1 Wav2Lip_TensorRT

1.1 System config

#follow the instructions here for system config,
https://github.com/Rudrabha/Wav2Lip/tree/master.
#download sample video and audio here, https://bhaasha.iiit.ac.in/lipsync/.
#download pre-trained models from here, https://github.com/Rudrabha/Wav2Lip/tree/master.
> clear && CUDA_VISIBLE_DEVICES=1 /usr/local/bin/nsys profile /opt/conda/bin/python inference.py --checkpoint_path ./checkpoints/wav2lip.pth --face ./samples/game.mp4 --audio ./samples/game.wav --export_wav2lip_onnx=./onnx_trt/wav2lip_bs1_fp32_6000.onnx --export_wav2lip_trt=./onnx_trt/wav2lip _bs1_fp32_6000.engine --warm_up=2 --export_ s3fd _onnx=./onnx_trt/s3fd_bs43_fp32_6000.onnx --export_s3fd _trt=./onnx_trt/s3fd_bs43_fp32_6000.engine
> CUDA_VISIBLE_DEVICES=1 trtexec --loadEngine=./onnx_trt/s3fd_bs1_fp32_6000.engine --loadInputs="imgs:./results/imgs.dat" --exportOutput=output.json --verbose --batch=1
The original video, driven audio, the generated video is:


game.wav

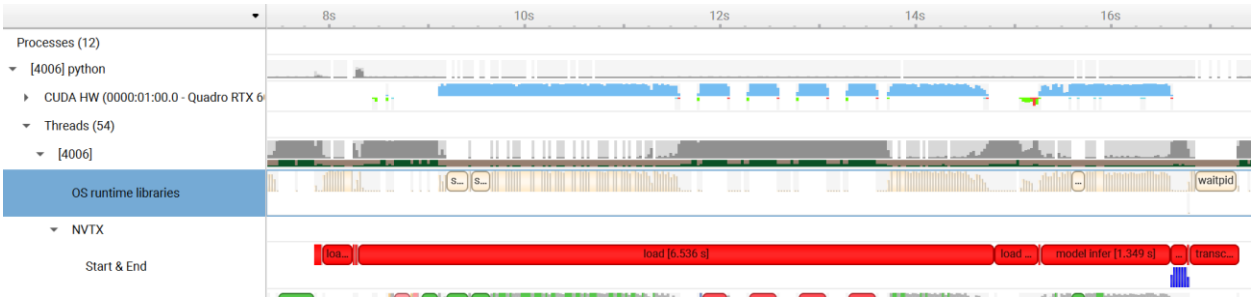

game.mp4


result_voice.mp4

I use system config:
Ubuntu 18.04
GPU Driver 530
CUDA 11.7
TensorRT 8.6.0
GPU RTX 6000
Torch 2.0.1+cu117

1.2 Baseline profiling

stage(gpu0)	device mem usage(gpu0)
prev-E2E	0.47 GB device memory.
load all frames	0.04 GB device memory.
load wav	0.00 GB device memory.
mels	0.00 GB device memory.
mel_chunks	0.00 GB device memory.
datagen	0.00 GB device memory.
load sample0	0.68 GB device memory.
load model	0.03 GB device memory.
preproc batch	0.00 GB device memory.
model infer	9.33 GB device memory.
write frame1-n	0.00 GB device memory.
transcode video from avi to results/result voice.mp4	0.00 GB device memory.



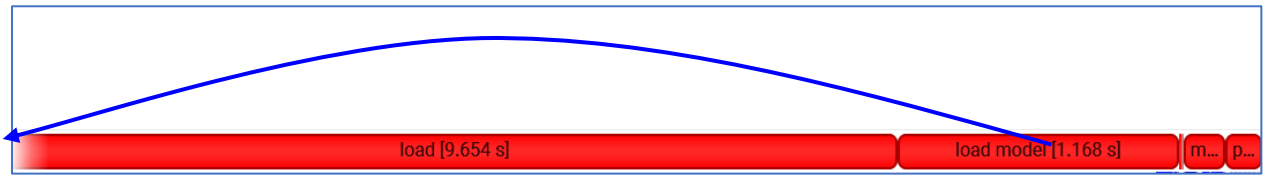
Data loading includes many GPU computing(s3fd face detection), and is the biggest time-consuming part in timeline, much more than wav2lip inference stage.

