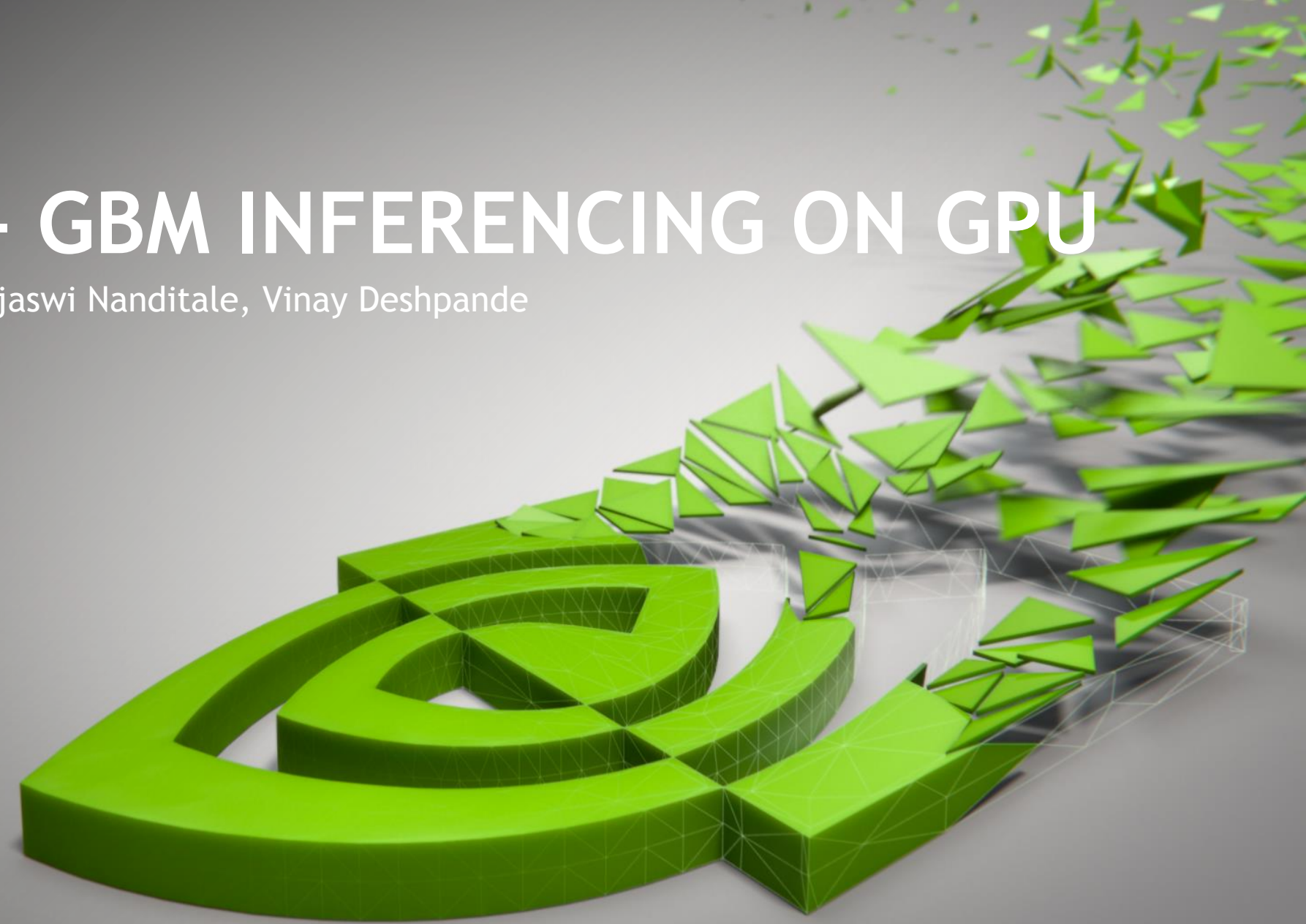
S8873 – GBM INFERENCING ON GPU

Shankara Rao Thejaswi Nanditale, Vinay Deshpande

NVIDIA



AGENDA

Introduction

Objective

Experimental Results

Implementation Details

Conclusion

INTRODUCTION

BOOSTING

What is it?

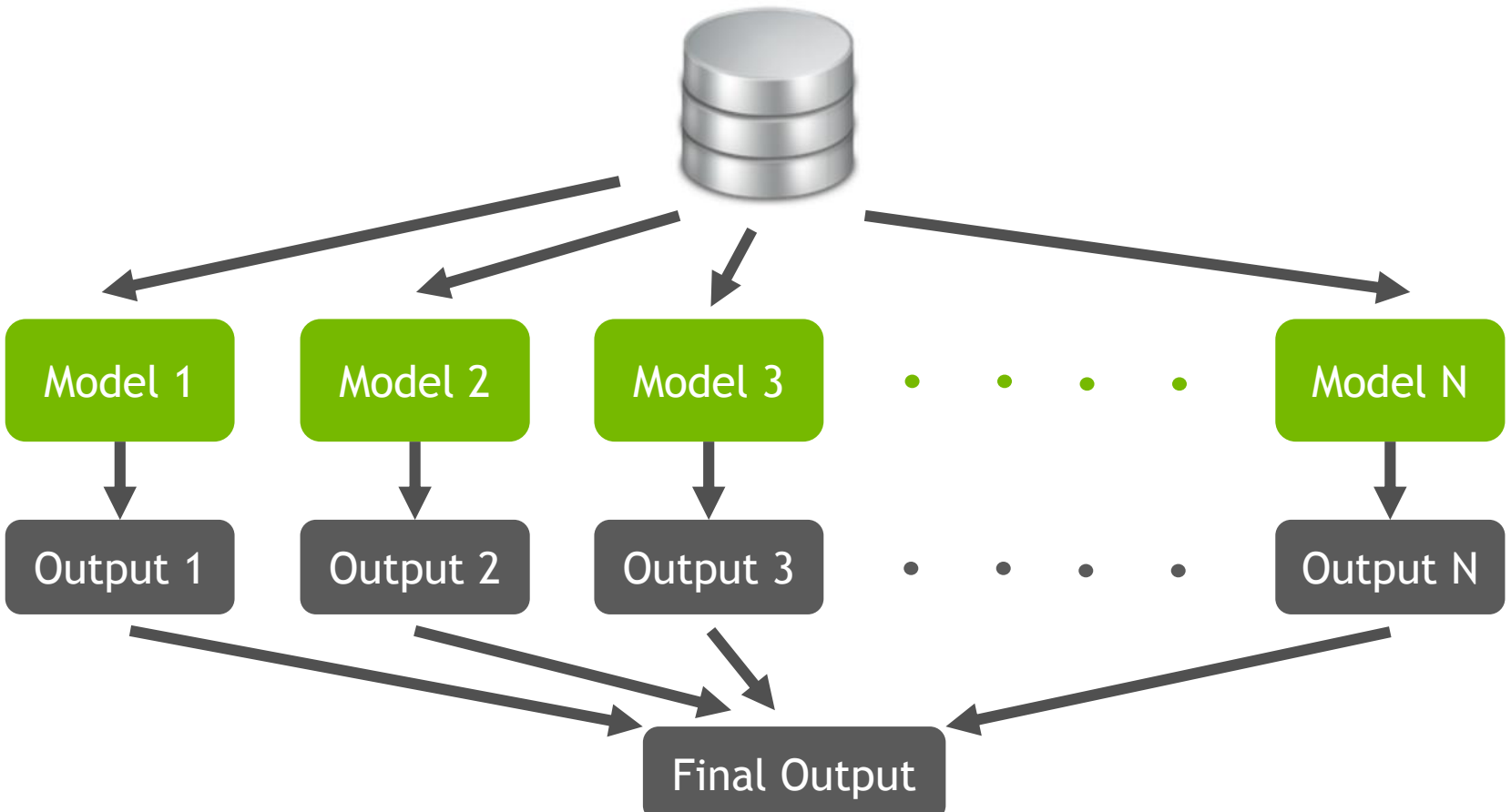
Training an ensemble of “weak learners”

