Videocomposer Pipeline Inference Performance Optimizing Project Overview

Contents

1 Videocomposer

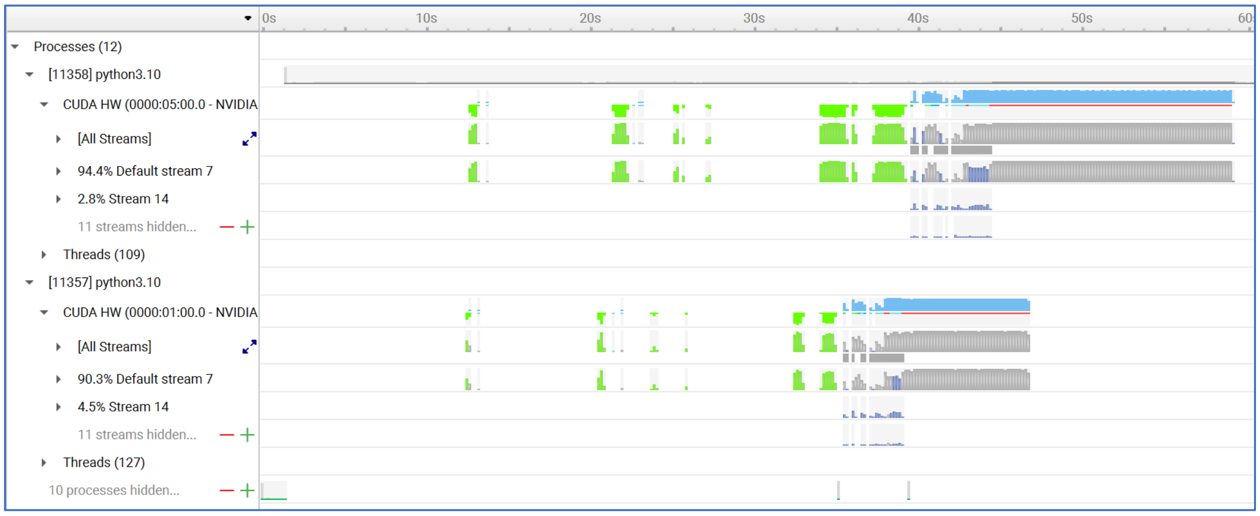
This is my system config scripts just for reference.

```
>docker run --gpus all -it -v /your_name/:/your_name --network host nvcr.io/nvidia/cuda:11.3.0-cudnn8-devel-
ubuntu20.04 bash
>nvcc --version
>apt-get update
>cd /your_name/projects/TRT-DeepSpeed/vid2vid/
#install python3.10
>apt-get install software-properties-common
>add-apt-repository ppa:deadsnakes/ppa
>apt-get update
>apt-get install python3.10
>apt-get install vim
>vim ~/.bashrc
>source ~/.bashrc
>apt-get install python3.10-distutils
>python3.10 get-pip.py
>pip --version
>pip3 --version
>python3 --version
>python --version
>pip install torch==1.12.0+cu113 torchvision functorch --extra-index-url https://download.pytorch.org/whl/cu113
>pip install absl-py==1.4.0    aiohttp==3.8.4    aiosignal==1.3.1    aliyun-python-sdk-core==2.13.36    aliyun-
python-sdk-kms==2.16.0    asttokens==2.2.1    async-timeout==4.0.2    attrs==22.2.0    backcall==0.2.0
cachetools==5.3.0    cffi==1.15.1    chardet==5.1.0    charset-normalizer==3.1.0    clean-fid==0.1.35
click==8.1.3    cmake==3.26.0    crcmod==1.7    cryptography==39.0.2    decorator==5.1.1    decord==0.6.0
easydict==1.10    einops==0.6.0    executing==1.2.0    fairscale==0.4.6    filelock==3.10.2    pytorch-
lightning==1.4.2    flash-attn==0.2.0    frozenlist==1.3.3    fsspec==2023.3.0    ftfy==6.1.1    future==0.18.3
google-auth==2.16.2    google-auth-oauthlib==0.4.6    grpcio==1.51.3    huggingface-hub==0.13.3    idna==3.4
imageio==2.15.0    importlib-metadata==6.1.0    ipdb==0.13.13    ipython==8.11.0    jedi==0.18.2
jmespath==0.10.0    joblib==1.2.0    lazy-loader==0.2    markdown==3.4.3    markupsafe==2.1.2    matplotlib-
inline==0.1.6    motion-vector-extractor==1.0.6    multidict==6.0.4    mypy-extensions==1.0.0    networkx==3.1
numpy==1.24.2    oauthlib==3.2.2    open-clip-torch==2.0.2    openai-clip==1.0.1    opencv-python==4.5.5.64
opencv-python-headless==4.7.0.68    oss2==2.17.0    packaging==23.0    parso==0.8.3    pexpect==4.8.0
pickleshare==0.7.5    pillow==9.4.0    pkgconfig==1.5.5    prompt-toolkit==3.0.38    protobuf==4.22.1
ptyprocess==0.7.0
>pip install pure-eval==0.2.2    pyasn1==0.4.8    pyasn1-modules==0.2.8    pycparser==2.21
pycryptodome==3.17    pydeprecate==0.3.1    pygments==2.14.0    pynvml==11.5.0    pyre-extensions==0.0.23
pywavelets==1.4.1    pyyaml==6.0    regex==2023.3.22    requests==2.28.2    requests-oauthlib==1.3.1    rotary-
embedding-torch==0.2.1    rsa==4.9    sacremoses==0.0.53    scikit-image==0.20.0    scikit-learn==1.2.2
scikit-video==1.1.11    scipy==1.9.1    simplejson==3.18.4    six==1.16.0    stack-data==0.6.2
tensorboard==2.12.0    tensorboard-data-server==0.7.0    tensorboard-plugin-wit==1.8.1    threadpoolctl==3.1.0
tifffile==2023.4.12    tokenizers==0.12.1    tomli==2.0.1
#install xformers
>git clone https://github.com/facebookresearch/xformers/
>cd xformers
>git submodule update --init --recursive
>pip install --verbose --no-deps -e .
```

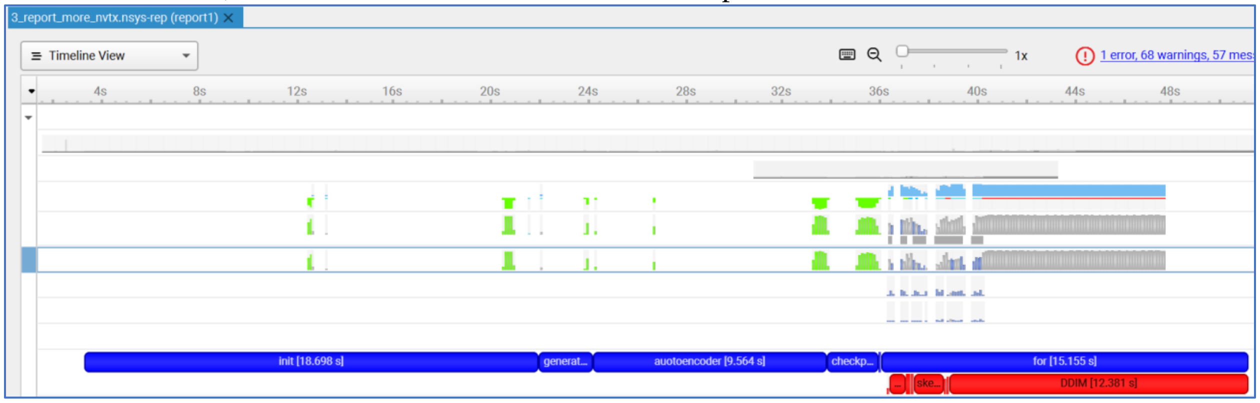
```
>pip install tqdm==4.65.0      traitlets==5.9.0      transformers==4.18.0      triton==2.0.0.dev20221120      typing-
extensions==4.5.0      typing-inspect==0.8.0      urllib3==1.26.15      wcwidth==0.2.6      werkzeug==2.2.3
yarl==1.8.2      zipp==3.15.0
>cd ..
>cd videocomposer/
#cp libGL*
>mv ../../../tmp/* /usr/lib/x86_64-linux-gnu/
>apt-get update
>apt install ffmpeg
># copy model weights
># mv ./model_weights/damo/VideoComposer/non_ema_228000.pth ./model_weights/
># mv ./model_weights/damo/VideoComposer/open_clip_pytorch_model.bin ./model_weights/
># mv ./model_weights/damo/VideoComposer/midas_v3_dpt_large.pth ./model_weights/
># mv ./model_weights/damo/VideoComposer/sketch_simplification_gan.pth ./model_weights/
># mv ./model_weights/damo/VideoComposer/*.pth ./model_weights/
># mv ./model_weights/damo/VideoComposer/*.ckpt ./model_weights/
>python run_net.py    --cfg configs/exp02_motion_transfer.yaml    --seed 9999    --input_video
"demo_video/motion_transfer.mp4"    --image_path "demo_video/moon_on_water.jpg"    --input_text_desc "A beautiful
big moon on the water at night"
#install nsys
>cd /your_name/projects/software/
>./NsightSystems-linux-public-2023.2.1.122-3259852.run
>export PATH=/your_name/projects/software/bin/:$PATH
```

```
> CUDA_VISIBLE_DEVICES=1 /your_name/projects/software/bin/nsys profile /usr/bin/python3.10 run_net.py    --cfg
configs/exp02motion_transfer.yaml    --seed 9999    --input_video "demo_video/motion_transfer.mp4"    --image_path
"demo_video/moon_on_water.jpg"    --input_text_desc "A beautiful big moon on the water at night"
> clear && CUDA_VISIBLE_DEVICES=0 /your_name/projects/software/bin/nsys profile -t
cuda,nvtx,cublas,cudnn,nvvideo  /usr/bin/python3.10 run_net.py    --cfg configs/exp02_motion_transfer.yaml    --seed
9999    --input_video "demo_video/motion_transfer.mp4"    --image_path "demo_video/moon_on_water.jpg"    --
input_text_desc "A beautiful big moon on the water at night"
```

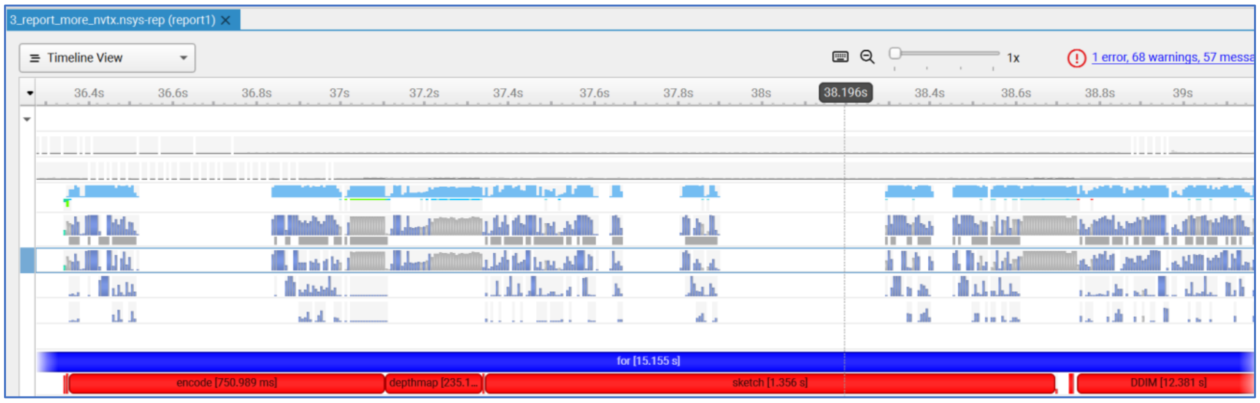
1.1 Baseline



videoComposer uses multi-GPU while inferencing, and GPU balancing is one key feature, always the lowest performance GPU matters. It is hard to find performance bottlenecks directly from the timeline, and nvtx annotations can do some help.



