TensorRT 教程

基于 8.6.1 版本

Part 5

去年基于 8.2.3 版本的教程视频：BV15Y4y1W73E
今年更新的内容会用 ## 标出

NVIDIA DevTech. Meng Wang

2023年7月6日

TensorRT性能优化

大纲

- **概述**
- 性能分析工具
- 性能优化技巧
- 性能优化实例

概述

- 性能优化的核心是充分发挥GPU算力
- Nsight system是分析性能瓶颈的关键工具
- trtexec除了构建engine，也是非常实用的性能测试工具
- 计算图优化和TRT plugin是性能优化的主要手段
- 本次讲座主要基于Framework->ONNX->TRT workflow

概述

GPU specifications

- 要充分发挥Tensor Core算力
- 数据类型：TF32，FP16，INT8，FP8

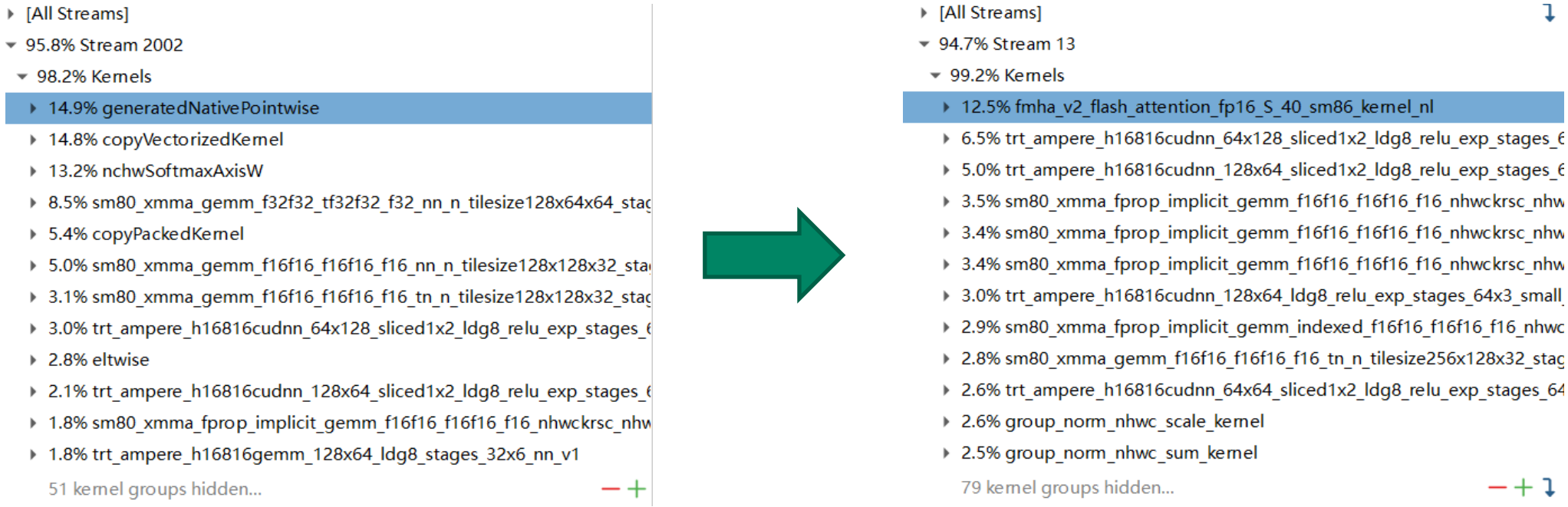
Technical Specifications			
	H100 SXM	**H100 PCIe**	**H100 NVL[1]**
FP64	34 teraFLOPS	26 teraFLOPS	68 teraFLOPS
FP64 Tensor Core	67 teraFLOPS	51 teraFLOPS	134 teraFLOPS
FP32	67 teraFLOPS	51 teraFLOPS	134 teraFLOPS
TF32 Tensor Core	989 teraFLOPS[2]	756 teraFLOPS[2]	1,979 teraFLOPS[2]
BFLOAT16 Tensor Core	1,979 teraFLOPS[2]	1,513 teraFLOPS[2]	3,958 teraFLOPS[2]
FP16 Tensor Core	1,979 teraFLOPS[2]	1,513 teraFLOPS[2]	3,958 teraFLOPS[2]
FP8 Tensor Core	3,958 teraFLOPS[2]	3,026 teraFLOPS[2]	7,916 teraFLOPS[2]
INT8 Tensor Core	3,958 TOPS[2]	3,026 TOPS[2]	7,916 TOPS[2]
GPU memory	80GB	80GB	188GB
GPU memory bandwidth	3.35TB/s	2TB/s	7.8TB/s[3]

https://resources.nvidia.com/en-us-tensor-core/nvidia-tensor-core-gpu-datasheet

概述

优化目标

- 推理框架的性能优化的目标：
  - 尽可能地把所有非GEMM kernel融合起来
  - GEMM kernel (Tensor Core)占比较高，例如在90%以上



概述

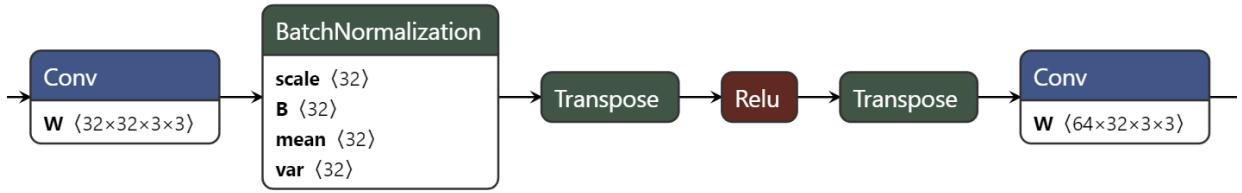
优化流程

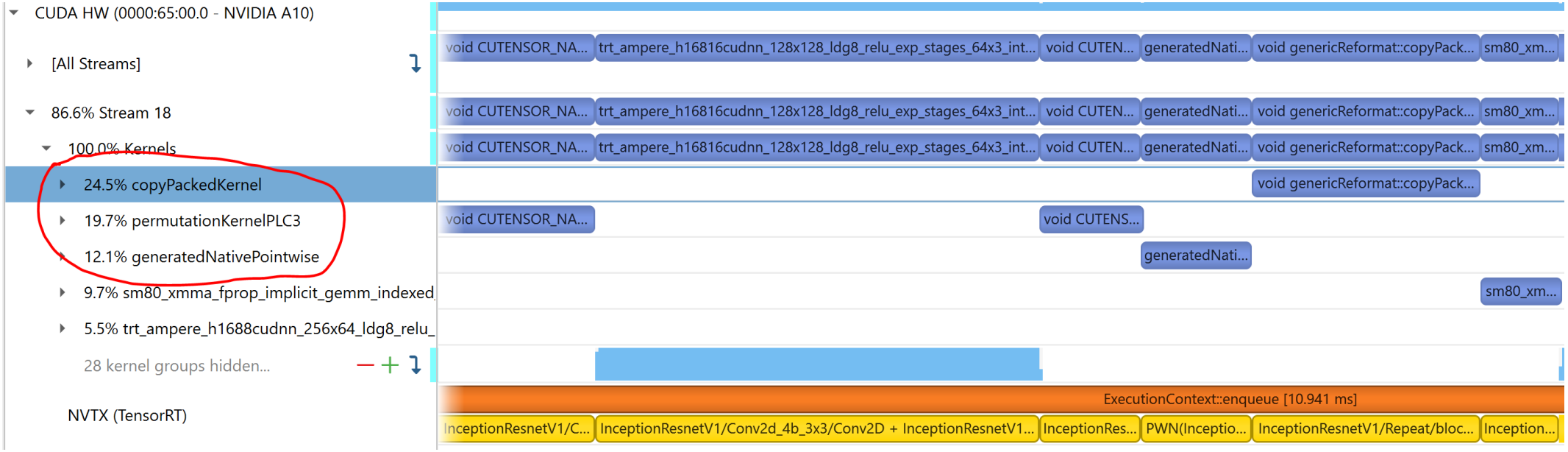
- 优化流程
  1. Run Framework->ONNX->TRT to get baseline
  2. Profile and find perf bottleneck
  3. Optimize with ONNX-graphsurgeon and TRT plugin
  - ➢ Repeat step2 and step3 until satisfied

概述

A simple optimization example

- FaceNet
  - CNN网络，人脸识别
- 性能瓶颈：
  - 重复的Transpose
  - 没有融合Conv+BatchNorm+Relu



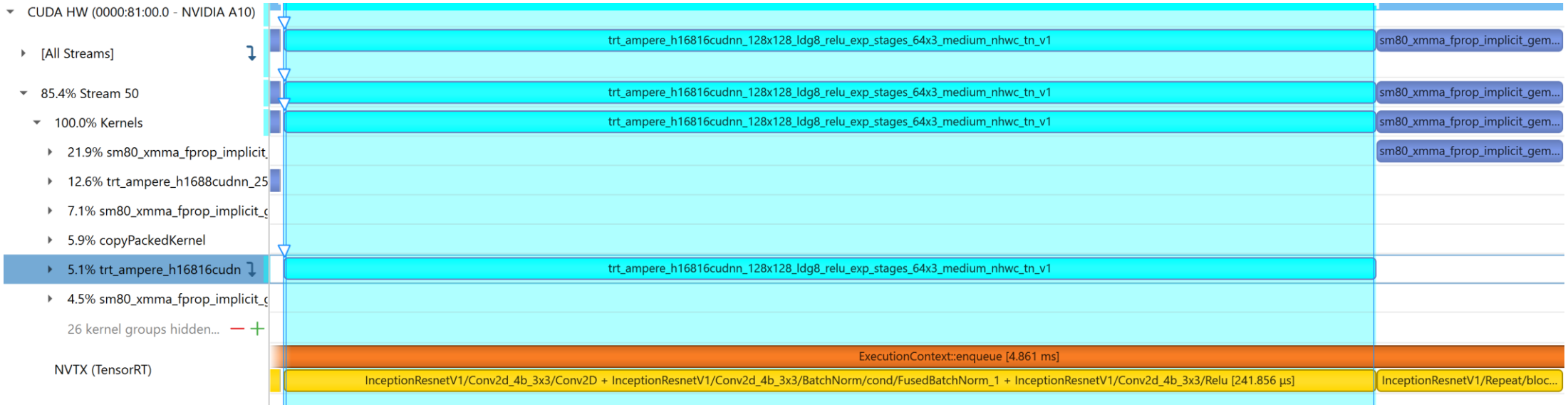


概述

A simple optimization example

- 解决方案：
  - 通过onnx-graphsurgeon去除重复的transpose
- 代码示例：
- 优化效果：
  - 去除Transpose后实现两倍吞吐
  - 使用INT8和multi stream可以达到四倍吞吐

```python
graph = gs.import_onnx(onnx.load("model.onnx"))
for node in graph.nodes:
    if "Relu" in node.op:
        if node.i().op == "Transpose" and node.o().op == "Transpose":
            node.inputs = node.i().inputs
            next_node = node.o()
            node.outputs = []
            next_node.outputs[0].inputs[0] = node
            graph.cleanup()
```