Together, this ensemble becomes a strong predictor or model

Old but popular Machine Learning (ML) technique

It helps reducing bias and variance in the models

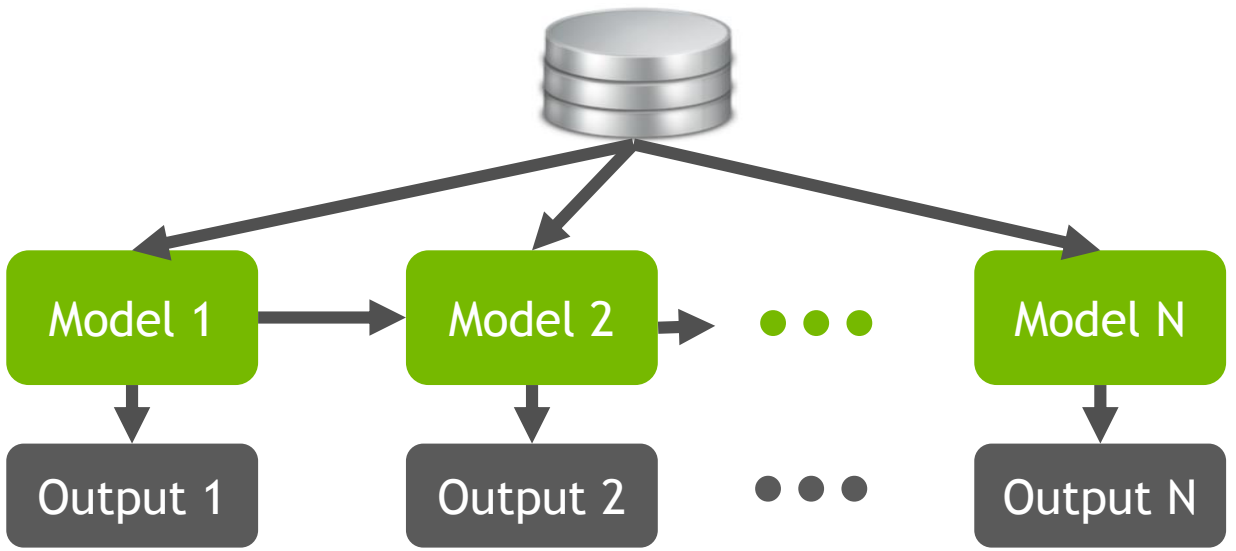
BOOSTING


Model 1
Model 2
Model 3
Model N
Output 1
Output 2
Output 3
Output N
Final Output

GRADIENT BOOSTING

Iteratively perform gradient descent on the function space

Build a weak learner at each step of this process

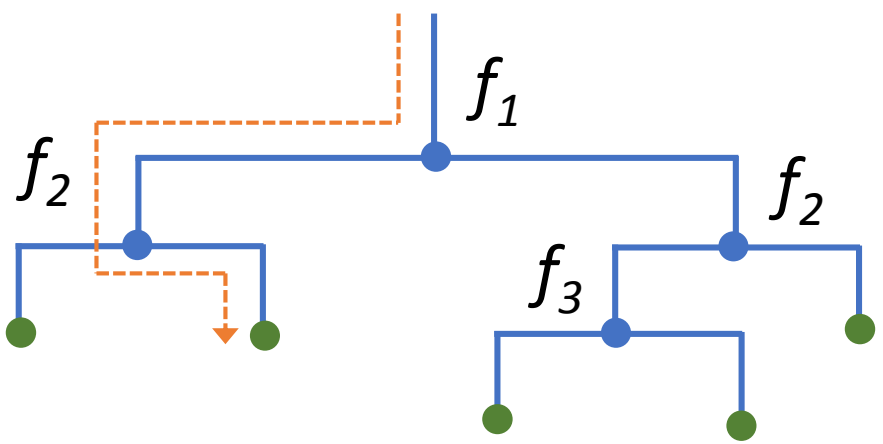


Successive learners weigh more on the mistakes by their predecessors

GRADIENT BOOSTED TREES

Form of gradient boosting where the “weak learners” are decision trees

This is what is being considered for our experiments



GRADIENT BOOSTED INFERENCE

AKA testing

AKA prediction

Process of applying learned model on real world data

Synonymous with deploying learned models in production

Batched-inference is inference across multiple such observations at once

GRADIENT BOOSTING

Libraries*

Scikit-learn – GradientBoostingClassifier

XGBoost

LightGBM

CatBoost

S8393 – CatBoost: Fast Open-Source Gradient Boosting Library for GPU

Tensorflow Boosted Trees (TFBT)

** In no particular order*

GRADIENT BOOSTED INFERENCE

Libraries

All Gradient Boosted libraries support inferencing

Treelite

- An inference-only library
- Multi-threaded on CPUs
- Converts the decision tree into C-code and executes it

OBJECTIVE

OBJECTIVE

Build a GPU-accelerated inference-only library for gradient boosting

Use decision trees as our “weak learners”

Assume batched inferencing scenarios

Limit ourselves to single-GPU*

** Batched inference is trivially parallelizable across multiple-GPUs (and also across multiple-nodes)*

GPU INFERENCING LIBRARY

Introduction

This talk introduces GIL (GPU/GBM Inferencing Library)

An efficient GPU accelerated inference-only library for gradient boosted trees

ASSUMPTIONS

For this work

Input dataset is dense

Ensemble of trees are also dense

Fully grown trees with leaves only at the deepest layer*

** If not, trivial to construct one such, from a sparse tree*

EXPERIMENTS

SETUP

Targets

- GPU Inferencing Library (GIL)
- xgboost.Booster.predict – On GPU @5ef6846
- Treelite – CPU only @332ffba

