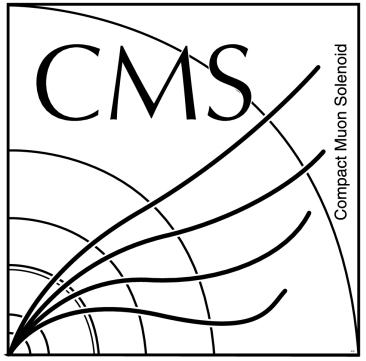
Fast Boosted Decision Tree inference on FPGAs for triggering at the LHC

CHEP 2018

Sioni Summers



Imperial College London

Contents

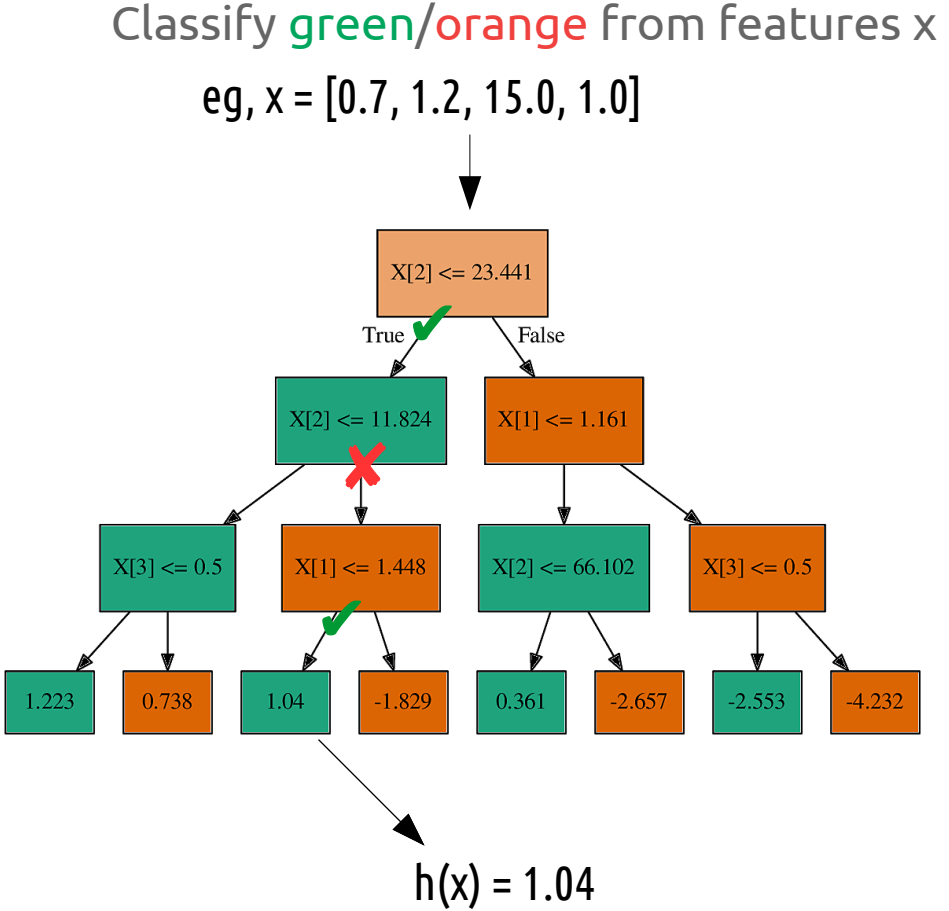
- Boosted Decision Tree inference
- FPGA implementation
- Performance
- Accelerating BDT inference

Introduction

- Boosted Decision Trees (BDTs) are multivariate classifiers popular in offline reconstruction and analysis
- More effective than simple cuts
- BDTs could be used in hardware triggers to improve object / event identification
  - Train offline, classify online
  - e.g. regression BDT assigns p_T in current CMS endcap muon trigger from large Look Up Table (J.F. Low, Boosted Decision Trees in the CMS Level-1 Endcap Muon Trigger, TWEPP-17)
- Seek a generic low latency implementation of BDT inference