From this timeline, prev-for(init, generators, autoencoder, loading checkpoint, etc) stages consumes about 2/3 times of a pipeline execution. So reuse prev-for stages for different user queries is necessary.



In the for stage, must first check DDIM, encode, depthmap, sketch block.

1.2 nvtx && memory footprint

two items are key to inferencing performance for online GPU inferencing, where is the bottleneck is and how to improve it, how many device memory consuming for each stage of the pipeline.

1.2.1 nvtx

Install:

>pip install nvtx

Usage:

Step 1: coding

```
import nvtx
rng = nvtx.start_range(message="deserialize", color="blue")
…stage_n…
```

```
nvtx.end_range(rng)
```

step 2: Nsight System profiling

Install Nsight System in GPU server, >./NsightSystems-linux-public-2023.2.1.122-3259852.run

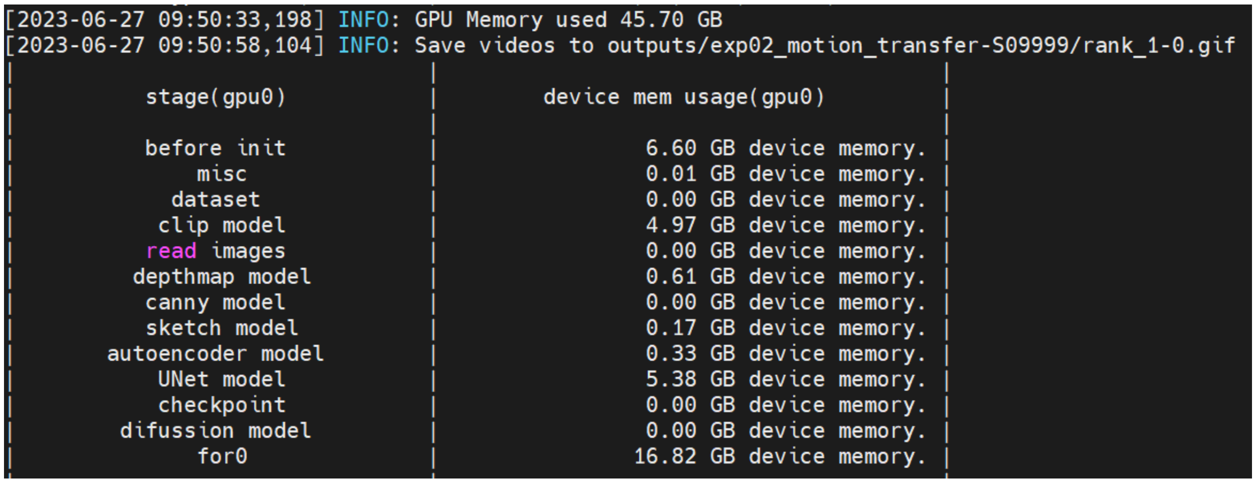
Profiling on GPU server, >/xxx/bin/nsys profile /usr/bin/python3.10 run_net.py --cfg configs/exp02_motion_transfer.yaml --seed 9999 --input_video "demo_video/motion_transfer.mp4" --image_path "demo_video/moon_on_water.jpg" --input_text_desc "A beautiful big moon on the water at night"

Install Nsight System in your local Windows desktop, open the profiling file *.nsys-rep and check the timeline.

1.2.2 Memory footprint

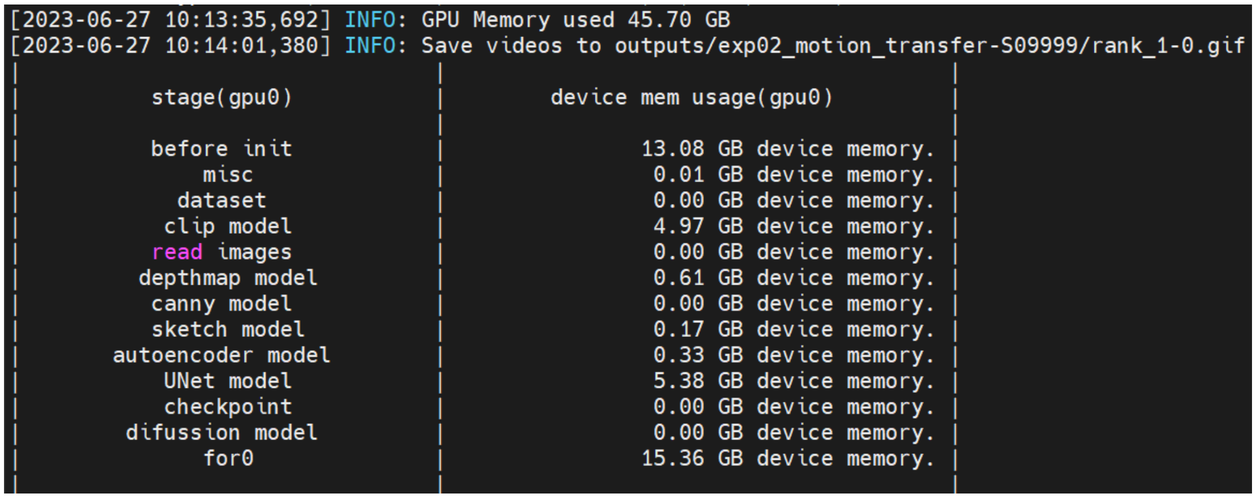
1.2.2.1 Fp32 vs fp16

Using one GPU only, A6000, fp32:



```
[2023-06-27 09:50:33,198] INFO: GPU Memory used 45.70 GB
[2023-06-27 09:50:58,104] INFO: Save videos to outputs/exp02_motion_transfer-S09999/rank_1-0.gif
|                                 |                                        |
|          stage(gpu0)            |        device mem usage(gpu0)          |
|                                 |                                        |
|          before init            |               6.60 GB device memory.  |
|             misc                |               0.01 GB device memory.  |
|            dataset              |               0.00 GB device memory.  |
|           clip model            |               4.97 GB device memory.  |
|          read images            |               0.00 GB device memory.  |
|         depthmap model          |               0.61 GB device memory.  |
|          canny model            |               0.00 GB device memory.  |
|          sketch model           |               0.17 GB device memory.  |
|       autoencoder model         |               0.33 GB device memory.  |
|           UNet model            |               5.38 GB device memory.  |
|           checkpoint            |               0.00 GB device memory.  |
|        difussion model          |               0.00 GB device memory.  |
|             for0                |              16.82 GB device memory.  |
```

Using one GPU only, A6000, fp16:

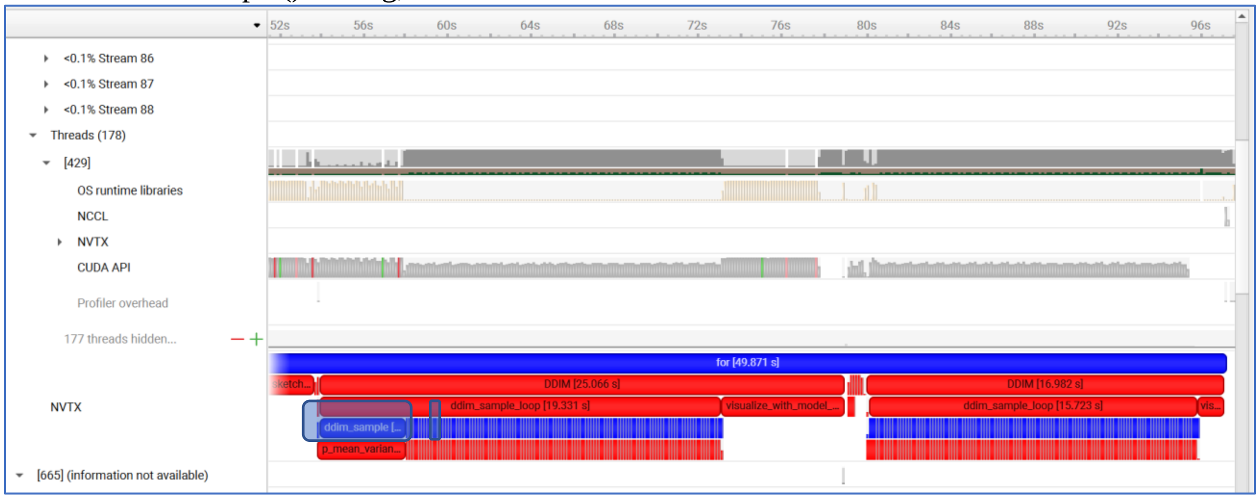


```
[2023-06-27 10:13:35,692] INFO: GPU Memory used 45.70 GB
[2023-06-27 10:14:01,380] INFO: Save videos to outputs/exp02_motion_transfer-S09999/rank_1-0.gif
|                                 |                                        |
|          stage(gpu0)            |        device mem usage(gpu0)          |
|                                 |                                        |
|          before init            |              13.08 GB device memory.  |
|             misc                |               0.01 GB device memory.  |
|            dataset              |               0.00 GB device memory.  |
|           clip model            |               4.97 GB device memory.  |
|          read images            |               0.00 GB device memory.  |
|         depthmap model          |               0.61 GB device memory.  |
|          canny model            |               0.00 GB device memory.  |
|          sketch model           |               0.17 GB device memory.  |
|       autoencoder model         |               0.33 GB device memory.  |
|           UNet model            |               5.38 GB device memory.  |
|           checkpoint            |               0.00 GB device memory.  |
|        difussion model          |               0.00 GB device memory.  |
|             for0                |              15.36 GB device memory.  |
|                                 |                                        |
```

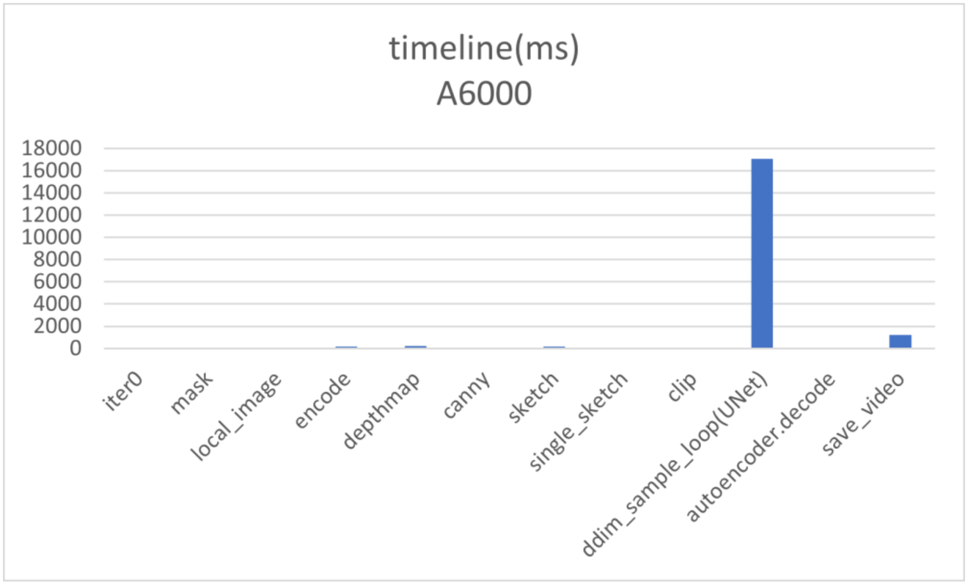
There is no much difference between them. Because DDIM used fp16 already.