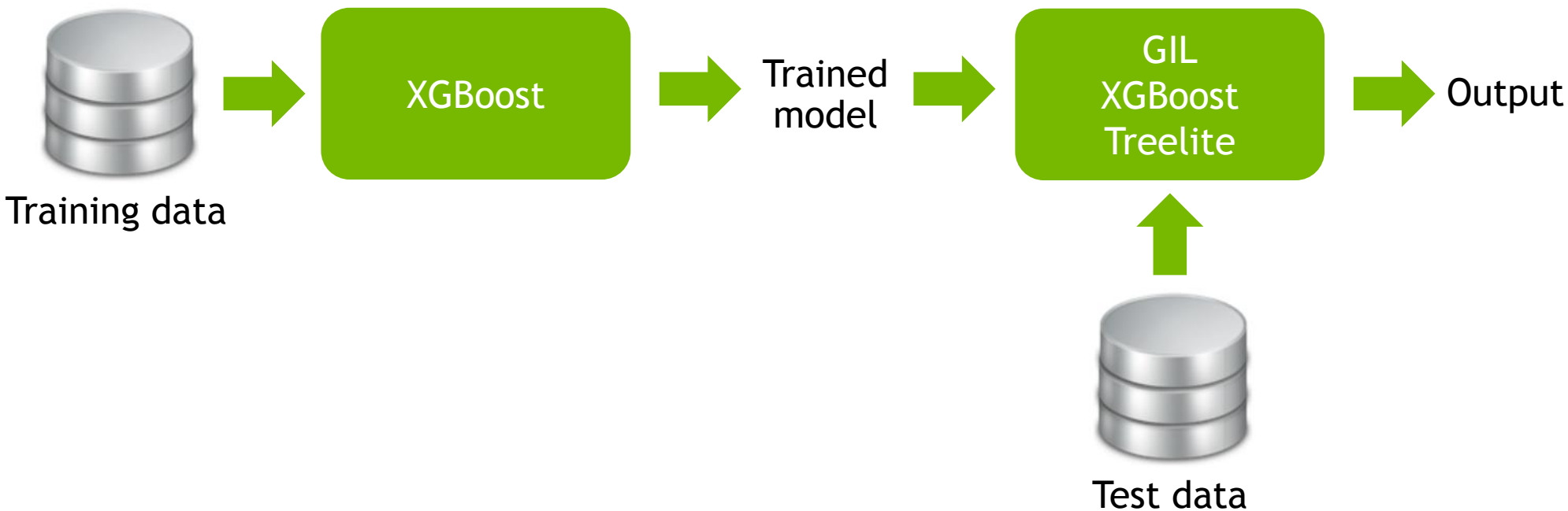
Datasets

- Airline
- Higgs

HW/SW

- Centos – 7.3
- Python 2.7
- V100 @ CUDA9.1
- Driver Version – 387.26

EXPERIMENT WORKFLOW

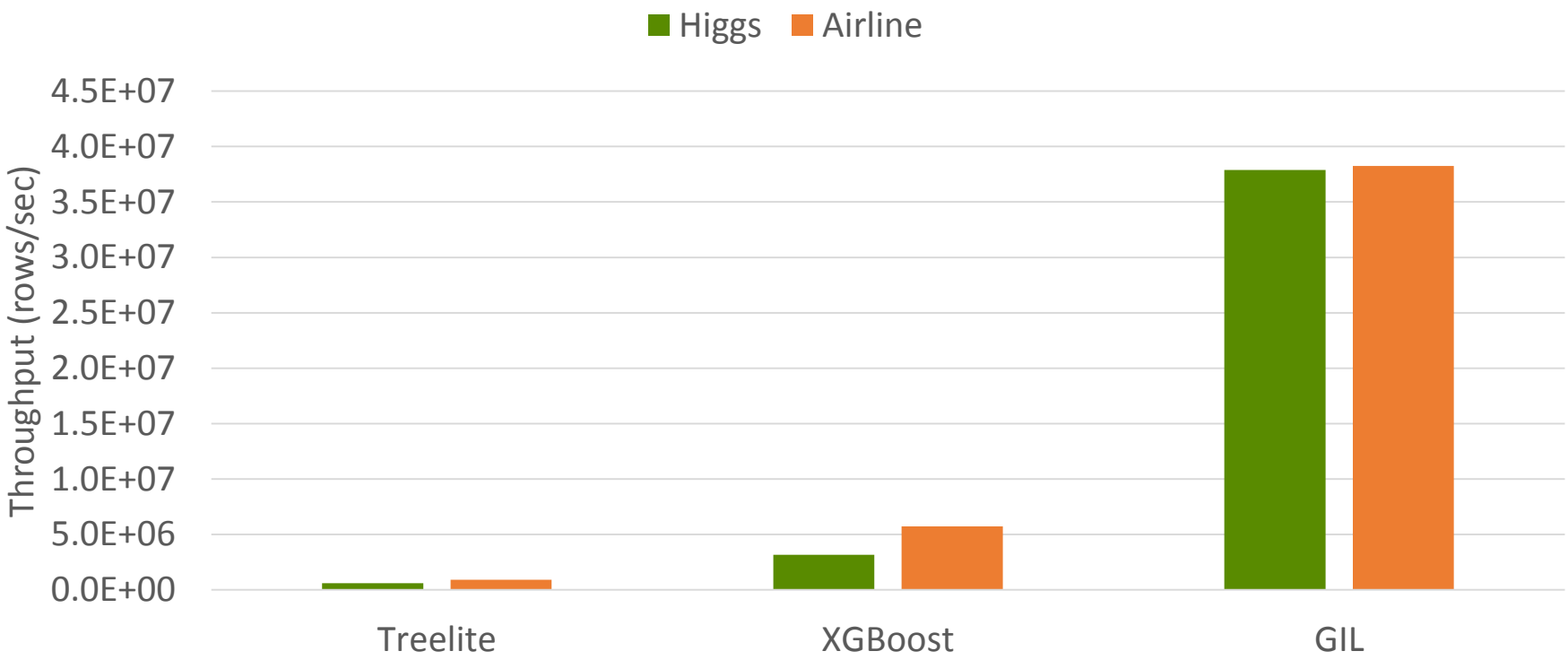


CORRECTNESS

For testing correctness of our implementation, we match the output value of the ensemble before classification

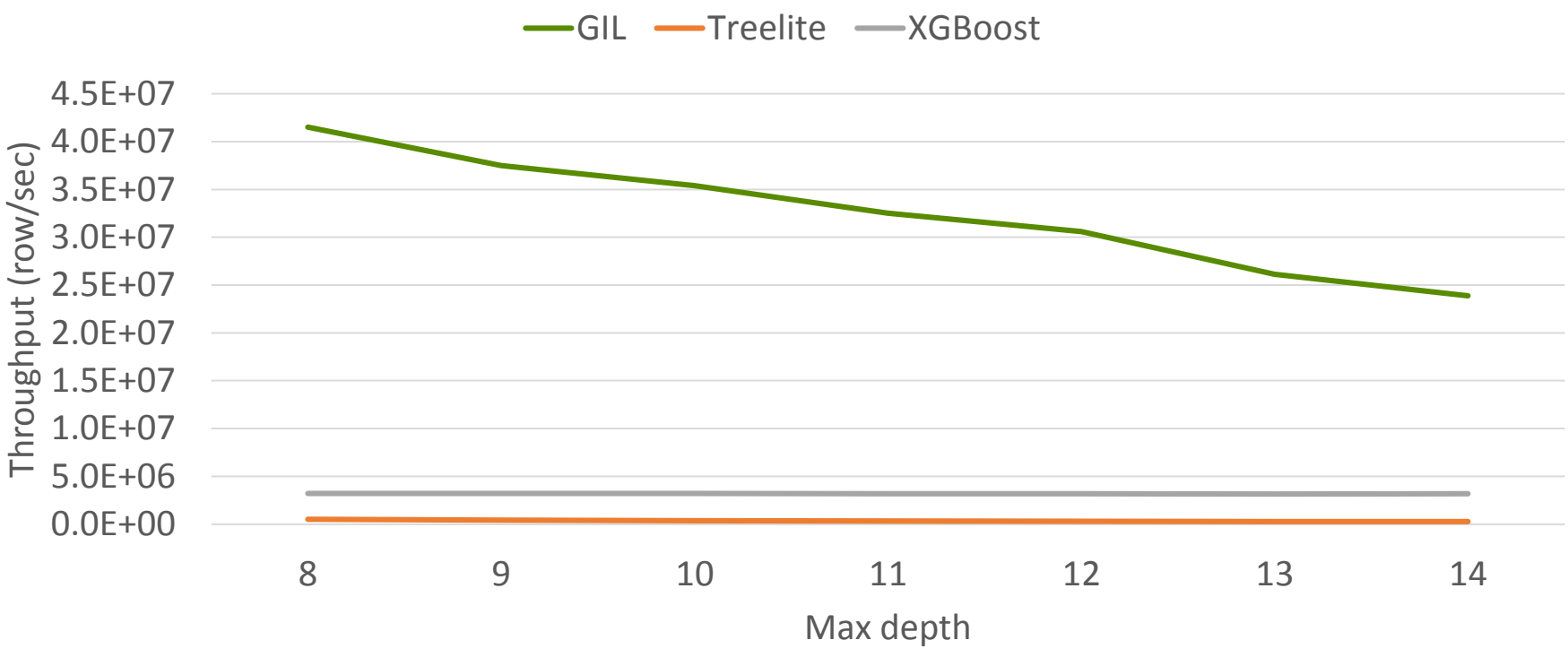
This means exact matching with the other libraries

