Accelerate RecSys and Increase GPU Utilization with Multiple CUDA Streams

Randy Wang, DevTech Engineer, NVIDIA

Robin Zhang, DevTech Engineer, NVIDIA

GTC 2024

Agenda

- RecSys Status & Problems
- CUDA Stream & Frameworks
- Accelerate RecSys Inference
- Accelerate RecSys Training

Agenda

- RecSys Status & Problems
- CUDA Stream & Frameworks
- Accelerate RecSys Inference
- Accelerate RecSys Training

RecSys Overview

RecSys development status on NVIDIA GPUs

- NVIDIA GPUs serve most of the top RecSys customers worldwide.
- Customers are willing to migrate their RecSys workloads to GPU to save costs. Firstly, only the dense part, then the embedding lookup, then the data preprocessing, and gradually the whole pipeline is on GPU.
- The most common feedback we receive during the migrations is that **GPU utilization** is too low. WHY?

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 550.54.14              Driver Version: 550.54.14      CUDA Version: 12.4     |
|-------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA A10                     On  |   00000000:65:00.0 Off |                    0 |
| 95%   43C    P0             125W /  150W |   22134MiB /  23028MiB |     33%      Default |
|                                         |                        |                  N/A |
+-------------------------------+----------------------+----------------------+
|   0  NVIDIA A10                     On  |   00000000:65:00.0 Off |                    0 |
| 90%   45C    P0             123W /  150W |   21346MiB /  23028MiB |     29%      Default |
|                                         |                        |                  N/A |
+-------------------------------+----------------------+----------------------+
|   0  NVIDIA A10                     On  |   00000000:65:00.0 Off |                    0 |
| 95%   42C    P0             115W /  150W |   21586MiB /  23028MiB |     28%      Default |
|                                         |                        |                  N/A |
+-------------------------------+----------------------+----------------------+
|   0  NVIDIA A10                     On  |   00000000:65:00.0 Off |                    0 |
| 90%   43C    P0             117W /  150W |   20041MiB /  23028MiB |     32%      Default |
|                                         |                        |                  N/A |
+-------------------------------+----------------------+----------------------+
```

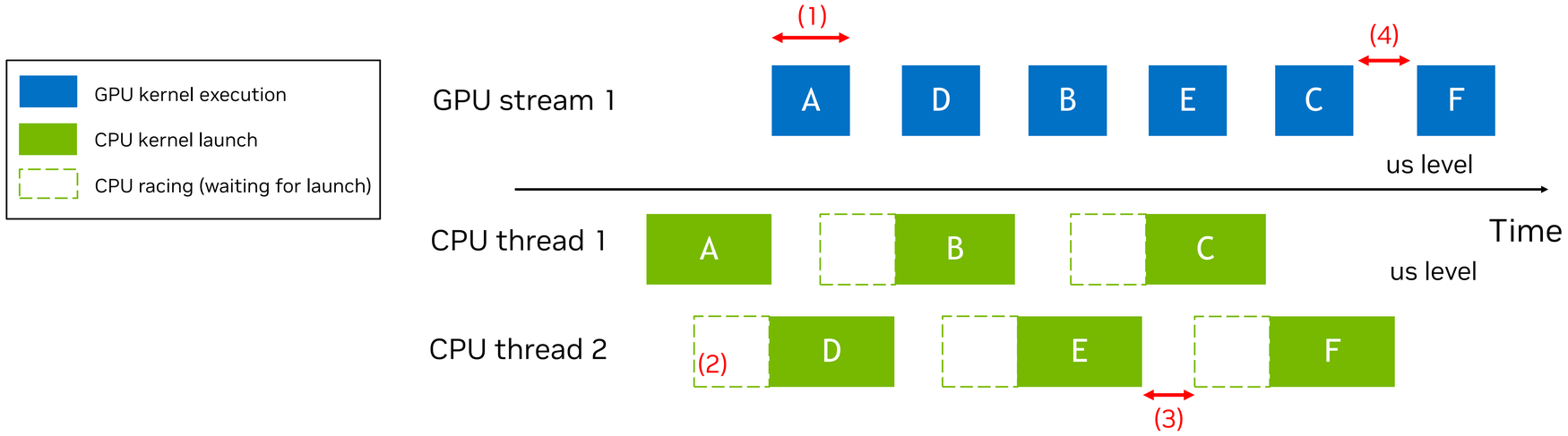
One typical RecSys training scenario. The average GPU utilization is around 30%.