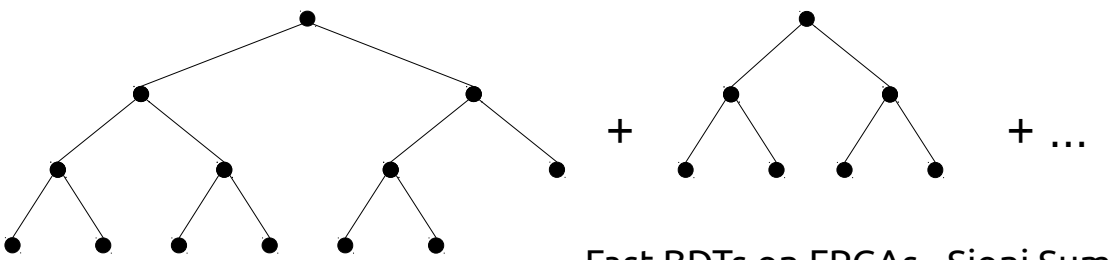
Decision Tree Inference

- Input feature vector x
- Compare a feature with threshold, follow decision path
- In DT final leaf gives class prediction
- In BDT final leaf gives tree score in ensemble



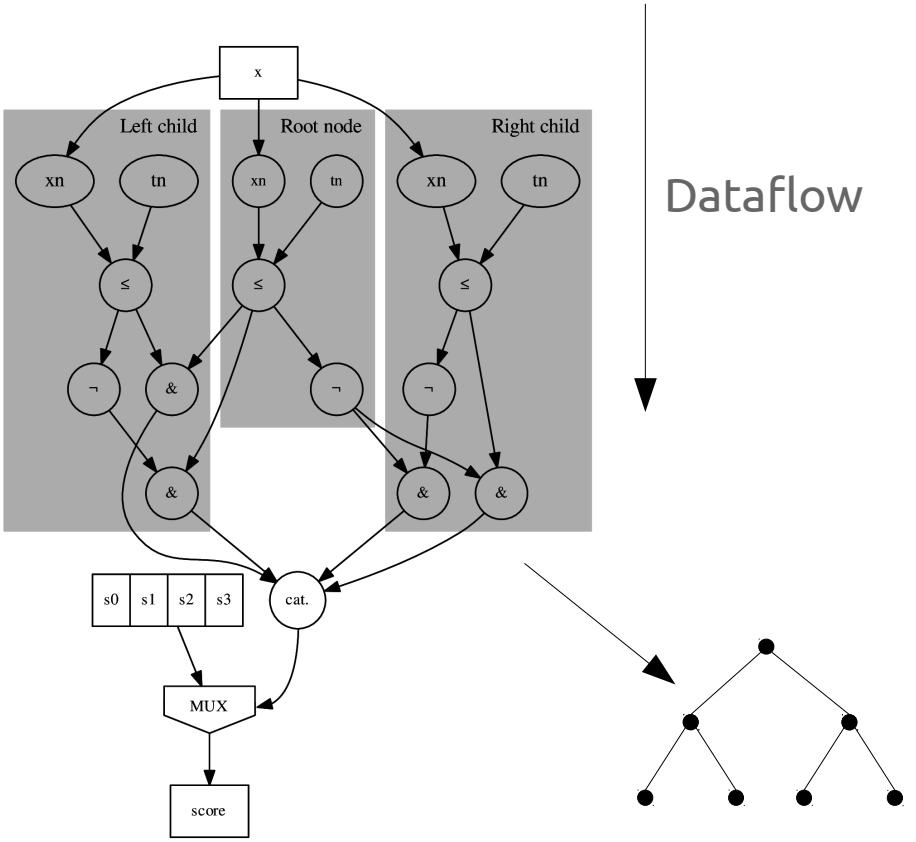
Boosted Decision Tree Inference

- Predict class of input using ensemble of decision trees
- $F(x) = k + \sum_i w_i h_i(x)$
- Score F is sum over weighted (w) scores of weak learners (decision trees), h
- Each h is independent → opportunity for parallelism
- An ensemble of weak learners makes a strong one