AGGREGATE COMPARISON



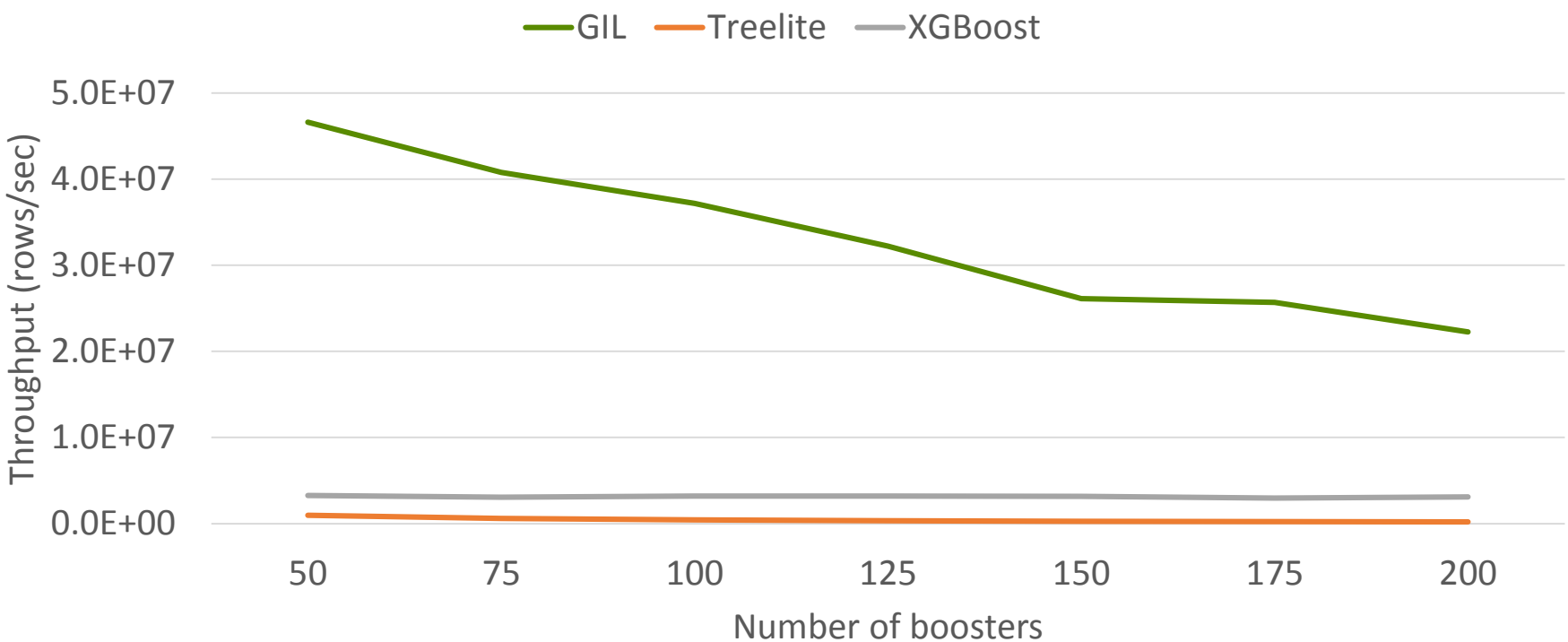
Dataset: Higgs and airline| Number of boosters: varying |
Max depth: varying | GPU – V100 | Batch size: 4 million

EFFECT OF MAX DEPTH



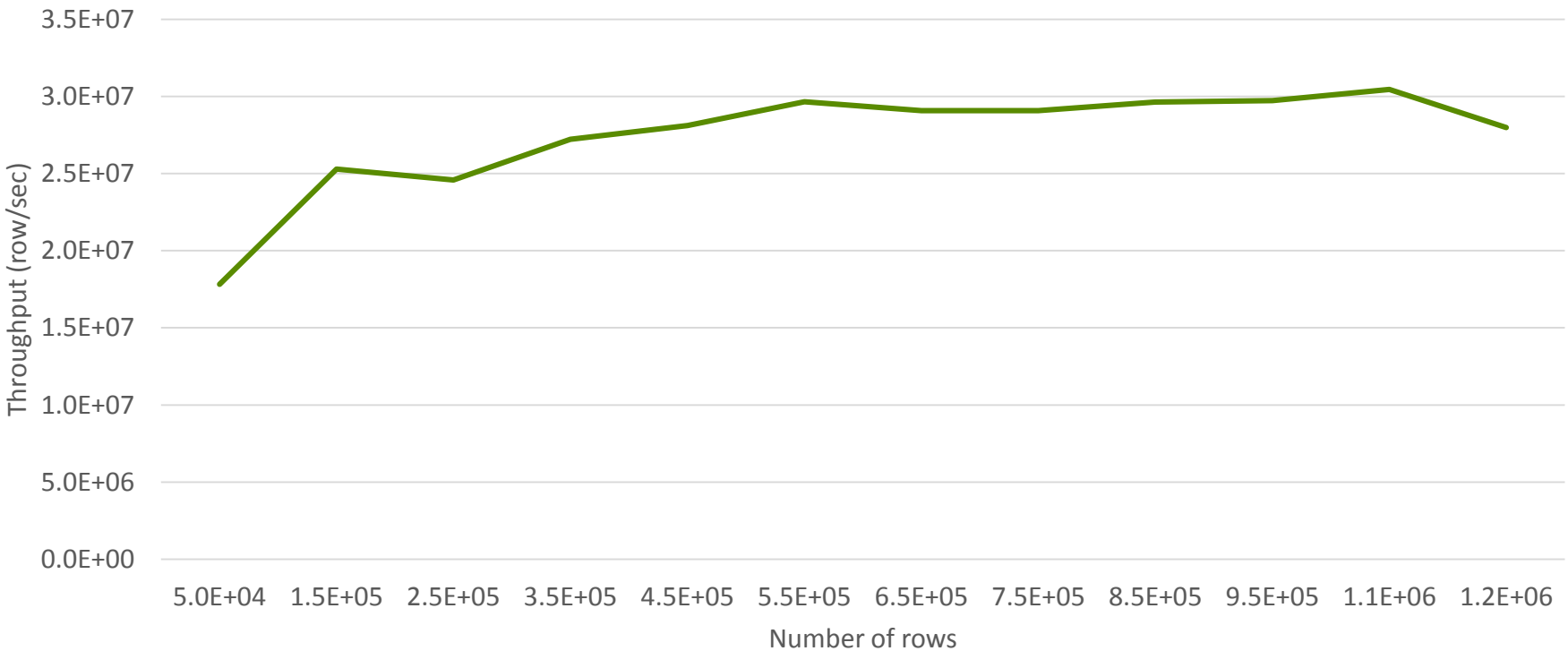
Dataset: Higgs| Number of boosters: 150 | Max depth: varying | GPU – V100 | Batch size: 4 million