RecSys on GPU

GPU cannot be well used without careful optimization

- The widely used CVR/CTR models are of small kernels. This causes significant **kernel launch overhead** and reduce the **GPU utilization** and **SM utilization**. The system running time is the **CPU working time**.
- It's worse when there are multiple threads launching kernels, as the launching will be serialized on the CPU side.

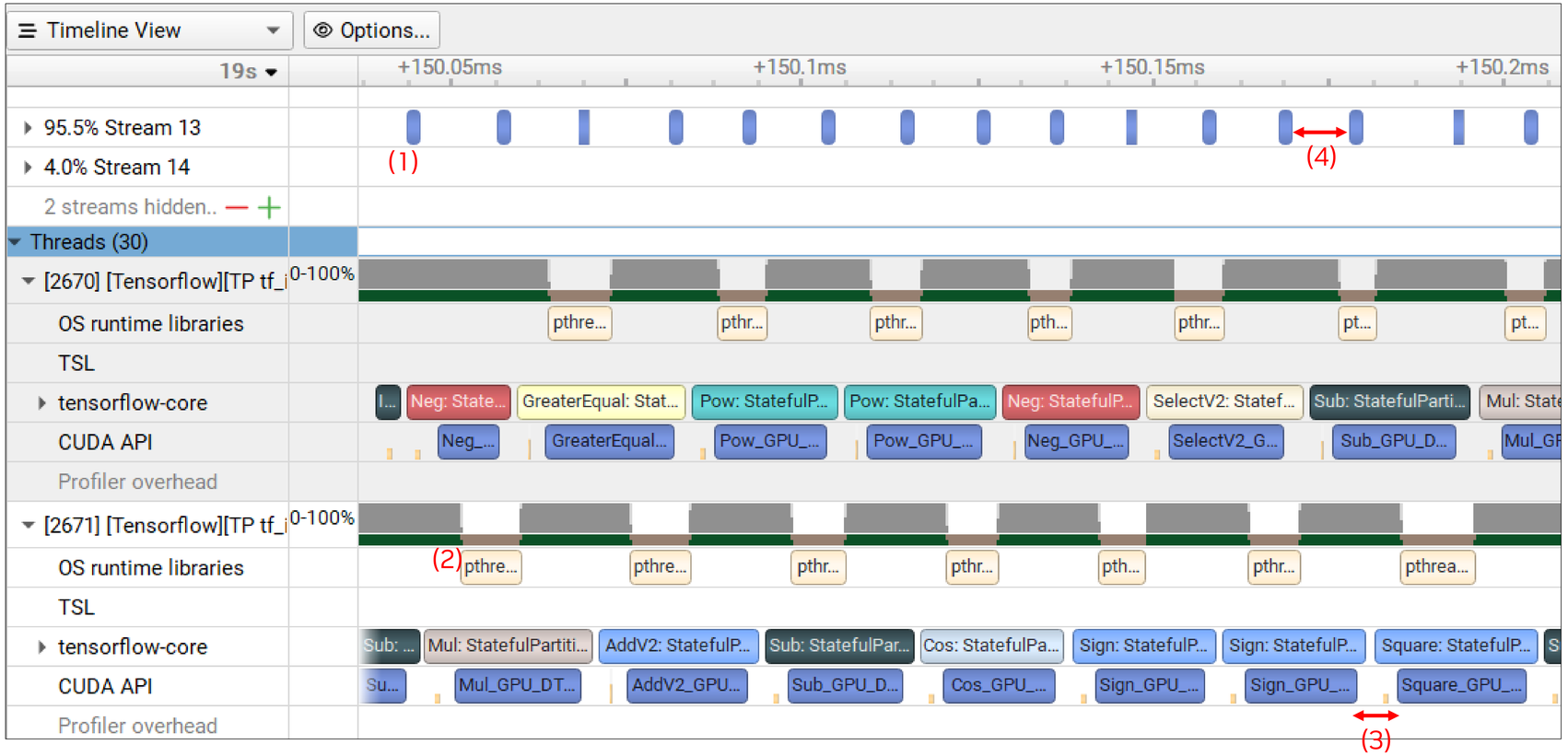
(1) Small kernel size leads to short execution time, sometimes <10us.
(2) Multiple threads launching to one stream, causing CPU racing.
(3) Framework (usually TensorFlow) scheduling latency & locks.
(4) large GPU idle time, causing low GPU utilization.



What really happened in most CTR/CVR models

A Concrete Case

CTR inference model from our customer



Two CPU threads running two inference queries. There is only one CUDA stream.

(1) Small kernel size leads to short execution time, sometimes <10us.

(2) Multiple threads launching to one stream, causing CPU racing.

(3) Framework (usually TensorFlow) scheduling latency & locks.

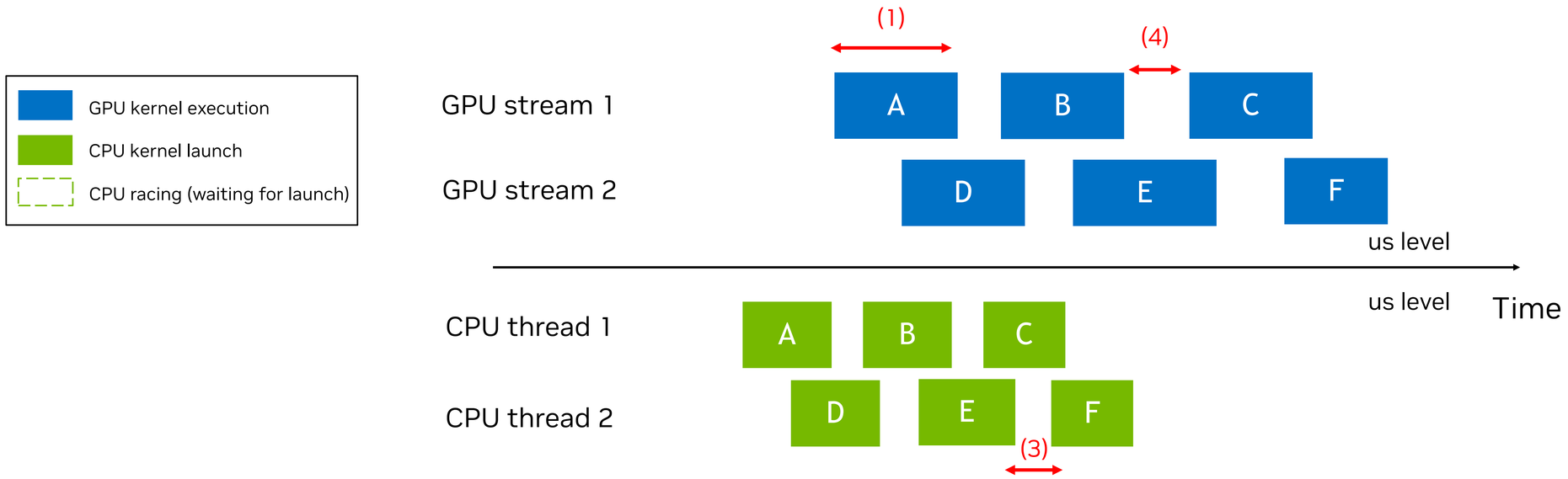
(4) large GPU idle time, causing low GPU utilization.

Multiple Stream

Why multiple stream benefits

If have multiple stream:

(1) Kernels sizes are still small, but can overlap with each other.
(2) CPU racing on the same stream no longer exist.
(3) Framework scheduling latency is the same. But locks contentions may be eliminated.
(4) Idle on one stream is filled by another stream, GPU util is higher.



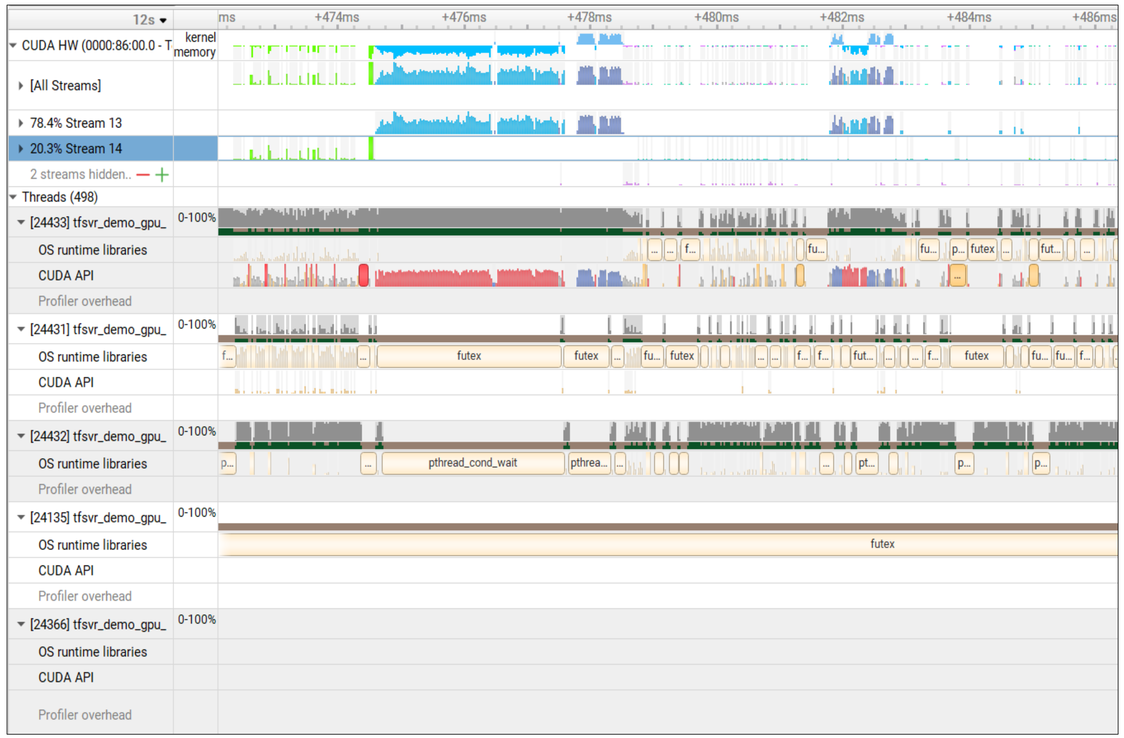
Under multiple stream

Diagnostic

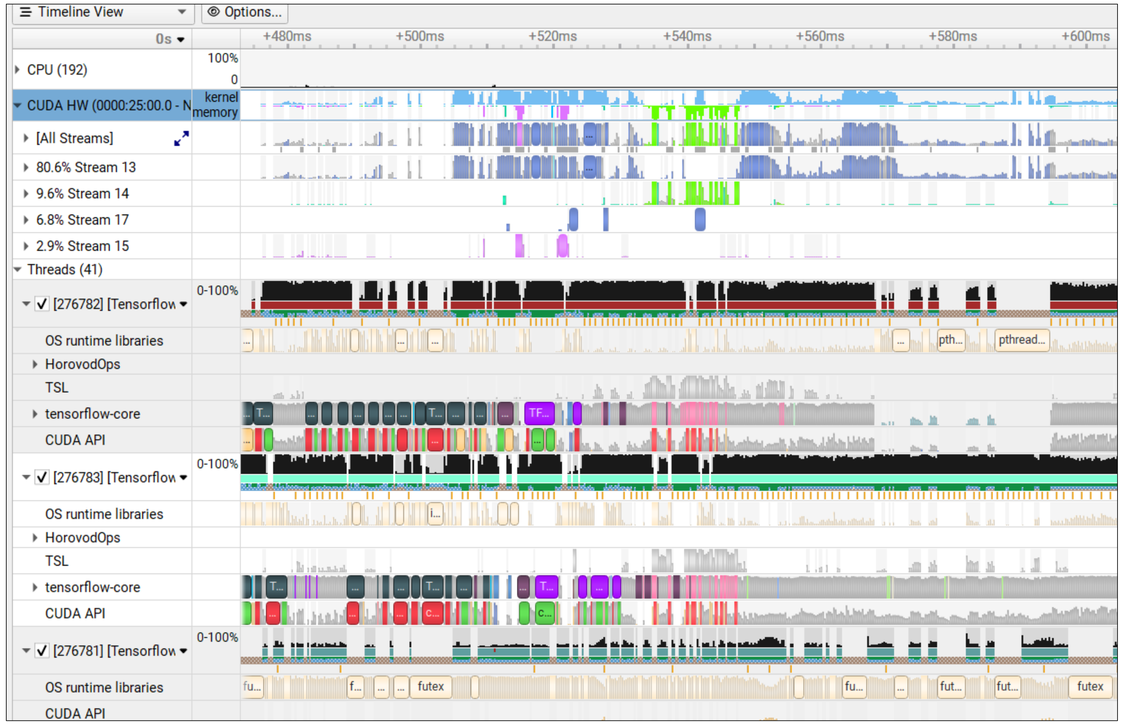
In what situation, you need multi-stream in your workload?

- Parallelizable workload : paralleled query in inference, or multiple branch structure (like multi-task learning) in training.
- Short kernel : Same level with kernel launch. It is the reason of low GPU utilization. Ideally GPU utilization =100%. At least >90%.
- Small kernel size : kernels with only a few blocks, which means SMs is not been fully utilized.

Observe them via Nsight System! You'll see many blanks on the GPU side and busy launching or racing on the CPU side.



RecSys Inference



RecSys Training

Agenda

- RecSys Status & Problems
- CUDA Stream & Frameworks
- Accelerate RecSys Inference
- Accelerate RecSys Training

CUDA Stream in TensorRT

- TensorRT supports both within-inference multi-streaming[1] and cross-inference multi-streaming[2] to enhance GPU utilization.
  - Within-inference multi-streaming:

```
// Allow 1+3 streams at maximum in builder config.
IBuilderConfig* config = builder->createBuilderConfig();
config->setMaxAuxStreams(3);

// Build engine and create execution context.
ICudaEngine engine = builder->buildEngineWithConfig(network, &config);
IExecutionContext *context = engine->createExecutionContext();

// Start inference using a CUDA stream. TensorRT will automatically run some layers on the
auxiliary streams.
context->enqueueV3(stream);
```

  - Cross-inference multi-streaming:

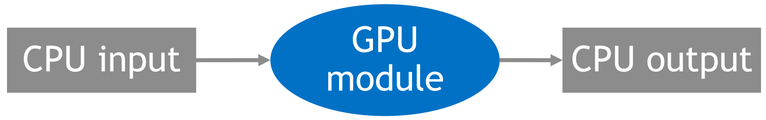
```
// Build engine and create multiple execution contexts.
ICudaEngine engine = builder->buildEngineWithConfig(network, &config);
std::vector<IExecutionContext*> contexts;
for (size_t i = 0; i < 3; ++i) {
    contexts.push_back(engine->createExecutionContext());
}

// Execute inference in different CUDA streams in parallel.
for (size_t i = 0; i < 3; ++i) {
    contexts[i]->enqueueV3(streams[i]);
}
```

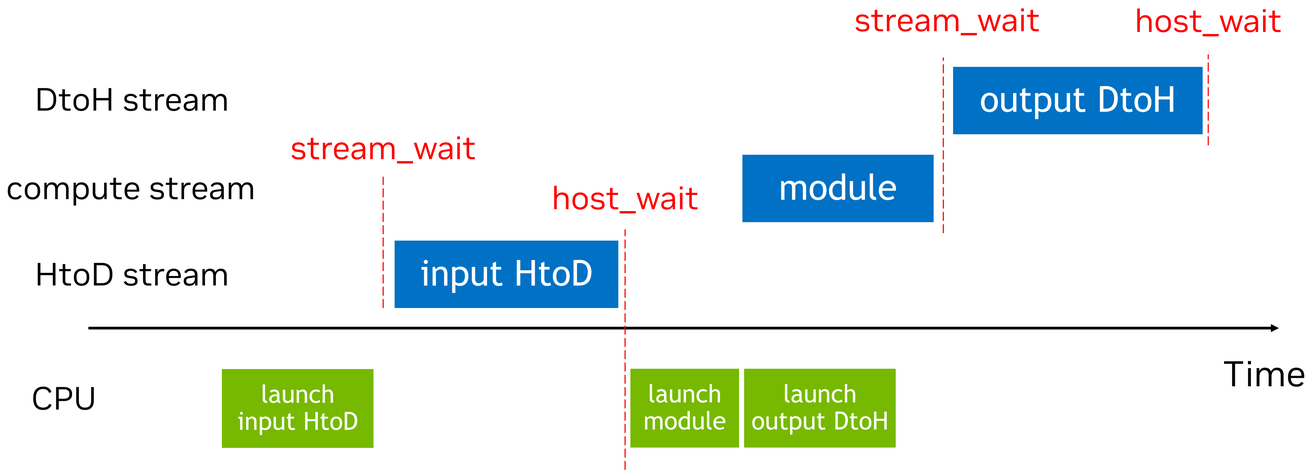
- More practices please see [**S61715: Optimizing Online Recommender Services with NVIDIA Hopper**].

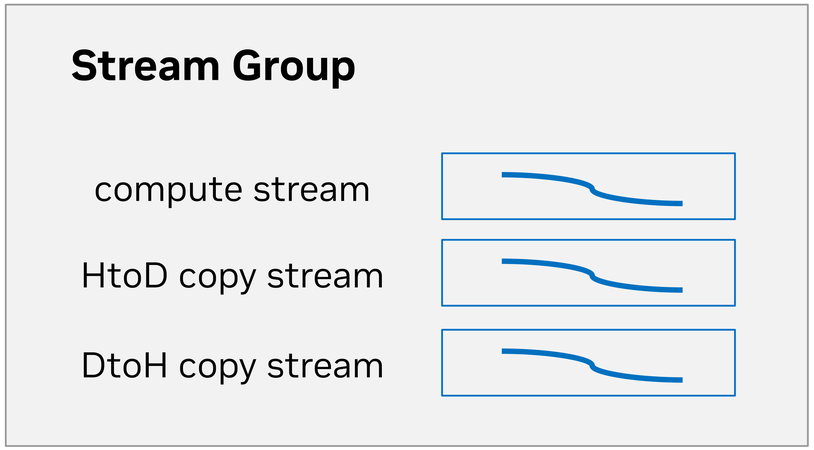
1. https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html#within-inference-multi-streaming
2. https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html#streaming

CUDA Stream in TensorFlow



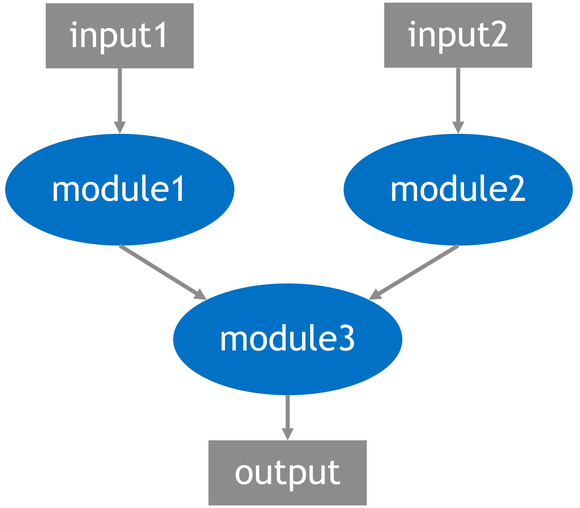
- TF is the most widely used framework for RecSys.
- TF uses “Stream Group”: a compute stream, a HtoD copy stream, and a DtoH copy stream. Assigning HtoD/DtoH/computation to different streams allows overlaps between copy and computation.
- The design of TF introduces two overheads:
  - **Kernel launch overhead** when multiple CPU threads launching kernels, because there is always only one compute stream.
  - **Stream synchronization overhead**, because every HtoD/DtoH needs a **stream_wait** before it, and a **host_wait** after it.

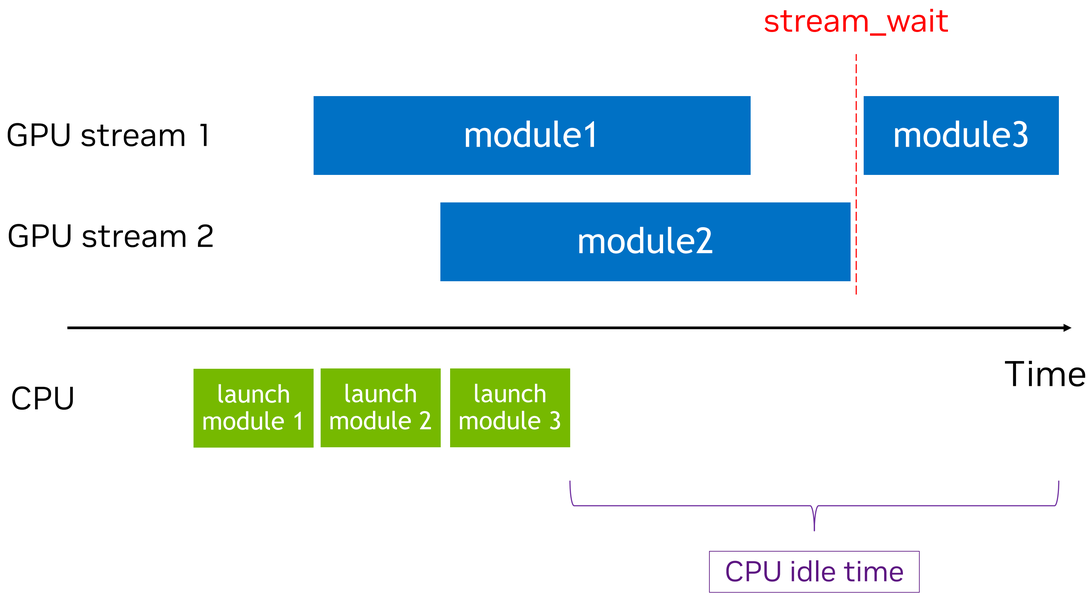




CUDA Stream in PyTorch

- PyTorch exposes stream related CUDA APIs.
- Users can create streams and assign CUDA kernels to streams easily.
- If not specified, the kernels will be executed in the default stream.
- *PyTorch's design is well suited for doing multi-stream acceleration during **training**.*





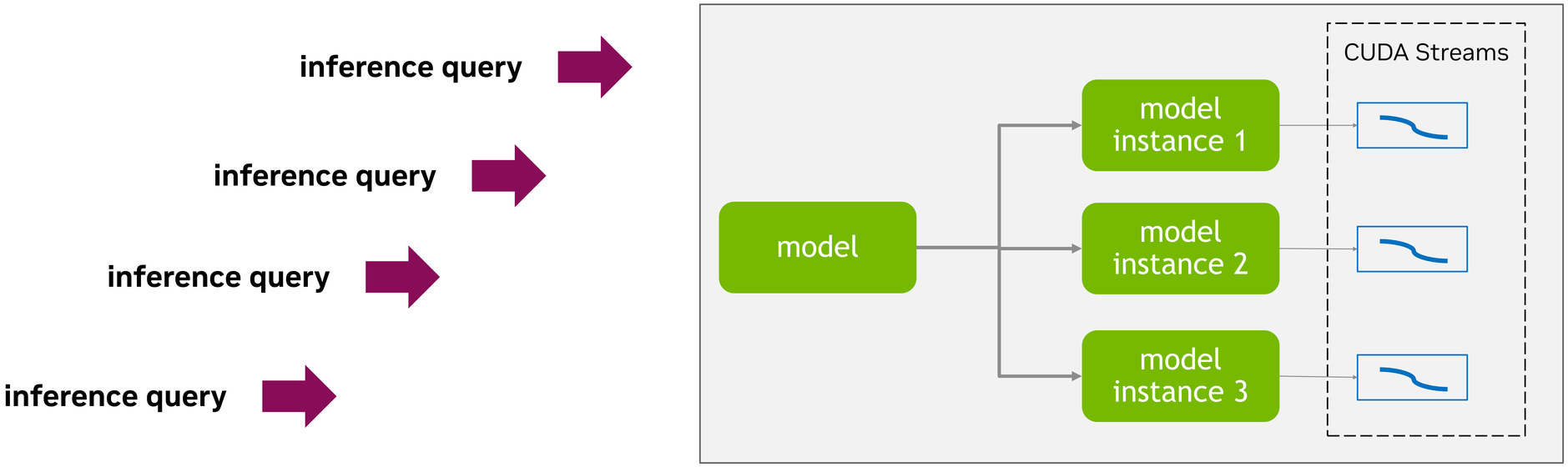