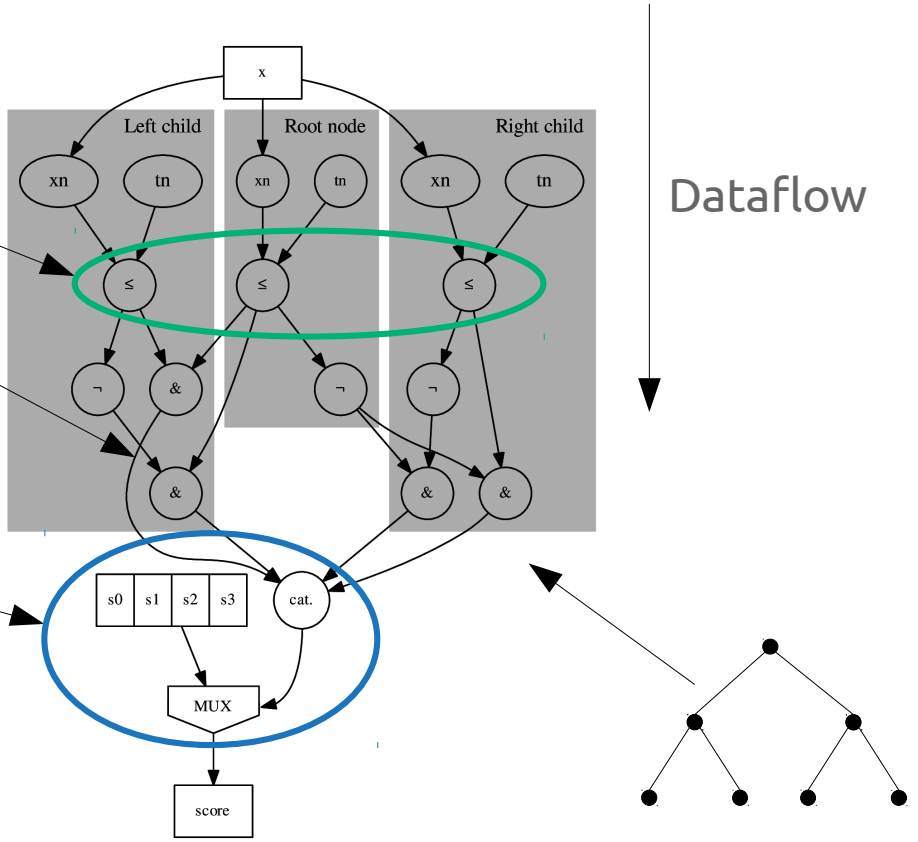
FPGA Implementation – Decision Tree

- Target lowest latency: unroll everything
- Do all node comparisons in parallel
- Represent each leaf with a boolean: True if decision path reaches leaf, False otherwise (only one leaf can be True)
  - Boolean combination of comparison results
- Use leaf bits to address table of values (small lookup)
- Fully pipelined (new data every clock)