EFFECT OF NUMBER OF BOOSTERS



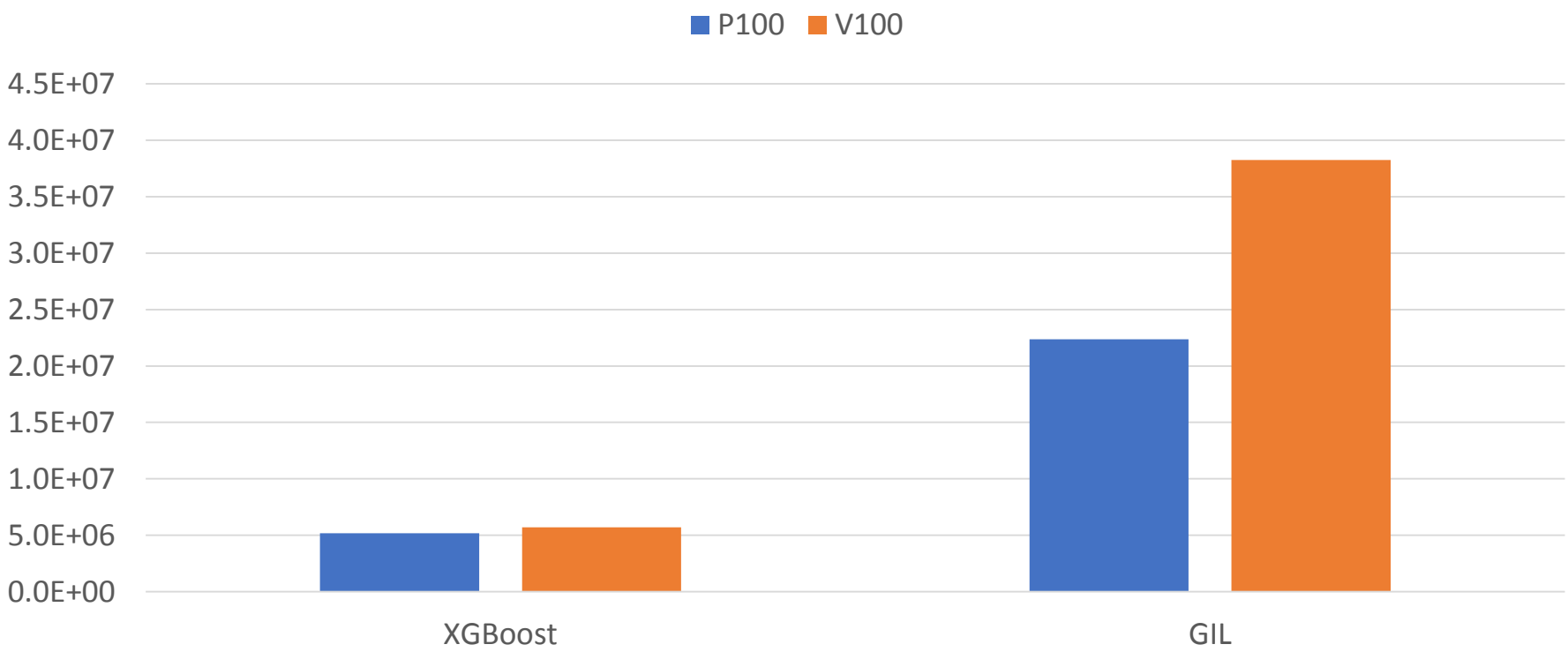
Dataset: Higgs| Number of boosters: varying | Max depth: 13 | GPU – V100 | Batch size: 4 million

EFFECT OF BATCH SIZE



Dataset: Airline| Number of boosters: 200 | Max depth: 10 | GPU – V100 | Batch size: varying

V100 VS P100



Dataset: Airline| Number of boosters: varying | Max depth: varying | GPU – P100 & V100 | Batch size: varying

IMPLEMENTATION

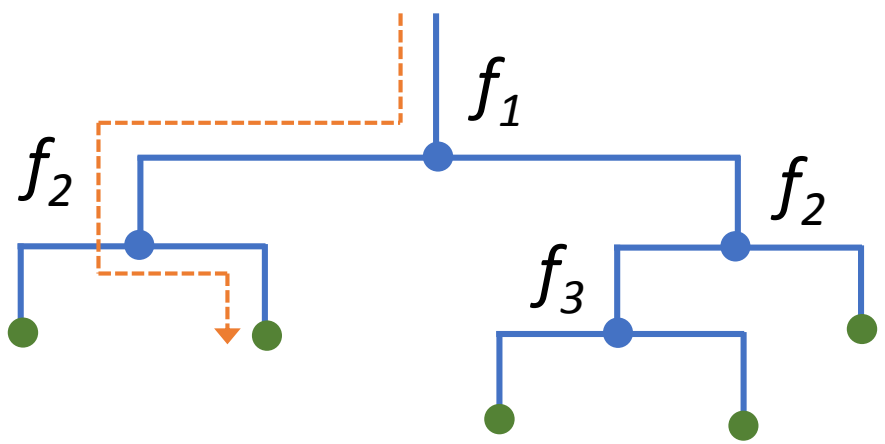
CHALLENGES

Control divergence – potentially different traversals across trees in the ensemble

Uncoalesced memory accesses

Different traversals will also lead to requests to far-off memory locations

Which also means, Treelite-like code generation is not an optimal for GPUs



REDUCING MEMORY UNCOALESCING

Shared Memory

Store the input row for a given threadblock in shared memory

Helps eliminate uncoalesced accesses to input data

Reduce global memory traffic

ELIMINATING CONTROL DIVERGENCE

control divergence

```
if( value <= threshold ) {
    next = 2 * current + 1;
} else {
    next = 2 * current + 2;
}
```

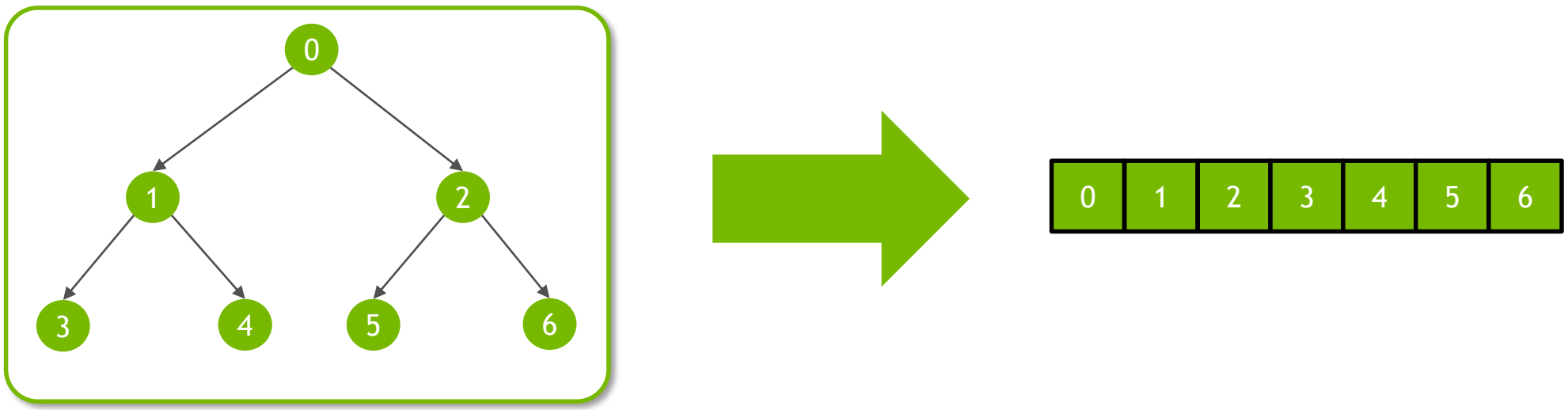
no control divergence

```
next = 2 * current + 1 + (value <=
threshold)
```