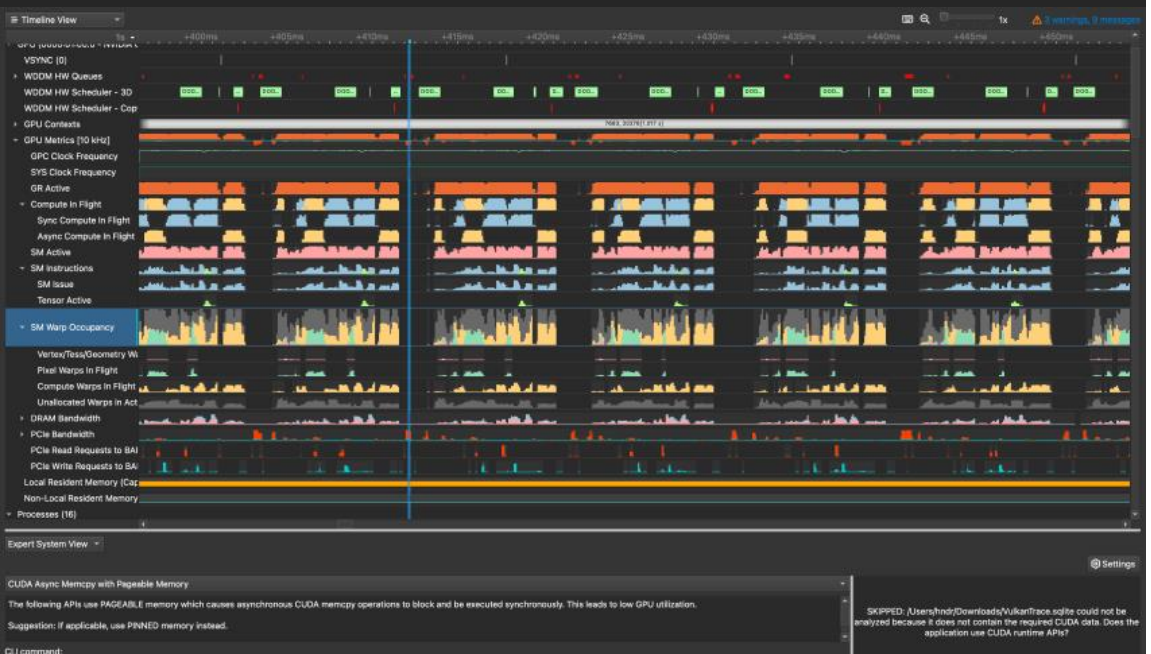
TensorRT性能优化

大纲

- 概述
- **性能分析工具**
- 性能优化技巧
- 性能优化实例

Nsight System

Nsight Compute



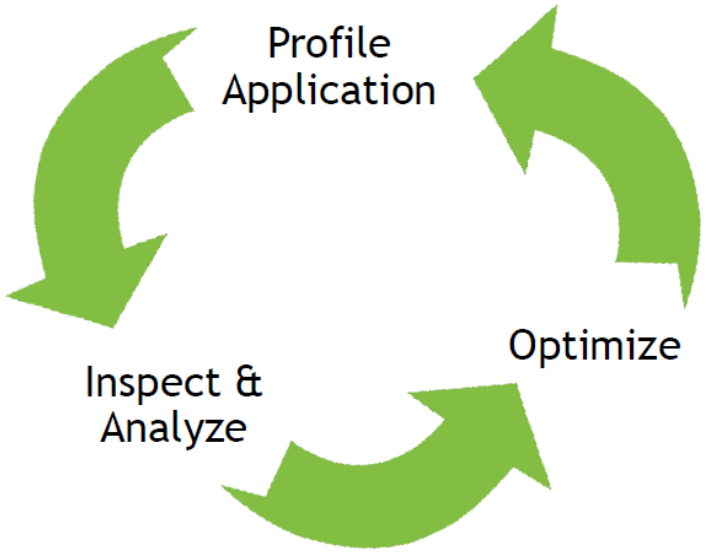
Profiling

- Nsight system
- NVTX
- Nsight compute
- Understand nsys timeline
- trtexec
- Trex

Profiling

To find the bottleneck

- Nsight systems
  - A system-wide performance analysis tool
- NVTX
  - User annotation API that adds marks in profiling
- Nsight compute
  - An interactive kernel profiler for CUDA applications



https://developer.nvidia.com/nsight-systems
https://developer.nvidia.com/nsight-compute
https://resources.nvidia.com/ent-gtccn20/gtccn20-CNS20632 *《深入理解Nsight System 与Nsight Compute 性能分析优化工具》*

Nsight System

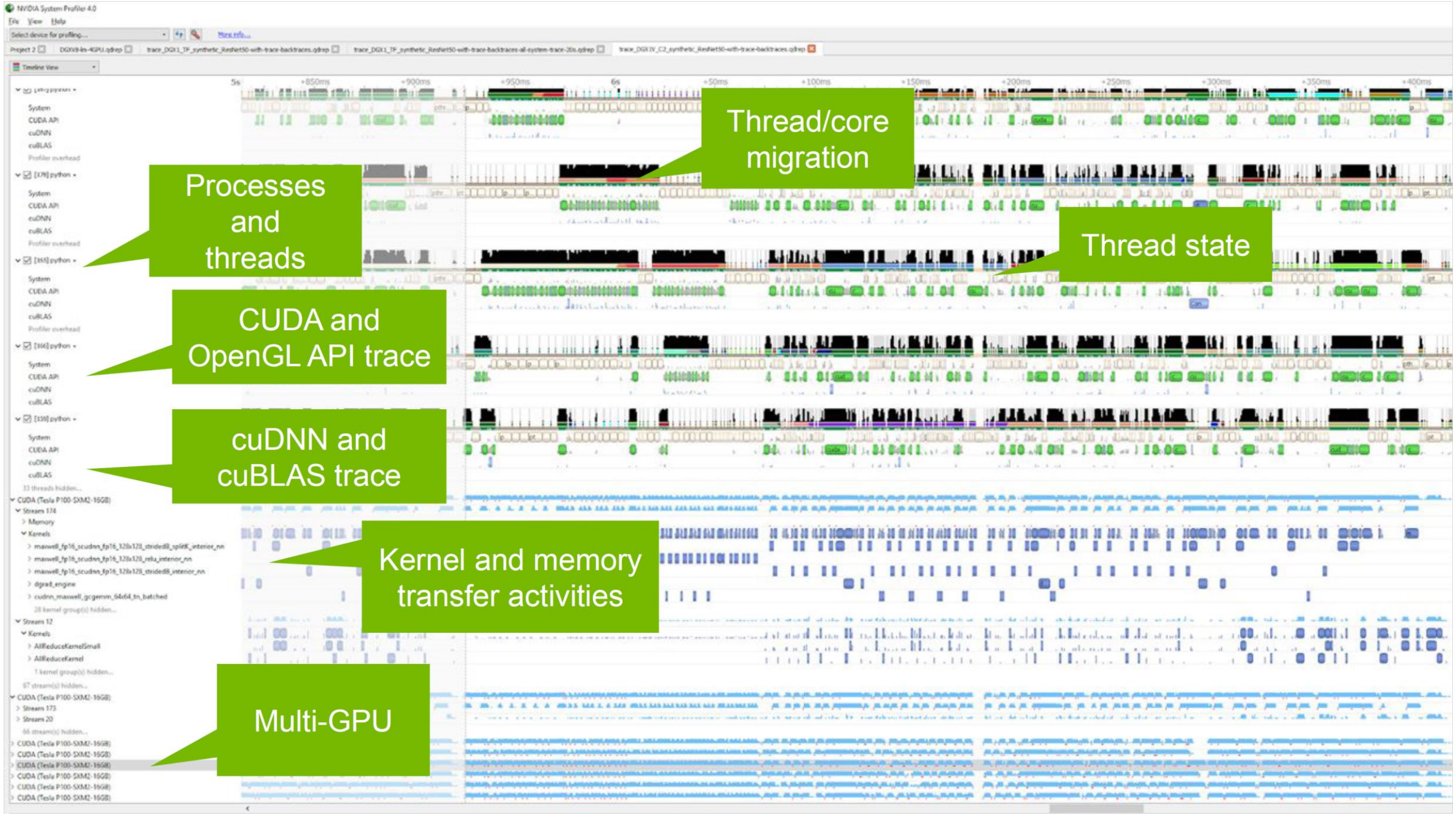
Overview

- System-wide application
- Locate optimization opportunities
  - See gaps of unused CPU and GPU time
- Balance your workload across multiple CPUs and GPUs
  - GPU streams, kernels, memory transfers, etc.
- Support for Linux & windows, x86-64

Nsight System

- Compute
  - CUDA API, kernel launch and execution
  - Libraries: cuBLAS, cuDNN, TensorRT