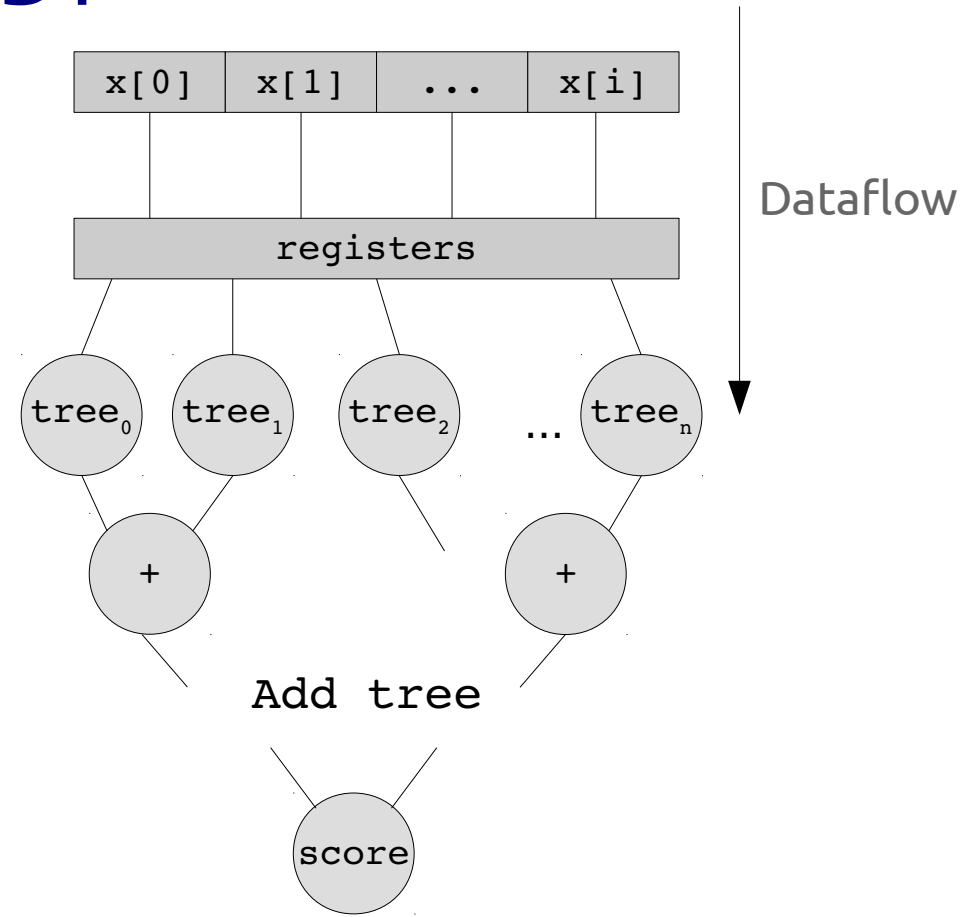
FPGA Implementation – Decision Tree

- Target lowest latency: unroll everything
- Do all node comparisons in parallel
- Represent each leaf with a boolean: True if decision path reaches leaf, False otherwise (only one leaf can be True)
  - Boolean combination of comparison results
- Use leaf bits to address table of values (small lookup)
- Fully pipelined (new data every clock)