1.3 Warm up

Most always, warmup helps to make the profiling more accurate, here shows two run of the for block. For the first ddim_sample() calling, it consumes much more time than after ddim_sample() callings. In this figure, left rect has the same operations to the right thin rect. For the first ddim_sample() calling,

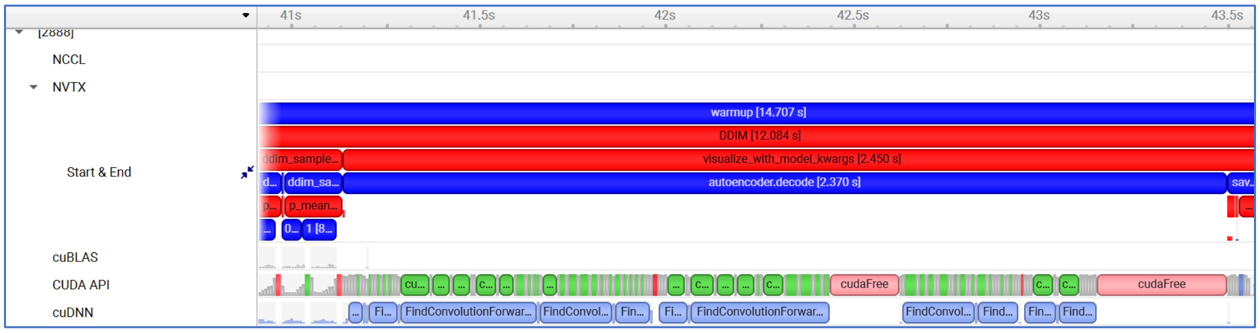


For one inference, the timeline consumed by each stage looks like this:



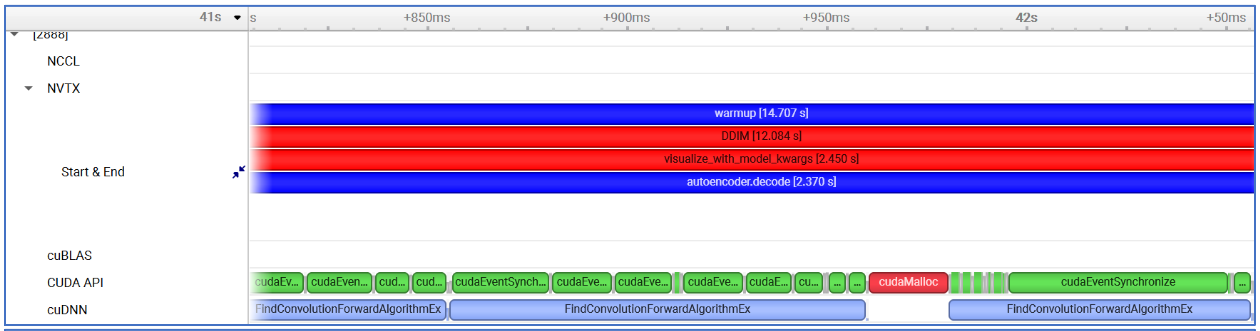
In this chart, UNet part consumes ~90% of the time and it is the most computing intensive part too. The second highest one is save_video part.

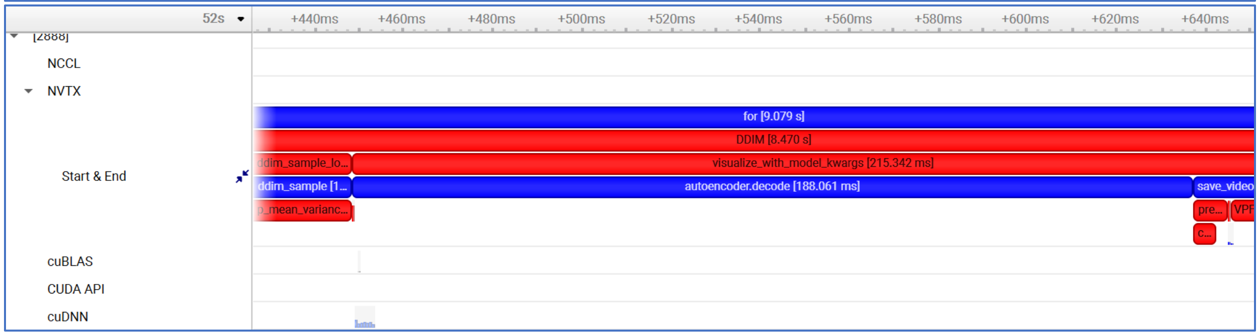
1.3.1 Warmup for cuDNN



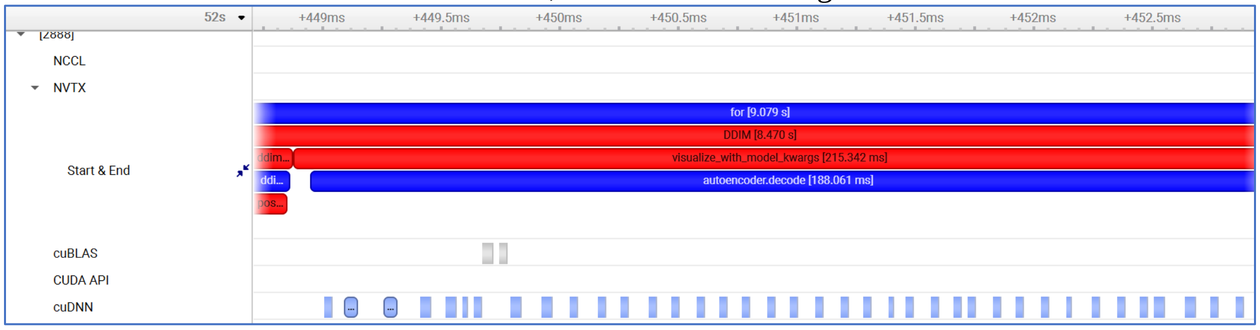
First run, in warmup

In warmup, autoencoder.decode calling cuDNN/cuBLAS APIs, which calls a lot of FindConvolutionForwardAlgorithmEx and involves stream synchronizing and device memory allocation. These operations should be in the initialization stage. For warmup autoencoder.decode stage, it consumes 2370ms, and for later real inference run, it only consumes 188ms.

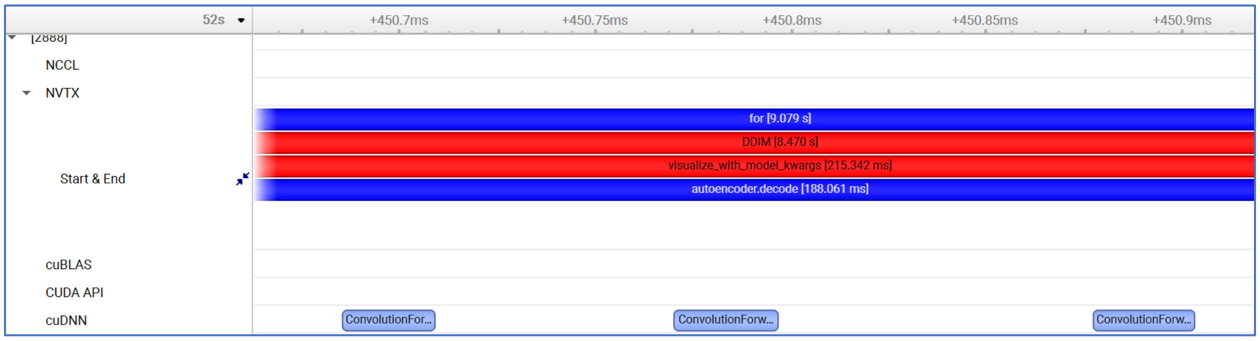

41s
+850ms
+900ms
+950ms
42s
+50ms
[2888]
NCCL
NVTX
Start & End
cuBLAS
CUDA API
cuDNN
warmup [14.707 s]
DDIM [12.084 s]
visualize_with_model_kwargs [2.450 s]
autoencoder.decode [2.370 s]
cudaEventSynch...
cudaMalloc
cudaEventSynchronize
FindConvolutionForwardAlgorithmEx
FindConvolutionForwardAlgorithmEx
FindConvolutionForwardAlgorithmEx



Second run, in real inference stage



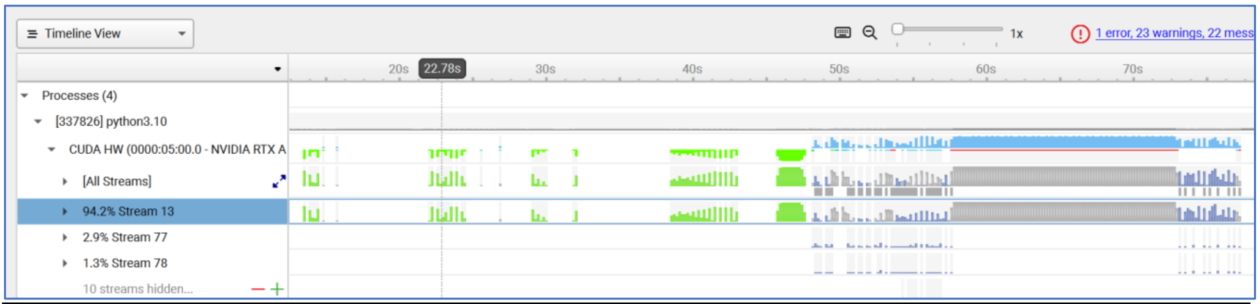
Zoom in (a)

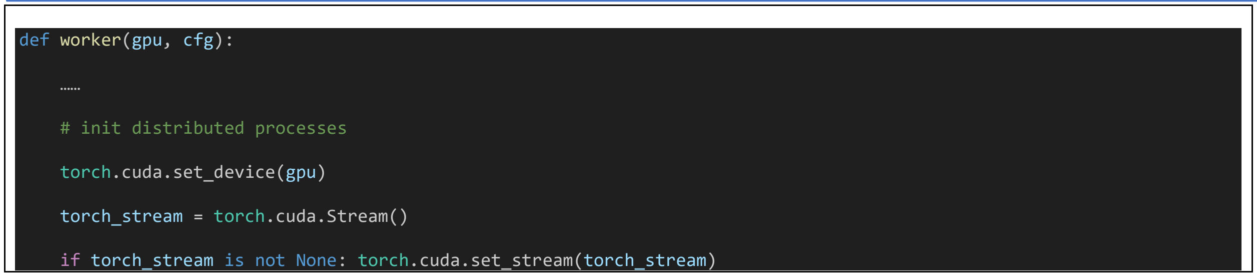


Zoom in (b)

In real refence run, it only includes computing kernels and FindConvolutionForwardAlgorithmEx calls are gone.