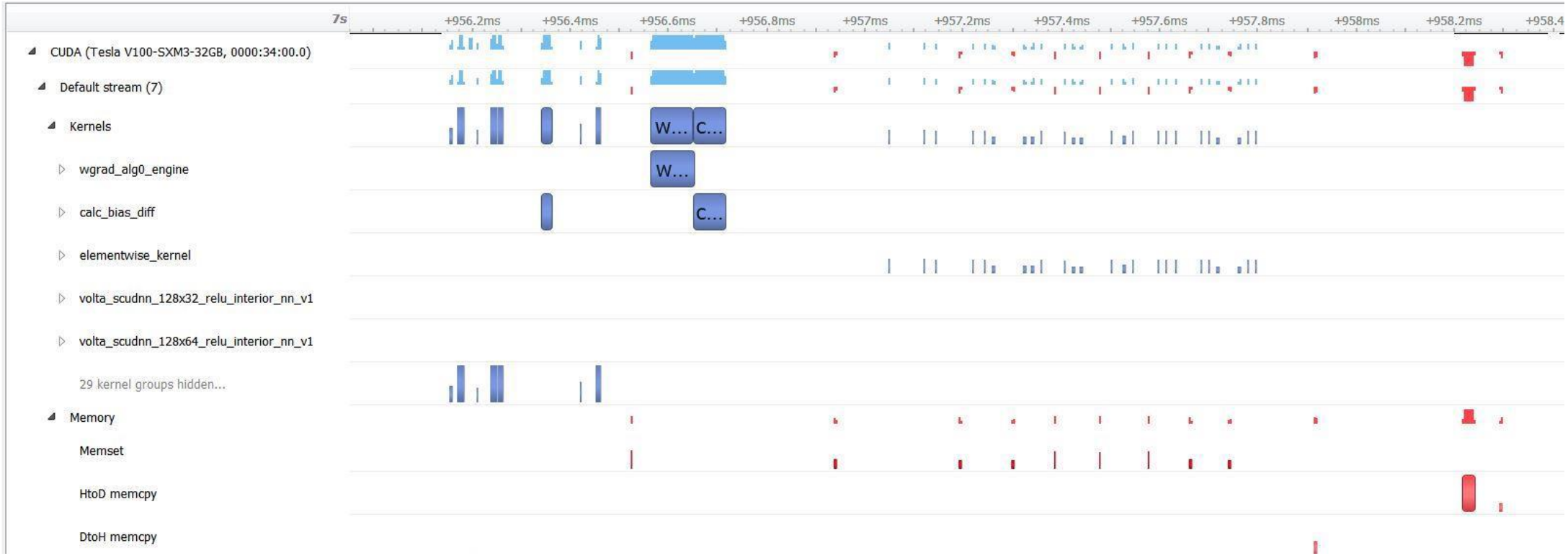
https://developer.nvidia.com/nsight-systems
https://resources.nvidia.com/ent-gtccn20/gtccn20-CNS20632 *《深入理解Nsight System 与Nsight Compute 性能分析优化工具》*


Thread/core migration
Processes and threads
Thread state
CUDA and OpenGL API trace
cuDNN and cuBLAS trace
Kernel and memory transfer activities
Multi-GPU

GPU Workload

In nsight system

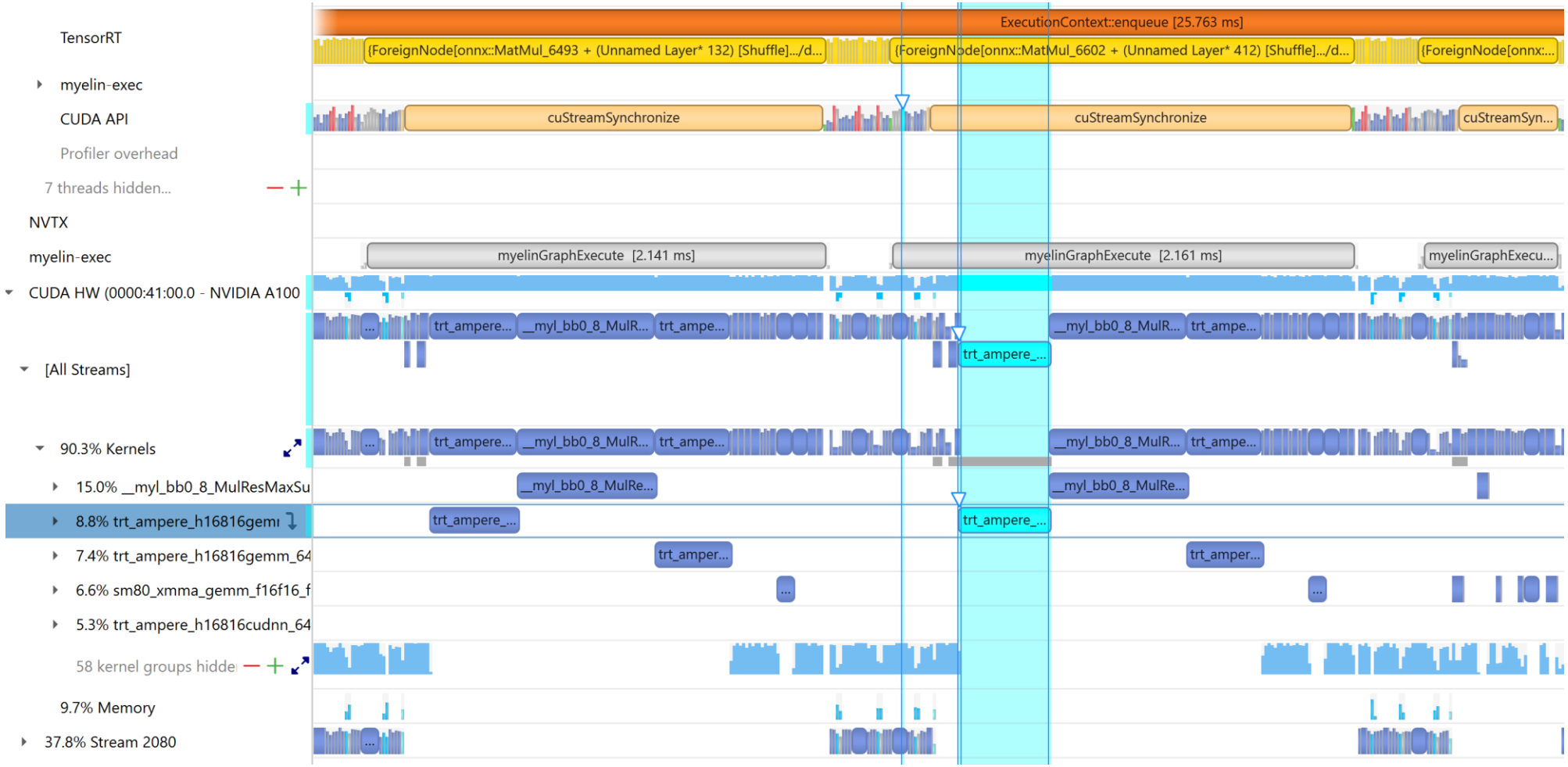
- See CUDA workloads execution time
- Locate idle GPU times



Correlation Ties API to GPU Workload

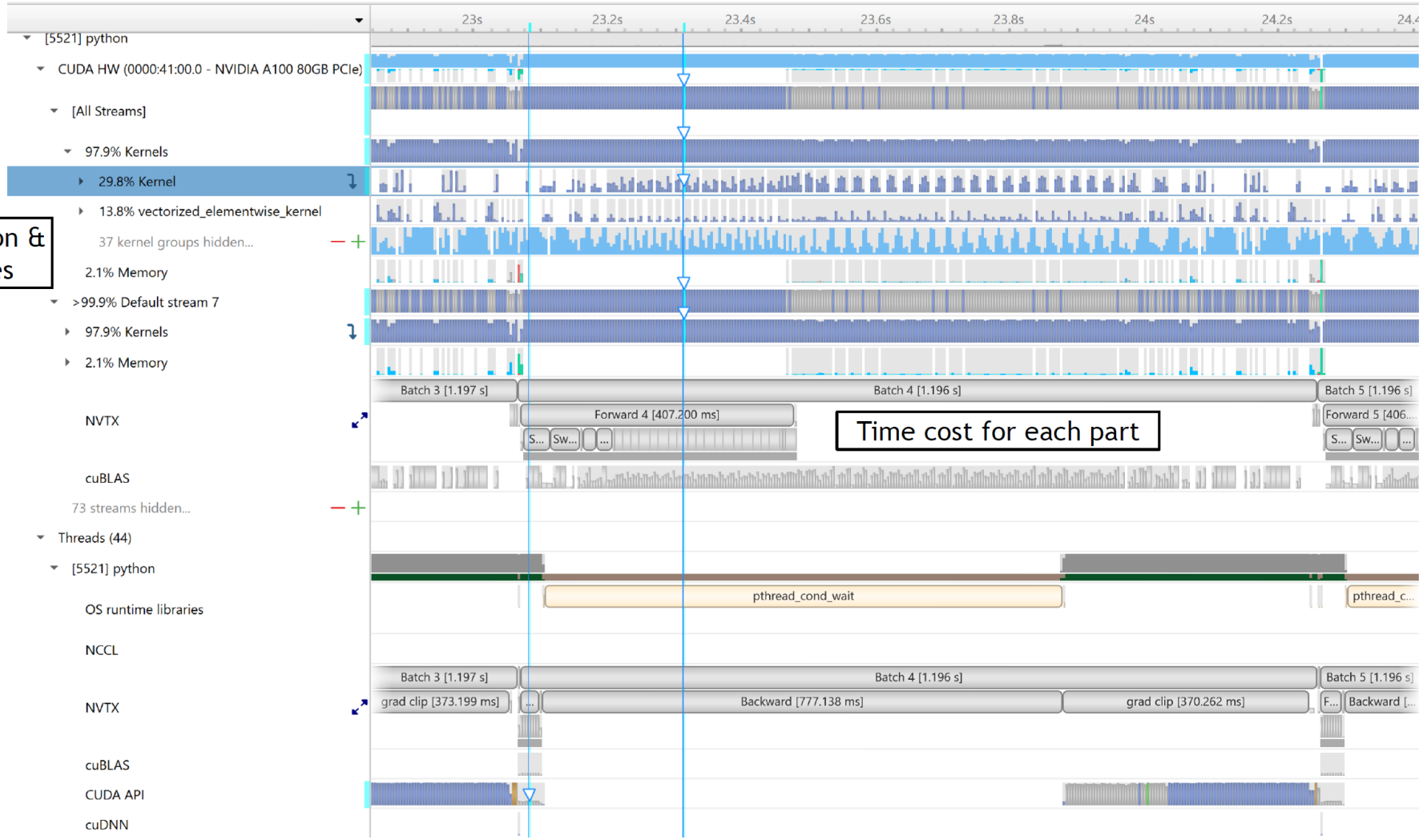
Click and highlight

- Automatically bound



NVTX Instrumentation

Take SwinTransformer as an example



Time occupation & Kernel names

Time cost for each part

Kernel runtime

Kernel launch

Nsight Compute

Next-Gen Kernel Profiling Tool

- Interactive kernel profiler
  - Graphical profile report. For example, the SOL and Memory Chart
  - Differentiating results across one or multiple reports using baselines
  - Fast Data Collection