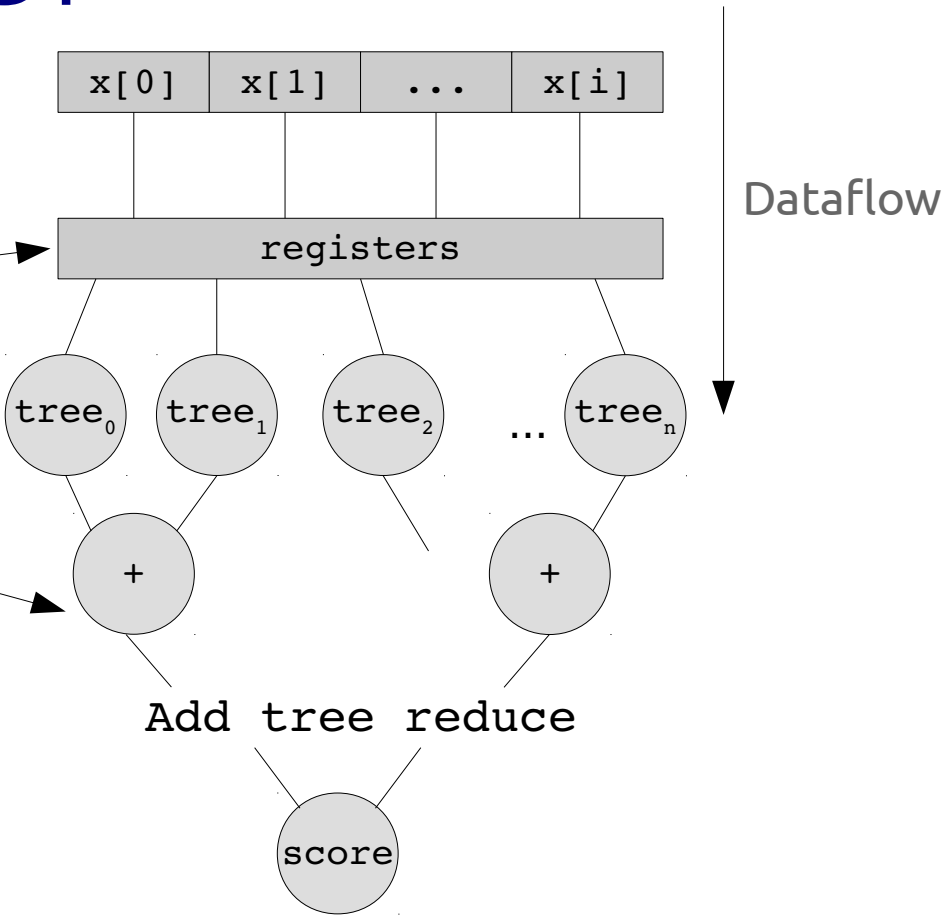
FPGA Implementation - BDT

- Target lowest latency: unroll everything
- Fanout input features to trees through layers of registers
  - For faster clock
- Execute all trees in parallel
- Perform sum of scores with balanced adder tree
- Output score
- Fully pipelined (new data every clock)



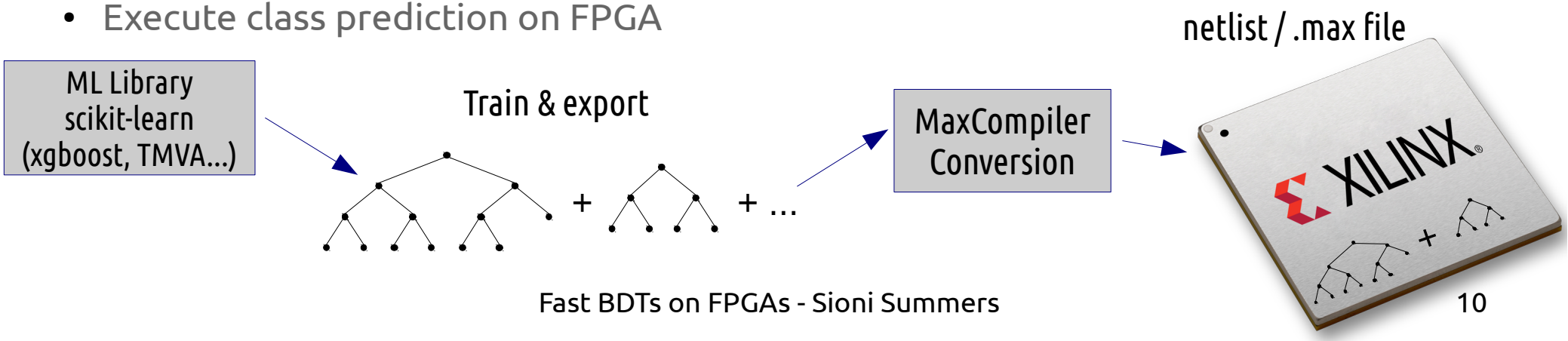
FPGA Implementation - BDT

- Target lowest latency: unroll everything
- Fanout input features to trees through layers of registers
  - For faster clock
- Execute all trees in parallel
- Perform sum of scores with balanced adder tree
- Output score
- Fully pipelined (new data every clock)