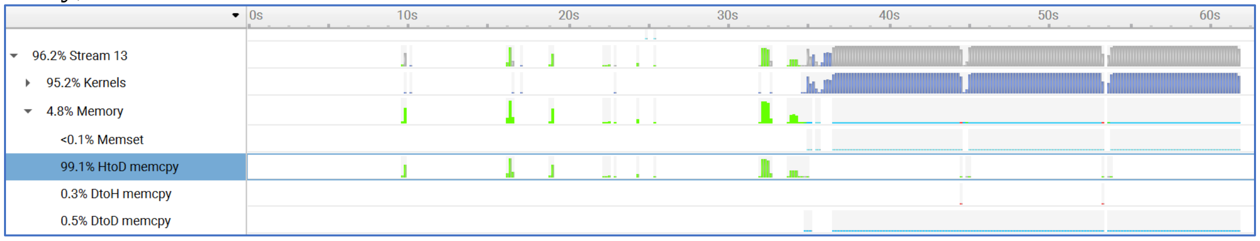
1.4 User created CUDA stream





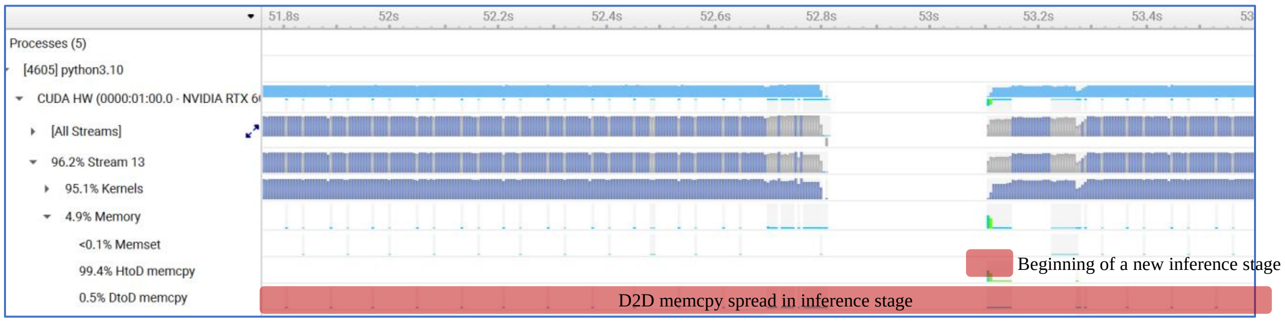
```
def worker(gpu, cfg):

    ......

    # init distributed processes

    torch.cuda.set_device(gpu)

    torch_stream = torch.cuda.Stream()

    if torch_stream is not None: torch.cuda.set_stream(torch_stream)
```

One benefit of using your own CUDA steams is you can check H2D/D2H/D2D memory copies easily, as shown here:



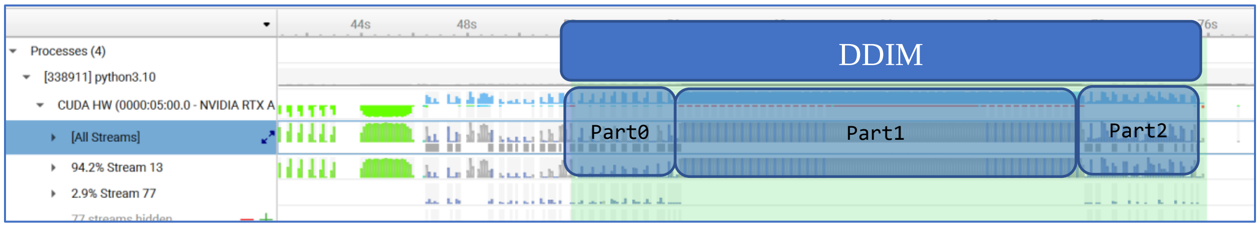
All H2D/D2H/D2D memory copies are list in your own CUDA stream timeline

And we can see from this timeline, that for the inference stage, only H2D/D2H at the beginning of it, this is good that all later operations are working on GPU:



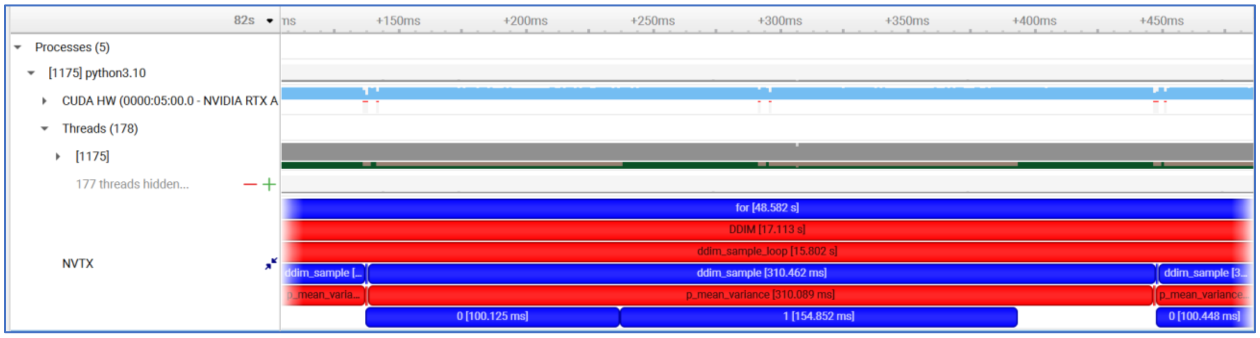
D2d memcpy is happening here and there as needed in the inference stage.

1.5 DDIM stage optimization



DDIM can break into 3 parts, part0 and part1 is in diffusion.ddim_sample_loop function, part2 in visualize_with_model_kwargs. Part0 and part2 may can be optimized, part1 is hard to optimized as it consumes almost GPU computing power already.

1.5.1 UNetSD_temporal forward optimization



Each p_mean_variance() calling includes 2 calling of UNetSD_temporal.forward, main inference time is consumed by it.

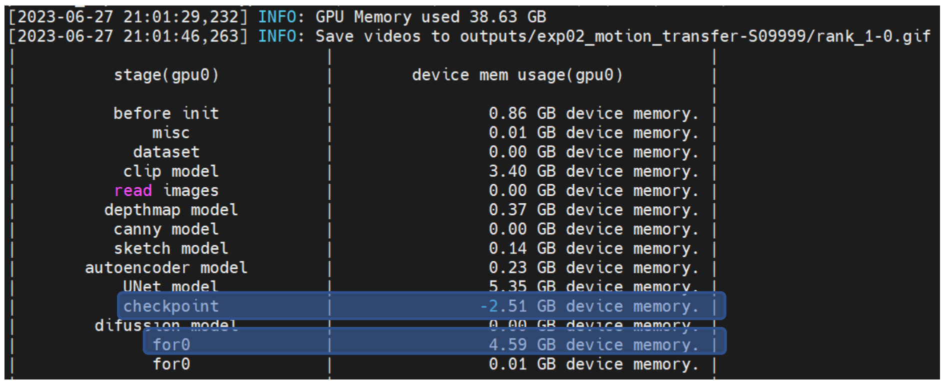
1.5.1.1 Fp16

Set all models to fp16:

```
clip_encoder.to(torch.float16)

clip_encoder_visual.model.to(torch.float16)

pidinet.to(torch.float16)

cleaner.to(torch.float16)

pidi_mean.to(torch.float16)
```

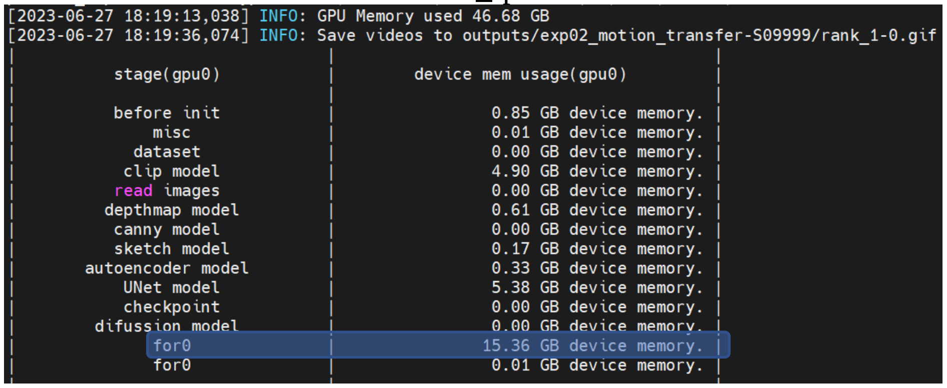
```
pidi_std.to(torch.float16)

if MODEL_TO_FLOAT16_ENABLED: model.to(torch.float16)
```

The profiling output shows here, the line of fp16 in rect is much lower than its fp32 counterpart, 4.59GB vs 15.36GB, and total GPU memory used is 38.63GB vs 46.68GB.



```
[2023-06-27 21:01:29,232] INFO: GPU Memory used 38.63 GB
[2023-06-27 21:01:46,263] INFO: Save videos to outputs/exp02_motion_transfer-S09999/rank_1-0.gif
|                                |                                       |
|          stage(gpu0)           |        device mem usage(gpu0)         |
|                                |                                       |
|          before init           |              0.86 GB device memory.   |
|              misc              |              0.01 GB device memory.   |
|            dataset             |              0.00 GB device memory.   |
|           clip model           |              3.40 GB device memory.   |
|          read images           |              0.00 GB device memory.   |
|         depthmap model         |              0.37 GB device memory.   |
|          canny model           |              0.00 GB device memory.   |
|          sketch model          |              0.14 GB device memory.   |
|       autoencoder model        |              0.23 GB device memory.   |
|           UNet model           |              5.35 GB device memory.   |
|           checkpoint           |             -2.51 GB device memory.   |
|        difussion model         |              0.00 GB device memory.   |
|              for0              |              4.59 GB device memory.   |
|              for0              |              0.01 GB device memory.   |
```

model_fp16



```
[2023-06-27 18:19:13,038] INFO: GPU Memory used 46.68 GB
[2023-06-27 18:19:36,074] INFO: Save videos to outputs/exp02_motion_transfer-S09999/rank_1-0.gif
|                                |                                       |
|          stage(gpu0)           |        device mem usage(gpu0)         |
|                                |                                       |
|          before init           |              0.85 GB device memory.   |
|              misc              |              0.01 GB device memory.   |
|            dataset             |              0.00 GB device memory.   |
|           clip model           |              4.90 GB device memory.   |
|          read images           |              0.00 GB device memory.   |
|         depthmap model         |              0.61 GB device memory.   |
|          canny model           |              0.00 GB device memory.   |
|          sketch model          |              0.17 GB device memory.   |
|       autoencoder model        |              0.33 GB device memory.   |
|           UNet model           |              5.38 GB device memory.   |
|           checkpoint           |              0.00 GB device memory.   |
|        difussion model         |              0.00 GB device memory.   |
|              for0              |             15.36 GB device memory.   |
|              for0              |              0.01 GB device memory.   |
```

model_fp32

1.6 Codec

Known VPF bug on motion vector:
https://github.com/NVIDIA/VideoProcessingFramework/blob/3347e555ed795ba7de98b4e6b9bf7fe441784663/tests/test_reported_bugs.py line 45: nvDec.GetMotionVectors()