NAÏVE

Tree Layout

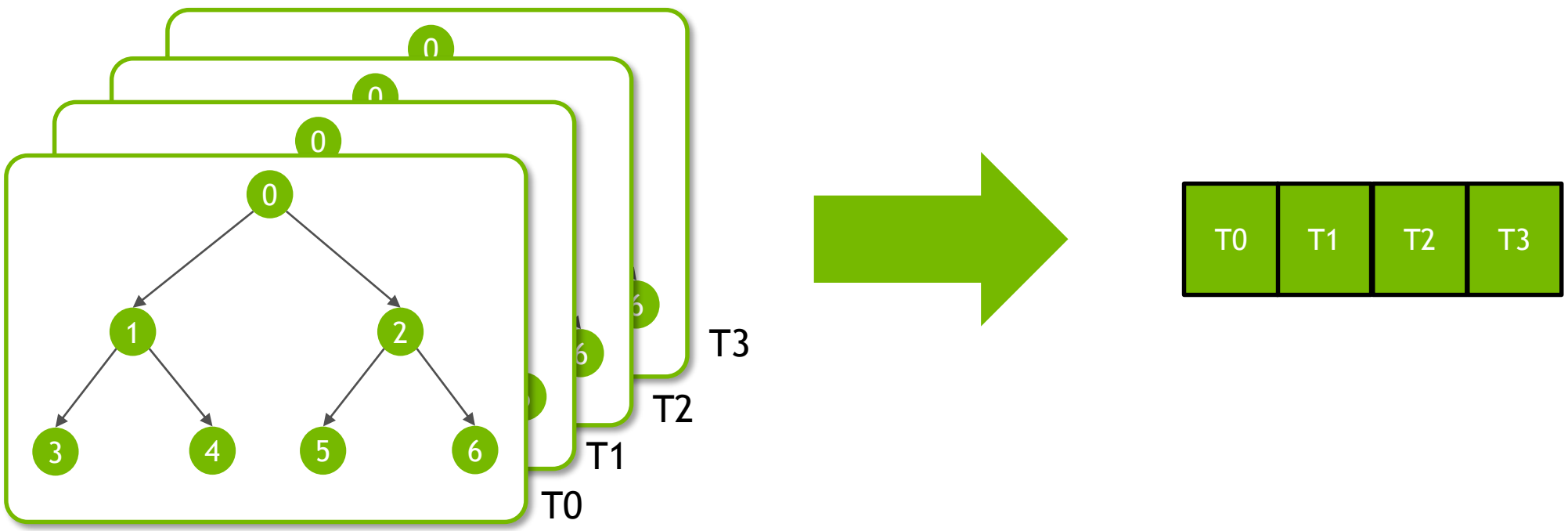
Store the nodes in the usual BFS order



NAÏVE

Tree Layout – Contd.

Store all the trees themselves adjacent to each other



NAÏVE

Work distribution

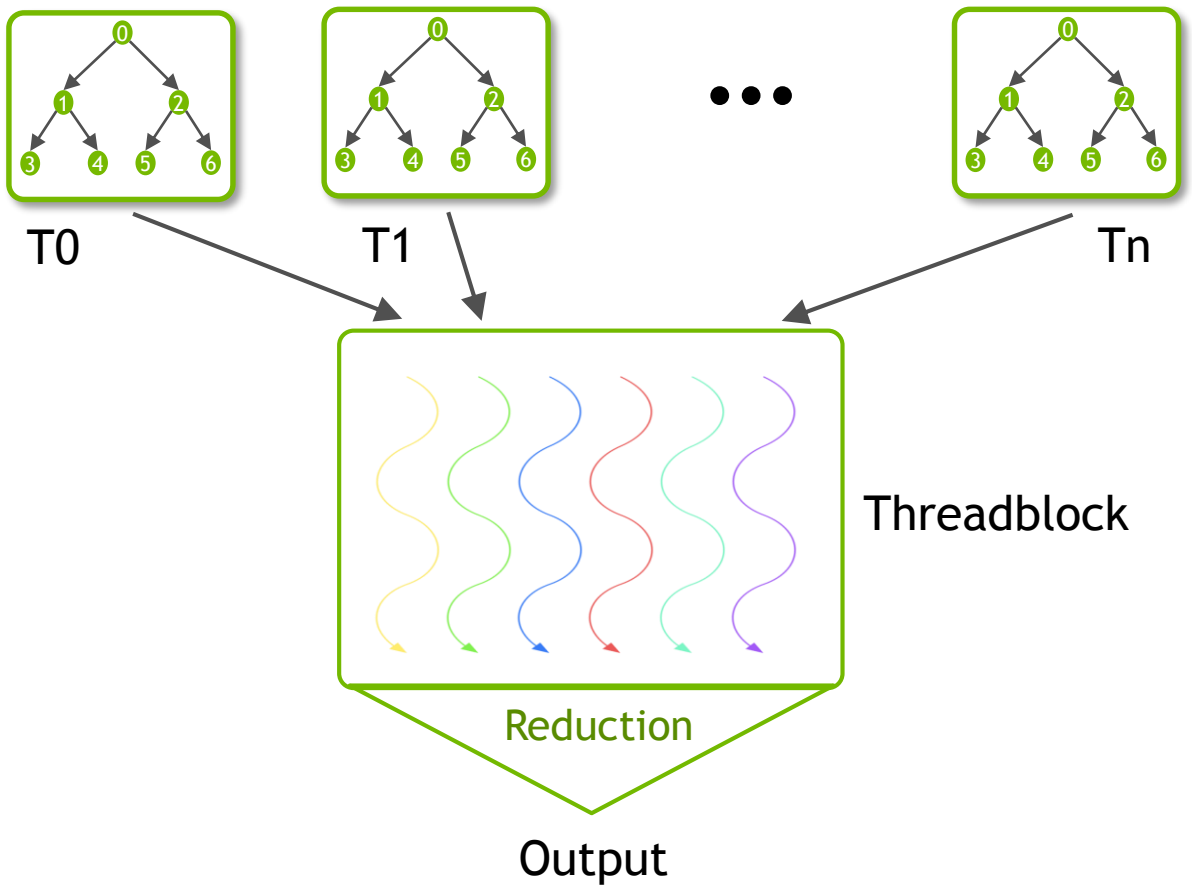
Each threadblock works on a row of the input dataset