```
// Create two streams.
stream1 = torch.cuda.Stream()
stream2 = torch.cuda.Stream()

// module1 and module2 can overlap in two streams.
with torch.cuda.stream(stream1):
    act1 = module1(input1)
with torch.cuda.stream(stream2):
    act2 = module2(input2)

// Ensure module2 is finished before executing module3.
stream1.wait_stream(stream2)
with torch.cuda.stream(stream1):
    output = module3(act1, act2)
```

CUDA Stream in HugeCTR HPS

- HugeCTR HPS[1] initializes multiple "model instances" for inference service. Each instance has its own stream.
- An inference query is served by one model instance.
- *HPS's design is well suited for doing multi-stream acceleration during **inference**.*



1. https://nvidia-merlin.github.io/HugeCTR/main/hierarchical_parameter_server/index.html

Agenda

- RecSys Status & Problems
- CUDA Stream & Frameworks
- **Accelerate RecSys Inference**
- Accelerate RecSys Training

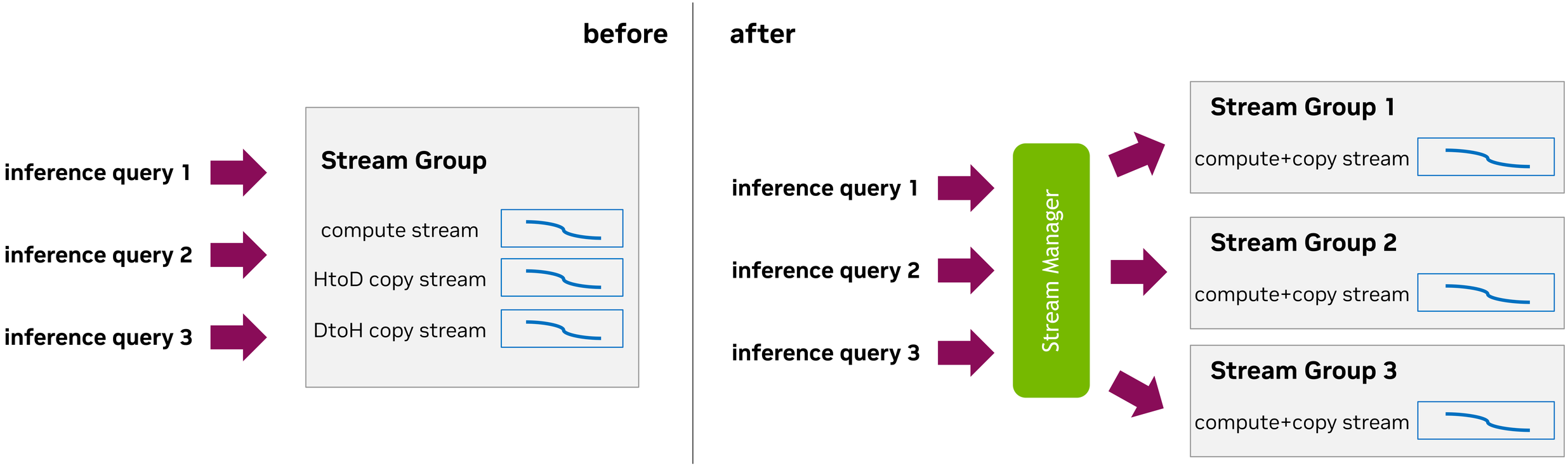
Multiple Streams in Inference

Three key contributions: multiple stream groups + stream merging + stream manager

- Most companies serve RecSys inference with TF-Serving, so our implementation[1] is also based on TF.
- **Multiple Stream Groups**: the inference queries can be served by different stream groups independently.
- **Stream Merging**: use one stream in one stream group to do all the compute and copy of one query.
- **Stream Manager**: Every coming inference query picks the lowest loaded stream to execute.

kernel launch overhead

stream synchronization overhead



1. https://docs.google.com/document/d/1yL3lWk_iFKqLTyekkuaiKXZ78I0lPmD5kM1fghHRs4Y/edit?usp=sharing

Ablation Study

The contribution of multiple stream groups and stream merging

- WDL model inference service.
- A10 GPU. The number of concurrent inference queries (4 or 8) is the same as the number of stream groups.

	4 queries	8 queries
Official TF	66.8	67.2
+ multiple stream groups	230.8 (3.45x)	427.7 (6.34x)
+ multiple stream groups + stream merging	244.8 (3.66x)	463.0 (**6.89x**)

- Throughput (the higher, the better)

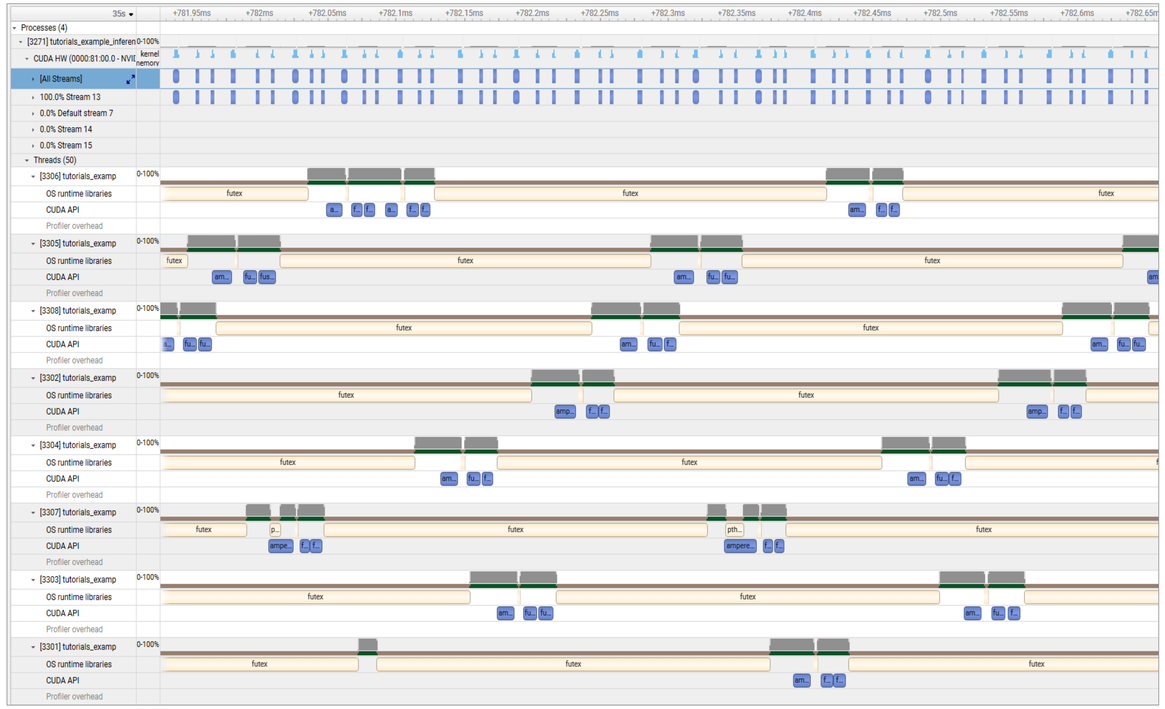
	4 queries	8 queries
Official TF	70.2	175.1
+ multiple stream groups	22.6 (0.32x)	23.2 (0.13x)
+ multiple stream groups + stream merging	20.2 (0.29x)	20.9 (**0.12x**)

- TP99 Latency, in ms (the lower, the better)

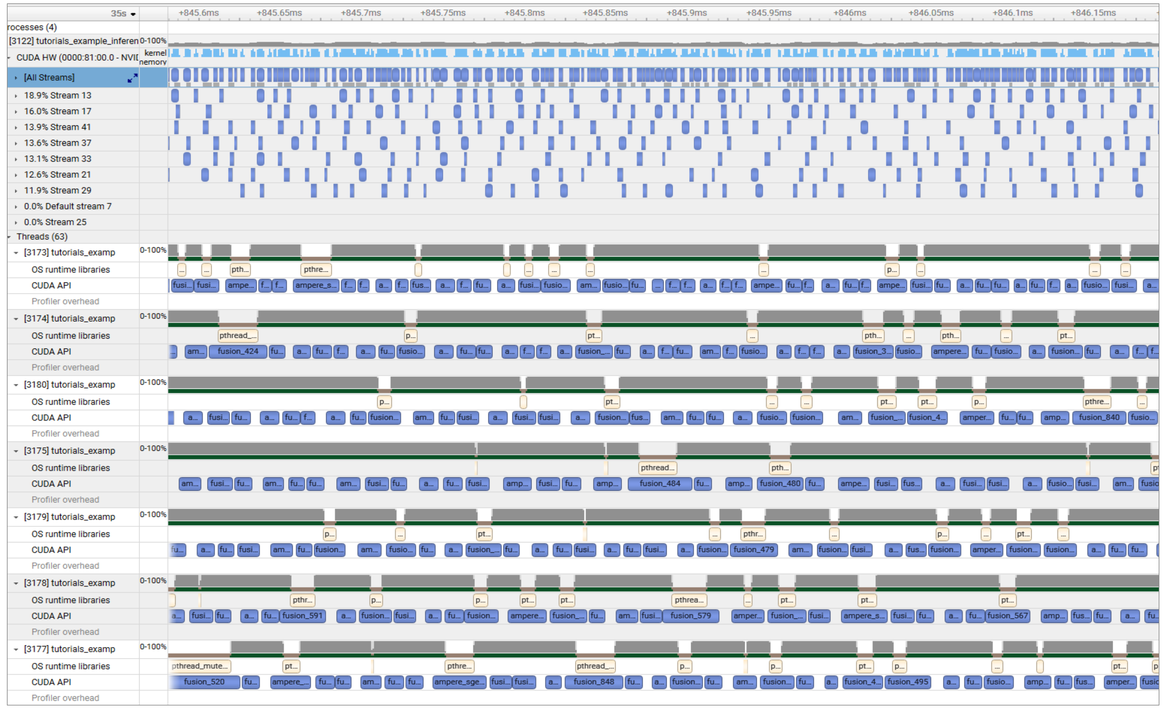
Nsight Systems Timeline

Multi-stream help increase GPU utilization

- A10 GPU. 8 inference queries concurrently.
- Optimization target for inference: higher throughput under tolerable TP99 latency (usually <50ms).



- w/o multi-stream
- GPU Utilization: 29%
- Throughput: 88.6 queries/s
- TP99 Latency: 153.5 ms

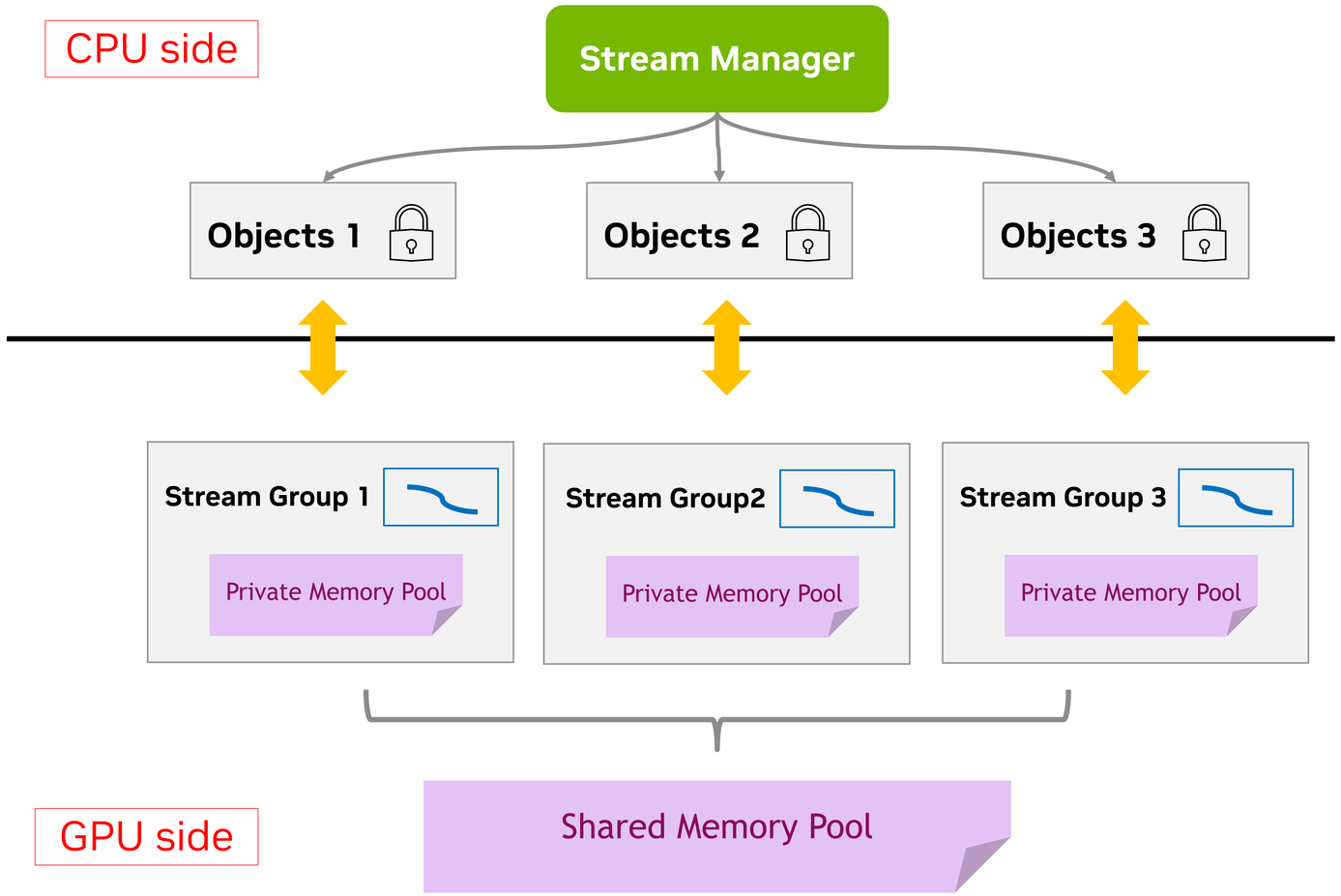


- w/ multi-stream (8 stream groups)
