436 tuiles en 256x1024




```
Speed per image at shape (1, 1, 1024, 256):
        0.0µs preprocess
        562.7µs inference
        1483.8µs postprocess
```

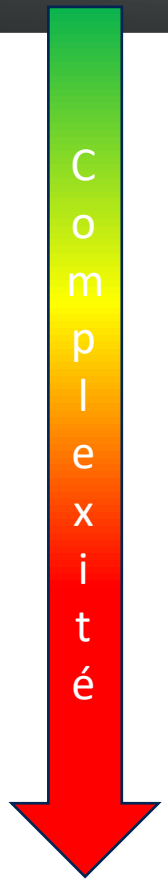
Ratio 325 à gagner sur l'inférence

Ratio 858 à gagner sur le NMS

A même nombre de pixels, il est préférable d'avoir moins d'images pour faire tourner moins souvent NMS.

A voir comment optimiser NMS

Optimisations pour l'inférence



~~SiLU vs ReLU~~

H100 (x3), FP8 (x2)

Export TensorRT (x5) → en cours de debug

Post Training Quantization (INT8)

NVIDIA Triton Server ?

Modèle + petit

YOLOv8 → YOLOv10 (x3)

Pruning

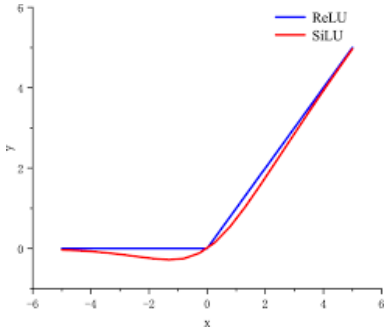
Knowledge distillation

Optimize architecture bottlenecks

Les facteurs de gains renseignés sont hypothétiques

ReLU vs SiLU

YOLOv8 utilise la fonction d'activation SiLU connu pour avoir surpassé ReLU en performances.
Idée : Remplacer SiLU par ReLU pour gagner en temps de calcul



Après test : léger avantage à SiLU

```
Time taken with activation function SiLU: 193.6 µs
Time taken with activation function ReLU: 228.9 µs
```

TensorRT

Test de l’exemple d’ultralytics qui crash. Debug à la rentrée

```python
import pickle
import torch
from ultralytics import utils
from ultralytics.nn.autobackend import AutoBackend

device = "cuda:1"
pickle_path = "sxxs_tiles.pkl"
sxxs_tiles = pickle.load(open(pickle_path, 'rb'))
sxxs_tiles = sxxs_tiles.to(device)

model = AutoBackend("/data/SIGINT/CBE/bt202g-norm-eps/tuning_nano_Adam_fraction_1_reg_max_16_single_cls_with_dfl_with_mosaic_and_mixup/tune/weights/best.engine",
                    device=torch.device(device),
                    fp16=True)
# Warmup
model.warmup()
preds = model(sxxs_tiles)
```

✓ 4.4s Python

```
Loading /data/SIGINT/CBE/bt202g-norm-eps/tuning_nano_Adam_fraction_1_reg_max_16_single_cls_with_dfl_with_mosaic_and_mixup/tune/weights/best.engine for TensorRT inference...
[08/14/2024-17:27:29] [TRT] [I] Loaded engine size: 7 MiB
[08/14/2024-17:27:29] [TRT] [I] [MemUsageChange] TensorRT-managed allocation in IExecutionContext creation: CPU +0, GPU +1, now: CPU 0, GPU 6 (MiB)
[08/14/2024-17:27:29] [TRT] [E] 3: [executionContext.cpp::setInputShape::2037] Error Code 3: API Usage Error (Parameter check failed at: runtime/api/executionContext.cpp::setInputShape::2037, condition: engineDims.
[08/14/2024-17:27:29] [TRT] [E] 1: [gemmBaseRunner.cpp::executeGemm::468] Error Code 1: Cask (Cask Gemm execution)
[08/14/2024-17:27:29] [TRT] [E] 1: [checkMacros.cpp::catchCudaError::181] Error Code 1: Cuda Runtime (an illegal memory access was encountered)
[08/14/2024-17:27:30] [TRT] [E] 3: [executionContext.cpp::setInputShape::2037] Error Code 3: API Usage Error (Parameter check failed at: runtime/api/executionContext.cpp::setInputShape::2037, condition: engineDims.
[08/14/2024-17:27:30] [TRT] [E] 1: [reformat.cu::operator()::1609] Error Code 1: Cuda Runtime (an illegal memory access was encountered)
```

AVANTIX

MERCI