- The UI executable is called **nv-nsight-cu**, and the command-line one is **nv-nsight-cu-cli**
- GPUs: Pascal, Volta, **Turing, Ampere**

Nsight Compute

https://developer.nvidia.com/nsight-compute
https://resources.nvidia.com/ent-gtccn20/gtccn20-CNS20632 *《深入理解Nsight System 与Nsight Compute 性能分析优化工具》*

File Connection Debug Profile Tools Window Help

Connect Disconnect Terminate Profile Kernel

API Stream

11236 > vectorAdd

Next Trigger: vector

ID	API Name	Details	Fu
186	cuDeviceGetAttribute		CU
187	cuDeviceGetAttribute		CU
188	cuDeviceGetAttribute		CU
189	cuDeviceGetAttribute		CU
190	cuDeviceGetAttribute		CU
191	cuDeviceGetAttribute		CU
192	cudaMalloc		cu
193	cuCtxGetCurrent		CU
194	cuCtxSetCurrent		CU
195	cuDevicePrimaryCtxRe…		CU
196	cuCtxGetCurrent		CU
197	cuCtxGetDevice		CU
198	cuModuleGetFunction		CU
199	cuMemAlloc_v2		CU
200	cudaMalloc		cu
201	cuMemAlloc_v2		CU
202	cudaMalloc		cu
203	cuMemAlloc_v2		CU
204	cudaMemcpy		cu
205	cuMemcpyHtoD_v2		CU
206	cudaMemcpy		cu
207	cuMemcpyHtoD_v2		CU
208	cudaConfigureCall		cu
209	cudaSetupArgument		cu
210	cudaSetupArgument		cu
211	cudaSetupArgument		cu
212	cudaSetupArgument		cu
213	cudaLaunch		
214	cuLaunchKernel		
215	vectorAdd	vectorAdd	

Sections/Rules Info

Reload Enable All Disable All

Enter filter

Name	Pric
Memory Workload Analysis	7
Memory Workload Analysis Chart	7
Memory Workload Analysis Tables	7
Scheduler Statistics (1)	8
Warp State Statistics (17)	9
Instruction Statistics	10

Sections/Rules Info | API Statistics | NVTX

Launch: 0 - 215 - vectorAdd

Copy as Image

Current 215 - vectorAdd (50176, 1, 1) T…

Frequency: 883.21 cycle/usecond CC: 7.0 Process: [11236] vectorAdd

High-level overview of the utilization for compute and memory resources of the GPU. For each unit, the Speed Of Light (SOL) reports the achieved percentage of utilization with respect to the theoretical maximum.

SOL SM [%]	3.72	Duration [usecond]	4.77
SOL Memory [%]	38.46	Elapsed Cycles [cycle]	4,232
SOL TEX [%]	6.60	SM Active Cycles [cycle]	2,368.09
SOL L2 [%]	9.77	SM Frequency [cycle/usecond]	883.21
SOL FB [%]	38.46	Memory Frequency [cycle/usecond]	626.40

GPU Utilization

Recommendations

Bottleneck [Warning] This kernel grid is too small to fill the available resources on this device. Look at `Launch Statistics` for more details.

Compute Workload Analysis

Detailed analysis of the compute resources of the streaming multiprocessors (SM), including the achieved instructions per clock (IPC) and the utilization of each available pipeline. Pipelines with very high utilization might limit the overall performance.

Executed Ipc Elapsed [inst/cycle]	6.61
Executed Ipc Active [inst/cycle]	3.67
Issued Ipc Active [inst/cycle]	4.48

Memory Workload Analysis All

Detailed analysis of the memory resources of the GPU. Memory can become a limiting factor for the overall kernel performance when fully utilizing the involved hardware units (Mem Busy), exhausting the available communication bandwidth between those units (Max Bandwidth), or by reaching the maximum throughput of issuing memory instructions (Mem Pipes Busy). Detailed chart of the memory units. Detailed tables with data for each memory unit.

Memory Throughput [Gbyte/second]	185.04	Mem Busy [%]	9.77
L1 Hit Rate [%]	0	Max Bandwidth [%]	38.46
L2 Hit Rate [%]	34.75	Mem Pipes Busy [%]	2.32

Scheduler Statistics

Summary of the activity of the schedulers issuing instructions. Each scheduler maintains a pool of warps that it can issue instructions for. The upper bound of warps in the pool (Theoretical Warps) is limited by the launch configuration. On every cycle each scheduler checks the state of the allocated warps in the pool (Active Warps). Active warps that are not stalled (Eligible Warps) are ready to issue their next instruction. From the set of eligible warps the scheduler selects a single warp from which to issue one or more instructions (Issued Warp). On cycles with no eligible warps, the issue slot is skipped and no instruction is issued. Having many skipped issue slots indicates poor latency hiding.

Active Warps Per Scheduler [warp]	4.26	Instructions Per Active Issue Slot [inst/cycle]	1
Eligible Warps Per Scheduler [warp]	0.05	No Eligible [%]	96.17
Issued Warp Per Scheduler	0.04	One or More Eligible [%]	3.83

Warp State Statistics

Analysis of the states in which all warps spent cycles during the kernel execution. The warp states describe a warp's readiness or inability to issue its next instruction. The warp cycles per instruction define the latency between two consecutive instructions. The higher the value, the more warp parallelism is required to hide this latency. For each warp state, the chart shows the average number of cycles spent in that state per issued instruction. Stalls are not always impacting the overall performance nor are they completely avoidable. Only focus on stall reasons if the schedulers fail to issue every cycle.

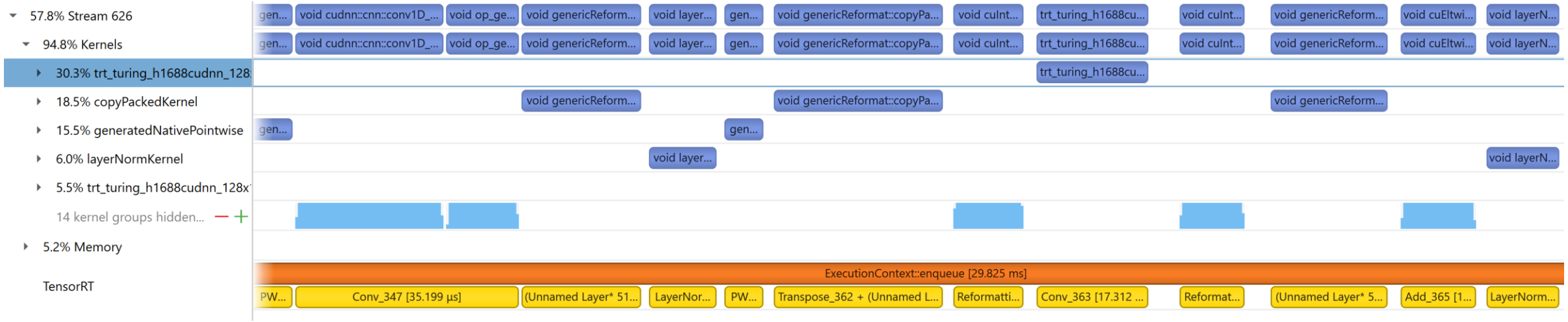
Warp Cycles Per Issued Instruction	111.19	Avg. Active Threads Per Warp	31.99

vectorAdd [11236]

解读nsys timeline

How to map CUDA kernel to model layer

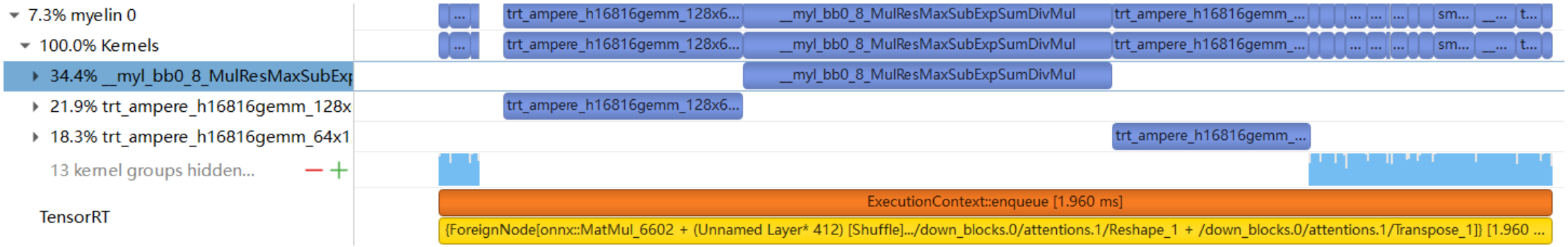
- 因为各种优化，nsys timeline显示的CUDA kernel和模型节点并不能一一对应，我们需要充分理解TRT的行为，进行快速匹配
- TRT自带NVTX，根据layer name匹配kernel和layer
- TRT在graph optimization过程中会进行大量的layer fusion，NVTX显示layer1 name + layer2 name + …
- TRT在build engine过程中会自动加入一些reformat layer，这些reformat layer在原模型中并不存在，NVTX显示Unnamed Layer
- TRT对pointwise类算子可以自动生成generatedNativePointwise kernel