1.3 Profiling/performance optimization

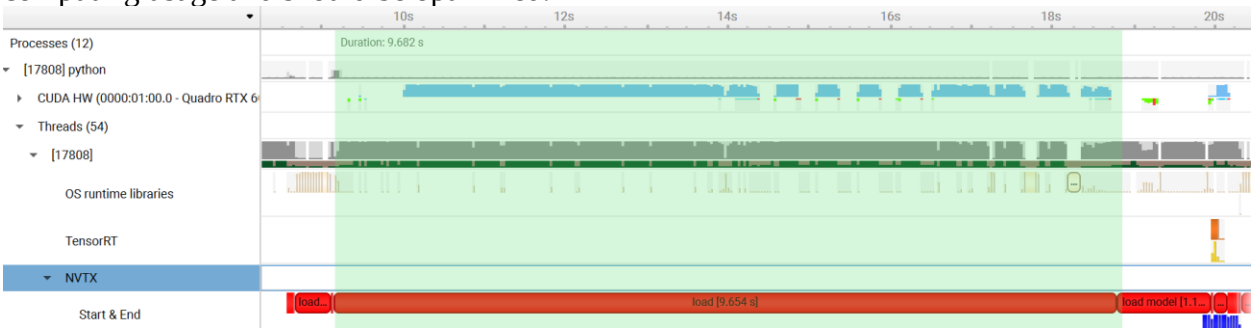
1.3.1 Wav2Lip→TensorRT

1.3.2 Data loading

In the original implementation, model loading stage is in the for block and it should be move to initialization stage.

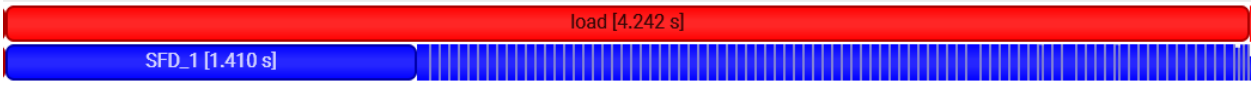


While data loading, it involves face detection model (S3FD detection) and other low CUDA computing usage and should be optimized.

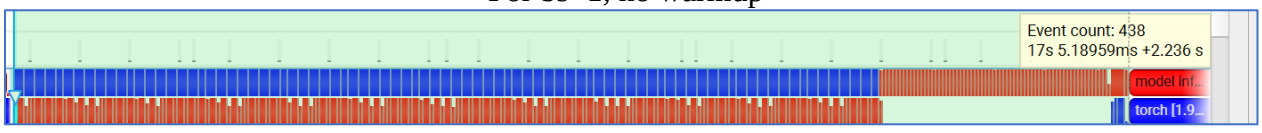


For the SFD face detection on an 86 frames sample video(game.mp4), original implementation (torch native GPU) involves several runs of SFD model in batches:

For bs=16, no warmup



For bs=1, no warmup

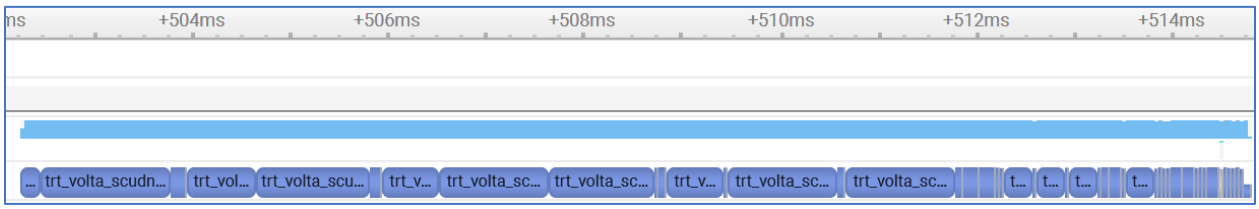


For bs=1 with warmup (torch native GPU, 2.236s)
This is a good case to optimize via TRT.

