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 → fixé car quasi aucune influence sur le ratio de gain
- IMG_HEIGHT = 1024 si non précisée
- 16 842 752 échantillons (4,211ms) → Donne SXX à IMG_WIDTH colonnes pour nperseg=131075

Script : CBE/_temps_contraint/sxx/

Calcul des spectrogrammes

```
def sxx():
     # Reshape en temps
     iq = iq.unfold(0, nperseg, nperseg – noverlap)
     # Windowing par tukey (alpha = 0,25)
     window = tukey(nperseg, alpha=0,25)
     iq = iq * window[:, None]
     # Zeros padding if nfft > nperseg (avec nfft = max(IMG_HEIGHT, nperseg)
     iq = torch.cat(iq, torch.zeros(nfft-nperseg, iq.size(1))
     # FFT
     zxx = torch.fft.fft(iq)
     # FFTSHIFT
     zxx = torch.fft.fftshift(zxx) les[i*bs: (i+1)*bs])
     # SXX
     sxx = (conj(zxx)*zxx).real
     return sxx

def compute_all_sxxs(NPERSEGS):
    sxxs_lin = {}
    for nperseg in NPERSEGS:
           sxxs_lin[nperseg] = sxx(is, nperseg, noverlap, nfft)
    return sxxs_lin
```

Ratio 42 à gagner

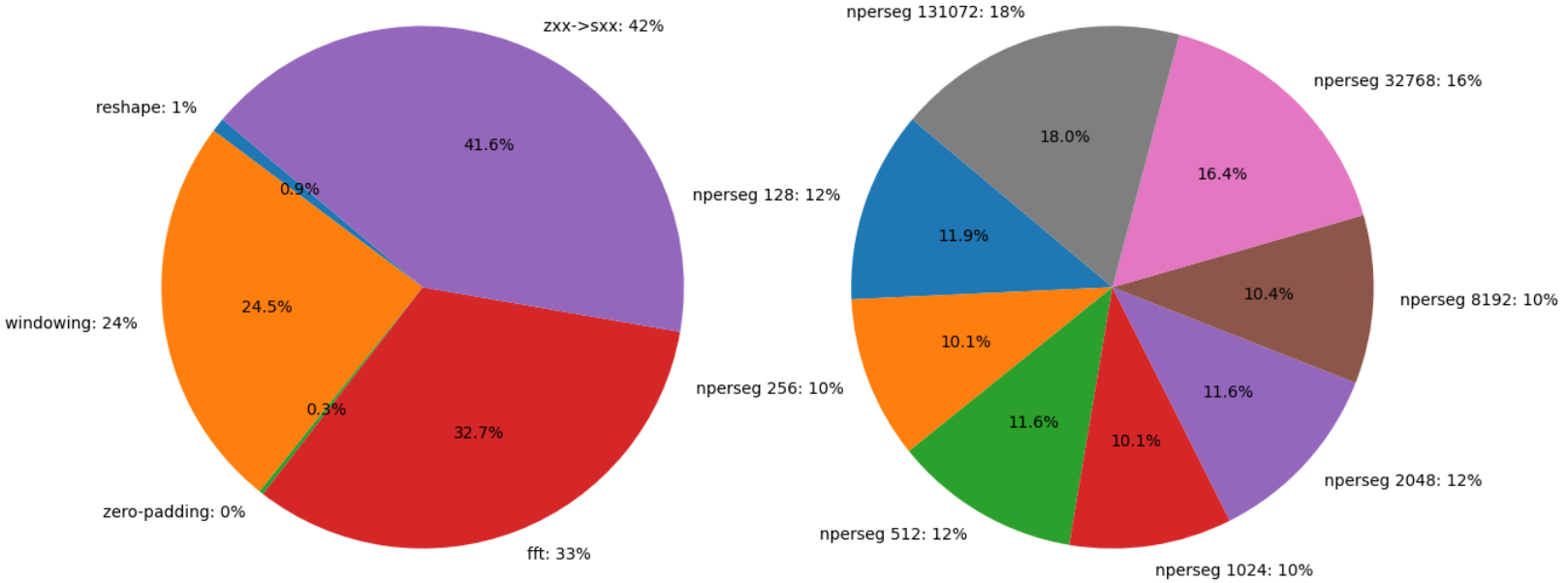
Nouveau calcul des spectrogrammes

```
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
```