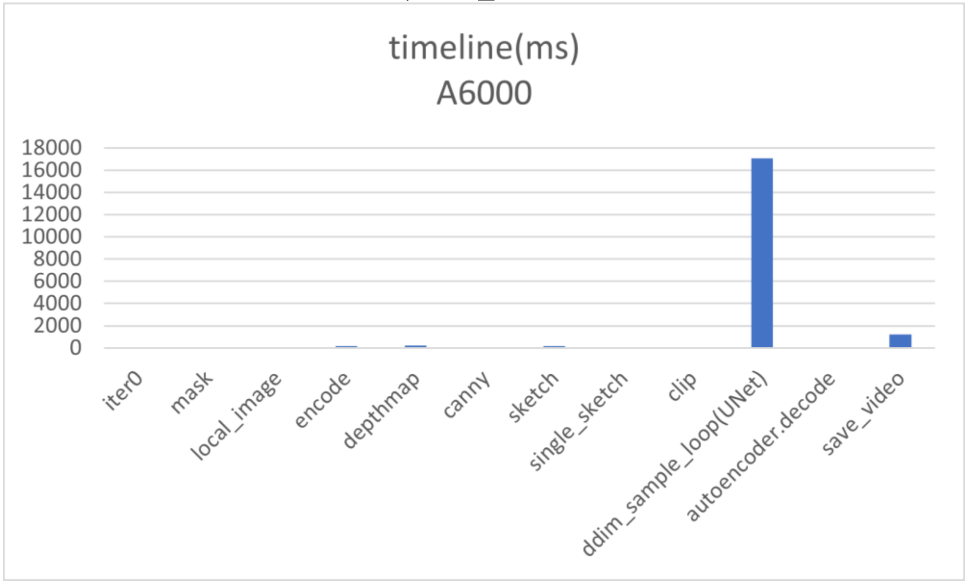
1.6.1 Install VPF

```
>cd /your_name/projects/TRT-DeepSpeed/vid2vid/videocomposer/VideoProcessingFramework/
>apt install -y libavfilter-dev     libavformat-dev     libavcodec-dev     libswresample-dev     libavutil-dev
wget     build-essential     git
>pip3 install .
>apt update
>apt install libtorch
>pip install src/PytorchNvCodec
>make run_samples_without_docker
>find /usr -name _PytorchNvCodec.cpython-310-x86_64-linux-gnu.so
>cp /usr/local/lib/python3.10/dist-packages/_PytorchNvCodec.cpython-310-x86_64-linux-gnu.so
/usr/local/lib/python3.10/dist-packages/PytorchNvCodec/
>ldd /usr/local/lib/python3.10/dist-packages/PytorchNvCodec/_PytorchNvCodec.cpython-310-x86_64-linux-gnu.so
```

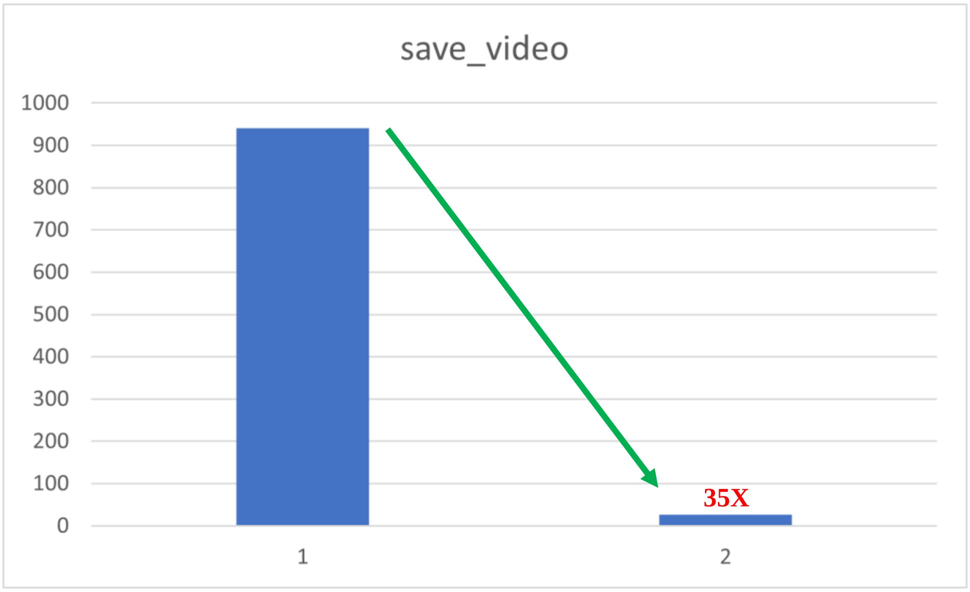
```
>find /usr -name libtorch*
>export LD_LIBRARY_PATH=/usr/local/lib/python3.10/dist-packages/torch/lib/:$LD_LIBRARY_PATH
>ldd /usr/local/lib/python3.10/dist-packages/PytorchNvCodec/_PytorchNvCodec.cpython-310-x86_64-linux-gnu.so
>python ./samples/SamplePyTorch.py
>clear && CUDA_VISIBLE_DEVICES=0,1 /usr/bin/python3.10 run_net.py    --cfg configs/exp02_motion_transfer.yaml    --
seed 9999    --input_video "demo_video/motion_transfer.mp4"    --image_path "demo_video/moon_on_water.jpg"    --
input_text_desc "A beautiful big moon on the water at night"
>cd ../videocomposer/VideoProcessingFramework/
>clear && CUDA_VISIBLE_DEVICES=0 /usr/bin/python3.10 run_net.py    --cfg configs/exp02_motion_transfer.yaml    --
seed 9999    --input_video "demo_video/motion_transfer.mp4"    --image_path "demo_video/moon_on_water.jpg"    --
input_text_desc "A beautiful big moon on the water at night"
>cd ./VideoProcessingFramework/
>clear && python ./tests/test_PyNvEncoder.py
```

1.6.2 Video encoding

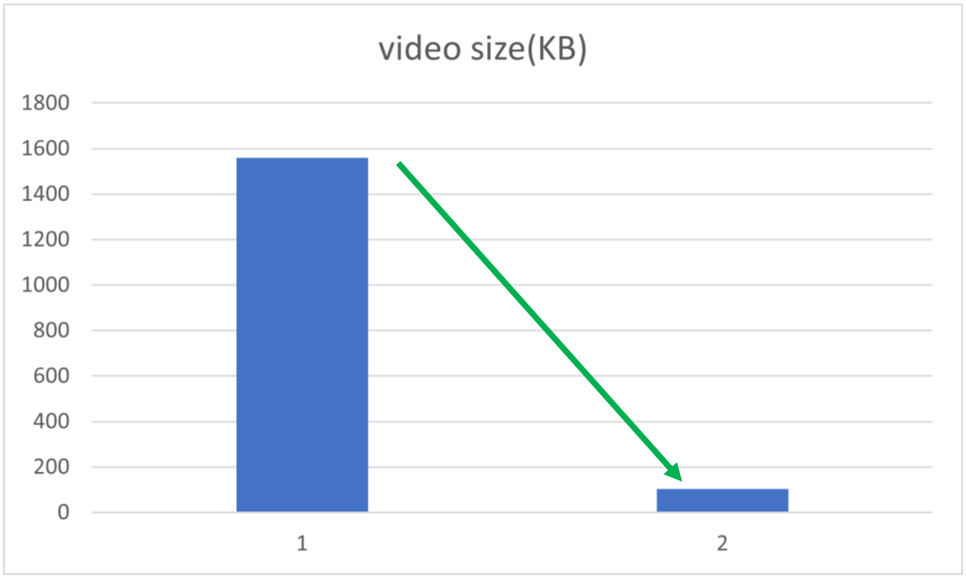
Original implementation calls imageio.mimwrite and output a git of 16 frames. The input frame size is 256x1024x3. As shows in this chart, save_video consumes about 1s.



Here I replace video encoding part with NVIDIA VPF, and the performance comparing to baseline is:



Comparing baseline gif file format, using H264 makes the video file size much smaller:



1.6.3 Torch Tensor to Codec Surface

Sample scripts from https://github.com/NVIDIA/VideoProcessingFramework/tree/master/samples.

```
def cuda_tensor_to_surface( tensor, gpu_id, torch_stream=None ):
    tensor_h = tensor.shape[0]
    tensor_w = tensor.shape[1]
    tensor_c = tensor.shape[2] #channel
    surf_dst = nvc.Surface.Make(nvc.PixelFormat.RGB_PLANAR, tensor_w, tensor_h, gpu_id)

    assert not surf_dst.Empty()
    dst_plane = surf_dst.PlanePtr()

    cuda.cuMemcpyDtoDAsync(dst_plane.GpuMem(), tensor.data_ptr(), tensor_h * tensor_w * tensor_c, torch_stream.cuda_stream)
    trt_util_2.synchronize(torch_stream)
    return surf_dst
```

1.6.4 Codec Surface to Torch Tensor

```
     def test_pycuda_memcpy_Surface_Tensor(self):
       while True:
           surf_src = self.nvDec.DecodeSingleSurface()
           if surf_src.Empty():
               break
           src_plane = surf_src.PlanePtr()

           surface_tensor = torch.zeros(
               src_plane.Height(),
               src_plane.Width(),
               1,
               dtype=torch.uint8,
               device=torch.device(f"cuda:{self.gpu_id}"),
           )
           dst_plane = surface_tensor.data_ptr()

           memcpy_2d = cuda.Memcpy2D()
           memcpy_2d.width_in_bytes = src_plane.Width() * src_plane.ElemSize()
           memcpy_2d.src_pitch = src_plane.Pitch()
           memcpy_2d.dst_pitch = self.nvDec.Width()
           memcpy_2d.width = src_plane.Width()
           memcpy_2d.height = src_plane.Height()
           memcpy_2d.set_src_device(src_plane.GpuMem())
           memcpy_2d.set_dst_device(dst_plane)
           memcpy_2d(self.cuda_str)

           frame_src = np.ndarray(shape=(0), dtype=np.uint8)
           if not self.nvDwn.DownloadSingleSurface(surf_src, frame_src):
               self.fail("Failed to download decoded surface")

           frame_dst = surface_tensor.to("cpu").numpy()
           frame_dst = frame_dst.reshape((src_plane.Height() * src_plane.Width()))

           if not np.array_equal(frame_src, frame_dst):
               self.fail("Video frames are not equal")
```

1.6.5 Numpy NDArray to Codec Surface

```
    def run(self):
        try:
            while True:
                frameSize = self.nvEnc.Width() * self.nvEnc.Height() * 3 / 2
                rawFrame = np.fromfile(self.rawFile, np.uint8, count=int(frameSize))
                if not (rawFrame.size):
                    print("No more video frames.")
                    break

                rawSurface = self.nvUpl.UploadSingleFrame(rawFrame)
                if rawSurface.Empty():
                    print("Failed to upload video frame to GPU.")
                    break

                cvtSurface = self.nvCvt.Execute(rawSurface, self.cc_ctx)
                if cvtSurface.Empty():
                    print("Failed to do color conversion.")
                    break
                self.nvEnc.EncodeSingleSurface(cvtSurface, self.encFrame)

            # Encoder is asynchronous, so we need to flush it
            success = self.nvEnc.Flush(self.encFrame)

        except Exception as e:
            print(getattr(e, "message", str(e)))
```

```
    def __init__(self, gpuID: int, width: int, height: int, rawFilePath: str):
        Thread.__init__(self)

        res = str(width) + "x" + str(height)

        # Initialize color conversion context.
        # Accurate color rendition doesn't matter in this sample so just use
        # most common bt601 and mpeg.
        self.cc_ctx = nvc.ColorspaceConversionContext(
            color_space=nvc.ColorSpace.BT_601, color_range=nvc.ColorRange.MPEG
        )

        self.nvUpl = nvc.PyFrameUploader(
            width, height, nvc.PixelFormat.YUV420, self.ctx.handle, self.str.handle
        )

        self.nvCvt = nvc.PySurfaceConverter(
            width,
            height,
            nvc.PixelFormat.YUV420,
            nvc.PixelFormat.NV12,
            self.ctx.handle,
            self.str.handle,
        )

        self.nvEnc = nvc.PyNvEncoder(
            {"preset": "P1", "codec": "h264", "s": res},
            self.ctx.handle,
            self.str.handle,
        )

        self.rawFile = open(rawFilePath, "rb")

        self.encFrame = np.ndarray(shape=(0), dtype=np.uint8)
```