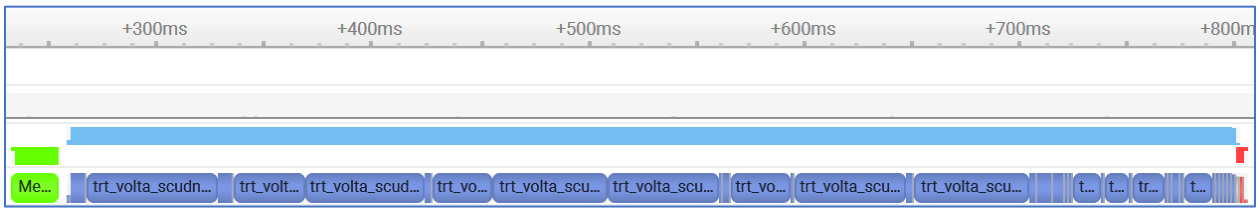
1.3.2.1 S3FD TensorRT conversion

For the game.mp4 sample, it has 86 frames of size 480x720x3, I tried to build a TensorRT engine with bs=86 but failed of OOM, and bs=43 is work with 6000 (24GB).
With warmup, loading performance for bs=1 and bs=43 with TRT is:

No much speed up with higher batchsize. This is because s3fd is a heavy AI model and bs=1 is already making GPU run at high utility:



bs=1

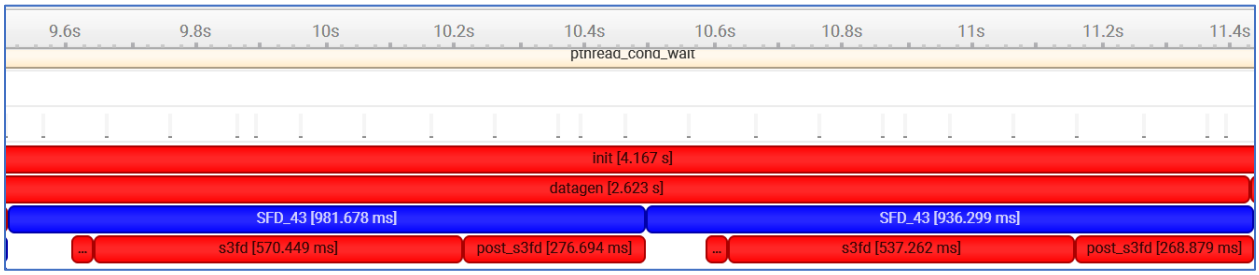


bs=43

for bs=1 already occupies all GPU hardware and setting higher batchsize does not make the GPU run with more efficiency.
Check the timeline deeper and we find there are many empty areas between neighboring s3fd model inference, for bs=1 and bs=43:

These empty areas should be removed and makes GPU run at higher utility, we will optimize this next section.

1.3.2.2 *Hide CPU operations in loading data*



We use bs=43, and there are 2 batches. We group all data loading GPU operations and CPU operations into different code block. We can see from this figure that post_s3fd operations consumes about half the time of s3fd inference and it is executing on CPU in original source code. We want to hide this CPU operations of first SFD_43 inference in the timeline for the second SFD_43 inference.