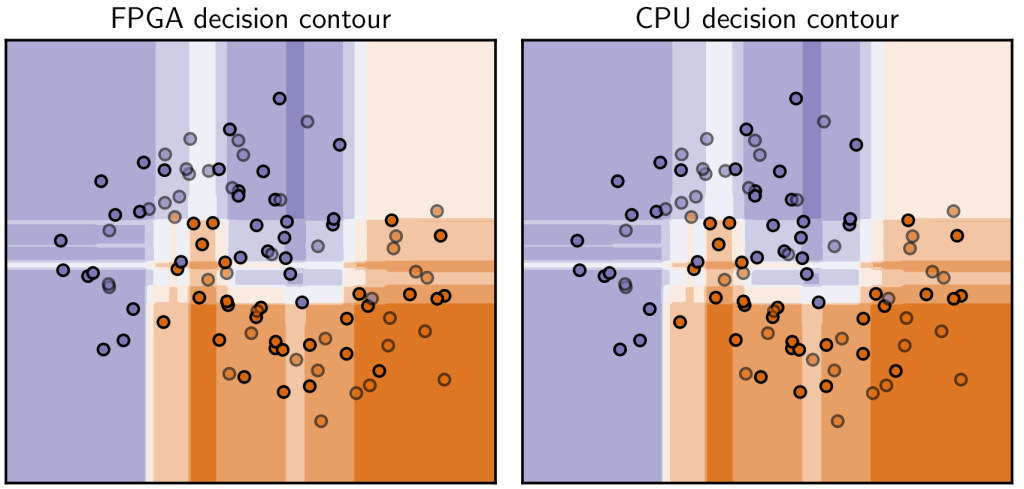
FPGA implementation - overview

- Train and export ensemble with ML library on standard hardware (scikit-learn on a PC here)
- Specify data formats: fixed-/floating-point types
- MaxCompiler project reads saved ensemble and generates:
  - VHDL / netlist for integration into bigger project (e.g. L1 Trigger)
  - .max file for execution on Maxeler hardware (e.g. acceleration)
- Execute class prediction on FPGA



Implementation Validation

- Validate with randomly generated, separable, dataset with 2 classes (red, blue) and 2 features
- Train BDT with scikit-learn
- Measure class predictions from FPGA and CPU (scikit-learn)
- Same results ✓



Points are training data, shading is class prediction for finely sampled mesh (separately for each architecture)

Vital statistics

- BDT with 100 tress, max depth 3
- Algorithm latency: 12clks, 30ns @ 400 MHz
  - Pipelining means classification rate is 400 MHz
- Tested at 400 MHz on Stratix V (should go faster on newest chips)
- Post synthesis resource usage for some popular Xilinx chips, and scaling:

Device	Virtex 7 690	Kintex Ultrascale 115	Virtex Ultrascale 9+
LUTs (%)	2.24	1.46	0.82
Registers (%)	1.15	0.75	0.42
Parameter	nTrees	maxDepth	nFeatures
Latency	log(n) (score sum)	n	-
Resources	n (tree & add)	2^n	-

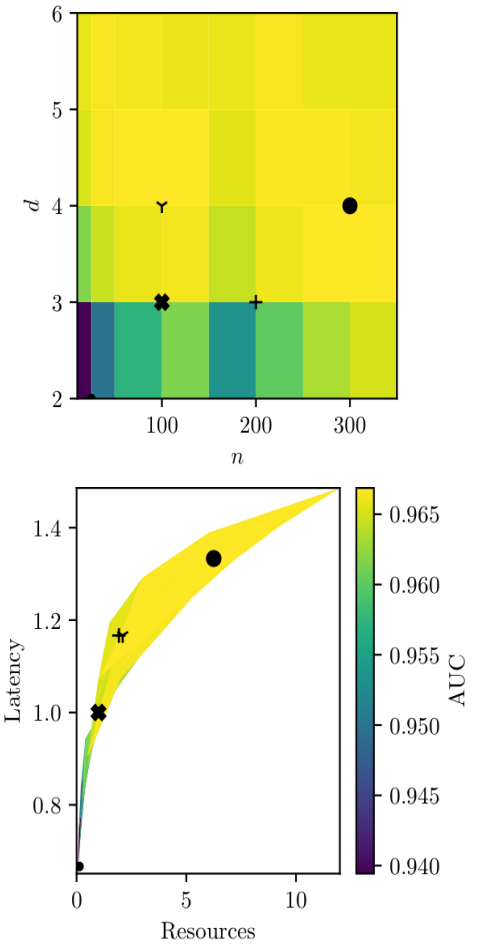
Hyperparameter Tuning

- Scaling shown on previous slide gives model for resources & latency relative to reference:

$$R = \frac{n}{n_{\text{ref}}} \times 2^{d-d_{\text{ref}}}$$

$$T = \frac{5}{12} + \frac{7}{24}\left(\log_2\left(\frac{n}{n_{\text{ref}}}\right) + \frac{d}{d_{\text{ref}}}\right)$$