1.6.6 Codec Surface to Numpy NDArray

```
def test_pycuda_memcpy_Surface_Surface(self):
        while True:
            surf_src = self.nvDec.DecodeSingleSurface()
            if surf_src.Empty():
                break
            src_plane = surf_src.PlanePtr()

            surf_dst = nvc.Surface.Make(
                self.nvDec.Format(),
                self.nvDec.Width(),
                self.nvDec.Height(),
                self.gpu_id,
            )
            self.assertFalse(surf_dst.Empty())
            dst_plane = surf_dst.PlanePtr()

            memcpy_2d = cuda.Memcpy2D()
            memcpy_2d.width_in_bytes = src_plane.Width() * src_plane.ElemSize()
            memcpy_2d.src_pitch = src_plane.Pitch()
            memcpy_2d.dst_pitch = dst_plane.Pitch()
            memcpy_2d.width = src_plane.Width()
            memcpy_2d.height = src_plane.Height()
            memcpy_2d.set_src_device(src_plane.GpuMem())
            memcpy_2d.set_dst_device(dst_plane.GpuMem())
            memcpy_2d(self.cuda_str)

            frame_src = np.ndarray(shape=(0), dtype=np.uint8)
            if not self.nvDwn.DownloadSingleSurface(surf_src, frame_src):
                self.fail("Failed to download decoded surface")

            frame_dst = np.ndarray(shape=(0), dtype=np.uint8)
            if not self.nvDwn.DownloadSingleSurface(surf_dst, frame_dst):
                self.fail("Failed to download decoded surface")

            if not np.array_equal(frame_src, frame_dst):
                self.fail("Video frames are not equal")
```

1.7 Autoencoder to TRT

Separate encoder/decoder from AutoencoderKL, for exporting them to ONNX and then to TensorRT, as exporting ONNX need them inherited from nn.Module.

```
class AutoencoderKL_Encoder(nn.Module):
    def __init__(self,
                 ddconfig,
                 embed_dim,):
        super().__init__()
        self.encoder = Encoder(**ddconfig)
        self.training = False
        self.quant_conv = torch.nn.Conv2d(2*ddconfig["z_channels"], 2*embed_dim, 1)

    def encode(self, x):
        h = self.encoder(x)
        moments = self.quant_conv(h)
        posterior = DiagonalGaussianDistribution(moments)
        return posterior

    def forward(self, input):
        posterior = self.encode(input)
        return posterior

class AutoencoderKL_Decoder(nn.Module):
    def __init__(self,
                 ddconfig,
                 embed_dim,):
        super().__init__()
        self.decoder = Decoder(**ddconfig)
        self.post_quant_conv = torch.nn.Conv2d(embed_dim, ddconfig["z_channels"], 1)

        self.training = False

    def decode(self, z):
        z = self.post_quant_conv(z)
        dec = self.decoder(z)
        return dec

    def forward(self, input):
        dec = self.decode(input)
        return dec
```

Using torch.onnx.export API to generate ONNX files, for decoder, the scripts is:

```python
def torch_onnx_export_autoencoder_decode( model, onnx_model_path ):

    if os.path.exists(onnx_model_path):
        check_onnx(onnx_model_path)
        return
    device = torch.device("cuda:0")
    dummy_inputs = {
        "input": torch.randn((16, 4, 32, 32),dtype=torch.float32).to(device),
    }
    output_names = ["posterior"]

    #import apex
    with torch.no_grad():
        with warnings.catch_warnings():
            warnings.filterwarnings("ignore", category=torch.jit.TracerWarning)
            warnings.filterwarnings("ignore", category=UserWarning)
            #with open(onnx_model_path, "wb") as f:
            with torch.onnx.select_model_mode_for_export(model, torch.onnx.TrainingMode.EVAL):
                torch.onnx.export(
                    model,
                    tuple(dummy_inputs.values()),
                    onnx_model_path, #f,
                    export_params=True,
                    verbose=True,
                    opset_version=12,
                    do_constant_folding=False,
                    input_names=list(dummy_inputs.keys()),
                    output_names=output_names,
                    #dynamic_axes=dynamic_axes,
                )
```

1.7.1.1 TRT engine from ONNX

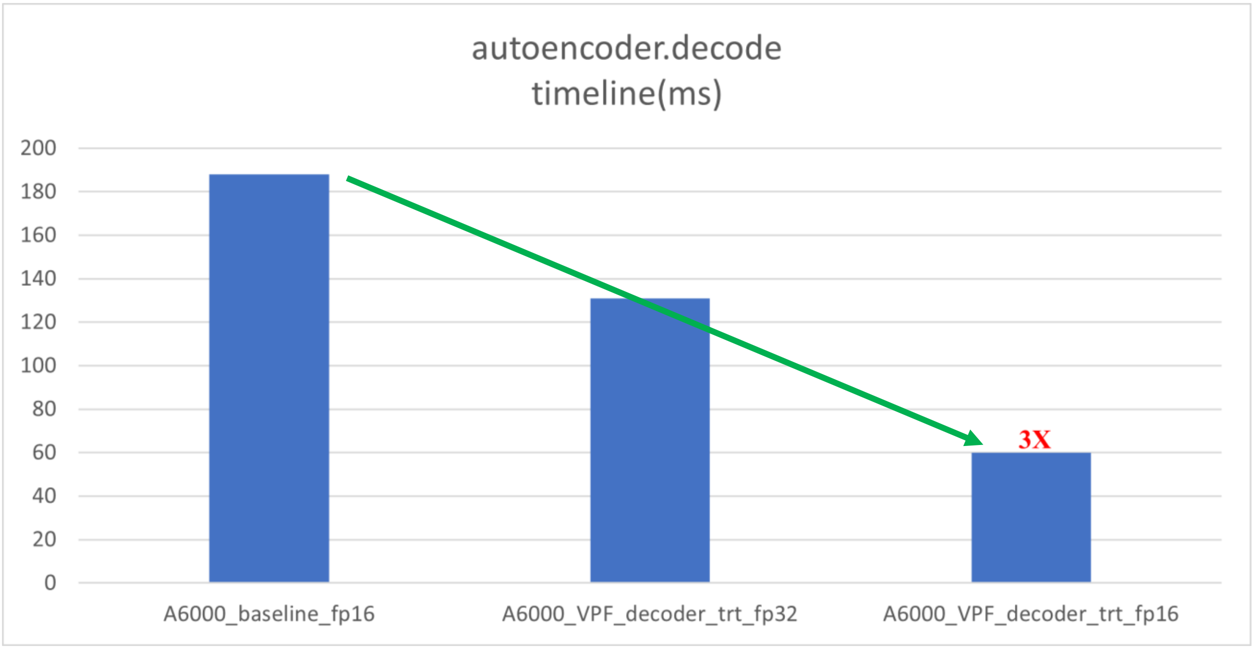
My script to build TRT engine from ONNX file:

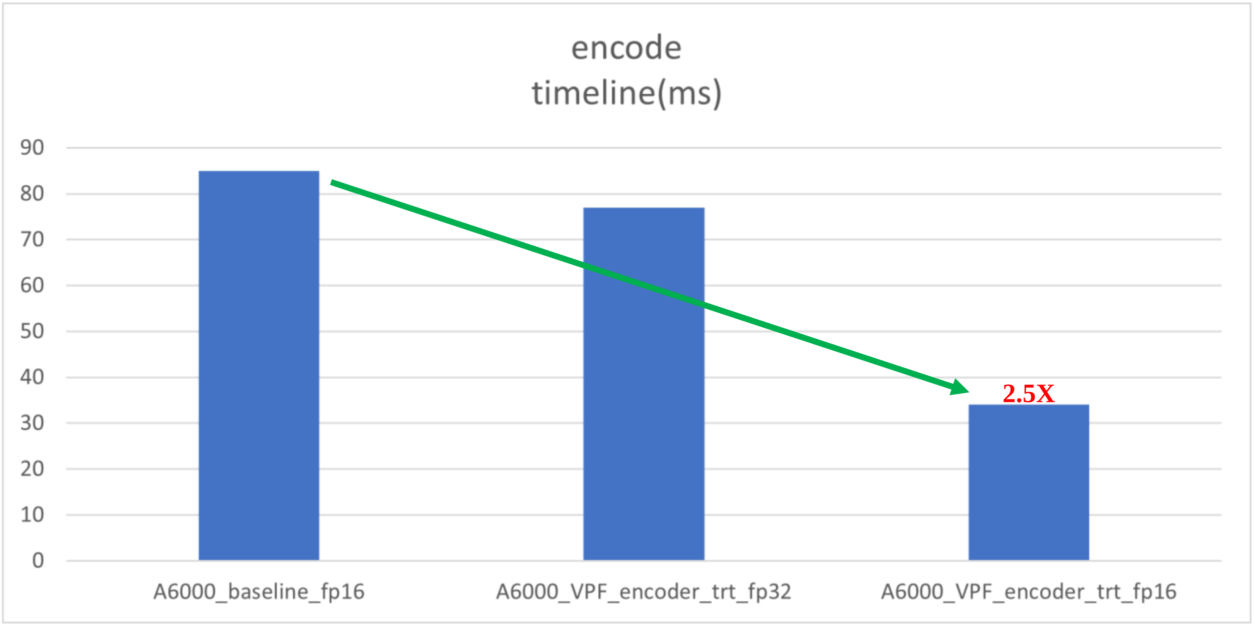
trtexec --onnx=./onnx/autoencoder_decode.onnx --saveEngine=./onnx/autoencoder_decode_fp32_6000Ada.engine --workspace=4096
trtexec --onnx=./onnx/autoencoder_decode.onnx --saveEngine=./onnx/autoencoder_decode_fp32_6000Ada.engine --workspace=4096
trtexec --onnx=./onnx/autoencoder_decode.onnx --saveEngine=./onnx/autoencoder_decode_fp16_6000Ada.engine --workspace=4096
trtexec --onnx=./onnx/autoencoder_decode.onnx --saveEngine=./onnx/autoencoder_decode_fp16_6000Ada.engine --workspace=4096

Just an example trtexec for reference:

```
./TensorRT-8.6.0.12/bin/trtexec --onnx=unet.onnx --minShapes=\'sample\':2x4x8x8,\'encoder_hidden_states\':2x77x1024 --
optShapes=\'sample\':4x4x64x64,\'encoder_hidden_states\':4x77x1024 --
maxShapes=\'sample\':8x4x96x96,\'encoder_hidden_states\':8x77x1024  --buildOnly --saveEngine=unet.plan --
memPoolSize=workspace:13888 --device=1 --fp16
```

1.7.2 Benchmark





1.8 Midas/deepthmap TRT optimizing

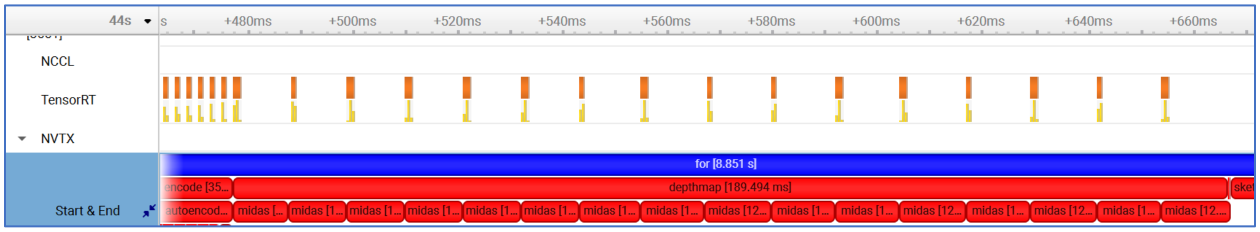
Create ONNX model from fp32 model, I can’t create ONNX model from its torch_model.half() conversion.