1.3.2.2.1 Ansynchronzing D2H/H2D data transfer

Cpu tensor should be created in pinned memory by:

```
cpu_tensor.pin_memory()
```

Or

```
torch.empty(pin_memory=True)
```

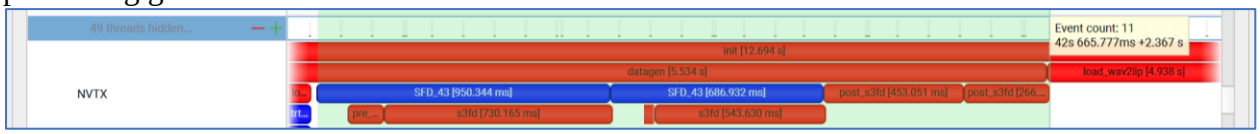
while copying, use

```
dst_tensor.copy_(src_tensor, non_blocking=True)
```

and we need to move the pinned memory CPU tensor to initialization stage. As shown here, before using pinned memory, SFD_43 block can't overlap with post_s3fd block.

After improvement, it overlaps with post_s3fd block and time for datagen stage goes from 2640ms→

After performing this GPU/CPU overlapping, the performance for SFD inference+its post-processing goes from 2367ms→1639:



Before GPU/CPU overlapping

After GPU/CPU overlapping

Please be care for the cuda synchronizing, all after-ward CPU operations need to begin after the data provided by GPU operations is done. I use CUDA event to make sure this:

```
    For loops of CUDA operations:
        CUDA operations……
        events_list.append( torch.cuda.Event() )
        events_list[-1].record()
```

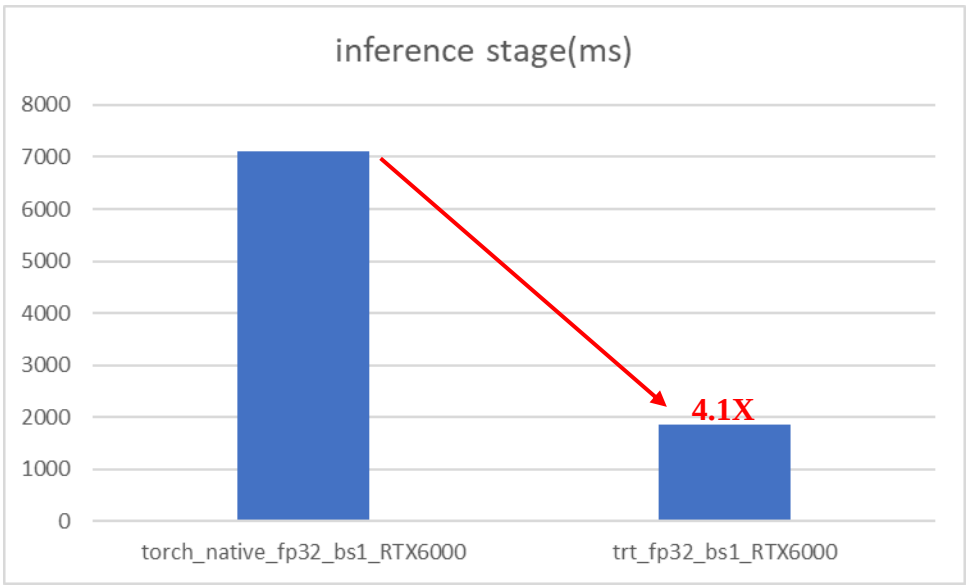
```
    For loops of CPU operations:
        events_list[i].synchronize()
        CPU operations……
```

1.3.3 Inference stage performance

Final inferencing part(s3fd+wav2lip) consumes 1845ms.

Comparing to original baseline 7561ms-463ms(load wav2lip model)=7098ms:

Performance speed up is 4.1X to here:



1.4 Discussion

We find two very interesting points that may many developers encounter. One is that while we using TensorRT for inference, we convert torch tensor to pointer and sent to TensorRT, TensorRT use predefined tensor shape and performing the indexing of each element. But torch CUDA tensor may not in contiguous in device memory, check here for more details, https://saturncloud.io/blog/what-does-contiguous-do-in-pytorch/. Anyway, you have to make sure it is contiguous before sending input to TensorRT. This issue wastes me 2 days to find it out.

Another issue relates to onnx model format and TensorRT engine format. After checking the output of onnx model file with Netron, we find its output tensor order is not the same as TensorRT's, which wastes me about 1 days to positioning it.