- GPU Utilization: 84%
- Throughput: 287.6 queries/s (3.25x)
- TP99 Latency: 39.7 ms (0.26x)

Optimize Resource Management for Multi-Stream

Achieve lock-free on the CPU side and weights sharing on the GPU side

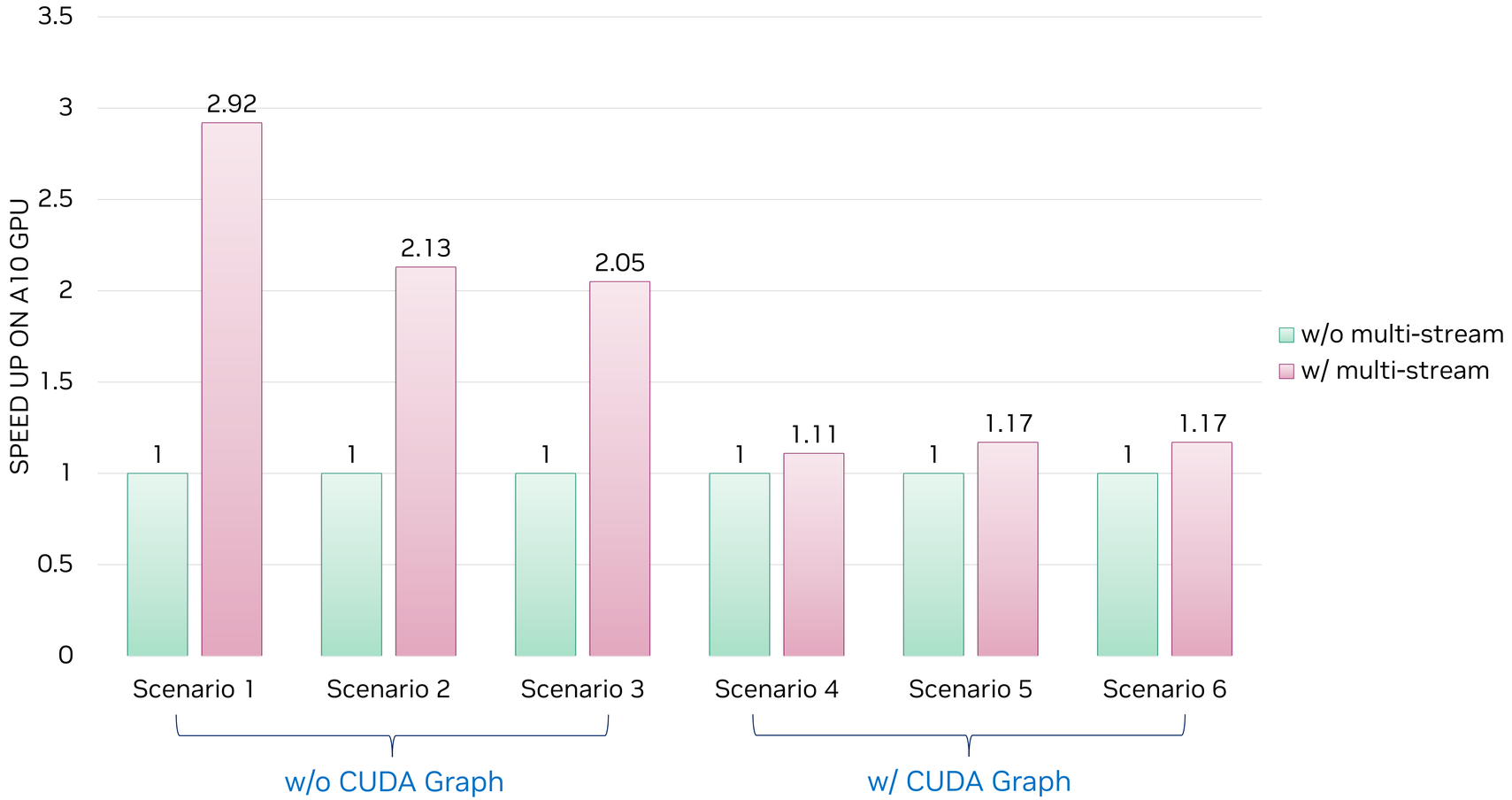
- On the CPU side, decouple data dependencies between streams.
  - Every stream has its own copy of lock-required objects to avoid cross-stream racing.
  - Customer-1 got 1.79x throughput boost by decoupling the host memory allocator objects between stream groups.
- On the GPU side, share whatever can be shared between streams.
  - Private pool: for activations, cublas workspace, etc.
  - Shared pool: for model weights, XLA bin, TRT engine, etc.
  - Customer-2 got a significant memory and warmup time decrease, leading to more streams used and a 30%~50% end-to-end speedup.



Case Study

Combine with CUDA Graph for end2end speedup

- Some of our customers use CUDA Graph to accelerate searching and recommendation inference. It's difficult to generalize CUDA Graph to all models due to its strict constraint on shape and memory. Multi-stream can benefit these models (#1-#3).
- For models already accelerated with CUDA Graph, multi-stream is also helpful (#4-#6).



Performance

Use multi-stream in customers' online system

- Selected top customers that are using our multi-stream for RecSys online inference service:

	scenario	multi-stream landmark
Taobao RTP[1]	searching & recommendation	up to 3x throughput
DeepRec[2]	hundreds of models via cloud service	0.72x P99 Latency for vivo. 60% cost saved.
Alibaba Ads	ads recommendation	1.3x~1.5x throughput
WeChat Video Channels	video recommendation	40% cost saved in terms of ROI
IQIYI	video recommendation	1.4x throughput

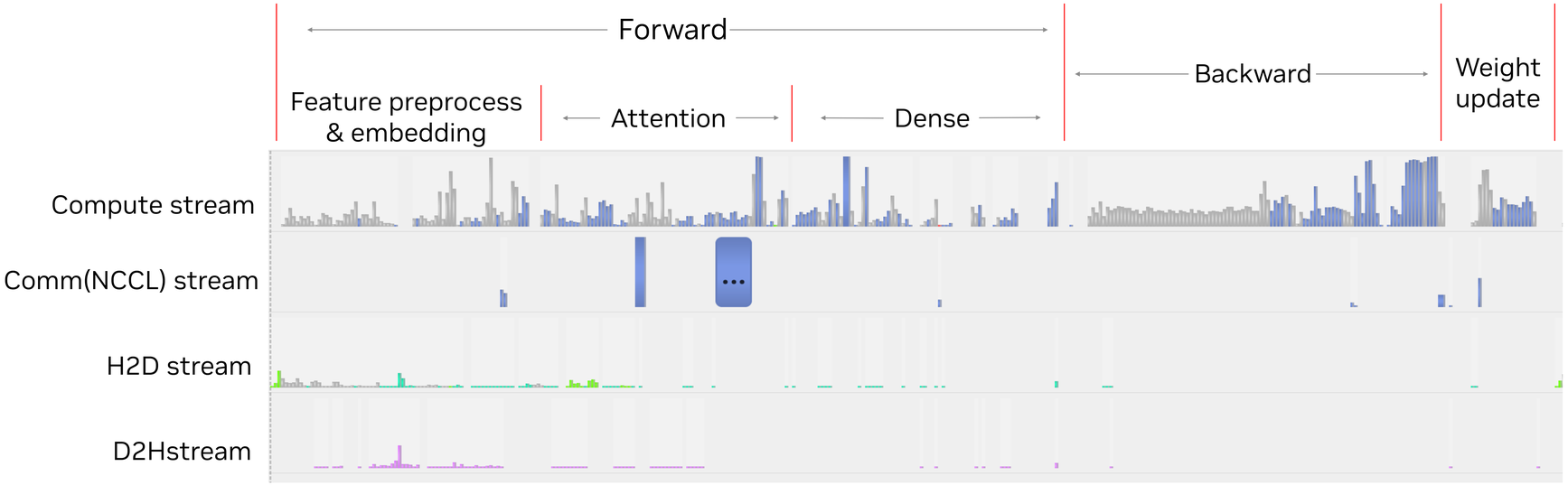
1. https://mp.weixin.qq.com/s/kqlCVXD2rF2EMkh7-qCa1Q
2. https://mp.weixin.qq.com/s/LQ6xjHqv0YYDRXtQTB5qPw

Agenda

- RecSys Status & Problems
- CUDA Stream & Frameworks
- Accelerate RecSys Inference
- Accelerate RecSys Training

Multi-Stream in Training

A typical RecSys training workload



- GPU util is low, kernel launch bound.
- Tens of thousands of small ops (as well as kernels).
- Can't apply inference multi-stream because of only 1 training instance.

- TF creates relatively fragmented small ops and kernels for some high level APIs.

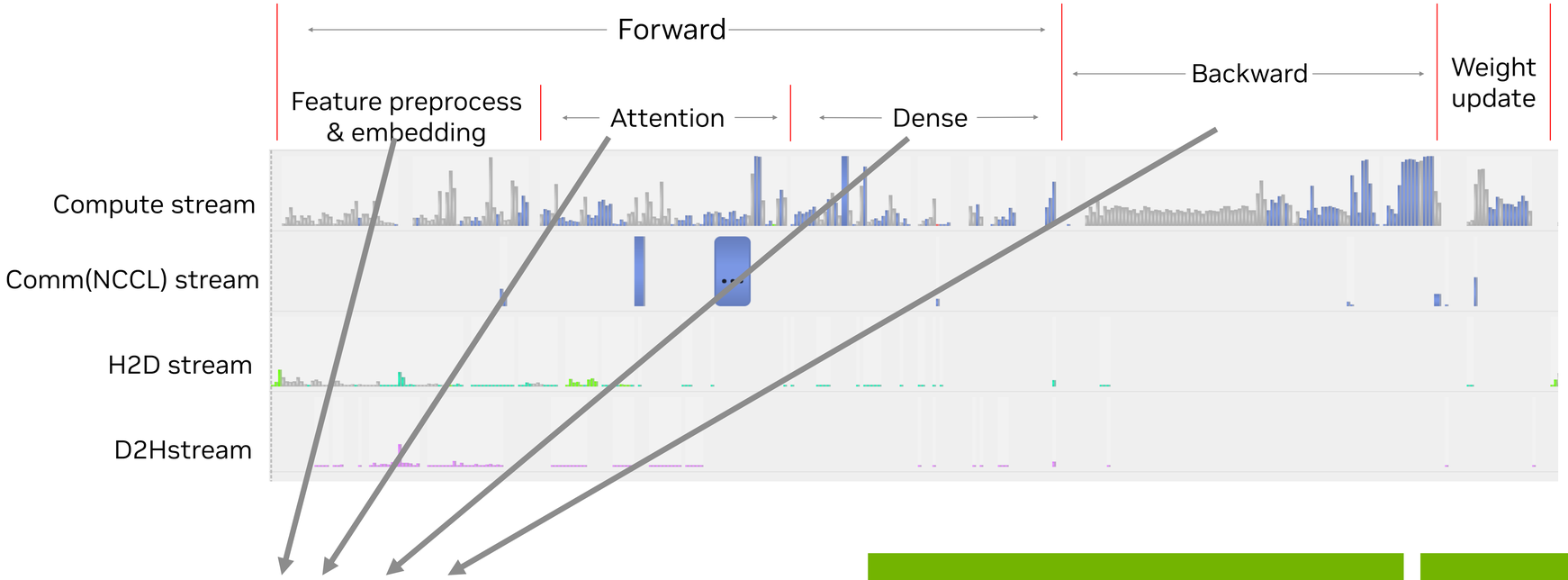
```
def unsorted_segment_mean(data, segment_ids, num_segments, name=None):
  with ops.name_scope(name, "UnsortedSegmentMean"):
    data = ops.convert_to_tensor(data)
    segment_ids = ops.convert_to_tensor(segment_ids)
    N = _unsorted_segment_N(data, segment_ids, num_segments)
    summed = gen_math_ops.unsorted_segment_sum(data, segment_ids, num_segments)
    return summed / N