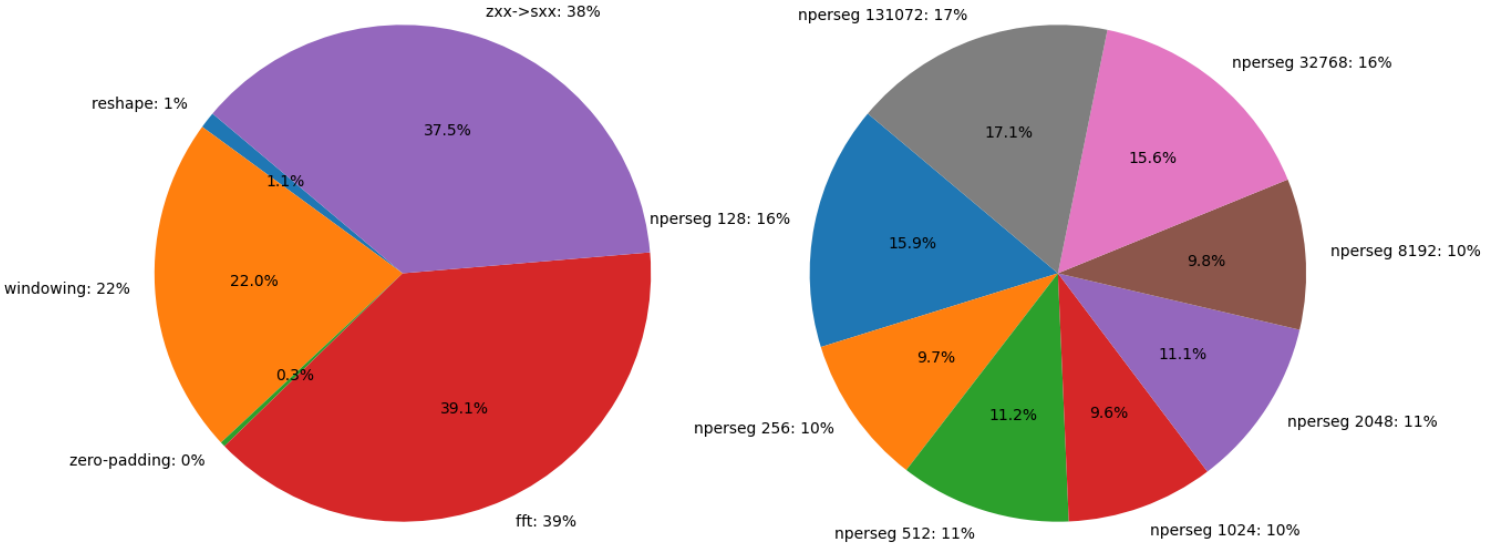
Détails des durées avec warmup double exec