解读nsys timeline

How to map CUDA kernel to model layer

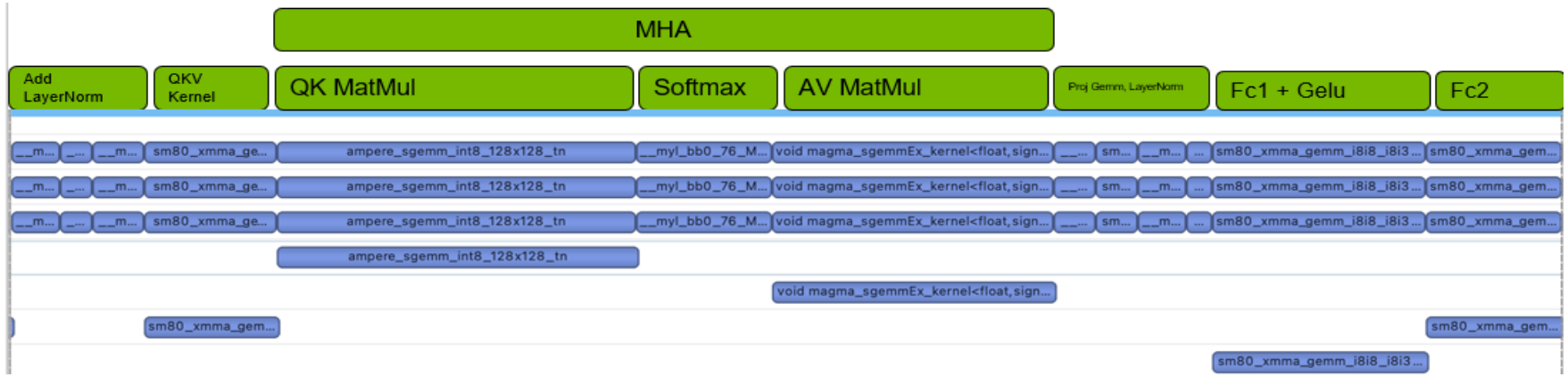
- Myelin作为一个deep learning compiler，有独特的layer和kernel的命名规则
  - TRT 8.6之前Myelin nodes只有非常简单的NVTX tag，类似{ForeignNode[onnx::MatMul_6493 + (Unnamed Layer* 132) [Shuffle]…/down_blocks.0/attentions.0/Reshape_1 + /down_blocks.0/attentions.0/Transpose_1]}，难以解读
  - TRT 8.6开始Myelin nodes增加了更细力度的NVTX tag，比如$$myelin$$/time_embedding/act/sigmoid，便于匹配
  - TRT会持续优化Myelin的可理解性
- Myelin fused kernel命名规则
  - 以LayerNorm为例，**__myl_bb0_8_ResTraCasAddMeaSubMulMeaAddSqrDivMulMulAddRes_com_vec**，可以通过两个Mea (reduce_mean)识别，另外Res代表reshape，Tra代表transpose
- 寻找关键节点，比如Layernorm和MHA block的Softmax
  - Softmax对应的Myelin fused kernel一般命名为**ResNegExpResAddDivMul**



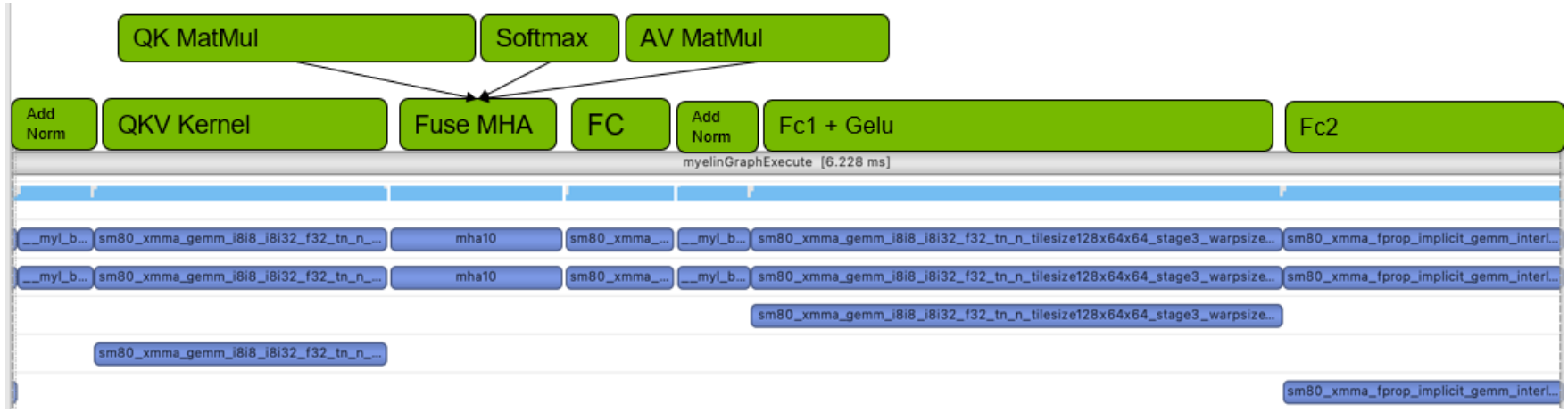
解读nsys timeline

How to map CUDA kernel to model layer

Unfused MHA



Fused MHA



通过trtexec做性能分析

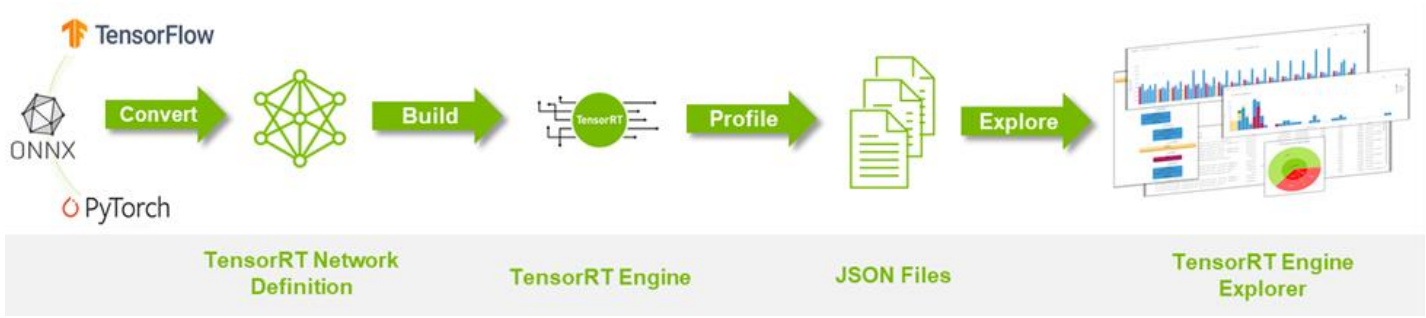
trtexec

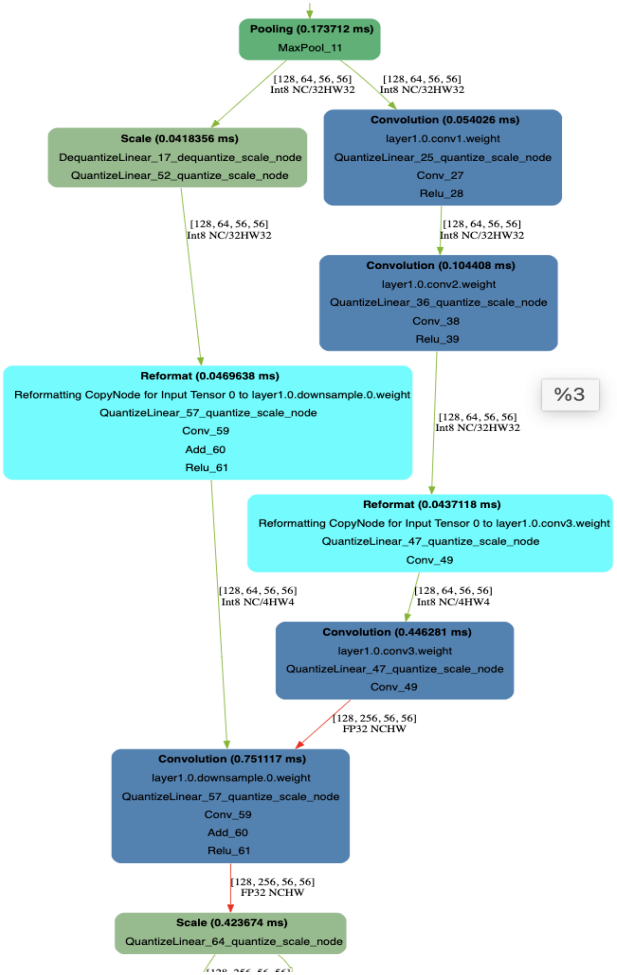
- 常用的参数
  - --verbose: Enable verbose logging.
  - --nvtxMode=verbose: Enable verbose NVTX marking so that you can see layer details in Nsys profile.
  - --dumpProfile --separateProfileRun: Gather per-layer perf profile.
  - --useCudaGraph: Enable CUDA graph to reduce enqueue time. If graph capturing fails, it will automatically fall back to non-graph path.
  - --noDataTransfers: Disable H2D/D2H data copies. Add this if you do not care about H2D/D2H transfers.
  - --warmUp=<N>: Controls how many ms to run warm-up. This part is not included in the perf metrics.
  - --duration=<N>: Controls how many seconds to run the inference for minimally. Set this to 0 if you only want to run a few iterations.
  - --iterations=<N>: Set the minimal number of iterations to run the inference for. If N iterations have reached but "duration" has not been reached yet, trtexec will continue to run the inference until "duration" is reached.