1.8.1 System configuration

```
>cd /your_name/projects/TRT-DeepSpeed/vid2vid/videocomposer/
```

```
>clear && CUDA_VISIBLE_DEVICES=0 /usr/bin/python3.10 run_net.py    --cfg
configs/exp02_motion_transfer.yaml    --seed 9999    --input_video "demo_video/motion_transfer.mp4"    --
image_path "demo_video/moon_on_water.jpg"    --input_text_desc "A beautiful big moon on the water at night"
>pip install onnx onnxruntime onnxruntime-gpu
>dpkg -i ../nv-tensorrt-local-repo-ubuntu1804-8.6.0-cuda-11.8_1.0-1_amd64.deb
>cp /var/nv-tensorrt-local-repo-ubuntu1804-8.6.0-cuda-11.8/nv-tensorrt-local-D279C523-keyring.gpg /usr/sh
are/keyrings/
>apt update && apt install tensorrt
>find /usr -name trtexec
>export PATH=/usr/src/tensorrt/bin/:$PATH
>trtexec --onnx=./onnx/midas.onnx --saveEngine=./onnx/midas_fp32_6000Ada.engine --workspace=4096
>trtexec --onnx=./onnx/midas.onnx --fp16 --saveEngine=./onnx/midas_fp16_6000Ada.engine --workspace=4096
```

1.8.2 Benchmark

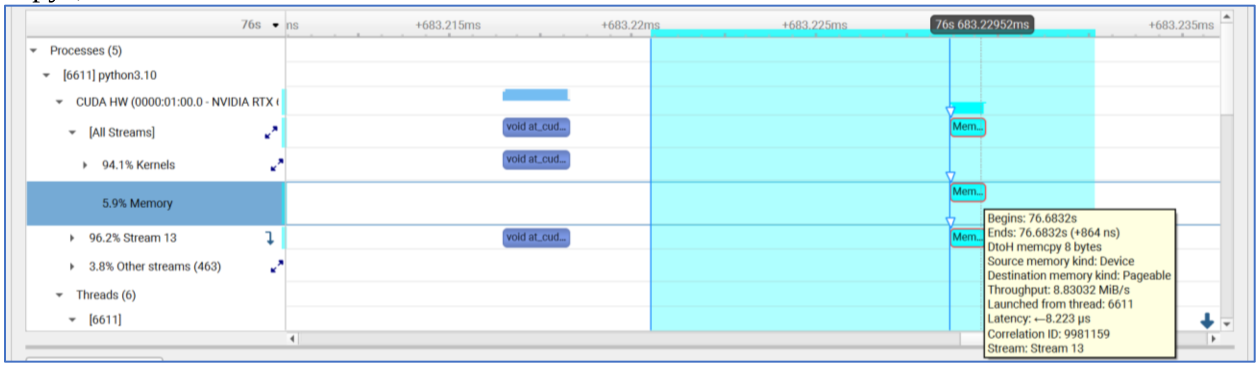


Fp32_trt

1.9 Unnormal D2H/H2D/D2D memory operations

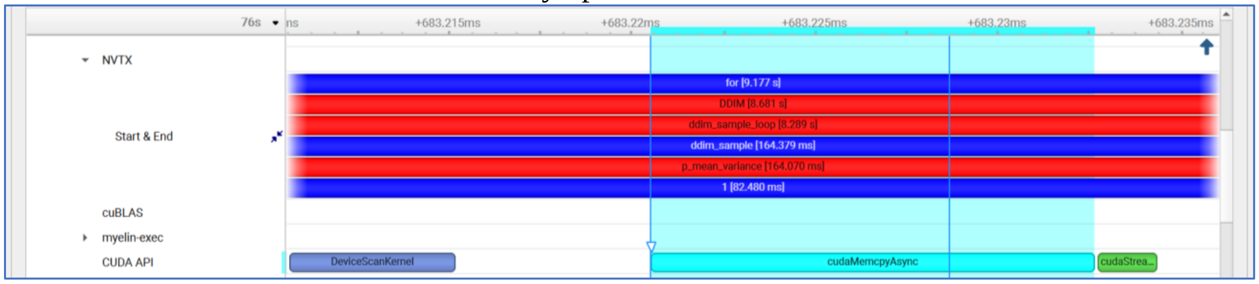
1.9.1

While calling _scale_timesteps in p_mean_variance, it involves unnormal D2H/H2D memory copys, looks like this:



Begins: 76.6832s
Ends: 76.6832s (+864 ns)
DtoH memcpy 8 bytes
Source memory kind: Device
Destination memory kind: Pageable
Throughput: 8.83032 MiB/s
Launched from thread: 6611
Latency: ←8.223 μs
Correlation ID: 9981159
Stream: Stream 13

Find its memory operations in cuda streams

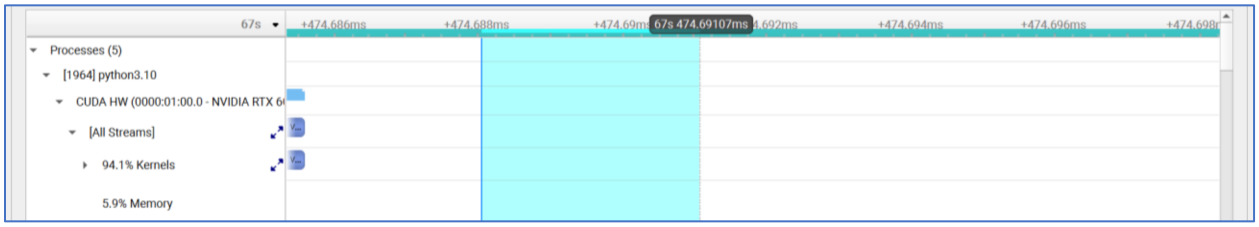


Find its corresponding CUDA API calling and nvtx annotation

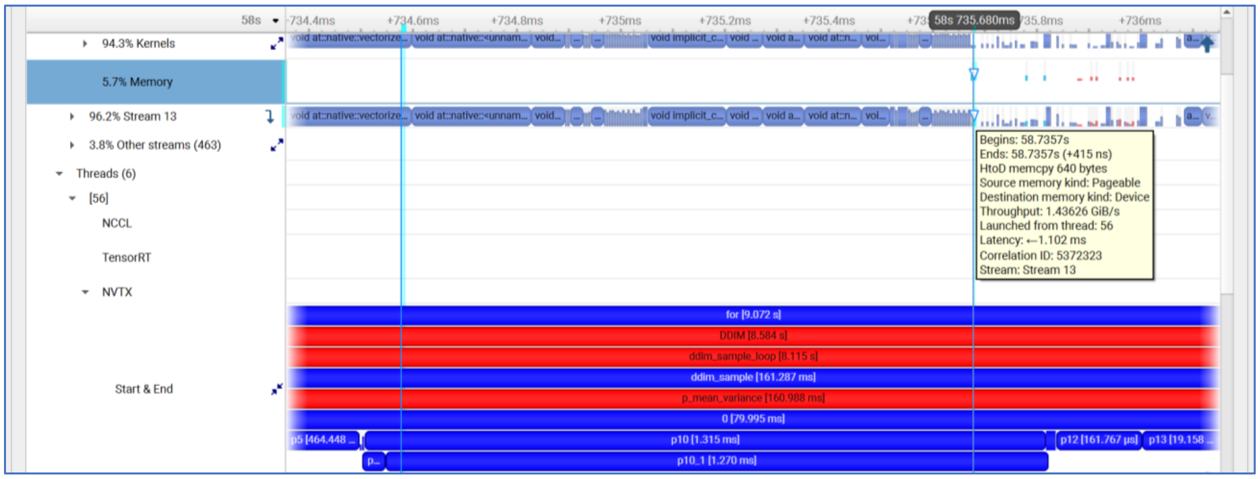
```
class UNetSD_temporal(nn.Module):
```

```python
def forward(self,
    ):
    ……
    if True:
        misc_dropout = partial(self.misc_dropout, zero = None, keep = None)
    else:
        zero = torch.zeros(batch, dtype=torch.bool).to(x.device)
        keep = torch.zeros(batch, dtype=torch.bool).to(x.device)
        if self.training:
            nzero = (torch.rand(batch) < self.p_all_zero).sum()
            nkeep = (torch.rand(batch) < self.p_all_keep).sum()
            index = torch.randperm(batch)
            zero[index[0:nzero]] = True
            keep[index[nzero:nzero + nkeep]] = True
        assert not (zero & keep).any()
        misc_dropout = partial(self.misc_dropout, zero = zero, keep = keep)
```

After double check source code, we find *zero*/*keep* is not used in inference, so we can commented out this safely. After replacing this, there is no memory operations at here in the timeline:

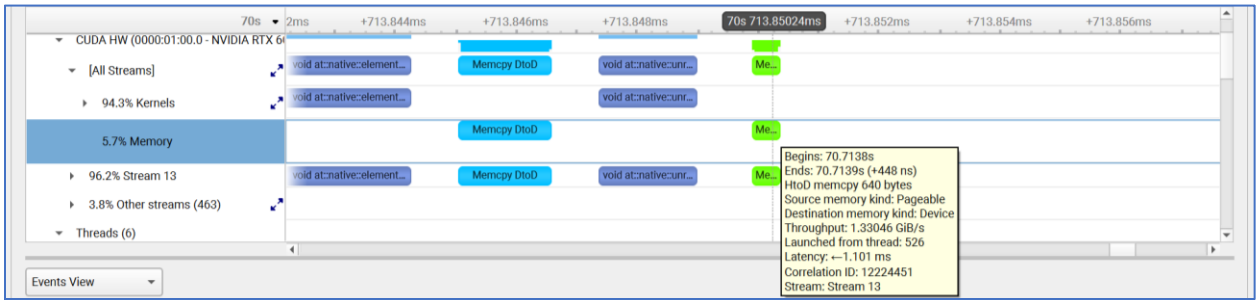


1.9.2



Begins: 58.7357s
Ends: 58.7357s (+415 ns)
HtoD memcpy 640 bytes
Source memory kind: Pageable
Destination memory kind: Device
Throughput: 1.43626 GiB/s
Launched from thread: 56
Latency: ←1.102 ms
Correlation ID: 5372323
Stream: Stream 13