Total duration = 14.298 ms for 4.211 ms of iq
->overall ratio to gain: 3.4



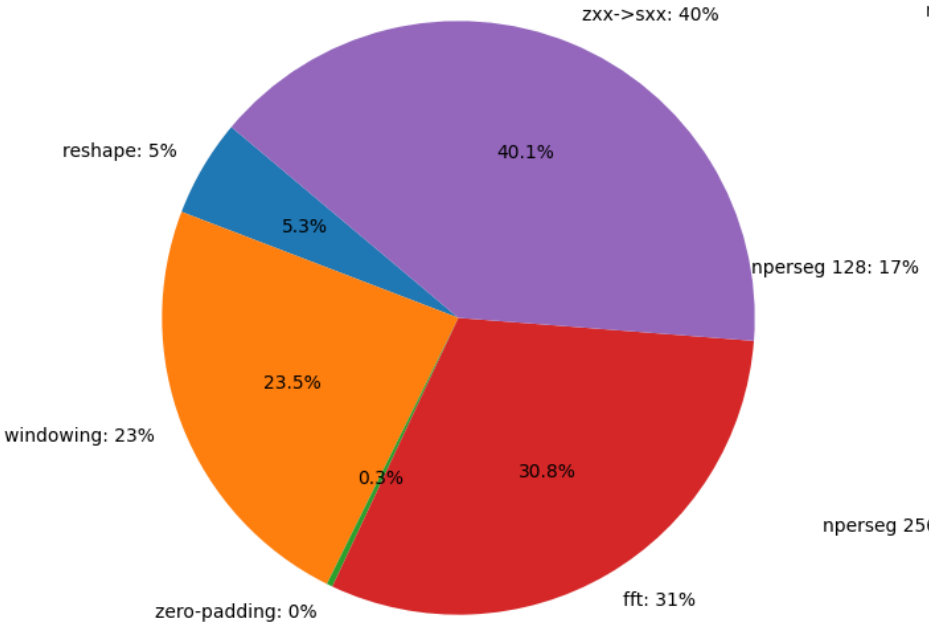
Détails des durées avec warmup random

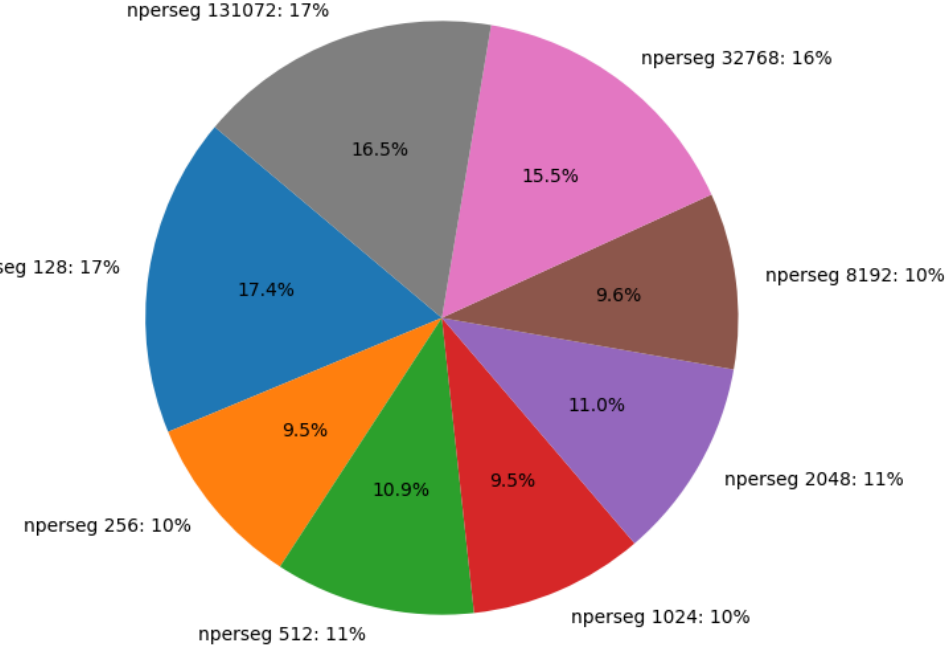
Total duration = 16.413 ms for 4.211 ms of iq
->overall ratio to gain 3.9



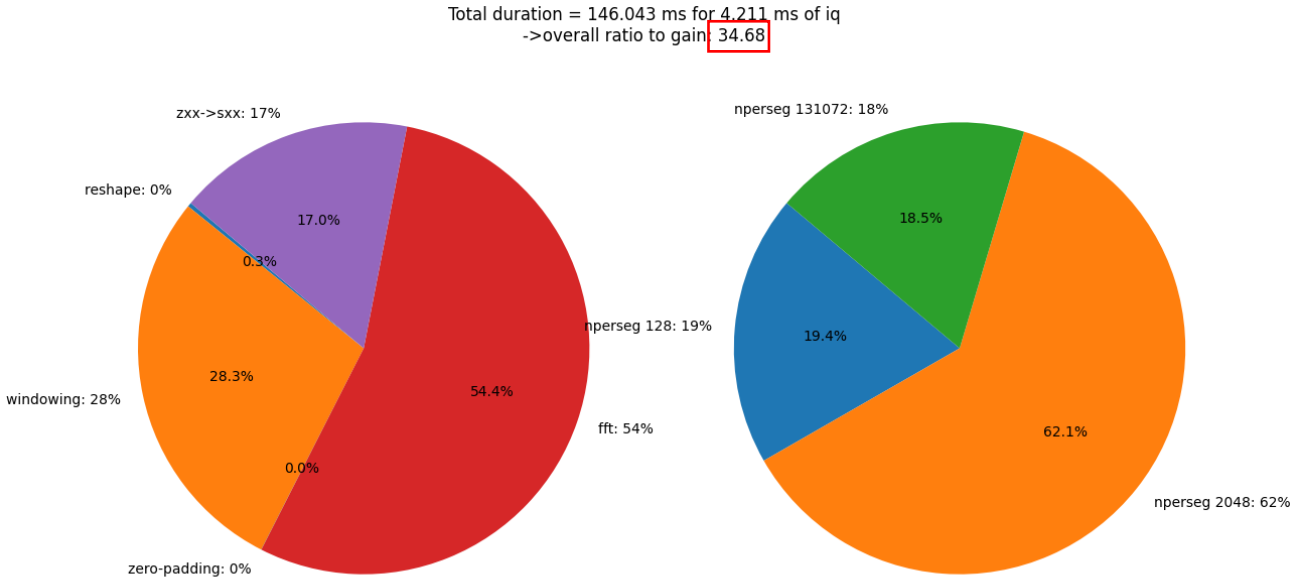
Détails des durées avec warmup double exec sur iq différents