def _unsorted_segment_N(data, segment_ids, num_segments):
  num_segments = ops.convert_to_tensor(num_segments)
  # bincount doesn't support negative indices so we use unsorted_segment_sum
  segment_ids_shape = array_ops.shape_internal(segment_ids)
  ones_tensor = array_ops.ones(segment_ids_shape, dtype=data.dtype)
  n = gen_math_ops.unsorted_segment_sum(ones_tensor, segment_ids, num_segments)
  # add dimensions for all non-reduced axes
  broadcastable_shape = array_ops.concat(
      [num_segments[array_ops.newaxis],
       array_ops.ones([array_ops.rank(data)
                       - array_ops.rank(segment_ids)],
                      dtype=num_segments.dtype)],
      axis=0)
  n = array_ops.reshape(n, broadcastable_shape)
  return gen_math_ops.maximum(n, 1)
```

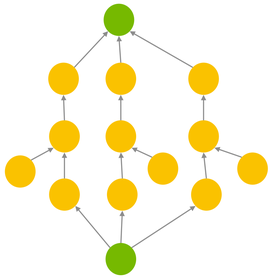
- Multi-branches structure of model. *branches* x *ops_each_branch* is high.

Multi-Stream in Training

Parallel branches structure



Parallel branches structure



Constructed by for-loop in Python