Zoom out, to find the D2H/H2D drawback



Begins: 70.7138s
Ends: 70.7139s (+448 ns)
HtoD memcpy 640 bytes
Source memory kind: Pageable
Destination memory kind: Device
Throughput: 1.33046 GiB/s
Launched from thread: 526
Latency: ←1.101 ms
Correlation ID: 12224451
Stream: Stream 13

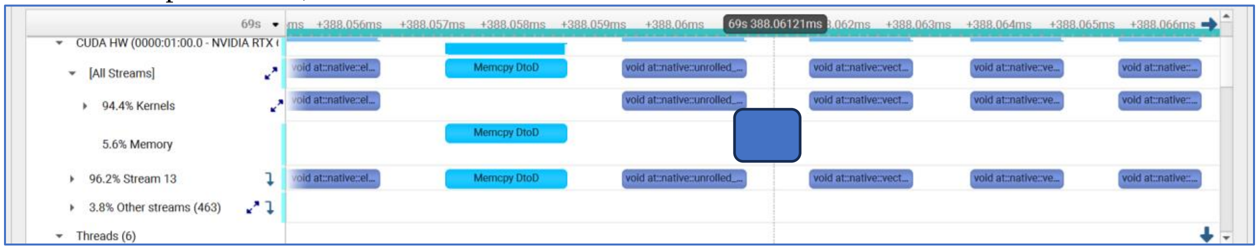
Zoom in, what the H2D copy is

In this circumstance, this H2D is from this line:

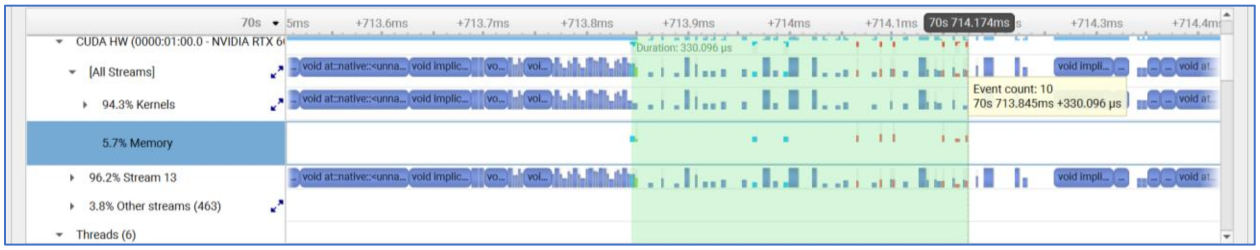
```
torch.pow(10000, -torch.arange(half).to(timesteps).div(half))
```

torch.arange create a tensor in system memory which we find can be pre-allocated and move to initialization stage and create in GPU device.

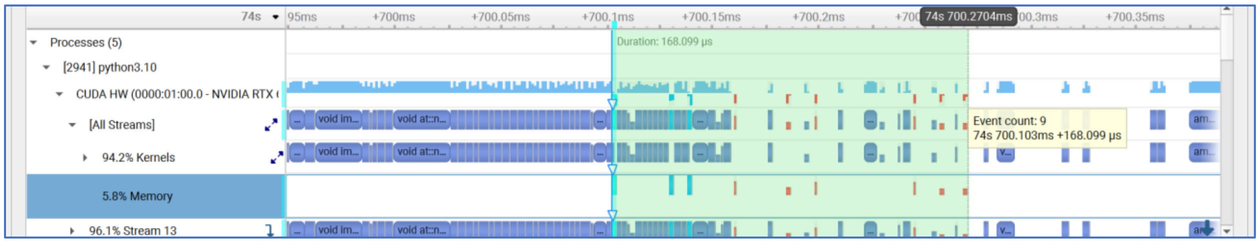
After this improvement, there is no H2D at here in the timeline:



And the neighboring kernels are getting much closer:



↓

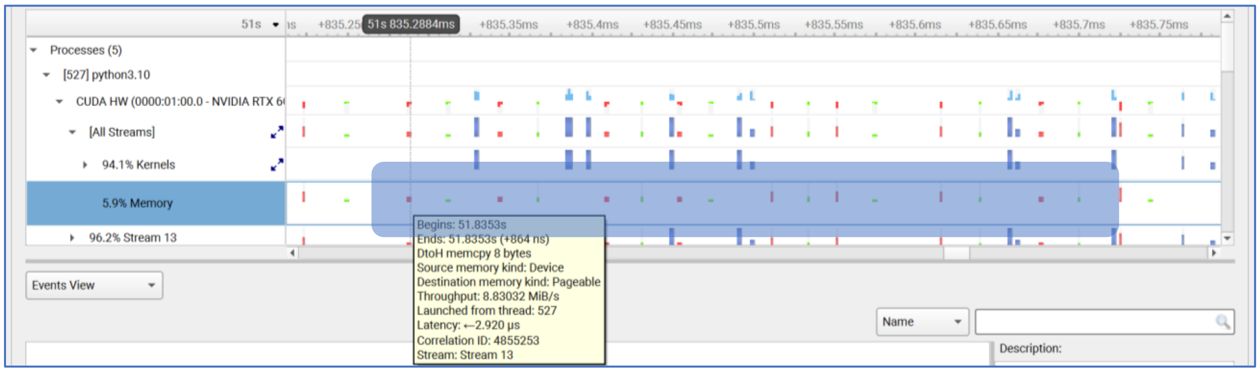


1.9.3 Double check tensors on same device

Torch supports tensors operations on different device, and this may make implicit data transfer between CPU and GPU, and CUDA stream synchronize. Like this code script shows:

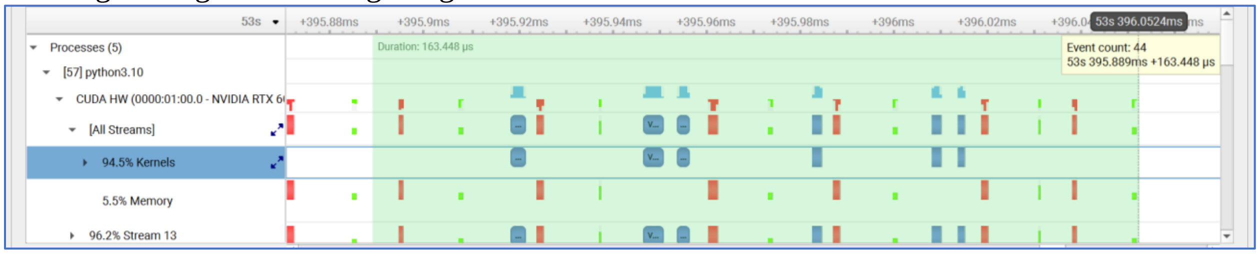
```
x0 = _i(self.sqrt_recip_alphas_cumprod, t, xt) * xt - \

                _i(self.sqrt_recipm1_alphas_cumprod, t, xt) * out
```

Where `self.sqrt_recip_alphas_cumprod` is on CPU and `t`/`xt` are on GPU. Implicit data transfer occurs on timeline:



Begins: 51.8353s
Ends: 51.8353s (+864 ns)
DtoH memcpy 8 bytes
Source memory kind: Device
Destination memory kind: Pageable
Throughput: 8.83032 MiB/s
Launched from thread: 527
Latency: ←2.920 µs
Correlation ID: 4855253
Stream: Stream 13

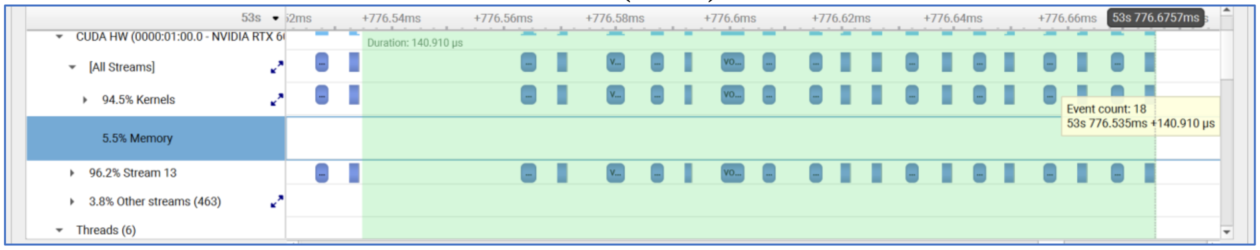
Initialize `self.sqrt_recip_alphas_cumprod` on GPU will make this drawback disappear and neighboring kernels are getting closer:



Duration: 163.448 µs

Event count: 44
53s 395.889ms +163.448 µs

Before(163us)



Duration: 140.910 µs

Event count: 18
53s 776.535ms +140.910 µs

After(140us), more CUDA kernels replacing CPU operations with no data transfer between CPU/GPU

1.9.4 Replace `repeat_interleave` with `expand`

Torch repeat_interleave is not very high performance designed and

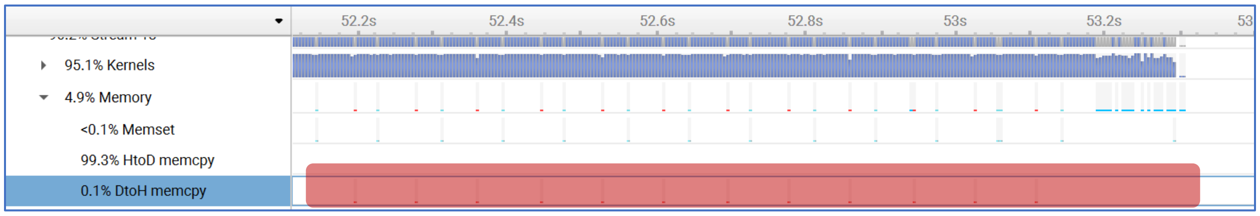
```
e = e.repeat_interleave(repeats=f, dim=0)
```

Is the same as

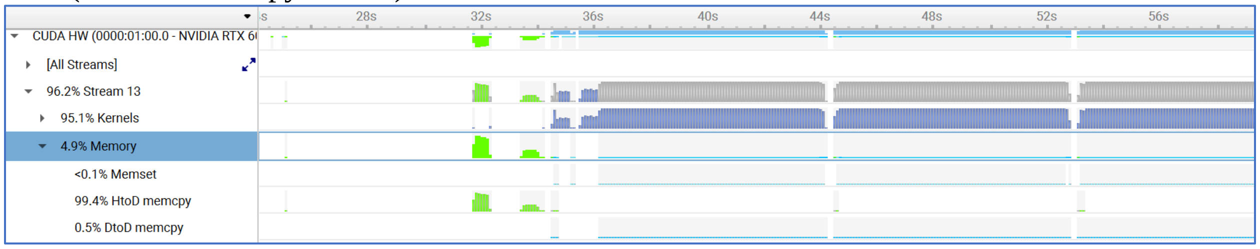
```
e_shape = list(e.shape)

e_shape[0] = e_shape[0]*f

e = e.expand(tuple(e_shape))
```

And data transfer between CPU/GPU inside of repeat_interleave() is gone in timeline.

Before:


95.1% Kernels
4.9% Memory
<0.1% Memset
99.3% HtoD memcpy
0.1% DtoH memcpy

After(no D2H memcpy at here):


CUDA HW (0000:01:00.0 - NVIDIA RTX 6
[All Streams]
96.2% Stream 13
95.1% Kernels
4.9% Memory
<0.1% Memset
99.4% HtoD memcpy
0.5% DtoD memcpy