- n trees (n_{ref} = 100), depth d (d_{ref} = 3)
- R resources, L latency

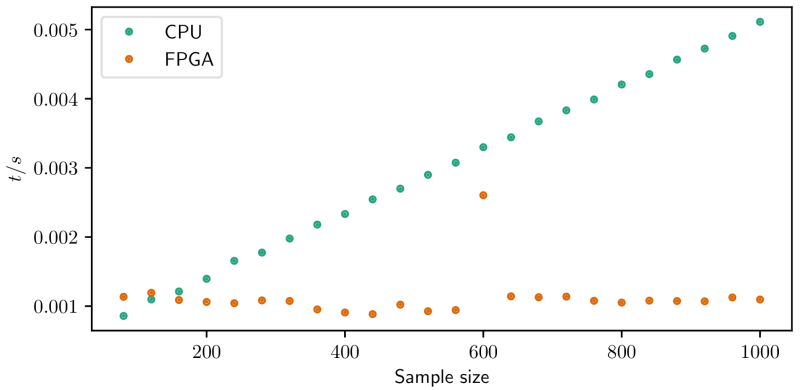


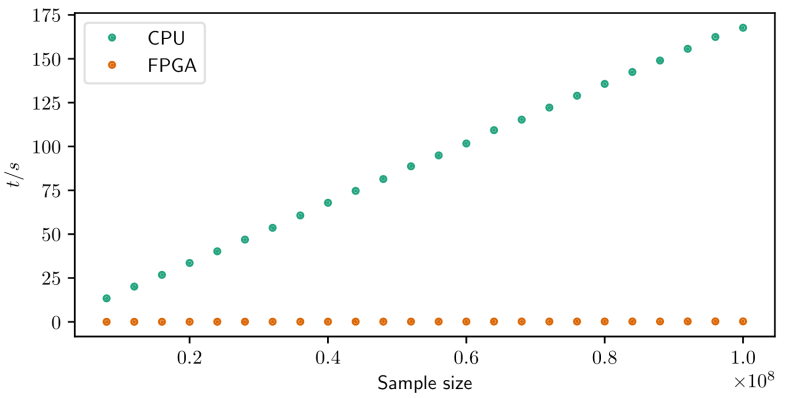
Acceleration - setup

- Using Intel Xeon CPU & 1 Maxeler “Maia” card in MPC-X
  - Connected via infiniband & PCIe switch
  - Altera Stratix V FPGA
- Train 100 trees, depth 3, 2 random features
- Generate random feature vectors
- Classify either on CPU (python, scikit-learn, single threaded) or FPGA

Acceleration - results

- Latency penalty to transfer data host → FPGA → host
- For sample sizes ≳ 100, FPGA is faster
- Up to 600x speedup for large samples
- FPGA classification rate limited at IO bandwidth





Conclusion

- Presented fast BDT inference for real-time applications
  - Low latency: ~30 ns
    - Suitable for hardware triggers
  - High throughput: ~ 400 x 10^6 classifications s^{-1}
    - suitable for compute acceleration on Maxeler Dataflow Engines

- Maxeler : http://maxeler.com/
- MaxCompiler BDT: https://github.com/thesps/MaxBDT

Application – CMS Level 1 Track Trigger

- Reconstructing tracks at Level 1 for HL-LHC
  - Tracks with p_T > 2-3 GeV
  - Latency < 4μs
  - New event every 25ns
- 20% of tracks in $t\bar{t}$ + 200 pileup are fake
  - Degrades performance of, e.g., vertexing, track MET
- Classify tracks as genuine/fake using reconstructed track parameters