性能分析工具

Trex

- TensorRT Engine可视化工具：Trex





TensorRT性能优化

大纲

- 概述
- 性能分析工具
- **性能优化技巧**
- 性能优化实例

性能优化技巧

General Tips

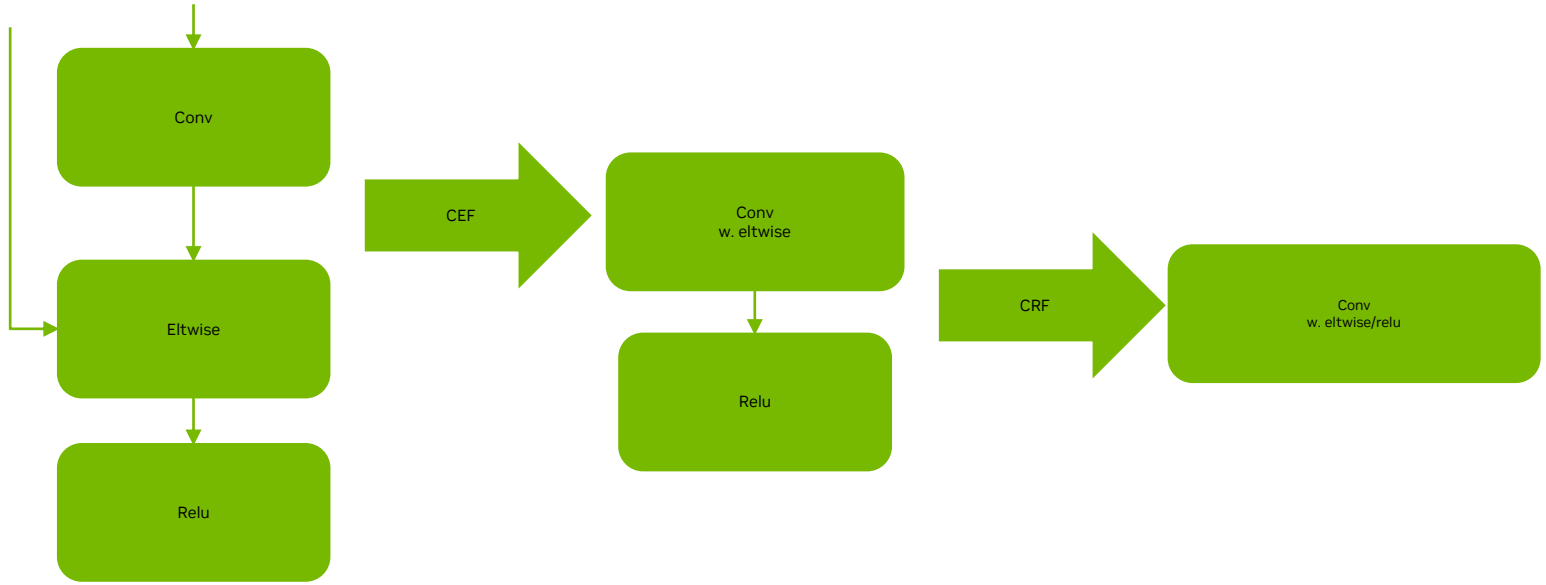
- Batching
  - 增大batch size可以更好的发挥GPU算力，从而达到更高吞吐
  - 如果吞吐随着batch size线性增加，一般说明GPU已打满
- Stream
  - 单个execution context内多流并行：TRT 8.6提供了IBuilderConfig::setMaxAuxStreams() API 来设置steam数量
  - 多个execution context间多流并行：设置不同stream
  - 尽量避免使用default stream，否则会引入额外的implicit sync
- CUDA Graph
  - 对于kernel launch bounded情况，可以通过CUDA graph来加速
  - 不支持dynamic shape，需要capture多个CUDA Graph
  - 设置nsys --cuda-graph-trace=node查看kernel
- Overhead of Shape Change and Optimization Profile Switching
  - 当切换execution context或者改变graph input shape后，TRT可能会有额外的开销
  - 可以尝试禁掉 kEDGE_MASK_CONVOLUTIONS tactic source，或者通过多个execution context切换使用来隐藏overhead

https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html#optimize-performance

性能优化技巧

Graph Fusion

- Graph fusion是非常有效的手段，TRT内部支持了丰富的fusion pattern，比如经典的conv+bias+relu
- 如果你发现了通用而且有明显收益的pattern，但是TRT没有fuse，建议发个bug
- 我们可以通过onnx-graphsurgeon和TRT plugin来手动实现graph fusion，比如经典的LayerNorm plugin
- 我们可以利用TRT外部的资源，比如Cutlass，FlashAttn， xformer，将kernel implementation封装到TRT plugin



性能优化技巧

Optimizing Layer Performance

- Reduce Layer
  - 尽量保证reduce axis在最后一维，这样能保证数据连续性，从而实现更好的性能
  - 如果reduce axis不在最后一维，可以通过Shuffle (transpose) 调整
- Shuffle Layer
  - 如果input和output tensor有相同的memory layout，就不会有实际的kernel
- MatMul Layer
  - 如果使用FP16，而且K不是8的倍数，TRT可能会引入reformat kernel，可以通过手动padding到8的倍数来规避
  - 如果input是4维，可以尝试通过reshape转化为3维或者2维，可能有更好的性能

https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html#optimize-layer

性能优化技巧

Optimizing for Tensor Cores

- 使用Tensor Core才能发挥GPU的最大算力，TF32，FP16，INT8，FP8
- TRT的MatrixMultiply, FullyConnected, Convolution, and Deconvolution等计算密集的算子会使用Tensor Core
- Tensor Core对data alignment有要求：

Table 4. Types of Tensor Cores

Tensor Core Operation Type	Suggested Tensor Dimension Alignment in Elements
TF32	4
FP16	8 for dense math, 16 for sparse math
INT8	32

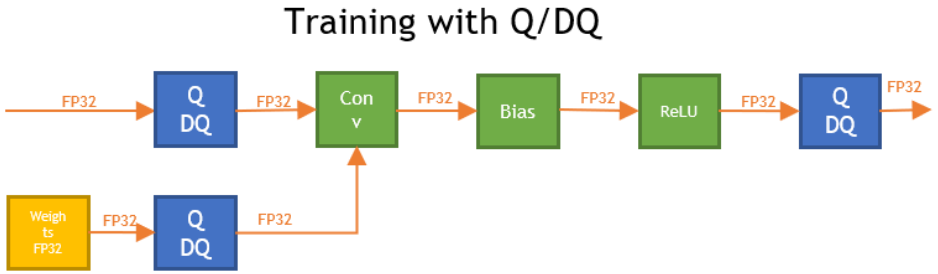
- 当tensor不满足alignment条件时，TRT会隐式地做padding，比如插入reformat layer
- 可以设置nsys --gpu-metrics-device all来查看Tensor Core使用情况

https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html#optimize-tensor-cores

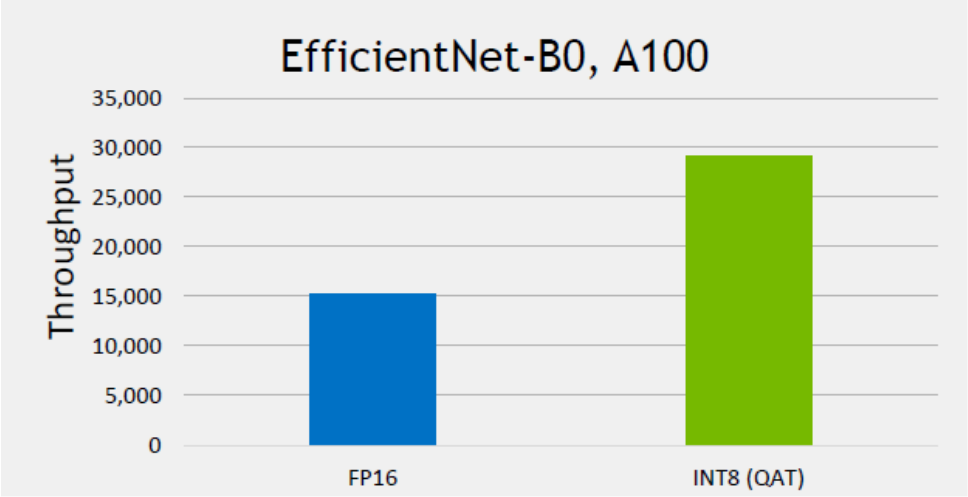
性能优化技巧

Low Precision

- 使用低精度可以降低memory开销，其中FP16比较成熟，INT8需要PTQ或者QAT来保证精度，TRT接下来的版本会支持FP8
- INT8 QAT
  - 量化感知训练（QAT）相比于训练后量化（PTQ）有更高的精度
  - 训练阶段，TensorRT提供了基于PyTorch的INT8量化工具包，帮助生成QAT模型，并维持原有精度
  - 模型转换阶段， TensorRT ONNX parser新增了对ONNX QuantizeLinear和DequantizeLinear算子的支持
  - 推理阶段，TensorRT 8.0新增了Quantize和Dequantize层以及基于Q/DQ的图优化



	FP32 & FP16	INT8 PTQ	INT8 QAT
Top-1 acc	77.4	33.9	76.8

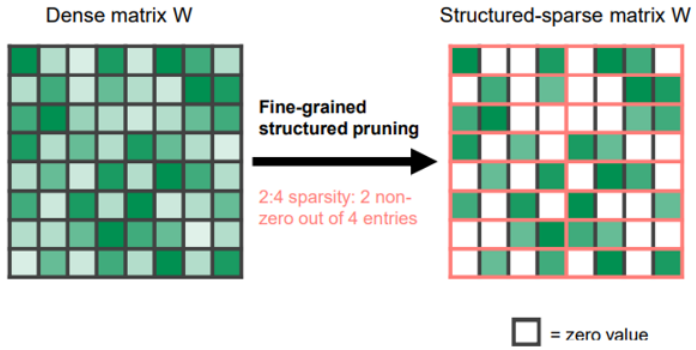


https://www.nvidia.com/en-us/on-demand/session/gtcspring21-s31876/

性能优化技巧

Sparsity

- TRT8.0支持了结构化稀疏，它是安培架构引入的新特性
- 利用结构化稀疏(structured sparsity)加速推理
  - 通过2:4细粒度结构化稀疏减少一半参数，并利用Sparse Tensor Core进行加速，在Bert模型上能实现1.3倍的加速
  - 训练阶段，通过稀疏化工具ASP (Automatic SParsity)修剪参数，然后微调模型以维持原有精度
  - 推理阶段，通过设置build_config的SPARSE_WEIGHTS flag来挑选sparse kernel
- 可以通过trtexec --sparsity=enable/force来快速测试加速效果




```
# Import ASP (Automatic SParsity)
from apex.contrib.sparsity import ASP
ASP.prune_trained_model(model, optimizer)
```

Training Phase

Enable **Sparsity** by setting the `kSPARSE_WEIGHTS` flag in `IBuilderConfig`

Inference Phase

https://www.nvidia.com/en-us/on-demand/session/gtcspring21-s31876/
https://www.nvidia.com/en-us/on-demand/session/gtcspring21-s31552/
https://developer.nvidia.com/blog/accelerating-inference-with-sparsity-using-ampere-and-tensorrt/

性能优化技巧

Misc

- Optimizing Plugins
  - The performance of plugins depends on the CUDA code performing the plugin operation.
  - Support as many formats as possible in the plugin. This removes the need for internal reformat operations during the execution of the network
- Optimizing Builder Performance
  - Timing cache
  - Tactic Selection Heuristic
- Builder Optimization Level
  - Set the optimization level in builder config to adjust how long TRT should spend searching for tactics with potentially better performance.