Each thread works on a tree in the ensemble

Final stage involves a reduction operation across threads in the threadblock

NAÏVE

Work distribution



NAÏVE

Issues

Each thread works on nodes from different trees

Suffers from memory uncoalescing on node accesses

Uncoalesced accesses also result from reading input data while tree traversal

CUSTOM TREE LAYOUT

GPU cache-friendly layout for nodes

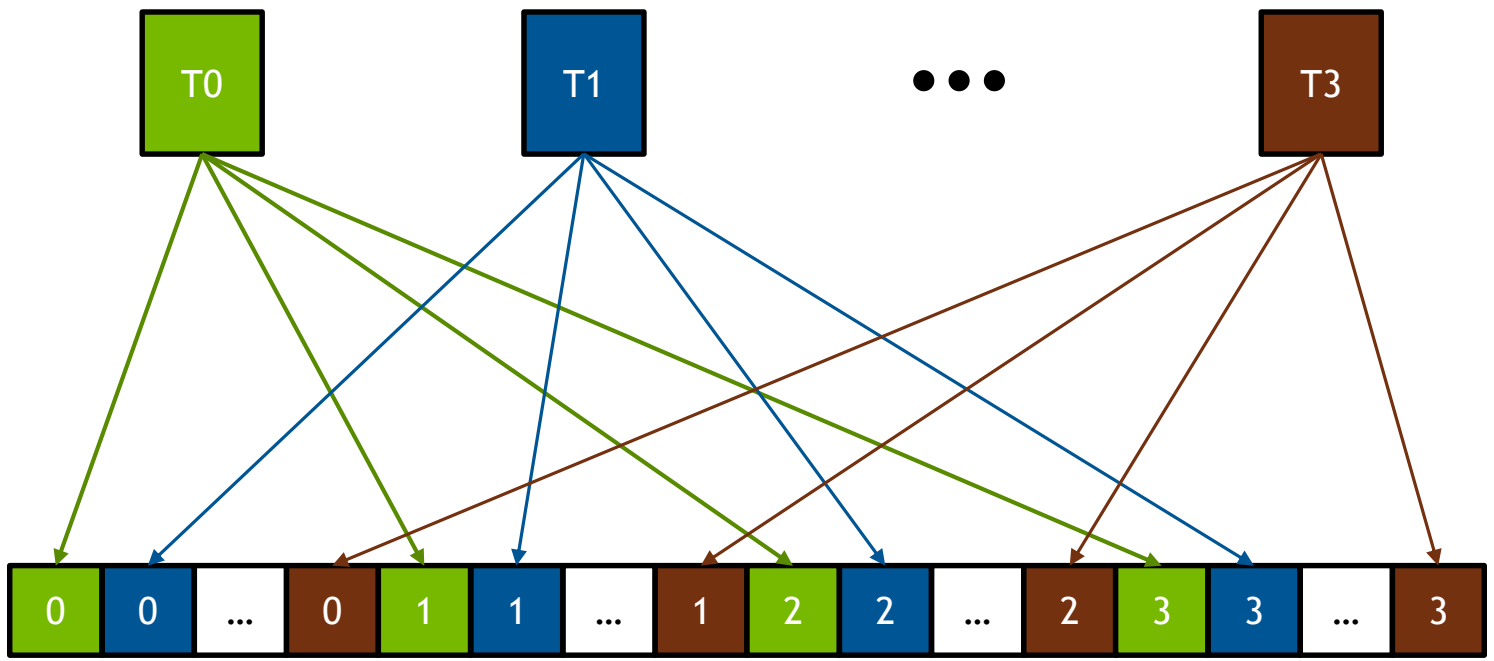
Store a given node of all trees adjacent to each other