Total duration = 15.762 ms for 4.211 ms of iq
->overall ratio to gain: 3.74





Parallélisation SXX via torch.multiprocessing



Parallélisation SXX via shell script

nperseg = 128 -> duration = 28.857ms
nperseg = 512 -> duration = 28.041ms
nperseg = 2048 -> duration = 14.488ms
nperseg = 131072 -> duration = 17.799ms
nperseg = 1024 -> duration = 19.168ms
nperseg = 32768 -> duration = 12.034ms
nperseg = 8192 -> duration = 18.750ms
nperseg = 256 -> duration = 22.861ms

Moins rapide que for loop sequentielle (15ms) !

Parallélisation SXX via shell script

nperseg = 128 -> duration = 15.322ms
nperseg = 131072 -> duration = 18.743ms
nperseg = 2048 -> duration = 14.148ms

Même si on n'en lance que 3

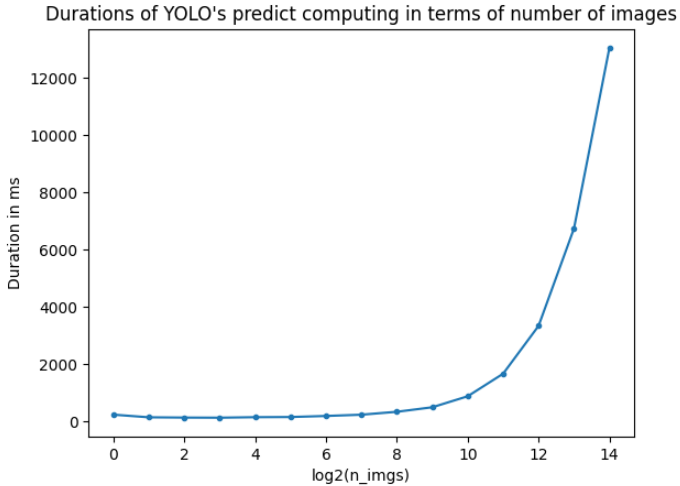
Continuer à investiguer la possibilité de calculer les SXX en parallèle en vérifiant « GPU Utilisation % »

AVANTIX

Temps contraint : Inférence + NMS (postprocess)

10/01/2024

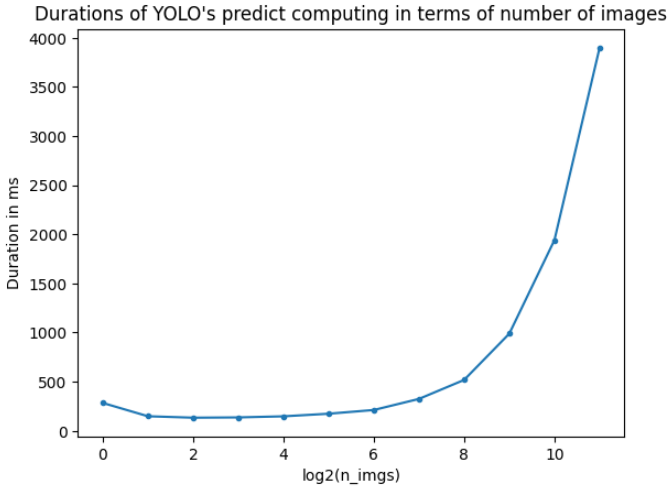
19488 tuiles en 256x128




```
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
```

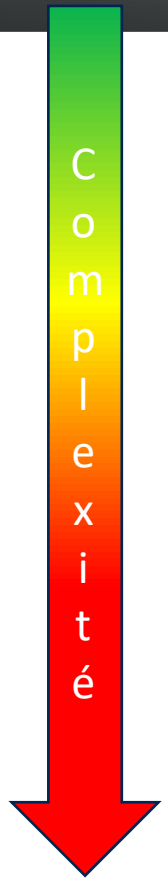
Ratio 325 à gagner sur l'inférence

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

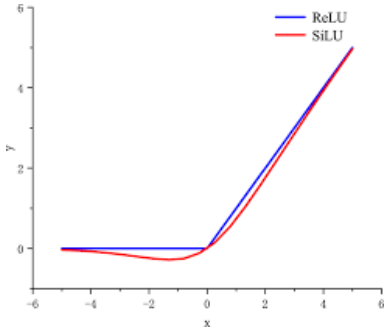
Knowledge distillation

Optimize architecture bottlenecks

Les estimations de gains lorsque renseignées 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