https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html#performance

TensorRT性能优化

大纲

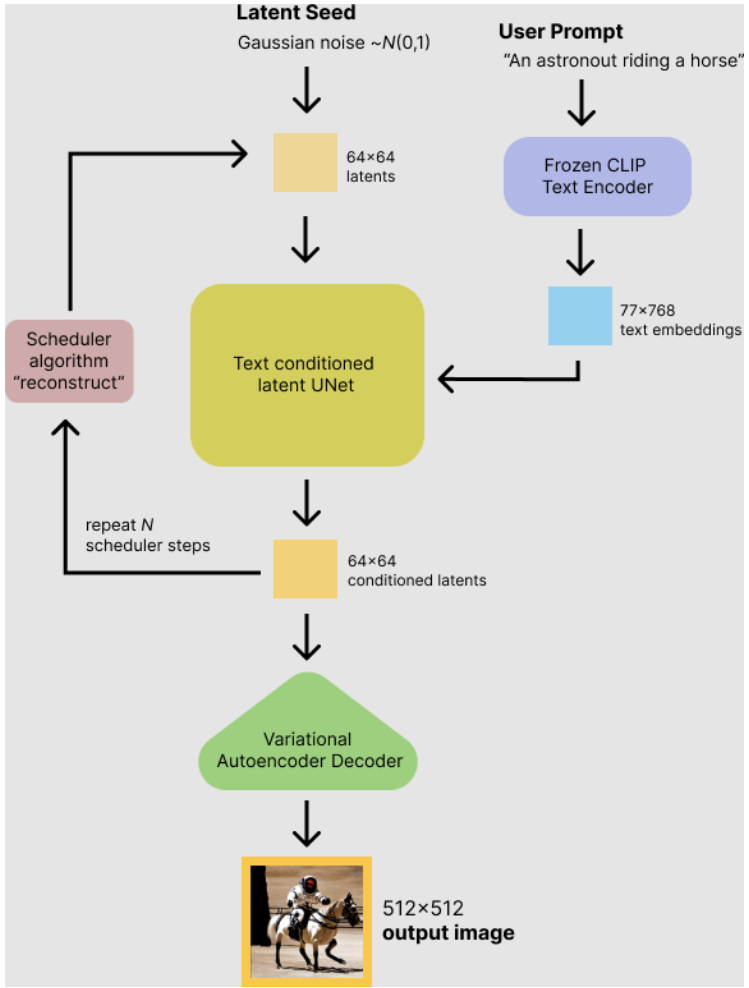
- 概述
- 性能分析工具
- 性能优化技巧
- **性能优化实例**

性能优化实例

TRT v8.5 Demo Stable Diffusion

- Stable Diffusion，爆火的文本生成图像的扩散模型，pipeline由三部分组成：
- text-encoder (CLIP)
- U-Net （运行50次，占绝大部分时间）
- Autoencoder (VAE)

Prompt: an astronaut riding a horse



性能优化实例

TRT v8.5 Demo Stable Diffusion

- 优化流程
  1. Run PyTorch->ONNX->TRT to get baseline
  2. Profile and find perf bottleneck
  3. Optimize with ONNX-graphsurgeon and TRT plugin
  4. Repeat step2 and step3 until satisfied

- Stable Diffusion
  - Main perf bottleneck: unfused MHA，占30% end2end time

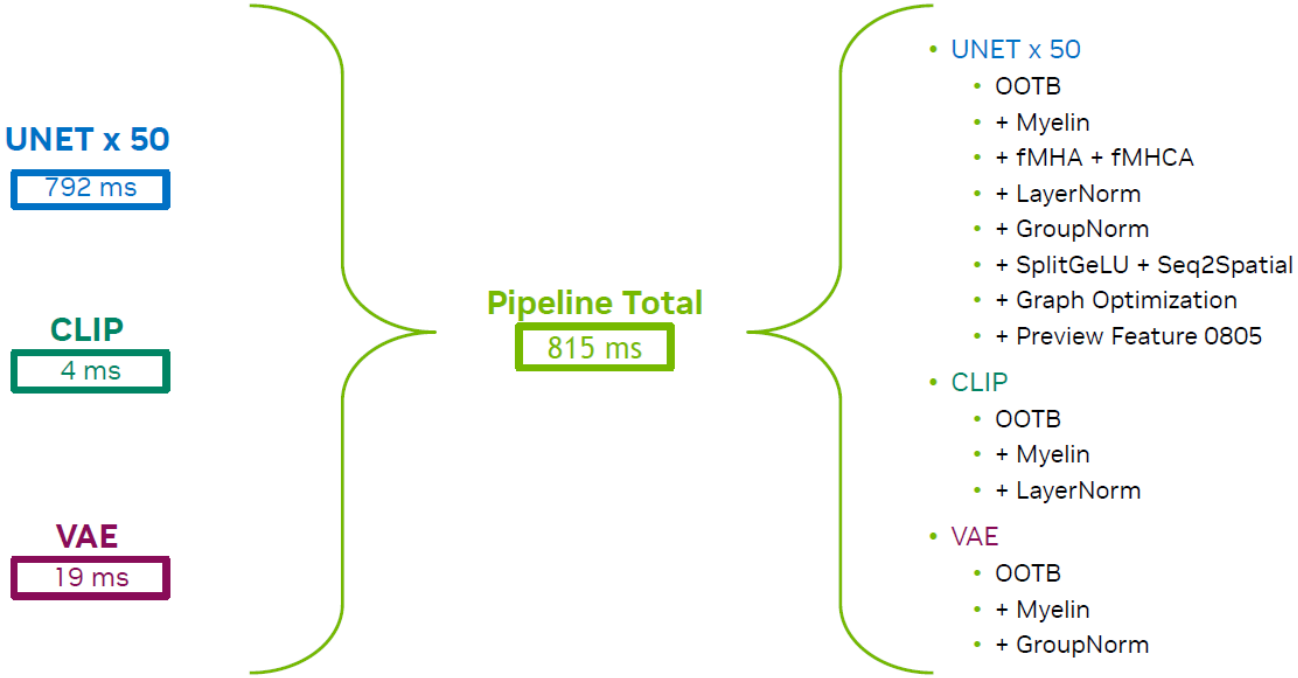
- TRT OSS 8.5, plugin solution
  - https://github.com/NVIDIA/TensorRT/tree/release/8.5/demo/Diffusion

U-Net	Time
MHA	30%
GroupNorm	12%
FC-GatedGelu	6%
MHCA	4%
…	

性能优化实例

TRT v8.5 Demo Stable Diffusion

- Graph optimization code and associated plugin
  - https://github.com/NVIDIA/TensorRT/blob/release/8.5/demo/Diffusion/models.py
  - https://github.com/NVIDIA/TensorRT/tree/release/8.5/plugin



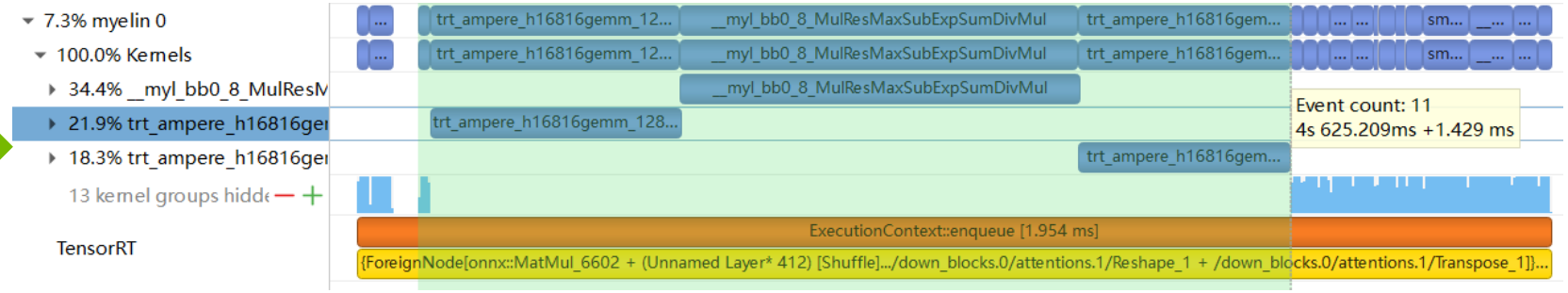
- TensorRT 8.5.2.2, bs=1, wh=512x512

性能优化实例

TRT v8.5 Demo Stable Diffusion

- **MHA plugin**
  - https://github.com/NVIDIA/TensorRT/blob/release/8.5/demo/Diffusion/models.py#L637
  - https://github.com/NVIDIA/TensorRT/tree/release/8.5/plugin/multiHeadFlashAttentionPlugin
- 图优化关键点在于找寻MatMul (Q*K) -> SoftMax -> MatMul (Score*V) subgraph pattern
- Plugin kernel is based on Flash-attention, can save lots of memory during engine-building and inference phase, also significantly speedup Unet model
- Only supports GPUs with compute capability 7.5, 8.0, 8.6 and 8.9

Unfused MHA: 1.429ms



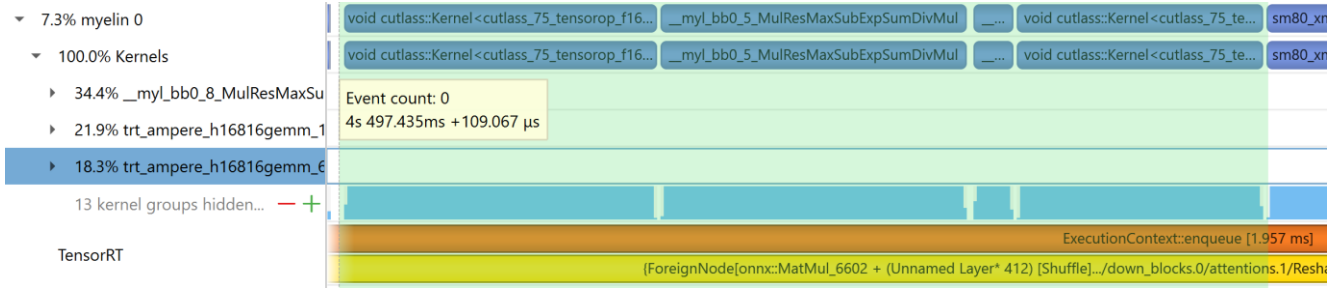
Fused MHA: 0.556ms

性能优化实例

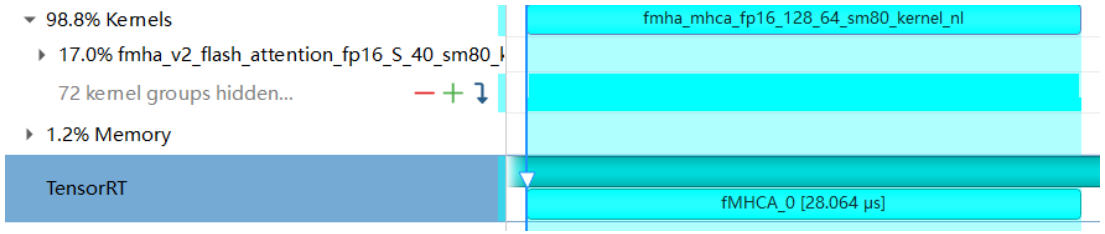
TRT v8.5 Demo Stable Diffusion

- **MHCA plugin**
  - https://github.com/NVIDIA/TensorRT/blob/release/8.5/demo/Diffusion/models.py#L631
  - https://github.com/NVIDIA/TensorRT/tree/release/8.5/plugin/multiHeadCrossAttentionPlugin
- 图优化关键点在于找寻MatMul (Q*K) -> SoftMax -> MatMul (Score*V) subgraph pattern，与MHA区别主要在于QKV输入是否相同
- MHCA时间占比和显存占用都不高，Plugin kernel is not based on Flash-attention, can speedup Unet model.
- Only supports GPUs with compute capability 7.5, 8.0, 8.6 and 8.9