Have these collection of nodes themselves in the usual BFS order

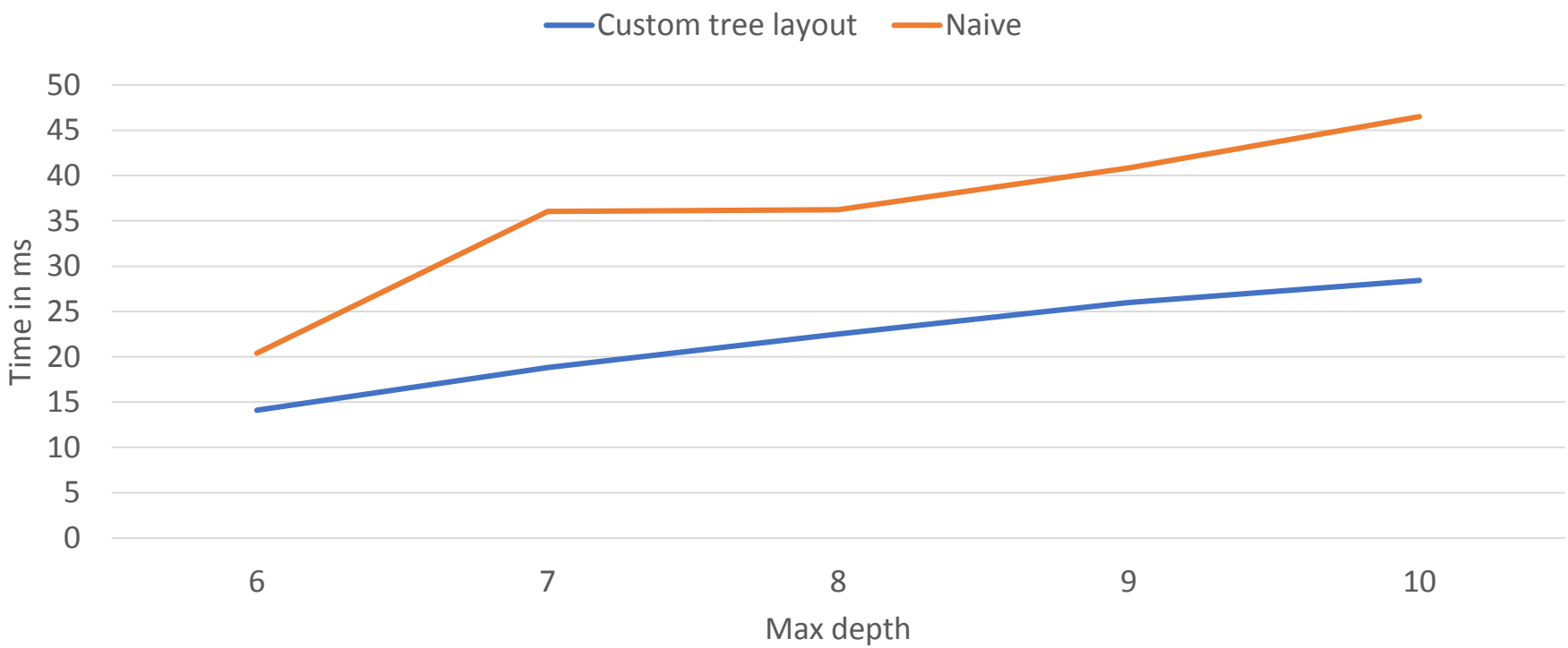
Helps reduce uncoalesced accesses

CUSTOM TREE LAYOUT

Illustration

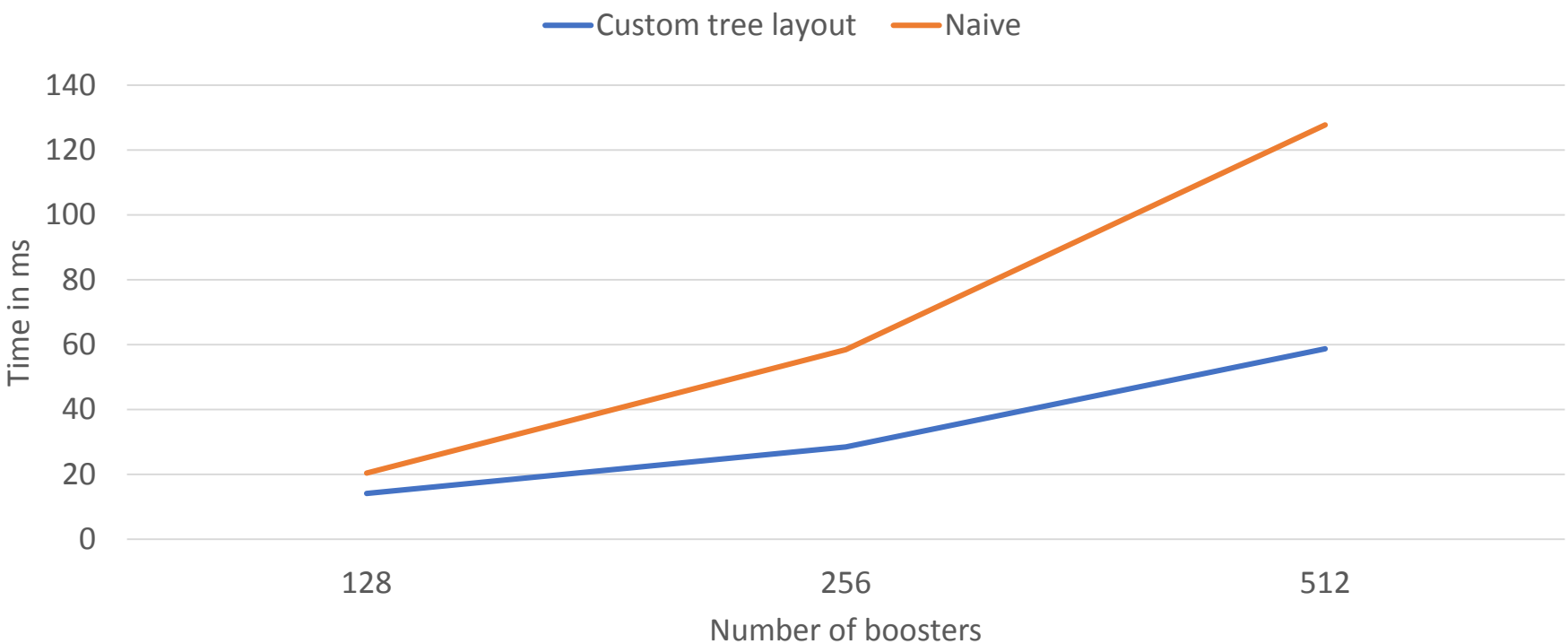


EFFECT OF CUSTOM TREE LAYOUT



Dataset: Synthetic| Number of boosters: 128| Max depth: varying | GPU - P100 | Batch size: 1 million

EFFECT OF CUSTOM TREE LAYOUT



Dataset: Synthetic| Number of boosters: varying| Max depth: 6 | GPU – P100 | Batch size: 1 million

CONCLUSION

TAKEAWAYS

There is a need for efficient and optimized batched inferencing

Large scope for improvements in the existing solutions

A custom GPU cache-friendly tree layout to reduce memory uncoalescing

GIL is a high-throughput batched inference-only library for GPU

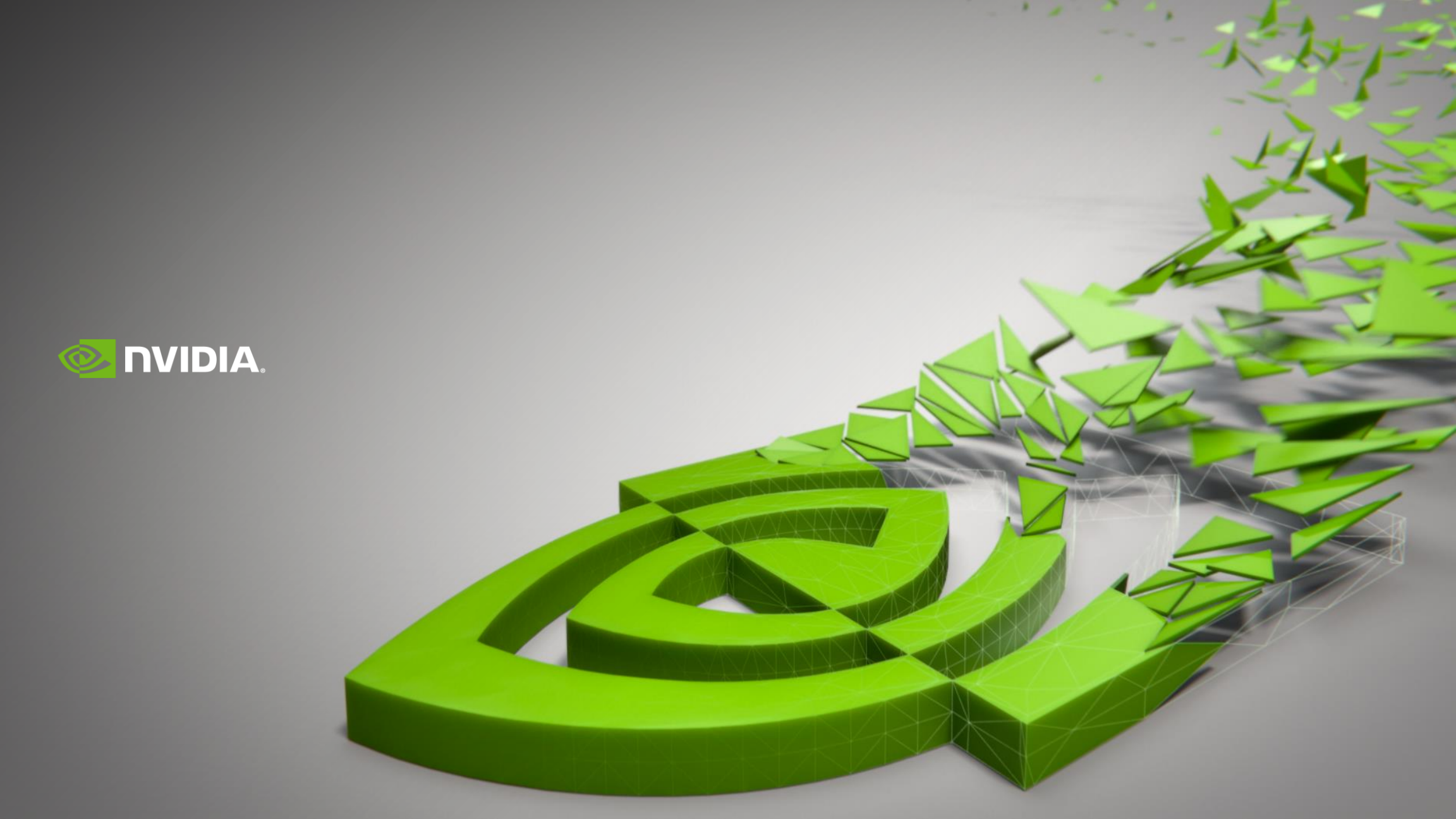
FUTURE WORK

Support for sparsity in binary trees

Support for sparsity in input data

- Challenges: need to tackle warp-divergence and uncoalesced memory accesses

Optimizing throughput for larger tree depths (>= 16) as well as number of trees


NVIDIA