```
for feature in all_features:
  # preprocess feature
  ...
  keys = custom_hash_op(feature)
  keys = tf.unique(feature)
  ...
  # embedding
  emb = sparse_lookup(
    keys, emb_var)
  tf.unsorted_segment_mean(
    emb, indices)
  ...
```

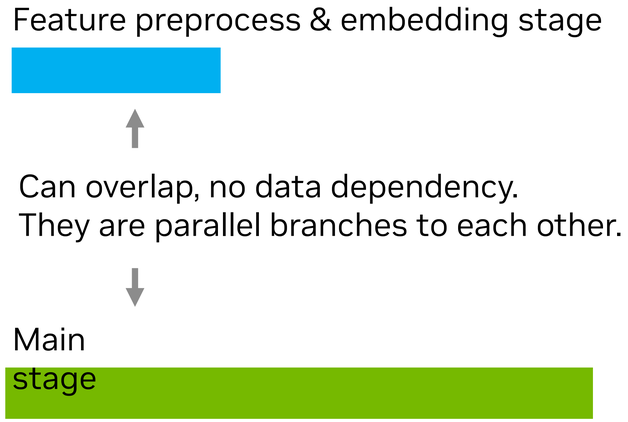
```
for attn in config["attn_pairs"]:
  # build attn branch
  score = attn["feat_1"] * attn["feat_2"]
  score = softmax(score)
  ...
...
for target in config["targets"]:
   # build target branch
   act = tf.dense(target["input"])
   ...
   loss = loss_fn(output, target["label"])
   ...
```

Multi-Stream in Training

Parallel branches structure

Customer's advanced development, decouple model to two stages, eliminate data dependency.

- Feature preprocess & embedding stage: Consume input data and prepare embeddings for next train step.
- Main stage: Consume embeddings which was ready in last train step's feature preprocess & embedding stage and do following forward & backward.



Multi-Stream Training Runtime

Stream assignment

Method 1: Manual stream assignment

- 1.1. Assign streams with Python API:

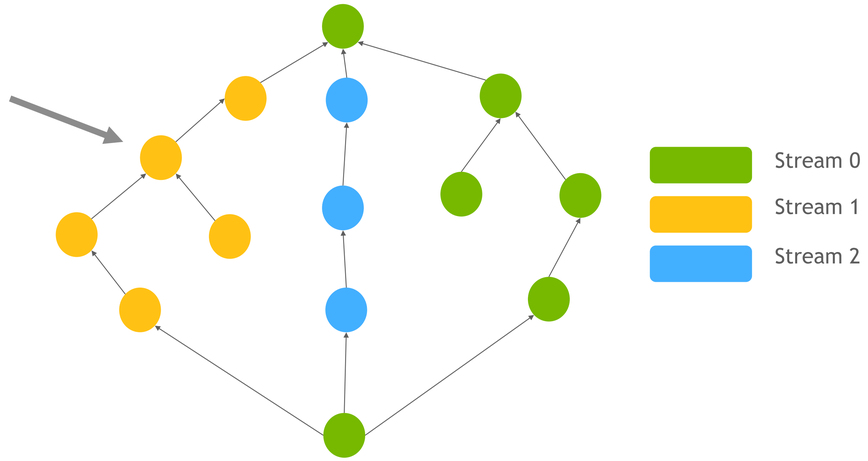
```
with tf.cuda.stream_scope(stream=tf.cuda.Stream(),
                          include_grad=True):
  # create branch subgraph in yellow
  ...
with tf.cuda.stream_scope(stream=tf.cuda.Stream(),
                          include_grad=True):
  # create branch subgraph in blue
  ...
with tf.cuda.stream_scope(include_grad=True):
  # create branch subgraph in default stream, green
  ...
```

or

```
for branch in branches:
  with tf.cuda.stream_scope(stream=tf.cuda.Stream(),
                            include_grad=True):
    # create branch subgraph
    ...
```

- Backward branch subgraph will use the same stream as the forward branch subgraph, if specify "***include_grad = True***".

```
node {
  name: "add_95"
  op: "AddV2"
  input: "global_step/read"
  input: "add_95/y"
  attr {
    key: "_stream_id"
    value {
      i: 1
    }
  }
```



- 1.2. Assign streams in config file in op level:

```
Attention/user_idx_to_item/* : 1

Attention/user_gender_to_item/* : 2

...
```

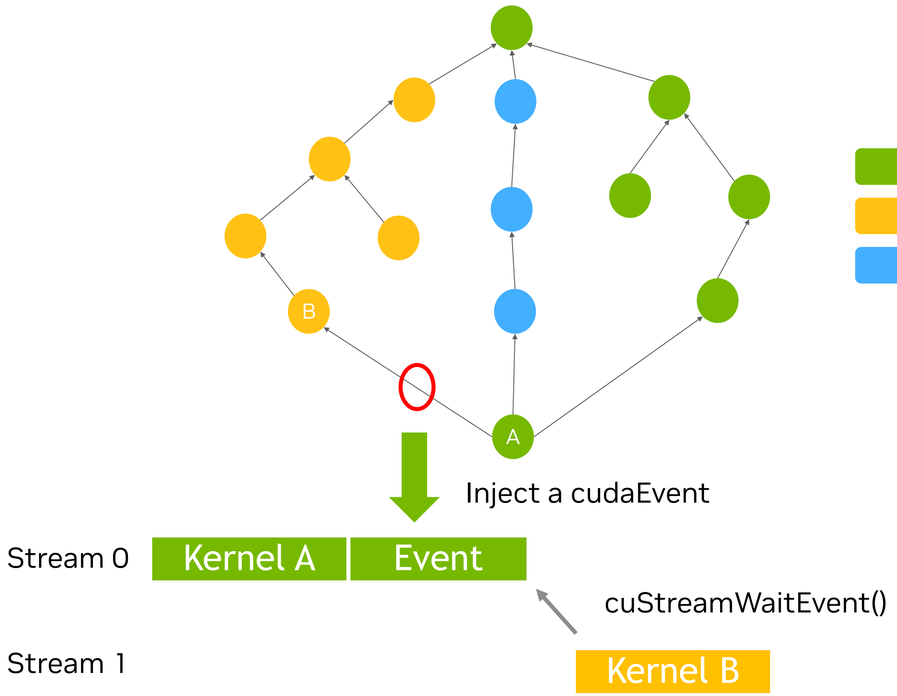
- Each line maps a node name (regex supported) to a stream.

Method 2: Auto stream assignment (WIP)

- Auto assign stream groups to nodes, through graph partitioning algorithm and dynamic runtime stats.
- Under developing.

Multi-Stream Training Runtime

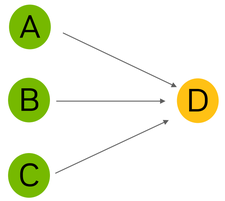
Mechanism & Challenges



TF Multi-stream training runtime

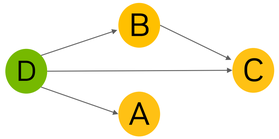
- Manage stream sync & memory allocation automatically.
- Option to use stream merge.
- Other options, like reduce stream sync, constrain memory usage, etc.

- Reducing stream sync through graph structure analysis



Statical analysis:

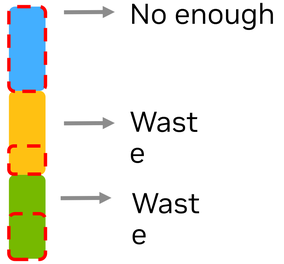
D only need to sync with stream 0 once, despite 3 input nodes from stream 0.

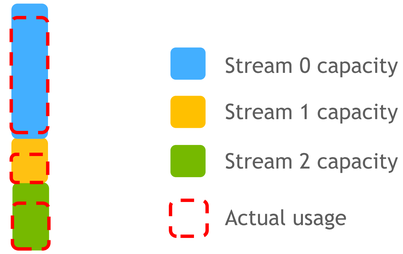


Dynamic analysis:

When launching B or C, if D has already synced with A on stream 0, no need to fire a new stream sync for B or C.

- Reduce memory consumption and fit to different workload size on different streams on the fly.





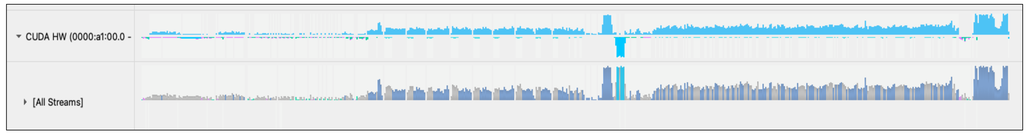
Memory capacity is equally divided originally. Hard to fit different workload size among streams.

Share capacity counter and allow each stream's space to grow dynamically to fit different workload size.

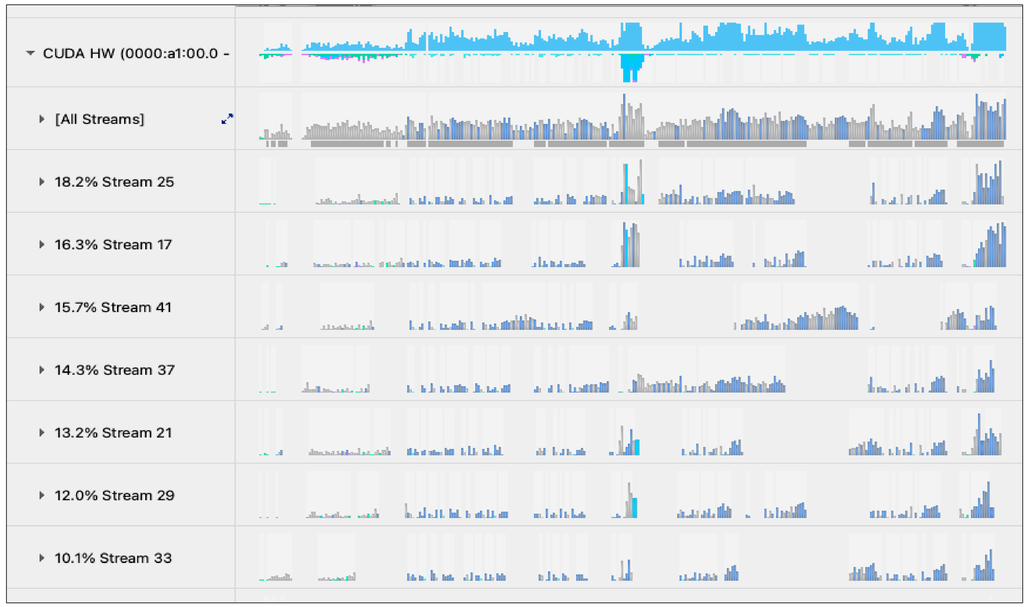
Performance

In-the-lab performance

- PLE model training, from model zoo. Driver r525.
  - Before:

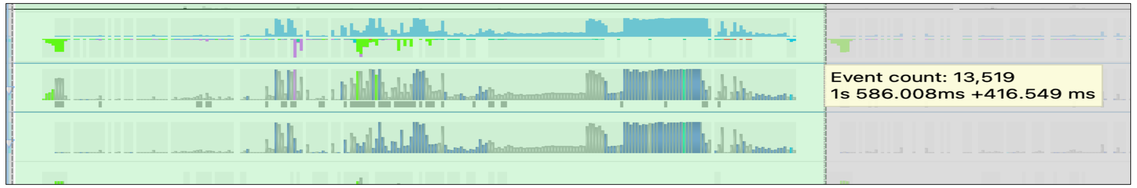


  - After multi-stream (stream = 7)

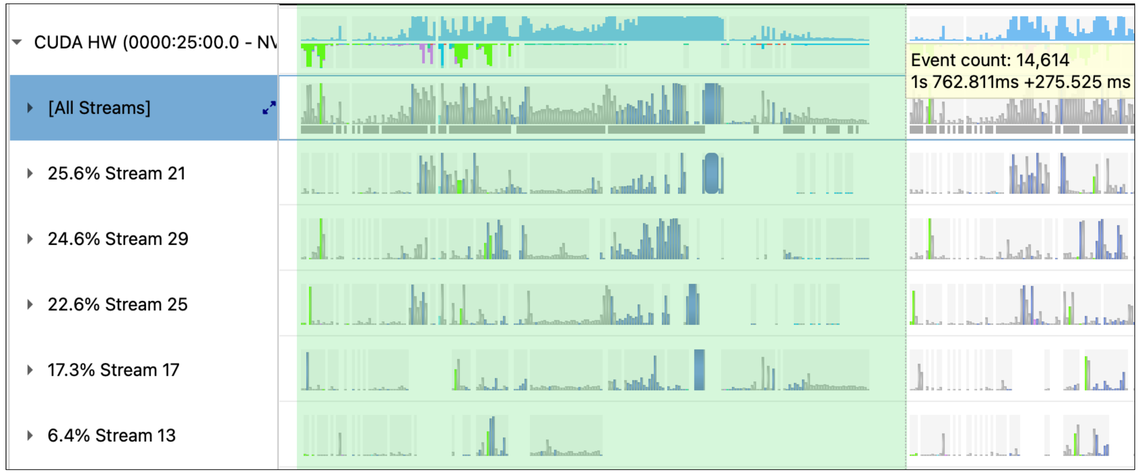


- Performance boost by ~30%.

- A reranking model resembles customer's real model. Driver r525.
  - Before:



  - After multi-stream (stream = 5):



- Performance boost by ~33%.

Performance

Industrial customer

- Applied on customer’s RecSys models training (hundreds of millions DAU).
- Only using multi-stream for embedding pipeline overlapping. 2 streams.

scenario	Configs	Speed up
Reranking	1 node x 8 * GPUs	15%
Reranking	4 nodes x 8 * GPUs	8%

- Cooperating with the development team to spread to more scenario.

Conclusion and Future Work

Multi-stream inference and training

- Inference code is under the review process and merging to community TF: PR[1], Doc[2].
  - https://mp.weixin.qq.com/s/kqlCVXD2rF2EMkh7-qCa1Q
  - https://mp.weixin.qq.com/s/LQ6xjHqv0YYDRXtQTB5qPw
  - …More success stories on the way!

- Multi-stream training is still in beta testing
  - Auto stream assignment.
  - Optimization for memory consumption and stream sync overhead.

- Welcome to try the menu! contact robinz@nvidia.com/ruotongw@nvidia.com

- Special thanks to our key customers for working closely with us to improve the solution!

1. https://github.com/tensorflow/tensorflow/pull/61185
2. https://docs.google.com/document/d/1yL3lWk_iFKqLTyekkuaiKXZ78I0lPmD5kM1fghHRs4Y/edit?usp=sharing