Unfused MHCA: 109us



Fused MHCA: 28us

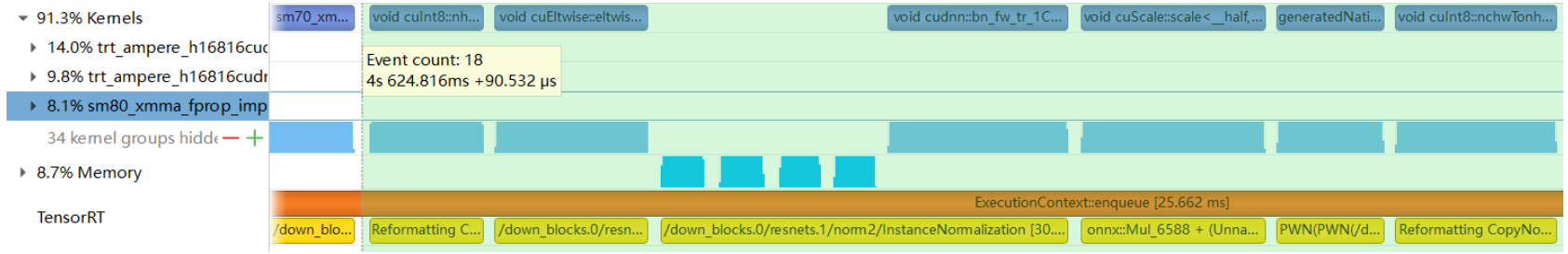


性能优化实例

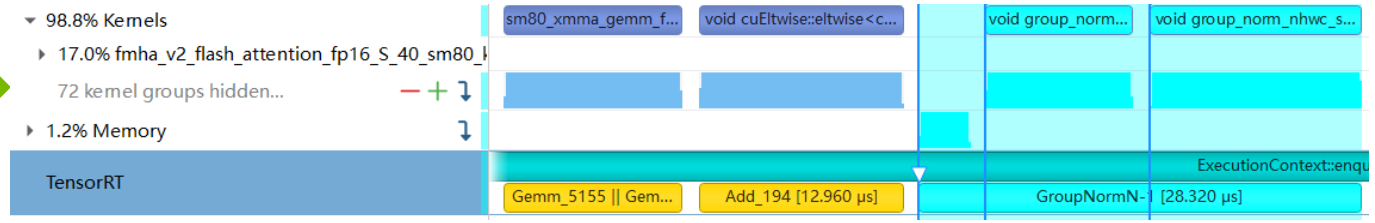
TRT v8.5 Demo Stable Diffusion

- **GroupNorm plugin**
  - https://github.com/NVIDIA/TensorRT/blob/release/8.5/demo/Diffusion/models.py#L206
  - https://github.com/NVIDIA/TensorRT/tree/release/8.5/plugin/groupNormPlugin
- 图优化关键点在于先将InstanceNormalization算子拆分成ReduceMean等小算子，再向下融合为GroupNorm自定义算子，因为计算图存在Conv-GroupNorm-Swish pattern，所以plugin增加了对Swish的融合
- GroupNorm的存在方式一般为Conv-GroupNorm[optional swish]-Conv，其中Conv kernel为NHWC格式，而未经优化前TRT对GroupNorm采用的是NCHW格式，所以前后都会存在Transpose算子
- 为解决该问题，GroupNorm kernel采用NHWC格式，避免了大量Transpose开销

Unfused GroupNorm: 90us



Fused GroupNorm : 28us



性能优化实例

TRT v8.5 Demo Stable Diffusion

- **LayerNorm plugin**
  - https://github.com/NVIDIA/TensorRT/blob/release/8.5/demo/Diffusion/models.py#L255
  - https://github.com/NVIDIA/TensorRT/tree/release/8.5/plugin/layerNormPlugin
- 跟我们在2022 hackathon的问题/解决方案是一致的

- **SplitGeLU plugin**
  - https://github.com/NVIDIA/TensorRT/blob/release/8.5/demo/Diffusion/models.py#L313
  - https://github.com/NVIDIA/TensorRT/tree/release/8.5/plugin/splitGeLUPlugin
- 存在于FFN block，TRT会拆分为多个kernel，使用plugin进行彻底的融合
- 更激进的做法是通过cutlass将MatMul+SplitGeLU融合为一个kernel，但未被添加进 v8.5 plugin方案中

- **SeqLen2Spatial plugin**
  - https://github.com/NVIDIA/TensorRT/blob/release/8.5/demo/Diffusion/models.py#L336
  - https://github.com/NVIDIA/TensorRT/tree/release/8.5/plugin/seqLen2SpatialPlugin
- 为了去除冗余的Transpose节点

性能优化实例

TRT v8.6 Demo Stable Diffusion

- 我们不想要复杂而且依赖性高的手动图优化和plugin
- 我们希望TensorRT能自动识别并完成极致的性能优化

- TRT OSS 8.6, OOTB solution
  - https://github.com/NVIDIA/TensorRT/tree/release/8.6/demo/Diffusion
- 充分借鉴和整合8.5 plugin solution的优化思路和代码
- **无需手动图优化，无需手写plugin，直接导入TRT即可达到与plugin solution相近的性能！**
- 增加img2img和inpaint pipeline

- 另外，OSS 8.5专门为demo stable diffusion准备的上述plugin，在8.6已经移除，不会再更新

性能优化实例

TRT v8.6 Demo Stable Diffusion

- Scheduler optimization
  - Scheduler本身的工程优化，从diffuser到demo diffusion的实现
  - Scheduler算法的优化，比如从LMSD到DPMSolver，减少迭代次数

- Interactions between PyTorch and TRT
  - context.set_tensor_address(name, tensor.data_ptr())

- 千万注意要保证提供给TRT input的Torch tensor是连续的
  - Torch tensor经过一些Transpose类算子后在内存上可能并不连续
  - 可以通过tensor.is_contiguous()来检查是否在GPU内存上连续
  - 通过tensor.contiguous()来保证连续性，reshape(-1)也可以

- 千万注意要保证提供给TRT input的Torch tensor的数据类型是相符的
  - TRT不支持INT64/Float64，会自动转成INT32/Float32，需要将Torch tensor转成相应datatype
  - timestep_float = timestep.float()

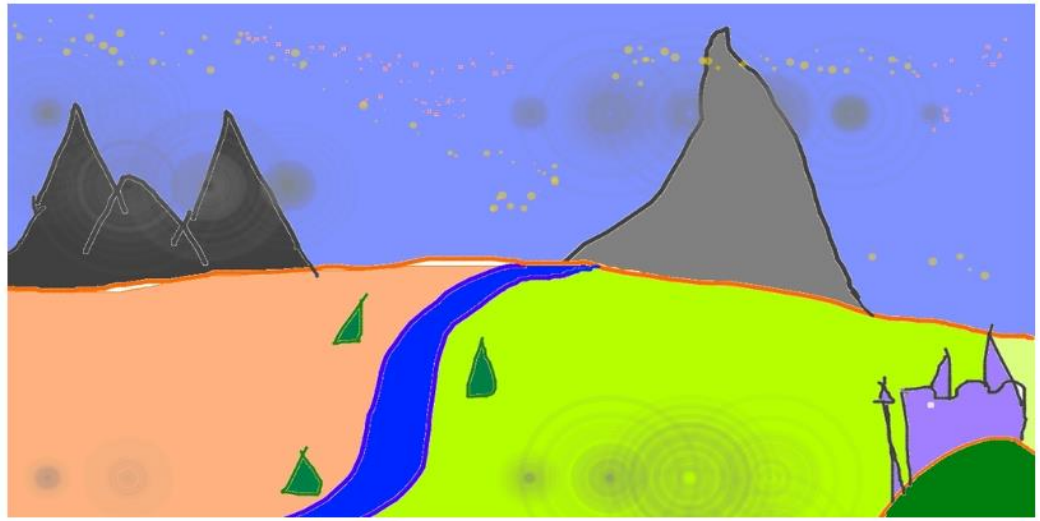
https://github.com/NVIDIA/TensorRT/blob/release/8.6/demo/Diffusion/utilities.py#L377
https://github.com/huggingface/diffusers/blob/main/src/diffusers/schedulers/scheduling_ddim.py

性能优化实例

TRT v8.6 Demo Stable Diffusion

- Img2img pipeline
  - https://github.com/NVIDIA/TensorRT/blob/release/8.6/demo/Diffusion/demo_img2img.py
  - https://huggingface.co/docs/diffusers/api/pipelines/stable_diffusion/img2img
- 类似txt2img pipeline，不同的是增加了一个image作为输入，通过VAE Encoder进行处理，输出init_latents

Prompt: A fantasy landscape, trending on artstation
Input image:



Output image:

https://replicate.com/stability-ai/stable-diffusion-img2img

TensorRT性能优化

总结

- TensorRT性能优化的核心是充分发挥GPU算力
- TensorRT性能优化的目标是尽可能地把所有非GEMM kernel融合起来
- TensorRT性能优化的手段主要是计算图优化和plugin
- TensorRT性能优化的工具主要是Nsight system，trtexec等

- TensorRT在快速持续更新中，以此契合更新的GPU/CUDA，应对更多的模型需求
- 本次讲座只是介绍性能优化的整体概念，内容可能有所遗漏或过时，还请大家取其精华，也欢迎各位选手分享好的性能问题和优化思路，谢谢！